UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

DISEÑO DE PRÁCTICAS DE LABORATORIO PARA TRABAJO EN JAVA CON PROCESSING Y VAADIN

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Juan José Moreno Escámez
Dirigido por: Pepa López López
Convocatoria: *Terrassa, 10 de mayo de 2019*
Curso Académico 2018-2019

AGRADECIMIENTOS

A mi tutora, Pepa López, por dirigirme el proyecto y estar siempre dispuesta a ayudar.

A mi amigo Roberto Flores, por prestarme Arduino para poder trabajar en las prácticas de Processing.

A mis padres, por siempre estar a mi lado apoyándome.

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

RESUMEN

En este documento se presenta el desarrollo del diseño de prácticas de laboratorio para trabajo en Java con Processing y Vaadin. También se introducirá la comunicación con Arduino para hacer diferentes prácticas y acondicionarlas en la ingeniería.

El trabajo incluye los manuales previos de instalación de las aplicaciones, los enunciados de las prácticas y sus soluciones. Las prácticas están hechas en base a la taxonomía de Bloom, trabajando por niveles ascendentes de complejidad y dificultad.

En la primera parte del proyecto se describen las diferentes tecnologías utilizadas, así como también el lenguaje de programación, Java. Finalmente, se explica de qué tratan las prácticas y sus niveles de complejidad.

ÍNDICE

1. **OBJETO** ..... - 9 -

2. **ALCANCE** ..... - 10 -

3. **ESPECIFICACIONES BÁSICAS** ..... - 11 -

4. **JUSTIFICACIÓN** ..... - 12 -

5. **DESCRIPCIÓN TECNOLOGÍAS UTILIZADAS** ..... - 13 -

5.1. *JAVA* ..... - 13 -

5.2. *PROCESSING* ..... - 15 -

5.3. *VAADIN* ..... - 17 -

5.4. *ARDUINO* ..... - 19 -

5.5. *TAXONOMÍA DE BLOOM* ..... - 21 -

6. **DESARROLLO** ..... - 23 -

6.1. *ANTECEDENTES* ..... - 23 -

6.2. *PLANTEAMIENTO Y DECISIÓN SOBRE SOLUCIONES ALTERNATIVAS* ..... - 24 -

6.3. *ESTUDIO DE MERCADO ARDUINO* ..... - 25 -

6.4. *DESARROLLO DE LAS SOLUCIONES ESCOGIDAS* ..... - 32 -

6.4.1. *PRÁCTICAS* ..... - 32 -

6.4.1.1. *PROCESSING* ..... - 32 -

6.4.1.1.1. Práctica 0: Introducción a Processing ..... - 32 -

6.4.1.1.2. Práctica 1: Primeros programas ..... - 33 -

6.4.1.1.3. Práctica 2: Paint ..... - 34 -

6.4.1.1.4. Práctica 3: Pelota saltarina ..... - 35 -

6.4.1.1.5. Práctica 4: Encuentra los elementos. ..... - 36 -

6.4.1.1.6. Práctica 5: Flappy Bird ..... - 38 -

6.4.1.1.7. Práctica 6: Puzzle ..... - 39 -

6.4.1.1.8. Práctica 7: Domótica ..... - 41 -

6.4.1.2.*ARDUINO* ..... - 42 -

6.4.1.2.1. Práctica 0: Introducción al Arduino ..... - 42 -

6.4.1.3.*VAADIN* ..... - 43 -

6.4.1.3.1. Práctica 0: Ejecución primer diseño Vaadin ..... - 43 -

6.4.1.3.2. Práctica 1: Formulario ..... - 44 -

6.4.1.3.3. Práctica 2: Noticias ..... - 45 -

6.4.1.3.4. Práctica 3: Registro ..... - 46 -

6.4.2. *MANUALES Y DOCUMENTACIÓN PARA EL PROFESORADO* ..... - 48 -

6.4.2.1. Manual Instalación Vaadin Designer ..... - 48 -

6.4.2.2. Manual funcionamiento del Arduino ..... - 49 -

6.4.2.3. Conceptos teóricos ..... - 50 -

7. **RESUMEN DE RESULTADOS** ..... - 51 -

7.1. PLANIFICACIÓN TEMPORAL ..... - 51 -

7.2. *PRESUPUESTO* ..... - 53 -

7.3. *CONCLUSIONES* ..... - 54 -

7.4. *GLOSARIO* ..... - 56 -

7.5. BIBLIOGRAFÍA ..... - 57 -

ANEXO A: ENUNCIADOS DE LAS PRÁCTICAS ..... - 59 -

ANEXO B: SOLUCIONES DE LAS PRÁCTICAS ..... - 100 -

ANEXO C: MANUALES Y DOCUMENTACIÓN PARA EL PROFESORADO ..... - 215 -

ANEXO D: AUTOINFORME DE CALIDAD ..... - 246 -

ÍNDICE DE FIGURAS

Fig. 1 Software de Processing..... - 16 -
Fig. 2 Entorno de Vaadin Designer en Eclipse..... - 18 -
Fig. 3 Niveles de la taxonomía de Bloom en la dimensión cognitiva..... - 22 -

ÍNDICE DE TABLAS

Tabla 1 Comparación de tipos de Arduino para la venta al público en Noviembre de 2018 .....- 25 -
Tabla 2 Comparación Kits de Arduino.....- 30 -
Tabla 3 Distribución de horas.....- 51 -
Tabla 4 Diagrama de Gantt. .....- 52 -
Tabla 5 Presupuestos del proyecto. .....- 53 -

1. OBJETO

El objetivo principal del proyecto es la realización de prácticas para complementar la asignatura de Programación Avanzada Orientada a Objetos, es decir, realizar diversas prácticas que sean creativas en las cuales el alumnado podrá mejorar sus conocimientos. Será de especial apoyo la plataforma Arduino.

Además del objetivo prioritario, tenemos otros más claros en cuanto a nivel práctico:

- **Utilización del entorno Processing y Vaadin**. El primero, Processing, para entrar en contacto con la interfaz y con su lenguaje, el cual incorpora ciertas clases y ciertas funciones dinámicas. Se puede hacer trabajos bastante entretenidos ya que permite trabajar tanto con imágenes como con vídeos, así como con oyentes desde teclado o el ratón. Por otro lado, Vaadin, que nos permite construir páginas web de una manera diferente y sencilla, en la cual se puede diseñar clases directamente colocando los componentes en el sitio deseado.
- **Utilización del Arduino**. Entrar más en detalle con esta plataforma de código abierto, para darle un uso sencillo a la vez que productivo incorporando la utilización de sensores, interruptores o pantallas LCD.

Todos estos objetivos, por lo tanto, pretenden que el alumno pueda aprender, mejorar sus aptitudes y, en definitiva, disfrutar de la programación con independencia total del nivel o conocimientos que tenga.

2. ALCANCE

Las prácticas realizadas complementarán la asignatura de Programación Avanzada Orientada a Objetos. Esta es una materia que imparte la unidad de Departamento de Ciencias de la Computación.

Es una asignatura optativa del itinerario general, es decir, la pueden cursar alumnos de diferentes grados, desde electrónicos que dominan más la programación hasta químicos que prácticamente no han programado a lo largo de la carrera. Por lo tanto, las prácticas que trata el proyecto deberán abarcar diferentes niveles de dificultad, porque los alumnos de cada grado tienen unos conocimientos diferentes de programación.

Las prácticas están pensadas para una dedicación de 10 horas en el laboratorio y 15 horas de aprendizaje autónomo. Las prácticas se han realizado con la idea de tener prácticas con distintos niveles para que el profesorado pueda elegir varias en función del alumnado matriculado a la asignatura.

Podrían matricularse una gran cantidad de alumnos que estudien la especialidad de electrónica, con lo cual, ya habrán trabajo previamente distintos lenguajes de programación, y se podría trabajar con prácticas de bastante nivel; por el contrario, se podría formar un grupo de alumnos de otra especialidad que no hayan programado mucho, y en ese caso se podría trabajar más con Arduino, que es una plataforma con la que no habrán trabajado. La idea principal es tener una variedad de prácticas con las que poder trabajar independientemente del tipo de carrera que estén cursando los alumnos.

3. ESPECIFICACIONES BÁSICAS

El proyecto trata de diseñar prácticas utilizando el lenguaje de programación Java, para realizar un taller de iniciación a Processing y a interfaces visuales utilizando Vaadin. Se deberán diseñar retos creativos e implementar las soluciones.

La materia tiene unos objetivos de aprendizaje, los cuales se pondrán a prueba durante todo el curso y los exámenes. Estos son los siguientes: utilizar el paradigma de programación a objetos de forma avanzada, diseñar e implementar interfaces gráficas de usuario, trabajar con eventos y comprender y utilizar clases de las librerías de estructuras de datos.

Al finalizar el curso, los alumnos deberán ser capaces de realizar implementaciones en Java para resolver problemas haciendo servir las librerías de Java, la orientación a objetos y las interfaces gráficas. Las propias prácticas tendrán sus objetivos, los cuales son que los alumnos aprendan a programar en Processing con la ayuda del Arduino y que diseñen interfaces gráficas de una manera más práctica. Pero, además, ayudarán al alumno a superar los objetivos de la asignatura ya que se potenciará las capacidades de los alumnos, sobretodo, el diseño de interfaces gráficas, la orientación a objetos y el trabajo con eventos.

4. JUSTIFICACIÓN

Con el objetivo de formar a los alumnos y por la motivación de realizar actividades de los conocimientos que obtuve cursando la asignatura de Complementos de Programación, decidí escoger este proyecto final de carrera.

Cursando esta asignatura aprendí y me divertí, ya que la rama que más me gusta del grado que he cursado es la programación, por lo tanto, era una buena oportunidad para refrescar conocimientos y ampliarlos utilizando nuevas aplicaciones y seguir aprendiendo del gran mundo que es Java.

Nunca había utilizado las plataformas de Processing ni Vaadin. Vaadin permite el diseño de páginas web arrastrando los componentes directamente y soltándolos en la posición que se desee, y Processing es el lenguaje ideal para los gráficos. Por estas razones, tengo que decir que me entusiasmó empezar a programar y aprender con estas aplicaciones.

Todas las decisiones y soluciones que se han adoptado a lo largo del proyecto son fruto de una documentación y formación previa realizada antes de empezar a desarrollar las prácticas, con objetivo de coger ideas para la realización de estas.

5. DESCRIPCIÓN TECNOLOGÍAS UTILIZADAS

5.1. *JAVA*

Java es un lenguaje de programación orientado a objetos el cual se empezó a comercializar como plataforma informática en 1991 por Sun Microsystems. Java en la actualidad está en todas partes, desde teléfonos móviles hasta bases de datos pasando por videoconsolas, la mayoría de estas no funcionarían sin tener Java instalado.

La creación de aplicaciones Java sigue en aumento, ya que utiliza un lenguaje de programación rápido, sencillo y muy bien estructurado debido a su orientación a objetos.

La programación a objetos es un paradigma de programar, que viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.[1]

Está basada en diferentes técnicas como herencia, polimorfismo, abstracción y cohesión. Estas aportan unas ventajas a esta forma de programar. La más importante es la reusabilidad, es decir, se permite utilizar clases diseñadas en distintas partes del programa y distintos proyectos. Es mucho más fiable, que el programa queda dividido y así es más fácil corregir posibles errores, y permite con facilidad añadir, suprimir y modificar nuevos objetos.

Uno de los grandes pilares de Java es que los programas funcionan en cualquier plataforma, es decir, pueden ejecutarse en diferentes sistemas operativos a diferencia de otros lenguajes que necesitan de versiones específicas para cada sistema operativo. Esto es posible ya que no es el sistema el que ejecuta los programas, sino la consola. Esta consola es llamada máquina virtual (JVM), es realmente quien va a ejecutar el código. Una vez compilado un programa en Java, este se compila mediante el compilador *Javac* a un lenguaje denominado *Byte Code*, y es en este momento donde la máquina virtual ejecuta el código en el sistema operativo que sea. La máquina virtual interpretará las instrucciones y las ejecutará. Es el enlace entre el código del programa que se quiera ejecutar y cualquier sistema operativo.

Otra de las ventajas que tiene Java es que tiene un recolector de basura (*garbage collector*), que tiene como función detectar cuando una variable no va a ser utilizada de nuevo dentro del código y por lo tanto libera la memoria de ejecución. Esto ayuda a evitar errores durante la ejecución, ya que existiría la posibilidad de que la memoria se llenase durante la ejecución de un código.

En Java, existen dos conjuntos de herramientas que nos permiten trabajar, el JDK (Herramientas de Desarrollo) y el JRE (Entorno de ejecución).

Por un lado, tenemos la herramienta de desarrollo de Java, es un conjunto de librerías y programas que permiten el desarrollo del lenguaje, es decir, compilar, ejecutar generar documentación… Aquí se encuentran diferentes programas: *Jar*, que sirve para manipular ficheros donde se encuentren clases de Java; el *Javadoc*, que genera la documentación de las clases; y uno de los más importantes, el compilador *Javac*, previamente mencionado. Este último es el encargado de compilar nuestro código y convertirlo a un código *Byte Code*, para que luego la máquina virtual lo vuelva a compilar al sistema operativo.

Por otro lado, tenemos el entorno de ejecución de Java, es el que permite la ejecución de los programas. Está formado principalmente por la máquina virtual y una serie componentes necesarios para la ejecución de los programas. Un usuario solo necesita el JRE para ejecutar las aplicaciones, mientras que para desarrollar nuevas aplicaciones necesitaría el JDK.

Hay varios entornos de desarrollo para Java, los más utilizados son NetBeans, IntelliJ y Eclipse. La que se usará para el desarrollo del proyecto será Eclipse.

Eclipse es una plataforma de desarrollo, diseñada para ser extendida de forma indefinida a través de plug-ins. No tiene en mente un lenguaje específico, sino que es un IDE genérico, aunque goza de mucha popularidad entre la comunidad de desarrolladores del lenguaje Java usando el plug-in JDT (Java Development Tools) que viene incluido en la distribución estándar del IDE.[2]

Proporciona herramientas para la gestión de espacios de trabajo, escribir, desplegar, ejecutar y depurar aplicaciones, que mediante perspectivas, editores y vistas permitirán trabajar de una forma óptima.

5.2. *PROCESSING*

Processing es un lenguaje de programación y entorno de desarrollo integrado de código abierto basado en Java, de fácil utilización, y que sirve como medio para la enseñanza y producción de proyectos multimedia e interactivos de diseño digital.[3]

Desde 2001, Processing ha promovido su software para decenas de miles de estudiantes, artistas o diseñadores. Es totalmente gratuito, interactivo en 2D, 3D o salidas *SVG*, y extendiéndose en más de 100 librerías.

Processing es una plataforma integra donde es muy fácil aprender a programar. Una de sus principales ventajas es que utiliza un método de aprendizaje *plug and play,* como la plataforma Arduino. En castellano se denomina aprendizaje "enchufar, conectar y usar". Este tipo de aprendizaje no necesita ir instalando softwares específicos o configurar parámetros para poder programar. Si se quiere hacer programas complejos utilizando diferentes librerías y clases no se conseguiría tener esa rapidez y esa facilidad que sí tendría si se trabajase directamente con su propio software, donde directamente se compila el código y se ejecuta.

Se puede programar desde programas muy básicos hasta programas muy complejos potenciando la orientación a objetos a un alto nivel, siempre teniendo un código muy bien estructurado.

Tiene su propio entorno de desarrollo, su propio software, aunque también se puede utilizar en otros entornos como Eclipse si el proyecto tiene un alto nivel de complejidad. Se recomienda utilizar su propio software para aplicaciones muy sencillas, y una vez se vaya aumentando el nivel se vaya emigrando a otro entorno más completo para programar. También puede ejecutar aplicaciones en navegador, ya que contiene una Applet en la cual se pueden generar diferentes ejecutables.

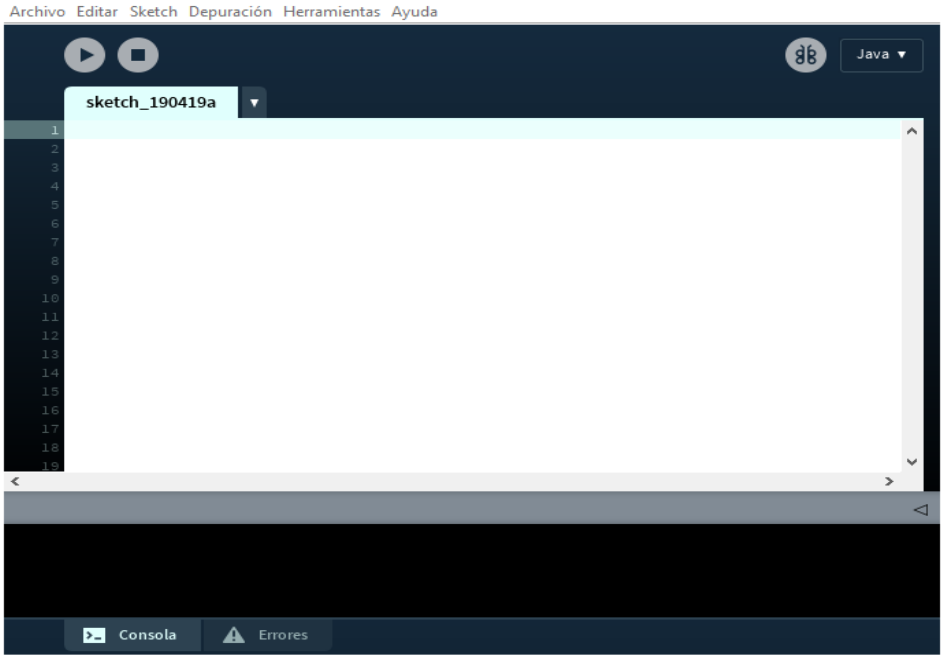


Fig. 1 Software de Processing

Otra característica importante de Processing es la capacidad de combinarse con otras aplicaciones como Java o Arduino. También tiene la posibilidad de generar ejecutables para diferentes plataformas Mac, Windows e incluso aplicaciones móviles para Android.

Como se ha comentado anteriormente, Processing está basado en Java, aunque tiene algunas diferencias. En modo básico se programa sin clases ni funciones y sin tener una estructura, solamente con líneas de código; en modo en modo estructural, donde ya se pueden crear funciones propias y llamarlas posteriormente; y en modo avanzado, donde se trabaja con la orientación orientada a objetos.

Para empezar a programar hay que entender la estructura del programa, consta de tres métodos principales:

- *settings(),* este método solamente se ejecuta una vez, y solamente se puede inicializar variables y utilizar la función *size(),* para dar tamaño a la ventana.
- *setup(),* este método también se ejecutará solo una vez donde ya se podrá ejecutar cualquier función.
- *draw(),* este método se ejecutará infinitamente en bucle siempre que se esté ejecutando el programa

A partir de aquí ya se podría empezar a programar, hay que mencionar la posibilidad de añadir oyentes como los periféricos, como el teclado o el ratón con los cuales podremos actuar directamente con el código.

5.3. *VAADIN*

Vaadin es un framework para desarrollar aplicaciones SPA (Single Page Application) que permiten programar en diversos lenguajes, en las cuales se encuentra Java.[4]

Desde que se empezó a desarrollar aplicaciones web, se ha buscado la forma para que el desarrollo de estas páginas fuese lo más sencillo posible, y Vaadin ha conseguido ese objetivo.

Previamente se ha intentado encontrar esa solución pero hasta Vaadin, no se había conseguido. El framework JSF (Java Server Faces), basado en la API de Servelts MVC (Model View Controller) quedó bastante cerca ya que contenía potentes implementaciones pero difíciles de extender y más limitadas a la hora de componer una aplicación.

Vaadin sí consiguió esa solución, presenta muchas ventajas para un rápido desarrollo de aplicaciones. La principal característica que diferencia a Vaadin del resto, es que presenta una arquitectura centrada con el servidor, al contrario que otros frameworks o librerías de JavaScript. Además, este framework permite extenderse a GWT (Google Web Toolkit). Es una tecnología desarrollada por Google que permite desarrollar las aplicaciones de manera sencilla sin necesidad de plugins en el navegador.

Otras ventajas que tiene Vaadin es la compatibilidad, más allá de la web, con soportes para móviles. También proporciona una arquitectura segura con plugins para diferentes aplicaciones como Eclipse o Intellij o para testear UI's (Diseños de interfaz de usuario).

El entorno de trabajo es muy intuitivo. Está compuesto por un extenso framework de componentes que permiten un fácil *Drag and Drop* (un arrastrar y soltar) en la posición que se desee de la pantalla. Además, también contiene diferentes tipos de layout para el diseño de las páginas. La facilidad y la rapidez con la que se permite diseñar aplicaciones hace de este framework uno de los más potentes que hay.

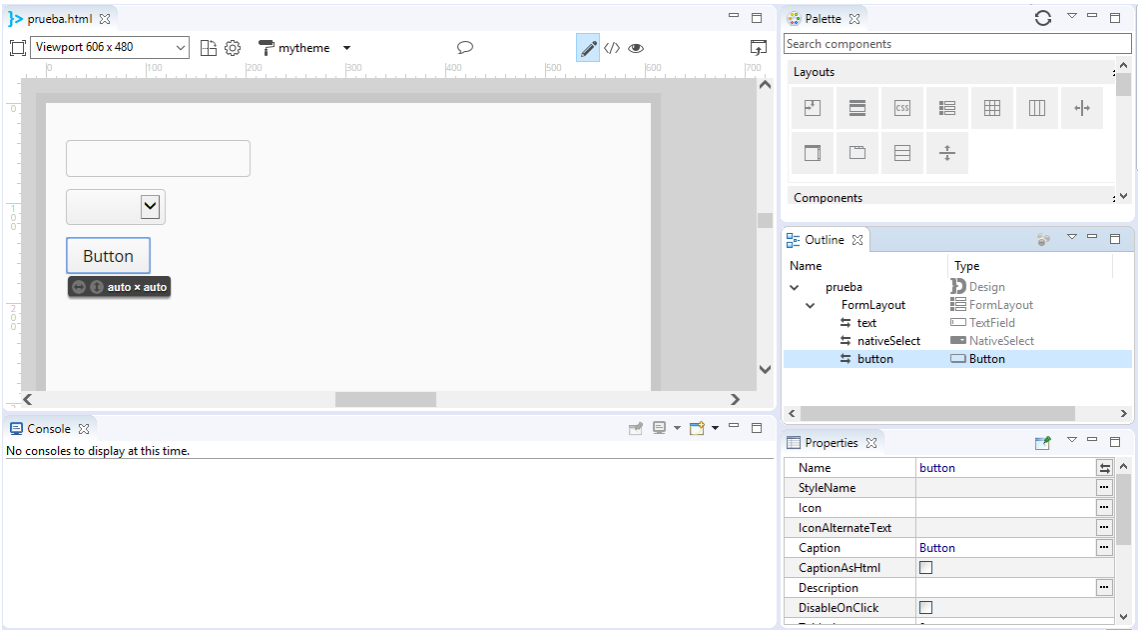


Fig. 2 Entorno de Vaadin Designer en Eclipse.

Vaadin se podría resumir en una palabra, simplicidad. Se reduce líneas de código, por lo tanto, también reduce errores, así como tiempos de desarrollo. El resultado, una aplicación eficiente que necesita poco mantenimiento y con un bajo coste.

5.4. *ARDUINO*

Arduino es la plataforma electrónica más utilizada en prototipos de circuito. Es un micro controlador que mezcla el mundo del software con el del hardware, en el cual se programa de una forma sencilla con un código libre que ha simplificado mucho la forma de trabajar con ellos. Como ambos son libres, se utiliza un código accesible para que cualquiera pueda modificarlo o utilizarlo, de forma gratuita, donde todo el mundo puede crear sus propios programas en función de sus necesidades.

Utiliza un lenguaje de programación basado en C++, utiliza dos métodos principales, *setup(),* el cual solo se ejecuta una vez; y *loop()* o *draw(),* el cual se ejecuta en bucle. Previamente se declaran las variables que se van a utilizar para luego utilizarlas en los métodos posteriores.

Por lo que respecta a los periféricos, se pueden utilizar gran cantidad de ellos, como periféricos de lectura (como cámaras para obtener imágenes o vídeos) hasta todo tipo de sensores; así como salidas, desde pantallas a altavoces.

Las características o especificaciones de una placa Arduino son las siguientes:

- Micro controlador: es el circuito integrado que contiene el programa Arduino, en el cual se programarán previamente las órdenes que ejecutará. Internamente contiene los puertos de entradas y salidas más la CPU. Los más sencillos y más utilizados son el ATmega168 y el ATmega32u4.[5]

- Memoria Flash: es donde se guarda el programa después de compilarlo y cargarlo en la placa. Ese programa queda guardado una vez se apague la placa hasta que se vuelva a cargar otro programa. Suelen ser de 16 o 32 KB.

- SRAM: es la memoria donde, mientras se ejecuta el programa, se guardan las variables locales y calculadas. Al contrario que la Memoria Flash, éstas se pierden cuando se apaga el Arduino. Suelen ser entre 8 y 32 KB.

- EEPROM: guarda los datos constantes una vez se apaga la placa Arduino, es una memoria no volátil como la SRAM, pero más lenta debido a que solo puede leer byte a byte.

- Tensión de trabajo: es la tensión de alimentación a la cual el Arduino hará trabajar a todos sus componentes. Los Arduinos suelen trabajar con 5V, aunque últimamente se están implementando de 3,3V.

- Entradas y salidas analógicas o digitales: es el número de pines que tendrá el Arduino, habrá pines analógicos y pines digitales, y todos se podrán utilizar como entradas o como salidas

5.5. *TAXONOMÍA DE BLOOM*

La taxonomía de Bloom es un conjunto de tres modelos jerárquicos usados para clasificar objetivos de aprendizaje en niveles de complejidad para una herramienta para educar, estos modelos son la dimensión afectiva, la cognitiva y la psicomotora.[6]

Primero tenemos la dimensión afectiva, que es el modo en el que una persona reacciona emocionalmente, en su capacidad para sentir las emociones o sensaciones de otras personas y en cómo expresar sus sentimientos. Consta de 5 niveles: recepción, respuesta, valoración, organización y caracterización. Siendo la recepción el nivel más bajo de complejidad y la caracterización el más alto.

Por otro lado tenemos la dimensión psicomotora, que es la capacidad de las personas para manipular instrumentos físicamente. Sus niveles son percepción, disposición, respuesta dirigida, respuesta automática, adaptación y creación. Siendo la percepción el nivel de complejidad más bajo y la creación el más alto.

Y por último, tenemos la dimensión cognitiva, que es la habilidad que tenemos sobre los estudios para pensar. Con ella se pretende potenciar las competencias del estudiante. Esta es la dimensión en la cual va a basarse la realización de las prácticas, de una manera escalonada, pasando por diferentes niveles de complejidad siempre en un orden cronológico. Los seis niveles son los siguientes:

- NIVEL 1 : Recordar o conocer:

Este nivel trata de recordar sin necesidad de entender. Es decir, tener la capacidad de almacenar en la memoria diferentes tipos de información: teorías, conceptos... Es el nivel más sencillo de alcanzar.

Verbos que se utilizan en este nivel: enumerar, nombrar, describir, identificar, repetir…

- NIVEL 2 - Comprender:

Este nivel trata a parte de recordar la información, entenderla. Se demuestra una comprensión de esa información mediante interpretaciones, traducciones o descripciones.

Verbos que se utilizan en este nivel: resumir, ejemplificar, expresar, localizar…

- NIVEL 3 - Aplicar:

Este nivel trata de utilizar esos conocimientos adquiridos y comprendidos previamente como reglas o técnicas para conseguir resolver problemas en nuevas situaciones.

Verbos que se utilizan en este nivel: calcular, construir, solucionar, establecer, practicar, mostrar…

- NIVEL 4: - Analizar:

Este nivel trata de examinar la información, descomponerla y encontrar la relación entre los componentes. Esto se consigue buscando evidencias y observando las consecuencias de esa información. Elaborar hipótesis y contrastarlas también está dentro de este nivel.

Verbos que se utilizan en este nivel: experimentar, inspeccionar, cuestionar, clasificar…

- NIVEL 5 - Crear:

En este nivel trata de crear algo diferente o proponer soluciones alternativas aplicando el conocimiento y combinando toda la información. Anticiparse a lo que pueda ocurrir y prever los problemas que puedan surgir.

Verbos que se utilizan en este nivel: diseñar, resolver, elaborar, actualizar…

- NIVEL 6 - Evaluar:

Este es el último nivel, el más complejo. Trata de calificar o dar validez a algo o justificar o juzgar una información, siempre en base a unas opiniones o criterios preestablecidos.

Verbos que se utilizan en este nivel: valorar, juzgar, calificar…

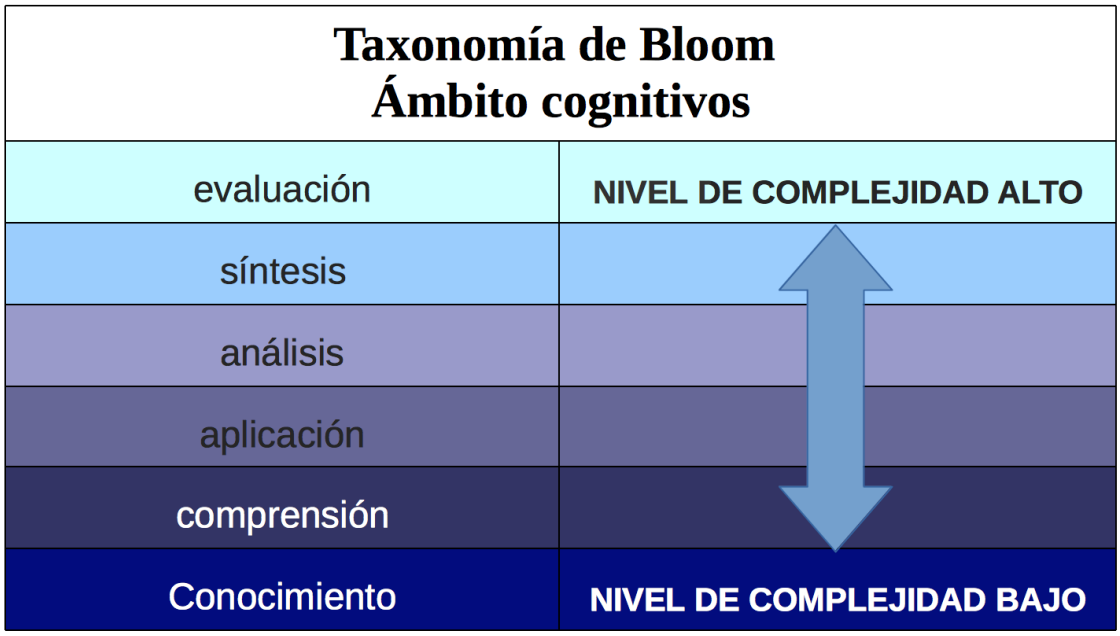


Fig. 3 Niveles de la taxonomía de Bloom en la dimensión cognitiva. [7]

De acuerdo a estos diferentes niveles se ha hecho la realización de las prácticas.

6. DESARROLLO

6.1. *ANTECEDENTES*

Previamente a la realización de las prácticas se hizo un estudio de mercado para la compra de un Arduino. Se miraron diferentes kits con diferentes Arduinos y se eligió el ARDUINO STARTER KIT. En el siguiente apartado se mostrará ese estudio. Con Arduino se pueden hacer muchísimas cosas, sobretodo para procesos automatizados, desde domótica a controlar procesos a pequeña escala. De ese modo se ha diseñado una práctica de domótica para poner en situación a los estudiantes de lo que se puede llegar a hacer.

Por parte de Processing se ha hecho una documentación de lo que se puede llegar a hacer, he hecho varios ejemplos que ofrece la web de Processing. Y a partir de aquí saqué las ideas lo más creativas posibles y diseñé las prácticas con sus implementaciones. Nunca hay solo una solución correcta, siempre hay diferentes maneras de solucionarlas.

Y por último, por parte de Vaadin, primeramente me planteé algunas prácticas que posteriormente ya que al utilizar el trial de Vaadin, no podía acceder a todos los componentes que quería y tuve que buscar nuevas ideas.

De esta manera, teniendo la base después de haberme documentado y ver hasta que punto podía trabajar me puse manos a la obra a crear prácticas originales donde los alumnos se diviertan a la vez que aprende.

6.2. *PLANTEAMIENTO Y DECISIÓN SOBRE SOLUCIONES ALTERNATIVAS*

El proyecto consta de trece prácticas, repartidas en siete de Processing, una de Arduino y cuatro de Vaadin. Estas prácticas se han realizado, como bien se ha comentado, basándose en la Taxonomía de Bloom. Como se ha explicado en el apartado anterior cogí unas ideas haciendo ejemplos e informándome para después tener la capacidad de hacer prácticas nuevas y originales.

La primera práctica de cada plataforma son prácticas introductorias en las cuales se explica como crear un proyecto y el código principal para hacer un programa muy sencillo. Por parte de Processing, se explicará como importar las librerías de Processing desde Eclipse para trabajar con ella. La práctica de Arduino es simplemente introductoria explicando como se programa en Arduino. Y por último, por parte de Vaadin, se explicará como crear un diseño y como ejecutarlo, además de explicar en el manual de instalación de Vaadin Designer la instalación de este.

Los enunciados de las prácticas se encuentran en el ANEXO A de la memoria, y las soluciones en el ANEXO B. Además, se podrá encontrar los enunciados de las prácticas y los archivos de las soluciones en una carpeta compartida.

https://drive.google.com/open?id=1-DIEboc26nhb4imTDuZHFiuZ7K1gMc1-

Además de la realización de los enunciados y la implementación de las soluciones, se ha realizado, a parte del manual de instalación de Vaadin Designer indicado anteriormente, algo más de documentación. Por un lado, se ha realizado un manual de Arduino donde se explica la forma de programar en él y como se debe montar la placa para poder trabajar. Y por último, un manual de conceptos básicos de las prácticas indicando de que tratan, que clases y que funciones nuevas se van utilizar.

Estos dos últimos manuales serán para el profesorado y no para los alumnos, de esta manera el profesor sabrá que es lo que tiene que explicar en cada práctica, y por otro lado, también podrá dejar el Arduino preparado para las clases para que los alumnos no pierdan el tiempo montándolo y se dediquen a programar. Los manuales y la documentación para el profesorado se encuentran en el ANEXO C de la memoria. Estos documentos también se encontrarán el la web previamente mencionada.

6.3. *ESTUDIO DE MERCADO ARDUINO*

Para la realización de algunas de las prácticas de Processing se va a necesitar un Arduino y algunos componentes. Previamente se ha buscado el Arduino más adecuado para la realización de las prácticas.

Primeramente, hay que diferenciar entre los diferentes tipos de Arduinos que hay en el mercado. Estos vienen diferenciados por el tamaño, la conectividad, el número de pines de entradas y salidas y la cantidad de memoria interna para programas que requieran mucho código, cuanta más capacidad y más salidas, mayor es el precio del *Arduino*. Actualmente hay quince tipos de Arduinos en el mercado.

Arduino	Micro controlador	Tensión (V) trabajo	Pines I/O Digitales	Pines I/O análogas	Sram (KB)	Memoria Flash (KB)
PRO	ATmega168	3.3	14	6	1	16
PRO MINI	ATmega168	3.3	14	8	1	16
NANO	ATmega168	5	14	8	1	16
UNO	ATmega328	5	14	6	2	32
ESPLORA	ATmega32u4	5	0	0	2.5	32
ROBOT	ATmega32u4	5	4	8	2	32
ETHERNET MEGA 2560	ATmega328	5	14	6	2	32
MINI	ATmega328	5	14	8	2	32
FIO	ATmega328P	3.3	14	8	2	32
LEONARDO	ATmega32u4	5	20	12	2	32
YUN	ATmega32u4	5	20	12	2.5	32
MICRO	ATmega32u4	5	20	12	2.5	32
MEGA 2560	ATmega2560	5	54	16	8	256
MEGA ADK	ATmega2560	5	54	16	8	256
DUE	AT91SAM3X8 E	3.3	54	12	96	512

Tabla 1 Comparación de tipos de Arduino para la venta al público en Noviembre de 2018

Como podemos observar en la tabla, la primera gran diferencia es el microcontrolador, el cual te indica la Memoria Flash del *Arduino*, con la cual nosotros tendríamos suficiente

con uno de 32 KB. Por otro lado, tenemos el número de entradas y salidas, que va a marcar nuestra elección. Con catorce pines I/O digitales y seis pines de entradas analógicas sería suficiente. Por lo tanto el Arduino que más se adapta a la realización sería el Arduino Uno, además, bastante económico.

Para la búsqueda de material o dispositivos electrónicos para *Arduinos*, se ha consultado en varias webs de comercialización. Cetronic, Electan y BricoGeek han sido las que ofrecían los productos originales a buen precio. Se puede comprar los Arduinos y los materiales por separado, pero no es una opción muy económica; por suerte, existe la posibilidad de comprar kits, los cuales incluyen una placa Arduino y diverso material de interés, como pueden ser leds, interruptores, sensores, etc. Aquí comparamos las tres diferentes opciones de kit que aparecen en las tres previas webs de comercialización.

- KIT BASICO ARDUINO en Electan.[8]

Su precio es de 43.30€ y contiene:

- 1 x Arduino UNO R3
- 1 x cable USB
- 1 x placa protoboard
- 25 x cables Jumper
- 1 x mini servo
- 1 x zumbador
- 1 x potenciómetro
- 1 x caja de plástico Verde
- 4 x cajas de plástico azul pequeñas
- 5 x leds rojos
- 5 x leds verdes
- 10 x condensadores cerámicos 10nF
- 10 x condensadores cerámicos 100nF
- 5 x condensadores electrolíticos de 100uF
- 10 x resistencias 330 Ω
- 10 x resistencias 10 K Ω
- 1 x sensor de inclinación
- 1 x termistor
- 1 x foto resistencia
- 1 x diodo
- 5 x pulsadores

- 5 x interruptores

- KIT ARDUINO AVANZADO en BricoGeek.[9]

Su precio es de 95,00€ y contiene:

- 1 x placa Frearduino UNO
- 1 x pantalla TFT1.8 serial (compatible con alimentación 3.3V y 5V)
- 1 x teclado de 5 botones
- 1 x zumbador
- 1 x sensor de gas MQ3
- 1 x módulo MP3
- 1 x sensor de color
- 1 x sensor inercial 9 DOF
- 1 x módulo Wifi ESP8266
- 1 x adaptador Bluetooth BLE
- 1 x relé de 1 canal
- 1 x porta pilas (pilas no incluidas)
- 1 x cable USB
- 30 x cables de conexión

- ARDUINO STARTER KIT en Cetronic (Componentes eléctricos).[10]

Su precio es de 88.95€ y contiene:

- 1 x Arduino Projects Book
- 1 x Arduino UNO Rev3
- 1 x cable USB
- 1 x placa protoboard de 400 puntos
- 1 x adaptador para la batería de 9 Voltios
- 70 x puentes de conexión para la protoboard
- 6 x foto resistencias (VT90N2 LDR)
- 3 x potenciómetros de 10K
- 10 x pulsadores
- 1 x sensor de temperatura (TMP36)
- 1 x sensor de inclinación
- 1 x LCD alfanumérico (16x2 caracteres)
- 1 x LED (Blanco)

- 1 x LED (RGB)
- 8 x LED (Rojo)
- 8 x LED (Verde)
- 8 x LED (Amarillo)
- 3 x LED (Azul)
- 1 x motor CC 6 y 9 Voltios
- 1 x servo motor
- 1 x zumbador
- 2 x opto acopladores 4N35
- 5 x transistores BC547
- 2 x mosfet IRF520
- 5 x condensadores 100nF
- 3 x condensadores 100uF
- 5 x condensadores 100pF
- 5 x diodos 1N4007
- 1 x tiras de pines macho (40x1)
- 20 x resistencias 220 Ω
- 5 x resistencias 560 Ω
- 5 x resistencias 1K Ω
- 5 x resistencias 4.7K Ω
- 10 x resistencias 10K Ω
- 5 x resistencias 1M Ω
- 5 x resistencias 10M Ω

Tabla comparadora de los Kits de Arduino:

KIT	BASICO ARDUINO	ARDUINO AVANZADO	ARDUINO STARTER
Comercial	Electan	BricoGeek	Cetronic
Arduino	UNO R3 con cable USB y protoboard	Frearduino UNO con cable USB	UNO R3 con cable USB y protoboard
Leds	5 rojos 5 verdes 1 RGB	0	1 blanco 8 rojos 8 verdes

			8 amarillos 3 azules 1 RGB
Resistencias	10 x 330 Ω, 10 x 10K Ω	0	20 x 220 Ω, 5 x 560 Ω, 5 x 1K Ω, 5 x 4.7K Ω, 10 x 10K Ω, 5 x 1M Ω, 5 x 10M Ω
Condensadores	10 cerámicos 10nF, 10 cerámicos 100nF, 5electrolíticos 100uF	0	3 de 100uF, 5 de 100nF, 5 de 100pF
Otros componentes electrónicos	5 interruptores, 5 pulsadores, 1 termistor, 1 foto resistencia, 1 mini servo, 1 zumbador, 1 diodo, 1 potenciómetro	1 relé de 1 canal, 1 zumbador	10 pulsadores, 6 foto resistencias, 3 potenciómetros de 10K, diodos 1N4007, 1 motor CC 6 y 9 V, 1 servo motor, 1 zumbador, 2 opto acopladores 4N35, 5 transistores BC547, 2 mosfet IRF520
Sensores	1 de inclinación	1 de color, 1 inercial 9 DOF, 1 de gas MQ3	1 de temperatura (TMP36), 1 de inclinación

Extras	25 cables Jumper	1 módulo Wifi ESP8266, 1 adaptador Bluetooth BLE, 1 pantalla TFT1.8 serial, 1 teclado de 5 botones, 1 módulo MP3, 30 cables conexión, 1 porta pilas	1 LCD alfanumérico, 70 puentes de conexión, 1 libro de proyectos para Arduino
Precio	43.30€	95.00€	88.95€

Tabla 2 Comparación Kits de Arduino.

Como podemos observar en la tabla, la primera opción trata de un kit simple bastante económico con un Arduino Uno R3 y varios componentes electrónicos como leds e interruptores. Es una opción con muy poca variedad de materiales y muy limitado, apenas un sensor de inclinación con el que poder hacer aplicaciones.

En la segunda opción, con el precio más elevado de las tres opciones, tenemos un kit con un Frearduino Uno. Este Arduino es casi idéntico al anterior, la una diferencia es que en los pines para entradas y salidas se permite la salida y entrada a diferentes tensiones (3.3V y 5V) para facilitar la conexión con periféricos como sensores o servos. El kit también incluye una gran variedad de materiales un poco más sofisticados para un nivel más avanzado, como pueden ser un sensor de gas, un módulo Wifi ESP8266 o un adaptador Bluetooth.

Y por último, tenemos un kit con un Arduino UNO R3, el mismo que en la primera opción. A diferencia del primer kit, este contiene más variedad de material, como un sensor de temperatura, un LCD alfanumérico… En definitiva, bastante más completo, pero sin llegar al nivel de dificultad de la segunda opción, la cual contenía material con un nivel bastante avanzado respecto a conocimientos.

En conclusión, podemos decir que el primer dispositivo es económico pero es bastante simple e incluye poco material para poder trabajar, por lo tanto no es una buena opción; por otro lado, la segunda, todo lo contrario, poco económico y con un material de uso más complejo, lo cual no serían las más idóneas para muchas de las prácticas si seguimos la taxonomía de Bloom, explicada en el siguiente apartado. Por estas razones, la última opción es la más adecuada para este caso: Tiene una gran variedad de material con el que podemos trabajar bien sin muchas dificultades. Por lo tanto, el Arduino que se ofrece es el ARDUINO STARTER KIT del comercial Cetronic (Componentes eléctricos) por 88,95€.

6.4. *DESARROLLO DE LAS SOLUCIONES ESCOGIDAS*

6.4.1. *PRÁCTICAS*

6.4.1.1. *PROCESSING*

6.4.1.1.1. PRÁCTICA 0: INTRODUCCIÓN A PROCESSING

Esta es la primera práctica de Processing, donde los alumnos crearán un proyecto *Java* e importarán las librerías de Processing y crearán su primer programa "Hola Mundo!".

Los objetivos de la práctica son:

- Importar las librerías de Processing para trabajar en Eclipse.
- Aprender cómo crear un proyecto Processing.
- Aprender el lenguaje básico para programar en Processing.

Previamente los alumnos deberán descargar las librerías necesarias desde la web de Processing. https://processing.org/download/

Una vez se hayan seguido todos los pasos y se hayan importado esas bibliotecas ya estarán listos para empezar a programar. En la práctica se explicará el lenguaje básico de Processing y como desarrollar una aplicación muy sencilla. Finalmente, se creará un primer programa "Hola mundo" donde se mostrará un mensaje por pantalla.

Esta es una práctica de nivel 1 de complejidad, en la taxonomía de Bloom. En la práctica se explican todos los pasos que hay que seguir y se muestran uno a uno, simplemente deberán de seguir las instrucciones. Los alumnos deben recordar todo lo explicado en la práctica ya que estos conocimientos aprendidos se deberán utilizar en el resto de prácticas de Processing. Una vez acabada la práctica, los alumnos deberán ser capaces de recordar los pasos y conocimientos que se han adquirido de cara a las próximas prácticas.

6.4.1.1.2. PRÁCTICA 1: PRIMEROS PROGRAMAS

En esta segunda práctica de Processing se pretende introducir al alumnado en el mundo de Processing, enseñándole diferentes funciones para que las vayan conociendo.

Los objetivos de la práctica son:

- Trabajar con clases y herencia.
- Crear diferentes formas geométricas utilizando las funciones 2D primitivas de Processing.
- Aprender cómo crear un proyecto Processing.
- Aprender el lenguaje básico para programar en Processing.
- Trabajar con métodos continuos.
- Crear movimiento de objetos por pantalla.
- Mostrar imágenes por ventana.
- Utilizar eventos desde el teclado.
- Utilizar eventos desde el ratón.

Esta práctica está dividida en tres apartados:

En el primer apartado, los alumnos deberán crear una cara sonriente con diferentes formas geométricas. Se utilizarán círculos para los ojos, un rectángulo para la nariz y un triángulo para la boca. Se deberán de crear las clases necesarias para realizarlas.

En el segundo apartado, se creará un programa en el cual se visualizará el movimiento a una línea. Haciendo clic al botón izquierdo del ratón, se pondrá en marcha o se parará el movimiento. El movimiento de la línea no deberá parar por el mismo. La línea de desplazará verticalmente hacia arriba de manera que cuando llegue a la parte superior de la pantalla vuelva a aparecer por la parte inferior y de esta manera el movimiento sea continuo.

Y finalmente, en el último apartado, los alumnos deberán dar movimiento a una imagen a través del teclado. Pulsando diferentes teclas del teclado la imagen se moverá en una dirección o en otra.

Esta práctica está dentro del nivel 2 de complejidad en la taxonomía de Bloom. Se necesitará comprender y entender lo aprendido en la primera práctica. En esta se explicarán conceptos nuevos y funciones nuevas, pero se necesitará recordar la base aprendida en la práctica anterior. Los alumnos irán siguiendo la práctica y necesitarán haber comprendido toda la información para poder realizarla.

6.4.1.1.3. PRÁCTICA 2: PAINT

En esta práctica se va a simular la creación de un Paint. Se creará, en la parte superior de la pantalla, una zona donde no se permitirá dibujar y donde podremos elegir si dibujar o borrar y donde daremos tamaño a la línea de dibujo.

- Los objetivos de la práctica son:
- Conocer diferentes funciones de Processing.
- Crear programas entretenidos.
- Simular un Paint.
- Estructurar bien las diferentes zonas de la pantalla.
- Trabajar con las variables mouse X y mouse Y.

Constará de tres apartados, el primero, en el cual solamente permitiremos dibujar a través de líneas. Se utilizarán las variables de *Processing*, mouse X y mouse Y las cuales nos indicarán en qué posición de la pantalla se encuentra el cursor.

En el segundo apartado se creará un rectángulo dentro de la parte superior de la pantalla donde se podrá elegir que grosor de línea se quiere para dibujar. Aparecerán tres opciones diferentes.

Y en el último apartado, se creará la simulación de la "goma" de Paint para borrar lo dibujado. Se volverá a utilizar el espacio de la parte superior de la pantalla, para que el dibujante pueda seleccionar si dibuja o borra.

En esta práctica se volverá a trabajar en el nivel 2 de complejidad en la taxonomía de Bloom. Se introducirán nuevas funciones, pero será suficiente solo con recordarlas, los alumnos deberán entenderlas para posteriores prácticas.

6.4.1.1.4. PRÁCTICA 3: PELOTA SALTARINA

En esta práctica se dará movimiento a una esfera que irá moviéndose por pantalla y reaccionando a los choques con las paredes y con los cruces con otras esferas.

Los objetivos de esta práctica son:

- Utilización de otras *librerías* de Java.
- Conocer la clase *PVector* de Processing.
- Tener conocimientos matemáticos de Vectores.

Primeramente, se va a crear una esfera que vaya continuamente moviéndose por la pantalla. La dirección será aleatoria, es decir, una vez se ejecute la esfera se desplazará en cualquier dirección. No siempre irá en esa dirección ya que en el momento que esa esfera rebote con la pared rebotará, es decir, simulando el choque de una pelota en una pared. Por ejemplo si la esfera choca con la pared izquierda mientras estaba desplazándose hacia arriba, la esfera continuará desplazándose hacia arriba pero esta vez buscando la pared derecha.

Para darle la dirección correcta en cada momento a la esfera se utilizará la clase de Processing, *PVector*, la cual contiene diferentes funcione que permitirán un desplazamiento correcto de la esfera. También se dará color y radio a la esfera de manera aleatoriamente.

En el segundo apartado, se introducirá otra esfera en la pantalla y se añadirá alguna dificultad más. En el momento que las dos esferas se crucen, no chocarán, ya que conseguir la dirección en la que las esferas se crucen es una tarea más complicada y se perdería bastante tiempo aquí. Por esta razón cuando las esferas se crucen, ambas cambiarán de color aleatoriamente otra vez.

Está práctica estaría dentro del nivel 3 de complejidad de la taxonomía de Bloom. Este nivel también incluye los dos anteriores, ya que se necesitará recordar y comprender la información aprendida hasta aquí. Pero, además, en esta práctica se necesitará calcular direcciones y dimensiones, por lo tanto, aquí sí que se necesita aplicar unos conocimientos y utilizar unas leyes y datos, en este caso, fórmulas. Para esta parte de la práctica se necesitará calcular las distancias que hay entre las esferas y la pared para determinar cuándo hay un choque con pared o un cruce con otra esfera. El choque entre esferas con su posterior variación de la dirección que tomarían las esferas hubiese complicado demasiado la práctica.

6.4.1.1.5. PRÁCTICA 4: ENCUENTRA LOS ELEMENTOS.

En esta práctica los alumnos simularán un juego llamado Encuentra los elementos, donde el jugador deberá encontrar diferentes objetos escondidos en diferentes imágenes. Los alumnos buscarán imágenes en las cuales aparezcan diferentes elementos y elegirán varios de ellos en cada imagen.

Los objetivos de la práctica son los siguientes:

- Trabajar con arrays de imágenes.
- Aprender a seleccionar elementos aleatoriamente dentro de un array.
- Mostrar por pantalla las coordenadas del puntero del ratón.
- Controlar los márgenes de la pantalla de manera que las imágenes siempre queden centradas.

Habrá un primer programa donde se mostrará una imagen por pantalla y cuando hagamos clic con el botón izquierdo del ratón en cualquier punto de la pantalla se mostrarán las coordenadas del puntero del ratón en ese momento por la consola. Estas coordenadas nos servirán para determinar el área en el cual se encuentran los elementos. Es necesario determinar el área de estos para cuando se ejecute el juego, el jugador encuentre el elemento y lo seleccione.

En la segunda parte de la práctica se creará el programa que simule el juego. Se mostrarán las diferentes imágenes por pantalla una a una. Se seleccionarán cinco de los nueve elementos de cada imagen aleatoriamente y se mostrará el nombre de estos por pantalla. De esta manera el jugador verá el nombre de los elementos que tendrá que encontrar. Cada vez que el jugador clique en la pantalla el programa comprobará si ese punto de la imagen pertenece al área de alguno de los elementos pendientes. Si se clica dentro del área de alguno de los elementos pendientes este elemento habrá sido encontrado. Cuando esto ocurra, se eliminará el nombre del elemento de la lista de nombres pendientes de encontrar y se señalizará en la misma imagen con un círculo verde. En caso de no acertar, se acumulará el número de fallos en una variable, y cuando esta exceda del permitido, el jugador habrá perdido la partida.

Por otro lado, si el jugador encuentra todos los elementos de una imagen, se irá a por la siguiente. Una vez se haya encontrado todos los elementos de todas las imágenes el jugador habrá ganado el juego, también se mostrará un mensaje por pantalla indicándolo.

Esta práctica se encuentra dentro del nivel 3 de complejidad en la taxonomía de Bloom como en la práctica anterior, pero en esta se va un paso más allá a nivel de dificultad y complejidad. Al estar en el nivel tres la práctica cumple con su nivel y con los anteriores,

es decir, recordar, comprender y aplicar. Se pautan varios métodos necesarios para el desarrollo de la práctica los cuales los alumnos necesitarán pensar bien como harán el programa. Se necesitará hacer un buen análisis antes de empezar a programar.

El alumno deberá recordar toda la información adquirida y entendida, y aplicarla en la práctica para poder llegar a resolverla.

6.4.1.1.6. PRÁCTICA 5: FLAPPY BIRD

Esta es una práctica bastante sencilla. Se creará un mini juego parecido a un famoso juego llamado Flappy Bird con alguna variante.

Los objetivos de la práctica son los siguientes:

- Crear un programa más dinámico en el que poder entretenerse
- Aumentar la velocidad de movimiento de imágenes
- Recordar funciones de Processing explicadas previamente.

Se deberá crear un programa en el cual aparezca una línea horizontal desplazándose de abajo a arriba continuamente. En el momento que se clique a la pantalla el movimiento de la línea se pondrá en marcha. Pero, esta línea tendrá una separación entre medias, es decir, estará formada por dos líneas separadas por un pequeño hueco. Este hueco irá variando aleatoriamente cada vez que la línea llegue a la parte superior de la pantalla y aparezca por la parte inferior para volver a desplazarse para arriba.

También se deberá crear un pequeño icono o una pequeña imagen como "muñeco", el cual estará fijo en el eje de la Y pero permitirá moverse en el eje de la X mediante el cursor del ratón. El jugador deberá intentar colocar el icono en una posición de manera que cuando la línea llegue a su altura, la imagen pase por el hueco y la línea no toque la imagen.

En el momento que la línea toca la imagen, se detiene el juego ya que el jugador ha perdido. La velocidad de desplazamiento de la línea irá incrementándose en función del número de veces que el jugador ha conseguido esquivar la línea. De esta manera el juego se irá complicando hasta un punto donde el jugador ya no será capaz de esquivar esa línea.

Esta práctica se encuentra dentro del nivel 2 de complejidad en la taxonomía de Bloom. Requiere de conocer unos conocimientos aprendidos en la práctica 1 y entenderlos para poder desarrollar la práctica sin problemas.

6.4.1.1.7. PRÁCTICA 6: PUZZLE

La práctica consiste en crear un Puzzle. Se creará un programa el cual al ejecutarlo aparecerán diferentes piezas de una imagen y habrá que colocarlas en un recuadros para completar la imagen.

Los objetivos de la práctica son los siguientes:

- Dividir una imagen en varias piezas.
- Trabajar con Arrays de objetos.
- Comunicar Arduino con Eclipse.
- Enviar datos desde Arduino al programa de Eclipse

Esta práctica está dividida en tres apartados.

El primero de ellos será hacer un programa el cual divida una imagen en varias piezas. La imagen tendrá que ser cuadrada, es decir, que tenga el mismo número de pixeles tanto de alto como de ancho. El número de piezas en el que se querrá dividir la imagen tiene ser un número que sea cuadrado de otro número, como podrían ser 4, 9, 16, 25… De esta manera cada imagen estaría formado por ese número de piezas que formarían la imagen completa de forma 2x2, 3x3, 4x4, 5x5…

Processing tiene una forma de dividir una imagen en varias. Trata de mostrar por pantalla la parte de la imagen que se quiera mostrar, es decir la pieza. Si se muestra la imagen en unas coordenadas negativas y en tamaño de la pantalla es del tamaño que se quiere que sean las piezas, en la pantalla solo se mostrará esa pequeña parte de la pieza que se quiere ver, por lo tanto, en ese momento, se hará un guardado de pantalla. Haciendo un guardado, el programa creará una imagen con lo que se vea por pantalla en ese momento. De esta manera, haciendo un método que vaya mostrando las zonas de la imagen que se desee y guardándolas en cada momento se consigue dividir la imagen en un número determinado de piezas.

El segundo apartado consiste en crear el Puzzle. Se creará un programa que muestre por pantalla todas las imágenes creadas y una zona con varios recuadros donde deberán ir esas piezas. Mediante el ratón se seleccionará la imagen que quiere moverse, y mediante las teclas del teclado se moverán en las cuatro direcciones. Las imágenes saldrán desordenadas y se deberá colocar cada imagen en su posición correcta, por lo tanto se creará un array de objetos el cual tendrá como atributos las imágenes y sus coordenadas finales. Una vez se mueva una imagen hasta su coordenada cuando esta esté lo suficientemente cerca, se colocará. Cuando se coloquen todas las imágenes se habrá completado el Puzzle.

Este último apartado tiene el mismo funcionamiento que el anterior. Se creará el Puzzle igual pero el movimiento de las piezas será a través del Arduino y no a través del teclado.

Arduino tendrá cuatro interruptores para poder mover la imagen en las cuatro direcciones. Mediante el programa se comunicará con el Arduino y cada vez que se pulse un interruptor del Arduino, este enviará un *char* al programa, el cual estará leyendo para ejecutar la orden de mover la imagen en una determinada dirección. Y finalmente, una vez estén todas las piezas colocadas se habrá terminado el Puzzle.

Esta práctica estaría dentro del nivel 5 de complejidad en la taxonomía de Bloom. Para realizar la práctica los alumnos deberán recordar toda la información aprendida, haberla comprendido y haberla aplicado en las prácticas anteriores. Pero además, en esta práctica, los alumnos deberán compilar toda esa información de diferentes modos combinando elementos aprendidos para proponer una solución alternativa.

Los alumnos deberán combinar ideas, prever errores que puedan salir y aplicar las habilidades aprendidas para producir esta práctica, una práctica original y complicada.

6.4.1.1.8. PRÁCTICA 7: DOMÓTICA

En esta práctica se va a simular la domótica de una casa: el aire acondicionado o calefacción y la iluminación.

Los objetivos de la práctica son los siguientes:

- Trabajar con entradas analógicas en el Arduino.
- Enviar datos desde el programa de Eclipse al Arduino.
- Poner a prueba todos los conocimientos adquiridos.

Esta práctica está dividida en tres apartados.

Esta práctica tiene dos apartados. El primero, el control de un aire acondicionado de una casa. Mediante el Arduino y un sensor de temperatura TMP-36 se leerá la temperatura ambiental. Por comunicación serie, se enviará el dato al programa de Processing donde mediante unos valores prefijados se comparará la temperatura actual. Cuando la temperatura supere el límite superior deberemos "activar" el aire acondicionado y cuando la temperatura sea más baja que el límite inferior se "desactivará" el aire. Habrá que implementar por pantalla un *SCADA* (Supervisory Control And Data Acquisition) donde se vean la temperatura actual, las dos temperaturas prefijadas y las señalizaciones de cuando se activa el aire acondicionado.

En el segundo apartado se simulará la iluminación de una casa. En la pantalla se dibujará un esquema de una casa con 4 o 5 habitaciones. También se mostrará una persona o un icono el cual se irá moviendo por la casa. Aparte, se hará un pequeño montaje en el Arduino simulando las habitaciones, un led por cada habitación. En el momento que la "persona" entre en una habitación, el programa enviará una señal al Arduino para indicar que alguna persona ha entrado en esa habitación, por lo tanto, se encenderá la luz de la habitación se encenderá el *LED*). En el momento que se salga de la habitación la luz se apagará.

Esta práctica también estaría dentro del nivel 5 de complejidad en la taxonomía de Bloom, como la anterior. Son las últimas prácticas de Processing y las más complejas. Se busca que el alumno sea capaz de crear un pequeño sistema de iluminación y el funcionamiento de un aire acondicionado. Para ello deberá poner sobre la mesa todos sus conocimientos de Processing, aplicarlos y buscar soluciones al planteamiento inicial.

6.4.1.2. *ARDUINO*

Solamente hay una práctica de Arduino, que deberá realizarse antes de las prácticas 6 y 7 de Processing, ya que en ellas se utiliza el Arduino y se trabaja con él. Una vez realizada esta práctica se podrá hacer sin problema las dos prácticas ya mencionadas.

6.4.1.2.1. PRÁCTICA 0: INTRODUCCIÓN AL ARDUINO

En esta práctica se hará una pequeña introducción al Arduino. Se explicará cómo instalar el software y cómo es el lenguaje para programar.

Los objetivos de la práctica son los siguientes:

- Instalar el Software de Arduino para trabajar en *Eclipse*.
- Aprender el lenguaje básico para programar en Processing.

En la práctica se explicará paso por paso como descargar e instalar el software del Arduino para poder hacer un programa y descargarlo en la placa. Se mostrará el montaje que habrá que hacer.

Esta práctica estaría dentro del nivel 1 de complejidad en la taxonomía de Bloom. En la práctica se explican todos los pasos uno a uno para descargar e instalar el software del Arduino. Los alumnos solamente deberán seguir esas instrucciones. Los alumnos deberán recordar todo lo que se explique en la práctica ya que los conocimientos aprendidos en esta práctica se utilizarán en las dos prácticas de Processing que se utiliza el Arduino.

6.4.1.3. *VAADIN*

Por parte de *Vaadin* hay 4 prácticas, la primera introductoria de nivel 1, dos más de nivel y la última de nivel 3. Los alumnos deberán haber realizado la instalación de Vaadin Designer en Eclipse antes de la realización de las prácticas.

6.4.1.3.1. PRÁCTICA 0: EJECUCIÓN PRIMER DISEÑO VAADIN

En esta práctica se creará un primer diseño de *Vaadin*. Con el manual ya se habrá explicado cómo instalar el plugin de Vaadin y como crear y ejecutar una clase.

En esta práctica los objetivos son los siguientes:

- Conocer los diferentes tipos de layouts.
- Utilizar diferentes tipos de layouts.
- Diseñar una clase con Vaadin Designer.
- Ejecutar una clase diseñada por Vaadin Designer.

El diseño será cuatro *TextFields* colocados en la pantalla. Estarán colocados dos arriba y dos abajo. Se deberá de crear un programa de manera que lo que se escriba en el *TextField* superior izquierda se escriba en el inferior derecho; y por otro lado, lo que escriba en el superior derecho que se escriba en el inferior izquierdo. En los *TextFields* inferiores no se permitirá escribir.

Esta práctica estará dentro del nivel 1 de complejidad en la taxonomía de Bloom. En la práctica se explica cómo se diseñará la clase y como se declarará y se mostrará el diseño. Los alumnos deberán seguir los pasos simplemente y deberán recordar la información para la siguiente práctica.

6.4.1.3.2. PRÁCTICA 1: FORMULARIO

En la práctica se hará un formulario como los que aparecen en muchas páginas web.

En esta práctica los objetivos son los siguientes:

- Utilizar componentes teóricos como *TextField*, *Button* y *Label*.
- Crear un formulario.
- Utilizar un *TextArea* para mostrar los datos recogidos.
- Validar datos de un formulario.

Primeramente se deberá diseñar la página. Se colocarán 3 *TextFields* para escribir el nombre y los apellidos, un *ComboBox* para elegir la nacionalidad, un *RadioButtonGroup* para elegir el género y un *inLineDateField* para seleccionar la fecha de nacimiento. Aquí se introducirán los datos todos los datos.

Hay tres apartados, en los cuales se irá añadiendo cosas. En el primero simplemente se mostrará el formulario y el usuario rellenará los campos.

En el segundo apartado se añadirá un *TextArea* y un *Button*. Cuando se apriete al botón se mostrará una descripción de la persona que ha introducido sus datos en el área de texto. Se recogerán los datos de los componentes y se escribirá en el área de texto.

Finalmente, en el tercer apartado de la práctica se validarán los datos introducidos. Si algún dato introducido no es correcto, no se permitirá hacer la descripción y mostrará un mensaje con un *Label* donde se informará de porque no se permite, es decir, que dato es incorrecto y porque. Nombre y apellidos serán correctos siempre que tengan un mínimo de 3 caracteres, por lo tanto, no permitirá ningún campo vació o con uno o dos caracteres. La nacionalidad y el género serán correctos siempre que se haya seleccionado alguna opción, es decir, si no se selecciona ninguna opción, no se permitirá hacer la descripción. Y por último, la fecha de nacimiento será correcta siempre que la fecha elegida no sea posterior a la fecha actual.

Cuando se cumplan todas las restricciones se permitirá hacer la descripción de la persona con los datos introducidos.

Está práctica está dentro del nivel 2 de complejidad en la taxonomía de Bloom. Los alumnos deberán comprender la información aprendida en la práctica anterior para la realizar la práctica. Realizando la práctica, los alumnos demostrarán haber comprendido lo aprendido.

6.4.1.3.3. PRÁCTICA 2: NOTICIAS

En la práctica se simulará un muro de cualquier red social o cualquier diario online donde se redacta una noticia o simplemente una frase y se publica en un muro de noticias.

En esta práctica los objetivos son los siguientes:

- Crear la simulación de un muro de noticias.
- Limitar el número de caracteres de los *TextFields*.
- Añadir y eliminar componentes de un layout.

La práctica es muy sencilla, primeramente se diseñará la clase donde se añadirá un *TextField* y un *Button* en la parte izquierda de la pantalla. En el área de texto se escribirá lo que se quiera escribir y con el botón se publicará en la parte derecha de la pantalla.

Cada noticia que se publique deberá aparecer en primer lugar, es decir, las noticias antiguas irán desplazándose hacia la parte inferior de la pantalla. Se limitará el número de noticias para que cuando se llegue a ese límite, la noticia más antigua desaparecerá a favor de la nueva que se publique.

También se deberá limitar el número de caracteres del área de texto para que no se supere el ancho de la pantalla y solo se publicará la noticia si hay algo en área de texto, si no hay nada y pulsamos el botón de publicar no publicará nada.

Esta práctica también se encuentra en el nivel 2 de complejidad en la taxonomía de Bloom. Los alumnos deberán haber comprendido la información de la práctica 0 para poder realizar esta.

6.4.1.3.4. PRÁCTICA 3: REGISTRO

Esta es la práctica más complicada de Vaadin. Se creará un registro de usuario donde se las personas se registrarán y posteriormente podrán iniciar sesión para modificar algún dato o para eliminar la cuenta creada.

En esta práctica los objetivos son los siguientes:

- Crear un registro de usuarios con contraseña.
- Crear un archivo de texto en el cual se almacenen datos.
- Leer archivos de texto para cargar datos.
- Iniciar sesión con usuarios.
- Modificar datos de estos.
- Añadir y eliminar usuarios del registro.
- Saltar de un diseño a otro.

Se crearán diferentes diseños. El primero será la página principal donde aparecerán dos *Buttons* que al pulsarlos hará aparecer otras pantallas. Una pantalla para registrarse y otra para inicio de sesión.

La pantalla de registro tendrá un *TextField* para escribir el usuario, un *PasswordField* para escribir la contraseña del usuario, otro *TextField* para el nombre de la persona, un *TextArea* para escribir una pequeña descripción y un *Button* para completar el registro. Las funciones de la pantalla serán permitir registrar un usuario siempre que en todos los campos haya un mínimo de tres caracteres. Una vez se registre se añadirá ese usuario a un *TreeMap* donde quedarán guardados. Siempre que se ejecute la aplicación se deberá cargar esa lista y cada vez que se registre un usuario nuevo se guardará. No se permitirá añadir un usuario ya existente. Desde esta pantalla también se podrá ir a la de inicio de sesión.

La pantalla de inicio de sesión se compondrá de un *TextField* para escribir el usuario, otro para la contraseña y un botón para iniciar sesión. No permitirá iniciar sesión si el usuario no existe o si el usuario si existe pero la contraseña no es correcta. Desde esta pantalla también se permitirá ir a la de registro.

Cuando un usuario inicio sesión aparecerá un cuarto diseño. Se abrirá el perfil del usuario. Este deberá contener tres *Buttons*, mostrar el nombre de la persona y el *TextArea* con la descripción de la persona. El primer botón será para cerrar sesión, con el cual se cerrará la sesión del usuario y se volverá a mostrar la pantalla de inicio de sesión. El segundo botón será para eliminar el perfil, el cual lo eliminará de la lista de usuarios. Y finalmente, un botón para modificar el perfil del usuario. En el momento que

se pulse, el área de texto se desbloqueará y permitirá modificar la descripción, una vez modificada, además de que deberá aparecer otro botón para guardar la modificación.

Estos son todos los diseños y la función de cada uno de ellos, solo faltaría añadir un *Button* en el diseño de registro el cual elimine todos los usuarios registrados.

Esta práctica se encuentra en el nivel 3 de complejidad de la taxonomía de Bloom. Además de haber comprendido toda la información de las prácticas anteriores, en esta, se tendrán que aplicar esos conocimientos. Los alumnos deberán utilizar los componentes y funciones aprendidas para resolver el problema, se construirán diferentes diseños y se irán mostrando en función del programa.

6.4.2. *MANUALES Y DOCUMENTACIÓN PARA EL PROFESORADO*

Como se ha explicado anteriormente se ha realizado tres diferentes documentos: manual de instalación de Vaadin Designer, manual de funcionamiento del Arduino y conceptos básicos.

6.4.2.1. MANUAL INSTALACIÓN VAADIN DESIGNER

En el manual se explica como se instala el plugin de Vaadin para Eclipse. Seguidamente se muestra como se crea un proyecto de Vaadin, y una vez creado, como se crea un diseño.

Y finalmente una vez se ha creado el diseño, es el momento de ejecutarlo. El manual muestra paso por paso como dar Run al proyecto a través del Maven Built. Se ejecutará un pequeño programa muy sencillo para ver el funcionamiento.

6.4.2.2. MANUAL FUNCIONAMIENTO DEL ARDUINO

Se ha desarrollado un manual para el profesorado donde se explicará el funcionamiento del *Arduino*. Primeramente se explicarán los componentes que van a ser utilizados en las prácticas y como se colocarían en la placa protoboard. Se explicará el funcionamiento de cada uno de ellos y en qué posición tienen que colocarse los pines.

Por otro lado, también se explicará cómo se programa el Arduino. Que lenguaje utiliza, como se declaran las variables y que funciones hay. Primeramente se explicará un lenguaje básico con entradas y salidas digitales donde simplemente habrá que escribir o leer valores digitales de estos pines y las entradas y salidas analógicas, donde se escribirán o se leerán valores entre 0 y 1023.

Más adelante, se enseñará un lenguaje más avanzado, en el cual el Arduino comunicará mediante el puerto serie con la ejecución del programa Eclipse, y de esta manera podrán compartir información entre una aplicación y otro.

Este manual de funcionamiento del Arduino será exclusivo para el profesorado. Ellos serán los encargados de realizar los montajes del Arduino con su programa descargado en él para que los alumnos no pierdan tanto tiempo en la programación y el montaje del Arduino.

6.4.2.3. CONCEPTOS TEÓRICOS

También se ha realizado un manual de conceptos teóricos para el profesorado. En él se explica en que consiste cada práctica y que clases y métodos se van a utilizar. El profesor de la asignatura preparará sus clases en función a las prácticas que se vayan hacer. Al indicarse los métodos y clases nuevos que aparecerán en las prácticas, los profesores deberán explicar previamente en horas de clase lo imprescindible para la realización de las prácticas.

Los propios enunciados de las prácticas ya tienen unas pautas para que los alumnos sean capaces de realizarlas, pero en alguna de ellas, en la que se utilizan clase diferentes o se establece una comunicación con el puerto serie, los alumnos necesitarán previamente haber visto algún ejemplo o saber para qué sirve las clases introducidas.

7. RESUMEN DE RESULTADOS

7.1. PLANIFICACIÓN TEMPORAL

Trabajo 5 horas al día en horario de 08:00h a 13:00h. He dedicado 4 horas diarias al proyecto, de 15:00h a 19:00h de lunes a viernes. Tuve un parón entre el 23 de octubre de 2018 y el 14 de enero de 2019 en el cual no dedique tiempo al proyecto y me hizo pedir la prórroga. Tuve una fractura en la pierna que me hizo coger la baja laboral ya que necesitaba reposo absoluto. La pierna tenía que estar en una postura determinada que era incómodo el trabajar.

Aquí se puede ver el número de horas que he dedicado a cada parte del proyecto y el diagrama de Gantt.

Actividad	Horas
Documentación previa	60
Presupuesto estimado	20
Preparación manuales	80
Preparación enunciados	80
Resolución prácticas	140
Redacción memoria	180
Preparación presentación	40

Tabla 3 Distribución de horas.

DIAGRAMA DE GANTT

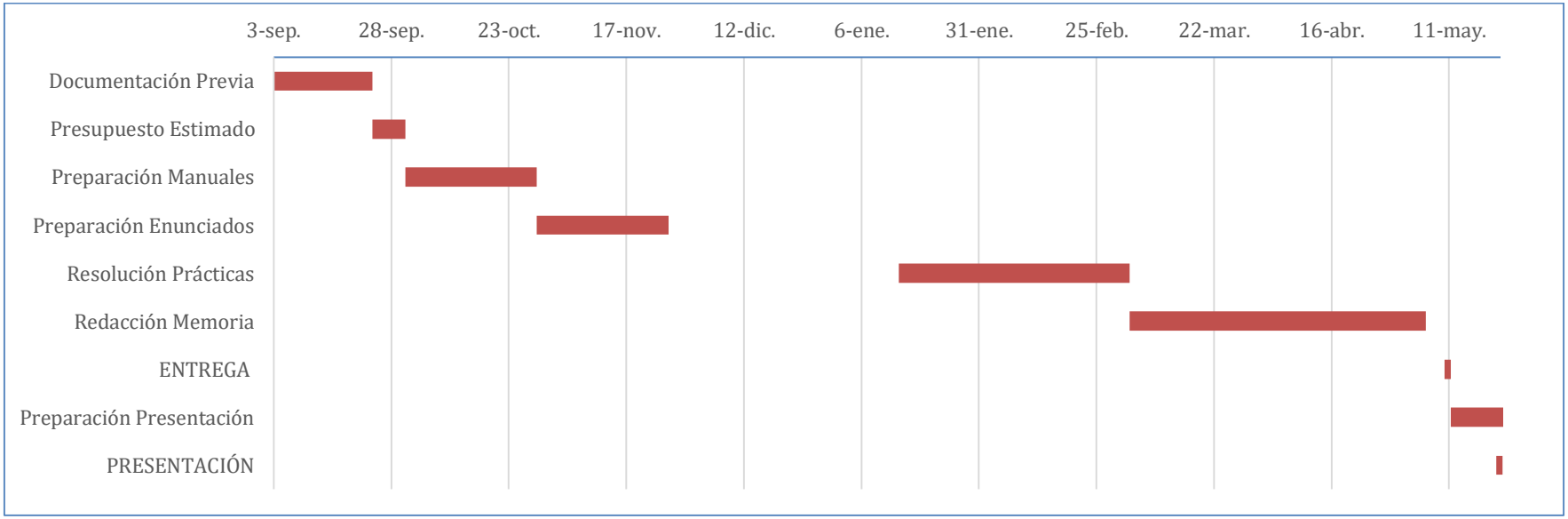


Tabla 4 Diagrama de Gantt.

7.2. *PRESUPUESTO*

Para este proyecto se ha utilizado el software de *Processing* y de *Arduino* que son gratuitos. Y por parte de *Vaadin*, se utilizará la trial de 30 días gratuita, por lo tanto por parte de software no ha habido gasto ninguno.

Se han comprado tres Arduinos para la realización de las prácticas, a un precio de 88.95€ cada uno.

Se ha dedicado 80 horas a la preparación de los enunciados y 140 horas a resolución de las prácticas, por lo tanto, se ha trabajado 220 horas de formación. El precio por hora dedicado se ha establecido a partir del precio orientativo de las prácticas académicas externas de la UPC. El precio orientativo es de 8 euros por hora.[11]

	Precio	Cantidad	Total
Arduino	88.95€/unidad	3 unidades	266,85€
Coste horas de trabajo	8€/hora	220 horas	17602,6€
			2026,85€

Tabla 5 Presupuestos del proyecto.

7.3. *CONCLUSIONES*

Para concluir el proyecto, se puede asegurar que por lo que respecta a los objetivos del proyecto, se ha cumplido, se han creado prácticas creativas y originales con Processing y Vaadin para la formación de los alumnos en la asignatura *Programación Avanzada Orientada a Objetos*.

Se han redactado los manuales necesarios para las instalaciones de softwares necesarios, se han redactado un total de doce prácticas y se han implementado sus soluciones. Las prácticas se han hecho siempre clasificando los objetivos educativos gracias a la taxonomía de Bloom.

Por parte de Processing se han hecho siete prácticas incrementando el nivel de complejidad y en nivel de dificultad. Todo lo que se aprenda en una práctica va a ser útil para la siguiente. Se ha trabajado con imágenes, movimiento de elementos y con eventos desde el teclado o el ratón. En las últimas dos prácticas de Processing se ha utilizado un Arduino para comunicar y trabajar con Eclipse de una manera en la cual los alumnos no están acostumbrados a trabajar. Quizás los alumnos de Electrónica sí que estarán acostumbrados, pero del resto de grados no, ya que esta asignatura es una optativa para todas las carreras. Puede ser muy interesante para alumnos que nunca han trabajado con Arduino trabajar con él.

Se ha trabajado con entradas y salidas tanto digitales como analógicas y se han utilizado diferentes componentes de una placa protoboard, como interruptores, leds, resistencias y un sensor de temperatura. Por parte de Arduino, además se ha hecho una práctica 0 para que los alumnos entren en contacto con el entorno y se ha hecho un manual para el profesorado explicando el lenguaje que utiliza y el montaje de la placa.

Por último, por parte de Vaadin se han diseñado cuatro prácticas, sin un gran nivel de dificultad ya que al utilizar el trial de Vaadin Designer, muchos componentes y funciones no están habilitados y por lo tanto, no se pueden trabajar con ellos. Se ha diseñado prácticas comunes que pueden ser utilizadas en la vida diaria, como formularios para ingresar en algunas páginas web; el funcionamiento de un muro de publicaciones donde las nuevas aparecen siempre al principio y van desplazando las antiguas; o registros de usuarios para registrarse en redes sociales o cualquier tipo de páginas.

He aprendido mucho haciendo este proyecto, es una rama de programación que me gusta y la he potenciado bastante conociendo aplicaciones nuevas.

Para un futuro, se pueden hacer prácticas nuevas. El kit de Arduino elegido y utilizado tiene más componentes así que si en futuro se quiere potenciar esta aplicación se pueden realizar nuevas prácticas. Contiene un sensor de inclinación, un servomotor que

puede girar hasta 180 grados., un timbrador para usarse como bocina y una pantalla LCD que contiene 2 filas de 16 caracteres cada una.

Por parte de Processing, se podría añadir el abanico de prácticas trabajando con imágenes ya que ofrece diferentes funciones con las cuales se puede sobreponer imágenes, cambiar tonalidades…

Y por parte de Vaadin, se podría comprar la licencia y trabajar con toda su potencia y realizar prácticas ya a un nivel bastante más comercial ya que incluye más componentes.

7.4. *GLOSARIO*

CPU - Central Processing Unit	Es un componente básico de la computadora personal u ordenador que procesa datos y realiza cálculos matemáticos einformáticos.[12]
GWT - Google Web Toolkit	Es un framework creado por Google que permite ocultar la complejidad de varios aspectos de tecnologías de desarrollo web.
IDE - Integorated Development Environmoent	Un IDE es un entorno de programación que ha sido que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica.[13]
JDT - Java Development Tools	Es quien proporciona los plugins que implementan un IDE Java apoyando el desarrollo de cualquier aplicación Java, incluyendo plugins.[14]
JSF - Java Server Faces	Es una tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java.
MVC - Model View Controller	Es un patrón de arquitectura de software que, utilizando 3 componentes (Vistas, Modelos y Controladores) separa la lógica de la aplicación de la lógica de la vista en una aplicación. [15]
LCD - Liquid Cristal Display	Tipo de pantalla que se caracteriza por ser plana y por estar formada por píxeles que contienen moléculas de cristal líquido. [16]
LED - Light Emitting Diode	Es un diodo semiconductor que genera luz cuando recibe tensión.
RGB - Red Green Blue	Consiste en representar distintos colores a partir de la mezcla de los tres colores primarios.
SCADA - Supervisory Control And Data Acquisition	Es un sistema de software que permite controlar y supervisar procesos industriales a distancia.[17]
SPA - Single Page Application	Es un tipo de aplicación web donde todas las pantallas las muestra en la misma página, sin recargar el navegador.[18]
SRAM - Static Random Access Memory	Es un tipo de memoria basada en semiconductores que es capaz de mantener los datos mientras esté alimentada.[19]
SVG - Scalable Vectors Graphics	Es un formato de gráficos vectoriales bidimensionales.
UI - User Interface	Es el medio con que el usuario puede comunicarse con una máquina, equipo, computadora o dispositivo.

Tabla 6 - Definiciones.

7.5. BIBLIOGRAFÍA

La bibliografía presentada está escrita según la normativa IEEE.

[1] ASOCICACION AEPI, "Programación Orientada a Objetos en Java - AEPI Escuela de programacion en Madrid." [Online]. Available: https://asociacionaepi.es/programacion-orientada-a-objetos-en-java/. [Accessed: 17-Mar-2019].

[2] GENBETA, "Eclipse IDE." [Online]. Available: https://www.genbeta.com/desarrollo/eclipse-ide. [Accessed: 17-Mar-2019].

[3] C. Reas and B. Fry, *Processing : a programming handbook for visual designers and artists*. .

[4] BEEVA, "Desarrollo rapido de aplicaciones con Vaadin." [Online]. Available: https://www.beeva.com/beeva-view/desarrollo/desarrollo-rapido-de-aplicaciones-con-vaadin/. [Accessed: 25-Nov-2018].

[5] PRACTICAS CON ARDUINO, "Información básica sobre Arduino." [Online]. Available: http://www.practicasconarduino.com/manualrapido/informacin_bsica_sobre_arduino.html. [Accessed: 12-Mar-2019].

[6] I. Hernán-Losada, *Diseño de SW educativo basado en la taxonomía de Bloom Aplicado a la enseñanza de la programación orientada a objetos*. Editorial Académica Española, 2012.

[7] IMAGEN TAXONOMIA, "La taxonomía de Bloom, una herramienta imprescindible para enseñar y aprender » CENTRO DEL PROFESORADO Tenerife Sur." [Online]. Available: http://www3.gobiernodecanarias.org/medusa/edublog/cprofestenerifesur/2015/12/03/la-taxonomia-de-bloom-una-herramienta-imprescindible-para-ensenar-y-aprender/. [Accessed: 10-May-2019].

[8] "Kit Iniciación Arduino UNO Básico con 22 Componentes Arduino, Electronica y Robotica Electan, Tienda On Line." [Online]. Available: https://www.electan.com/arduino-uno-pack-con-kit-basico-arduino-p-3056.html?gmeltn=1&gclid=Cj0KCQiA8_PfBRC3ARIsAOzJ2uoY-O1k8ncAUBglnU65gHeGFc7wRhk9nol4I8S20zZQW1he5Nn-0vUaAjjbEALw_wcB. [Accessed: 27-Nov-2018].

[9] BRICO GEEK, "Kit Arduino Avanzado Elecfreaks EF08062 | BricoGeek.com." [Online]. Available: https://tienda.bricogeek.com/descatalogado/827-kit-arduino-avanzado.html. [Accessed: 27-Nov-2018].

[10] componentes electrónicos CETRONIC, "Arduino Starter Kit en Castellano - Cetronic." [Online]. Available: https://www.cetronic.es/sqlcommerce/disenos/plantilla1/seccion/producto/DetalleProducto.jsp?idIdioma=&idTienda=93&codProducto=151185026&cPath=1339. [Accessed: 27-Nov-2018].

[11] UPC, "Prácticas académicas externas — Prácticas académicas externas — UPC. Universitat Politècnica de Catalunya." [Online]. Available: https://www.upc.edu/cce/es/estudiantes/practicas-academicas-externas#r-gimen-econ-mico--precio-orientativo-. [Accessed: 1-May-2019].

[12] CONCEPTO.DE, "CPU: Concepto, Características y Lenguaje." [Online]. Available: https://concepto.de/cpu/. [Accessed: 17-Mar-2019].

[13] ECURED, "IDE de Programación - EcuRed." [Online]. Available: https://www.ecured.cu/IDE_de_Programación. [Accessed: 17-Mar-2019].

[14] ECLIPSE FOUNDATION, "Eclipse Java development tools (JDT) | The Eclipse Foundation." [Online]. Available: https://www.eclipse.org/jdt/. [Accessed: 17-Mar-2019].

[15] CODIGO FACILITO, "MVC (Model, View, Controller) explicado." [Online]. Available: https://codigofacilito.com/articulos/mvc-model-view-controller-explicado. [Accessed: 18-Mar-2019].

[16] DEFINICION.DE, "Definición de LCD - Qué es, Significado y Concepto." [Online]. Available: https://definicion.de/lcd/. [Accessed: 25-Nov-2018].

[17] ECURED, "Memoria SRAM - EcuRed." [Online]. Available: https://www.ecured.cu/Memoria_SRAM. [Accessed: 25-Nov-2018].

[18] DESARROLLOWEB, "Qué es una SPA." [Online]. Available: https://desarrolloweb.com/articulos/que-es-una-spa.html. [Accessed: 16-Mar-2019].

[19] PARADISO-FP7, "SCADA: qué es y sus beneficios." [Online]. Available: https://paradiso-fp7.eu/scada/. [Accessed: 17-Mar-2019].

ANEXO A: ENUNCIADOS DE LAS PRÁCTICAS

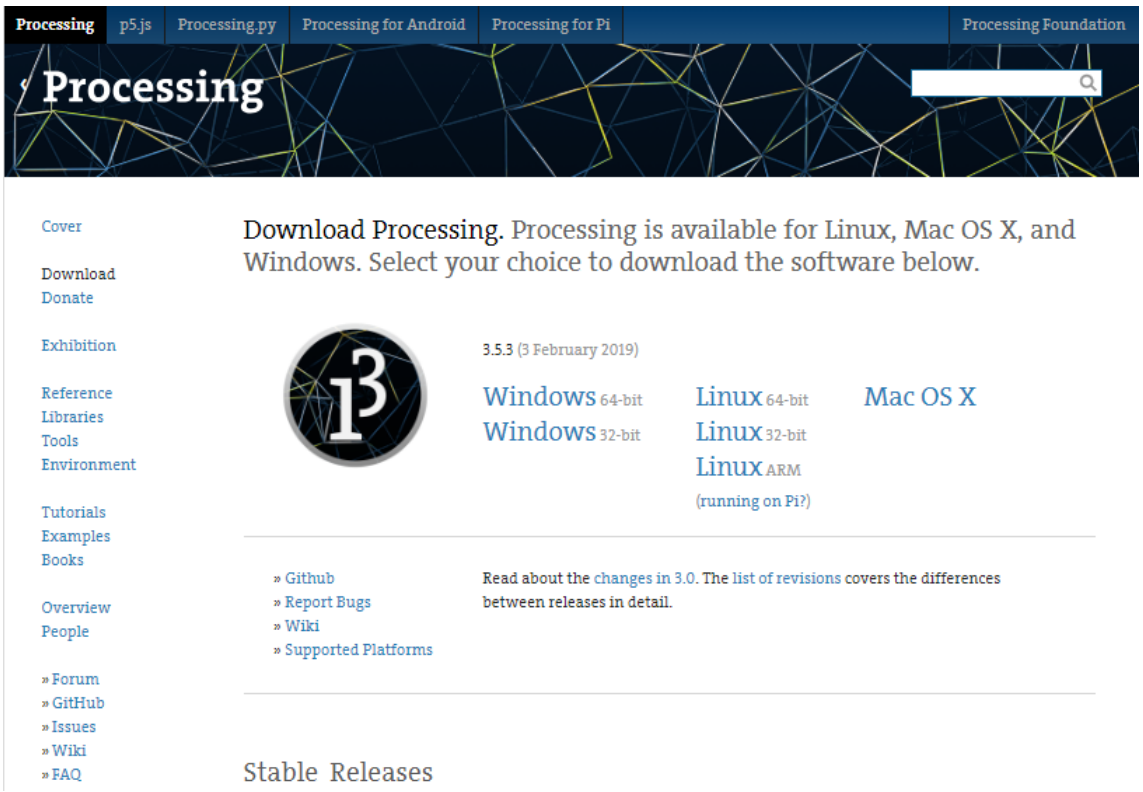
PRÁCTICA 0: INTRODUCCIÓN A PROCESSING

Objetivos:

- Importar las librerías de Processing para trabajar en Eclipse.
- Aprender cómo crear un proyecto Processing.
- Aprender el lenguaje básico para programar en Processing.

A continuación, se describen los pasos para llegar a crear un proyecto java desde Eclipse que pueda utilizar las librerías de Processing:

Descargar las librerías de Processing. Las librerías están disponibles en la web de Processing, en el apartado de descargas: https://processing.org/download/.



Apartado de descargas de la web de Processing.

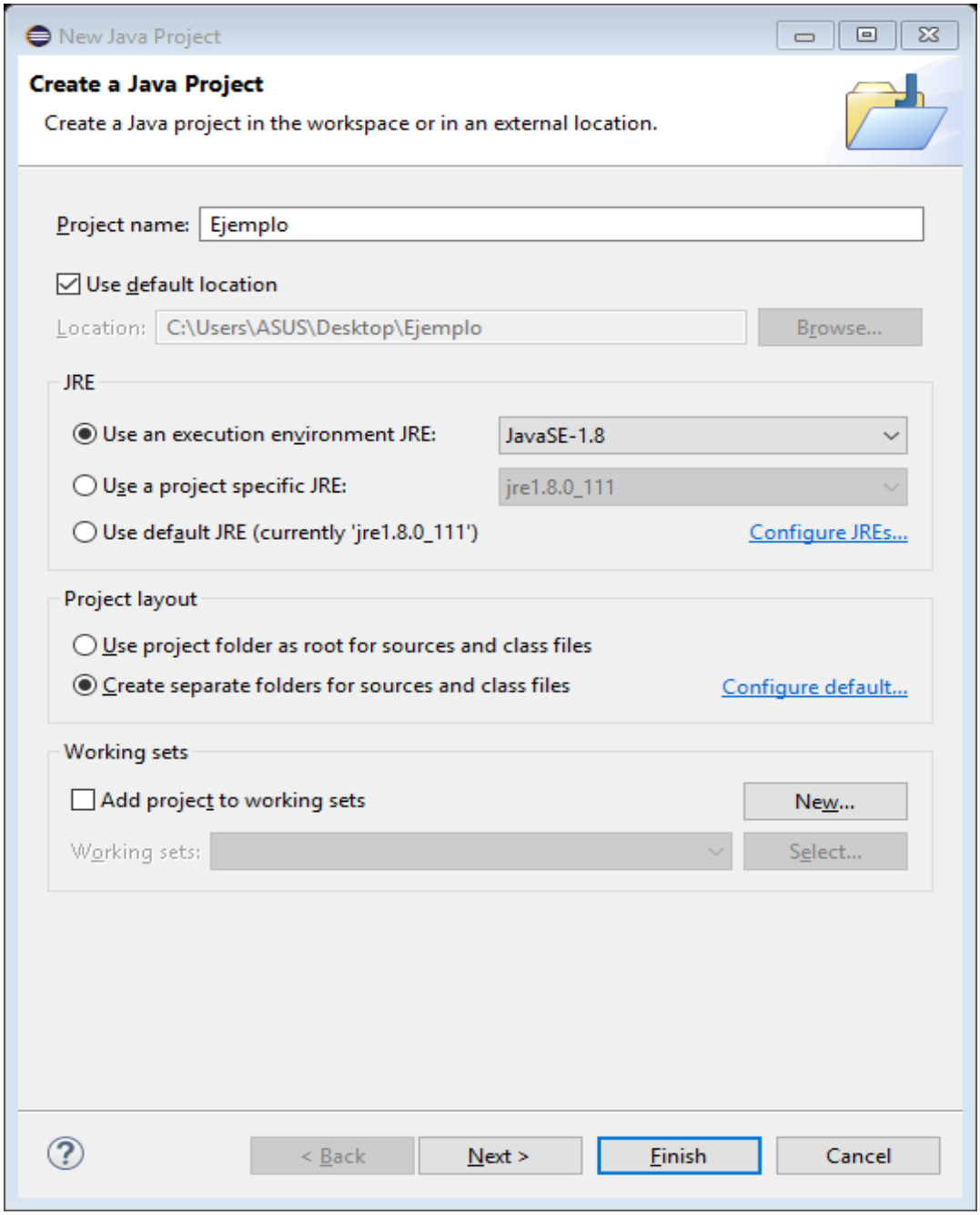
Descomprimir el archivo descargado en el punto 2 para poder utilizar dichas librerías en la ubicación deseada.

Abrir *Eclipse*, el cual se ha instalado previamente, y elegir la ubicación de trabajo, también conocida como '*workspace'*.

En *Eclipse*, crear un nuevo *Proyecto de Java*:

File > New > Java Project

Hay que dar un nombre al proyecto antes de darle clic al botón '*Finish'*.



Ventana de Eclipse para crear un nuevo proyecto.

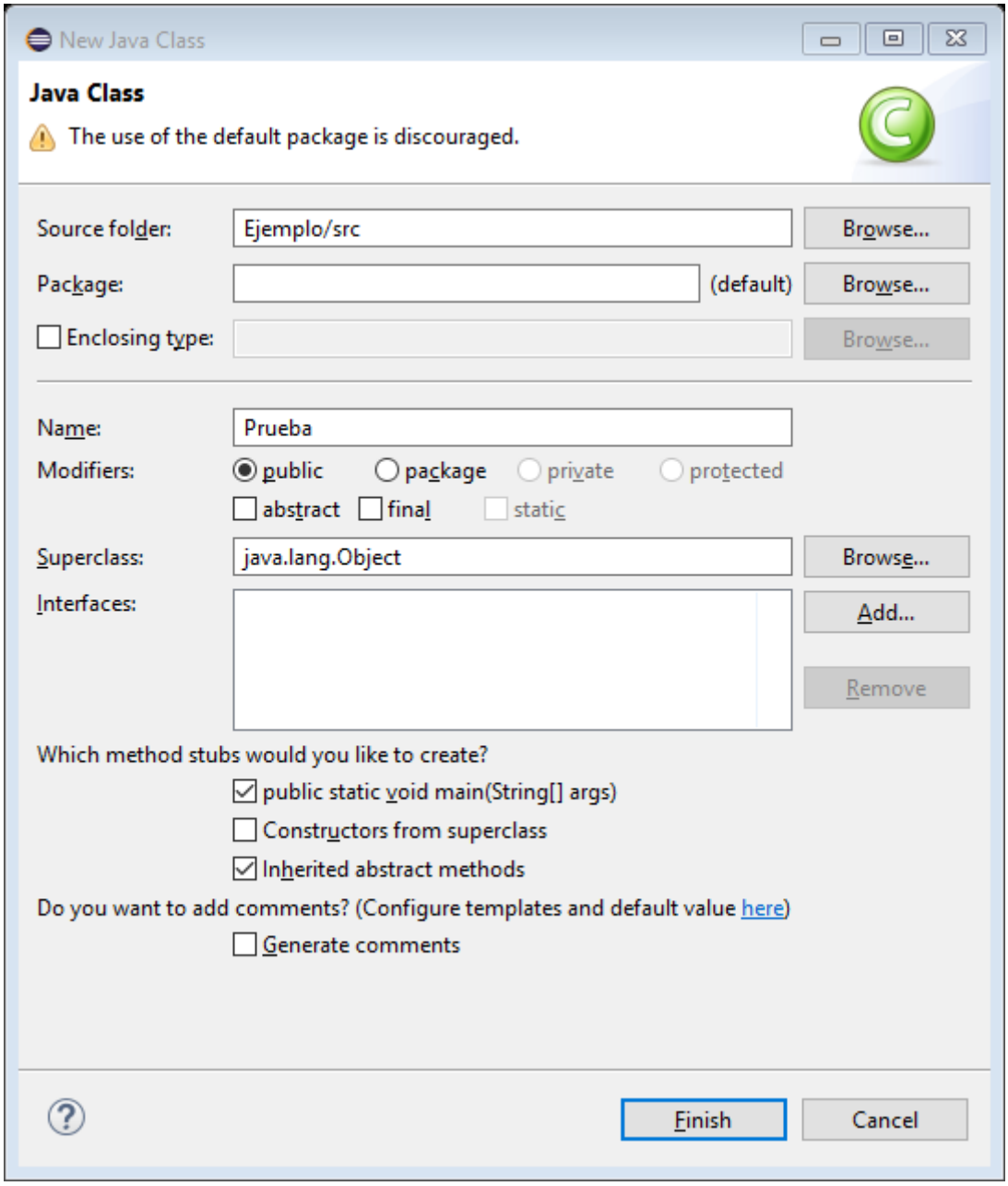
Se crea una clase *java*:

File > New > Class

Dentro de la ventana que aparece, hay que:

Dar un nombre a dicha clase

Marcar el checkbox 'public static void main(Strings[] Args)', para crear la clase con dicho método ya implementado.



Ventana de Eclipse para crear una nueva clase.

Importar la librería de *Processing*.

File > Import > General > File System

Para ello, realizamos las siguientes acciones:

Clic en el botón 'browse' para poder buscar la carpeta del archivo que se ha descomprimido en el paso 3.

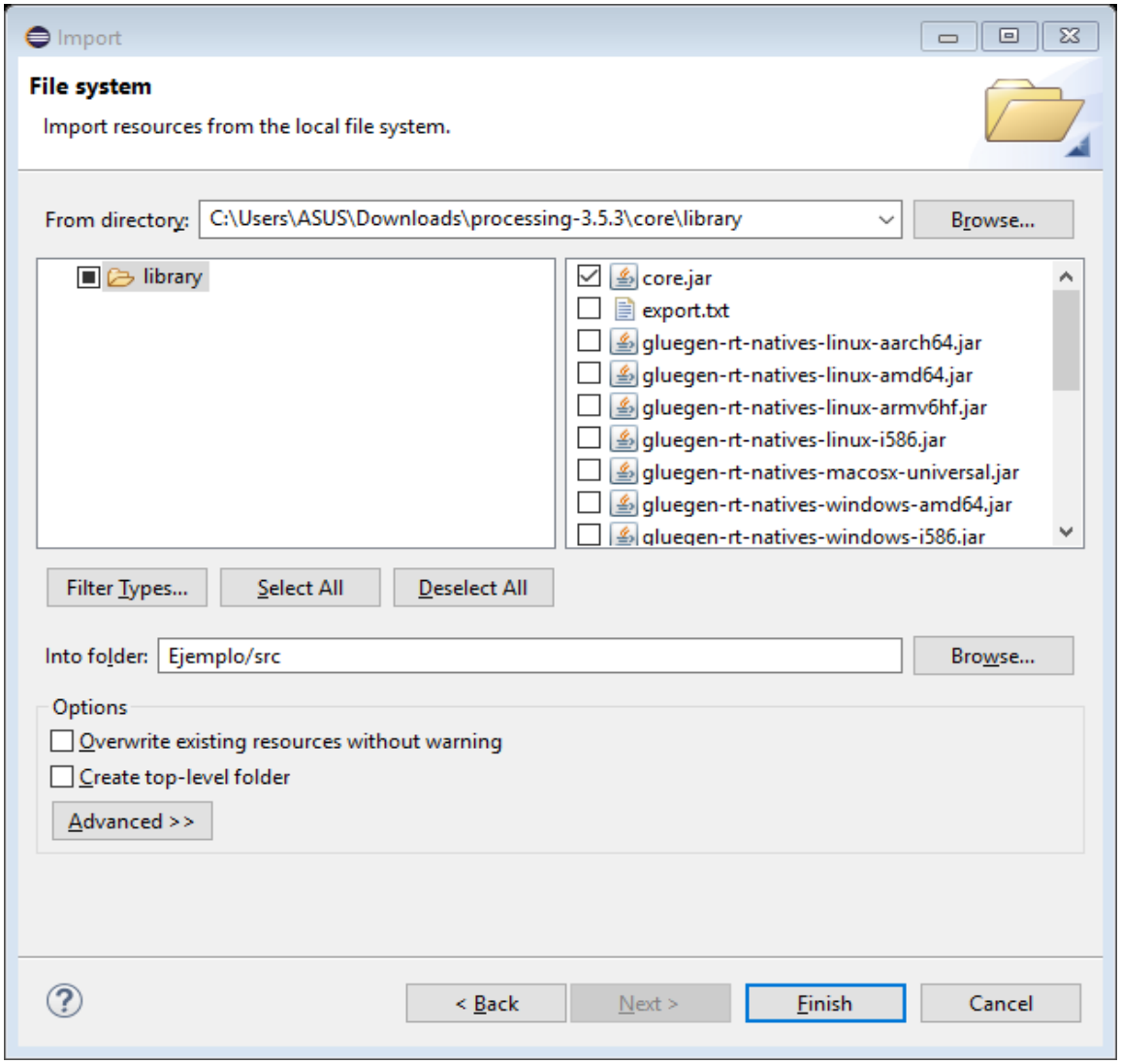
Entrar dentro de dicha carpeta.

Entrar dentro de la de la carpeta 'core'.

Dentro de la carpeta core, seleccionar la carpeta 'library' y aceptar.

Marcar el checkbox 'core.jar'

Clic en el botón 'Finish'



Ventana de Eclipse para importar una nueva librería.

Descomprimir las librerías ‘core.jar’ para poder utilizarlas. Se generan automáticamente al realizar los siguientes pasos:

Clic derecho en ‘core.jar*’:*

Package Explorer > core.jar

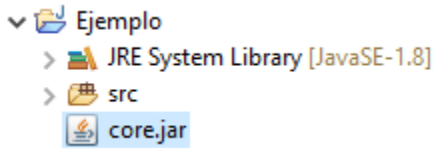
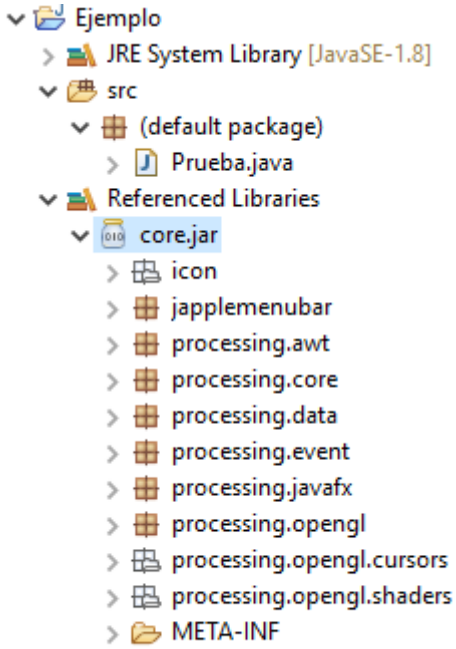


Fig. 5 – Librería ‘core.jar’ en el explorador de proyectos de *Eclipse*.

Click en:

Build Path > Add to Build Path

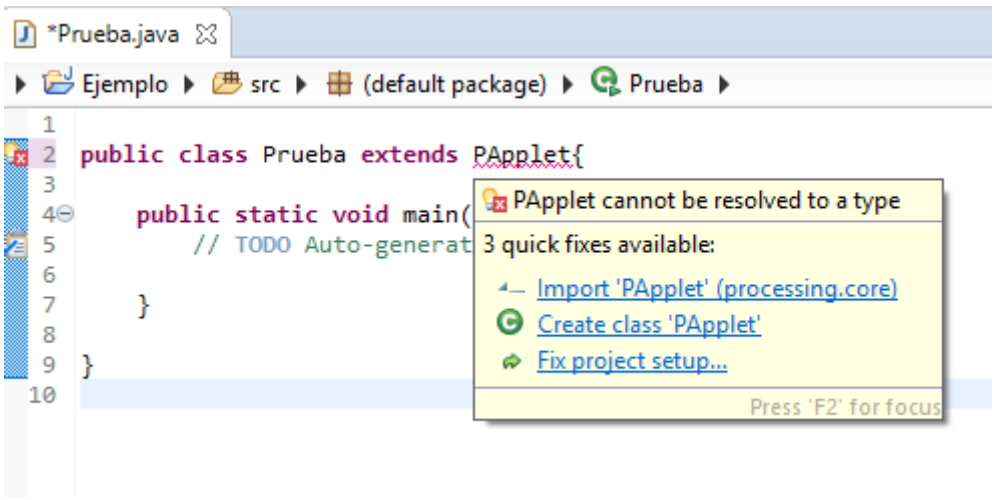


Librería ‘core.jar’ descomprimida en el explorador de proyectos de Eclipse.

Después de seguir estos pasos ya se dispondrá de un proyecto java que incorpora las librerías de Processing.

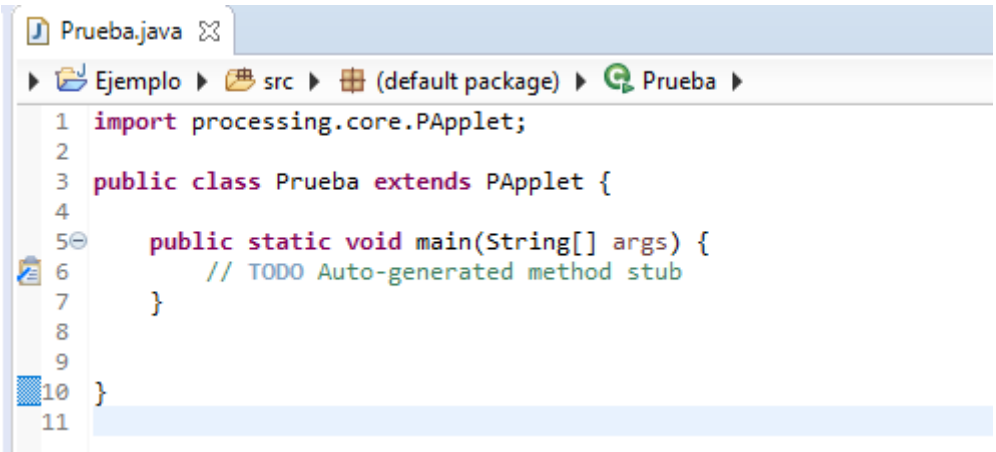
Con el proyecto creado y las librerías importadas, ahora es el momento de trabajar sobre la clase java (la creada en el punto 6). Para indicar a java que la clase va utilizar las

librerías, esa clase debe heredar (en java sería realizar un 'extend') la clase '*PApplet*' perteneciente a la Processing.



Clase 'Prueba' que realiza una herencia de la clase '*PApplet*', de la librería Ventana de Eclipse para importar una nueva librería.

Clicar sobre Import '*PApplet*' (processing.core) y la clase estará extendida a la *Applet* de Processing, el siguiente paso es iniciarla.



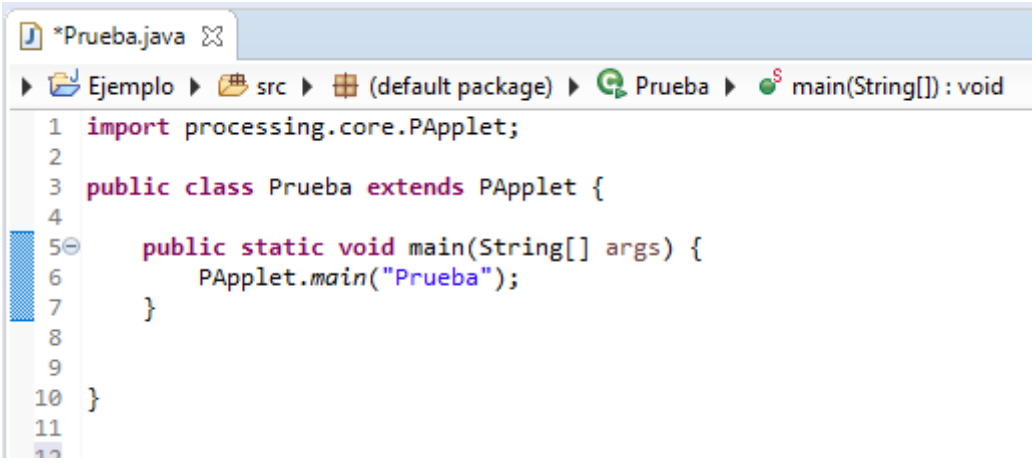
Clase 'Prueba' una vez se ha importado la *PApplet* "processing.core".

Para poder iniciar la clase es suficiente con escribir una línea dentro del main.

PApplet.main("nombre de la clase");

O en nuestro caso:

PApplet.main("Prueba");



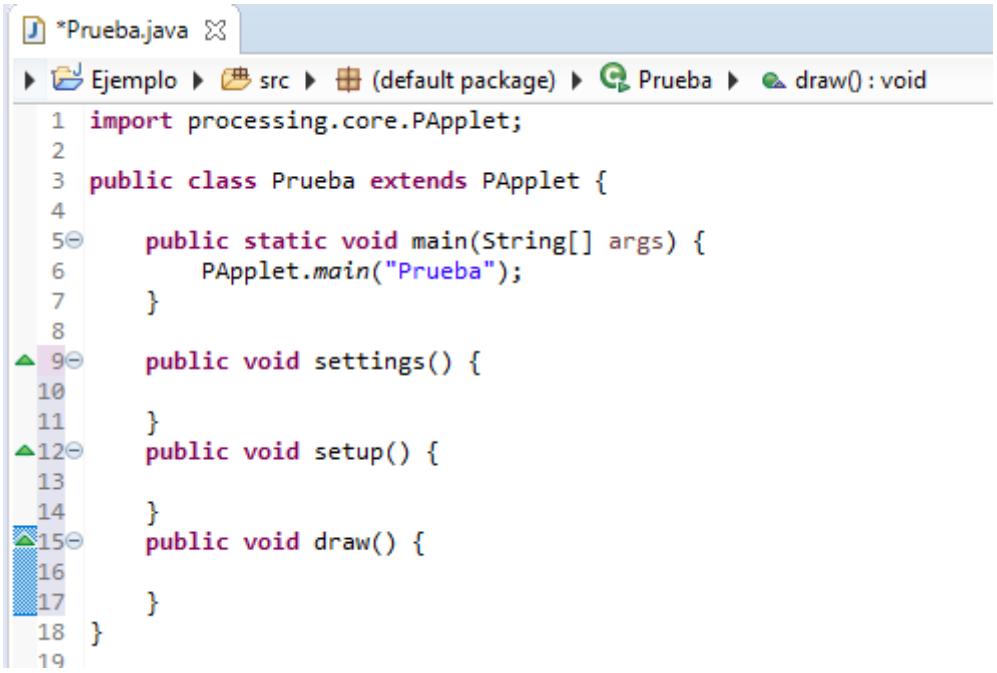
Clase 'Prueba' con la *PApplet* inicializada para poder ejecutar cualquier programa.

Crear los tres métodos "obligatorios" para iniciar un programa en *Processing*:

public void settings(), este método que se ejecuta primero en el cual se pueden inicializar variables y en el cual solo se puede utilizar una función de *Processing*, *size(int x, int y)* y se utiliza para dar tamaño a la ventana.

public void setup(), este método solo se ejecutará una vez, la primera vez que se ejecute el programa

public void draw(), este método se ejecuta infinitamente en bucle siempre que estemos ejecutando el programa



Clase 'Prueba' con los tres métodos principales con los que funciona Processing.

De esta forma, se empieza a programar utilizando todas las funciones que contiene las librería core de Processing.

Ejercicio 1: Hola mundo!

A continuación se va a hacer una pequeña toma de contacto con el entorno Processing con un programa muy sencillo. Mostrar el texto “Hola mundo!” con letras negras en el centro de una pantalla con fondo blanco de 500 de ancho y 300 de alto.

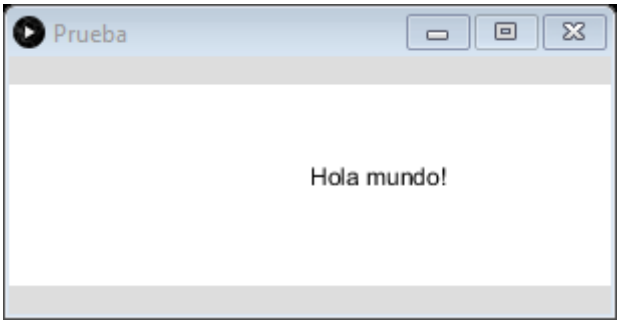
Métodos necesarios:

size(int ancho, int altura): para dar el tamaño indicado a la ventana.

background(int x): para dar color al fondo de la ventana, 0 corresponde a negro y 255 a blanco.

text(string “texto”, int posición X, int posición Y): para escribir en la ventana cualquier String en la posición X e Y que se desee.

fill(int x): para dar relleno a cualquier forma, línea o letra que se escriba a continuación, en este caso se definirá el color del texto



Primer programa mostrando el texto “Hola mundo! “ en el centro de la ventana.

Práctica 1: Primeros programas

Objetivos:

- Trabajar con clases y herencia.
- Crear diferentes formas geométricas utilizando las funciones 2D primitivas de Processing.
- Trabajar con métodos continuos.
- Crear movimiento de objetos por pantalla.
- Mostrar imágenes por ventana.
- Utilizar eventos desde el teclado.
- Utilizar eventos desde el ratón.

1. CREACIÓN DE FORMAS

Para este primer apartado se tiene que crear una cara sonriente con formas geométricas. Se utilizarán círculos para los ojos, un rectángulo para la nariz y un triángulo para la boca.

El proyecto debe de tener la clase *Formas*, con los siguientes atributos: una coordenada X, una coordenada Y y tres variables *integer* para dar color a esas formas. Todas las variables deberán tener sus correspondientes *setters* y *getters*. A continuación se crearán las tres clases que heredarán de la clase Formas: Círculo, Triángulo y Rectángulo.

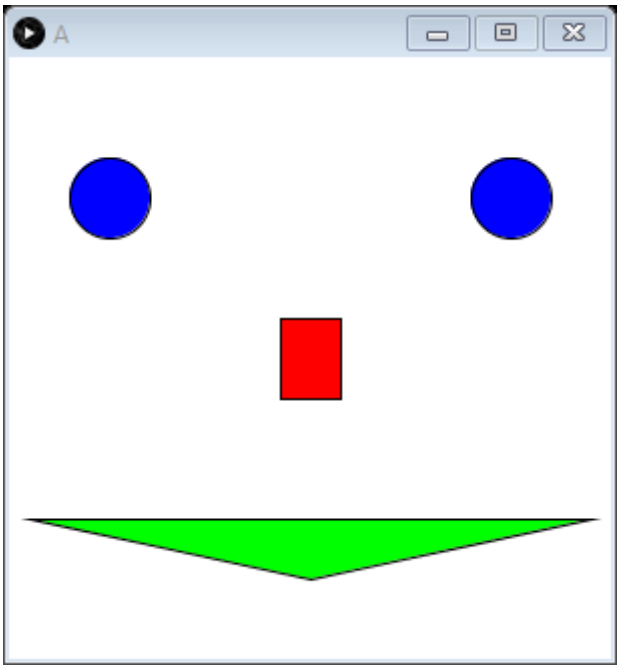
La clase *Círculo* deberá contener el radio del ojo y un método *construirCirculo(),* el cual deberá crear un circulo en las coordenadas correspondientes con su color y radio.

La clase *Triángulo* deberá contener el ancho y el alto de la boca y un método *construirTriangulo(),* el cual deberá *crear* un triángulo en las coordenadas correspondientes con su color y medidas.

La clase Rectángulo deberá contener el ancho y el alto de la nariz y un método *construirRectangulo(),* el cual deberá crear un rectángulo en las coordenadas correspondientes con su color y medidas.

Para dar color en el lenguaje *Processing*, ya sea en un contorno o un relleno, se utilizará una variable integer dentro de un rango que va desde 0(negro) a 255(blanco). En caso de querer un color en la escala de grises, se utilizará solamente una variable; y en caso de querer otro color, se utilizarán tres variables (*RGB*), el primer número nos indicará el color rojo; el segundo, el verde; y el tercer el azul. De esta manera jugando con esos tres valores, siempre dentro del rango, se tendrá el color deseado.

Una vez estén las clases creadas es el momento de la creación del test. Se deben crear los objetos necesarios dar forma a nuestra cara de colores.



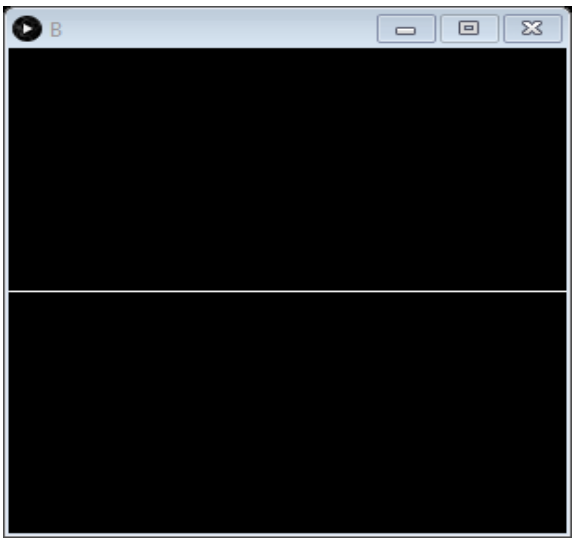
Pantalla mostrando la cara sonriente.

2. MOVIMIENTO DE UNA LÍNEA

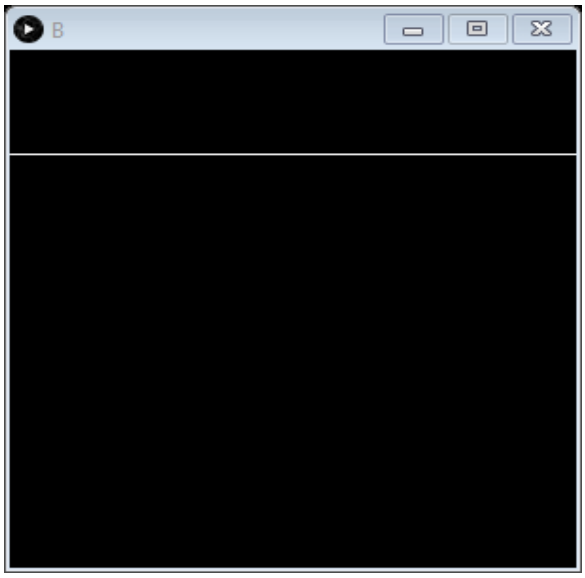
Para empezar se tiene que hacer un programa en el cual aparezca una línea blanca horizontal en un fondo negro.

Una vez creada se le va a dar movimiento, se buscará un desplazamiento de la línea de abajo a arriba de manera que una vez llegue al punto más alto de la pantalla vuelva a aparecer en el punto más bajo y continúe moviéndose. La marcha y paro del movimiento se va hacer mediante el clic izquierdo del ratón, clicando la primera vez la pondremos en marcha y con una segunda vez la pararemos. Y así de manera continua.

Para hacer aparecer una línea hay que utilizar el método *line(),* indicando la coordenada inicial y la final.



Línea en posición inicial.



Línea un segundo después del primer clic.

3. MOVIMIENTO DE UNA IMAGEN A TRAVÉS DEL TECLADO

Se tiene que desarrollar un programa en el cual se muestre una imagen en la ventana y se pueda mover a través del teclado.

Primeramente se mostrará por la pantalla una imagen cualquiera dándole su posición. La imagen deberá estar guardada en el proyecto java. Para poder mostrar una imagen

hay que utilizar la clase *PImagen* de Processing la cual permite trabajar con ella (mostrarla, reducir su tamaño, hacer filtros de imagen…)

Una vez se muestre la imagen por pantalla se debe poder desplazar mediante el teclado. Se utilizarán las teclas A, S, D y W.

Tecla A: desplazamiento hacia la izquierda
Tecla S: desplazamiento hacia abajo
Tecla D: desplazamiento hacia la derecha
Tecla W: desplazamiento hacia arriba

Habrá que utilizar el método de Processing *keyPressed()* el cual se ejecutará cada vez que se pulse una tecla. Para saber que tecla se ha pulsado habrá que hacer una comparación con la variable *key*.

Y para acabar se habrá de limitar el desplazamiento de la imagen de manera que no supere los límites de la pantalla y siempre se vea completa.

Imagen mostrada por pantalla preparada para desplazarse mediante el teclado.

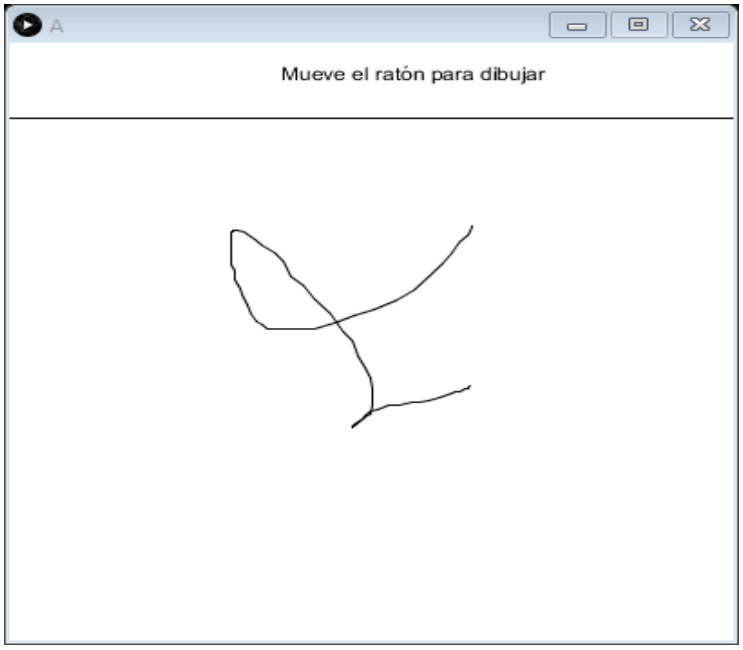
PRÁCTICA 2: PAINT

Objetivos:

- Conocer diferentes funciones de Processing.
- Crear programas entretenidos.
- Simular un Paint.
- Estructurar bien las diferentes zonas de la pantalla.
- Trabajar con las variables mouse X y mouse Y.

1. DIBUJAR

Se tiene que crear un programa el cual permita dibujar en toda la pantalla exceptuando la parte superior en la cual se podrán textos instructivos como “barra de herramientas” para la realización de la práctica. Para pode saber cuándo se está dibujando (clicando el botón izquierdo del ratón a la vez que se mueve), se utilizará el método *mouseDragged().* De esta manera siempre que se clique el botón izquierdo del ratón a la vez que se mueve se ejecutará el método y se permitirá dibujar.

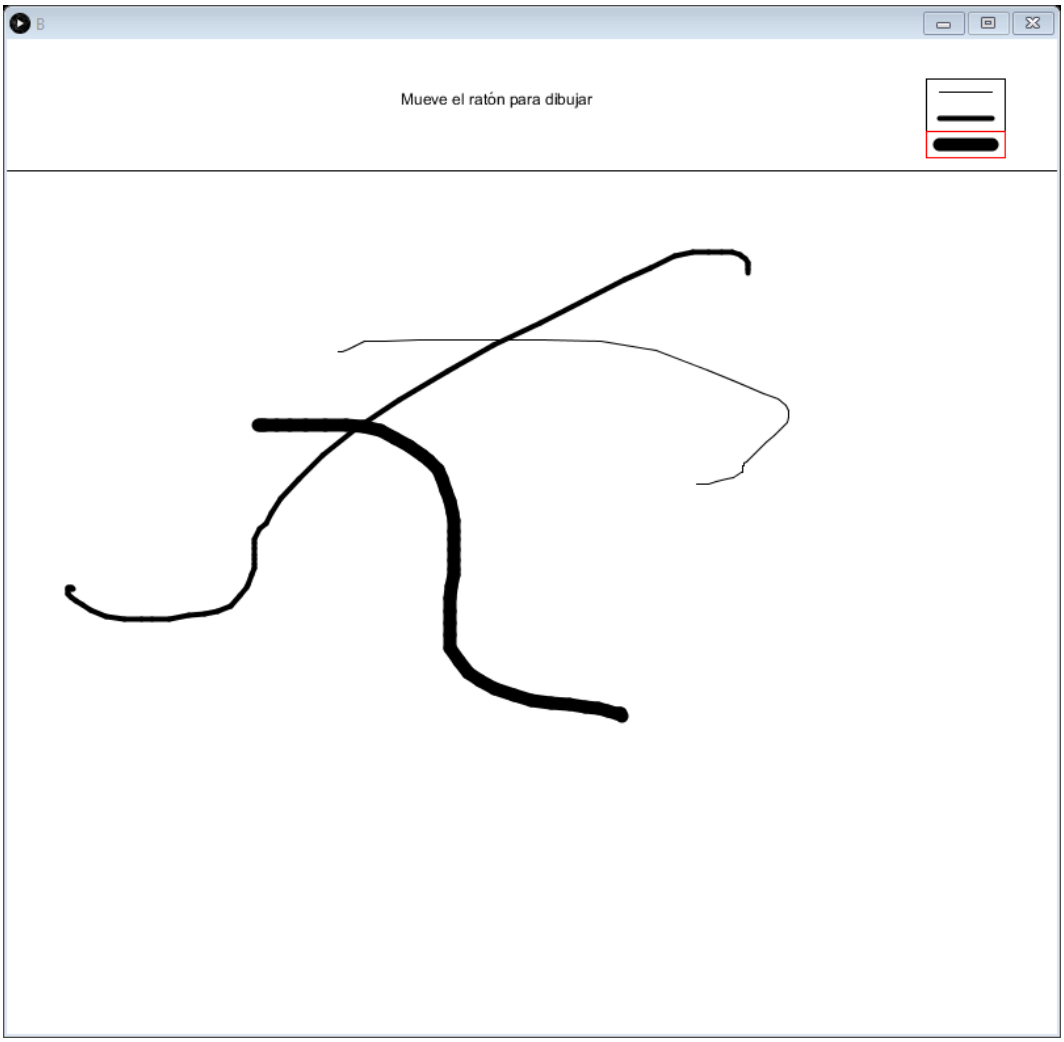


Ventana del programa dibujar.

2. DIBUJAR CON DIFERENTES ESPESORES

En este apartado se va a ir un poco más lejos y se va a permitir dibujar con varios espesores. Se dibujará un recuadro en la parte superior de la pantalla en la cual se podrá elegir entre 3 tamaños de espesor diferentes.

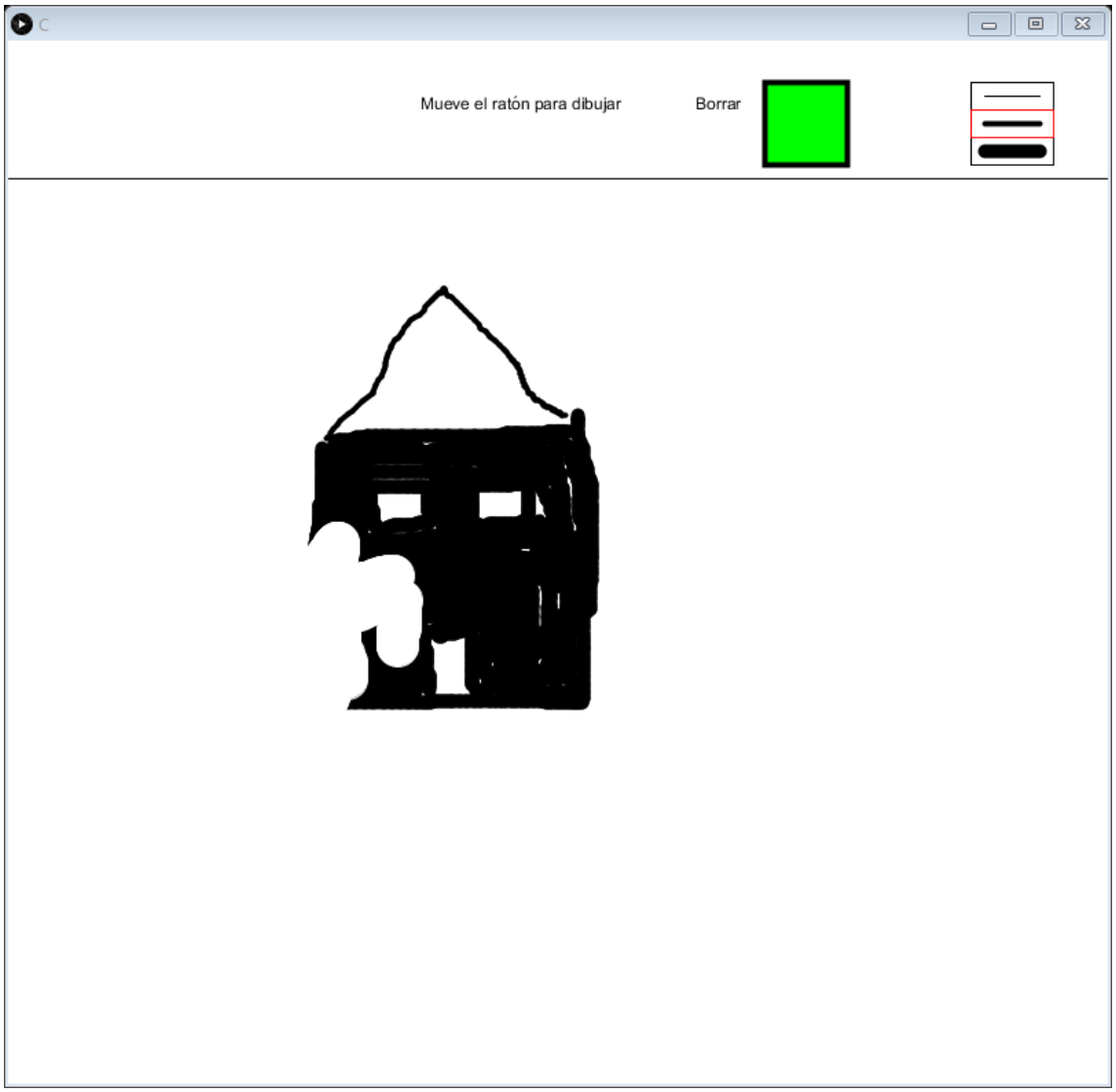
Para dar grosor a cualquier línea, tamaño de letra o grosor del contorno de cualquier figura geométrica se utiliza el método *StrokeWeight() al* cual se le introducirá una variable integer indicando el grosor.



Ventana del programa dibujar con varios espesores.

3. BORRAR

Para hacerlo más completo, se permitirá borrar el dibujo de la ventana. Se puede hacer con el desplazamiento de una elipse blanca por pantalla de manera que vaya tapando los dibujos o garabatos previamente creados. También se dibujará un recuadro en la parte superior de la pantalla de manera que se pueda elegir entre dibujar o borrar.



Ventana del programa dibujar con varios espesores

PRÁCTICA 3: PELOTA SALTARINA

Objetivos:

- Utilización de otras librerías de Java.
- Conocer la clase PVector de Processing.
- Tener conocimientos matemáticos de Vectores.

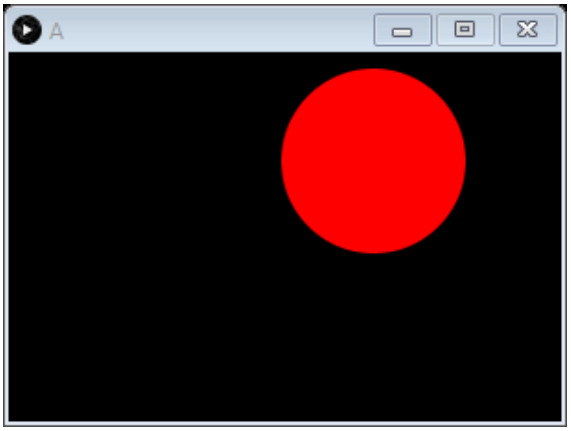
1. UNA PELOTA QUE REBOTA

Se tiene que crear un programa en el cual aparezca una esfera en la ventana rebotando entre las paredes y cambiando de dirección a cada choque con pared.

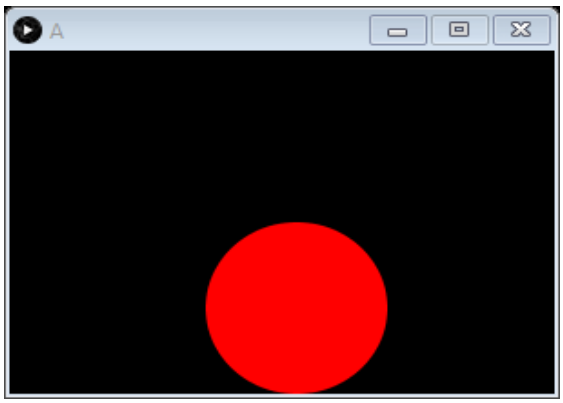
Para empezar se debe crear una clase *Esfera* la cual tendrá como atributos tres variables de clase integer para el color, una variable integer para el radio de la esfera, dos variables integer para la dirección de la esfera y una variable de clase *PVector* para posicionar siempre la esfera.

La clase PVector contiene dos variables, X e Y, y muchos métodos, permite hacer operaciones aritméticas con el vector, cambiar la dirección, calcular la magnitud… Se dará una posición inicial, un color y una dirección aleatoria a la esfera importando la librería *java.util.concurrent-ThreadLocalRandom*.

La clase *Esfera* deberá contener los siguientes métodos: el método *mostrarEsfera(),* el cual mostrará la esfera por pantalla; el método *actualizarPosición(),* en el cual se actualizará continuamente la dirección de la esfera; y el método *choqueConPared(),* en el cual en función de que pared choque se cambiará a una dirección o a otra.



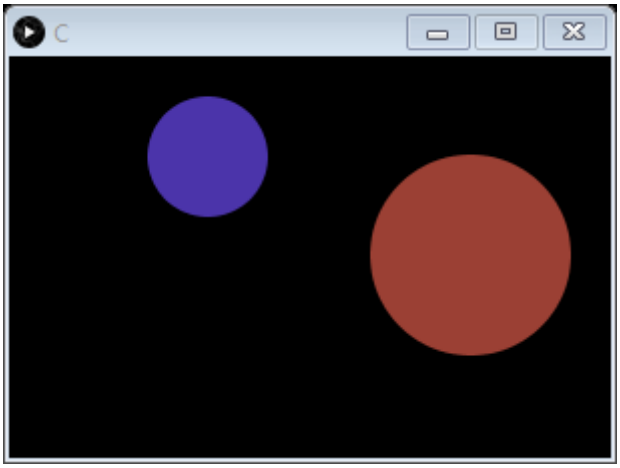
Esfera en una determinada posición



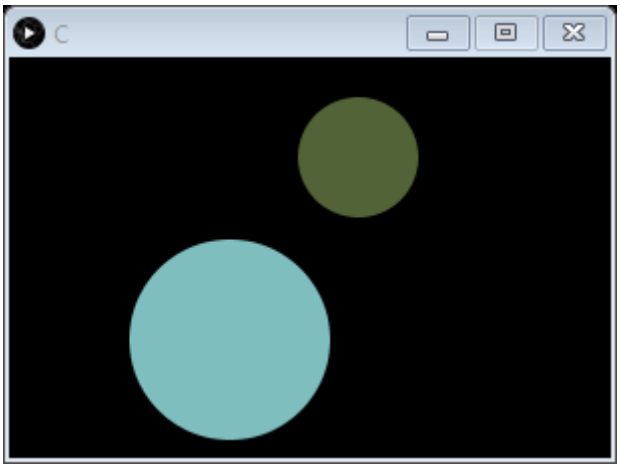
Esfera en otra posición

2. CRUCE DE PELOTAS

Se tiene que mantener el mismo programa pero añadiendo otra esfera más de manera que las dos se muevan una dirección totalmente aleatoria. Una vez aparezcan las dos esfera en la pantalla rebotando con las paredes y cruzándose entre ellas se deberá añadir un método *cruceEntreEsferas()* en el cual las esferas cambarán de color aleatoriamente cada vez que se crucen entre ellas.



Esferas al arrancar el programa.



Esferas después del primer cruce

PRÁCTICA 4: ENCUENTRA LOS OBJETOS

Objetivos:

- Trabajar con arrays de imágenes y mostrarlas aleatoriamente.
- Mostrar por pantalla las coordenadas del puntero del ratón.
- Controlar los márgenes de la pantalla de manera que as imágenes siempre queden centradas.
- Aprender a seleccionar objetos aleatoriamente dentro de un array.

En esta práctica se creará el juego de encontrar los objetos.

Se elegirán imágenes en las cuales aparezcan diferentes y se irán mostrando por pantalla. Irá que mostrar las imágenes aleatoriamente con varios nombres de objetos que aparecen en la imagen. Los jugadores deberán encontrar esos objetos en la imagen y seleccionarlos. De esta manera habrán encontrado un objeto, cuando hayan encontrado todos los objetos nombrados debajo de la imagen se habrá completado esa imagen. Cuando se complete una imagen, se mostrará la siguiente, de esta manera, cuando el jugador haya encontrado todos los objetos de todas las imágenes habrá ganado.

1. OBTENCIÓN DE COORDENADAS DE LOS ELEMENTOS

Primeramente, se tiene que elegir varias imágenes, cualquier imagen en la que aparezcan diferentes elementos que se vean a simple vista, algunos objetos serán más difíciles de encontrar que otros.

Para poder determinar si un jugador ha encontrado un objeto o no, se comprobará mirando el puntero del ratón, si este está lo suficientemente cerca o no del objeto. Para ello se necesitará saber las coordenadas de los objetos. Por lo tanto en este apartado se creará un programa para saber esas coordenadas. Una vez se hayan elegido las imágenes habrá que elegir que objetos se querrán encontrar. Cuando ya se haya decidido cuales, se creará el programa con el método que devuelva las coordenadas exactas del puntero del ratón. Por lo tanto, el método tendrá que escribir por consola las coordenadas del puntero del ratón en el momento de clic del ratón.

De esta manera, sabremos en que área de la pantalla se encuentran los elementos para posteriormente en el siguiente apartado asignarla a los objetos.

2. CREACIÓN JUEGO

Una vez se ha hecho el programa permite obtener las coordenadas de los elementos seleccionados, hay que hacer el programa del juego.

Primeramente se deberá crear la clase *Elemento* para los objetos, la cual tendrá como atributos el nombre del elemento, las coordenadas(X e Y) del centro del elemento, una variable booleana y el ancho y el alto del elemento. Con las coordenadas del elemento y el ancho y el alto de este se construirá el área de la imagen donde se encuentra el objeto. La variable booleana servirá para saber si el objeto se ha encontrado o no.

Se declararán entre ocho o nueve objetos por imagen, de los cuales se elegirán cinco aleatoriamente para encontrar. Se mostrarán las imágenes aleatoriamente, y deberá aparecer un recuadro de bajo de la imagen donde aparezcan los nombres de los objetos elegidos aleatoriamente. De esta manera, el jugador sabrá que objetos son los que tiene que encontrar. De manera que se vayan encontrando los objetos, el nombre de los objetos encontrados irán desapareciendo hasta encontrarlos todos. Cuando esto suceda, deberá aparecer la siguiente imagen con los nombres de los cinco objetos elegidos aleatoriamente a encontrar. Cuando se hayan encontrado todos los objetos de todas las imágenes el jugador habrá ganado el juego.

El programa también deberá contener un contaje de fallos, de manera que cada clic incorrecto, es decir, fuera de todas las áreas de los objetos, se incrementará en uno el número de fallos. Cuando se llegue a número determinado de fallos, la partida se dará por perdida.

El programa deberá contener los siguientes métodos:

- *mostrarImagen():* este método cual mostrará la imagen que corresponda en la ventana y mostrará el recuadro con el nombre de los elementos a encontrar debajo de la imagen, a manera que se vayan encontrado los objetos, su nombre irá desapareciendo.
- *métodoEncontrado():* este método irá continuamente comprobando si se han encontrado todos los objetos de manera que cuando todos los elementos de una misma imagen se hayan encontrado se mostrará la siguiente imagen a mostrar en la ventana. Y si ya no quedan imágenes por mostrar se mostrará un mensaje por pantalla “ Has ganado!”.

- *mousePressed():* este método comprobará en el momento de hacer el clic con el botón izquierdo del ratón si las el puntero del ratón se encuentra dentro de alguna de las áreas. Cada vez que se acierte y se seleccione correctamente uno de los elementos que se piden de encontrar, se señalizará el objeto encontrado con un de los elementos. si se ha acertado o no. Cada vez que se encuentre un objeto se señalará con un círculo verde señalizando el acierto y se eliminará de la lista.

PRÁCTICA 5: FLAPPY BIRD

Objetivos:

- Crear un programa más dinámico en el que poder entretenerse
- Aumentar la velocidad de movimiento de imágenes
- Recordar funciones de Processing explicadas previamente

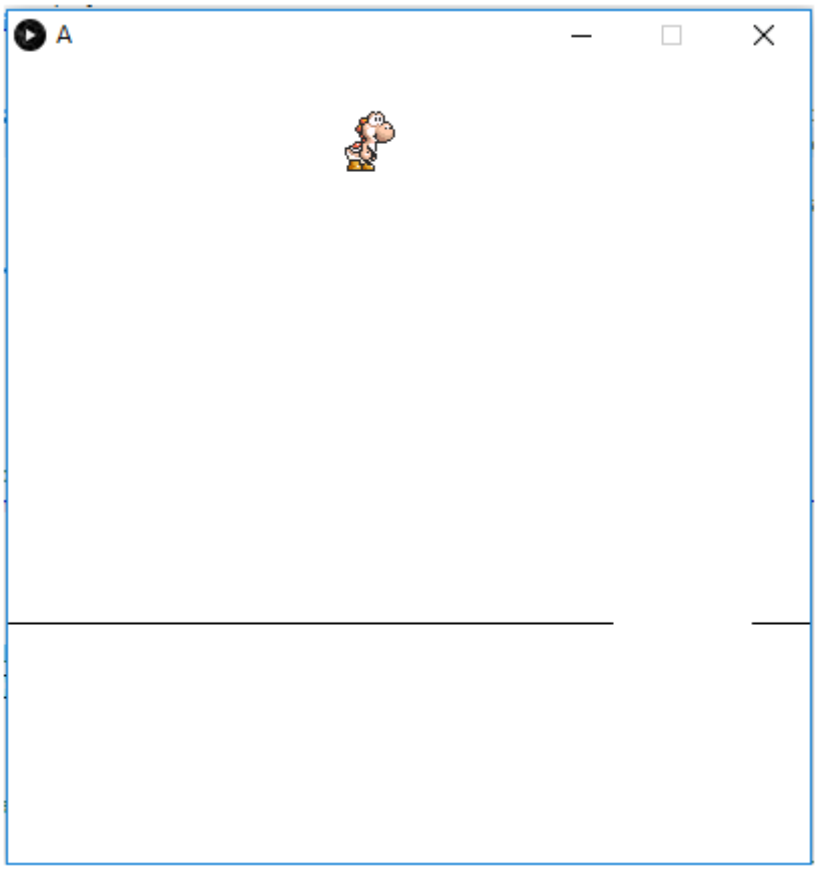
1. JUEGO

Se va a crear la simulación de un juego llamado Flappy Bird.

Se creará una línea horizontal que irá desplazándose continuamente de abajo a arriba de manera que cada x veces aumente la velocidad. Esta línea estará compuesta por 2 líneas de manera que siempre quede un pequeño hueco entre ellas. En la parte superior de la ventana se colocará una pequeña imagen (un humano, una animal…) de manera que mediante la posición del ratón se moverá la imagen de izquierda a derecha para pasar por el hueco que habrá entre las dos líneas. Conforme se vaya esquivando la línea irá subiendo de velocidad y cada vez será más difícil. Con el clic se dará marcha al movimiento de la línea. Una vez el jugador se haya chocado y haya perdido, clicando otra vez se volverá a poner en marcha el juego.

Métodos necesarios:

- *movimiento():* el cual irá desplazando la línea verticalmente dejando huecos en ellas aleatoriamente.
- *choque():* el cual una vez choque la imagen con la línea mostrará un mensaje en la ventana conforme hemos perdido.



Ventana de la simulación del juego *Flappy Bird*

PRÁCTICA 6: PUZZLE

Objetivos:

- Dividir una imagen en varias piezas.
- Trabajar con *Arrays* de objetos.
- Comunicar Arduino con Eclipse.
- Enviar datos desde Arduino al programa de Eclipse.

En esta práctica se va a realizar un Puzzle. Se elegirá una imagen cuadrada (que tenga las mismas medidas de ancho que de alto) y se dividirá en partes iguales, es necesario que se divida en un número de piezas que tenga raíz cuadrada, es decir 4, 9, 16, 25… para que se pueda dividir la imagen en 3x3, 4x4…Una vez se tengan las piezas creadas se dibujará un cuadrado con el hueco para sus piezas y sus piezas colocadas aleatoriamente para que estas se coloquen en su posición correcta.

1. DIVISIÓN DE UNA IMAGEN

.En ese primer apartado se debe dividir la imagen. Desde Processing, la forma que hay para dividir una imagen trata de crear una imagen del tamaño de las piezas y colocar la imagen en unas coordenadas fuera del rango de la pantalla de manera que solo se vea una pequeña parte de esa foto. Una vez se muestra la pieza por la ventana hacer un guardado y guardar la imagen. Para ello se deberá crear un método *dividirImagen() (PImage, número de piezas)* que introduciendo el número de piezas cree las correspondientes imágenes para que en el siguiente apartado se puedan colocar por pantalla para completar el Puzzle. El programa deberá ser apto para cualquier número de piezas siempre que sea un cuadrado de otro número. Se recomienda empezar con 4 piezas.

Si el número de piezas no es correcto se deberá mostrar un mensaje por pantalla indicando que el número de piezas no es correcto.

2. PUZZLE CON MOVIMIENTO POR TECLADO

Una vez se han creado las imágenes¸ se tiene que crear el Puzzle.

Para empezar se deberá crear los objetos de las piezas. Se creará una clase *Imagen* la cual tendrá como atributos una imagen, unas coordenadas actuales, unas coordenadas finales (en las cuales deberá ir la pieza) y una variable booleana que indique si la pieza está colocada o no. Si el número de piezas no es correcto se deberá mostrar un mensaje por pantalla indicando que el número de piezas no es correcto.

En el *setup()* se deberá crear un método *crearArrayDeImagenes()* que cree un array de clase Imagen, en el cual se carguen las imágenes creadas en el apartado anterior asociándoles sus coordenadas correctamente.

Una vez se han creado el array, se deberá crear un método *visualizarImagenes()* que muestre las imágenes en sus coordenadas actuales. También habrá que mostrar un recuadro para la imagen completa y sus correspondientes recuadros del tamaño de las piezas dentro del grande para la colocación de las piezas en su coordenada correcta.

Cuando se han creado los métodos constructivos es el momento de crear los métodos que ofrezcan acciones al programa. Para empezar se deberá crear un *mousePressed()* que permita seleccionar la imagen que se quiera mover. Si la pieza ya está colocada no se permitirá seleccionarla.

Una vez se tiene seleccionada se permitirá su desplazamiento. Este último se realizará mediante teclado, con el método *keyPressed().*

La tecla W permitirá el desplazamiento de la pieza vertical hacia arriba.
La tecla S permitirá el desplazamiento de la pieza vertical hacia abajo.
La tecla A permitirá el desplazamiento horizontal de la pieza hacia la izquierda.
La tecla D permitirá el desplazamiento horizontal de la pieza hacia la derecha.

Se irán acercando las piezas a sus coordenadas correctas, pero es difícil posicionarlas en su coordenada exacta, ya que puede que con el incremento o decremento de coordenadas desde la coordenada actual a la correcta no sea posible. Para corregir se permitirá un pequeño margen en el cual si está dentro de esos márgenes se dará por pieza colocada. Una vez estén colocadas todas las piezas se mensaje de juego finalizado.

3. PUZZLE CON MOVIMIENTO CON ARDUINO

En este apartado se va a sustituir el movimiento de las piezas mediante el teclado por el Arduino.

Para comunicar con el Arduino por el puerto serie es necesario importar dos librerías, *Serial.jar* y *jssc.jar*. Habrá que declarar una variable de clase *Serial*.

Serial miPuerto;

E iniciarla en el *setup()*.

miPuerto = new Serial(this, Serial.list() [“número de puerto en el que se conecta”] ,9600);

Cuando esté la comunicación creada habrá que estar continuamente leyendo el puerto para cuando haya algún valor leerlo. Por lo tanto, habrá que escribir este código:

```
If(miPuerto.avaliable() >0) {
    valor = miPuerto.read();
}
```

Y ahora se puede trabajar con ese valor.

El valor 4 permitirá el desplazamiento de la pieza vertical hacia arriba.

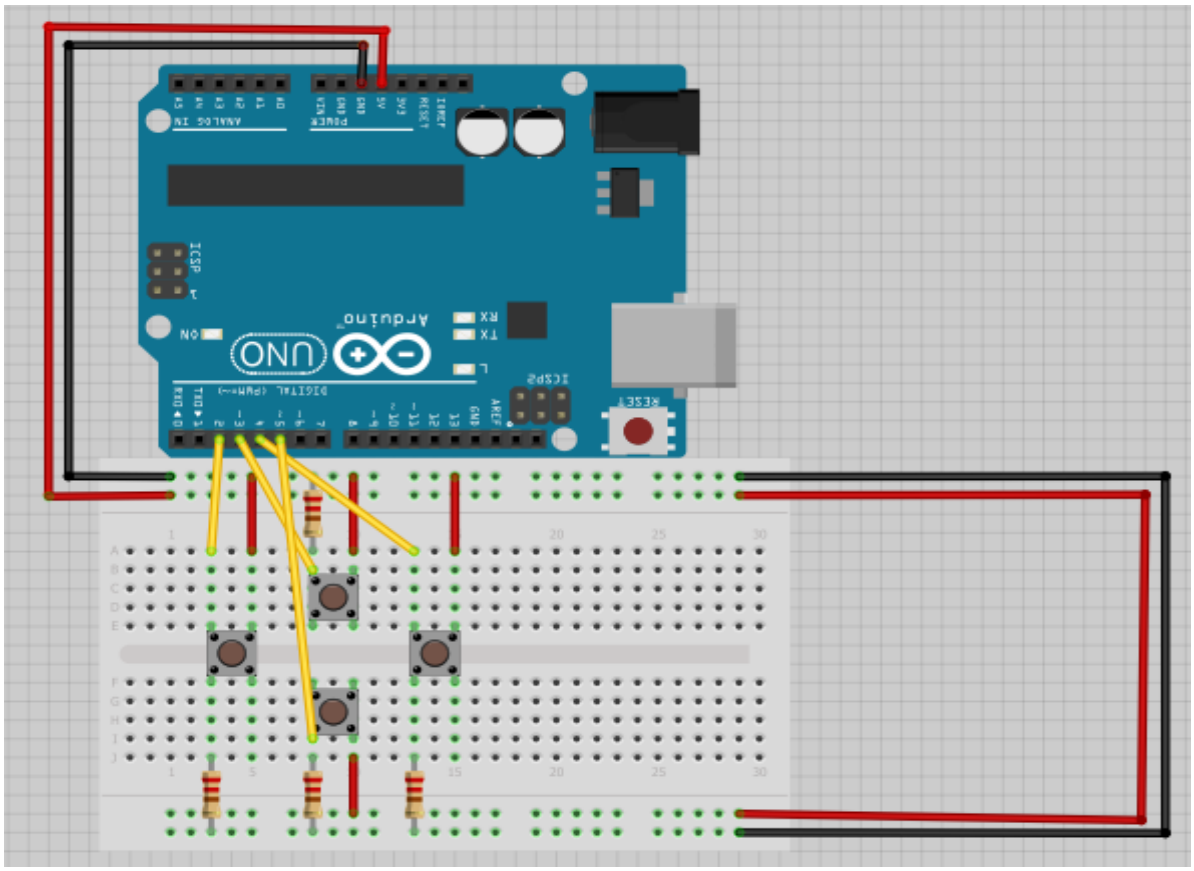
El valor 2 permitirá el desplazamiento de la pieza vertical hacia abajo.

El valor 3 permitirá el desplazamiento horizontal de la pieza hacia la izquierda.

El valor 5 permitirá el desplazamiento horizontal de la pieza hacia la derecha.

Por otro lado, se tendrá que coger el Arduino, montarlo y cargar el programa en la placa.

El programa está colgado en atenea.



Montaje Arduino para la práctica.

PRÁCTICA 7: DOMÓTICA

Objetivos:

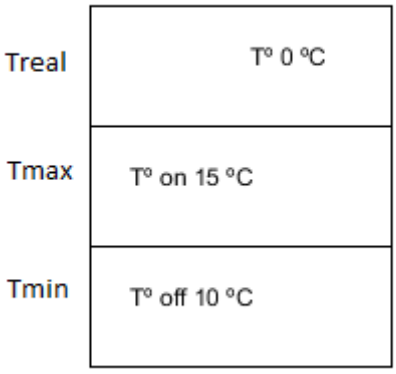
- Trabajar con entradas analógicas en el Arduino.
- Enviar datos desde el programa de Eclipse al Arduino.
- Poner a prueba todos los conocimientos adquiridos.

En esta práctica se va a realizar la simulación de un piso, concretamente la iluminación y el aire acondicionado.

1. AIRE ACONDICIONADO

.En ese primer apartado se va a realizar el funcionamiento del aire acondicionado de una casa. Se utilizará el sensor de temperatura TMP36 del kit de Arduino para medir la temperatura ambiente. Se enviará este dato al programa de manera que se pueda comprobar si se está dentro de los márgenes establecidos. Se establecerán dos límites, uno máximo, el cual una vez superado se activará el aire; y una vez esa temperatura baje por debajo del valor mínimo se desactivará el aire.

Habrá que hacer un recuadro en la pantalla mostrando las dos temperaturas establecidas (la mínima y la máxima) y la temperatura real.



Mando del aire acondicionado de la práctica

Como en la práctica anterior, se habrá de declarar la variable de clase *Serial* e inicializarla en el *setup():*

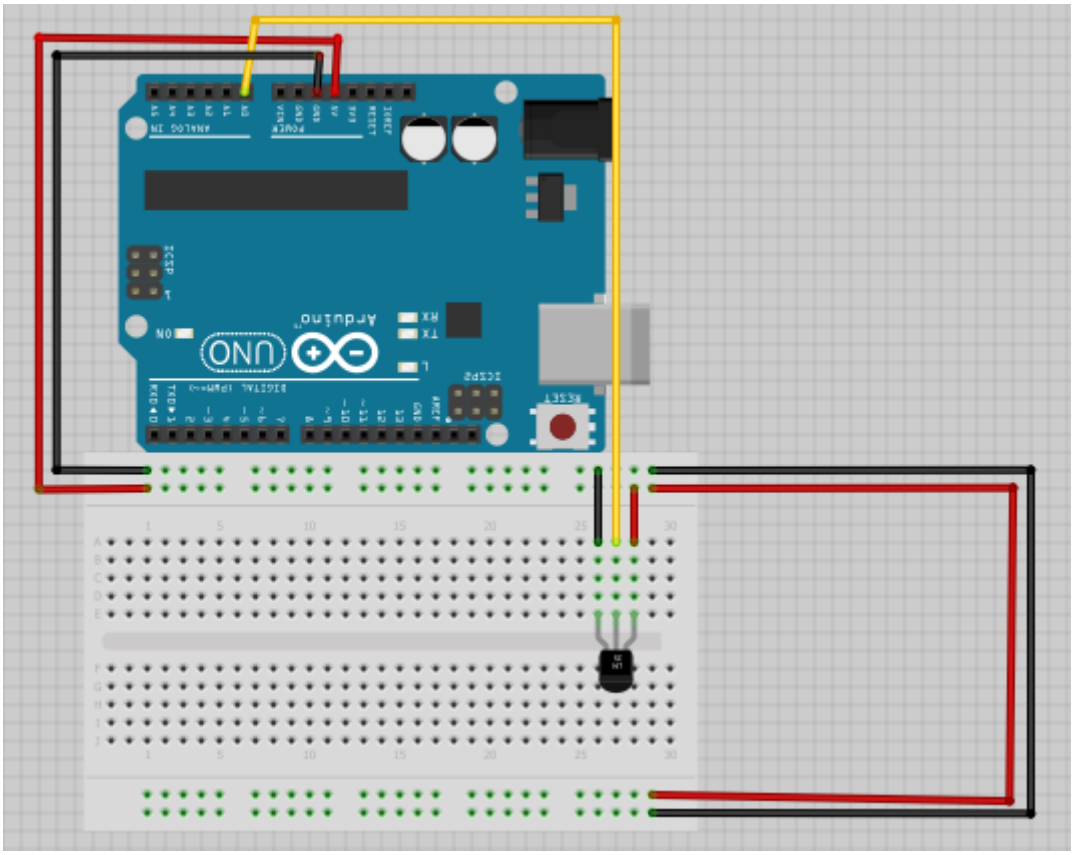
miPuerto = new Serial(this, Serial.list() [“número de puerto en el que se conecta”] ,9600);

Una vez declarado, también habrá que redactar en el método draw(), la lectura del dato de la temperatura que se leerá desde el puerto siempre el puerto esté disponible.

```
If(miPuerto.avaliable() >0) {
        valor = miPuerto.read();
}
```

Por lo tanto, en todo momento se estará leyendo la temperatura, así que hay que hacer un método *activaciónAire()* el cual deberá comprar esta última con los valores prefijados de manera que en cuanto supere el máximo se active el aire y en cuanto esté por debajo del mínimo se vuelva a desactivar. Se tendrá que mostrar por pantalla que el aire está activado de la manera que se crea conveniente, desde un piloto a un texto.

Se deberá que coger el Arduino, montarlo y cargar el programa en la placa. El programa está colgado en atenea.



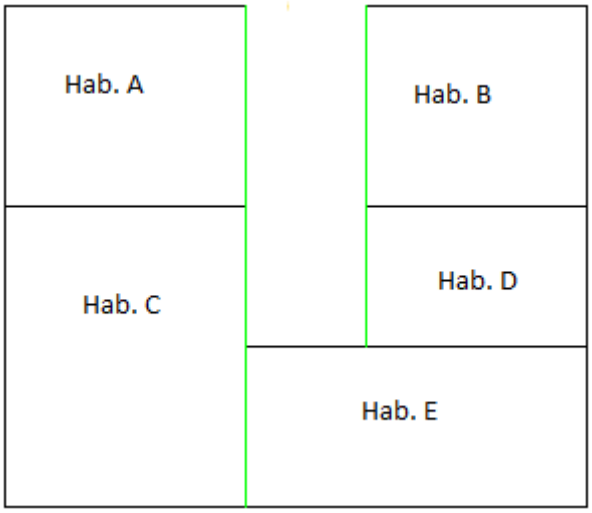
Montaje Arduino para el primer apartado de la práctica.

2. Iluminación de la casa

En este apartado se va simular una casa donde cada habitación tiene un sensor de movimiento el cual cuando detecta enciende la luz.

Se necesitará crear una clase *Habitacion* la cual tendrá como atributos, un nombre (utilizar caracteres como A, B, C…), dos coordenadas, dos longitudes (alto y ancho) y una variable booleana para indicar si hay alguien dentro o no.

Se deberá crear un rectángulo grande a modo de plano de una casa donde dentro tendrá sus habitaciones. Habrá que declarar 5 habitaciones¸ por ejemplo de esta manera:

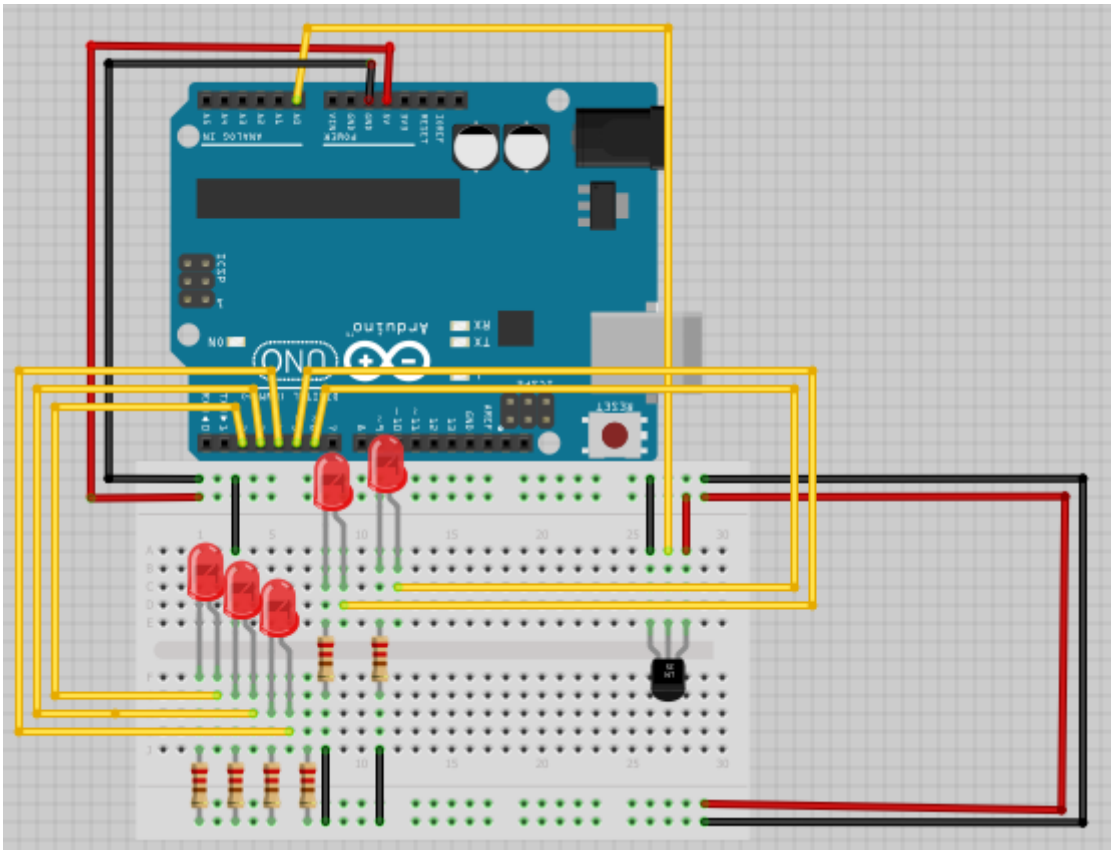


Distribución de las habitaciones de la casa

Se mostrará una pequeña silueta en forma de *PImage* y se posicionará en el centro del pasillo. Con el teclado se permitirá el desplazamiento de este hacia arriba, abajo, izquierda y derecha. No se le permitirá salir de la casa ni atravesar las paredes (líneas de color negro) y si atravesar las puertas (líneas verdes).

El siguiente paso será crear un método *busquedaHabitaciones()* el cual buscará si la imagen está dentro de alguna una habitación, y en caso que sea correcto, se mandará un carácter al Arduino para que encienda un led. El carácter que se mandará será el nombre de la habitación (A, B, C…) ya que con el Arduino se puede enviar o leer un carácter.

Se deberá que coger el Arduino, montarlo y cargar el programa en la placa. El programa está colgado en atenea.



Montaje Arduino para la práctica.

PRÁCTICA 0: INTRODUCCIÓN AL ARDUINO

Objetivos:

- Instalar el Software de Arduino para trabajar en Eclipse.
- Aprender el lenguaje básico para programar en Processing.

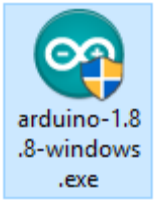
A continuación, se describen los pasos para crear un programa de Arduino:

1. Descargar e instalar el software de Arduino. Dicho programa está disponible en la propia página de Arduino, en el apartado 'software/downloads':

https://www.arduino.cc/en/Main/Software

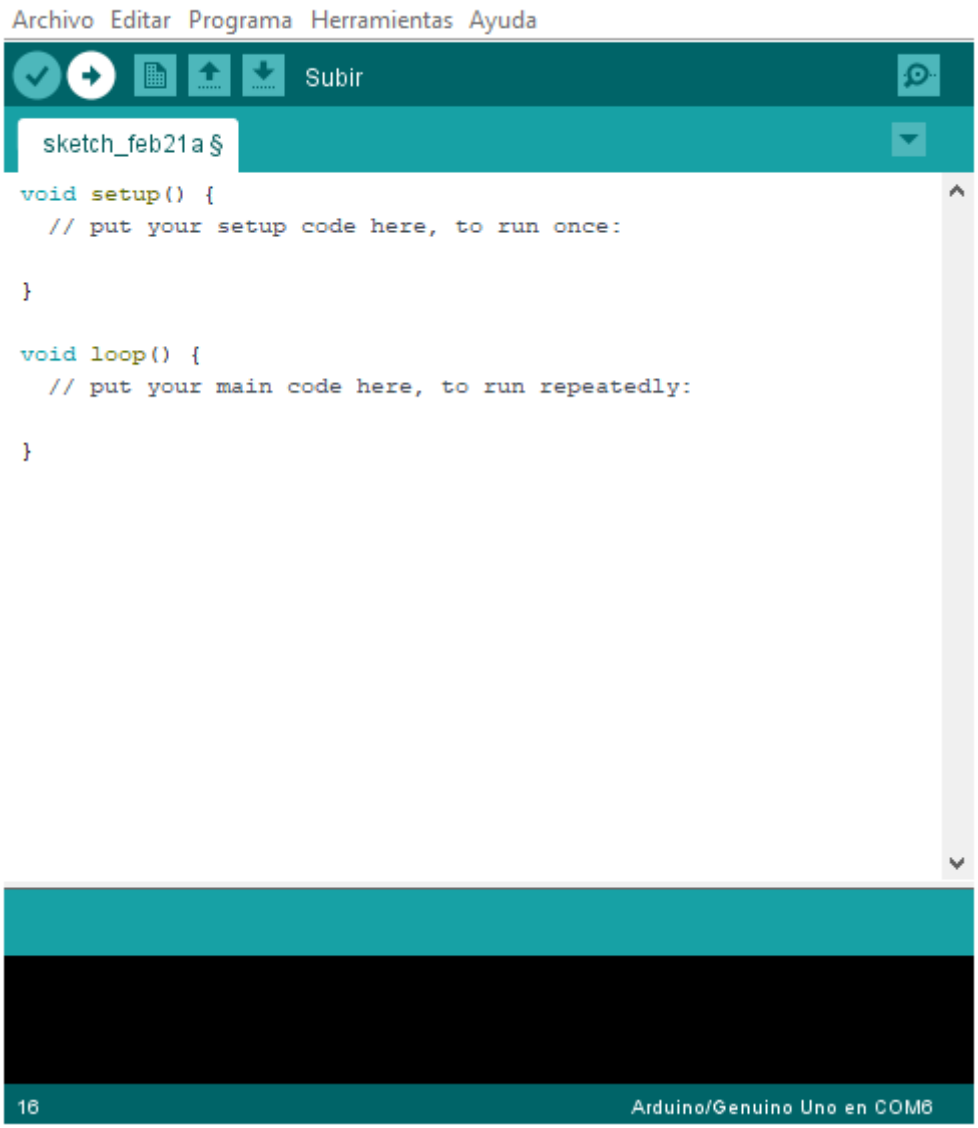
Apartado de 'downloads' de la web de Arduino.

2. Descargar y ejecutar el archivo descargado



Archivo ejecutable para instalar el software de Arduino.

3. Una vez instalado, abrir el software y ya se puede empezar a programar.



Ventana de Arduino lista para programar.

Para programar en Arduino se utilizan dos métodos obligatorios para el funcionamiento del programa:

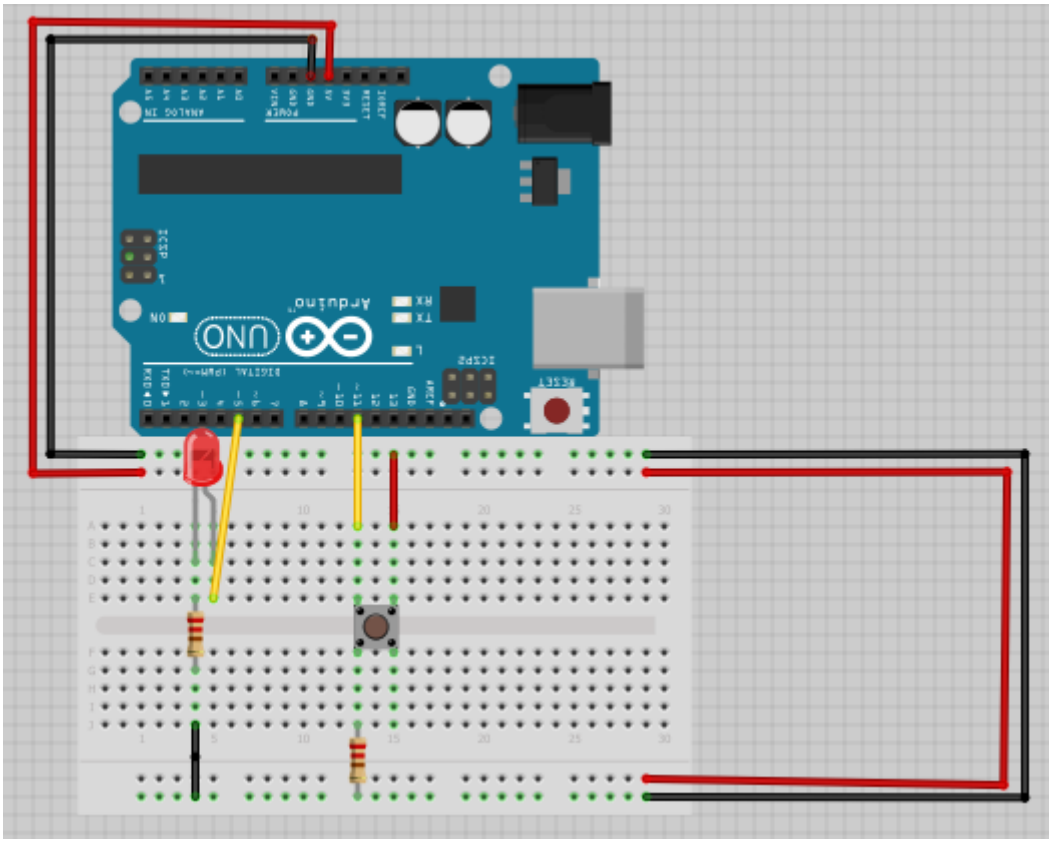
1. *void setup(),* este método solo se ejecutará una vez, la primera vez que se ejecute el programa.
2. *void loop(),* este método se ejecuta infinitamente en bucle siempre que estemos ejecutando el programa

Ejercicio 1:

A continuación se va a hacer una pequeña toma de contacto con el Arduino con un programa muy sencillo. Se encenderá un led siempre que se mantenga pulsado un interruptor.

Primeramente se deberá montar la placa, por un lado habrá que colocar un interruptor de manera que cuando se le pulse se de tensión a uno de los pines digitales de la placa; y por otro lado, un led en serie con una resistencia (para proteger el led de manera que no descargue el solo toda la tensión) de manera que cuando se pulse ese interruptor desde otro pin digital de la placa se de tensión al led para que se mantenga encendido. Una vez se deje de pulsar el pulsador, el led vuelva a apagarse. Es importante que en la salida del interruptor se conecte una resistencia en paralelo de manera que cuando no haya tensión en la entrada del interruptor se escriba 0v en la entrada digital. Se utilizarán resistencias de 220 Ohmios.

Una vez está el montaje hecho, es la hora de hacer el código. Habrá que declarar los pines digitales como entradas o salidas, leer la entrada y en función de esta escribir en la salida o no.



Montaje Arduino para la práctica.

PRÁCTICA 1: EJECUCIÓN PRIMEROS DISEÑOS VAADIN

Objetivos:

- Conocer los diferentes tipos de layouts.
- Utilizar diferentes tipos de layouts.
- *Diseñar una clase con* Vaadin Designer.
- Ejecutar una clase diseñada por Vaadin Designer.

Una vez seguido el manual de instalación de Vaadin y la ejecución de la clase creada por defecto, ahora se creará un propio diseño.

Para ello se utilizará la clase creada solamente para ejecutar el diseño que se creará.

Se abrirá la clase ejemplo creada previamente en *html* donde se colocarán los layouts y los componentes. Y en clase ejemplo.java será donde se declararán los métodos y las acciones necesarias.

Project Explorer > prueba > src > main > resources > com > exemple > prueba > ejemplo.html

Project Explorer > prueba > src > main > java > com > exemple > prueba > ejemplo.java

En la clase MyUI se declarará una clase *Ejemplo* con la cual se mostrará en el contenido en el servidor a través de la función *setContent().*

```
public class MyUI extends UI {

    @Override
    proyecte void init(VaadinRequest vaadinRequest) {

        ejemplo layout = new ejemplo();

     setContent(layout);

    }

     @WebServlet(urlPatterns = “/*”, name = “MyUIServlet”, asyncSupported = true)
```

```
    @VaadinServletConfiguration(ui = MyUI.class, productionMode = false)
    public static class MyUIServlet extends VaadinServlet {
  }
}.
```

En estos momentos ya se podría ejecutar el servidor en el cual se vería el contenido del diseño Ejemplo. Por lo tanto, ya se puede empezar a diseñar la página.

Para este primer apartado se deberá dividir la pantalla en cuatro partes: Arriba izquierda, arriba derecha, abajo izquierda y abajo derecha. Se deberá utilizar los layouts correspondientes para organizar la pantalla.

Habrá que poner cuatro *TextFields*, uno en cada parte de manera que lo que se escriba en el *TextField* de arriba a la izquierda se escriba en el contrario, es decir, en el de abajo a la derecha. Y por otro lado, lo que se escriba en el TextField de arriba a la derecha se escribirá en su contraria, abajo a la izquierda. En los dos *TextFields* inferiores no se permitirá escribir.

PRÁCTICA 1: FORMULARIO

Objetivos:

- Utilizar componentes teóricos como *TextField*, *Button*, *Label*.
- Crear un formulario.
- Utilizar un *TextArea* para mostrar los datos recogidos.
- Validar datos de un formulario.

En esta primera práctica se va diseñar un formulario como si de una inscripción se tratase. Se pedirán datos al usuario y se trabajará con ellos.

1. CREACIÓN DE UN FORMULARIO

Para este primer apartado se creará un formulario. Primeramente habrá que diseñar la clase. En el diseño se añadirán diferentes componentes para recoger los siguientes datos: nombre, apellidos, fecha de nacimiento, si la persona es hombre o mujer, y su nacionalidad. Nombre y apellidos se introducirán como variable *String*; la fecha de cumpleaños con un calendario donde se seleccione el día de nacimiento; hombre o mujer seleccionando una de las dos opciones que aparezcan; y por último, la nacionalidad seleccionándola de una lista desplegable donde aparezcan varios países. Se comprobará que se puedan rellenar todos los campos.

2. LECTURA DE DATOS INTRODUCIDOS

Una vez se ha creado el formulario, se trabajará con los datos. Se leerán y se mostrarán en otra parte de la pantalla. Se deberá añadir un *TextArea*, donde hará una descripción de todos los datos del usuario en forma de *String*. En vez de mostrar los datos uno a uno, se deberá mostrar un solo texto el cual informe de todos los datos.

3. VALIDACIÓN DE DATOS

Para acabar esta práctica, se van a tener que validar estos datos. Es decir, antes de hacer la descripción del persona se deberá que comprobar si los datos son válidos o no. Los siguientes campos serán válidos según las siguientes restricciones:

- Nombre y apellidos deberán tener como mínimo 3 caracteres.
- Nacionalidad y género tienen que haber sido seleccionadas.
- La fecha de nacimiento no puede ser posterior a la actual.

Siempre que se introduzca un dato erróneo se deberá mostrar un mensaje por pantalla indicando el error y no dejará hacer la descripción hasta que todos los datos sean válidos.

PRÁCTICA 2: NOTICIAS

Objetivos:

- Crear la simulación de un muro de noticias.
- Limitar el número de caracteres de los *TextFields*
- Añadir y eliminar componentes de un layout.

Esta práctica se va a crear un muro de noticias en el cual se irán publicando por orden cronológico, es decir, las publicaciones más antiguas se verán adelantadas por las nuevas.

1. MURO

Primeramente se deberá crear el diseño, será un diseño muy sencillo, se dividirá la pantalla horizontalmente en dos partes. La parte izquierda de la pantalla será donde se escribirán las publicaciones y en la parte derecha el muro donde irán publicando las publicaciones. Una vez se tenga la publicación o frase redactada por medio de un Button se publicará la noticia.

Las publicaciones deberán ir publicándose en el muro por orden cronológico, es decir, las más nuevas deberán aparecer siempre en la parte superior; y las antiguas irán bajando hacia la parte inferior. Se deberá poner un límite de publicaciones de manera que si se llega a ese límite, se elimine la publicación más antigua a favor de publicar una noticia nueva de nuevo en la parte superior de la pantalla.

Y por último, se deberá limitar el número de caracteres de la noticia, para no rebasar el ancho de la pantalla.

PRÁCTICA 3: REGISTRO

Objetivos:

- Crear un registro de usuarios con contraseña.
- Crear un archivo de texto en el cual se almacenen datos.
- Leer archivos de texto para cargar datos.
- Iniciar sesión con usuarios.
- Modificar datos de estos.
- Añadir y eliminar usuarios del registro.
- Saltar de un diseño a otro.

Esta práctica consiste en crear un club de usuarios donde puedan registrarse o eliminar su cuenta, y posteriormente puedan iniciar sesión para ver sus datos y modificarlos.

1. CREACIÓN DE LOS DISEÑOS

Para esta práctica se va a necesitar crear varios diseños para posteriormente en la ejecución del programa saltar de uno al otro. Primeramente se debe crear una ventana principal donde se pueda elegir si ir a la página de registros o a la de iniciar sesión.

En la pantalla *Registro* los usuarios que se quieran registrar deberán introducir un usuario y una contraseña, además de otros datos, nombre y descripción como mínimo. Una vez estén todos los campos introducidos se deberá pulsar a botón de registrar, siempre que todos los campos tengan como mínimo 3 caracteres. Todos los usuarios que se registren deberán quedar guardados de manera que cuando se quiera registrar un usuario ya existente no se permita ese registro. En esta pantalla también deberá aparecer un botón el cual elimine todos los usuarios registrados.

Por otro lado, habrá que diseñar la pantalla *InicioSesion*, la cual pedirá al usuario el usuario y la contraseña, si estas son correctas se permitirá abrir el perfil del usuario con otra ventana Perfil. Si el usuario no existe no se permitirá abrir ningún perfil y se mostrará un mensaje indicando por qué no se ha permite el inicio de sesión; si el usuario existe y la contraseña no es correcta, tampoco se permitirá y se mostrará su correspondiente mensaje notificándolo.

En caso que el usuario introduzca sus datos correctamente se abrirá la pantalla de perfil, en la cual se visualizarán los datos del mismo. No se deberá permitir la edición de estos, excepto cuando se pulse el botón editar, en ese caso de permitirá la modificación del nombre y de la descripción. A parte de la modificación, deberá aparecer un botón para cerrar sesión, el cual simplemente dejara de mostrar la ventana de perfil para volver a mostrar la ventana de inicio de sesión; y por último, un botón de eliminar cuenta el cual elimine al usuario de la lista de registros.

Desde las pantallas de *Registro* e *InicioSesion* se permitirá poder ir de una la otra de la manera que se desee.

ANEXO B: SOLUCIONES DE LAS PRÁCTICAS

Processing practica 0

Clase Prueba

```
import processing.core.PApplet; //Se importa la Applet de Processing

public class Prueba extends PApplet {

	public static void main(String[] args) {
		PApplet.main("Prueba");
	}

	int anchoPantalla = 300; //Se declarán las variables
	int alturaPantalla = 100;

	public void settings() {
		size(anchoPantalla, alturaPantalla); //Se da tamaño a la pantalla
	}

	public void setup() {
		background(255); //Color blanco(255) al fondo de pantalla

		fill(0); //Se cambia el color de trabajo por el negro(0)
		//Se escribe el texto en el centro de la pantalla
		text("Hola mundo!", anchoPantalla/2,alturaPantalla/2);
	}
	public void draw() {
	}
}
```

Práctica 1

Clase Formas

```
//Clase Formas
public class Formas{
		//Se declaran las variables de la clase Formas
		private int xCoordenada,yCoordenada;
		private int colorRed, colorGreen,colorBlue;

		//Constructor de la clase Formas
		public Formas(int xCoordenada, int yCoordenada, int colorRed, int colorGreen, int colorBlue) {
			this.xCoordenada = xCoordenada;
			this.yCoordenada = yCoordenada;
			this.colorRed = colorRed;
			this.colorGreen = colorGreen;
			this.colorBlue = colorBlue;
		}

		//Setters y Getters de las variables
		public int getxCoordenada() {
			return xCoordenada;}
		public void setxCoordenada(int xCoordenada) {
			this.xCoordenada = xCoordenada;
		}

		public int getyCoordenada() {
			return yCoordenada;}
		public void setyCoordenada(int yCoordenada) {
			this.yCoordenada = yCoordenada;
		}

		public int getColorRed() {
			return colorRed;}
		public void setColorRed(int colorRed) {
			this.colorRed = colorRed;
		}

		public int getColorBlue() {
			return colorBlue;}
		public void setColorBlue(int colorBlue) {
			this.colorBlue = colorBlue;
		}

		public int getColorGreen() {
			return colorGreen;	}
		public void setColorGreen(int colorGreen) {
```

```
                this.colorGreen = colorGreen;
        }
}
```

Clase Rectangulo

```
//Clase Rectangulo, que hereda de la clase Formas
import processing.core.PApplet; //Se improta la Applet de Processing

public class Rectangulo extends Formas {
	//Se declaran las variables de la clase Rectangulo
	private int rectanguloAlto,rectanguloAncho;

	//Constructor de la clase Rectangulo, utlizando las variables de su clase heredada añadiendo las prévias
	public Rectangulo(int xCoordenada, int yCoordenada,int colorRed, int colorGreen, int colorBlue,
				int rectanguloAlto, int rectanguloAncho) {
		super(xCoordenada, yCoordenada, colorRed, colorBlue, colorGreen);
		this.rectanguloAlto = rectanguloAlto;
		this.rectanguloAncho=rectanguloAncho;
	}

	//Método constructor del rectángulo
	public void construirRectangulo(PApplet app) {
		//Se da color al rectángulo
		app.fill(getColorRed(), getColorGreen(),getColorBlue());
		//Se construye el rectángulo
		app.rect(getxCoordenada()-getRectanguloAlto()/2,getyCoordenada()-
getRectanguloAncho()/2,getRectanguloAlto(),getRectanguloAncho());
	}

	//Setters y Getters de las variables
	public int getRectanguloAncho() {
		return rectanguloAlto;		}
	public void setRectanguloAncho(int rectanguloAncho) {
		this.rectanguloAlto = rectanguloAncho;
	}

	public int getRectanguloAlto() {
		return rectanguloAncho;		}
	public void setRectanguloAlto(int rectanguloAlto) {
		this.rectanguloAncho = rectanguloAlto;
	}
}
```

Clase Circulo

```
//Clase Circulo, que hereda de la clase Formas
import processing.core.PApplet; //Se importa la Applet de Processing

public class Circulo extends Formas {

	//Se declaran las variables de la clase Circulo
	private int radioCirculo,margenOjos;

	//Constructor de la clase Circulo, utlizando las variables de su clase heredada añadiendo las prévias
	public Circulo(int xCoordenada, int yCoordenada,int colorRed, int colorGreen, int colorBlue,int radioCirculo, int
margenOjos) {
		super(xCoordenada, yCoordenada, colorRed, colorBlue, colorGreen);
		this.radioCirculo=radioCirculo;
		this.margenOjos=margenOjos;
	}

	//Método constructor de los círculos
	public void construirCirculo(PApplet app) {
		//Se da color a los círculos
		app.fill(getColorRed(), getColorGreen(),getColorBlue());
		//Se construyen las 2 elipses
		app.ellipse(getxCoordenada()-getMargenOjos(),getyCoordenada()-
getMargenOjos(),getRadioCirculo(),getRadioCirculo());
		app.ellipse(getxCoordenada()+getMargenOjos(),getyCoordenada()-
getMargenOjos(),getRadioCirculo(),getRadioCirculo());
	}

	//Setters y Getters de las variables
	public int getRadioCirculo() {
		return radioCirculo;}
	public void setRadioCirculo(int radioCirculo) {
		this.radioCirculo = radioCirculo;
	}

	public int getMargenOjos() {
		return margenOjos;}
	public void setMargenOjos(int margenOjos) {
		this.margenOjos = margenOjos;
	}
}
```

Clase Triangulo

```
//Clase Triangulo, que hereda de la clase Formas
import processing.core.PApplet; //Se importa la Applet de Processing
public class Triangulo extends Formas {
		//Se decalaran las variables de la clase Triangulo
		private int  margenBocaVertical, anchoBoca, alturaBoca;

		//Constructor de la clase Triangulo, utlizando las variables de su clase heredada añadiendo las prévias
		public Triangulo(int xCoordenada, int yCoordenada, int colorRed, int colorGreen, int colorBlue,int margenBocaVertical,
int anchoBoca, int alturaBoca) {
			super(xCoordenada, yCoordenada, colorRed, colorBlue, colorGreen);
			this.margenBocaVertical=margenBocaVertical;
			this.anchoBoca=anchoBoca;
			this.alturaBoca=alturaBoca;
		}

		//Método constructor deL triángulo
		public void construirTriangulo(PApplet app){
			//Se da color al triángulo
			app.fill(getColorRed(), getColorGreen(),getColorBlue());
			//Se contrsuye el triángulo
			app.triangle(getxCoordenada()-
getAnchoBoca()/2,getyCoordenada()+getMargenBocaVertical(),getxCoordenada()+getAnchoBoca()/2,getyCoordenada()+getMarge
nBocaVertical(),getxCoordenada(),getyCoordenada()+getMargenBocaVertical()+getAlturaBoca());
		}

		//Setters y Getters de las variables
		public int getMargenBocaVertical() {
			return margenBocaVertical;}
		public void setMargenBocaVertical(int margenBocaVertical) {
			this.margenBocaVertical = margenBocaVertical;
		}

		public int getAnchoBoca() {
			return anchoBoca;}
		public void setAnchoBoca(int anchoBoca) {
			this.anchoBoca = anchoBoca;
		}

		public int getAlturaBoca() {
			return alturaBoca;	}
		public void setAlturaBoca(int alturaBoca) {
			this.alturaBoca = alturaBoca;
		}
}
```

Apartado A

```
//Se crea una cara sonriente a partir de diferentes formas
import processing.core.PApplet; //Se importa la Applet de Processing
public class A extends PApplet {
		//Se declaran la altura y ancho de la pantalla
		public static final int altura=680;
		public static final int ancho=680;

		//Coordenadas centrales, la cuales compartiran todas las formas
		int xCoordenada = altura/2;
		int yCoordenada = ancho/2;

		//Con estos tres números, que van de 0 a 255, se podrán combinar entre ellos para poder conseguir los colores deseados
		int colorRed=0;
		int colorBlue=0;
		int colorGreen=0;

		//Datos de las formas
		int rectanguloAncho = 30;
		int rectanguloAltura = 40;
		int radioCirculo = 40;

		//Márgenes de separación entre las formas
		int margenOjos = 150;
		int margenBocaVertical = 80;
		int anchoBoca = 300;
		int alturaBoca = 30;

		//Se crea los objetos para poder crear la cara sonriente
		Rectangulo rect=new Rectangulo(xCoordenada,yCoordenada, colorRed, colorBlue, colorGreen, rectanguloAncho,
rectanguloAltura);
		Circulo circ =new Circulo(xCoordenada,yCoordenada, colorRed, colorBlue, colorGreen, radioCirculo, margenOjos);
		Triangulo tri =new Triangulo(xCoordenada,yCoordenada, colorRed, colorBlue, colorGreen, margenBocaVertical,
anchoBoca, alturaBoca);

		public static void main(String[] args) {
			PApplet.main("A");
		}

		public void settings(){
			size(ancho, altura); //Se da tamaño a la pantalla
		}
```

```
	//Se ejecuta 1 sola vez
	public void setup(){
		background(255); //Color del fondo de la pantalla, 255 equivale a blanco
		construirCara();                                    //Se llama al método
construirPantalla
	}

	public void draw(){

	}
	//Método construirCara
	public void construirCara() {
		rect.setColorRed(255); //Cambiamos el color del rectangulo a rojo, dándole el máximo valor posible
		rect.construirRectangulo(this); //Construimos la nariz de nuestra cara (rectángulo)
		tri.setColorGreen(255); //Cambiamos el color del trangulo a verde, dándole el máximo valor posible
		tri.construirTriangulo(this); //Cosntruimos la boca (un triangulo)
		circ1.setColorBlue(255); //Cambiamos el color del primer círculo a azul, dádole el máximo valor posible
		circ2.setColorBlue(255); //Cambiamos el color del segundo círculo a azul, dádole el máximo valor posible
		circ1.construirCirculo(this); //Construimos el primer círculo
		circ2.construirCirculo(this); //Construimos el segundo círculo
	}
}
```

Apartado B

```
//Se crea el movimiento de una línea
import processing.core.PApplet; //Se importa la Applet de Processing

public class B extends PApplet {

	//Se declara la altura y ancho de la pantalla
	public static final int altura=700;
	public static final int ancho=700;


	int xLinea = 0; //Coordenada x de la línea
	int yLinea = ancho/2; //Coordenada x de la línea
	int velocidad=5; //Velocidad a la cual se quiere que avance la línea
	boolean marcha=false; //Variable que permite el movimiento de la línea

	public static void main(String[] args) {
		PApplet.main("BB");
	}

	public void settings(){
		size(ancho, altura); //Se da tamaño a la pantalla
	}

	//Se ejecuta 1 sola vez
	public void setup(){
		background(0); //Color del fondo de la pantalla, 0 equivale a negro
		stroke(255);		//Se pinta el contorno de color blanco, para que asi la posterior línea sea blanca
		line(xLinea, yLinea,ancho , yLinea); //Se crea la línea
	}

	//Se ejecuta continuamente
	public void draw(){ //En caso que esté habilitada la marcha, permitirá el movimiento a la línea
		if(marcha) {		//Cada vez que se ejecute el draw, se disminuirá la coordenada Y, en la velocidad que
se desee
			//para que se mueva la línea
			yLinea = yLinea - velocidad;

			//En caso que la línea llegue a su punto más alto,
			//volverá empezar su recorrido desde el punto más bajo
			if (yLinea < 0) {
				yLinea = altura;
			}
			background(0); //Color del fondo de la pantalla, 0 equivale a negro
			stroke(255);		//Se pinta el contorno de color blanco, para que asi la posterior líinea sea
blanca
```

```
                        line(xLinea, yLinea,altura , yLinea);  //Se crea la línea continuamente
                }
        }

        //Cada vez que se clique el ratón, se dará marcha o paro al movimiento de la linea
        public void mousePressed(){
                marcha=!marcha;
        }
}
```

Apartado C

```
//Se importa una foto y se permitirá moverla mediante el teclado
import processing.core.PApplet; //Se importa la Applet de Processing
import processing.core.PImage; //Se importa la librería PImage de Processing para poder trabajar con imagenes

public class C extends PApplet {

	public static final int margen=300;
	PImage imagen;
	int ancho,altura;
	int xImagen =0,yImagen = 0;
	int velocidad=1; //Velocidad a la cual se desea que se desplace la imagen
	boolean marcha=false;

	public static void main(String[] args) {
		PApplet.main("C");
	}

	public void settings(){
		imagen=loadImage("logo.png"); //Se carga la imagen deseada
		//El ancho de la pantalla será el ancho de la imagen más un pequeño margen
		ancho=imagen.width+margen; //La altura de lapantalla será la latura de la imagen más un pequeño
margenaltura=imagen.height+margen;
		size(ancho,altura); //Se da tamaño a la pantalla

		//Se centran las coordenadas de la imagen para que aparezca en el centro de la pantalla
		xImagen=(ancho-imagen.width)/2;
		yImagen=(altura-imagen.height)/2;
	}

	//Se ejecuta 1 sola vez
	public void setup(){
		background(255); //Color del fondo de la pantalla, 255 equivale a blanco
		image(imagen,xImagen,yImagen); //Se indica que imagen se va a mostar, y en que coordenadas
	}

	//Se ejecuta continuamente
	public void draw(){
		background(255); //Color del fondo de la pantalla, 255 equivale a blanco
		image(imagen,xImagen,yImagen); //Se indica que imagen se va a mostar, y en que coordenadas
	}

	//Cuando se pulsa una tecla entra en la función
	public void keyPressed(){		//En caso de que la tecla pulsada sea la W, el logo se desplazará hacia arriba
		//Se limita el borde superior de la pantalla para que no desaparezca la imagen
		if((key=='w'||key=='W')&& yImagen>0){
```

```
                    yImagen-=velocidad;; //Se disminuye la coordenada Y
                }

                //En caso de que la tecla pulsada sea la S, el logo se desplazará hacia abajo
                //Se limita el borde inferior de la pantalla para que no desaparezca la imagen

                else if((key=='s'||key=='S')&& yImagen<margen){
                    yImagen+=velocidad; //Se aumenta la coordenada Y
                }

                //En caso de que la tecla pulsada sea la A, el logo se desplazará hacia la izquierda
                //Se limita el borde izquierdo de la pantalla para que no desaparezca la imagen

                else if((key=='a'||key=='A')&& xImagen>0){ xImagen-=velocidad; //Se disminuye la coordenada X
                }

                //En caso de que la tecla pulsada sea la W, el logo se desplazará hacia la derecha
                //Se limita el borde derecho de la pantalla para que no desaparezca la imagen

                else if((key=='d'||key=='D')&& xImagen<margen){ xImagen+=velocidad; //Se aumenta la coordenada X
                }
            }
}
```

Práctica 2

Apartado A

```
//Se crea un aplicación que permite dibujar con el ratón
import processing.core.PApplet; //Se importa la Applet de Processing

public class A extends PApplet{

	//Se declara la altura y ancho de la pantalla
	public static final int altura=400;
	public static final int ancho=400;

	//Se declaran más variables
	boolean encimaArea=false;

	public static void main(String[] args) {
		PApplet.main("A");
	}

	public void settings(){
		construirPantalla(ancho, altura); //Se llama al método construirPantalla
	}

	//Se ejecuta 1 sola vez
	public void setup(){
		construirParteSuperior(ancho,altura); //Se llama al método construirParteSuperior
	}

	//Se ejecuta contínuamente
	public void draw(){ //Se sabra si el ratón está dentro del rectángulo superior de la pantalla
		if( mouseX >= 0 && mouseX <= ancho && mouseY >= 0 && mouseY <= altura/8){
			encimaArea=true;}
		else{
			encimaArea=false;}
	}

	//Cuando se clique y se mueva el ratón simultáneamente
	public void mouseDragged() { //Si el ratón no está dentro del rectángulo superior, se podrá dibujar
		if(!encimaArea && mouseY > altura/8) {
			dibujar(mouseX, mouseY, pmouseX, pmouseY); //Se llama al método Dibujar
		}
	}

	//Método construirPantalla
	public void construirPantalla(int x, int y) {
		size(x,y); //Se da tamaño a la pantalla
	}
```

```
	//Método construirParteSuperior
	public void construirParteSuperior(int x, int y) {
		background(255); //Color de fondo de la pantalla, 255 equivale a blanco
		fill(0); //Letras negras

		//Se escribe un pequeño texto en unas determianadas coordenadas
		text("Mueve el ratón para dibujar", x/8*3, y/16);
		line(0,y/8,x,y/8);	//Línea separadora
	}

	//Método dibujar
	public void dibujar(int x, int y, int a, int b) {
		line(x,y,a,b); //Se crea una línea en esas coordenadas
	}
}
```

Apartado B

```
//Se crea un aplicación que permite dibujar con el ratón y a diferentes espesores
import processing.core.PApplet; //Se importa la Applet de Processing

public class B extends PApplet{
	//Se declara la altura y el ancho de la pantalla
	public static final int altura=800;
	public static final int ancho=800;

	//Declaración variables
	boolean encimaArea=false;

	//Datos
	int xRectanguloEspesor=700;
	int yRectanguloEspesor=30;
	int alturaRectanguloEspesor=60;
	int anchoRectanguloEspesor=60;

	public static void main(String[] args) {
		PApplet.main("B");
	}

	public void settings(){
		construirPantalla(ancho, altura); //Se llama al método construirPantalla
	}

	//Se ejecuta 1 sola vez
	public void setup(){
		//Se llama al método construirParteSuperior
		construirParteSuperior(ancho,altura);

		//Se llama al método construirZonaEspesor
	construirZonaEspesor(xRectanguloEspesor,yRectanguloEspesor,anchoRectanguloEspesor,alturaRectanguloEspesor,1);
	}

	//Se ejecuta continuamente
	public void draw(){ //Se sabrá si el ratón está en le rectángulo superior de la pantalla
		if((mouseX >= 0 && mouseX <= ancho && mouseY >= 0 && mouseY <= altura/8) ){
			encimaArea=true;
		}
		else {
			encimaArea  =false;
		}
	}

	//Cuando se clique y se mueva el ratón simultáneamente
	public void mouseDragged() { //Si el ratón no está dentro del rectangulo superior, se podrá dibujar
```

```
                if(!encimaArea && mouseY > altura/8) {
                        dibujar(mouseX, mouseY, pmouseX, pmouseY); //Se llama al método Dibujar
                }
        }

        //Cada vez que cliquemos el ratón entraremos en el método
        public void mousePressed() { //Se sabrá si el ratón está en la zona espesor pequeño
                if(mouseX > xRectanguloEspesor && mouseX < xRectanguloEspesor+anchoRectanguloEspesor && mouseY >
yRectanguloEspesor && mouseY <yRectanguloEspesor+alturaRectanguloEspesor/3){ //Se llama al método construir zona espesor

        construirZonaEspesor(xRectanguloEspesor,yRectanguloEspesor,anchoRectanguloEspesor,alturaRectanguloEspesor,1);
                }
                //Se sabrá si el ratón está en la zona espesor mediano
                else if(mouseX > xRectanguloEspesor && mouseX < xRectanguloEspesor+anchoRectanguloEspesor &&
mouseY > yRectanguloEspesor+alturaRectanguloEspesor/3 && mouseY <yRectanguloEspesor+alturaRectanguloEspesor/3*2){ //Se
llama al método construirZonaEspesor

        construirZonaEspesor(xRectanguloEspesor,yRectanguloEspesor,anchoRectanguloEspesor,alturaRectanguloEspesor,2);
                }
                //Se sabrá si el ratón está en la zona espesor grande
                else if(mouseX > xRectanguloEspesor && mouseX < xRectanguloEspesor+anchoRectanguloEspesor &&
mouseY > yRectanguloEspesor+alturaRectanguloEspesor/3*2 && mouseY <yRectanguloEspesor+alturaRectanguloEspesor){ //Se
llama al método construirZonaEspesor

        construirZonaEspesor(xRectanguloEspesor,yRectanguloEspesor,anchoRectanguloEspesor,alturaRectanguloEspesor,3);
                }
        }

        //Método construirPantalla
        public void construirPantalla(int x, int y) {
                size(x,y); //Damos tamaño a la pantalla
        }

        //Método construirParteSuperior
        public void construirParteSuperior(int x, int y) {
                background(255); //Color de fondo de la pantalla, 255 equivale a blanco
                fill(0); //Letras negras
                text("Mueve el ratón para dibujar", x/8*3, y/16); //Escribimos un pequeño texto en unas determinadas
coordenadas
                line(0,y/8,x,y/8);      //Línea separadora
        }

        //Método dibujar
        public void dibujar(int x, int y, int a, int b) {
                line(x,y,a,b); //Se crea una línea en con esas coordenadas
        }

        //Método construirZonaEspesor
```

```
	public void construirZonaEspesor(int xrectanguloespesor, int yrectanguloespesor,
				int anchorectanguloespesor, int alturarectanguloespesor, int i) {
			//Se llama al método construirRectánguloGrandeEspesor

	construirRectanguloGrandeEspesor(xrectanguloespesor,yrectanguloespesor,anchorectanguloespesor,alturarectangulo
espesor);

			switch (i) { //En función de la variable i, se seleccionaá un tamaño u otro
			case 1:
				strokeWeight(1); //Grosor línea 1
				stroke(255,0,0); //Contorno rojo

				//Rectángulo rojo en el espesor seleccionado
	rect(xrectanguloespesor,yrectanguloespesor,anchorectanguloespesor,alturarectanguloespesor/6*2);
				stroke(0); //Contorno negro
				//Línea de grossor 1
				line(xrectanguloespesor + anchorectanguloespesor/6,
yrectanguloespesor+alturarectanguloespesor/6, xrectanguloespesor+anchorectanguloespesor/6*5,
yrectanguloespesor+alturarectanguloespesor/6);
				break;
			case 2:
				stroke(255,0,0); //Contorno rojo

				//Rectángulo rojo en el espesor seleccionado
	rect(xrectanguloespesor,yrectanguloespesor+alturarectanguloespesor/6*2,anchorectanguloespesor,alturarectanguloes
pesor/3);
				stroke(0); //Contorno negro
				strokeWeight(4); //Grosor línea 4
				//Línea de grossor 4
				line(xrectanguloespesor+anchorectanguloespesor/6,
yrectanguloespesor+alturarectanguloespesor/6*3, xrectanguloespesor+anchorectanguloespesor/6*5,
yrectanguloespesor+alturarectanguloespesor/6*3);
				break;
			case 3:
				stroke(255,0,0); //Contorno rojo

				//Rectángulo rojo en el espesor seleccionado
	rect(xrectanguloespesor,yrectanguloespesor+alturarectanguloespesor/6*4,anchorectanguloespesor,alturarectanguloes
pesor/3); stroke(0); //Contorno negro
				strokeWeight(10); //Grosor línea 10
				//Línea de grossor 10
				line(xrectanguloespesor +anchorectanguloespesor/6,
yrectanguloespesor+alturarectanguloespesor/6*5,xrectanguloespesor+anchorectanguloespesor/6*5,
yrectanguloespesor+alturarectanguloespesor/6*5);
				break;
			default:
				break;
			}
```

```
}

//Método construirRectanguloGrandeEspesor
public void construirRectanguloGrandeEspesor(int a,int b,int c,int d) {
        strokeWeight(1); //Grosor línea 1
        fill(255); //Relleno blanco
        stroke(0); //Contorno negro
        rect(a,b,c,d); //Rectángulo grande

        //Líneas con su grosor
        strokeWeight(10); //Grosor línea 10
        line(a +c/6, b+d/6*5, a+c/6*5, b+d/6*5); //Línea de grossor 10
        strokeWeight(4);  //Grosor línea 4
        line(a +c/6 , b+d/6*3, a+c/6*5, b+d/6*3); //Línea de grossor 4
        strokeWeight(1); //Grosor línea 1
        line(a+c/6,b+d/6, a+c/6*5, b+d/6); //Línea de grossor 1

}
```

Apartado C

```
//Se crea un aplicación que permite dibujar con el ratón y a diferentes espesores con la posibilidad de borrar
import processing.core.PApplet;  //Se importa la Applet de Processing

public class C extends PApplet{ //Se declaran la altura y el ancho de la pantalla
		public static final int altura=800;
		public static final int ancho=800;

		// Declaración varables
		boolean estado=false;
		boolean encimaArea=false;

		//Datos
		int xRectanguloEspesor=700;
		int yRectanguloEspesor=30;
		int alturaRectanguloEspesor=60;
		int anchoRectanguloEspesor=60;
		int xRectanguloBorrar= 550;
		int yRectanguloBorrar=30;
		int alturaRectanguloBorrar=60;
		int anchoRectanguloBorrar=60;

		public static void main(String[] args) {
			PApplet.main("C");
		}

		public void settings(){
			construirPantalla(ancho, altura); //Se llama al método construirPantalla
		}

		//Se ejecuta 1 sola vez
		public void setup(){
			//Se llama al método construirParteSuperior
			construirParteSuperior(ancho,altura);

			//Se llama al método construirZonaBorrar
		construirZonaBorrar(xRectanguloBorrar,yRectanguloBorrar,anchoRectanguloBorrar,alturaRectanguloBorrar,2);
			fill(0); //Letras negras
			text("Borrar", altura/8*5, ancho/16);		//Se escribe un pequeño texto en unas determinadas
coordenadas

			//Se llama al método construirZonaEspesor

		construirZonaEspesor(xRectanguloEspesor,yRectanguloEspesor,anchoRectanguloEspesor,alturaRectanguloEspesor,1);
		}
```

```
	//Se ejecuta continuamente
	public void draw(){ //Se sabrá si el ratón está en le rectángulo superior de la pantalla
		if(mouseX >= 0 && mouseX <= ancho && mouseY >= 0 && mouseY <= altura/8 ){
			encimaArea=true;
		}
		else {
			encimaArea =false;
		}
	}

	//Cuando se clique y se mueva el ratón simultáneamente
	public void mouseDragged() {
		//Si el ratón no está dentro del rectángulo superior, se podrá dibujar
		if(!encimaArea && mouseY > altura/8+15) {
			if(estado) {		//En función de la variable estado, dibuajeremos o borraremos

				borrar(mouseX,mouseY, 30); //Se llama al método Borrar
			}
			else {
				dibujar(mouseX, mouseY, pmouseX, pmouseY); //Se llama al método dibujar
			}
		}
	}

	//Cada vez que se clique el ratón
	public void mousePressed() {
		//Se sabrási el ratón está en la zona espesor pequeño
		if(mouseX > xRectanguloEspesor && mouseX < xRectanguloEspesor+anchoRectanguloEspesor && mouseY >
yRectanguloEspesor && mouseY <yRectanguloEspesor+alturaRectanguloEspesor/3){

			//Se llama al método construir zona espesor
	construirZonaEspesor(xRectanguloEspesor,yRectanguloEspesor,anchoRectanguloEspesor,alturaRectanguloEspesor,1);
		}
		//Se sabrá si el ratón está en la zona espesor mediano
		else if(mouseX > xRectanguloEspesor && mouseX < xRectanguloEspesor+anchoRectanguloEspesor &&
mouseY > yRectanguloEspesor+alturaRectanguloEspesor/3 && mouseY <yRectanguloEspesor+alturaRectanguloEspesor/3*2){

			//Se llama al método construirZonaEspesor
	construirZonaEspesor(xRectanguloEspesor,yRectanguloEspesor,anchoRectanguloEspesor,alturaRectanguloEspesor,2);
		}
		//Se sabrá si el ratón está en la zona espesor grande
		else if(mouseX > xRectanguloEspesor && mouseX < xRectanguloEspesor+anchoRectanguloEspesor &&
mouseY > yRectanguloEspesor+alturaRectanguloEspesor/3*2 && mouseY <yRectanguloEspesor+alturaRectanguloEspesor){

			//Se llama al método construirZonaEspesor
	construirZonaEspesor(xRectanguloEspesor,yRectanguloEspesor,anchoRectanguloEspesor,alturaRectanguloEspesor,3);
		}
		//Se sabrá si el ratón está en la zona borrar
```

```
                else if((mouseX >= xRectanguloBorrar && mouseX <= xRectanguloBorrar+anchoRectanguloBorrar && mouseY
>= yRectanguloBorrar && mouseY <= yRectanguloBorrar+alturaRectanguloBorrar) ){
                        if(!estado) {  //En función de la variable estado, se mostrará la zona borrar activada o desactivada
                                estado=true;
                                 //Se llama al método construirZonaBorrar
                                construirZonaBorrar(xRectanguloBorrar,yRectanguloBorrar,   anchoRectanguloBorrar,
alturaRectanguloBorrar, 1);
                        }
                        else {
                                estado=false;
                                //Se llama al método construirZonaBorrar
                                construirZonaBorrar(xRectanguloBorrar,  yRectanguloBorrar,  anchoRectanguloBorrar,
alturaRectanguloBorrar, 2);
                        }
                }
        }

        //Método construirPantalla
        public void construirPantalla(int x, int y) {
                size(x,y); //Se da tamaño a la pantalla
        }

        //Método construirParteSuperior
        public void construirParteSuperior(int x, int y) {
                background(255); //Color de fondo de la pantalla, 255 equivale a blanco
                fill(0); //Letras negras

                //Se escribe un pequeño texto en unas determinadas coordenadas
                text("Mueve el ratón para dibujar", x/8*3, y/16);                line(0,y/8,x,y/8);        //Línea separadora
        }

        //Método dibujar
        public void dibujar(int x, int y, int a, int b) {
                line(x,y,a,b); //Se crea una línea en esas coordenadas
        }

        //Método construirZonaEspesor
        public void construirZonaEspesor(int xrectanguloespesor, int yrectanguloespesor,
                        int anchorectanguloespesor, int alturarectanguloespesor, int i) {

//Se llama al método construirRectánguloGrandeEspesor
        construirRectanguloGrandeEspesor(xrectanguloespesor,yrectanguloespesor,anchorectanguloespesor,alturarectangulo
espesor);

                switch (i) {                //En función de la variable i, se seleccionará un tamaño u otro
                case 1:
                        strokeWeight(1); //Grosor línea 1
                        stroke(255,0,0); //Contorno rojo
```

```
                        //Rectángulo rojo en el espesor seleccionado

        rect(xrectanguloespesor,yrectanguloespesor,anchorectanguloespesor,alturarectanguloespesor/6*2);
                        stroke(0); //Contorno negro

                        //Línea de grossor 1
                        line(xrectanguloespesor + anchorectanguloespesor/6,
yrectanguloespesor+alturarectanguloespesor/6, xrectanguloespesor+anchorectanguloespesor/6*5,
yrectanguloespesor+alturarectanguloespesor/6);
                        break;
                case 2:
                        stroke(255,0,0); //Contorno rojo

                        //Rectángulo rojo en el espesor seleccionado
        rect(xrectanguloespesor,yrectanguloespesor+alturarectanguloespesor/6*2,anchorectanguloespesor,alturarectanguloes
pesor/3);
                        stroke(0); //Contorno negro
                        strokeWeight(4); //Grosor línea 4

                        //Línea de grossor 4
                        line(xrectanguloespesor+anchorectanguloespesor/6,
yrectanguloespesor+alturarectanguloespesor/6*3, xrectanguloespesor+anchorectanguloespesor/6*5,
yrectanguloespesor+alturarectanguloespesor/6*3);
                        break;
                case 3:
                        stroke(255,0,0); //Contorno rojo

                        //Rectángulo rojo en el espesor seleccionado
        rect(xrectanguloespesor,yrectanguloespesor+alturarectanguloespesor/6*4,anchorectanguloespesor,alturarectanguloes
pesor/3); stroke(0); //Contorno negro
                        strokeWeight(10);  //Grosor línea 10

//Línea de grossor 10
                        line(xrectanguloespesor +anchorectanguloespesor/6,
yrectanguloespesor+alturarectanguloespesor/6*5,xrectanguloespesor+anchorectanguloespesor/6*5,
yrectanguloespesor+alturarectanguloespesor/6*5);
                        break;
                default:
                        break;
                }
        }

        //Método construirRectanguloGrandeEspesor
        public void construirRectanguloGrandeEspesor(int a,int b,int c,int d) {
                strokeWeight(1); //Grosor línea 1
                fill(255); //Relleno blanco
                stroke(0); //Contorno negro
                rect(a,b,c,d); //Rectángulo grande
```

```
                //Líneas con su grosor
                strokeWeight(10); //Grosor línea 10
                line(a +c/6, b+d/6*5, a+c/6*5, b+d/6*5); //Línea de grossor 10
                strokeWeight(4); //Grosor línea 4
                line(a +c/6 , b+d/6*3, a+c/6*5, b+d/6*3); //Línea de grossor 4
                strokeWeight(1); //Grosor línea 1
                line(a+c/6,b+d/6, a+c/6*5, b+d/6); //Línea de grossor 1

        }

        //Método borrar
        public void borrar(int x, int y, int a) {
                ellipse(x,y,a,a); //Se crea una ellipse en esas coordenadas
        }

        //Método construirZonaBorrar
        public void construirZonaBorrar(int xrectanguloborrar, int yrectanguloborrar, int anchorectanguloborrar, int
alturarectanguloborrar,int r){
                switch (r) {            //En función de la variables r, seleccionaremos un rectangulo u otro
                case 1:
                        fill(0, 255,0); //Relleno color verde
                        strokeWeight(4); //Grosor línea 4

                        //Rectángulo con contorno grosor 4 y relleno verde
        rect(xrectanguloborrar,yrectanguloborrar,anchorectanguloborrar,alturarectanguloborrar);
                        stroke(255);            //Contorno blanco
                        fill(255); //Relleno blanco
                        strokeWeight(1); //Grosor línea 1
                        break;

                case 2:
                        stroke(0); //Contorno negro
                        fill(255); //Relleno blanco
                        strokeWeight(1); //Grosor línea 1

                        //Rectángulo con contorno grosor 1 y relleno blanco
        rect(xrectanguloborrar,yrectanguloborrar,anchorectanguloborrar,alturarectanguloborrar);
                        break;
                default:
                        break;
                }
        }
}
```

Práctica 3

Clase Esfera

```
//Se crea la clase Esfera con sus diferentes métodos
import java.util.concurrent.ThreadLocalRandom; //Se importa el la librería random de java
import processing.core.PApplet; //Se importa la Applet de Processing
import processing.core.PVector; //Se importa la librería de PVector de Processing

class Esfera {
	//Se declaran las variables de la clase Esfera
	PVector posicion;
	private int radio;
	private int xDireccion, yDireccion; //Será la velocidad con la que se desplace la esfera
	private int color1,color2,color3;
	boolean choque = false;

	//Constructor de la clase Esfera
	public Esfera(PApplet app, int xCoordenadaInicial, int yCoordenadaInicial, int radioCirculo, int colourRed, int
colourGreen,int colourBlue) {
		posicion = new PVector(xCoordenadaInicial, yCoordenadaInicial);
		this.radio = radioCirculo;
		this.xDireccion=ThreadLocalRandom.current().nextInt(-10,10);
		this.yDireccion=ThreadLocalRandom.current().nextInt(-10,10);
		this.color1=colourRed;
		this.color2=colourGreen;
		this.color3=colourBlue;
	}

	//Método display, muestra la elipse por pantalla
	void display(PApplet app) {
		app.noStroke(); //Sin contorno
		app.fill(getColor1(),getColor2(),getColor3()); //Se rellena co el color que corresponda
		app.ellipse(getPosicion().x,getPosicion().y, getRadio()*2, getRadio()*2); //Se muestra la elipse por pantalla
	}

	//Método update, cambia la posición de la esfera
	void update() {
		getPosicion().add(getxDireccion(),getyDireccion());	//Se cambia la posición de la esfera
	}

	//Método choqueConPared, cuando la esfera choca con la pared, rebota
	void choqueConPared(float altura, float ancho) {

		//Si choca con la pared derecha, se cambia la dirección hacia izquierda
		if (getPosicion().x > ancho-getRadio()) {
			getPosicion().x= ancho-getRadio();
			setxDireccion(getxDireccion()*-1);
		}
```

```
		//Si choca con la pared izquierda, se cambia la dirección hacia derecha
		else if (getPosicion().x < getRadio()) {
			getPosicion().x= getRadio();
			setxDireccion(getxDireccion()*-1);
		}

		//Si choca con la pared inferior, se cambia la dirección hacia arriba
		else if (getPosicion().y > altura-getRadio()) {
			getPosicion().y= altura-getRadio();
			setyDireccion(getyDireccion()*-1);
		}

		//Si choca con la pared superior, se cambia la dirección hacia abajo
		else if (getPosicion().y < getRadio()) {
			getPosicion().y= getRadio();
			setyDireccion(getyDireccion()*-1);
		}
	}

	//Método choqueEntreEsferas, cuando dos esferas se cruzan, estas cambian de color aleatóriamente
	void cruceEntreEsferas(PApplet app, Esfera other,float altura, float ancho) {
		//Se calcula la diferencia entre las 2 esferas
		PVector distanceVect = PVector.sub(other.getPosicion(),getPosicion());

		//Se calcula la magnitud del vector
		float distanceVectMag = distanceVect.mag();

		//Se calcula la distancia mínima teniendo cuenta el radio de ambas
		float minDistance = getRadio() + other.getRadio();

		//Si la distancia entre las 2 esferas es menor que la mínima, están en contacto, y por lo tanto
		//se cambia el color de ésta

		if (distanceVectMag < minDistance) {
			if(!choque) { //Este método es para que cambie solamente una vez hasta que dejen de chocarse
				setColor1(ThreadLocalRandom.current().nextInt(0,255));
				setColor2(ThreadLocalRandom.current().nextInt(0,255));
				setColor3(ThreadLocalRandom.current().nextInt(0,255));
				choque=true;
			}
		}
		else {
			choque=false;
		}
	}

	//Setters y Getters de las variables
```

```
public PVector getPosicion() {
		return posicion;}
public void setPosicion(PVector posicion) {
		this.posicion = posicion;
}

public int getRadio() {
		return radio;}
public void setRadio(int radio) {
		this.radio = radio;
}

public int getColor1() {
		return color1;}
public void setColor1(int color1) {
		this.color1 = color1;
}

public int getColor2() {
		return color2;}
public void setColor2(int color2) {
		this.color2 = color2;
}

public int getColor3() {
		return color3;}
public void setColor3(int color3) {
		this.color3 = color3;
}

public int getxDireccion() {
		return xDireccion;}
public void setxDireccion(int xDireccion) {
		this.xDireccion = xDireccion;
}

public int getyDireccion() {
		return yDireccion;}
public void setyDireccion(int yDireccion) {
		this.yDireccion = yDireccion;
}
```

Apartado A

```
//Se crea una esfera que se mueve por pantalla y rebota en las paredes
import processing.core.PApplet;  //Se importa la Applet de Processing

public class A extends PApplet{
	//Declaración altura y ancho de la pantalla
	public static final int altura=200;
	public static final int ancho=300;

	//Datos
	int colorRed=255;
	int colorGreen=0;
	int colorBlue=0;
	int radioCirculo=50;

	//Declaración variables, 1 esfera
	Esfera esfera;

	public static void main(String[] args) {
		PApplet.main("A");
	}

	public void settings(){
		//Creamos una esfera
		esfera = new Esfera(this,altura/2, altura/2,radioCirculo,colorRed,colorGreen,colorBlue);

		construirPantalla(ancho, altura); //Se llama al método construirPantalla
	}

	public void setup(){
		background(0); //Color de fondo de la pantalla, 0 equivale a negro
	}

	public void draw(){
		background(0); //Color de fondo de la pantalla, 0 equivale a negro
		esfera.display(this); //Se llama al método display
		esfera.update(); //Se llama al método update
		esfera.choqueConPared(altura, ancho); //Se llama al método choqueConPared
	}

	//Método construirPantalla
	public void construirPantalla(int x, int y){
		size(x,y);  //Se da tamaño a la pantalla
	}
}
```

Apartado B

```
//Se crean dos esferas que se mueven por pantalla y rebotan en las paredes
import processing.core.PApplet; //Se importa la Applet de Processing
public class B extends PApplet{

	//Declaración altura y ancho de la pantalla
	public static final int altura=400;
	public static final int ancho=600;

	//Datos
	int colorRed=255;
	int colorGreen=0;
	int colorBlue=0;
	int radioCirculo = 50;

	public static void main(String[] args) {
		PApplet.main("B");
	}

	//Declaración variables, 2 Ball
	Esfera bola1;
	Esfera bola2;

	public void settings(){
		bola1 = new Esfera(this,altura/2, altura/2,radioCirculo,colorRed,colorGreen,colorBlue); //Se crea Ball1
		bola2 = new Esfera(this,altura/4, altura/4,radioCirculo*6/10,colorRed,colorGreen,colorBlue); //Se crea Ball2
		construirPantalla(ancho, altura); //Se llama al método construirPantalla
	}

	public void setup(){
		background(0); //Color del fondo de la pantalla, 0 equivale a negro
		//Se cambian los colores de Ball2
		bola2.setColor1(0);
		bola2.setColor2(255);
	}

	public void draw(){
		background(0); //Color del fondo de la pantalla, 0 equivale a negro
		bola1.display(this); //Se llama al método display de Ball1
		bola1.update(); //Se llama al método update de Ball1
		bola1.choqueConPared(altura, ancho); //Se llama al método choqueConPared de Ball1

		bola2.display(this); //Se llama al método display de Ball2
		bola2.update(); //Se llama al método update de Ball2
		bola2.choqueConPared(altura, ancho); //Se llama al método choqueConPared de Ball2
	}
```

```
        //Metodo construir Pantalla
        public void construirPantalla(int x, int y){
                size(x,y); //Se da tamaño a la pantalla
        }

}
```

Apartado C

```
//Se crean dos esferas que se mueven por pantalla, rebotan en las paredes y cambian de color cuando se cruzan
import processing.core.PApplet; //Se importa la Applet de Processing
public class C extends PApplet{

	//Se declaran la altura y el ancho de la pantalla
	public static final int altura=200;
	public static final int ancho=300;

	//Datos
	int colorRed=255;
	int colorGreen=0;
	int colorBlue=0;
	int radioCirculo=50;

	//Se decalaran las 2 variables principales , 2 esferas
	Esfera esfera1;
	Esfera esfera2;

	public static void main(String[] args) {
		PApplet.main("C");
	}

	public void settings(){
		//Se crea la primera esfera
		esfera1 = new Esfera(this,altura/2, altura/2,radioCirculo,colorRed,colorGreen,colorBlue);
		//Se crea la segunda esfera
		esfera2 = new Esfera(this,altura/4, altura/4,radioCirculo*6/10,colorRed,colorGreen,colorBlue);

		construirPantalla(ancho, altura);                                    //Se
llama al método construirPantalla
	}

	public void setup(){
		background(0); //Color de fondo de la pantalla, 0 equivale a negro

		esfera2.setColor1(0); //Se cambia el color de la segunda esfera, a 0 el rojo
		esfera2.setColor2(255); //Se cambia el color de la segunda esfera, a 255 el verde
	}

	public void draw(){
		background(0); //Color de fondo de la pantalla, 0 equivale a negro
		esfera1.display(this); //Se llama  al método display de la primera esfera
		esfera1.update(); //Se llama al método update de la primera esfera
		esfera1.choqueConPared(altura, ancho);     //Se llama al método choqueConPared  de la primera esfera
		esfera1.cruceEntreEsferas(this,esfera2,altura, ancho); //Se llama  al método cruceEntreEsferas de la primera
esfera con la segunda
```

```
                esfera2.display(this); //Se llama  al método display de la segunda esfera
                esfera2.update(); //Se llama  al método update de la segunda esfera
                esfera2.choqueConPared(altura, ancho); //Se llama  al método choqueConPared de la segunda esfera
                esfera2.cruceEntreEsferas(this,esfera1,altura, ancho); //Se llama  al método cruceEntreEsferas de la segunda
esfera con la primera
        }

        //Método construirPantalla
        public void construirPantalla(int x, int y){
                size(x,y); //Se da tamaño a la pantalla
        }

}
```

Práctica 4

Clase Elemento

```
//Clase Elemento
public class Elemento {
	private String nombre;
	private int xCoordenada,yCoordenada,xlength,ylength;
	private boolean encontrado;
	//Constructor de la clase Elemento
	public Elemento(String nombre, int xCoordenada, int yCoordenada,int xlength, int ylength, boolean encontrado) {
		this.nombre = nombre;
		this.xCoordenada = xCoordenada;
		this.yCoordenada = yCoordenada;
		this.xlength = xlength;
		this.ylength = ylength;
		this.encontrado = encontrado;
	}
	//Setters y Getters de las variables
	public String getNombre() {
		return nombre;}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	public int getXcoordenada() {
		return xCoordenada;}
	public void setxCoordenada(int xCoordenada) {
		this.xCoordenada = xCoordenada;
	}

	public int getYcoordenada() {
		return yCoordenada;}
	public void setyCoordenada(int yCoordenada) {
		this.yCoordenada = yCoordenada;
	}

	public boolean getEncontrado() {
		return encontrado;}
	public void setEncontrado(boolean encontrado) {
		this.encontrado = encontrado;
	}

	public int getXlength() {
		return xlength;}
	public void setXlength(int xlength) {
		this.xlength = xlength;
	}
```

```
	public int getYlength() {
		return ylength;}
	public void setYlength(int ylength) {
		this.ylength = ylength;
	}
}
```

Apartado A

```
//Se consiguen las coordenadas en cualquier punto de la imagen, para la posterior asignación a los objetos
import processing.core.PApplet;  //Se importa la Applet de Processing
import processing.core.PImage; //Se importa la librería PImage de Processing para poder trabajar con imagenes

public class A extends PApplet {
	//Declaración variables
	int xImagen =0,yImagen = 0;
	PImage imagen;

	public static void main(String[] args) {
		PApplet.main("A");
	}

	public void settings(){
		imagen=loadImage("foto2.png"); //Se carga la imagen deseada
		construirPantalla(imagen.width, imagen.height+100);   //Se llama al metodo construirPantalla
	}

	//Se ejecuta 1 sola vez
	public void setup(){
		background(255); //Color de fondo de la pantalla, 255 equivale a blanco
		mostrarImagen(imagen,xImagen, yImagen); //Se llama al método construirImagen
	}

	//Se ejecuta continuamente
	public void draw(){
	}


	//Cada vez que se clique el botón izquierdo del ratón
	public void mousePressed(){
		//Se muestra por consola las coordenadas en el momento del click
		println(mouseX,mouseY);

	}

	//Método construirPantalla
	public void construirPantalla(int x, int y) {
		size(x,y); //Se da tamaño a la pantalla
	}
```

```
		//Método mostrarImagen
		public void mostrarImagen(PImage imagen, int xImagen, int yImagen) {
			//Se muestra la imagen correspondiente en las coordenadas correspondientes
			image(imagen,xImagen,yImagen);

		}
}
```

Apartado B

```
//Se cosntruyen imágenes aleatoriamente con diferentes elementos y hay que encontrarlos
import java.util.ArrayList; //Se importa la libreria ArrayList de java
import java.util.concurrent.ThreadLocalRandom; //Se importa la librería random de java
import processing.core.PApplet; //Se importa la Applet de Processing
import processing.core.PImage; //Se importa la librería PImage de Processing para poder trabajar con imágenes

public class B extends PApplet {
		//Declaración de variables
		int margen=100;
		int xImagen =0,yImagen = 0;
		int fotoActual=0;
		int
numerosEncontrados=0,numeroImagenes=3,numeroElementosDefinitivosPorImagen=5,numeroElementosPorImagen=9;
		int xMax=0,yMax=0;
		int fallos=0,dado;
		int[] orden;

		PImage imagen1,imagen2, imagen3;
		PImage[] imagenes;
		Elemento[][] elementos,elementosDefinitivos;

		public static void main(String[] args) {
			PApplet.main("B");
		}


		public void settings(){
			//Se cargan las imágenes
			imagen1=loadImage("foto1.png");
			imagen2=loadImage("foto2.png");
			imagen3=loadImage("foto3.png");
			imagenes=new PImage[numeroImagenes];

			//Se declaran las variables en la cuales se almacenaremos los elementos
			elementos=new Elemento[numeroImagenes][numeroElementosPorImagen];
			elementosDefinitivos=new Elemento[numeroImagenes][numeroElementosDefinitivosPorImagen];
```

```
//Se almacenan las imágenes y elementos
imagenes[0]=imagen1;
imagenes[1]=imagen2;
imagenes[2]=imagen3;

//Se declaran los elementos
elementos[0][0]=new Elemento("Dinosaurio",298,41,26,13,false);
elementos[0][1]=new Elemento("Mariposa",59,280,19,16,false);
elementos[0][2]=new Elemento("Fresa",199,300,13,13,false);
elementos[0][3]=new Elemento("Ardillas",394,241,18,19,false);
elementos[0][4]=new Elemento("Lampara",20,78,20,22,false);
elementos[0][5]=new Elemento("Abanico",120,36,19,14,false);
elementos[0][6]=new Elemento("BolaMundo",111,210,19,18,false);
elementos[0][7]=new Elemento("Helado",400,280,15,18,false);
elementos[0][8]=new Elemento("Pimiento",186,254,12,11,false);

elementos[1][0]=new Elemento("Zapatos",294,282,12,14,false);
elementos[1][1]=new Elemento("Barco",138,134,21,18,false);
elementos[1][2]=new Elemento("Espejo",316,166,13,21,false);
elementos[1][3]=new Elemento("Guitarra",450,124,10,19,false);
elementos[1][4]=new Elemento("BolaMundo",499,268,33,30,false);
elementos[1][5]=new Elemento("Trompeta",364,220,21,25,false);
elementos[1][6]=new Elemento("Mascara",218,85,10,13,false);
elementos[1][7]=new Elemento("Camara",270,182,14,10,false);
elementos[1][8]=new Elemento("Pala",341,331,16,17,false);

elementos[2][0]=new Elemento("Diana",302,145,22,25,false);
elementos[2][1]=new Elemento("Camiseta",38,74,38,41,false);
elementos[2][2]=new Elemento("Telefono",352,368,40,30,false);
elementos[2][3]=new Elemento("OsoPeluche",145,187,15,17,false);
elementos[2][4]=new Elemento("BotasEsqui",491,136,19,33,false);
elementos[2][5]=new Elemento("Herramientas",260,331,33,38,false);
elementos[2][6]=new Elemento("Iman",388,103,11,15,false);
elementos[2][7]=new Elemento("Champiñon",455,330,27,16,false);
elementos[2][8]=new Elemento("Ordenador",103,310,28,25,false);

//Se llama al método cogerElementosAleatorios el cual cogerá 5 elementos aleatorios de cada imagen
elementosDefinitivos=cogerElementosAleatorios(elementos);

//Se llama al método cogerNumerosAleatorios dará aleatoriamente el orden de las imágenes
orden=cogerNumerosAletorios(numeroImagenes);

//Se llama amamos a estos dos métodos los cuales darán el ancho y el alto más grande entre todas las
imágenes
yMax=yMasGrande(imagenes);
xMax=xMasGrande(imagenes);
construirPantalla(xMax,yMax+margen); //Se llama al método construirPantalla
```

```
	}

	//Se ejecuta 1 sola vez
	public void setup(){
		background(255); //Color de fondo de la pantalla, 255 equivale a blanco

		//Se llama al método construirImagen con la imagen actual
		construirImagen(imagenes[orden[fotoActual]],xImagen, yImagen);
		}

	//Se ejecuta continuamente
	public void draw(){
		//Llamamos al método metodoEncontrado con la imagen y elementos actuales
		metodoEncontrado(imagenes[orden[fotoActual]], elementosDefinitivos[orden[fotoActual]]);
	}

	//Cada vez que se clique el boton derecho el ratón
	public void mousePressed(){
		dado=0;

		//Se comprueba si el punto en el cual se ha clicado está dentro del área de algún elemento
		for(Elemento e :elementosDefinitivos[orden[fotoActual]] ) {
			if(mouseX > e.getXcoordenada()-e.getXlength() && mouseX < e.getXcoordenada()+e.getXlength()
&& mouseY > e.getYcoordenada()-e.getYlength() && mouseY <e.getYcoordenada()+e.getYlength()){ //En caso de haber acertado
				e.setEncontrado(true);
				noFill(); //Sin relleno
				stroke(0,255,0); //Color Verde
				strokeWeight(4); //Grosor línea 4
				ellipse(e.getXcoordenada(),e.getYcoordenada(),40,40); //Se crea la elipse para mostrar
el acierto
				strokeWeight(1); //Grosor línea 1
				stroke(0); //Se vuelve al color negro
			}
			else { //En caso de haber falllado
				dado++;
			}
		}
		//En caso de que el punto no corresponda con ningun elemento, se suma un fallo al contador
		if(dado!=0) {
			fallos++;
		}
	}

	//Método métodoEncontrado, el cual una vez encontrado todos los elementos de una imagen
	//crea la siguiente, o en casp de que sea la última, se acaba el juego
	public void metodoEncontrado(PImage imagen, Elemento[] elementos) {
		if(numerosEncontrados==numeroElementosDefinitivosPorImagen) {
```

```
                    //Si ya se han encontrado todos los elementos de la imagen
                    stroke(0,255,0);
                    if(fotoActual==orden.length-1) {  //Si la foto actual es la última
                              background(255); //Color de fondo de la pantalla, 255 equivale a blanco
                              fill(0); //Letras negras

                              //Se muestra por pantalla que se ha acabado el juego
                              text("Has ganado", xMax/2, yMax/10);

                              }
                    else {        //Si las foto actual no es la última
                              fotoActual++; //Siguiente imagen

                              //Se vuelve a poner el contador de las imagenes a 0
                              numerosEncontrados=0;

                              //Color de fondo de la pantalla, 255 equivale a blanco
                              background(255);


                              //Se llama al método construirImagen con la imagen actual
                              construirImagen(imagenes[orden[fotoActual]],xImagen, yImagen);              }
          }

          //Si aún no se han encontrado todos los elementos de la imagen
          else {

                    fill(255); //Relleno blanco
                    stroke(255);          //Contormo blanco

                    //Se crea un rectángulo donde irán los nombres de los elementos a encontrar
                    rect(0, yMax, xMax, yMax+margen);                                                  int i=0;
                    numerosEncontrados=0;

                    //Se muestra el rectángulo con el nombre de los elementos que quedan por encontrar
                    for(Elemento e :elementos) {
                              if(!e.getEncontrado()) {
                                        fill(0); //Letras negras
                                        text(e.getNombre(), xMax/elementos.length*i+20,yMax+margen/2);
                                        i++;
                              }
                              else {
                                        numerosEncontrados++;
                              }
                    }
          }
}
```

```
//Método xMasGrande, el cual devuelve el ancho más grande entre todas las imágenes
public int xMasGrande(PImage[] imagenes) {
		int x=0;
		//En caso de que el ancho actual sea más grande que el anterior, se substituye
		for (int i = 0; i < imagenes.length; i++) {
			if(imagenes[i].width>x) {
				x=imagenes[i].width;
			}
		}
		return x;
}

//Método yMasGrande, el cual devuelve la altura más grande entre todas las imágenes
public int yMasGrande(PImage[] imagenes) {
		int y=0;
		//En caso de que la altura actual sea más grande que el anterior, se substituye
		for (int i = 0; i < imagenes.length; i++) {
			if(imagenes[i].height>y) {

				y=imagenes[i].height;
			}
		}
		return y;
}

//Método cogerNumerosAleatorios, el cual asignará un orden para que las imágenes se muestren aleatóriamente
public int[] cogerNumerosAletorios(int length) {
		int[] aux3= new int[length];
		int num1;

		//Se inicia una lista del 1 al X escribiendo los números
		ArrayList<Integer> aux2 = new ArrayList<Integer>();
		for(int i = 0; i < length; i++){
			aux2.add(i);
		}

		//Se cogen X números aleatorios de esa lista y se añaden
		for(int i = 0; i < length; i++){
			num1 = ThreadLocalRandom.current().nextInt(0, aux2.size());

			//Se recupera uno de los elementos, borrándolo en el proceso
			num1 = aux2.remove(num1);

			aux3[i] = num1; //Se asigna el número aleatorio recuperado
		}
		return aux3;
}
```

```
//Método cogerElementosAleatorios, el cual asignará un orden para que las imágenes se muestren aleatoriamente
public Elemento[][] cogerElementosAleatorios(Elemento[][] elementos2) {

		Elemento[][] aux=new Elemento[elementos2.length][numeroElementosDefinitivosPorImagen];
		int[] aux2= new int[numeroElementosPorImagen];

		for(int j = 0; j < aux.length; j++){
			aux2=cogerNumerosAletorios(aux2.length); //Se coge una lista de X números
			for(int k = 0; k < numeroElementosDefinitivosPorImagen; k++){
				aux[j][k]=elementos2[j][aux2[k]];  //Se cargan los elementos aleaóoriamente
			}
		}
		return aux;
	}

	//Método construirPantalla
	public void construirPantalla(int x, int y) {
		size(x,y); //Se da tamaño a la pantalla
	}

	//Método mostrarImagen
	public void construirImagen(PImage imagen, int xImagen, int yImagen) {
		xImagen=0;

		//Si el ancho de la imagen actual es menor al máximo, se centra la imagen
		if(imagen.width<xMax) {

			xImagen=(xMax-imagen.width)/2; //Se corrigen la coordenadas de los elementos después de
centrarla
			for (int i = 0; i <elementosDefinitivos[fotoActual].length; i++) {

	elementosDefinitivos[orden[fotoActual]][i].setxCoordenada(elementosDefinitivos[orden[fotoActual]][i].getXcoordenad
a()+xImagen);
			}
		}
		//Se muestra la imagen correspondiente en las coordenadas correspondientes
		image(imagen,xImagen,yImagen);

	}
```

Práctica 5

Apartado A

```
import processing.core.PApplet; //Se importa la Applet de Processing
import processing.core.PImage; //Se importa la librería PImage de Processing para poder trabajar con imagenes
import java.util.concurrent.ThreadLocalRandom; //Se importa la librería random de java

public class A extends PApplet {
	//Se declaran las variables
	public static final int altura=400,ancho=400;
	int yLinea = altura/10*8;
	int yFija=altura/10;
	int velocidad=5;
		//Velocidad a la cual se desea que avance la línea
	int xHueco,x, hueco=70;
	int cont=0;
	boolean marcha=false;

	PImage dino;

	public static void main(String[] args) {
		PApplet.main("A");
	}

	public void settings(){
		dino=new PImage();
		dino=loadImage("dinosaurio.png"); //Se carga la imagen
		size(ancho, altura); //Dar tamaño a la pantalla
	}

	//Se ejecuta 1 sola vez
	public void setup(){
		//Se coge un valor al azar para tener aleatóriamente el hueco
		xHueco= ThreadLocalRandom.current().nextInt(0, ancho-hueco);
		movimiento(ThreadLocalRandom.current().nextInt(0, ancho-hueco));	//Se llama al método
movimiento
	}


	// se ejecuta continuamente
	public void draw(){

		if(cont==3) {		//Cada 3 líneas, se aumenta en 1 la velocidad
			cont=0;
			velocidad++;
		}
		if(marcha) {		//Si está en marcha
			yLinea = yLinea - velocidad;
```

```
                        //Cuando la línea llegue al punto más alto, se baja al punto más bajo para que continue subiendo
                        if (yLinea < 0) {
                                yLinea = altura;
                                //Cada vez que aparece una nueva línea, se cambia la zona libre aleatóriamente
                                xHueco= ThreadLocalRandom.current().nextInt(0, ancho-hueco);
                                cont++;
                        }
                        //Se limita el movimiento de la imagen, en los bordes izquierdos y derechos para no perder de
vista la imagen
                        x=mouseX-dino.width/2;
                        if(x<=0) {
                                x=0;
                        }
                        if(x+dino.width>=ancho) {
                                x=ancho-dino.width;
                        }

                        movimiento(xHueco); //Se llama al método movimiento
                        choque(); //Se llama a la función choque
                }

                //Si no está en marcha, se muestra por pantalla que se ha perdido
                else{

                        fill(0); //Letras negras
                        text("Has perdido",ancho/2,altura/2);
                }

        }

        //Método choque
        private void choque() {
                //Si la imagen choca con la línea, se detiene el movimiento
                if(yFija>=yLinea-dino.height/2&&                yFija<=yLinea+dino.height/2&&(((mouseX-
dino.width/2)<=xHueco)||(mouseX+dino.width/2>=xHueco+hueco))) {
                        marcha=false;
                }
        }

        //Cada vez que se clique el botón izquierdo del ratón,
        public void mousePressed(){
                marcha=true; //Se da marcha a la línea
        }

        //Método movimiento, se muestra la línea y la imagen
        public void movimiento(int huecoLinea) {
                //Se cambia el color del fondo de la pantalla, 255 equivale a blanco
```

```
        background(255);


        //Se da el color negro al contorno, para que asi la posterior línea sea negra
        stroke(0);

        //Se dibuja la línea con el hueco
        line(0, yLinea,huecoLinea, yLinea);
        line(huecoLinea+hueco, yLinea,ancho, yLinea);
        //Se muestra la imagen correspondiente en las coordenadas correspondientes
        image(dino,x,yFija-dino.height/2);

}
```

Práctica 6

Clase Elemento

```
//Clase Imagen
import processing.core.PImage; //Se importa la librería PImage de Processing para poder trabajar con imágenes


public class Imagen {
	//Se declaran las variables de la clase Imagen
	private PImage foto;
	private int xCoordenadaActual,xCoordenadaCorrecta, yCoordenadaActual,yCoordenadaCorrecta;
	private boolean colocada;

	//Constructor de la clase Imagen
	public Imagen(PImage foto, int xCoordenadaActual,  int yCoordenadaActual,int xCoordenadaCorrecta,
				int yCoordenadaCorrecta, boolean colocada) {
		super();
		this.foto = foto;
		this.xCoordenadaActual = xCoordenadaActual;
		this.yCoordenadaActual = yCoordenadaActual;
		this.xCoordenadaCorrecta = xCoordenadaCorrecta;
		this.yCoordenadaCorrecta = yCoordenadaCorrecta;
		this.colocada=colocada;
	}

	//Setters y Getters de las variables
	public PImage getFoto() {
		return foto;
	}
	public int getxCoordenadaActual() {
		return xCoordenadaActual;}
	public void setxCoordenadaActual(int xCoordenadaActual) {
		this.xCoordenadaActual = xCoordenadaActual;
	}

	public int getyCoordenadaActual() {
		return yCoordenadaActual;}
	public void setyCoordenadaActual(int yCoordenadaActual) {
		this.yCoordenadaActual = yCoordenadaActual;
	}

	public boolean isColocada() {
		return colocada;}
	public void setColocada(boolean colocada) {
		this.colocada = colocada;
	}

	public int getxCoordenadaCorrecta() {
```

```
                return xCoordenadaCorrecta;
        }

        public int getyCoordenadaCorrecta() {
                return yCoordenadaCorrecta;
        }
}
```

Apartado A

```
//Ejercicio en el cual se divide la imagen en x piezas y se crean x imágenes
import processing.core.PApplet; //Se importa la Applet de Processing
import processing.core.PImage; //Se importa la librería PImage de Processing para poder trabajar con imágenes


public class A extends PApplet {
	//Se declaran las variables
	int piezas=4;
	int raizPiezas;
	int alturaAux,anchoAux;
	boolean valido=false;
	PImage imagen;

	public static void main(String[] args) {
		PApplet.main("A");
	}

	public void settings(){
		//Se carga la imagen
		imagen=loadImage("fuji.png");


		//En caso que el número de piezas sea cuadrado de otro número
		if(!(piezas%2==0&&piezas%3==0&&piezas%5==0&&piezas%7==0&&piezas%11==0)) {
			raizPiezas=(int) Math.sqrt(piezas);
			//Se da tamaño a la pantalla, el tamaño de las fúturas piezas
			size(imagen.width/raizPiezas,imagen.height/raizPiezas);

			valido=true;		//Se da el okey
		}

		//En caso que el número de piezas no sea cuadrado de otro número
		else {

			//Creamos una pantalla vacía
			alturaAux=800;
			anchoAux=800;
			size(anchoAux,alturaAux);		//Se da tamaño a la pantalla

			//Se escribe un pequeño texto en unas determinadas coordenadas
			text("Numero de Piezas incorrecto", anchoAux/2, alturaAux/2);


		}
	}
```

```
	//Se ejecuta 1 sola vez
	public void setup(){
		background(255); //Se cambia el color de fondo de la pantalla, 255 equivale a blanco
		if(valido) {  //Si el número de piezas es cuadrado de otro número
			dividirImagen(imagen,piezas); //Se llama al método dvidirImagen
		}
		else {	//Si el número es inválido, se muestra un mensaje
			fill(0); //Letras negras

			//Se escribe un pequeño texto en unas determinadas coordenadas
			text("Numero de piezas invalido",anchoAux/3,alturaAux/3);

		}
	}

	//Se ejecuta continuamente
	public void draw(){
	}

	//Método dividirImagen, se crean x piezas
	public void dividirImagen(PImage foto, int numeroPiezas) {
		int x;
		int y;
		int columna=0; //Columna a 0
		int fila=-1;

		//Por X piezas, se juegacon la posición en la quiere que se muestre la imagen para así conseguir las piezas colocadas correctamente
		for (int i = 0; i < numeroPiezas; i++) {
			if (i%Math.sqrt(numeroPiezas)==0){
				columna=0; //Columna a 0
				fila++; //Se aumenta una fila
			}
			else {

				columna++; //Se auumenta una colunmna
			}
			//Se asignan las coordenadas correctas para que en pantalla se muestre la parte de la imagen que se desees
			x=(int) (-foto.width/Math.sqrt(numeroPiezas)*columna);
			y=(int) (-foto.height/Math.sqrt(numeroPiezas)*fila);

			//Se muestra la imagen correspondiente en las coordenadas correspondientes
			image(foto,x,y);

			//Se guarda lo que se vea en la pantalla con un nombre determinado
			save("part"+i+".png");
```

```
                }
            }
        }
```

Apartado B1

```
//Se crea el puzzle y colocamos las piezas con el teclado
import java.util.ArrayList; //Se importa la librería ArrayList de java
import java.util.concurrent.ThreadLocalRandom; //Se importa la librería random de java
import processing.core.PApplet; //Se importa la Applet de Processing
import processing.core.PImage; //Se importa la librería PImage de Processing para poder trabajar con imágenes


public class B1 extends PApplet{
	//Se declaran las variables
	int altura,ancho;
	int piezas=4,margenCorrecto=5,margenAutoajuste=5,numeroCoordenadas=2, velocidad=8;
	int raizPiezas=(int) Math.sqrt(piezas);
	int imagenSeleccionada=0,totalColocadas=0;
	int[] orden;
	boolean valido=false;

	PImage imagen;
	PImage[] imagenes;
	Imagen foto;
	Imagen[] conjuntoDeImagenes;

	public static void main(String[] args) {
		PApplet.main("B1");
	}

	public void settings(){

		//En caso que el número de piezas sea cuadrado de otro número
		if(!(piezas%2==0&&piezas%3==0&&piezas%5==0&&piezas%7==0&&piezas%11==0)) {

			//Se cargan la imagen
			imagen=loadImage("fuji.png");

			conjuntoDeImagenes=new Imagen[piezas];
			orden=new int[piezas];
			ancho=imagen.width*2;
			altura=imagen.height*3;
			size(ancho,altura);	//Se da tamaño a la pantalla
			valido=true;		//Se da el okey
		}

		//En caso que el número de piezas no sea cuadrado de otro número
		else {

			//Creamos una pantalla vacía
			altura=800;
```

```
                ancho=800;
                size(ancho,altura);     //Se da tamaño a la pantalla

                //Se ecribe un pequeño texto en unas determinadas coordenadas
                text("Numero de Piezas incorrecto", ancho/2, altura/2);

            }
        }

        //Se ejecuta 1 sola vez
        public void setup(){
            if(valido) {
                orden=numerosAleatorios(piezas);            //Se crea una lista de valores desordenados

                //Se llama al método crearConjuntoImagenes, que nos devolverá una lista con todas las piezas desordenadas
                conjuntoDeImagenes=crearConjuntoImagenes(orden);
                construirPantalla(); //Se llama al método construirPantalla
            }
        }

        //Se ejecuta continuamente
        public void draw(){
            if(valido) {
                if(totalColocadas==conjuntoDeImagenes.length) {                    //En el momento que están todas las imágenes colocadas

                    background(255); //Color de fondo de la pantalla, 255 equivale a blanco

                        image(imagen,(ancho-imagen.width)/2,0); //Se muestra la imagen completa en su lugar
                        fill(0); //Letras negras

                        //Se escribe un pequeño texto en unas determinadas coordenadas
                        text("Has ganado",(ancho-imagen.width)/2,altura/2);
                }
            }
        }

        //Cada vez que cliquemos el ratón, daremos marcha o paro al movimiento de la linea
        public void mousePressed(){
            int f=-1;
            for(Imagen e :conjuntoDeImagenes ) {
        //Se busca por las imágenes, cual se ha seleccionado
                if(!e.isColocada()) { //Si ya está colocada, no se permitirá seleccionarla
                    if(mouseX > e.getxCoordenadaActual()&& mouseX < e.getxCoordenadaActual()+e.getFoto().width && mouseY > e.getyCoordenadaActual()&& mouseY <e.getyCoordenadaActual()+e.getFoto().height) {
```

```
                                        imagenSeleccionada=f+1;
                                        construirPantalla(); //Se llama al método construirPantalla
                                    }
                                }
                                f++;
                            }
                        }

                        //Método construirPantalla
                        public void construirPantalla() {
                            //Color de fondo de la pantalla, 255 equivale a blanco
                            background(255);


                            if(totalColocadas==conjuntoDeImagenes.length) { //Si ya se han colocado todas las imágenes
                                image(imagen,(ancho-imagen.width)/2,altura/3); //Se muestra la imagen completa
                            }
                            else { //Si no están todas las imágenes colocadas
                                autoAjuste(); //Se llama al método Autoajuste
                                construirRectangulos(); //Se llama al método construirRectangulos

                                //Se llama Lamamos al método visualizarImagenes para mostrar todas las piezas
                                visulaizarImagenes(conjuntoDeImagenes);
                            }
                        }

                        //Método visualizarImagenes, se muestran todas las piezas en la pantalla
                        private void visulaizarImagenes(Imagen[] conjuntoDeImagenes2) {
                            //Se colocan las x piezas en sus coordenadas actuales
                            for (int i = 0; i < piezas; i++) {

                                //Si la imagen i no está seleccionada, se mostrará tal cual
                                if(i!=imagenSeleccionada) {


image(conjuntoDeImagenes2[i].getFoto(),conjuntoDeImagenes2[i].getxCoordenadaActual(),conjuntoDeImagenes2[i].getyCoordenadaActual());// indicamos que imagen queremos mostar, y en que coordenadas
                                }
                            }
                            //La imagen seleccionada se mostrará con un contorno rojo
                            stroke(250,0,0); //Contorno rojo
                            //Se creea un rectángulo rojo alrededor de la imagen seleccionada
                            rect(conjuntoDeImagenes2[imagenSeleccionada].getxCoordenadaActual()-
1,conjuntoDeImagenes2[imagenSeleccionada].getyCoordenadaActual()-
1,conjuntoDeImagenes2[imagenSeleccionada].getFoto().width+1,conjuntoDeImagenes2[imagenSeleccionada].getFoto().height+1)
;

                            image(conjuntoDeImagenes2[imagenSeleccionada].getFoto(),conjuntoDeImagenes2[imagenSeleccionada].getxCoorde
```

```
nadaActual(),conjuntoDeImagenes2[imagenSeleccionada].getyCoordenadaActual());// indicamos que imagen queremos mostar, y en que coordenadas
                    stroke(0); //Se vuelve a dejarlo sin contorno
          }

          //Método construiRectangulo
          public void construirRectangulos() { //Se crean los x rectángulos donde irán las piezas
                    for (int i = 0; i < piezas; i++) {
                              rect(conjuntoDeImagenes[i].getxCoordenadaCorrecta(),
conjuntoDeImagenes[i].getyCoordenadaCorrecta(),conjuntoDeImagenes[i].getFoto().width,conjuntoDeImagenes[i].getFoto().heig
ht);
                    }
          }

          //Método Autoajuste,
          //En caso que la imagen que se esté moviendo este relativamente cerca de su rectángulo correcto
          public void autoAjuste(){
                    if(conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaActual()-
margenAutoajuste<=conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaCorrecta()&&conjuntoDeImagenes[imagenSelecc
ionada].getxCoordenadaActual()+margenAutoajuste>=conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaCorrecta()&&c
onjuntoDeImagenes[imagenSeleccionada].getyCoordenadaActual()-
margenAutoajuste<=conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaCorrecta()&&conjuntoDeImagenes[imagenSelecc
ionada].getyCoordenadaActual()+margenAutoajuste>=conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaCorrecta()) {

          conjuntoDeImagenes[imagenSeleccionada].setxCoordenadaActual(conjuntoDeImagenes[imagenSeleccionada].getxCoo
rdenadaCorrecta());

          conjuntoDeImagenes[imagenSeleccionada].setyCoordenadaActual(conjuntoDeImagenes[imagenSeleccionada].getyCoo
rdenadaCorrecta());

          image(conjuntoDeImagenes[imagenSeleccionada].getFoto(),conjuntoDeImagenes[imagenSeleccionada].getxCoordenad
aActual(),conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaActual());// indicamos que imagen queremos mostar, y en
que coordenadas
                              conjuntoDeImagenes[imagenSeleccionada].setColocada(true);
                              totalColocadas++; //Se aumenta el número de piezas colocadas
                    }
          }


          //Método cogerNumerosAleatorios, el cual asignará un orden para que las imágenes se muestren aleatóriamente
          public int[] numerosAleatorios(int length) {
                    int[] aux3= new int[length];
                    int num1;

                    //Se inicia lista del 1 al X para escribir los números
                    ArrayList<Integer> aux2 = new ArrayList<Integer>();
                    for(int i = 0; i < length; i++){
                              aux2.add(i);
```

```
                }

                //Se cogen X números aleatorios de esa lista y los añadimos
                for(int i = 0; i < length; i++){
                        num1 = ThreadLocalRandom.current().nextInt(0, aux2.size());
                        num1 = aux2.remove(num1);      //Se recupera uno de los elementos, borrandolo en el proceso
                        aux3[i] = num1; //Se asignan el número aleatorio recuperado
                }
                return aux3;
        }


        //Método crearConjuntoImagenes
        public Imagen[] crearConjuntoImagenes(int[] orden) {
                Imagen[] aux= new Imagen[piezas];

                int j=0,k=-1,l=0, m=0;
                //Se declaran la separación entre las piezas para que no estén juntas
                int margenVertical=((altura-imagen.height)-imagen.height)/(raizPiezas+1);
                int margenHorizontal=(ancho-imagen.width)/(raizPiezas+1);

                //Se asignan las variables correctas para que se guarde cada pieza con su coordenada correcta
                for (int i = 0; i < piezas; i++) {
                        if (i%raizPiezas==0){
                                j=0;
                                k++;
                        }
                        else {
                                j++;
                        }
                        if(orden[i]-raizPiezas<0) {
                                l=orden[i];
                        }
                        else {
                                l=orden[i]%raizPiezas;
                        }
                        m=orden[i]/raizPiezas;

                        //Se cargan todas las imágenes creadas anteriormente llamándolas por su nombre
                        aux[i]=new
Imagen(loadImage("part"+(orden[i])+".png"),margenHorizontal*(j+1)+imagen.width/raizPiezas*j,imagen.height+margenVertical*(
k+1)+imagen.height/raizPiezas*k,imagen.width/2+imagen.width/raizPiezas*l,imagen.height/raizPiezas*m,false);
                }
                return aux;
        }

        //Método en el cual entrará cuando se pulse una tecla
        public void keyPressed(){
```

```
                    if(!conjuntoDeImagenes[imagenSeleccionada].isColocada()) {

                    //En caso de que la tecla pulsada sea la W, el logo se desplazará hacia arriba
            if((key=='w'||key=='W')&&conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaActual()>0){          //Se
disminuye la coordenada Y
                                conjuntoDeImagenes[imagenSeleccionada].setyCoordenadaActual(
conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaActual()-velocidad); //La pieza se deplazará hacia arriba
                                construirPantalla();
                            }

                            //En caso de que la tecla pulsada sea la W, el logo se desplazará hacia abajo
                            else
if((key=='s'||key=='S')&&conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaActual()                          <altura-
conjuntoDeImagenes[imagenSeleccionada].getFoto().height){ //Se aumenta la coordenada Y
                                conjuntoDeImagenes[imagenSeleccionada].setyCoordenadaActual(
conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaActual()+velocidad);
                                //La pieza se deplazará hacia abajo
                                construirPantalla();
                            }

                            //En caso de que la tecla pulsada sea la W, el logo se desplazará hacia la izquierda
                            else
if((key=='a'||key=='A')&&conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaActual() >0){ //Se disminuye la coordenada X
                                conjuntoDeImagenes[imagenSeleccionada].setxCoordenadaActual(
conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaActual()-velocidad);
                                //La pieza se desplazará hacia la izquierda
                                construirPantalla();
                            }

                            //En caso de que la tecla pulsada sea la W, el logo se desplazará hacia la derecha
                            else
if((key=='d'||key=='D')&&conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaActual()                          <ancho-
conjuntoDeImagenes[imagenSeleccionada].getFoto().width){ //Se aumenta la coordenada X
                                conjuntoDeImagenes[imagenSeleccionada].setxCoordenadaActual(
conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaActual()+velocidad);
                                //La pieza se desplazará hacia la derecha
                                construirPantalla();
                            }

                    }

            }
}
```

Apartado B2

```
//Se crea el puzzle y se colocan las piezas con el arduino
import java.util.ArrayList; //Se importa la librería ArrayList de java
import java.util.concurrent.ThreadLocalRandom; //Se importa la librería random de java
import processing.core.PApplet; //Se importa la Applet de Processing
import processing.core.PImage; //Se importa la librería PImage de Processing para poder trabajar con imágenes

import processing.serial.*; //Se importa la librería Serial de Processing

public class B2 extends PApplet{
	//Se declaran las variables
	int piezas=81,margenCorrecto=5,margenAutoajuste=5,numeroCoordenadas=2,velocidad=8;
	int altura,ancho;
	int raizPiezas=(int) Math.sqrt(piezas);
	int imagenSeleccionada=0,totalColocadas=0;
	int valor;
	int[] orden;
	boolean valido=false;

	PImage imagen;
	PImage[] imagenes;
	Imagen foto;
	Imagen[] conjuntoDeImagenes;
	Serial miPuerto;


	public static void main(String[] args) {
		PApplet.main("B2");
	}

	public void settings(){
		//En caso que el número de piezas sea cuadrado de otro número
		if(!(piezas%2==0&&piezas%3==0&&piezas%5==0&&piezas%7==0&&piezas%11==0)) {
			//Se carga la imagen
			imagen=loadImage("fuji.png");

			conjuntoDeImagenes=new Imagen[piezas];
			orden=new int[piezas];
			ancho=imagen.width*2;
			altura=imagen.height*3;
			size(ancho,altura);	//Se da tamaño a la pantalla
			valido=true;		//Se da el okey
		}

		//En caso que el número de piezas no sea cuadrado de otro número
		else {
```

```
                        //Se crea una pantalla vacía
                        altura=800;
                        ancho=800;

                        size(ancho,altura);     //Se da tamaño a la pantalla

                        //Se escribe un pequeño texto en unas determinadas coordenadas
                        text("Numero de Piezas incorrecto", ancho/2, altura/2);

                }
        }

        //Se ejecuta 1 sola vez
        public void setup(){
                if(valido) {
                        miPuerto= new Serial(this, Serial.list()[2], 9600);          //Se conecta con el puertoSerie de nuestro
PC
                        orden=numerosAleatorios(piezas);
        //Se crea una lista de valores desordenados
                        //Se llama al método crearConjuntoImagenes, que nos devolverá una lista con todas las
desordenadas
                        conjuntoDeImagenes=crearConjuntoImagenes(orden);
                        construirPantalla(); //Se llama al método construirPantalla
                }
        }

        //Se ejecuta continuamente
        public void draw(){
                if(valido) {
                        if(totalColocadas==conjuntoDeImagenes.length) { //En el momento que estén todas las imágenes
colocadas
                                background(255); //Color de fondo de la pantalla, 255 equivale a blanco
                                image(imagen,(ancho-imagen.width)/2,0);  //Se muestra la imagen completa en su
lugar
                                fill(0); //Letras negras

                                //Se escribe un pequeño texto en unas determinadas coordenadas
                                text("Has ganado",(ancho-imagen.width)/2,altura/2);
                        }
                        else{
                                        //En caso de que el puzzle no esté acabado
                                valor=0;
                                if(miPuerto.available() > 0){ //Se mira si hay algún dato disponible en el puerto
                                        valor=miPuerto.read(); //Se lee el dato y lo almacena en la variable "valor"
                                }
                                if(!conjuntoDeImagenes[imagenSeleccionada].isColocada()) { //En caso de que la tecla
pulsada sea la W, el logo se desplazará hacia arriba
                                        if(valor==4){           //Si el valor es un 4
```

```
                                                //Se disminuye la coordenada Y

            conjuntoDeImagenes[imagenSeleccionada].setyCoordenadaActual(
conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaActual()-velocidad);
                                                construirPantalla();
                                            }
                                            //En caso de que la tecla pulsada sea la W, el logo se desplazará hacia abajo
                                            else if(valor==2){      //Si el valor es un 2
                                                //Se aumentas la coordenada Y

            conjuntoDeImagenes[imagenSeleccionada].setyCoordenadaActual(
conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaActual()+velocidad);
                                                construirPantalla();
                                            }

                                            //En caso de que la tecla pulsada sea la W, el logo se desplazará hacia la
izquierda
                                            else if(valor==3){      //Si el valor es un 3
                                                //Se disminuye la coordenada X

            conjuntoDeImagenes[imagenSeleccionada].setxCoordenadaActual(
conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaActual()-velocidad);
                                                construirPantalla();
                                            }
                                            //En caso de que la tecla pulsada sea la W, el logo se desplazará hacia la
derecha
                                            else if(valor==5){      //Si el valor es un 5
                                                //Se auumentaa la coordenada X

            conjuntoDeImagenes[imagenSeleccionada].setxCoordenadaActual(
conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaActual()+velocidad);
                                                construirPantalla();
                                            }
                                        }
                                    }
                                }
            }

            //Cada vez que se clique el botón izquierdo del ratón, se dará marcha o paro al movimiento de la línea
            public void mousePressed(){
                int f=-1;
                for(Imagen e :conjuntoDeImagenes ) { //Se busca por las imágenes cual se ha seleccionado
                    if(!e.isColocada()) { //Si ya está colocada, no se permitirá seleccionarla
                        if(mouseX > e.getxCoordenadaActual()&& mouseX <
e.getxCoordenadaActual()+e.getFoto().width && mouseY > e.getyCoordenadaActual()&& mouseY
<e.getyCoordenadaActual()+e.getFoto().height) {
                            imagenSeleccionada=f+1;
                            construirPantalla(); //Se llama al método construirPantalla
```

```
                                        }
                                }
                                f++;
                        }
                }

                //Método construirPantalla
                public void construirPantalla() {
                        background(255); //Color de fondo de la pantalla, 255 equivale a blanco

                        if(totalColocadas==conjuntoDeImagenes.length) { //Si ya se han colocado todas las imagenes
                                image(imagen,(ancho-imagen.width)/2,altura/3); //Se muestra la imagen completa
                        }
                        else {
                                        //Si no están todas las imágenes colocadas
                                autoAjuste(); //Se llama al método Autoajuste
                                construirRectangulos(); //Se llama al método construirRectangulos

                                //Se llama al método visualizarImagenes para mostrar todas las piezas
                                visulaizarImagenes(conjuntoDeImagenes);
                        }
                }

                //Método visualizarImagenes, se muestran todas las piezas en la pantalla
                private void visulaizarImagenes(Imagen[] conjuntoDeImagenes2) {
                        //Se colocan las x piezas en sus coordenadas actuales
                        for (int i = 0; i < piezas; i++) {
                                if(i!=imagenSeleccionada) { //Si la imagen i no está seleccionada, se muestra tal cual

                image(conjuntoDeImagenes2[i].getFoto(),conjuntoDeImagenes2[i].getxCoordenadaActual(),conjuntoDeImagenes2[i].getyCoordenadaActual()); //indicamos que imagen queremos mostar, y en que coordenadas
                                }
                        }
                        //La imagen seleccionada se muestra con un contorno rojo
                        stroke(250,0,0); //Contorno rojo
                        //Se crea un rectángulo rojo alrededor de la imagen seleccionada
                        rect(conjuntoDeImagenes2[imagenSeleccionada].getxCoordenadaActual()-1,conjuntoDeImagenes2[imagenSeleccionada].getyCoordenadaActual()-1,conjuntoDeImagenes2[imagenSeleccionada].getFoto().width+1,conjuntoDeImagenes2[imagenSeleccionada].getFoto().height+1);

                image(conjuntoDeImagenes2[imagenSeleccionada].getFoto(),conjuntoDeImagenes2[imagenSeleccionada].getxCoordenadaActual(),conjuntoDeImagenes2[imagenSeleccionada].getyCoordenadaActual());// indicamos que imagen queremos mostar, y en que coordenadas
                        stroke(0); //Se vuelve a dejar sin contorno
                }

                //Método construiRectangulos
```

```
	public void construirRectangulos() { //Se crean los x rectángulos donde irán las piezas
		for (int i = 0; i < piezas; i++) {
			rect(conjuntoDeImagenes[i].getxCoordenadaCorrecta(),
conjuntoDeImagenes[i].getyCoordenadaCorrecta(),conjuntoDeImagenes[i].getFoto().width,conjuntoDeImagenes[i].getFoto().heig
ht);
		}
	}

	//Método Autoajuste,
	//En caso que la imagen que se esté moviendo este relativamente cerca de su rectángulo correcto
	public void autoAjuste(){
		if(conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaActual()-
margenAutoajuste<=conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaCorrecta()&&conjuntoDeImagenes[imagenSelecc
ionada].getxCoordenadaActual()+margenAutoajuste>=conjuntoDeImagenes[imagenSeleccionada].getxCoordenadaCorrecta()&&c
onjuntoDeImagenes[imagenSeleccionada].getyCoordenadaActual()-
margenAutoajuste<=conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaCorrecta()&&conjuntoDeImagenes[imagenSelecc
ionada].getyCoordenadaActual()+margenAutoajuste>=conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaCorrecta()) {

	conjuntoDeImagenes[imagenSeleccionada].setxCoordenadaActual(conjuntoDeImagenes[imagenSeleccionada].getxCoo
rdenadaCorrecta());

	conjuntoDeImagenes[imagenSeleccionada].setyCoordenadaActual(conjuntoDeImagenes[imagenSeleccionada].getyCoo
rdenadaCorrecta());

	image(conjuntoDeImagenes[imagenSeleccionada].getFoto(),conjuntoDeImagenes[imagenSeleccionada].getxCoordenad
aActual(),conjuntoDeImagenes[imagenSeleccionada].getyCoordenadaActual());// indicamos que imagen queremos mostar, y en
que coordenadas
			conjuntoDeImagenes[imagenSeleccionada].setColocada(true);
			totalColocadas++;	//Se aumenta el número de piezas colocadas
		}
	}

	//Método cogerNumerosAleatorios, el cual asignará un orden para que las imágenes se muestren aleatóriamente
	public int[] numerosAleatorios(int length) {
		int[] aux3= new int[length];
		int num1;

		//Se inicia una lista del 1 al X para escribir los números
		ArrayList<Integer> aux2 = new ArrayList<Integer>();
		for(int i = 0; i < length; i++){
			aux2.add(i);
		}

		//Se cogen X números aleatorios de esa lista y se añaden
		for(int i = 0; i < length; i++){
			num1 = ThreadLocalRandom.current().nextInt(0, aux2.size());
			num1 = aux2.remove(num1);	//Se recupera uno de los elementos, borrándolo en el proceso
			aux3[i] = num1; //Se asigna el número aleatorio recuperado
```

```
                }
                return aux3;
        }

        //Método crearConjuntoImagenes
        public Imagen[] crearConjuntoImagenes(int[] orden) {
                Imagen[] aux= new Imagen[piezas];

                int j=0,k=-1,l=0, m=0;
                //Se declara la separación entre las pieazs para que no estén juntas
                int margenVertical=((altura-imagen.height)-imagen.height)/(raizPiezas+1);
                int margenHorizontal=(ancho-imagen.width)/(raizPiezas+1);

                //Se asignan las variables correctas para que se guarde cada pieza con su coordenada correcta
                for (int i = 0; i < piezas; i++) {
                        if (i%raizPiezas==0){
                                j=0;
                                k++;
                        }
                        else {
                                j++;
                        }
                        if(orden[i]-raizPiezas<0) {
                                l=orden[i];
                        }
                        else {
                                l=orden[i]%raizPiezas;
                        }
                        m=orden[i]/raizPiezas;
                        //Se cargan las imagenes creadas anteriormente llamándolas por su nombre
                        aux[i]=new
Imagen(loadImage("part"+(orden[i])+".png"),margenHorizontal*(j+1)+imagen.width/raizPiezas*j,imagen.height+margenVertical*(
k+1)+imagen.height/raizPiezas*k,imagen.width/2+imagen.width/raizPiezas*l,imagen.height/raizPiezas*m,false);
                }
                return aux;
        }
}
```

Código Arduino

```
int switchState2 = 0;
int switchState3 = 0;
int switchState4 = 0;
int switchState5 = 0;

void setup() {
  Serial.begin(9600);

  pinMode(2,INPUT);
  pinMode(3,INPUT);
  pinMode(4,INPUT);
  pinMode(5,INPUT);
  pinMode(6,OUTPUT);
}

void loop() {
  switchState2 = digitalRead(2);
  switchState3 = digitalRead(3);
  switchState4 = digitalRead(4);
  switchState5 = digitalRead(5);

  if(switchState2){
    Serial.write(2);
  }
  else if(switchState3){
    Serial.write(3);
  }
  else if(switchState4){
    Serial.write(4);
  }
  else if(switchState5){
    Serial.write(5);
```

```
 }
 else{
  Serial.write(0);
 }
 delay(100);
}
```

Práctica 7

Habitación

```
//Clase Habitación
public class Habitacion {

	private String nombre;
	private int xCoordenada,yCoordenada,xlength,ylength;
	private boolean dentro;

	//Constructor de la clase Habitación
	public Habitacion(String nombre, int xCoordenada, int yCoordenada,int xlength, int ylength,boolean dentro) {
		this.nombre = nombre;
		this.xCoordenada = xCoordenada;
		this.yCoordenada = yCoordenada;
		this.xlength = xlength;
		this.ylength = ylength;
		this.dentro=dentro;
	}

	//Setters y Getters de las variables
	public String getNombre() {
		return nombre;}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	public int getXcoordenada() {
		return xCoordenada;}
	public void setxCoordenada(int xCoordenada) {
		this.xCoordenada = xCoordenada;
	}

	public int getYcoordenada() {
		return yCoordenada;}
	public void setyCoordenada(int yCoordenada) {
		this.yCoordenada = yCoordenada;
	}
```

```
	public boolean getDentro() {
		return dentro;}
	public void setDentro(boolean dentro) {
		this.dentro = dentro;
	}

	public int getXlength() {
		return xlength;}
	public void setXlength(int xlength) {
		this.xlength = xlength;
	}

	public int getYlength() {
		return ylength;}
	public void setYlength(int ylength) {
		this.ylength = ylength;
	}
}
```

Apartado A

```
//Se crea un aire acondicionado que se activará según la temperatura
import processing.core.PApplet; //Se importa la Applet de Processing
import processing.serial.*; //Se importa la librería Serial de Processing

public class A extends PApplet{
	//Declaración de variables
	int ancho=800,altura=800;
	Serial miPuerto;
	int valor,valorOn=15,valorOff=10;
	int xCoordenadaAire=100,yCoordenadaAire=125;
	int achoAire=150,alturaAire=60;
	int margen=10,radioElpise=10;
	boolean aireActivo=false;


	public static void main(String[] args) {
		PApplet.main("A");
	}

	public void settings(){
		size(ancho,altura);	//Se da tamaño a la pantalla
	}

	//Se ejecuta 1 sola vez
	public void setup(){
		miPuerto= new Serial(this, Serial.list()[2], 9600);		//Se conecta el puertoSerie con el PC
		construirPantalla(); //Se llama al método construirPantalla
	}

	//Se ejecuta continuamente
	public void draw(){
		valor=0;
		if(miPuerto.available() > 0){ //Se mira si hay algún dato disponible en el puerto

			valor=miPuerto.read(); //Se lee el dato y lo almacenamos en la variable "valor"
		}
		delay(250);
		activacionAire(); //Se llama al método activacióAire para la posible activación del aire
		construirMandoAire(aireActivo); //Se llama al método construirMandoAire para actualizar la temperatura
	}


	public void activacionAire() {
		if(valor>valorOn) {	//Si el valor supera la consigna de activación, se activa el aire
			aireActivo=true;
```

```
		}
		else {
			if(valor<valorOff) { //Si el valor es inferior a la consgina de desactivación, se desactiva el aire
				aireActivo=false;
			}
		}

}

//Método construirPantalla
public void construirPantalla() {
		background(255); //Color de fondo de la pantalla, 255 equivale a blanco
		construirMandoAire(aireActivo); //Se llama al método construirMandoAire
}


//Método construirMandoAire
public void construirMandoAire(boolean activado) {
		//Primer recuadro, temperatura actual
		stroke(0);
		fill(255); //Relleno Blanco
		rect(xCoordenadaAire,yCoordenadaAire,achoAire,alturaAire);	//Primer rectángulo
		fill(0); //Letras negras
		text("Tº "+valor+" ºC",xCoordenadaAire+8*margen,yCoordenadaAire+alturaAire/2); //Temperatura actual

		//Segundo recuadro, temperatura de on
		fill(255); //Relleno Blanco
		rect(xCoordenadaAire,yCoordenadaAire+alturaAire,achoAire,alturaAire);
//Segundo rectángulo
		if(activado) { //Si el aire está activado
			fill(0,255,0); //Relleno Verde

			//Círculo verde para decir que el aire está en modo on
ellipse(xCoordenadaAire+margen,yCoordenadaAire+alturaAire+alturaAire/2,radioElpise,radioElpise);
		}
		else { //Si el aire no está activado
			fill(255,0,0); //Relleno Rojo

			//Círculo rojo para decir que el aire no está en modo off
ellipse(xCoordenadaAire+margen,yCoordenadaAire+alturaAire+alturaAire/2,radioElpise,radioElpise);
		}

		//Tercer recuadro, temperatura de off
		fill(0); //Letras negras
		//Temperatura On
		text("Tº  on  "+valorOn+"  ºC",xCoordenadaAire+2*margen,yCoordenadaAire+alturaAire+alturaAire/2);
```

```
                fill(255); //Relleno Blanco
                rect(xCoordenadaAire,yCoordenadaAire+alturaAire*2,achoAire,alturaAire);                 //Tercer rectángulo
                if(!activado) { //Si el aire no está activado
                        fill(0,255,0); //Relleno Verde
                        //Círculo verde para decir que el aire esta en modo off

        ellipse(xCoordenadaAire+margen,yCoordenadaAire+alturaAire*2+alturaAire/2,radioElpise,radioElpise);
                }
                else {      //Si el aire está activado
                        fill(255,0,0); //Relleno Rojo
                        //Círculo rojo para decir que el aire no está en modo on

        ellipse(xCoordenadaAire+margen,yCoordenadaAire+alturaAire*2+alturaAire/2,radioElpise,radioElpise);
                }
                fill(0); //Letras negras
                //Temperatura off
                text("Tº   off   "+valorOff+"   ºC",xCoordenadaAire+2*margen,yCoordenadaAire+alturaAire*2+alturaAire/2);

        }
}
```

Apartado B

```
//Se añaden unos sensores los cuales encenderían las luces de las habitaciones
import processing.core.PApplet; //Se importa la Applet de Processing
import processing.core.PImage; //Se importa la librería PImage de Processing para poder trabajar con imágenes
import processing.serial.*; //Se importa la librería Serial de Processing

public class B extends PApplet{
	//Se declaran las variables
	int ancho=800,altura=800;
	Serial miPuerto;
	int valor,valorOn=15,valorOff=10;
	int xCoordenadaAire=100,yCoordenadaAire=125;
	int achoAire=150,alturaAire=60;
	int margen=10,radioElpise=10;
	int xImagen=470,yImagen=115;
	int velocidad=1;
	boolean aireActivo=false;

	Habitacion[] casa;
	PImage foto;

	public static void main(String[] args) {
		PApplet.main("B");
	}

	public void settings(){
		foto=new PImage();
		foto=loadImage("foto.png"); //Se carga la imagen

		//Se declara la variable en la cual se almacenarán las habitaciones
		casa=new Habitacion[5];

		//Se almacenan las habitaciones con sus coordenadas y longitudes correspondientes
		casa[0]=new Habitacion("A", 330,125, 120, 100,false);
		casa[1]=new Habitacion("B", 330,225, 120, 150,false);
		casa[2]=new Habitacion("C", 510,125, 110, 100,false);
		casa[3]=new Habitacion("D", 510,225, 110, 70,false);
		casa[4]=new Habitacion("E", 450,295, 170, 80,false);

		size(ancho,altura);
		//Se da tamaño a la pantalla
	}

	//Se ejecuta 1 sola vez
	public void setup(){
		mipuerto= new Serial(this, Serial.list()[2], 9600);		//Se conecta el puertoSerie de con el PC
		construirPantalla(); //Se llama al método construirPantalla
```

```
	}

	//Se ejecuta continuamente
	public void draw(){
		valor=0;
		if(miPuerto.available() > 0){ //Se mira si hay algún dato disponible en el puerto

			valor=miPuerto.read(); //Se lee el dato y lo se almacena en la variable "valor"
		}
		delay(250);
		activacionAire(); //Se llama al método activacióAire para la posible activación del aire
		construirMandoAire(aireActivo); //Se llama al método construirMandoAire para actualizar la temperatura
			busquedaHabitaciones();
	}


	private void busquedaHabitaciones() {
		for (Habitacion h: casa) {
			if(h.getNombre().equals("A")||h.getNombre().equals("B")) {		//Las
dos primeras habitaciones
				if(xImagen >= h.getXcoordenada()&& xImagen+foto.width<=
h.getXcoordenada()+h.getXlength()&&yImagen>= h.getYcoordenada()&&yImagen+foto.height
<=h.getYcoordenada()+h.getYlength()) {

					//Envía el nombre de la habitación para que el Arduino encienda el led
correspondiente
					miPuerto.write(h.getNombre());
					h.setDentro(true);
				}
				else {
					if(xImagen>= h.getXcoordenada()+h.getXlength()) {
						h.setDentro(false);
					}
				}
			}
			else {
					//Las otras habitaciones
				if(xImagen >= h.getXcoordenada()&& xImagen+foto.width<=
h.getXcoordenada()+h.getXlength()&&yImagen>= h.getYcoordenada()&&yImagen+foto.height
<=h.getYcoordenada()+h.getYlength()) {
					//Envía el nombre de la habitación para que el Arduino encienda el led
correspondiente
					miPuerto.write(h.getNombre());
					h.setDentro(true);
				}
				else {
					if(xImagen+foto.width <= h.getXcoordenada()) {
						h.setDentro(false);
```

```
                                        }
                                }
                        }
                }
                if(!dentroHabitación()) {
                        miPuerto.write("N");
                }

        }

        public void activacionAire() {
                if(valor>valorOn) {     //Si el valor supera la consigna de activación, se activa el aire
                        aireActivo=true;
                }
                else {
                        if(valor<valorOff) { //Si el valor es inferior a la consgina de desactivación, se desactiva el aire
                                aireActivo=false;
                        }
                }

        }

        //Método construirPantalla
        public void construirPantalla() {
                background(255); //Color de fondo de la pantalla, 255 equivale a blanco
                image(foto, xImagen,yImagen); //Se muestra la imagen en las coordenadas correspondientes
                construirMandoAire(aireActivo); //Se llama al método construirMandoAire
                construirHabitaciones(); //Se llama método construirHabitaciones
                construirPuertas(); //Se llama al método construirPuertas
        }

        //Método ConstruirHabitaciones, se crea un rectángulo por cada habitación en sus coordenadas correspondientes
        public void construirHabitaciones() {
                for (Habitacion h: casa) {
                        noFill();
                        rect(h.getXcoordenada(),h.getYcoordenada(), h.getXlength(), h.getYlength());
                        stroke(0);
                }
        }

        //Método ConstruirPuertas
        public void construirPuertas() {
                for (Habitacion h: casa) {
                        //Las dos primera habitaciones tendrán la puerta a la derecha
                        if(h.getNombre().equals("A")||h.getNombre().equals("B")) {
                                stroke(0,255,0);
                                line(h.getXcoordenada()+h.getXlength(),h.getYcoordenada(),
h.getXcoordenada()+h.getXlength(),h.getYcoordenada()+h.getYlength());
```

```
                                        }
                                        else {

                                                //Las otras habitaciones tendrán la puerta a la izquierda
                                                line(h.getXcoordenada(),h.getYcoordenada(),
h.getXcoordenada(),h.getYcoordenada()+h.getYlength());
                                        }
                                }
                        }

                        //Método construirMandoAire
                        public void construirMandoAire(boolean activado) {
                                //Primer recuadro, temperatura actual
                                stroke(0);
                                fill(255); //Relleno Blanco
                                rect(xCoordenadaAire,yCoordenadaAire,achoAire,alturaAire);
                                //Primer rectángulo
                                fill(0); //Letras negras
                                text("Tº "+valor+" ºC",xCoordenadaAire+8*margen,yCoordenadaAire+alturaAire/2); //Temperatura actual

                                //Segundo recuadro, temperatura de on
                                fill(255); //Relleno Blanco
                                rect(xCoordenadaAire,yCoordenadaAire+alturaAire,achoAire,alturaAire);
                        //Segundo rectángulo
                                if(activado) { //Si el aire está activado
                                        fill(0,255,0); //Relleno Verde

                                        //Círculo verde para decir que el aire está en modo on
                        ellipse(xCoordenadaAire+margen,yCoordenadaAire+alturaAire+alturaAire/2,radioElpise,radioElpise);
                                }
                                else { //Si el aire no está activado
                                        fill(255,0,0); //Relleno Rojo

                                        //Círculo rojo para decir que el aire no está en modo off
                        ellipse(xCoordenadaAire+margen,yCoordenadaAire+alturaAire+alturaAire/2,radioElpise,radioElpise);
                                }

                                //Tercer recuadro, temperatura de off
                                fill(0); //Letras negras
                                //Temperatura On
                                text("Tº    on    "+valorOn+"    ºC",xCoordenadaAire+2*margen,yCoordenadaAire+alturaAire+alturaAire/2);


                                fill(255); //Relleno Blanco
                                rect(xCoordenadaAire,yCoordenadaAire+alturaAire*2,achoAire,alturaAire);                 //Tercer rectángulo
                                if(!activado) { //Si el aire no está activado
                                        fill(0,255,0); //Relleno Verde
```

```
                    //Círculo verde para decir que el aire esta en modo off
ellipse(xCoordenadaAire+margen,yCoordenadaAire+alturaAire*2+alturaAire/2,radioElpise,radioElpise);
                }
                else {        //Si el aire está activado
                    fill(255,0,0); //Relleno Rojo

                    //Círculo rojo para decir que el aire no está en modo on
ellipse(xCoordenadaAire+margen,yCoordenadaAire+alturaAire*2+alturaAire/2,radioElpise,radioElpise);
                }
                fill(0); //Letras negras
                //Temperatura off
                text("Tº  off  "+valorOff+"  ºC",xCoordenadaAire+2*margen,yCoordenadaAire+alturaAire*2+alturaAire/2);

            }

            //Método en el cual entrará cuando se pulse una tecla
            public void keyPressed(){        //En caso de que la tecla pulsada sea la W, la persona se desplazará hacia arriba
                if((key=='w'||key=='W')){
                    if(dentroHabitación()) { //Dentro de la habitación, no se permite atravesar la pared superior
                        for (Habitacion h: casa) {
                            if(h.getDentro()) {
                                if(h.getYcoordenada()<yImagen) {
                                    yImagen-=velocidad; //Se disminuye la coordenada Y
                                    //La persona se deplazará hacia arriba
                                    construirPantalla();    //Se  llama  al  método
construirPantalla
                                }
                            }
                        }
                    }
                    else { //Fuera de la habitación, no se permite subir más la persona de el punto inicial
                        if(casa[0].getYcoordenada()<yImagen) {
                            yImagen-=velocidad; //Se disminuye la coordenada Y
                            //La persona se deplazará hacia arriba
                            construirPantalla(); //Se llama al método construirPantalla
                        }
                    }
                }

                //En caso de que la tecla pulsada sea la S, la persona se desplazará hacia abajo
                else if((key=='s'||key=='S')){
                    if(dentroHabitación()) { //Dentro de la habitación, no se permitirá atravesar la pared inferior
                        for (Habitacion h: casa) {
                            if(h.getDentro()) {
                                if(h.getYcoordenada()+h.getYlength()>yImagen+foto.height) {
                                    yImagen+=velocidad; //Se aumenta la coordenada Y
                                    //La persona se deplazará hacia abajo
```

```
                                                    construirPantalla();    //Se llama al método
construirPantalla
                                                }
                                            }
                                        }
                                    }
                                    else {      //Fuera de la habitación, no se permite bajar más desde la pared superior de la ultima
habitacion
                                        if(casa[4].getYcoordenada()>yImagen+foto.height) {
                                            yImagen+=velocidad; //Se aumenta la coordenada Y
                                            //La persona se deplazará hacia abajo
                                            construirPantalla(); //Se llama al método construirPantalla
                                        }
                                    }
                                }

                                //En caso de que la tecla pulsada sea la A, la persona se desplazará hacia la izquierda
                                else if((key=='a'||key=='A')){
                                    if(dentroHabitación()) { //Dentro de la habitación, no se permite atravesar la pared izquierda en
las 2 primeras habitaciones
                                        for (Habitacion h: casa) {
                                            if(h.getNombre().equals("A")||h.getNombre().equals("B")) {
                                                if(h.getXcoordenada()<xImagen) {
                                                    xImagen-=velocidad; //Se disminuye la coordenada X
                                                    //La persona se desplazará hacia la izquierda
                                                    construirPantalla();    //Se llama al método
construirPantalla
                                                }
                                            }
                                        }
                                    }
                                    else {
                                        xImagen-=velocidad;  //Se disminuye la coordenada X
                                        //La persona se desplazará hacia la izquierda
                                        construirPantalla(); //Se llama al método construirPantalla
                                    }
                                }

                                //En caso de que la tecla pulsada sea la D, la persona se desplazará hacia la derecha
                                else if((key=='d'||key=='D')){

                                    //Dentro de la habitación, no se permite atravesar la pared derecha en las ultimas 3 habitaciones
                                    if(dentroHabitación()) {
                                        for (Habitacion h: casa) {
                                            if(!(h.getNombre().equals("A")||h.getNombre().equals("B"))) {
                                                if(h.getXcoordenada()+h.getXlength()>xImagen+foto.width) {
                                                    xImagen+=velocidad; //Se aumenta la coordenada X
                                                    //La persona se desplazará hacia la derecha
```

```
                                                                        construirPantalla();    //Se    llama    al    método
construirPantalla
                                                                                }
                                                                        }
                                                                }
                                                        }
                                                        else {
                                                                xImagen+=velocidad; //Se aumenta la coordenada X
                                                                //La persona se desplazará hacia la derecha
                                                                construirPantalla(); //Se llama al método construirPantalla
                                                        }
                                                }

                                        }

                                        //Método dentroHabitación
                                        public boolean dentroHabitación(){
                                                for (Habitacion h: casa) {
                                                        if(h.getDentro()) { //Si está dentro de alguna habitación, devolverá true; si está en el pasillo, false
                                                                return true;
                                                        }
                                                }
                                                return false;
                                        }
}
```

Código Arduino

```
int led2=2;
int led3=3;
int led4=4;
int led5=5;
int led6=6;
int temp;

void setup() {
 Serial.begin(9600);
 pinMode(led2,OUTPUT);
 pinMode(led3,OUTPUT);
 pinMode(led4,OUTPUT);
 pinMode(led5,OUTPUT);
 pinMode(led6,OUTPUT);
 }

void loop() {

  temp=analogRead(A0);
 temp=(5*temp*100)/1023.0;
 Serial.write(temp);
 delay(250);


 if(Serial.available()>0){
  byte dato=Serial.read();

  if(dato==69){
    digitalWrite(led2,HIGH);
    digitalWrite(led3,LOW);
    digitalWrite(led4,LOW);
    digitalWrite(led5,LOW);
```

```
  digitalWrite(led6,LOW);
 }
  else if(dato==68){
  digitalWrite(led3,HIGH);
    digitalWrite(led2,LOW);
  digitalWrite(led4,LOW);
  digitalWrite(led5,LOW);
  digitalWrite(led6,LOW);
 }
 else if(dato==67){
  digitalWrite(led4,HIGH);
    digitalWrite(led3,LOW);
  digitalWrite(led2,LOW);
  digitalWrite(led5,LOW);
  digitalWrite(led6,LOW);
 }
 else if(dato==66){
  digitalWrite(led5,HIGH);
    digitalWrite(led3,LOW);
  digitalWrite(led4,LOW);
  digitalWrite(led2,LOW);
  digitalWrite(led6,LOW);
 }
else   if(dato==65){
  digitalWrite(led6,HIGH);
    digitalWrite(led3,LOW);
  digitalWrite(led4,LOW);
  digitalWrite(led5,LOW);
  digitalWrite(led2,LOW);
 }
 else {
  digitalWrite(led2,LOW);
```

```
      digitalWrite(led3,LOW);
    digitalWrite(led4,LOW);
    digitalWrite(led5,LOW);
    digitalWrite(led6,LOW);
   }
  }
}
```

Arduino

Práctica 0

Código Arduino

```
int entrada= 11;
int salida = 5;

void setup() {

  pinMode(entrada,INPUT);
  pinMode(salida,OUTPUT);
}

void loop() {



  if(digitalRead(entrada)==HIGH){
  digitalWrite(salida,HIGH);
  }
 if(digitalRead(entrada)==LOW){
  digitalWrite(salida,LOW);
  }
}
```

Vaadin

Práctica 0

Diseño Prueba

```
//Diseño  ejemplo
package com.example.prueba;

//Se importan las librerías necesarias
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.annotations.DesignRoot;
import com.vaadin.data.HasValue.ValueChangeEvent;
import com.vaadin.data.HasValue.ValueChangeListener;
import com.vaadin.ui.Button;
import com.vaadin.ui.TextField;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.declarative.Design;
@DesignRoot
@AutoGenerated
@SuppressWarnings("serial")
public class ejemplo extends VerticalLayout {
	protected TextField text1;
	protected TextField text2;
	protected TextField text3;
	protected TextField text4;

	//Constructor del diseño
	public ejemplo() {
		Design.read(this);
	}

	//Setters y  Getters de las variables
	public TextField getText1() {
		return text1;}
	public void setText1(TextField text1) {
		this.text1 = text1;
	}

	public TextField getText2() {
		return text2;}
	public void setText2(TextField text2) {
		this.text2 = text2;
	}

	public TextField getText3() {
		return text3;}
	public void setText3(TextField text3) {
		this.text3 = text3;
```

```
	}

	public TextField getText4() {
		return text4;}
	public void setText4(TextField text4) {
		this.text4 = text4;
	}
}
```

Test

```
/Test
package com.example.prueba;

//Se importan las librerías necesarias
import javax.servlet.annotation.WebServlet;

import com.vaadin.annotations.Theme;
import com.vaadin.annotations.VaadinServletConfiguration;
import com.vaadin.data.HasValue.ValueChangeEvent;
import com.vaadin.data.HasValue.ValueChangeListener;
import com.vaadin.server.VaadinRequest;
import com.vaadin.server.VaadinServlet;
import com.vaadin.ui.Button;
import com.vaadin.ui.Label;
import com.vaadin.ui.TextField;
import com.vaadin.ui.UI;
import com.vaadin.ui.VerticalLayout;

@Theme("mytheme")
public class MyUI extends UI {

	@Override
	protected void init(VaadinRequest vaadinRequest) {

		//Declaración del diseño ejemplo
		ejemplo layout = new ejemplo();

		//Se muestra el diseño ejemplo por pantalla
		setContent(layout);

		//No se permite escribir en los texField 3 y 4
		layout.getText3().setEnabled(false);
		layout.getText4().setEnabled(false);

		//Cuando se modifique el textField 1
		layout.getText1().addValueChangeListener(new ValueChangeListener<String>() {
			@Override
			public void valueChange(ValueChangeEvent<String> event) {
```

```
                        //Se escribe automáticamente del texto 1 al texto 4
                        layout.getText4().setValue(layout.getText1().getValue());
                    }
                });{
                }

                //Cuando se modifique el textField 2
                layout.getText2().addValueChangeListener(new ValueChangeListener<String>() {
                    @Override
                    public void valueChange(ValueChangeEvent<String> event) {
                        //Se escribe automáticamente del texto 2 al texto 3
                        layout.getText3().setValue(layout.getText2().getValue());
                    }
                });
            }

            @WebServlet(urlPatterns = "/*", name = "MyUIServlet", asyncSupported = true)
            @VaadinServletConfiguration(ui = MyUI.class, productionMode = false)
            public static class MyUIServlet extends VaadinServlet {
            }
}
```

Práctica 1

Diseño A

```
//Diseño  A
package com.example.formulario;

//Se importan las librerías necesarias
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.annotations.DesignRoot;
import com.vaadin.ui.ComboBox;
import com.vaadin.ui.FormLayout;
import com.vaadin.ui.InlineDateField;
import com.vaadin.ui.RadioButtonGroup;
import com.vaadin.ui.TextField;
import com.vaadin.ui.declarative.Design;

/**
 * !! DO NOT EDIT THIS FILE !!
 *
 * This class is generated by Vaadin Designer and will be overwritten.
 *
 * Please make a subclass with logic and additional interfaces as needed,
 * e.g class LoginView extends LoginDesign implements View { }
 */
@DesignRoot
@AutoGenerated
@SuppressWarnings("serial")
public class A extends FormLayout {
		protected TextField textNombre;
		protected TextField textApellido1;
		protected TextField textApellido2;
		protected RadioButtonGroup<java.lang.String> radioButtonGenero;
		protected ComboBox<java.lang.String> comboPais;
		protected InlineDateField dataFieldDia;

		//Constructor del diseño
		public A() {
			Design.read(this);
		}
}
```

Test A

```
//Test
package com.example.formulario;

//Se importan las librerías necesarias
import javax.servlet.annotation.WebServlet;
import com.example.formulario.A;
import com.vaadin.annotations.Theme;
import com.vaadin.annotations.VaadinServletConfiguration;
import com.vaadin.server.VaadinRequest;
import com.vaadin.server.VaadinServlet;
import com.vaadin.ui.Button;
import com.vaadin.ui.Label;
import com.vaadin.ui.TextField;
import com.vaadin.ui.UI;
import com.vaadin.ui.VerticalLayout;

/**
 * This UI is the application entry point. A UI may either represent a browser window
 * (or tab) or some part of an HTML page where a Vaadin application is embedded.
 * <p>
 * The UI is initialized using {@link #init(VaadinRequest)}. This method is intended to be
 * overridden to add component to the user interface and initialize non-component functionality.
 */
@Theme("mytheme")
public class MyUI extends UI {

	@Override
	protected void init(VaadinRequest vaadinRequest) {

		//Declaración del diseño A
		A layout = new A();

		//Se muestra el diseño A por pantalla
		setContent(layout);
	}

	@WebServlet(urlPatterns = "/*", name = "MyUIServlet", asyncSupported = true)
	@VaadinServletConfiguration(ui = MyUI.class, productionMode = false)
	public static class MyUIServlet extends VaadinServlet {
	}
}
```

Diseño B

```
//Diseño  B
package com.example.formulario;

//Se importan las librerías necesarias
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.annotations.DesignRoot;
import com.vaadin.ui.Button;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.Button.ClickListener;
import com.vaadin.ui.ComboBox;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.InlineDateField;
import com.vaadin.ui.RadioButtonGroup;
import com.vaadin.ui.TextArea;
import com.vaadin.ui.TextField;
import com.vaadin.ui.declarative.Design;

/**
 * !! DO NOT EDIT THIS FILE !!
 *
 * This class is generated by Vaadin Designer and will be overwritten.
 *
 * Please make a subclass with logic and additional interfaces as needed,
 * e.g class LoginView extends LoginDesign implements View { }
 */
@DesignRoot
@AutoGenerated
@SuppressWarnings("serial")
public class B extends HorizontalLayout {
	protected TextField textNombre;
	protected TextField textApellido1;
	protected TextField textApellido2;
	protected ComboBox<java.lang.String> comboPais;
	protected RadioButtonGroup<java.lang.String> radioButtonGenero;
	protected InlineDateField dataFieldDia;
	protected TextArea textAreaDatos;
	protected Button botonGuardar;

	//Constructor del diseño
	public B() {
		Design.read(this);
	}

	//Setters Getters de las variables
	public TextField getTextNombre() {
```

```
                return textNombre;}
        public void setTextNombre(TextField textNombre) {
                this.textNombre = textNombre;
        }

        public TextField getTextApellido1() {
                return textApellido1;}
        public void setTextApellido1(TextField textApellido1) {
                this.textApellido1 = textApellido1;
        }

        public TextField getTextApellido2() {
                return textApellido2;}
        public void setTextApellido2(TextField textApellido2) {
                this.textApellido2 = textApellido2;
        }

        public ComboBox<java.lang.String> getComboPais() {
                return comboPais;}
        public void setComboPais(ComboBox<java.lang.String> comboPais) {
                this.comboPais = comboPais;
        }

        public RadioButtonGroup<java.lang.String> getRadioButtonGenero() {
                return radioButtonGenero;}
        public void setRadioButtonGenero(RadioButtonGroup<java.lang.String> radioButtonGenero) {
                this.radioButtonGenero = radioButtonGenero;
        }

        public InlineDateField getDataFieldDia() {
                return dataFieldDia;}
        public void setDataFieldDia(InlineDateField dataFieldDia) {
                this.dataFieldDia = dataFieldDia;
        }

        public TextArea getTextAreaDatos() {
                return textAreaDatos;}
        public void setTextAreaDatos(TextArea textAreaDatos) {
                this.textAreaDatos = textAreaDatos;
        }

        public Button getBotonGuardar() {
                return botonGuardar;}
        public void setBotonGuardar(Button botonGuardar) {
                this.botonGuardar = botonGuardar;
        }
}
```

Test B

```
//Test
package com.example.formulario;

//Se importan las librerías necesarias
import javax.servlet.annotation.WebServlet;

import com.example.formulario.B;
import com.vaadin.annotations.Theme;
import com.vaadin.annotations.VaadinServletConfiguration;
import com.vaadin.server.VaadinRequest;
import com.vaadin.server.VaadinServlet;
import com.vaadin.ui.Button;
import com.vaadin.ui.Label;
import com.vaadin.ui.TextField;
import com.vaadin.ui.UI;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.Button.ClickListener;


@Theme("mytheme")
public class MyUI extends UI {

	@Override
	protected void init(VaadinRequest vaadinRequest) {

		//Declaración del diseño B
		B layout = new B();

		//Se muestra el diseño B por pantalla
		setContent(layout);


		//Cuando se clique el botón de Guardar
		layout.getBotonGuardar().addClickListener(new ClickListener() {
			@Override
			public void buttonClick(ClickEvent event) {
				//Se escribe en el Area de texto una pequeña descripción de la persona con la
información de los campos
				layout.getTextAreaDatos().setValue("Mi nombre es
"+layout.getTextNombre().getValue()+" "+layout.getTextApellido1().getValue()+" "+layout.getTextApellido2().getValue()+".");
				layout.getTextAreaDatos().setValue(layout.getTextAreaDatos().getValue()+"\n"+"Soy
"+layout.getRadioButtonGenero().getValue()+" y vengo de "+layout.getComboPais().getValue()+".");
				layout.getTextAreaDatos().setValue(layout.getTextAreaDatos().getValue()+"\n"+"Nací
el día "+layout.getDataFieldDia().getValue());
```

```
                        }
                    });
            }

            @WebServlet(urlPatterns = "/*", name = "MyUIServlet", asyncSupported = true)
            @VaadinServletConfiguration(ui = MyUI.class, productionMode = false)
            public static class MyUIServlet extends VaadinServlet {
            }
}
```

Diseño C

```
//Diseño  C
package com.example.formulario;

//Se importan las librerías necesarias
import java.time.LocalDate;
import java.time.chrono.ChronoLocalDate;
import java.util.Date;

import com.vaadin.annotations.AutoGenerated;
import com.vaadin.annotations.DesignRoot;
import com.vaadin.ui.Button;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.Button.ClickListener;
import com.vaadin.ui.ComboBox;
import com.vaadin.ui.DateField;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.InlineDateField;
import com.vaadin.ui.Label;
import com.vaadin.ui.RadioButtonGroup;
import com.vaadin.ui.TextArea;
import com.vaadin.ui.TextField;
import com.vaadin.ui.declarative.Design;

/**
 * !! DO NOT EDIT THIS FILE !!
 *
 * This class is generated by Vaadin Designer and will be overwritten.
 *
 * Please make a subclass with logic and additional interfaces as needed,
 * e.g class LoginView extends LoginDesign implements View { }
 */
@DesignRoot
@AutoGenerated
@SuppressWarnings("serial")
public class C extends HorizontalLayout {
	protected TextField textNombre;
	protected TextField textApellido1;
	protected TextField textApellido2;
	protected ComboBox<java.lang.String> comboPais;
	protected RadioButtonGroup<java.lang.String> radioButtonGenero;
	protected InlineDateField dataFieldDia;
	protected TextArea textAreaDatos;
	protected Button botonGuardar;
	protected Label label1;

	//Constructor del diseño
```

```
public C() {
		Design.read(this);
}

//Setters Getters de las variables
public TextField getTextNombre() {
		return textNombre;}
public void setTextNombre(TextField textNombre) {
		this.textNombre = textNombre;
}

public TextField getTextApellido1() {
		return textApellido1;}
public void setTextApellido1(TextField textApellido1) {
		this.textApellido1 = textApellido1;
}

public TextField getTextApellido2() {
		return textApellido2;}
public void setTextApellido2(TextField textApellido2) {
		this.textApellido2 = textApellido2;
}

public ComboBox<java.lang.String> getComboPais() {
		return comboPais;}
public void setComboPais(ComboBox<java.lang.String> comboPais) {
		this.comboPais = comboPais;
}

public RadioButtonGroup<java.lang.String> getRadioButtonGenero() {
		return radioButtonGenero;}
public void setRadioButtonGenero(RadioButtonGroup<java.lang.String> radioButtonGenero) {
		this.radioButtonGenero = radioButtonGenero;
}

public InlineDateField getDataFieldDia() {
		return dataFieldDia;}
public void setDataFieldDia(InlineDateField dataFieldDia) {
		this.dataFieldDia = dataFieldDia;
}

public TextArea getTextAreaDatos() {
		return textAreaDatos;}
public void setTextAreaDatos(TextArea textAreaDatos) {
		this.textAreaDatos = textAreaDatos;
}

public Button getBotonGuardar() {
```

```
                return botonGuardar;}
        public void setBotonGuardar(Button botonGuardar) {
                this.botonGuardar = botonGuardar;
        }

        public Label getLabel1() {
                return label1;}
        public void setLabel1(Label label1) {
                this.label1 = label1;
        }
}
```

Test C

```
//Test
package com.example.formulario;

import java.time.LocalDate;

//Se importan las librerías necesarias
import javax.servlet.annotation.WebServlet;

import com.example.formulario.C;
import com.vaadin.annotations.Theme;
import com.vaadin.annotations.VaadinServletConfiguration;
import com.vaadin.server.VaadinRequest;
import com.vaadin.server.VaadinServlet;
import com.vaadin.ui.Button;
import com.vaadin.ui.DateField;
import com.vaadin.ui.Label;
import com.vaadin.ui.TextField;
import com.vaadin.ui.UI;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.Button.ClickListener;

/**
 * This UI is the application entry point. A UI may either represent a browser window
 * (or tab) or some part of an HTML page where a Vaadin application is embedded.
 * <p>
 * The UI is initialized using {@link #init(VaadinRequest)}. This method is intended to be
 * overridden to add component to the user interface and initialize non-component functionality.
 */
@Theme("mytheme")
public class MyUI extends UI {

	@Override
	protected void init(VaadinRequest vaadinRequest) {


		//Declaración del diseño C
		C layout = new C();

		//Se muestra el diseño C por pantalla
		setContent(layout);

		//Cuando se clique el botón de Guardar
		layout.botonGuardar.addClickListener(new ClickListener() {
			@Override
			public void buttonClick(ClickEvent event) {
```

```
boolean datosCorrectos = true;

//Se limpian el area de texto
layout.getTextAreaDatos().clear();

//No se muestra nada por el label1
layout.getLabel1().setValue(null);

//Se crea una variable de clase DateField con la fecha actual
DateField date = new DateField();

//Si el Nombre es correcto, se permitirá escribir
if (!stringCorrecto(layout.getTextNombre().getValue())&datosCorrectos) {
        datosCorrectos=false;
        layout.getLabel1().setValue("Nombre incorrecto,mínimo 3 caracteres.");
}

//Si el primer apellido es correcto, se permitirá escribir
if (!stringCorrecto(layout.getTextApellido1().getValue())&datosCorrectos) {
        datosCorrectos=false;
        layout.getLabel1().setValue("Primer apellido incorrecto, mínimo 3
caracteres.");
}

//Si el segundo apellido es correcto, se permitiáos escribir
if (!stringCorrecto(layout.getTextApellido2().getValue())&datosCorrectos) {
        datosCorrectos=false;
        layout.getLabel1().setValue("Segundo apellido incorrecto, mínimo 3
caracteres.");
}

//Si se ha seleccionado algun país, se permtiirá escribir
if(layout.getComboPais().getValue()==null&datosCorrectos) {
        datosCorrectos=false;
        layout.getLabel1().setValue("No ha elegido ningún país.");
}

//Si se ha seleccionado algun género, se permtiirá escribir
if(layout.getRadioButtonGenero().getValue()==null&datosCorrectos) {
        datosCorrectos=false;
        layout.getLabel1().setValue("No ha elegido ningún género.");
}

//Si se ha seleccionado una fecha anterior a la fecha actual, se permitirá escribir
if(layout.getDataFieldDia().getValue().isAfter(LocalDate.now())&datosCorrectos) {
        datosCorrectos=false;
        layout.getLabel1().setValue("La fecha es posterior a hoy");
}
```

```
                                        //Si todos los datos son correctos
                                        if(datosCorrectos) {
                                                //Se escribe en el Area de texto una pequeña descripción de la persona con la información de los campos
                                                layout.getTextAreaDatos().setValue("Mi nombre es "+layout.getTextNombre().getValue()+" "+layout.getTextApellido1().getValue()+" "+layout.getTextApellido2().getValue()+".");

                layout.getTextAreaDatos().setValue(layout.getTextAreaDatos().getValue()+"\n"+"Soy "+layout.getRadioButtonGenero().getValue()+" y vengo de "+layout.getComboPais().getValue()+".");

                layout.getTextAreaDatos().setValue(layout.getTextAreaDatos().getValue()+"\n"+"Nací el día "+layout.getDataFieldDia().getValue());
                                        }
                                }
                        });
        }

        //método StringCorrecto, devuelve un booleano en función de si el String tiene más de 2 caracteres
        boolean stringCorrecto(String palabra) {
                if(palabra.length()>2){
                        return true;
                }
                return false;
        }

        @WebServlet(urlPatterns = "/*", name = "MyUIServlet", asyncSupported = true)
        @VaadinServletConfiguration(ui = MyUI.class, productionMode = false)
        public static class MyUIServlet extends VaadinServlet {
        }
}
```

Práctica 2

Diseño A

```
//Diseño A
package com.example.noticias;

//Se importan las librerías necesarias
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.annotations.DesignRoot;
import com.vaadin.ui.Button;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.TextField;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.declarative.Design;

@DesignRoot
@AutoGenerated
@SuppressWarnings("serial")
public class A extends HorizontalLayout {
	protected VerticalLayout verticalPublicacion;
	protected TextField textoPublicacion;
	protected Button botonPublicar;
	protected VerticalLayout verticalMuro;

	//Constructor del diseño
	public A() {
		Design.read(this);
	}

	//Setters y Getters de las variables
	public VerticalLayout getVerticalPublicacion() {
		return verticalPublicacion;}
	public void setVerticalPublicacion(VerticalLayout verticalPublicacion) {
		this.verticalPublicacion = verticalPublicacion;
	}

	public TextField getTextoPublicacion() {
		return textoPublicacion;}
	public void setTextoPublicacion(TextField textoPublicacion) {
		this.textoPublicacion = textoPublicacion;
	}

	public Button getBotonPublicar() {
		return botonPublicar;}
	public void setBotonPublicar(Button botonPublicar) {
		this.botonPublicar = botonPublicar;
	}
```

```
	public VerticalLayout getVerticalMuro() {
		return verticalMuro;}
	public void setVerticalMuro(VerticalLayout verticalMuro) {
		this.verticalMuro = verticalMuro;
	}
}
```

Test

```
//Test
package com.example.noticias;

//Se importan las librerías necesarias
import javax.servlet.annotation.WebServlet;

import com.vaadin.annotations.Theme;
import com.vaadin.annotations.VaadinServletConfiguration;
import com.vaadin.server.ClassResource;
import com.vaadin.server.FileResource;
import com.vaadin.server.VaadinRequest;
import com.vaadin.server.VaadinServlet;
import com.vaadin.ui.Button;
import com.vaadin.ui.Label;
import com.vaadin.ui.TextField;
import com.vaadin.ui.UI;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.Button.ClickListener;
import com.vaadin.ui.Image;

@Theme("mytheme")
public class MyUI extends UI {

	@Override
	protected void init(VaadinRequest vaadinRequest) {
		final int numeroMaximoPublicaciones=8;
		final int numeroMaximoCaracteres=30;
		//Declaración del diseño
		A layout = new A();


		//Se muestra el diseño por pantalla
		setContent(layout);


		layout.getTextoPublicacion().setMaxLength(numeroMaximoCaracteres);

		//Cuando se clique el botón publicar
		layout.getBotonPublicar().addClickListener(new ClickListener() {
			@Override
			public void buttonClick(ClickEvent event) {

				String texto = new String();
				//Se recoge lo escrito en el TextField
				texto=layout.getTextoPublicacion().getValue();
```

```
                                        if (texto!="") {          //Si hay algo escrito
                                                //Si ya hay x publicaciones hechas, se elimina la última
                                                if
(layout.getVerticalMuro().getComponentCount()==numeroMaximoPublicaciones) {

            layout.getVerticalMuro().removeComponent(layout.getVerticalMuro().getComponent(7));
                                                }

                                                //Se publica la publicación arriba de todo
                                                layout.getVerticalMuro().addComponent(new Label(texto),0);

                                                //Se vuelve limpiar el textField
                                                layout.getTextoPublicacion().setValue("");
                                        }

                                }
                        });
            }

            @WebServlet(urlPatterns = "/*", name = "MyUIServlet", asyncSupported = true)
            @VaadinServletConfiguration(ui = MyUI.class, productionMode = false)
            public static class MyUIServlet extends VaadinServlet {
            }
}
```

Práctica 3

Clase Club

```
//Clase Club
package com.example.registro;

//Se importan las librerías necesarias
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.Map;
import java.util.TreeMap;


public class Club implements Serializable{
	private Map<String, Persona> usuarios;

	//Constructor de la clase Club
	public Club(){
		usuarios=new TreeMap<String, Persona> ();
	}
	//Getters de la variable
	public Map<String, Persona> getAlumnos(){
		return usuarios;
	}
	//Método registrar, se añade un usuario nuevo al club
	public void registrar(String usu,String contra, String nombre, String descripcion){
		usuarios.put(usu,new Persona(usu,contra,nombre,descripcion));
	}


	//Se cargan todos los usuarios que aparezcan en un fichero de texto, y añaden al club
	public void cargar(String nomFitxer) {
		Club o;
		try {
			FileInputStream fis = new FileInputStream(nomFitxer);
			ObjectInputStream ois = new ObjectInputStream(fis);
			o = (Club) ois.readObject();
			ois.close();
		} catch (IOException e) {
			o = new Club();
		} catch (ClassNotFoundException e) {
			o = new Club();
		}
		this.usuarios = o.usuarios;
```

```
	}

	//Se guardan todos los usuarios del club en un fichero de texto
	public void guardar(String nombreArchivo) {
		try {
			FileOutputStream fos = new FileOutputStream(nombreArchivo);
			ObjectOutputStream oos = new ObjectOutputStream(fos);
			oos.writeObject(this);
			oos.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	//Método que devuelve si existe ese usuario o no
	public boolean containsKey(String key) {
		return this.usuarios.containsKey(key);
	}

	//método que devuelve la Persona con ese usuario
	public Persona getPersona(String key) {
		return this.usuarios.get(key);
	}

	//Método eliminarPerfil, elimina el perfil abierto del club
	public void eliminarPerfil(String usuarioActual) {
		usuarios.remove(usuarioActual);
	}

	//Método eliminarTodosLosPerfiles, limpia el treemap, es decir, elimina todos los usuarios del club
	public void eliminarTodosLosPerfiles() {
		usuarios.clear();
	}
}
```

Clase Persona

```
//Clase Persona
package com.example.registro;

import java.io.Serializable;

public class Persona implements Serializable {
	private String usuario;
	private String contraseña;
	private String nombre;
	private String descripcion;
	//Constructor clase Persona
	public Persona(String usuario,String contraseña,String nombre, String descrpicion) {

		this.usuario = usuario;
		this.contraseña = contraseña;
		this.nombre=nombre;
		this.descripcion=descrpicion;
	}
	//Setters y getters
	public String getUsuario() {
		return usuario;}
	public void setUsuario(String usuario) {
		this.usuario = usuario;
	}

	public String getContraseña() {
		return contraseña;}
	public void setContraseña(String contraseña) {
		this.contraseña = contraseña;
	}

	public String getNombre() {
		return nombre;}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	public String getDescripcion() {
		return descripcion;}
	public void setDescripcion(String descripcion) {
		this.descripcion = descripcion;
	}
}
```

Diseño InicioSesion

```
//Diseño de Inicio Sesion
package com.example.registro;

//Se importan las librerías necesarias
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.annotations.DesignRoot;
import com.vaadin.ui.Button;
import com.vaadin.ui.Label;
import com.vaadin.ui.PasswordField;
import com.vaadin.ui.TextField;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.declarative.Design;

/**
 * !! DO NOT EDIT THIS FILE !!
 *
 * This class is generated by Vaadin Designer and will be overwritten.
 *
 * Please make a subclass with logic and additional interfaces as needed,
 * e.g class LoginView extends LoginDesign implements View { }
 */
@DesignRoot
@AutoGenerated
@SuppressWarnings("serial")
public class InicioSesion extends VerticalLayout {
	protected Button inicioBotonRegistrar;
	protected TextField inicioUsuario;
	protected PasswordField inicioContraseña;
	protected Button inicioBoton;
	protected Label inicioMensaje;

	//Constructor del diseño
	public InicioSesion() {
		Design.read(this);
	}

	//Setters y Getters de las variables
	public Button getInicioBotonRegistrar() {
		return inicioBotonRegistrar;}
	public void setInicioBotonRegistrar(Button inicioBotonRegistrar) {
		this.inicioBotonRegistrar = inicioBotonRegistrar;
	}

	public TextField getInicioUsuario() {
		return inicioUsuario;}
	public void setInicioUsuario(TextField inicioUsuario) {
```

```
                this.inicioUsuario = inicioUsuario;
        }

        public PasswordField getInicioContraseña() {
                return inicioContraseña;}
        public void setInicioContraseña(PasswordField inicioContraseña) {
                this.inicioContraseña = inicioContraseña;
        }

        public Button getInicioBoton() {
                return inicioBoton;}
        public void setInicioBoton(Button inicioBoton) {
                this.inicioBoton = inicioBoton;
        }

        public Label getInicioMensaje() {
                return inicioMensaje;}
        public void setInicioMensaje(Label inicioMensaje) {
                this.inicioMensaje = inicioMensaje;
        }
}
```

Diseño Perfil

```
//Diseño de Perfil
package com.example.registro;

//Se importan las librerías necesarias
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.annotations.DesignRoot;
import com.vaadin.ui.Button;
import com.vaadin.ui.Label;
import com.vaadin.ui.TextArea;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.declarative.Design;

/**
 * !! DO NOT EDIT THIS FILE !!
 *
 * This class is generated by Vaadin Designer and will be overwritten.
 *
 * Please make a subclass with logic and additional interfaces as needed,
 * e.g class LoginView extends LoginDesign implements View { }
 */
@DesignRoot
@AutoGenerated
@SuppressWarnings("serial")
public class Perfil extends VerticalLayout {
		protected Button perfilCerrar;
		protected Button perfilEliminar;
		protected Button perfilEditar;
		protected Label perfilUsuario;
		protected Label perfilNombre;
		protected TextArea perfilDescripcion;
		protected Button perfilGuardar;
		protected Label perfilMensaje;

		//Constructor del diseño
		public Perfil() {
			Design.read(this);
		}

		//Setters y Getters de las variables
		public Button getPerfilCerrar() {
			return perfilCerrar;}
		public void setPerfilCerrar(Button perfilCerrar) {
			this.perfilCerrar = perfilCerrar;
		}

		public Button getPerfilEliminar() {
```

```
                return perfilEliminar;}
        public void setPerfilEliminar(Button perfilEliminar) {
                this.perfilEliminar = perfilEliminar;
        }

        public Button getPerfilEditar() {
                return perfilEditar;}
        public void setPerfilEditar(Button perfilEditar) {
                this.perfilEditar = perfilEditar;
        }

        public Label getPerfilUsuario() {
                return perfilUsuario;   }
        public void setPerfilUsuario(Label perfilUsuario) {
                this.perfilUsuario = perfilUsuario;
        }

        public Label getPerfilNombre() {
                return perfilNombre;  }
        public void setPerfilNombre(Label perfilNombre) {
                this.perfilNombre = perfilNombre;
        }

        public TextArea getPerfilDescripcion() {
                return perfilDescripcion;            }
        public void setPerfilDescripcion(TextArea perfilDescripcion) {
                this.perfilDescripcion = perfilDescripcion;
        }

        public Button getPerfilGuardar() {
                return perfilGuardar;}
        public void setPerfilGuardar(Button perfilGuardar) {
                this.perfilGuardar = perfilGuardar;
        }

        public Label getPerfilMensaje() {
                return perfilMensaje;}
        public void setPerfilMensaje(Label perfilMensaje) {
                this.perfilMensaje = perfilMensaje;
        }
}
```

Diseño Principal

```
//Diseño Principal
package com.example.registro;

//Se importan las librerías necesarias
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.annotations.DesignRoot;
import com.vaadin.ui.Button;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.declarative.Design;

/**
 * !! DO NOT EDIT THIS FILE !!
 *
 * This class is generated by Vaadin Designer and will be overwritten.
 *
 * Please make a subclass with logic and additional interfaces as needed,
 * e.g class LoginView extends LoginDesign implements View { }
 */
@DesignRoot
@AutoGenerated
@SuppressWarnings("serial")
public class Principal extends HorizontalLayout {
	protected Button principalBotonSesion;
	protected Button principalbotonRegistrar;

	//Constructor del diseño
	public Principal() {
		Design.read(this);
	}

	//Setters y Getters de las variables
	public Button getPrincipalBotonSesion() {
		return principalBotonSesion;}
	public void setPrincipalBotonSesion(Button principalBotonSesion) {
		this.principalBotonSesion = principalBotonSesion;
	}

	public Button getPrincipalbotonRegistrar() {
		return principalbotonRegistrar;}
	public void setPrincipalbotonRegistrar(Button principalbotonRegistrar) {
		this.principalbotonRegistrar = principalbotonRegistrar;
	}
}
```

Diseño Registro

```
//Diseño  Registro
package com.example.registro;
//Se importan las librerías necesarias
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.annotations.DesignRoot;
import com.vaadin.ui.Button;
import com.vaadin.ui.Label;
import com.vaadin.ui.PasswordField;
import com.vaadin.ui.TextArea;
import com.vaadin.ui.TextField;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.declarative.Design;

/**
 * !! DO NOT EDIT THIS FILE !!
 *
 * This class is generated by Vaadin Designer and will be overwritten.
 *
 * Please make a subclass with logic and additional interfaces as needed,
 * e.g class LoginView extends LoginDesign implements View { }
 */
@DesignRoot
@AutoGenerated
@SuppressWarnings("serial")
public class Registro extends VerticalLayout {
		protected Button registrarBotonSesion;
		protected Button registrarBotonEliminarPerfiles;
		protected TextField registrarUsuario;
		protected PasswordField registrarContraseña;
		protected TextField registrarNombre;
		protected TextArea regstrarDescrpicion;
		protected Button registrarBoton;
		protected Label registrarMensaje;

		//Constructor del diseño
		public Registro() {
			Design.read(this);
		}

		//Setters Getters de las variables
		public Button getRegistrarBotonSesion() {
			return registrarBotonSesion;}
		public void setRegistrarBotonSesion(Button registrarBotonSesion) {
			this.registrarBotonSesion = registrarBotonSesion;
		}
```

```
	public Button getRegistrarBotonEliminarPerfiles() {
		return registrarBotonEliminarPerfiles;}
	public void setRegistrarBotonEliminarPerfiles(Button registrarBotonEliminarPerfiles) {
		this.registrarBotonEliminarPerfiles = registrarBotonEliminarPerfiles;
	}

	public TextField getRegistrarUsuario() {
		return registrarUsuario;}
	public void setRegistrarUsuario(TextField registrarUsuario) {
		this.registrarUsuario = registrarUsuario;
	}

	public PasswordField getRegistrarContraseña() {
		return registrarContraseña;}
	public void setRegistrarContraseña(PasswordField registrarContraseña) {
		this.registrarContraseña = registrarContraseña;
	}

	public TextField getRegistrarNombre() {
		return registrarNombre;}
	public void setRegistrarNombre(TextField registrarNombre) {
		this.registrarNombre = registrarNombre;
	}

	public TextArea getRegistrarDescrpicion() {
		return regstrarDescrpicion;}
	public void setRegistrarDescrpicion(TextArea regstrarDescrpicion) {
		this.regstrarDescrpicion = regstrarDescrpicion;
	}

	public Button getRegistrarBoton() {
		return registrarBoton;}
	public void setRegistrarBoton(Button registrarBoton) {
		this.registrarBoton = registrarBoton;
	}

	public Label getRegistrarMensaje() {
		return registrarMensaje;}
	public void setRegistrarMensaje(Label registrarMensaje) {
		this.registrarMensaje = registrarMensaje;
	}
}
```

Test

```
/Test
package com.example.registro;

//Se importan las librerías necesarias
import java.util.TreeMap;

import javax.servlet.annotation.WebServlet;

import com.vaadin.annotations.Theme;
import com.vaadin.annotations.VaadinServletConfiguration;
import com.vaadin.data.HasValue.ValueChangeEvent;
import com.vaadin.data.HasValue.ValueChangeListener;
import com.vaadin.event.selection.SingleSelectionEvent;
import com.vaadin.event.selection.SingleSelectionListener;
import com.vaadin.server.VaadinRequest;
import com.vaadin.server.VaadinServlet;
import com.vaadin.ui.Button;
import com.vaadin.ui.Label;
import com.vaadin.ui.TextField;
import com.vaadin.ui.UI;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.Button.ClickListener;
import com.vaadin.ui.themes.ValoTheme;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

/**
 * This UI is the application entry point. A UI may either represent a browser
 * window (or tab) or some part of an HTML page where a Vaadin application is
 * embedded.
 * <p>
 * The UI is initialized using {@link #init(VaadinRequest)}. This method is
 * intended to be overridden to add component to the user interface and
 * initialize non-component functionality.
 */
@Theme("mytheme")
public class MyUI extends UI {
        //Declaración archivo donde se guardarán los datos
        final String nombreArchivo = "Lista_usuarios.dat";
        String usuarioActual;

        @Override
```

```
protected void init(VaadinRequest vaadinRequest) {
        //Declaración del club
        Club miClub = new Club();

        //Recuperar los usuarios previamente registrados
        miClub.cargar(nombreArchivo);

        //Declaración de los diferentes diseños
        Registro layoutRegistrar = new Registro();
        InicioSesion layoutSesion = new InicioSesion();
        Principal layoutPrincipal = new Principal();
        Perfil layoutPerfil= new Perfil();

        //Se muestra el diseño Principal por pantalla
        setContent(layoutPrincipal);

        //En la pantalla principal, cuando se clique el botón de Página de Inicio Sesión
        layoutPrincipal.getPrincipalBotonSesion().addClickListener(new ClickListener() {
                @Override
                public void buttonClick(ClickEvent event) {
                        //Se muestra el diseño InicioSesión por pantalla
                        mostrarPantallaInicio(layoutSesion);
                }
        });

        //En la pantalla principal, cuando se clique el botón de Página de Registro
        layoutPrincipal.getPrincipalbotonRegistrar().addClickListener(new ClickListener() {
                @Override
                public void buttonClick(ClickEvent event) {
                        //Se muestra el diseño Registro por pantalla
                        mostrarPantallaRegistrar(layoutRegistrar);
                }
        });

        //En la pantalla de inicio sesión, cuando se clique el botón de Página de Registro
        layoutSesion.getInicioBotonRegistrar().addClickListener(new ClickListener() {
                @Override
                public void buttonClick(ClickEvent event) {
                        //Se muestra el diseño Registro por pantalla
                        mostrarPantallaRegistrar(layoutRegistrar);
                }
        });

        //En la pantalla de registrar, cuando se clique el botón de Página de Inicio
        layoutRegistrar.getRegistrarBotonSesion().addClickListener(new ClickListener() {
                @Override
                public void buttonClick(ClickEvent event) {
                        //Se muestra el diseño InicioSesión por pantalla
```

```
                    mostrarPantallaInicio(layoutSesion);
                }
            });

            //Cuando se clique al Boton de Registrar
            layoutRegistrar.getRegistrarBoton().addClickListener(new ClickListener() {
                @Override
                public void buttonClick(ClickEvent event) {
                    //Si todos los campos tienen un mínimo de 3 caracteres, se permitirá registrar
                    if
(!stringCorrecto(layoutRegistrar.getRegistrarUsuario().getValue())||!stringCorrecto(layoutRegistrar.getRegistrarContraseña().getV
alue())||!stringCorrecto(layoutRegistrar.getRegistrarNombre().getValue())||!stringCorrecto(layoutRegistrar.getRegistrarDescrpicio
n().getValue())) {
                        //No se permite registrar por campos rellenados incorrectamente, se
muestra un mensaje por pantalla
                        layoutRegistrar.getRegistrarMensaje().setValue("Datos incorrectos, mínimo
3 caracteres en cada campo.");
                    }
                    else{
                        //Si el usuario no existe, se permitirá registrar
                        if(miClub.containsKey(layoutRegistrar.getRegistrarUsuario().getValue())) {
                            //No se permite registrar por que el usuario ya existe, se muestra
mensaje por pantalla
                            layoutRegistrar.getRegistrarMensaje().setValue("Usuario      ya
existente");
                        }
                        else {
                            //Se registra el usuario en el club

        miClub.registrar(layoutRegistrar.getRegistrarUsuario().getValue(),
layoutRegistrar.getRegistrarContraseña().getValue(),layoutRegistrar.getRegistrarNombre().getValue(),layoutRegistrar.getRegistrar
Descrpicion().getValue());

                            //Cada vez que se registre algún usuario nuevo, se sobreescribe
el archivo
                            miClub.guardar(nombreArchivo);

                            //Se muestra mensaje por pantalla conforme el usuario se ha
registrado correctamente
                            layoutRegistrar.getRegistrarMensaje().setValue("Usuario
registrado correctamente");
                        }
                    }
                    //Se limpian los campos
                    layoutRegistrar.getRegistrarMensaje().setVisible(true);
                    layoutRegistrar.getRegistrarUsuario().setValue("");
                    layoutRegistrar.getRegistrarContraseña().setValue("");
                    layoutRegistrar.getRegistrarNombre().setValue("");
```

```
                    layoutRegistrar.getRegistrarDescrpicion().setValue("");
                }
            });

            //Cuando se clique al Boton de Eliminar Todos los perfiles
            layoutRegistrar.getRegistrarBotonEliminarPerfiles().addClickListener(new ClickListener() {
                @Override
                public void buttonClick(ClickEvent event) {
                    //Se eliminan todos los usuarios del club
                    miClub.eliminarTodosLosPerfiles();

                    //Cada vez que se eliminen todos los usuarios, se sobreescribe el archivo
                    miClub.guardar(nombreArchivo);

                    //Se muestra mensaje por pantalla conforme se han eliminado todos los perfiles correctamente
                    layoutRegistrar.getRegistrarMensaje().setValue("Todos los perfiles han sido eliminados");
                }
            });

            //Cuando se clique al Boton de InicioSesion
            layoutSesion.getInicioBoton().addClickListener(new ClickListener() {
                @Override
                public void buttonClick(ClickEvent event) {
                    //Si no se rellena los campos Usuario y Contraseña no se permitirá iniciar sesión
                    if (layoutSesion.getInicioUsuario().getValue()!=null||layoutSesion.getInicioContraseña().getValue()!=null) {
                        //Si el usuario no existe, no se permitirá iniciar sesión
                        if (miClub.containsKey(layoutSesion.getInicioUsuario().getValue())) {
                            //Si el usuario y contraseña no coinciden, no se permitirá inicar sesión
                            if(miClub.getPersona(layoutSesion.getInicioUsuario().getValue()).getContraseña().equals(layoutSesion.getInicioContraseña().getValue())) {
                                //Se guarda el usuario en esta variable
                                usuarioActual=layoutSesion.getInicioUsuario().getValue();

                                //Se muestra el diseño Perfil por pantalla
                                setContent(layoutPerfil);

                                //No permite visualizar la tecla guardar, ya que de momento no se modifica el perfil del usuario
                                layoutPerfil.getPerfilGuardar().setVisible(false);

                                //Se muestran los datos del usuario
```

```
                                                    layoutPerfil.getPerfilUsuario().setValue("Bievenido
"+miClub.getPersona(usuarioActual).getUsuario()+"!");
                                                    //Nombre

            layoutPerfil.getPerfilNombre().setValue(miClub.getPersona(usuarioActual).getNombre());
                                                    //Descripción

            layoutPerfil.getPerfilDescripcion().setValue(miClub.getPersona(usuarioActual).getDescripcion());

                                                    //No se permite modificar la descipcion, ya que de
momento no se modifica el perfil
                                                    layoutPerfil.getPerfilDescripcion().setEnabled(false);

                                                    //No se permite la visualización del mensaje
                                                    layoutPerfil.getPerfilMensaje().setVisible(false);
                                                }
                                                else {
                                                    //Se permite visualizar el mensaje
                                                    layoutSesion.getInicioMensaje().setVisible(true);

                                                    //Se muestra el mensaje por pantalla conforme el
usuario y contraseña no son correctos
                                                    layoutSesion.getInicioMensaje().setValue("Usuario y
contraseña incorrectos");

                                                    //Se limpian los campos
                                                    layoutSesion.getInicioUsuario().setValue("");
                                                    layoutSesion.getInicioContraseña().setValue("");

                                                }
                                            }
                                            else{ //Se muestra el mensaje por pantalla conforme el usuario no existe
                                                    layoutSesion.getInicioMensaje().setValue("Usuario no
existente");

                                                    //Se limpian los campos
                                                    layoutSesion.getInicioUsuario().setValue("");
                                                    layoutSesion.getInicioContraseña().setValue("");

                                                }
                                        }
                                }
                        });

                        //Cuando se clique al Boton de CerrarSesión
                        layoutPerfil.getPerfilCerrar().addClickListener(new ClickListener() {
                                @Override
                                public void buttonClick(ClickEvent event) {
```

```
                    //Se muestra el diseño InicioSesion por pantalla
                    mostrarPantallaInicio(layoutSesion);
                }
            });

            //Cuando se clique al Boton de EliminarCuenta
            layoutPerfil.getPerfilEliminar().addClickListener(new ClickListener() {
                @Override
                public void buttonClick(ClickEvent event) {
                    //se elimina el usuario seleccionado del club
                    miClub.eliminarPerfil(usuarioActual);

                    //Se muestra el diseño InicioSesion por pantalla
                    mostrarPantallaInicio(layoutSesion);
                }
            });

            //Cuando se clique al Boton de EditarCuenta
            layoutPerfil.getPerfilEditar().addClickListener(new ClickListener() {
                @Override
                public void buttonClick(ClickEvent event) {
                    //Se permite modificar los datos del usuario
                    layoutPerfil.getPerfilDescripcion().setEnabled(true);

                    //La tecla para guardar la modificacion se hace visible
                    layoutPerfil.getPerfilGuardar().setVisible(true);
                }
            });

            //Cuando se clique al Boton de GuardarModificación
            layoutPerfil.getPerfilGuardar().addClickListener(new ClickListener() {
                @Override
                public void buttonClick(ClickEvent event) {
                    //Se modifica la descripción del usuario

        miClub.getPersona(usuarioActual).setDescripcion(layoutPerfil.getPerfilDescripcion().getValue());

                    //Se permite la visualizacion del mensaje
                    layoutPerfil.getPerfilMensaje().setVisible(true);

                    //Se muestra mensaje por pantalla conforme se ha modificado el perfil del usuario
correctamente
                    layoutPerfil.getPerfilMensaje().setValue("Modifcación guardada");

                    //se vuelve a no permitir la modificación de la descripción
                    layoutPerfil.getPerfilDescripcion().setEnabled(false);

                    //Se vuelve a ocultar el boton de guardar la modificación
```

```
                                layoutPerfil.getPerfilGuardar().setVisible(false);

                                //Siempre que se modifique el perfil de un usuario, se sobreescribe el archivo
                                miClub.guardar(nombreArchivo);
                        }
                });
}

//método StringCorrecto, devuelve un booleano en función de si el String tiene más de 2 caracteres
boolean stringCorrecto(String palabra) {
                if (palabra.length() > 2) {
                        return true;
                }
                return false;
}

//Se muestra por pantalla el diseño de Inicio Sesion
public void mostrarPantallaInicio(InicioSesion layout) {
                //Cambio de pantalla a la de Inicio Sesión
                setContent(layout);

                //Siempre que se entre en la pantalla se limpian los textFields y los label
                layout.getInicioMensaje().setValue("");
                layout.getInicioMensaje().setVisible(false);
                layout.getInicioUsuario().setValue("");
                layout.getInicioContraseña().setValue("");
}

//Se muestra por pantalla el diseño de Registro
public void mostrarPantallaRegistrar( Registro layout) {
                //Cambio de pantalla a la de Registro
                setContent(layout);

                //Siempre que se entre en la pantalla se limpian los textFields y los label
                layout.getRegistrarMensaje().setValue("");
                layout.getRegistrarMensaje().setVisible(false);
                layout.getRegistrarUsuario().setValue("");
                layout.getRegistrarContraseña().setValue("");
                layout.getRegistrarNombre().setValue("");
                layout.getRegistrarDescrpicion().setValue("");
}

@WebServlet(urlPatterns = "/*", name = "MyUIServlet", asyncSupported = true)
@VaadinServletConfiguration(ui = MyUI.class, productionMode = false)
public static class MyUIServlet extends VaadinServlet {

}
```

```
}
```

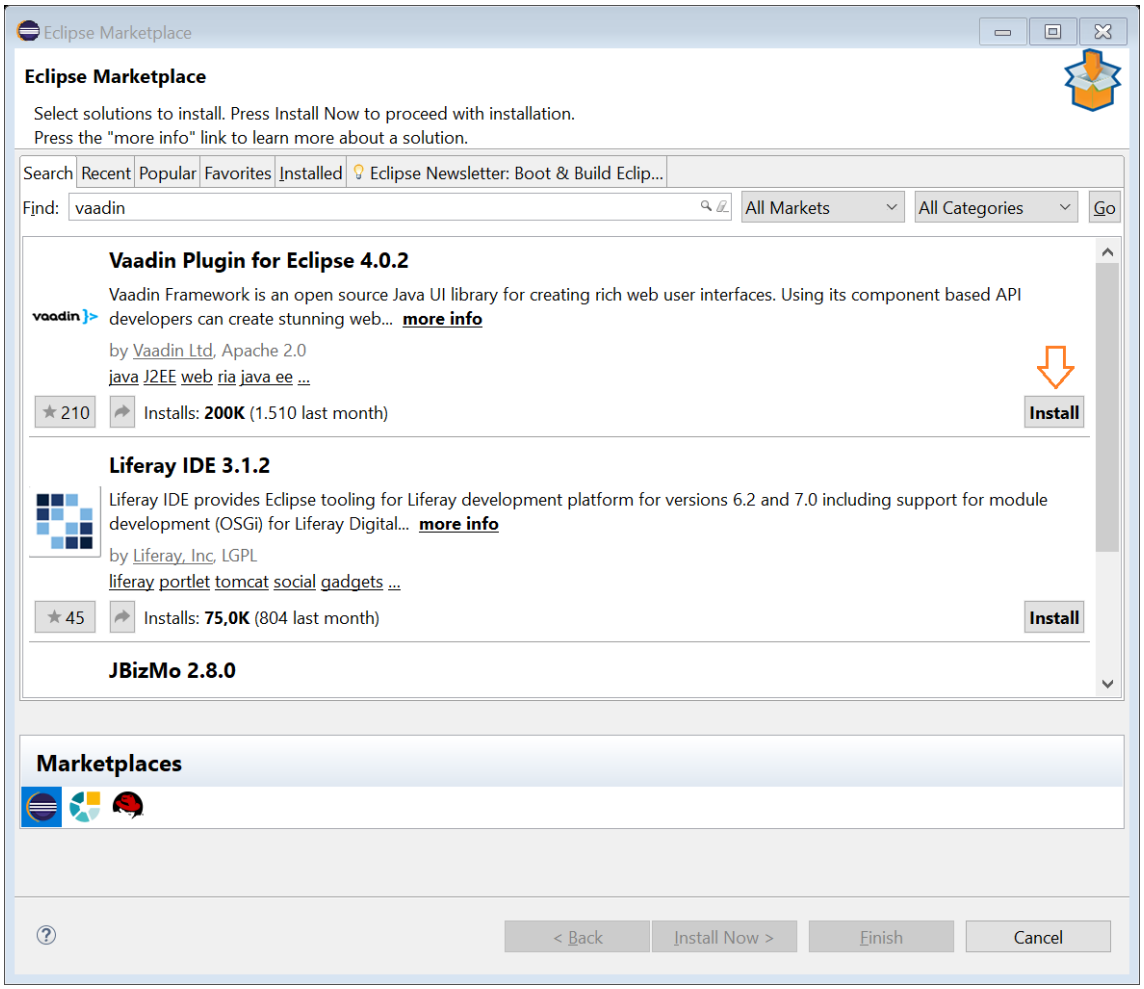
ANEXO C: MANUALES Y DOCUMENTACIÓN PARA EL PROFESORADO

MANUAL INSTALACIÓN VAADIN DESIGNER

1. Instalar el Plugin de Vaadin para eclipse.

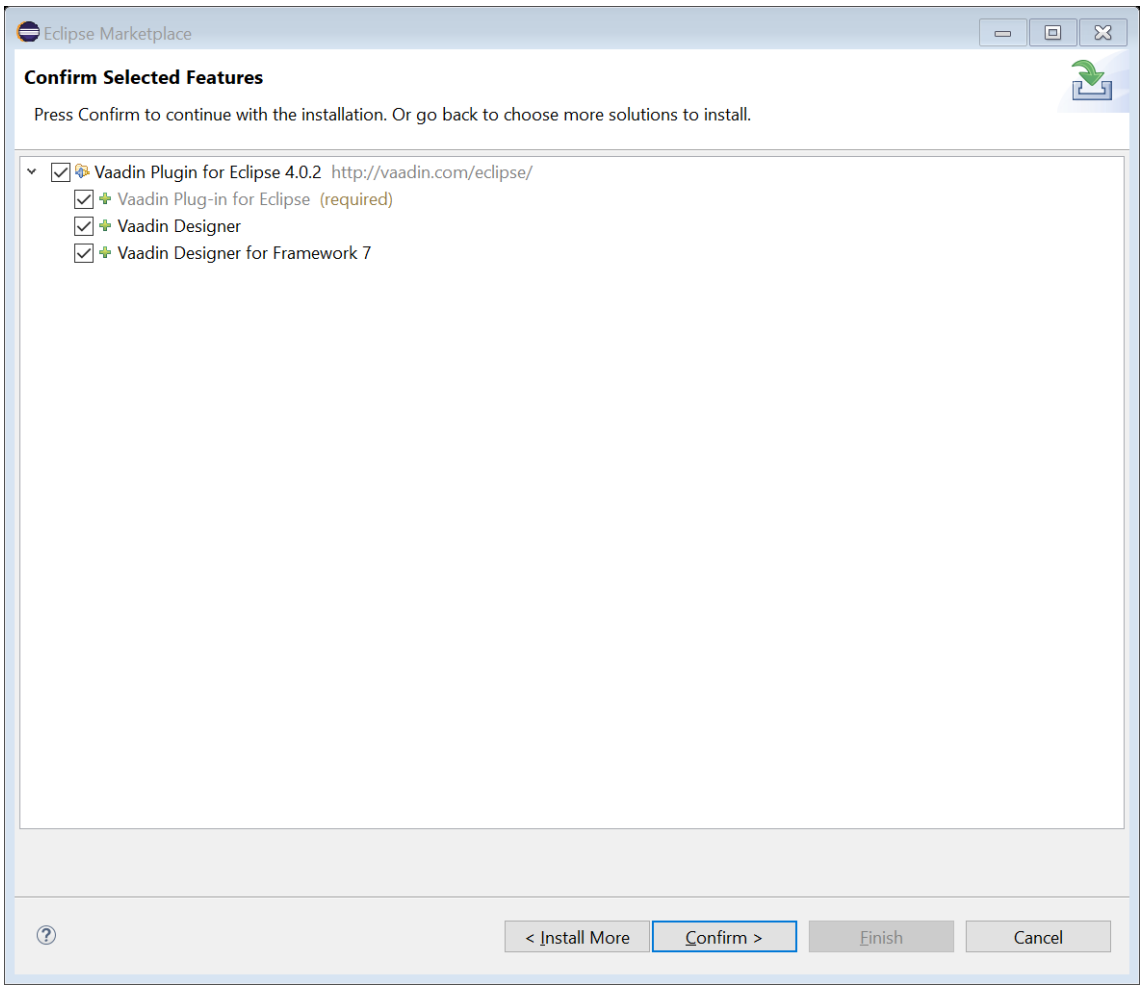
   Help > Eclipse MarketPlace…

   Se hace una búsqueda para encontrar el Plugin de Vaadin fácilmente.



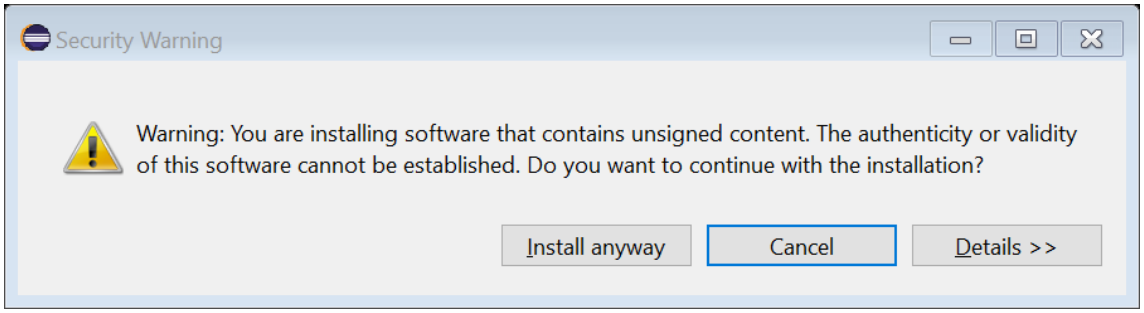
Marketplace de Eclipse donde se instalará el plugin de Vaadin.

2. Seleccionar todas las características y confirmar.

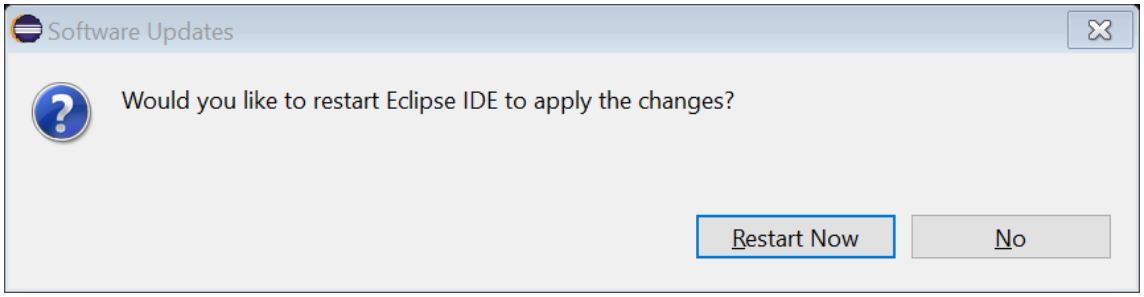


Selección de características de Vaadin.

3. Durante la instalación aparecerá una advertencia, seleccionamos instalar de todas formas y se reseteará Eclipse.



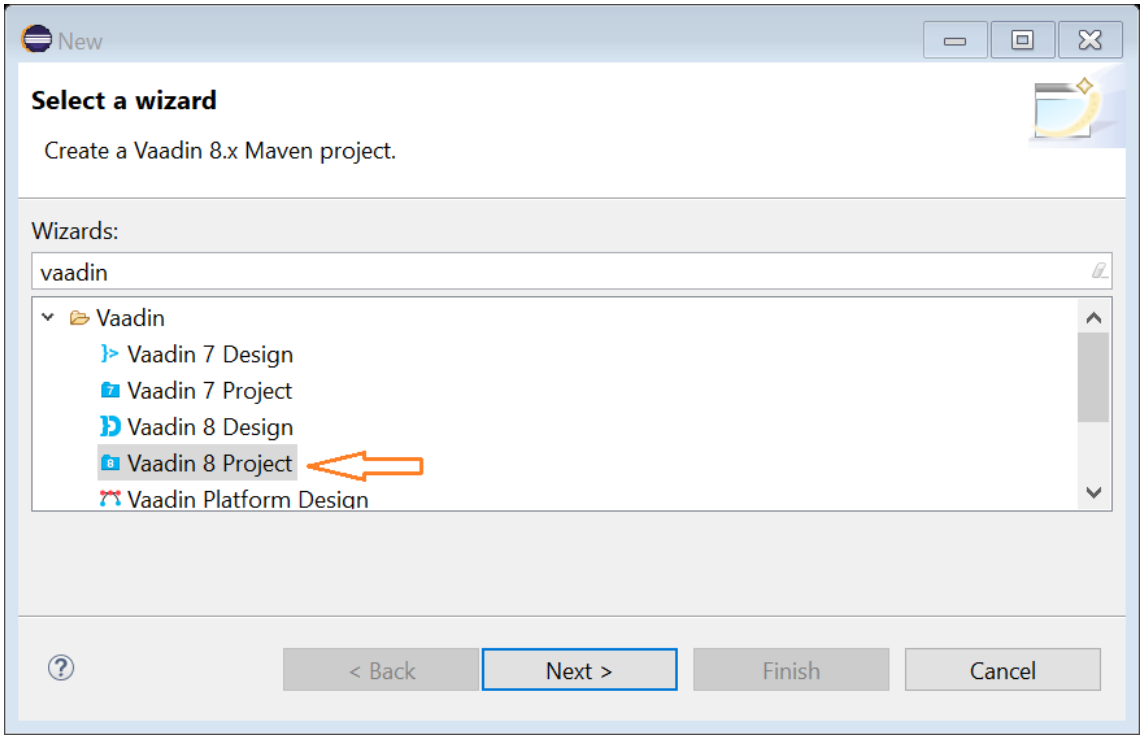
Selección Instalar de todas formas.



Selección Resetear Ahora.

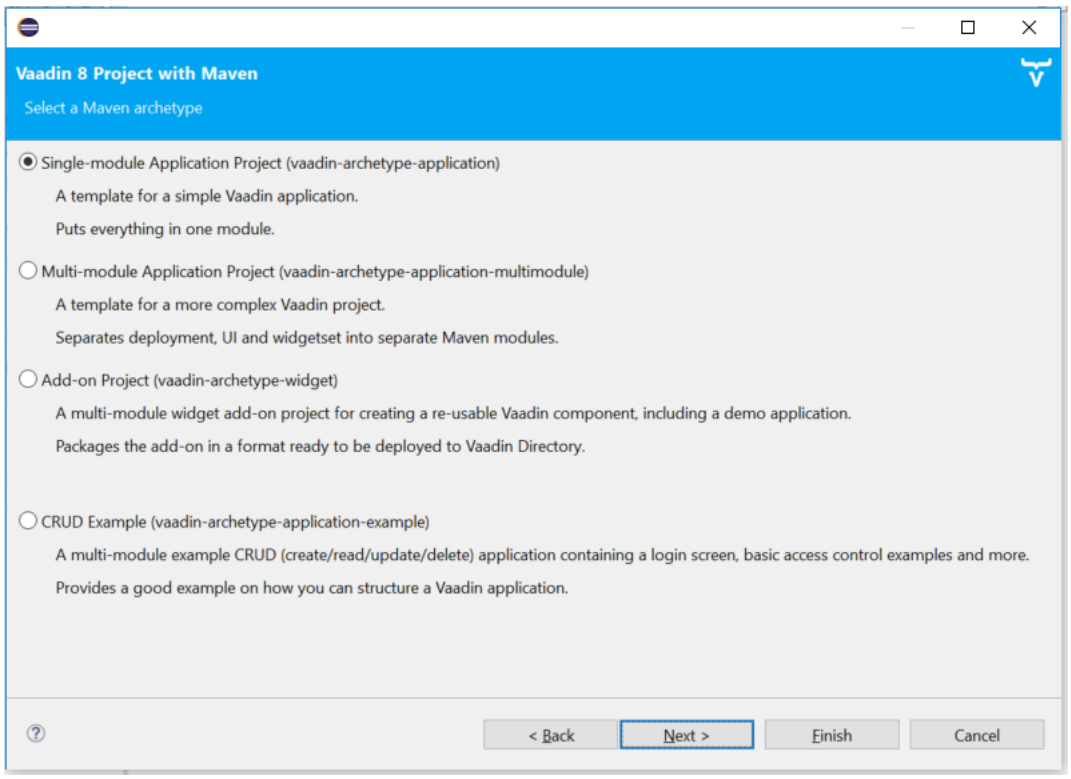
4. Crear un nuevo Proyecto de *Vaadin*.

File > New > Other > Vaadin > Vaadin 8 Project;



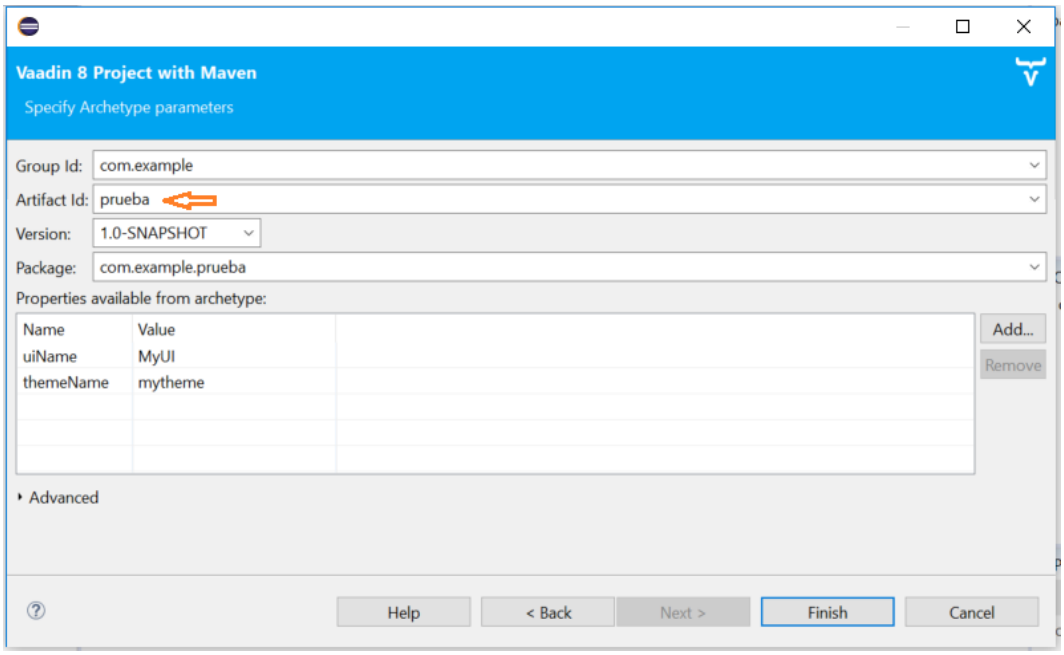
Creación proyecto *Vaadin*.

5. Seleccionar el prototipo Maven sencillo.



Selección Single-Mode Application Project.

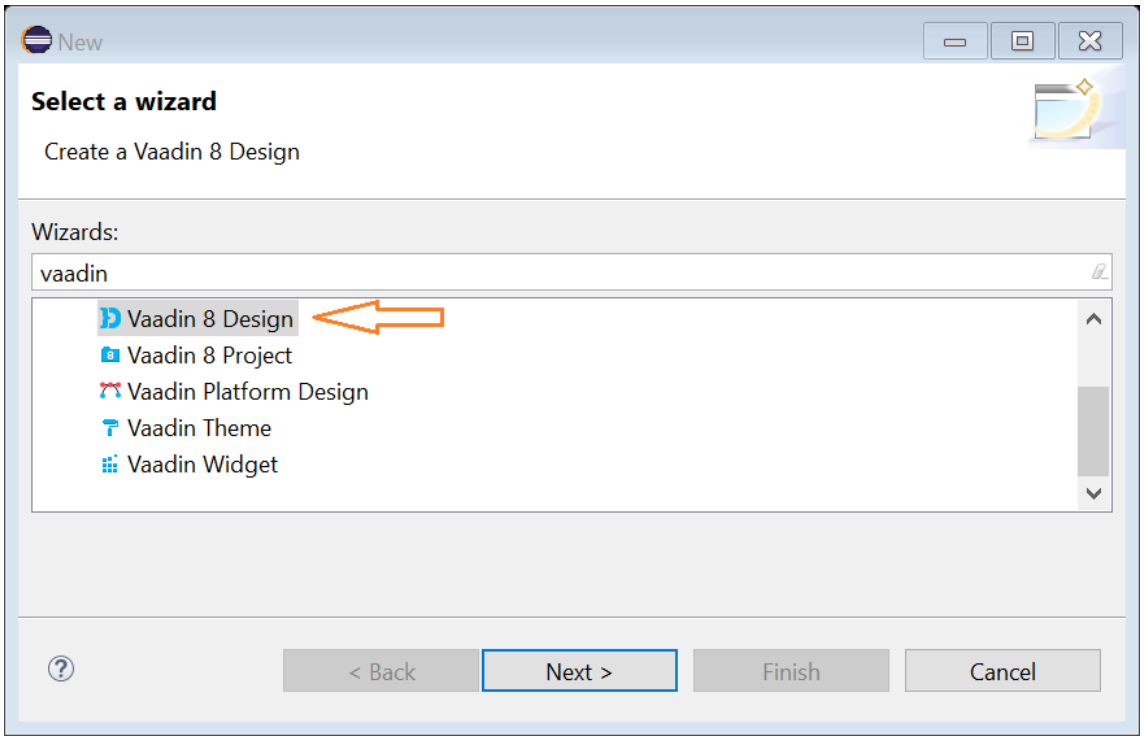
6. Dar nombre al proyecto.



Nombramiento del proyecto.

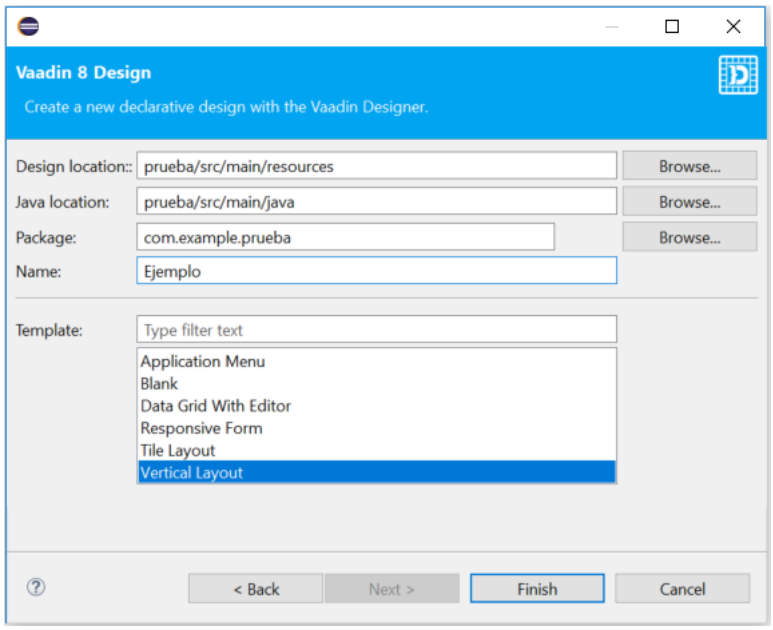
7. Crear el diseño de Vaadin.

File > New > Other > Vaadin > Vaadin 8 Design;



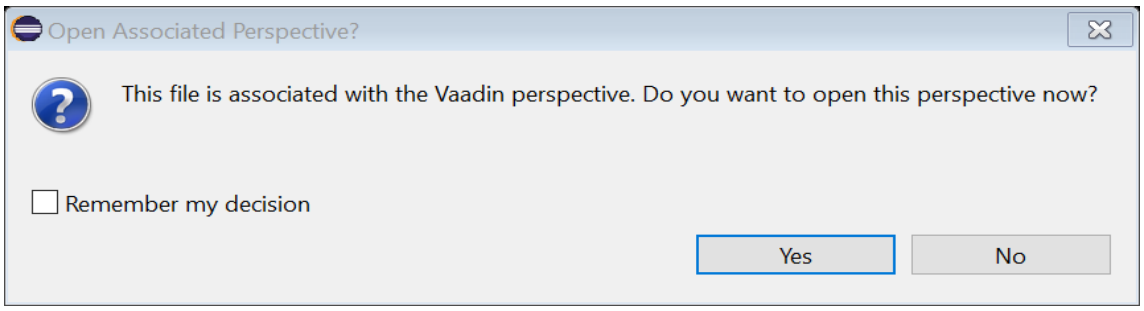
Creación diseño de Vaadin.

8. Dar nombre al diseño.



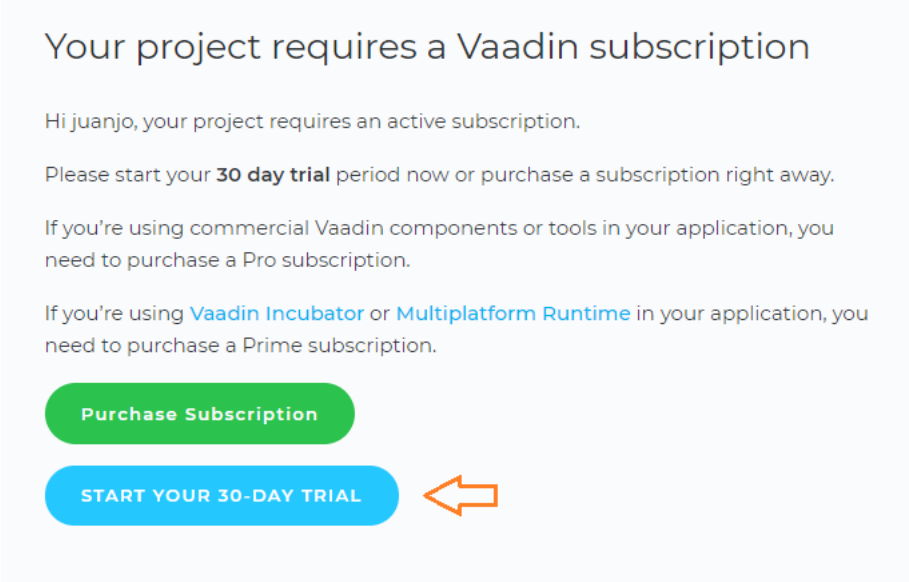
Nombramiento del diseño.

9. Aceptar cambiar la vista de Eclipse



Aceptación de cambiar la vista de Eclipse.

10. *Eclipse* pedirá la licencia mediante una advertencia, se entrará en la página web que se indica y en ella se seleccionará usar un trial de 30 días.



Selección prueba trial de 30 días.

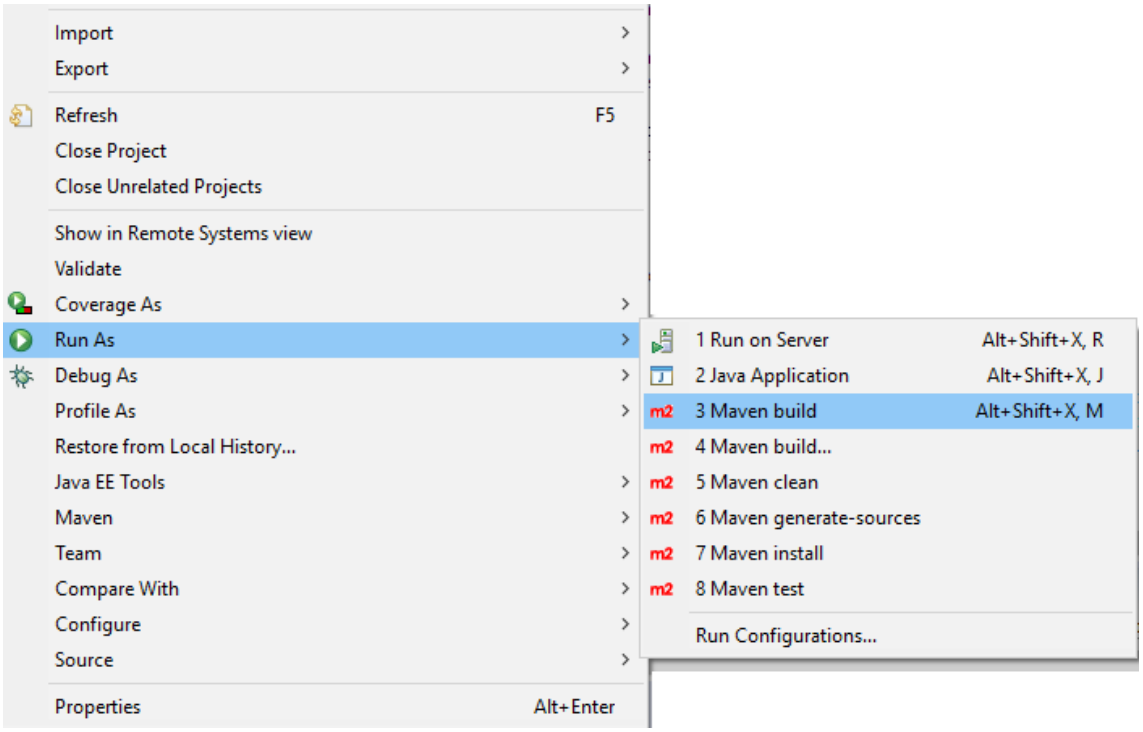
De esta forma, ya se podría empezar a diseñar la página web.

Por defecto, se crea una clase *MyUI* la cual consta de una página web muy sencillo con un *TextField*, un *Button* y un *Label*. Se escribirá cualquier nombre en el *TextField*, y clicando al Button se mostrará un mensaje “Hola + nombre +, It Works” en el *Label*.

```java
@Theme("mytheme")
public class MyUI extends UI {

    @Override
    protected void init(VaadinRequest vaadinRequest) {
        final VerticalLayout layout = new VerticalLayout();

        final TextField name = new TextField();
        name.setCaption("Type your name here:");

        Button button = new Button("Click Me");
        button.addClickListener(e -> {
            layout.addComponent(new Label("Thanks " + name.getValue()
                    + ", it works!"));
        });

        layout.addComponents(name, button);

        setContent(layout);
    }

    @WebServlet(urlPatterns = "/*", name = "MyUIServlet", asyncSupported = true)
    @VaadinServletConfiguration(ui = MyUI.class, productionMode = false)
    public static class MyUIServlet extends VaadinServlet {
    }
}
```

Clase MyUI creada por defecto.

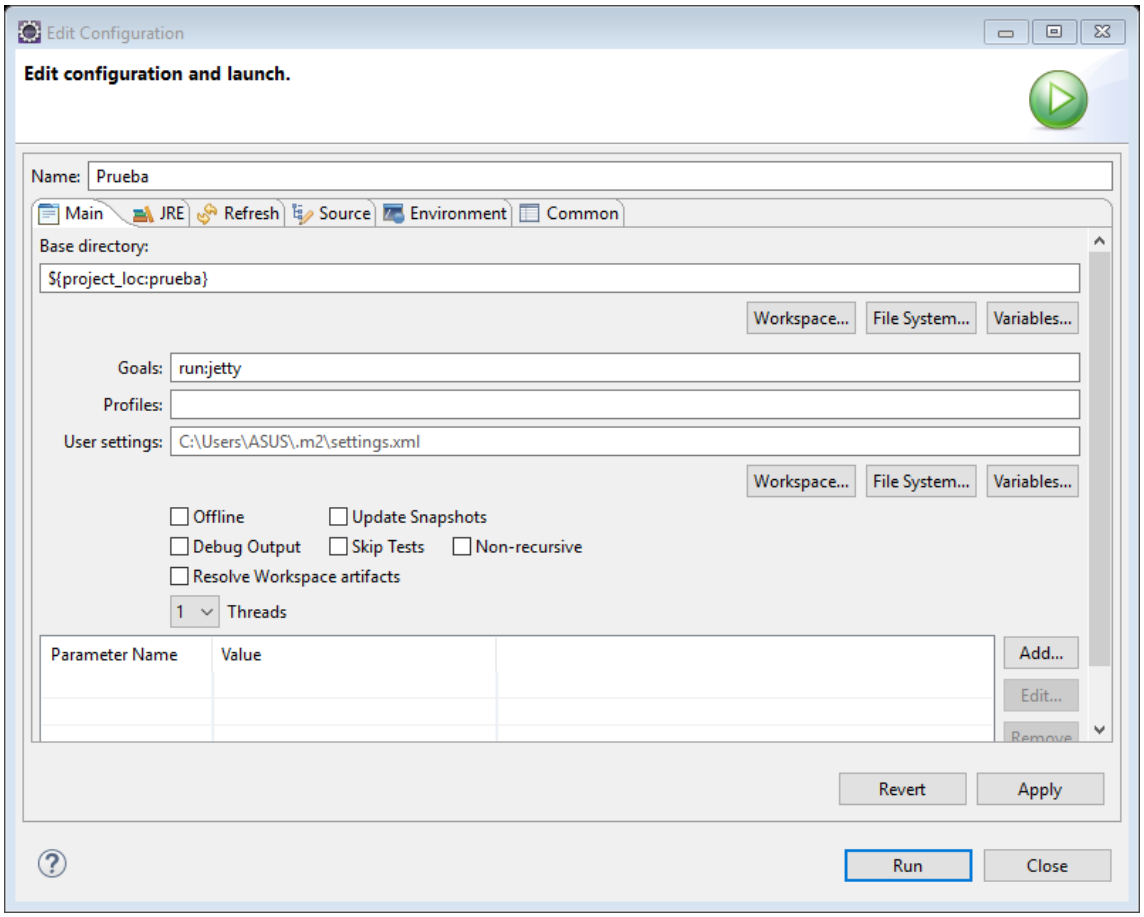
En el momento de ejecutarlo, el run no debe de ejecutarse como JavaApplication, sino como Maven Build.



Selección run como Maven Build.

La primera vez que se ejecute el Maven build aparecerá una ventana (Edit configuration and launch) en la cual habrá que escribir en el apartado Goals, “run:jetty”.

t

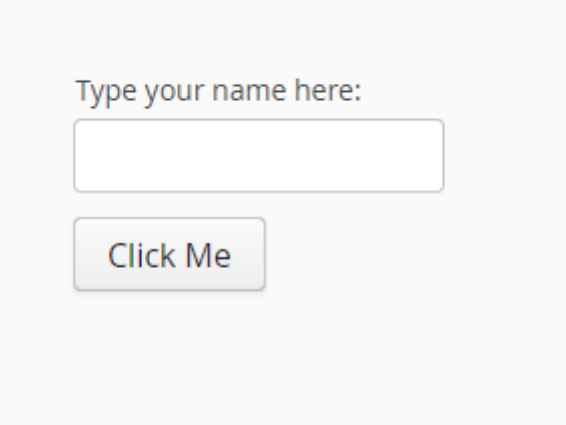


Configuración del lanzamiento.

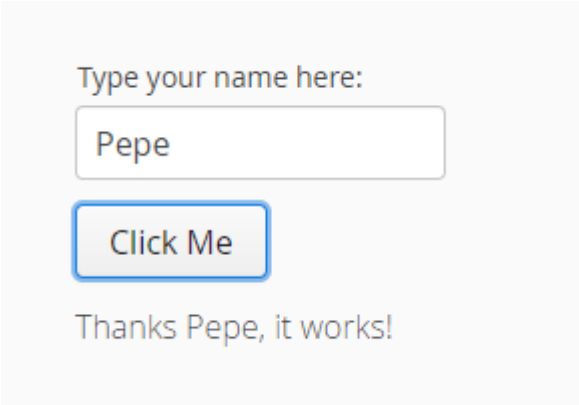
Por defecto, el PC trabajará en el puerto 8080, pero se podría dar el caso de que ese puerto esté ocupado y no permita ejecutar el servidor. En caso de estar ocupado cuando se ejecute el servidor saltará un error Failure: *Address already in use*. Si esto ocurre hay que modificar el código del pom.xml del proyecto para que el servidor se ejecute en el puerto especificado, 8090 o 8070 por ejemplo. Entre la línea 124 y 125 del pom.xml se deberá escribir:

```
<httpConnector>
    <!--host>localhost</host-->
    <port>8090</port>
</httpConnector>
```

Una vez el servidor se está ejecutando, hay que abrir el navegador y escribir localhost:8080. Aparecerá en programa creado por defecto ejecutándose.

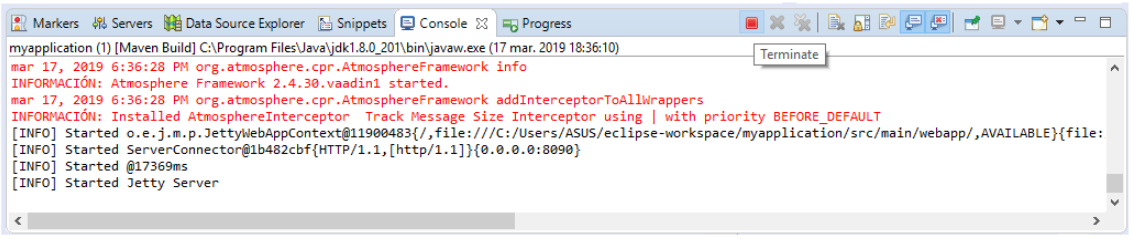


Programa inicial ejecutado en el servidor.



Programa final ejecutado en el servidor.

Importante siempre antes de cerrar Eclipse, Terminar la ejecución del programa, para que no se quede ejecutándose.



Fin de la ejecución del servidor.

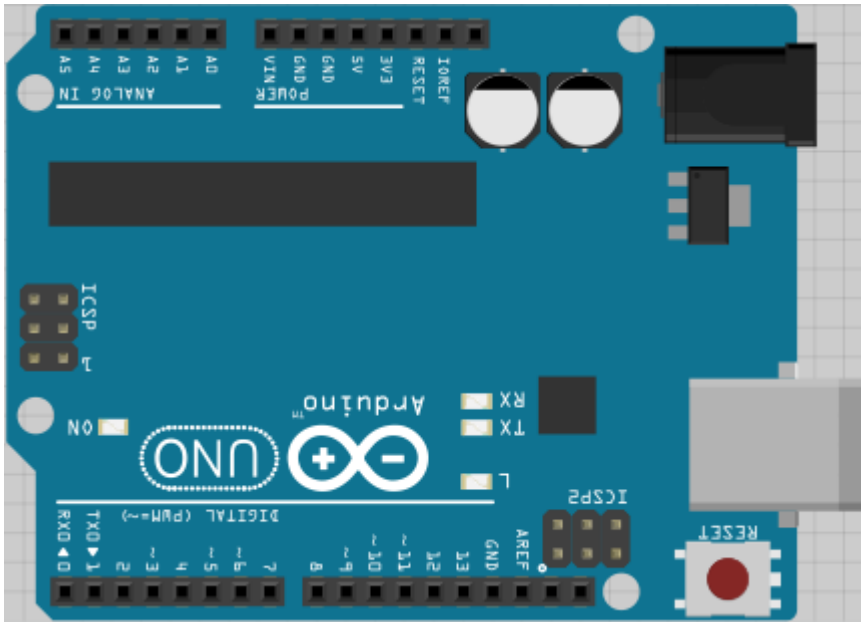
FUNCIONAMIENTO DEL ARDUINO

1. Montaje Arduino

El kit contiene, aparte del arduino y la placa protoboard, varios componentes. Los principales, los que más se utilizarán, son las resistencias, los interruptores, los leds y los cables para los puentes. También contiene potenciómetros, sensores y varios componentes más con los que se podrían hacer más programas.

Aquí se explica el funcionamiento y montaje de los componentes a utilizar:

1.1. Arduino



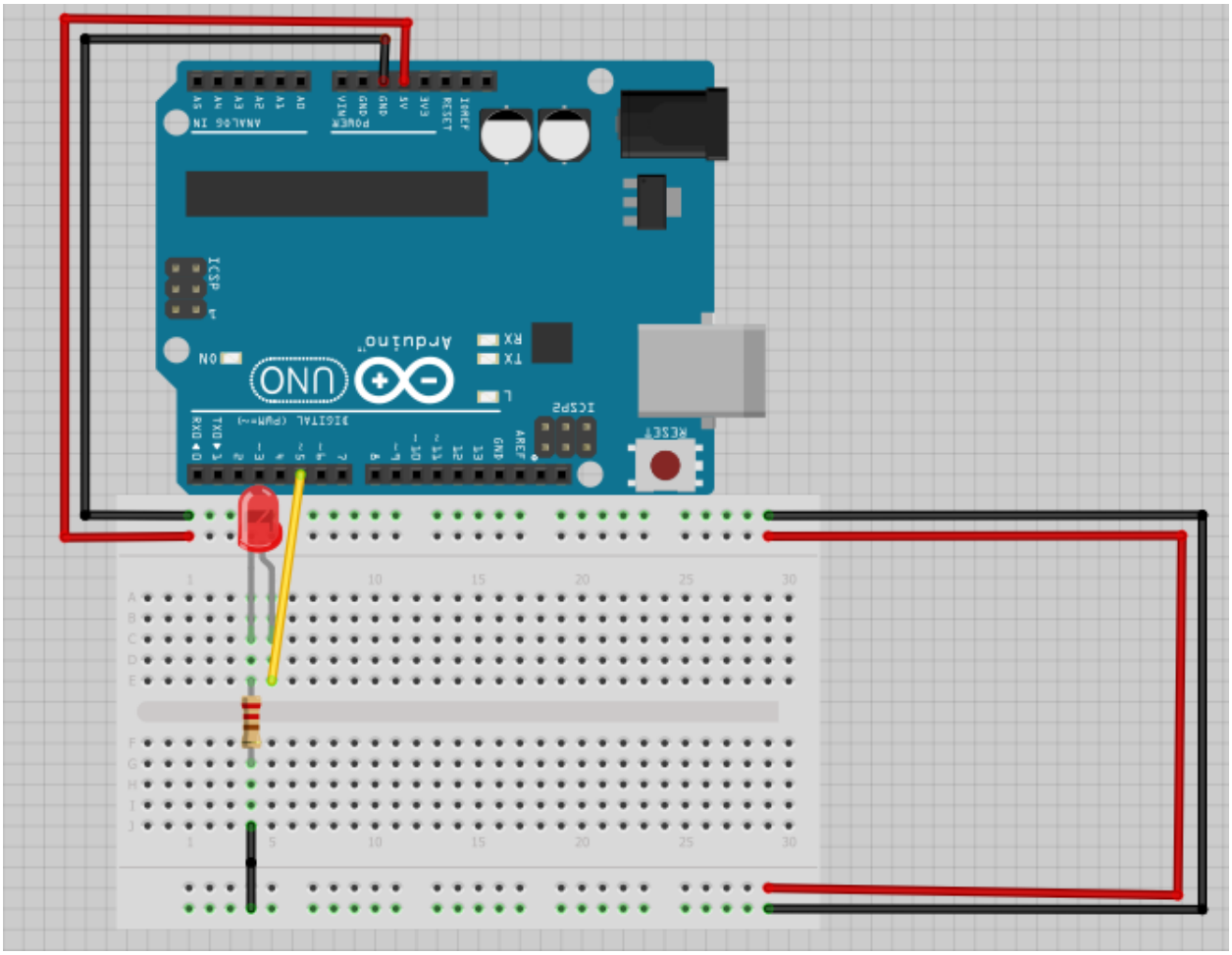
Bornes del Arduino.

En la parte superior izquierda se pueden ver las seis entradas o salidas analógicas. En la parte superior central se pueden ver ocho bornes, de los cuales se utilizarán o bien los 3.3v o 5v y tierra (GND). En la parte inferior se pueden ver las catorce salidas o entradas digitales y algún borne más de los cuales solo se utilizarán los catorce pines digitales

1.2. Led

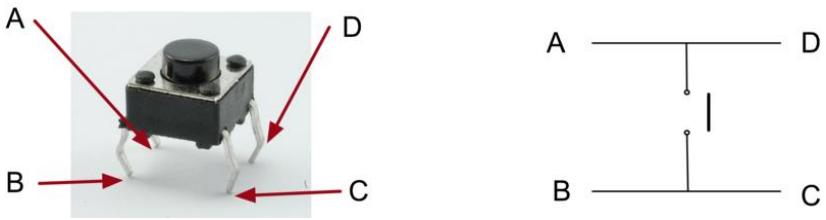
Led de Arduino.

Siempre la pata más corta del led es la negativa y la pata más larga la positiva. Siempre tendrán que ir acompañados con una resistencia, de manera que la caída de tensión se divida y no caiga toda por el led.



Demostración de conexión de un led.

1.3. Interruptores

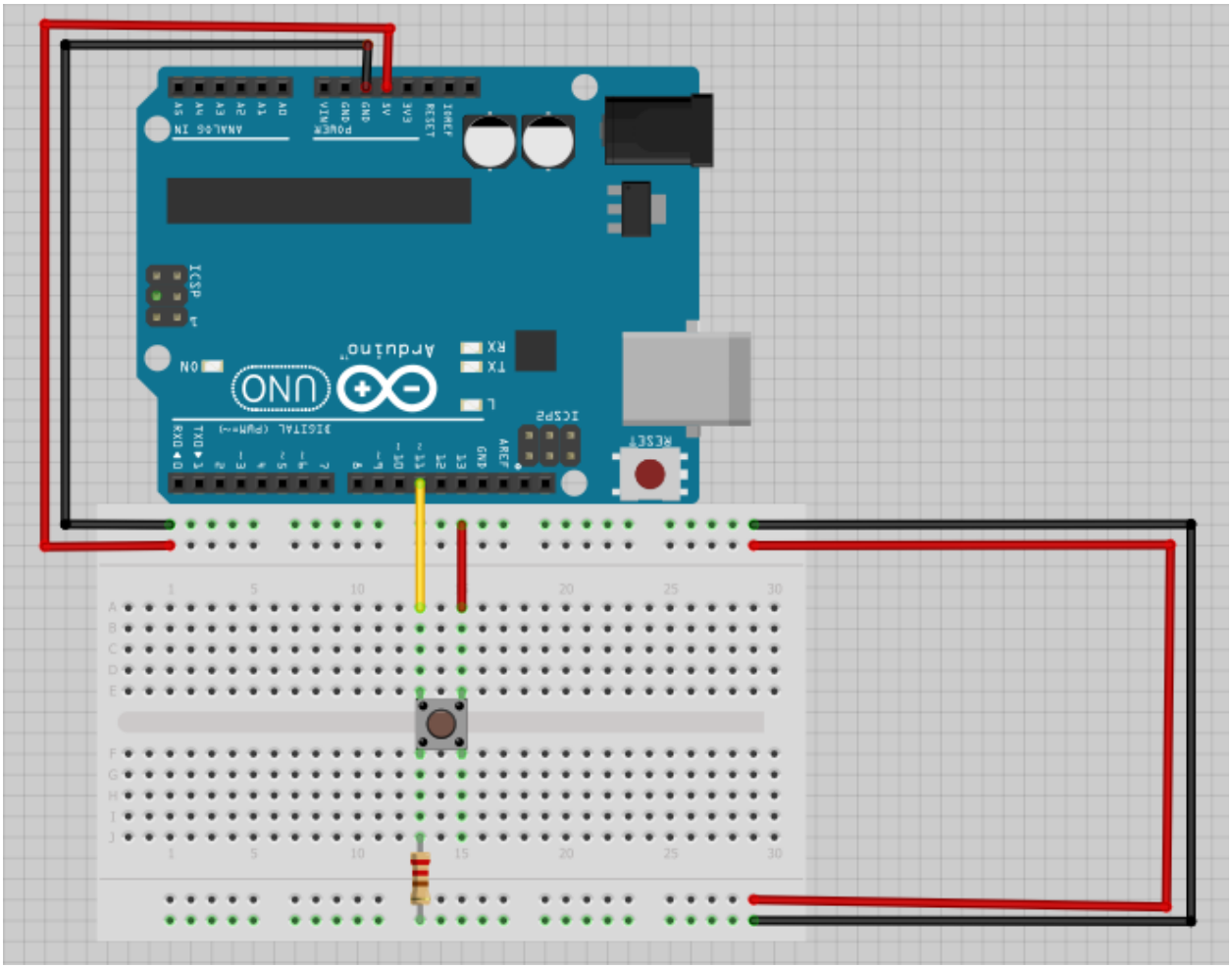


Interruptor pulsador.

Permiten el paso de corriente al ser pulsados, una vez se deja de pulsar, deja de pasar corriente.

Es importante saber que los puntos A y D están conectados internamente y los puntos B y C también.

Los interruptores deberán llevar siempre una resistencia entre la salida del interruptor y 0 V, de manera que cuando no esté el interruptor pulsado en la salida haya 0 V.



Demostración de conexión de un interruptor pulsador.

1.4. Resistencias

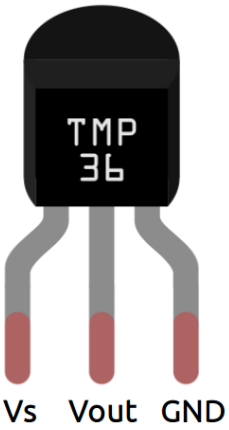
Resistencia 220 Ohmios.

Las resistencias no tienen polaridad, así que no importa la orientación.

1.5. Sensor de temperatura

Sensor de temperatura TMP36.

En la práctica se utilizará un sensor de temperatura TMP36, es un sensor de precisión y de bajos voltajes. Tiene tres pines, uno de 5V, otro de 0V y la salida de la tensión. Es importante la colocación de estos, ya que una mala colocación puede dañar el sensor y dejarlo inutilizable. El sensor tiene una parte plana y otra cilíndrica. Colocaremos el sensor con la cara plana mirando hacia nosotros con los pines hacia abajo y la parte cilíndrica hacia detrás. Una vez tenemos el sensor con la cara plana hacia nosotros, el pin que quede a la izquierda de los tres será el pin donde conectaremos los 5V; el pin que quede a la derecha de los tres será la tierra, es decir, 0V; y por último, el pin central será la salida del sensor. La salida variará en función de la temperatura que esté leyendo el sensor. Ese pin lo conectaremos a una de las entradas analógicas para leerlas desde el PC.



Indicación correcta de los pines del sensor donde Vs serán los 5V, GND los 0V y Vout la salida.

El valor de la temperatura que lee el Arduino no es el real. El valor que se lee deberá ser escalado entre los 5V máximos que puede leer y los 1024 pulsos que puede leer a máxima tensión. Por lo tanto, para obtener el valor de la tensión real será necesario hacer una regla de 3, multiplicar el valor leído por 5 y dividirlo entre 1023. Por las especificaciones del sensor, se sabe que 10 mV representan 1 grado centígrado real, por lo tanto hay que convertir el valor obtenido en temperatura real. Para ello habrá que volver a hacer otra regla de 3. Se multiplicará el valor por 1000 y se dividirá entre 10, es decir, multiplicar por 1000. De esta manera se obtendría la temperatura real en grados centígrados. Se puede simplificar la fórmula:

2. Código Arduino

2.1. Lenguaje básico

El código de programación de Arduino es muy sencillo, consiste principalmente en dos métodos obligatorios:

void setup(): el cual solo se ejecutará una vez
void loop(): el cual se ejecutará en bucle infinito

Se puede declarar variables tanto antes de la declaración de los métodos como dentro de ellos.

La placa tiene 14 pines digitales que se pueden utilizar como entradas o salidas (el número de pin se declara como un número del 0 al 13) y 6 pines analógicos que se pueden utilizar como entradas o salidas (el número de pin se declara como A0, A1, A2, A3, A4 y A5).

Por parte de las salidas y entradas digitales, primeramente se deben declarar en el *setup()*, es decir, definirlas como entradas o salidas con la función *pinMode*(número de pin, OUTPUT o INPUT).

Por ejemplo, queremos configurar el pin 1 digital como salida y el pin 2 digital como entrada. Pues debemos declarar:

```
void setup() {
        pinMode(1,OUTPUT);              // Se declara el pin 1 como entrada
        pinMode(2,INPUT);               // Se declara el pin 2 como salida
 }
```

Estos pines en el lenguaje de Arduino pueden tener dos valores HIGH(1), para cuando se lea tensión en la entrada o cuando se quiera dar tensión en la salida; o LOW (0), para cuando no se lea tensión en la entrada o cuando se quiera no dar tensión en la salida. Una vez declaradas ya se puede leer o escribir en ellas.

Para leer la señal utilizaremos la función digitalRead (número de pin). Y para darle valor utilizaremos la función digitalWrite (número de pin).

```
If (digitalRead(1)==HIGH) {             // Se lee el valor del pin 1 y se compara
}
    digitalWrite(2,HIGH);               // Se da tensión al pin 2
```

Aquí se puede ver un ejemplo en el cual se leerá el valor del pin 1 y en función de si hay tensión o no, daremos tensión al pin 2 o no.

```
void setup() {
  pinMode(1,OUTPUT);        // Se declara el pin 1 como entrada
            pinMode(2,INPUT);           // Se declara el pin 2 como salida
}

void loop() {
 if(digitalRead(1)==HIGH){  // Se lee el valor del pin 1, y si hay tensión
                            // se cumple la condición, por lo tanto se ejecutará
                                // lo que haya dentro del condicional If
       digitalWrite(2,HIGH); // Se da tensión al pin 2
 }
}
```

Por otro lado, las entradas y salidas analógicas no se tienen que declarar previamente. Las funciones para leer y escribir son *analogWrite* (número de pin, valor) y *analogRead* (número de pin).

Cuando se lea la tensión de un pin analógico se leerá un valor entre 0 y 1023. Por lo tanto si se quiere saber el valor exacto de la tensión, el valor que se lea se tiene que multiplicar por 5 y dividir entre 1023, de esta manera se habrá calculado el valor exacto de tensión que hay en el pin analógico 0. Y en el momento que se quiera dar una tensión a un pin analógico el valor habrá que darlo en un rango de 0 a 255, cuando se escriba un 255 se estará escribiendo 5 V en la salida analógica.

```
"variable int" = analogRead(A0);
// Se lee el valor del pin analógico 0 y se guarda en una // variable


analogWrite(A1,"variable int");
// Se da un valor de tensión al pin analógico 1
```

Aquí se puede ver un ejemplo en el cual se leerá el valor del pin analógico 1 y se escribirá el mismo en el pin analógico 2.

```
void setup() {
        int val =0;                    // Se declara la variable integer "var"
}


void loop() {
        val = analogRead(A1) * 5 / 1023;      // Se guarda el valor de la tensión
        // real de la tensión leída en el pin analógico 1
analogWrite(A2, var * 255 / 5);       // Se escribe en el pin analógico 2 la misma
                                      // tensión que se ha leído en el pin 1
}
```

2.2. Lenguaje avanzado

En este apartado se va a explicar cómo comunicarse con el puerto serie de nuestro PC.

Para abrir el puerto basta con iniciarlo desde el *setup(),* la función Serial.begin(9600).

De esta manera se abre la comunicación del puerto para la posterior utilización. Una vez establecida la comunicación se permite leer y escribir a través de ese puerto. Cuando se escriba en el puerto se mandará un dato de manera que desde otra aplicación (eclipse por ejemplo) se pueda leer lo que haya en el puerto. De la misma manera se lee desde el puerto, desde otra aplicación se escribirá en el puerto de manera que desde el código leamos lo que haya en el puerto.

Escribir en el puerto se puede hacer en cualquier momento con la instrucción *Serial.write(char),* siempre escribiendo un carácter de la tabla ASCII; pero leer desde el puerto hay que hacerlo en todo momento, por lo tanto las instrucciones tendrán que estar en el método void loop(), y la función para leer es *Serial.read().*

Por lo tanto, dentro del método *loop(),* se deberá estar leyendo constantemente de manera que cuando haya algo en el puerto serie se lea, se hace de la siguiente manera:

```
if(Serial.available()>0){  //Se mira si hay algo que leer en el puerto, en caso que si
                           // se lee lo que haya en el puerto y se guarda en una
        byte dato=Serial.read();              // variable tipo Byte
  }
```

Una vez se ha guardado el valor del carácter que se ha leído se puede trabajar con él, por ejemplo comparando con otro valor fijo de manera que cuando ese valor sea el mismo que el prefijado hacer cualquier acción.

```
  void loop() {
        if(Serial.available()>0){
        byte dato=Serial.read();
     if(dato==69){
        digitalWrite(1,HIGH);
              }
        }
}
```

//Cuando se lea un 69 en el puerto serie, se dará tensión a la salida analógica 1.

CONCEPTOS BÁSICOS

1. Práctica 0 Processing: Introducción al entorno Processing

1.1. ¿De qué trata?

Esta práctica consiste en explicar cómo descargar las librerías de Processing, cómo importarlas a un proyecto java de eclipse y cómo implementar la Applet para poder utilizar las funciones de Processing.

Se hace un programa muy simple mostrando un texto por una pantalla. Se dan a conocer los métodos esenciales de Processing y unas pocas funciones.

1.2. Conceptos teóricos

1.2.1. Creación de un proyecto importando las librerías de Processing.

Utilización de los métodos principales de Processing.

1.3. Clases

No se utilizan clases no explicadas anteriormente

1.4. Métodos:

1.4.1. void settings()

Este método sirve para declarar e inicializar variables y para dar tamaño a la ventana. Solo puede contener el método *size().* Se ejecutará solamente una vez.

1.4.2. void setup()

Este método también se ejecutará solamente una vez. Podrá contener todo tipo de funciones.

1.4.3. size

Este método da el tamaño a la ventana.

1.4.4. Background

Este método da color al fondo de la ventana.

1.1.1. fill()

Este método da el color al relleno de las posteriores funciones que se llamen, como el color de la letra, el color de una línea o el color del relleno de las formas.

1.1.2. text()

Este método muestra un el texto deseado en las coordenadas deseadas.

2. Práctica 1 Processing: Primeros programas

2.1. ¿De qué trata?

Esta práctica se constituye de 3 apartados.

El primero consiste en utilizar figuras geométricas de dos dimensiones para dibujar una cara sonriente. El segundo en dar un movimiento infinito a una línea. Y el tercero, en mostrar una imagen en la ventana y desplazarla.

2.2. Conceptos teóricos

2.2.1. Utilización de eventos con el teclado y el ratón.

2.2.2. Dar movimiento a objetos utilizando el método draw().

2.2.3. Introducción de la clase PImage.

2.3. Clases:

2.3.1. PImage

Esta clase se utiliza para mostrar imágenes por pantalla y trabajar con ellas.

2.4. Métodos:

2.4.1. void draw()

Este método se ejecutará en bucle. Podrá contener todo tipo de funciones.

2.4.2. mousePressed()

Este es un evento el cual se ejecutará siempre que el usuario haga un clic izquierdo con el ratón. Podrá contener todo tipo de funciones.

2.4.3. keyPressed()

Este es un evento el cual se ejecutará siempre que el usuario pulse una tecla. Una vez dentro de ese programa se podrá comprobar que tecla se ha pulsado.

2.4.3.1. Key – Esta variable char será la tecla pulsada

2.4.4. rectangle()

Este método dibuja un rectángulo en la posición deseada y con las medidas deseadas

2.4.5. triangle()

Este método dibuja un triángulo en la posición deseada y con las medidas deseadas

2.4.6. elipse()

Este método dibuja una elipse en la posición deseada y con las medidas deseadas

2.4.7. line()

Este método dibuja una línea entre dos coordenadas

2.4.8. stroke()

Este método da el color del contorno a las posteriores funciones que se llamen, como el color de la letra, el color de una línea o el color del contorno de las formas.

2.5. Métodos de PImage:

2.5.1. loadimage(string “nombre de la imagen”)

Este método carga la imagen con el nombre escrito en una variable PImagen

2.6. image()

Este método muestra por pantalla la imagen previamente guardad en las coordenadas deseadas.

3. Práctica 2 Processing: Paint

3.1. ¿De qué trata?

En esta práctica se creará un Paint. Primeramente permitiendo dibujar con el ratón; seguidamente permitiendo hacerlo con diferentes espesores; y por último, permitir borrar.

3.2. Conceptos teóricos:

3.2.1. Utilizar más eventos como MouseDragged

3.2.2. Distribución de pantalla

3.3. Clases:

No se utilizan clases no explicadas anteriormente

3.4. Métodos:

3.4.1. mouseDragged()

Este es un evento el cual se ejecutará siempre que el usuario haga un clic izquierdo con el ratón y mueva el ratón. Podrá contener todo tipo de funciones.

3.4.1.1. mouseX – Esta variable integer devuelve la coordenada X de la pantalla en la cual está el puntero del ratón

3.4.1.2. mouseY – Esta variable integer devuelve la coordenada Y de la pantalla en la cual está el puntero del ratón

3.4.1.3. pmouseX – Esta variable integer devuelve la coordenada X de la pantalla en la cual estaba el puntero del ratón antes de la coordenada actual

3.4.1.4. pmouseY – Esta variable integer devuelve la coordenada Y de la pantalla en la cual estaba el puntero del ratón antes de la coordenada actual

3.4.2. strokeWeight()

Este método da el grosor del contorno a las posteriores funciones que se llamen, como el grosor de la letra, el grosor de una línea o el grosor del contorno de las formas.

4. Práctica 3 Processing: Pelota saltarina

4.1. ¿De qué trata?

En esta práctica se creará una esfera que irá desplazándose por la ventana de manera que rebote cuando choque con las paredes cambiando de dirección.

Posteriormente, se añadirá otra esfera de manera que cuando se crucen entre ellas cambien de color aleatoriamente.

4.2. Conceptos teóricos

4.2.1. Introducción de la clase PVector

4.2.2. Conceptos teóricos de vectores como magnitud y operaciones entre ellos.

4.2.3. Librería Maths de java para utilización de método random.

4.3. Clases:

4.3.1. PVector

Es una clase que describe un vector, tanto en 2 como en 3 dimensiones. Incluye muchas funciones para trabajar con los vectores, como cambios de dirección, rotaciones o cálculos de magnitud. Siempre que se vaya a mover algún objeto por pantalla de forma

libre, es decir, en cualquier dirección, la clase PVector te facilita los movimientos y direcciones de este.

4.4 Métodos de PVector:

4.4.1 sub()

Este método substrae o “resta” componentes a un vector, suma un vector a otro o suma dos vectores independientes.

4.4.2 add()

Este método añade o “suma” componentes a un vector, suma un vector a otro o suma dos vectores independientes.

4.4.3 mag()

Este método devuelve la magnitud del vector un vector a otro o suma dos vectores independientes.

5. Práctica 4 Processing: Encuentra los objetos

5.1. ¿De qué trata?

Esta práctica consiste en crear un juego en el cual aparezcan una imagen con varios elementos, aleatoriamente se elijan 5 de ellos y se muestre el nombre de estos. El jugador deberá localizar donde se encuentran y seleccionarlos, si es correcto, eliminará ese objeto de la lista. Una vez se complete una imagen, deberá saltar a otra y hacer lo mismo. Una vez se hayan encontrado todos los objetos de todas las imágenes el jugador habrá ganado la partida. Por otro lado, si el jugador falla un número de veces determinado, el jugador habrá perdido.

Todo esto es la segunda parte de la práctica, que es lo de que consta el juego; pero primeramente se deberá crear un programa sencillo el cual sirva para determinar las coordenadas de esos objetos. Se mostrarán las imágenes por la pantalla y a cada clic en la pantalla, la consola mostrará las coordenadas para posteriormente poder declarar los elementos.

5.2. Conceptos teóricos:

5.2.1. Conocer arrays y trabajar con ellos.

5.3. Clases:

No se utilizan clases no explicadas anteriormente

5.4. Métodos:

No se utilizan métodos no explicados anteriormente

6. Práctica 5 Processing: Flappy bird

6.1. ¿De qué trata?

Esta práctica consiste en crear un juego en el cual una línea con un hueco en medio vaya desplazándose verticalmente de abajo a arriba y el jugador mueva con el ratón una pequeña imagen de manera que intente pasar entre medio del hueco. Cada vez que la línea pase de la parte superior de la ventana a la inferior se deberá de cambiar aleatoriamente la posición del hueco, y cada un número determinado de veces se deberá aumentar la velocidad de desplazamiento de la línea de manera que cada vez sea más difícil mantener nuestra imagen dentro del hueco.

6.2. Conceptos teóricos

6.2.1. Reforzar lo aprendido hasta ahora.

6.3. Clases:

No se utilizan clases no explicadas anteriormente.

6.4. Métodos:

No se utilizan métodos no explicados anteriormente.

7. Práctica 0 Arduino: Primer programa

7.1. ¿De qué trata?

Esta práctica consiste en tomar contacto con el Arduino. Es una práctica muy sencilla en el cual mediante una entrada y una salida digital se encenderá un led siempre que se pulse un interruptor.

7.2. Conceptos teóricos:

7.2.1. Toma de contacto con el Arduino

7.2.2. Conocer y utilizar correctamente las entradas y salidas digitales de la placa Arduino

8. Práctica 6 Processing: Puzzle

8.1. ¿De qué trata?

Esta práctica de que construir con un puzle moviendo las piezas con el Arduino. Se comunicará a través del puerto serie y se leerán se escribirán caracteres a través de este.

Primeramente se creará un programa el cual divida una imagen. Siempre se dividirá en un número de piezas que sea potencia de otro, es decir 4, 9 ,16 ,25… para que se pueda construir un puzle 2x2, 3x3, 4x4, 5x5… El programa deberá crear una ventana con el tamaño deseado de las piezas, si se ha elegido hacer un puzle de 2x2, habrá que crear una ventana con un tamaño de la mitad de la imagen completa. De esta manera se colocará la imagen en unas coordenadas determinadas para que en la pequeña ventana se muestre el trozo de imagen que se quiere obtener como pieza. Y una vez se haya mostrado el trozo de imagen que se desea, se salvará y se creará una pieza del puzle. De esta manera jugando con la posición de la imagen, se obtendrán todas las piezas.

Una vez ya se tiene todas las piezas creadas.se creará un programa que cree el puzle, es decir, que muestre todas las piezas en la parte inferior de la pantalla y un gran rectángulo del tamaño de la imagen completa con las divisiones dentro de él para la colocación de las piezas.

Mediante el ratón se seleccionará la pieza que se desea mover y mediante el teclado se moverá hasta su posición correcta. Y para acabar, se deberá crear un último programa que mueva las piezas con el Arduino en vez de con el teclado.

8.2. Conceptos teóricos:

8.2.1. Introducción de la comunicación Serie a través del puerto del PC.

8.2.2. Leer datos desde el Arduino.

8.2.3. Conocer y utilizar correctamente las entradas y salidas digitales de la placa Arduino

8.3. Clases:

8.3.1. Serial

Esta clase permite leer y escribir bytes de uno en uno a través del puerto serie. Se ha de inicializar el puerto en el método setup().

```
miPuerto=new Serial(this, Serial.list()[2],9600)
```

8.4. Métodos:

8.4.1. save()

Este método guarda lo que se vea en pantalla en el momento de llamar al programa, lo guarda como una imagen del tamaño que tenga la pantalla.

8.5. Métodos de la clase Serial:

8.5.1. avaliable()

Este método devuelve el número de bytes que hay esperando en memoria para leer. Si devuelve un 0 es que no hay ningún byte esperando.

8.5.2. read()

Este método lee el byte que hay en la memoria.

9. Práctica 7 Processing: Domótica

9.1. ¿De qué trata?

Esta práctica consiste en simular la domótica de una casa. Tendrá dos apartados, el aire acondicionado del piso y las luces de las habitaciones. Se creará una especie de SCADA en el cual se vea una simulación de un mando de aire acondicionado y una simulación de un piso con 5 habitaciones.

El primero apartado consta de la simulación del aire acondicionado de la casa. Se utilizará el sensor TMP36 para leer la temperatura, se leerá a través de las entradas analógicas del Arduino, y se enviará el dato corregido al programa de Processing, el cual se comparará con valores prefijados. Si supera el límite de temperatura máxima se activará el aire y cuando sea inferior a la temperatura mínima se desactivará. En el mando del aire acondicionado se visualizaran las temperaturas prefijadas, la temperatura en ese momento y un indicador led indicando que el aire está activado.

Y para el segundo apartado, se habrá dibujado la simulación de un piso con 5 habitaciones, pues bien, se creará una imagen o icono simulando una persona de manera que cada vez que entre en una habitación se encienda la luz de esta. Cada vez que la persona entre en la habitación se enviará un dato al Arduino de manera que detecte que alguien ha entrado en la habitación y encenderá un led, habrá uno por cada habitación.

9.2. Conceptos teóricos

9.2.1. Enviar datos al Arduino

9.3. Clases:

No se utilizan clases no explicadas anteriormente

9.4. Métodos:

No se utilizan métodos no explicados anteriormente.

9.5. Métodos de la clase Serial:

9.5.1. write()

Este método escribe el byte deseado para que el Arduino lo lea.

10. Práctica 8 Vaadin: Ejecución diseño Vaadin

10.1. ¿De qué trata?

Esta práctica consiste en aprender como diseñar una clase de Vaadin y ejecutar la clase Test que ejecute el diseño creado. Se crearán varios textFields, en dos de ellos no se podrá de escribir, y en los otros dos si, lo que se escriba en esos dos se visualizarán en los otros dos simulando una “conversación”.

10.2. Conceptos teóricos:

10.2.1. Conocer los diferentes tipos de layout que tiene Vaadin.
10.2.2. Seguir los pasos para ejecutar un diseño creado.
10.2.3. Conocer diferentes tipos de Vaadin
10.2.4. Conocer diferentes eventos de los componentes de Vaadin

10.3. Componentes:

10.3.1. TextField

Este componente es en un campo de texto en la cual se puede escribir.

10.4. Métodos:

10.4.1. setContent()

Este método muestra la clase declarada en la pantalla.

10.4.2. setEnable()

Este método activa o desactiva el componente, aparece en la mayoría de ellos. En caso de un TextField por ejemplo, permitirá o no escribir en él.

10.4.3. setValue()

Este es método que solo tienen unos componentes determinados, los más utilizados son los TextFields o los TextArea. Cambia el valor de estos.

10.4.4. getValue()

Este es método que solo tienen unos componentes determinados, los más utilizados son los TextFieldso los TextArea. Devuelve el valor de estos en el momento de la llamada al programa.

10.5. Eventos:

10.5.1. addValueChangeListener()

Este es un evento el cual se ejecutará siempre que el *TextField* se modifique, es decir, siempre que se añada o se elimine algún carácter.

11. Práctica 9 Vaadin: Formulación

11.1. ¿De qué trata?

Esta práctica consiste hacer un formulario para una página web, que la web te pida varios campos como nombre, nacionalidad o edad. Se deberá diseñar un a clase con Vaadin Designer en al cual se rellenen todos los campos. En el siguiente apartado se recogerán esos datos y se rellenará un *TextArea* con esos datos haciendo una pequeña descripción. Y finalmente se dará validez a esos datos antes de hacer esa descripción. Por ejemplo nombre y apellidos deberán tener un mínimo de tres caracteres o la fecha de nacimiento debe ser anterior al día actual.

11.2. Conceptos teóricos:

11.2.1. Conocer diferentes componentes que contiene Vaadin.

11.2.2. Conocer los diferentes métodos que contienen esos componentes.

11.2.3. Diseñar un formulario con sus respectivos campos como de una página web se tratase.

11.2.4. Validación de datos.

11.3. Componentes:

11.3.1. ComboBox

Este componente es en una lista desplegable donde aparecen diferentes opciones de las cuales se debe elegir una, también tiene aplicada la búsqueda de datos, es decir, puedes filtrar la lista escribiendo la opción que quieras seleccionar.

11.3.2. RadioButton

Este componente es en mostrar diferentes opciones a seleccionar de las cuales se deberá seleccionar una. A diferencia del *ComboBox*, aquí aparecerán todas las opciones directamente, sin ningún desplegable.

11.3.3. TextArea

Este componente es en una versión extendida de un *TextField*, ya que simula varios TextFields juntos, un gran campo de texto donde puedes escribir en diferentes líneas.

11.3.4. InlineDateField

Este componente es un calendario en el cual se puede elegir un día en él.

11.3.5. Button

Este componente es un simple botón el cual contendrá sus eventos para cada vez que se actúe sobre él realizar alguna acción.

11.3.6. Label

Este componente es una etiqueta que muestra un texto no editable, se utiliza para dar información.

11.4. Métodos:

11.4.1. clear().

Este método limpia cualquier campo, ya sean *radioButton*, *TextFields* o *TextAreas*.

11.4.2. isAftet()

Este es un método que devuelve si la una fecha es anterior o no a otra. Se utiliza con variables de tipo *LocalDate*.

11.5. Eventos:

11.5.1. addClickListener ()

Este es un evento el cual se ejecutará siempre que se cique sobre el *Button*.

12. Práctica 10 Vaadin: Noticias

12.1. ¿De qué trata?

Esta práctica consiste hacer un formulario para una página web, que la web te pida varios campos como nombre, nacionalidad o edad. Se deberá diseñar un a clase con Vaadin Designer en al cual se rellenen todos los campos. En el siguiente apartado se recogerán esos datos y se rellenará un *TextArea* con esos datos haciendo una pequeña descripción. Y finalmente se dará validez a esos datos antes de hacer esa descripción. Por ejemplo nombre y apellidos deberán tener un mínimo de tres caracteres o la fecha de nacimiento debe ser anterior al día actual.

12.2. Conceptos teóricos:

12.2.1. Añadir componentes de un layout.

12.2.2. Eliminar componentes de un layout.

12.3. Componentes:

No se utilizan componentes no explicados anteriormente

12.4. Métodos:

12.4.1. setMaxLength()

Este método limita el número de caracteres de un *TextField* o un *TextArea*. Los campos no permitirán escribir más caracteres una vez se ha llegado a ese número de caracteres.

12.4.2. getComponentCount()

Este método devuelve la cantidad de componentes que contiene un layout.

12.4.3. removeComponent()

Este método elimina el componente indicado de un layout.

12.4.4. addComponent()

Este método añade el componente indicado a un layout.

12.5. Eventos:

12.5.1. addClickListener ()

Este es un evento el cual se ejecutará siempre que se cique sobre el *Button*.

13. Práctica 11 Vaadin: Registro

13.1. ¿De qué trata?

Esta práctica consiste en crear un registro de usuarios con algunos datos para posteriormente iniciar sesión y ver y modificar estos datos. A la hora de registrarse se deberá escribir un usuario y una contraseña de manera que solo se pueda iniciar sesión introduciendo los datos correctamente. Deberán guardarse todos los usuarios con sus datos de manera que cada vez que se abra la aplicación se siga teniendo los usuarios guardados.

13.2. Conceptos teóricos:

13.2.1. Leer archivos de texto para cargar datos.

13.2.2. Crear un archivo de texto para almacenar datos.

13.2.3. Saltar de un diseño a otro.

13.3. Componentes:

No se utilizan componentes no explicados anteriormente

13.4. Métodos:

13.4.1. setVisible()

Este método permite visualizar o no el componente.

13.5. Eventos:

No se utilizan eventos no explicados anteriormente.

ANEXO D: AUTOINFORME DE CALIDAD

PARÀMETRE A VERIFICAR: ASPECTES FORMALS - CONTINENT	RESULTATS			
	1	2	3	COMENTARIS
A1 - Formats portades			X	
A2 - Sumari de continguts			X	
A3 - Sumari de taules i figures			X	
A4 - Ortografia / Unitats		X		Se ha revisado el documento, aunque siempre puede haber alguna falta de ortografía o unidades inadecuadas que no se hayan tenido en cuenta.
A5 - Taules / Gràfics			X	
A6 - Formats dels documents			X	
A7 - Extensió de la memòria			X	
A8 - Bibliografia			X	
A9 - Relació de documents			X	
PARÀMETRE A VERIFICAR: ASPECTES FORMALS - CONTINGUT	RESULTATS			
	1	2	3	COMENTARIS
B1 - Plantejament problema			X	
B2 - Antecedents i estat de l'art			X	
B3 - Plantejament i justificació solucions			X	
B4 - Acompliment abast i especificacions			X	
B5 - Aspectes econòmics, ambientals i seguretat			X	No proceden aspectos ambientales ni de seguridad
B6 - Aspectes temporals			X	
B7 - Conclusions i recomanacions			X	