Universidad Politécnica de Cartagena

Manual de prácticas

FUNDAMENTOS DE PROGRAMACIÓN

Grado en Ingeniería en Sistemas de Telecomunicación
Grado en Ingeniería Telemática

Área LSI

Curso 2021-2022

Práctica 1: Introducción al entorno de programación Eclipse.

Objetivos:

- Presentar las herramientas que hacen falta para desarrollar programas.
- Aprender a editar, compilar y ejecutar programas en Java con el entorno Eclipse.

La estructura de esta práctica difiere bastante del resto, ya que no se trata de poner en práctica ningún conocimiento teórico, sino de presentar el entorno en el que se van a realizar las prácticas y poner este entorno en el contexto más general del proceso de desarrollo de software. No obstante, al final del boletín se incluye un pequeño ejercicio.

El proceso de edición, compilación y ejecución de programas.

Programar es algo más que *codificar* programas en un lenguaje de programación. En la figura 1.1 se resumen los pasos típicos que incluye el desarrollo de un programa, desde la definición del problema hasta la obtención de un programa ejecutable correcto. La numeración que aparece en este apartado se refiere a los pasos que se describen en dicha figura. La figura no asume ningún lenguaje de programación en concreto, en apartados posteriores particularizaremos para el lenguaje Java.

Definición del problema (1) y diseño de la solución (2).

Desarrollar un programa, incluso para programas tan sencillos como los que se van a realizar en esta asignatura, es parte de un proceso que empieza por (1) la definición del problema y (2) el diseño de una solución. Estos pasos se abordan *antes* de empezar a codificar. Si el problema es grande y complicado será necesario aplicar técnicas que permitan descomponerlo en problemas más pequeños cuyas soluciones puedan integrarse para resolver el problema global. Pero aunque el problema se reduzca a plantear un pequeño algoritmo, sigue siendo necesario comprender el problema y solucionarlo antes de empezar a codificar la solución[1]. Aunque las soluciones se diseñan pensando en los términos de las abstracciones que nos proporciona el lenguaje de programación, la codificación es un paso posterior al planteamiento de la solución.

Codificación (3)

Una vez que hemos resuelto el problema podemos pasar a codificar la solución, creando un archivo o un conjunto de archivos que contengan el código fuente. Para ello se necesita un programa *editor* que permita escribir y guardar nuestro programa. En las prácticas se va a utilizar el editor que proporciona el entorno de desarrollo Eclipse, pero vale cualquier editor que utilice el conjunto de caracteres que admite Java. Durante la codificación es muy habitual incorporar código disponible en bibliotecas software, que puede haber sido desarrollado por el propio programador o por terceros. Los lenguajes de programación establecen mecanismos para importar y utilizar dichas librerías aunque no estén disponibles los archivos con su código fuente.

Compilación y ensamblado del programa (4)

El programa que hemos editado es código fuente y no puede ser ejecutado directamente por ningún computador: es necesario traducirlo a código máquina. Para ello necesitamos un *compilador* o un *intérprete*[2]. Además, los programas habitualmente no se encuentran en un único archivo, sino distribuidos en muchos, que deben

[1] Cuando el problema está muy acotado, por ejemplo un algoritmo para solucionar un problema muy concreto, la solución puede expresarse directamente en el lenguaje de programación. Esta forma de actuar deja de ser viable en cuanto el problema tiene una entidad que hace necesaria su descomposición en problemas parciales, lo cual ocurre sorprendentemente pronto.
[2] O ambos, como veremos que ocurre en Java.

ensamblarse para dar lugar a un programa ejecutable. La obtención de un programa ejecutable implica, por un lado, la traducción de los archivos en código fuente a archivos con código objeto y, por otro, el ensamblado de los archivos con código objeto.

Ejecución, prueba y depuración del programa (5)

Dependiendo del lenguaje y de la plataforma de ejecución que utilicemos, para ejecutar un programa puede bastar con escribir su nombre como un comando en una consola o hacer doble click en un icono o puede ser necesario utilizar algún programa auxiliar que lance al nuestro. En cualquier caso, que nuestro programa sea capaz de ejecutarse no significa que realice correctamente la tarea que tiene encomendada. En programación es muy difícil acertar a la primera. Normalmente es necesario ejecutar el programa una y otra vez hasta conseguir detectar y corregir todos sus fallos. Existen técnicas para probar los programas de forma exhaustiva y utilidades (*depuradores*) que ayudan a detectar y corregir los errores de programación. Normalmente los lenguajes de alto nivel proporcionan soporte en tiempo de ejecución, es decir realizan ciertas comprobaciones para asegurarse de que el programa no viola ninguna restricción impuesta por el lenguaje. La corrección de un error puede dar lugar a replantearse el problema desde el principio, de ahí que en la figura 1.1 se pueda regresar desde la prueba del programa a cualquier paso anterior.

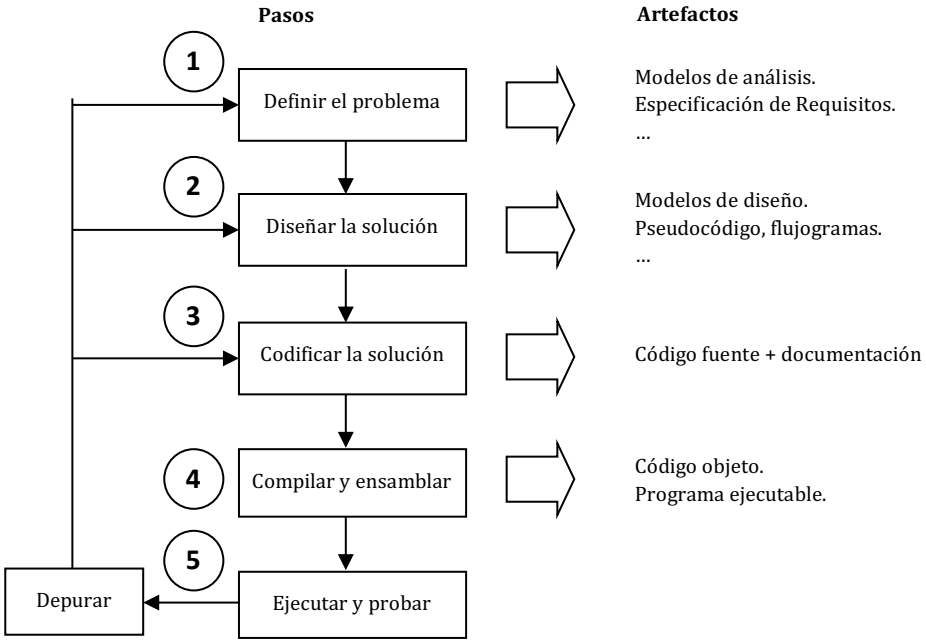

Figura 1.1: Desarrollo de programas.

Los ciclos de la figura 1.1 revelan una característica esencial del proceso de desarrollo software: es un proceso iterativo e incremental que incluye varios ciclos en cada uno de los cuales se resuelven los problemas encontrados en el ciclo anterior, se refina la solución y se incorpora nueva funcionalidad. Cada fase de desarrollo produce sus propios artefactos que sirven para abordar la siguiente fase y que son revisados y actualizados en cada iteración.

Escritura y ejecución de programas en Java. La plataforma Java.[3]

La figura 1.2 muestra el proceso de escritura, compilación, ensamblado y ejecución de un programa en Java. La figura 1.2 es la particularización a partir del punto 3 de la figura 1.1 para el lenguaje Java. En Java todo el código fuente se escribe en ficheros de texto con la extensión `.java`. Los ficheros fuente son compilados por el compilador `javac` a ficheros `.class`. Un fichero `.class` no contiene el código máquina correspondiente a ningún procesador real, sino el código máquina o código de bytes (*bytecodes*) de una *máquina virtual*, la máquina virtual de Java o JVM[4]. Una vez que se dispone de los ficheros `.class`, un programa especial, el lanzador (*launcher*), es capaz de ejecutar el programa sobre una instancia de la JVM. La JVM *interpreta* el código de bytes y genera las instrucciones en código máquina correspondientes al procesador *real* en el que se ejecuta la aplicación.

Figura 1.2: Compilación y ejecución de un programa Java.

Los ficheros `.class` pueden ejecutarse en cualquier plataforma (microprocesador + sistema operativo) para los que exista una versión de la JVM. Como, de hecho, existe una JVM para casi todas las plataformas comunes (incluyendo todas las versiones de Microsoft Windows, Solaris, Linux, Mac OS, etc.) se dice que los programas Java son plenamente *portables*, es decir pueden ejecutarse sobre cualquier plataforma (figura 1.3)[5].

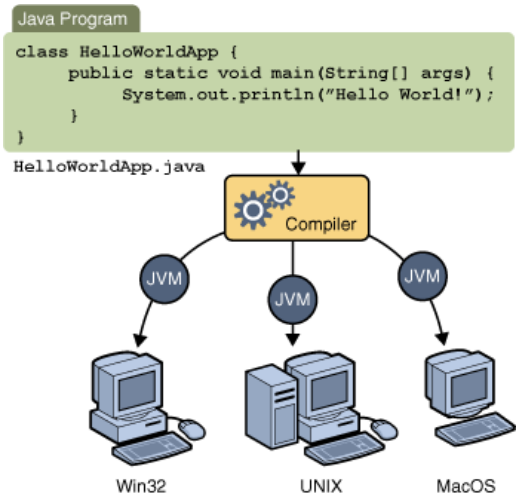

Figura 1.3: Portabilidad de los programas Java.

La figura 1.4 compara el enfoque de Java con el seguido por otros lenguajes (en la figura se toma como ejemplo C++) en los que es necesario desarrollar un compilador específico para cada plataforma de ejecución. Puesto que en Java también hay que desarrollar una máquina virtual específica para cada plataforma, el alumno puede preguntarse dónde está la ventaja. El hecho es que usualmente la implementación de un intérprete es más sencilla que la de un compilador y más sencilla todavía si, como es el caso, el lenguaje a interpretar es cercano al lenguaje máquina. Lo

[3] Fuente: http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html
[4] Conocida como JVM por sus siglas en inglés (Java Virtual Machine).
[5] *Write once, run anywhere*: escríbelo una vez, ejecútalo en cualquier sitio.

cierto es que el enfoque Java ha tenido un enorme éxito consiguiendo la portabilidad real de los programas escritos en dicho lenguaje.

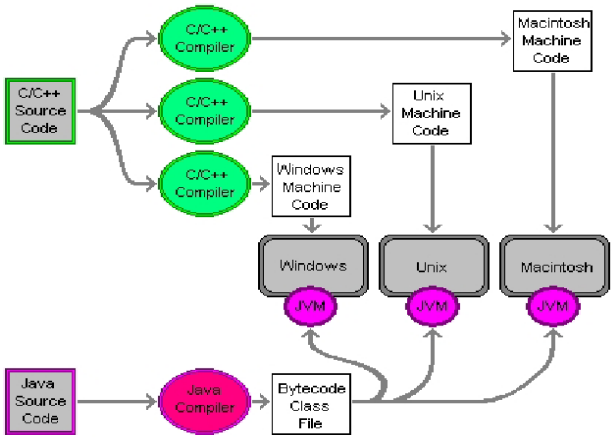

Figura 1.4: El enfoque Java frente al enfoque tradicional.

La plataforma Java

Una plataforma es el entorno hardware y software en el que se ejecuta un programa. La mayoría de las plataformas pueden describirse como una combinación de un sistema operativo y un microprocesador determinado. La plataforma Java difiere de otras plataformas en que es una plataforma exclusivamente software que se ejecuta sobre otra plataforma y que consta principalmente de dos componentes:

- La máquina virtual de Java
- La interfaz de programación de aplicaciones (API: *Application Programming Interface*)

La máquina virtual de Java ya ha sido comentada, la API es una extensa colección de componentes software a disposición de los programadores Java. La API se organiza en paquetes (*packages*) que contienen clases e interfaces. La figura 1.5 muestra esquemáticamente la ubicación de la plataforma Java, que ocupa un lugar intermedio entre nuestro programa y la plataforma de ejecución subyacente.

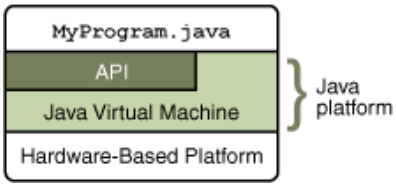

Figura 1.5: La plataforma Java.

Toda implementación de la plataforma de ejecución de Java incluye:

- Herramientas de desarrollo de software necesarias para compilar, ejecutar, depurar y documentar las aplicaciones Java, siendo las principales el compilador (javac), el lanzador y la herramienta de generación de documentación (javadoc).
- La API de Java. La API proporciona la funcionalidad esencial del lenguaje Java y ofrece una enorme cantidad de software que puede incorporarse a los programas, desde estructuras de datos (conjuntos, mapas, tablas, listas),

comunicaciones vía internet, componentes para realizar aplicaciones gráficas, gestión de la seguridad, etc. La API de Java está además muy bien documentada.

La plataforma Java puede descargarse de forma completa o parcial. Si un usuario solamente desea ejecutar aplicaciones Java, pero no desarrollarlas, basta con que tenga el JRE (*Java Runtime Environment*). Si, por el contrario, como es nuestro caso, desea además desarrollar aplicaciones Java necesita el JDK (*Java Development Kit*). Existen diversas versiones del JDK en función del tipo de aplicaciones que hay que desarrollar y del tipo de plataforma sobre las que deben ejecutarse. Por ejemplo:

- Plataforma Java, Edición Estándar (Java Platform, Standard Edition), o Java SE (antes J2SE)
- Plataforma Java, Edición Empresa (Java Platform, Enterprise Edition), o Java EE (antes J2EE)
- Plataforma Java, Edición Micro (Java Platform, Micro Edition), o Java ME (antes J2ME)

En nuestro caso vamos a trabajar con la edición estándar que pueden obtener en:

https://www.oracle.com/java/technologies/javase-downloads.html

La instalación simplemente consiste en hacer doble click sobre el ejecutable descargado, aceptar la licencia e instalar los productos que se muestran, salvo los ejemplos.

También es recomendable tener disponible la documentación, que puede obtenerse de la misma página web. La documentación está en formato html, lo que facilita en gran medida la navegación dentro de la propia documentación y el salto a páginas relacionadas.

El entorno Eclipse.[6]

Eclipse es un IDE y un IDE es un entorno integrado de desarrollo (*Integrated Development Environment*). Los IDE incluyen de forma coherente todo lo que hace falta para desarrollar programas en un lenguaje o en varios lenguajes de programación (editores, compiladores, enlazadores, depuradores, etc.) y pueden ser (Eclipse lo es) *entornos abiertos*, en los que pueden incorporarse herramientas de desarrollo software desarrolladas por terceros que pueden estar dirigidas a cualquiera de los pasos descritos en la figura 1.1 De Eclipse sólo vamos a ver lo estrictamente necesario para realizar las prácticas, que es una parte ínfima de todo lo que ofrece Eclipse[7]. El alumno interesado puede conectarse a la página de Eclipse. Otros ejemplos de IDEs son NetBeans[8] (también para Java) y Microsoft Visual Studio (que soporta todos los lenguajes de la plataforma .NET)[9].

Obtención e instalación de Eclipse.

El IDE Eclipse se puede obtener bajándolo directamente del sitio web oficial del Proyecto Eclipse - www.eclipse.org. La versión **Eclipse for Java Developers**, puede obtenerse en:

http://www.eclipse.org/downloads/packages

La instalación es extremadamente simple. Los ficheros descargados se descomprimen en una carpeta que suele ser `c:/Archivos de programa/eclipse` en la que hay que encontrar un ejecutable cuyo nombre es precisamente `eclipse` (véase figura 1.6). Pueden crear un acceso directo y copiarlo en el escritorio para tenerlo disponible.

---

[6] Fuente: http://ubuntulife.files.wordpress.com/2008/03/intro_eclipse_espanol.pdf

[7] El alumno interesado puede conectarse a la página de Eclipse www.eclipse.org

[8] Tanto Eclipse creado por IBM como Netbeans creado por Sun Microsystems son muy robustos y completos permitiendo crear casi toda clase de aplicaciones. La elección de uno u otro depende de lo cómodo y atractivo que sea para cada desarrollador.

[9] La plataforma .NET es la propuesta de Microsoft para competir con la plataforma Java.

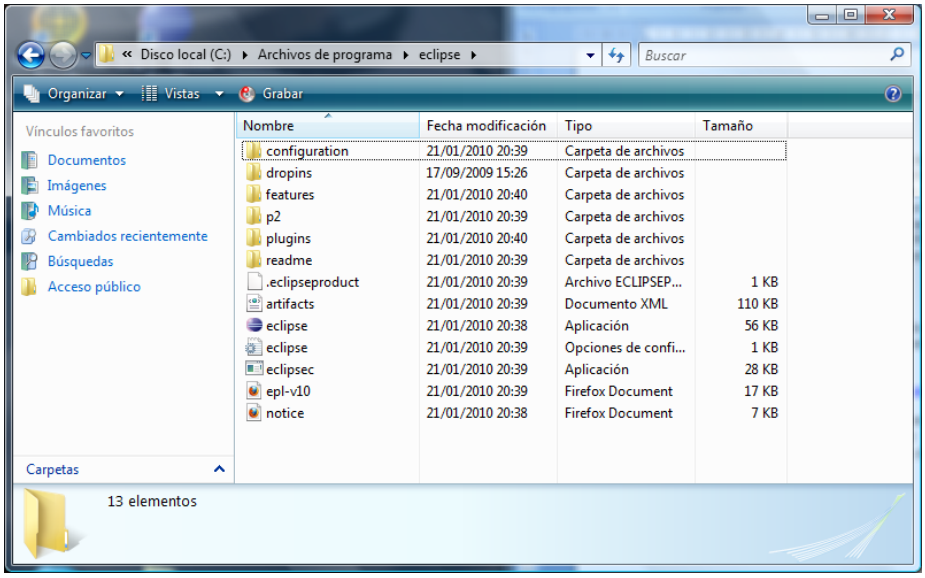

Figura 1.6: Archivos para instalación de Eclipse

Eclipse incorpora herramientas para poder compilar y ejecutar programas en Java. En principio para esta asignatura es suficiente. Para poder elegir desde Eclipse otras versiones de Java más completas o potentes debe haberse descargado e instalado previamente el JDK de Java.

Ejecución de Eclipse.

Para ejecutar Eclipse basta con hacer doble click sobre su icono. Lo primero que aparece es una ventana como se muestra en la figura 1.7.

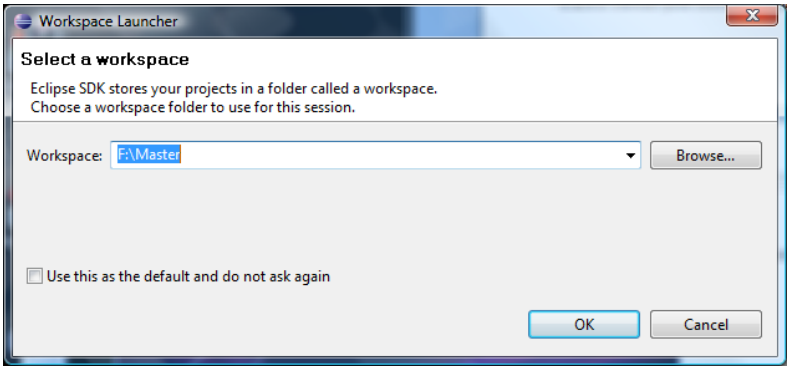

Figura 1.7: Selección del espacio de trabajo en Eclipse.

Esta ventana es un cuadro de diálogo en el que Eclipse les pregunta en qué espacio de trabajo (*workspace*) quieren trabajar, que coincide con un directorio determinado. Si el espacio de trabajo que les sugiere Eclipse no les conviene creen un directorio de trabajo (por ejemplo `../Mis Documentos/Fundamentos de Programacion`) y elíjanlo como espacio de trabajo. Una vez elegido el espacio de trabajo aparece la ventana[10] que se muestra en la figura 1.8.

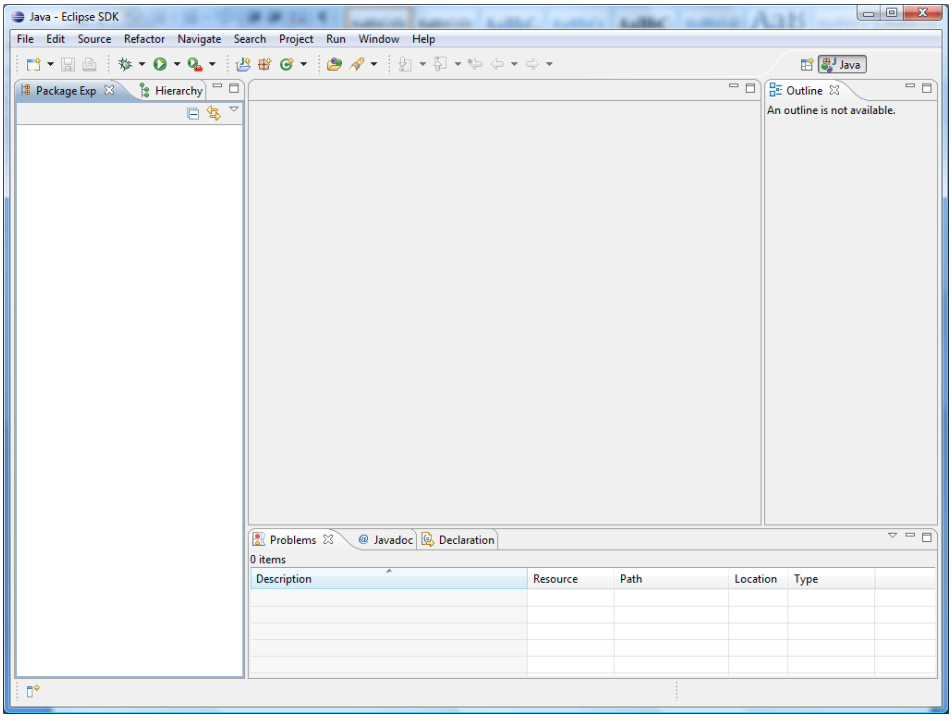

Figura 1.8: Ventanas en entorno Eclipse.

[10] Es posible que aparezca una ventana de bienvenida, ciérrenla y aparecerá la de la ilustración.

Creación de un proyecto Java.

Para poder realizar un programa en Java primero tenemos que crear un proyecto. Vamos entonces a crear un proyecto llamado `PracticasFP`[11]. Para ello, seleccionamos:

`File` → `New` → `Java Project`

Aparecerá el diálogo de la figura 1.9 en el que se nos pide el nombre del proyecto y se asume, si no lo cambiamos, que utilizamos el entorno de ejecución JDK indicado por defecto y que las fuentes y los códigos de bytes se almacenan en carpetas separadas.

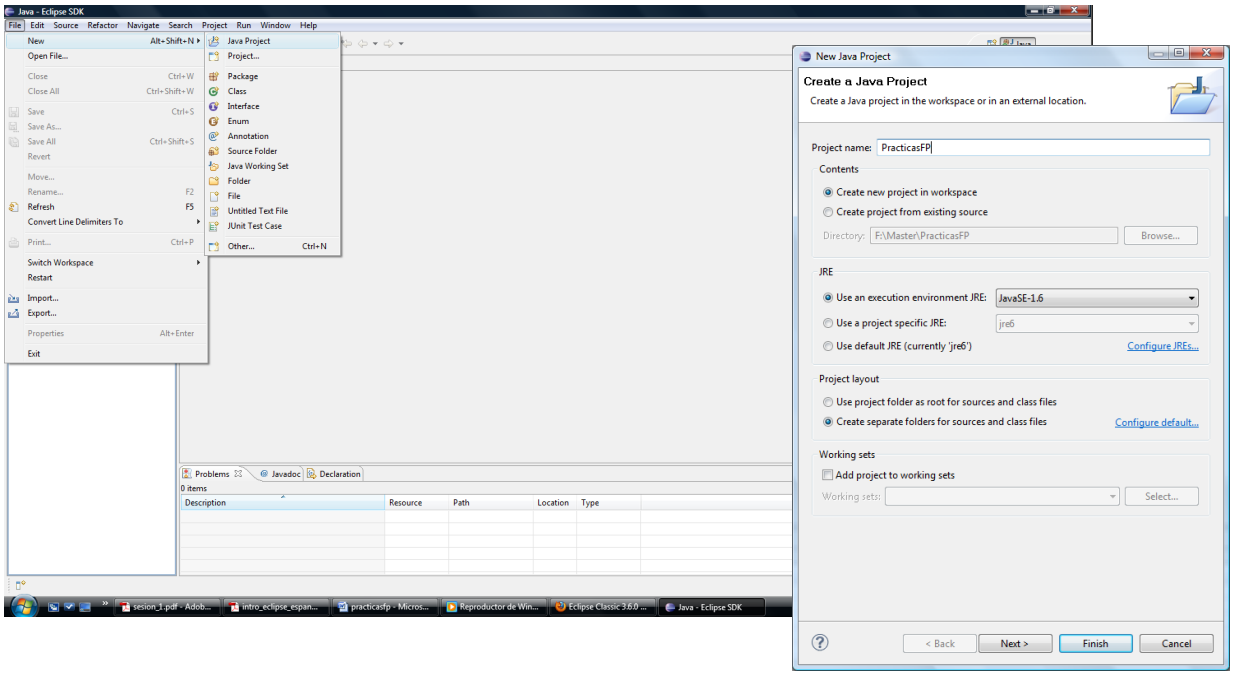

Figura 1.9: Creación de un proyecto en Eclipse.

Si estamos de acuerdo con la configuración propuesta en el diálogo damos directamente a **`Finish`** y nos aparecerá una carpeta en la parte izquierda de la ventana correspondiente a nuestro proyecto (véase figura 1.10). Aunque puede crearse más de un proyecto en un mismo espacio de trabajo nosotros trabajaremos siempre en el mismo proyecto y distinguiremos unas prácticas de otras colocándolas en paquetes diferentes.

[11] Llámenlo como mejor les parezca.

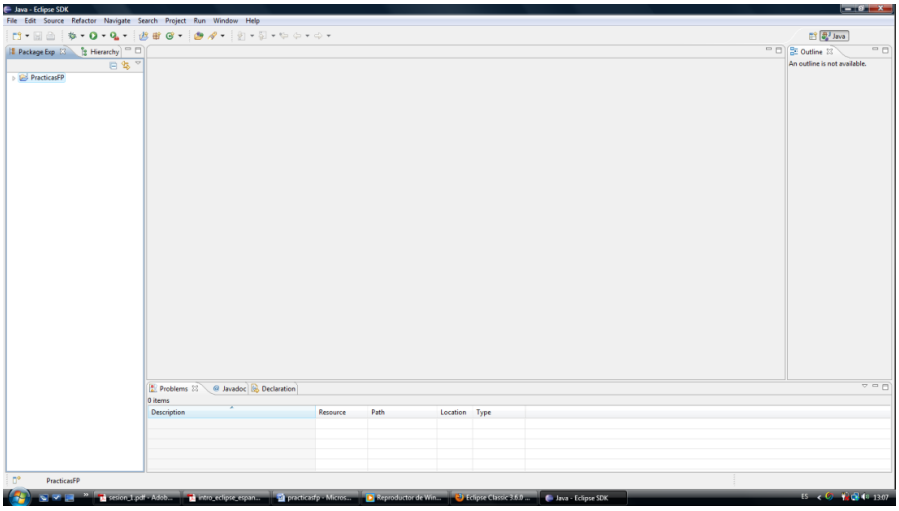

Figura 1.10: Carpeta de proyecto en entorno Eclipse.

Creación de un paquete en un proyecto Java.

Un paquete es un directorio donde agrupamos a otros paquetes y a los elementos básicos de un programa Java que pueden ser usados por otros programas Java: clases e interfaces. La forma de crear un paquete en Eclipse es abrir la carpeta del proyecto y seleccionar el paquete dentro del cual deseemos crear el nuestro. Actualmente el único paquete que debemos tener es `src`, así que lo seleccionamos con el ratón y pulsamos el botón derecho. Aparecen una serie de menús (véase figura 1.11) en los que seleccionamos:

`New` → `Package`

Nos aparece entonces un diálogo en el que se nos indica el paquete contenedor (en el ejemplo `practicasFP/src`) y el nombre del nuevo paquete, en nuestro caso `p1`.

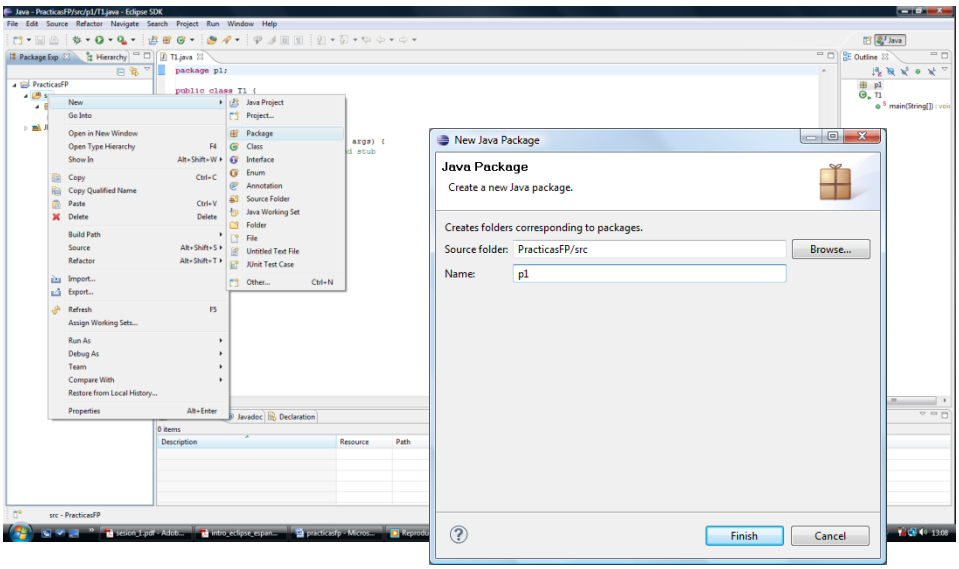

Figura 1.11: Creación de un paquete en entorno Eclipse.

Creación de una clase en un paquete.

Todos los programas Java están compuestos de clases e interfaces. Para crear una clase dentro de un paquete, seleccionamos con el ratón el paquete donde queremos crear la clase y pulsamos el botón derecho. Aparecen una serie de menús (véase figura 1.12) en los que seleccionamos:

```
New → Class
```

Nos aparece entonces un diálogo en el que se nos indica el paquete contenedor (en el ejemplo `practicasFP/src/p1`) y se pide el nombre de la clase, en nuestro caso `Programa1`. Se pregunta además si queremos que se nos genere de forma automática la cabecera de un método `main`. En nuestro caso hemos seleccionado que sí.

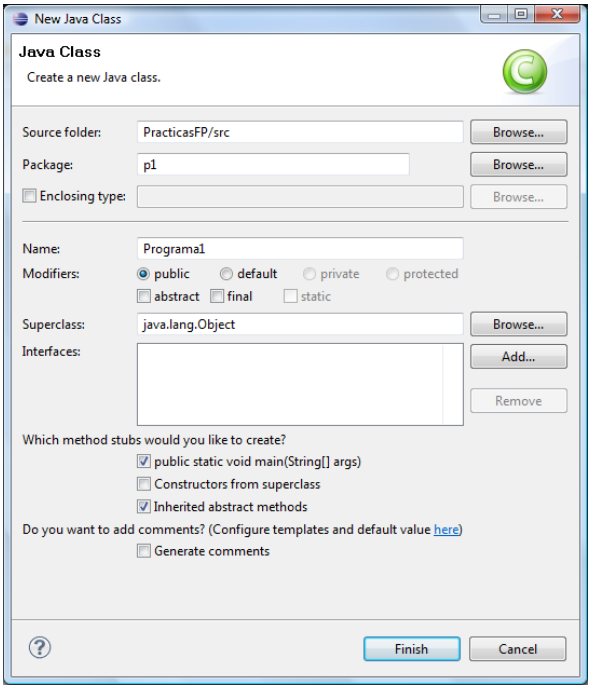

Figura 1.12: Creación de una clase en entorno Eclipse.

Cuando pulsamos `Finish` aparece en el editor un esquema de nuestra clase que podemos continuar editando.

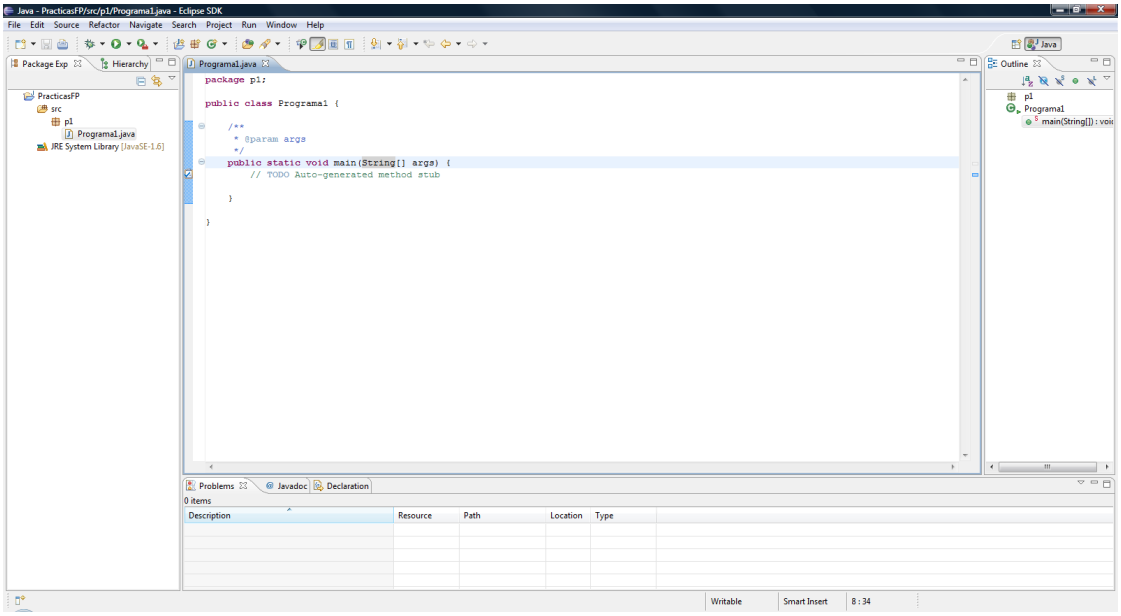

Figura 1.13: Esquema de una clase en entorno Eclipse.

Observe en la figura 1.13 que la primera línea del fichero es

```
package p1;
```

La primera línea de un fichero fuente debe indicar siempre el paquete en el que está dicho fichero, salvo que esté en el paquete por defecto. En las prácticas nunca se va a trabajar en dicho paquete. Cada práctica se realizará en su propio paquete y los elementos comunes a todas las prácticas se guardarán en el paquete `utilidades`.

Observe también que el nombre del fichero debe ser igual al de la clase pública que contiene y que Java distingue entre mayúsculas y minúsculas.

Edición, compilación y ejecución de un fichero fuente.

Cuando se edita un programa en el editor de Eclipse se va compilando al mismo tiempo que se va escribiendo. Nuestro primer programa va a ser el siguiente:

```
package p1;

public class Prg1 {

      public static void main(String[] args) {
         System.out.println("Hola Mundo");
         System.out.printf("%s \n", "Hola Mundo");
      }
}
```

En el código de **`main`** aparece una invocación a un primer método de un objeto:

```
System.out.println("Hola Mundo");
```

El objeto en cuestión se identifica mediante la referencia `System.out` y el método es `println`. Estas sentencias pueden interpretarse como un mensaje que mandamos al objeto `System.out` para que imprima en salida estándar `Hola Mundo`.

Otra forma de mostrar en pantalla un saludo es haciendo uso del método `printf` que muestra los datos en un determinado formato. En el código de main aparece una invocación a este método:

```
System.out.printf("%s \n,"Hola Mundo");
```

El primer argumento suele tener especificadores de formato que comienzan con `%` y van seguidos de un carácter que representa el tipo de datos. En este caso `%s` representa un `String` que es sustituido por el segundo argumento[12].

Una vez que el programa ha sido editado y compilado puede ejecutarse. Hay varias formas de hacer esto. Una es seleccionar el archivo que contiene el método `main` que se quiere ejecutar y como se muestra en la figura 1.14 seleccionar en el menú emergente:

```
Run as → java application
```

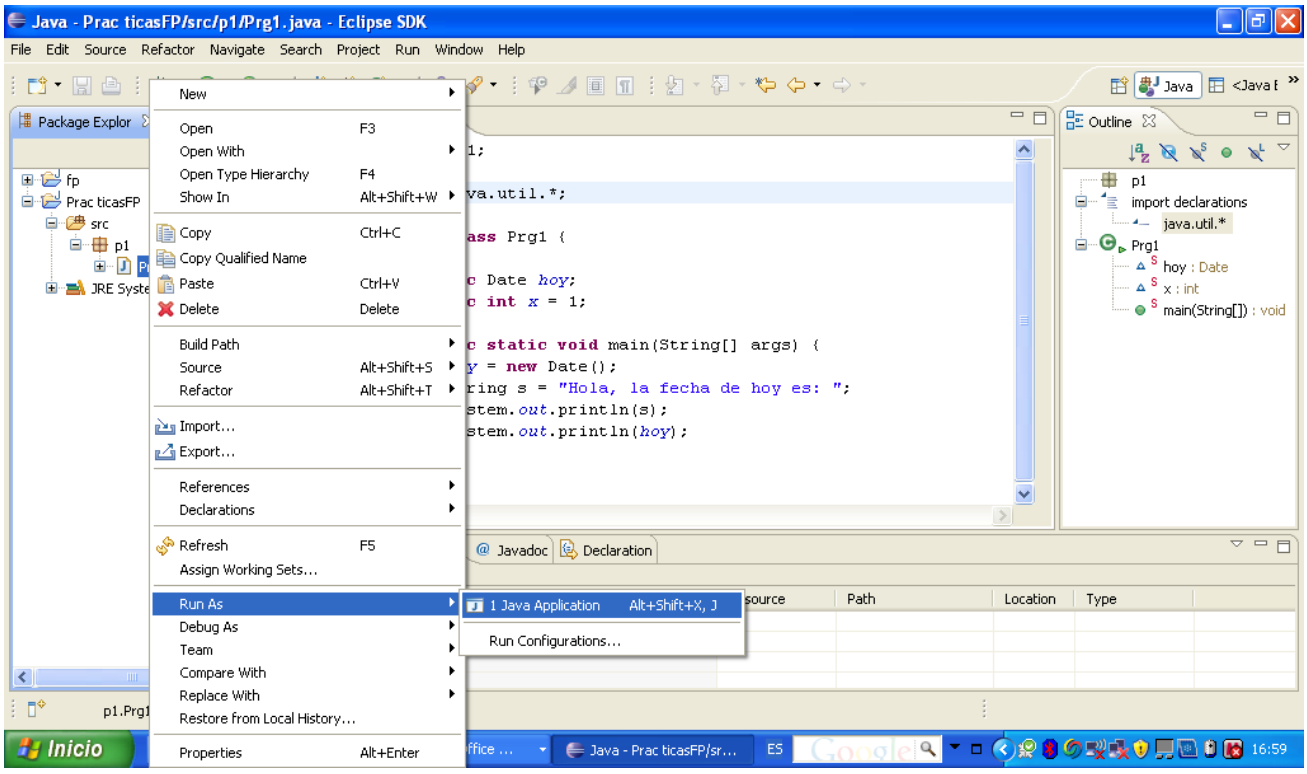

Figura 1.14: Ejecución de una aplicación en entorno Eclipse.

[12] Se remite al alumno a la lección 3 del Manual de Apuntes para más detalles.

Ejercicio.

Cree un paquete llamado `utilidades` y copie en él el fichero `Teclado.java` que puede encontrar en el aula virtual. Este fichero contiene la clase `Teclado` que se ha desarrollado para recoger información por parte del usuario a través del teclado del ordenador. Haciendo uso de esta clase podrá introducir números enteros, reales, etc., invocando los métodos correspondientes. Para que Eclipse detecte el nuevo fichero deberá "refrescar el proyecto". Seleccione el proyecto y en el menú emergente (ver figura anterior) seleccione `refresh`.

Cree un nuevo programa `Prg2` en el paquete `p1`, con el siguiente código:

```
package p1;
import utilidades.Teclado;
public class Prg2 {
   public static void main (String args[]) {
      System.out.println ("Este es mi segundo programa");
      System.out.println ("Introduzca un entero");
      int valor = Teclado.readInt();
      System.out.print("Se ha leído el valor: ");
      System.out.println (valor);
   }
}
```

Las dos últimas sentencias se pueden sustituir por:

```
System.out.println ("Se ha leído el valor: " + valor);
```

En este caso el valor es escrito en la misma línea porque cuando uno de los operandos del operador + es un String el otro se convierte a String y son concatenados[13].

Complete el programa anterior para poder introducir por teclado distintos tipos de datos primitivos utilizando las diferentes funciones de la clase `Teclado`.

Para las siguientes prácticas:

En el aula virtual se irán poniendo recursos para la realización de las prácticas. Se irán poniendo además las soluciones de las prácticas ya realizadas.

[13] Todos los objetos tienen un método toString que devuelve la representación String del objeto. Cuando un objeto es concatenado con un String, el método toString es invocado implícitamente para obtener la representación String del objeto.

Práctica 2: Tipos de datos.

Objetivos:

- Aprender a declarar y a usar variables de diferentes tipos en expresiones, en especial numéricas.
- Aprender las limitaciones de los tipos de datos numéricos.
- Aprender a escribir mensajes en consola mediante `System.out`.
- Aprender a leer valores desde teclado mediante la clase `Teclado`.
- Presentar la clase `Math` y la clase `Integer`.

Recursos:

Paquete `utilidades` con `Teclado.java`.

Paquete `Ejemplos` con los archivos `Ejemplo21.java`, `Ejemplo22.java` y `Ejemplo23.java`.

Ejemplos:

Ejecute los programas de las clases `Ejemplo21, Ejemplo22 y Ejemplo23` e inspeccione el código.

Estos programas incluyen declaraciones de variables de diferentes tipos y su uso en expresiones. En concreto la clase `Ejemplo21` trabaja con caracteres, `Ejemplo22` con variables numéricas y `Ejemplo23` con expresiones booleanas y operaciones sobre bits. Esta última muestra ejemplos de uso de la clase `Math` y de la clase `Integer`.

La clase `Math` proporciona un conjunto de métodos matemáticos muy útiles (operaciones trigonométricas, algebraicas, etc.). Consulte la ayuda (tendrá que tenerla instalada o utilice Google).

La clase `Integer` proporciona métodos para manejar números del tipo `int`. En el ejemplo se usa para obtener las representaciones hexadecimal y binaria de un número entero. La clase `Integer` es una *clase envoltorio*. Existe una por cada tipo de datos primitivo cuyo uso y significado se verán durante el curso. De momento la vamos a considerar una clase para manipular enteros.

Las clases `Math` e `Integer` son parte de `java.lang` por lo que no es necesario importarlas.

Mi Ejemplo.

Con objeto de recordar lo aprendido en los ejemplos proporcionados añada al paquete `Ejemplos` una clase `MiEjemplo` con un método `main` en el que:

- Se comparen los caracteres iniciales de tus apellidos y se muestre por pantalla el resultado de dicha operación.
- Convierta una variable numérica de tipo `int` con el valor de tu edad multiplicada por diez en tipo `char` y se muestre en pantalla el resultado de la operación.
- Muestre en pantalla la tabla de verdad del operador lógico OR (|).
- Muestre en pantalla el resultado de aplicar el operador lógico OR (|) entre dos variables de tipo short cuyos valores son 0xaa y 0x55.

Ejercicio 1.

Añada al paquete `p2` una clase `Calculadora` con un método `main` en el que:

- Se solicite al usuario que introduzca dos enteros por teclado (vea la interfaz de la clase `Teclado`).
- Se calculen la suma, la resta, el producto y la división de los dos enteros introducidos.
- Se calculen el resto de la división entera y el valor absoluto del primer operando.
- Se muestren los resultados de todas las operaciones por pantalla, indicando a qué operación corresponden.

Pruebe a introducir números muy grandes, cercanos al máximo del tipo de datos que elija. Pruebe también a realizar una división por cero.

Ejercicio 2.

Añada al paquete `p2` una clase `ConversorRadianes` con un método `main` en el que:

- Se solicite al usuario que introduzca un valor `double` por teclado. Este valor representará un ángulo medido en grados sexagesimales.
- Transforme este ángulo a radianes multiplicándolo por `PI` y dividiéndolo por 180. Observe que la clase `Math` proporciona el valor de la constante `PI`.
- Calcule el seno, coseno y tangente del ángulo transformado y muestre los resultados por pantalla. Recuerde siempre que las funciones trigonométricas de la clase `Math` trabajan en radianes por ello previamente hay que hacer la conversión de grados a radianes.

Pruebe al menos con los valores correspondientes a 0, 30, 45 y 90 grados.

Ejercicio 3.

Añada al paquete `p2` una clase `Ecuacion` con un método `main` en el que:

- Se soliciten por teclado tres enteros: a, b y c, que serán los coeficientes de la ecuación de segundo grado:

  $y = ax^2 + bx + c$

- Se calculen y se muestren por pantalla los valores de x que resuelven la ecuación.

Pruebe con diferentes valores de los coeficientes.

Ejercicio 4.

Implemente un programa que calcule el área de un cilindro a partir del radio de la base y la altura. El método `main` debe pedir al usuario que introduzca por teclado el radio de la base y la altura, y mostrar en la consola el área calculada.

NOTA:

El área del cilindro será dos veces el área de la base más el área lateral que puede calcular a partir de la altura y el perímetro de dicha base.

Ejercicio 5.

Implemente un programa que calcule el área de un prisma regular hexagonal a partir del lado o radio de la base y la altura. El método `main` debe pedir al usuario que introduzca por teclado el lado de la base y la altura, y mostrar en la consola el área calculada.

NOTA:

Para ello deberá calcular a priori el área de la base a partir del lado. Dicho área es la de un hexágono regular:

`Area = perimetro * apotema / 2` siendo la `apotema = (lado/2) *` $\sqrt{3}$

El área del prisma será dos veces el área de la base más el área lateral que puede calcular a partir de la altura y el perímetro de dicha base.

Ejercicio 6.

Implemente un programa que obtenga la suma de los n primeros términos de una serie aritmética. El método `main` debe pedir al usuario que introduzca por teclado el primer término, la distancia y el número de términos que se desean sumar, y mostrar en consola el resultado obtenido.

NOTA:

El cálculo debe hacerse aplicando la fórmula: $S = n*(a_1 + a_n)/2$ siendo el n-ésimo término $a_n = a_1 + (n - 1)*d$. Deberá trabajar sólo con enteros.

Ejercicio 7.

Implemente un programa que obtenga la suma de los n primeros términos de una serie geométrica. El método `main` debe pedir al usuario que introduzca por teclado tres enteros: el primer término, la razón y el número de términos, y mostrar en la consola el resultado.

NOTA:

El cálculo debe hacerse aplicando la fórmula: $S = \frac{a_n * r - a_1}{r-1}$ siendo el n-ésimo término $a_n = a_1 \cdot r^{n-1}$

Utilice la función *pow* de la clase `Math` para elevar un número a otro.

Práctica 3: Estructuras de control.

Objetivos:

- Aprender a utilizar la sentencia de selección `if/else` y el selector `switch`.
- Aprender a utilizar las sentencias de repetición.
- Utilizar en expresiones variables de diferentes tipos, en especial numéricas.
- Introducir el concepto de función.

Recursos:

Paquete `utilidades` con la clase `Teclado`.

Ejercicio 1.

Añada al paquete `p3` la clase `Ecuacion` con un método `main` en el que:

- Se soliciten por teclado tres enteros: `a`, `b` y `c`, que serán los coeficientes de la ecuación de segundo grado:

  $y = ax^2 + bx + c$

- Se calculen `y` se muestren por pantalla los valores de `x` que resuelven la ecuación.
- Muestre el mensaje "La ecuación tiene raíces complejas" si las raíces de la ecuación son complejas.

Ejercicio 2.

Añada al paquete `p3` la clase `Calculadora` con un método `main` en el que:

- Se solicite al usuario que introduzca dos enteros por teclado (clase `Teclado`).
- Después de introducir los operandos pida mediante un código numérico la operación a realizar. Por ejemplo:
  - Pulse: (1) para sumar, (2) para restar, (3) para multiplicar, (4) para dividir, y (5) para residuo.
- Se realice la operación seleccionada y se muestre el resultado por pantalla. En las operaciones de división y residuo, si el segundo operando es igual a cero debe mostrarse el mensaje "Imposible dividir por cero".

  NOTA: Utilice la estructura de control `switch`.

Ejercicio 3.

Añada al paquete `p3` la clase `DivisiblePor11` con un método `main` tal que:

- Muestre por pantalla todos los números divisibles por 11 existentes en el rango `(1..N)`, donde `N` es un valor solicitado al usuario por teclado. Hágalo empleando:

  a) Un bucle `for`.

  b) Un bucle `while`.

  c) Un bucle `do/while`.

- Modifique el programa para que sólo muestre los divisores PARES de 11.

Ejercicio 4.

Añada al paquete `p3` la clase `MCD` con un método `main` tal que:

- Pida por teclado dos números enteros.
- Calcule su máximo común divisor (mcd) utilizando el siguiente algoritmo:

- Supongamos que a y b son los números.
- Si a es mayor que b y su mcd es f, entonces el mcd de a-b y b será también f.
- Según la propiedad anterior si reemplazamos repetidamente el mayor de los dos números por su diferencia, hasta que los dos números sean el mismo, entonces ese número será el mcd de los números a y b.

Ejercicio 5.

Añada al paquete `p3` la clase `Fibonacci`.

Fibonacci fue un matemático italiano del siglo XIII que definió la serie que lleva su nombre. Cada nuevo valor de esta serie es el resultado de la suma de los dos anteriores. Y los dos primeros elementos de la serie son unos.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Escriba un método `main` que imprima por pantalla los 50 primeros términos de la serie de `Fibonacci`. Compruebe que para los últimos términos se produce desbordamiento.

Ejercicio 6.

En este ejercicio y en los sucesivos, vamos a encapsular el código dentro de funciones que ubicaremos en una clase `Calculos`. El concepto de función se explica con detalle en la lección 5, y vamos a empezar a usarla desde este momento. Una función tiene un nombre, una lista de parámetros de entrada y produce un valor de salida. Esta información se indica en la cabecera de la función. Ya se han utilizado algunas funciones de la clase `Teclado` y de la clase `Math`.

El número PI puede calcularse de forma iterativa utilizando diferentes expresiones. Una de ellas es la fórmula de Leibniz, que se muestra a continuación.

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

Puesto que es imposible realizar infinitas iteraciones se pide que implemente la función `piLeibniz` cuyo cabecara es la siguiente:

```
public static double piLeibniz(long iters)
```

Esta función, toma como argumento el valor máximo de n y retorna el valor de pi correspondiente. Si el número de iteraciones es menor que cero, la función retorna -1.

Ejercicio 7.

El factorial de n responde a las siguientes expresiones:

$$n \geq 0,\ 0! = 1,\ n > 0\ n! = n \cdot (n-1)!$$

Se pide que implemente la función factorial cuya cabecera es la siguiente:

```
public static long factorial(long n)
```

Esta función, toma como argumento el valor de n y retorna el valor del factorial de n. Si n es menor que cero, la función retorna -1.

Ejercicio 8.

Se pide que implemente la función `potenciaEntera` cuya cabecera es la siguiente:

```
public static long potenciaEntera(long base, int exp)
```

Esta función, toma como argumentos la base y el exponente y calcula $base^{exp}$. Si `exp` es menor que cero, la función retorna -1.

NOTA: No utilizar la función pow de la clase Math porque trabaja con números reales. En este ejercicio se trata de calcular la potencia entera.

Ejercicio 9.

El número PI puede calcularse de forma iterativa utilizando diferentes expresiones. Una de ellas es la fórmula de Euler, que se muestra a continuación.

$$\sum_{n=0}^{\infty} \frac{2^n n!^2}{(2n+1)!} = 1 + \frac{1}{3} + \frac{1 \cdot 2}{3 \cdot 5} + \frac{1 \cdot 2 \cdot 3}{3 \cdot 5 \cdot 7} + \cdots = \frac{\pi}{2}$$

Puesto que es imposible realizar infinitas iteraciones se pide que implemente la función `piEuler` cuyo prototipo es el siguiente:

```
public static double piEuler(long iters)
```

Esta función, toma como argumento el valor máximo de n y retorna el valor de pi correspondiente. Si el número de iteraciones es menor que cero, la función retorna -1. Si funcionan correctamente los métodos anteriores, puede utilizarlos en este. ¿Con cuántas iteraciones deja de funcionar el método? ¿A qué se debe?

Ejercicio 10.

Añada a la clase `Calculos` un método `main` en el que:

- Pida mediante un código numérico una de las siguientes operaciones:
  - (1) pi según Leibniz, (2) pi según Euler, (3) potencia entera, (4) factorial .
- Se solicite al usuario que introduzca por teclado (clase `Teclado`) los datos asociados a la opción elegida.
- Se muestre el resultado de la operación seleccionada por pantalla indicando a que operación corresponde.
- Si la operación realizada produce un error, debe mostrarse el mensaje correspondiente.

Práctica 4: Arrays.

Objetivos:

- Aprender a utilizar arrays para almacenar y modelar datos y a utilizarlos como argumentos y valores de retorno.

Recursos:

Paquete `utilidades` con la clase `Teclado`.

Las funciones implementadas en los diez primeros ejercicios las vamos a encapsular dentro de una clase `ProcesarArrays` que contendrá un `main` para probarlas. Las funciones implementadas en los cuatro últimos ejercicios las vamos a encapsular dentro de otra clase `ProcesarMatrices` que también contendrá el programa principal para probarlas.

Ejercicio 1.

Implemente una función que tome como argumento un array de números enteros y devuelva la suma de sus elementos.

Prototipo:

```
public static int sumaElementos(int [] a)
```

Comprobación de argumentos:

- Devuelve `-1` si: (1) el array de entrada es igual a `null` o está vacío.

Ejercicio 2.

Implemente una función que tome como argumentos dos arrays de números enteros y devuelva otro array resultado de sumar los argumentos elemento a elemento.

Prototipo:

```
public static int [] sumaArrays(int [] a, int [] b)
```

Comprobación de argumentos:

- Devuelve `null` si: (1) alguno de los arrays de entrada es igual a `null` o está vacío, (2) si los arrays no son de igual longitud.

Ejercicio 3.

Implemente una función que tome como argumentos un array de números enteros y un entero y devuelva el número de ocurrencias de dicho entero en el array.

Prototipo:

```
public static int cuentaOcurrencias(int [] a, int k)
```

Comprobación de argumentos:

- Devuelve `-1` si: (1) el array de entrada es igual a `null` o está vacío.

Ejercicio 4.

Implemente una función que tome como argumentos un array de números enteros sin elementos repetidos y otro número entero que representa un elemento clave, y devuelva el índice del array en el que se encuentra almacenada dicha clave utilizando el algoritmo de búsqueda lineal.

Prototipo:

```
public static int busquedaLineal(int [] a, int clave)
```

Comprobación de argumentos:

- Devuelve `-1` si: (1) el array de entrada es igual a `null` o está vacío, (2) si no encuentra la clave.

NOTA: Suponga que no hay elementos repetidos. No hace falta que haga esta comprobación.

Ejercicio 5.

Dados dos arrays unidimensionales de números enteros de igual longitud, escriba un método que tome como argumentos ambos arrays y devuelva otro array resultado de tomar alternativamente un elemento de cada uno de los arrays originales tomando el primer elemento del primer argumento.

Por ejemplo, si los argumentos son `a={1,2,3,4,5}` y `b={6,7,8,9,0}` entonces el método devuelve `{1,6,2,7,3,8,4,9,5,0}`.

Prototipo:

```
public static int [] alternarElementos(int [] a, int [] b)
```

Comprobación de argumentos:

- Si algún array de entrada es `null` o está vacío devuelve `null`.
- Si los arrays de entrada no tienen la misma longitud devuelve `null`.

Ejercicio 6.

a) Implemente una función que tome un array de números enteros como argumento y devuelva un **nuevo** array resultado de invertir el orden de sus elementos. Antes de retornar el array deberá mostrarlo en pantalla. Por ejemplo:

`a = {1, 2, 3, 4, 5, 6}` → `invertir(a), b = {6, 5, 4, 3, 2, 1}`

Prototipo:

```
public static int [] invertir(int [] a)
```

Comprobación de argumentos:

- Devuelve `null` si: (1) el array de entrada es igual a `null` o está vacío.

b) Repita el ejercicio sin crear un nuevo array, es decir, el método modificará y devolverá el array **original** invirtiendo sus elementos.

Ejercicio 7.

Implemente una función que tome como argumentos un array de números enteros y el número entero que deberá eliminar y devuelva otro array con los mismos elementos del argumento excepto el entero a eliminar. Por ejemplo:

`int a = {1, 2, 3, 4, 5, 6}` → `eliminarK(a,1), devuelve = {2, 3, 4, 5, 6}`
`int b = {1, 2, 2, 3, 5, 2}` → `eliminarK(b,2), devuelve = {1, 3, 5}`

El argumento no se modifica.

Prototipo:

```
public static int [] eliminarK(int [] a, int k)
```

Comprobación de argumentos:

- Si el array es `null` devuelve `null`.
- Si el array tiene longitud 0 devuelve un array de longitud 0.
- Si todos los elementos del array son iguales al entero a eliminar devuelve un array de longitud 0.

Utilice la función `cuentaOcurrencias` del ejercicio 3 para calcular el tamaño del nuevo array.

Ejercicio 8.

Implemente una función que tome como argumento un array ordenado de números enteros y compruebe si hay elementos repetidos.

Prototipo:

```
public static boolean elementosRepetidos(int [] a)
```

Comprobación de argumentos:

- Devuelve `false` si : (1) el array de entrada es `null`, sin elementos o con un solo elemento, (2) si no hay elementos repetidos.

Ejercicio 9.

Escriba un método que tome como argumento un array de enteros y un número entero y devuelva un array de enteros conteniendo los índices de las posiciones del array original que contienen al número pasado como argumento. Por ejemplo, si los argumentos fueran:

El array `a = {1,3,7,3,4}` y el número `n = 3`, el método debería devolver `b={1,3},` ya que el número 3 se encuentra en las posiciones 1 y 3 del array `a.`

Prototipo:

```
public static int [] indices(int [] a, int n)
```

Comprobación de argumentos:

- Devuelve `null` si: (1) el array de entrada es `null` o está vacío (2) el array no contiene el número dado.

Ejercicio 10.

a) Implemente una función que tome como argumentos un array de números enteros y un valor booleano de forma que compruebe si está o no ordenado de manera ascendente o descendente en función del segundo argumento.

Prototipo:

```
public static boolean enOrden(int [] a, boolean ascendente)
```

Comprobación de argumentos:

- Devuelve `false` si: (1) el array de entrada es igual a `null` o está vacío, (2) está desordenado.

b) Implemente otra función que tome como argumentos un array de números enteros y un valor booleano de forma que devuelva otro array con los elementos del argumento ordenados en orden ascendente o descendente según indique el segundo argumento. Muéstrelo en pantalla antes de retornarlo. No se modifican los elementos del argumento, es decir, deberá crear un array **nuevo** para manipularlo.

Prototipo:

```
public static int [] ordenar(int [] a, boolean ascendente)
```

Comprobación de argumentos:

- Devuelve `null` si: (1) el array de entrada es igual a `null`, está vacío o tiene un solo elemento .
- Si el array de entrada está ordenado el método devuelve una referencia al array original. Utilice la función anterior para comprobarlo.

Ejercicio 11.

Implemente una función que tome como argumento un array bidimensional de números reales y compruebe si es matriz. Si es una matriz la función devolverá `true` y `false` en caso contrario.

Prototipo:

```
public static boolean esMatriz(double [][] a)
```

Comprobación de argumentos:

Devuelve `false` si el array de entrada es `null`, y `true` si es de longitud cero.

Ejercicio 12.

Se pide implementar una función que tome como argumento un array de dos dimensiones, compruebe que es una matriz y devuelva otra matriz como la original, pero eliminando una de sus filas. La fila a eliminar se pasa también como argumento.

Prototipo:

```
public static double [][] quitarFila(double [][] m, int n)
```

Comprobación de argumentos:

- Si el array de entrada es `null` o está vacío devuelve `null`.
- Si el array de entrada no es matriz devuelve `null`.
- Si no existe la fila devuelve `null`.

Observaciones:

No se puede modificar la matriz original. Utilice la función anterior para comprobar que es matriz.

Ejercicio 13.

Se pide implementar una función que tome como argumento un array de dos dimensiones, compruebe que es una matriz y devuelva otra matriz como la original, pero eliminando una de sus columnas. La columna a eliminar se pasa también como argumento.

Prototipo:

```
public static double [][] quitarCol(double [][] m, int n)
```

Comprobación de argumentos:

- Si el array de entrada es `null` o está vacío devuelve `null`.
- Si el array de entrada no es matriz devuelve `null`.
- Si no existe la columna devuelve `null`.

Observaciones:

No se puede modificar la matriz original. Utilice la función anterior para comprobar que es matriz.

Ejercicio 14.

Se pide implementar una función que tome como argumento un array de dos dimensiones, compruebe que es una matriz y devuelva el menor complementario. El menor complementario es otra matriz resultante de eliminar una fila y una columna. El número de fila y columna a eliminar se pasan también como argumentos a la función.

Prototipo:

```
public static double [][] menorComplementario(double [][] m, int i,int j)
```

Observaciones:

No se puede modificar la matriz original. Utilice las funciones anteriores.

Práctica 5: Programación orientada a objetos.

Objetivos:

- Entender los principios de encapsulación y ocultamiento de la información.
- Aprender a definir y a utilizar tipos abstractos de datos y ser capaces de crear y manejar objetos.

Recursos:

- Realice los dos primeros ejercicios en un paquete 51.
- Para realizar el ejercicio 4 se suministra la interfaz **IForma**. Haga este ejercicio en un paquete 52.

Ejercicio 1.

Se trata de modelar objetos geométricos de diferentes tipos que puedan ser trasladados y girados en un espacio de dos dimensiones. Para ello, debe comenzar por implementar la clase `Punto` para que modele una posición en un espacio de dos dimensiones.

Los objetos de esta clase son **inmutables**. Observe que puesto que los objetos no pueden modificarse, los métodos trasladar y girar no modifican el objeto sobre el que se invocan, sino que crean un nuevo objeto correspondiente a haber trasladado o girado al objeto cuyo método ha sido invocado.

Trabaje con el tipo `double` para modelar números reales.

Constructores:

Se deben proporcionar:

- Un constructor sin argumentos que cree un punto en [0, 0].
- Un constructor que tome como argumentos las coordenadas del punto.

Métodos:

Un objeto de la clase `Punto` tiene los siguientes métodos:

- Métodos de acceso para obtener las coordenadas del punto.
- Un método `getOrientacion` que devuelve cero. NOTA: Este método será útil para poder realizar los siguientes ejercicios.
- Un método de giro, que devuelve otro punto correspondiente al giro del punto sobre el cual se invoca el método. Dicho giro se realiza respecto de un centro que es otro punto cuyas coordenadas x e y se pasan como argumento. El ángulo de giro también se pasa en grados como argumento. Este método tendría el siguiente prototipo:

  ```
  public Punto girar(double grados, double x0, double y0)
  ```

  El signo del ángulo de giro determina el sentido de giro, *positivo antihorario*, *negativo, horario*. El ángulo debe introducirse en grados (véase figura 5.1 para la obtención de las coordenadas del punto resultante).
- Un método de giro, que devuelve otro punto correspondiente al giro del punto sobre sí mismo. Dicho método devuelve una copia del punto y tendría el siguiente prototipo:

  ```
  public Punto girar(double grados)
  ```

- Un método de traslación, que devuelve otro punto correspondiente a la traslación del punto sobre el cual se invoca el método. La traslación viene definida por un desplazamiento en x y otro en y.
- Un método que devuelve la distancia entre el punto sobre el cual se invoca el método y otro punto que se pasa como argumento.

- Un método `toString` que debe devolver una cadena de caracteres con el siguiente formato [x, y]. En este caso, como se utiliza una clase auxiliar para dar formato a los números se proporciona el código de dicho método:

```
public String toString(){

        // Crea formato de representación con dos decimales.
        DecimalFormat form = new DecimalFormat();
        form.setMaximumFractionDigits(2);

        // Obsérvese cómo se formatean los números para
        // representarlos con dos decimales
        return "[" + form.format(x) + ", " + form.format(y) + "]";
}
```

NOTA: Deberá importar la clase `DecimalFormat` (`import java.text.DecimalFormat`).

Pruebas sobre la clase `Punto` en un método `main`.

Cree una clase TestP51 que en su método `main` incluya las siguientes operaciones:

- Defina un array, llámelo `arrayPuntos1`, de 8 puntos distribuidos uniformemente en una circunferencia de radio 100, con el primero de ellos localizado en [100, 0]. El resto de los puntos obténgalos girando el anterior 45 grados. Para ello, utilice el método girar *del punto inmediatamente anterior respecto del origen de coordenadas*.
- Imprima las coordenadas de dichos puntos (escribiendo cada punto en una línea diferente).
- Cree un nuevo array, llámelo `arrayPuntos2`, resultado de trasladar los puntos del array anterior 25 hacia la derecha y 100 hacia arriba.
- Imprima las coordenadas de dichos puntos.
- Imprima las distancias entre el tercer punto de `arrayPuntos1` y todos los puntos de `arrayPuntos2`.

Coordenadas de un punto (x1, y1) resultado de girar un punto (x, y) respecto de un centro de giro (x0, y0) un ángulo α:

$$x1 = x0 + (x - x0) * \cos \alpha - (y-y0) * \text{sen } \alpha$$

$$y1 = y0 + (x - x0) * \text{sen } \alpha + (y-y0) * \cos \alpha$$

Figura 5.1

Advertencias y consejos:

- No ignore los errores. Como verá la clase `Vector` se implementa a partir de la clase `Punto`. Por ejemplo, un vector está definido por dos puntos. No trate de implementar la clase `Vector` si antes no ha implementado bien la clase `Punto`.

Ejercicio 2.

Implemente la clase `Vector` para modelar un vector en el espacio de dos dimensiones. El vector que se pide queda definido por sus puntos origen y extremo. Las únicas variables de instancia que pueden definirse deben representar a dichos puntos. No se admite el uso de ninguna otra variable de instancia adicional.

Constructores:

Se deben proporcionar:

- Un constructor sin argumentos. Devuelve el vector unitario según la dirección x.
- Un constructor que tome como argumentos los puntos origen y extremo.

Métodos:

Un objeto de la clase `Vector` tiene los siguientes métodos:

- Un método `getOrientacion` que devuelva la orientación del vector, es decir, el ángulo en grados que forma el vector con el eje x.
- Métodos que devuelvan los puntos origen y extremo del vector.
- Un método que devuelva el módulo, es decir, la distancia entre el punto origen del vector y el punto extremo del vector.
- Métodos de obtención de las componentes del vector sobre los ejes coordenadas, es decir, las proyecciones del vector sobre dichos ejes (véase figura 5.2).
- Un método de giro. Girar el vector respecto de un centro de giro significa girar sus puntos origen y extremo respecto de dicho centro de giro. Nuevamente, no se modifica el objeto vector original, sino que se devuelve un nuevo vector resultado de haber girado el original. Dicho método tendría el siguiente prototipo:

```
Vector girar(double grados, double x0, double y0)
```

- Un método de giro de un vector respecto de su punto de origen. Este método funciona como el anterior, pero el centro de giro es el origen del vector. Obsérvese que el origen sigue siendo el mismo, pero cambia el extremo. Este método cambia el ángulo que forma el vector con el eje X. Nuevamente, no se modifica el objeto vector original, sino que se devuelve un nuevo vector resultado de haber girado el original. Dicho método tendría el siguiente prototipo:

```
Vector girar(double grados)
```

- Un método de traslación. Trasladar el vector significa trasladar sus puntos origen y extremo. No se modifica el objeto vector original, sino que se devuelve un nuevo vector resultado de haber trasladado el original.
- Un método `toString` que debe devolver una cadena de caracteres con el siguiente formato [[O_x, O_y], módulo, orientación], donde [O_x, O_y] representa el punto origen del vector. En este caso, como se utiliza una clase auxiliar para dar formato a los números se proporciona el código de dicho método:

```
public String toString(){

        // Crea formato de representación con dos decimales.
        DecimalFormat form = new DecimalFormat();
        form.setMaximumFractionDigits(2);

        // Obsérvese cómo se formatean los números para
        // representarlos con dos decimales
        return "[" + origen.toString() + ", " + form.format(modulo()) + ", "
        + form.format(getOrientacion()) + "]";
}
```

NOTA: Deberá importar la clase `DecimalFormat` (`import java.text.DecimalFormat`).

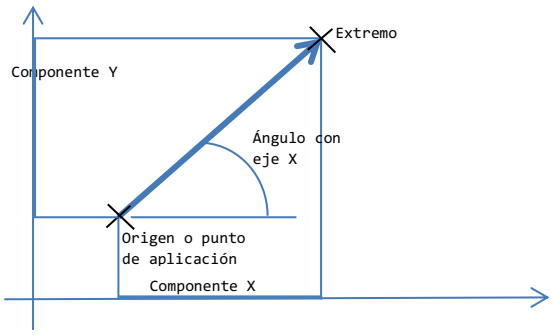

Figura 5.2

Pruebas sobre la clase `Vector` en un método `main`.

Complete la clase TestP51 para que en su método `main` incluya las siguientes operaciones:

- Defina un array, llámelo `arrayVectores1`, de 8 vectores distribuidos uniformemente en una circunferencia de radio 100. El primer vector sobre el eje x tiene como origen el punto [100,0] y como extremo el punto [110, 0]. Puede obtener el siguiente de los vectores *girando el primero de ellos 45 grados con respecto al origen de coordenadas [0,0]. Así sucesivamente vaya girando 45 grados cada vector para obtener el siguiente.*
- Imprima el array de vectores con cada vector en una línea diferente.
- Cree un nuevo array, llámelo `arrayVectores2`, resultado de trasladar los vectores del array anterior 25 hacia la derecha y 100 hacia arriba.
- Imprima el nuevo array de vectores con cada vector en una línea diferente.
- Imprima las distancias entre el origen del tercer vector de `arrayVectores1` y los orígenes del resto de vectores de `arrayVectores2`.

Ejercicio 3.

Dada la interfaz **`IForma`** defina nuevamente las clases `Punto` y `Vector` para que implementen dicha interfaz. No olviden sobrescribir también el método `toString()`. Al sobrescribir los métodos de la interfaz observe que pueden devolver un objeto de la clase que la implementa en lugar de un objeto del tipo de la interfaz. De esta manera ahorrará hacer un *casting* para concretar el tipo del objeto posteriormente. Por ejemplo, el método:

```
IForma trasladar(double x, double y)
```

Se puede sobreescribir en la clase `Punto` como :

```
Punto trasladar(double x, double y)
```

La interfaz `IForma` modela un objeto geométrico que está situado en una posición de un espacio de dos dimensiones y que puede ser trasladado o girado respecto de un centro de giro.

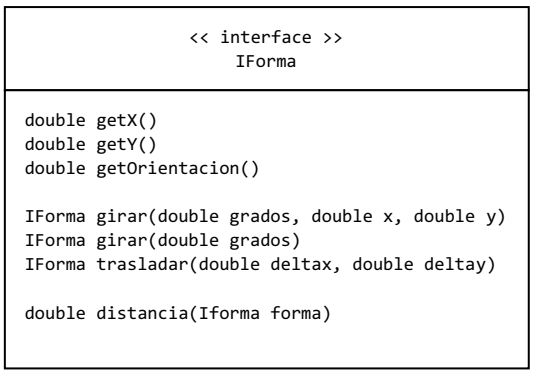

Figura 5.3

La interfaz (véase figura 5.3) define métodos para obtener la posición y orientación del objeto en el espacio y para trasladarlo y girarlo. Define además un método que devuelve la distancia del objeto que recibe el mensaje respecto del argumento.

Los detalles de los métodos están documentados en el código fuente.

Por ultimo, implemente la clase `TestP52` para incluir en su método `main` todas las operaciones de la clase `TestP51`.

NOTA: Haga este ejercicio en un paquete separado (paquete 52), copiando la solución de los ejercicios anteriores y modificándola.