TEMA 1. ANÁLISIS DE LAS FASES EN EL DESARROLLO DE UN PROGRAMA. ELEMENTOS DEL LENGUAJE

Índice

Introducción

La principal finalidad de la programación es la de resolver un problema, de cualquier índole, mediante un sistema informático. Para ello lo primero que se debe realizar es un estudio detallado del problema. Este estudio detallado tendrá como resultado unas especificaciones que vendrán expresadas en forma de pseudocódigo, ordinograma, o cualquier otro medio de representación. Estos medios de representación son muy útiles pero presentan el inconveniente de que no son directamente comprensibles por el sistema informático.

Un sistema informático es en último término un conjunto de dispositivos físicos que sólo comprenden señales de tipo eléctrico (ausencia / presencia de tensión). El objetivo de la programación es el de convertir la solución obtenida mediante los anteriores medios de representación a una representación comprensible por el sistema informático.

En un principio la solución consistió en asignar a un estado eléctrico el valor 0 y al otro el valor 1. De esta forma se obtuvo un alfabeto binario que si bien era difícil de manejar permitía cierto grado de programación. Utilizando ese alfabeto binario se desarrolló el código máquina que consistía en una serie de instrucciones compuestas de ceros y unos que realizaban ciertas operaciones.

Como la programación mediante código máquina era muy compleja pues había que saberse que secuencia de ceros y unos correspondía con cada operación lo que provocaba muchos errores, se desarrolló un sistema que asignaba nemotécnicos a cada una de las instrucciones en código máquina (Ej.: MOV para mover algo). Esto es lo que se conoce como lenguaje ensamblador. En lenguaje ensamblador la correspondencia entre instrucciones en código máquina e instrucciones en código ensamblador es de una a una, lo que hace que la ejecución de los programas escritos en ensamblador sea muy rápida. La conversión entre el código fuente en ensamblador y el código máquina se realizaba mediante traductores.

El lenguaje ensamblador aportó muchas ventajas pero seguía siendo un lenguaje muy complicado para la mayoría de las personas. La siguiente evolución de los lenguajes de programación trató de acercar más al lenguaje natural humano las instrucciones que eran necesarias para realizar un programa. En esta época surgieron los lenguajes de alto nivel entre los que se encuentran C, Cobol, Pascal, y Basic. Los lenguajes de alto nivel tienen la ventaja de que son más intuitivos y por lo tanto facilitan la programación pero presentan los inconvenientes de que los programas escritos en estos lenguajes no son ejecutables directamente por la máquina, y de que una instrucción de estos lenguajes se corresponde con varias instrucciones de código máquina, lo que hace que su rendimiento no sea tan bueno.

Para poder ejecutar un programa fuente escrito en lenguaje de alto nivel deberemos usar un compilador o un intérprete. Un compilador convierte todo el código fuente en código objeto, mientras que un intérprete va convirtiendo el código fuente en código objeto por partes. El código objeto no es de por sí ejecutable, para poder ejecutarlo deberemos utilizar un enlazador (linker). El enlazador coge los ficheros objeto que compondrán el programa final y los enlaza convirtiendo ese código objeto en código máquina.

Java

Java es un lenguaje de alto nivel Orientado a Objetos que puede ser interpretado o compilado según las necesidades.

Su principal ventaja es que es independiente de la máquina por lo que una misma aplicación Java puede ejecutarse en varios dispositivos.

Lo único que necesitan las aplicaciones Java para ejecutarse correctamente sea cual sea el dispositivo es que dicho dispositivo tenga instalada su Java Virtual Machine (JVM) correspondiente.

Esto es así porque las aplicaciones Java producen en vez de código máquina ejecutable directamente por el dispositivo un bytecode (código java) que tiene que ser ejecutado por una Java Virtual Machine.

Esto hace que el código Java no sea muy eficiente pero permite desarrollar aplicaciones multiplataforma de manera sencilla. Queda en mano del desarrollador la decisión de usar Java o no a la hora de desarrollar aplicaciones.

Algoritmos. Pseudocódigo

En la vida real se nos presentan un sinfín de situaciones en las que nos vemos obligados a resolver algún tipo de problema. La solución a estos problemas la conseguimos siguiendo una serie de pasos determinados. La mayoría de las veces, sobre todo en situaciones que se producen a menudo, seguimos esos pasos sin percatarnos de que lo estamos haciendo. El motivo es que ya tenemos almacenada en nuestro cerebro la secuencia de acciones necesaria para resolver ese tipo de problemas.

Un ejemplo típico de problema que se nos presenta a menudo es calentar el desayuno en el microondas. Aunque aparentemente pueda parecer algo sencillo, debido a que normalmente estamos familiarizados con el uso del microondas, para una persona que nunca ha visto uno su uso puede representar un problema. Algunos aspectos que para muchos son obvios como por ejemplo la elección de la potencia correcta (500 W, 600 W, 700 W, ...), la selección del tiempo justo para calentar el desayuno, el cuidado de no introducir objetos metálicos en el interior,..., son totalmente desconocidos para esa persona.

Esa persona deberá tratar el problema de calentarse el desayuno con un microondas como un nuevo problema y no nos le quedará más remedio que intentar encontrar la secuencia de pasos necesaria para poder calentar su desayuno correctamente.

A toda secuencia de acciones que se utiliza para resolver un problema se la denomina **algoritmo**. Así bien, un algoritmo es la secuencia de acciones necesaria para resolver un determinado problema.

Supongamos ahora que somos esa persona que quiere calentarse el desayuno en el microondas. El primer paso que deberemos dar será el de asegurarnos de que contamos con los componentes necesarios. Para ello nos haremos la siguiente pregunta ¿Qué necesitamos para poder calentar nuestro desayuno en el microondas? A la que responderemos que, entre otras cosas, necesitaremos: un microondas, corriente eléctrica, una taza de desayuno no metálica, y lo que vayamos a calentar (leche, café, cola cao, ...).

En este primer paso lo que hemos hecho es una detección de las **necesidades previas**. Si cuando vamos a resolver un problema nos percatamos de que no se cumple alguna de las necesidades previas deberemos posponer la resolución del mismo hasta que éstas estén plenamente cubiertas.

Una vez que las necesidades previas están cubiertas deberemos determinar la **secuencia de pasos** correcta o algoritmo. Una secuencia es un conjunto de acciones o pasos que siguen un orden determinado. Esta es la parte más importante ya que un pequeño cambio de orden en los pasos puede provocar una solución incorrecta. Por ejemplo, imaginemos que el usuario pone en marcha el microondas antes de introducir la taza, ambas acciones forman parte de la secuencia de pasos a seguir pero ese orden no es el correcto.

Una solución aceptable al problema de calentarse el desayuno en el microondas sería:

Necesidades previas: Microondas, corriente eléctrica, taza de desayuno no metálica, lo que vayamos a calentar (leche, café, cola cao, ...)

Secuencia de pasos o Algoritmo:

1. Comprobar si hay corriente eléctrica.
2. Enchufar el microondas a la corriente eléctrica (Si no lo está)
3. Verter lo que vayamos a calentar en la taza de desayuno no metálica.
4. Abrir la puerta del microondas.
5. Meter la taza de desayuno no metálica en el microondas.
6. Cerrar la puerta del microondas.
7. Seleccionar la potencia deseada (Por ejemplo 500 W).
8. Seleccionar el tiempo deseado (Por ejemplo 2 minutos).
9. Poner en marcha el microondas.
10. Esperar a que pase el tiempo deseado.
11. Abrir la puerta del microondas.
12. Sacar la taza de desayuno no metálica caliente.
13. Desayunar.

Como práctica vamos a intentar encontrar el algoritmo que resuelve cada una de las siguientes situaciones de la vida cotidiana:

- Preparar una pizza en el horno.
- Hacer una llamada de teléfono.
- Lavarse las manos con jabón.
- Preparar unos espaguetis.
- Programar el vídeo para grabar un programa de la tele.

Variables

A la hora de ejecutar una aplicación o un programa podemos distinguir dos partes fundamentales, una que se mantiene siempre igual y que corresponde a las **instrucciones** que ejecuta la aplicación, o sea, a su código y otra que cambia de una ejecución a otra y que corresponde a los **datos** que usa la aplicación en esa ejecución en concreto.
Los datos que necesita la aplicación para ejecutarse correctamente se deben especificar a la hora de crear la aplicación.
Al especificar un dato se reserva en **memoria** el espacio necesario para guardar sus valores.
Esos datos que necesita la aplicación para ejecutarse correctamente reciben el nombre de **variables**.
Para crear una variable debemos darle un nombre válido, ese nombre es el **identificador** de la variable y sirve para acceder al valor de la variable.
También debemos especificar el **tipo** de valores que va almacenar la variable (numérico entero, numérico real, cadena de caracteres, carácter, booleano, …)
En algunos casos deberemos definir también el **ámbito** o parte del código desde dónde se puede acceder al valor de la variable.
Cuando se declara una variable el compilador reserva el espacio necesario para guardar un dato del tipo de la variable en memoria y asocia la dirección de memoria en que se encuentra ese espacio con el identificador, de tal forma que a partir de ese momento podemos utilizar el identificador para acceder al valor guardado en esa zona de memoria.

Ejemplo
a <- 0 pseudocódigo, PSeInt
a = 0; Java

En una variable podemos distinguir dos partes:

Valor. Es el contenido de la variable. Se accede a él mediante el identificador de la variable. Se utiliza para asignar un valor a la variable, o para obtener el valor de la variable

Dirección. Es la dirección de memoria donde se encuentra la variable. Se utiliza normalmente para el paso por referencia.

Ámbito. Las variables se pueden utilizar sólo dentro de su ámbito. Este ámbito depende del modificador de ámbito que se haya utilizado al declarar la variable (Public, Private, Protected, …) y de dónde se haya declarado la variable.

Constantes

Se refieren a los valores fijos que no pueden ser modificados por el programa. Pueden ser de cualquier tipo de datos básicos. Las constantes de **carácter** van encerradas en **comillas simples(')**, y las constantes de **cadena de caracteres** entre **comillas dobles(")**. Las constantes **enteras** se especifican con números sin parte decimal y las **reales** con su parte entera separada por un **punto** (.) de su parte **decimal**.

En Java las constantes son variables con estado **Final** que significa que ese estado no puede cambiar.

Tipos De Datos

Para poderse ejecutar correctamente en los dispositivos las aplicaciones necesitan datos. Esos datos deben de especificarse a la hora de desarrollar la aplicación. Para ello se usan normalmente variables cuya misión es almacenar en memoria el valor de esos datos.

Según el tipo de aplicación necesitará datos de un tipo u otro.

Los tipos de datos más utilizados son:

Enteros. Son datos numéricos sin decimales. Ejemplos: 5, 0, -1. Para definir una variable de tipo entero y de nombre N debemos escribir

```
ENTERO N              pseudocódigo
Definir N Como Entero PSeInt
int N;                Java – entero simple
long N;               Java – entero de mayor rango
```

Reales. Son datos numéricos con decimales. La parte entera está separada por un punto (.) de su parte decimal. Ejemplos: 5.3, 0.73, -23.8346. Para definir una variable de tipo real y de nombre R debemos escribir

```
REAL R                pseudocódigo
Definir R Como Real   PSeInt
float R;              Java – precisión simple
double R;             Java – mayor precisión
```

Carácter. Almacenan un único carácter. Las constantes de tipo carácter van entre comillas simples ('). Ejemplo: 'A', 'B', 'C', … Para definir una variable de tipo carácter y de nombre C debemos escribir

```
CARACTER C            pseudocódigo
Definir car Como Carácter PSeInt
char C;               Java
```

Lógicos o Booleanos. Son datos que sirven para comprobar si una expresión es verdadera o falsa. Sólo tienen dos valores posibles Verdadero (True) y Falso (False). Para definir una variable de tipo booleano y de nombre B debemos escribir

```
BOOLEANA B            pseudocódigo
Definir B como Logica PSeInt
boolean B;            Java
```

Cadenas de Caracteres o Strings. Almacenan un grupo o cadena de caracteres. Las constantes de tipo cadena de caracteres van entre comillas dobles ("). Ejemplo: "Hola Mundo". Para definir una variable de tipo cadena de caracteres y de nombre S debemos escribir

```
CADENA S              pseudocódigo
Definir S como Cadena PSeInt
String S;             Java
```

Sentencias

En Java todas las sentencias sencillas deben terminar con el carácter punto y coma (;).

Operador De Asignación

La operación de Asignación sirve para dar un valor a una variable. A la hora de asignar un valor a una variable el tipo del valor a asignar y el tipo de la variable deben de ser iguales o compatible. Por ejemplo, para asignar el valor 5 a la variable entera N que hemos definido antes debemos escribir

```
N ← 5   pseudocódigo, PSeInt
N = 5;  Java
```

Operadores Aritméticos

Los Operadores Aritméticos sirven para realizar operaciones aritméticas con los datos.

OPERADOR	*DESCRIPCIÓN*	*ORDEN*
-	Resta.	**3**
+	Suma	**3**
*****	Multiplicación	**2**
/	División	**2**

% (MOD en pseudocódigo)	Resto de una división entre números enteros	**2**
-	Cambio de signo (unitario).	**2**
--	Decremento en 1.	**1**
++	Incrementa en 1.	**1**

Operadores Lógicos y Relacionales

Los Operadores Lógicos y Relacionales sirven para realizar comparaciones de expresiones.

OPERADORES RELACIONALES		
OPERADOR	***DESCRIPCIÓN***	***ORDEN***
<	Menor que.	**5**
>	Mayor que.	**5**
<=	Menor o igual.	**5**
>=	Mayor o igual	**5**
= =	Igual	**6**
! =	Distinto	**6**

OPERADORES LÓGICOS		
OPERADOR	***DESCRIPCIÓN / PSEUDOCODIGO***	***ORDEN***
&&	Y (AND)	**10**
| |	O (OR)	**11**
!	NO (NOT)	**1**

Entrada y Salida De Datos. Pseudocódigo. PSeInt

La representación algorítmica mediante pseudocódigo de la entrada y salida estándar se realiza mediante las órdenes **LEER** y **ESCRIBIR**.
En PSeInt se utiliza la misma sintaxis.
Por ejemplo, si queremos leer un número N por teclado y después mostrarlo por pantalla mediante pseudocódigo debemos escribir

```
LEER N
ESCRIBIR N
```

Salida de Datos. Concatenación de Valores. PSeInt

Cuando queremos escribir el valor de una o varias variables acompañado por un texto tenemos que convertir el valor de la variable a texto y unirlo con el otro texto.
A esto se le conoce como concatenación de textos.
Para concatenar valores de variables y textos en PSeInt debemos poner los textos entre comillas dobles (") y separar los textos de los valores de las variables usando **comas** (,).

Por ejemplo, para escribir en PSeInt una línea que contenga un mensaje y el valor de la variable N escribimos
ESCRIBIR "El valor de la variable es ", N

Ejercicios de Entrada y Salida de Datos. Pseudocódigo. PSeInt

1. Realiza el pseudocódigo para el programa LEERNUM que lee un número entero por teclado y lo muestra por pantalla.

2. Realiza el pseudocódigo para el programa LEERNUM1 que lee un número entero por teclado, le suma 1, y lo muestra por pantalla.

3. Realiza el pseudocódigo para el programa DOBLENUM que lee un número entero por teclado, calcula su doble, y lo muestra por pantalla.

4. Realiza el pseudocódigo para el programa SIGNONUM que lee un número entero por teclado, lo cambia de signo, y lo muestra por pantalla

5. Realiza el pseudocódigo para el programa LONCIRCU que lee el valor del radio por teclado y muestra el valor de la circunferencia de ese radio por pantalla.

6. Realiza el pseudocódigo para el programa AREACIRC que lee el valor del radio por teclado y muestra el valor del área de un círculo de ese radio por pantalla.

7. Realiza el pseudocódigo para el programa CUBO que lee un número por teclado, calcula el cubo de ese número y muestra el resultado por pantalla.

8. Realiza el pseudocódigo para el programa CAPITAL que pide una cantidad en euros, un tipo de interés (ej: 3,6 %), y un período de tiempo expresado en días por pantalla y calcula el interés producido en ese tiempo en base a la fórmula
   Interes=(Cantidad*TipoInteres*Tiempo)/(360*100)

Estructura de una Clase Java

Una clase Java se define de la siguiente manera

```
public class Hola {
}
```

Dónde **Hola** es el nombre de la clase, **class** es la palabra Java para definir una clase y **public** es la palabra Java que debemos usar para permitir que la clase se pueda ejecutar desde el exterior (por ejemplo desde el Editor de Comandos).

Método main de una Clase Java

El método **main** de una clase Java es el método por el que comienza la ejecución de dicha clase. Si una clase Java no dispone de método main no se ejecutará.

El método main de una clase Java se define de la siguiente manera

```
public static void main(String[] args) {
}
```

Dónde **main** es el nombre del método, **void** indica que el método no devuelve ningún valor, **static** indica que sólo hay un método main para todos los objetos de la misma clase (comparten su contenido), **public** indica que se puede llamar al método para que se ejecute desde fuera de la clase. Lo que aparece entre paréntesis **(String[] args)** son datos que necesita el método para poder ejecutarse correctamente. En el caso del método main esos datos que se pasan a través del argumento de nombre **args** y de tipo array de cadenas de caracteres (**String[]**), se introducen como parámetros al ejecutar la clase desde el Editor de Comandos.

Salida de Datos en Java

La salida de datos en Java se realiza a través de clases predefinidas. Para la **salida estándar** de datos que normalmente consiste en sacar datos por pantalla se usan métodos de la clase **System.out**. Por ejemplo, para sacar el mensaje "Hola." en una línea debemos usar el código **System.out.println**("Hola.");

Si queremos que al terminar de escribir el texto el cursor no salte a la línea siguiente, es decir se quede justo a continuación del texto, debemos usar la función **System.out.print**. Por ejemplo, para sacar el mensaje "Hola." por pantalla sin que el cursor salte de línea debemos usar el código **System.out.print**("Hola.");

9. Crea la clase Java **Hola,** que muestra por pantalla el mensaje "Hola.", usando un editor de textos. Usa el Editor de Comandos para Compilar y Ejecutar la clase Java. El contenido de **Hola.java** es

```
public class Hola {

    public static void main(String[] args) {
        System.out.println("Hola");
    }

}
```

Ejecutar una Clase Java desde la Línea de Comandos

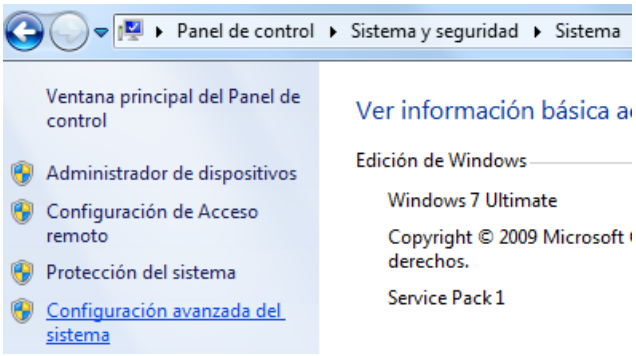
Para poder compilar (usando javac) y ejecutar una clase Java (usando java) desde la línea de comandos debemos instalar la versión **JDK** de Java. Hemos elegido una versión del JDK de **64 bits** porque la versión de Eclipse que vamos a usar es de 64 bits. La versión que hemos descargado es la **Windows x64** que aparece en http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Una vez instalado el JDK, para facilitar la compilación y ejecución de Java debemos incluir la ruta de la instalación del JDK de Java en las variable del sistema PATH.

Para ellos comprobamos el contenido de la variable del sistema Path. Para ello abrimos una ventana del Editor de Comandos e introducimos el comando **echo %Path%**

Dentro de la variable Path puede aparecer la cadena **C:\ProgramData\Oracle\Java\javapath** que permite el acceso al JRE de Java (que no tiene el compilador de Java javac).

Para tener acceso al compilador de Java javac tenemos que incluir en el Path la ruta del JDK. Para ello vamos a **Panel de Control -> Sistema y Seguridad -> Sistema** y allí en la ventana de la izquierda elegir **Configuración Avanzada del Sistema**.



En la ventana que aparece (Propiedades del Sistema), en la pestaña **Opciones Avanzadas**, hacemos clic sobre el botón **Variables de entorno …**

Buscamos en Variables del Sistema la variable Path. La seleccionamos y pulsamos Editar. Añadimos al final del texto que tengamos un punto y coma (;) y la ruta donde se ha instalado el JDK de Java en el Sistema **(;C:\Program Files\Java\jdk1.8.0_102\bin**) y pulsamos Aceptar.

Para comprobar que Java está correctamente instalado abrimos una ventana del Editor de Comandos e introducimos el comando **java -version**

Para comprobar que tenemos acceso al compilador de Java javac abrimos una ventana del Editor de Comandos e introducimos el comando **javac -version**

Podemos evitar el paso de Editar las variables del sistema si usamos rutas completas a la hora de compilar. Aunque en principio sea más lioso como sólo vamos a compilar usando la interfaz de comandos este ejemplo para nosotros es más sencillo.

Para compilar la clase Java Hola, una vez actualizada la variable de Sistema Path, abrimos el interfaz de comando y vamos a la carpeta donde esté la clase Java Hola.java.

Al haber añadido a la variable de Sistema Path la ruta de Java, para compilar basta con poner **javac Hola.java** y para ejecutar basta con poner **java Hola**

Si todo ha ido bien en la carpeta dónde está Hola.java se habrá creado la clase Java ejecutable **Hola.class**.

Para ejecutarla introducimos el comando **java Hola**

Eclipse. Instalación

Antes de realizar nuestro primer programa en Java en Eclipse vamos a instalarlo. Eclipse es el IDE que vamos a utilizar para desarrollar en Java.

Si existe una versión previa de Eclipse instalada la desinstalamos. Además, si existe alguna versión de Java instalada en el equipo también la desinstalamos antes de comenzar la instalación de Eclipse.

La versión que vamos a utilizar es **Eclipse IDE for Java Developers Windows 64-bit**. La última versión de Eclipse la podemos descargar desde:

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

La versión que vamos a usar es la **eclipse-java-2023-06-R-win32-x86_64**

Una vez descargada descomprimimos el contenido del fichero zip en la carpeta **c:\eclipse**. Es muy importante que esté en esta carpeta para evitar problemas.

Una vez descomprimido en esta carpeta, creamos un **acceso directo** a eclipse.exe en el Escritorio.

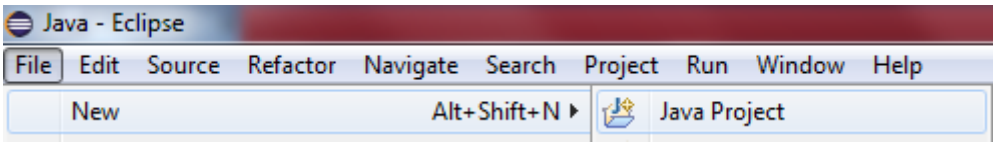
Importante. Para que Eclipse funcione correctamente las versiones de Eclipse y de Java instaladas en el sistema deben de coincidir. Es decir, como nosotros vamos a utilizar **Eclipse IDE for Java Developers Windows 64-bit** debemos usar una versión Java de 64 bits.

Tras crear el acceso directo lo ejecutamos.

Si todo va bien nos aparecerá una ventana en la que se nos pide que creemos un **Workspace**. Para evitar problemas vamos a crear el Workspace en la unidad de **DATOS** (D: en mi caso) dentro de la carpeta **PROG** (si no existe la creamos) en una carpeta de nombre **eclipse**.

Una vez dentro de Eclipse ya podemos empezar a trabajar.

Cerramos la pestaña de Bienvenida (Welcome) y, con todo el Workspace visible, vamos a **File -> New -> Java Project** para crear un nuevo proyecto Java.

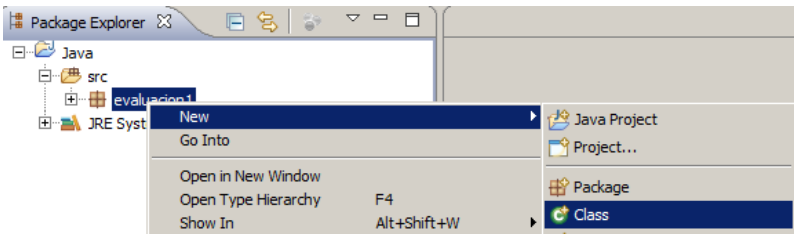


Ponemos como nombre del proyecto **Java** y lo guardamos en la ubicación por defecto que debe de ser **X:\PROG\eclipse\Java** (dónde X es la unidad de DATOS de nuestro equipo, la unidad D en mi caso) y pulsamos Finish para terminar el proceso de creación del proyecto.

Desplegamos el contenido del proyecto haciendo clic con el botón izquierdo en el icono situado a la izquierda del nombre del proyecto Java y hacemos **clic con el botón derecho** sobre la carpeta **src** y seleccionamos New -> **Package** para crear un nuevo paquete donde guardar los ejercicios de la primera evaluación que va a tener de nombre **evaluacion1**.

Después hacemos **clic con el botón derecho** sobre el paquete evaluacion1 y seleccionamos New -> **Class** para crear una **nueva clase Java**.

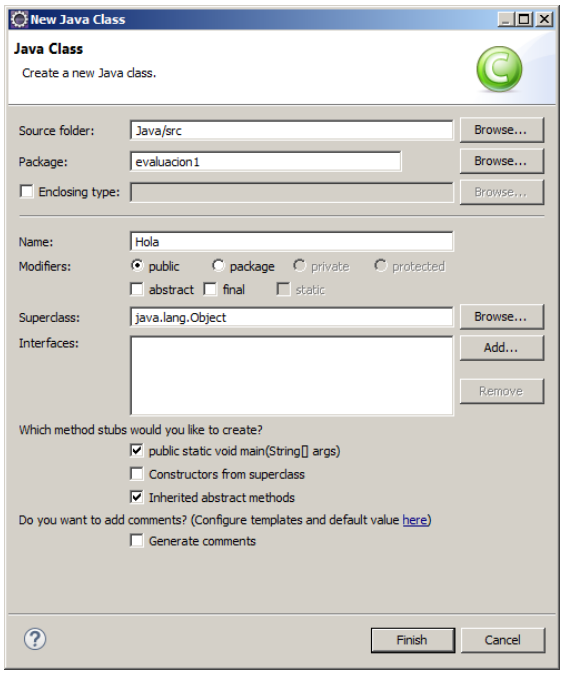
Para ello en el menú que aparece vamos a **New -> Class**



Como name (nombre) de la clase ponemos **Hola**.

Marcamos la casilla **public static void main(String [] args**) para que nos cree ese método automáticamente ya que es el método en el que tenemos que escribir el código que queremos ejecutar.

Pulsamos Finish para terminar de crear la clase Hola.



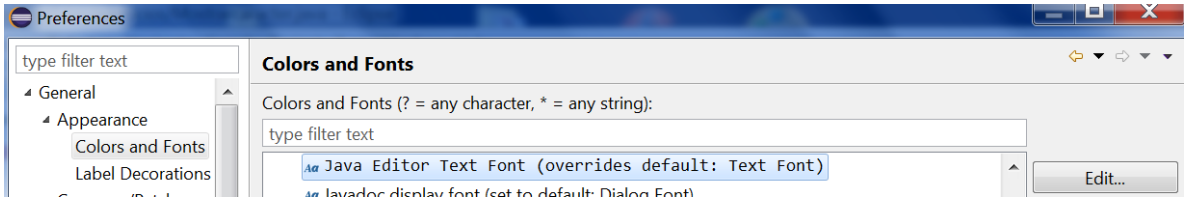
En este punto ya tenemos la clase Hola preparada trabajar con ella.

10. Realiza en Eclipse la clase Java **Holamundo** que muestra por pantalla el mensaje "Hola Mundo".

11. Ejecuta la clase Java **Holamundo** usando Eclipse.

Eclipse. Personalizar Editor

Para Personalizar el **tamaño de la fuente** debemos ir a **Window -> Preferences**. Una vez allí vamos a **General -> Appearance -> Colors and Fonts**, buscamos en la ventana de la derecha **Java - > Java Editor Text Font**, y pulsamos el botón **Edit…**
En la ventana que aparece seleccionamos el tamaño de fuente que queramos. Por ejemplo, elegimos un tamaño de 14.



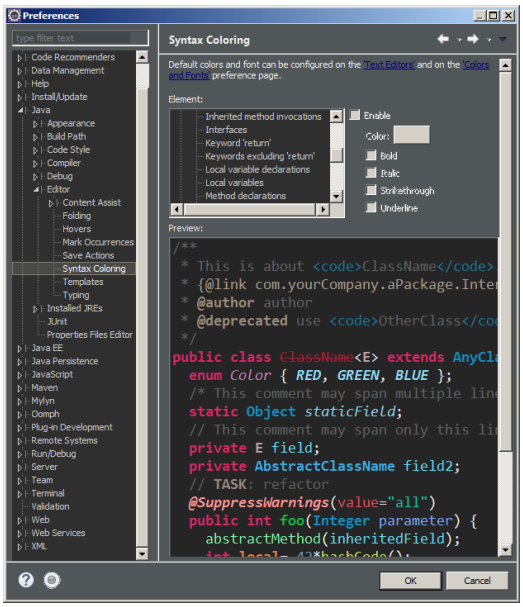
También podemos cambiar el tamaño del texto que se muestra en el Editor pulsando CTRL+SHIFT+'+' para hacer más grande o pulsando CTRL+SHIFT+'-' para hacer más pequeño.

Para personalizar el **tamaño de las tabulaciones** debemos ir a **Window -> Preferences**. Una vez allí vamos a **Java -> Code Style -> Formatter**, seleccionamos como perfil **Eclipse [built-in]** y pusamos el botón **Edit…**

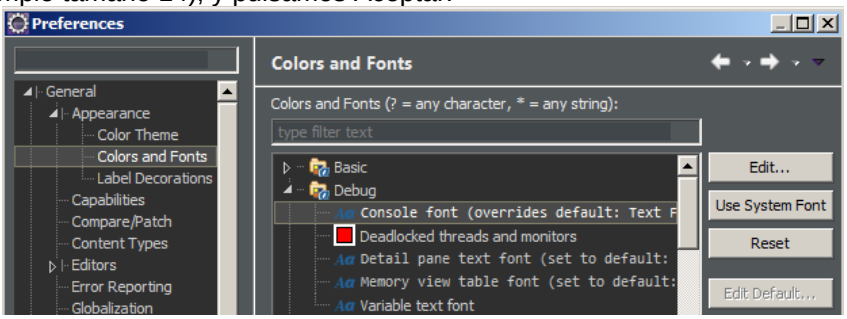
Cambiamos el nombre del perfil por **Eclipse - PROG**. En la pestaña **Indentation** ponemos como **Tab Size** el valor **2**.
Pulsamos el botón Apply, y comprobamos que el perfil activo del Formatter sea Eclipse – PROG y pulsamos el botón OK.

También podemos cambiar el tema de Eclipse. Para ellos vamos a **Window -> Preferences**. Una vez allí vamos **General -> Appearance** y allí seleccionamos el tema que queremos usar. Por ejemplo, el tema **Dark** para mitigar el efecto del brillo de la pantalla.

También podemos cambiar el color de los elementos del editor de Java. Por ejemplo, para cambiar el color de las palabras reservadas de Java vamos a **Window -> Preferences -> Java -> Editor -> Syntax Coloring** y seleccionamos en **Element Keyword 'return'** y **Keywords excluding 'return'**.



También podemos personalizar la fuente de la Consola. Para ello vamos a **Window -> Preferences**. Una vez allí vamos a **General -> Appearance -> Colors and Fonts**, buscamos en la ventana de la derecha **Debug**, seleccionamos Console Font, pulsamos el botón Edit…, seleccionamos las características que queramos (por ejemplo tamaño 14), y pulsamos Aceptar.



También podemos personalizar la apariencia de Eclipse instalando temas nuevos. Por ejemplo, podemos instalar el tema Moonrise para dar otra apariencia al entorno de Eclipse. Los pasos para hacerlo se encuentran en el enlace https://marketplace.eclipse.org/content/eclipse-moonrise-ui-theme

Salida de Datos. Concatenación de Valores. Java

Podemos usar el método **System.out.println** para mostrar por pantalla el valor de una o varias variables mezclado con uno o varios textos. Para ello usamos el operador de concatenación (**+**) dentro de los paréntesis.

Por ejemplo, para mostrar un mensaje y el valor de la variable N podemos escribir
System.out.println("El valor de la variable es " + N);

12. Realiza la clase Java Mostrarentero que coge el valor de una variable de tipo entero (**int**) y lo muestra por pantalla.

13. Realiza la clase Java Mostrarreal que coge el valor de una variable de tipo real con decimales (**double**) y lo muestra por pantalla.

14. Realiza la clase Java Mostrarcaracter que coge el valor de una variable de tipo caracter (**char**) y lo muestra por pantalla.

15. Realiza la clase Java Mostrarcadena que coge el valor de una variable de tipo **String** y lo muestra por pantalla.

Salida de Datos con Formato

Si queremos que los datos aparezcan por pantalla con un formato determinado, por ejemplo si queremos que un valor de tipo real aparezca con dos decimales, no podemos usar el método System.out.println, debemos usar el método **System.out.printf** cuya sintaxis es diferente.

El método printf usa unos identificadores de formato de los cuales los más usados son:

%d datos de tipo int
%f datos de tipo float
%f datos de tipo double
%c datos de tipo carácter
%s datos de tipo String

En función del tipo de datos se pueden especificar unas opciones u otras. Para simplificar vamos a ver sólo como especificar los decimales de los datos de tipo float o double. Para especificar que queremos que un dato de tipo double aparezca con 2 decimales debemos usar **%.2f**.

*System.out.printf("El valor de la variable es **%.2f",**n);*

16. Realiza la clase Java Mostarreal2 que coge el valor de una variable de tipo real con decimales y lo muestra por pantalla con 2 decimales.

17. Realiza la clase Java Mostrarformato que coge el valor de una variable de tipo String y el valor de una variable de tipo real y muestra por pantalla la cadena y la variable real con 2 decimales.

Entrada de Datos en Java

La **entrada estándar** de datos consiste, normalmente, en introducir datos por teclado. Para realizar la entrada de datos en Java se usan métodos de la clase **System.in** y de otras clases dedicadas.

Para facilitar la entrada de datos vamos a utilizar la clase **Scanner** que se encuentra dentro de la clase **java.util**. Esta clase no está incluida en las clases que se cargan por defecto por lo que para poderla utilizar la debemos **importar**. Para importarla debemos escribir la siguiente línea antes de la definición de la clase **import java.util.Scanner;**

Una vez importada la clase debemos debemos definir dentro de la función **main** un objeto de la clase Scanner. Por ejemplo, para definir un objeto de la clase **Scanner** y de nombre **teclado** debemos escribir el siguiente código

```
// defino un objeto de la clase Scanner
Scanner teclado;
// creo un nuevo objeto de la clase Scanner
teclado = new Scanner(System.in);

// tambien puedo escribirlo de la siguiente manera
Scanner teclado = new Scanner(System.in);
```

Al definir el nuevo objeto de tipo Scanner pasamos como parámetro el dispositivo de entrada estándar (**System.in**) porque en este caso vamos a leer los datos por teclado. En caso de querer los datos desde otro lugar (por ejemplo desde un fichero) lo indicaríamos allí.

Una vez finalizado el proceso de lectura de datos, si no vamos a usar más el objeto de tipo Scanner lo debemos cerrar. Por ejemplo, para cerrar un objeto de la clase **Scanner** y de nombre **teclado** debemos escribir el siguiente código

teclado.close();

Entrada de Datos de Tipo Entero

Para leer el valor de una variable de tipo **entero** mediante la clase **Scanner** se usa el método **nextInt()**. Por ejemplo, para leer el valor de una variable de tipo **entero** y de nombre **n** por teclado usando un objeto de la clase **Scanner** y de nombre **teclado** debemos escribir el siguiente código

n = teclado.nextInt();

18. Realiza la clase Java Leerentero que lee un número entero por teclado y muestra por pantalla el mensaje *"El valor de la variable introducida es "*.

Entrada de Datos de Tipo Real

Para leer el valor de una variable de tipo **real double** mediante la clase **Scanner** se usa el método **nextDouble()**. Por ejemplo, para leer el valor de una variable de tipo **double** y de nombre **d** por teclado usando un objeto de la clase **Scanner** y de nombre **teclado** debemos escribir el siguiente código

d = teclado.nextDouble();

A la hora de introducir datos de tipo real por teclado hay que tener mucho cuidado con el separador decimal ya que debemos usar el que tenga la Configuración Regional de nuestro equipo (**,** en nuestro caso) y no el propio de los datos de tipo real de Java (que es el punto). Por ejemplo, para usar el valor de pi debemos poner

3.14159	en la clase Java
3,14159	en la consola a la hora de introducir datos por teclado

19. Realiza la clase Java Leerreal que lee un número real por teclado y muestra por pantalla el mensaje *"El valor de la variable introducida es "*.

Entrada de Datos de Tipo String (Cadena de Caracteres)

Para leer el valor de una variable de tipo **String** (**cadena de caracteres)** mediante la clase **Scanner** se usa el método **next()**. Por ejemplo, para leer el valor de una variable de tipo **String** y de nombre **s** por teclado usando un objeto de la clase **Scanner** y de nombre **teclado** debemos escribir el siguiente código

s = teclado.next();

20. Realiza la clase Java Leerstring que lee un String por teclado y muestra por pantalla el mensaje *"El valor de la variable introducida es "*.

Para leer el valor de una variable de tipo **String** en cuyo interior aparece algún carácter separador (por ejemplo un espacio en blanco, un salto de línea, un tabulador,…) mediante la clase **Scanner** se usa el método **nextLine()**. Por ejemplo, para introducir el valor "Hola Mundo." en una variable de tipo **String** y de nombre **s** por teclado usando un objeto de la clase **Scanner** y de nombre **teclado** debemos escribir el siguiente código

s = teclado.nextLine();

21. Realiza la clase Java Leerstringcompuesto que lee un String por teclado y muestra por pantalla el mensaje *"El valor de la variable introducida es "*.

Entrada de Datos de Tipo Caracter

Para leer el valor de una variable de tipo **caracter** mediante la clase **Scanner** debemos leer un String completo y coger el primer carácter de ese String. Para leer un **String** se usa el método **nextLine()**. El inconveniente de usar nextLine es que si se pulsa enter directamente, sin introducir otro carácter, se produce una excepción que debemos controlar.
Podemos usar next pero no reconoce los espacios en blanco al inicio de la línea y si se pulsa enter pasa a la siguiente línea.
Por ejemplo, para leer el valor de una variable de tipo **char** y de nombre **c** por teclado usando un objeto de la clase **Scanner** y de nombre **teclado** debemos escribir el siguiente código

c = teclado.nextLine().charAt(0);

22. Realiza la clase Java Leercaracter que lee un caracter por teclado y muestra por pantalla el mensaje *"El valor de la variable introducida es "*.

Forzado de Tipo de Datos. Casting

Un forzado de tipo de datos o casting es una operación especial que nos permite realizar una conversión entre tipos, por ejemplo cuando queremos que un dato de tipo real se comporte como uno de tipo entero. Para ello se antepone al nombre de la variable que queremos convertir el tipo de datos al que lo queremos convertir entre paréntesis.

Por ejemplo, para hacer que el valor de la variable real R se comporte como una variable de tipo entero escribimos en pseudocódigo

```
REAL R
ENTERO E
E ← (ENTERO) R
```

Para hacer lo mismo en Java escribimos

```
double r;
int e;
e = (int) r;
```

En PSeInt no existe el operador de casting y para realizar un forzado de tipo de datos debemos usar la función **trunc**. Por ejemplo, para hacer que el valor de la variable real R se comporte como una variable de tipo entero en PSeInt escribimos

```
Definir R Como Real
Definir E Como Entero
E <- trunc(R)
```

Ejercicios de Entrada y Salida de Datos. Java

23. Realiza la clase Java Leernum que lee un número entero por teclado y lo muestra por pantalla.

24. Realiza la clase Java Leernum1 que lee un número entero por teclado, le suma 1, y lo muestra por pantalla.

25. Realiza la clase Java Doblenum que lee un número entero por teclado, calcula su doble, y lo muestra por pantalla.

26. Realiza la clase Java Signonum que lee un número entero por teclado, lo cambia de signo, y lo muestra por pantalla

27. Realiza la clase Java Loncircu que lee el valor del radio por teclado, calcula el valor de la circunferencia de ese radio, y muestra por pantalla el resultado. Para el cálculo podemos usar la constante Java ***Math.PI***.

28. Realiza la clase Java Areacirc que lee el valor del radio por teclado, calcula el área de un círculo de ese radio, y muestra por pantalla el resultado. Para el cálculo podemos usar la constante Java ***Math.PI***.

29. Realiza la clase Java Cubo que lee un número por teclado, calcula el cubo de ese número, y muestra por pantalla el resultado.

30. Realiza la clase Java Capital que lee por teclado una cantidad en euros, un tipo de interés (ej: 5,5%), y un período de tiempo expresado en días por pantalla y calcula el interés producido en ese tiempo en base a la fórmula ***Interes=(Cantidad*TipoInteres*Tiempo)/(360*100)*** Nota: Para mostrar el carácter '%' dentro de printf debemos escribir '**%%**'. Sino mostrará una excepción.

31. Realiza la clase Java EnteroReal que lee un número real por teclado y muestra por separado su parte entera y su parte real.

32. Realiza la clase Java CentimosEuro que lee una cantidad en euros por teclado y muestra por separado el número de euros y el número de céntimos de euro.