Introducción al uso de Java

Iván González Rincón,
Manuel Méndez Calvo, Juan Carlos Garrote Gascón

GUI
Grupo Universitario de Informática
Escuela de Ingeniería Informática, Universidad de Valladolid

Hour of Code, 2019
Miércoles, 27 de Noviembre de 2019

Origen de Java

- Tenía como objetivo televisores pero era demasiado avanzado.
- Se desarrolló con una sintaxis similar a C/C++ para hacerlo familiar.
- Java 1.0 lanzado en 1996 por Sun Microsystems.
- Noviembre de 2006, Java se declara Open Source y gratis.
- Oracle adquiere Sun Microsystems en 2009/10.

Figura: Logo de Java

¿Hacia adonde apunta Java?

- Java es un lenguaje concurrente, basado en clases y orientado a objetos.
- Utiliza punteros, pero nosotros no lo sabemos.
- Posee una de las mejores documentaciones, si no la mejor.
- En la implementación diferenciamos JRE y JDK.
- Utilizado en desarrollo Android.

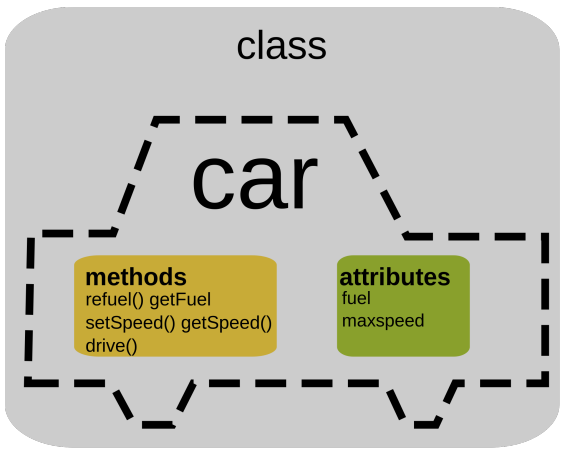

Figura: Breve idea sobre la Orientación a Objetos.

Sintaxis-Hola Mundo!

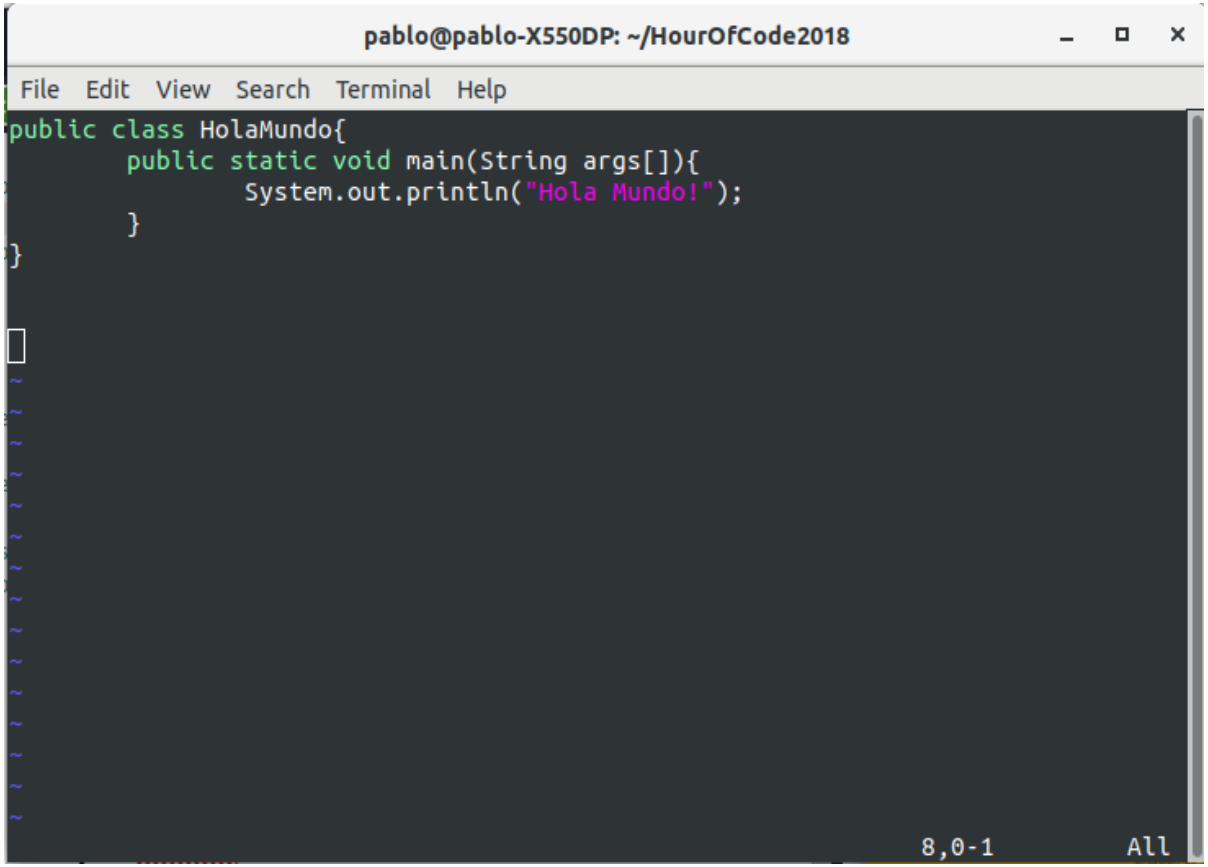

Figura: Hola Mundo en Java

Compilar y Ejecutar

Para poder ejecutar nuestro programa primeramente debemos compilarlo a través del compilador de Java:

Compilación

```
javac HolaMundo.java
```

Tras esto, pasamos a la ejecución:

Ejecución

```
java HolaMundo
```

¿Qué es una variable?

- Una variable es una asignación, nos permite darle un nombre a un dato.
- Las variables pueden inicializarse, o asignarse.
- Hay varios tipos básicos y compuestos de variables

- Los tipos básicos más utilizados
  - int
  - char
  - double
  - boolean
  - String (No es básico)

Inicialización y asignación de variables

```
// Podemos inicializar una variable sin asignarle un valor,
    para asignarselo mas tarde
int soyUnEntero;
// O podemos inicializarlo asignndole un valor directamente
double soyUnDouble=2.0;
// Y si quisieramos ahora podriamos asignarle un valor a la
    primera variable
soyUnEntero=1;
```

¿Cómo operar con variables?

- Los tipos básicos tienen operaciones como sumar, restar, multiplicar, comparar etc.
- Sólo podemos operar con variables del mismo tipo
- También podemos asignar a una variable que tiene un valor, el dato de otra variable (si son del mismo tipo, o de tipos compatibles).

```
int soyUnEntero=1;
int soyOtroEntero=1000000

// Esto es asignacion
soyOtroEntero=soyUnEntero

// Esto es una
    comparacion (que
    devolvera un booleano
    dependiendo de si se
    cumple o no, en este
    caso sera True)
soyOtroEntero==soyUnEntero
```

Métodos

Los métodos son fragmentos de código que definimos dentro de nuestro programa para poder utilizarlos con una llamada. Estos métodos podrán devolver (o no) un resultado, que podremos almacenar en una variable.
Los métodos pueden definirse tanto antes del *main*, como después del mismo, y puede utilizarse todas las veces que queramos y desde cualquier punto del código. Además los métodos pueden llamarse entre si.

```
public boolean comparaEnteros(int primero,int segundo){
    return(primero==segundo);
}

public static void main (String [] args){
    int soyUnEntero=2;
    int soyOtroEntero=5;

    boolean compara=comparaEnteros(soyUnEntero,soyOtroEntero);
}
```

Estructuras condicionales if/if-else

Podemos utilizar estructuras condicionales para controlar la ejecución nuestro código. Esto nos permite que un fragmento de código se ejecute si se dan unas condiciones impuestas por nosotros mismos.
Nota: Nos gustaría mencionar que hay estructuras más complejas pero no las vamos a ver en este taller inicial.

Estructuras condicionales if/if-else

```
char soyUnCaracter='a';
char soyOtroCaracter='b';

if(soyUnCaracter==soyOtroCaracter){
    System.out.println("somos iguales");
}else if(soyUnCaracter<soyOtroCaracter){
    System.out.println("el primer caracter es menor que el
        segundo");
}else{ // Este else no seria necesario
    System.out.println("el segundo caracter es menor que el
        primero");
}
```

switch

Switch es otra estructura de control, que nos permite gestionar en casos lo que queremos comprobar, el último caso deberá ser *default* y este se ejecutará siempre que el resto de condiciones haya fallado. Además después de cada caso tenemos que poner un *break* para salir del *switch*

switch código

switch

```
switch (variable) {
case valor_1:
case valor_2:
    // Ejecutar si el valor de variable es igual a valor_1 o a
        valor_2
    ejecutarA();
    break; // Salir del switch
case valor_3:
    // Ejecutar si el valor de variable es igual a valor_3
    ejecutarB();
    break; // Salir del switch
default:
    // Ejecutar si el valor de varible es distinto del del resto
        de casos
    ejecutarC();
    break; // Salir del switch
}
```

while y do while

El bucle while nos permite ejecutar un bucle mientras una condición sea cierta

```
int bucle=0;
while(bucle<=10){
  bucle +=1
}
```

El bucle do-while hace lo mismo, pero ejecuta minimo una vez el bucle

```
int bucle=0;

do{
  bucle +=1
}while(bucle<=10);
```

for

El bucle for es un bucle controlado por un iterador, resultando que antes de comenzar ya sabemos el límite de ejecuciones que tiene.

```
int j=100;
for(int i=0 ;i<=j ; i++){
    System.out.println("El contador se llega por el numero " +
        i);
}
```

Recursividad

Según Wikipedia recursividad es la forma en la cual se especifica un proceso basado en su propia definición". El factorial de un numero n o la serie de Fibonacci son buenos ejemplos de ello.
No se considera una buena practica, y debemos intentar evitarla siempre que podamos

Arrays

Un array puede compararse con un vector, es una asignación de celdas en las que podemos guardar informacion. Los arrays tienen tipo, y deben declararse e inicializarse como las variables. También podemos hacer operaciones con lo que hay guardado en sus celdas. Sin embargo, también podemos hacer operaciones como ver cuantas celdas tiene.
Se pueden declarar arrays de n dimensiones, pero en el caso que nos ocupa sólo vamos a ver hasta los de 2 dimensiones.

Ficheros

Java nos permite trabajar con ficheros, en este caso vamos a hablar de ficheros de texto. Aprovechando esto, introduciremos el bloque*try-catch*, que se usa para controlar posibles errores que surjan durante la ejecución del código. Para ello dentro del bloque catch pondremos las posibles excepciones que pueden haber, para que si surgen, conozcamos el error.
Para trabajar con los ficheros debemos abrirlos, operar con ellos y cerrarlos. Los ficheros pueden ser abiertos en modo lectura, o escritura, y podemos abrir más de un fichero para poder escribir cosas en uno, leer cosas en otro, o que interactúen entre ellos.

Estructura

```
try {
   ficheroLectura = new Scanner(new File("hola.txt"));

   // Cierre
   ficheroLectura.close();
}
// Debemos capturar la excepcion en caso de que se intente abrir
     un archivo que no existe
catch (FileNotFoundException e) {
   System.out.println("No he encontrado el fichero!");
}
```

Gracias por vuestra atención

Esperamos que el taller os haya resultado útil, si tenéis alguna cuestión, no dudéis en preguntar
Repositorio con código visto en el taller:
https://github.com/chivi94/Hour-Of-Code-19-20-Java