Introducción a Java

En esta lección introduciremos la estructura principal de un programa en Java, así como los métodos básicos para obtener entradas de información por parte del usuario y escribir los resultados obtenidos.
Como la sintaxis de Java se basa, indirectamente, en la de C, muchos aspectos os resultarán conocidos, por lo que veréis como sois capaces de entender y escribir programas Java desde el principio.
Además de presentar el lenguaje, introduciremos la biblioteca de la *ACM Java Task Force*[1] que está específicamente diseñada para simplificar algunos de los aspectos de Java y hacerlos más sencillos para facilitar su uso en los cursos introductorios de programación.
Enfatizaremos sobretodo las diferencias entre ambos lenguajes, que básicamente se circunscriben al ámbito de los vectores y matrices (*arrays*) y de las cadenas de caracteres.
Respecto de los ejemplos que veremos, algunos de ellos están programados de forma algo enrevesada para que aparezcan llamadas a las funciones o construcciones que se quieren mostrar.
Como es tradición en informática, empezaremos nuestra presentación de Java con el programa *Hola Mundo*.

[1] http://www-cs-faculty.stanford.edu/~eroberts/jtf/index.html

1. "Hola mundo" en Java

```
1 /* HelloProgram.java
2  * ----------------
3  * This program displays "hello, world" on the
4  * screen.
5  * It is inspired by the first program in Brian
6  * Kernighan and Dennis Ritchie's classic book,
7  * The C Programming Language.
8  */
9
10 import acm.program.ConsoleProgram;
11
12 public class HelloProgram extends ConsoleProgram {
13
14   public void run() {
15     println("hello, world");
16   }
17
18 }
```

Comentarios

Todo texto entre las marcas /* y */, incluso cuando ocupa varias líneas, es ignorado por el compilador y, por tanto, puede usarse para poner comentarios. También existe la posibilidad de usar el marcador // para comentarios que solamente se extienden hasta el final de la línea. Posteriormente veremos el marcador especial /** para indicar documentación en formato *javadoc*, que permite generar archivos html con la documentación de nuestras clases y funciones.

Importaciones

Cuando en un fichero Java usamos clases definidas fuera de él, hemos de indicar al compilador que importe la definición de dichas clases. Para ello usaremos la sentencia `import`, en la que indicaremos también la clase que deseamos importar.
Las clases en java se agrupan en paquetes, de modo parecido a los directorios y ficheros[2]. Por ejemplo, la clase

[2] Más adelante veremos que, para que todo funcione, la estructura de paquetes y clases ha de coincidir con la estructura de directorios y ficheros. Es decir, en el caso anterior, el código de la clase `ConsoleProgram` estaría en `acm/program/ConsoleProgram`

`acm.program.ConsoleProgram` es la clase `ConsoleProgram` que está definida en el paquete `acm.program`.
Si en un fichero usamos varias clases de un mismo paquete podemos incluirlas todas en una sola sentencia `import`. Por ejemplo, para incluir todas las clases del paquete `acm.program`, podemos hacer

```
import acm.program.*;
```

Es por ello que, de cara a que la clase `HelloProgram` del ejemplo pueda importarse desde cualquier fichero java, se ha mantener una correspondencia entre el nombre de la clase `HelloProgram` y el nombre del fichero java dónde se ha programado, en este caso, `HelloProgram.java`.

Tipo de programa

En Java las unidades principales en la que se dividen los programas son las clases, que se usan como plantillas para crear objetos, por lo que para definir un programa tendremos que definir una clase y, dentro de esa clase, una función que sea el punto de arranque de nuestro programa[3].
En la asignatura utilizaremos la biblioteca de clases Java definida por la ACM para simplificar algunas tareas que, con la biblioteca de clases estándar, resultan arduas y tediosas. Por ello, la clase que defina el punto inicial de ejecución (que en este caso se llama `run`), debe extender a una de las clases del paquete `acm.program`.
En concreto, el paquete incluye cuatro tipos diferentes de programas predefinidos, que se diferencian en la forma en cómo el usuario interactúa con ellos:

- `CommandLineProgram`: se interacciona con el programa de forma similar a C, es decir, mediante la línea de comandos. Por ejemplo, la llamada a println en el programa anterior, escribe el texto citado en la consola. Normalmente no lo usaremos.
- `ConsoleProgram`: parecido al anterior, pero el programa despliega una ventana que se convierte en la consola de la aplicación. Al acabarse el programa, la ventana desaparece.
- `DialogProgram`: se interacciona mediante ventanas emergentes (*pop-ups*) tanto para la entrada de datos, como para la salida de resultados. Por ejemplo, si en el ejemplo `HelloProgram`

[3] En C la unidad principal son las funciones y, para definir el punto inicial de ejecución del programa, tan solo teníamos que definir la función `main`

extendiera `DialogProgram`, el mensaje no se escribiría en la consola, sino que aparecería una ventana con él.
- `GraphicsProgram`: la interacción se hace mediante elementos gráficos, pudiéndose dibujar con el ratón, arrastrar y soltar, etc.

Para indicar el estilo de interacción que deseamos para nuestro programa utilizaremos la directiva `extends` cuando definamos la clase para nuestro programa:

```
public class HelloProgram extends ConsoleProgram
```

Convenciones de nombres

En Java los nombres de las clases se escriben en letras minúsculas, excepto la inicial que va en mayúscula y, si el nombre consiste en la concatenación de varias palabras, simplemente se concatenan, manteniendo las mayúsculas para cada una de las palabras. Por ejemplo, `console + program` da lugar a `ConsoleProgram`.
Es importante respetar las convenciones en cuanto a los nombres ya que así, a simple vista, solamente viendo el nombre sabremos si se refiere a una clase, a una variable o función, o a una constante. Por ejemplo, si tenemos la clase alumno programador, la denominaríamos como `AlumnoProgramador`

¿Qué es esto de `public`?

El formato de las funciones[4] en Java es similar al de las funciones en C. La única diferencia que apreciamos es que delante de la definición de la función (y también de la de la clase) aparece la palabra `public`.
De momento, para simplificar, la pondremos siempre tanto delante de la definición de la clase y como de todas las funciones. Más adelante veremos qué otras cosas pueden ponerse y para qué.

[4] Técnicamente, y como se verá más adelante, son métodos pero de momento continuaremos con la nomenclatura que ya conocemos de C.

2. Raíz entera de un número natural

```
1 /*
2  * File: SquareRoot.java
3  * ---------------------
4  * This program calculates the square root of a
5  * given positive integer
6  */
7
8 import acm.program.ConsoleProgram;
9
10 public class SquareRoot extends ConsoleProgram {
11
12   public int squareRoot(int n) {
13     int lower = 0;
14     while ((lower + 1) * (lower + 1) <= n) {
15       lower = lower + 1;
16     }
17     return lower;
18   }
19
20   public void run() {
21     int n = readInt("Enter a natural number: ");
22     int root = squareRoot(n);
23     println("The root is " + root);
24   }
25 }
```

Función "principal"

La estructura de la función principal es típica:

- Pedir datos al usuario
- Calcular, usando una función auxiliar
- Mostrar el resultado

En este caso, como necesitábamos un entero, hemos usado la función `readInt`[5], que muestra un mensaje al usuario (indicativo de lo que se pide) y que devuelve el valor que el usuario ha entrado, en este caso un entero.

[5] Como siempre todo suele ser más complicado que lo que explicamos inicialmente ya que hay varias versiones de `readInt`, que permiten variaciones sobre el tema de pedir al usuario un número entero (p.e. determinar el rango válido).

Existen también funciones `readBoolean`, `readDouble` y `readLine`, para los tipos boolean, double (coma flotante) y cadena de caracteres (String)[6].

Funciones auxiliares

Dentro de la clase podemos definir también funciones auxiliares como la que, en este caso, realiza el cálculo. Como hemos dicho anteriormente, precederemos su definición con la palabra `public`.
En Java, para las funciones la convención es la misma que para las clases salvo que la letra inicial del nombre completo va en minúsculas. Es decir, `read+int` da lugar a `readInt`.

Declaración de variables

En Java, podemos realizar en una misma instrucción la declaración de una variable y su inicialización. El resultado es el mismo que declarar primero la variable para luego asignarle un valor, es decir:

```
int lower = 0;
```

es equivalente a:

```
int lower;
lower = 0;
```

La convención para nombrar variables es la misma que para las funciones auxiliares.

Combinación de cadenas

La sentencia

```
println("The root is " + root);
```

combina varios aspectos de tratamiento de cadenas que es necesario indicar. Para entender qué hace, primero pondremos su equivalente en C, que sería:

```
printf("The root is %d\n", root);
```

La sentencia del `println` requiere tres cosas:

- Como lo primero que se encuentra es una cadena de caracteres, se considera que + es la concatenación de cadenas (la interpretación de + como suma de enteros es imposible).
- Se convierte `root` que es un entero a cadena de caracteres, para poder concatenar
- Se obtiene la cadena resultado de la concatenación.
- Se escoge la versión adecuada de `println` (en este caso la de cadenas de caracteres).

[6] El tema de `Strings` se verá más adelante.

Otra forma de obtener un resultado similar sería:

```
print(“The root is ”);
println(root);
```

3. Suma elementos de un vector

```
1 /* ArraySum.java
2  * -------------
3  * This programs fills an array of máximum size of
4  * 10 and sums its contents until the first zero is
5  * found
6  */
7
8 import acm.program.ConsoleProgram;
9
10 public class ArraySum extends ConsoleProgram {
11
12   public int MAX = 10;
13
14   public  void run() {
15     int[] numbers = readArray();
16     int sum = sumArray(numbers);
17     println("The array sum is " + sum);
18   }
19
20   public int[] readArray() {
21     int[] nums = new int[MAX];
22     int i = 0;
23     int num;
24
25     do {
26       num = readInt("Enter an integer (0 to end): ");
27       nums[i] = num;
28       i = i + 1;
29     } while ( num!=0 && i < nums.length );
30
31     return nums;
32   }
33
34   public int sumArray(int[] numbers) {
35     int sum = 0;
36     int i = 0;
37
38     while (i < numbers.length && numbers[i] != 0) {
39       sum += numbers[i];
40       i = i + 1;
41     }
42
43     return sum;
```

```
44    }
45 }
```

Definición de "constantes"

De la misma manera que podemos definir funciones auxiliares, podemos definir constantes que luego usaremos en cualquiera de las funciones. Su objetivo es similar al que en C lográbamos con `#define MAX 10`

Se definen como si fueran variables locales pero, en vez de hacerlo dentro de una función, se definen fuera de ella (pero dentro de la clase).

Técnicamente **no son constantes** y más tarde ya veremos la sintaxis para conseguir que lo sean.

Eso sí, seguiremos la convención de definir sus nombres en letras mayúsculas y, en caso de ser un nombre compuesto, va todo en mayúsculas con los componentes separados por el carácter '_' (guión bajo). Es decir, en caso de ser `max+int` el nombre que usaríamos para la constante sería `MAX_INT`.

Declaración de un vector

Aquí observamos una de las principales diferencias respecto de C, pero que una vez entendida, veremos que es para bien y simplificará algunas cosas que en C eran ligeramente *complicadas*.

Dentro del ejemplo hemos hecho

```
int[] nums = new int[MAX];
```

que, como hemos visto antes es lo mismo que

```
int[] nums;          // Declaración
nums = new int[MAX]; // Inicialización
```

que, en esta segunda forma nos será más fácil de explicar.

- La primera línea declara la variable `nums` y dice que su tipo es `int[]`, es decir, es una variable que se referirá a un vector de enteros (de la misma manera que una declaración `int i;` nos dice que i se referirá a un número entero).
- La segunda línea crea un vector de `MAX` posiciones (numeradas desde `0` hasta `MAX-1`) cada una de las cuales es un número entero y lo asigna a la variable `nums`.

Pero la diferencia más importante es que un vector en Java *recuerda* el número máximo de elementos con el que se inicializó. En el caso del ejemplo, `nums.length` devuelve `10`.

Funciones que devuelven vectores

De la misma manera que podemos hacer funciones que retornen tipos simples (como enteros), podemos hacer funciones que retornen un vector. El tipo de retorno de la función es int[], es decir, vector de enteros.
Fijaros que dentro de la función se declara una variable local (`nums`) y se crea en vector el vector (con la instrucción new) pero fuera de la función solamente se declara una variable para referirse al valor retornado (`numbers`) y le asigna el valor retornado por la función.

Funciones que reciben vectores

Las funciones también pueden recibir vectores como parámetros. Simplemente hace falta definir el tipo de parámetro como vector, int[] en el ejemplo.
En el caso de los vectores, el paso de parámetros de hace **por referencia**, es decir, si dentro de la función hacemos modificaciones sobre el vector que hemos pasado como parámetro, dichas modificaciones persisten al salir de la función.
En el caso de los **tipos primitivos** de Java:

- `boolean` : Puede contener los valores *true* o *false*.
- `byte` : Enteros. Tamaño 8-bits. Valores entre -128 y 127.
- `short` : Enteros. Tamaño 16-bits. Entre -32768 y 32767.
- `int` : Enteros. Tamaño 32-bits. Entre -2147483648 y 2147483647.
- `long` : Enteros. Tamaño 64-bits. Entre -9223372036854775808 y 9223372036854775807.
- `float` : Números en coma flotante. Tamaño 32-bits.
- `double` : Números en coma flotante. Tamaño 64-bits.
- `char` : Caracteres. Tamaño 16-bits. Unicode. Desde '\u0000' a '\uffff' inclusive. Esto es desde 0 a 65535

el paso de parámetros es **por valor**, es decir, aunque modifiquemos los parámetros dentro de la función, dichos cambios **no** afectan a las variables originales.

4. Un ejemplo con matrices

```
 1 import acm.program.ConsoleProgram;
 2
 3 public class MatrixTranspose extends ConsoleProgram {
 4
 5   public void transpose(int[][] input,
 6                         int[][] output) {
 7
 8     for (int i=0; i<input.length; ++i) {
 9       for (int j=0; j<input[i].length; ++j) {
10         output[j][i] = input[i][j];
11       }
12     }
13   }
14
15   public void print(int[][] matrix) {
16     print("{");
17     for (int i = 0; i<matrix.length; ++i) {
18       print("{");
19       for (int j=0; j<matrix[i].length; ++j) {
20         print(matrix[i][j]);
21         if ( j != matrix[i].length-1) {
22           print(", ");
23         }
24       }
25       print("}");
26       if ( i != matrix.length-1) {
27         print(", ");
28       }
29     }
30     println("}");
31   }
32
33   public  void run() {
34     int[][] source = {{1, 2, 3}, {4, 5, 6}};
35     int[][] destination = new int[3][2];
36     transpose(source, destination);
37     print(destination);
38   }
39 }
```

Declaración de las matrices

Las matrices son, simplemente, vectores de vectores. Para declararlo, se añaden tantas parejas de [] como dimensiones tenga la matriz (usualmente 2). Por ello, las matrices de enteros se declaran como

```
int[][] source;
```

Inicialización de matrices

La forma más simple de inicialización es similar a la que hemos visto en el caso de los vectores: indicar el máximo número de elementos en cada dimensión. En el ejemplo:

```
destination = new int[3][2];
```

que crea una matriz de 3 filas (numeradas de 0 a 2) i dos columnas (numeradas 0 y 1).

La otra forma de inicialización, que también puede usarse con vectores, consiste en indicar, en el caso de matrices bidimensionales, los elementos que contiene. En el caso que nos ocupa, se indican los vectores que constituyen cada una de las filas, por tanto:

```
source = {{1, 2, 3}, {4, 5, 6}}
```

representa la matriz

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

De hecho, los mismo puede hacerse con vectores. Por ejemplo, si quisiéramos inicializar un vector de caracteres con las letras vocales, podemos hacer:

```
char[] vocals = {‘a’, ‘e’, ‘i’, ‘o’, ‘u’}
```

Tamaño de la matriz

Como una matriz no es más que un vector que contiene vectores, si hacemos `matriz.length` obtenemos el número de vectores, es decir, el número de filas. Para obtener el de columnas, hemos de ver el tamaño de uno de los vectores que hay en cada fila, por tanto, `matriz[0].length`

5. Ejemplo básico con cadenas de caracteres

```
 1 import acm.program.ConsoleProgram;
 2
 3 public class UsingStrings extends ConsoleProgram {
 4
 5   public boolean isVocal(char c) {
 6     String vocals = "AEIOUaeiou";
 7     char[] vocalsChars = vocals.toCharArray();
 8     int i = 0;
 9     while ( i < vocalsChars.length &&
10             vocalsChars[i] != c ) {
11       i = i + 1;
12     }
13     return  i < vocalsChars.length ;
14   }
15
16   public String removeVocals(String str) {
17     char[] resultChars = new char[str.length()];
18     int resultLength = 0;
19     for (int i=0; i<str.length(); i++) {
20       char current = str.charAt(i);
21       if ( ! isVocal(current) ) {
22         resultChars[resultLength] = current;
23         resultLength = resultLength + 1;
24       }
25     }
26     return new String(resultChars, 0, resultLength);
27   }
28
29   public  void run() {
30     String sentence = readLine("Enter a sentence: ");
31     print("The sentence w/o vocals is: ");
32     println(removeVocals(sentence));
33   }
34 }
```

Diferencias fundamentales entre las cadenas de C y los Strings de Java

En primer lugar debemos recordar que en C no existe un tipo especial para las cadenas de caracteres sino que éste se representa como un vector de caracteres acabado en el carácter especial `'\0'`.
En Java existe un tipo especial para representar cadenas de caracteres que es la clase `String`. Dicha clase nos permitirá hacer algunas de las

cosas que hacíamos en C (entre muchas otras). Además, siempre podremos, a partir de un `String`, obtener un vector con los caracteres que contiene, y tratarlo de forma parecida a como haríamos en C; y, a partir de un vector de caracteres, obtener el `String` equivalente[7].

Dado un String obtener el char[]

Si lo que queremos es obtener un vector con todos los caracteres que contiene un `String`, simplemente hemos de hacer

```
vocalsChars = vocals.toCharArray();
```

De esta manera transformaremos la cadena "AEIOUaeiou" en el vector `{'A', 'E', 'I', 'O', 'U', 'a', 'e', 'i', 'o', 'u'}`.
Dos cosas a resaltar: la primera, que **no** se trata de una invocación a una función pasando alguna cosa como parámetro, `toCharArray(vocals)`; y segundo, el vector no contiene el carácter `'\0'` al final. La sintaxis consistente en escribir la cadena `vocals` seguida de un punto y algo que parece una llamada a función (esta sintaxis forma parte de la **programación orientada a objetos**). De momento tendremos que memorizar su uso y, más adelante, ya veremos el sentido que tiene hacerlo de esta manera. Respecto del carácter `'\0'` en Java no es necesario ya que podemos usar `vocalChars.length` para saber que el vector tiene diez caracteres.

Acceso a los caracteres que contiene un String

Si queremos acceder a los caracteres de un `String` sin necesidad de pasarlo a vector de caracteres podemos hacer

```
current = vocals.charAt(i);
```

dónde el parámetro `i` va desde cero hasta el número de caracteres -1
Por cierto, para saber el número de caracteres de una cadena podemos hacer

```
vocals.length();
```

Un aspecto un poco lioso es que en el caso de los vectores se pone directamente `length`, por ejemplo, `vocalsChars.length`; pero en el caso de un `String` se han de colocar paréntesis al final, es decir, `vocals.length()`.

[7] Por supuesto que también podremos tratar directamente los Strings sin necesidad de convertirlos en vector de caracteres.

Podéis pensar en que un `char[]` ya tiene los corchetes y su `Length` no necesita los paréntesis, pero un `String` no los tiene y, por ello, su `Length()` los necesita[8].

Dado un char[] obtener el String

Esta operación viene a ser la inversa de `toCharArrary`. El problema radica en que hay que indicar, además del vector en el que se encuentran los caracteres que deberán formar el `String`, el intervalo de posiciones de éstos dentro del vector. Ello es debido a que, por ejemplo, hemos reservado más espacio del necesario. Por ejemplo, en el programa, hemos reservado `str.length()` caracteres (ya que potencialmente la cadena podría consistir en todo no vocales) pero hemos encontrado `resultLength` no vocales. Los dos parámetros a pasar son:

- el primer elemento válido del vector (en nuestro caso 0)
- el número de caracteres que queremos tenga el String (en nuestro caso `resultLength`) ya que son válidas las posiciones desde la `0` hasta la `resultLength-1`.

6. Lecturas adicionales

- Artículo de la Wikipedia, Java (lenguaje de programación)
- Capítulos 1 y 2 del libro "*The Art and Science of Java (Preliminary Draft)" de Eric S. Roberts.* Lo tenéis disponible en sakai.

[8] Técnicamente en el primer caso (array) se trata de una propiedad y en el segundo caso (String) de una llamada a un método.