Curso de programación Java II

Artículo publicado originalmente en la revista Sólo Programadores

En el número anterior presentamos brevemente qué es la plataforma Java y en qué consisten cada una de las tres ediciones de la plataforma (Java ME, Java SE y Java EE). Vimos cuáles eran los componentes básicos del kit de desarrollo Java y aprendimos a manejarlos desde una consola. Finalmente, presentamos cuáles son los tipos de datos primitivos y los operadores aritméticos, relacionales y lógicos básicos del lenguaje. En esta segunda entrega de este curso de programación Java presentaremos brevemente la librería matemática, introduciremos nuevos tipos de datos no primitivos como cadenas de caracteres, arrays y enumeraciones, y presentaremos las estructuras de control de flujo (bucles y condicionales) del lenguaje. Para trabajar con los códigos de ejemplo de este artículo os recomiendo que sigáis usando el JDK para familiarizaros con él. En el número siguiente comenzaremos a usar un entorno de desarrollo.

La librería matemática: la clase Math

La mayor parte de las funciones matemáticas básicas (raíces cuadradas, exponenciación, logaritmos, senos, cosenos, senos hiperbólicos, etc.) están disponibles en Java a través de la clase Math. Hasta el siguiente número de este curso de programación Java no presentaremos qué son las clases y cómo construirlas. No obstante, no es necesario tener conocimientos de programación orientada a objetos para emplear esta clase: todos sus métodos son estáticos, esto es, se comportan como funciones. Por tanto, podemos invocarlos directamente sin necesidad de crear ningún objeto de la clase. La sintaxis de estos métodos es:

```
Math.metodo(argumentos);
```

Donde metodo es el nombre del método concreto que queremos invocar. Por ejemplo, para realizar un seno emplearíamos Math.sin (ángulo) , y para calcular x^y en emplearíamos Math.pow (x,y) . En estos momentos probablemente te estés realizando preguntas como ¿Tengo que saber de memoria todos los métodos de la clase Math ? ¿Y también tengo que saber el orden en el que se le pasan los argumentos? y... ¿a Math.sin (ángulo) el ángulo se lo debo de pasar en grados o en radianes?. Tranquilo, no es necesario saberse de memoria los métodos, ni los argumentos. Tampoco tienes que saberte de memoria si los ángulos se pasan en grados o en radianes. Aunque con la práctica acabarás sabiendo la respuesta a muchas de estas preguntas, no creo que tenga ningún sentido que te estudies la clase Math de memoria.

Java tiene unas librerías extensas (extensísimas). Ningún desarrollador Java, por más gurú que sea, se las sabe de memoria. Pero lo que sí sabemos hacer los gurús es buscar rápidamente aquellas cosas que no sabemos en el javadoc de las librerías. Las librerías Java están formadas principalmente por código

fuente Java. El javadoc de estas librerías no es más que el resultado de aplicar la herramienta javadoc, que presentamos en el primer artículo de esta serie, a dicho código fuente.

Todo programador Java que se precie debe tener siempre a mano el javadoc de la librería estándar. Este javadoc puede descargarse desde la misma página de descarga donde se obtiene el JDK de Sun. Si el lector se ha instalado Netbeans, el entorno de desarrollo tiene esta documentación integrada y puede accederse poniendo el cursor sobre la clase o método del cual se quiera consultar el javadoc y a continuación pulsando Alt + F1. Esta documentación también está disponible online en http://java.sun.com/javase/6/docs/api/, aunque personalmente prefiero tenerla descargada en mi equipo ya que así puedo navegar de un modo más rápido través de ella y no dependo de tener conectividad para poder consultarla.

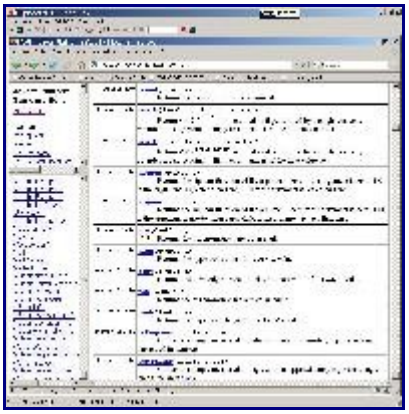

FIGURA 1: El javadoc de la librería estándar es una herramienta indispensable para cualquier programador Java.

Como podemos ver en la figura 1, en el recuadro situado en la esquina inferior izquierda de la página principal de la documentación hay un listado, ordenado alfabéticamente, con todas las clases que componen la documentación de la librería estándar de Java. Tras hacer clic en Math vemos la documentación de esta clase. Una de las primeras cosas que nos encontraremos es un listado de los métodos de la clase, junto con una descripción corta de su funcionalidad. Cada método es un enlace que nos lleva a una descripción más detallada, situada más abajo en la misma página web.

Volviendo a nuestra pregunta inicial ¿a Math.sin (ángulo) el ángulo se lo debo pasar en grados o en radianes?. No tengo intención de responderte. Antes de seguir leyendo este artículo tú mismo deberías responder a esa pregunta consultando la documentación de la clase Math. Es muy importante acostumbrarse a usar el javadoc de la librería estándar; es un recurso esencial para cualquier programador Java. En el código del listado 1 podemos ver un programa donde se emplean varios de los métodos de la clase Math .

```
//LISTADO 1: Este código muestra el funcionamiento de la clase Math

//código Ejemplo6.java del CD
```

```
int i = 45, j=2;
System.out.println ("Cos i : " + Math.cos(i));//coseno
System.out.println ("Sen i : " + Math.sin(i));//seno
System.out.println ("j^i : " + Math.pow(j,i)); //exponenciacion
System.out.println ("sqrt(j) : " + Math.sqrt(j)); //raiz cuadrada
System.out.println (" N&uacute;mero aleatorio : " + Math.random());//generacion de un numero aleator
io
System.out.println ("Valor absoluto: " + Math.abs(-5));//valor absoluto
```

Cadenas de caracteres: la clase String

Para representar cadenas de caracteres en Java se emplea una clase, String , que es la que proporciona soporte para las operaciones que se realizan más comúnmente con cadenas de caracteres. La definición de un String es similar a la de un tipo primitivo. Las cadenas de caracteres que vayan incluidas en el código Java como literales deben ir rodeadas por comillas dobles:

```
String e ;   //declarado pero no inicializado
String e ="";   //cadena vacia
String e = "Hola";  //inicializacion y asignacion
```

La concatenación de dos cadenas de caracteres en Java es muy sencilla: se realiza con el operador + , es decir “sumando” las cadenas de caracteres. Lo ilustraremos con un ejemplo:

```
String saludo = "hola";
String nombre = "Pepe";
String saludaPepe = "";
saludaPepe = saludo + nombre;// saludaPepe toma el valor "holaPepe"
```

Si una cadena la intentamos concatenar con otro tipo de variable automáticamente se convierte la otra variable a String (en el caso de que la otra variable sea un objeto para realizar la conversión se invocará el método toString () de dicho objeto), de tal modo que en Java es perfectamente válido el siguiente código:

```
String saludo = "hola";
int  n = 5;
saludo = saludo + “"" + n;// saludo toma el valor "hola 5"
```

Algunos métodos útiles de la clase String

A diferencia de C, las operaciones comunes sobre cadenas de caracteres, como la comparación o la extracción de una subcadena, no se realizan a través de funciones (o métodos estáticos), sino que se realizan a través de métodos de la propia clase String . Por ejemplo, en la clase String hay un método que permite la extracción de una subcadena de caracteres de otra. Su sintaxis es:

cadena.substring(int posiciónInicial, int posiciónFinal);

donde posiciónInicial y posiciónFinal son, respectivamente, la posición del primer carácter que se desea extraer y del primer carácter que ya no se desea extraer. Veamos un ejemplo de uso:

```
String saludo = "hola";
String subsaludo = saludo.substring(0,2);// subsaludo toma el valor "ho"
```

También puede extraerse un char de una cadena; para ello se emplea el método charAt(posición) , siendo posición la posición del carácter que se desea extraer. Se empieza a contar en 0.Para comparar cadenas de caracteres no puede emplearse el operador == . Este operador realiza una comparación de identidad, es decir, nos permitiría comprobar si dos objetos String son realmente un mismo objeto. Pero no nos permite comprobar si dos objetos String diferentes contienen el mismo texto. El segundo caso es el que se corresponde con la semántica de "igualdad" que habitualmente empleamos los seres humanos: no nos importa si son el mismo objeto, sino si su contenido es el mismo. Para comparar dos cadenas de caracteres debe emplearse otro método de String: equals. Su sintaxis es:

```
cadena1.equals(cadena2);
```

el método devuelve true si las dos cadenas son iguales y false si son distintas.¿Es posible pasar una cadena de caracteres a minúsculas o a mayúsculas ? ¿y reemplazar un carácter por otro? ¿O encontrar la primera ocurrencia de un carácter de una cadena?. La respuesta es sí, todas estas operaciones, y muchas otras, son posibles. ¿Cómo se realizan?. Empleando métodos de la clase String . ¿Qué métodos?. A esta pregunta, ya me niego a responder. Recomiendo al lector que antes de continuar leyendo el artículo consulte el javadoc de la clase String y se familiarice con las operaciones que soporta. En el listado 2 vemos un código que demuestra varias operaciones que se realizan con Strings. Este código también permite mostrar como en Java se distingue entre mayúsculas y minúsculas, de tal modo que "Hola" no es la misma cadena que caracteres que "hola".

```
//LISTADO 2: Operaciones frecuentes con cadenas de caracteres
```

```
//código Ejemplo7.java del CD

  String saludo = "Hola";
  String saludo2 ="hola";
  int n = 5;
//Imprime por consola la subcadena formada por los caracteres

//comprendidos entre el caractero 0 de saludo y hasta el  carácter 2

  System.out.println( saludo.substring(0,2));
//ejemplo de concatenacion

  System.out.println( saludo +" " + n);
//Imprime el resultado del test de igualdad entre saludo y saludo2.

  System.out.println( "saludo == saludo2 "+ saludo.equals(saludo2));
```

Enumeraciones

Esta característica del lenguaje sólo está disponible en Java 5 y superior. Si estás utilizando Java 1.4.X o inferior no podrás emplearla. La versión actual de Java es la 6.

Los tipos de datos enumerados son un tipo de dato definido por el programador (no como ocurre con los tipos de datos primitivos). Sirven para representar variables que toman valor en un conjunto finito y, normalmente, pequeño de datos. Algunos ejemplos podrían ser los palos de la baraja española, o los días de la semana. Podríamos representar un palo de la baraja mediante un número entero; por ejemplo, podríamos asignar el 1 a los oros, el 2 a los bastos, el 3 a las copas y el 4 a las espadas. Esta estrategia tiene dos problemas: si una variable tipo int es usada para representar un palo de la baraja, nada me impide asignarle el valor "-143", aunque no tenga ningún sentido. El segundo problema es que dentro de un mes no me voy a acordar qué palo representaba el 2, o si comencé a enumerar en 0 o en 1.

Las enumeraciones proporcionan una solución elegante a este problema e incrementan la legibilidad del programa. Para definirlas el programador debe indicar un conjunto de valores finitos sobre los cuales las variables de tipo enumeración deberán tomar valor. La mejor forma de comprender qué son es viéndolo; para definir un tipo de dato enumerado se emplea la sintaxis:

```
modificadores enum NombreTipoEnumerado{ VALOR1,VALOR2,.. }
```

Los posibles valores de modificadores serán vistos en el siguiente capítulo de esta serie. El caso más habitual, y el que emplearemos por la de ahora, es que modificadores tome el valor public . El nombre puede ser cualquier nombre válido dentro de Java. Entre las llaves se ponen los posibles valores que podrán tomar las variables de tipo enumeración, valores que habitualmente se escriben en letras mayúsculas. Un ejemplo de enumeración podría ser:

public enum Semana {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO}

Las definiciones de los tipos enumerados deben realizarse fuera del método main y, en general, fuera de cualquier método; es decir, deben realizarse directamente dentro del cuerpo de la clase. Para definir una variable de la anterior enumeración se emplearía la siguiente sintaxis:

```
Semana variable;
```

y para darle un valor a las variables de tipo enumeración éstas deben asignarse a uno de los valores creados en su definición. El nombre del valor debe ir precedido del nombre de la propia enumeración:

variable = Semana.DOMINGO;

A diferencia de las enumeraciones de C o C++, las enumeraciones de Java son "typesafe", esto es, es imposible asignarle una variable de una enumeración un valor distinto de aquellos que se declararon al definir la enumeración. En C las enumeraciones internamente se representan como números enteros, por lo que nada nos impide asignarle a una variable de una enumeración un valor entero fuera del rango de las constantes simbólicas que se emplearon en su definición. En Java cada uno de los valores de las enumeraciones se representa mediante una instancia de la clase de la enumeración y el compilador garantiza que cada vez que se inicialice una enumeración se inicializa obligatoriamente a uno de los valores que se declararon en su definición, ya que estos valores son todas las instancias que existen de dicha clase.

En el listado 3 podemos ver un ejemplo de uso de enumeraciones.

```
//LISTADO 3: Ejemplo de uso de enumeraciones.

//codigo Ejemplo8.java del CD

public enum Semana {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO};
```

```
   public static void main(String[] args) {
          Semana hoy = Semana.MARTES;

//observa como gracias a la enumeración el programa es mas facil de leer

          if(hoy == Semana.DOMINGO  || hoy == Semana.SABADO){

        System.out.println("Hoy toca descansar");
          } else{

        System.out.println("Hoy toca trabajar");
          }

    }
```

Arrays

Un array es una colección de variables del mismo tipo, que tienen un nombre común y están dispuestos en posiciones consecutivas de la memoria del ordenador. Cada elemento del array se distingue de los demás por su índice (número de orden dentro del array). Un array se caracteriza por su tipo base (el tipo de cada elemento), el número de dimensiones del mismo y la longitud en cada dimensión.

En Java los arrays son un objeto. Como tales se crean mediante el operador de new (en el siguiente capítulo veremos el uso de este operador de un modo detallado). Quitando esa diferencia, la forma de crear el array y la forma de trabajar con ellos es idéntica a la forma en la que se trabaja con los arrays en C. La sintaxis en la definición de un array de una dimensión es la siguiente:

```
Tipo_datos[] nombreArray = new Tipo_datos[tamañoArray];
```

Tipo_datos es el tipo de los datos que se almacenarán en el array (int , char , String ... o cualquier objeto). TamañoArray es el tamaño que le queremos dar a este array. Por ejemplo, la sentencia:

int[] array = new int[10];

crea un array llamado array , en el que podremos almacenar 10 datos tipo entero. Para crear una matriz, esto es, un array de dos dimensiones, emplearemos el siguiente código:

int[][] matriz = new int[3][9];

Como se muestra en la figura 2, el primer elemento de un array se sitúa en la posición 0, exactamente igual que en C. Una vez definido, para acceder a los elementos de un array usaremos el nombre del array y a continuación, entre corchetes, el índice del elemento al cual queremos acceder. Por ejemplo, la instrucción:

```
array[5] = array[0] + array[3];
```

hace que la sexta posición del array tome el valor de la suma de la primera posición del array más la cuarta (una vez más, hagamos énfasis en que se comienza a contar en cero). En el caso de las matrices, o poliedros en general, deberemos emplear un índice por cada dimensión del poliedro.

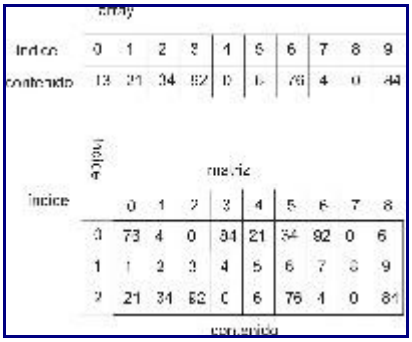

FIGURA 2: En Java el primer índice de todas las dimensiones de un array comienza a contar en 0.

El listado 4 muestra un código donde se inicializa un array con números aleatorios entre 0 y 80 y a continuación se calcula la suma de dichos números.

```
//LISTADO 4: Codigo de ejemplo que muestra el uso de arrays en Java.

//codigo Ejemplo9.java del CD

    int[] edades = new int[10];
   for(int i= 0; i< 10; i++){
  edades[i] = (int)Math.random()% 80;
            System.out.println("Elemento" + i + edades[i]);
```

```
            }
```

```
int sum = 0;
   for(int i= 0; i< 10; i++){
    sum  = sum + edades[i];

   }
```

```
System.out.println("Suma " + sum);
```

El control de flujo en Java

El modo de ejecución de un programa Java en ausencia de elementos de control de flujo es secuencial, es decir, las instrucciones se van ejecutando una detrás de otra, se ejecutan todas, y sólo se ejecutan una vez. Esto nos permite hacer programas muy limitados; para evitarlo Java emplea la misma solución que emplean la inmensa mayoría de los lenguajes de programación: estructuras de control de flujo. Estas estructuras de control de flujo se dividen en dos grupos: los condicionales, que permiten ejecutar o no un bloque de código dependiendo de que se cumpla una determinada condición; y los bucles, que permiten repetir la ejecución de un bloque de código mientras se cumpla una determinada condición.

Aquellos lectores familiarizados con C comprobarán cómo el 90% de la sintaxis de Java referente a estructuras de flujo es idéntica a la de este lenguaje.

Estructuras de control de flujo condicionales

Las estructuras de control de flujo de tipo condicional ejecutan un código u otro en función de que se cumpla o no una determinada condición. En Java existen dos tipos de condicional: if y switch .

Condicional tipo if

La sintaxis más sencilla para el condicional if es:

```
if(condicion) {

Grupo de sentencias;}
```

condicion tiene que ser un valor tipo boolean ; este valor puede obtenerse evaluando una expresión indicada por condición, pero el valor de evaluar dicha expresión tiene que ser obligatoriamente un boolean (a diferencia de C, no se pueden emplear números enteros como condiciones). El grupo de sentencias que se encuentra entre llaves se ejecuta sólo si la condición toma un valor true . En caso contrario se ignora el grupo de sentencias. Si el condicional afecta a una única sentencia las llaves son opcionales.

Un if puede ir acompañado de un bloque else :

```
if(condicion) {

Grupo de sentencias;}

else{

Grupo2 de sentencias;}
```

Si condicion toma el valor true se ejecuta Grupo de sentencias , en caso contrario se ejecuta Grupo2 de sentencias . Entre el if y el else no puede ir ninguna sentencia.

Es posible anidar varios condicionales tipo if :

```
if(condicion) {

Grupo de sentencias;}

else if (condicion2){

Grupo2 de sentencias;}

else if (condicion3){

Grupo3 de sentencias;}

...

else{

Grupo_n de sentencias;}
```

Si condicion toma el valor true se ejecuta Grupo de sentencias y se ignoran todos los demás condicionales y el else . En caso contrario, si condicion2 toma el valor true se ejecuta Grupo2 de sentencias ... y así sucesivamente hasta acabarse todas las condiciones. Si no se cumple ninguna se ejecuta el grupo de sentencias asociadas con el else . Este último else es opcional.

En el listado 5 mostramos un ejemplo de uso de condicional tipo if . Este es el primer ejemplo donde usamos otro método además del main , el método static int test(int val, int val2) . Un método estático es equivalente a una función; es decir, puede invocarse directamente sin necesidad de crear ninguna instancia de la clase. El nombre del método es test y recibe dos parámetros (val y val2) de tipo entero. El método devolverá como resultado de su ejecución un valor entero; dicho valor debe indicarse empleando la sentencia al return . En el siguiente artículo de esta serie presentaremos en detalle la sintaxis para declarar métodos en Java.

El método test es invocado varias veces desde el main . Este método devuelve -1 si el primer argumento es menor que el segundo, +1 si el primer argumento es mayor que el segundo y 0 si ambos son iguales.

```
//LISTADO 5: Ejemplo de uso del condicional tipo if.

//codigo Ejemplo8.java del CD

 static int test(int val, int val2) {
  int result = 0;
    if(val > val2)

      result = +1;

    else if(val < val2)

      result = -1;

    else

      result = 0;

    return result;

  }

 public static void main(String[] args) {
  System.out.println(test(10, 5));
  System.out.println(test(5, 10));
  System.out.println(test(5, 5));
  }
```

Switch

Esta estructura de control de flujo permite elegir entre varios bloques de código cuáles se deben ejecutar y cuáles no. Veamos su sintaxis:

```
switch(selector) {

        case valor1 : Grupo de sentencias1; break;

        case valor2 : Grupo de sentencias2; break;

        case valor3 : Grupo de sentencias3; break;

        case valor4 : Grupo de sentencias4; break;

        case valor5 : Grupo de sentencias5; break;

                // ...

        default: statement;

}
```

al ejecutarse la sentencia switch compara el valor de selector con valorX . Si el valor coincide se ejecuta su respectivo grupo de sentencias. Si no se encuentra ninguna coincidencia se ejecutan las sentencias del default . Si no se pusieran los break una vez hubiese una coincidencia entre un valor y el selector se ejecutarían todos los grupos de sentencias, incluida la del default . En la gran mayoría de los casos, éste no es el comportamiento que se desea de la sentencia switch , de ahí que cada uno de los case habitualmente termine con un break , es decir, con un salto incondicional fuera de la estructura de control de flujo.

La variable selector sólo puede ser de tipo char o cualquier tipo de valor entero menos long (si estás empleando Java 5 o posterior también se pueden emplear enumeraciones). No podemos emplear ningún tipo de número real, Strings, u objetos en general.

En el listado 6 mostramos un ejemplo de uso de la sentencia switch . En este caso, tomamos ventaja de que una vez que hay una coincidencia entre la variable que hace las veces de selector y uno de los valores de los case se ejecutan todos los bloques de código de todos los case que haya a continuación y el bloque de código del default . El programa genera una letra aleatoria del alfabeto y emplea un switch para comprobar si la letra es una vocal o una consonante; si es una vocal imprime "vocal" por la consola, y si es una consonante imprime "consonante". El principio códigos o promotores switch se emplea para comprobar si el carácter es una de las cinco vocales del alfabeto, por tanto hay que hacer cinco comparaciones diferentes. Pero la acción a realizar en los cinco casos es la misma: imprimir

"vocal". De ahí que podamos tomar ventaja del particular comportamiento del switch .

```
//LISTADO 6: Ejemplo de la estructura de control de flujo switch.

//codigo Ejemplo11.java del CD

//Bucle for. Ejecutará 100 veces el codigo que tiene dentro.

   for(int i = 0; i < 100; i++) {
//generamos un carácter aleatorio

      char c = (char)(Math.random() * 26 + 'a');

    System.out.print(c + ": ");
      switch(c) {

      case 'a':

      case 'e':

      case 'i':

      case 'o':

      case 'u':

//Si el caracter es ‘a’, ‘e’, ‘i’, ‘o’ o ‘u’ imprimimos vocal.

         System.out.println("vocal");
                 break;

      default:

//Si no era ninguna de las anterioes imprimos consonate.

         System.out.println("consonante");
      }

    }

  }
```

Bucles

Son instrucciones que nos permiten repetir un bloque de código mientras se cumpla una determinada condición. En Java existen cuatro tipos diferentes: while , do while , for , y (en Java 5) "for each".

Bucle while

Cuando en la ejecución de un código se llega a un bucle while se comprueba si se verifica la condición asociada con él, si se verifica se ejecuta el bloque de código asociado con el bucle y se vuelve a comprobar la condición; mientras dicha condición siga verificándose se seguirá ejecutando el código del bucle. Su sintaxis es:

```
while(condición){

Grupo de sentencias;}
```

El listado 7 muestra el funcionamiento de este bucle. En él se emplea el método random de la librería matemática de Java para generar un número aleatorio. Mientras el número aleatorio que se genera sea menor que 0.99 (ver condición del bucle) se vuelve a generar otro número aleatorio y se imprime dicho número por consola.

```
//LISTADO 7: Ejemplo de bucle tipo while.

//codigo Ejemplo12.java del CD

   double r = 0;
//Mientras que r < 0.99 sigue ejecutando el cuerpo del bucle.

    while(r < 0.99) {

//Genera un nuevo r aleatario entr 0 y 1.

      r = Math.random();

    System.out.println(r);
     }
```

El bucle do while

Su comportamiento es semejante al bucle while , sólo que aquí la condición va al final del código del

bucle, por lo que el código se va a ejecutar al menos una vez. Como norma general, debemos emplear un bucle tipo while cuando haya un conjunto de instrucciones que se puedan repetir de 0 a n veces, y un bucle do while cuando tengamos un conjunto de instrucciones que se puedan repetir de 1 a n veces. La sintaxis de do while es:

```
do {

Grupo de sentencias;

}while(condición);
```

En el listado 8 podemos ver el código de listado 7 implementado mediante un bucle do while.

```
//LISTADO 8: Ejemplo de bucle tipo do while.

//código Ejemplo13.java del CD

  double r;
    do {

      r = Math.random();

   System.out.println(r);
    } while(r < 0.99d);

  }
```

Bucle for

La sintaxis de este bucle es la siguiente siguiente:

```
for(expresion1;expresion2;expresion3){

Grupo de sentecias;}
```

Expresion1 es una expresión cualquiera, en la cual es posible declarar y/o asignar valor a una variable, la variable-condición del bucle. Expresion2 es la condición del bucle. Mientras dicha condición tome el valor true el bucle se repetirá una y otra vez. Habitualmente, esta expresión comprueba alguna condición sobre una variable declarada o asignada en la primera expresión. Expresion3 indica una operación que se realiza en cada iteración a partir de la primera (en la primera iteración el valor de la variable del bucle es el que se le asigna en expresion1); nuevamente, lo más habitual es que esta operación se realice sobre la variable del bucle. El bucle for anterior es completamente equivalente al siguiente código:

```
expresion1;

while (expresion2)

  {

  Grupo de sentecias;

 expresion3;

 }
```

En el listado 9 se emplea un bucle for para imprimir por consola los caracteres de la tabla ASCII. Cuando el carácter se imprime haciéndole un cast a int se mostrará el valor entero correspondiente, y cuando se imprime como carácter se mostrará el carácter asociado a dicho valor entero en la tabla ASCII.

```
//LISTADO 9: Este pequeño código imprime por consola la tabla ASCII empleando un
bucle tipo for.

//código Ejemplo14.java del CD

  for( char c = 0; c < 128; c++)

    System.out.println("valor: " + (int)c + " caracter: " + c);
  }
```

Bucle for-each

Esta característica sólo está disponible en Java 5 y versiones posteriores. Se trata de un bucle diseñado con el propósito de recorrer un conjunto de objetos. Por lo de ahora la única estructura que hemos visto

que permite almacenar un conjunto de objetos son los arrays, pero en números posteriores de esta serie de artículos introduciremos el framework de colecciones de datos de Java. Esta estructura resulta particularmente útil para trabajar con dichas colecciones (listas, pilas, mapas...). La sintaxis de este bucle es:

```
for(Tipo elemento: colecciónElementos){

Grupo de sentecias;}
```

Tipo es el tipo de dato de los elementos del conjunto; elemento es una variable auxiliar que la primera vez que se ejecute el bucle tomará el valor del primer elemento del conjunto, la segunda vez tomará el valor del segundo elemento del conjunto y así sucesivamente. La colecciónElementos es el conjunto de elementos sobre los cuales queremos iterar (por lo de ahora, un array). El bucle se repetirá una vez para cada elemento de la colección.

Este bucle "foreach" se llama "for", aunque hubiese sido mejor emplear el primer nombre por similitud con otros lenguajes de programación que poseen un bucle similar, por haber sido introducido en el lenguaje de programación a posteriori. foreach no es una palabra reservada dentro de Java; si cuando se diseñó Java 5 se hubiese empleado esta palabra para este tipo de bucle podrían producirse incompatibilidades con código fuente Java escrito antes de ese momento.

En el listado 10 podemos ver cómo se emplea este tipo de bucle para calcular la suma de los elementos de un array.

```
//LISTADO 10: Ejemplo de bucle "foreach" en Java.

//código Ejemplo15.java del CD

    int array[] = new int[10];
    int suma = 0, contador = 0;
        //con este bucle damos valores a los elementos del array

    for (int i = 0; i < array.length; i++) {
            array[i]= 2*i;

        }

    for (int e : array){  //para cada elemento del array
            suma = suma + e;

        }
```

```
System.out.println(suma);
}
```

break y continue

Estas dos instrucciones no son ni un bucle ni un condicional, pero sí son sentencias íntimamente relacionada con estas estructuras de control de flujo. El encontrarse una sentencia de break en el cuerpo de cualquier bucle detiene la ejecución de su cuerpo y produce un salto incondicional a la sentencia que se encuentra inmediatamente después del bucle. Como ya hemos mostrado en este artículo, esta sentencia también se puede usar para forzar la salida del bloque de código asociado con uno de los case de una instrucción tipo switch.

continue también detiene la ejecución del cuerpo del bucle, pero en esta ocasión no se sale del bucle, sino que se ignora el resto del cuerpo del bucle que quedase por ejecutar y se vuelve al principio.

En el listado 11 podemos ver un ejemplo de uso de estas sentencias. En el primer bucle for , continue se emplea para ignorar todos los números enteros que no sean múltiplos de 9. En el segundo bucle, break se emplea para salir de un bucle infinito.

```
//LISTADO 11: Ejemplo de uso de las sentencias break y continue.

//codigo Ejemplo16.java del CD

   for(int i = 0; i < 100; i++) {
//Salto a la siguiente iteracion si i no es divisible entre 9

      if(i % 9 != 0) continue;

//Si i es divisible entre 9 se imprime

     System.out.println(i);
     }

   int i = 0;
     // Lazo infinito del cual se sale con break:

     while(true) {

       i++;

       if(i == 120) break; // Salimos del lazo
```

```
    System.out.println(i);
  }
```

Conclusiones

En este segundo artículo de la serie hemos presentado la librería matemática de Java, varios tipos de datos no primitivos como arrays, cadenas de caracteres y enumeraciones, y los elementos básicos de control de flujo (bucles y condicionales) del lenguaje.

En el siguiente artículo de la serie comenzaremos a introducir la programación orientada a objetos en Java. Explicaremos qué es una clase, cómo trabajar con ellas, y en qué consiste la herencia. También comenzaremos emplear un entorno de desarrollo, BlueJ, para trabajar con los códigos de ejemplo. Os espero a todos el mes que viene.

Descargas

- Códigos del artículo

Cápitulos anteriores del curso:

Curso de programación Java I - Abraham Otero