Tema 2. Estructuras de control.

1. Condiciones.

   Una condición es una expresión booleana, es decir, que devuelve *true* o *false*.

2. Estructuras if e if-else.

   La estructura if tiene la forma

   ```
   if (condición) {
     acciones;
   }
   ```

   y se traduce en que las *acciones* se ejecutarán si la *condición* se evalúa como *true*. En caso contrario se saltarán y se seguirá ejecutando el programa a partir de la siguiente línea al bloque *if*.

   La sentencia if – else tiene la forma

   ```
   if (condición) {
     acciones;
   } else {
     otras_acciones;
   }
   ```

   Su efecto es el siguiente: si la *condición* se evalúa como *true*, se ejecutarán las *acciones* y a continuación se seguirá ejecutando el programa a partir del final del bloque *if – else*. Si la *condición* se evalúa como *false*, se ejecutarán las *otras_acciones* y el programa continuará después del bloque *if – else*.

3. Estructura if- else if.

   Las sentencias if e if -else se pueden anidar:

   ```
   if (condición1) {
     if (condición2) {
       acciones1;
     } else {
       acciones2;
     }
   }
   ```

   Además, si la acción consta de una única sentencia, se pueden omitir las llaves (en otras palabras, no es necesario definir un bloque). Aprovechando esta característica, cuando tenemos una serie de condiciones excluyentes, podemos anidar sentencias *if – else* de la siguiente forma:

```
if (condición1) {
  acciones1;
} else if (condición2) {
  acciones2;
} else if (condición3) {
  acciones3;
.
.
}
```

Escribiendo todas las sentencias *if* en el mismo nivel de identación, evitamos que las líneas se nos vayan a la derecha, mejorando la legibilidad del código, aunque realmente estén anidadas.

Es muy importante que las condiciones sean excluyentes, es decir, que no haya ningún caso (ninguna combinación de valores) que permita evaluar como *true* más de una condición.

4. Estructura switch.

Cuando tenemos una decisión múltiple basada en los posibles valores de una expresión, en lugar de usar la estructura anterior podemos usar la estructura *switch* de la siguiente forma:

```
switch (expresión) {
  case valor1:
    acciones1;
    break;
  case valor2:
    acciones2;
    break;
  .
  .
  default:
    acciones_por_defecto;
}
```

Su efecto es el siguiente: se evalúa la *expresión*, que debe ser de tipo entero, *String* o enumerado (los tipos enumerados se verán más adelante) y se busca una clausula *case* cuyo valor coincida con el resultado de la *expresión*, en orden de aparición. Si se encuentra una coincidencia, se ejecutan las acciones correspondientes a dicha clausula *case*. Si no se encuentra ninguna coincidencia y se ha especificado una clausula *default*, se ejecutarán las acciones por defecto.

La clausula *default* es opcional, y en caso de aparecer tiene que ser la última.

Las sentencias *break* sirven para que una vez ejecutadas las acciones de una clausula case, salte la ejecución a la línea siguiente al final del bloque *switch*.
En algunos casos, nos encontramos con casos equivalentes, es decir, varios valores que deben producir el mismo resultados. En esas circunstancias podemos encadenar varias clausulas case sin separarlas por *break*, para que todas ellas ejecuten las mismas acciones.

5. Expresiones condicionales.

Existe un tipo de expresiones llamado expresión condicional o expresión ternaria, ya que consta de tres operandos, y tiene la sintaxis

```
condición?expresión1:expresión2
```

Cuando aparece una expresión de este tipo, se evalúa la *condición*, y si el resultado es *true*, se devuelve el resultado de evaluar la *expresión1*. En caso contrario se devuelve el resultado de evaluar *expresión2*.

6. Introducción Bucles.

Los bucles son estructuras repetitivas (también llamadas iterativas) que permiten ejecutar un bloque de instrucciones un número de veces. En Java todos los bucles dependen de una condición y su bloque de instrucciones se ejecutará repetidamente mientras la condición se evalúe como *true*.

Es importante que durante la ejecución del bucle, en algún momento se llegue a un estado en el que la condición pase a valer *false*, ya que de lo contrario el bucle se volvería infinito (no terminaría nunca).

7. Bucle while.

El bucle while tiene la sintaxis

```
while (condición) {
    acciones;
}
```

Este bucle ejecuta las acciones mientras la *condición* se evalúe como *true*. Su funcionamiento detallado es así: cuando el programa llega a la sentencia *while*, se evalúa la *expresión*. Si el resultado es *true* se ejecuta el bloque de *acciones*. Si es *false* se salta a la siguiente instrucción después del bloque *while*. Si se ejecuta el bloque de *acciones*, se vuelve a la sentencia *while* y se procede de nuevo de la misma forma.

Se puede ver fácilmente que si la condición es *false* la primera vez, nunca se ejecuta el bloque de *acciones*.

8. Bucle do-while.

El bucle *do – while* tiene la forma

```
do {
    acciones;
} while (condición);
```

Aunque es parecido al bucle *while*, tiene un orden de ejecución distinto y ciertas características que los diferencian.

En este caso, cuando el programa llega a la sentencia *do*, se ejecuta el bloque de *acciones*. Después se evalúa la *condición*. Si el resultado es *true*, se vuelve a la sentencia *do* y se procede del mismo modo. Si es *false*, se sale.

Como vemos, en este caso el bloque de *acciones* se ejecuta al menos una vez.

9. Bucle for.

El bucle *for* se usa normalmente para realizar bucles que se repiten un número determinado de veces o que recorre una secuencia de valores. Su sintaxis es:

```
for (inicialización; condición; actualización) {
    acciones;
}
```

Su funcionamiento es el siguiente: cuando el programa llega a la sentencia *for* se ejecuta la sentencia de *inicialización*. A continuación se evalúa la *condición*, y si el resultado es true, se ejecuta el bloque de *acciones*. Después se ejecuta la sentencia de *actualización*, y se vuelve al *for*, pero sin volver a ejecutar la *inicialización*. De esta forma se vuelve a comprobar la *condición* para repetir o no el bloque de *acciones*, y así sucesivamente.

La sentencia de *inicialización* se usa normalmente para inicializar el índice o contador que controlará la ejecución del bucle. La *condición* se usa para decidir si el bucle continúa ejecutándose o termina y la sentencia de *actualización* se usa normalmente para actualizar el índice o contador al final de cada vuelta.

Los tres tipos de bucles son equivalentes en cuanto a su funcionalidad, es decir, se pueden intercambiar con pocas modificaciones en el código. Por ejemplo, un bucle *for* se puede sustituir por un bucle *while* con la misma *condición*, poniendo la sentencia de *inicialización* antes del bucle y la de *actualización* como última instrucción del bloque.

10. Sentencias break y continue.

La sentencia `break` sirve para salirse del bucle en el mismo punto en el que aparece, sin completar el resto del bloque, al igual que en la estructura *switch*.

La sentencia `continue` rompe la ejecución actual del bucle en el lugar en que aparece y continúa con la siguiente iteración, es decir, se salta el resto del bloque de *acciones* en la iteración en curso.

Tanto la sentencia *break* como *continue* están encarecidamente desaconsejadas por el código de honor de los programadores, el consejo de ancianos informáticos y los profesores de este módulo, ya que rompe la secuencia de ejecución de las estructuras y puede introducir mayor complejidad en el programa, derivando en un riesgo mayor de cometer errores de programación.

11. Bucles anidados y bucles dependientes.

Los bucles se pueden anidar, es decir, dentro del bloque de *acciones* de un bucle pueden aparecer otro u otros bucles. De esta forma el bucle anidado se ejecutará (realizará todas sus iteraciones) para cada vuelta del bucle que lo contiene.

Un caso especial de bucles anidados son los bucles dependientes en los que el número de iteraciones (o la *condición*) del bucle anidado, depende de la iteración actual (del índice, por ejemplo) del bucle exterior.