Sesión 3:
Sentencias de control

Etapas en la solución de problemas

- **Sentencias de selección: condicionales**
- **Sentencias de iteración: bucles**
- **Sentencias de salto**

Sentencias de selección: if

- **if** es la sentencia de bifurcación de Java. Permite llevar el programa hacia dos caminos diferentes. Su formato es:

```
if (condicion) sentencia_1;
else sentencia_2;
```

```
if (condicion) {
        sentencia_1;
} else {
        sentencia_2;
}
```

- Cada sentencia puede ser única o encerrada entre llaves, es decir, un bloque.
- La condición tiene que ser cualquier expresión que devuelva un **boolean**.
- La expresión else es opcional.
- Funciona así: Si la condición es verdadera, se ejecuta la sentencia_1, si no, se ejecuta la sentencia_2

Sentencias de selección: Otras formas de if

if anidados

- Un if anidado es una sentencia if que está contenida dentro de otro if o else.
- Son muy habituales en programación.
- Muy importante: las llaves para definir los bloques, y las tabulaciones para determinar a simple vista donde empieza cada bloque.

if-else-if múltiples

- Se basa en una secuencia de if anidados.
- Se ejecuta de arriba a abajo.
- En cuanto una se cumple, se saltan las demás.
- El else final actúa como condición por defecto.

```
if (condicion_2) {
        sentencia_1;
} else if (condicion_3) {
      sentencia_3;
} else {
        sentencia_2;
}
```

Sentencias de selección: el operador “?”

El operador “?”

- Es un operador especial que sustituye a sentencias if-then-else

- Puede resultar confuso al principio, pero una vez se ha practicado con él, resulta muy efectivo.

- El operador tiene la siguiente forma: ***expresion_1 ? expresion_2 : expresion_3***
  - Donde *expresion_1* es cualquier expresión con resultado **boolean**.
  - Si *expresion_1 = true*, se evalúa la *expresion_2*. Por el contrario, se evalúa la *expresion_3*

- Por ejemplo, dada esta expresión:

```
int ratio = num == 0 ? 0 : a / num
```

  - Se evalúa la expresión 1 (num == 0)
  - Si num es 0, la variable ratio valdrá 0;
  - Si la expresión 1 es falsa, ratio valrá a / num.

Sentencias de selección: switch

- La sentencia switch es una sentencia de bifurcación múltiple de Java
- Proporciona una forma sencilla de dirigir la ejecución a distintas partes del programa en función del valor de una expresión.
- Su formato es:

```
switch (expresion_1) {
    case valor1:
       //secuencia de sentencias
       break;

    case valor2:
    //sentencias
    break;

    default:
    //por defecto
    break;
}
```

Sentencias de selección: switch

- Funciona así: se compara el valor de la expresión con cada uno de los casos. Si coincide con alguno, se ejecuta el código que tenga dentro. Si no coincide con ningún caso, se ejecuta la sentencia *default*.
- La sentencia *default* es opcional. Si no se coloca y no coincide la expresión con ningún caso, no se ejecuta ninguna acción.
- En muchas ocasiones, es una alternativa mejor que una serie larga de if-else-if.
- La expresión debe ser de tipo byte, short, int o char.
- Cada uno de los valores del caso es un literal único, no puede ser una constante o una variable. Tampoco permite valores duplicados en las sentencias case.
- La sentencia *break* también es opcional. Si se omite, la ejecución continúa hasta el siguiente *case*.

Sentencias de iteración: while

- Con el bucle while se repite una sentencia o bloque mientras que la condición asociada sea verdadera.

- Su forma general es:

```
while (condicion) {
    //cuerpo del bucle
}
```

- La condición puede ser cualquier condición booleana. El cuerpo del bucle se ejecutará mientras la expresión sea verdadera. Cuando la condición sea falsa, se ejecutará la siguiente línea de código.

- Ya que en el bucle while la expresión condicional se evalúa al principio, el bucle no se ejecutará nunca si la condición es falsa.

Sentencias de iteración: do-while

- El bucle do-while es un bucle que, a diferencia del while, se ejecuta al menos una vez, ya que la expresión condicional está al final del mismo.

- Su forma general es:

```
do {
      //cuerpo del bucle
} while (condicion)
```

- En cada iteración del bucle do-while, se ejecuta en primer lugar el cuerpo, y a continuación se evalúa la condición. Si es verdadera, el bucle se repetirá. En caso contrario, el bucle finalizará. Como en todos los bucles, la expresión debe ser de tipo boolean.

- El bucle do-while es muy útil cuando se procesa un menú de selección, ya que se desea que el menú se ejecute al menos una vez.

Sentencias de iteración: for

- Es el bucle más versátil. Permite repetir unas sentencias un número conocido de veces.
- Su formato es el siguiente:

```
for (inicialización; condición; iteración) {
    sentencia
}
```

- En la inicialización, se declara una variable de control (contador) y se inicializa con un valor.
- La condición es una expresión booleana que examina al contador. Si es verdadero, el bucle sigue iterando. Si es falso, el bucle termina.
- La iteración determina cómo va a cambiar el contador cada vez que se da una vuelta al bucle.

Sentencias de iteración: for

- A menudo, el contador del bucle sólo se necesita para el bucle y no se usa en ninguna otra parte externa. En ese caso, se puede declarar la variable dentro del for.

```
for (int i = 0; i < 100; i++) {
      sentencia
}
```

- Se puede usar el bucle for controlado por dos variables, separando las expresiones con comas.

```
for (a = 1, b = 4; a < b; a++, b--) {
      // sentencia
}
```

- También permite el uso de bucles anidados. Es decir, un bucle dentro de otro.

Sentencias de salto: break

- Tiene 3 usos en Java:
  - Para finalizar un caso en un switch
  - Se puede usar para salir de un bucle
  - Se puede usar para ir a partes etiquetadas del código. Aunque de momento, esta opción no la vamos a estudiar.
- Mediante break se puede forzar la finalización inmediata de un bucle. Se puede usar con cualquier bucle. Si se usa en un bucle anidado, se saldrá del más interno.

```
for (int i = 0; i < 100; i++) {
        if (i == 10) {
                break;
        }
        System.out.println(i);
}
```

Sentencias de salto: continue

- La sentencia continue se utiliza en un bucle para que realice otra iteración saltándose lo que restaba de código.
- Se puede utilizar con cualquiera de los 3 bucles.
- Es raro su uso, pero es una instrucción útil en ocasiones especiales

```
for (int i = 0; i < 10; i++) {
      System.out.print (i + " ");
        if (i % 2 == 0) {
              continue;
      }
      System.out.println("");
}
```

Sentencias de salto: return

- La sentencia return se utiliza para salir explícitamente de un método, es decir, hace que el programa vuelva a la parte del código donde el método fue llamado.
- La sentencia return, además, hace que finalice inmediatamente el método en el que se ejecuta.
- Suele ir precedida de un if, una condición que si se cumple, se salga del método.