Taller de preparación para la OCI - PUC

Control de flujo if - else

1. Introducción

En este capítulo se tratará el control de flujo condicional utilizando `if`, `else if` y `else`. Estos son elementos presentes en cualquier lenguaje de programación.

Cuando estamos programando nos gustaría que algunas instrucciones se ejecuten si se cumplen ciertas condiciones. Por ejemplo, pensemos que existen dos personas: Alice y Bob. Ellos quieren usar la aplicación de su banco para transferirse dinero. Si Alice quiere transferirle $2,000 a Bob, Alice al menos tiene que tener $2,000 disponibles en su cuenta para transferirle, por lo que nos gustaría efectuar la transferencia solamente **si Alice tiene el dinero suficiente**.

De la misma forma, cuando una condición no se cumplió, a veces nos gustaría saber si existe otra condición que sí se cumple para ejecutar otra instrucción. Supongamos que Charles, un estudiante de 16 años, quiere ir a ver una película para mayores de 17 años pero que puede ser vista por mayores de 15 si va en compañía de un adulto. Suponga que Charles fue al cine con su padre. Nuestro estudiante no cumple con la condición "ser mayor de 17 años", pero de todas formas es posible que él compre su entrada porque cumple con la condición **ser mayor de 15 años e ir en compañía de un adulto**. En este caso la primera condición no se cumplía, pero la segunda sí.

Finalmente, nos gustaría que ciertas instrucciones de código se ejecutar si ninguna de nuestras condiciones se cumplió. Piense ahora en Doris, una estudiante de 13 años que desea ir a ver la misma película que fue a ver Charles. Doris va con su madre a ver esta película. Sin embargo cuando intenta comprar una entrada **no cumple con ninguna de las condiciones** ya que no tiene más de 17 años, y si bien va en compañía de un adulto, no es mayor a 15 años. Por lo tanto, queremos que se le prohíba a Doris comprar su entrada.

Para poder describir algoritmos que solucionen los problemas planteados necesitamos usar `if`, `else if` y `else`. La descripción de cada uno se presenta a continuación:

- `if`: ejecuta un bloque de código si se cumple la condición.
- `else if`: ejecuta un bloque de código si no se cumplió la condición anterior pero se cumple una nueva condición.
- `else`: ejecuta un bloque de código si no se cumplió ninguna de las condiciones anteriormente señaladas.

2. Comparación entre valores

2.1. Comparación entre números

Antes de seguir, recordemos las expresiones que nos permiten comparar números. Estas son:

- Comparar si dos variables son iguales (`==`).
- Comparar si dos variables son distintas (`!=`).

- Comparar si una variable es menor o mayor que otra (`<` o `>`). Para menor igual o mayor igual usar (`<=` o `>=`).

Es importante notar que cuando hacemos estas comparaciones, el resultado es una expresión *booleana*, es decir, puede tomar los valores `true` o `false`. Por lo tanto, cuando escribo:

```
1 int n = 4;
2 boolean comparison = (n == 4);
```

El valor `comparison` va a ser `true`. En cambio si escribo:

```
1 int n = 4;
2 boolean comparison = (n >= 5);
```

El valor de `comparison` va a ser `false`.

2.2. Comparación entre Strings

Para comparar *strings* utilizamos una función de Java ya implementada llamada `equals`. **No podemos comparar *strings*** utilizando `==` por razones que no son parte del contenido de este curso. Como ejemplo tomemos el código a continuación.

```
1 String s = "Hola";
2 boolean comparison = (s.equals("Hola"));
3 System.out.println(comparison);
```

En este caso `comparison` toma el valor `true`. En cambio en el siguiente código:

```
1 String s1 = "Hola";
2 String s2 = "Chao";
3 boolean comparison = (s1.equals(s2));
4 System.out.println(comparison);
```

La variable `comparison` toma el valor `false`.

Observación 2.1. En la comparación de *strings* es importante el uso de mayúsculas y minúsculas, por lo que el *string* con valor ``Hola'' es distinto al *string* con valor ``hola'', ya que el primero tiene el caracter `H` (mayúscula), y el segúndo el caracter `h` (minúscula).

3. Control de flujo

3.1. Control de flujo utilizando if

Un `if` es una instrucción que me permite ejecutar código si se cumple cierta condición. La estructura de un `if` en Java se ve de la siguiente forma:

```
1 if (<condición>) {
2   // bloque de código que se ejecuta en caso
3   // de que se cumpla la condición.
4 }
```

La condición es una expresión de tipo booleana (es decir, puede ser solamente `true` o `false`). Si esta asignación es `true`, entonces se ejecuta el bloque de código que está dentro de los paréntesis de llave ({ ... }). Recordemos que las condiciones que conocemos hasta ahora son:

- Comparar si dos variables son iguales (==).
- Comparar si una variable es menor o mayor que otra (< o >). Para menor igual o mayor igual usar (<= o >=).

Veamos ahora como se vería el ejemplo planteado en la introducción, en que Alice quiere transferirle dinero a Bob.

Ejemplo 3.1. Alice quiere transferirle $2,000 a Bob, pero sólo puede hacerlo si tiene dinero suficiente en su cuenta.

```
1  int moneyAlice = 3000;
2  int moneyBob = 1000;
3
4  if (moneyAlice >= 2000) {
5    moneyAlice = moneyAlice - 2000;
6    moneyBob = moneyBob + 2000;
7    System.out.println("Se realizó la transferencia");
8  }
9
10 System.out.println("Dinero Alice: " + moneyAlice);
11 System.out.println("Dinero Bob: " + moneyBob);
```

En el Ejemplo 3.1 si se cumple la condición de que el dinero de Alice es mayor o igual a $2,000. Esto hace que `moneyAlice >= 2000` sea equivalente a `true`. Por lo tanto, se ejecuta el código que le resta $2,000 a su cuenta. Luego se suman $2,000 a la de Bob y se imprime un mensaje que señala que se hizo la transferencia. Luego se imprime el saldo de Alice ($1,000) y el de Bob ($3,000). Ahora veamos un ejemplo en el que la condición no se cumple:

Ejemplo 3.2. Alice quiere transferirle $2,000 a Bob, sólo puede hacerlo si tiene dinero suficiente en su cuenta y esta vez no dispone de dicha cantidad.

```
1  int moneyAlice = 1000;
2  int moneyBob = 1000;
3
4  if (moneyAlice >= 2000) {
5    moneyAlice = moneyAlice - 2000;
6    moneyBob = moneyBob + 2000;
7    System.out.println("Se realizó la transferencia");
8  }
9
10 System.out.println("Dinero Alice: " + moneyAlice);
11 System.out.println("Dinero Bob: " + moneyBob);
```

En el Ejemplo 3.2 no se cumple la condición de que el dinero de Alice es mayor o igual a $2,000, por lo tanto, no se ejecuta el bloque de código de la transferencia. Luego se imprime el saldo de Alice ($1,000) y el de Bob ($1,000).

3.2. Control de flujo utilizando if - else

Ahora introduciremos la instrucción `else`. Esta es una instrucción que premite ejecutar un bloque de código si la condición anterior no se cumple. Un `else` debe venir después de un `if`. La estructura de un bloque `if - else` es como la que se muestra a continuación:

```
if (<condición>) {
  // bloque de código que se ejecuta en caso
  // de que se cumpla la condición.
}
else {
  // bloque de código que se ejecuta en caso
  // de que no se cumpla la condición
}
```

Es importante notar que en caso de que se cumpla la condición impuesta en el `if`, el bloque de código dentro del `else` es ignorado. Veamos un ejemplo:

Ejemplo 3.3. Escriba un código en Java que pida al usuario un número. Si el número es mayor que 0 debe imprimir un mensaje que señale que el número es mayor que 0. En caso contrario debe imprimir un mensaje que indica que el número es menor o igual al 0.

```
Scanner scanner = new Scanner(System.in);
System.out.println("Ingrese el número: ");
int n = scanner.nextInt();

if (n > 0) {
  System.out.println("El número era mayor que 0");
}
else {
  System.out.println("El número era menor o igual que 0");
}
```

En el ejemplo 3.3 si ingresamos como input el número 10, la condición `n > 0` será equivalente a `true`, por lo que se va a imprimir el mensaje "El número era mayor que 0" y el código del `else` será ignorado.

Si ingresamos -1, no se va a cumplir la condición `n > 0`, por lo que se va a ingresar al bloque de código dentro del `else` y se va a imprimir el mensaje "El número era menor o igual que 0".

3.3. Control de flujo utilizando if - else if - else

Ahora incluiremos al control de flujo la instrucción `else if`. Esta es una instrucción que me permite ejecutar un bloque de código si no se cumplió una instrucción anterior y se cumple una nueva condición. Un `else if` viene después de un bloque `if` o de otro bloque `else if`. La estructura se ve de la siguiente forma:

```
if (<condición 1>) {
  // bloque de código que se ejecuta en caso
  // de que se cumpla la condición.
}
else if (<condición 2>) {
  // bloque de código que se ejecuta en caso
  // de que se cumpla la condición 2.
}
else if (<condición 3>) {
  // ...
}
// puede haber una serie arbitraria de else if...
else if (<condición n>) {
```

```
14    // ...
15  }
16  else {
17    // ...
18  }
```

Las condiciónes $1, ..., n$ son condiciones *booleanas*. Después de un `if` puede haber tantos `else if` como el programador quiera. Finalmente, el bloque de código dentro del `else` se va a ejecutar sin no se cumplió ninguna de las condiciones (tanto la del `if`, como la de los `else if`).

Observación 3.1. Es importante notar que los bloques `if - else if` se van comprobando en secuencia. Una vez que una condición se cumple y se ejecuta, no se va a comprobar ningún bloque más. Por ejemplo, si se cumple la condición 3, no se van a comprobar las condiciones desde la 4 en adelante y se va a ignorar el bloque dentro del `else`, saliendo de toda la secuencia `if - else if - else`. Veamos un ejemplo.

Ejemplo 3.4. Escriba un código en Java que pida al usuario un número. Imprima un mensaje señalando si el número es mayor, menor o igual a 0.

```
1  Scanner scanner = new Scanner(System.in);
2  System.out.println("Ingrese el número: ");
3  int n = scanner.nextInt();
4
5  if (n > 0) {
6    System.out.println("El número era mayor que 0");
7  }
8  else if (n == 0) {
9    Systen.out.println("El número es igual a 0");
10 }
11 else {
12   System.out.println("El número era menor que 0");
13 }
```

En el ejemplo 3.4 si ingresamos como input el número 7, la condición `n > 0` será equivalente a `true`, por lo que se va a imprimir el mensaje "El número era mayor que 0". Así, los bloques de código del `else if` y del `else` serán ignorados.

Si ingresamos el número 0, la condición `n > 0` será equivalente a `false` y el bloque de código dentro del `if` será ignorado. Luego se pasará a comprobar la condición del `else if`. En este caso la expresión `n == 0` es equivalente a `true`, por lo que se va a ejecutar el bloque de código dentro del `else if` y el `else` será ignorado.

Si ingresamos -2, no se va a cumplir la condición `n > 0`, luego en el `else if` la condición `n == 0` tampoco se cumple (la expresión es equivalente a `false`), por lo que se va a ingresar al bloque de código dentro del `else` y se va a imprimir el mensaje "El número era menor que 0".

3.4. Anidación de bloques if - else

El código que puede ir dentro de un bloque `if`, `else if` o `else` puede ser código válido en Java, incluyendo código que contiene `if - else if - else`. Por lo mismo es posible de anidar los `if - else if - else` dentro de otros. Veamos un ejemplo.

Ejemplo 3.5. Escriba un código en Java que pida al usuario un número. Si el número es mayor que 0 imprima un mensaje señalando si el número es par o impar.

```
1  Scanner scanner = new Scanner(System.in);
2  System.out.println("Ingrese el número: ");
3  int n = scanner.nextInt();
4
5  if (n > 0) {
6    if ((n % 2) == 0) {
7      System.out.println("El número era par");
8    }
9    else {
10     System.out.println("El número era impar");
11   }
12 }
13 else {
14   System.out.println("El número no era mayor que 0");
15 }
```

El Ejemplo 3.5 muestra la anidación de un bloque `if - else` dentro de un `if`. Cabe destacar que en la línea 6 estamos preguntando si el resto de la división entre el número `n` y 2 es igual a 0[1]. En caso de que esto se cumpla la expresión `(n% 2) == 0` va a ser `true` y se va a imprimir el mensaje señalando que el número es par. En caso de que el número no haya sido mayor a 0 se procede directamente a la línea 13, saltándose todo el bloque que intenta comprobar si el número era par.

4. Ejemplos de if - else if - else

A continuación presentamos algunos ejemplos resueltos.

Ejemplo 4.1. Haz un programa que pida al usuario tres números enteros y calcule su promedio. Luego debe imprimir un mensaje "Aprobado" si el promedio es mayor o igual a 4.0 y reprobado si es menor.

```
1  public class Main {
2    public static void main(String[] args) {
3
4      Scanner scanner = new Scanner(System.in);
5      System.out.println("Ingrese el número 1: ");
6      int n1 = scanner.nextInt();
7      System.out.println("Ingrese el número 2: ");
8      int n2 = scanner.nextInt();
9      System.out.println("Ingrese el número 3: ");
10     int n3 = scanner.nextInt();
11
12     double avg = ((double) (n1 + n2 + n3)) / 3.0;
13     if (avg >= 4.0) {
14       System.out.println("Aprobado");
15     }
16     else {
17       System.out.println("Reprobado");
18     }
19
```

[1]Si esto se cumple el número es par.

```
20    }
21  }
```

En el ejemplo no validamos el rango de los números, aprenderemos a hacer eso en otra clase. El mensaje es acorde al valor de la variable `avg`, que es de tipo `double` y guarda el promedio.

Ejemplo 4.2. Haz un programa que reciba un string y un número. El número representa una temperatura y el string puede ser ``celcius'' para indicar que la temperatura está en grados celcius y ``fahrenheit'' si la temperatura está en fahrenheit. Su programa debe transformar a grados fahrenheit si la temperatura está en celcius y a celcius si la temperatura está en fahrenheit.

```
1  public class Main {
2    public static void main(String[] args) {
3
4    Scanner scanner = new Scanner(System.in);
5    System.out.println("Ingrese la temperatura: ");
6    double temp = scanner.nextDouble();
7    scanner.nextLine();
8    System.out.println("Ingrese el tipo de temperatura: ");
9    String tempType = scanner.nextLine();
10
11   if (tempType.equals("celcius")) {
12     double far = 9.0/5.0 * temp + 32.0;
13     System.out.println("Temperatura en fahrenheit: " + far);
14   }
15   else if (tempType.equals("fahrenheit")) {
16     double cel = (temp - 32.0) * 5.0/9.0;
17     System.out.println("Temperatura en fahrenheit: " + cel);
18   }
19   else {
20     System.out.println("Error en la temperatura");
21   }
22   }
23 }
```

En la solución vemos cómo comparar *strings*. También notamos la importancia de capturar en el `else` el caso de un *input* incorrecto. Además notamos que en la línea 7 hay un `scanner.nextLine()`. Esto es necesario porque cuando usamos `nextDouble()` la línea no ha terminado, y cuando intentamos obtener el tipo de temperatura necesitamos leer una nueva línea, por eso debemos hacer un `nextLine()` adicional antes de pedir el tipo de temperatura (`String tempType = scanner.nextLine()`)

Ejemplo 4.3. Haz un programa que pida al usuario dos números enteros $n1$ y $n2$. Luego debe pedir un tercer número que representa una operación matemática (1 para sumar, 2 para restar, 3 para multiplicar y 4 para dividir). Debe imprimir en pantalla el resultado de operar $n1$ con $n2$ mediante el operador que quería usar el usuario.

```
1  public class Main {
2    public static void main(String[] args) {
3
4      Scanner scanner = new Scanner(System.in);
5      System.out.println("Ingrese el número n1: ");
6      int n1 = scanner.nextInt();
7      System.out.println("Ingrese el número n2: ");
8      int n2 = scanner.nextInt();
9      System.out.println("Ingrese la operación: ");
```

```
10      int op = scanner.nextInt();
11
12      if (op == 1) {
13        int sum = n1 + n2;
14        System.out.println("La suma vale: " + sum);
15      }
16      else if (op == 2) {
17        int sub = n1 - n2;
18        System.out.println("La resta vale: " + sub);
19      }
20      else if (op == 3) {
21        int mult = n1 * n2;
22        System.out.println("La multiplicación vale: " + mult);
23      }
24      else if (op == 4) {
25        double div = ((double) n1) / ((double) n2);
26        System.out.println("La división vale: " + div);
27      }
28      else {
29        System.out.println("Operación incorrecta");
30      }
31
32    }
33  }
```

En el ejemplo guardamos dos números en las variables `n1` y `n2`. Además guardamos el número que representa en la operación en `op`. El caso especial es cuando `op` vale 4, porque el resultado es de tipo `double`. También desplegamos un mensaje si el número de la operación es incorrecto.

5. Uso de operadores lógicos

Hasta ahora hemos visto que las instrucciones `if` - `else if` - `else` nos permiten ejecutar ciertos bloques de código dependiendo si se cumplen o no determinadas condiciones. Vimos también, que estas condiciones tenían que ser expresiones booleanas y que resultaban de la comparación entre números o textos. Sin embargo, es posible generar y combinar expresiones más complejas con **operadores lógicos**.

Es posible validar si algo no se cumple (`!` - not), si dos condiciones se cumplen simultáneamente (`&&` - and) o si se cumple al menos una de dos condiciones (`||` - or). Estos operadores tienen su origen en la lógica de predicados y permiten operar valores de verdad (variables de tipo `boolean`) para generar nuevos valores de verdad.

Muchas veces no nos basta con una condición simple, sino que queremos alguna combinación de varias condiciones. Por ejemplo, en el caso de rendir un examen de conducir, el postulante debe ser mayor de 18 años o ser mayor de 17 y acreditar un curso en una escuela de conductores. También podemos tomar decisiones en base a condiciones que no se cumplen. Por ejemplo, si no tengo dinero suficiente para comprar una entrada al cine, no podré entrar a verla a una sala de cine, y deberemos buscar otras opciones.

5.1. Negación - !

El primer operador lógico es la negación (`!` - not). Este operador invierte el valor de verdad de una variable de tipo boolean: Si es `false` devuelve `true` y viceversa. Para usarlo se debe anteponer un

signo de exclamación (`!`) al boolean que se desea negar. Por ejemplo (suponiendo que una entrada al cine vale $3000):

```
1  int dinero = 2000;
2  boolean suficienteDinero = dinero > 3000;
3  boolean insuficienteDinero = !suficienteDinero;
```

En este caso en la línea 2 se asigna a la variable `suficienteDinero` el valor `false`, ya que dinero tiene asignado el valor `2000` y la comparación `2000 >3000` retorna `false`. En la línea 3 se asigna el valor `true` a la variable `insuficienteDinero`, ya que `suficienteDinero` tiene valor `false` y al anteponer el `!` su valor de verdad será el opuesto, quedando la variable con valor `true`.

Es importante tener en cuenta que no solo se pueden negar variables, sino que cualquier expresión que sea booleana. En el caso anterior se podría haber hecho lo siguiente:

```
1  int dinero = 2000;
2  boolean insuficienteDinero = !(dinero > 3000);
```

En este caso, la comparación `2000 >3000` retorna `false` y entonces, al anteponer `!` se obtiene el valor `true`. Finalmente se le asigna este valor (`true`) a la variable insuficiente dinero.

5.2. Y - &&

El siguiente operador lógico es (`&&` - and), que también es llamado conjunción. Este evalúa dos expresiones y nos dice si ambas son ciertas. Por ejemplo, puedo comprar una entrada para ver una película si:

1. Quedan asientos disponibles
2. Tengo dinero suficiente

Si alguna de las condiciones no se cumple, entonces no puedo comprar la entrada.

Este operador funciona de la siguiente forma: el operador evalúa el valor de verdad de dos variables de tipo `boolean`: si ambos son `true` devuelve `true` y en cualquier otro caso retorna `false`. El cuadro 1 muestra el resultado de evaluar `&&` con dos variables `a` y `b`.

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

Cuadro 1: resultado de evaluar `a && b`

Supongamos ahora que tenemos una sala de cine con 40 asientos y la entrada tiene un valor de $3000:

```
1  int asientosVendidos = 20;
2  int dinero = 4000;
3  boolean posibleComprar = (asientosVendidos < 40) && (dinero > 3000)
```

En el ejemplo anterior (`asientosVendidos < 40`) va a tener valor `true`, al igual que la expresión (`dinero > 3000`). Luego `true && true` tiene valor `true` y ese es el valor que se asigna a la variable `posibleComprar`.

5.3. O - ——

El siguiente operador lógico es (`||` - or), también llamado conjunción. Este operador evalúa dos expresiones y nos dice si al menos alguna de las 2 es verdad (pueden ser ambas verdaderas). Por ejemplo, iré al cine si:

1. Están dando la película Avengers: Infinity War, o bien
2. Están dando la película Interstellar[2]

Si cualquiera de las condiciones se cumple, entonces voy a ir al cine.

Este operador funciona de la siguiente forma: el operador evalúa el valor de verdad de dos variables de tipo `boolean`: si alguna de ellas es `true`, entonces devuelve el valor `true`. Este operador solamente retorna `false` cuando ambas variables eran `false`. El cuadro 2 muestra el resultado de evaluar `||` con dos variables `a` y `b`.

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Cuadro 2: resultado de evaluar `a || b`

Por ejemplo, supongamos un cine que solamente está dando la película Interstellar:

```
1 String cartelera = "Interstellar";
2 boolean voyAlCine = cartelera.equals("Interstellar") ||
3                     cartelera.equals("Avengers: Infinity War");
```

En el ejemplo anterior, `cartelera.equals(.Avengers: Infinity War")` se evalúa con el valor `false`, mientras que `cartelera.equals(Ïnterstellar")` va a tener el valor `true`. Luego `false || true` tiene el valor `true`. Por lo tanto, la variable `voyAlCine` tiene el valor `true`.

6. Ejemplos de operadores lógicos

Ejemplo 6.1. En este ejercicio vamos a determinar si un número está dentro de un rango. Suponga que va a recibir tres números, `a`, `b` y `c`. Su programa debe determinar si el número `c` está entre los otros dos (incluyéndolos). Puede asumir que $a \leq b$.

```
1 public class Main {
2   public static void main(String[] args) {
3     Scanner scanner = new Scanner(System.in);
```

[2]Que para uno de los autores de este apunte, es la mejor película de la historia.

```
4       System.out.println("Ingrese a:");
5       int a = scanner.nextInt();
6       System.out.println("Ingrese b:");
7       int b = scanner.nextInt();
8       System.out.println("Ingrese c:");
9       int c = scanner.nextInt();
10
11      if (a <= c && c <= b) {
12        System.out.println("c está entre a y b");
13      }
14      else{
15        System.out.println("c no está entre a y b");
16      }
17    }
18  }
```

En este ejemplo notamos que en la línea 11 se está haciendo uso del operador `&&` (and). En ese momento se verifica que `c` sea mayor o igual que `a` y además que sea menor o igual que `b`.

Ejemplo 6.2. En este ejercicio vamos representar la situación de la introducción. Haremos un programa que reciba la edad de una persona y un string que representa si la persona viene en compañía de un adulto (puede tener valores "sí" o "no"). El programa debe imprimir un mensaje que anuncie si la persona puede entrar o no al cine. La persona puede entrar al cine si es mayor de 17 años o bien si es mayor de 15 y va en compañía de un adulto. Primero resolveremos el ejercicio sin operadores lógicos, y luego con operadores lógico, para mostrar la mejora en el código.

6.1. Solución sin operadores lógicos

```
1   public class Main {
2     public static void main(String[] args) {
3       Scanner scanner = new Scanner(System.in);
4       System.out.println("Ingrese su edad:");
5       int edad = scanner.nextInt();
6       scanner.nextLine();
7       System.out.println("Viene junto a un adulto? (sí/no)");
8       String vieneAdulto = scanner.nextLine();
9
10      if (edad > 17){
11        System.out.println("Puede entrar al cine");
12      }
13      else if (edad > 15) {
14        if (vieneAdulto.equals("sí")) {
15          System.out.println("Puede entrar al cine");
16        }
17        else {
18          System.out.println("No puede entrar al cine");
19        }
20      }
21      else {
22        System.out.println("No puede entrar al cine");
23      }
24    }
25  }
```

6.2. Solución con operadores lógicos

```
1  public class Main {
2    public static void main(String[] args) {
3      Scanner scanner = new Scanner(System.in);
4      System.out.println("Ingrese su edad:");
5      int edad = scanner.nextInt();
6      scanner.nextLine();
7      System.out.println("Viene junto a un adulto? (sí/no)");
8      String vieneAdulto = scanner.nextLine();
9
10     if ((edad > 17) || ((edad > 15) && vieneAdulto.equals("sí"))) {
11       System.out.println("Puede entrar al cine");
12     }
13     else{
14       System.out.println("No puede entrar al cine");
15     }
16   }
17 }
```

Vemos que en la línea 11 hay un `if` cuya condición es el resultado de aplicar muchos operadores lógicos. Los paréntesis nos permiten agrupar las expresiones. En este caso tenemos 2 expresiones que se combinan mediante el operador `or`:

1. `(edad >17)`
2. `(edad >15) && vieneAdulto.equals("si")`

Basta que se cumpla 1) o 2) para que la expresión sea `true` y entre al `if`. Para ver si 1) es true se aplica la comparación `edad > 17`. Mientras tanto para 2) se descompone a su vez en 2 expresiones que se combinan mediante el operador `and`:

1. `(edad >15)`
2. `vieneAdulto.equals("si")`

Para que sea `true` en este caso, 1) y 2) deben serlo: la edad debe ser supeior a `15` y `vieneAdulto` debe tener el valor `sí¨`.