Ministerio de Producción
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos

Programa 111 mil
VOS PODÉS SER UNO.

Estructuras de Control

Agenda

- Operadores relacionales
- Operadores condicionales
- Estructuras de control
  - Sentencias de selección: if, if-else, switch
  - Sentencias de iteración: while y for

Operadores Relacionales y Condicionales

- Los operadores relacionales comparan dos valores del mismo tipo para determinar si son iguales o distintos, si uno es mayor o menor que el otro. Una expresión que usa un operador relacional es una expresión lógica, por lo tanto el resultado es un valor de tipo boolean (true o false).

- Los operadores condicionales u operadores lógicos permiten combinar los resultados de múltiples expresiones lógicas. El resultado es también un valor de tipo boolean (true o false).

Operadores Relacionales: ==, !=, <

Operador	Nombre	Ejemplo
==	Igual a	`int i = 1; (i == 1)` **(TRUE)** `Empleado e1= new Empleado(); Empleado e2 = new Empleado(); e1 == e2;` **(FALSE)**

!=	Distinto a	`int i = 2; (i != 1)` **(TRUE)**
		`Persona[] empleados = new Persona[10];`
		`empleados[0].setSalario(1000);`
		`empleados[1].setSalario(1200);`
		`empleados[0].getSalario() != empleados[1].getSalario()` **(TRUE)**

<	Menor que	`Persona[] empleados = new Persona[10];`
		`empleados[0].edad = 40; empleados[1].edad=50;`
		`empleados[0].edad < empleados[1].edad` **(TRUE)**

Operadores Relacionales: <=, >, >=

Operador	Nombre	Ejemplo

<=	Menor o igual que	`Empleado e1= new Empleado(); Empleado e2 = new Empleado(); Empleado e3 = new Empleado();`
		`e1.setSalario(1000); e2.setSalario(1200);`
		`e3.setSalario(1200);`
		`e2.getSalario() <= e1.getSalario();` **(FALSE);**
		`e2.getSalario() <= e3.getSalario()` **(TRUE)**

>	Mayor que	`char[] caracteres = new char[3]; caracteres[0] = 'a'; caracteres [1]='Z'; caracteres[2]='Z';`
		`caracteres[0] > caracteres[1]à false;`
		`caracteres[2] > caracteres[0]` **(TRUE);**
		`caracteres[0] >= caracteres[2] à false;`
		`caracteres[1] >= caracteres[2]` **(TRUE)**

>=	Mayor o igual que	

Operadores Condicionales o Lógicos

Operador	Nombre	Significado	Ejemplo

&& (circuito corto)
&(circuito largo)
	AND lógico	Devuelve **true** si TODAS las condiciones son verdaderas y **false** si al menos una es falsa	`Persona[] empleados = new Persona[10];`
			`empleados[0].edad = 40;`
			`empleados[0].nombre="Jose";`
			`(empleados[0].getEdad()==40) &&`
			`(empleados[0].getNombre.equals("Jose"))`
			(TRUE)

|| (circuito corto) | (circuito largo)	OR lógico	Devuelve **false** si TODAS las condiciones son falsas y **true** si al menos una es verdadera	`int i = 2; int j = 7;` `((i < 3) | (j > 10))` **(TRUE)**
!	NOT lógico	Devuelve **false** si la condición en verdadera y **true** si es falsa	`int i = 2; (!(i < 3))` **(FALSE)**

Operadores Condicionales o Lógicos

La diferencia entre el **circuito corto** y el **circuito largo** es la forma de evaluación de las condiciones. En el **circuito largo se evalúan todas las condiciones**, mientras que en el **circuito corto** alcanza con evaluar solo una.

```
String[] nombres = {"Juan", "Pedro", "Julia", "Elena"};
int i=4;

((i==nombres.length)&&(nombres[i].equals("Manuel")))
```

¿Cuál es el resultado de esta evaluación?
false
Evalua solamente la primera condición

```
((i==nombres.length)&(nombres[i].equals("Manuel")))
```

¿Cuál es el resultado de esta evaluación?
Se produce un error en ejecución porque nombres[4] no está definido
Evalua todas las condiciones

Sentencias de Selección

- Las **sentencias de selección** nos permiten especificar condiciones para cambiar el flujo de control en un programa.

  - **if**: ejecuta un bloque de código si la evaluación de determinada expresión lógica es verdadera (true).
  - **if/else**: ejecuta un bloque de código si la evaluación de determinada expresión lógica es verdadera (true), y otro bloque diferente de código si el resultado es falso (false).
  - **switch**: es un if con multi bifurcaciones. Ejecuta diferentes bloques de código de acuerdo a los múltiples valores diferentes que puede tomar una variable.

La sentencia IF

- Forma general de la sentencia if:

```
if (expresión_lógica){
    sentencias;
}
```

Si la **expresión lógica** es verdadera se ejecutan las sentencias del bloque de código, también llamado **cuerpo** del if. Si es falsa, se saltea el cuerpo.

La sentencia IF: Ejemplo

- Tenemos que implementar una clase llamada **DivisionSegura**, que dados 2 números los divide solo si el denominador es distinto de cero.

```
public class DivisionSegura {
  public static void main(String args[]){
  int x = Integer.parseInt(args[0]);
  int y = Integer.parseInt(args[1]);
  int z=0;
  if( y !=0 ) z = x / y;
  System.out.println("El resultado es : " + z);
  }
}
```

Es posible omitir las llaves en la sentencia if cuando el cuerpo tiene solamente una sentencia.

Esta expresión lógica verifica si el valor de la variable **y** es distinto de **0**, en cuyo caso hace la división

La sentencia IF-ELSE

- Forma general de la sentencia if/else:

```
if (expresión_lógica) {
    sentenciasA;
}else{
   sentenciasB;
}
```

Si la **expresión lógica** es verdadera se ejecuta el bloque de código **sentenciasA** y si es falsa, se ejecuta el bloque de código **sentenciasB**.

La sentencia IF-ELSE: Ejemplo

- Ejemplo **DivisionSegura**

```
public class DivisionSegura {
    public static void main(String args[]){
   int x = Integer.parseInt(args[0]);
   int y = Integer.parseInt(args[1]);
   int z = 0;
   if ( y !=0 ) {
        z = x / y;
        System.out.println("El resultado es : " + z);
   }
   else
        System.out.println("Atención! se pretende dividir por 0");
   }
}
```

¿Qué resultado obtengo si ejecuto **DivisionSegura 4 2**?

El resultado es: 2

¿Y si ejecuto **DivisionSegura 4 0**?

Atención! Se pretende dividir por 0.

Esta expresión lógica evalúa si el valor de la variable **y** es distinto de **0**, en cuyo caso hace la división e imprime el resultado en pantalla; en caso contrario (el valor de y es igual a **0**) imprime un mensaje de advertencia

La sentencia IF-ELSE: Ejemplo (2)

- Ejemplo con operadores condicionales e if anidados:

```
if ((( año % 4 == 0 ) && ( año % 100 != 0 )) || ( año % 400 == 0 )) {
  System.out.println("Es bisiesto"); }
else {
  System.out.println("No es bisiesto");
 }
```

Operadores condicionales o lógicos

El operador % retorna el resto de la división entera. Ej: 5 % 4 retorna 1.

La sentencia IF-ELSE: Ejemplo (3)

- Ejemplo con if anidados:

```
public class DivisionSegura {
  public static void main(String args[]){
  int x = Integer.parseInt(args[0]);
  int y = Integer.parseInt(args[1]);
  int z = 0;
  if( y !=0 ){
    if (y > 0)
        System.out.println("Estoy dividiendo por un número positivo");
    else
        System.out.println("Estoy dividiendo por un número negativo");
    z = x / y;
    System.out.println("El resultado es : " + z);
  }
  else {
    System.out.println("Atención! se pretende dividir por 0");}
  }
}
```

Sentencias if anidadas

Anidamiento IF-ELSE

```
public class DivisionSegura {
  public static void main(String args[]){
    if (args.length == 0) System.out.println("Falta pasar 2 argumentos");
    else if (args.length == 1) System.out.println("Falta pasar un argumento");
    else {
        int x = Integer.parseInt(args[0]);
        int y = Integer.parseInt(args[1]);
        int z = 0;
        if( y !=0 ) {
            z = x / y;
            System.out.println("El resultado es : " + z);
        }
         else
            System.out.println("Atención! se pretende dividir por 0");
      }
 }
```

Es importante la indentar el código cuando se usan if anidados, mejora la legibilidad

Uso de operadores condicionales - IF-ELSE

```
public class Persona {
    private String apellido = "";
    private String nombre = "";
    private String ocupacion = "";
    private int edad;
    // setter y getters

}
```

```
public class TestPersona {
public static void main(String[] args) {
    Persona p = new Persona();
    p.setApellido("Gomez");
    p.setEdad(50);
    p.setOcupacion("Ingeniero");
    p.setNombre("Jorge");
    Persona p2 = new Persona();
    p2.setApellido("Garcia");
    p2.setEdad(50);
    p2.setOcupacion("Arquitecto");
    p2.setNombre("Jorge");
```

```
        if ((p.getApellido().equals(p2.getApellido()))&&(p.getEdad()==p2.getEdad())
            System.out.println("Tienen el mismo apellido y la misma edad");
        else
            System.out.println("El apellido es distinto  y/o no tienen la misma edad");
        }
    }
```

AND Lógico (circuito corto)

¿Qué imprime?

El apellido es distinto y/o no tienen la misma edad

La sentencia Switch

- Forma general de la sentencia switch:

```
switch (variable) {
  case selector:
  {sentencias;
  break;
  }
  case selector:
   {sentencias;
  break;
  }
  case selector:
   {sentencias;
  break;
  }
    ….
   default:
  {sentencias;
  break;
  }
}
```

variable: es la variable cuyo valor se quiere testear. No puede ser de cualquier tipo. Los tipos permitidos son: char, byte, short o int y enumerativos

case selector: representa un valor posible de la variable. La combinación de **case** con un valor literal se conoce como **etiqueta case**. Se debe tener una etiqueta case por cada valor que se desea testear. Esta cláusula es obligatoria.

Los valores literales NO pueden ser: Variables, Expresiones, Llamadas a métodos

Los valores literales pueden ser: Constantes, Literales (por ejemplo: ‘A’ o 10)

default: indica el bloque de código de default que se ejecutará si no hay coincidencia con alguno de los case establecidos. La cláusula default es opcional.

break: esta sentencia indica que el switch termina y el control pasa a la sentencia siguiente al switch. Aunque es opcional, en términos prácticos siempre se incluye la sentencia break al final de cada case.

La sentencia Switch: ejemplo

- Tenemos una variable de tipo char que almacena el talle de una remera y dependiendo de su valor imprimimos diferentes mensajes en la pantalla.

```
char talle = 'm';
switch (talle) {
  case 's':
     {System.out.println("La remera es Small");
     break;}
  case 'm':
    {System.out.println("La remera es Medium");
     break;}
  case 'l':
    {System.out.println("La remera es Large");
     break;}
   default:
     {System.out.println("La remera es Extra
Large");}
}
```

if talle == ´s´

if talle == ´m´

if talle == ´1´

if (talle !== ´s´) & (talle !== ´m´) & (talle !== ´l´)

¿Qué imprime?

La remera es Medium

La remera es Large

La remera es Extra Large

¿Por qué imprime esto?

Porque cuando un caso coincide si no se pone la sentencia **break** ejecuta los restantes casos hasta el final del switch.

¿Si agregamos los break?

La remera es Medium

Algunas restricciones del switch

- Sólo se pueden testear igualdades
- No se puede usar para realizar un comparación del tipo *“es mayor que”*, *“es menor que”*, *“es mayor o igual que”*, *“es menor o igual que”* (>, <, >=, <=)
- No se puede usar para comparar contra más de un valor
- Funciona con tipos primitivos: char, byte, short, char, int. Wrappers: Character, Byte, Short, and Integer. Adicionalmente String (Java 7+)

La sentencia de iteración - WHILE

- Un bucle **while** itera un bloque de código mientras una condición es verdadera.
- Forma general de la sentencia **while**:

```
while (expresión_lógica) {
  bloque de código
}
```

while
es un bucle iterativo del tipo cero a muchas veces

expresión_lógica: toma el valor **true** o **false** y es evaluada **antes** de cada iteración.

bloque de código: representa las líneas de código que son ejecutas si la **expresión lógica es verdadera**.

WHILE: Ejemplo

```
int i = 0;
while (i < 5){
    System.out.println(i +" )Hola Mundo");
    i++;
}
System.out.println("fin");
```

¿Cuál es la salida en pantalla?

1) Hola Mundo

2) Hola Mundo

3) Hola Mundo

4) Hola Mundo

5) Hola Mundo

fin

WHILE: Ejemplo (2)

```
int i = 0;
 String[] palabras = {"Norte", "Sur", "Este", "Oeste"};
 System.out.println("Los puntos cardinales son:");
 while (i < 4){
        System.out.println(i +" )" + palabras[i]);
        i++;
 }
```

¿Cuál es la salida en pantalla?

Los puntos cardinales son:

1) Norte

2) Sur

3) Este

4) Oeste

WHILE: Ejemplo (3)

```
int i = 0;
while (i<5){
    System.out.println("Hola Mundo");
    i++;
}
```

- **¿Qué sucede si omito poner la instrucción i++?**
  - Se transforma en un bucle infinito
- **¿Qué sucede si modifico la primer línea por i=5 ?**
  - No se ejecuta nunca el cuerpo
- **¿Y si se modifica expresión lógica del while (i< 5) por while (2==2)?**
  - Se transforma en un bucle infinito
- **¿Qué ocurre si se modifica la expresión de while (i< 5) por while (true)?**
  - Se transforma en un bucle infinito

Sentencias WHILE anidadas

- Supongamos que queremos dibujar el siguiente rectángulo:

```
######
######
######
```

```
int fila = 0;
while (fila < 3){
      int columna = 0;
      while (columna < 6) {
         System.out.print("#");
         columna ++;
      }
   System.out.println();
   fila++;
}
```

Las sentencias while anidadas son útiles para recorrer / crear / modificar estructuras de multidimensionales (Ej., matrices)

La sentencia de iteración - FOR

- Un bucle `for` itera un bloque de código una cantidad **predeterminada** de veces.
- Forma general de la sentencia **for**:

```
for (inicializador; expresión_lógica; modificador) {
      bloque de código
}
```

for
es un bucle iterativo del tipo cero a muchas veces

inicializador: contiene las sentencias que inicializan las variables que funcionan como contadores del bucle. El inicializador es ejecutado solamente una vez antes que cualquier otra parte del bucle.
expresión_lógica: es una expresión que devuelve **true** o **false**. Es procesada cada vez antes de cada iteración.
modificador: contiene las variables del bucle (contadores del bucle) que son incrementadas o decrementadas. Es procesada después del cuerpo pero antes de la verificación de la expresion_lógica.

FOR: Ejemplo

```
String[] palabras = {"Norte", "Sur", "Este", "Oeste"};
for(int i = 0; i < 4; i++){
     System.out.println(i +" )" +palabras[i]);
}
System.out.println("fin");
```

La salida por consola de la instrucción es:

1) Norte

2) Sur

3) Este

4) Oeste

fin

- Una forma más segura de escribir lo mismo:

```
String[] palabras = {"Norte", "Sur", "Este", "Oeste"};
for(int i = 0; i < palabras.length; i++){
  System.out.println(i +" )" +palabras[i]);
}
System.out.println("fin");
```

Sentencias FOR anidadas

- Supongamos que queremos dibujar el siguiente rectángulo:

```
######
######
######
```

```
for(int fila=0;fila<3; fila++)
    for (int columna=0;columna<6;columna++) {
        System.out.print("#");
    }
    System.out.println();
}
```

De forma similar al WHILE, los FOR anidados nos permiten recorrer estructuras multidimensionales. El código queda más compacto utilizando FOR

FOR: Ejemplo (2)

```
for (int i = 0; i < 5; i++){
    System.out.println("Hola Mundo");
}
```

- **¿Qué sucede si modifico i++ por i--?**
  - Se transforma en un bucle infinito
- **¿Qué sucede si modifico el inicio i=0 por i=5 ?**
  - No se ejecuta nunca el cuerpo
- **¿Y si se modifica expresión (i< 5) por (2==2)?**
  - Se transforma en un bucle infinito
- **¿Qué ocurre si se reemplaza la expresión (i< 5) por el literal true?**
  - Se transforma en un bucle infinito

La sentencia de iteración - FOREACH

- Un bucle **`foreach`** itera un bloque de código sobre cada uno de los elementos de una lista de manera secuencial.
- Forma general de la sentencia **foreach**:

```
for (variable : colección) {
      bloque de código
}
```

inicializador: contiene el Tipo y nombre de la variable donde se vinculan secuencialmente los elementos de la colección

colección: puede ser una colección iterable o un arreglo

FOREACH: Ejemplo

```
int i = 0;
String[] palabras = {“Norte”, “Sur”, “Este”, “Oeste”};
System.out.println(“Los puntos cardinales son:”);
for (String palabra : palabras) {
      System.out.println(i +“ )” + palabra);
      i++;
}
```

¿Cuál es la salida en pantalla?

Los puntos cardinales son:

1) Norte

2) Sur

3) Este

4) Oeste