Capítulo 4

Operadores

Java World

¿Querés ser un SCJP?

Revista Digital - Hecha por estudiantes para estudiantes

Operadores

Los operadores son aquellos elementos que producen un resultado realizando operaciones con uno o varios operandos (estos son los elementos que se encuentran a la izquierda y derecha del operador).

Operadores de asignación

El operador de asignación es el **=**. Algunos conceptos a tener en cuenta:

- Cuando se le asigna un valor a una primitiva, el tamaño si importa. (Es importante saber cuando ocurrirá un casteo implícito, explícito, y cuando se trunca el valor).
- Una variable de referencia no es un objeto, es una manera de obtener el objeto al que hace referencia.
- Cuando se asigna un valor a una referencia, el tipo importa. (Es importante saber las reglas para superclases, subclases, y arrays).

Operadores de asignación compuestos

Actualmente, existen alrededor de 11 operadores de asignación compuestos, pero en el examen solo aparecerán los 4 más comunes (**+=**, **-=**, ***=**, **/=**) se encuentran en el examen.
Pero, que es un operador de asignación compuesto?
Simplemente, es una manera abreviada de realizar una operación con el valor de la variable en donde se asigna el resultado con otro

```
int a = 2;
a = a * 3;

//Diferente sintaxis, mismo significado.
int a = 2;
a *= 3;
```

conjunto. Veamos un ejemplo:

```
int a = 2;
int b = 3;
int c = 5;

a = a * ( b + c );

//Diferente sintaxis, mismo significado.
int a = 2;
int b = 3;
int c = 5;
a *= b + c;
```

A simple vista, parece sencillo, pero no hay que confundirse. El operador de asignación compuesto se refleja en realizar una opeación con el resultado de todo lo que se encuentra luego del igual. Veamos un ejemplo:

Es posible confundirse con esto, creyendo que la operación fuera " (a * b) + c", pero reitero, el operador de asignación compuesto realiza los siguientes pasos:

1. Realiza todas las operaciones contenidas a la derecha del operador de asignación (**=**).
2. Realiza la operación compuesta (Multiplicar, Dividir, Sumar, Restar, ...).
3. Asigna el valor de la operación a la variable.

Operadores relacionales

El examen cubre sies operadores relaciones (**<**, **<=**, **>**, **>=**, **==** y **¡=**).
Los operadores relacionales siempre producen como resultado un valor de tipo **boolean**.
Los operadures se traducen como:

- `<` menor que
- `<=` menor o igual que
- `>` mayor que
- `>=` mayor o igual que
- `==` igual que
- `¡=` distinto de

Es muy común ver en una comparación un operador `=`. Esto es una asignación, no un operador relacional. Como dijimos, estos tipos de operadores devuelven un valor boolean, asi que pueden ser utilizados en sentencias condicionales (`if`, `else`, `while`), como ser asignados a una variable.

Veamos algunos ejemplos:

```
int x = 7;
if ( x < 9 ) {
  //Codigo ...
} else {
  //Codigo ...
}
//Entra en el if

boolean a = 100 > 90;
if ( a ) {
  //Codigo ...
} else {
  //Codigo ...
}
//Entra en el else
```

Ahora, veamos un ejemplo un poco más complicado:

```
int x = 12;
if ( (x = 2 * 3 ) == 6 ) {
  //Codigo ...
} else {
  //Codigo ...
}
```

En este código, se ejecuta el `if`.
Los pasos que realizó el programa fueron:

1. Realizar la operación “2 * 3”.
2. Asignar el resultado de la operación anterior a x.
3. Comparar el valor de x con el literal 6.

Vuelvo a recalcar la importancia de entender la diferencia entre `=` e `==`, y saber identificar cuando podría haber un posible error.
Quiero mostrar otro ejemplo, en donde se aprecia aun más la importancia de detectar las diferencias entre dichos operadores:

```
boolean x = false;

if ( x = true ) {
  //Codigo ...
} else {
  //Codigo ...
}
```

El resultado de el código anterior ejecuta el código dentro del `if`.
¿Cómo? Simple:

1. x es false.
2. Dentro de la condición, se asigna true a x.
3. Se evalua x (este ahora vale true).

Hay que recordar que una sentencia de condición, en Java, solo evalua los tipos boolean. Es válido asignar un valor a una variable dentro de una condición, pero si el tipo de dato almacenado no es de tipo boolean, se generará un error de compilación. De manera que una sentencia como "if (x = 9)" generará un error de compilación.

No es lo mismo que:

```
boolean x = false;

if ( x == true ) {
  //Codigo ...
} else {
  //Codigo ...
}
```

En este código se ejecuta el **else**.
También es posible realizar una comparación como:

```
char a = 7;
if ( a < 'g' ) {
  //Codigo ...
} else if ( a < 16 ) {
  //Codigo ...
}
```

Cuando se compara un valor de tipo **char**, se utiliza el valor unicode del caracter.

Operadores de igualdad

Los operadores de igualdad son:

- **==** Igual que
- **¡=** Distinto de

Estos operadores solo comparan el patrón de bits. Esto es importante dado que en una variable de tipo referencia, el valor es la posición de memoria donde se encuentra el objeto.

Igualdad en primitivas

Algunos ejemplos:

```
'a' == 'a'              //TRUE
'a' == 'b'              //FALSE
7 != 9                  //TRUE
1.0 == 1L               //TRUE (se realiza un casteo implícito)
true == false           //FALSE
```

Igualdad en variables de tipo referencia

Cuando se compara una referencia con los operadores de igualdad, solo se compara el objeto al que referencian (el valor de la variable es la dirección de memoria en donde se encuentra el objeto realmente).

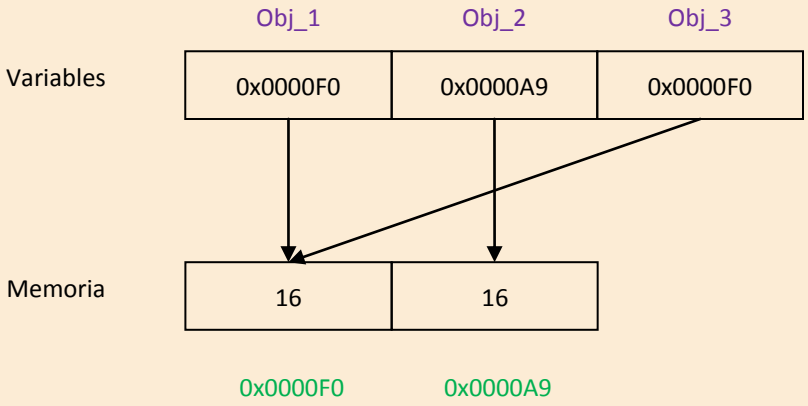
```
class Simple {
  int x;
  Simple(int x){ this.x = x; }
}

public static void main(String[] args) {
  Simple obj_1 = new Simple(16);
  Simple obj_2 = new Simple(16);
  Simple obj_3 = obj_1;

  System.out.println("Objeto 1 es igual a objeto 2?  " + (obj_1 == obj_2));
  System.out.println("Objeto 1 es igual a objeto 3?  " + (obj_1 == obj_3));
}
```

```
Objeto 1 es igual a objeto 2?  False
Objeto 1 es igual a objeto 3?  True
```

A pesar de que el único atributo que tiene el objeto Simple es igual en obj_1 como en obj_2, su comparación de igualdad da false. Vamos a hacerlo un poco más gráfico para dejar este importante concepto bien en claro.



En el gráfico podemos apreciar facilmente el porque la comparación entre obj_1 y obj_2 fallo, cuando entre obj_1 y obj_3 fue exitosa. Más alla de que obj_1 y obj_2 contentan su único atributo con el mismo valor.

Igualdad en enumeradores

Se aplica la política de variables de referencia. Al final, un enumerador es un objeto.

Comparación con instanceof

El operador **instanceof** devuelve un valor de tipo boolean, que indica si el objeto Es-Un objeto del tipo especificado (o subtipo). También aplica a interfaces, cuando el objeto Implementa una interfaz especificada (o alguna de sus superclases la implementa).

El operador **instanceof** solo funciona para comparar tipos que se encuentren en el mismo árbol de herencia.

Veamos un ejemplo de esto:

```
interface Dibujable {}
class Triangulo implements Dibujable {}
class Cuadrado extends Triangulo {}
class Circulo implements Dibujable {}
```

Tipo	Se puede comparar con	No se puede comparar con
null	Cualquier clase o interface	-
Dibujable	Dibujable – Object – Circulo – Triangulo – Cuadrado	-
Circulo	Circulo – Dibujable – Object	Triangulo – Cuadrado
Triangulo	Cuadrado – Triangulo – Dibujable – Object	Circulo
Cuadrado	Cuadrado – Triangulo – Dibujable – Object	Circulo

http://thechroniclesofbinary.googlecode.com/files/ejemploInstanceof.java

Para compilarlo: Javac -g ejemploInstanceof.java

Para ejecutarlo: java ejemploInstanceof

Operadores aritméticos

No hay mucho que decir acerca de estos, salvo que todos los utilizamos.

El operador de resto

Cualdo realizamos una operación de división, existe un resto. Este valor puede ser obtenido a partir del operador %.

```
public class ejemploResto {
  static public void main(String[] arg) {
    System.out.println("9 dividido 2 es " + (9 / 2) + ". Y el resto es " + (9 % 2));
  }
}
```

Un ejemplo del operador:

```
9 dividido 2 es 4. Y el resto es 1.
```

El operador de concatenación

El operador + también puede ser utilizado para concatenar cadenas de caracteres.

La regla para saber cuando se comporta como suma y cuando como concatenación es simple: Si alguno de los dos valores es un String, el + funciona como concatenador.

Algunos ejemplos:

```
System.out.println(67 + 45 + 9);                    //121
System.out.println(67 + 45 + " - valores" + 67 + 45);    //112 - valores6745
System.out.println("hola" + " " + "mundo");              //hola mundo
```

Operadores de incremento y decremento

Estos operadores sirven para incrementar o decrementar en 1 el valor de cualquier variable numérica.

http://thechroniclesofbinary.googlecode.com/files/ejemploOperadoresIncrementoDecremento.java
Para compilarlo: Javac -g ejemploOperadoresIncrementoDecremento.java
Para ejecutarlo: java ejemploOperadoresIncrementoDecremento

En el ejemplo se puede ver como el operador de incremento y decremento es aplicado a todos los valores de tipo numérico, y como funciona el de preincremento y posincremento.

Existe una gran diferencia entre los operadores de preincremento y posincremento.

- Preincremento ++variable Antes de realizar cualquier acción, el valor de la variable es incrementado en 1. Luego continúa con la lógica.
- Posincremento variable++ Primero realiza la lógica, y al finalizar incrementa el valor de la variable en 1.

Veamos un ejemplo:

```
int x;

x = 0;
System.out.println(x++ + 6);  //6
System.out.println(x);        //7

x = 0;
System.out.println(++x + 6);  //7
System.out.println(x);        //7
```

Hay que tener cuidado en codigos de condición complejos, veamos algunos ejemplos:
http://thechroniclesofbinary.googlecode.com/files/ejemploIncrementoDecrementoCondicionales.java
Para compilarlo: Javac -g ejemploIncrementoDecrementoCondicionales.java
Para ejecutarlo: java ejemploIncrementoDecrementoCondicionales

```
int x, y, z;

x = 0;y = 2;z = 2;
if ( (x++ == y) || (z == ++y) ){
  System.out.println("( (x++ == y) || (z == ++y) ) ? true");
} else {
  System.out.println("( (x++ == y) || (z == ++y) ) ? false");
}
//Resultado: false

x = 0;
if ( (x++ == ++x) ) {
  System.out.println("( (x++ == ++x) ) ? true");
} else {
  System.out.println("( (x++ == ++x) ) ? false");
}
//Resultado: false

x = 0;
if ( (++x == x++) ) {
  System.out.println("( (++x == x++) ) ? true");
} else {
  System.out.println("( (++x == x++) ) ? false");
}
//Resultado: true

x = 0;y = 1;z = 2;
if ( (--z > y) && (x == z) ) {
  System.out.println("( (--z > y) && (++x < z) ) ? true");
} else {
  System.out.println("( (--z > y) && (++x < z) ) ? false");
}
//Resultado: false

x = 2;y = 1;z = 2;
if ( (--z > y) && (x > z) ) {
  System.out.println("( (--z > y) && (x > z) ) ? true");
} else {
  System.out.println("( (--z > y) && (x > z) ) ? false");
}
//Resultado: false
```

Veamos a fondo uno por uno:

```
x = 0;y = 2;z = 2;
if ( (x++ == y) || (z == ++y) ){ //...

( (0 == 2) || (2 == 3) ) //false || false
```

```
x = 0;
if ( (x++ == ++x) ) { //...

( (0 == 1) ) //false
```

```
x = 0;
if ( (++x == x++) ) { //...

( (1 == 1) ) //true
```

```
x = 2;y = 1;z = 2;
if ( (--z > y) && (x > z) ) { //...

( (1 > 1) && (2 > 1) ) ) //false && true
```

Operador condicional

Este operador requiere tres términos, y es utilizado para la asignación de valores a una variable.
Su funcionamiento es similar al de un **if**, verifica una condición, y si esta es true asigna un valor, caso contrario asigna la otra.
Ejemplo de la sintaxis:

```
variable = (expresion booleana) ? valor a asignar si la condicion fue true : valor a
asignar si la condicion fue false;

int x;
x = ( true )  ? 1 : 2;  //x ahora contiene el valor 1
x = ( false ) ? 1 : 2;  //x ahora contiene el valor 2
```

Hay que recordar que los operadores lógicos solo funcionan con valores de tipo **boolean**, salvo aquellos que se utilizan a nivel de bits.

Operadores lógicos

Los operadores lógicos necesarios para el examen son: **&**, **|**, **^**, **¡**, **&&** y **||**.

Operadores lógicos sobre bits (no entra en el examen)

Los siguientes operadores **&**, **|** y **^** pueden utilizarse en diferentes situaciones, una de ellas es para comparación a nivel de bits. Estos fueron incluidos en versiones de examenes previas, pero para nuestra versión, no son necesarios.

```
byte b01 = 5 & 15; //Compuerta AND
/*
    0 0 1 0 1   - 5
    0 1 1 1 1   - 15
    ----------
    0 0 1 0 1   - 5
*/

byte b02 = 3 | 8; //Compuerta OR
/*
    0 0 0 1 1   - 3
    0 1 0 0 0   - 8
    ----------
    0 1 0 1 1   - 11
*/

byte b03 = 29 ^ 11; //Compuerta XOR
/*
    1 1 1 0 1   - 29
    0 1 0 1 1   - 11
    ----------
    1 0 1 1 0   - 21
*/
```

Si deseas comprender un poco más acerca del funcionamiento de las operaciones lógicas con compuertas en código binario, puedes buscar en internet por el nombre de las mismas (AND, OR, XOR).

Operadores lógicos de circuito corto

Estos son:

- && AND de circuito corto
- || OR de circuito corto

Pero, ¿por qué de circuito corto?
Se denominan de esta manera porque cuando se evalua una expresión, si la primer evaluación hace que la condición no se cumpla no se evalua la siguiente.
Veamos algunos ejemplos:

```
class Valores {
  public int getValor1() {
    System.out.println("Devuelvo el valor 1");
    return 1;
  }

  public int getValor2() {
    System.out.println("Devuelvo el valor 2");
    return 2;
  }
}

public class ejemploOperadoresLogicosCircuitoCorto {
  static public void main(String[] args) {
    Valores valores = new Valores();

    if ( (valores.getValor1() == 2) && (valores.getValor1() == 1) ) {
      System.out.println("( (valores.getValor1() == 2) && (valores.getValor1() ==
1) ) ? true");
    } else {
      System.out.println("( (valores.getValor1() == 2) && (valores.getValor1() ==
1) ) ? false");
    }

    if ( (valores.getValor2() > 0) || (valores.getValor2() != 5) ) {
      System.out.println("( (valores.getValor2() > 0) || (valores.getValor2() !=
5) ) ? true");
    } else {
      System.out.println("( (valores.getValor2() > 0) || (valores.getValor2() !=
5) ) ? false");
    }
  }
```

```
Devuelvo el valor 1
( (valores.getValor1() == 2) && (valores.getValor1() == 1) ) ? false
Devuelvo el valor 2
( (valores.getValor2() > 0) || (valores.getValor2() != 5) ) ? true
```

Como se puede apreciar, en los dos casos, al evaluar la primer condición no es necesario evaluar la segunda, de manera que esta última, directamente es obviada.

Operadores lógicos de circuito completo

Estos son:

- |
- &

Al contrario de los de circuito corto, estos siempre evaluan todas las condiciones, por más que ya se conozca de antemano el resultado de la operación lógica (estos operadores son ineficientes).
Veamos el ejemplo con los operadores de circuito completo:

A diferencia de los operadores lógicos de circuito corto, vemos como se verifica la segunda condición.

```
class Valores {
  public int getValor1() {
    System.out.println("Devuelvo el valor 1");
    return 1;
  }

  public int getValor2() {
    System.out.println("Devuelvo el valor 2");
    return 2;
  }
}

public class ejemploOperadoresLogicosCircuitoCompleto {
  static public void main(String[] args) {
    Valores valores = new Valores();

    if ( (valores.getValor1() == 2) & (valores.getValor1() == 1) ) {
      System.out.println("( (valores.getValor1() == 2) & (valores.getValor1() == 1)
) ? true");
    } else {
      System.out.println("( (valores.getValor1() == 2) & (valores.getValor1() == 1)
) ? false");
    }

    if ( (valores.getValor2() > 0) | (valores.getValor2() != 5) ) {
      System.out.println("( (valores.getValor2() > 0) | (valores.getValor2() != 5) )
? true");
    } else {
      System.out.println("( (valores.getValor2() > 0) | (valores.getValor2() != 5) )
? false");
    }
  }
}
```

```
Devuelvo el valor 1
Devuelvo el valor 1
( (valores.getValor1() == 2) & (valores.getValor1() == 1) ) ? false
Devuelvo el valor 2
Devuelvo el valor 2
( (valores.getValor2() > 0) | (valores.getValor2() != 5) ) ? true
```

Lo que se viene

En la proxima entrega estaremos adentrandonos en las sentencia para control de flujo del programa.

Veremos las sentencias de condición y bucles.

Nos adentraremos en el mundo de la excepciones para saber como tratar los errores en Java.

Veremos las afirmaciones, un mecanismo que Java nos otorga para verificar aquellas secciones de código que estamos seguros no deberían de fallar jamás.

Esto y mucho más, en el próximo JavaWorld.

Operadores lógicos ^ y ¡

Estos operadores cumplen la unción de las siguientes compuertas:

- ¡ Not
- ^ Xor

Not

El operador Not solo toma un término, y el resultado refleja el valor de entrada invertido.

Xor

El operador Xor evalua dos términos. Si estos son diferentes, el resultado es `true`, de lo contrario es `false`.

Valor Creativo

¿Que es JavaWorld para Valor Creativo?

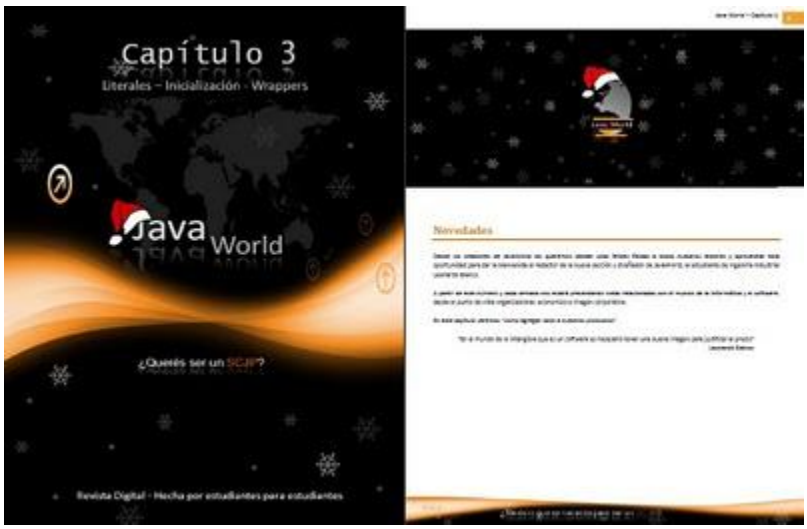
Muchos de los visitantes de Valor Creativo provienen de la revista digital JavaWorld de Gustavo Alberola y Matias Álvarez. Recientemente se agrego a la revista una nueva sección relacionado con la importancia de la creatividad en la industria del Software.

JavaWorld es más bien un resumen, para gente de habla hispana, de un libro de más de 900 páginas totalmente en ingles (por ello serán 10 ediciones). El libro habla de programación en lenguaje Java y esta orientado a programadores.
Valor Creativo se unió a la iniciativa y su objetivo es transformar este resumen en un revista creando, con el paso del tiempo, múltiples secciones relacionado con Java y el mundo de la Informática.

Este proyecto llevo muchas horas de planificación antes del lanzamiento de la primera edición, con un simple objetivo realizar la revista lo mejor posible con los recursos disponibles.

Decisiones

Cuando empezamos a diagramar la revista surgio el problema de que los párrafos de doble columna eran incómodos a la vista por la información que manejaban, de ahí nuestra primer decisión: realizar una Revista tipo Informe.
La segunda decisión era como llamarla había que darle un nombre y un logo, que más simple que el lema Java para todo el mundo, de aquí nació el mundo dentro de una taza.



La tercera decisión fue como proponer la revista que las personas sean atraídas a el, y nacen los lemas ¿Querés ser un SCJP? en el portada, ¿Tenes lo que se necesita para serlo? en la contratapa -en un comentario personal cuando se pensó en estos lemas simplemente nacieron por mi incapacidad de entender los temas de JavaWorld, la verdad es que no entendía nada y por ello el desafío-.
La cuarta decisión fue muy importante debido a que la revista era gratuita y los costos de diseño son elevados, se decidió realizar una plantilla única y mantenerla durante todas las ediciones. Fue muy difícil esta decisión porque si queremos algo prolijo tenemos que respetar lo anterior y la idea es que las revistas sean coleccionables, por ello si ahora no nos gusta la letra, por ejemplo, será un error que no se debe corregir.

Resultados

A partir de todas nuestras decisiones se lanzo la primera edición.

Nadie se esperaba el éxito de la revista tan pronto, y con mente fría se lanzo el segundo capitulo con mayor aceptación. A partir de la tercera edición se anexa la sección Valor Creativo con el fin de darles a los lectores herramientas para que agreguen valor a sus trabajos intangibles.

Estamos muy contentos con todos los comentarios que nos dejan los usuarios, y ello nos motiva aún más a seguir adelante. Es un esfuerzo en conjunto muy grande y que la gente lo valore y reconozco es nuestra recompensa.

Seguiremos trabajando por más ediciones, y cualquier adhesión, apoyo o sugerencia es bien recibida.

Saludos a todos y feliz año nuevo.

¿Tenés lo que se necesita para ser un SCJP?

Autores
Gustavo Alberola
Matias Álvarez

Diseño
Leonardo Blanco