SCJP – Capitulo 4

Operadores y Asignaciones

Agenda

- Operadores
- Operadores Lógicos
- Parámetros de Métodos

Operadores – Asignación

- int x = 6;
- Button b = new Button();
- Button b = null;

¿Que es lo que se asigna realmente?

R.-
Patrones de bits

Excepto el null que quiere decir que no referencia a un objeto…

Asignación de Primitivos

=	*=	/=	%=
+=	-=	<<=	>>=
>>>=	&=	^=	|=

Casting de primitivos

- byte b = 3; //OK 3 cabe en un byte
- byte c = (byte) 8; //OK 8 cabe en un byte
- byte d = b + c; // 11 cabe en un byte??
- byte d = (byte) (b + c); //Hacer Casting

- d = (byte) (d + 7); //casting requerido
- d += 7; // casting automático

Casting con perdida de precisión

- int x = (int) 32.5F; //aquí se trunca la parte decimal
- byte a = 128; // byte almacena hasta 127
- byte a = (byte) 128; //se toman los 8 bits significativos (de la derecha)

Casting con perdida de precisión

- byte a = (byte) 128;

00000000 00000000 00000000 10000000

Expresado en “complemento a dos”

1ro : complemento : 01 11 11 11

2do: sumar 1 : 00 00 00 01

Resultado (-128) : 10 00 00 00

Casting con perdida de precisión

- byte a = (byte) 768;

00000000 00000000 00000011 00000000

Expresado en “complemento a dos”

1ro : complemento : …

2do: sumar 1 : …

Resultado (0) : 00 00 00 00

Casting con perdida de precisión

- byte a = (byte) 876;

00000000 00000000 00000011 01101100

Expresado en “complemento a dos”

1ro : complemento : …

2do: sumar 1 : …

Resultado (+108) : 01 10 11 00

Casting con perdida de precisión

- byte a = (byte) 960;

00000000 00000000 00000011 11000000

Expresado en “complemento a dos”

1ro : complemento : 00 11 11 11

2do: sumar 1 : 00 00 00 01

Resultado (-64) : 01 00 00 00

Asignación de variables de Ref.

```
class Perro {
  String nombre;
}

class TestReferencia {
  public static void main(String[ ] args){
    Perro a = new Perro();
    a.nombre = “Snoopy”;
    Perro b = a; //a y b hacen referencia al mismo objeto
    b.nombre = “Boby”;
  }
}
```

Asignación de String

```
class RefString {
  public static void main (String[ ] args){
    String x = “Java”;
    String y = x;
    x = x + “ Bean”; //se crea un nuevo String
    x.toUpperCase(); //nuevo String sin referencia
  }
}
```

Operadores de Comparación

- > Mayor que
- >= Mayor igual que
- < Menor que
- <= Menor igual que

- Sólo para Primitivos

```
boolean b = 100 > 99; // true
boolean b = ( ‘a’ < ‘b’ ); // true
```

Comparación con instanceof

```
String str = “Soy un String”;

If (str instanceof String) {
   System.out.println(“str es un String”);
}
```

Comparación con instanceof

```
class Animal { }
class Perro extends Animal { }


…
Perro p = new Perro();
If (p instanceof Animal) {
    System.out.println(“p es un Animal”);
}
…
```

Comparación con instanceof

```
class Animal { }
class Perro extends Animal { }


…
Perro p = new Perro();
If (p instanceof Object) {
    System.out.println(“p definitivamente Object”);
}
…
```

Comparación con instanceof

```
interface Correr { }
class Animal implements Correr { }
class Perro extends Animal { }
…
Animal a = new Animal();
Perro p = new Perro();

a instanceof Animal // true
p instanceof Perro // true
p instanceof Animal // true
p instanceof Correr // true, aunque no la implemente directamente
```

Comparación con instanceof

```
interface Correr { }
class Animal implements Correr { }
class Perro extends Animal { }
…
Animal a = new Animal();
Perro p = null;

a instanceof Perro // false, no conoce las subclases
p instanceof Perro // false, si esta en nulo no se
    considera instancia
null instanceof Perro // false (siempre)
```

Comparación con instanceof

```
interface Correr { }
class Animal implements Correr { }
class Perro extends Animal { }
```

Variable de Referencia	instanceof	Resultado
null	Cualquier clase o interface	Falso
Instancia de Perro	Perro, Animal, Correr, Object	Verdadero
Instancia de Animal	Animal, Correr, Object	Verdadero
Instancia de Animal	Perro	Falso
Perro[ ]	Perro, Animal, Correr	Falso
Perro[ ]	Object	Verdadero
Perro[1]	Perro, Animal, Correr, Object	Verdadero

Operadores de Igualdad

- == igual
- != no igual (distinto)

```
char c = '@';
if (c == '@') {
   System.out.println("Var c es igual a @");
}
```

Igualdad en primitivos

```
//aquí se esta asignando un valor boolean
boolean b = false;
if (b = true) {
   System.out.println(“b es true”);
} else {
   System.out.println(“b es false”);
}
```

Igualdad en Referencias

```
Animal a = new Animal();
Animal b = new Animal();
Animal c = a;
…
(a == b) //false, apuntan a objetos distintos
(a == c) //true, apuntan al mismo objeto
```

Operaciones aritméticas

- + - * / %

```
int x = 7 / 3; // división entera = 2
int y = 7 % 3; // resto = 1

double a = 7.0 / 3.0; // división =
   2.33333…
double b = 7.0 % 3.0; // resto = 1.0
```

Operaciones aritméticas

- Si se dividen Enteros (int) por cero se consigue una AritmeticException
- Si se divide un coma flotante (double o float) por cero, **No** provoca Excepcion. Y la operación retorna infinito positivo o infinito negativo (según sea el caso)
- Lo mismo se aplica para el Módulo o calculo del resto (%)

Concatenación de String

- Operador +

```
String a = “String”;
int b = 3;
int c = 7;
System.out.println(a + b + c);
```

String37

```
System.out.println(a + (b + c));
```

String10

Incremento y Decremento

- ++ Incremento
- -- decremento

```
int x = 0;
x++; // evalúa y después incrementa
++x; // incrementa después evalúa
int y = ++x * 2; // y = 6
```

Operador Shift

- >> Mueve los bits a la derecha
- << Mueve los bits a la izquierda
- >>> Mueve los bits a la derecha (pero sin mover el bits del signo)

Sólo para Enteros (no para punto flotante)

Operador Shift

- 8 >> 1

00000000 00000000 00000000 00001000

00000000 00000000 00000000 000001000

Entra : repite bit de signo

Sale : bits de la derecha

- Resultado : 4

Operador Shift

int x = 0x80000000;

Convertir Hexa a Bin, tomar 1 hexa por 4 bits

8	0	0	0	0	0	0	0
1000	0000	0000	0000	0000	0000	0000	0000

x = x << 1;

1000	0000	0000	0000	0000	0000	0000	0000	0

Entra : bit en cero

Sale : bits de la izquierda

Resultado :

Antes de shift : -2147483648

Después del shift : 0

Operador Shift

int x = 0x80000000;

Convertir Hexa a Bin, tomar 1 hexa por 4 bits

8	0	0	0	0	0	0	0
1000	0000	0000	0000	0000	0000	0000	0000

x = x >> 28;

1111 1111 1111 1111 1111 1111 1111 1000

Entra : repite bit de signo

Sale : bits de la derecha

Resultado :

Antes de shift : -2147483648

Después del shift : -8

Operador Shift

int x = 0x80000000;

Convertir Hexa a Bin, tomar 1 hexa por 4 bits

8	0	0	0	0	0	0	0
1000	0000	0000	0000	0000	0000	0000	0000

x = x >>> 28;

0000	0000	0000	0000	0000	0000	0000	1000

Entra : bit en cero (nótese que son tres : >>>)

Sale : bits de la derecha

Resultado :

Antes de shift : -2147483648

Después del shift : -8

Operadores Bitwise

- & AND
- | OR inclusivo (el normal)
- ^ OR exclusivo

- ~ Complemento

Sólo para Enteros en su expresión binaria

Operadores Bitwise

X	Y	& (AND) X & Y	| (OR) X | Y	^(XOR) X ^ Y	~ (compl) ~X
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Operadores Bitwise

- int x = 10 & 9; //1010 AND 1001

Resultado x = 8

	1	0	1	0
&	1	0	0	1
=	1	0	0	0

Operadores Bitwise

- int x = 10 | 9; //1010 OR 1001

Resultado x = 11

	1	0	1	0
|	1	0	0	1
=	1	0	1	1

Operadores Bitwise

- int x = 10 ^ 9; //1010 XOR 1001

Resultado x = 3

	1	0	1	0
|	1	0	0	1
=	0	0	1	1

Operadores Bitwise

- int x = ~5; //~0101

11111111 11111111 11111111 11111010

Es negativo !!!!, convertir “complemento a dos”

De izquierda a derecha traspasar todos los bits hasta el primer 1 (inclusive) y luego negar el resto de bits

00000000 00000000 00000000 00000110

Resultado -6

Operador Condicional

var = (expresión booleana)
? Valor a asignar si es verdadero
: Valor a asignar si es falso;

```
int x = 18;
String estado = (x > 18) ?
   “Mucho”:“Poco”;
```

Resultado : estado = “Poco”

Operador Condicional

```
double nota = 6.2
...
nota = (nota < 4.0)
              ? 4.0
              : (nota > 6.0)
                    ? 6.0
                    : nota;
...
Resultado : nota = 6.0;
```

Operador Condicional

```
interface Accion { }
class SuperHeroe implements Accion { }
class Persona implements Accion { }
…
boolean superPoderes = false;
Accion accion = (superPoderes)
                        ? new SuperHeroe()
                        : new Persona();
…
```

Resultado : (accion instanceof Persona)

Operadores lógicos

- & AND
- | OR

- && AND (corto circuito)
- || OR (corto circuito)

Operadores lógicos

```
int x = 5;
int y = 1;
if (++x > 7 && y++ == 1){
    …
}
System.out.println(x + y); // 7

int x = 5;
int y = 1;
if (++x > 7 & y++ == 1){
  …
}
System.out.println(x + y); // 8
```

Pasando Variables a Métodos

```
class Gato {
  String nombre;
  Gato (String nombre) { this.nombre = nombre;  }
}

class Test {
  Gato crearGato(){
    Gato g = new Gato(“Garfield”);
    hacerAlgo(g);
    return g;
  }
  void hacerAlgo(Gato g){
    g.nombre = “Gato Volador”;
  }
}
…
System.out.println((new Test()).crearGato().nombre); // Gato Volador
```

Pasando Variables a Métodos

```
class Gato {
  String nombre;
  Gato (String nombre) { this.nombre = nombre;  }
}

class Test {
  Gato crearGato(){
    Gato g = new Gato(“Garfield”);
    hacerAlgo(g);
    return g;
  }
  void hacerAlgo(Gato g){
    g = new Gato(“Gato Volador”);
  }
}
…
System.out.println((new Test()).crearGato().nombre); // Garfield
```

Pasando Variables a Métodos

```
class MiClase {
    static int size = 7;
    static void cambialo(int size){
        size += 200;
        System.out.println("size en cambialo es " + size);
    }
    public static void main (String[ ] args){
        MiClase mc = new MiClase();
        System.out.println("size = " + size);
        cambialo(size);
        System.out.println("size después de cambialo es " + size);
    }
}
…
```

size = 7
size en cambialo es 207
size después de cambialo es 7

Pasando Variables a Métodos

```
class MiClase {
    static int size = 7;
    static void cambialo(int size){
        size += 200;
        System.out.println("size en cambialo es " + size);
    }
    public static void main (String[ ] args){
        MiClase mc = new MiClase();
        System.out.println("size = " + size);
        cambialo(size);
        System.out.println("size después de cambialo es " + size);
    }
}
…
size = 7
size en cambialo es 207
size después de cambialo es 7
```

Pasando Variables a Métodos

```
class Algo { int algoNum = 28; }
class MiClase {
        Algo miAlgo = new Algo();
        void cambialo(Algo miAlgo){
                 miAlgo.algoNum = 99;
                System.out.println("miAlgo.algoNum en cambialo es " + miAlgo.algoNum); //99
                miAlgo = new Algo();
                 miAlgo.algoNum = 420;
                System.out.println("miAlgo.algoNum en cambialo es ahora " + miAlgo.algoNum);  //420
                System.out.println("miAlgo.algoNum en cambialo es ahora " + this.miAlgo.algoNum);  //99
        }
        public static void main (String[ ] args){
                MiClase mc = new MiClase();
                System.out.println("mc.miAlgo.algoNum es " + mc.miAlgo.algoNum);
                mc.cambialo(mc.miAlgo);
                System.out.println("mc.miAlgo.algoNum después de cambialo es " + mc.miAlgo.algoNum);
        }
}
…
```

mc.miAlgo.algoNum es 28
miAlgo.algoNum en cambialo es 99
miAlgo.algoNum en cambialo es ahora 420
mc.miAlgo.algoNum después de cambialo es 99

Pasando Variables a Métodos

```
class Algo { int algoNum = 28; }
class MiClase {
        Algo miAlgo = new Algo();
        void cambialo(Algo miAlgo){
                 miAlgo.algoNum = 99;
                System.out.println("miAlgo.algoNum en cambialo es " + miAlgo.algoNum);
                miAlgo = new Algo();
                 miAlgo.algoNum = 420;
                System.out.println("miAlgo.algoNum en cambialo es ahora " + miAlgo.algoNum);
        }
        public static void main (String[ ] args){
                MiClase mc = new MiClase();
                System.out.println("mc.miAlgo.algoNum es " + mc.miAlgo.algoNum);
                cambialo(mc.miAlgo);
                System.out.println("mc.miAlgo.algoNum después de cambialo es " + mc.miAlgo.algoNum);
        }
}
…
```

mc.miAlgo.algoNum es 28
miAlgo.algoNum en cambialo es 99
miAlgo.algoNum en cambialo es ahora 420
mc.miAlgo.algoNum después de cambialo es 99