Programación orientada a objetos con JAVA

Definición de variables

type nombreVariable[,nombreVariable];*
type nombreVariable = valor;

Ejemplo

```
int cantCuotas, nroTarjeta;
int cantCuotas = 10;
```

El alcance de una variable corresponde al bloque dentro del cual fue definida.

Expresiones aritméticas

```
int i = 10 + 1;
float f = ( 3 * 10 / 2 ) + 10 ;
```

Expresiones lógicas

```
boolean b = ( f  > 10 ) || ( f  == 0 );
```

Salida por dispositivo estándar

System.out.println (*mensaje*);

El contenido de *mensaje* puede ser de distinto tipo.

Ejemplo

```
System.out.println (123 );
System.out.println ("Total:" + total );
```

Paso de parámetros a la aplicación

java *ClasePrincipal param1 param2 ... paramN*

```
class ClasePrincipal {
   public static void main (String args[]) {
      for (int i = 0; i < args.length; i++) {
         System.out.println(args[i]);
      }
   }
}
```

Conversión de cadenas a tipos numéricos

```
Integer.parseInt ( cadena );  // Retorna "cadena" convertida a int.
Float.parseFloat ( cadena );  // Retorna "cadena" convertida a float.
Double.parseDouble ( cadena );  // Retorna "cadena" convertida a double.
```

Ejemplo

```
int i = Integer.parseInt (“123”);
float f = Float.parseFloat (“123.45”);
```

Conversión de tipos (cast)

(*type*) *expresion*;

Ejemplo

```
int i = ( int ) 10.5;
int i = ( int ) (( 2 * 4 ) / 3);
float f = ( float ) i  * 2;
```

Funciones matemáticas

```
Math.sin(a);
Math.abs(a);
Math.max(a,b);
Math.random();
```

Ejemplo

```
System.out.println((int) ((Math.random() * 6)+1) );
```

Comparación de cadenas

La sentencia

cadena1.equals (*cadena2*)

retorna true si cadena1 es igual a cadena2.

Ejemplo

```
nombre.equals ( “Jose” )
“Jose”.equals ( nombre )
nombre1.equals ( nombre2 )
```

Sentencias de control de flujo

Bucles

```
while (boolean expression) {
   ...
   [continue;]
   [break;]
   ...
}
```

Ejemplo

```
while (true) {
   System.out.println("error");
}
```

```
do {
   ...
   [continue;]
   [break;]
   ...
} while (expression);
```

Ejemplo

```
      do {
         System.out.println("error");
      } while (true)
```

```
for (initialization ; termination ; increment) {
   ...
   [continue;]
   [break;]
   ...
}
```

Ejemplo

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

Sentencias de toma de decisión

```
if (boolean expression) {
   ...
}
```

Ejemplo

```
      if ( i < 10 ) {
         System.out.println(i);
      }
```

```
if (boolean expression) {
   ...
} else {
   ...
}
```

Ejemplo

```
      if ( i < 10 ) {
         System.out.println(i);
      } else {
         System.out.println(j);
      }
```

```
if (boolean expression) {
   ...
} else if (boolean expression) {
   ...
} else if (boolean expression) {
```

```
   ...
} else {
   ...
}
```

Ejemplo

```
      if ( i == 1 ) {
         System.out.println("uno");
      } else ( i == 2) {
         System.out.println("dos");
      } else {
         System.out.println("");
      }
```

```
switch (integer expression) {
   case integer expression:
      ...
      [break;]
   ...
   default:
      ...
}
```

Ejemplo

```
      switch (i) {
         case 1:
            System.out.println("uno");
            break;
         case 2:
            System.out.println("dos");
            break;
         default:
            System.out.println("");
      }
```

Resumen de operadores aritméticos

Operador	Uso	Descripción
+	op1 + op2	Suma op1 y op2
–	op1 – op2	Resta op2 de op1
*	op1 * op2	Multiplica op1 y op2
/	op1 / op2	Divide op1 por op2
%	op1 % op2	Computa el resto de dividir op1 por op2

Incremento y decremento en uno de un número.

Operador	Uso	Descripción
++	op++	Incrementa op en 1; evalúa op antes de incrementar
++	++op	Incrementa op en 1; evalúa op después de incrementar
--	op--	Decrementa op en 1; evalúa op antes de decrementar
--	--op	Decrementa op en 1; evalúa op después de decrementar

Resumen de operadores relacionales

Operado	Uso	Retorna true si
>	op1 > op2	op1 es mayor a op2
>=	op1 >= op2	op1 es mayor o igual a op2
<	op1 < op2	op1 es menor que op2
<=	op1 <= op2	op1 es menor o igual que op2
==	op1 == op2	op1 y op2 son iguales
!=	op1 != op2	op1 y op2 no son iguales

Resumen de operadores condicionales

Operador	Uso	Retorna true si
`&&`	`op1 && op2`	`op1` y `op2` son ambos `true`, condicionalmente se evalúa `op2`
`||`	`op1 || op2`	`op1` o `op2` son `true`, condicionalmente se evalúa `op2`
`!`	`! op`	`op` es `false`
`&`	`op1 & op2`	`op1` y `op2` son ambos `true`, se evalúa `op1` y `op2`
`|`	`op1 | op2`	`op1` o `op2` es `true`, se evalúa `op1` y `op2`
`^`	`op1 ^ op2`	op1 y op2 son diferentes - uno es true y el otro false

Resumen de operadores de corrimiento

Operador	Uso	Operación
`>>`	`op1 >> op2`	Corre op2 bits de `op1` a la derecha
`<<`	`op1 << op2`	Corre op2 bits de `op1` a la izquierda

Resumen de operadores lógicos binarios

Operador	Uso	Operación
`&`	`op1 & op2`	and
`|`	`op1 | op2`	or
`^`	`op1 ^ op2`	xor
`~`	`~op2`	complemento

Resumen de operadores de asignación

Operador	Uso	Equivalente a

+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
^=	op1 ^= op2	op1 = op1 ^ op2
<<=	op1 <<= op2	op1 = op1 << op2
>>=	op1 >>= op2	op1 = op1 >> op2