Variables y operadores en Java

Programación I

Grado en Ingeniería Informática

MDR, JCRdP y JDGD

Variables (contenido)

- Tipos:
  - Primitivas
  - Referencias a objetos
  - Objetos
- Nombres de identificadores
- Ámbito

Variables primitivas y objetos

- Las variables primitivas (**enteros, reales, referencias, etc.**) son diferentes a los objetos en **uso** y **almacenamiento**
- Variables primitivas
  - Se pueden tener en la pila como variables locales a una función o bloque
  - Pueden ser parte de un objeto
- Objetos
  - sólo pueden existir en memoria dinámica
  - Se manejan siempre mediante referencias

Primitivas

- Almacenan un dato simple
- Deben ser declaradas antes de ser usadas
- Tipos:
  - Entero
  - Real
  - Lógico y carácter
- Java determina el tamaño de cada tipo primitivo
  - No varía de una plataforma a otra

Enteros (signo)

Tipo	Bytes	Nombre	Rango de valores
byte	1	Octeto	-128 a 127
short	2	entero corto	-32768 a 32767
int	4	entero	-2^{31} a $2^{31}-1$
long	8	entero largo	-2^{63} a $2^{63}-1$

Real

Tipo	Bytes	Nombre	Rango de valores
float	4	Real simple precisión	$-3.4x10^{+38}$ a $3.4x10^{+38}$
double	8	Real doble precisión	$-1.8x10^{+308}$ a $1.8x10^{+308}$

Lógico y carácter

Tipo	Bits	Nombre	Rango de valores
boolean	1	lógico	false o true
char	16	carácter	formato Unicode

Nombres de identificadores

- El primer carácter debe ser una letra (a-z, A-Z), el carácter _ o el carácter $
- Se puede utilizar la tilde, diéresis, etc.
- No pueden contener caracteres en blanco
- Es sensible a mayúsculas y minúsculas
- No se pueden utilizar como nombres de variables las palabras reservadas de Java

Literales

- Si el tipo del valor literal es ambiguo se añade información en forma de caracteres asociados con el literal
- Se establece un carácter sufijo
- Sea en mayúsculas o en minúsculas L significa **long**, F **float**, D **double**
- La base hexadecimal funciona con todos los tipos de datos enteros y se representa por 0x o 0X seguidos de 0-9 o a-f |A-F
- La base octal se representa por 0 seguido de 0-7

Literales

Variables	Valores	Descripción
char c =	0xffff	Máximo valor hexadecimal
byte b =	0x7f	Máximo valor hexadecimal
short s =	0x7fff	Máximo valor hexadecimal
int i1 =	0x2f	Hexadecimal (minúsculas)
int i2 =	0X2F	Hexadecimal (mayúsculas)
int i3 =	0177	Octal (precedido de cero)
long n1 =	200L	Sufijo long
long n2 =	200l	Sufijo long

Literales (double por defecto)

Variables	Valores	Descripción
float f1 =	1	
float f2 =	1F	Sufijo float
float f3 =	1f	Sufijo float
float f4 =	1e-45f	10^{-45} (error si no lleva f. El tipo de los exponenciales es double)
float f5 =	1e+9f	10^{9} (error si no lleva f)
double d1 =	1d	Sufijo double
double d2 =	1D	Sufijo double
double d3 =	45e+49d	$45*10^{49}$ (Superfluo)

Ejemplo de mostrado de variables

```
public class Variables {
  public static void main(String[] args) {
      int i=10;
      float f=1.1E-2f;
      System.out.println("i="+i+" f="+f);
      //se usa el operador de concatenación +
      //se convierte el dato numérico a String
      //se invoca al método toString()
  }
}
```

Ámbito

- Sólo tienen existencia dentro del bloque en el que se definen

Fichero “AmbitoVariables.java”:

```
public class AmbitoVariables {
  public static void main(String args[]) {
    int x=10;
    {
          int q=8;  //tanto x como q están disponibles
    }
    System.out.println(“x=”+x);   //q está fuera de ámbito
  }
}
```

- Los objetos no tienen ámbito
- Se mantienen en memoria dinámica hasta que el recolector de basura libera la memoria que ocupan

Expresiones, operadores

- Una expresión:
  - Es un conjunto de variables, constantes y funciones unidas por operadores
  - Su resultado es un dato numérico, lógico, carácter o una cadena de caracteres
- Los operadores:
  - Permiten relacionar datos y evaluar el resultado de las operaciones
  - No se pueden sobrecargar
  - Casi todos funcionan únicamente con datos primitivos
    - Excepto: =, == y != que se pueden usar con referencias
    - La clase String tiene sobrecargado el operador + y +=

Asignaciones

- La asignación permite evaluar una expresión y asignar el resultado a una variable
- Se puede considerar un tipo de operador con el efecto colateral de modificar la variable asignada
- Devuelve el valor asignado
- ¡Cuidado! no confundir con la comparación

Operadores

- **Aritméticos**: Permiten manipular datos numéricos
- **De relación**: Permiten comparar datos numéricos o conjuntos de caracteres. Generan un resultado de tipo boolean
- **Lógicos**: Permiten evaluar expresiones lógicas. Se evalúan hasta que se pueda determinar sin ambigüedad la certeza o falsedad de la expresión (cortocircuito)
- **De asignación**: Permiten transferir datos desde una variable a otra
- **A nivel de bits**: Permiten modificar o comparar valores numéricos a nivel de bits
- **Instanceof**: Permite determinar si un objeto pertenece a una clase

Operadores aritméticos

Operador	Descripción	Ejemplo	Resultado (a=7 y b=2)
*	Multiplicación	a*b	14
/	División	a/b	3
%	Resto de la división	a%b	1
+	Suma	a+b	9
-	Resta	a-b	5
++	Incremento	a++	8
--	Decremento	a--	6
-	Cambio de signo	-a	-7

Ejemplo de autoincremento

```
public class Autoincremento {
  public static void main(String[] args) {
      int i=1; //podría ser pasado por parámetro
      System.out.println(“i = ”+ i);
      System.out.println(“++i = “+ ++i);
      System.out.println(“i++ = “+ i++);
      System.out.println(“i = “+ i);
      System.out.println(“--i = “+ --i);
      System.out.println(“i-- = “+ i--);
      System.out.println(“i = “+ i);
  }
}
```

Operadores de relación

Operador	Descripción	Ejemplo	Resultado
>	Mayor que	7>2	true
>=	Mayor o igual que	7>=2	true
<	Menor que	7<2	false
<=	Menor o igual que	7<=2	false
==	Igual que	7==2	false
!=	Distinto de	7!=2	true

== y != comparan referencias, no comparan objetos

Para comparar objetos se utiliza el método `equals()`

Comparación entre reales

- Las operaciones entre reales, por su naturaleza, no tienen una precisión exacta
- El siguiente código muestra los números 49, 98, 103, 107, 161, 187, 196 y197 indicando que 1.0 dividido y después multiplicado por dichos números no vuelve a ser1.0

```
for(int i=1; i<200; i++){
   if((1.0/i*i) != 1.0)
        System.out.println(i);
}
```

Comparación entre reales

- Para comparar números reales se usa una cierta tolerancia.
- Por ejemplo para comparar x e y con una tolerancia delta se emplearía el siguiente código :

```
if(Math.abs(x-y)<delta){
    //Ejecutar si x e y se consideran iguales
}
```

Operadores lógicos

Operador	Descripción	Ejemplo	Resultado (a=7, b=2 y c=3)
&&	Conjunción	(a>b)&&(c>0)	true
||	Disyunción	(a>b)||(c==0)	true
!	Negación lógica	!(c==3)	false

Se aplican a valores lógicos

Ejemplo de relación y lógico

```
public class RelacionYLogico {
  public static void main(String[] args) {
       int i=83, j=4; //podrían ser valores pasados por parámetro
       System.out.println(“i = ”+i);
       System.out.println(“j = “+j);
       System.out.println(“i > j es ”+ (i>j));
       System.out.println(“i < j es “+ (i<j));
       System.out.println(“i >= j es ”+ (i>=j));
       System.out.println(“i <= j es “+ (i<=j));
       System.out.println(“i == j es ”+ (i==j));
       System.out.println(“i != j es “+ (i!=j));
       System.out.println(“(i<10) && (j<10) es ”+
              ((i<10) && (j<10)) );
       System.out.println (“(i<10) || (j<10) es ”+
              ((i<10) || (j<10)) );
  }
}
```

Operadores a nivel de bits

Operador	Descripción	Ejemplo	Resultado (a=1, b=2 y c=3)
&	Conjunción (*)	a&b	0
|	Disyunción (*)	a|b	3
^	Disyunción exclusiva	a^c	2
<<	Desplazamiento a la izquierda	a<<b	4
>>	Desplazamiento a la derecha	c>>a	1
>>>	Desplazamiento a la derecha sin signo	c>>a	1
~	Complemento	~a	-2

(*)Circuito completo en expresiones lógicas

Operadores de asignación

Operador	Descripción	Ejemplo	Expresión equivalente
=	Asignación	a=b	
-=	Resta	a-=b	a=a-b
+=	Suma	a+=b	a=a+b
*=	Multiplicación	a*=b	a=a*b
/=	División	a/=b	a=a/b
%=	Resto de división	a%=b	a=a%b

Operadores de asignación a nivel de bits

Operador	Descripción	Ejemplo	Expresión equivalente
<<=	Desplazamiento izq	a<<=b	a=a<<b
>>=	Desplazamiento der	a>>=b	a=a>>b
&=	Conjunción	a&=b	a=a&b
|=	Disyunción	a|=b	a=a|b
^=	Disyunción exclusiva	a^=b	a=a^b

Operador ternario

Es de la forma: `Expresión_booleana ? Valor1:Valor2`

Si la expresión booleana es true el resultado del operador es Valor1 y si es false es Valor2

```
public class Ternario {
    public static void main(String[] args) {
            int i=5, a;
            a=(i<10)?100:10;
            System.out.println(a);
    }
}
```

Operadores de conversión

- Se permite convertir un tipo primitivo en cualquier otro excepto **boolean**
- Si la conversión es extensora no es necesaria una conversión explícita
- Si la conversión es reductora se corre el riesgo de perder información. La conversión debe ser explícita

```
public class Conversion {
    public static void main(String[] args) {
            int i=5;
            double j=99e+8;
            i=(int) j;
            System.out.println(i);
    }
}
```

Prioridad de los operadores
(de mayor a menor)

Nombre	Operadores
Expresión	. ( ) [ ]
Unitarios	- ! * & ++ --
Multiplicativos	* / %
Aditivos	+ -
Desplazamiento de bits	>> <<
Relacionales	< <= > >= == !=
Lógicos (de izquierda a derecha)	& ^ | && ||
Condicionales	?:
Asignación	= *= /= %= += -=

Referencias bibliográficas

- **Language Basics**
  http://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html