I.P.N

Instituto Politécnico Nacional

ESCOM

Escuela Superior de Computo

Programación Orientada a Objetos

Practica4

Tipos y variables

Alumno.- Santuario Parra Luis Fernando

Introducción.-

Operadores

Los operadores de Java son muy parecidos en estilo y funcionamiento a los de C. En la siguiente tabla aparecen los operadores que se utilizan en Java, por orden de precedencia:

```
.      []     ()
++     --
!      ~      instanceof
*      /      %
+      -
<<     >>     >>>
<      >      <=      >=      ==      !=
&      ^      |
&&     ||
?  :
=      op=    (*=     /=      %=      +=      -=      etc.)    ,
```

Los operadores numéricos se comportan como esperamos:

int + int = int, long + long = long, float + float = float, double + double = double.
int - int = int, long - long = long, float - float = float, double - double = double.
Int * int = int, long * long = long, float * float = float, double * double = double.
int / int = int, long / long = long, float / float = float, double / double = double.

Para realizar operaciones aritméticas con asignación utilizamos +=, -=, *=, /=, %=.

Los operadores relacionales devuelven un valor booleano.

Para las cadenas, se pueden utilizar los operadores relacionales para comparaciones además de + y += para la concatenación:

```
String nombre = "nombre" + "Apellido";
```

El operador de igualdad (=) siempre hace copias de objetos, marcando los antiguos para borrarlos, y ya se encargará el Recolector de basura de devolver al sistema la memoria ocupada por el objeto eliminado.

Algunos de los principales problemas para utilizar los operadores con un flujo de datos proveniente del teclado es que Java da una importancia mayor a los datos de tipo String, claro esta que un String es una clase, por lo tanto es más fácil introducir por teclado datos en formato de cadenas, más sin embargo Java soluciona el problema utilizando CLASES para los tipos de datos y por medio de estas se realizan conversiones entre tipos de datos y entre tipos de objetos. Dichas clases se muestran a continuación:

LA CLASE Character

Al trabajar con caracteres se hace necesario utilizar funciones de comprobación y traslación. Estas funciones están empleadas en la clase Character. De esta clase sí que se pueden crear instancias, al contrario que sucede con la clase Math.

Declaraciones

La primera sentencia creará una variable carácter y la segunda un objeto Character:

```
char c;
Character C;
```

Comprobaciones booleanas

```
Character.isLowerCase( c )
Character.isUpperCase( c )
Character.isDigit( c )
Character.isSpace( c )
```

En este caso, para un objeto Character C, no se podría hacer C.isLowerCase, porque no se ha hecho un new de Character. Estas funciones son estáticas y no conocen al objeto, por eso hay que crearlo antes.

Traslaciones de caracteres

```
char c2 = Character.toLowerCase( c );
char c2 = Character.toUpperCase( c );
```

Traslaciones de carácter a dígito

```
int i = Character.digit( c,base );
char c = Character.forDigit( i,base );
```

Métodos de la clase Character

```
C = new Character( 'J' );
char c = C.charValue();
String s = C.toString();
```

LA CLASE Float

Cada tipo numérico tiene su propia clase de objetos. Así el tipo float tiene el objeto Float. De la misma forma que con la clase Character, se han codificado muchas funciones útiles dentro de los métodos de la clase Float.

Declaraciones

La primera sentencia creará una variable float y la segunda un objeto Float:

```
float f;
Float F;
```

Valores de Float

```
Float.POSITIVE_INFINITY
Float.NEGATIVE_INFINITY
Float.NaN
Float.MAX_VALUE
Float.MIN_VALUE
```

Conversiones de Clase/Cadena

```
String s = Float.toString( f );
f = Float.valueOf( "3.14" );
```

Comprobaciones

```
boolean b = Float.isNaN( f );
boolean b = Float.isInfinite( f );
```

La función isNaN() comprueba si f es un No-Número. Un ejemplo de no-número es raiz cuadrada de -2.

Conversiones de Objetos

```
Float F = new Float( Float.PI );
```

```
String s = F.toString();
int i = F.intValue();
long l = F.longValue();
float F = F.floatValue();
double d = F.doubleValue();
```

Otros Métodos

```
int i = F.hashCode();
boolean b = F.equals( Object obj );
int i = Float.floatToIntBits( f );
float f = Float.intBitsToFloat( i );
```

LA CLASE Double

Cada tipo numérico tiene su propia clase de objetos. Así el tipo double tiene el objeto Double. De la misma forma que con la clase Character, se han codificado muchas funciones útiles dentro de los métodos de la clase Double.

Declaraciones

La primera sentencia creará una variable double y la segunda un objeto Double:

```
double d;
Double D;
```

Valores de Double

```
Double.POSITIVE_INFINITY
Double.NEGATIVE_INFINITY
Double.NaN
Double.MAX_VALUE
Double.MIN_VALUE
```

Métodos de Double

```
D.isNaN();
Double.isNaN( d );
D.isInfinite();
Double.isInfinite( d );
boolean D.equals();
String D.toString();
int D.intValue();
long D.longValue();
float D.floatValue();
double D.doubleValue();
int i = D.hashCode();
Double V.valueOf( String s );
long l = Double.doubleToLongBits( d );
double d = Double.longBitsToDouble( l );
```

LA CLASE Integer

Cada tipo numérico tiene su propia clase de objetos. Así el tipo int tiene el objeto Integer. De la misma forma que con la clase Character, se han codificado muchas funciones útiles dentro de los métodos de la clase Integer.

Declaraciones

La primera sentencia creará una variable int y la segunda un objeto Integer:

```
int i;
Integer I;
```

Valores de Integer

```
Integer.MIN_VALUE;
Integer.MAX_VALUE;
```

Métodos de Integer

```
String Integer.toString( int i,int base );
String Integer.toString( int i );
int I.parseInt( String s,int base );
int I.parseInt( String s );
Integer Integer.valueOf( String s,int base );
Integer Integer.valueOf( String s );
int I.intValue();
long I.longValue();
float I.floatValue();
double I.doubleValue();
String I.toString();
int I.hashCode();
boolean I.equals( Object obj );
```

En los métodos toString(), parseInt() y valueOf() que no se especifica la base sobre la que se trabaja, se asume que es base 10.

LA CLASE Long

Cada tipo numérico tiene su propia clase de objetos. Así el tipo long tiene el objeto Long. De la misma forma que con la clase Character, se han codificado muchas funciones útiles dentro de los métodos de la clase Long.

Declaraciones

La primera sentencia creará una variable long y la segunda un objeto Long:

```
long l;
Long L;
```

Valores de Long

```
Long.MIN_VALUE;
Long.MAX_VALUE;
```

Métodos de Long

```
String Long.toString( long l,int base );
String Long.toString( long l );
long L.parseLong( String s,int base );
long L.parseLong( String s );
Long Long.valueOf( String s,int base );
Long Long.valueOf( String s );
int L.intValue();
long L.longValue();
float L.floatValue();
double L.doubleValue();
```

```
String L.toString();
int L.hashCode();
boolean L.equals( Object obj );
```

En los métodos toString(), parseInt() y valueOf() que no se especifica la base sobre la que se trabaja, se asume que es base 10.

LA CLASE Boolean

Los valores boolean también tienen su tipo asociado Boolean, aunque en este caso hay menos métodos implementados que para el resto de las clases numéricas.

Declaraciones

La primera sentencia creará una variable boolean y la segunda un objeto Boolean:

```
boolean b;
Boolean B;
```

Valores de Boolean

```
Boolean.TRUE;
Boolean.FALSE;
```

Métodos de Boolean

```
boolean B.booleanValue();
String B.toString();
boolean B.equals( Object obj );
```

LA CLASE String

Java posee gran capacidad para el manejo de cadenas dentro de sus clases String y StringBuffer. Un objeto String representa una cadena alfanumérica de un valor constante que no puede ser cambiada después de haber sido creada. Un objeto StringBuffer representa una cadena cuyo tamaño puede variar.

Los Strings son objetos constantes y por lo tanto muy baratos para el sistema. La mayoría de las funciones relacionadas con cadenas esperan valores String como argumentos y devuelven valores String.

Hay que tener en cuenta que las funciones estáticas no consumen memoria del objeto, con lo cual es más conveniente usar Character que char. No obstante, char se usa, por ejemplo, para leer ficheros que están escritos desde otro lenguaje.

Existen muchos constructores para crear nuevas cadenas:

```
String();
String( String str );
String( char val[] );
String( char val[],int offset,int count );
String( byte val[],int hibyte );
String( byte val[],int hibyte,int offset,int count );
```

Tal como uno puede imaginarse, las cadenas pueden ser muy complejas, existiendo muchas funciones muy útiles para trabajar con ellas y, afortunadamente, la mayoría están codificadas en la clase String.

Funciones Básicas

La primera devuelve la longitud de la cadena y la segunda devuelve el carácter que se encuentra en la posición que se indica en indice:

```
int length();
char charAt( int indice );
```

Funciones de Comparación de Strings

```
boolean equals( Object obj );
boolean equalsIgnoreCase( Object obj );
```

Lo mismo que equals() pero no tiene en cuenta mayúsculas o minúsculas.

```
int compareTo( String str2 );
```

Devuelve un entero menor que cero si la cadena es léxicamente menor que str2. Devuelve cero si las dos cadenas son léxicamente iguales y un entero mayor que cero si la cadena es léxicamente mayor que str2.

Funciones de Comparación de Subcadenas

```
boolean regionMatch( int thisoffset,String s2,int s2offset,int len );
boolean regionMatch( boolean ignoreCase,int thisoffset,String s2,
int s2offset,int 1 );
```

Comprueba si una región de esta cadena es igual a una región de otra cadena.

```
boolean startsWith( String prefix );
boolean startsWith( String prefix,int offset );
boolean endsWith( String suffix );
```

Devuelve si esta cadena comienza o termina con un cierto prefijo o sufijo comenzando en un determinado desplazamiento.

```
int indexOf( int ch );
int indexOf( int ch,int fromindex );
int lastIndexOf( int ch );
int lastIndexOf( int ch,int fromindex );
int indexOf( String str );
int indexOf( String str,int fromindex );
int lastIndexOf( String str );
int lastIndexOf( String str,int fromindex );
```

Devuelve el primer/último índice de un carácter/cadena empezando la búsqueda a partir de un determinado desplazamiento.

```
String substring( int beginindex );
String substring( int beginindex,int endindex );
String concat( String str );
String replace( char oldchar,char newchar );
String toLowerCase();
String toUpperCase();
String trim();
```

Ajusta los espacios en blanco al comienzo y al final de la cadena.

```
void getChars( int srcBegin,int srcEnd,char dst[],int dstBegin );
void getBytes( int srcBegin,int srcEnd,byte dst[],int dstBegin );
String toString();
char toCharArray();
int hashCode();
```

Funciones ValueOf

La clase String posee numerosas funciones para transformar valores de otros tipos de datos a su representación como cadena. Todas estas funciones tienen el nombre de valueOf, estando el método sobrecargado para todos los tipos de datos básicos.

Veamos un ejemplo de su utilización:

```
String Uno = new String( "Hola Mundo" );
float f = 3.141592;

String PI = Uno.valueOf( f );
String PI = String.valueOf( f );     // Mucho más correcto
```

Funciones de Conversión

```
String valueOf( boolean b );
String valueOf( int i );
String valueOf( long l );
String valueOf( float f );
String valueOf( double d );
String valueOf( Object obj );
String valueOf( char data[] );
String valueOf( char data[],int offset,int count );
```

Usa arrays de caracteres para la cadena.

```
String copyValueOf( char data[] );
String copyValueOf( char data[],int offset,int count );
```

Crea un nuevo array equivalente para la cadena.

Objetivo.-

El alumno comprenderá y utilizará correctamente las clases empleadas para cada uno de los tipos de datos en Java, así como los operadores que el este lenguaje emplea para realizar operaciones de nivel de bits, numéricas, relacionales etc. en programas (aplicaciones o Applets) que resuelvan algún tipo de problema, ya sea que utilicen o no un algoritmo.

Desarrollo.-

Calculadora sin pila

```
A continuacion se muestran las instrucciones para utilizar el programa
Introduzca un operador aritmetico(+,-,*,/) seguido de un numero
Para terminar el programa introdusca >> o << seguido de un numero
No introduzca numeros sin operadores ni operadores sin numeros
Valor:
+7
7.0
Valor:
-8
-1.0
Valor:
+1
0.0
Valor:
-56
-56.0
Valor:
/2
-28.0
Valor:
<<1
El total es:-28.0
```

Calculadora con pila

```
A continuacion se muestran las instrucciones para utilizar el programa
Introdusca >>  para terminar.
Introduzca un operador(+,-,/,*) seguido de un numero :
Valor:
+3
Valor:
+4
Valor:
+9
Valor:
-4
Valor:
-9
Valor:
<<
Vuelta de carro detectado, imprimiendo resultado:
El resultado de sus operaciones X= 3.0
```

Cuestionario.-

1.- Qué clase de conversión utilizo en los programas?
Cadena a flotante

2.- Piensa usted que en Java se puedan implementar programas inteligentes?
Si pero no son fáciles de implementar

3.- Qué operador le causó mayor complejidad en implementar?
<<,>>

4.- Piensa que están bien pensadas las clases para los tipos de datos?
Si

5.- Implemente el programa 3 del desarrollo con pilas.

Conclusiones.-

En esta práctica utilizamos los elementos aprendidos en clase para poder solucionar los problemas planteados en los programas requeridos (Programa inteligente con y sin pila) cada uno de los cuales represento un problema en su mayoría enfocados a el uso de los operadores, los dientitos envoltorios de datos y poder llevarlos del paradigma estructurado al orientado a objetos, introducir constructores además de hacer que funcionen claro está.