WRAPPERS

Los ***Wrappers*** (*envoltorios*) son clases diseñadas para ser un ***complemento*** de los ***tipos primitivos***. En efecto, los tipos primitivos son los únicos elementos de ***Java*** que no son objetos. Esto tiene algunas ventajas desde el punto de vista de la ***eficiencia***, pero algunos inconvenientes desde el punto de vista de la ***funcionalidad***. Por ejemplo, los ***tipos primitivos*** siempre se pasan como argumento a los métodos ***por valor***, mientras que los ***objetos*** se pasan ***por referencia***. No hay forma de modificar en un método un argumento de tipo primitivo y que esa modificación se trasmita al entorno que hizo la llamada. Una forma de conseguir esto es utilizar un ***Wrapper***, esto es un objeto cuya variable miembro es el tipo primitivo que se quiere modificar. Las clases ***Wrapper*** también proporcionan métodos para realizar otras tareas con lo tipos primitivos, tales como conversión con cadenas de caracteres en uno y otro sentido.

Existe una clase ***Wrapper*** para cada uno de los tipos primitivos numéricos, esto es, existen las clases ***Byte***, ***Short***, ***Integer***, ***Long***, ***Float*** y ***Double*** (obsérvese que los nombres empiezan por mayúscula, siguiendo la nomenclatura típica de ***Java***).

Los *Wrappers* ***Byte***, ***Short*** y ***Long*** son similares a ***Integer***.

Los *Wrappers* son utilizados para convertir cadenas de caracteres (texto) en números. Esto es útil cuando se leen valores desde el teclado, desde un fichero de texto, etc. Los ejemplos siguientes muestran algunas conversiones:

Cree en java, una clase nueva, y en el método main, ingrese las siguientes órdenes:

```
String numDecimalString = "8.978";
float numFloat=Float.valueOf(numDecimalString).floatValue();
// numFloat = 8,979
double
numDouble=Double.valueOf(numDecimalString).doubleValue();//
numDouble = 8,979
String numIntString = "1001";
int numInt=Integer.valueOf(numIntString).intValue(); //
numInt = 1001
```

Luego haga que se muestren el pantalla los valores convertidos.

Métodos	Función que desempeñan
Integer(int) y Integer(String)	Constructores de la clase
doubleValue(), floatValue(), longValue(), intValue(), shortValue(), byteValue()	Conversores con otros tipos primitivos
Integer decode(String), Integer parseInt(String), String toString(), Integer valueOf(String)	Conversores con String
String toBinaryString(int), String toHexString(int), String toOctalString(int)	Conversores a cadenas representando enteros en otros sistemas de numeración
Integer getInteger(String)	Determina el valor de una propiedad del sistema a partir del nombre de dicha propiedad
MAX_VALUE, MIN_VALUE, TYPE	Constantes predefinidas

Tabla 4.4. Métodos y constantes de la clase Integer.

Métodos	Función que desempeñan
Double(double) y Double(String)	Los constructores de esta clase
doubleValue(), floatValue(), longValue(), intValue(), shortValue(), byteValue()	Métodos para obtener el valor del tipo primitivo
String toString(), Double valueOf(String)	Conversores con la clase String
isInfinite(), isNaN()	Métodos de chequear condiiciones
equals(Object)	Compara con otro objeto
MAX_DOUBLE, MIN_DOUBLE, POSITIVE_INFINITY, NEGATIVE_INFINITY, NaN, TYPE	Constantes predefinidas. TYPE es el objeto Class representando esta clase

Tabla 4.3. Métodos y constantes de la clase Double.

En el caso de que el texto no se pueda convertir directamente al tipo especificado se lanza una excepción de tipo ***NumberFormatException***, por ejemplo si se intenta convertir directamente el texto "4.897" a un número entero. El proceso que habrá que seguir será convertirlo en primer lugar a un número ***float*** y posteriormente a número entero.

CLASE MATH

La clase ***java.lang.Math*** deriva de ***Object***. La clase ***Math*** proporciona métodos ***static*** para realizar las operaciones matemáticas más habituales. Proporciona además las constantes ***E*** y ***PI***, cuyo significado no requiere muchas explicaciones.

Métodos	Significado	Métodos	Significado
abs()	Valor absoluto	sin(double)	Calcula el seno
acos()	Arcocoseno	tan(double)	Calcula la tangente
asin()	Arcoseno	exp()	Calcula la función exponencial
atan()	Arcotangente entre -PI/2 y PI/2	log()	Calcula el logaritmo natural (base e)
atan2(,)	Arcotangente entre -PI y PI	max(,)	Máximo de dos argumentos
ceil()	Entero más cercano en dirección a infinito	min(,)	Mínimo de dos argumentos
floor()	Entero más cercano en dirección a -infinito	random()	Número aleatorio entre 0.0 y 1.0
round()	Entero más cercano al argumento	power(,)	Devuelve el primer argumento elevado al segundo
rint(double)	Devuelve el entero más próximo	sqrt()	Devuelve la raíz cuadrada
IEEEremainder(double , double)	Calcula el resto de la división	toDegrees(double)	Pasa de radianes a grados (Java 2)
cos(double)	Calcula el coseno	toRadians()	Pasa de grados a radianes (Java 2)

Tabla 4.5. Métodos matemáticos de la clase Math.

¿Qué es la JVM?

Se planteó la necesidad de conseguir un código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código "neutro" el cual estuviera preparado para ser ejecutado sobre una "*máquina hipotética o virtual*", denominada ***Java Virtual Machine*** (***JVM***). Es esta ***JVM*** quien ***interpreta*** este código neutro convirtiéndolo a código particular de la CPU utilizada. Se evita tener que realizar un programa diferente para cada CPU o plataforma.

La ***JVM*** es el intérprete de ***Java***. Ejecuta los "***bytecodes***" (ficheros compilados con extensión ****.class***) creados por el compilador de ***Java*** (***javac.exe***). Tiene numerosas opciones entre las que destaca la posibilidad de utilizar el denominado ***JIT*** (*Just-In-Time Compiler*), que puede mejorar entre 10 y 20 veces la velocidad de ejecución de un programa.

Clases de operaciones

♦ *Creadores: crean objetos de su tipo de dato sin tomar ningún objeto del mismo tipo como entrada. Todos los creadores son constructores, aunque no ocurre lo contrario.*
♦ *Productores: crean objetos del tipo de dato tomando algún objeto del mismo tipo como entrada*
♦ *Modificadores: modifican los objetos del tipo de dato ⇒sólo se dan en las abstracciones modificables*
♦ *Observadores: toman como entrada objetos del tipo de dato y devuelven resultados de otros tipos.*

3.7.2 La clase Object

Como ya se ha dicho, la clase ***Object*** es la raíz de toda la jerarquía de clases de ***Java***. Todas las clases de ***Java*** derivan de ***Object***.

La clase ***Object*** tiene métodos interesantes para cualquier objeto que son heredados por cualquier clase. Entre ellos se pueden citar los siguientes:

1. Métodos que pueden ser redefinidos por el programador:

   clone() Crea un objeto a partir de otro objeto de la misma clase. El método original heredado de ***Object*** lanza una *CloneNotSupportedException*. Si se desea poder clonar una clase hay que implementar la interface ***Cloneable*** y redefinir el método ***clone()***. Este método debe hacer una copia miembro a miembro del objeto original. No debería llamar al operador ***new*** ni a los constructores.

   equals() Indica si dos objetos son o no iguales. Devuelve ***true*** si son iguales, tanto si son referencias al mismo objeto como si son objetos distintos con iguales valores de las variables miembro.

   toString() Devuelve un ***String*** que contiene una representación del objeto como cadena de caracteres, por ejemplo para imprimirlo o exportarlo.

   finalize() Este método ya se ha visto al hablar de los ***finalizadores***.

Un ***finalizador*** es un método que se llama automáticamente cuando se va a destruir un objeto (antes de que la memoria sea liberada de modo automático por el sistema)). ***garbage collector*** sólo libera la memoria reservada con ***new***.