



Java Básico

Otros conceptos 2

Copyright

- Copyright (c) 2004
José M. Ordax
- Este documento puede ser distribuido solo bajo los términos y condiciones de la Licencia de Documentación de javaHispano v1.0 o posterior.
- La última versión se encuentra en
<http://www.javahispano.org/licencias/>







Clase java.lang.String

- A diferencia de C++, en Java se usa la clase String para el manejo de cadenas de caracteres.
- Existen distintas formas de crear un String:
 - Mediante su constructor por defecto:
String s = **new** String(); // Se inicializa a "" (y no a " ").
 - Mediante su constructor con un parámetro de tipo String:
String s = **new** String("Hola");
 - Mediante la asignación de una cadena de caracteres:
String s = "Hola"; // Es la única excepción al uso del new.
- Otros constructores:
 - http://java.sun.com/javase/6/docs/api/java/lang/String.html#constructor_detail







Clase java.lang.String

- Para la clase String, la primera posición de la cadena de caracteres es la cero (no la uno).
- Implementa una serie de métodos muy útiles para el manejo de las cadenas de caracteres:
- Por ejemplo:
 - **public char** charAt(**int** index);
Devuelve el carácter de la posición *index*.
 - **public** String concat(String str);
Devuelve la cadena con *str* añadido al final (igual que +).
 - **public int** indexOf(**int** ch);
Devuelve la primera ocurrencia de *ch* (-1 si no está).

Clase java.lang.String

-  **public int** compareTo(String str);
Compara la cadena con *str*. Devuelve 0 si son iguales, <1 si es menor o >1 si es mayor (siguiendo el abecedario).
-  **public** String replace(**char** oldChar, **char** newChar);
Cambia todas las ocurrencias de *oldChar* por *newChar*.
-  **public int** lastIndexOf(String str);
Devuelve la última ocurrencia de *str* (-1 si no se encuentra).
-  **public int** length();
Devuelve la longitud de la cadena de caracteres.
-  **public** String substring(**int** beginIndex);
Devuelve la cadena desde *beginIndex* hasta el final.
-  **public** String substring(**int** beginIndex, **int** endIndex);
Devuelve la cadena desde *beginIndex* hasta *endIndex* - 1.

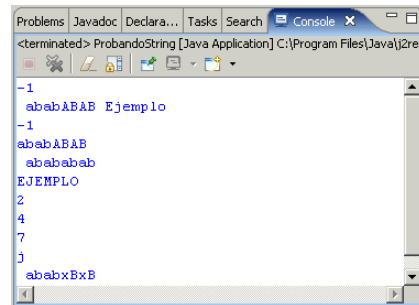
Clase java.lang.String

-  **public** String toLowerCase();
Devuelve la cadena en minúsculas.
-  **public** String toUpperCase();
Devuelve la cadena en mayúsculas.
-  **public** String trim();
Devuelve la cadena sin espacios en blanco ni por delante ni por detrás.
-  **public static** String valueOf(**double** d);
Devuelve la cadena de caracteres que representa *d*. Este método está sobrecargado varias veces recibiendo como parámetro otros tipos primitivos (char, int, float, long...).
-  Otros métodos:
-  http://java.sun.com/javase/6/docs/api/java/lang/String.html#method_detail

Ejemplo

```
public class StringTest
{
    public static void main(String[] args)
    {
        String s1 = new String();
        String s2 = new String(" ababABAB "); // Tiene espacios en blanco.
        String s3 = "Ejemplo";

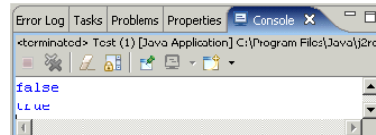
        System.out.println(s1.compareTo(" ")); // " " es un espacio en blanco.
        System.out.println(s2.concat((s3)));
        System.out.println(s1.indexOf('h'));
        System.out.println(s2.trim());
        System.out.println(s2.toLowerCase());
        System.out.println(s3.toUpperCase());
        System.out.println(s2.indexOf('b'));
        System.out.println(s2.lastIndexOf('b'));
        System.out.println(s3.length());
        System.out.println(s3.charAt(1));
        System.out.println(s2.replace('A','x'));
    }
}
```



Clase java.lang.String

- La clase String, sobrescribe el método equals() de la clase java.lang.Object.
- De esta forma, podemos saber si dos objetos String distintos, representan la misma cadena de caracteres o no.
- Ejemplo:

```
String s1 = new String("abc");
String s2 = new String("abc");
System.out.println(s1 == s2);
System.out.println(s1.equals(s2));
```



Clase java.lang.String

- Hay un concepto muy importante relacionado con el uso de Strings: son inmutables.
- Jamás se modifica el valor de un String si no que se crean nuevos objetos.
- Por ejemplo:

```
String s = "0";  
for(int i=1; i <10; i++)  
    s = s + i;
```
- No existe un solo objeto de tipo String al que se le ha ido cambiando su valor interno (atributos), sino que se han creado 10 objetos distintos.

Clase java.lang.String

- Pero aun hay mas... la JVM reserva un espacio en memoria llamado *String Pool* donde va guardando todos los String.
- Y el Garbage Collector jamás los elimina.
- Cada vez que se crea un objeto nuevo del tipo String, la JVM mira antes si ese String ya existe, y si es así lo reutiliza.
- Por este motivo, el uso de String es un tema no tan trivial como pudiera parecer. Un uso indebido puede provocar problemas de rendimiento.

Clase java.lang.StringBuffer

- StringBuffer es otra clase relacionada con las cadenas de caracteres. Pero no son inmutables.
- Existen distintas formas de crear un StringBuffer:
 - Mediante su constructor por defecto:
`StringBuffer s = new StringBuffer();` // Se inicializa a "".
 - Mediante su constructor con un parámetro de tipo String:
`StringBuffer s = new StringBuffer("Hola");`
 - Mediante su constructor con un parámetro de tipo int:
`StringBuffer s = new StringBuffer(3);` // Capacidad inicial 3.
- Otros constructores:
http://java.sun.com/javase/6/docs/api/java/lang/StringBuffer.html#constructor_detail

Clase java.lang.StringBuffer

- Para la clase StringBuffer, la primera posición de la cadena de caracteres es la cero (no la uno).
- Alguno de sus métodos:
 - `public StringBuffer append(char c);`
Añade el carácter `c` al valor del StringBuffer. Este método está sobrecargado varias veces recibiendo como parámetro otros tipos: String, Object, int, float, long...
 - `public char charAt(int index);`
Devuelve el carácter de la posición `index`.
 - `public int indexOf(int ch);`
Devuelve la primera ocurrencia de `ch` (-1 si no está).

Clase java.lang.StringBuffer

- **public int** length();
Devuelve la longitud de la cadena de caracteres.
- **public int** capacity();
Devuelve el número de caracteres que puede contener sin necesidad de alocar mas memoria.
- **public** String toString();
Devuelve un String representado por el StringBuffer.
- **public** String substring(**int** beginIndex);
Devuelve un String desde *beginIndex* hasta el final.
- **public** StringBuffer reverse();
Devuelve la cadena invertida.

Clase java.lang.StringBuffer

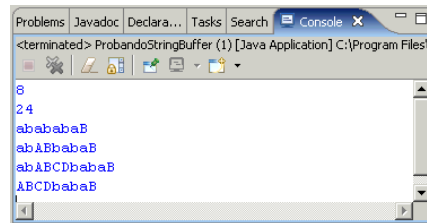
- **public void** setCharAt(**int** index, **char** ch);
Reemplaza el carácter de la posición *index* por *ch*.
- **public** StringBuffer replace(**int** start, **int** end, String str);
Reemplaza la cadena entre *start* y *end* con *str*.
- **public** StringBuffer insert(**int** offset, **char** c);
Inserta *c* en la posición *offset* de la cadena. Este método está sobrecargado varias veces recibiendo como parámetro a insertar otros tipos: int, float, long...
- **public** StringBuffer delete(**int** start, **int** end);
Elimina de los caracteres entre las posiciones *start* y *end*.
- Otros métodos:
○ http://java.sun.com/javase/6/docs/api/java/lang/StringBuffer.html#method_detail

```
public class StringBufferTest
```

```
{  
    public static void main(String[] args)
```

```
{  
    StringBuffer sb1 = new StringBuffer("abababab");  
    System.out.println(sb1.length());  
    System.out.println(sb1.capacity());  
    sb1.setCharAt(sb1.length()-1,'B');  
    System.out.println(sb1);  
    sb1.replace(2,3,"AB");  
    System.out.println(sb1);  
    sb1.insert(4,"CD");  
    System.out.println(sb1);  
    sb1.delete(0,2);  
    System.out.println(sb1);  
}  
}
```

Ejemplo



```
public class StringBufferTest
```

```
{  
    public static void main(String[] args)
```

```
{  
    StringBuffer sb2 = new StringBuffer(2);
```

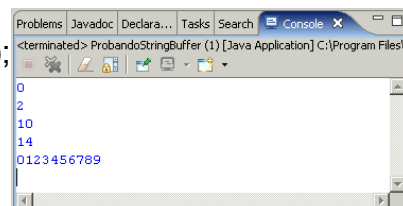
```
  
    System.out.println(sb2.length());  
    System.out.println(sb2.capacity());
```

```
  
    for(int i=0; i<10; i++)  
        sb2.append(i);
```

```
  
    System.out.println(sb2.length());  
    System.out.println(sb2.capacity());  
    System.out.println(sb2);
```

```
    }  
}
```

Ejemplo



Clase java.lang.StringBuffer

- El uso mas habitual es la creación de Strings cuyo valor se calcula de forma dinámica.
- Al no ser inmutable, permite la creación del String final sin otros objetos intermedios que consumirán memoria de forma innecesaria.
- Por ejemplo:

```
StringBuffer tmp = new StringBuffer(10);
for(int i=0; i <10; i++)
    tmp.append(i);
String s = tmp.toString();
```
- Esta vez solo se han creado 2 objetos en memoria: un StringBuffer (el GC puede limpiarlo) y un String.

Clase java.lang.StringBuilder

- Java SE 5.0 añade la clase StringBuilder al tratamiento de cadenas de caracteres.
- Su funcionalidad (constructores y métodos) es idéntica a la de StringBuffer.
- La única diferencia es que sus métodos no están sincronizados (veremos qué significa esto en el capítulo de threads).
- Tiene mejor rendimiento que StringBuffer.
- Mas información:
<http://java.sun.com/javase/6/docs/api/java/lang/StringBuilder.html>

```

public class StringBuilderTest
{
    public static void main(String[] args)
    {
        StringBuilder sb = new StringBuilder("0");

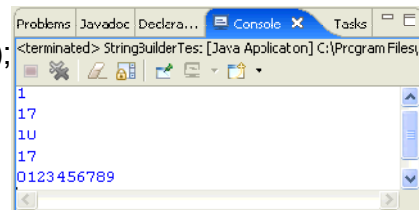
        System.out.println(sb.length());
        System.out.println(sb.capacity());

        for(int i=1; i<10; i++)
            sb.append(i);









        System.out.println(sb.length());
        System.out.println(sb.capacity());
        System.out.println(sb);
    }
}

```

Ejemplo



Clase java.lang.System

-  Se trata de una clase con utilidades genéricas del sistema.
-  Todos sus atributos y métodos son estáticos.
-  Tiene tres atributos muy utilizados:
 -  **public static final** PrintStream out;
Representa por defecto al stream de salida en pantalla.
 -  **public static final** InputStream in;
Representa por defecto al stream de entrada del teclado.
 -  **public static final** PrintStream err;
Representa por defecto al stream de salida de errores.
-  Mas información:
 -  http://java.sun.com/javase/6/docs/api/java/lang/System.html#field_detail

Clase java.lang.System



Y entre sus métodos más utilizados están:



public static long currentTimeMillis();

Devuelve la diferencia de entre la hora actual y la medianoche del 1 de enero de 1970 en milisegundos.



public static long nanoTime(); 

Devuelve la hora actual en nanosegundos.



public static void exit(**int** status);

Termina la ejecución de la JVM devolviendo status como código de retorno (cero significa que todo ha ido bien).



public static void gc();

Pide a la JVM que ejecute el Garbage Collector. Se trata de un método peligroso. Utilizado sin cuidado puede afectar muy negativamente al rendimiento.

Clase java.lang.System



public static String getProperty(String key);

Devuelve el valor de la propiedad del sistema *key*, o null si no existiese.



public static Properties getProperties();

Devuelve una instancia de java.util.Properties encapsulando todas las propiedades del sistema.



public static void loadLibrary(String libname);

Carga la librería nativa *libname*. Se utiliza con la programación JNI (Java Native Interface).



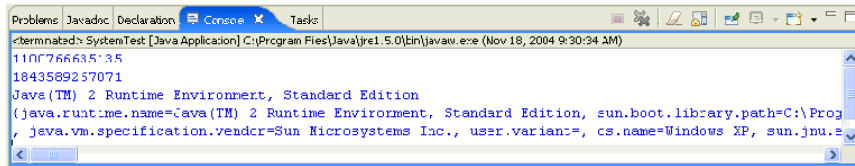
Otros métodos:



http://java.sun.com/javase/6/docs/api/java/lang/System.html#method_detail

Ejemplo

```
public class SystemTest
{
    public static void main(String[] args)
    {
        System.out.println(System.currentTimeMillis());
        System.out.println(System.nanoTime());
        System.out.println(System.getProperty("java.runtime.name"));
        System.out.println(System.getProperties());
        System.exit(0);
        System.out.println("Aquí nunca llega.");
    }
}
```



Clase java.lang.Math

- Se trata de una clase con utilidades matemáticas.
- Todos sus atributos y métodos son estáticos.
- Tiene dos atributos (constantes) muy utilizados:
 - public static final double E;**
Es el número e, utilizado en los logaritmos neperianos.
 - public static final double PI;**
Es el número π , utilizado en trigonometría.
- Más información:
http://java.sun.com/javase/6/docs/api/java/lang/Math.html#field_detail

Clase java.lang.Math



Y alguno de sus métodos son:



public static double abs(**double** a);

Devuelve el valor absoluto de *a*. Este método se encuentra sobrecargado recibiendo otros tipos: int, long...



public static double log10(**double** a); 

Devuelve el logaritmo base 10 de *a*.



public static int max(**int** a, **int** b);

Devuelve el número mayor de *a* y *b*. Este método se encuentra sobrecargado recibiendo otros tipos: long, float...



public static int min(**int** a, **int** b);

Devuelve el número menor de *a* y *b*. Este método se encuentra sobrecargado recibiendo otros tipos: long, float...

Clase java.lang.Math



public static double pow(**double** a, **double** b);

Calcula potencias de base *a* y exponente *b*.



public static double random();

Devuelve un número aleatorio entre 0.0 y 1.0 (0.0 incluido pero no así 1.0).



public static long round(**double** a);

Redondea *a* al número entero mas cercano. Este método se encuentra sobrecargado con otros tipos: float.



public static double sqrt(**double** a);

Calcula la raíz cuadrada de *a*.



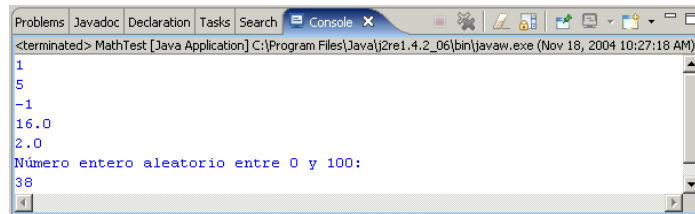
Otros métodos:



http://java.sun.com/javase/6/docs/api/java/lang/Math.html#method_detail

Ejemplo

```
public class MathTest
{
    public static void main(String[] args)
    {
        System.out.println(Math.abs(-1));
        System.out.println(Math.max(-1,5));
        System.out.println(Math.min(-1,5));
        System.out.println(Math.pow(4,2));
        System.out.println(Math.sqrt(4));
        System.out.println("Número entero aleatorio entre 0 y 100:");
        System.out.println((int)Math.round(Math.random() * 100));
    }
}
```



Otras clases

- Para manejar fechas:
 - java.util.Date y java.util.Calendar
- Para formatear fechas y números:
 - java.text.DateFormat y java.text.NumberFormat
- Para dividir una cadena en tokens:
 - java.util.StringTokenizer
- Para trabajar con expresiones regulares:
 - java.util.regex.Pattern y java.util.regex.Matcher
- Mas información:
 - <http://java.sun.com/javase/6/docs/api/>

Wrappers de tipos primitivos

- Hay ocasiones en las que necesitaríamos usar un tipo primitivo como un objeto (tipo complejo). Por ejemplo, cuando queremos guardar números en una colección que solo admite `java.lang.Object`
- En el paquete `java.lang.*` existe un wrapper para cada tipo primitivo (no siempre coincide el nombre):
 - Boolean
 - Character
 - Byte, Short, Integer, Long
 - Float, Double

Wrappers de tipos primitivos

- Casi siempre, suelen tener los siguientes métodos:
 - Constructores que recibe un `String` o el tipo primitivo que representan: `Integer a = new Integer(3);`
 - Convertidores de tipo `String` a su tipo complejo (wrapper): `Integer b = Integer.valueOf("3");`
 - Convertidores de tipo `String` al tipo primitivo que representan: `int c = Integer.parseInt("3");`
 - Convertidores de tipo primitivo a `String`: `String d = Integer.toString(c);`
 - Extractores del tipo primitivo que representan: `int e = b.intValue();`

Wrappers de tipos primitivos

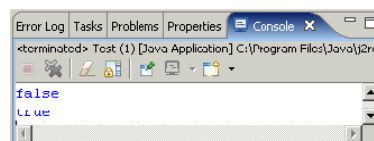
- Todas estas conversiones son susceptibles de producir errores. ¿Qué pasaría si se intenta crear un Integer utilizando “Hola” como parámetro?
- Siempre que la conversión no sea posible, la JVM lanzará una excepción del tipo:
 - java.lang.NumberFormatException
- Ya veremos como capturar y manejar este tipo de errores en el capítulo dedicado a las excepciones.

Wrappers de tipos primitivos

- Todos los wrappers sobreescriben el método equals de la clase java.lang.Object.
- De esta forma, podemos saber si dos objetos distintos de un mismo tipo de wrapper, representan el mismo valor primitivo o no.

Ejemplo:

```
Integer i1 = new Integer(3);  
Integer i2 = new Integer(3);  
System.out.println(i1 == i2);  
System.out.println(i1.equals(i2));
```



Clase java.lang.Boolean



Es el wrapper del tipo primitivo *boolean*.



public Boolean(**boolean** value);

Constructor de un Boolean con el boolean *value*.



public Boolean(String s);

Constructor de un Boolean con el String *s*. Si *s* no vale "true" entonces siempre cogerá el valor false.



public static Boolean valueOf(String s);

Convierte el String *s* en un Boolean. Si *s* no vale "true" entonces siempre devuelve un Boolean con false.



public static boolean parseBoolean(String s);

Convierte el String *s* en un boolean. Si *s* no vale "true" entonces siempre devuelve false.

Clase java.lang.Boolean



public static String toString(**boolean** b);

Convierte el boolean *b* en un String.



public String toString();

Devuelve su representación String (sobrescribe el método toString() de java.lang.Object).



public boolean booleanValue();

Extrae el boolean que representa.



Mas información y métodos:



<http://java.sun.com/javase/6/docs/api/java/lang/Boolean.html>

Clase java.lang.Character



Es el wrapper del tipo primitivo *char*.



public Character(**char** value);
Constructor de un Character con el char *value*.



public static String toString(**char** c);
Convierte el char *c* en un String.



public String toString();
Devuelve su representación String (sobrescribe el método toString() de java.lang.Object).



public char charValue();
Extrae el char que representa.



Mas información y métodos:



<http://java.sun.com/javase/6/docs/api/java/lang/Character.html>

Clase java.lang.Byte



Es el wrapper del tipo primitivo *byte*.



public Byte(**byte** value);
Constructor de un Byte con el byte *value*.



public Byte(String s);
Constructor de un Byte con el String *s*.



public static Byte valueOf(String s);
Convierte el String *s* en un Byte.



public static byte parseByte(String s);
Convierte el String *s* en un byte.



public static String toString(**byte** b);
Convierte el byte *b* en un String.

Clase java.lang.Byte



public String toString();
Devuelve su representación String (sobrescribe el método toString() de java.lang.Object).



public byte byteValue();
Extrae el byte que representa.



Más información y métodos:



<http://java.sun.com/javase/6/docs/api/java/lang/Byte.html>

Clase java.lang.Short



Es el wrapper del tipo primitivo *short*.



public Short(**short** value);
Constructor de un Short con el short *value*.



public Short(String s);
Constructor de un Short con el String *s*.



public static Short valueOf(String s);
Convierte el String *s* en un Short.



public static short parseShort(String s);
Convierte el String *s* en un short.



public static String toString(**short** s);
Convierte el short *s* en un String.

Clase java.lang.Short



public String toString();
Devuelve su representación String (sobrescribe el método toString() de java.lang.Object).



public short shortValue();
Extrae el short que representa.



Más información y métodos:



<http://java.sun.com/javase/6/docs/api/java/lang/Short.html>

Clase java.lang.Integer



Es el wrapper del tipo primitivo *int*.



public Integer(**int** value);
Constructor de un Integer con el int *value*.



public Integer(String s);
Constructor de un Integer con el String *s*.



public static Integer valueOf(String s);
Convierte el String *s* en un Integer.



public static int parseInt(String s);
Convierte el String *s* en un int.



public static String toString(**int** i);
Convierte el int *i* en un String.

Clase java.lang.Integer



public String toString();
Devuelve su representación String (sobrescribe el método toString() de java.lang.Object).



public int intValue();
Extrae el int que representa.



Más información y métodos:



<http://java.sun.com/javase/6/docs/api/java/lang/Integer.html>

Clase java.lang.Long



Es el wrapper del tipo primitivo *long*.



public Long(**long** value);
Constructor de un Long con el long *value*.



public Long(String s);
Constructor de un Long con el String *s*.



public static Long valueOf(String s);
Convierte el String *s* en un Long.



public static long parseLong(String s);
Convierte el String *s* en un long.



public static String toString(**long** l);
Convierte el long *l* en un String.

Clase java.lang.Long



public String toString();
Devuelve su representación String (sobrescribe el método toString() de java.lang.Object).



public long longValue();
Extrae el long que representa.



Más información y métodos:



<http://java.sun.com/javase/6/docs/api/java/lang/Long.html>

Clase java.lang.Float



Es el wrapper del tipo primitivo *float*.



public Float(**float** value);
Constructor de un Float con el float *value*.



public Float(String s);
Constructor de un Float con el String *s*.



public static Float valueOf(String s);
Convierte el String *s* en un Float.





public static float parseFloat(String s);
Convierte el String *s* en un float.





public static String toString(**float** f);
Convierte el float *f* en un String.

Clase java.lang.Float

 **public** String toString();
Devuelve su representación String (sobrescribe el método toString() de java.lang.Object).


 **public float** floatValue();
Extrae el float que representa.

 Mas información y métodos:


 <http://java.sun.com/javase/6/docs/api/java/lang/Float.html>


Clase java.lang.Double


 Es el wrapper del tipo primitivo *double*.

 **public** Double(**double** value);
Constructor de un Double con el double *value*.

 **public** Double(String s);
Constructor de un Double con el String *s*.

 **public static** Double valueOf(String s);
Convierte el String *s* en un Double.

 **public static double** parseDouble(String s);
Convierte el String *s* en un double.

 **public static** String toString(**double** d);
Convierte el double *d* en un String.

Clase java.lang.Double

public String toString();

Devuelve su representación String (sobrescribe el método toString() de java.lang.Object).

public double doubleValue();

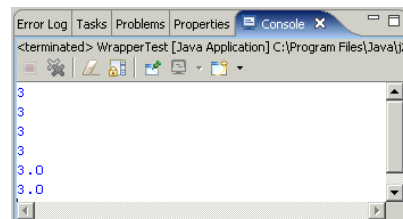
Extrae el double que representa.

Mas información y métodos:





<http://java.sun.com/javase/6/docs/api/java/lang/Double.html>

Ejemplo





```
public class WrapperTest
{
    public static void main(String[] args)
    {
        String texto = new String("3");
        byte b = Byte.parseByte(texto);
        System.out.println(Byte.toString(b));
        short s = Short.parseShort(texto);
        System.out.println(Short.toString(s));
        int i = Integer.parseInt(texto);
        System.out.println(Integer.toString(i));
        long l = Long.parseLong(texto);
        System.out.println(Long.toString(l));
        float f = Float.parseFloat(texto);
        System.out.println(Float.toString(f));
        double d = Double.parseDouble(texto);
        System.out.println(Double.toString(d));
    }
}
```



Autoboxing/Auto-unboxing

-  Java SE 5.0 añade una novedad al respecto, permitiendo que las conversiones entre tipos primitivos y sus wrappers se hagan de forma automática.
-  Antes, para crear un wrapper a partir de un tipo primitivo se utilizaba su constructor:
`Integer i = new Integer(1);`
-  Sin embargo ahora se puede hacer directamente:
`Integer i = 1;`
-  El compilador se encarga de realizar la conversión de forma automática (autoboxing).

Autoboxing/Auto-unboxing

-  De igual forma, antes para extraer un tipo primitivo de su wrapper utilizábamos el siguiente método:
`Integer a = new Integer(1); int b = a.intValue();`
-  Sin embargo ahora se puede hacer directamente:
`int b = a;`
-  El compilador se encarga de realizar la extracción de forma automática (auto-unboxing).
-  Esto nos permite también operar con los wrappers.
`Integer a = 10; Integer b = 3; int c = a + b;`

Autoboxing/Auto-unboxing

- También se permiten las comparaciones:

```
Integer a = 5; int b = 6;  
if(a == b)  
    System.out.println("Iguales");
```

- El wrapper Boolean también se ve favorecido por esta nueva funcionalidad. Antes no podía participar en condiciones, pero ahora si:

```
Boolean a = true; boolean b = false;  
Boolean c = a && b;
```

Autoboxing/Auto-unboxing

- ¿Y qué pasa con la sobrecarga de métodos?

```
public void metodo(double param) { };  
public void metodo(Integer param) { };  
int a = 5;  
this.metodo(a);
```

- Para evitar diferencias en la funcionalidad de una aplicación al migrar de versiones anteriores, primero se busca el método a ejecutar sin tener en cuenta el autoboxing y auto-unboxing.

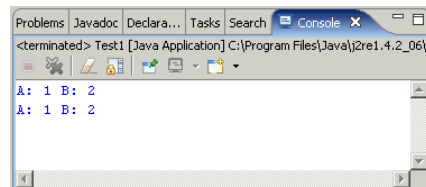
- Si no se encuentra ninguno, entonces se busca teniendo en cuenta el autoboxing y auto-unboxing.
Se llamaría a: **public void** metodo(**double** param) { };

Paso por valor o referencia

- En programación existen dos formas de pasar parámetros:
 - Por valor (o copia): se realiza una copia del parámetro.
 - Por referencia: se pasa una referencia al parámetro.
- En C se decidía mediante la gestión de punteros con los operadores: * y &
- En Java sin embargo no hay decisión posible: todo se pasa por valor.
- Si se modifica el valor de la variable recibida, no se modifica la variable original.

Ejemplo

```
public class Test1
{
    public static void main(String[] args)
    {
        int a = 1;
        int b = 2;
        System.out.println("A: " + a + " y B: " + b);
        cambiar(a,b);
        System.out.println("A: " + a + " y B: " + b);
    }
    public static void cambiar(int a, int b)
    {
        int tmp = a;
        a = b;
        b = tmp;
    }
}
```

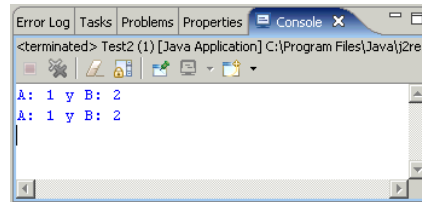


Ejemplo

```
public class MiClase
{
    private int valor = 0;
    public MiClase(int param) { this.valor = param; }
    public void setValor(int param) { this.valor = param; }
    public int getValor() { return this.valor; }
    public String toString() { return Integer.toString(valor); }
}

public class Test2
{
    public static void main(String[] args)
    {
        MiClase a = new MiClase(1);
        MiClase b = new MiClase(2);
        System.out.println("A: " + a + " y B: " + b);
        cambiar(a,b);
        System.out.println("A: " + a + " y B: " + b);
    }

    public static void cambiar(MiClase a, MiClase b)
    {
        MiClase tmp = a;
        a = b;
        b = tmp;
    }
}
```

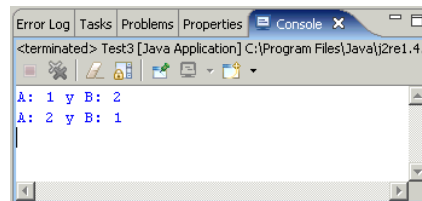


Ejemplo

```
public class MiClase
{
    private int valor = 0;
    public MiClase(int param) { this.valor = param; }
    public void setValor(int param) { this.valor = param; }
    public int getValor() { return this.valor; }
    public String toString() { return Integer.toString(valor); }
}

public class Test3
{
    public static void main(String[] args)
    {
        MiClase a = new MiClase(1);
        MiClase b = new MiClase(2);
        System.out.println("A: " + a + " y B: " + b);
        cambiar(a,b);
        System.out.println("A: " + a + " y B: " + b);
    }

    public static void cambiar(MiClase a, MiClase b)
    {
        int tmp = a.getValor();
        a.setValor(b.getValor());
        b.setValor(tmp);
    }
}
```



Nota: Realmente no se han cambiado las referencias sino los atributos internos de esas referencias.

Qué ocurre por defecto

- Las clases que no importan explícitamente el paquete `java.lang.*` lo hacen de forma implícita:
`import java.lang.*;`
- Las clases que no heredan explícitamente de otra clase, heredan implícitamente de `java.lang.Object`:
`public class MiClase extends Object`
- Las clases que no definan ningún constructor contienen implícitamente uno sin parámetros:

```
public MiClase()
{
    super();
}
```

Qué ocurre por defecto

- Los constructores que no llamen a otro constructor de la misma clase o del padre, contienen una llamada implícita al del padre sin parámetros:

```
public MiClase(int param)
{
    super();
    this.valor = param;
}
```
- Siempre que se haga referencia un atributo o método de la propia clase, implícitamente se añade `this`:
`this.miMetodo();`

Qué ocurre por defecto

- Todos los métodos de un interfaz son definidos como abstract de forma implícita:
`public abstract int miMetodo();`
- Todos los atributos son inicializados a su valor por defecto si no se inicializan de forma explícita. Ojo, que no ocurre lo mismo con las variables locales.
- Si no se especifica ningún package, la clase pertenece al package por defecto.
- Si no se especifica ningún modificador de acceso al definir una clase, atributo o método, se le aplica el modificador package.

instanceof

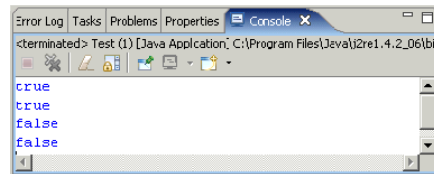
- Se trata de un operador especial del lenguaje Java representado por la keyword: instanceof
objeto instanceof clase
- num **instanceof** Integer ó num **instanceof** java.util.Date
- Este operador permite comprobar si un objeto es instancia de una clase o no.
- Devuelve un boolean por lo que puede utilizarse en expresiones lógicas (condiciones).
- Básicamente lo que hace es comprobar si un casting concreto se puede realizar o no.

Ejemplo





```

public class Test
{
    public static void main(String[] args)
    {
        Test.test(new String("Probando instanceof"));
    }
    public static void test(Object o)
    {
        System.out.println(o instanceof String);
        System.out.println(o instanceof Object);
        System.out.println(o instanceof Integer);
        System.out.println(o instanceof java.util.Calendar);
    }
}

```



Bibliografía

- 
Head First Java (2nd edition)
 Kathy Sierra y Bert Bates.
 O'Reilly
- 
Learning Java (2nd edition)
 Patrick Niemeyer y Jonathan Knudsen.
 O'Reilly.
- 
Thinking in Java (4th edition)
 Bruce Eckel.
 Prentice Hall.
- 
The Java tutorial
<http://java.sun.com/docs/books/tutorial/>

