Grado en Ingeniería Informática
Introducción a la Programación
2021-2022

Tema 5. Librerías básicas

- La clase `String`
- La clase `StringBuffer`
- Envoltorios de los tipos básicos. Wrappers
- La clase `Math`
- Las clases `BigInteger` y `BigDecimal`
- La clase `Arrays`

Tema 5. Librerías básicas

La clase `String`

Un *string* es una secuencia de caracteres. En la mayoría de los lenguajes de programación son tratados como array de caracteres, pero en Java un *string* es un objeto de la clase **`String`**. La clase **`String`** tiene 11 constructores y más de 40 métodos para manipular cadenas.

`String` se encuentra en el paquete **java.lang**

Está orientada a manejar cadenas de caracteres constantes, es decir, que yo no voy a poder cambiar. Se dice que son **inmutables**.

La clase **`StringBuffer`** está orientada a manejar cadenas de caracteres variables ó modificables.

Tema 5. Librerías básicas

La clase `String`

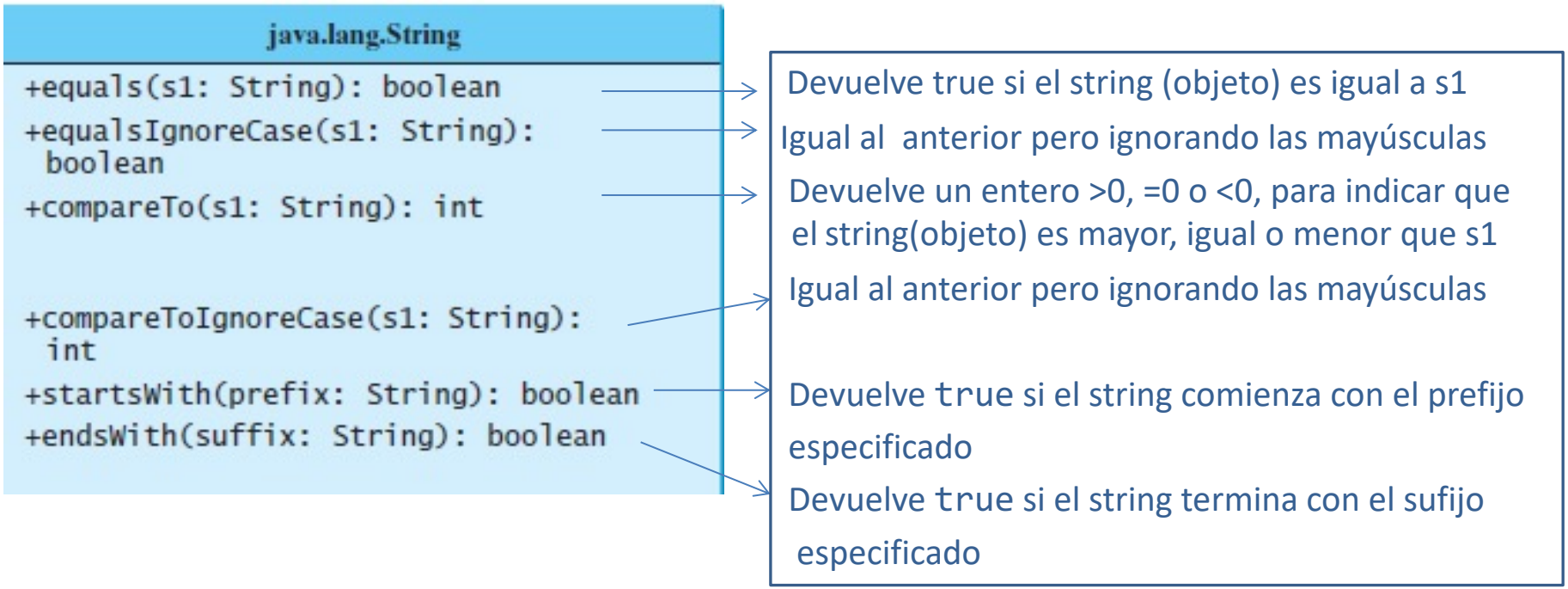
Construcción de un objeto de la clase `String`

- `String mensaje = new String("Bienvenido a Java");`
- `String mensaje = "Bienvenido a Java";`

Tema 5. Librerías básicas

La clase `String`

Comparación de cadenas



Tema 5. Librerías básicas

La clase `String`

```
String s1 = new String("Welcome to Java");
String s2 = "Welcome to Java";
```

Ejemplos de uso

```
if (s1.equals(s2))   // true  => compara contenidos
        System.out.println("s1 y s2  tienen el mismo contenido");
else
        System.out.println("s1 y s2  no tienen el mismo contenido");

if (s1 == s2)   // false  =>  compara referencias
        System.out.println("s1 y s2  son los mismos objetos");
else
        System.out.println("s1 y s2  no son los mismos objetos");
```

Tema 5. Librerías básicas

La clase `String`

```
String s1 = "Tomas";
String s2 = "Alberto";
```

Ejemplos de uso

```
if (s1.compareTo(s2)  > 0)
        System.out.println("s1  es  mayor que s2 ");   ← Salida
else if (s1.compareTo(s2)  < 0)
        System.out.println("s1  es menor que s2 ");
else
        System.out.println("s1  y s2  son iguales");
```

```
String str = "El primer programa";
```

```
boolean resultado = str.startsWith("El");     → true
boolean resultado = str.endsWith("programa"); → true
```

Tema 5. Librerías básicas

La clase `String`

Longitud del string, acceso a caracteres individuales y combinación de string

java.lang.String
+length(): int
+charAt(index: int): char
+concat(s1: String): String

Devuelve el número de caracteres del string

Devuelve el carácter cuya posición se especifica en **index**

Devuelve un string nuevo concatenando el string con s1

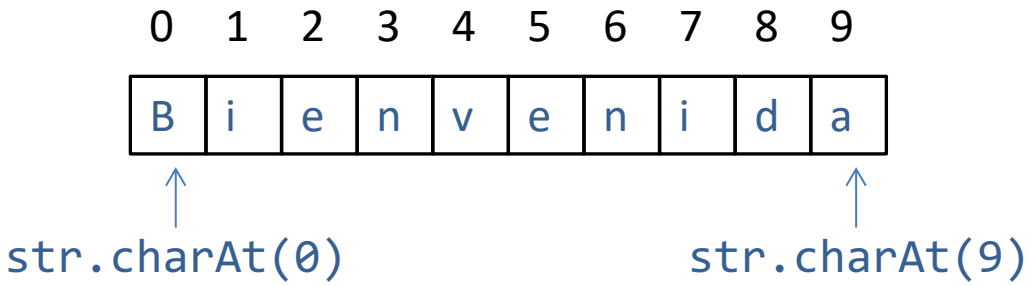
Tema 5. Librerías básicas

La clase String

Ejemplos de uso

```
int longitud = str.length();  → 10
```

```
String str = "Bienvenida";
```



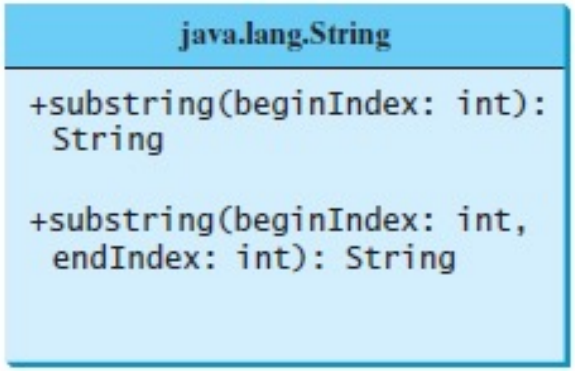

```
String s3 = s1.concat(s2)  equivalente a
String s3 = s1 + s2
```

```
String s1 = "Hola";
String s2 = " que tal";
```

Tema 5. Librerías básicas

La clase `String`

Obtener `substring` (subcadenas)



Devuelve una subcadena que comienza con el carácter cuya posición se especifica en **beginIndex** hasta el final del string
Devuelve una subcadena que comienza en el carácter cuya posición se especifica en **beginIndex** y termina en el carácter cuya posición es **endIndex** - 1

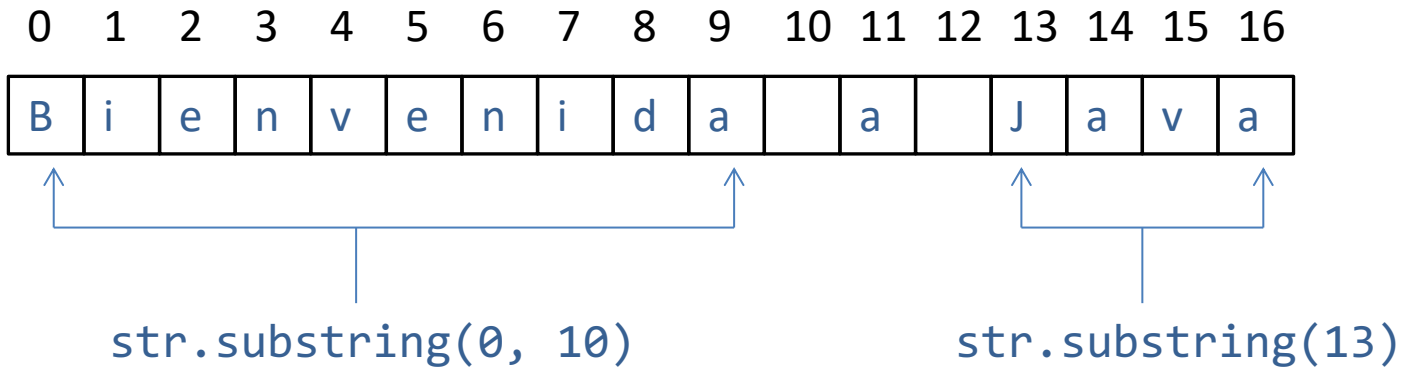
Métodos sobrecargados

Tema 5. Librerías básicas

La clase String

Ejemplos de uso

```
String str = "Bienvenida a Java";
```



Tema 5. Librerías básicas

La clase `String`

Convertir string

java.lang.String
+toLowerCase(): String
+toUpperCase(): String
+trim(): String

Devuelve un nuevo string con los caracteres en minúscula

Devuelve un nuevo string con los caracteres en mayúscula

Devuelve un nuevo string eliminando los caracteres blancos al principio y al final

Tema 5. Librerías básicas

La clase `String`

Ejemplos de uso

`"Hola".toLowerCase()`	devuelve un nuevo string, **hola**
`"Hola".toUpperCase()`	devuelve un nuevo string, **HOLA**
`"  Hola  ".trim()`	devuelve un nuevo string, **Hola** (sin blancos)

Tema 5. Librerías básicas

La clase `String`

Encontrar un carácter o un substring en un string

java.lang.String	
+indexOf(ch: char): int	Devuelve el índice (posición) de la 1ª ocurrencia de ch en el string. -1 si no está
+indexOf(ch: char, fromIndex: int): int	Devuelve el índice de la 1ª ocurrencia de ch después de **fromIndex**. -1 si no está
+indexOf(s: String): int	Devuelve el índice de la 1ª ocurrencia del string s en el string. -1 si no está
+indexOf(s: String, fromIndex: int): int	Devuelve el índice de la 1ª ocurrencia del string s en el string después de **fromIndex**. -1 si no está

¡OJO! **Métodos sobrecargados**

Tema 5. Librerías básicas

La clase `String`

Ejemplos de uso

```
"Bienvenida a Java".indexOf('B')          devuelve 0
"Bienvenida a Java".indexOf('e')          devuelve 2
"Bienvenida a Java".indexOf('e', 2)       devuelve 5
"Bienvenida a Java".indexOf("nida")       devuelve 6
"Bienvenida a Java".indexOf("Java", 5)    devuelve 12
"Bienvenida a Java".indexOf("java", 5)    devuelve -1
```

Tema 5. Librerías básicas

La clase `String`

Convertir caracteres y valores numéricos en string

java.lang.String
+valueOf(c: char): String
+valueOf(data: char[]): String
+valueOf(d: double): String
+valueOf(f: float): String
+valueOf(i: int): String
+valueOf(l: long): String
+valueOf(b: boolean): String

Devuelve un string que contiene el (carácter) `char c`

Devuelve un string que contiene los caracteres del array `data`

Devuelve un string que tiene el valor `d` `double`

Devuelve un string que tiene el valor `f` `float`

Devuelve un string que tiene el valor `i` `int`

Devuelve un string que tiene el valor `l` `long`

Devuelve un string que tiene el valor `b` `boolean`

¡OJO! → **Métodos de clase (sobrecargados)**

Tema 5. Librerías básicas

La clase `String`

Ejemplos de uso de `valueOf`

`String.valueOf(5.44)` devuelve un string que contiene "5.44"

`String.valueOf(10)` devuelve un string que contiene "10"

Tema 5. Librerías básicas

La clase `StringBuffer`

La clase **`StringBuffer`** es una alternativa a la clase **`String`**, siendo la primera más flexible ya que se puede añadir, insertar, es decir modificar el contenido de un objeto de esa clase. Los objetos **`StringBuffer`** son cadenas ***modificables*** a diferencia de los objetos de **`String`** que son ***inmutables***

Constructores

1) `StringBuffer s1 = new StringBuffer();`

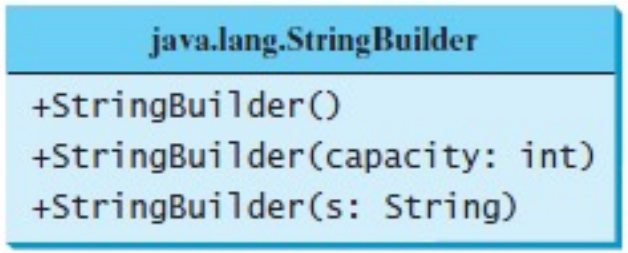
Crea un objeto `StringBuffer` sin caracteres y con una capacidad para 16 caracteres.

2) `StringBuffer s2 = new StringBuffer(10);`

Recibe un argumento entero y crea un objeto que no contiene caracteres y tiene una capacidad inicial que es la indicada en el parámetro.

Tema 5. Librerías básicas

La clase `StringBuffer`



3) `StringBuffer s3 = new StringBuffer("hola");`

Crea un objeto que contiene los caracteres que hemos pasado como parámetro y con una capacidad inicial igual al número de caracteres del argumento +16.

`StringBuffer` and **StringBuilder** son más flexibles que **String**. La clase **StringBuilder** es similar a **`StringBuffer`** excepto que los métodos para modificar el buffer en **`StringBuffer`** están *sincronizados*, lo que significa que sólo una tarea es permitida para ejecutar los métodos. Utilizaremos **`StringBuffer`** si la clase puede ser accedida por múltiples tareas al mismo tiempo. El uso de **StringBuilder** es más eficiente si se accede por una sola tarea, ya que no se necesita la sincronización en este caso.

Tema 5. Librerías básicas

La clase `StringBuffer`

Modificar strings con `StringBuffer`

java.lang.StringBuilder
+append(v: *aPrimitiveType*): StringBuilder
+append(s: String): StringBuilder
+delete(startIndex: int, endIndex: int): StringBuilder
+deleteCharAt(index: int): StringBuilder
+insert(offset: int, b: *aPrimitiveType*): StringBuilder
+insert(offset: int, s: String): StringBuilder
+replace(startIndex: int, endIndex: int, s: String): StringBuilder
+reverse(): StringBuilder
+setCharAt(index: int, ch: char): void

Añade un valor de tipo primitivo como un string

Añade un string al objeto de la clase `StringBuffer`

Borra caracteres desde la posición **startIndex** a **endIndex - 1**

Borra el carácter que está en la posición que indica **index**

Inserta un valor convertido a string en la posición **offset**

Inserta un string en la posición **offset**

Sustituye los caracteres desde **starIndex** hasta **endIndex -1** por el string **s**

Invierte los caracteres

Pone el nuevo carácter (**ch**) en la posición indicada por **index**

Tema 5. Librerías básicas

La clase `StringBuffer`

Ejemplos de uso

```
StringBuffer stringBuffer = new StringBuffer();
StringBuffer.append("Bienvenida");
StringBuffer.append(' ');
StringBuffer.append("a");
StringBuffer.append(' ');
```

StringBuffer.append(**"Java"**);	StringBuffer contiene **"Bienvenida a Java"**
StringBuffer.insert(13, **"HTML y ");**	StringBuffer contiene **"Bienvenida a HTML y Java"**
StringBuffer.delete(13, 20);	StringBuffer contiene **"Bienvenida a Java"**
StringBuffer.reverse();	StringBuffer contiene **"avaJ a adinevneiB"**
StringBuffer.setCharAt(0, 'b');	StringBuffer contiene **"bvaJ a adinevneiB“**

Tema 5. Librerías básicas

Clases Envoltorios de los tipos básicos. Wrappers

Los ***Wrappers*** (*envoltorios)* son clases diseñadas para ser un complemento de los **tipos primitivos.** Dichos tipos, son los únicos elementos de Java que no son objetos, ello supone un inconveniente ya que, en su inmensa mayoría, los métodos de Java requieren el uso de parámetros o argumentos que son objetos. Para evitar ello, Java ofrece una manera de *envolver* el tipo primitivo y convertirlo en objeto utilizando para cada tipo una clase:

`int` la clase envoltorio **`Integer`**

`double` la clase envoltorio **`Double`**

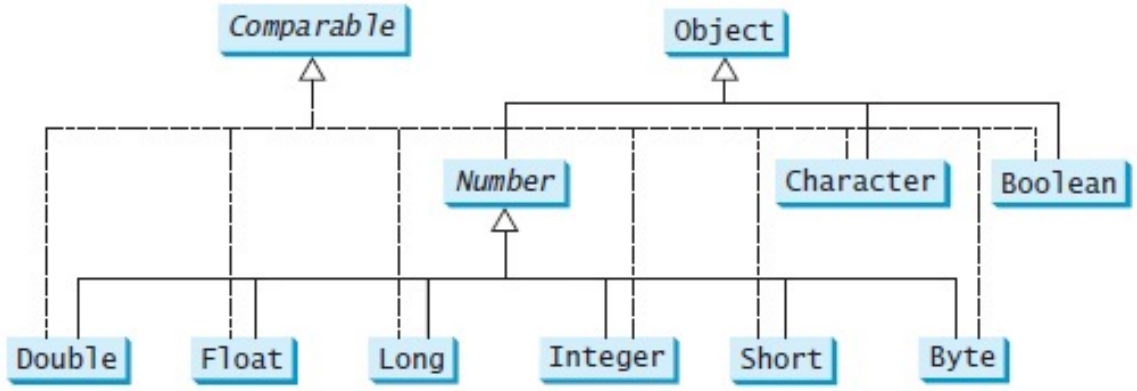
`byte` la clase envoltorio **`Byte`**

`char` la clase envoltorio **`Character`**, etc.

Tema 5. Librerías básicas

Clases Envoltorios de los tipos básicos. Wrappers

Jerarquía de clases



La clase **`Number`** es una superclase abstracta para **`Double`**, **`Float`**, **`Long`**, **`Integer`**, **`Short`** y **`Byte`**. Estas clases son extensiones, es decir heredan de **`Number`** los métodos abstractos

Además, las clases envoltorios implementan la interface **`Comparable`**

Tema 5. Librerías básicas

La clase `Character`

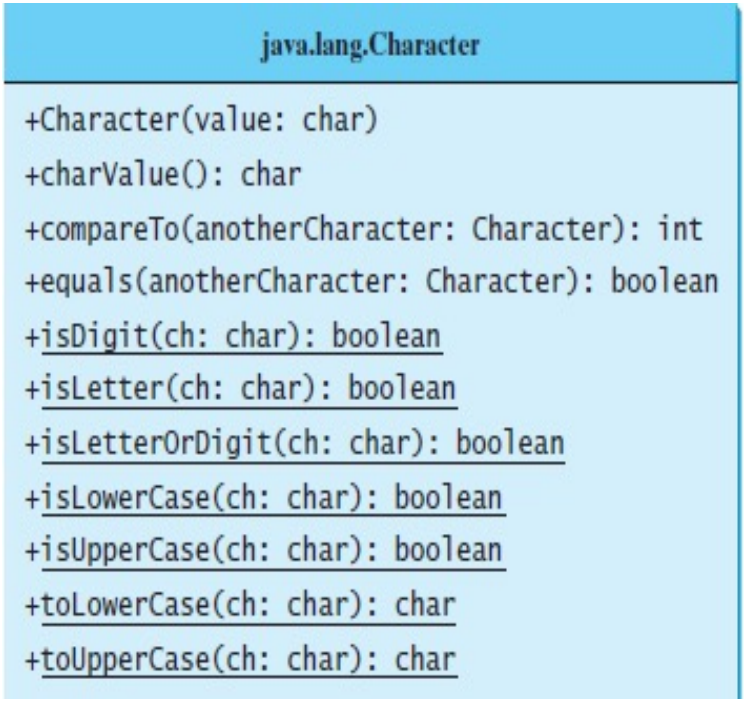
Es la clase envoltorio para el tipo primitivo **`char`.**

Podemos crear un objeto de la clase **`Character`** a partir de un valor de tipo **`char`**, utilizando el constructor:

```
Character caracter = new Character('a');
                    // el objeto caracter contiene el char 'a'
```

Tema 5. Librerías básicas

La clase Character



Constructor de un objeto carácter con un valor de tipo char

Devuelve el valor de tipo char de ese objeto

Compara un carácter con otro

Devuelve true si un carácter es igual a otro

Devuelve true si el carácter ch es un digito

Devuelve true si el carácter ch es una letra

Devuelve true si el carácter ch es un digito o una letra

Devuelve true si el carácter ch está en minúscula

Devuelve true si el carácter ch está en mayúscula

Devuelve la minúscula del carácter ch especificado

Devuelve la mayúscula del carácter ch especificado

Tema 5. Librerías básicas

La clase Integer

Es la clase envoltorio para el tipo primitivo **`int`**

Podemos crear un objeto de la clase **`Integer`** a partir de un valor de tipo **`int`** utilizando el constructor:

```
Integer entero1 = new Integer(25);
        // el objeto entero1 contiene el int 25
Integer entero2 = new Integer("456");
        // el objeto entero2 contiene el int 456
```

Tema 5. Librerías básicas

La clase Double

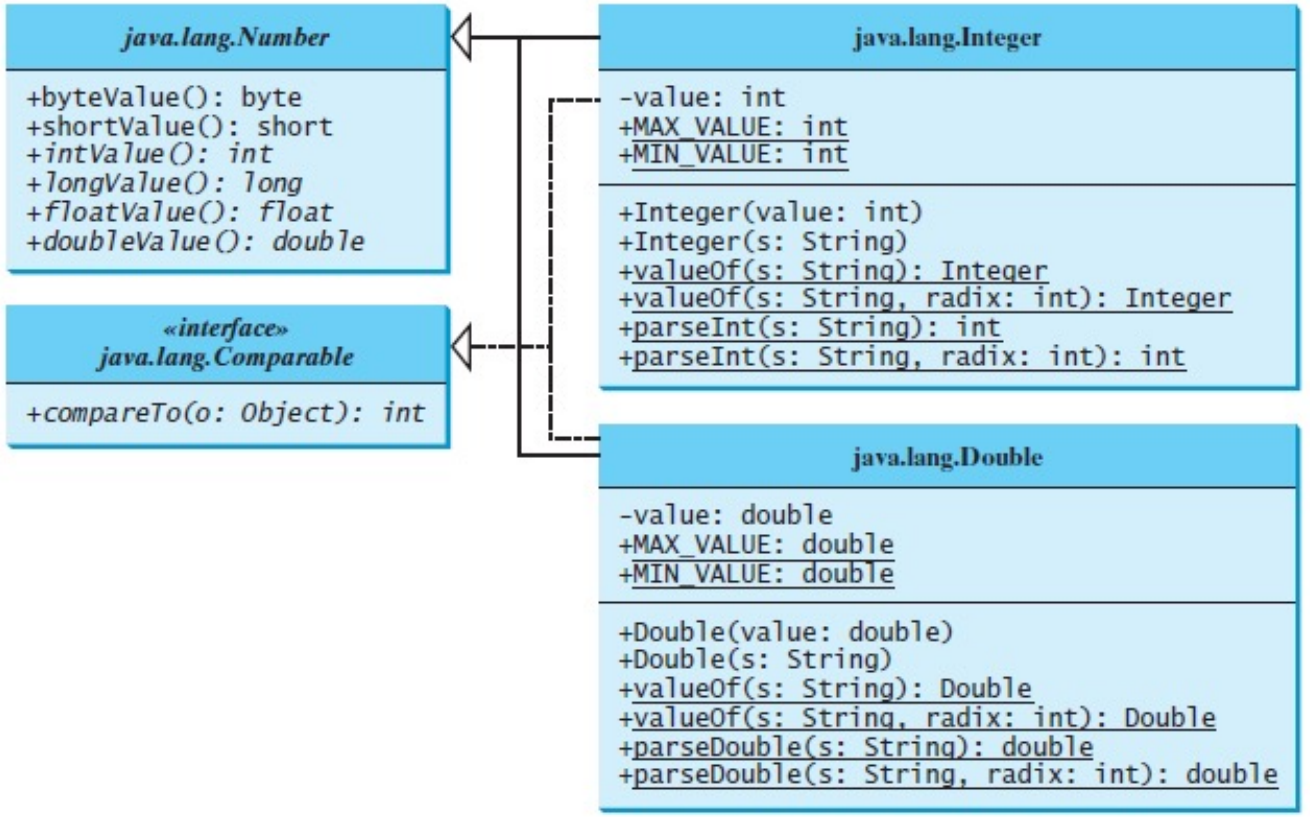
Es la clase envoltorio para el tipo primitivo **`double`**

Podemos crear un objeto de la clase **`Double`** a partir de un valor de tipo **`double`** utilizando el constructor:

```
Double real1 = new Double(25.0);
        // el objeto real1 contiene el double 25.0
Double real2 = new Double("456.75");
        // el objeto real2 contiene el double 456.75
```

Tema 5. Librerías básicas

Las clases `Integer` y `Double`



Ejemplo de uso

```java
package org.ip.tema04;

public class EjemplosEnvoltorios {

    public static void main(String[] args) {
        Character caracter = new Character('a');
        boolean digito = Character.isDigit('5');    // metodo de clase
        System.out.println(digito);
        int valor = caracter.compareTo('j');    // metodo de objeto, interfaz Comparable
        System.out.println(valor);
        char letra = Character.toUpperCase('a');    // metodo de clase
        System.out.println(letra);
        Integer entero1 = new Integer(45);
        Integer entero2 = new Integer("456");
        System.out.println("El máximo valor entero es " + Integer.MAX_VALUE);
        System.out.println("El mínimo valor real es " + Double.MIN_VALUE);
        int valor1 = entero1.intValue();    // metodo de Number y de objeto
        System.out.println(valor1);
        int valor2 = entero2.intValue();    // metodo de Number y de objeto
        System.out.println(valor2);
        Double real1 = new Double(25.0);
        Double real2 = new Double("456.75");
        double valor3 = real1.doubleValue();    // metodo de Number y de objeto
        System.out.println(valor3);
        double valor4 = real2.doubleValue();    // metodo de Number y de objeto
        System.out.println(valor4);
        double valor5 = Double.parseDouble("788.90");   // metodo de clase
        System.out.println(valor5);
    }
}
```

Salida

```
true
-9
A
El máximo valor entero es 2147483647
El mínimo valor real es 4.9E-324
45
456
25.0
456.75
788.9
```

Ejemplo de uso

```
1  package org.ip.tema04;
2
3  public class ConvertirBinarioADecimal {
4
5      public static void main(String[] args) {
6          String strNumeroBinario = "111000";
7
8          // Convierte un número binario a decimal utilizando
9          // el método de clase ParseInt de la clase Integer
10         // El segundo argumento indica la base
11
12         int numeroDecimal = Integer.parseInt(strNumeroBinario, 2);
13         System.out.println("El número binario " + strNumeroBinario
14                 + " convertido a número decimal es " + numeroDecimal);
15     }
16 }
```

Salida `El número binario 111000 convertido a número decimal es 56`

Tema 5. Librerías básicas

La clase Math

Es una clase que tiene:

- métodos de clase
- constantes E y PI

Ejemplos de uso:

```
double area = Math.PI * radio * radio;
double raiz = Math.sqrt(25);
```

Tema 5. Librerías básicas

Las clases `BigInteger` y `BigDecimal`

Si necesitamos utilizar enteros muy grandes o reales con muy alta precisión, debemos usar las clases `BigInteger` y `BigDecimal` del paquete **java.math**. Ambas son *inmutables*, heredan de la clase `Number` e implementan la interface `Comparable`. El mayor entero de tipo `long` es Long.MAX_VALUE (9223372036854775807). Una instancia o ejemplar u objeto de `BigInteger` puede representar un entero de cualquier tamaño.

Usaremos: `new BigInteger(String)` y `new BigDecimal(String)` para crear una instancia de `BigInteger` y `BigDecimal`

Usaremos: **`add`**, **`subtract`**, **`multiply`**, **`divide`** para realizar operaciones aritméticas y **`compareTo`** para comparar dos números grandes

Tema 5. Librerías básicas

Las clases BigInteger y BigDecimal

Ejemplo de uso:

```
BigInteger a = new BigInteger("92233720368547758O7");
BigInteger b = new BigInteger("2");
BigInteger c = a.multiply(b); // 9223372036854775807 * 2
System.out.println(c);
```

Salida: 18446744073709551614

Tema 5. Librerías básicas

Las clases BigInteger y BigDecimal

Ejemplo de uso:

```
BigDecimal a = new BigDecimal(1.0);
BigDecimal b = new BigDecimal(3);
BigDecimal c = a.divide(b, 20, BigDecimal.ROUND_UP);
System.out.println(c);
```

Salida: 0.33333333333333333334

Ejemplo de uso:

```java
package org.ip.tema04;

import java.math.BigInteger;

public class FactorialGrande {

    public static BigInteger factorial(int n) {
        BigInteger resultado = BigInteger.ONE;  // equivalente a new BigInteger("1")
        for (int i = 1; i <= n; i++)
            resultado = resultado.multiply(new BigInteger(i + ""));
        return resultado;
    }

    public static void main(String[] args) {
        System.out.println("50! = " + factorial(50));
    }
}
```

Salida `50! = 30414093201713378043612608166064768844377641568960512000000000000`

Tema 5. Librerías básicas

La clase `Arrays`

El paquete **java.util** contiene la clase **`Arrays`** que proporciona varios **métodos estáticos** (`static`) para realizar distintas operaciones con arrays, tales como:

`Arrays.sort(a)` ordena los elementos del array **`a`**, siempre y cuando los elementos de array implementen la interface `Comparable`

`Arrays.equals(a, b)` comprueba si los arrays **`a`** y **`b`** son iguales. Verifica que contienen el mismo nº de elementos, y que cada elemento es igual a su correspondiente en el otro array usando el método `equals` del elemento

`Arrays.fill(a, valor)` rellena el array **`a`** con el valor **`valor`**

`Arrays.toString(a)` devuelve una cadena (`String`) con el contenido del array **`a`** que se pasa como parámetro

Tema 5. Librerías básicas

La clase `Arrays`

- **`Arrays.binarySearch(a, k)`** busca el valor **`k`** en el array **`a`**. Es requisito que a esté ordenado, implementando los elementos la interface `Comparable`
- **`copyOf()`** y **`copyOfRange()`** copian un array origen en otro destino (creando el destino). Permite varios parámetros como: array de origen, posición inicial del array de origen desde la que vamos a empezar a copiar, array de destino (debe ser del mismo tipo de elementos que el de origen, si no, obtendremos un error), posición inicial del array de destino donde vamos a empezar a copiar y número de elementos a copiar. Ver System.arrayCopy()
- `http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html`

Diagrama de métodos de la clase `Arrays` muy resumido

Method	Purpose
`static int binarySearch(type [] a, type key)`	Searches the specified array for the specified key value using the binary search algorithm
`static boolean equals(type[] a, type[] a2)`	Returns `true` if the two specified arrays of the same type are equal to one another
`static void fill(type[] a, type val)`	Assigns the specified value to each element of the specified array
`static void sort(type[] a)`	Sorts the specified array into ascending order
`static void sort(type[] a, int fromIndex, int toIndex)`	Sorts the specified range of the specified array into ascending order

Ejemplo de uso

```
1  package org.ip.tema04;
2
3  import java.util.Arrays;
4
5  public class Ejemplo3Arrays {
6
7      public static void main(String[] args) {
8          int [] array1 = {2, 4, 7, 10};
9          int [] array2 = {2, 4, 7, 10};
10         int [] array3 = {20, 30, 40, 50};
11         int valor = 50;
12         if (Arrays.equals(array1, array2))
13             System.out.println(Arrays.toString(array1) + " coincide con "
14                     + Arrays.toString(array2));
15         else
16             System.out.println(Arrays.toString(array1) + " no coincide con "
17                     + Arrays.toString(array2));
18         int posicion = Arrays.binarySearch(array3, valor);
19         if (posicion < 0)
20             System.out.println("El valor " + valor + " no está en "
21                     + Arrays.toString(array3));
22         else
23             System.out.println("El valor " + valor + " está en la posición "
24                     +(posicion + 1) + " del array " + Arrays.toString(array3));
25     }
26 }
```

Salida

```
[2, 4, 7, 10] coincide con [2, 4, 7, 10]
El valor 50 está en la posición 4 del array [20, 30, 40, 50]
```

¡MUCHAS GRACIAS!

UNIVERSIDAD DE ALMERÍA