

---

# P00 en Java

## Destacados del lenguaje

---

UNQ - PO2

---

Profundizar con bibliografía y con pruebas todo lo marcado con la etiqueta **Leer**

# Temario

- Paquetes
- Clases
- Test
- Variables de instancia
- Visibilidad
- Sistema de tipos en Java
  - Tipos primitivos
  - Tipos referenciados (los que usaremos)
  - Clases inmutables: String, Wrappers.
- Constructores
- Métodos
- ArrayList
- Implementar uno de los ejemplos enviados

# Paquetes

- Agrupa clases con características comunes.
- Convención para el Formato del nombre. De lo más general a lo más particular. Ejemplo: ar.edu.unq.po2....

You form a unique package name by first having (or belonging to an organization that has) an Internet domain name, such as `oracle.com`. You then reverse this name, component by component, to obtain, in this example, `com.oracle`, and use this as a prefix for your package names, using a convention developed within your organization to further administer package names. Such a convention might specify that certain package name components be division, department, project, machine, or login names.

- Se corresponde con folders en el file system.
- Tienen una estructura jerárquica
- Útil para manejar niveles de accesibilidad
- Evita colisión de nombres

# Clases

- Tiene una declaración y un cuerpo

```
ClassDeclaration:
```

```
  NormalClassDeclaration
```

```
  EnumDeclaration
```

```
NormalClassDeclaration:
```

```
  {ClassModifier} class TypeIdentifier [TypeParameters] [Superclass] [Superinterfaces] ClassBody
```

- Modificadores
  - **public**
  - abstract. Veamos un ejemplo.
  - final

¿Tiene sentido que una clase sea abstract y final?

# Clases

- Tiene una declaración y un cuerpo

```
ClassDeclaration:
```

```
  NormalClassDeclaration
```

```
  EnumDeclaration
```

```
NormalClassDeclaration:
```

```
  {ClassModifier} class TypeIdentifier [TypeParameters] [Superclass] [Superinterfaces] ClassBody
```

- Modificadores
  - public
  - **abstract**
  - final

# Clases

- Tiene una declaración y un cuerpo

*ClassDeclaration:*

*NormalClassDeclaration*

*EnumDeclaration*

*NormalClassDeclaration:*

*{ClassModifier} class TypeIdentifier [TypeParameters] [Superclass] [Superinterfaces] ClassBody*

- Modificadores

- public
- abstract
- **final**

```
public final abstract class Plato {
```

```
//Variables de instancia  
String nombre;
```

❌ The class Plato can be either abstract or final, not both

Press 'F2' for focus

# Clases

- Convención para Nombre de la clase
- Nombre largo y nombre corto
- Archivo .java dentro de la carpeta del paquete
- Las clases que quieran usar y no estén en el mismo paquete se deben importar.

# Clases

- 3 tipos de comentarios. [Leer.](#)
- El cuerpo es un bloque (código Java encerrado entre dos llaves). ¿Qué contiene?



# Clases - Variables de instancia

```
public class AtencionHospitalaria {  
    //Variables de instancia  
    int ejemploTipoPrimitivo = 3;  
    Integer ejemploWrapperDeTipoNoPrimitivo;  
    String ejemploClaseInmutable = "un string";  
    Paciente ejemploClasePropia = new Paciente();  
  
    //Constructores  
  
    //métodos  
}
```

# Variables de instancia - Nombre

[modificadores] <tipo de dato> *<nombre>* [=valor inicial]

Convención de nombres

# Variables de instancia - Modificadores

*[modificadores]* <tipo de dato> <nombre> [=valor inicial]

Modificador de accesibilidad: **public**, **protected**, **private**

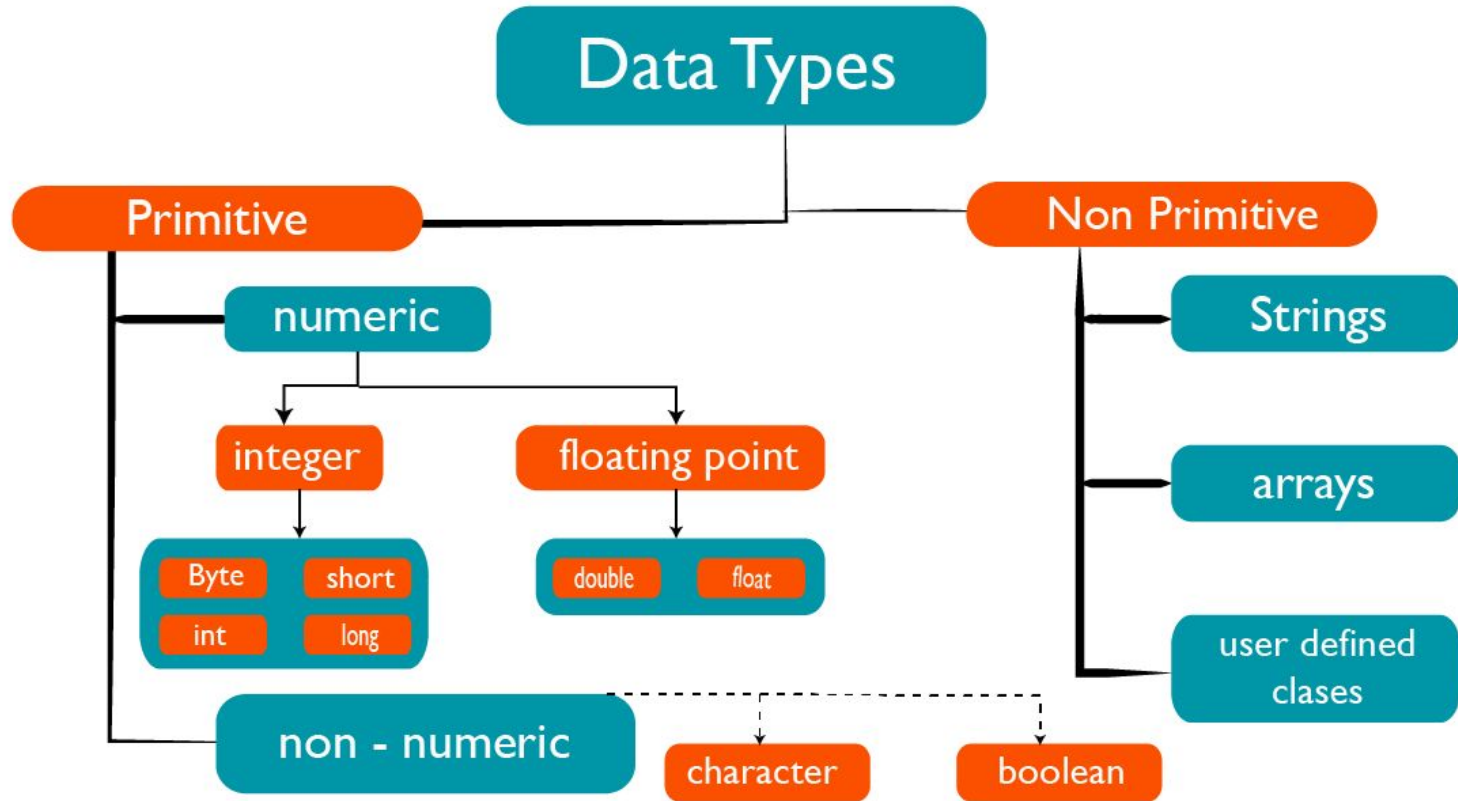
Access Levels				
Modifier	Class	Package	Subclass	Everyone
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗

# Variables de instancia - Modificadores

*[modificadores]* <tipo de dato> <nombre> [=valor inicial]

- Otros modificadores (que no son de accesibilidad)
  - **final**: su valor es obligatorio al declararla. No se puede modificar.
  - **static**: variables de clase

# Sistema de tipos en Java



# Primitive vs. non Primitive

```
int unNumero = 3;
```

```
Punto unOjeto = new Punto(3,4);
```

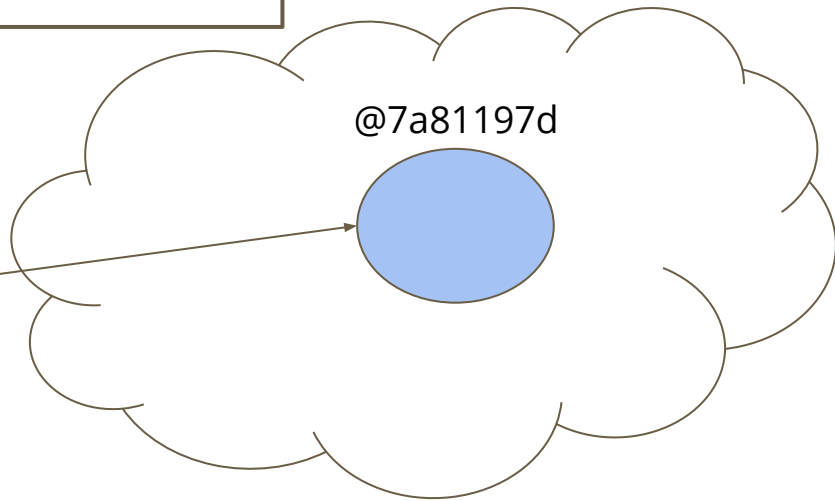
# Sistema de tipos en Java

La variable de tipo primitivo almacena valor literal y la no primitiva la referencia al objeto

```
int unNumero = 3;  
Punto unObjeto = new Punto(3,4);
```

unNumero (almacena el literal en la stack)	3
unObjeto (almacena la referencia a la memoria heap)	@7a81197d

**THE STACK**



**THE HEAP**

# Primitive vs. non Primitive

```
int unNumero = 3;
```

```
int copiaNumero = unNumero;
```

```
Punto unObjeto = new Punto(3,4);
```

```
Punto otroObjeto = unObjeto;
```



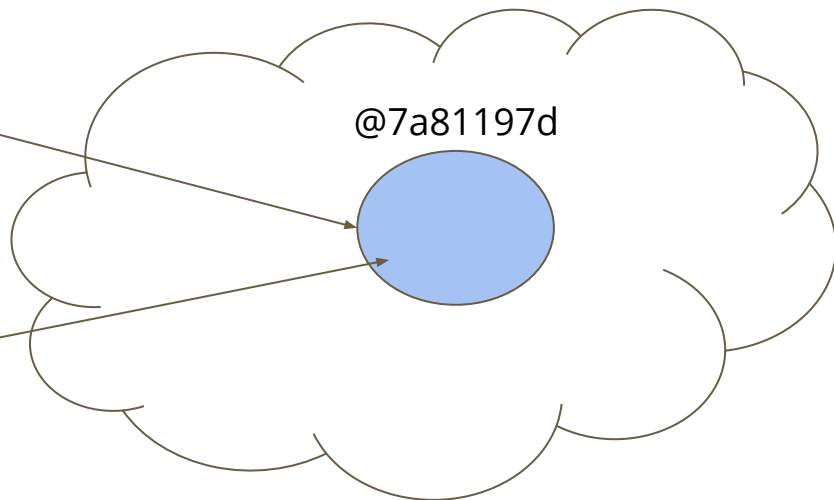
# Primitive vs. non Primitive

La variable de tipo primitivo almacena valor literal y la no primitiva la referencia al objeto

```
int unNumero = 3;  
int copiaNumero = unNumero;  
Punto unObjeto = new Punto(3,4);  
Punto otroObjeto = unObjeto;
```

unNumero (almacena el literal en la stack)	3
unObjeto (almacena la referencia a la memoria heap)	@7a81197d
copiaNumero (es una copia)	3
otroObjeto (se copia la referencia)	@7a81197d

**THE STACK**

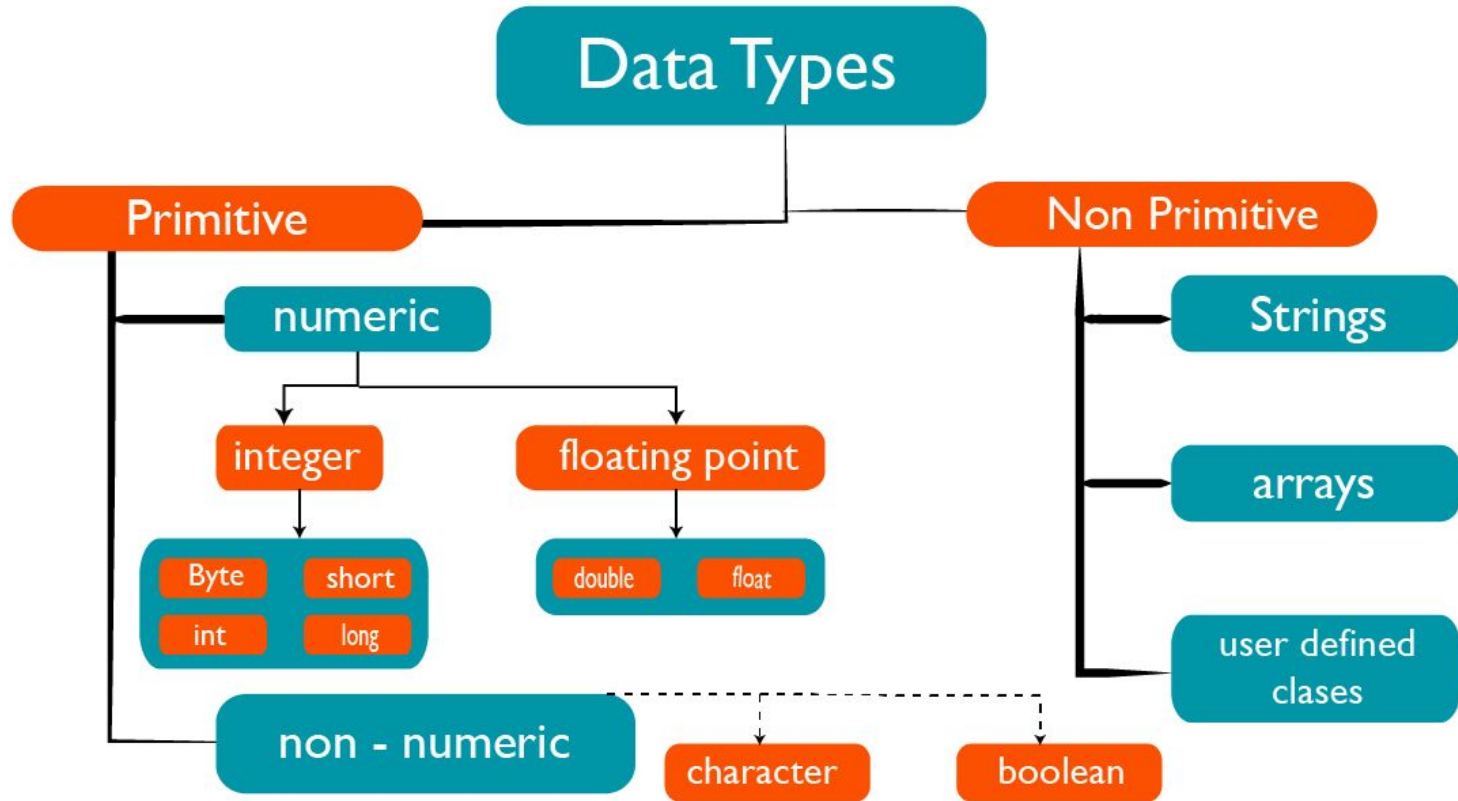


**THE HEAP**

# Sistema de tipos en Java

<b>Primitive</b>	<b>Non-primitive</b>
Pasaje por valor	Pasaje por referencia
El valor por defecto es un literal	El valor por defecto es null
Los valores que se asignan son literales	Los valores que se asignan son siempre referencias

# Sistema de tipos en Java



# Sistema de tipos en Java - Tipos primitivos

Reserved Word	Data Type	Size	Range of Values
byte	Byte Length Integer	1 bytes	$-2^8$ to $2^7 - 1$
short	Short Integer	2 bytes	$-2^{16}$ to $2^{16} - 1$
int	Integer	4 bytes	$-2^{32}$ to $2^{31} - 1$
long	Long Integer	8 bytes	$-2^{64}$ to $2^{63} - 1$
float	Single Precision	4 bytes	$-2^{32}$ to $2^{31} - 1$
double	Real number with double	8 bytes	$-2^{64}$ to $2^{62} - 1$
char	Character ( 16 bit unicode )	2 bytes	0 to 216 - 1
boolean	Has value true or false	A boolean value	true or false

# Sistema de tipos en Java - Tipos primitivos

## Casting automático

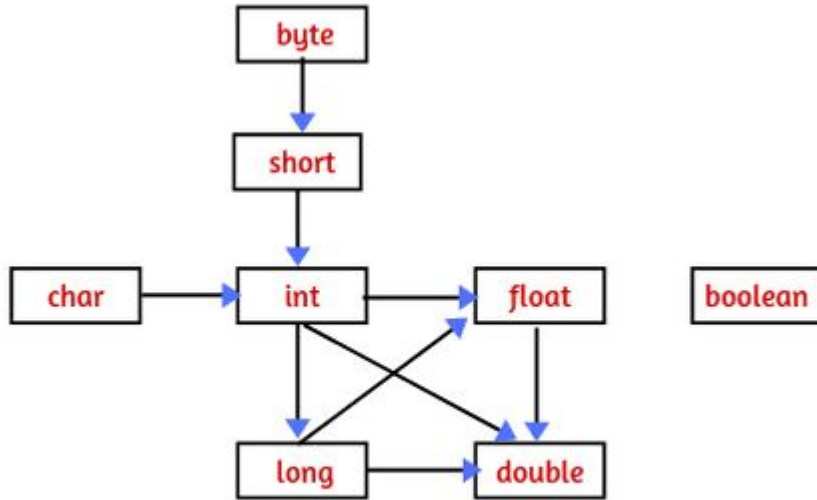


Fig: Automatic type conversion that Java allows.

```
float unFloat = 5;  
double unDouble = unFloat;  
  
double otroDouble = 5;  
float otroFloat = unDouble;
```

Type mismatch: cannot convert from double to float

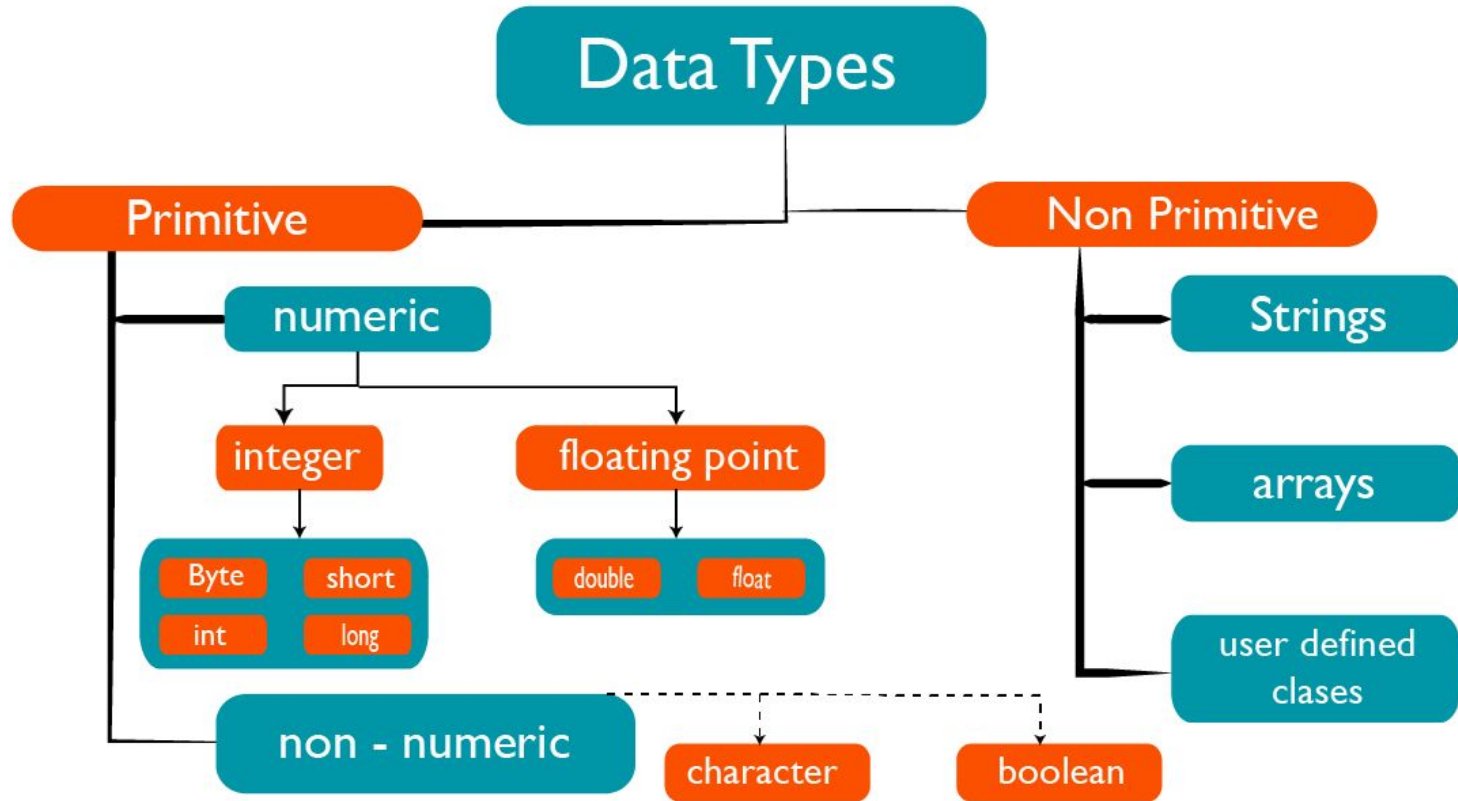
3 quick fixes available:

- [Add cast to 'float'](#)
- [Change type of 'otroFloat' to 'double'](#)
- [Change type of 'unDouble' to 'float'](#)

Press 'F2' for focus

Leer narrowing conversion vs widening conversion

# Sistema de tipos en Java



# Sistemas de tipos en Java - Non primitive

- Diferencia entre identidad e igualdad

# Sistemas de tipos en Java - Non primitive

- Redefinición del método equals. Por defecto hereda el de Object (compara por identidad). Leer.

The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object (x == y has the value true).

Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes.

**Parameters:**

obj - the reference object with which to compare.

**Returns:**

true if this object is the same as the obj argument; false otherwise.

**See Also:**

hashCode(), HashMap



# Redefinición de hashCode

```
public int hashCode()
```

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by `HashMap`.

The general contract of `hashCode` is:

- Whenever it is invoked on the `same object` more than once during an execution of a Java application, the `hashCode` method must consistently return the `same integer`, provided no information used in `equals` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the `same integer` result.
- It is *not* required that if two objects are unequal according to the `equals(java.lang.Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing `distinct integer results for unequal objects may improve the performance of hash tables`.

# Sistemas de tipos en Java - Non primitive

- Clases inmutables: no son modificables.
- Dos ejemplos importantes: clases Wrappers y Strings.

# Classes inmutables

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence,
               Constable, ConstantDesc {
```


```
    * @return  a string that represents the concatenation of this object's
    *           characters followed by the string argument's characters.
    */
    public String concat(String str) {
        if (str.isEmpty()) {
            return this;
        }
        return StringConcatHelper.simpleConcat(this, str);
    }
```

```
String a = "hola";
```

```
String b = "hola" + "Vicky";
```


```
String c = b.substring(0, 4);
```

```
System.out.println(a);
```




"hola"

```
System.out.println(c);
```




"hola"

```
System.out.println(a==c);
```



?

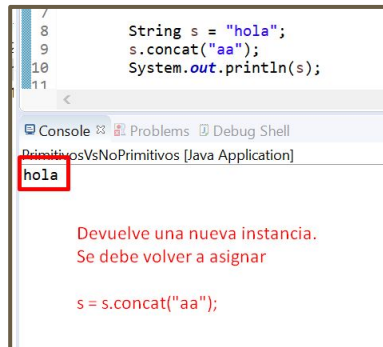
```
System.out.println(a.equals(c));
```



?

# Sistemas de tipos en Java - Strings

- Los Strings compárenlos siempre con equals.
- El método concat devuelve una nueva instancia. La tiene que reasignar



The screenshot shows an IDE window with a Java file named 'PrimitivosVsNoPrimitivos.java'. The code in the editor is as follows:

```
7  
8     String s = "hola";  
9     s.concat("aa");  
10    System.out.println(s);  
11
```

Below the code editor, the 'Console' tab is active, displaying the output of the program:

```
hola
```

Below the console output, there is a red text message:

```
Devuelve una nueva instancia.  
Se debe volver a asignar
```

At the bottom of the console output, the code snippet `s = s.concat("aa");` is shown in red.

- Es una clase inmutable → al pasarla como parámetro, los cambios no serán vistos desde afuera.

# Sistemas de tipos en Java - Strings

- Leer: <https://www.baeldung.com/java-compare-strings>
- StringBuffer en casos especiales en los que vayan a efectuarse muchos cambios sobre el string. Es siempre la misma instancia.

# Sistemas de tipos en Java - Strings

- Gran cantidad de métodos. **Leer.**
- Lean expresiones regulares.

```
• charAt(int index) : char - String
• chars() : IntStream - String
• codePointAt(int index) : int - String
• codePointBefore(int index) : int - String
• codePointCount(int beginIndex, int endIndex) : int - String
• codePoints() : IntStream - String
• compareTo(String anotherString) : int - String
• compareToIgnoreCase(String str) : int - String
• concat(String str) : String - String
• contains(CharSequence s) : boolean - String
• contentEquals(CharSequence cs) : boolean - String
• contentEquals(StringBuffer sb) : boolean - String
• describeConstable() : Optional<String> - String
• endsWith(String suffix) : boolean - String
• equals(Object anObject) : boolean - String
• equalsIgnoreCase(String anotherString) : boolean - String
• formatted(Object... args) : String - String
• getBytes() : byte[] - String
• getBytes(Charset charset) : byte[] - String
• getBytes(String charsetName) : byte[] - String
• getBytes(int srcBegin, int srcEnd, byte[] dst, int dstBegin) : void - String
• getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) : void - String
• getClass() : Class<?> - Object
• hashCode() : int - String
• indent(int n) : String - String
• indexOf(int ch) : int - String
• indexOf(String str) : int - String
• indexOf(int ch, int fromIndex) : int - String
• indexOf(String str, int fromIndex) : int - String
• intern() : String - String
• isBlank() : boolean - String
• isEmpty() : boolean - String
```

# Clases - métodos

```
public class AtencionHospitalaria {  
  
    //Variables de instancia  
    int ejemploTipoPrimitivo = 3;  
    Integer ejemploWrapperDeTipoNoPrimitivo;  
    String ejemploClaseInmutable = "un string";  
    Paciente ejemploClasePropia = new Paciente();  
  
    //Constructores  
  
    //métodos  
  
}
```

---



# Clases - Métodos

*[modificadores] tipoRetorno nombre ( listDeParámetros ) [ throws exceptions ]*

- **Modificadores**

- De acceso: los mismos que para variables de instancia
- Otros
  - static
  - abstract
  - final

# Clases - Métodos

*[modificadores] tipoRetorno nombre ( listDeParámetros ) [ throws exceptions ]*

- Variables locales
  - solo pueden tener el modificador final (no se pueden modificar una vez asignado el valor)
  - Se destruyen al terminar el método
- Excepciones: leer

# Colecciones - ArrayList

- Una instancia de la clase `java.util.ArrayList` constituye una colección lineal ordenada, de crecimiento dinámico, que puede contener elementos duplicados
- Ofrece acceso posicional en tiempo constante.
- Está indexada comenzando desde cero
- Cada vez que se agrega un elemento, la lista lo ubica en la última posición

# Colecciones - ArrayList

- Se pueden agregar elementos en otras posiciones, lo cual genera desplazamiento a derecha.
- También pueden eliminarse elementos de cualquier posición de la lista, lo cual genera desplazamiento a izquierda.

# Colecciones - ArrayList

- Java permite tipar una colección especificando el tipo de los elementos que contendrá. Este tipo podrá ser una clase o una interfaz, pero no un tipo primitivo (para esto último deberían utilizarse wrappers).
- El tipo de los elementos de la colección se especifica entre “<” y “>”, de la siguiente manera:  
`ArrayList<T> platos;` en donde “T” indica el nombre de una clase o interfaz.
- Leer cookbook de ArrayList.

# Creación de listas

```
List<Integer> i = new ArrayList<Integer>();  
i.add(1);  
i.add(2);  
i.add(3);
```

Camino corto con el mismo resultado

```
List<Integer> i2 = Arrays.asList(1,2,3);
```

# Colecciones - Map

- Es una estructura de datos que permite almacenar pares clave/valor (en otros lenguajes conocida como Dictionarios).

```
Map<Persona, Mapadre> padres = new HashMap<Persona, Mapadre>();
```

```
Persona juan = new Persona("Juan");  
Persona julian = new Persona("Julian");  
Persona mariana = new Persona("Mariana");  
Persona juana = new Persona("Juana");  
Persona norma = new Persona("Norma");  
|  
Mapadre julmar = new Mapadre(julian, mariana);  
  
padres.put(juana, julmar);
```

Importante: si usan objetos como claves redefinir equals y hashCode