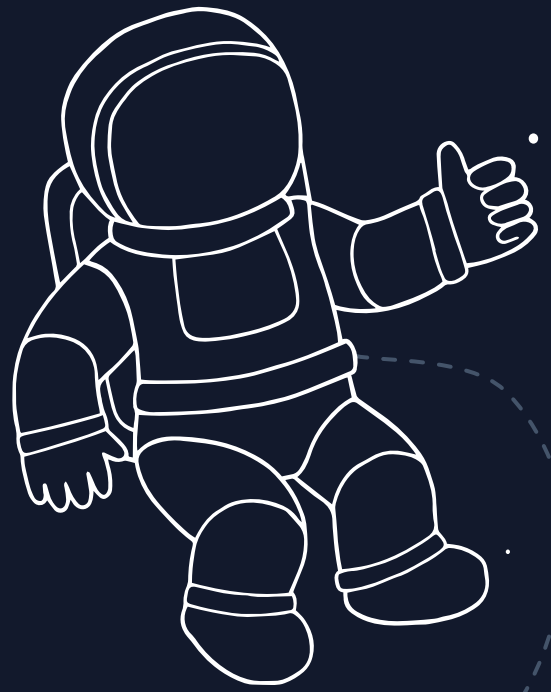
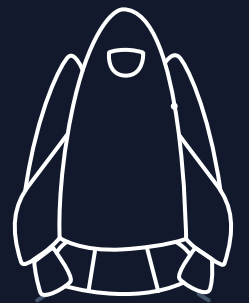


Programa académico CAMPUS

MODULO JAVA
Sesión 4
Cadenas y Conversiones

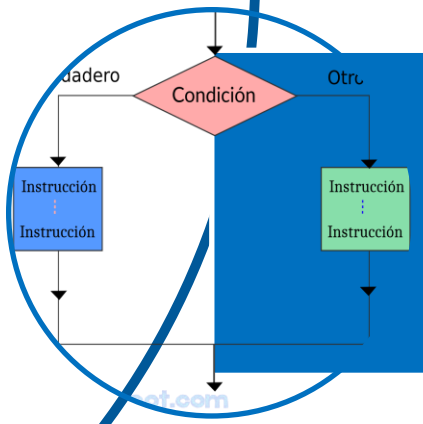


Trainer Carlos H. Rueda C.


Java™

TEMARIO

Cadenas



Conversiones

CREACIÓN DE CADENAS

Por lo general las cadena se crean a partir de cadenas literales. Cuando el compilador encuentra una serie de caracteres entre comillas (`""`), crea un objeto `String` cuyo valor es el propio texto. El esquema general es el siguiente:

```
String nombre = "Hola mundo";
```

```
String s = new String("Hola Mundo.");
```

CONCATENACIÓN DE CADENAS

Concatenación de cadenas Java permite concatenar cadenas fácilmente utilizando el operador `+`. La sentencia de código a continuación concatena tres cadenas para producir su salida:

```
"La entrada tiene " + contador + " caracteres."
```

Dos de las cadenas concatenadas son cadenas literales: "La entrada tiene " y " caracteres.". La tercera cadena (la del medio) es realmente un entero que primero se convierte a cadena y luego se concatena con las otras.

LONGITUD DE CADENAS

Uno de los métodos más habituales que se utilizan en un `String` es `length`, que devuelve el número. de caracteres de una cadena:

```
String s = "abc";
System.out.println(s.length());
```

EXTRACCIÓN DE CARACTERES

Para extraer un único carácter de una cadena, se puede referir a un carácter indexado mediante el método `charAt`, la sintaxis es la siguiente:

```
cadena.charAt(índice);
"abc".charAt(1)
```

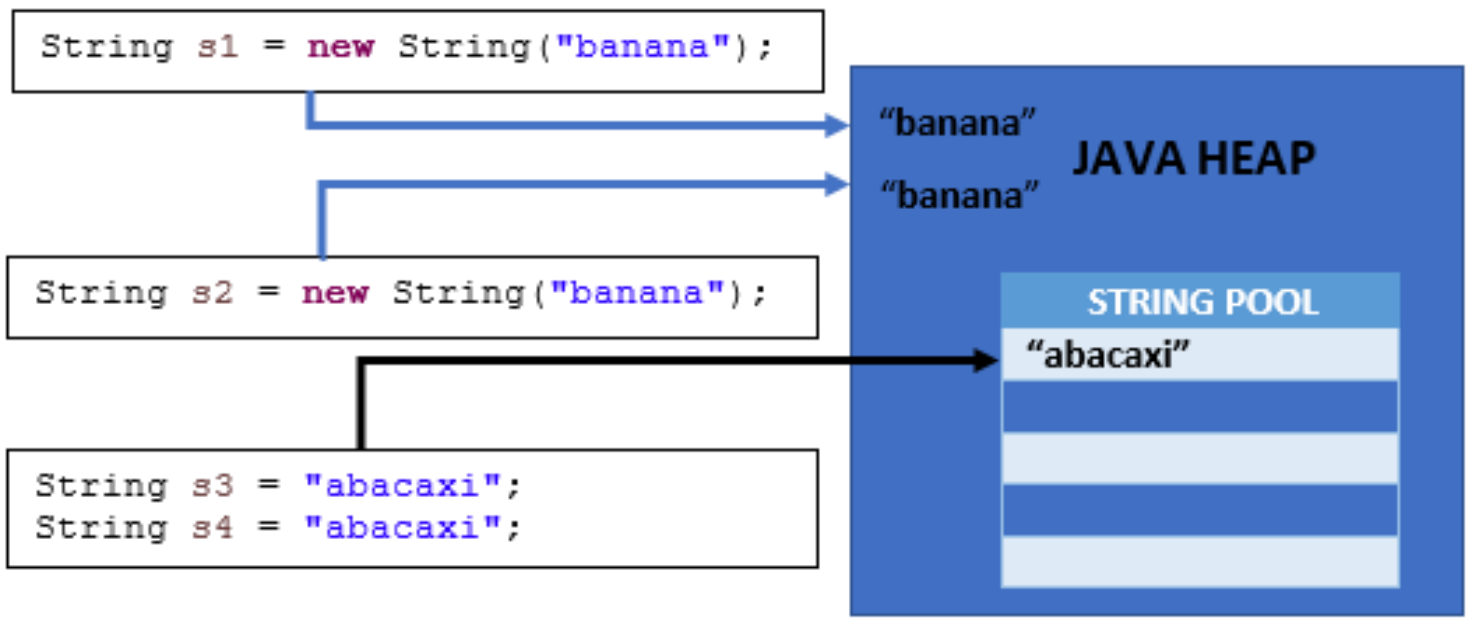
COMPARACIÓN DE CADENAS

Si se desea comparar dos cadenas para saber si son iguales, se puede utilizar el método `equals` de `String`. Devolverá `true` si el único parámetro está compuesto de los mismos caracteres que el objeto con el que se llama a `equals`.

```
String s1 = new String("banana");
String s2 = new String("banana");

System.out.println(s1.equals(s2));  //true
System.out.println(s1 == s2);   //false

String s3 = "abacaxi";
String s4 = "abacaxi";

System.out.println(s3.equals(s4));  //true
System.out.println(s3 == s4);   //true;
```

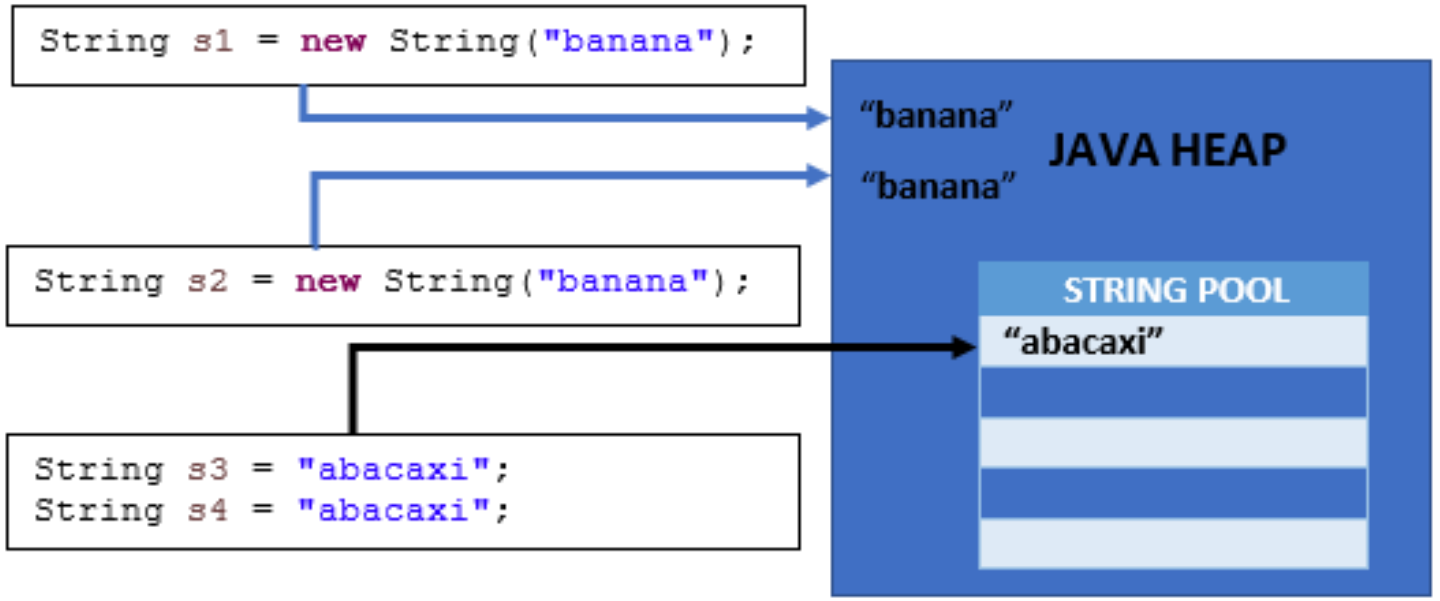
COMPARACIÓN DE CADENAS

```
String s1 = new String("banana");
String s2 = new String("banana");

System.out.println(s1.equals(s2));  //true
System.out.println(s1 == s2);   //false

String s3 = "abacaxi";
String s4 = "abacaxi";

System.out.println(s3.equals(s4));  //true
System.out.println(s3 == s4);   //true;
```



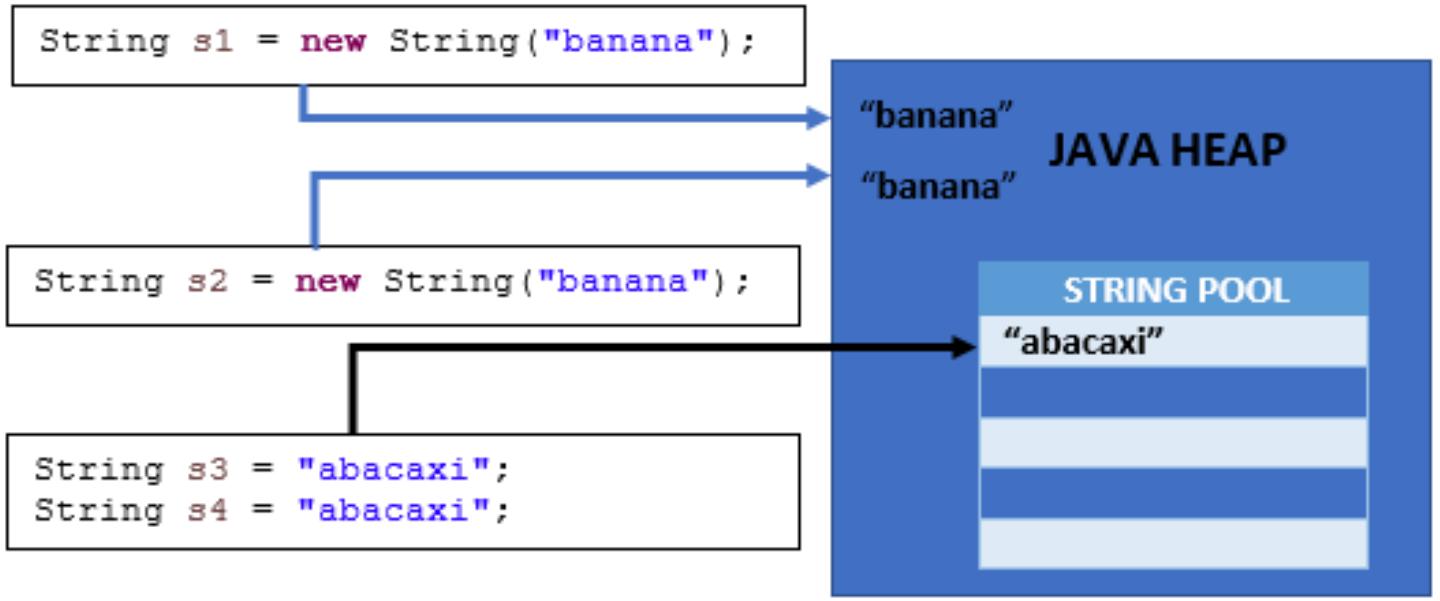
POOL DE STRINGS EN JAVA

El *pool de strings* es una área especial de la memoria, administrada por la JVM (Java Virtual Machine), que se utiliza para almacenar cadenas literales (strings que se escriben directamente en el código).
En lugar de crear un nuevo objeto de `String` cada vez que se declara una cadena, Java reutiliza objetos de `String` que ya existen en el pool.



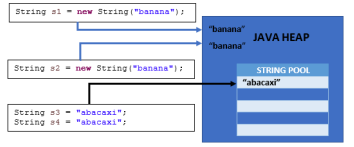
HEAP EN JAVA

El **heap** es una parte de la memoria donde se almacenan los objetos en Java, incluidos los objetos `String` que se crean con el operador `new`. A diferencia de los strings literales, los objetos `String` creados con `new` no van al pool de strings, sino que se almacenan directamente en el heap.



CÓMO AFECTAN AL CREAR STRING EN JAVA

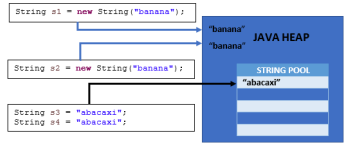
1. **Uso de memoria:**
   - Las cadenas que se crean directamente con literales (`String s1 = "Hola";`) se colocan en el pool de strings y son reutilizadas, lo que ahorra memoria.
   - Las cadenas creadas con `new` (`String s2 = new String("Hola");`) se almacenan en el heap, lo que puede llevar a un mayor uso de memoria, ya que se crean instancias separadas.
2. **Rendimiento:**
   - Comparar cadenas que están en el pool de strings usando `==` es más rápido, ya que solo se comparan las referencias, no el contenido.
   - Comparar objetos `String` que están en el heap generalmente requiere una comparación de contenido (usando el método `.equals()`), lo que es más lento.



SUBCADENAS DE CADENAS

En muchas ocasiones es necesario extraer una porción o substring de un `string` dado. Para este propósito hay una función miembro de la clase `String` denominada `substring`. Para extraer una subcadena desde una posición determinada hasta el final del `string` se usa la siguiente sintaxis:

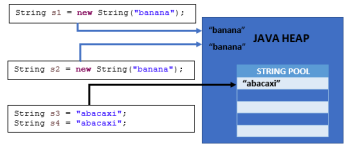
```
String str = "El lenguaje Java";
String subStr = str.substring(12);
```



SUBCADENAS DE CADENAS

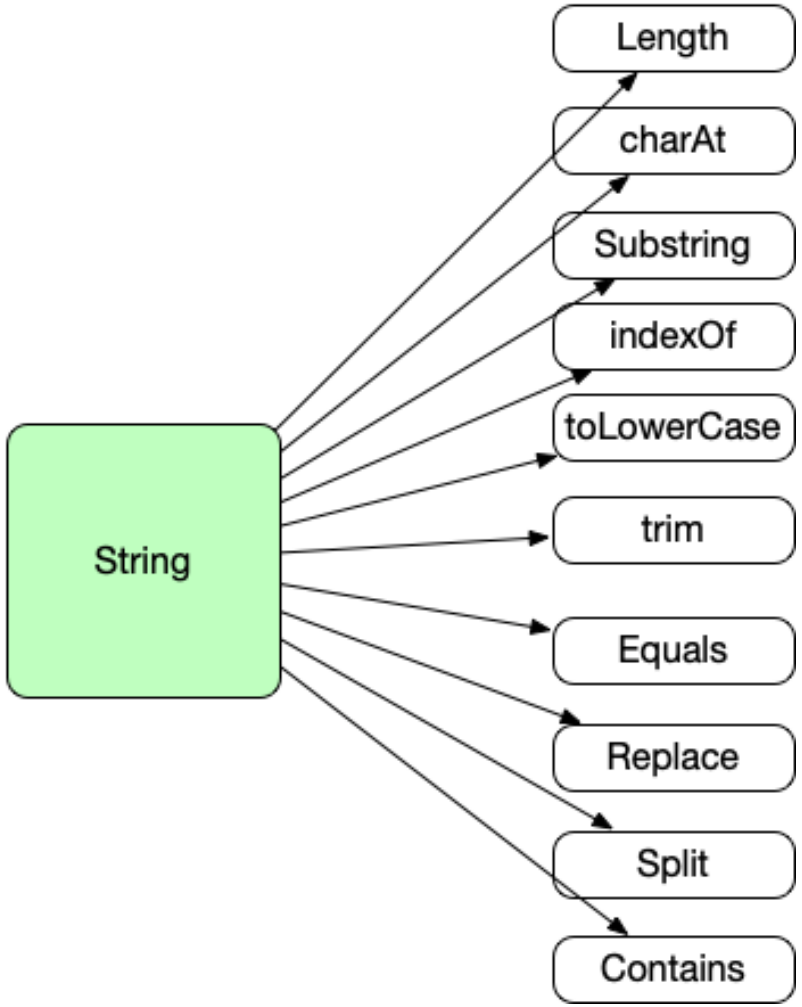
Una segunda versión de la función miembro `substring`, nos permite extraer ur substring especificando la posición de comienzo y la final.

```
String str="El lenguaje Java";
String subStr=str.substring(3, 11);
```



METODOS DE CADENAS

Realizar la actividad de los métodos de cadena de Java



CONVERSIONES

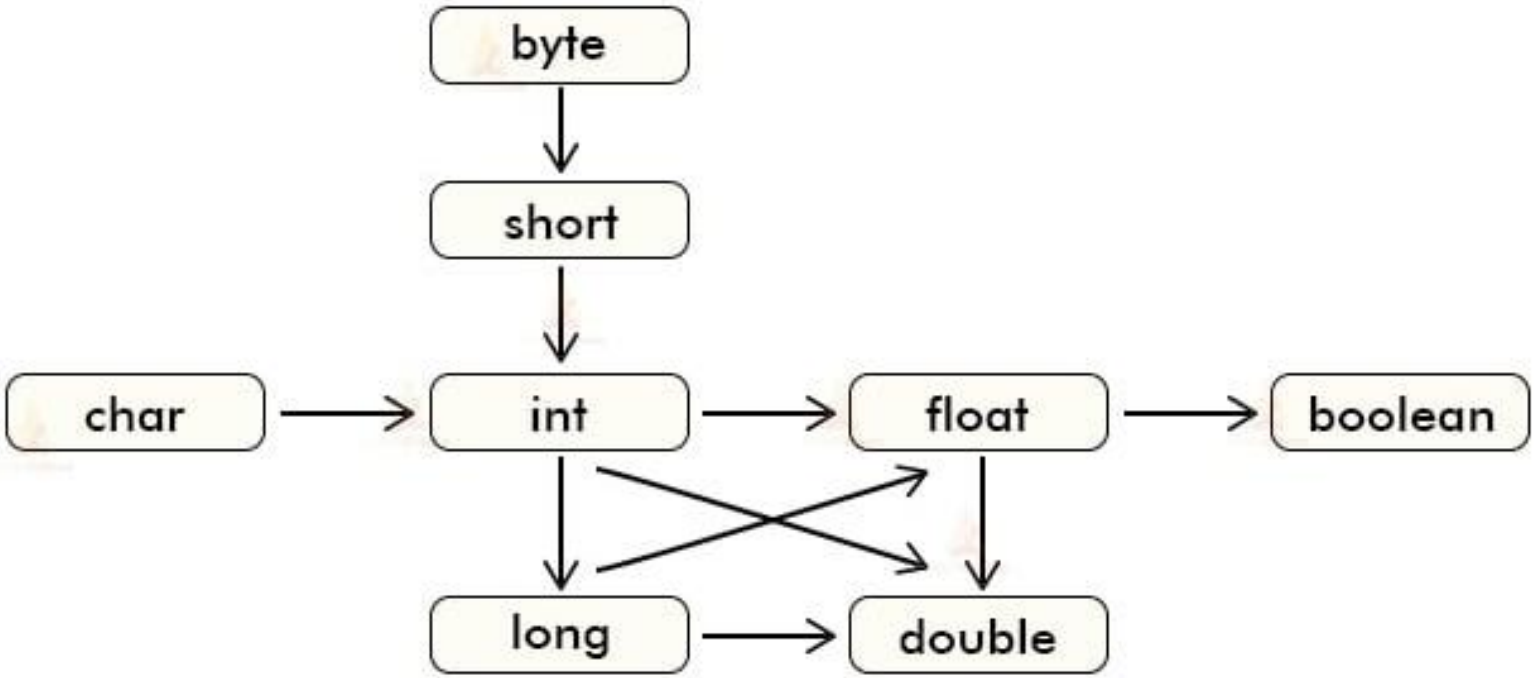
En Java, las conversiones se utilizan para convertir un valor de un tipo de dato a otro tipo de dato compatible. Hay dos tipos principales de conversiones: conversiones implícitas (o promoción) y conversiones explícitas (o casting).

CONVERSIONES IMPLÍCITAS

La conversión implícita ocurre cuando se asigna un valor de un tipo de dato más pequeño a una variable de un tipo de dato más grande sin perder información. Esto sucede de forma automática por el compilador de Java. Por ejemplo:

```
int numeroEntero = 10;
double numeroDouble = numeroEntero;
```

CONVERSIONES IMPLÍCITAS



CONVERSIONES IMPLÍCITAS

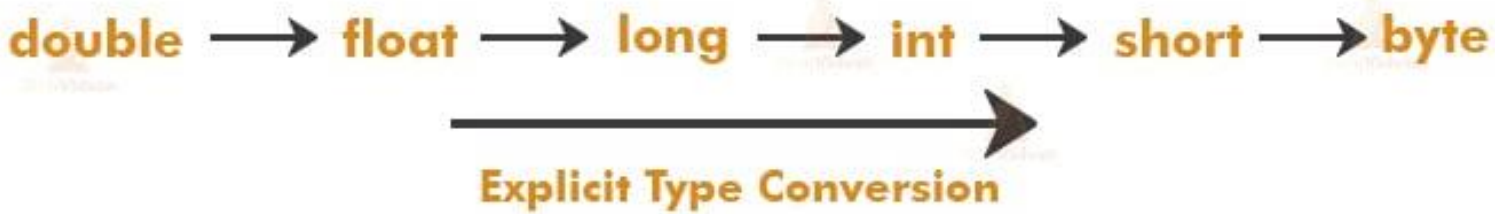
```java
int intVariable = 25;
long longVariable = intVariable;
float floatVariable = longVariable;
double doubleVariable = floatVariable;
System.out.println("Valor Entero: " + intVariable);
System.out.println("Valor Long: " + longVariable);
System.out.println("Valor Float: " + floatVariable);
System.out.println("Valor Double: " + doubleVariable);
```

CONVERSIONES EXPLÍCITAS

La conversión explícita, también conocida como casting, se utiliza cuando se necesita convertir un valor de un tipo de dato más grande a uno más pequeño o cuando se requiere convertir entre tipos de datos incompatibles. En este caso, se debe indicar la conversión de forma explícita. Por ejemplo:

```
double numeroDouble = 3.14;
int numeroEntero = (int) numeroDouble;
```

CONVERSIONES EXPLÍCITAS



CONVERSIONES EXPLÍCITAS

```
double doubleVariable = 135.78;
//explicit type casting
long longVariable = (long) doubleVariable;

//explicit type casting
int intVariable = (int) longVariable;

System.out.println("Double: " + doubleVariable);
System.out.println("Long: " + longVariable);
System.out.println("Integer: " + intVariable);
char charVariable = 'A';

//explicit type casting
int intVariable1 = (int) charVariable;
System.out.println("\nValor entero de: " + charVariable + " is " +
        intVariable1);

//explicit type casting
long longVariable1 = (long) intVariable1;
System.out.println("Long: " + longVariable1);
```

CLASES WRAPPERS (ENVOLVENTES)

Además de las conversiones entre tipos de datos primitivos, Java también proporciona clases predefinidas llamadas "wrappers" para cada tipo de dato primitivo. Estas clases son:

Wrapper	Dato primitivo
Byte	byte
Short	short
Integer	int
Long	long
Float	float
Double	double
Character	char
Boolean	boolean

Estas clases wrapper proporcionan métodos y funcionalidades adicionales para trabajar con los tipos de datos primitivos correspondientes. Además, se pueden utilizar para realizar conversiones entre tipos de datos primitivos y objetos. Ejemplo:

CLASES WRAPPERS (ENVOLVENTES)

Conversión de primitivo a objeto utilizando clases **wrapper**

```
int numeroEntero = 10;
Integer numeroObjeto = Integer.valueOf(numeroEntero);
```

Conversión de objeto a primitivo utilizando clases **wrapper**

```
Double numeroObjetoDouble = Double.valueOf(3.14);
double numeroDouble = numeroObjetoDouble.doubleValue();
```

CLASES WRAPPERS (ENVOLVENTES)

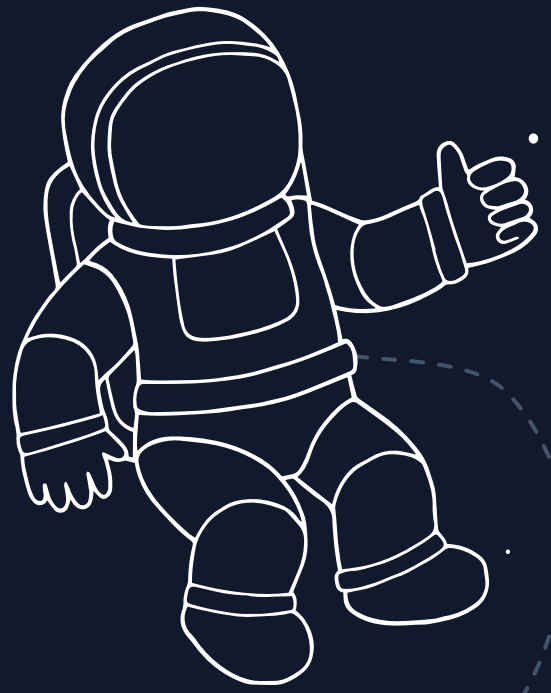
Es importante tener en cuenta que las conversiones entre tipos primitivos y sus clases **wrapper** se pueden realizar de forma automática mediante **autoboxing** (conversión de primitivo a objeto) y **unboxing** (conversión de objeto a primitivo). Ejemplo:

Autoboxing (la clase wapper recibe números primitivos)

```
int numeroEntero = 10;
Integer numeroObjeto = numeroEntero;

double numeroDouble = 3.14;
Double numeroObjetoDouble = numeroDouble;
```

Unboxing (Los valores primitivos recibe)

```
int otroNumeroEntero = numeroObjeto;
double otroNumeroDouble = numeroObjetoDouble;
```



Programa académico CAMPUS

Módulo JAVA