


# Estructuras de Datos:

## Strings y Arrays

- ▶ Unidad 6
- ▶ Apuntes referenciados:
  - ✓ A1.- Capítulo 3
  - ✓ A3.- La clase String. Los arrays



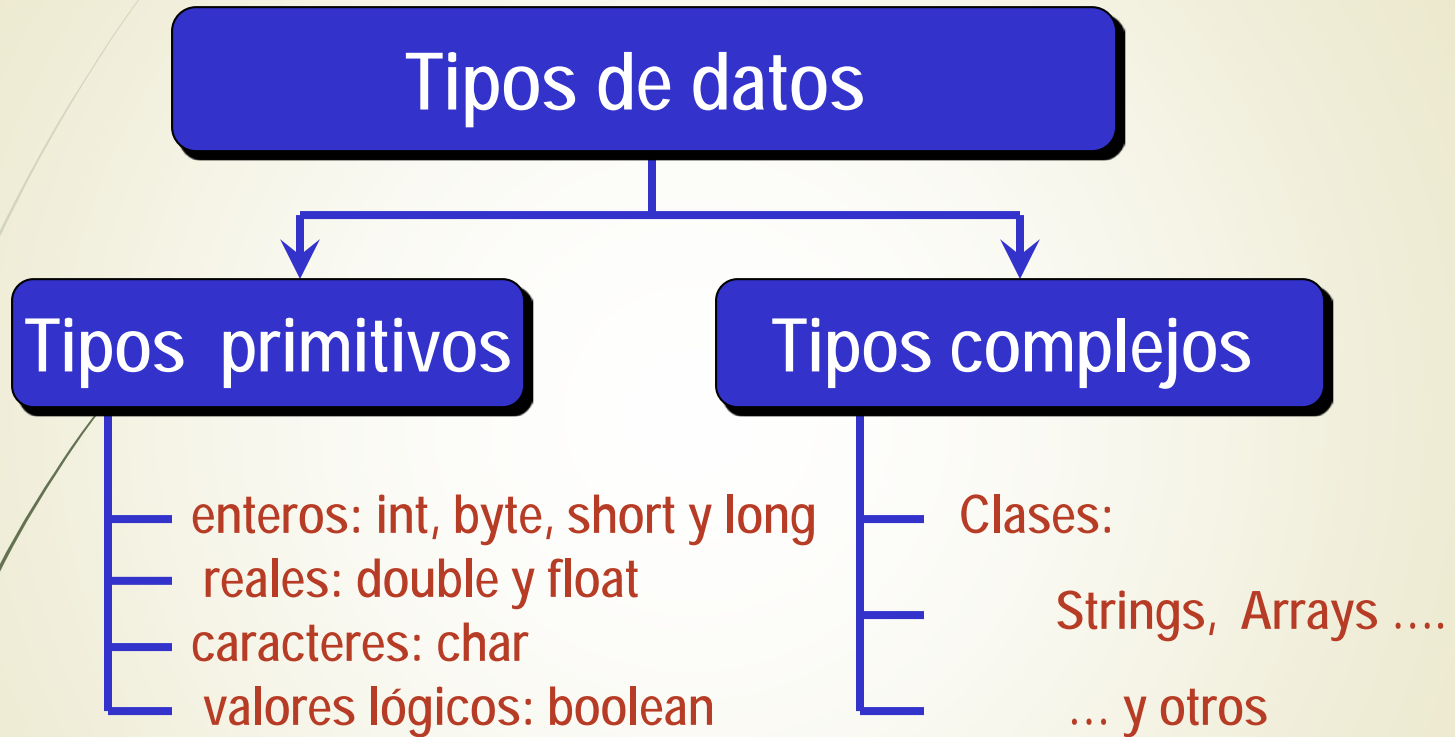
# Revisión de algunos conceptos sobre datos y variables

Tipos de datos

Paso por valor/por referencia

Ámbito de las variables

# Tipos de datos



# Tipo Primitivos Vs. Clases

- ▶ Variable de tipos primitivos (o variable valor):
  - ▶ Contiene directamente sus datos
  - ▶ Cada variable tiene su propia copia de datos, de modo que las operaciones en una variable no pueden afectar a otra variable
- ▶ Variable de clases (o variable referencia):
  - ▶ Contiene una referencia o puntero al valor de un objeto
  - ▶ Dos variables pueden referirse al mismo objeto, de modo que las operaciones en una variable pueden afectar al objeto referenciado por otra variable

# Paso de Argumentos (Repaso)

- ▶ En programación podemos pasar argumentos por valor o por referencia
  - ▶ **Por valor:** El procedimiento no puede modificar el valor de la variable original
  - ▶ **Por referencia:** El procedimiento puede modificar el valor de la variable original
- ▶ Para los argumentos de tipos básicos (primitivos) en el lenguaje Java, solamente se pasan argumentos por valor
  - ▶ Los argumentos tipo byte, short, int, long, float, double, boolean, char nunca se modifican en el programa llamante, aunque sus copias varíen en el método llamado

# Paso de Argumentos (Repaso)

Mecanismo de paso	Explicación	Implicaciones	Ventaja
Por valor	El procedimiento invocado recibe una copia de los datos cuando es invocado.	Si el procedimiento invocado modifica la copia, el valor original de la variable permanece intacto. Cuando la ejecución retorna al procedimiento de llamada, la variable contiene el mismo valor que tenía antes de que el valor se pasara.	Protege la variable de ser cambiada por el procedimiento invocado.
Por referencia	El procedimiento invocado recibe una referencia a los datos originales (la dirección de los datos en memoria) cuando es invocado.	El procedimiento invocado puede modificar la variable directamente. Cuando la ejecución retorna al procedimiento de llamada, la variable contiene el valor modificado.	El procedimiento invocado puede utilizar el argumento para devolver un nuevo valor al código de llamada.



# Paso de Argumentos. Paso por referencia

- ▶ Al método se le pasa una referencia al dato (o *una variable que apunta al dato, o la dirección de memoria del dato*)
- ▶ Aunque el método no puede alterar la referencia propiamente dicha, sí puede alterar aquello a que se refiere la referencia
- ▶ En este caso el método hace una copia de la referencia pero se comparte el objeto
- ▶ Java pasa por referencia todo aquello que no sean tipos primitivos: Strings, Arrays y Clases





# String

Cadenas de caracteres



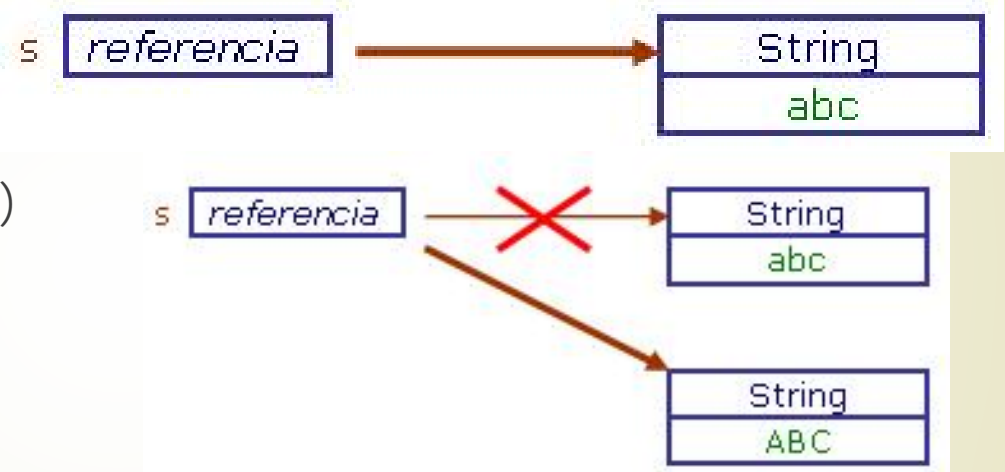


# Cadenas de caracteres o String

- ▶ Para Java una cadena de caracteres no forma parte de los tipos primitivos sino que es un objeto de la clase String (`java.lang.String`)
- ▶ Un objeto String representa una cadena de caracteres **no modificable** (o inmutable):
  - ▶ Una operación como convertir a mayúsculas no modificará el objeto original sino que devolverá un nuevo objeto con la cadena que resulte de la operación

String s="abc"

S=s.toUpperCase()



# Otro ejemplo

- Operador concatenación +
- El resultado de aplicar este operador es un nuevo String concatenación de los otros.

Por ejemplo:

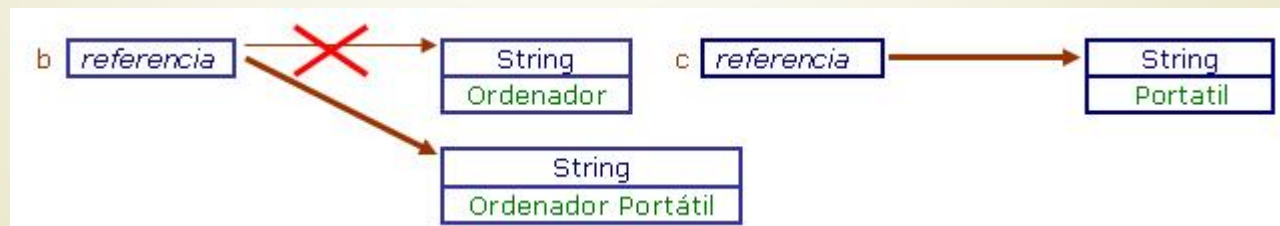
String b = "Ordenador";

String c = " Portátil";



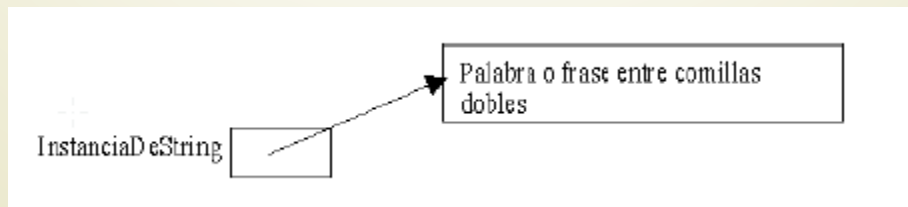
La operación:  $b = b + c$ ;

crea un nuevo String ( $b + c$ ) y le asigna su dirección a b:



# Declarar e inicializar un String

- ▶ Los objetos de la clase String se pueden crear
  - ▶ Implicítamente
    - ▶ Hay que poner una cadena de caracteres entre comillas dobles
      - ▶ Por ejemplo, al escribir `System.out.println("Hola")`
      - ▶ Java crea un objeto de la clase *String* automáticamente
  - ▶ **explícitamente**
    - ▶ `String str="Hola";` // modo tradicional
    - ▶ `String str=new String("Hola");` // modo constructor

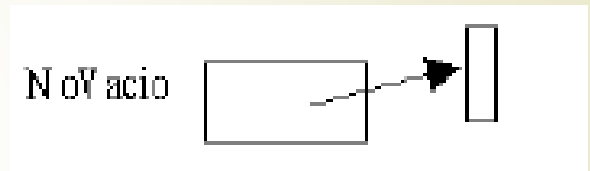


# Declarar e inicializar un String

- ▶ Para crear un String sin caracteres se puede hacer

- ▶ `String str="";` // o bien

- ▶ `String str=new String();`

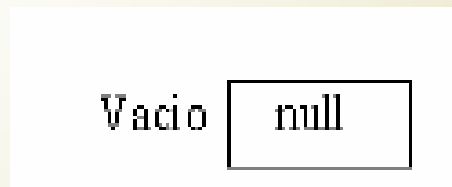


- ▶ Este es un string que no contiene caracteres pero es un objeto de la clase *String*

- ▶ Sin embargo si escribimos

- ▶ `String str;` // o bien

- ▶ `String str=null;`



- ▶ Se está declarando una variable *str* de la clase *String*, pero aún no se ha creado ningún objeto de esta clase (su valor es null)

# Algunos métodos de la clase String

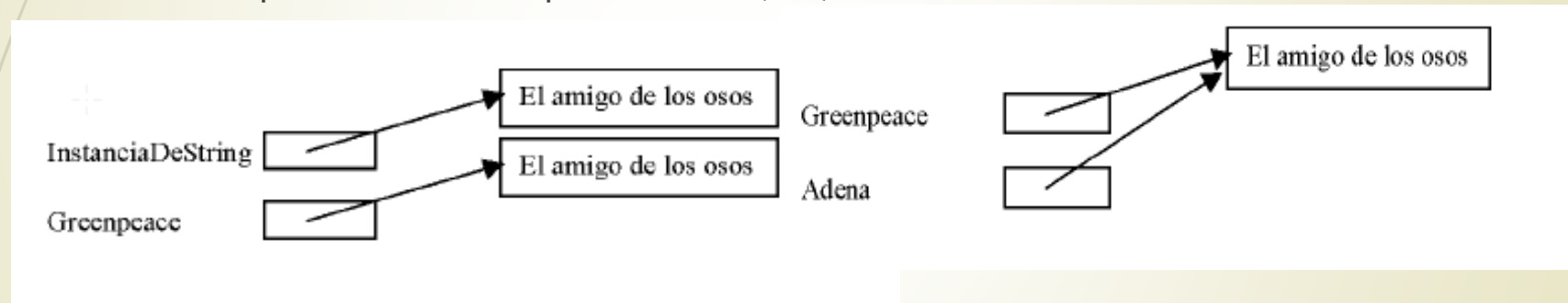
Para utilizarlos recuerda:

- ▶ Si el método es dinámico:
  - ▶ *variableString.método(argumentos)*
- ▶ Si el método es estático:
  - ▶ *String.método(argumentos)*

**La posición del primer carácter corresponde al cero (y no al uno)**

# Comparación entre objetos String

- Es importante darse cuenta de que si tenemos dos instancias del objeto String apuntando a contenidos idénticos, eso no significa que sean iguales aplicando el operador comparación (==)



- Dos Strings serán iguales (==) si apuntan a la misma estructura de datos
- Así pues, los objetos String no pueden compararse directamente con los operadores de comparación



# Comparación entre objetos String

- Disponemos de los métodos
  - **`cadena1.equals(cadena2)`**
    - El resultado es true si la cadena1 es igual a la cadena2
    - Ambas cadenas son variables de tipo String
  - **`cadena1.equalsIgnoreCase(cadena2)`**
    - Como la anterior, pero en este caso no se tienen en cuenta mayúsculas y minúsculas
  - **`cadena1.compareTo(cadena2)`**
    - Compara ambas cadenas considerando el orden alfabético (de la tabla ASCII)
    - Si la primera cadena es mayor en orden alfabético que la segunda devuelve 1, si son iguales devuelve 0 y si es la segunda la mayor devuelve -1
  - **`cadena1.compareToIgnoreCase(cadena2)`**
    - Igual que la anterior, sólo que además ignora las mayúsculas

# Algunos métodos de la clase String

## ▶ length()

- ▶ Permite devolver la longitud de una cadena (el número de caracteres de la cadena)

- ▶ `String texto1="Prueba";`

- ▶ `System.out.println(texto1.length());` // Escribe 6

- ▶ Para concatenar cadenas se puede hacer de dos formas, utilizando el método **concat()** o con el **operador +**

- ▶ `String s1="Buenos ", s2="días", s3, s4;`

- ▶ `s3 = s1 + s2;`

- ▶ `s4 = s1.concat(s2);`

# Algunos métodos de la clase String

## ▶ `charAt()`

- ▶ Devuelve un carácter de la cadena
- ▶ El carácter a devolver se indica por su posición
- ▶ Si la posición es negativa o sobrepasa el tamaño de la cadena, ocurre un error de ejecución
- ▶ `String s1="Prueba";`
- ▶ `char c1=s1.charAt(2);` // c1 valdrá 'u'

## ▶ `substring()`

- ▶ Da como resultado una porción del texto de la cadena
- ▶ La porción se toma desde una posición inicial hasta una posición final (sin incluir esa posición final)
- ▶ Si las posiciones indicadas no son válidas ocurre una excepción
- ▶ `String s1="Buenos días";`
- ▶ `String s2=s1.substring(7,10);` // s2 = día

# Métodos de la clase String

## ➤ indexOf()

- Devuelve la primera posición en la que aparece un determinado texto en la cadena
- En el caso de que la cadena buscada no se encuentre, devuelve -1
- El texto a buscar puede ser char o String
- **String s1="Quería decirte que quiero que te vayas";**
- **System.out.println(s1.indexOf("que"));**    *// Da 15*
  - Se puede buscar desde una determinada posición. En el ejemplo anterior:
    - **System.out.println(s1.indexOf("que",16));**    *// Ahora da 26*

## ➤ lastIndexOf()

- Devuelve la última posición en la que aparece un determinado texto en la cadena
- Es casi idéntica a la anterior, sólo que busca desde el final
- **String s1="Quería decirte que quiero que te vayas";**
- **System.out.println(s1.lastIndexOf("que"));**    *// Da 26*
  - También permite comenzar a buscar desde una determinada posición

# Algunos métodos de la clase String

## ➤ endsWith()

- Devuelve true si la cadena termina con un determinado texto
- `String s1="Quería decirte que quiero que te vayas";`
- `System.out.println(s1.endsWith("vayas")); //Da true`

## ➤ startsWith()

- Devuelve true si la cadena empieza con un determinado texto

# Algunos métodos de la clase String

## ➤ replace()

- Cambia todas las apariciones de un carácter por otro en el texto que se indique y lo almacena como resultado
- El texto original no se cambia, por lo que hay que asignar el resultado de replace a un String para almacenar el texto cambiado:
- `String s1="Mariposa";`
- `System.out.println(s1.replace('a','e'));` // Da Meripose
- `System.out.println(s1);` // Sigue valiendo Mariposa

## ➤ replaceAll()

- Modifica en un texto cada entrada de una cadena por otra y devuelve el resultado
- El primer parámetro es el texto que se busca (que **puede ser una expresión regular**), el segundo parámetro es el texto con el que se reemplaza el buscado. La cadena original no se modifica
- `String s1="Cazar armadillos";`
- `System.out.println(s1.replaceAll("ar","er"));` // Da Cazer ermadillos
- `System.out.println(s1);` // Sigue valiendo Cazar armadillos





# Algunos métodos de la clase String

- ▶ **toUpperCase()**

- ▶ Devuelve la versión en mayúsculas de la cadena

- ▶ **toLowerCase()**

- ▶ Devuelve la versión en minúsculas de la cadena

- ▶ **toCharArray()**

- ▶ Obtiene un array de caracteres a partir de una cadena





# Algunos métodos de la clase String

- ▶ **String.valueOf** (método static)

- ▶ Este método pertenece no sólo a la clase String, sino a otras y siempre es un método que convierte valores de una clase a otra
- ▶ En el caso de los objetos String, permite convertir valores que no son de cadena a forma de cadena
- ▶ Ejemplos:
  - ▶ `String numero = String.valueOf(1234);`
  - ▶ `String fecha = String.valueOf(new Date());`
- ▶ En el ejemplo se observa que es un método estático



# Clase StringBuilder

- ▶ La clase String representa una cadena de caracteres no modificable
  - ▶ Una operación como convertir a mayúsculas no modificará el objeto original sino que devolverá un nuevo objeto con la cadena que resulte de la operación
- ▶ La clase StringBuilder representa una cadena de caracteres modificable tanto en contenido como en tamaño
- ▶ NOTA: La clase StringBuffer tiene la misma funcionalidad (constructores y métodos) pero StringBuilder (Java SE 5.0) tiene mejor rendimiento



# Algunos métodos para StringBuilder

- ▶ Métodos **length()** y **capacity()**
  - ▶ Devuelven respectivamente la cantidad real de caracteres que contiene el objeto y la cantidad de caracteres que el objeto puede contener
- ▶ Método **append()**
  - ▶ Añadimos caracteres al final del objeto
- ▶ Métodos **delete()**, **replace()** y **insert()**
  - ▶ Modifican el objeto actual

# Ejercicio

Crea un programa que defina 2 cadenas de caracteres

- ▶ Cad1="En un lugar de la Mancha"

- ▶ Cad2="de cuyo nombre"

- ▶ Y tenga la siguiente salida:

Cad1:En un lugar de la Mancha

Cad2:de cuyo nombre

Con concat():En un lugar de la Manchade cuyo nombre

En un lugar de la Mancha de cuyo nombre

Longitud cad1:24

cad1.charAt(4): n

cad1.substring(3,9):un lug

Cad1. Primera a:9

Cad1. Primera a a partir de la posición 10:16

Cad1. Última posición en la que aparece una a:23

Cad1. Acaba con 'cha':true

Cad1. Reemplaza a y u por A:En An lAgAr de lA MAnchA

Cad1. Reemplaza un por UN:En UN lugar de la Mancha

En mayúsculas:EN UN LUGAR DE LA MANCHA