

INTRODUCCIÓN A LA PROGRAMACIÓN

**BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN**

Dra. Luz A. Sánchez Gálvez

sanchez.galvez@correo.buap.mx

luzsg@hotmail.com

JAVA

C

-
- ▶ Orientado a Objetos
 - ▶ Interpretado
 - ▶ Gestión de memoria
 - ▶ Referencias
 - ▶ Excepciones

Procedural

Compilado

Gestión de memoria (No hay)

Apuntadores (Pointers)

Errores de Código



JAVA

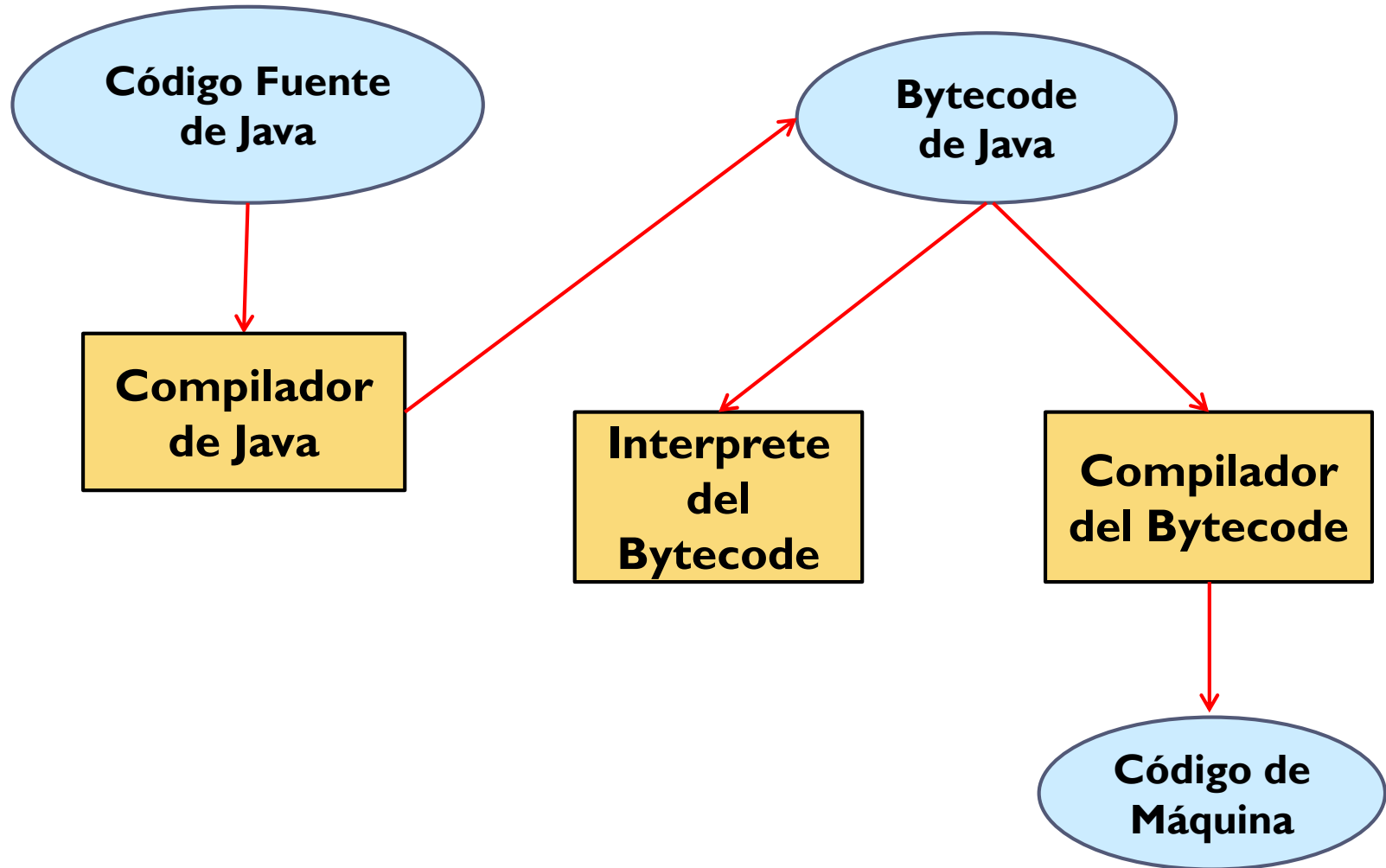
- ▶ El lenguaje de programación Java fue creado por **Sun Microsystems** por **James Gosling**. Se introdujo en 1995, y su popularidad ha crecido rápidamente.
- ▶ En 2009, Oracle adquirió Sun Microsystems
- ▶ Un lenguaje de programación especifica las palabras y los símbolos que se pueden utilizar para escribir un programa.
- ▶ Un lenguaje de programación utiliza un conjunto de reglas que determinan cómo las palabras y los símbolos se pueden poner para formar declaraciones válidas del programa



JAVA

- ▶ El compilador de Java traduce el código fuente de Java a una representación especial llamado **bytecode**.
- ▶ El **bytecode** de Java no es el lenguaje de máquina para cualquier CPU tradicional.
- ▶ Otra herramienta de software, llamado intérprete, traduce el **bytecode** a código de máquina y lo ejecuta.
- ▶ Por tanto, el compilador de Java no está vinculado a ningún equipo en particular.
- ▶ Java se considera como una arquitectura neutral

JAVA



Entornos de desarrollo JAVA

- ▶ Hay muchos programas que soportan el desarrollo de software en Java:
 - ▶ Java Development Kit (JDK)
 - ▶ Eclipse
 - ▶ NetBeans
 - ▶ BlueJ
 - ▶ jGRASP
 - ▶ JC
- ▶ Aunque los detalles de estos entornos difieren, el proceso básico de compilación y ejecución es esencialmente el mismo.



Palabras reservadas en JAVA

abstract	finally	short
assert	float	static
boolean	For	strictfp
break	goto*	super
byte	if	switch
case	implements	synchronized
catch	import	this
char	instanceof	throw
Class	int	throws
const*	interface	transient
continue	long	true
default	native	try
do	new	void
double	null	volatile
else	package	while
enum	private	
extends	protected	
false	public	
final	return	

Secuencias de escape en JAVA

Secuencias

Significado

`\b`

espacio

`\t`

tabulador

`\n`

línea nueva

`\r`

retorno de carro

`\“`

comillas

`\‘`

apostrofe

`\\`

slash



Tipos de datos primitivos

- ▶ Existen **8 tipos de datos primitivos** en **Java**
- ▶ **Cuatro** representan a los **enteros**:
 - ▶ byte, short, int, long
- ▶ **Dos** representan a los números de punto flotante:
 - ▶ float, double
- ▶ Uno representa a los caracteres:
 - ▶ char
- ▶ Uno representa a los valores booleanos:
 - ▶ boolean



Tipos de datos primitivos

- ▶ La diferencia entre los tipos primitivos numéricos es su tamaño y los valores que pueden almacenar:

Tipo	Almacena	Valor Mínimo	Valor Máximo
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
float	32 bits	$\pm 3.4 \times 10^{38}$	con 7 dígitos significantes
double	64 bits	$\pm 1.7 \times 10^{308}$	con 15 dígitos significantes

Caracteres

- ▶ Una variable de tipo **char** almacena un solo carácter
 - ▶ 'a' '7' 'X' '\$' ',' '\n'
- ▶ **Ejemplo de declaraciones :**
 - ▶ char grado = 'A';
 - ▶ espacio = ' ';
- ▶ Un objeto **String** puede contener múltiples caracteres



Estructura de un programa en JAVA

- ▶ En el lenguaje de programación JAVA:
 - ▶ Un programa se compone de una o más **clases**
 - ▶ Una **clase** contiene uno o **más métodos**
 - ▶ Un **método** contiene sentencias de programa
- ▶ Una aplicación en Java siempre contiene un método **main**



Estructura de un programa en JAVA

// Comentarios sobre la clase

```
public class MiClase
```

```
{
```

// encabezado de la clase

// cuerpo de la clase

// los comentarios pueden estar en cualquier parte

```
}
```

Java adopta la convención de que los nombres de las clases comienzan con una letra mayúscula, y el nombre del método con una letra minúscula.

Estructura de un programa en JAVA

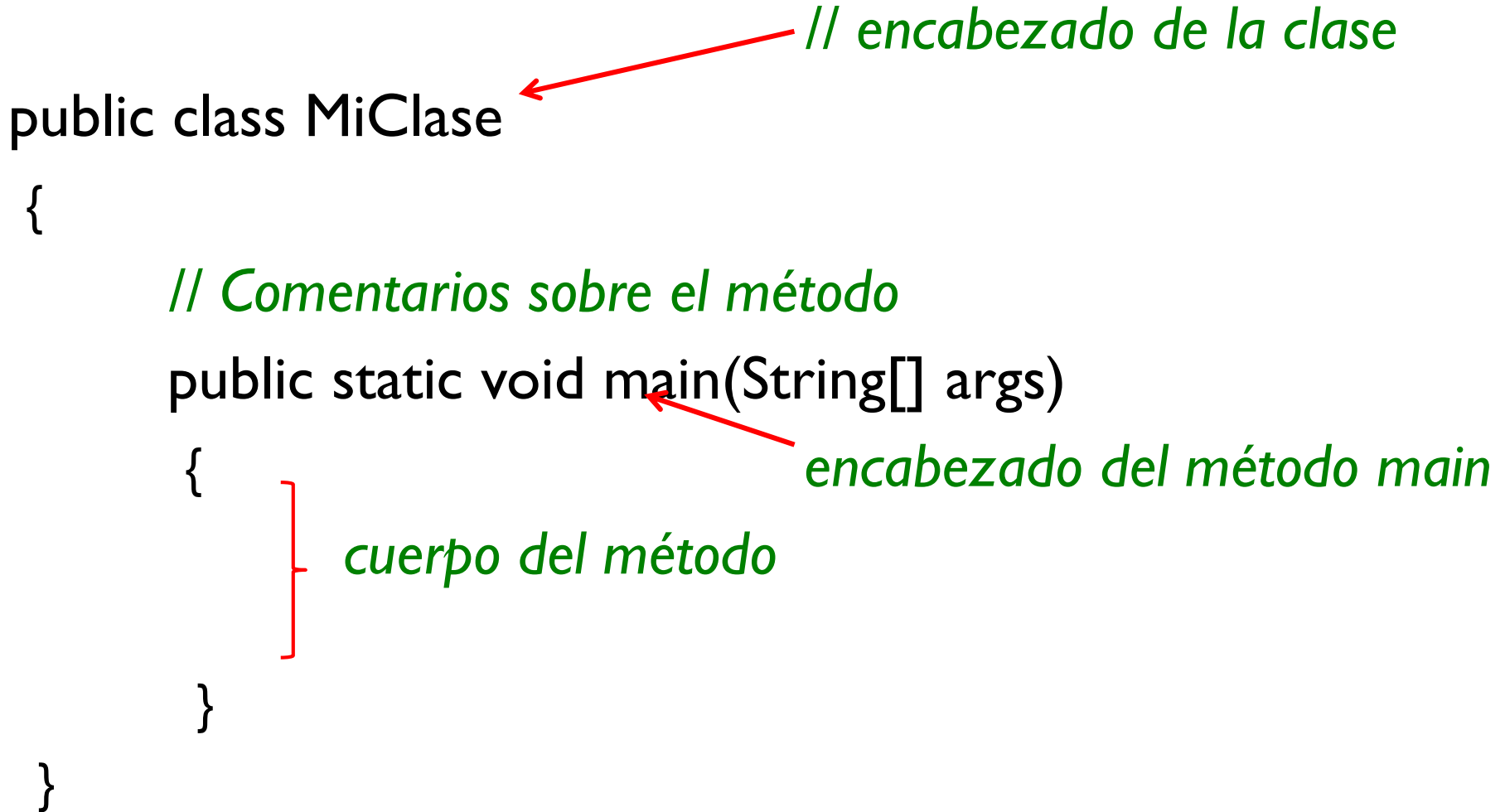
```
public class MiClase
{
    // Comentarios sobre el método
    public static void main(String[] args)
    {
    }
}
```

// encabezado de la clase

// Comentarios sobre el método

encabezado del método main

cuerpo del método



Estructura de un programa en JAVA

```
public class Mimensaje {  
    public static void main(String[] args) {  
        // objeto de invocación (denota una referencia)  
        System.out.println("JA VA ");  
    }  
}
```

// objeto de invocación (denota una referencia)

// método

parámetro que se pasa al método

- ▶ **System.out** es un objeto o *instancia* de la clase predefinida **PrintStream**. Este objeto está definido en la clase **System** y su propósito es manejar la salida del programa.
- ▶ **println (...)** es un método de la clase **PrintStream**.

Llamada a un método

▶ Sintaxis:

▶ **objeto.nombredelmétodo (parámetros)**

- ▶ **objeto** de invocación (denota una referencia)
- ▶ **nombredelmétodo** (...) invocado
- ▶ **parámetros**, que se pasan al método

▶ Semántica:

- ▶ Llamar a un método de un objeto, posiblemente, pasando parámetros.
- ▶ El efecto de una llamada a un método es la ejecución de la operación asociada al método, y a veces el retorno de un valor.



Objetos y clases en Java

- ▶ Los objetos son las entidades que pueden ser manipuladas por los programas (por lo general, invocando sus métodos).
- ▶ Cada objeto pertenece a una clase (se dice que es una instancia de la clase).
- ▶ Una clase está constituida por un conjunto de objetos (o instancias) que tienen las mismas propiedades.
- ▶ La clase a la que pertenece un objeto determina qué métodos están disponibles al objeto.



La clase **String**

- ▶ Los objetos que son instancias de la clase **String** denotan cadenas, es decir, secuencias de caracteres.
- ▶ Expresiones delimitadas por comillas: "java", denotan objetos de la clase String (son referencias predefinidas a objetos de la clase String).
 - ▶ " Una cadena de caracteres"
 - ▶ "123 es el número de la casa"
 - ▶ "Facultad de Ciencias de la Computación"
 - ▶ "X"



La clase **String**

► La clase **String** proporciona varios métodos:

<code>String</code>	<code>concat(String str)</code> Concatenates the specified string to the end of this string.
<code>int</code>	<code>length()</code> Returns the length of this string.
<code>String</code>	<code>substring(int beginIndex)</code> Returns a new string that is a substring of this string.
<code>String</code>	<code>substring(int beginIndex, int endIndex)</code> Returns a new string that is a substring of this string.
<code>char[]</code>	<code>toCharArray()</code> Converts this string to a new character array.
<code>String</code>	<code>toLowerCase()</code> Converts all of the characters in this <code>String</code> to lower case using the rules of the default locale.
<code>String</code>	<code>toLowerCase(Locale locale)</code> Converts all of the characters in this <code>String</code> to lower case using the rules of the given <code>Locale</code> .
<code>String</code>	<code>toString()</code> This object (which is already a string!) is itself returned.
<code>String</code>	<code>toUpperCase()</code> Converts all of the characters in this <code>String</code> to upper case using the rules of the default locale.
<code>String</code>	<code>toUpperCase(Locale locale)</code> Converts all of the characters in this <code>String</code> to upper case using the rules of the given <code>Locale</code> .
<code>String</code>	<code>trim()</code> Returns a copy of the string, with leading and trailing whitespace omitted.
<code>static String</code>	<code>valueOf(boolean b)</code> Returns the string representation of the <code>boolean</code> argument.
<code>static String</code>	<code>valueOf(char c)</code> Returns the string representation of the <code>char</code> argument.

La clase String

```
public class Mimensaje2 {  
    public static void main(String[] args) {  
        System.out.println("java".toUpperCase());  
        System.out.println("JAVA".toLowerCase());  
        System.out.println("java".length());  
    }  
}
```

- ▶ El método `toUpperCase ()` de la clase `String` convierte una cadena a mayúsculas.
- ▶ En este caso, `toUpperCase` es llamado sobre el objeto (denotado por la cadena) `"java"`, y devuelve una referencia a un nuevo objeto de la clase `String` que denota la cadena `"JAVA"`,
▶ que se imprime.

El distintivo de un método

- ▶ El distintivo de un método consiste del nombre del método y la descripción (tipo, número, y posición) de sus parámetros.
 - ▶ `toUpperCase ()`
 - ▶ `println (String s)`

Los nombres de los parámetros no tienen relevancia para el distintivo.



El encabezado de un método

- ▶ El encabezado de un método consiste del distintivo y la descripción (el tipo) del resultado.
 - ▶ **String toUpperCase ()**
 - ▶ **void println (String s)**
- ▶ La palabra clave **void** denota que el método no devuelve ningún valor (implementa una acción y no una función).
- ▶ Los métodos de la misma clase con el mismo nombre pero con diferentes distintivos se dice que están **sobrecargados**.
- ▶ El método `substring ()` de la clase `String`



Parámetros y resultados de un método

- ▶ Los **parámetros** de un método representan los argumentos que se pasan, y son necesarios para la operación que el método debe realizar.
- ▶ El método **println (String s)** puede ser llamado sobre el objeto `System.out`

System.out.println ("Adios").

- ▶ El parámetro **"Adios"** se utiliza en el método como la cadena a imprimir, denotado por `System.out`.
- ▶ Si el método tiene que devolver un resultado, éste se calcula y se devuelve al bloque de llamada.
- ▶ El método **String concat (String s)**



Evaluación de la llamada a un método

- ▶ Para llamar a un método se tiene que conocer el **objeto de invocación** y sus **parámetros**.
 - ▶ `"xxx".concat("yyy")`
 - ▶ objeto de invocación : `"xxx"`
 - ▶ parámetros: `"yyy"`
 - ▶ `"xxx".concat("yyy")` devuelve la cadena `"xxxyyy"`
 - ▶ En este caso, la evaluación del objeto de invocación y de los parámetros es inmediata.
 - ▶ En general, tanto el objeto de invocación como los parámetros que se pasan como argumentos, pueden necesitar ser obtenidos llamando primero a otros métodos.



Evaluación de la expresión denotando el objeto de invocación para un método

- ▶ `System.out.println("xxx".concat("yyy").concat("zzz"));`
- ▶ cómo la expresión es evaluada.
 1. Es posible evaluar `"xxx".concat("yyy")`;
 - ▶ “xxx” denota el objeto de invocación
 - ▶ “yyy” denota el parámetro
 - ▶ ambos están disponibles, se puede calcular `"xxx".concat("yyy")`, que devuelve la cadena “xxxyyy”
 2. Después continuar con `"zzz"`
 - ▶ “xxxyyy” denota el objeto de invocación
 - ▶ “zzz” denota el parámetro
 - ▶ Ambos están disponibles, se puede calcular `"xxxyyy".concat("zzz")`, que devuelve la cadena “xxxyyyzzz”



Evaluación de la expresión que denota el objeto de invocación de un método

- ▶ Por lo tanto, la declaración
- ▶ `System.out.println("xxx".concat("yyy").concat("zzz"))`
 - ▶ Imprime "xxxyyyzzz".
- ▶ La evaluación de la expresión que denota el objeto de invocación se realiza de izquierda a derecha, calculando los objetos de invocación y llamado a los métodos que necesitan ser evaluados.



Evaluación de la expresión denotando el objeto de invocación para un método

- ▶ `System.out.println("xxx".concat("yyy".concat ("zzz")));`
- ▶ cómo la expresión es evaluada.
 1. Es posible evaluar `"yyy".concat ("zzz");`
 - ▶ `"yyy"` denota el objeto de invocación
 - ▶ `"zzz"` denota el parámetro
 - ▶ ambos están disponibles, se puede calcular `"yyy".concat ("zzz")`, que devuelve la cadena `"yyyzzz"`
 2. Después continuar con `"xxx".concat(...)`
 - ▶ `"xxx"` denota el objeto de invocación
 - ▶ `"yyyzzz"` denota el parámetro calculado en 1
 - ▶ Ambos están disponibles, se puede calcular `"xxx".Concat ("yyyzzz")`, que devuelve la cadena `"xxxyyyzzz"`



Evaluación de la expresión que denota el objeto de invocación de un método

- ▶ Por lo tanto, la declaración
- ▶ `System.out.println("xxx".concat ("yyy".concat("zzz")));`
 - ▶ Imprime "xxxyyyzzz".
- ▶ La evaluación de las expresiones denotando los parámetros se realiza de dentro hacia fuera, calculando cada vez los parámetros de cada llamado antes de ser evaluado.



Métodos estáticos

- ▶ Métodos que no requieren un objeto de invocación.
- ▶ Sintaxis:
 - ▶ **Nombrede la Clase.Nombre del Metodo(parámetros)**
 - ▶ **Nombrede la Clase** es la clase a la que pertenece el método (es decir, la clase donde se define el método)
 - ▶ **Nombre del Metodo(...)** es el método llamado
 - ▶ **parámetros** que se pasan al método
- ▶ Llamar a un método estático es similar a llamar a un método no estático, excepto que no se especifica el objeto de invocación, sino sólo los parámetros. El nombre del método es precedido por el nombre de la clase a la que pertenece el método. En Java los métodos tienen un nombre que es local a la clase y por lo tanto, para identificar al método, se especifica la clase del método.

Métodos estáticos

► Ejemplo:

- `JOptionPane.showInputDialog ("Introduzca una cadena")`
 - Es la llamada del método `showInputDialog ()` definido en la clase predefinida `JOptionPane`. A este método se le pasa el parámetro " Introduzca una cadena" de tipo `String`.
 - Este método abre una ventana de diálogo que requiere de la entrada del usuario, donde muestra el mensaje "Introduzca una cadena" y un campo de entrada donde se puede escribir la cadena, que el método devolverá.
 - El método ***main()*** de una clase es un método estático que tiene como tipo ***void*** (es decir, no devuelve nada) y tiene como parámetro un vector (arreglo) de objetos de tipo cadena (*String*).
-

Variables

► Ejemplo:

```
public class JavaI {  
    public static void main(String[] args) {  
        System.out.println("java".toUpperCase());  
        System.out.println("java".toUpperCase());  
    }  
}
```

- *La expresión "java".toUpperCase() se evalúa dos veces. Para evitarlo, se podría almacenar el resultado de la evaluación de esta expresión en una variable y reusarlo para imprimirlo.*
-



Variables

```
public class Java2 {  
    public static void main(String[] args) {  
        String linea;  
        linea = "java".toUpperCase();  
        System.out.println(linea);  
        System.out.println(linea);  
    }  
}
```

- ▶ Una **variable** representa una ubicación en memoria, que se utiliza para almacenar la referencia a un objeto.
-



Declaración de variables

- ▶ Las variables en un programa son introducidas a través de su declaración.

- ▶ **Sintaxis:**

tipo NombredeVariable;

- ▶ **tipo** es el tipo de la variable
 - ▶ Para las variables de tipo de referencia a un objeto, es el nombre de la clase del cual el objeto es una instancia;
 - ▶ De lo contrario, se trata de un tipo primitivo predefinido
- ▶ **Nombredevariable** es el nombre de la variable declarada



Asignación

- ▶ **Ejemplo:**

`s = s.concat("yyy");`

- ▶ La ejecución de la sentencia equivale a:

1. evaluar la expresión del lado derecho, es decir, concatenar el valor actual de `s` (por ejemplo, "xxx") con la cadena "yyy", obteniendo como resultado "xxxyyy"
2. sustituir el valor actual de `s` ("xxx") con el valor calculado ("xxxyyy").

- ▶ Si el valor de `s` antes de la ejecución de la sentencia de asignación era "xxx", después es "xxxyyy"



Referencias a objetos

- ▶ En Java, las variables no pueden contener objetos, sino sólo las referencias a éstos.
- ▶ Los objetos se construyen y se asignan en memoria independientemente de las declaraciones de variables.
- ▶ Específicamente:
 - ▶ los objetos denotados por cadenas, ejemplo, “hola”, “ciao”, etc.) se asignan en memoria en tiempo de compilación;
 - ▶ todos los demás objetos deben ser contruidos y asignados a través de una declaración explícita.



Referencias a objetos

- ▶ Una variable cuyo tipo es una clase que contiene una referencia a un objeto de la clase (es decir, la dirección de la ubicación de memoria donde se asigna el objeto).

- ▶ **Ejemplo:**

```
String s;  
s = "xxx";
```

- ▶ La primera sentencia declara una variable **s** de tipo **String**.
- ▶ La segunda sentencia asigna a la variable la referencia al objeto indicado por "XXX".



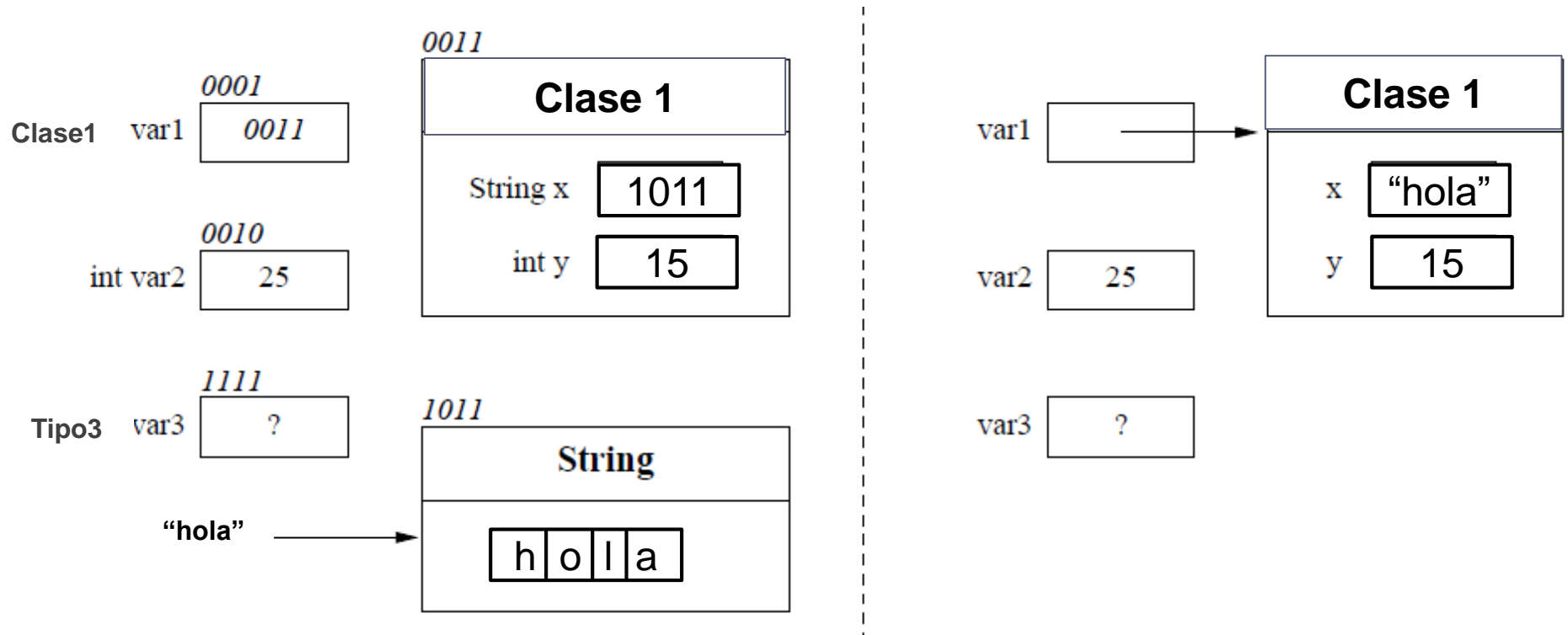
Referencias a objetos

- ▶ Dos variables pueden contener una referencia al mismo objeto.
- ▶ **Ejemplo:**
 - ▶ `String s, t;`
 - ▶ `s = "xxx";`
 - ▶ `t = s;`
- ▶ `t` y `s` contienen una referencia al objeto denotado por "xxx".
- ▶ Las variables del tipo referencia a un objeto pueden tener un valor especial, **null**,



Notación gráfica para representar las variables

- ▶ Una variable es una referencia a una ubicación de memoria en la que se almacena un objeto.
- ▶ La representación de variables y sus valores.



Notación gráfica para representar las variables

- ▶ En el diagrama se representa el nombre, tipo, dirección y valor de una variable.
 - ▶ El valor es representado por una flecha que apunta al objeto referenciado.
- ▶ Para un objeto se representa la clase de la que es una instancia, y el estado, es decir, los valores de sus atributos.
- ▶ Se muestra las direcciones de las ubicaciones en memoria, para aclarar el concepto de referencia del objeto.



Notación gráfica para representar las variables

- ▶ En Java las direcciones de los objetos nunca se representan de forma explícita.
- ▶ El tipo de una variable se omite, normalmente coincide con el tipo del objeto referenciado.

- ▶ **Ejemplo:**

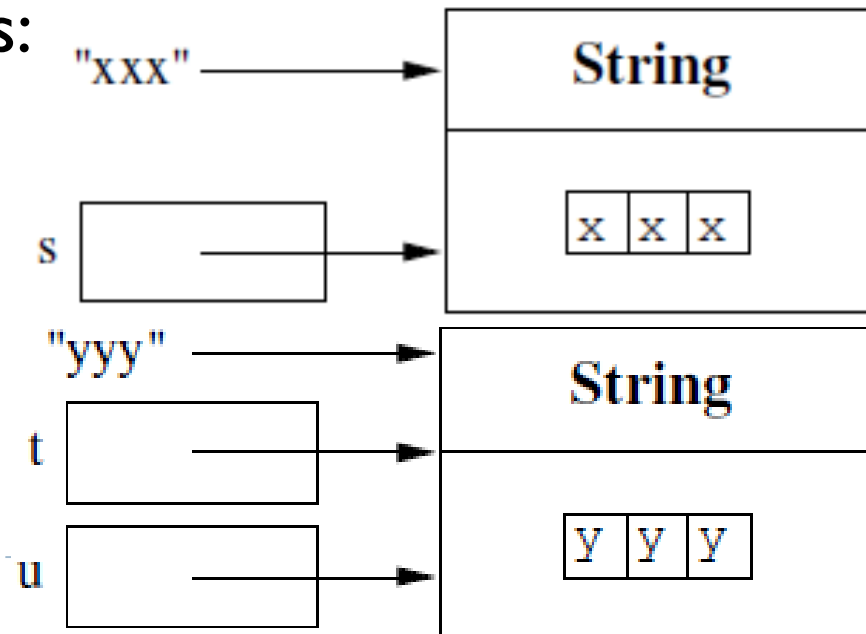
- ▶ La memoria con las sentencias:

```
String s, t, u;
```

```
s = "xxx";
```

```
t = "yyy";
```

```
u = t;
```



Ejemplo:

```
public class Computacion {  
    public static void main(String[] args) {  
        String s,t,u,v,z;  
        s = "Compu";  
        t = "tacion";  
        u = s.concat(t);  
        v = u.substring(0,5);  
        z = u.substring(5);  
        System.out.println("s = ".concat(s));  
        System.out.println("t = ".concat(t));  
        System.out.println("u = ".concat(u));  
        System.out.println("v = ".concat(v));  
        System.out.println("z = ".concat(z));  
    }  
}
```



Ejemplo: concatenación de cadenas

- ▶ La construcción de la cadena "xxxyyyzzz", usando variables y asignación:

```
String x = "xxx", y = "yyy", z = "zzz";
```

```
String temp = x.concat(y);
```

```
String result = temp.concat(z);
```

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

- ▶ Una variable se utiliza para cada objeto y para cada resultado intermedio.



Ejemplo: concatenación de cadenas

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

```
        String n = "Luz";  
        String a1 = "Sánchez";  
        String a2 = "Gálvez";  
        String iniciales;  
        String in, ia1, ia2;  
        in = n.substring(0,1);  
        ia1 = a1.substring(0,1);  
        ia2 = a2.substring(0,1);  
        iniciales = in + ia1 + ia2;  
        System.out.println(iniciales);  
    }  
}
```

```
}
```

// o simplemente

```
public class LSG {  
    public static void main(String[] args) {  
        String n = "Luz";  
        String a1 = "Sánchez";  
        String a2 = "Gálvez";  
        System.out.println(n.substring(0,1) +  
            a1.substring(0,1) +  
            a2.substring(0,1));  
    }  
}
```



Invocación de un constructor

- ▶ La creación del nuevo objeto se hace llamando a métodos especiales llamados constructores.
- ▶ **Sintaxis:**
 - ▶ `new NombredelaClase(parámetros)`
 - ▶ `new` es un operador predefinido
 - ▶ `NombredelaClase(parámetros)` es el distintivo de un método especial llamado **constructor**.
- ▶ El nombre de un **constructor coincide** con el nombre de la **clase** a la que pertenece.
- ▶ Una clase puede tener muchos constructores, que difieren en el número y/o tipos de sus parámetros (sobrecarga de constructores).



Invocación de un constructor

- ▶ **La invocación de un constructor crea un nuevo objeto de la clase, a la cual pertenece y devuelve una referencia del objeto creado.**
 - ▶ El objeto se construye utilizando los parámetros pasados al constructor.
 - ▶ **Ejemplo:**
new String("prueba")
new es el operador predefinido
String(String s) es un constructor de la clase String
 - ▶ La expresión construye un nuevo objeto de la clase String, igual a la cadena denotada por "prueba".
-



Invocación de un constructor

► Ejemplo:

```
public class Hola {  
    public static void main(String[] args) {  
        String s = new String ("hola mundo");  
        System.out.println(s);  
    }  
}
```

- la invocación del constructor **new String**("hola mundo"), crea un nuevo objeto, que es una instancia de **String** y representa la cadena "hola mundo",
- la referencia a tal objeto se asigna a la variable **s**
- el valor del objeto denotado por **s** ("hola mundo") se imprime.

Cadena vacía

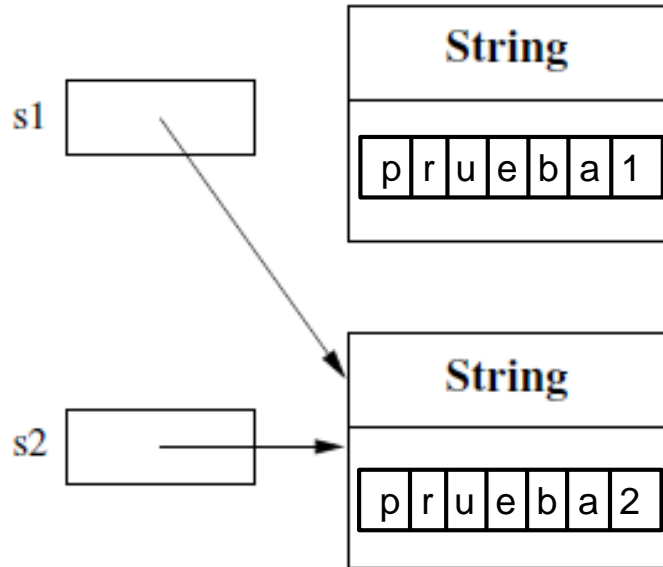
- ▶ La cadena vacía representa la secuencia de caracteres de longitud 0, y se denota por "".
- ▶ La clase String tiene un constructor sin parámetros que crea una cadena vacía:
 - ▶ `String CadenaVacía = new String();`
- ▶ El otro constructor para las cadenas toma una cadena como parámetro, por lo tanto, los dos constructores tienen diferentes distintivos. Este es un caso de sobrecarga.
- ▶ ***Nota:*** no confundir la cadena vacía con el valor null.



Accesibilidad de los objetos

- ▶ En las siguientes sentencias:
 - ▶ `String s1 = new String ("Prueba1");`
 - ▶ `String s2 = new String ("Prueba2");`
 - ▶ `s1 = s2;`
- ▶ Las referencias a **s1** y **s2** son inicialmente dos referencias a dos objetos recién creados.
- ▶ La instrucción de asignación establece la referencia de **s1** igual a la referencia de **s2** (dos referencias al mismo objeto "Prueba2"), mientras que la referencia al objeto "Prueba1", creado por
- ▶ la primera instrucción se pierde.

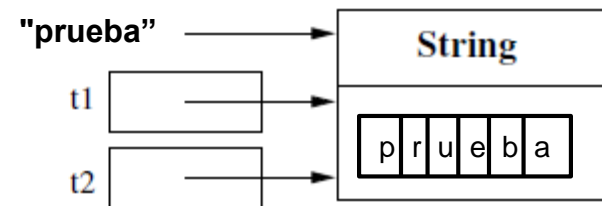
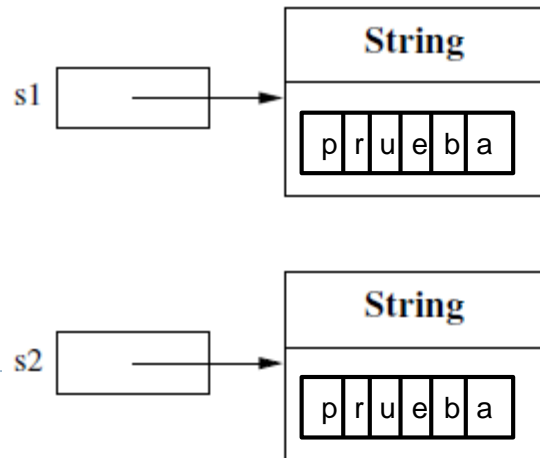
Accesibilidad de los objetos



- ▶ La operación de recuperación de la memoria utilizada por los objetos que han sido "perdidos" por el programa se llama **recolección de basura**.
- ▶ En Java, esta operación se realiza automáticamente por el sistema en tiempo de ejecución (**Máquina Virtual de Java**).

Referencias a objetos

- ▶ El operador `new` crea nuevas instancias de objetos.
- ▶ **Ejemplo:**
 - ▶ `String s1 = new String ("prueba");`
 - ▶ `String s2 = new String ("prueba");`
 - ▶ `String t1 = "prueba";`
 - ▶ `String t2 = "prueba";`
- ▶ Las referencias `s1` y `s2` son referencias a objetos diferentes, mientras que `t1` y `t2` son referencias al mismo objeto.



Objetos inmutables

- ▶ Los objetos de tipo String son objetos inmutables, porque no hay métodos para cambiar su estado, es decir, la cadena que representan.
- ▶ Los objetos que no pueden cambiar su estado se llaman inmutables. Estos representan exactamente la misma información durante todo su ciclo de vida.



Objetos inmutables

```
public class MayusculaMinuscula {  
    public static void main(String[] args) {  
        String s, mayuscula, minuscula;  
        s = new String("Hola");  
        mayuscula = s.toUpperCase();  
        minuscula = s.toLowerCase();  
        System.out.println(s);  
        System.out.print("mayuscula = ");  
        System.out.println(mayuscula);  
        System.out.print("minuscula = ");  
        System.out.println(minuscula);  
    }  
}
```

- ▶ El programa construye tres cadenas diferentes (que no se modifican):
 - ▶ la cadena "Hola", llamando al constructor,
 - ▶ la cadena "HOLA", denotada por la variable mayúscula, y
 - ▶ la cadena "hola", denotada por la variable minúscula.

Objetos mutables:

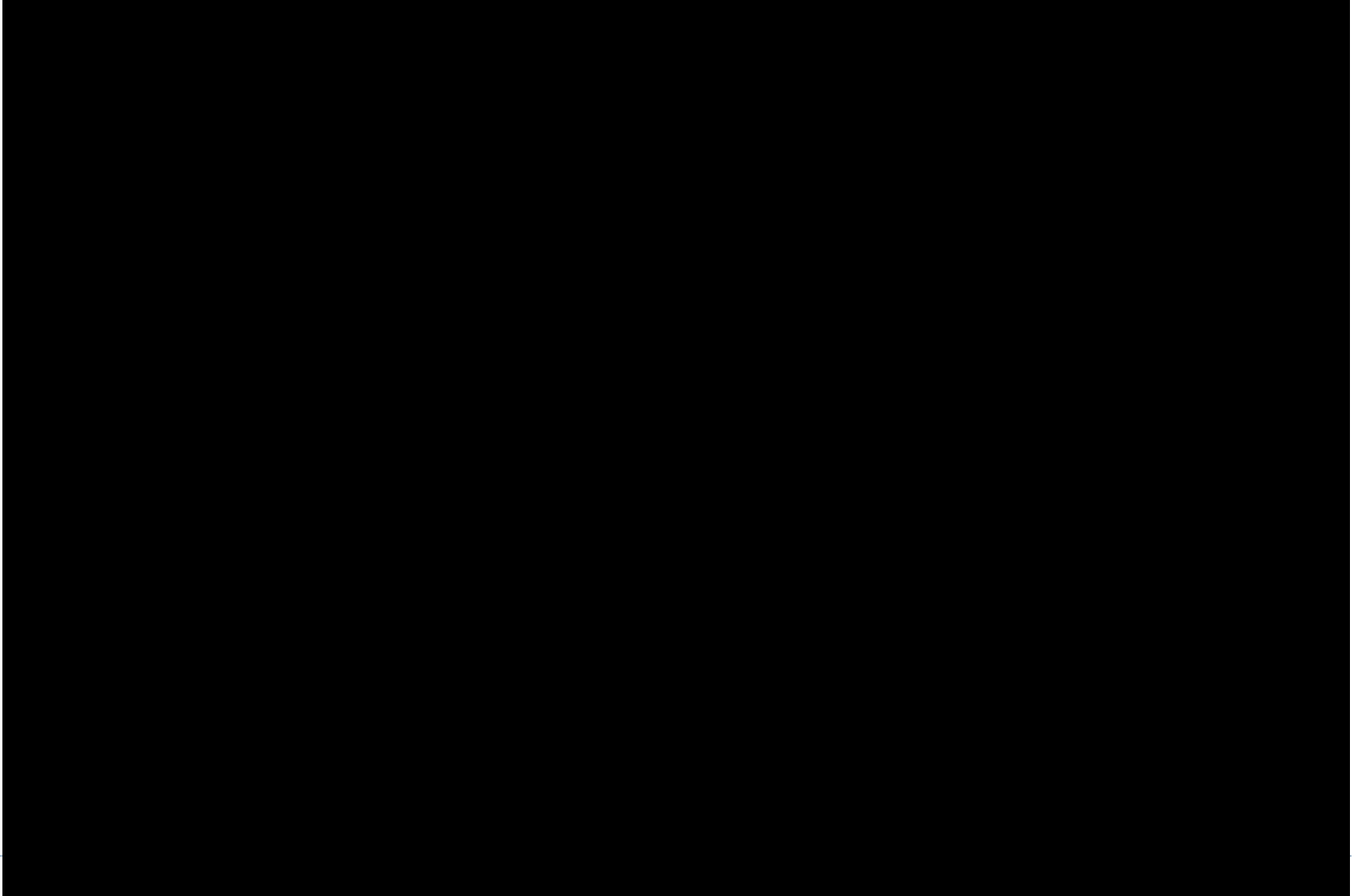
La clase StringBuffer

- ▶ Java tiene una clase muy similar a String, cuyas instancias son **objetos mutables**: la clase **StringBuffer**.
- ▶ Tiene métodos para modificar la cadena representada por un objeto.

StringBuffer	append(char c) Appends the string representation of the char argument to this sequence.
StringBuffer	append(char[] str) Appends the string representation of the char array argument to this sequence.
StringBuffer	append(char[] str, int offset, int len) Appends the string representation of a subarray of the char array argument to this sequence.
StringBuffer	append(CharSequence s) Appends the specified CharSequence to this sequence.
StringBuffer	append(CharSequence s, int start, int end) Appends a subsequence of the specified CharSequence to this sequence.
StringBuffer	append(double d) Appends the string representation of the double argument to this sequence.
StringBuffer	append(float f) Appends the string representation of the float argument to this sequence.
StringBuffer	append(int i) Appends the string representation of the int argument to this sequence.
StringBuffer	append(long lng) Appends the string representation of the long argument to this sequence.
StringBuffer	append(Object obj) Appends the string representation of the Object argument.
StringBuffer	append(String str) Appends the specified string to this character sequence.

Objetos mutables:

La clase StringBuffer



Objetos mutables:

Métodos con efectos secundarios

- ▶ Los objetos mutables permiten modificar su estado. Las modificaciones se denominan **efectos secundarios**.
- ▶ Los métodos que realizan tales modificaciones son denominados métodos con efecto secundario.

```
public class EfectoSecundarioI {  
    public static void main (String[] args) {  
        StringBuffer s = new StringBuffer("prueba");  
        StringBuffer t;  
        t = s;  
        s.append("!");  
        System.out.println(s.toString());  
        System.out.println(t.toString());  
    }  
}
```

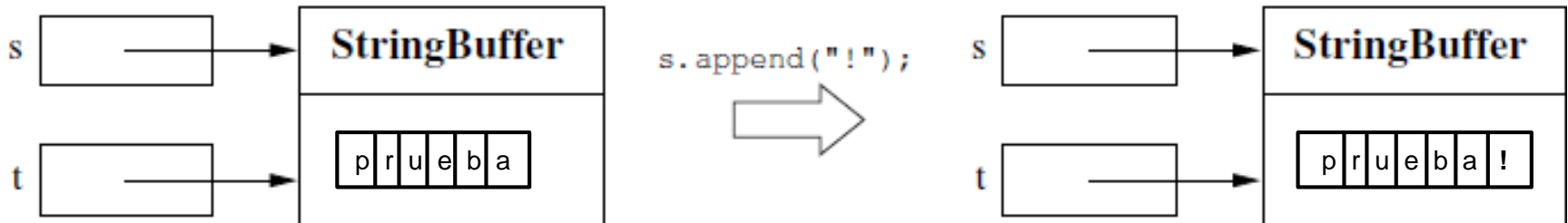


Objetos mutables:

Métodos con efectos secundarios

► Consideraciones:

- El programa imprime "prueba" dos veces.
- `append()` tiene un efecto secundario en el objeto de invocación. El objeto referenciado por `s` es modificado por el método `append()`.
- Después de la sentencia `t = s`; `s` y `t` hacen referencia al mismo objeto, también `t` denota un objeto que representa "prueba"
- El método `append()` devuelve una referencia al objeto de invocación modificado, aunque, en la sentencia `s.append("!");` no se utiliza esa referencia.



La clase StringBuffer

```
public class EfectoSecundario2 {  
    public static void main (String[] args) {  
        String s = "nombre apellido";  
        StringBuffer sbuf = new StringBuffer(s);  
        sbuf.replace(0,1,s.substring(0,1).toUpperCase());  
        sbuf.replace(7,8,s.substring(7,8).toUpperCase());  
        System.out.println(sbuf.toString());  
    }  
}
```



Entrada desde el teclado

- ▶ En Java, hay muchas maneras de leer cadenas.
- ▶ La más simple, con la clase **Scanner** de la biblioteca `java.util`.
- ▶ Un objeto se crea para leer desde el canal de entrada `System.in` :
 - ▶ `Scanner scanner = new Scanner (System.in);`
- ▶ El método `nextLine()` de la clase `Scanner` obtiene la siguiente línea de entrada (una secuencia de caracteres delimitados por un carácter de nueva línea):



Entrada desde el teclado

```
import java.util.Scanner;
public class Entradadeteclado {
    public static void main (String[] args) {
        ...
        Scanner scanner = new Scanner(System.in);
        String entracadena = scanner.nextLine();
        ...
        System.out.println(entracadena);
        ...
    }
}
```



Entrada desde el teclado

- ▶ `import java.util.Scanner;`
importa la clase `java.util.Scanner` de la biblioteca `java.util`.
 - ▶ `Scanner scanner = new Scanner(System.in);`
crea un objeto nuevo `scanner`, que se conecta a la entrada del teclado.
 - ▶ `String entracadena = scanner.nextLine();`
 1. temporalmente se suspende la ejecución del programa, esperando una entrada desde el teclado;
 2. la entrada es aceptada cuando se digita un retorno de carro;
 3. una referencia a la cadena introducida es devuelta por el método `nextLine ()`;
 4. la referencia a la cadena se asigna a la variable `entracadena`.
-



Ejemplo: las iniciales de un nombre

```
import java.util.Scanner;

public class Iniciales {

    public static void main (String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.println("Introduzca nombre");
        String n = scanner.nextLine();
        System.out.println("Introduzca apellido");
        String a = scanner.nextLine();
        String in = n.substring(0,1).toUpperCase();
        String ia = a.substring(0,1).toUpperCase();
        System.out.println("Nombre: " + n + " " + a);
        System.out.println("Iniciales: " + in + ia);

    }

}
```



Entrada desde una ventana de diálogo

- ▶ Una alternativa para leer cadenas es utilizar el método `showInputDialog()`, definido en la clase `JOptionPane`, que es parte de la biblioteca **swing**. Este método permite leer desde el teclado:

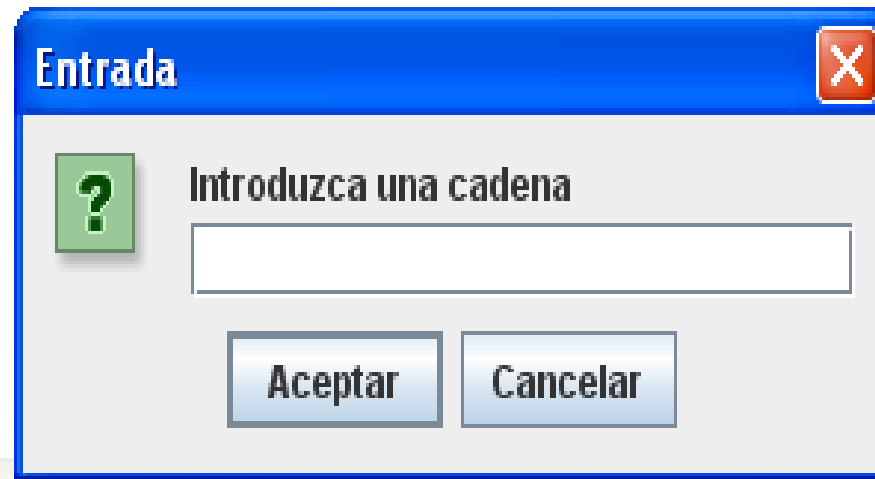
```
import javax.swing.JOptionPane;
public class Entradadeteclado {
    public static void main (String[] args) {
        ...
        String entracadena = JOptionPane.showInputDialog("Introduzca
        una cadena");
        ...
        System.out.println(entracadena);
        ...
        System.exit(0);
    }
}
```

Entrada desde una ventana de diálogo

- ▶ `import javax.swing.JOptionPane;`
importa la clase `JOptionPane` de la biblioteca `javax.swing`
 - ▶ `String entracadena=JOptionPane.showInputDialog ("Introduzca una cadena");`
 1. crea una ventana de diálogo que muestra el mensaje "Introduzca una cadena",
 2. lee una cadena desde el teclado,
 3. devuelve una referencia a una cadena, y
 4. asigna la referencia a la variable `entracadena`.
 - ▶ `System.exit(0);` debe añadirse al método `main()` cuando se utiliza `JOptionPane`, ya que la ventana de diálogo no es manejada directamente por `main()`, se debe proporcionar un comando para terminar.
-



Entrada desde una ventana de diálogo



Ejemplo: iniciales de un nombre utilizando una ventana de diálogo

```
import javax.swing.JOptionPane;
```

```
public class Iniciales {
```

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

```
        String n = JOptionPane.showInputDialog("Introduzca nombre");
```

```
        String a = JOptionPane.showInputDialog("Introduzca apellido");
```

```
        String in = n.substring(0,1).toUpperCase();
```

```
        String ia = a.substring(0,1).toUpperCase();
```

```
        System.out.println("Nombre: " + n + " " + a);
```

```
        System.out.println("Iniciales: " + in + ia);
```

```
        System.exit(0);
```

```
    }
```

```
}
```



Salida a una ventana

- ▶ La clase JOptionPane, envía la salida a una ventana de diálogo.
- ▶ El método showMessageDialog()

```
import javax.swing.JOptionPane;
```

```
public class SalidaVentana {  
    public static void main(String[] args) {  
        String nombre = JOptionPane.showInputDialog("¿Cuál es tu  
nombre?");  
        nombre = nombre.toUpperCase();  
        String cadenaamostar = "Hola" + " " + nombre + ", ¿Cómo  
estas?";  
        JOptionPane.showMessageDialog(null, cadenaamostar);  
        System.exit(0);  
    }  
}
```



Salida a una ventana

- ▶ `JOptionPane.showMessageDialog(null,stringToShow)`
 - ▶ crea una ventana de diálogo que muestra la cadena denotada por la variable `cadenaamostar`
 - ▶ el valor **null** para el primer parámetro indica que la ventana creada, no es una sub-ventana de una ventana existente.

