6

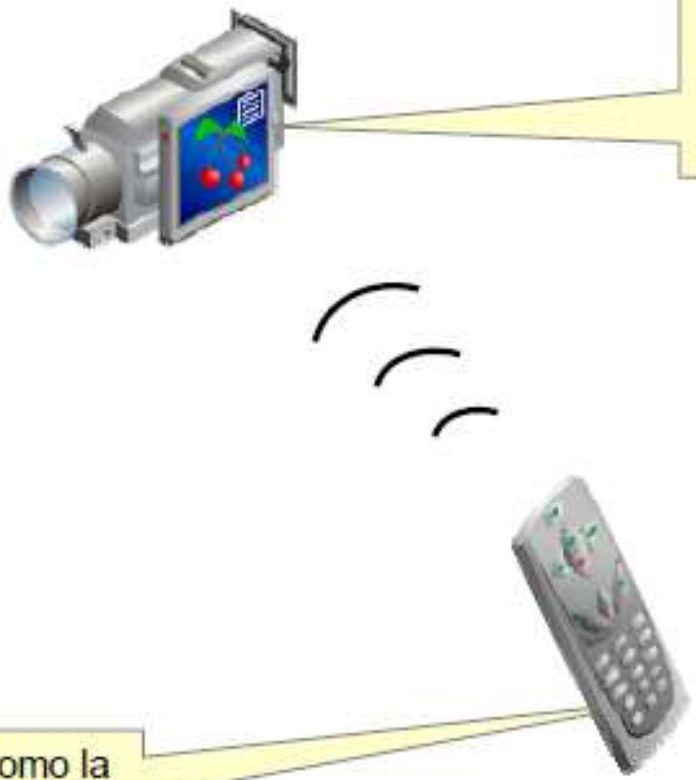
Trabajar con objetos

Trabajar con objetos: Introducción

Se accede a los objetos a través de referencias.

- Los objetos son versiones instanciadas de su clase.
- Los objetos constan de atributos y operaciones:
  - En Java, son campos y métodos.

Acceso a objetos mediante una referencia



Clase `Shirt`

```
public class Shirt {
  public int shirtID = 0; // Default ID for the shirt
  public String description =
        "-description required-"; // default
  // The color codes are R=Red, B=Blue, G=Green, U=Unset
  public char colorCode = 'U';
  public double price = 0.0; // Default price all items
  // This method displays the details for an item
  public void display() {
    System.out.println("Item ID: " + shirtID);
    System.out.println("Item description:" +
   description);
    System.out.println("Color Code: " + colorCode);
    System.out.println("Item price: " + price);
  } // end of display method
} // end of class
```

Trabajar con variables de referencia de objetos

Declaración:

```
Classname identifier;
```

Instanciación:

```
new Classname();
```

Este fragmento de código crea el objeto.

Asignación:

```
Object reference = new Classname();
```

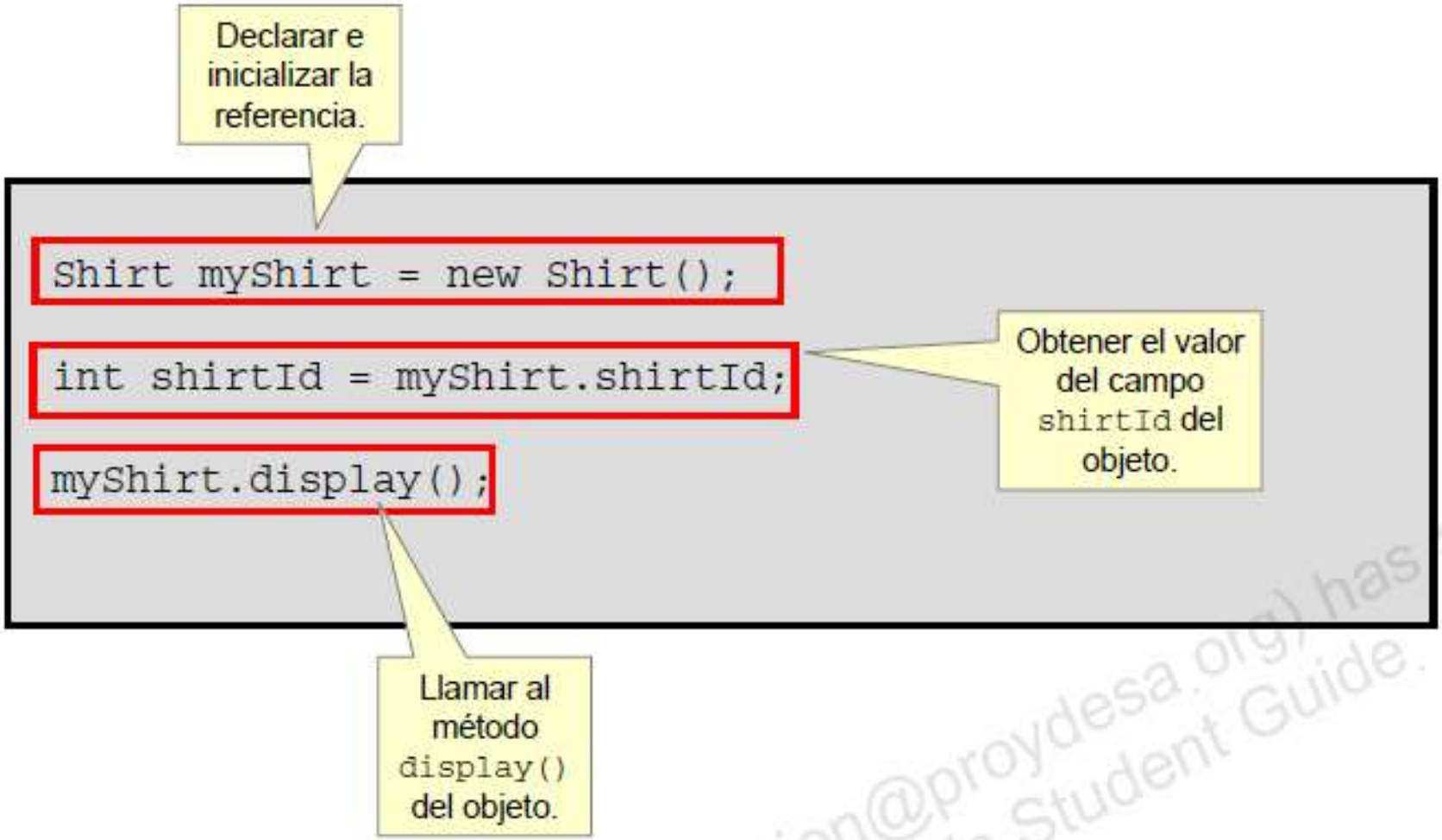
Identificador del paso de declaración

Operador de asignación

Para realizar la asignación a una referencia, la creación y la asignación deben estar en la misma sentencia.

Declaración e inicialización: Ejemplo

Trabajar con referencias de objetos



Trabajar con referencias de objetos

1. Coger el control remoto para obtener acceso a la cámara.
2. Pulsar los controles remotos para que la cámara haga algo.

1. Crear un objeto Shirt y obtener una referencia a él.

```
Shirt myShirt = new Shirt();
myShirt.display();
```

2. Llamar a un método para que el objeto Shirt haga algo.

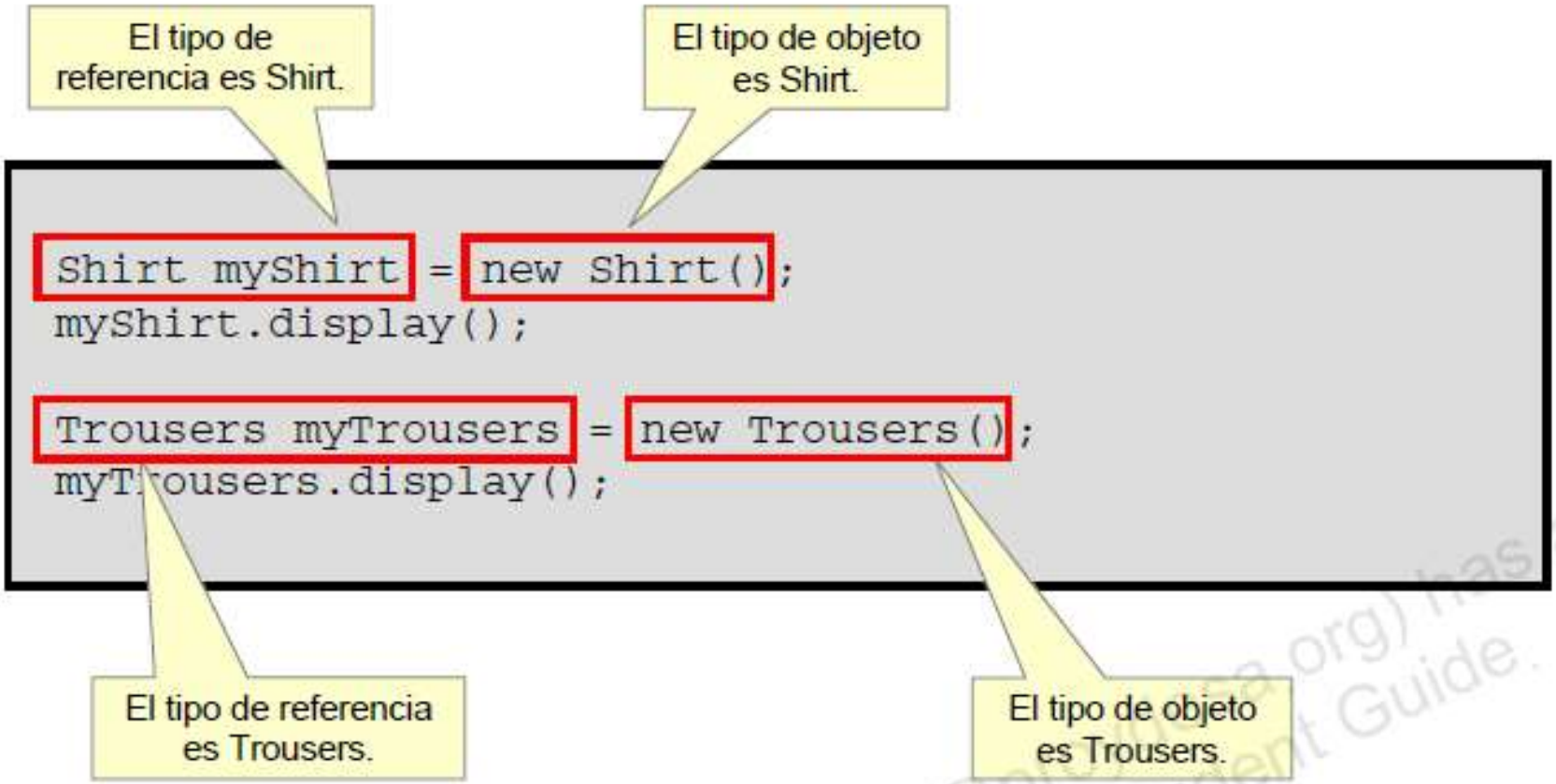
Trabajar con referencias de objetos

Solo hay un objeto Camera.

```
Camera remote1 = new Camera();

Camera remote2 = remote1;

remote1.play();

remote2.stop();
```

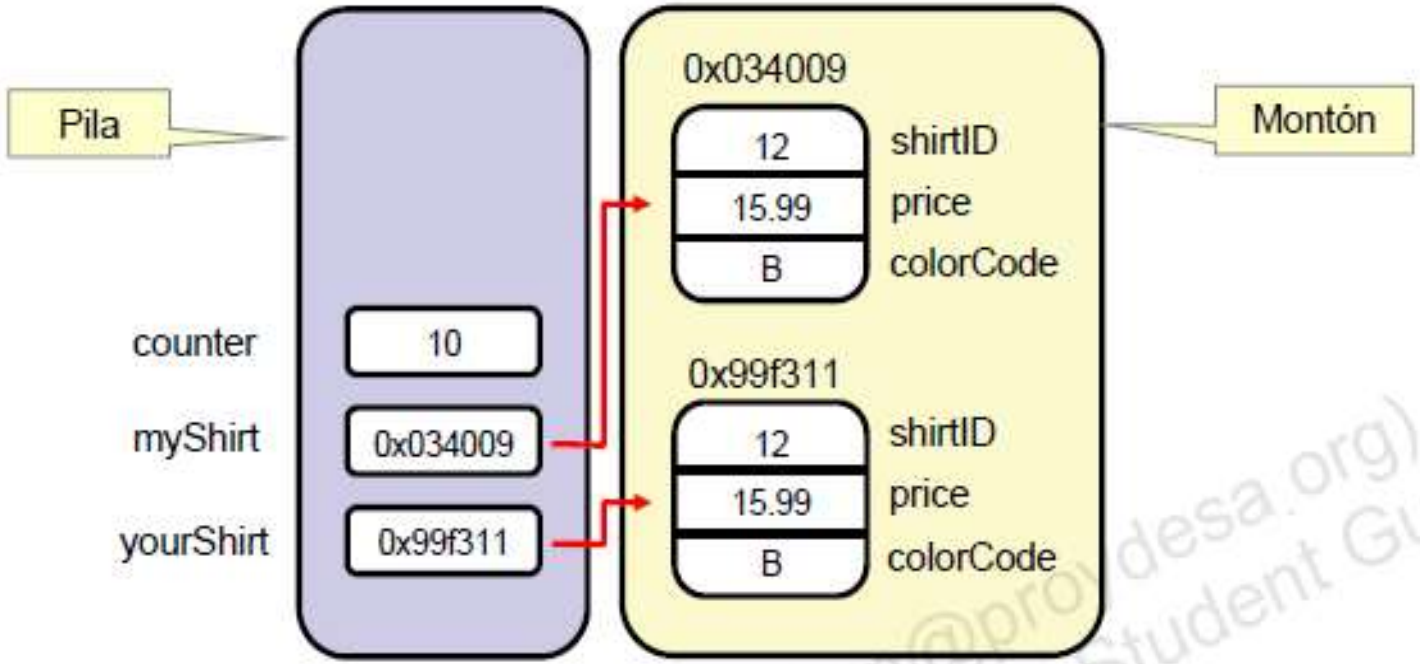
Referencias a diferentes objetos

Referencias a diferentes tipos de objetos



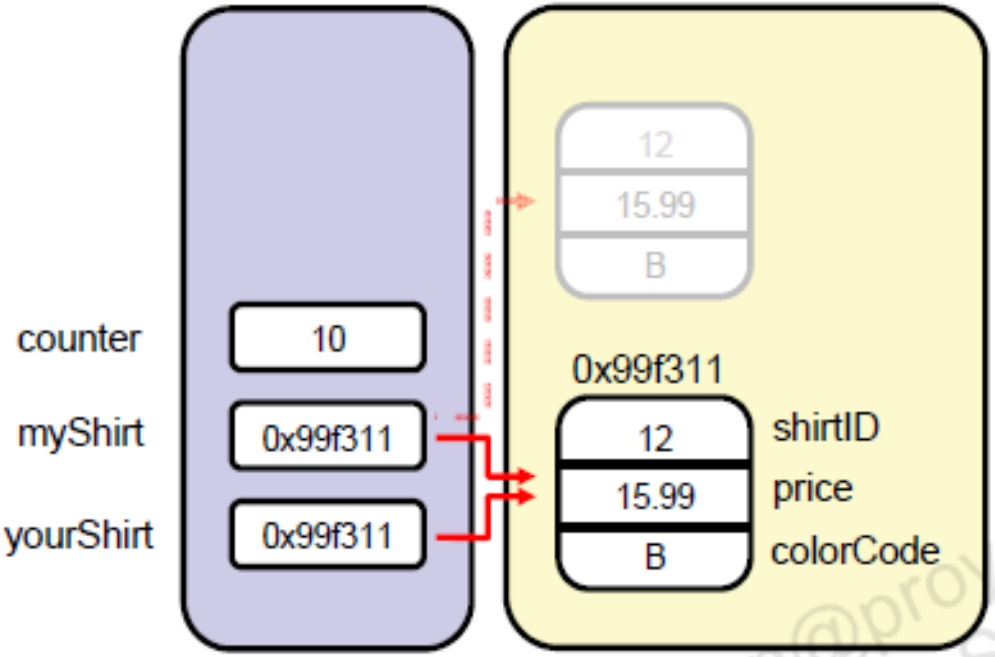
Referencias y objetos en memoria

```
int counter = 10;
Shirt myShirt = new Shirt();
Shirt yourShirt = new Shirt();
```



Asignación de una referencia a otra

```
myShirt = yourShirt;
```



Dos referencias, un objeto

Fragmento de código:

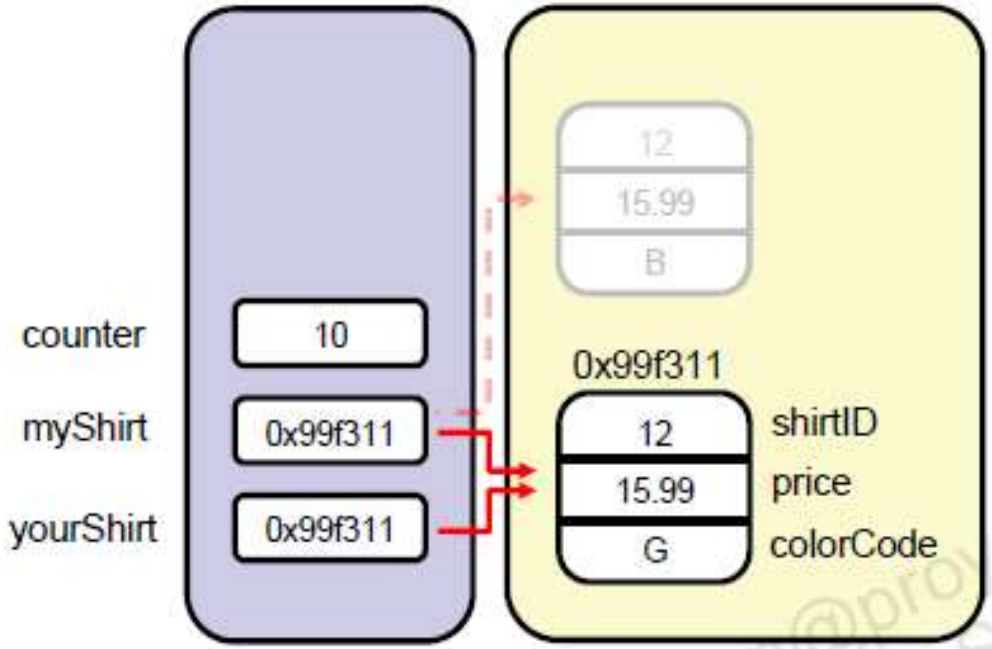
```
Shirt myShirt = new Shirt();
Shirt yourShirt = new Shirt();

myShirt = yourShirt;

myShirt.colorCode = 'R';
yourShirt.colorCode = 'G';

System.out.println("Shirt color: " + myShirt.colorCode);
```

Salida del fragmento de código:

```
 Shirt color: G
```

Asignación de una referencia a otra

```
myShirt.colorCode = 'R';
yourShirt.colorCode = 'G';
```



Clase String

La clase String soporta alguna sintaxis no estándar.

- Se puede instanciar un objeto String sin utilizar la palabra clave `new`; se prefiere esto:

```
String hisName = "Fred Smith";
```

  - Se puede utilizar la palabra clave `new`, pero *no* se recomienda:

```
String herName = new String("Anne Smith");
```

- Un objeto String es inmutable; su valor no se puede cambiar.
- Un objeto String se puede utilizar con el símbolo del operador de concatenación de cadenas (+) para la concatenación.

Concatenación de cadenas

Cuando utiliza un literal de cadena en el código Java, se instancia y se convierte en una referencia String.

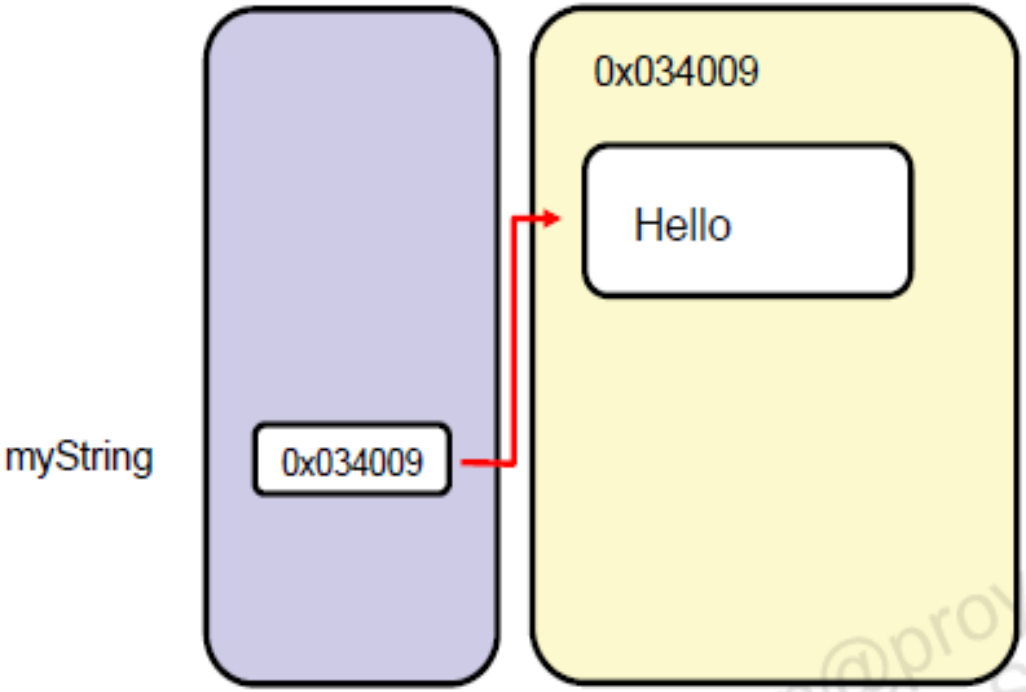
- Concatenar cadenas:

```
String name1 = "Fred"
theirNames = name1 + " and " +
             "Anne Smith";
```

- La concatenación crea una nueva cadena y la referencia String `theirNames` apunta ahora a esta nueva cadena.

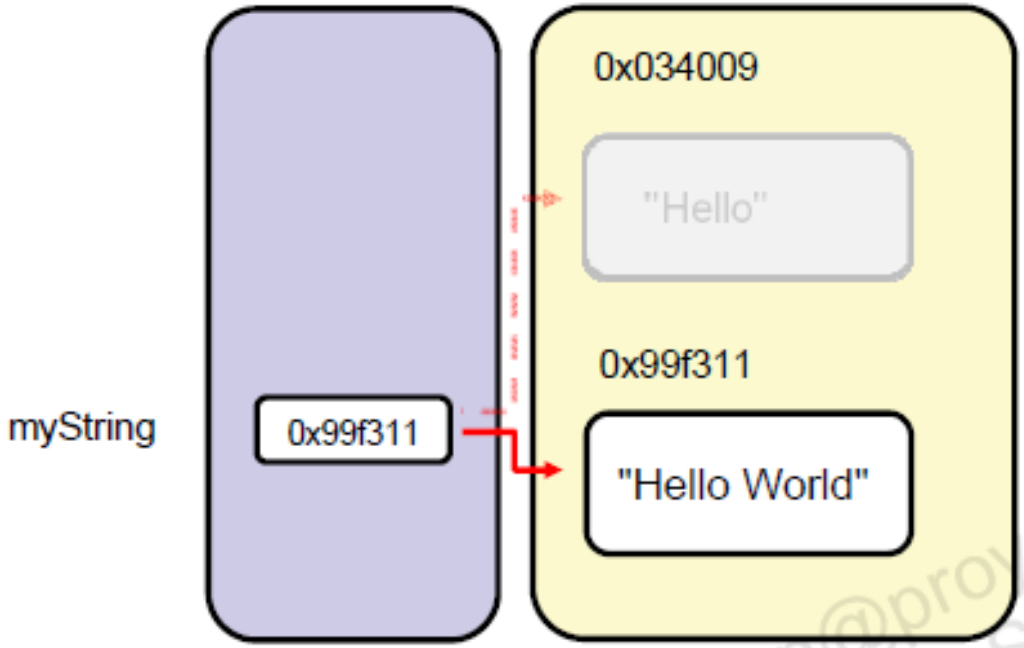
Concatenación de cadenas

```
String myString = "Hello";
```



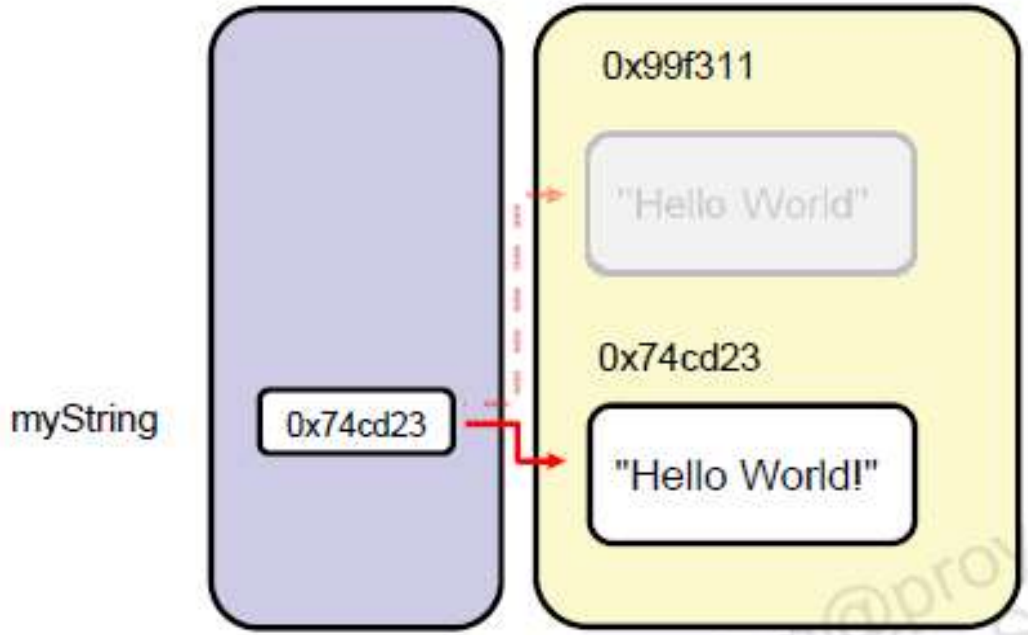
Concatenación de cadenas

```
String myString = "Hello";
myString = myString.concat(" World");
```



Concatenación de cadenas

```
String myString = "Hello";
myString = myString.concat(" World");
myString = myString + "!"
```



Llamadas al método String con valores de retorno primitivos

Una llamada a método puede devolver un único valor de cualquier tipo.

- Ejemplo de un método de tipo primitivo `int`:

```
String hello = "Hello World";
int stringLength = hello.length();
```

Llamadas al método String con valores de retorno de objeto

Llamadas a método que devuelven objetos:

```
String greet = " HOW ".trim();
String lc = greet + "DY".toLowerCase();
```

O bien

```
String lc = (greet + "DY").toLowerCase();
```

Llamadas a métodos que necesitan argumentos

Las llamadas a métodos pueden necesitar transferir uno o más argumentos:

- Transferir un primitivo

```
String theString = "Hello World";
String partString = theString.substring(6);
```

- Transferir un objeto

```
boolean endWorld =
            "Hello World".endsWith("World");
```

Documentación de la API de Java

Consta de un juego de páginas web.

- Muestra todas las clases de la API
  - Descripciones de la función de la clase
  - Lista de constructores, métodos y campos de la clase
- Gran cantidad de hiperenlaces para mostrar las interconexiones entre las clases y facilitar la búsqueda
- Disponible en el sitio web de Oracle en: http://download.oracle.com/javase/7/docs/api/index.html

Documentación de la plataforma Java SE 7

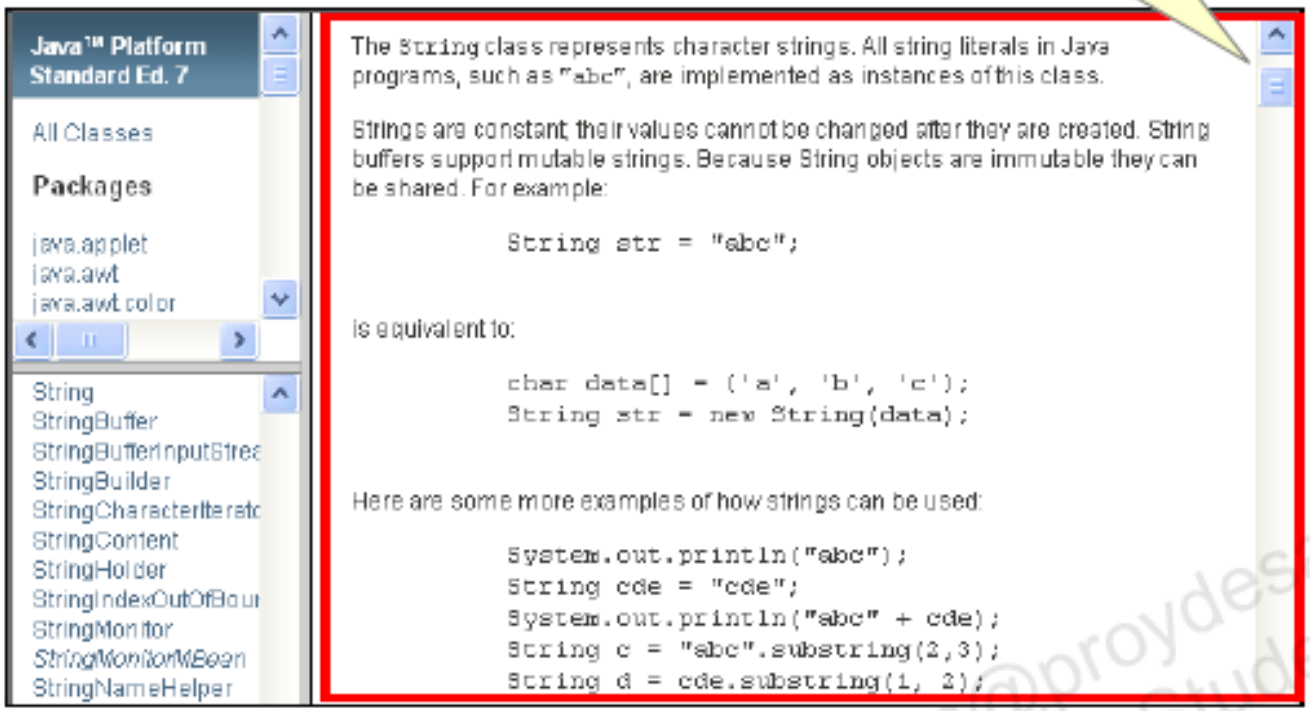
Aquí puede seleccionar All Classes o un paquete concreto.

En este panel se muestran detalles sobre la clase seleccionada.

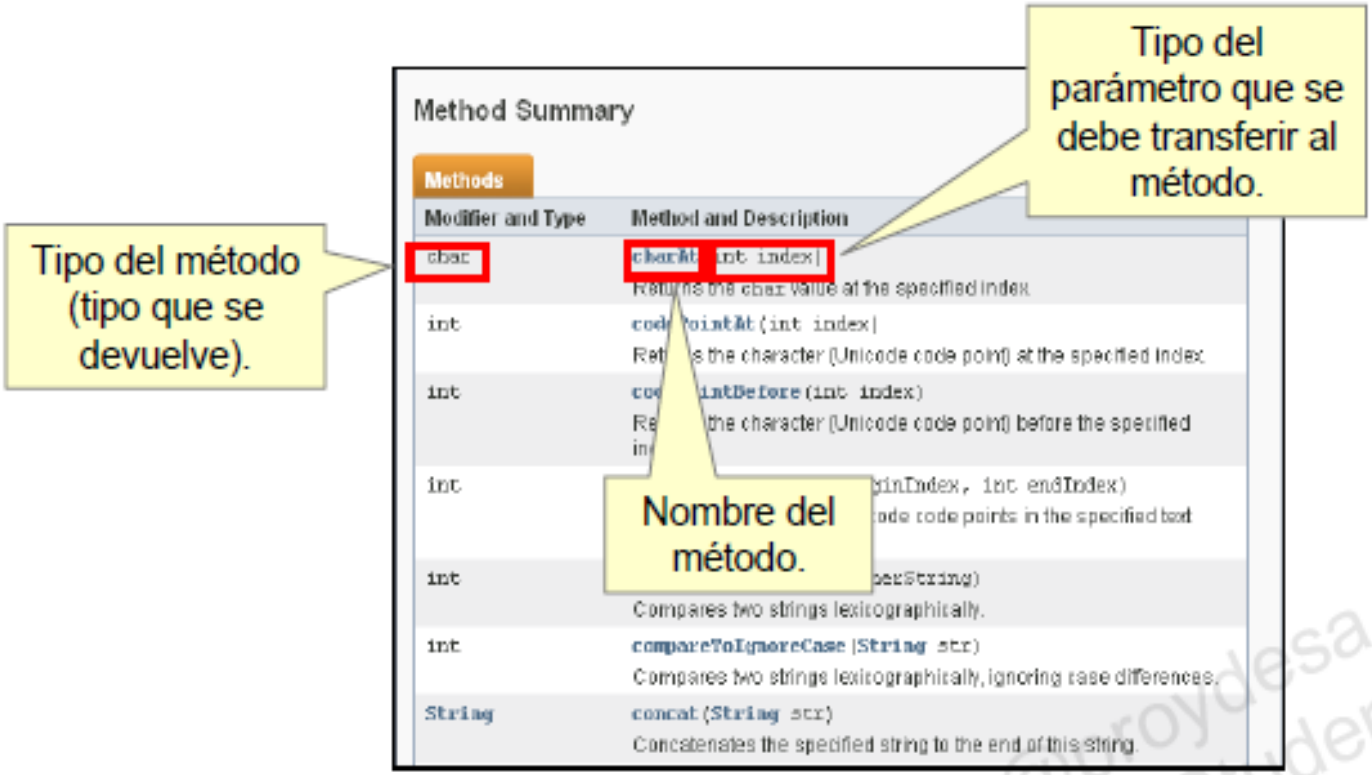
Según lo que seleccione, aquí se muestran las clases de un paquete concreto o todas las clases.

Documentación de la plataforma Java SE 7

Al desplazarse hacia abajo, se muestra más información de la clase String.

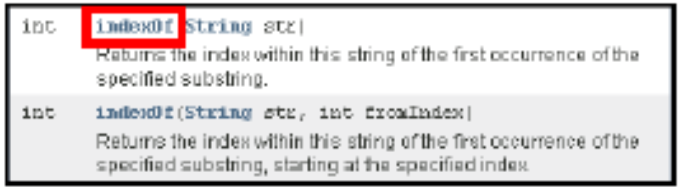


Plataforma Java SE 7: Resumen del método

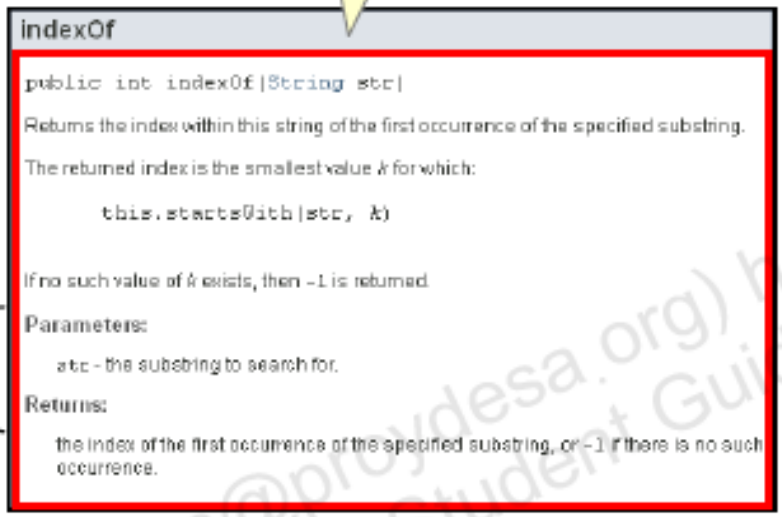


Plataforma Java SE 7: Detalles del método

Haga clic aquí para obtener la descripción detallada del método.



Descripción detallada del método `indexOf()`.



Se muestran más detalles sobre los parámetros y el valor de retorno en la lista de método.

Métodos `System.out`

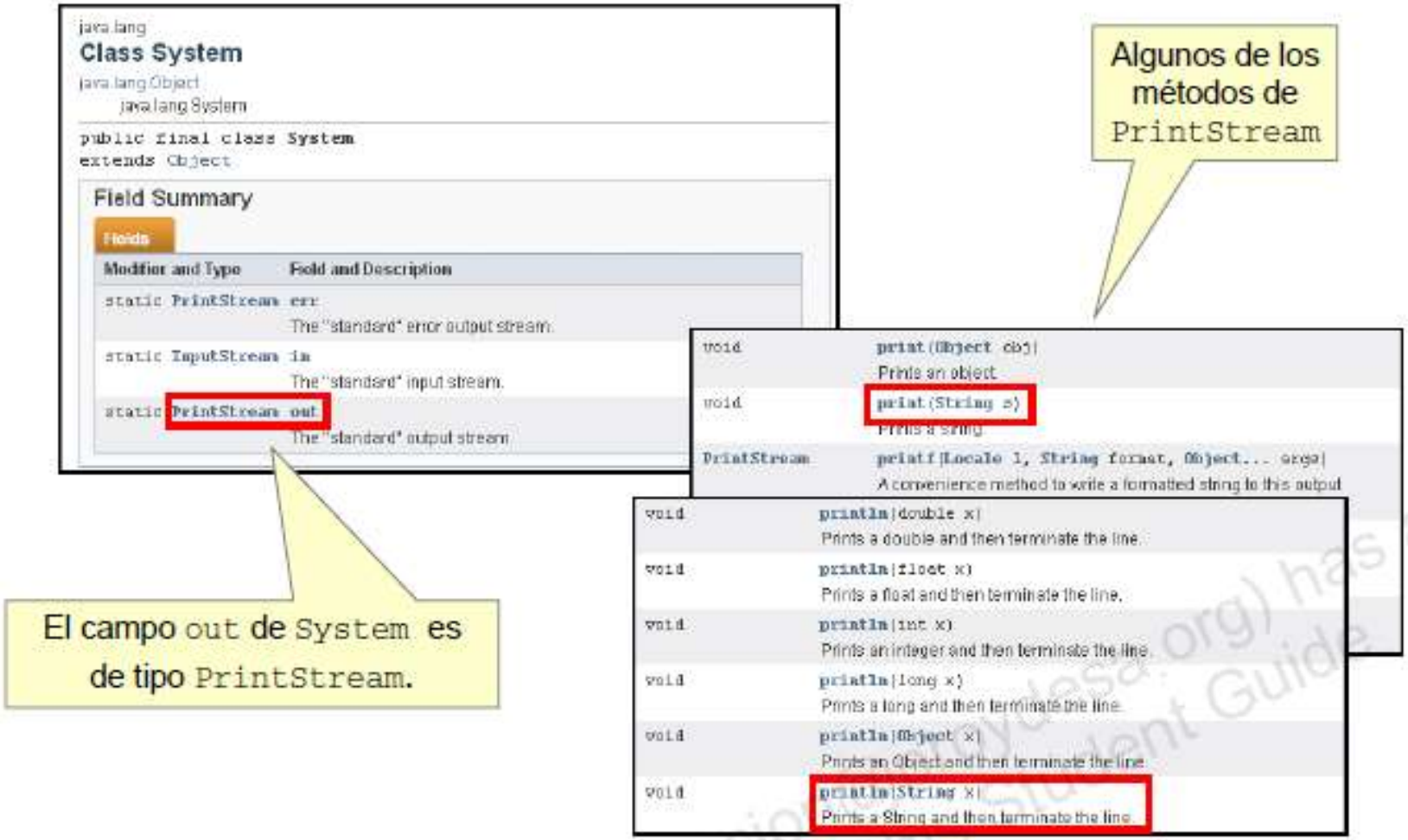
Para encontrar todos los detalles de `System.out.println()`, considere lo siguiente:

- `System` es una clase (en `java.lang`).
- `out` es un campo de `System`.
- `out` es un tipo de referencia que permite llamar a `println()` en el tipo de objeto al que hace referencia.

Para buscar la documentación:

1. Vaya a la clase `System` y busque el tipo del campo `out`.
2. Vaya a la documentación de dicho campo.
3. Revise los métodos disponibles.

Documentación sobre `System.out.println()`



Uso de los métodos `print()` y `println()`

- Método `println`:
  `System.out.println(`*`data_to_print`*`);`
- Ejemplo:

```
System.out.print("Carpe diem ");
System.out.println("Seize the day");
```

- Este método muestra lo siguiente:

```
Carpe diem Seize the day
```

Clase StringBuilder

StringBuilder proporciona una alternativa variable a String.

StringBuilder:

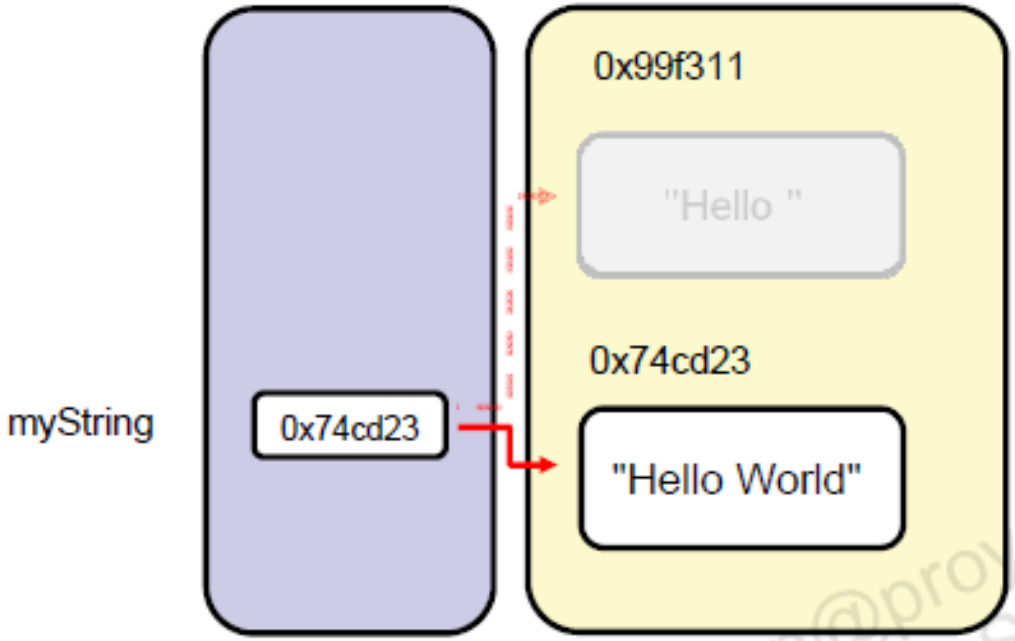
- Es una clase normal. Utilice `new` para instanciarla.
- Tiene un amplio juego de métodos para agregar, insertar y suprimir.
- Tiene muchos métodos para devolver una referencia al objeto actual. No hay ningún costo de instanciación.
- Se puede crear con la capacidad inicial que mejor se adapte a las necesidades.

String sigue siendo necesaria porque:

- Su uso puede ser más seguro que un objeto inmutable.
- Una clase de la API puede necesitar una cadena.
- Tiene muchos más métodos no disponibles en StringBuilder.

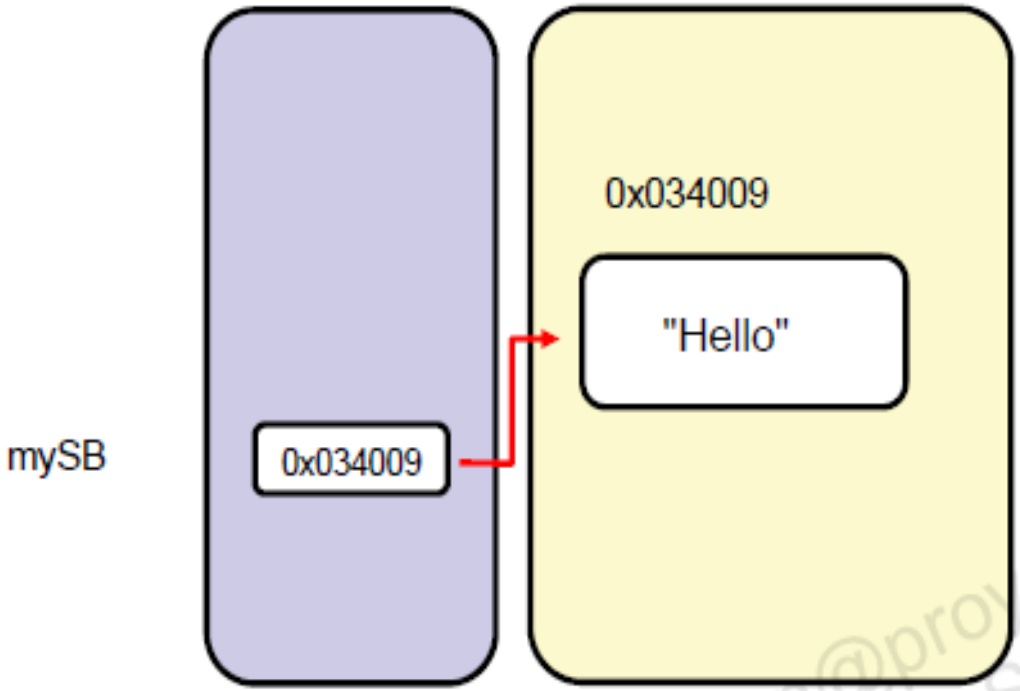
Ventajas de StringBuilder sobre String para la concatenación (o adición)

```
String myString = "Hello";
myString = myString.concat(" World);
```



StringBuilder: Declaración e instanciación

```
StringBuilder mySB = new StringBuilder("Hello");
```



Adición de StringBuilder

```
StringBuilder mySB = new StringBuilder("Hello");
mySB.append(" World");
```