



J01-06

使用 參考型別 (reference type)
操作 物件 (object)

曾瑞君 (Jim_Tzeng)

學習目標

1. 宣告 (Declaring)、實例化 (Instantiating)、初始化 (initializing) 物件 & 使用物件參考 (object reference)
2. 使用 String class
3. 使用 StringBuilder class
4. 使用 Java API Documentation
5. 基本型別的包裹類別
6. 使用 var 宣告變數





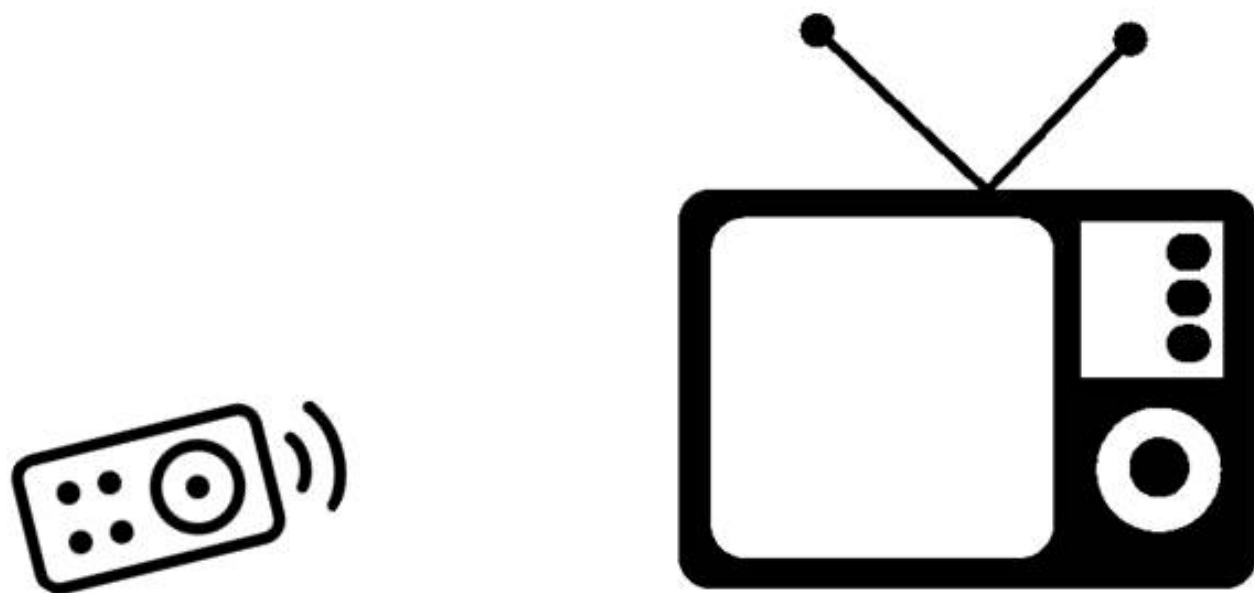
1/6

宣告、實例化、初始化物件 & 使用物件參考 (object reference)



使用「遙控器」

- 要使用物件 (object)，必須使用該物件的 **參考 (reference)**。類似我們使用「**遙控器**」由遠端操控「**電子產品**」：



Shirt 類別

```
public class Shirt {  
  
    public int shirtID = 11;  
    public double price = 100.5;  
    Public char colorCode = 'R';  
  
    public void display() {  
        System.out.println("shirtID = " + shirtID);  
        System.out.println("price = " + price);  
        System.out.println("colorCode = " + colorCode );  
    }  
}
```

由 class(類別)，建構 object(物件)

- Declaration (宣告):

Classname identifier (物件參考名稱);

```
Shirt myShirt;
```

- Instantiation (實例化):

new *Classname*();

```
new Shirt();
```

- Assignment 然後完成 Initialization (初始化):

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

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


使用物件參考，控制物件

- myShirt 是物件參考, 可以控制建立的 Shirt 物件：

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

- Java 是 Strong Type 的程式語言，重視型別 (Type)。
任何「變數」都需要宣告型別：

— 基本型別

```
int    x    =    10;
```

— 參考型別

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

- 「物件參考」即是宣告「參考型別」的「變數」

不同物件，使用不同參考(遙控器)

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

```
Trousers myTrousers = new Trousers();  
myTrousers.display();
```

- 使用 Shirt 的 reference type 去參照 Shirt 物件
- 使用 Trousers 的 reference type 去參照 Trousers 物件
- 目前，reference type 和所參照的 object 相同。但，不一定要完全相同。



物件參考與null

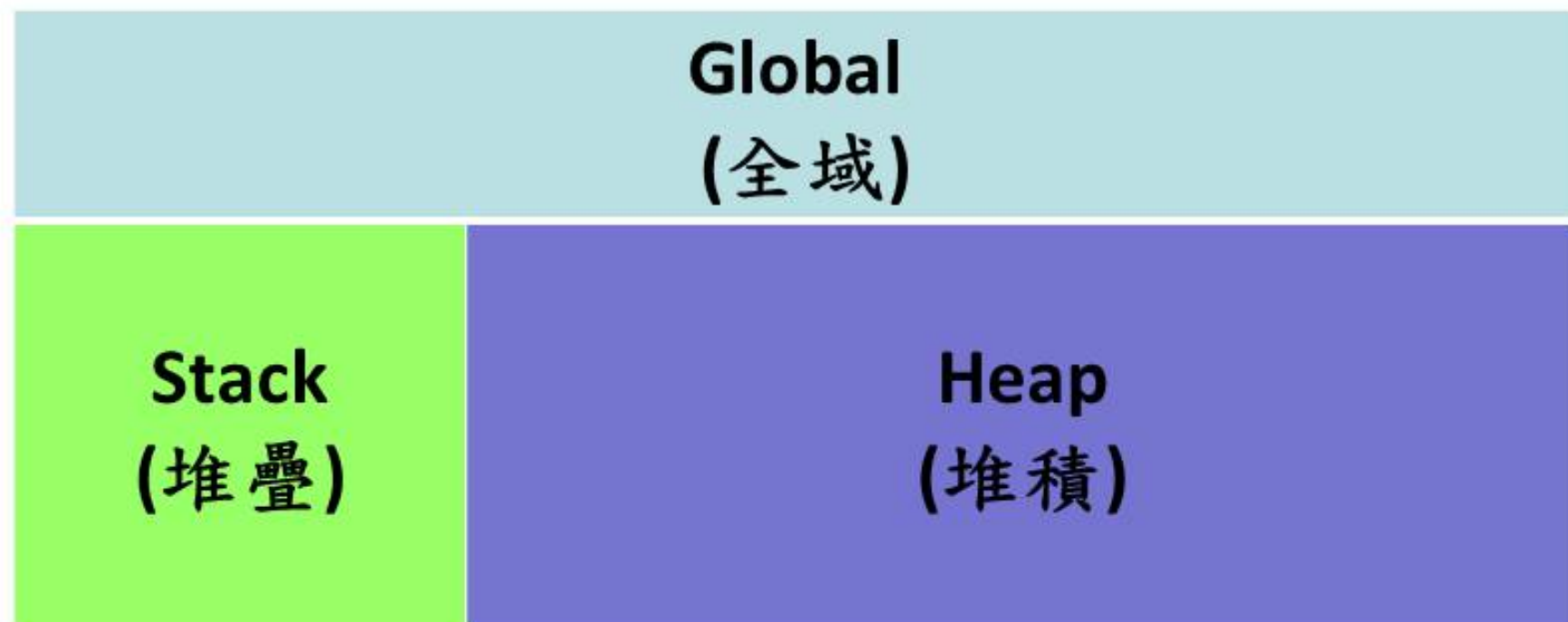
- 當物件參考變數沒有指向任何物件實例時，則該物件參考指向「null」；情形好比手上有一個遙控器，但該遙控器沒有指向任何電視。
- 當使用空的遙控器時，隨意點選按鍵不會有問題；但呼叫指向null的物件參考的方法，程式卻會出錯。



```
1 public class NullTest {  
2     public static void main(String[] args) {  
3         Shirt nullShirt = null;  
4         System.out.println(nullShirt);  
5  
6         String s1 = null + "Hi";  
7         System.out.println(s1);  
8  
9         String s2 = "Hi" + null;  
10        System.out.println(s2);  
11  
12        System.out.println(nullShirt.price); // will run failed!  
13    }  
14 }
```



Java 記憶體3區塊



JVM 記憶體分類

- Global (全域)
 - 存放著被宣告為 **static** 的類別成員變數。
- Stack (堆疊)
 - 存放**基本型別**(Primitive Type)的**變數**和**變數內容 (value)**的地方。
 - 存放**參考型別**(Reference Type)的**變數**的地方。
- Heap (堆積)
 - 存放**參考型別**(Reference Type)的**變數內容 (instance)**的地方。

JVM 記憶體分類

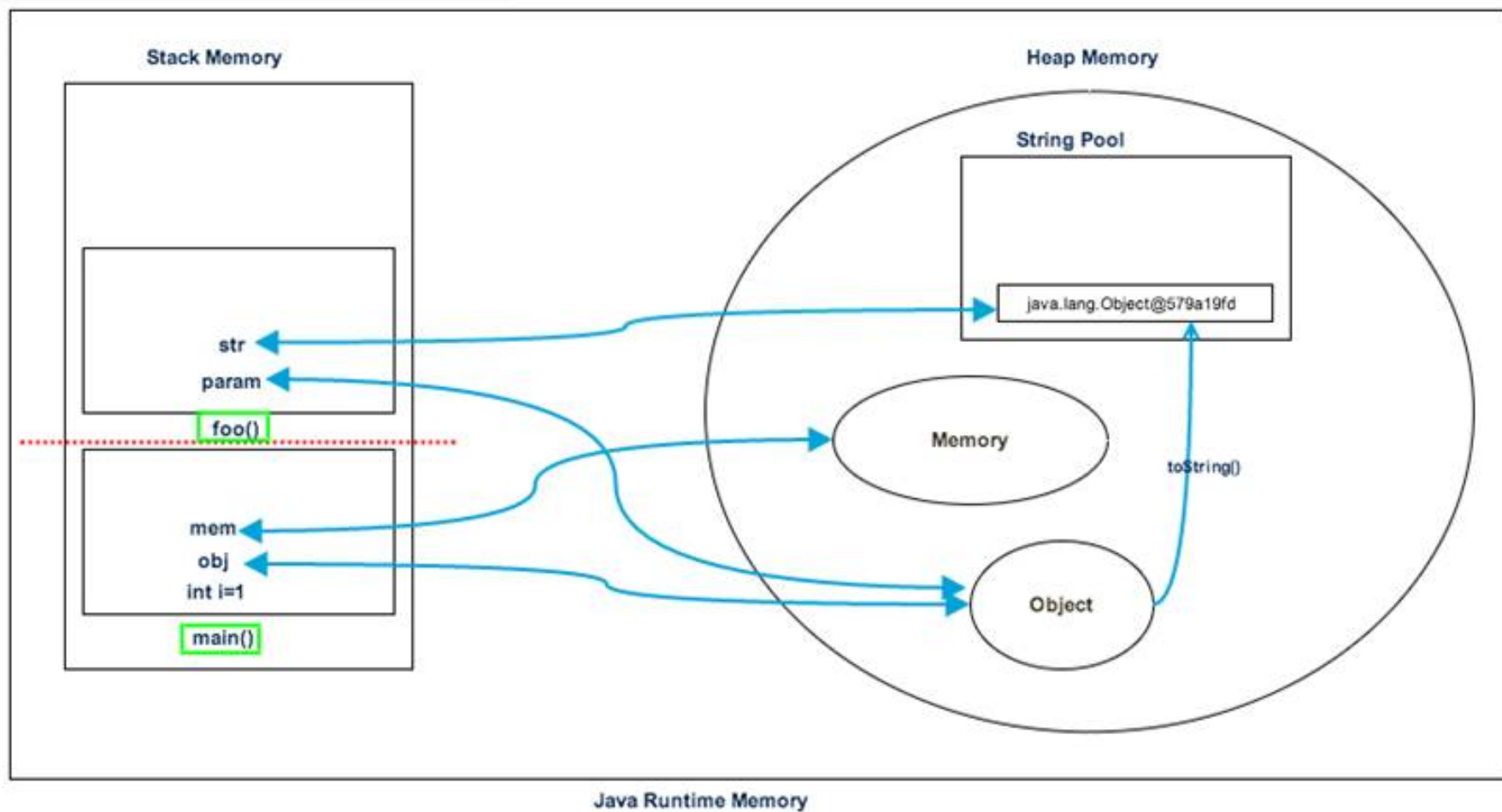
分類	變數 & 值	Stack (堆疊)	Heap (堆積)
基本型別	變數	★	
	值	★	
參考型別	變數(物件參考)	★	
	值(實例/instance)		★


```
public class Memory {
```

```
    public static void main(String[] args) { // Line 1  
        int i=1; // Line 2  
        Object obj = new Object(); // Line 3  
        Memory mem = new Memory(); // Line 4  
        mem.foo(obj); // Line 5  
    } // Line 9
```

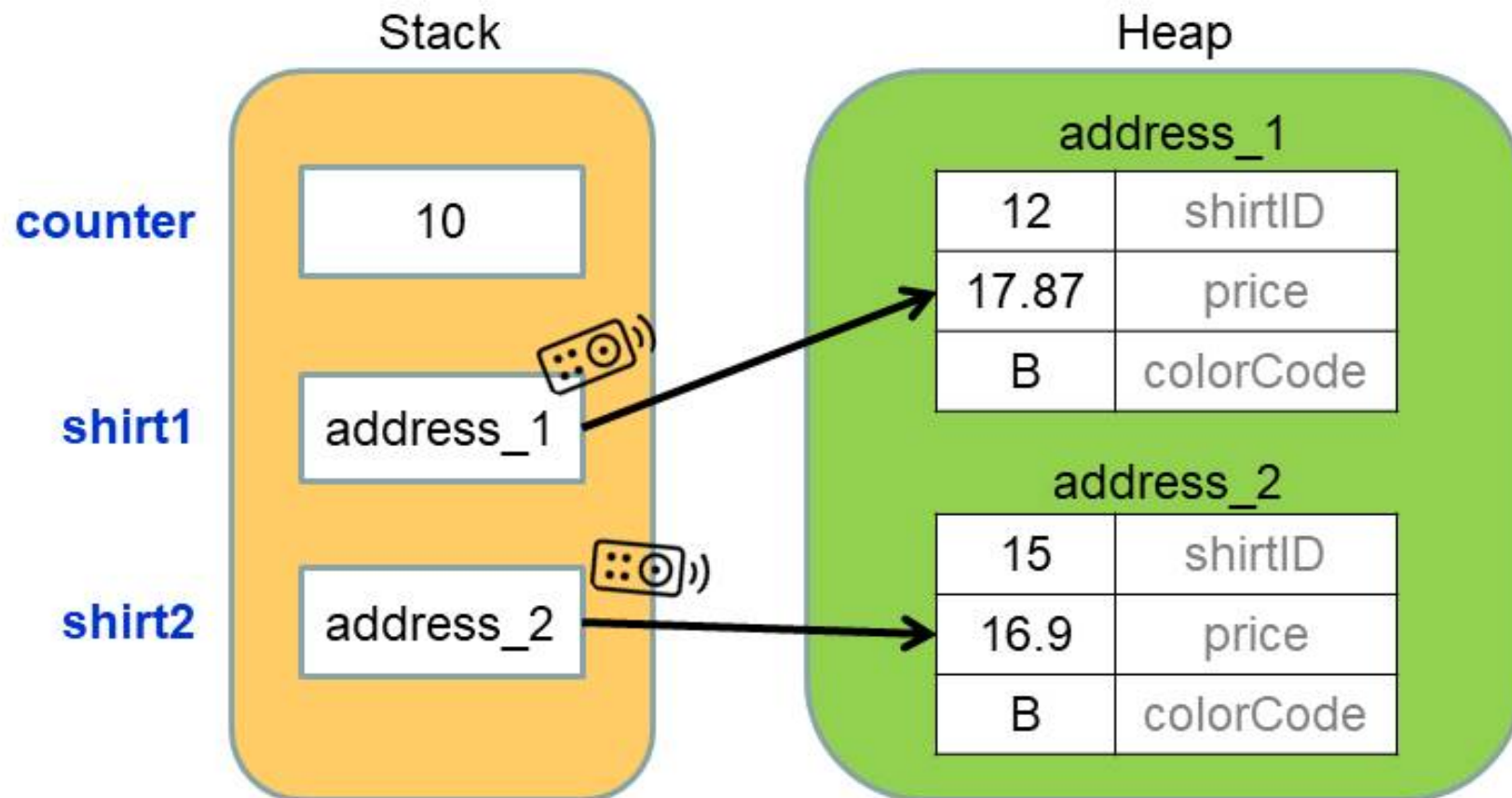
```
    private void foo(Object param) { // Line 6  
        String str = param.toString(); //// Line 7  
        System.out.println(str);  
    } // Line 8
```

```
}
```



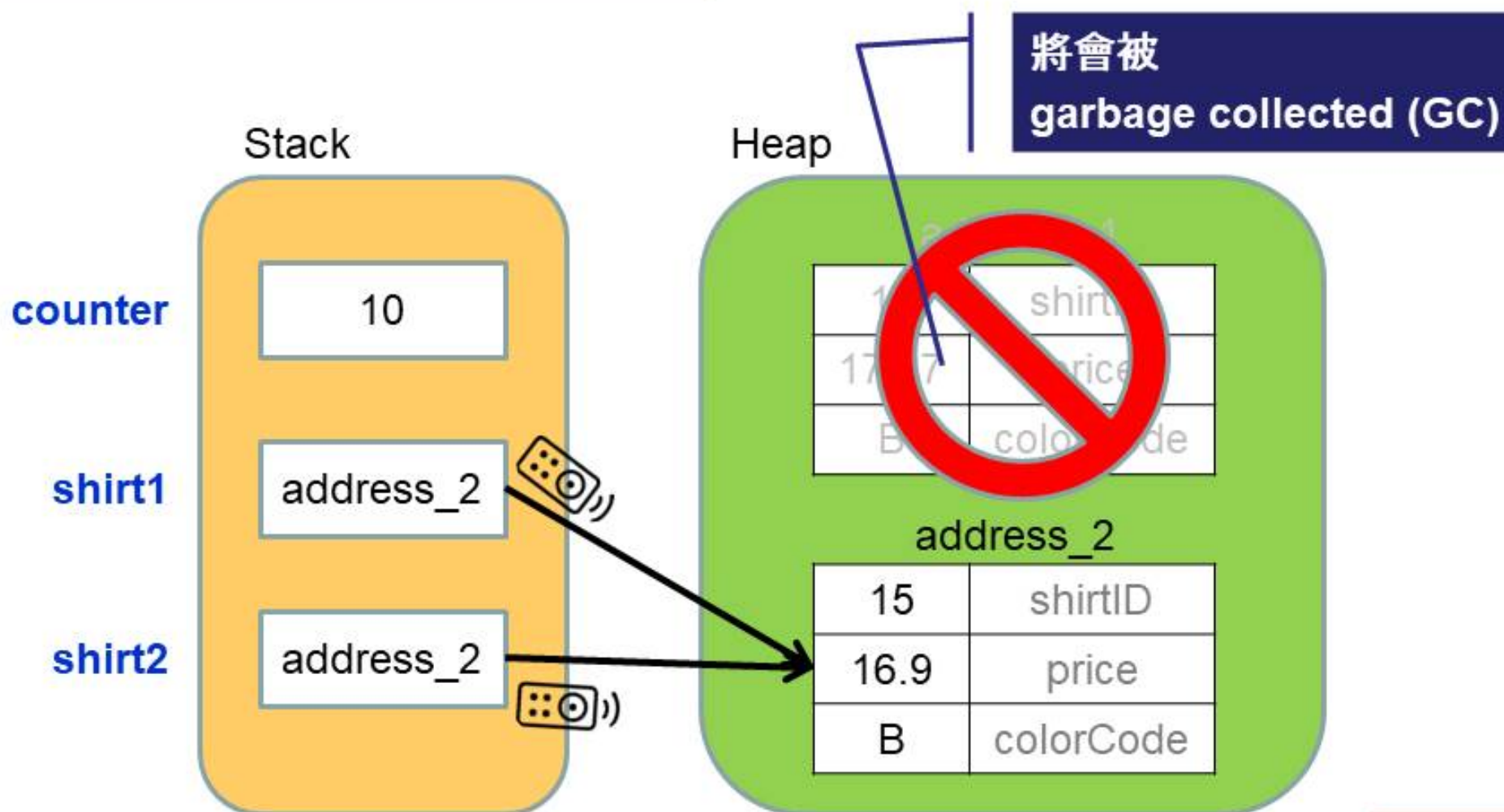
References and Objects In Memory

```
int counter = 10;  
Shirt shirt1 = new Shirt();  
Shirt shirt2 = new Shirt();
```



Assigning a Reference to Another Reference

```
shirt1 = shirt2;
```



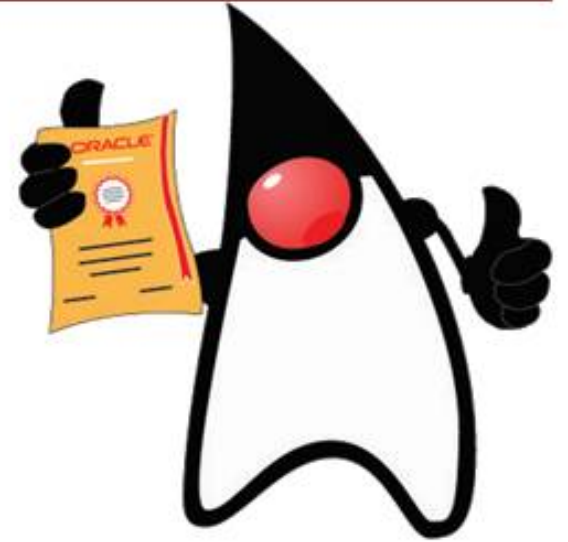


Two References, One Object

```
public class ReferenceTest {  
    public static void main(String[] args) {  
  
        Shirt shirt1 = new Shirt();  
        Shirt shirt2 = new Shirt();  
  
        shirt1 = shirt2;  
  
        shirt1.price = 1000;  
        shirt2.price = 500;  
  
        System.out.println("Shirt price: " + shirt1.price);  
    }  
}
```



Shirt price: 500.0



2/6

使用 **String** class



The String Class

String 類別支援非標準的語法：

- String 物件可以不需要使用“new”關鍵字進行實例化，這也是比較建議的方式。因為可以重複使用。
 - `String hisName = "Jim";`
- String 物件也可以使用“new”進行實例化，但不建議。因為將建立2個String物件：
 - `String hisName = new String ("Jim");`
- String 類別是 **immutable**；亦即 value 無法變更。
- 可使用“+”讓字串相連。

Concatenating Strings

- 當在程式碼中使用 String literal (字面常量)，將自動生成 String Object。
- 連接 String：

```
String name1 = "Jim"  
String name2 = name1 + " is teaching";
```

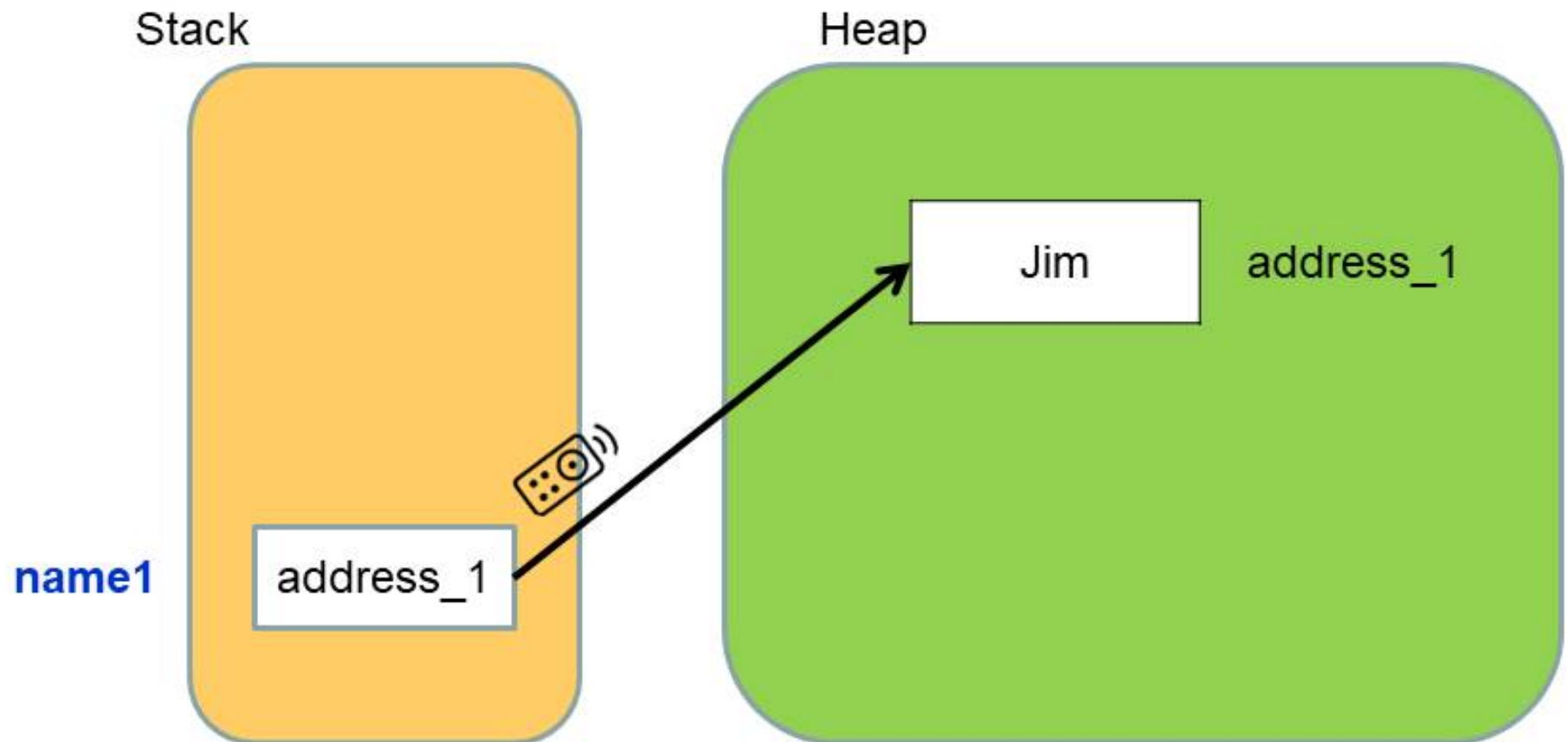
- String 相連後將產生新 String 物件。亦即 name2 將參照到不同於 name1 的一個新記憶體位址。

Immutable Demo

```
public class ImmutableDemo {  
    public static void main(String[] args) {  
        String name1 = "Jim";  
        name1 = name1.concat(" is teaching");  
        System.out.println(name1);  
  
        String name2 = "Jim";  
        name2.concat(" is teaching");  
        System.out.println(name2);  
    }  
}
```

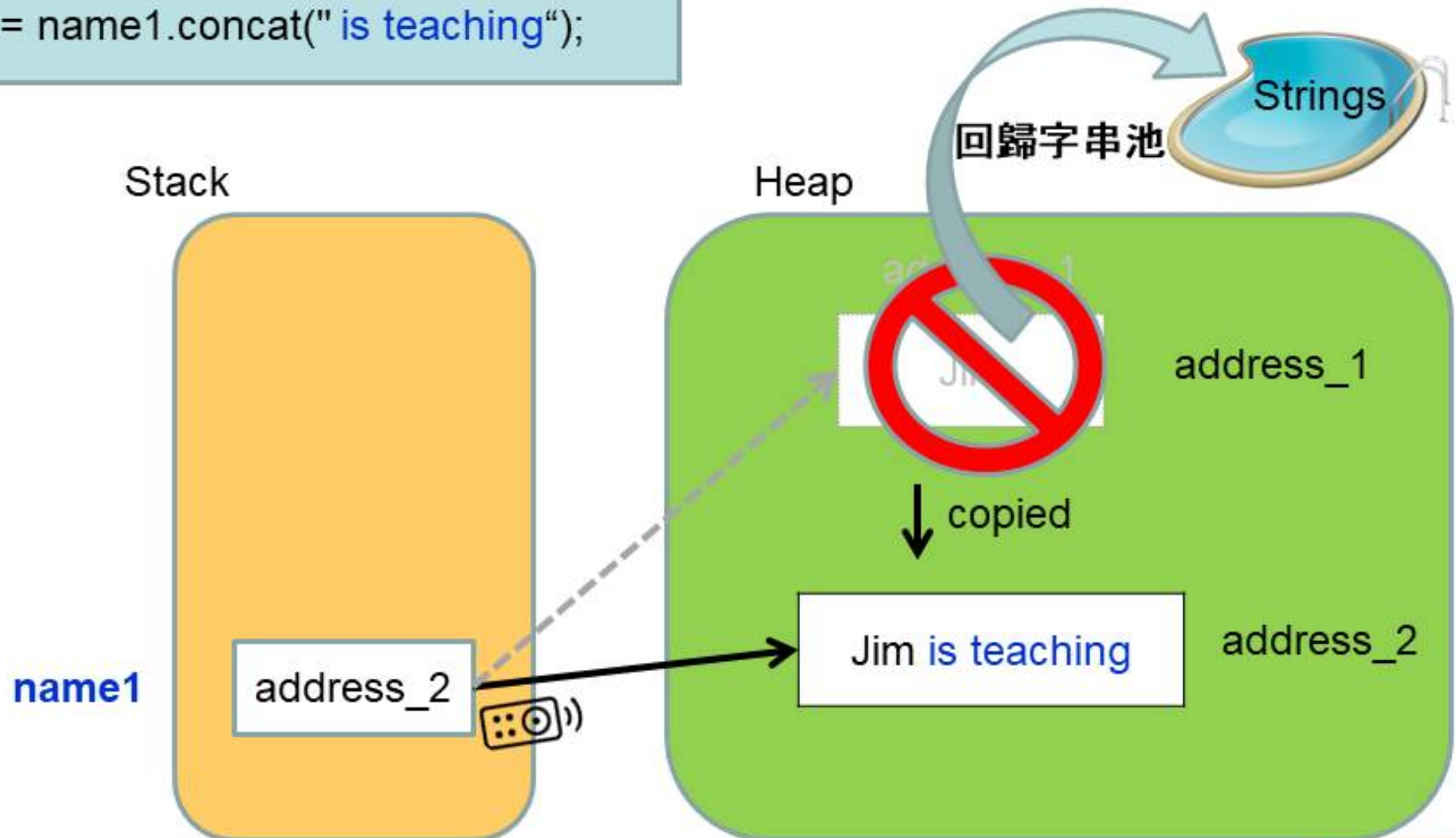
String Concatenation

```
String name1 = "Jim";
```



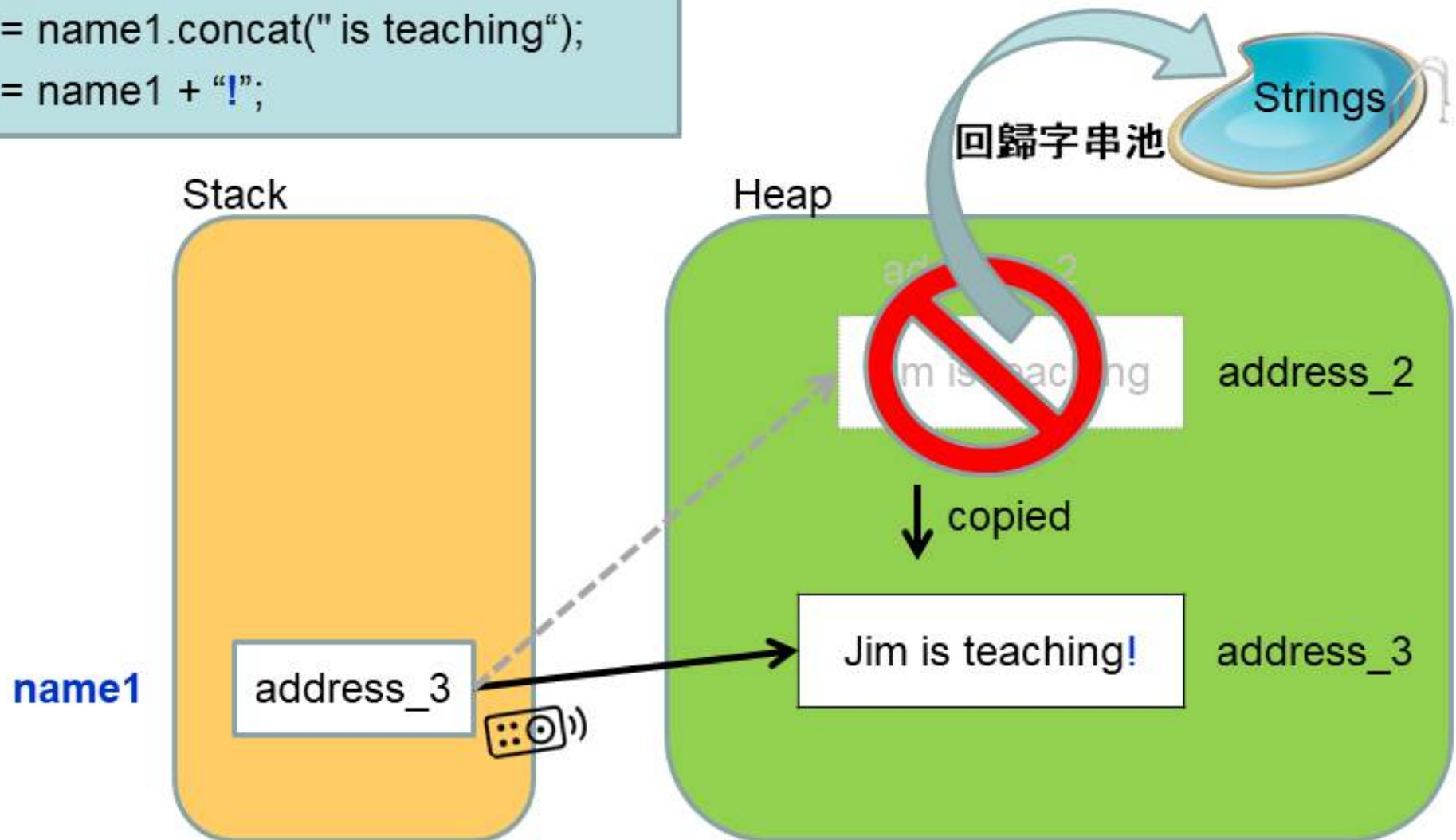
String Concatenation

```
String name1 = "Jim"  
name1 = name1.concat(" is teaching");
```



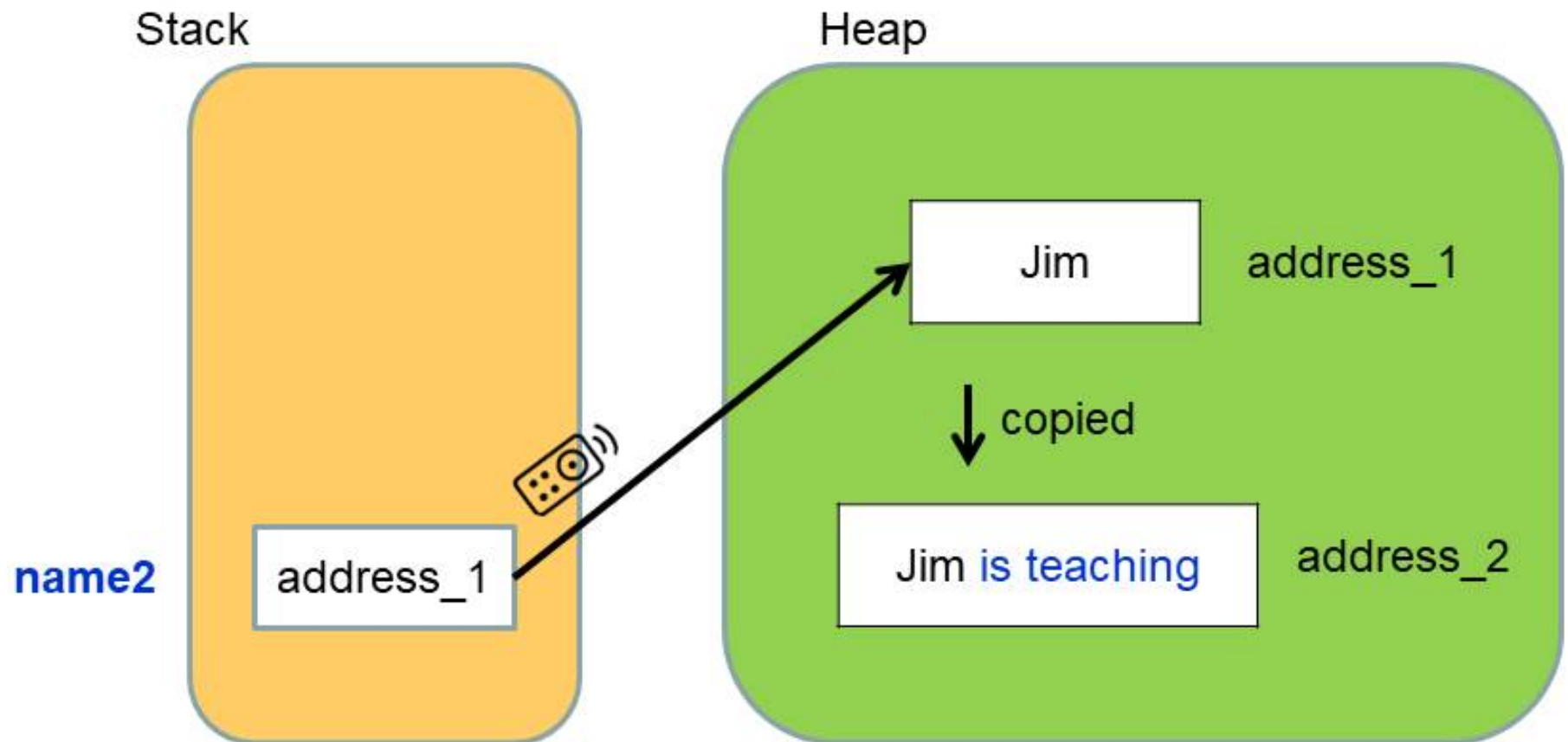
String Concatenation

```
String name1 = "Jim"  
name1 = name1.concat(" is teaching");  
name1 = name1 + "!";
```



String Concatenation

```
String name2 = "Jim";  
name2.concat(" is teaching");  
System.out.println(name2);
```



String 類別方法呼叫後 回傳 基本型別 / 參考型別

- Java 的 method 呼叫可以回傳任何型態的單一值
- 如回傳基本型別「int」：

```
String name1 = "Jim"  
int name1Length = name1.length();
```

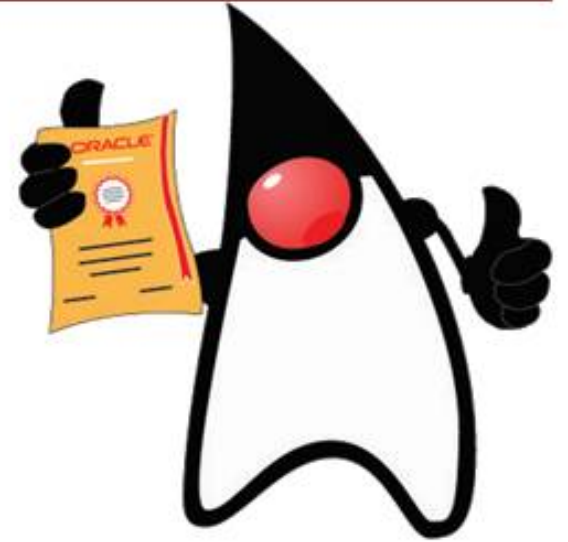
- 如回傳參考型別「String」：

```
String name1= " Jim ".trim();  
String lc = name1+ " TEACHES".toLowerCase();  
Or  
String lc = (name1+ " TEACHES").toLowerCase();
```

String 類別方法呼叫時 傳入 參數

- 不同方法需要不同種類、數量的參數：
- 傳入1, 2個基本型別和1個參考型別：

```
public static void main(String[] args) {  
    String name = "Jim Tzeng";  
  
    String lastName = name.substring(4);  
    System.out.println(lastName);  
  
    String lastName2 = name.substring(4, 9);  
    System.out.println(lastName2);  
  
    boolean end = name.endsWith("Tzeng");  
    System.out.println(end);  
}
```



3/6

使用 **StringBuilder** class



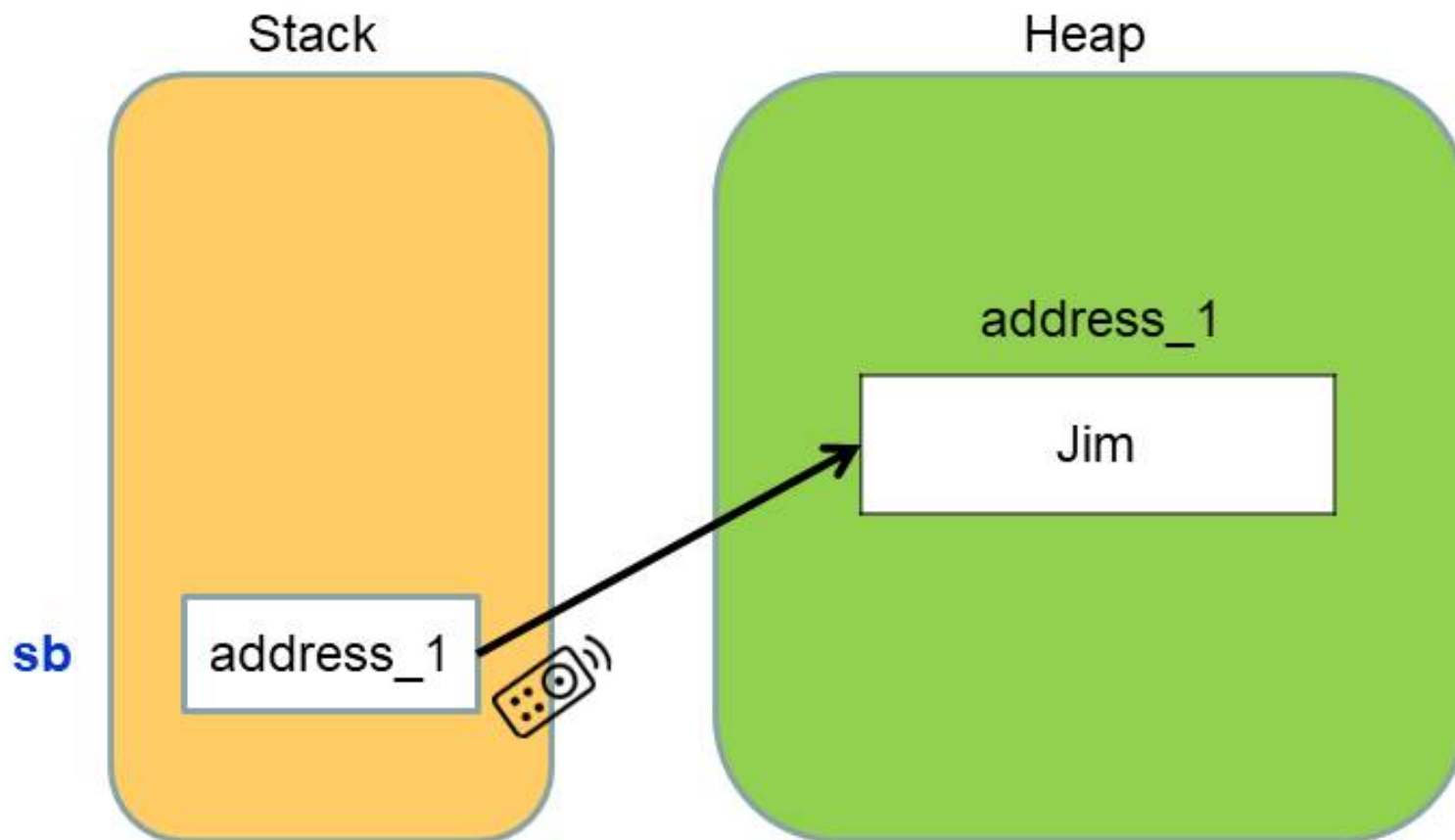


StringBuilder 類別

- 為 String 類別提供一個 **mutable** 的替代方案
- 大部分方法都回傳自己的參照，沒有實例化的成本。
- 必須使用 “new” 關鍵字進行物件實例化。
- 提供字串存取的擴充方法：append()，insert()，delete()。
- 建立時可以提供最佳化「initial capacity」。
- String 類別依然需要：
 - 使用 immutable 物件較安全
 - 其他 java class 仍需要
 - 擁有比 StringBuilder 更多的方法

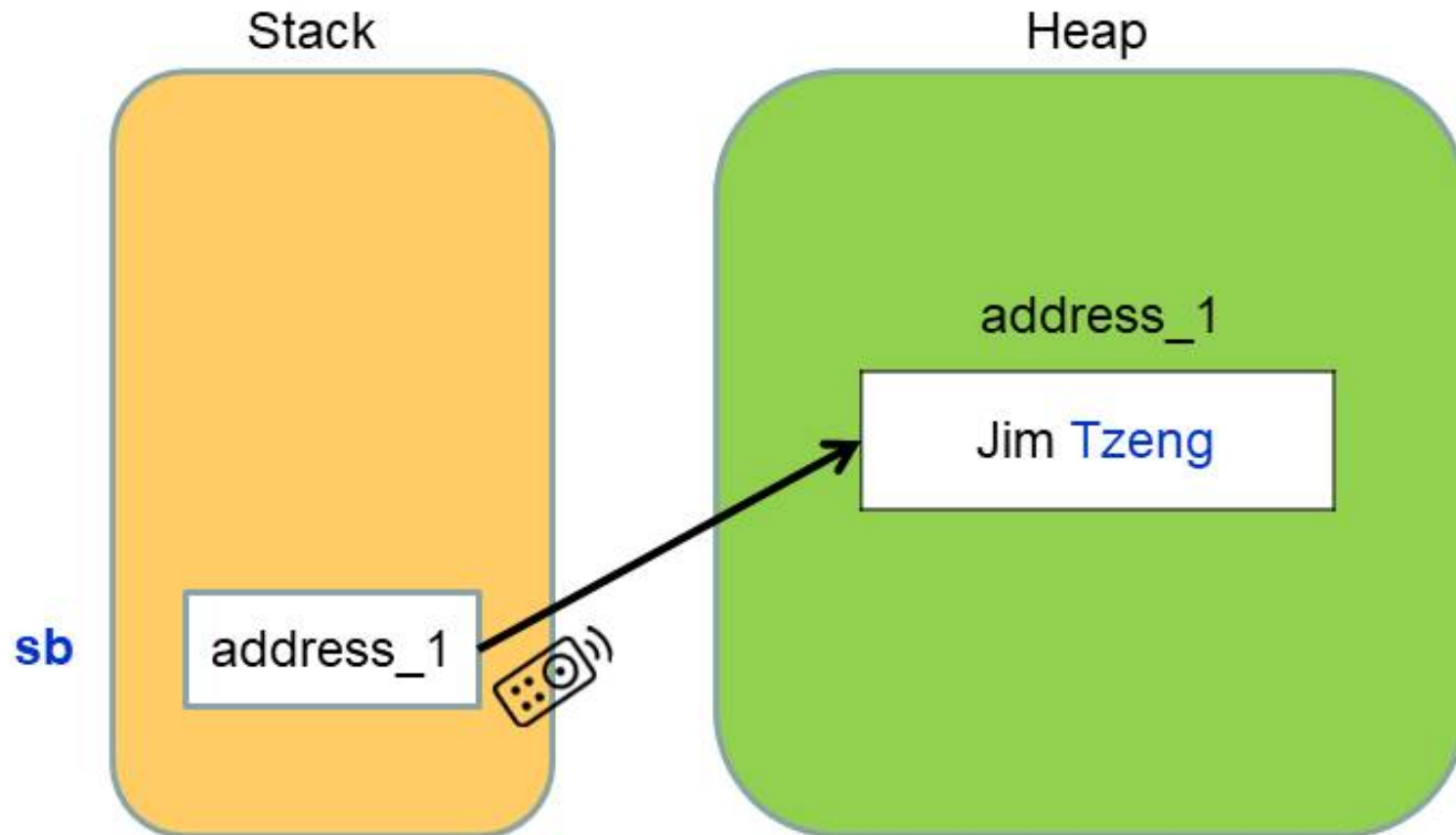
StringBuilder: Declare and Instantiate

```
StringBuilder sb = new StringBuilder("Jim");
```



StringBuilder: Append

```
StringBuilder sb = new StringBuilder("Jim");  
sb.append("Tzeng");
```



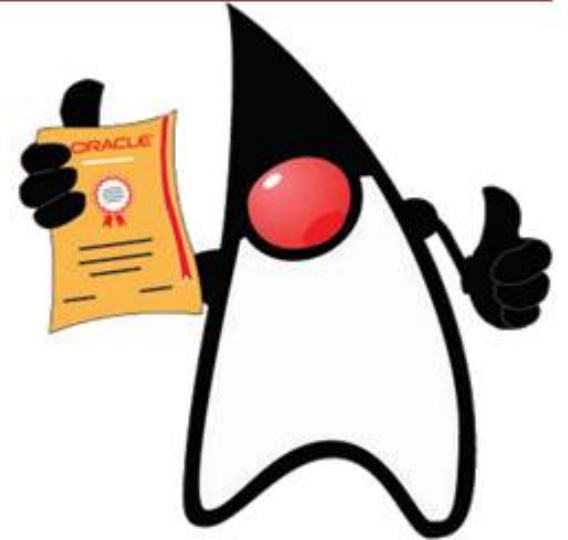
```
public static void main(String[] args) {
```

```
    StringBuilder sb1 = new StringBuilder(8);  
    sb1.append("jim");  
    sb1.append(" ");  
    sb1.append("tzeng");  
    System.out.println("sb1: " + sb1.toString());  
    System.out.println("sb1 object capacity: " + sb1.capacity());  
    System.out.println("sb1 sub string: " + sb1.substring(0, 5));  
    System.out.println("sb1 sub string: " + sb1.substring(0, 10)); // error at runtime
```

```
    StringBuilder sb2 = new StringBuilder();  
    sb2.append("123456789");  
    sb2.insert(3, "-");  
    sb2.insert(7, "-");  
    System.out.println("sb2: " + sb2.toString());
```

```
    StringBuilder sb3 = new StringBuilder("12345678");  
    sb3.delete(3, 5);  
    System.out.println("sb3: " + sb3.toString());
```

```
}
```



4/6

Java API Documentation





Java API Documentation

- <https://docs.oracle.com/en/java/javase/11/docs/api/index.html>
- 列出所有 classes (類別)
 - 類別的目的
 - 列出所有類別的 constructors, methods, & fields
- 關鍵文字有超連結互動



Java Module → java.base

Overview (Java SE 11 & JDK 11) × +

docs.oracle.com/en/java/javase/11/docs/api/index.html

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP Java SE 11 & JDK 11

ALL CLASSES SEARCH: Search

Java® Platform, Standard Edition & Java Development Kit Version 11 API Specification

This document is divided into two sections:

Java SE

The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.

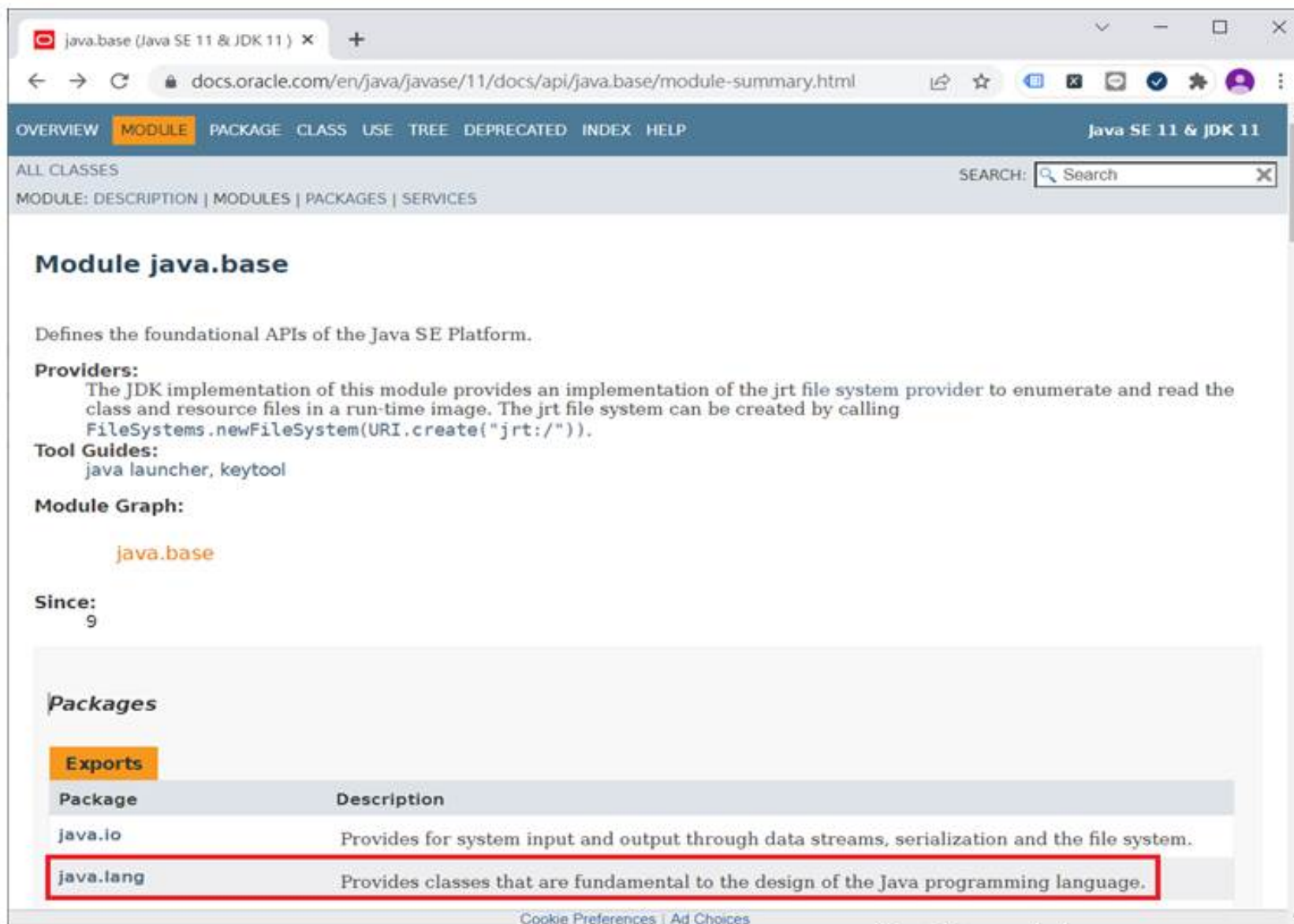
JDK

The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

All Modules	Java SE	JDK	Other Modules
Module	Description		
java.base	Defines the foundational APIs of the Java SE Platform.		
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.		
java.datatransfer	Defines the API for transferring data between and within applications.		
java.desktop	Defines the AWT and Swing user interface toolkits, the JavaFX API, and the Java 2D API.		

Cookie Preferences | Ad Choices

Java Package → java.lang



java.base (Java SE 11 & JDK 11) x

docs.oracle.com/en/java/javase/11/docs/api/java.base/module-summary.html

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java SE 11 & JDK 11

ALL CLASSES

MODULE: DESCRIPTION | MODULES | PACKAGES | SERVICES

SEARCH: Search

Module java.base

Defines the foundational APIs of the Java SE Platform.

Providers:
The JDK implementation of this module provides an implementation of the jrt file system provider to enumerate and read the class and resource files in a run-time image. The jrt file system can be created by calling `FileSystems.newFileSystem(URI.create("jrt:/"))`.

Tool Guides:
java launcher, keytool

Module Graph:

java.base

Since:
9

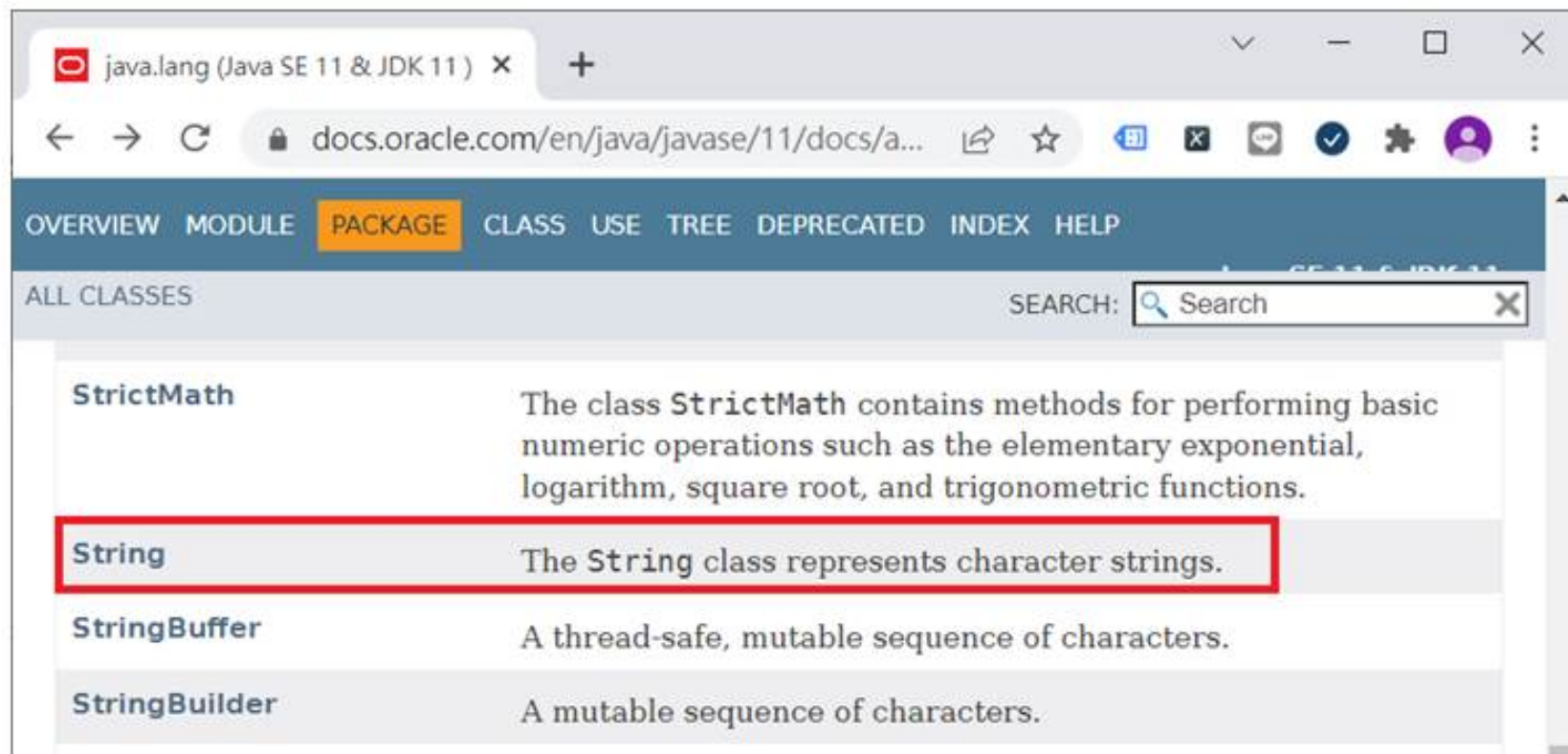
Packages

Exports

Package	Description
java.io	Provides for system input and output through data streams, serialization and the file system.
java.lang	Provides classes that are fundamental to the design of the Java programming language.

Cookie Preferences | Ad Choices

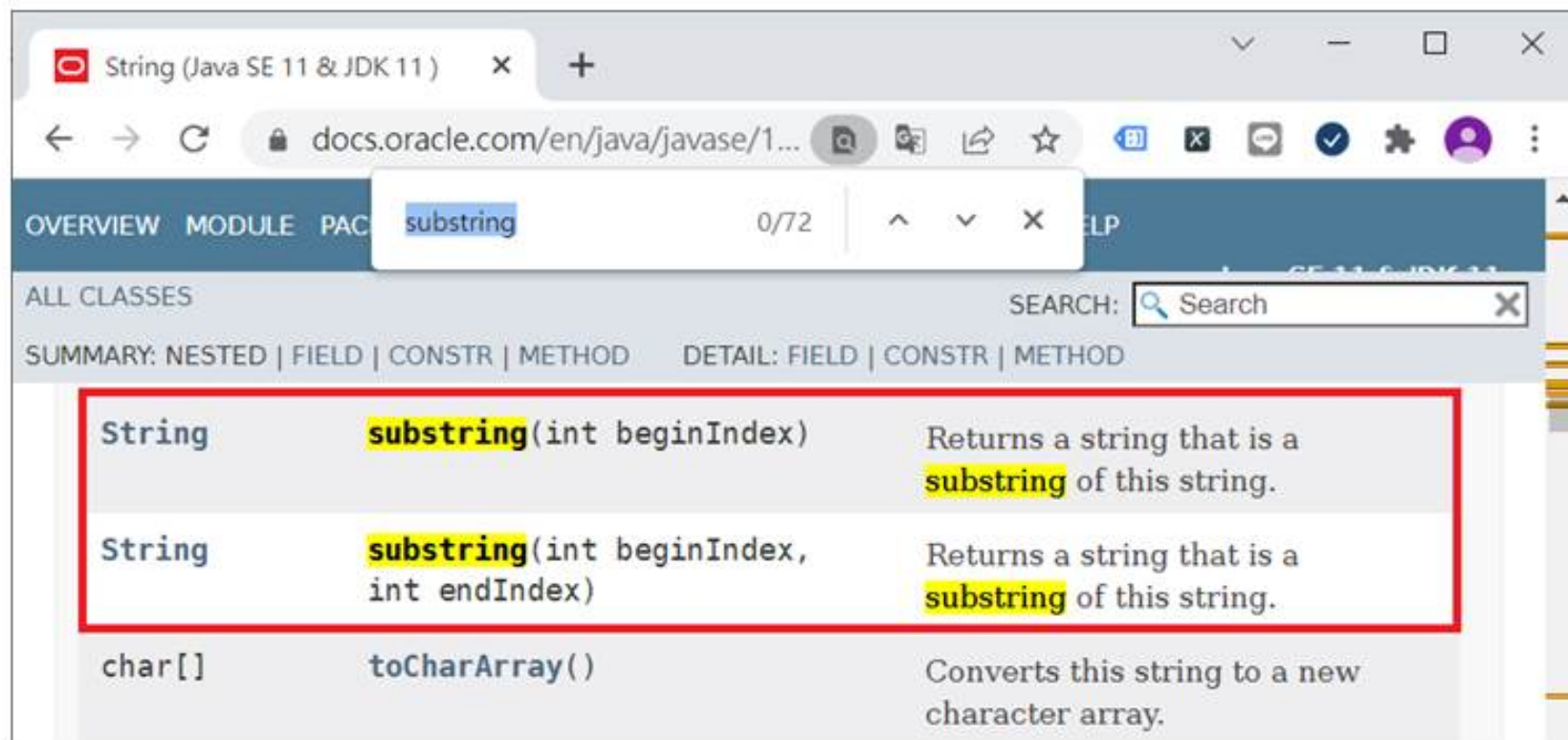
Java Class → String



The screenshot shows a web browser window with the URL `docs.oracle.com/en/java/javase/11/docs/a...`. The page displays the "PACKAGE" tab for the `java.lang` package. Under the "ALL CLASSES" section, a list of classes is shown. The `String` class is highlighted with a red rectangular border. The descriptions for the classes are as follows:

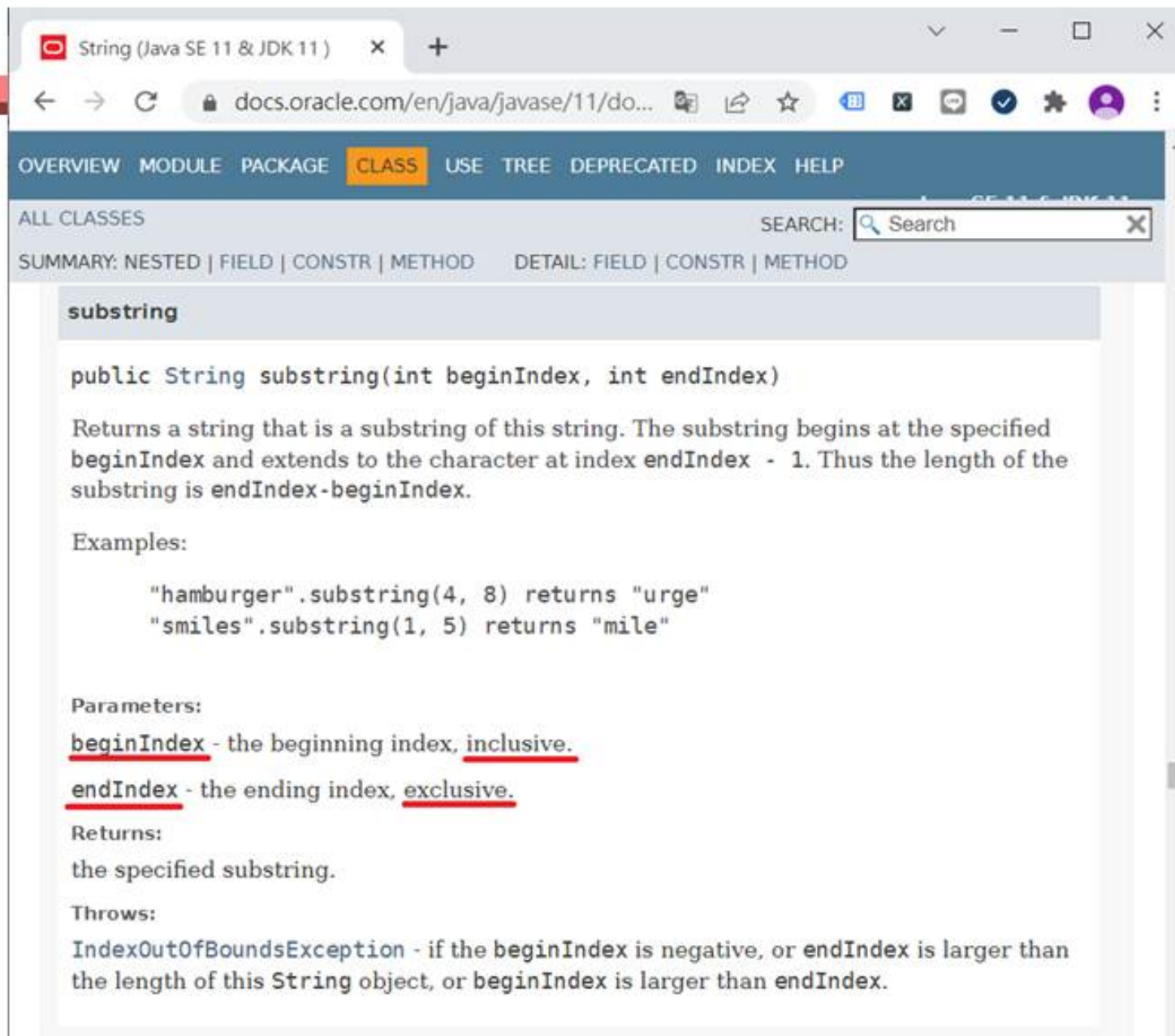
Class Name	Description
<code>StrictMath</code>	The class <code>StrictMath</code> contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.
<code>String</code>	The <code>String</code> class represents character strings.
<code>StringBuffer</code>	A thread-safe, mutable sequence of characters.
<code>StringBuilder</code>	A mutable sequence of characters.

Java Method → substring()



The screenshot shows the Oracle Java SE 11 & JDK 11 documentation page for the `String` class. The browser address bar shows `docs.oracle.com/en/java/javase/11...`. The page title is `String (Java SE 11 & JDK 11)`. The navigation bar includes tabs for `OVERVIEW`, `MODULE`, `PAC`, and `substring`. The `substring` tab is active, and a search bar is visible. The `substring` method is highlighted with a red box. The table below lists the methods:

Method	Signature	Description
<code>String</code>	<code>substring(int beginIndex)</code>	Returns a string that is a <code>substring</code> of this string.
<code>String</code>	<code>substring(int beginIndex, int endIndex)</code>	Returns a string that is a <code>substring</code> of this string.
<code>char[]</code>	<code>toCharArray()</code>	Converts this string to a new character array.



The screenshot shows a web browser window with the URL `docs.oracle.com/en/java/javase/11/do...`. The page title is "String (Java SE 11 & JDK 11)". The navigation bar includes links for OVERVIEW, MODULE, PACKAGE, CLASS (highlighted), USE, TREE, DEPRECATED, INDEX, and HELP. Below the navigation bar, there is a search bar and a summary section with links for NESTED, FIELD, CONSTR, and METHOD. The main content area is titled "substring" and contains the following information:

```
public String substring(int beginIndex, int endIndex)
```

Returns a string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex - beginIndex`.

Examples:

```
"hamburger".substring(4, 8) returns "urge"
"smiles".substring(1, 5) returns "mile"
```

Parameters:

- `beginIndex` - the beginning index, inclusive.
- `endIndex` - the ending index, exclusive.

Returns:

the specified substring.

Throws:

`IndexOutOfBoundsException` - if the `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`.

Eclipse

```
/**
 * Returns a string that is a substring of this string. The
 * substring begins at the specified {@code beginIndex} and
 * extends to the character at index {@code endIndex - 1}.
 * Thus the length of the substring is {@code endIndex-beginIndex}.
 * <p>
 * Examples:
 * <blockquote><pre>
 * "hamburger".substring(4, 8) returns "urge"
 * "smiles".substring(1, 5) returns "mile"
 * </pre></blockquote>
 *
 * @param    beginIndex    the beginning index, inclusive.
 * @param    endIndex      the ending index, exclusive.
 * @return    the specified substring.
 * @exception IndexOutOfBoundsException if the
 *           {@code beginIndex} is negative, or
 *           {@code endIndex} is larger than the length of
 *           this {@code String} object, or
 *           {@code beginIndex} is larger than
 *           {@code endIndex}.
 */
public String substring(int beginIndex, int endIndex) {
    int length = length();
    checkBoundsBeginEnd(beginIndex, endIndex, length);
    int subLen = endIndex - beginIndex;
    if (beginIndex == 0 && endIndex == length) {
        return this;
    }
    return isLatin1() ? StringLatin1.newString(value, beginIndex, subLen)
        : StringUTF16.newString(value, beginIndex, subLen);
}
```



System.out.println()

由文件裡了解：System.out.println()

- 1) Class : System (in java.lang)
- 2) Field : out
- 3) Link to : PrintStream class
- 4) method : println()

Class System

java.lang.Object
java.lang.System

```
public final class System  
extends Object
```

The `System` class contains several useful class fields and methods. It cannot be instantiated.

Among the facilities provided by the `System` class are standard input, standard output, and error output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.

Since:

JDK1.0

Field Summary

Fields

Modifier and Type	Field and Description
static <code>PrintStream</code>	<code>err</code> The "standard" error output stream.
static <code>InputStream</code>	<code>in</code> The "standard" input stream.
static <code>PrintStream</code>	<code>out</code> The "standard" output stream.

PrintStream (Java SE 11 & JDK 11)

docs.oracle.com/en...

OVERVIEW MODULE PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP

ALL CLASSES SEARCH:

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Module java.base
Package java.io

Class PrintStream

java.lang.Object
 java.io.OutputStream
 java.io.FilterOutputStream
 java.io.PrintStream

All Implemented Interfaces:
Closeable, Flushable, Appendable, AutoCloseable

Direct Known Subclasses:
LogStream

print

```
public void print(Object obj)
```

Prints an object. The string produced by the `String.valueOf(Object)` method is translated into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:

`obj` - The `Object` to be printed

See Also:

`Object.toString()`

println

```
public void println()
```

Terminates the current line by writing the line separator string. The line separator string is defined by the system property `line.separator`, and is not necessarily a single newline character (`'\n'`).

print() & println()

- 兩個方法差別在“ln”，亦即 line，表示“換行”
- 呼叫println()方法，和呼叫print()時在字串後方加上換行符號「\n」有相同效果：

```
public class PrintLineDemo {  
    public static void main(String[] args) {  
        System.out.print("helloWord\n");  
        System.out.println("helloWord");  
    }  
}
```




5/6

基本型別的包裹類別 (Wrapper Class)



Wrapper Class

基本型別	包裹類別	包裹類別的父類別
byte	Byte	Number
short	Short	
int	Integer	
long	Long	
float	Float	
double	Double	Object
char	Character	
boolean	Boolean	



```
Byte b = new Byte((byte) 1);  
Short s = new Short((short) 2);  
Integer i = Integer.parseInt("3");  
Long l = Long.valueOf(4);  
Float f = Float.valueOf("2.01");  
Double d = new Double(3.01);  
Character c = new Character('a');
```

```
Boolean bBrue = new Boolean(true);  
Boolean bFalse1 = new Boolean(false);  
Boolean bFalse2 = new Boolean(null);
```

```
System.out.println("Short is between " + Short.MAX_VALUE + " ~ " +  
Short.MIN_VALUE);
```

```
System.out.println(i.compareTo(5));  
System.out.println(l.compareTo(i.longValue()));  
System.out.println(Long.sum(2, 5));  
System.out.println(bFalse2);
```



6/6

使用 **var** 宣告變數



var 宣告的使用時機

- 從Java 10開始，開發者可以選擇在某些條件下使用關鍵字「var」取代區域變數的型別宣告。要使用此功能，開發者只需鍵入var而不是基本型別或參考型別。
- 這個功能的正式名稱是「區域變數型別推斷(local variable type inference)」，顧名思義就是只能套用在區域變數(local variable)，因此無法使用在實例變數；並由等號右側的變數值推斷(inference)變數型別(type)。

```
1 public void whatIsTheType() {  
2     var name = "Hello";  
3     var size = 7;  
4 }
```

Java & JavaScript 對 var 的用法不同

- Java使用var與JavaScript之類的語言不同。
 - JavaScript預期以var宣告的變數表示在執行時可以自動轉換成任何型別的變數。
 - 在Java中仍然是編譯時定義的特定型別，它不會在執行時更改型別。

1	public void reassignment() {
2	var number = 7;
3	number = 4;
4	number = "5"; // 編譯失敗
5	}

var 變數宣告的升等和轉型

- 以var宣告的變數在「升等和轉型」的運用上，和以非var宣告的變數是一致的。對於 short 型別的轉型測試範例：

```
1 public void showNonVarCasting() {  
2     short s = (short) 10;  
3     s = (byte) 5;  
4     s = 1_000_000; // 編譯失敗  
5 }
```

- 和改用var宣告的結果相同：

```
1 public void showVarCasting() {  
2     var s = (short) 10;  
3     s = (byte) 5;  
4     s = 1_000_000; // 編譯失敗  
5 }
```

以var宣告但無法推斷型別時編譯失敗

```
1 public void doesThisCompile(boolean check) {  
2     var question; // 編譯失敗  
3     question = 1;  
4     var answer; // 編譯失敗  
5     if (check) {  
6         answer = 2;  
7     } else {  
8         answer = 3;  
9     }  
10    System.out.println(answer);  
11 }
```

```
1 public void varAsNull() {  
2     var n = null; // 編譯失敗  
3 }
```


複合宣告(compound declaration)不適用於var變數

- 複合宣告的情境是同時有多個變數進行宣告：
 - 程式行開頭宣告型別，行間不再出現型別宣告；使用「,」區隔變數，敘述最終依然是「;」結尾。
 - 一個別變數可以自行決定是否初始化給值。

1	public void compoundDeclarationWithNonVar() {
2	int a, b = 3;
3	int c = 2, d = 3;
4	int e, f;
5	int g = 2, h;
6	int i, int j; // 編譯失敗, 非複合宣告
7	int k, double l; // 編譯失敗, 非複合宣告
8	}



- 以var宣告時Java需要推斷型別，只能一個一個來，無法應用於複合宣告上，因此以下無法編譯：

1	public void compoundDeclarationWithVar() {
2	var a, b = 3; // 編譯失敗
3	var c = 2, d = 3; // 編譯失敗
4	int e, var f = 3; // 編譯失敗, 非複合宣告
5	}

- 以下情形不是複合宣告，可以通過編譯：

1	public void notCompoundDeclaration1() {
2	int a; int b = 3;
3	int c; var d = 3;
4	}





END ~~

Thank you!!

Edited by Ruei-Jiun Tzeng

