

Java 程式設計進階

物件建構及使用

鄭安翔

ansel_cheng@hotmail.com

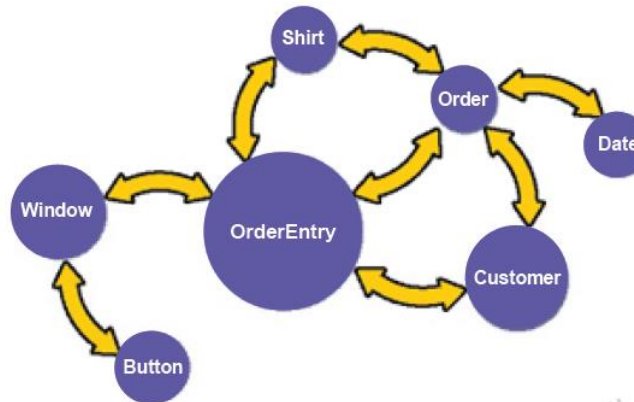
課程大綱

- 1) 物件建構、使用及移除
 - **Java** 應用程式靜態及動態觀點
 - 物件建構流程
 - 屬性 **vs.** 區域變數
- 2) **Java**資料型別
- 3) 陣列

靜態 vs. 動態觀點

■ 靜態觀點

- 一群組成應用程式的類別
- 程式設計師設計類別及類別之間的互動關係.



■ 動態觀點

- 執行時期,焦點放在類別的實體 - 物件身上
- 物件的建立、參考、使用

```
01 public class OrderEntry {  
02     public static void main (String[] args) {  
03         Order order = new Order();  
04         Shirt s1 = new Shirt(.....);  
05     }  
06 }
```

order:Order

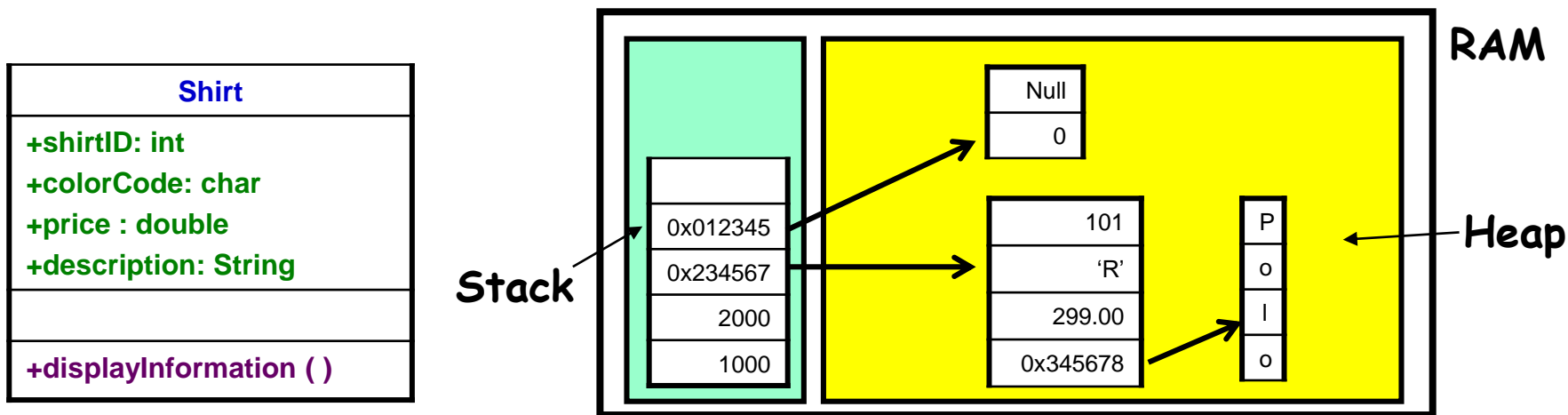
s1:Shirt



動態觀點

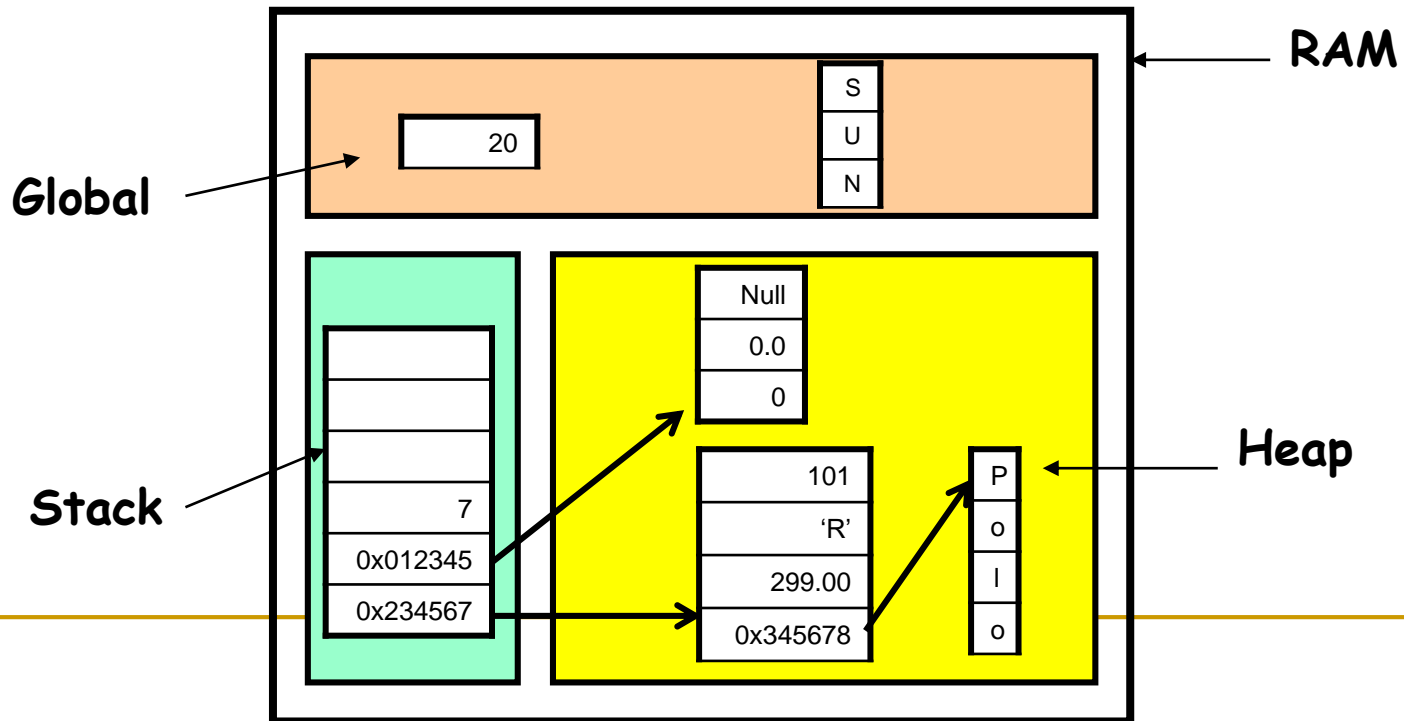
■ Java應用程式執行時期

- 物件建立：JVM 根據類別定義,在系統Heap記憶體中配置需要的儲存空間
- 物件參考：在Stack記憶體中儲存物件的記憶體位址
- 物件使用：處理物件之間互動,產生之狀態變化(記憶體中的儲存值的改變)



JVM記憶體配置

- Java Virtual Machine 將記憶體分為三大區塊
 - Global-儲存媒體：
 - 存放類別定義及類別成員變數



JVM記憶體配置

□ Stack-儲存媒體：

- 存放程式執行時期所參考之區域變數
- Stack 記憶體空間較小，搜尋存取內容速度快
- Stack 記憶體內的資料不會有初始值

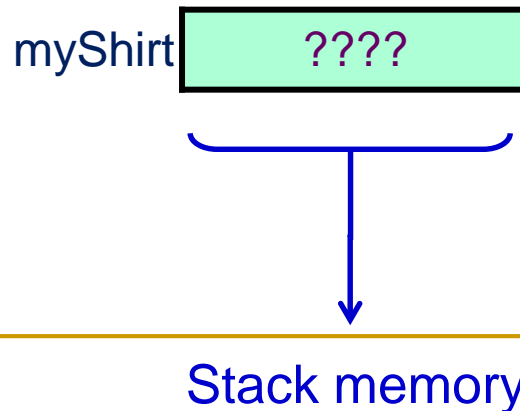
□ Heap-儲存媒體：

- 存放被宣告為參考資料型別 (Reference Type) 的物件實體
- Heap 記憶體空間較大，儲存的資料沒有一定限制，搜尋存取內容不易
- Heap 記憶體內的資料有預設初始值

物件建構流程 – 宣告

```
public class Shirt {  
  
    public int shirtID = 101;  
    public char colorCode = 'R';  
    public double price = 299.0;  
    public String description = "Polo Shirt";  
  
}
```

```
public class TestShirt {  
  
    public static void main(String[ ] args) {  
        Shirt myShirt;  
        myShirt = new Shirt( );  
    }  
}
```



物件建構流程 – 實體化

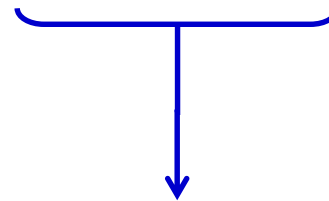
記憶體配置

```
public class Shirt {  
  
    public int shirtID = 101;  
    public char colorCode = 'R';  
    public double price = 299.0;  
    public String description = "Polo Shirt";  
  
}
```

```
public class TestShirt {  
  
    public static void main(String[ ] args) {  
        Shirt myShirt;  
        myShirt = new Shirt ( );  
    }  
}
```

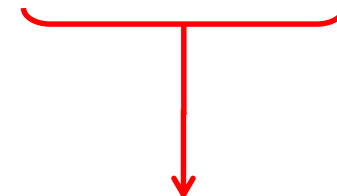
myShirt

????



Stack memory

0	shirtID
'R'	colorCode
0.0	price
NULL	description



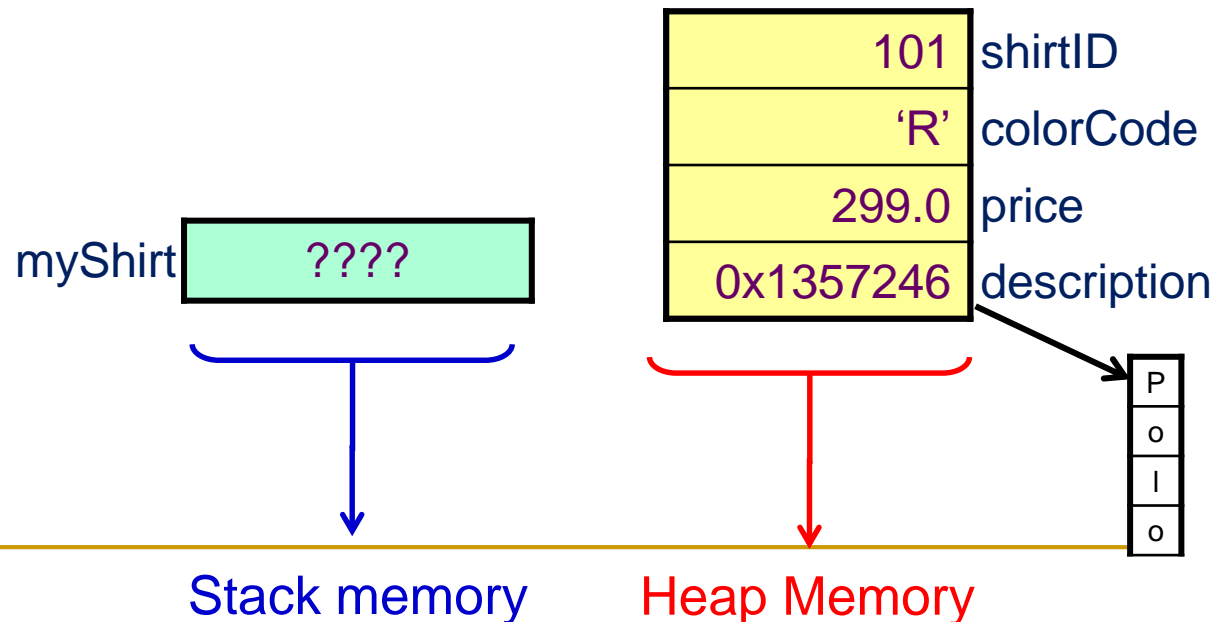
Heap Memory

物件建構流程 – 初始化

初始值賦值

```
public class Shirt {  
  
    public int shirtID = 101;  
    public char colorCode = 'R';  
    public double price = 299.0;  
    public String description = "Polo Shirt";  
  
}
```

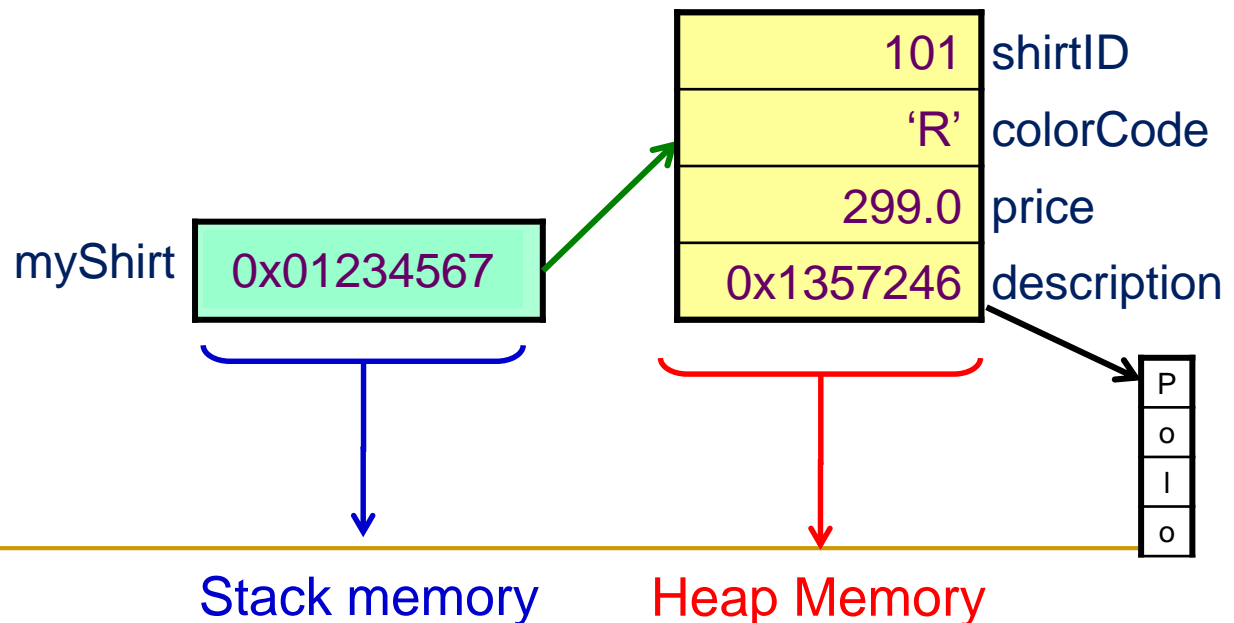
```
public class TestShirt {  
  
    public static void main(String[ ] args) {  
        Shirt myShirt;  
        myShirt = new Shirt ( );  
    }  
}
```



物件建構流程－儲存物件參考

```
public class Shirt {  
  
    public int shirtID = 101;  
    public char colorCode = 'R';  
    public double price = 299.0;  
    public String description = "Polo Shirt";  
  
}
```

```
public class TestShirt {  
  
    public static void main(String[ ] args) {  
        Shirt myShirt;  
        myShirt = new Shirt( );  
    }  
}
```

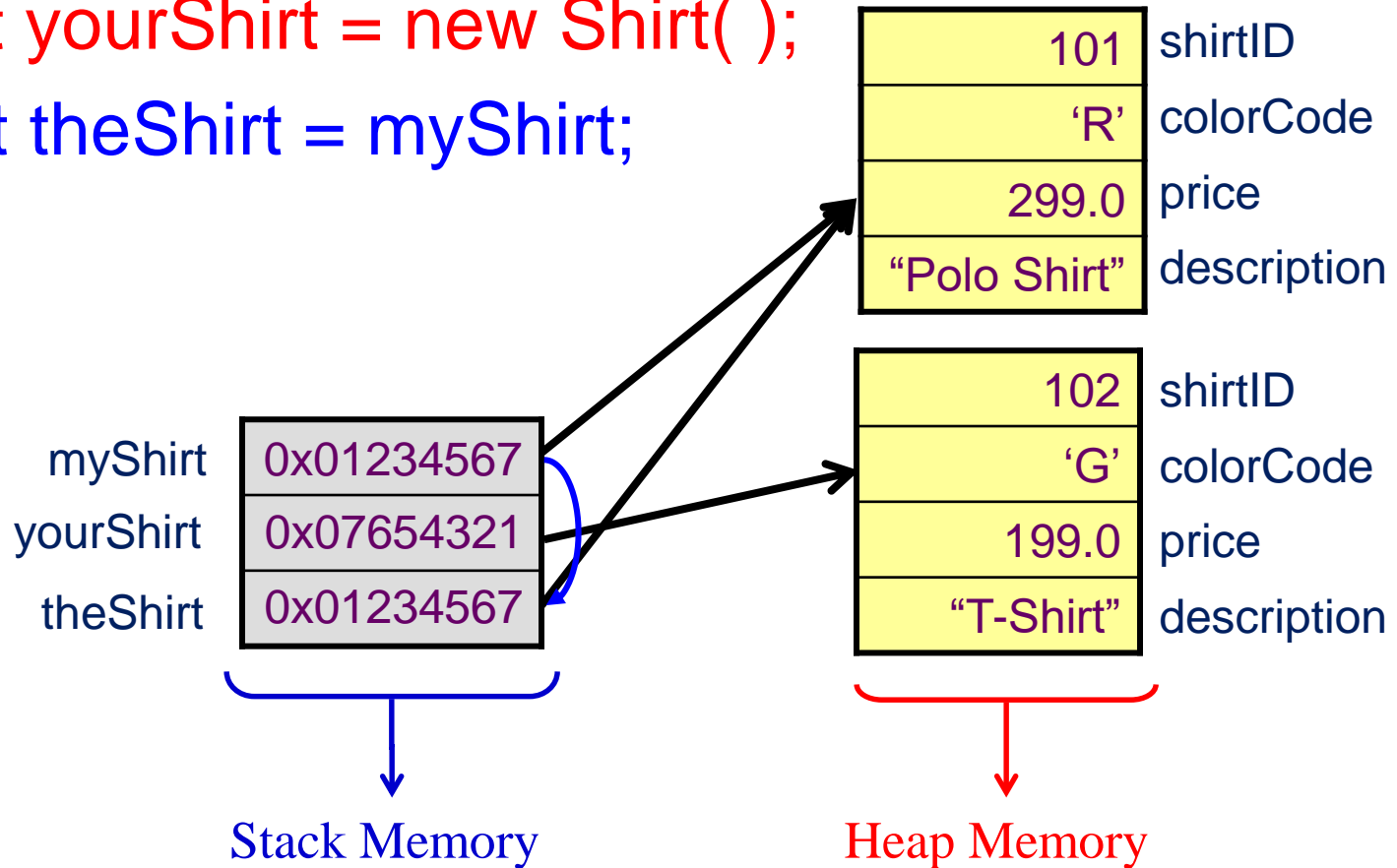


儲存物件參考

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

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

```
Shirt theShirt = myShirt;
```



物件互動操作

□ 物件參考.物件屬性

```
public class Shirt {  
  
    public int shirtID = 101;  
    public char colorCode = 'R';  
    public double price = 299.0;  
    public String description = "Polo Shirt";  
  
}
```

```
public class TestShirt {  
  
    public static void main(String[ ] args) {  
        Shirt myShirt = new Shirt( );  
        myShirt.colorCode = 'G';  
        System.out.println("Shirt ID: " + myShirt.shirtID);  
        System.out.println("Color Code: " + myShirt.colorCode);  
        System.out.println("Shirt Price: " + myShirt.price);  
        System.out.println("Description: " + myShirt.description);  
    }  
}
```

在螢幕上印
Shirt ID: 101
Color Code: G
Shirt Price: 299.0
Description: Polo

Stack
myShirt

0x01234

101	shirtID
'G'	colorCode
299.0	price
"Polo"	description

RAM

Heap

物件互動操作

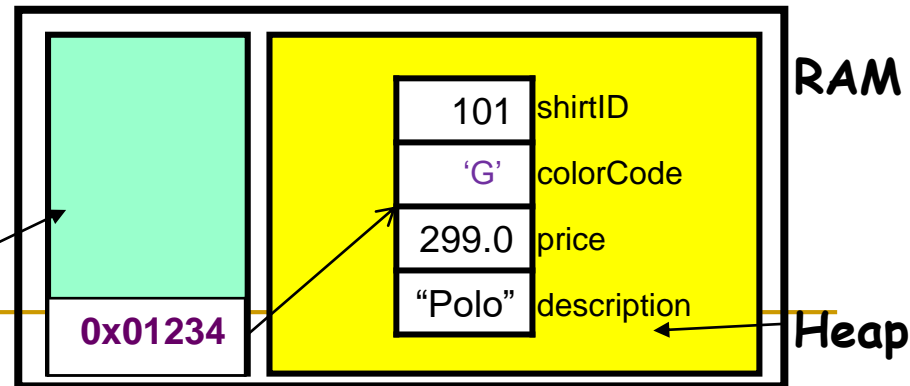
□ 物件參考.物件方法

```
public class Shirt {  
    public int shirtID = 101;  
    public char colorCode = 'R';  
    public double price = 299.0;  
    public String description = "Polo Shirt";  
  
    public void displayInformation() {  
        System.out.println("Shirt ID: " + shirtID);  
        System.out.println("Color Code: " + colorCode);  
        System.out.println("Shirt Price: " + price);  
        System.out.println("Description: " + description);  
    }  
}
```

```
public class TestShirt {  
  
    public static void main(String[] args) {  
        Shirt myShirt = new Shirt();  
        myShirt.colorCode = 'G';  
        myShirt.displayInformation();  
    }  
}
```

在螢幕上印
Shirt ID: 101
Color Code: G
Shirt Price: 299.0
Description: Polo

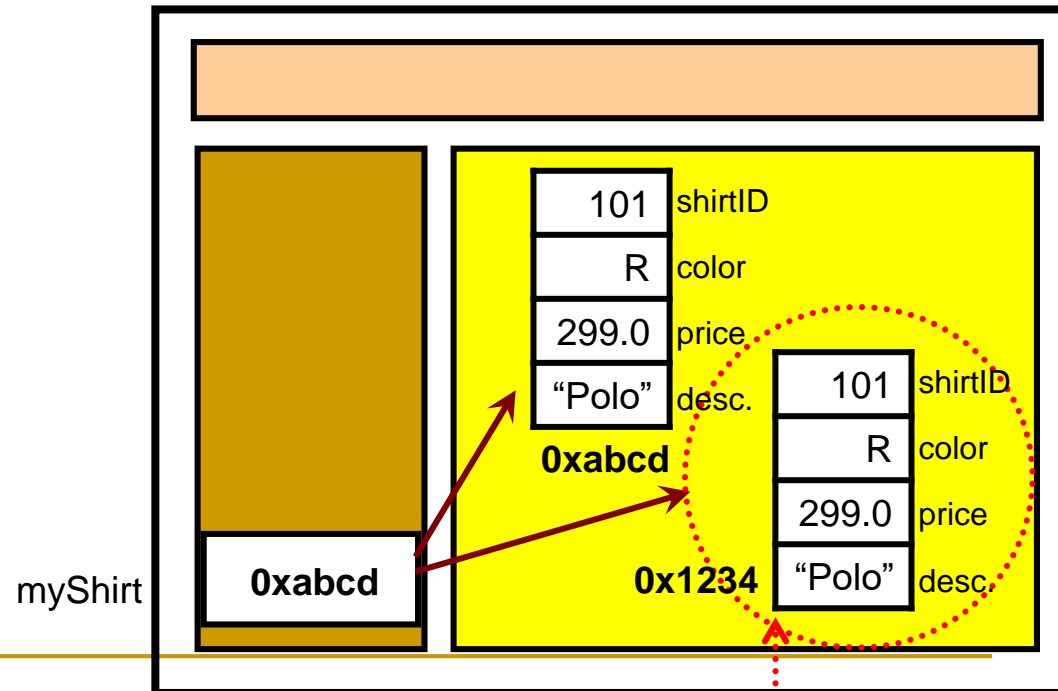
Stack
myShirt



物件移除

```
public class Shirt {  
  
    public int shirtID = 101;  
    public String description = "Polo Shirt";  
    public char colorCode = 'R';  
    public double price = 299.0;  
  
}
```

```
public class TestShirt {  
  
    public static void main(String[ ] args) {  
        Shirt myShirt = new Shirt( );  
        myShirt = new Shirt( );  
        System.gc();  
    }  
}
```



資源回收 Garbage Collection

課程大綱

- 1) 物件建構、使用及移除
- 2) **Java資料型別**
 - 基本資料型別 **vs.** 參考型別
 - 物件屬性與區域變數
- 3) 陣列

Java 資料型別

- Java的資料型別分為二種
 - 基本資料型別(Primitive Type)
 - 參考資料型別(Reference Type)
 - JDK類別函式庫 Library
 - 其他廠商開發之Java元件
 - 自行開發的類別
 - 二種資料型別在記憶體中實際儲存與用法上不同。

Java 基本資料型別

■ 基本資料型別之資料型態及範圍：

資料類別	資料型態	位元數(bits)	資料範圍 (Range)	範例	初始值
邏輯布林值	boolean	1	只能有 true 或 false	true, false	false
字元	char (Unicode)	16	'\u0000' ~ '\uFFFF' 0 ~ 65535	'A', 'x', '3', '中', '\n', '\u0063'	'\u0000'
整數	byte	8	-128 ~ 127	2, -114	0
	short	16	-32,768 ~ 32,767	2, -32699	0
	int	32	$-2^{31} \sim 2^{31}-1$	2, -147344778	0
	long	64	$-2^{63} \sim 2^{63}-1$	2, -2036854708L	0L
浮點數	float	32	$-3.4\text{E}+38 \sim 3.4\text{E}+38$	99F, -32699.01F	0.0F
	double	64	$-1.7\text{E}+308 \sim 1.7\text{E}+308$	-111, 21E12	0.0D

常用 JDK 類別函式庫

函式庫名稱	函式庫常用類別	用途
java.lang	Object, System, String, Enum, Integer, Double, Math, StringBuffer, Thread	Java語言的基本類別
java.util	Collections, Arrays, HashSet, ArrayList, TreeMap, Iterator, Properties, Locale, Calendar, Date, Scanner	工具類別
java.io	File, FileReader, FileWriter, InputStream, OutputStream	輸出輸入處理類別
java.text	DateFormat, DecimalFormat	文字格式處理類別
java.sql	ResultSet, Date, TimeStamp	結構化查詢語言(SQL)支援類別
java.math	BigDecimal, BigInteger	數學精算類別
java.net	Socket, URL, InetAddress	網路應用類別
javax.swing	JFrame, JPanel, JButton,	圖形化使用者介面類別
javax.xml.parsers	DocumentBuilder, SAXParser	可延伸標籤語言(XML)支援類別
javax.crypto	Cipher, KeyGenerator	資訊加密類別

不同資料型別之記憶體使用

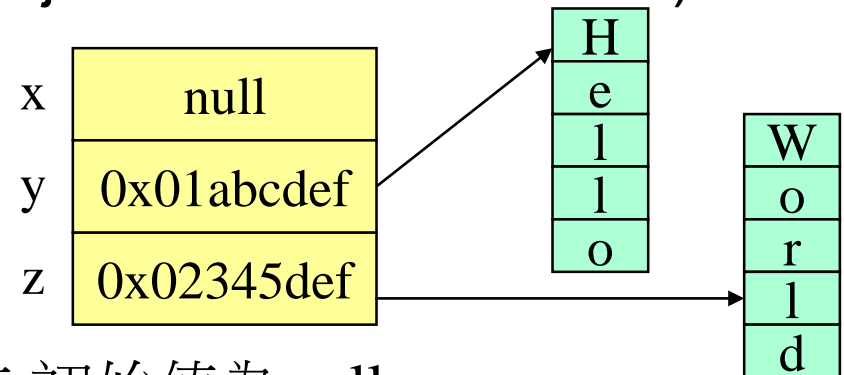
■ 基本資料型別(Primitive Type)記憶體使用

- 變數內直接儲存內容
- 宣告變數時即配置記憶體
 - 成員變數會自動填入初始值
 - 區域變數不會自動填入初始值

a	0
b	false
c	0.0

■ 參考資料型別記憶體使用

- 變數內存放物件內容參考(object reference variable)
- 使用步驟
 - 宣告
 - 實體化
 - 初始化
- 參考型別變數未經實體化前,初始值為null



物件屬性與區域變數

■ 物件屬性**Attribute**

- ❑ 變數宣告在class定義內,method區塊之外
- ❑ 也稱為member variable, instance variable
- ❑ 當物件實體化後,每一物件即各自擁有自己的變數資料,互不影響
- ❑ 如未定義初始值,會有default值 (int i; 此時i=0)
- ❑ 隨物件存放在Heap記憶體中

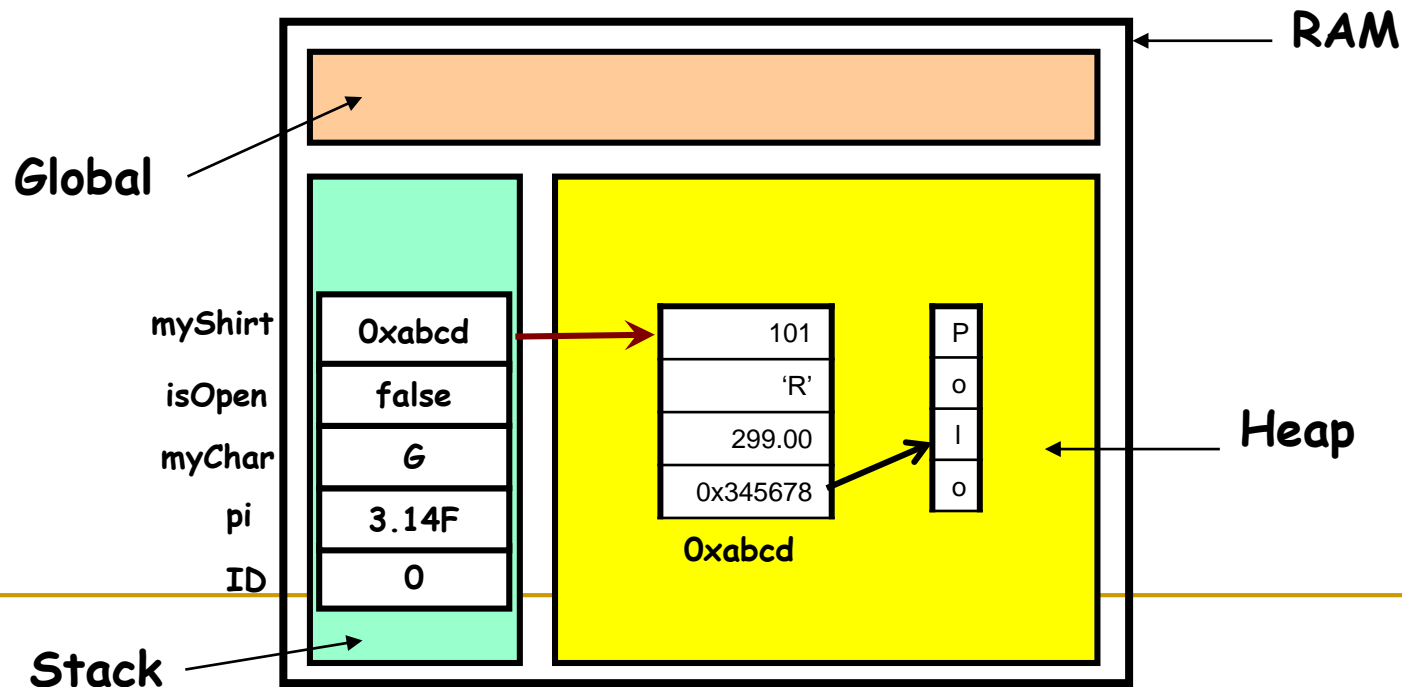
■ 區域變數**Local variable**

- ❑ 變數宣告method區塊之內
- ❑ 區域變數只在宣告的區段{ }中有效
- ❑ 宣告時不使用modifier(final例外)
- ❑ 必須自訂初始值
- ❑ 存放在Stack記憶體中

物件屬性與區域變數

```
01 public class TestVariable {  
02     public static void main(String[] args) {  
03         int ID = 0;  
04         float pi = 3.14F;  
05         char myChar = 'G';  
06         boolean isOpen = false;  
07         Shirt myShirt = new Shirt();  
08     }  
09 }
```

```
01 public class Shirt {  
02     public int shirtID;  
03     public char colorCode = 'R';  
04     public double price = 299.0;  
05     public String description = "Polo Shirt";  
06  
07     .....  
08 }
```



Var 型態推斷

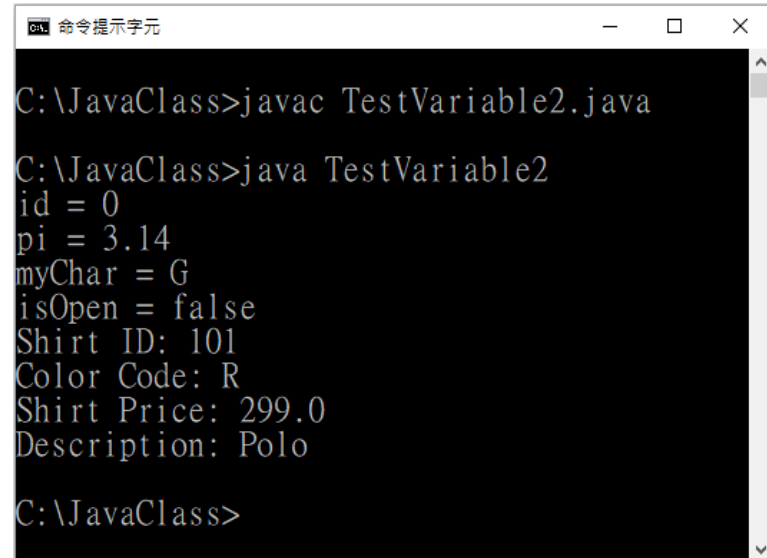
- 區域變數的型態可由從前後程式碼推斷

var age = 10;	// age 型態為 int
var pi = 3.14;	// pi 型態為 double
var largeNum = 10000000000L;	// largeNum 型態為 long
var floatPi = 3.14159F;	// floatPi 型態為 float
var color = 'R';	// color 型態為 char
var success = true;	// success 型態為 boolean
var shirt = new Shirt();	// shirt 型態為 Shirt

Var 型態推斷

```
01 public class Shirt {  
02     public int shirtID;  
03     public char colorCode = 'R';  
04     public double price = 299.0;  
05     public String description = "Polo Shirt";  
06  
07     .....  
08 }
```

```
01 public class TestVariable2 {  
02     public static void main(String[] args) {  
03         var id = 0;  
04         var pi = 3.14F;  
05         var myChar = 'G';  
06         var isOpen = false;  
07         var myShirt = new Shirt();  
08  
09         System.out.println("id = "+id);  
10         System.out.println("pi = "+pi);  
11         System.out.println("myChar = "+myChar);  
12         System.out.println("isOpen = "+isOpen);  
13         myShirt.displayInformation();  
14     }
```



```
C:\JavaClass>javac TestVariable2.java  
  
C:\JavaClass>java TestVariable2  
id = 0  
pi = 3.14  
myChar = G  
isOpen = false  
Shirt ID: 101  
Color Code: R  
Shirt Price: 299.0  
Description: Polo  
  
C:\JavaClass>
```

課程大綱

- 1) 物件建構、使用及移除
- 2) Java資料型別
- 3) 陣列
 - 一維陣列
 - 多維度陣列
 - 陣列特性

陣列

■ 陣列

- 將同樣型別的東西聚集在一起，用相同名稱來參考一群物件。
- 參考型別物件

■ 使用步驟：

- 宣告
- 實體化
- 初始化

一維陣列 – 宣告

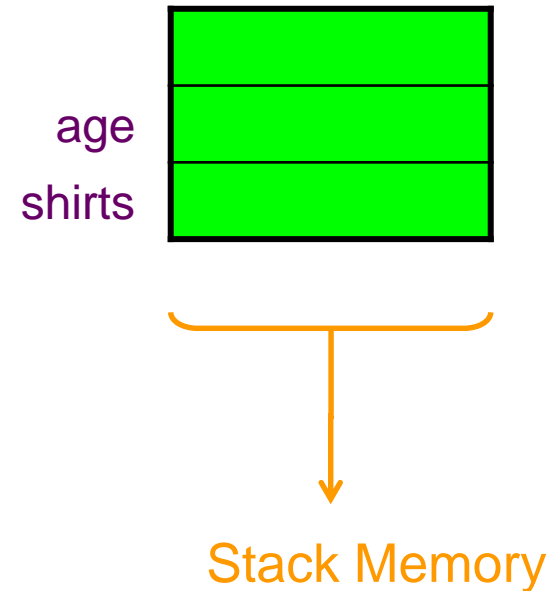
- 語法：

`type [] array_name;`

`type array_name [];`

Ex: `int [] age;`

`Shirt shirts[];`



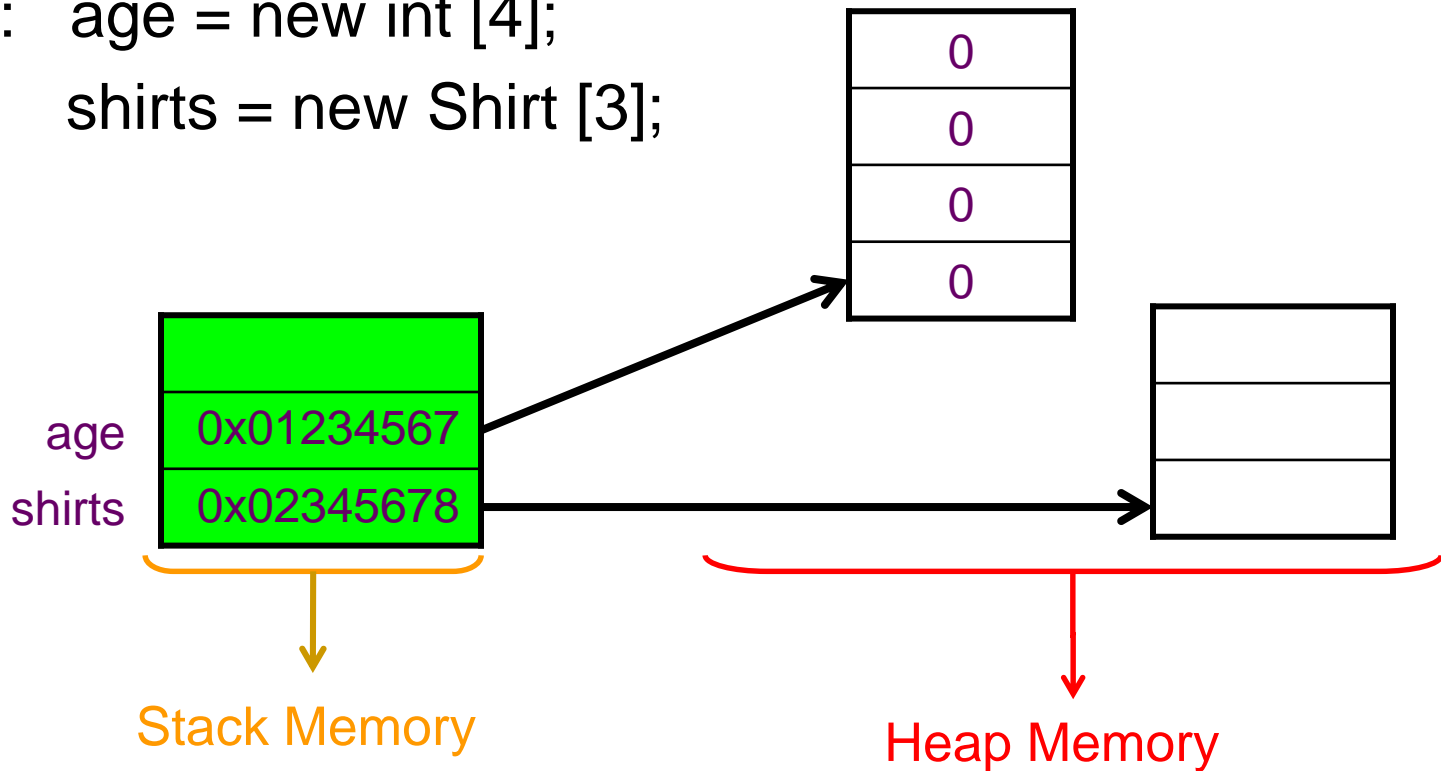
一維陣列 – 實體化

■ 語法：

`array_name = new type[length];`

Ex: `age = new int [4];`

`shirts = new Shirt [3];`



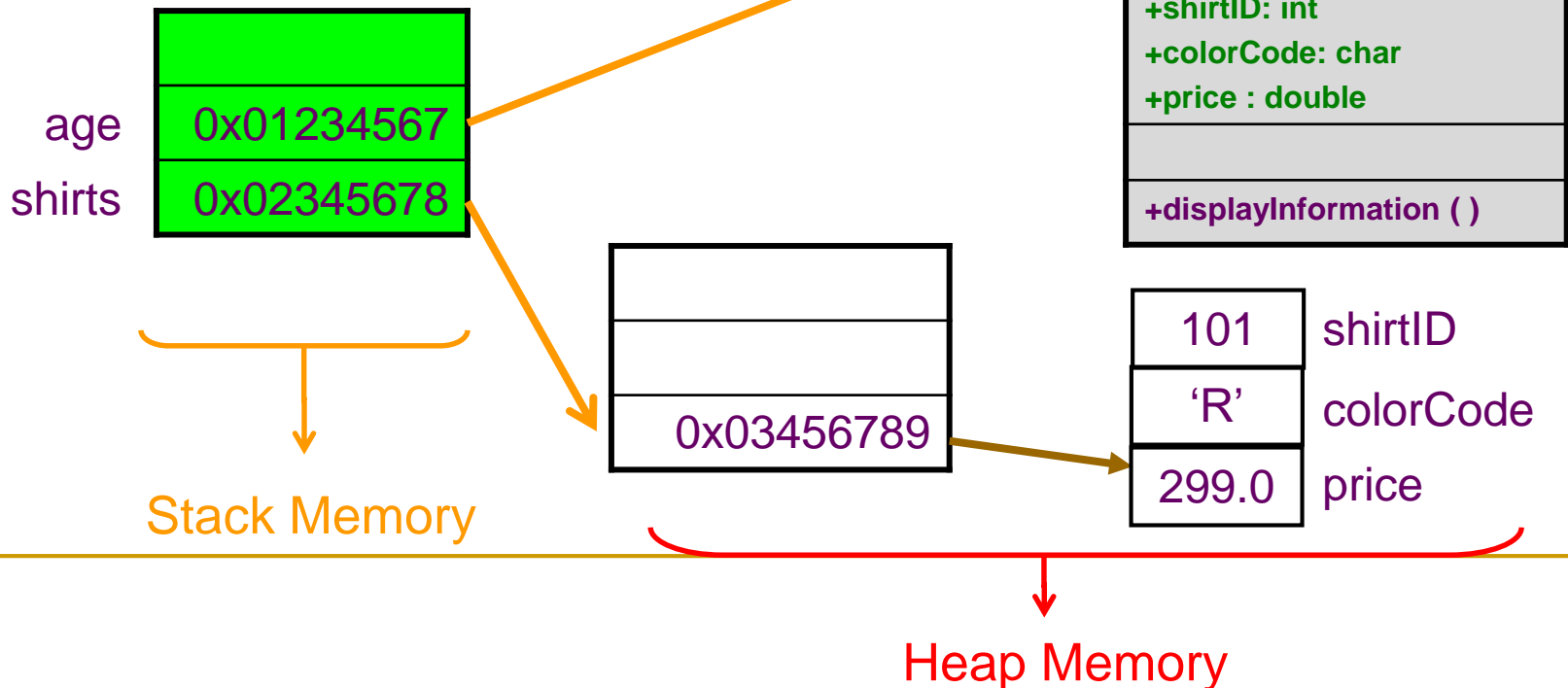
一維陣列 – 初始化

■ 語法：

`array_name [index] = value;`

Ex: `age [2] = 3;`

`shirts [0] = new Shirt();`



一維陣列的建構方式

宣告	<pre>int[] i; Shirt[] s;</pre>	<pre>int[] i = new int[3]; Shirt[] p = new Shirt[3];</pre>	<pre>int[] i = new int[] {1, 3, 5}; int[] i = {1, 3, 5}; Shirt[] s = new Shirt[] { new Shirt(), new Shirt(), new Shirt() } Shirt[] s = { new Shirt(), new Shirt(), new Shirt() }</pre>
實體化	<pre>i = new int[3]; s = new Shirt[3];</pre>		
初始化	<pre>i[0] = 1; i[1] = 3; i[2] = 5; s[0] = new Shirt(); s[1] = new Shirt(); s[2] = new Shirt();</pre>	<pre>i[0] = 1; i[1] = 3; i[2] = 5; s[0] = new Shirt(); s[1] = new Shirt(); s[2] = new Shirt();</pre>	

二維陣列

	Quarter1	Quarter2	Quarter3	Quarter4
<i>Year1</i>				
<i>Year2</i>				
<i>Year3</i>				
<i>Year4</i>				
<i>Year5</i>				

二維陣列 – 宣告

■ 語法：

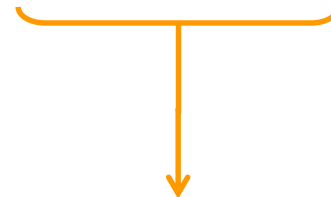
```
type [ ][ ] array_name;
```

```
type array_name [ ][ ];
```

Ex:

```
int [ ][ ] quarterSales;
```

quarterSales



Stack Memory

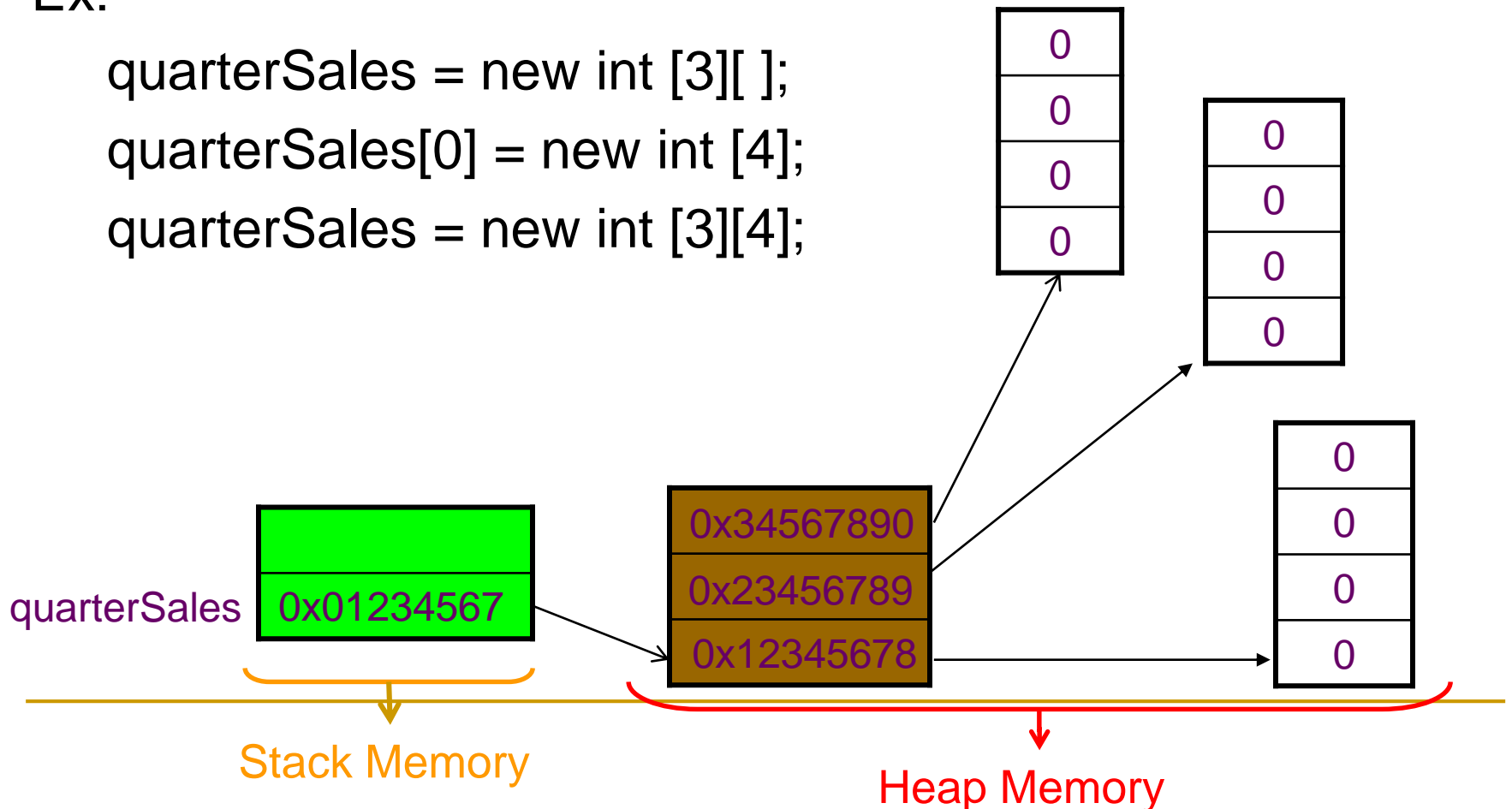
二維陣列 – 實體化

■ 語法：

`array_name = new type [number_of_arrays][length];`

Ex:

```
quarterSales = new int [3][ ];  
quarterSales[0] = new int [4];  
quarterSales = new int [3][4];
```



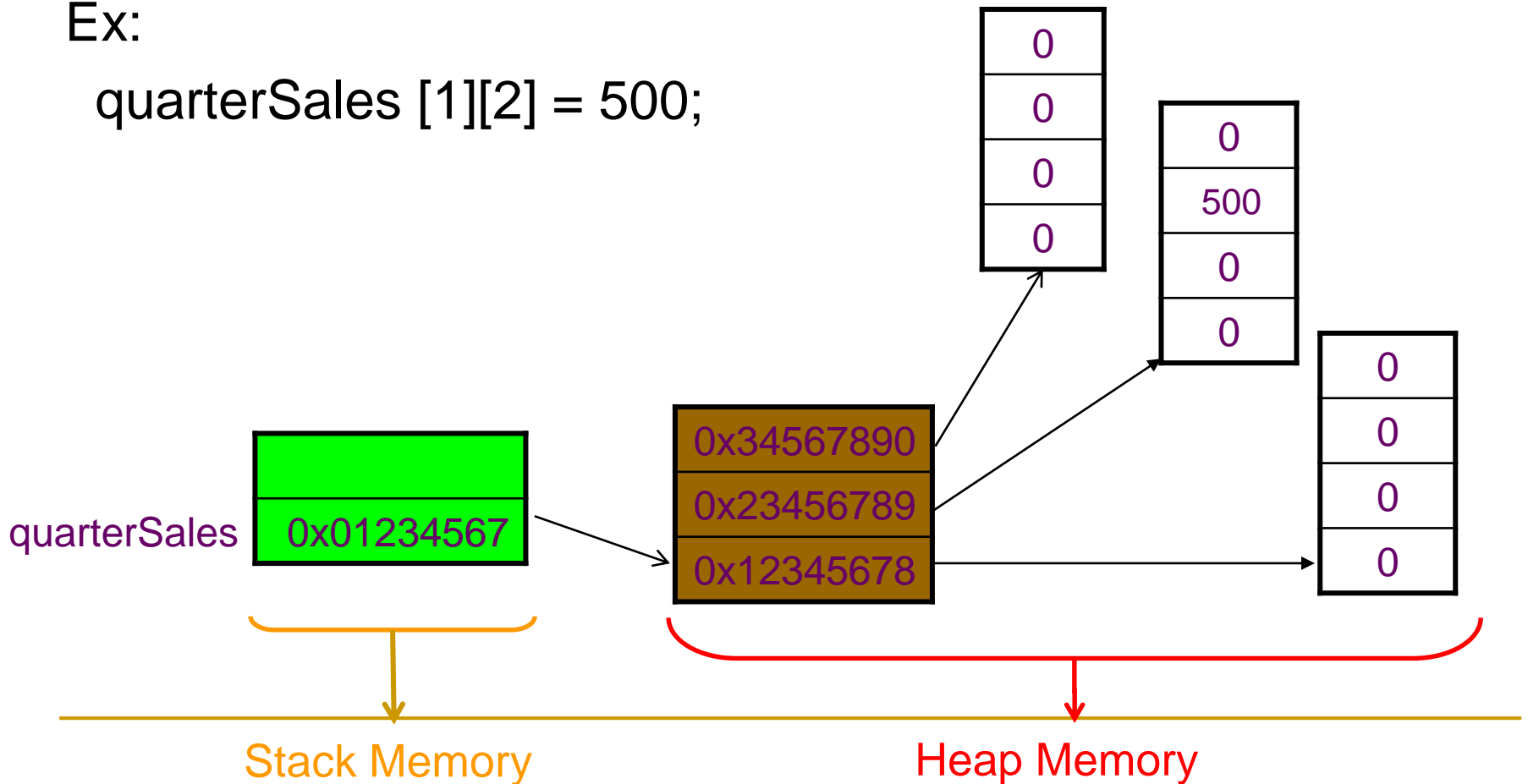
二維陣列 – 初始化

■ 語法：

`array_name` [`number_of_arrays`] [`index`] = `value`;

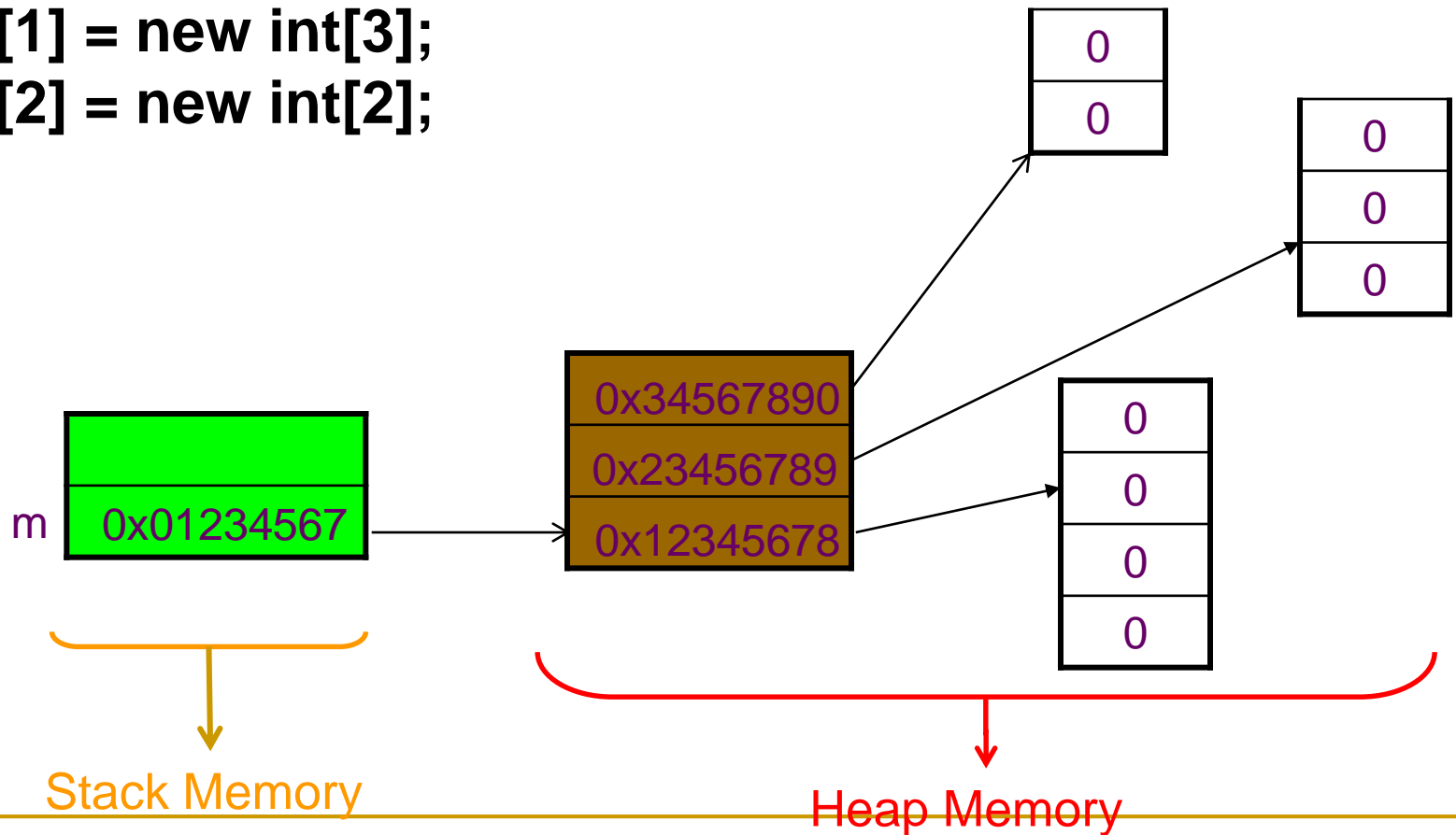
Ex:

`quarterSales [1][2] = 500;`



非對稱型陣列 - Non-Rectangular

```
int m[ ][ ]= new int[3][ ];  
m[0] = new int[4];  
m[1] = new int[3];  
m[2] = new int[2];
```



多維陣列建構

- 多維陣列的初始化內容宣告：
 - `int m[][] = { {1,2}, {1,2,3} };`
 - 分別以括號（{ }）來表示不同階層的陣列
- 陣列的宣告是從左至右的方式來初始陣列大小：
 - `int m[][] = new int[3][];` 正確
 - `int m[][] = new int[][3];`
此為錯誤的宣告方式，會產生編譯錯誤(Compile error)。

陣列大小

■ 取得陣列的大小

□ 陣列物件的length屬性(Attribute)

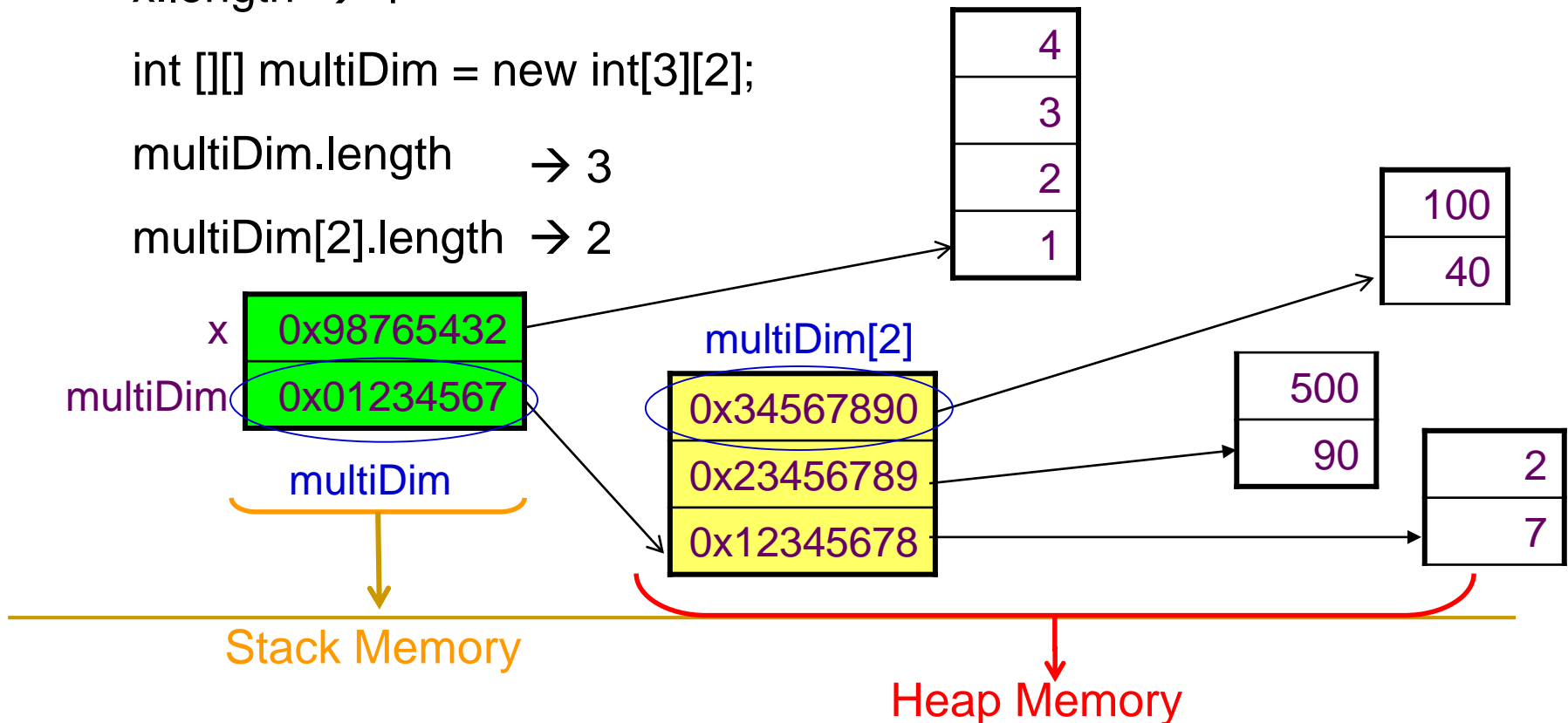
```
int [] x = new int[4];
```

x.length → 4

```
int [][] multiDim = new int[3][2];
```

multiDim.length → 3

multiDim[2].length → 2



陣列邊界 Array Bounds

- 陣列索引值

- 由 0 開始到 `length-1`
- 存取陣列的索引值超過邊界範圍，會有執行時期錯誤：`ArrayIndexOutOfBoundsException`
- 依序存取陣列元素

```
for (int i = 0; i < list.length; i++) {  
    System.out.println(list[i]);  
}
```

陣列長度不可改變

- 在**Java**中，陣列不被允許重新定義大小(**Resize**)
- 要改變陣列大小必須重新創建陣列
 - 物件參考值會指向新創建出來的物件陣列實體

```
int [ ] x = {1,2,3};
```

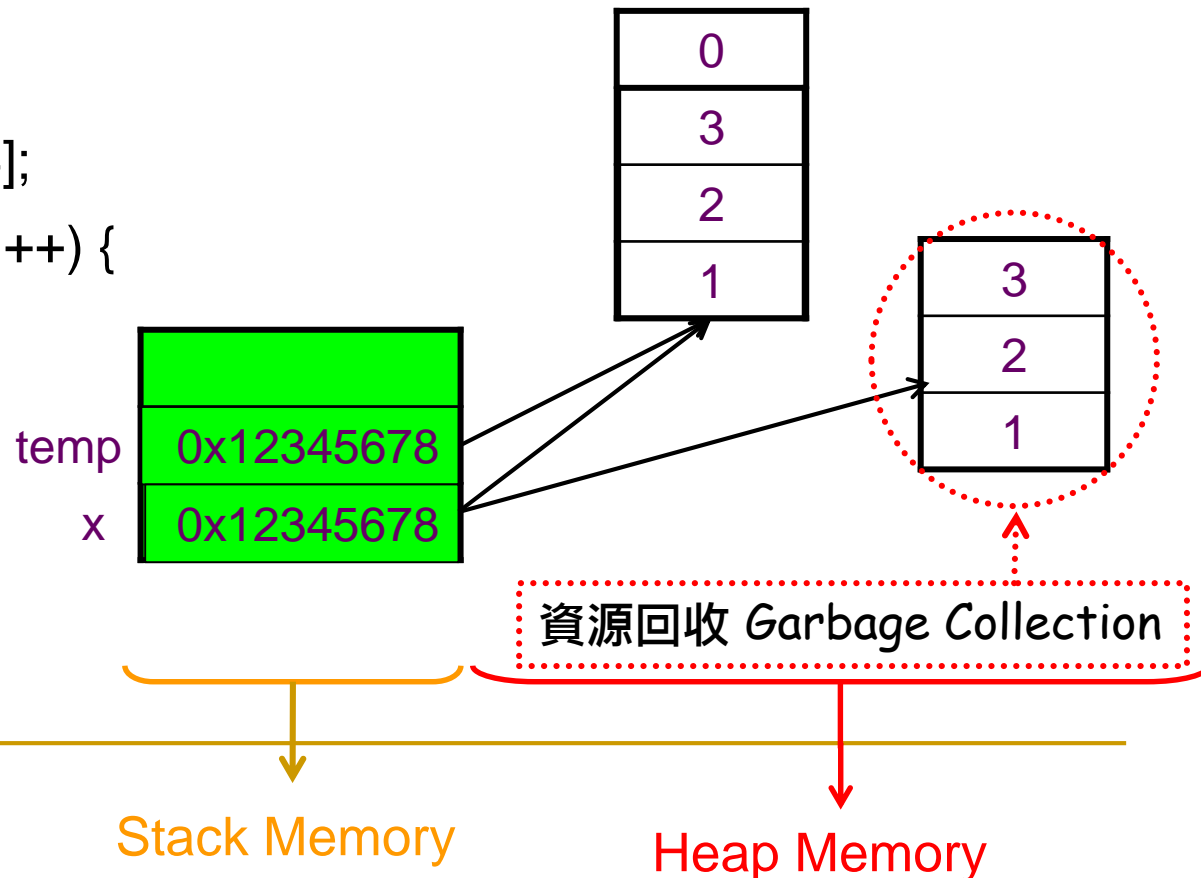
```
int [ ] temp = new int[4];
```

```
for(int i=0; i<x.length; i++) {
```

```
    temp[i] = x[i];
```

```
}
```

```
x = temp ;
```

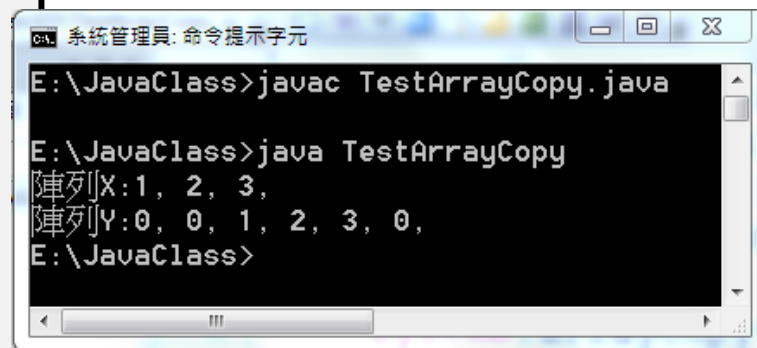


陣列複製

■ System類別提供陣列複製的方法

方法名稱	修飾字/ 傳回值	說明
arraycopy(Object src, int srcPos, Object dest, int destPos, int length)	static void	src:來源陣列, srcPos:複製來源陣列的起始位置 dest:目標陣列, srcPos:複製目標陣列的起始位置 length: 複製陣列元素個數

```
01 public class TestArrayCopy {
02     public static void main(String[] args) {
03         int[] x = {1,2,3};
04         int[] y = new int[6];
05
06         System.arraycopy(x, 0, y, 2, x.length);
07         System.out.print("陣列X:");
08         for(int i=0; i<x.length; i++){
09             System.out.print(x[i]+"", " ");
10         }
11         System.out.println();
12         System.out.print("陣列Y:");
13         for(int a : y){
14             System.out.print(a+"", " ");
15         }
16     }
17 }
```



```
C:\>系統管理員: 命令提示字元
E:\JavaClass>javac TestArrayCopy.java
E:\JavaClass>java TestArrayCopy
陣列X:1, 2, 3,
陣列Y:0, 0, 1, 2, 3, 0,
E:\JavaClass>
```

for-each 迴圈

■ for-each 迴圈 (for-each)

- JavaSE 5.0所新增的語法。
- 簡化存取集合性物件的元素內容。
 - 集合性物件：陣列(array)或集合(collection)
- 執行時自動找下一個元素，直到全部擷取完畢為
- 由 for-loop 實作出來的
- 讀取集合元素時不可修改元素內容

■ for-each 語法

for (元素資料型別 區域變數名稱 : 元素集合名稱)

for-each 迴圈

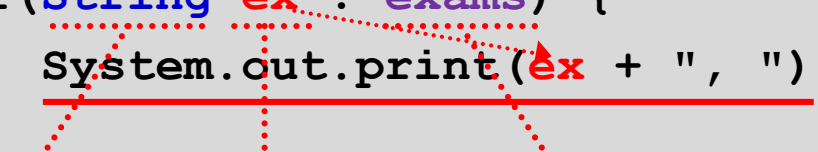
■ for- loop

```
01 String[] exams = {"SCJP", "SCWCD", "SCMAD"};  
02 for(int i=0;i<exams.length;i++) {  
03     System.out.print(exams[i] + ", ");  
04 }
```

執行結果：SCJP, SCWCD, SCMAD,

■ for-each

```
01 String[] exams = {"SCJP", "SCWCD", "SCMAD"};  
02 for(String ex : exams) {  
03     System.out.print(ex + ", ");  
04 }
```



元素資料型別 區域變數名稱 元素集合名稱

執行結果：SCJP, SCWCD, SCMAD,

for-each 迴圈

■ 分析多維陣列 - 使用 for-each

```
01 String[][] exams = {  
02     {"SCJP", "SCWCD", "SCMAD"},  
03     {"MCSD", "MCAD", "MCDBA"}  
04 };  
05 for(String[] ex : exams) {  
06     for(String e : ex) {  
07         System.out.print(e + ", ");  
08     }  
09     System.out.println();  
10 }
```

執行結果：SCJP, SCWCD, SCMAD,
MCSD, MCAD, MCDBA,