Method、Constructor & Static

方法 、建構式與靜態型態

Java Fundamental

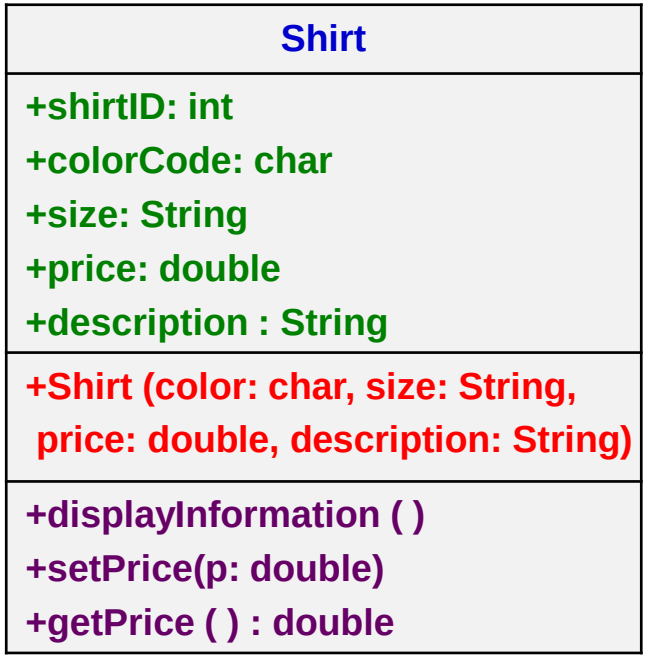
Outline

- 方法
- 建構式
- 類別成員

Outline

- 方法
- 建構式
- 類別成員

類別與方法

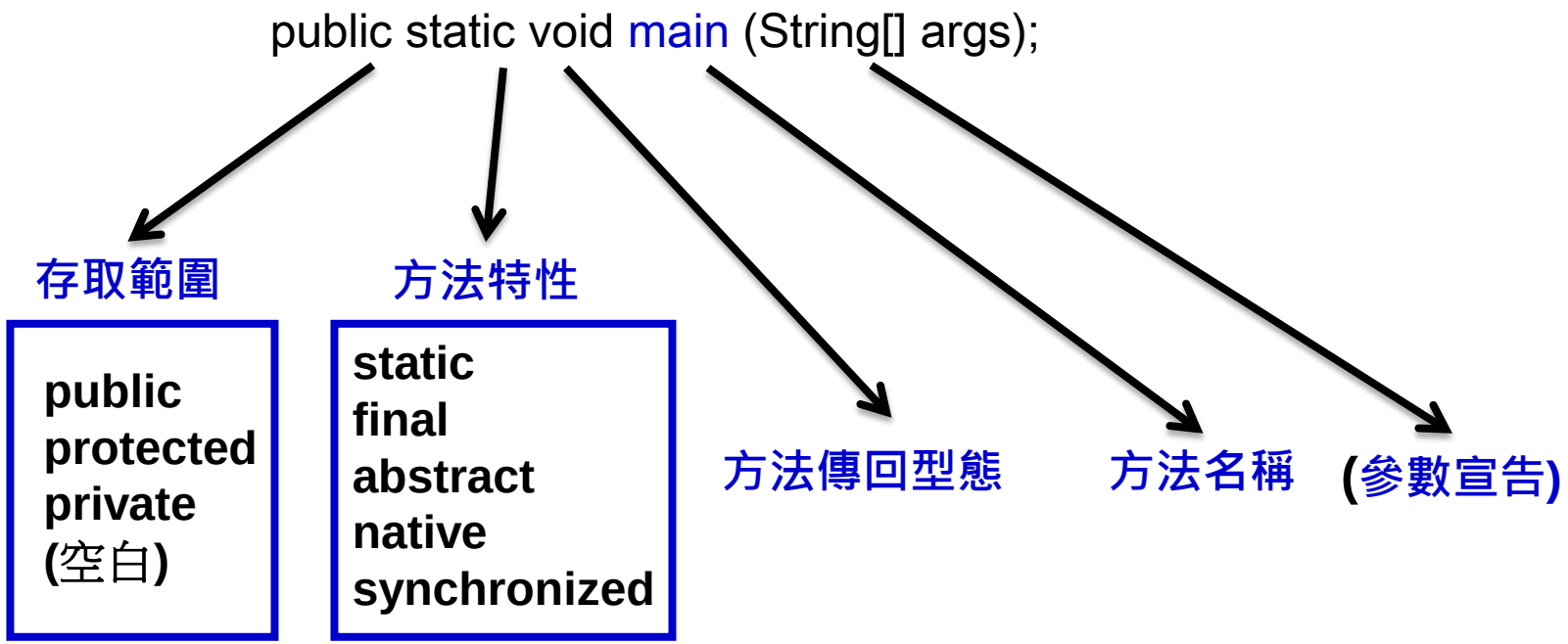



```
01  public class Shirt {
02
03      public int shirtID = 0;
04      public char colorCode = 'R';
05      public String size = "XL" ;
06      public double price = 299.00;
07      public String description = "Polo Shirt";
08
09      public Shirt(char color, String size,
10                   double price, String desc) {
11          this.colorCode = color;
12          this.size = size;
13          this.price = price;
14          this.description = desc;
15      }
16
17      public void displayInformation() {
18          System.out.println("Shirt ID:" + shirtID);
19          System.out.println("Color:" + colorCode);
20          System.out.println("Size:" + size);
21          System.out.println("Price:" + price);
22      }
23      public void setPrice(double p) {
24          price = p;
25      }
26      public double getPrice( ) {
27          return price;
28      }
29
30  }
```

物件方法
(操作)

方法 (Method)

- 定義
  - 可重複使用的程式碼片段
- 格式



Method

- Method 命名習慣
  - 應該採用英文動詞，可以串接多個單字：整個字串的第一個字母小寫，其他串接單字的第一個字母大寫。
    - 如：toString(), compareTo(), setX(), getX()…
  - 方法(Method)的名稱不能和類別(class)名稱一樣，因類別名稱是保留給建構式使用。
- 參數 – 將資料傳入method
  - Java 的基本及參考資料型態都可以傳入參數，例如： int、double、boolean、String或是物件。
  - Java 不允許將方法當成參數，傳入另一個方法中。

Method

- 在 Java 中，方法只能在類別中被創造
- 方法分成2個部分：
  - 方法簽章
  - 方法內容

```
void setPrice(double d)      //方法簽章

{d *= 0.9;}                  //方法內容
```

Method

- 方法的呼叫
  - 透過物件呼叫(instance methods)
    - 必須使用正確的方法名稱、參數個數與型態，例如：str.substring()
  - 透過類別呼叫(static methods or class methods)
    - 如：Math.power()
- 呼叫方法時會比對方法簽章的3個部分：
  - 名稱
  - 參數個數
  - 參數資料類型

```
void setPrice(double d)      //方法簽章
{d *= 0.9;}                  //方法內容
```

Method

◆ 參數傳遞(Passing Argument)

➢ 參數名稱

- 參數名稱在方法中用來指所傳入的變數
- 參數名稱可與類別中的成員變數名稱相同，此時成員變數將被隱藏(hide)，也就是說，在方法中所指稱的乃所傳入之變數，此時可用 this 來指定成員變數

➢ 參數傳遞方式

- 傳值 Pass-by-Value
  - 傳入參數為基本資料(primitive)型態，**對參數的改變不影響原來method 外變數的值**
- 傳址 Pass-by-Reference
  - 傳入參數為參考資料(reference)型態，無法更動參考對象(stack內容)，但可使用物件的方法與變數(屬性)

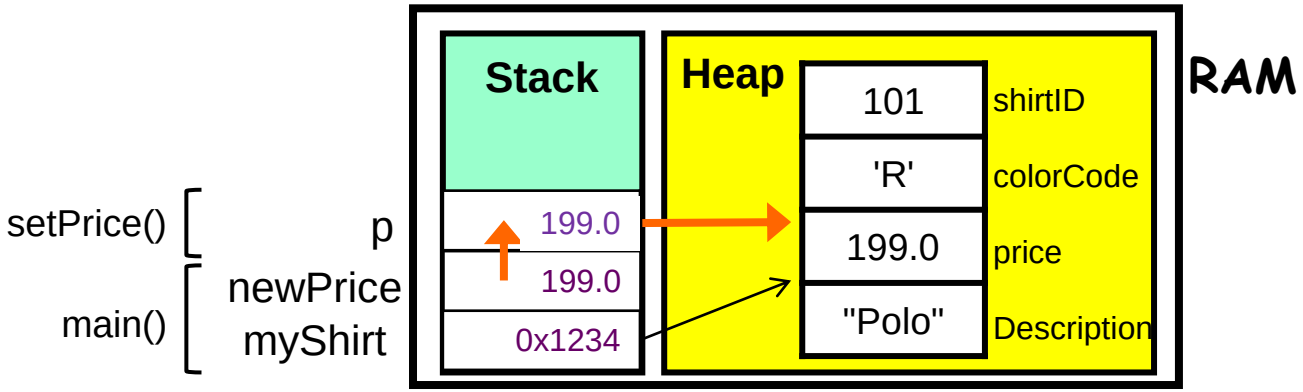
Pass by Value (傳值)

- Java 中參數的指派是傳遞目前 stack 中的內容
  - primitive type - 內容值

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

```
public class TestShirt {

  public static void main(String[ ] args) {
      Shirt myShirt = new Shirt( );
      double newPrice = 199.0;
      myShirt.setPrice(newPrice);
      ..........
  }
}
```



Method

```
static void myMethod1 (int i)
{
    i += 1;
    System.out.println("In MyMethod(): " + i);
}


public static void main(String[] args)
{
    int i = 10;
    // 參數以 primitive 的型態傳入
    myMethod1(i);
    // 不會改變在 main 中的 i 值
    System.out.println("In main(): " + i);
}
```

Pass by Value (傳址)

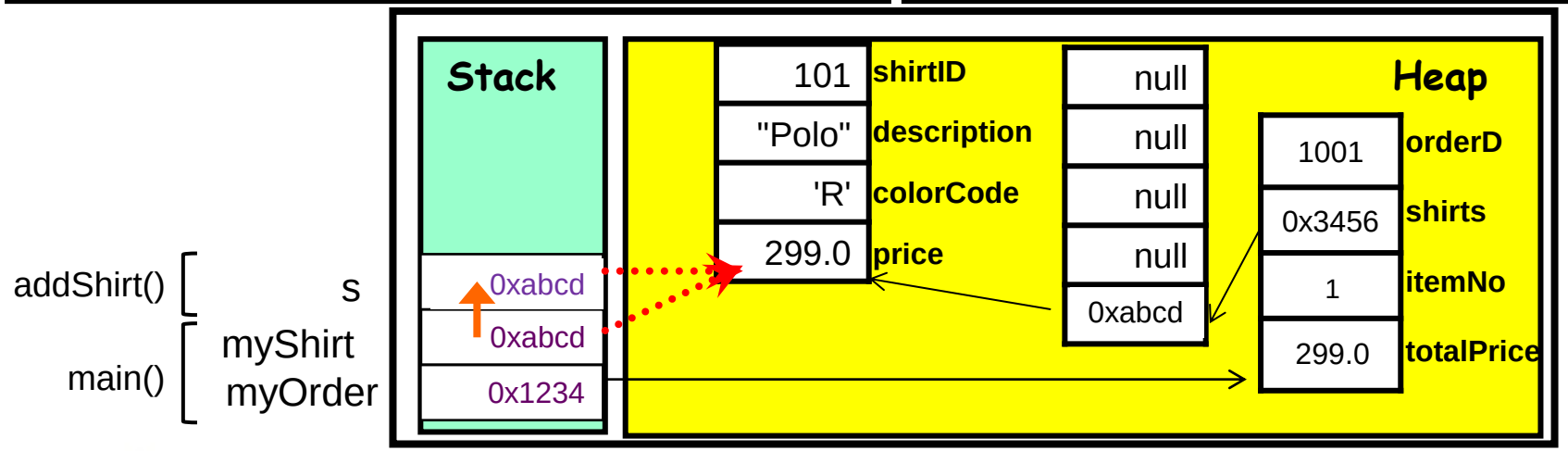
- Java 中參數的指派是傳遞目前 stack 中的內容
  - reference type - 參考值(位址)

```
public class Order {
   public int orderID = 1001;
   public Shirt[ ] shirts = new Shirt[5];
   public int itemNo = 0;
   public double totalPrice = 0.0;
   public void addShirt(Shirt s) {
      shirts[itemNo++] = s;
      totalPrice += s.price;
   }
}
```

```
public class TestOrder {

   public static void main(String[ ] args) {
      Order myOrder = new Order( );
      Shirt myShirt = new Shirt();

      myOrder.addShirt(myShirt);
      ..........
   }
}
```



Method

```
static void myMethod2 (int[] i)
{
    i[0] = i[0] +1;
    System.out.println("In MyFunc(): " + i[0]);
}

public static void main(String[] args)
{
     int[] i = {10};
     // 參數以 reference 的型態傳入
     myMethod2(i);
     // 會改變在 main 中的 i 值
     System.out.println("In main(): " + i[0]);
}
```

Method

- return
  - 宣告method時可設定回傳型態，在method主體中，必須使用 return 敘述，回傳適當的數值
  - 宣告為 void 的method，不能包含任何return敘述
  - 回傳的數值的資料型態，必須要符合method宣告設定的資料型態一致

Method

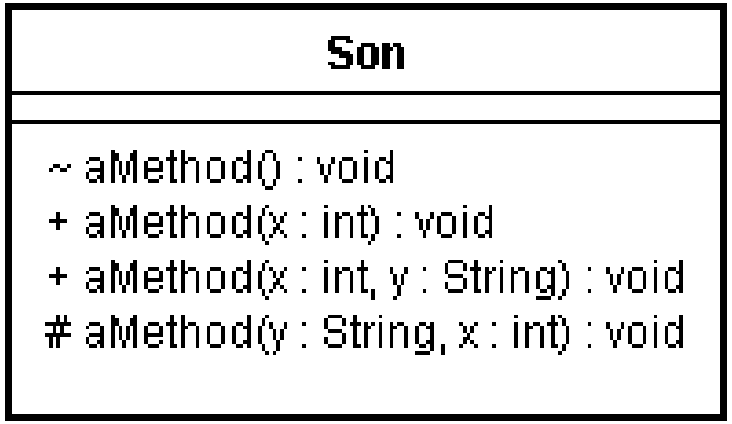
- Example: PassArgDemo.java

```
static int myMethod3 (int i)
{
    i *= 10;
    return i;
}

public static void main(String[] args)
{
    int i = 10;
    int j = myMethod3(i);
    System.out.println("The return value: " + j);
}
```

方法多載 (method overloading)

◆**同名方法名稱**根據其**不同的參數型別**以對應執行到不同的實作(內容)。




```
01 public class Son {
02     void aMethod() { }
03     public void aMethod(int x) { }
04     public void aMethod(int x, String y) { }
05     protected void aMethod(String y, int x) { }
06 }
```

方法多載 (method overloading)

```
public class Calculator {
   public int sum(int numberOne, int numberTwo){
     System.out.println("Method One");
     return numberOne + numberTwo;
   }
  public float sum(float numberOne, float numberTwo)
{
     System.out.println("Method Two");
     return numberOne + numberTwo;
  }
  public float sum(int numberOne, float numberTwo) {
    System.out.println("Method Three");
    return numberOne + numberTwo;
  }
}
```

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

    Calculator myCalculator = new Calculator();

    int totalOne = myCalculator.sum(2,3);
    System.out.println(totalOne);

    float totalTwo = myCalculator.sum(15.99F, 12.85F);
    System.out.println(totalTwo);

    float totalThree = myCalculator.sum(2, 12.85F);
    System.out.println(totalThree);
  }
}
```

Method

- 不定參數的處理
  - 如果無法確定呼叫方法時會傳遞幾個參數，就無法確切定義這個方法的參數個數，可以使用陣列來解決這種問題。

```
1.  class Varargs01{
2.      void showName(String[] names){
3.          System.out.print("購買的書籍：");
4.          for(String name : names)
5.              System.out.print(name + " ");
6.          System.out.println();
7.      }
8.
9.      public static void main(String[] args){
10.         Varargs01 var = new Varargs01();
11.         String[] twoNames = {"Java 程式設計", "JSP 程式設計"};
12.         var.showName(twoNames);
13.         String[] threeNames = {"C++概論", "資料庫概論", "網路概論"};
14.         var.showName(threeNames);
15.     }
16. }
```

------輸出------

購買的書籍：Java程式設計JSP程式設計

購買的書籍：C++概論資料庫概論網路概論

Method

- 參數列表(Varargs)
  - 專門用來處理參數個數不定的情形，比陣列更直覺。不過編譯器還是會將參數列表視為陣列。

```
1.  class Varargs02{
2.      void showName(String... names){
3.          System.out.print("購買的書籍：");
4.          for(String name : names)
5.              System.out.print(name + " ");
6.          System.out.println();
7.      }
8.
9.      public static void main(String[] args){
10.         Varargs02 var = new Varargs02();
11.         var.showName("Java 程式設計", "JSP 程式設計");
12.         var.showName("C++概論", "資料庫概論", "網路概論");
13.     }
14. }
```

```
------輸出------
購買的書籍：Java程式設計JSP程式設計
購買的書籍：C++概論資料庫概論網路概論
```

Method

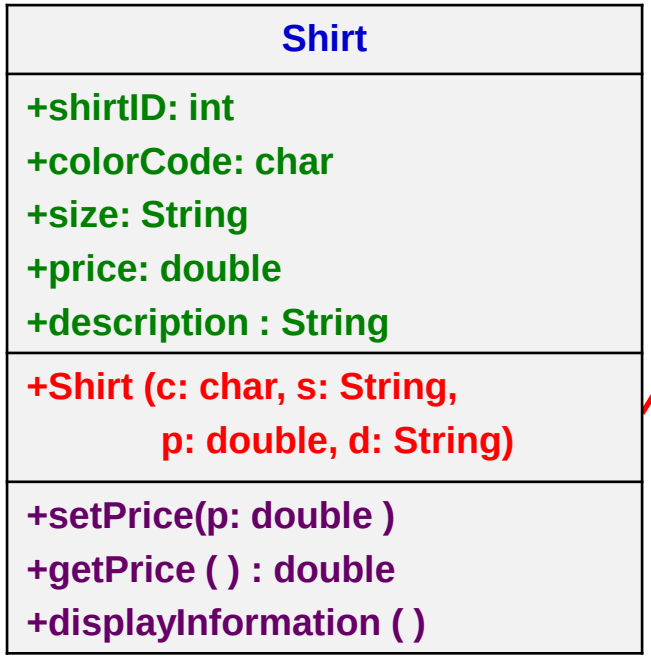
- 參數列表使用上的限制
  - 參數列表與陣列參數意義相同，不可同時出現。
  - 下列2個方法雖然寫法不同，但會被視為相同的方法
    - void showName(String... names)
    - void showName(String[] names)
  - 2. 參數列表要放在所有參數的後面。
    - void showName(String str, String... names)　　//正確
    - void showName(String... names, String str)　　//錯誤

    //編譯器無法判斷要如何切割，所以編譯失敗
    - showName("C++概論", "資料庫概論", "網路概論");
  - 3. 1個方法只能使用1次參數列表。
    - void showName(String str, String... names)　　//正確
    - void showName(String... str, String... names)　　//錯誤

Outline

- 方法
- 建構式
- 類別成員

Java 建構式




```
01  public class Shirt {
02
03     public int shirtID = 0;
04     public char colorCode = 'G';
05     public String size = "XL" ;
06     public double price = 299.00;
07     public String description = "Polo Shirt";
08
09     public Shirt(char c, String s, double p, String d) {
10        colorCode = c;
11        size = s;
12        price = p;
13        description = d;
14     }                                              建構式
15
16     public void setPrice(double p) {
17        price = p;
18     }
19     public double getPrice( ) {
20        return price;
21     }
22     public void displayInformation() {
23        System.out.println("Shirt ID:" + shirtID);
24        System.out.println("Color:" + colorCode);
25        System.out.println("Size:" + size);
26        System.out.println("Price:" + price);
27     }
28
29  }
30
```

Constructor

- 在定義類別時，可以使用「建構式」(Constructor)來進行物件的初始化。
- 「建構式」就是將類別實體化建立成物件時，所執行的方法。會在物件產生之後自動被呼叫，建構物件初始的狀態， 因此稱為建構式(建構方法)。
- 建構式名稱必須與類別名稱相同，建構式不得指定傳回值。
- 例如：

```
public class SafeArray {
   // ..
   public SafeArray() {  // 建構方法
      // ....
   }
   public SafeArray(參數列) {  //
      // ....
   }
}
```

如果沒有定義任何的建構方法，則編譯器會自動配置一個無參數且沒有陳述內容的建構式。

程式在運行時，會根據配置物件時所指定的引數資料型態等，來決定該使用哪一個建構式新建物件。

建構式 語法

```
[modifiers] constructor_name([arguments]) {
    code_blocks
  }
}
```

Accessibility

Same as class_name

Arguments list

- 與類別名稱一樣
- 沒有回傳型態
- 預設建構子
- 可以多載(Overloading)

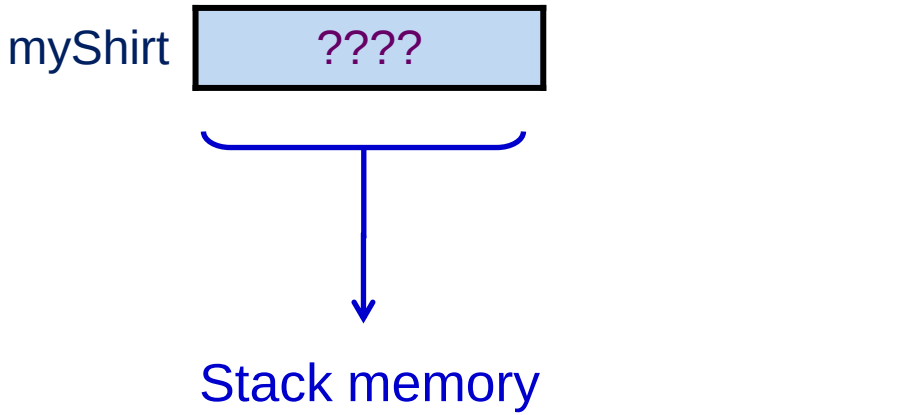
物件建構流程 – 1.宣告

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

    public Shirt(char c, double p, String d){
        colorCode = c;
        price = p;
        description = d;
    }
}
```

Shirt myShirt;

myShirt = new Shirt('G', 199.0 , "T-Shirt");

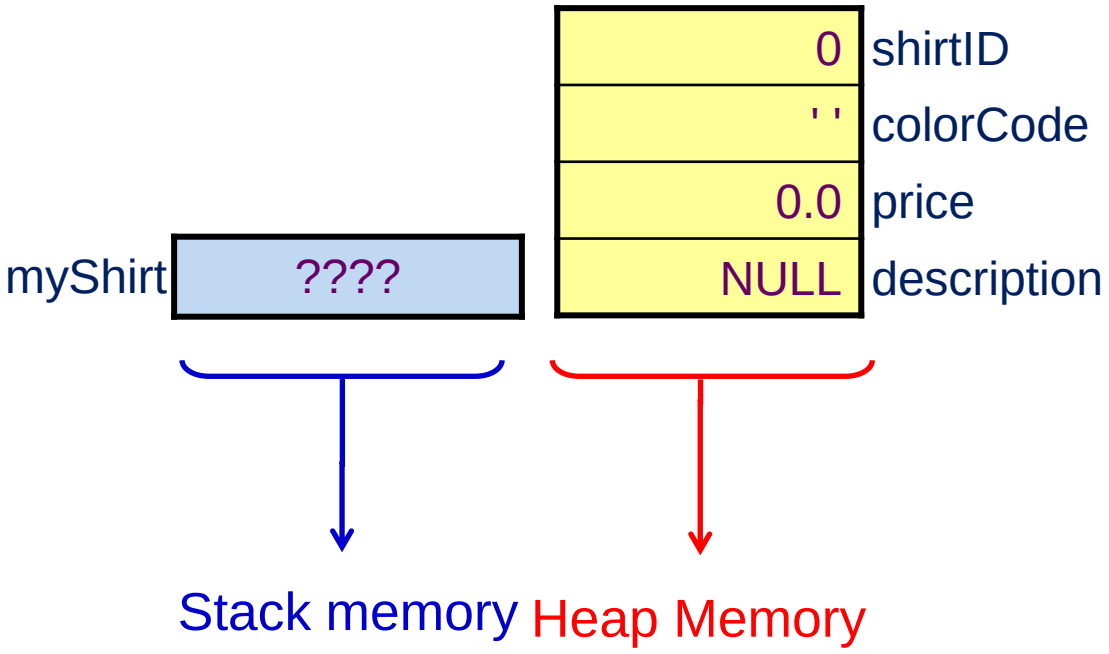


物件建構流程 – 2.實體化 記憶體配置

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

    public Shirt(char c, double p, String d){
        colorCode = c;
        price = p;
        description = d;
    }
}
```

```
Shirt myShirt;
myShirt = new Shirt('G', 199.0 , "T-Shirt" );
```

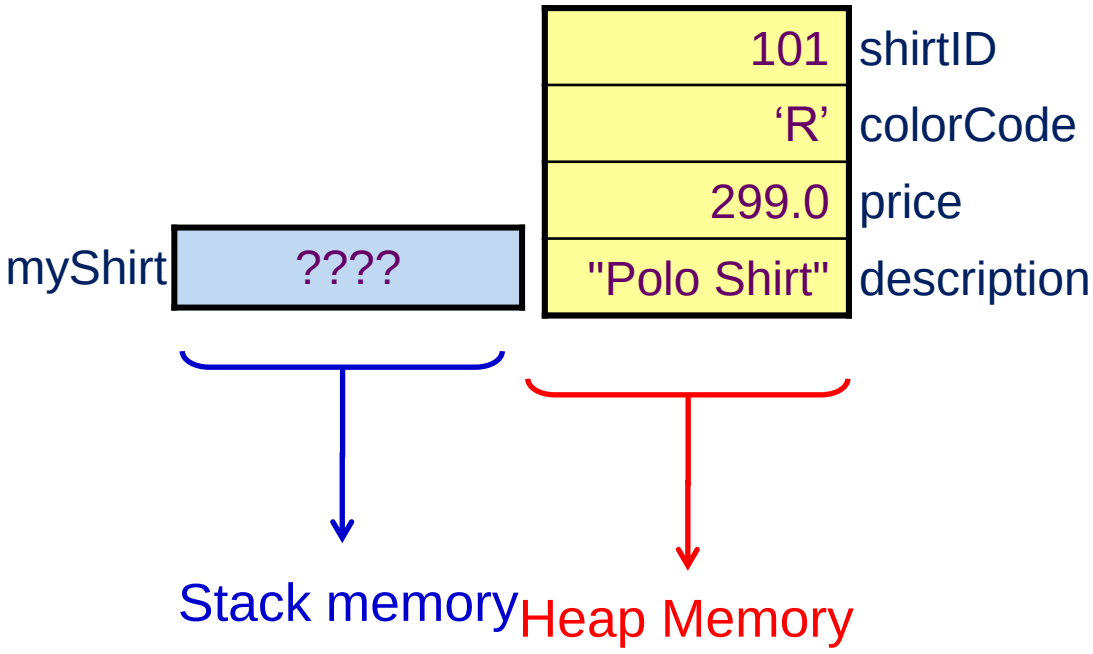


物件建構流程 – 3.初始化 初始值賦值

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

    public Shirt(char c, double p, String d){
        colorCode = c;
        price = p;
        description = d;
    }
}
```

```
Shirt myShirt;
myShirt = new Shirt('G', 199.0 , "T-Shirt" );
```

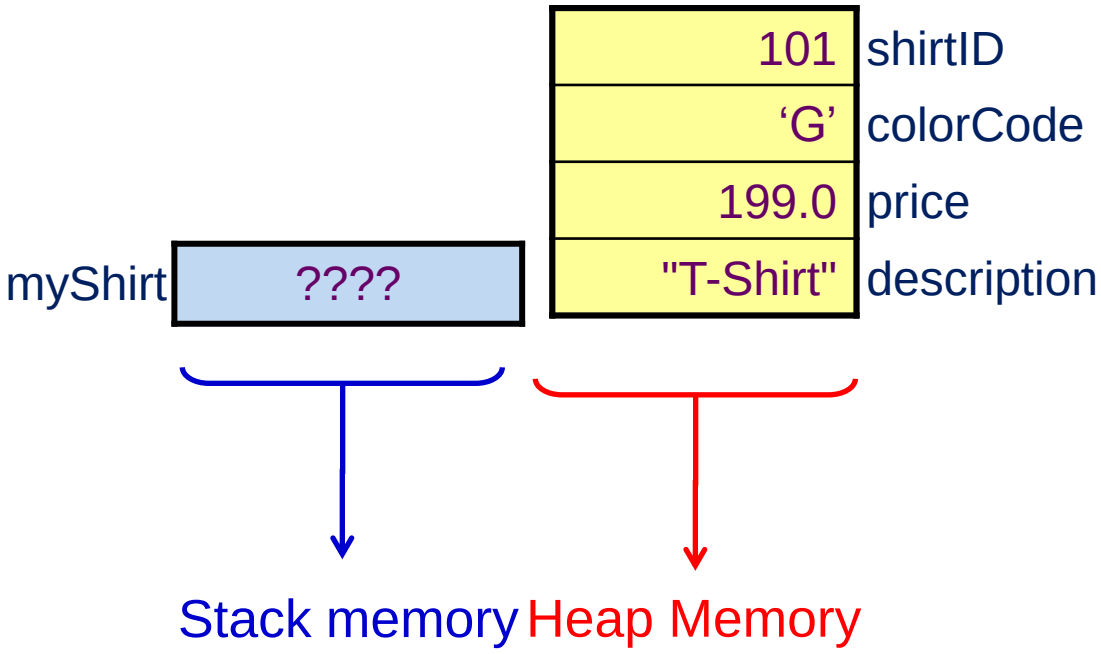


物件建構流程 – 4.初始化 執行建構式

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

    public Shirt(char c, double p, String d){
        colorCode = c;
        price = p;
        description = d;
    }
}
```

```
Shirt myShirt;
myShirt = new Shirt('G', 199.0 , "T-Shirt" );
```

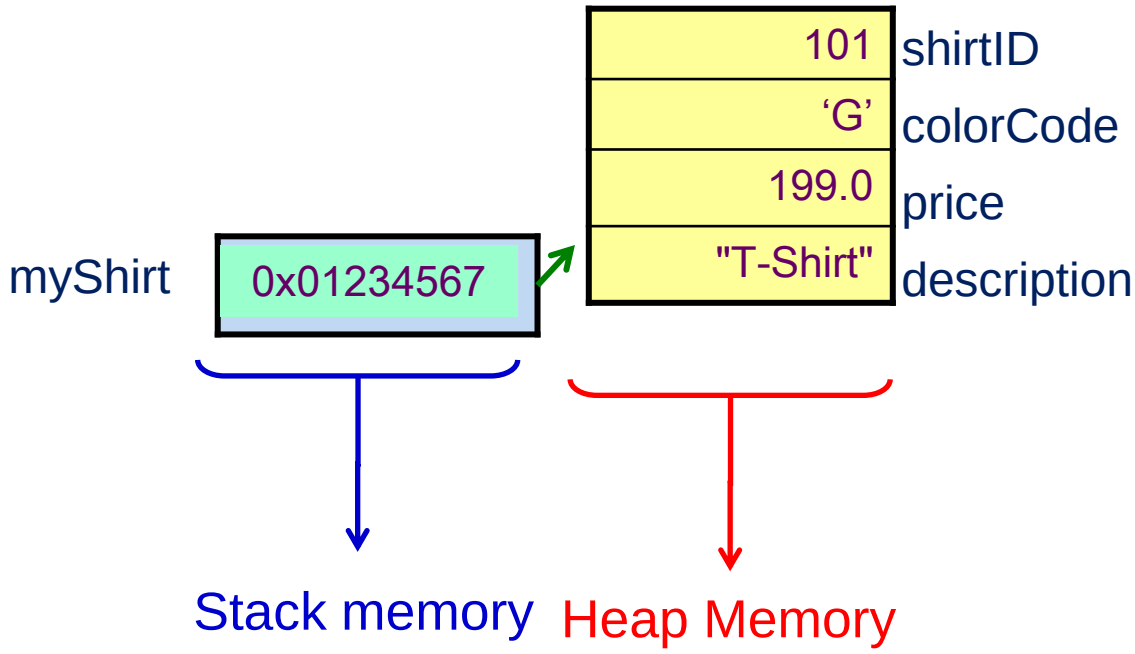


物件建構流程 – 5.儲存物件參考

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

    public Shirt(char c, double p, String d){
        colorCode = c;
        price = p;
        description = d;
    }
}
```

```
Shirt myShirt;
myShirt = new Shirt('G', 199.0 , "T-Shirt" );
```

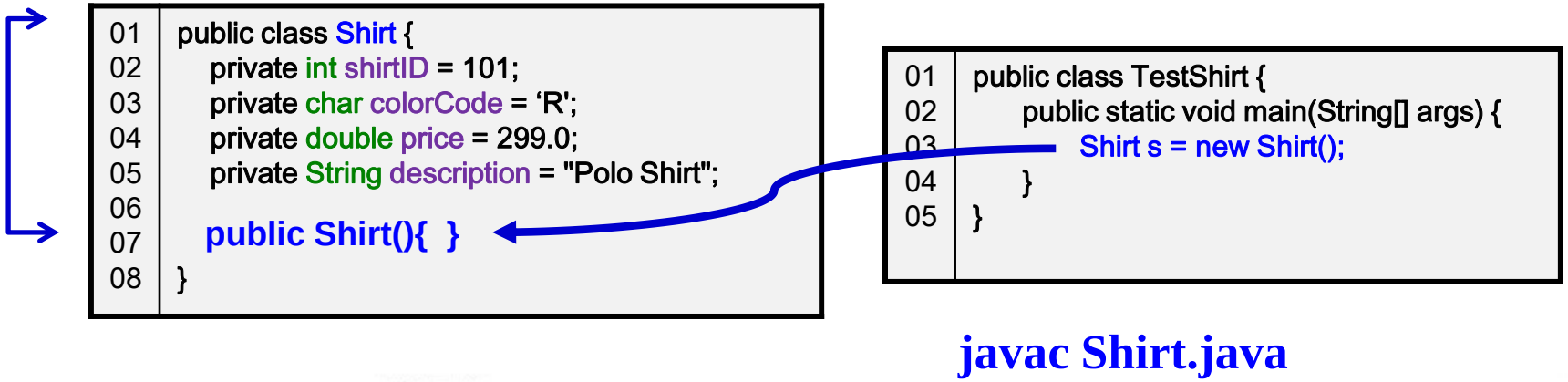


建構式分類

- 預設建構式 (Default Constructor)
- 非預設建構式

預設建構式 (Default Constructor)

- 物件裡面一定要有建構式，所以在撰寫類別時必須定義該物件的建構式。
- 程式中若沒有定義建構式，在編譯時期會自動加入，所加入的就稱之為預設建構式；
  - **預設建構式沒有參數列(no arguments)** 。
  - 除了初始物件變數或繼承時super()的定義外，預設建構式沒有其他的程式敘述(no body statement)。
  - 自行建立後預設建構式即失效。



預設建構式 & 非預設建構式

範例Constructor.java

```
1.  class Book{
2.      String name;
3.      double price;
4.      String author;
5.      Book(){ //預設建構式
6.          name = "不詳";                    //非預設建構式
7.          price = 0.0;
8.          author = "不詳";
9.      }
10.     Book(String n, double p, String a){ //非預設建構式
11.         name = n;
12.         price = p;
13.         author = a;
14.     }
15.     void show(){
16.         System.out.println("書名：" + name);
17.         System.out.println("定價：" + price);
18.         System.out.println("作者：" + author);
19.     }
20. }
21.
22. class Constructor{
23.     public static void main(String[] args){
24.         Book book1 = new Book("Java 程式設計", 580.0, "張搵風");
25.         book1.show();
26.         Book book2 = new Book();
27.         book2.show();
28.     }
29. }
```

------輸出------

書名：Java程式設計

定價：580.0

作者：張搵風

書名：不詳

定價：0.0

作者：不詳

建構子多載 (Constructor overloading)

- 提供多組建構子為物件設定初值
  - 傳入參數數量或型態不同

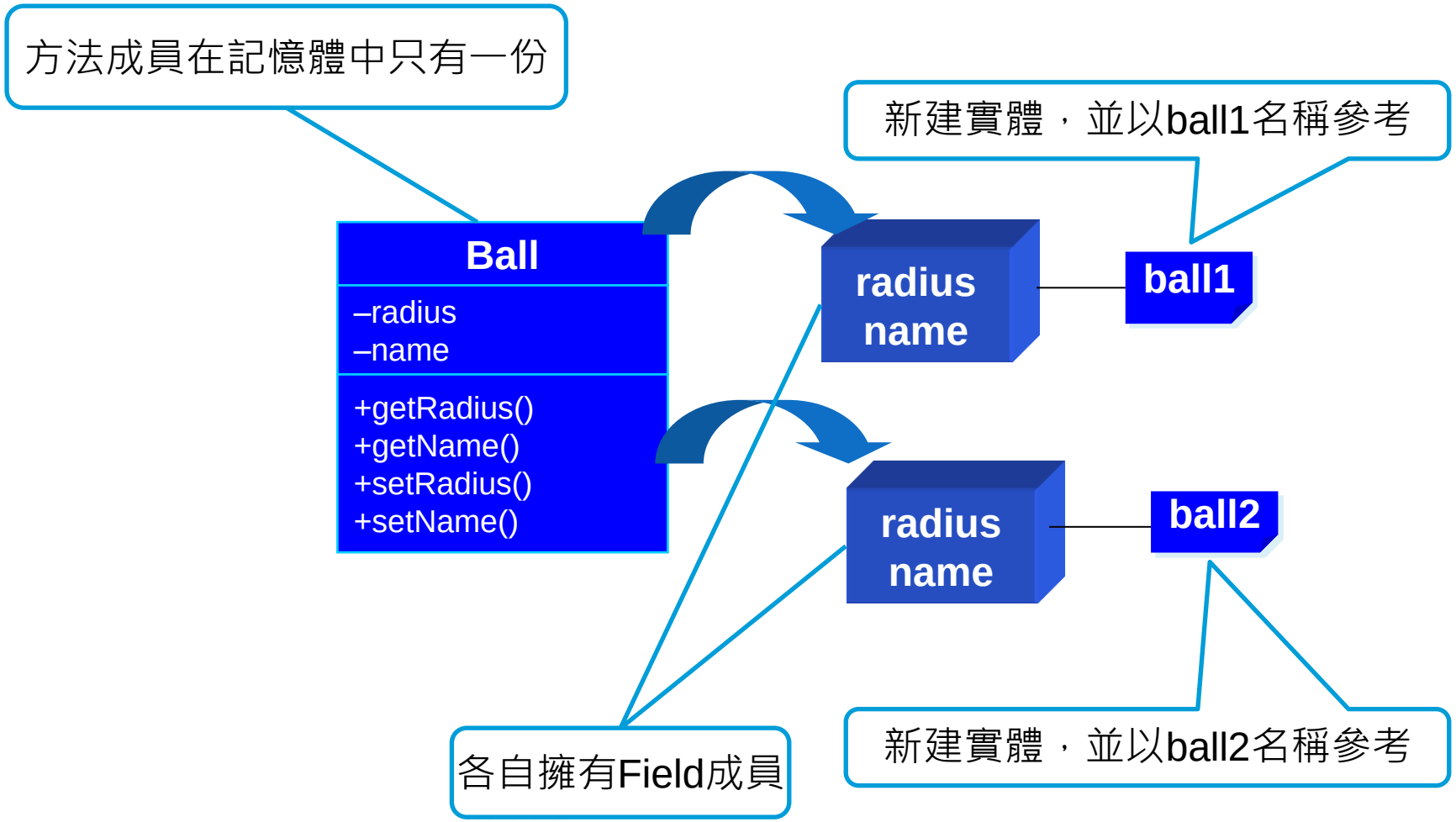
```
01  public class Shirt {
02      private int shirtID = 101;
03      private char colorCode = 'R';
04      private double price = 299.0;
05      private String description = "Polo Shirt";
06
07      public Shirt(int id) {
08          shirtID = id;
09      }
10      public Shirt(char color, double newPrice) {
11          colorCode = color;
12          price = newPrice;
13      }
14      public Shirt(char color, double newPrice,
15                   String desc) {
16          colorCode = color;
17          price = newPrice;
18          description = desc;
19      }
20  }
```

```
01  public class TestShirt {
02      public static void main(String[] args) {
03
04          Shirt s1 = new Shirt();
05
06          Shirt s2 = new Shirt(101);
07
08          Shirt s3 = new Shirt('G', 600.0);
09
10          Shirt s4 = new Shirt('Y', 199.0,
11                               "T-Shirt");
12
13      }
14  }
```

建構式 v.s 方法

- 建構式雖然很像方法，但是有3個不同點：
  - 呼叫時機不同
  - 建構式無回傳值
  - 建構式名稱與類別相同
- 建構式可分成：
  - 預設建構式
  - 非預設建構式

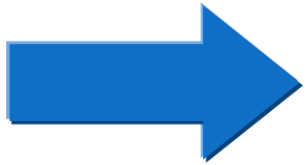
About this


方法成員在記憶體中只有一份
新建實體，並以ball1名稱參考
Ball
–radius
–name
+getRadius()
+getName()
+setRadius()
+setName()
radius
name
ball1
radius
name
ball2
各自擁有Field成員
新建實體，並以ball2名稱參考

About this

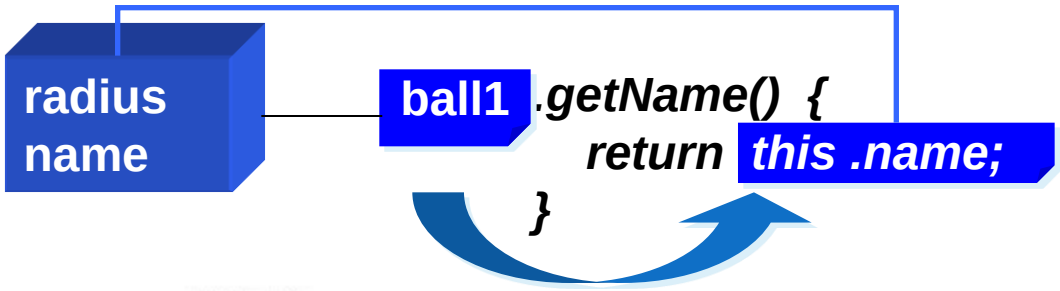
- 方法中所撰寫的每一個資料成員其實會隱含一個this參考名稱，這個this名稱參考至呼叫方法的物件，當呼叫getName()方法時，其實相當於執行：

```
public double getName()
{
        return name;
}
```



```
public double getName()
{
        return this.name;
}
```

- 當使用ball1並呼叫getRadius()方法時，this所參考的就是ball1所參考的物件：



About this

- 當在方法中使用資料成員時，都會隱含使用this名稱，當然也可明確的指定，例如在方法定義時使用：

```
public Ball(double radius, String name) {
    this.radius = radius;
    this.name = name;
}
```

- 參數名稱與資料成員名稱相同時，為了避免參數的作用範圍覆蓋了資料成員的作用範圍，必須明確的使用this名稱來指定，但如果參數名稱與資料成員名稱不相同則不用特別指定：

```
public Ball(double r, String n) {
    radius = r;     // 實際等於this.radius = r;
    name = n;     // 實際等於this.name = n;
}
```

Constructor & this

```
public class SafeArray {
    private int[] arr;

    public SafeArray() {
        this(10); // 預設 10 個元素
    }

    public SafeArray(int length) {
        arr = new int[length];
    }

....
}
```

使用this(10)，
這會呼叫另一個有參數的建構方法。

Outline

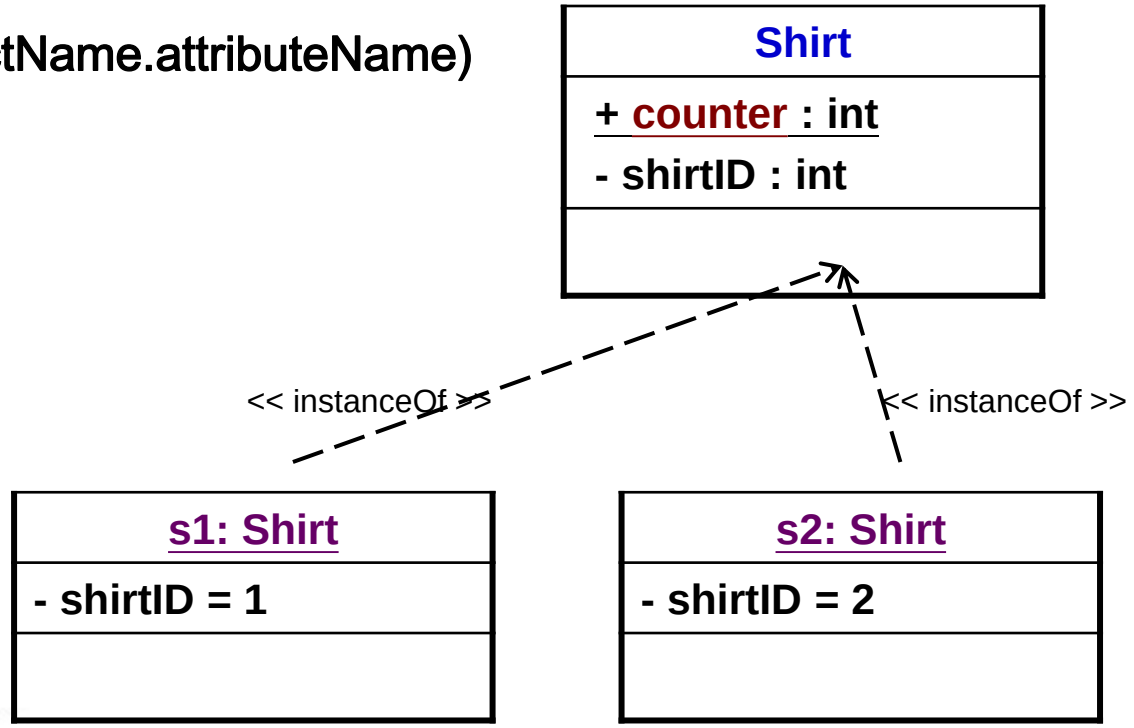
- 方法
- 建構式
- 類別成員

類別成員及物件成員

- 物件(實體)/非靜態/成員 instance (non-static) member
  - 物件屬性：每個物件各自擁有一份資料。
  - 物件方法：需透過特定物件來操作。
  - 物件實體化之後，物件屬性才存在，物件方法才可使用。
- 類別/靜態成員 class (static) member
  - Java用 static 修飾字來宣告類別屬性及類別方法。
  - 類別屬性：不伴隨物件，由同類別所有物件共享。
  - 類別方法：不需特定物件，可由類別來操作。

類別屬性

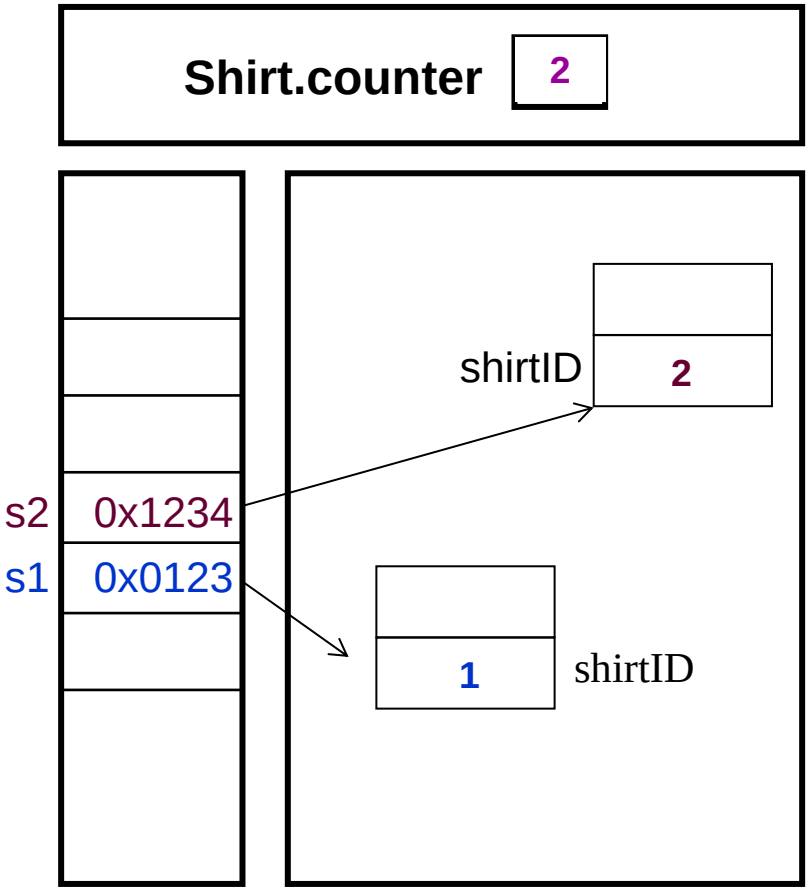
- 用來表示同一類別所有物件共用的屬性欄位 (類似全域變數 Global Variable)
- 可用以下兩種方法存取
  - **類別名稱.屬性名稱 (ClassName.attributeName)**
  - **物件名稱.屬性名稱 (ObjectName.attributeName)**



類別屬性 static attribute

```
01  public class Shirt {
02          public static int counter = 0;
03          public int ShirtID;
04
05          public Shirt() {
06              counter++;
07              shirtID = counter;
08          }
09  }
```

```
01  public class TestShirt {
02     public static void main(String []
03  args) {
04          Shirt s1 = new Shirt();
05          Shirt s2 = new Shirt();
06     }
    }
```



類別方法 static method

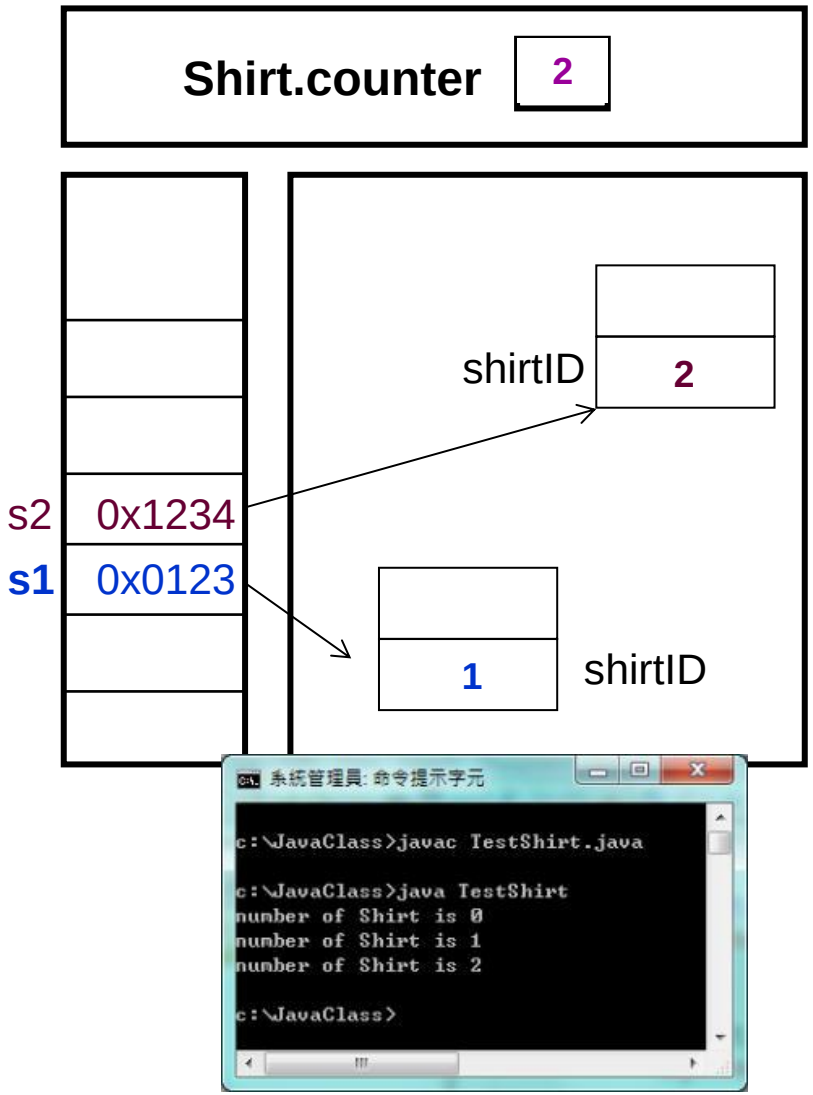
- **用途**
  - 只能存取類別屬性 (static variable)
- **語法**

  [modifiers] static return type name([arg_list]) { }
- **呼叫用法**
  1. ClassName.methodName();
  2. ObjectName.methodName();
  3. methodName();

類別方法 static method

```
01  public class Shirt {
02          public static int counter = 0;
03          public int shirtID;
04
05          public Shirt() {
06                  counter++;
07                  shirtID = counter;
08          }
09
10          public static int getTotalCount( ){
11                  return counter;
12          }
13  }
```

```
01  public class TestShirt {
02      public static void main(String [] args) {
03
04      System.out.println("number of Shirt is "
05                      + Shirt.getTotalCount());
06      Shirt s1 = new Shirt();
07      System.out.println("number of Shirt is "
08                      + s1.getTotalCount());
09      Shirt s2 = new Shirt();
10      System.out.println("number of Shirt is "
11                      + s2.getTotalCount());
12  }
```



類別方法 static method

```
01  public class Shirt {
02          public static int counter = 0;
03          public int shirtID;
04
05          public Shirt() {
06                  counter++;
07                  shirtID = counter;
08          }
09
10          public static int getTotalCount( ){
11                  return counter;
12          }
13
14          public static String convertShirtSize( int numericalSize) {
15                  if(numericalSize < 10){
16                          return "S" ;
17                  } else if(numericalSize < 14){
18                          return "M" ;
19                  } else if(numericalSize < 18){
20                          return "L" ;
21                  } else {
22                          return "XL" ;
23                  }
24          }
25
26  }
```

```
01  public class TestShirt {
02      public static void main(String [] args) {
03              ……..
04          System.out.println("size 14 equals  to " + Shirt.convertShirtSize(14);
05      }
06  }
```

Q & A