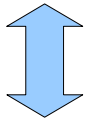
物件的建構

- 物件的建構—幫物件設定**初始狀態**
- 物件的建構方法 (Constructor)
  - 『建構方法』是一種 method
  - Constructor 在建立物件時由**系統自動**呼叫，例如當在建構物件時 Ootest ot = new Ootest(); 『()』就是系統自動呼叫的作用
- 若類別中沒有定義建構方法，則 Java 編譯器會自動幫該類別定義一個**預設建構方法** (Defualt constructor)

物件的建構 (Cont.)

```
5  import java.io.*;
6  class Test{
7      int x, y;
8  }
9  public class OOTest{
10     public static void main(String[] argv)
11         throws IOException{
12         BufferedReader br =
13             new BufferedReader(new InputStreamReader(System.in));
14         Test a = new Test();
15     }
16 }
```



```
5  import java.io.*;
6  class Test{
7      int x, y;
8      Test(){//預設的建構方法
9      }
10 }
11 public class OOTest{
12     public static void main(String[] argv)
13         throws IOException{
14         BufferedReader br =
15             new BufferedReader(new InputStreamReader(System.in));
16         Test a = new Test();
17     }
18 }
```

物件的建構 (Cont.)

- 物件的**建構方法**也可以自行定義
  - 需遵循一般方法 (method) 的寫法規則撰寫
  - 建構方法**不能**有回傳值，包括 void 也不能加
  - **建構方法名稱**必須要與**類別名稱**一模一樣

```
5  import java.io.*;
6  class Test{
7      int x, y;
8      Test(){//預設的建構方法
9          x = 10;
10         y = 20;
11     }
12 }
13 public class OOTest{
14     public static void main(String[] argv)
15         throws IOException{
16         BufferedReader br =
17             new BufferedReader(new InputStreamReader(System.in));
18         Test a = new Test();
19         System.out.println("x->"+a.x);
20         System.out.println("y->"+a.y);
21     }
22 }
```

不具參數的建構方法

```
x->10
y->20
```

物件的建構 (Cont.)

- 具有參數的**建構方法**
  - 讓物件的建構更具**彈性**
  - 建立物件時可透過 new 運算子及類別名稱之後的小括號傳入參數

```
 5 import java.io.*;
 6 class Test{
 7     int x, y;
 8     Test(int initX, int initY){//具參數的建構方法
 9         x = initX;
10         y = initY;
11     }
12 }
13 public class OOTest{
14     public static void main(String[] argv)
15         throws IOException{
16         BufferedReader br =
17             new BufferedReader(new InputStreamReader(System.in));
18         Test a = new Test(50, 100);
19         System.out.println("x->"+a.x);
20         System.out.println("y->"+a.y);
21     }
22 }
```

執行結果

```
x->50
y->100
```

物件的建構 (Cont.)

- 具有參數的**建構方法**注意事項
  - 一旦類別有自行定義的建構方法時，使用 new 運算子產生物件時必須要依建構方法的定義傳入相同數量及型別的參數，否則編譯時會出錯

```
 5 import java.io.*;
 6 class Test{
 7     int x, y;
 8     Test(int initX, int initY){//具參數的建構方法
 9         x = initX;
10         y = initY;
11     }
12 }
13 public class OOTest{
14     public static void main(String[] argv)
15         throws IOException{
16         BufferedReader br =
17             new BufferedReader(new InputStreamReader(System.in));
18         Test a = new Test(50);
19         System.out.println("x->"+a.x);
20         System.out.println("y->"+a.y);
21     }
22 }
```

```
Exception in thread "main" java.lang.Error: 尚未解決的編譯問題：
        建構子 Test(int) 是未定義的

        at OOTest.main(OOTest.java:18)
```

物件的建構 (Cont.)

- 建構方法的多種定義(Overloading)
  - 建構方法可依需求定義多種版本的建構方法
  - 編譯器會依所傳入的參數個數及型別來決定要用哪一個版本的建構方法

```
成員變數 x -> 50
成員變數 y -> 100
----------------------------
成員變數 x -> 137
成員變數 y -> 0
----------------------------
成員變數 x -> 0
成員變數 y -> 0
```

```
5  import java.io.*;
6  class Test{
7      int x, y;
8      Test(){
9      }
10     Test(int initX){
11         x = initX;
12     }
13     Test(int initX, int initY){//具參數的建構方法
14         x = initX;
15         y = initY;
16     }
17     void show(){
18         System.out.println("成員變數 x -> "+x);
19         System.out.println("成員變數 y -> "+y);
20     }
21 }
22 public class OOTest{
23     public static void main(String[] argv)
24         throws IOException{
25         BufferedReader br =
26             new BufferedReader(new InputStreamReader(System.in));
27         Test a = new Test(50, 100);
28         Test b = new Test(137);
29         Test c = new Test();
30         a.show();
31         System.out.println("----------------------------");
32         b.show();
33         System.out.println("----------------------------");
34         c.show();
35     }
36 }
```

物件的建構 (Cont.)

- 使用建構方法的多種定義 (Overloading) 需注意
  - 一旦類別定義有需要傳入參數的建構方法，則 Java 編譯器即不再替類別建立不需傳入參數的**預設建構方法**
  - 如果該類別需要一個不需傳入參數的建構方法，則需自行撰寫
  - 若類別備有不需傳入參數的建構方法，將可提供一種**預設狀態**給物件
  - 建議作法，先定義一個不需傳入參數的建構方法，其餘再依需求訂定各種版本的建構方法

物件的建構 (Cont.)

```
5  import java.io.*;
6  class Test{
7      int x, y;
8      Test(){
9      }
10     Test(int x){
11         this.x = x;
12     }
13     Test(int x, int y){//具參數的建構方法
14         this.x = x;
15         this.y = y;
16     }
17     void show(){
18         System.out.println("成員變數 x -> "+x);
19         System.out.println("成員變數 y -> "+y);
20     }
21 }
22 public class OOTest{
23     public static void main(String[] argv)
24         throws IOException{
25         BufferedReader br =
26             new BufferedReader(new InputStreamReader(System.in));
27         Test a = new Test(50, 100);
28         Test b = new Test(137);
29         Test c = new Test();
30         a.show();
31         System.out.println("----------------------------");
32         b.show();
33         System.out.println("----------------------------");
34         c.show();
35     }
36 }
```

類別的成員變數

區域變數

- 當建構方法定義的變數名與類別的成員變數名字一樣時，可用 this 保留字來標示所要存取的對象是成員變數

```
成員變數 x -> 50
成員變數 y -> 100
----------------------------
成員變數 x -> 137
成員變數 y -> 0
----------------------------
成員變數 x -> 0
成員變數 y -> 0
```

物件的建構 (Cont.)

- this 保留字的妙用
  - 假設某一類別有三個建構方法，分別為不需要傳入參數、傳入一個參數與傳入兩個參數，又，傳入一個參數與傳入兩個參數的部份程式碼相同，則此時可用 this 保留字達到
  - 多使用 this 保留字可提昇程式中重覆資料的一致性

```
5  import java.io.*;
6  class Test{
7      int x, y;
8      Test(){
9      }
10     Test(int x){
11         this.x = x;
12     }
13     Test(int x, int y){//具參數的建構方法
14         this(x);
15         this.y = y;
16     }
17     void show(){
18         System.out.println("成員變數 x -> "+x);
19         System.out.println("成員變數 y -> "+y);
20     }
21 }
22 public class OOTest{
23     public static void main(String[] argv)
24         throws IOException{
25         BufferedReader br =
26             new BufferedReader(new InputStreamReader(System.in));
27         Test a = new Test(50, 100);
28         Test b = new Test(137);
29         Test c = new Test();
30         a.show();
31         System.out.println("----------------------------");
32         b.show();
33         System.out.println("----------------------------");
34         c.show();
35     }
36 }
```

```
成員變數 x -> 50
成員變數 y -> 100
----------------------------
成員變數 x -> 137
成員變數 y -> 0
----------------------------
成員變數 x -> 0
成員變數 y -> 0
```

封裝與資訊隱藏

- 封裝與資訊隱藏
  - 透過類別的建構方法定義可**不必**在 main() 方法中直接存取類別的成員變數
  - 真正的物件導向程式設計必須要達到**不能修改**物件中的成員變數，此即所謂的『資訊隱藏』
  - 資訊隱藏 (Information hiding) ：類別外部不能看到、接觸到物件內部的資訊 (屬性)
  - 若真有必要更新或修改成員變數的值，該如何做？
    - 藉由類別公開給外部的方法來修改物件中的成員變數
    - 例：車子類別，可藉由『前進』方法來修改『載油量』這個成員變數

封裝與資訊隱藏 (Cont.)

- 為達到資訊隱藏的目的，程式設計時必須要能提供必要的方法，讓外部能正常操作物件
- 外界只要知道有哪些方法可用即可順利操作物件，而不用費心瞭解類別內部的構成
- 這樣的作法亦可達到程式碼重覆使用及提昇程式開發效率的目的
- 把類別的屬性、操作屬性的方法包裝在一起，僅對外公開必要的介面，這種即稱之為『封裝』(Encapsulation)

封裝與資訊隱藏 (Cont.)

- 如何讓外界無法直接存取封裝在類別內的成員變數 (屬性)?
  - 在類別中使用**存取控制字符**
  - private: 只有在成員變數所屬的類別裡才能存取
  - protected: 除了類別本身，在子類別或同一套件中的類別才能存取此成員變數
  - public: 任何類別都可以存取此成員變數

封裝與資訊隱藏 (Cont.)

```
5  import java.io.*;
6  class Test{
7      private int x = 19;
8      int y;
9      Test(){
10     }
11     Test(int x){
12         this.x = x;
13     }
14     void modifyX(int x){
15         this.x = x;
16     }
17     void show(){
18         System.out.println("成員變數 x -> "+x);
19         //System.out.println("成員變數 y -> "+y);
20     }
21 }
22 public class OOTest{
23     public static void main(String[] argv)
24         throws IOException{
25         BufferedReader br =
26             new BufferedReader(new InputStreamReader(System.in));
27         Test b = new Test();
28         b.show();
29         b.modifyX(79);
30         b.show();
31         b.x = 40;
32     }
33 }
```

- private 存取控制字符
  - 凡被標示為 private 的成員變數，除非透過類別所定義的方法進行存取，否則其他做法都是不合法的

```
Exception in thread "main" java.lang.Error: 尚未解決的編譯問題：
        欄位 Test.x 不可見

        at OOTest.main(OOTest.java:31)
```

封裝與資訊隱藏 (Cont.)

- 若沒有標示存取控制字符，Java 會將該成員變數設定成『預設控制』(Default Access)
- 預設控制：只有同一個套件 (Package) 的類別可以存取該成員變數
- 何謂套件 ?(正式介紹在第 13 章)
  - 所有編譯後產生的 .class 當都在同一個目錄下，則在此目錄下的類別就屬在同一個套件下
- 存取控制字符不僅用在成員變數，方法也可用來限制被使用的範圍

封裝與資訊隱藏 (Cont.)

- 要達到資訊隱藏的目的，最好的作法便是撰寫操作成員變數的方法
- 使用者不需瞭解類別內部的實作細節亦可撰寫程式
- 好處是日後該類別的實作方式改變，只要類別提供的操作方法不變，使用者即不用更動使用到該類別的程式碼
- 為了隱藏成員變數，適時適切的為成員變數加上存取字符是必要的

封裝與資訊隱藏 (Cont.)

- 為成員變數加存取限制的原則
  - 除特殊需求外，所有成員變數都加上 private 的存取字符
  - 若成員變數的內容需要修改，則由類別所提供的操作方法來完成
  - 進行成員變數內容修改的操作方法命名規則
    - 取得成員變數的值： getXXX, 其中 XXX 便是成員變數的名稱
    - 變定成員變數的值： setXXX, 其中 XXX 是成員變數的名稱

封裝與資訊隱藏 (Cont.)

- 為類別中定義的方法加上存取限制的通則
  - 提供給外界用：使用 public 字符
  - 僅供同類別中的其方法呼叫：使用 private 字符
- ★注意★
  - 對於建構方法，除非有特殊用途，否則都應加上 public 字符
  - 若使用 private 字符則 new 運算子就不能呼叫建構方法，無法設定物件的初始狀態

封裝與資訊隱藏 (Cont.)

```
5  class Test{
6      private int x, y;
7      public Test(int x, int y){
8          this.x = x;
9          this.y = y;
10     }
11     public int getX(){return x;}
12     public void setX(int x){this.x = x;}
13     public int getY(){return y;}
14     public void setY(int y){this.y = y;}
15 }
16 public class OOTest{
17     public static void main(String[] argv){
18         Test a = new Test(30, 40);
19         System.out.println("成員 X : "+a.getX());
20         System.out.println("成員 Y : "+a.getY());
21         a.setX(80);
22         a.setY(90);
23         System.out.println("成員 X : "+a.getX());
24         System.out.println("成員 Y : "+a.getY());
25     }
26 }
```

```
成員 X : 30
成員 Y : 40
成員 X : 80
成員 Y : 90
```

封裝與資訊隱藏 (Cont.)

- 利用合適的建構方法設計加上將成員變數設成 private 可達到下列好處
  - 將資訊隱藏起來
  - main() 方法裡的寫法會比較簡捷
  - 因為直接呼叫類別的方法，比較能看出 main() 中做了哪些事

封裝與資訊隱藏 (Cont.)

```
5  class Car{
6      private double gas;
7      private double eff;
8      public void showState(){
9          System.out.println("目前載油量爲："+gas+" 公升");
10         System.out.println("耗油量爲每公升跑 "+eff+" 公里");
11     }
12     public void move(double distance){
13         if(gas >= (distance/eff)){
14             gas -= (distance/eff);
15             System.out.println("行駛了 "+distance+" 公里");
16         }
17     }
18     public Car(double gas, double eff){
19         this.gas = gas;
20         this.eff = eff;
21     }
22     public Car(){
23         this(50, 20);
24     }
25 }
26 public class TestCar{
27     public static void main(String[] argv){
28         Car oldCar = new Car(30, 10);
29         Car newCar = new Car();
30         System.out.print("老爺車 oldCar ");
31         oldCar.move(50);
32         oldCar.showState();
33         System.out.print("新車 newCar ");
34         newCar.move(60);
35         newCar.showState();
36     }
37 }
```

```
老爺車 oldCar 行駛了 50.0 公里
目前載油量爲：25.0 公升
耗油量爲每公升跑 10.0 公里
新車 newCar 行駛了 60.0 公里
目前載油量爲：47.0 公升
耗油量爲每公升跑 20.0 公里
```

封裝與資訊隱藏 (Cont.)

- 在資訊隱藏之後，如果成員變數是另一個類別的物件，則在設計存取此成員物件相關的公開方法時，須注意是否要將此成員物件也公開到外部
- 例如：平面幾何中的圓，可以由圓心座標及半徑所表示，如果將座標點設計成一個物件，稱之為 point，然後再設計一個類別稱之為『圓』，則在『圓』這個物件中就包含了一個『 point 』的成員物件以代表圓心所在位置

封裝與資訊隱藏 (Cont.)

```
5  class Point{
6      private double x, y;
7      public void setX(double x){this.x = x;}
8      public void setY(double y){this.y = y;}
9      public String toString(){return "("+x+","+y+")";}
10     public Point(double x, double y){
11         this.x = x;
12         this.y = y;
13     }
14     public Point(){x = y = 0;}
15 }
16 class Circle{
17     private Point p;
18     private double r;
19     public Point getP(){return p;}
20     Circle(double x, double y, double r){
21         p = new Point(x, y);
22         this.r = r;
23     }
24     Circle(){this(0, 0, 1);}
25     public String toString(){
26         return "圓心"+p.toString()+" 半徑"+r;
27     }
28 }
29 public class TestPrivateMember{
30     public static void main(String[] argv){
31         Circle c = new Circle(3, 4, 5);
32         Point p = c.getP();
33         p.setX(6);
34         System.out.println(c.toString());
35     }
36 }
```

```
圓心(6.0,4.0) 半徑5.0
```

這樣的做法有達到『資訊隱藏』嗎？

外界可以直接存取 point 物件

封裝與資訊隱藏 (Cont.)

```
5  class Point{
6      private double x, y;
7      public void setX(double x){this.x = x;}
8      public void setY(double y){this.y = y;}
9      public String toString(){return "("+x+","+y+")";}
10     public Point(double x, double y){
11         this.x = x;
12         this.y = y;
13     }
14     public Point(){x = y = 0;}
15 }
16 class Circle{
17     private Point p;
18     private double r;
19     public Point getP(){return p;}
20     Circle(double x, double y, double r){
21         p = new Point(x, y);
22         this.r = r;
23     }
24     Circle(){this(0, 0, 1);}
25     public String toString(){
26         return "圓心 "+p.toString()+" 半徑 "+r;
27     }
28 }
29 public class TestPrivateMember{
30     public static void main(String[] argv){
31         Circle c = new Circle(3, 4, 5);
32         Point p = c.getP();
33         p.setX(6);
34         System.out.println(c.toString());
35     }
36 }
```

```
圓心 (6.0,4.0) 半徑 5.0
```

```
5  class Point{
6      private double x, y;
7      public void setX(double x){this.x = x;}
8      public void setY(double y){this.y = y;}
9      public String toString(){return "("+x+","+y+")";}
10     public Point(double x, double y){
11         this.x = x;
12         this.y = y;
13     }
14     public Point(){x = y = 0;}
15     public Point(Point p){x = p.x;  y = p.y;}
16 }
17 class Circle{
18     private Point p;
19     private double r;
20     public Point getP(){return new Point(p);}
21     Circle(double x, double y, double r){
22         p = new Point(x, y);
23         this.r = r;
24     }
25     Circle(){this(0, 0, 1);}
26     public String toString(){
27         return "圓心 "+p.toString()+" 半徑 "+r;
28     }
29 }
30 public class TestPrivateMember{
31     public static void main(String[] argv){
32         Circle c = new Circle(3, 4, 5);
33         Point p = c.getP();
34         p.setX(6);
35         System.out.println(c.toString());
36     }
37 }
```

```
圓心 (3.0,4.0) 半徑 5.0
```

Static 共享成員變數

- 每個類別所建構出來的物件都具有各自的成員變數以表現物件間的差異
- 若要讓這些物件的某一個屬性變成共通屬性，則可用 static 來達成要求
- 例如，由 Car 類別所建構出來的物件，不管新車或老爺車，我希望將其耗油率設成一樣，可藉由 static 做到
- 當類別中的成員變數加上 static 存取控制字符，表示所有此類別所建構出來的物件都共享這個成員變數，而非各自擁有一份

Static 共享成員變數 (Cont.)

```
5  class Test{
6      public int x;
7      public static int y;
8      public Test(int x, int y){
9          this.x = x;
10         this.y = y;
11     }
12     public String toString(){
13         return "(x, y):("+x+", "+ y +")";
14     }
15 }
16 public class StaticMember{
17     public static void main(String[] argv){
18         Test a = new Test(100, 40);
19         Test b = new Test(200, 50);
20         Test c = new Test(300, 60);
21         System.out.println("物件a"+a);
22         System.out.println("物件b"+b);
23         System.out.println("物件c"+c);
24
25     }
26 }
```

```
物件a(x, y):(100, 60)
物件b(x, y):(200, 60)
物件c(x, y):(300, 60)
```

Static 共享成員變數 (Cont.)

- static 成員變數可用一般成員變數的方式存取，此外其也可以用**類別名稱**存取。

```
5  class Test{
6      public int x;
7      public static int y;
8      public Test(int x, int y){
9          this.x = x;
10         this.y = y;
11     }
12     public String toString(){
13         return "(x, y):("+x+", "+ y +")";
14     }
15 }
16 public class StaticMember{
17     public static void main(String[] argv){
18         Test a = new Test(100, 40);
19         Test b = new Test(200, 50);
20         Test c = new Test(300, 60);
21         Test.y = 100;
22         System.out.println("物件a"+a);
23         System.out.println("物件b"+b);
24         System.out.println("物件c"+c);
25
26     }
27 }
```

```
物件a(x, y):(100, 100)
物件b(x, y):(200, 100)
物件c(x, y):(300, 100)
```

Static 共享成員變數 (Cont.)

- static 成員變數另一個特點是，在物件被建構起來之前，就能存取類別的 static 成員變數

```
5  class Test{
6      public int x;
7      public static int y;
8      public Test(int x){
9          this.x = x;
10         //this.y = y;
11     }
12     public String toString(){
13         return "(x, y):("+x+", "+ y +")";
14     }
15 }
16 public class StaticMember{
17     public static void main(String[] argv){
18         Test.y = 100;
19         Test a = new Test(100);
20         Test b = new Test(200);
21         Test c = new Test(300);
22         //Test.y = 100;
23         System.out.println("物件a"+a);
24         System.out.println("物件b"+b);
25         System.out.println("物件c"+c);
26
27     }
28 }
```

```
物件a(x, y):(100, 100)
物件b(x, y):(200, 100)
物件c(x, y):(300, 100)
```

Static 共享成員變數 (Cont.)

- 因 static 成員變數具共享特性，故其不需在物件被建構方法中給初始值
- 為了避免寫程式忘了設定 static 成員變數的初值，Java 提供了 **static 初始區塊** (static initializer) 以確保物件被建構起來對 static 成員進行初始化

```
5  class Test{
6      public int x;
7      public static int y;
8      static{
9          y = 100;
10     }
11     public Test(int x){ this.x = x;}
12     public String toString(){
13         return "(x, y):("+x+", "+ y +")";
14     }
15 }
16 public class StaticMember{
17     public static void main(String[] argv){
18         System.out.println(Test.y);
19         Test a = new Test(100);
20         Test b = new Test(200);
21         Test c = new Test(300);
22         //Test.y = 100;
23         System.out.println("物件a"+a);
24         System.out.println("物件b"+b);
25         System.out.println("物件c"+c);
26
27     }
28 }
```

```
100
物件a(x, y):(100, 100)
物件b(x, y):(200, 100)
物件c(x, y):(300, 100)
```

Static 共享成員變數 (Cont.)

- 因 static 成員變數是由同一類別所建構出來的物件所共享的，所以不需先建構物件就能以類別名稱進行存取，故該 static 成員變數又可稱之為**類別變數** (class variable)
- 其他非 static 成員變數，則因其必須在物件被建構出來後才能被存取，故稱此成員變數為**實體變數** (Instant variable)
- static 除了可用在成員變數上，也可將其應用在方法上

Static 共享方法

- 一個使用 static 的方法除了可以透過所屬類別的物件呼叫外，和 static 成員變數一樣，可以在沒有產生任何物件的情況下透過類別名稱進行呼叫

```
5  import java.io.*;
6  class Test{
7      public static void print(){
8          System.out.println("呼叫 static method");
9      }
10 }
11 public class StaticMethod{
12     public static void main(String[] argv) throws IOException{
13         BufferedReader br =
14             new BufferedReader(new InputStreamReader(System.in));
15         Test.print();
16         Test a = new Test();
17         a.print();
18     }
19 }
```

```
呼叫 static method
呼叫 static method
```

Static 共享方法 (Cont.)

- 由於 static 成員變數及方法不需產生物件即可使用的特性，因此可來提供一組相關聯的工具方法或常數值
- 像 java 類別庫中的 Math 類別，其提供了數學相關的 static 運算方法 (e.g. 次方、亂數、三角函數) 以及常數值 (e.g. PI)
- ★ 注意★
  - 在 static 區塊或方法中，不能用到任何非 static 成員變數及方法，亦不能使用 this 保留字

Static 共享方法 (Cont.)

- 因為非 static 的成員變數和方法是跟隨物件而生，而 static 區塊或方法卻可以在物件未產生前使用，所以因為沒有產生物件自然配置非 static 成員變數，再者， this 當然也就沒有物件可指
- main() 方法是一個 public static 的方法，而且不用回傳任何值

Final 存取控制

- 如何讓同一類別產生的物件都共用一份一致的成員變數資料？
  - 使用 static 保留字
  - 將 static 成員變數給定初始值後即予以鎖定，不讓任一物件對 static 成員變數進行修改，以達到資料『一致』的目的
  - final 可搭配 static 保留字達到成員變數資料一致的目的，讓有 final 的 static 成員一旦給定初始值後就不再允許更動

Final 存取控制 (Cont.)

```
5  import java.io.*;
6  class Test{
7      static final int x = 0;
8      public static void print(){
9          System.out.println("呼叫 static method");
10     }
11 }
12 public class StaticMethod{
13     public static void main(String[] argv) throws IOException{
14         BufferedReader br =
15             new BufferedReader(new InputStreamReader(System.in));
16         Test.print();
17         Test a = new Test();
18         a.x = 20;
19         a.print();
20     }
21 }
```

```
Exception in thread "main" java.lang.Error: 尚未解決的編譯問題：
        無法指派終態欄位 Test.x

        at StaticMethod.main(StaticMethod.java:18)
```

- ★注意★
  - 一旦成員變數宣告成 final，若是 static 成員變數，則必須在 static 初始區塊或宣告同時給定初始值
  - 若是非 static 成員變數，則必須在宣告的同時或在建構方法給定