



# 物件的建構

## 第 9 章



# 9-1 建構方法 (CONSTRUCTOR)

- 建構方法會在建立物件時由系統自動呼叫，以建構物件初始的狀態。
- 在使用 **NEW** 算符時，類別的名稱之後加上一對小括號，就是呼叫建構方法。

## 9-1-1 預設建構方法 (DEFAULT CONSTRUCTOR)

### 程式 NoConstructor.java 未定義建構方法

```
01 class Test {  
02     int x,y;  
03 }  
04  
05 public class NoConstructor {  
06  
07     public static void main(String[] argv){  
08         Test a = new Test();  
09     }  
10 }
```

# 預設建構方法 (DEFAULT CONSTRUCTOR)

**程式** DefaultConstructor.java 沒有內容的建構方法

```
01 class Test {
02     int x,y;
03
04     // 預設的建構方法
05     Test() {
06     }
07 }
08
09 public class DefaultConstructor {
10
11     public static void main(String[] argv){
12         Test a = new Test();
13     }
14 }
```

## 9-1-2 自行定義建構方法

定義建構方法時有以下幾點需要注意：

- 建構方法不能傳回任何值
- 建構方法一定要和類別同名

# 無參數的建構方法

## 程式 NoArgument.java 定義無參數的建構方法

```
01 class Test {  
02     int x,y;  
03  
04     // 不具參數的建構方法  
05     Test() {  
06         x = 10;  
07         y = 20;  
08     }  
09 }  
10
```

# 無參數的建構方法

```
11 public class NoArgument {  
12  
13     public static void main(String[] argv) {  
14         Test a = new Test();  
15         System.out.println("成員變數x：" + a.x);  
16         System.out.println("成員變數y：" + a.y);  
17     }  
18 }
```

## 執行結果

成員變數x：10

成員變數y：20

# 具有參數的建構方法

## 程式 WithArgument.java 撰寫具有參數的建構方法

```
01 class Test {  
02     int x,y;  
03  
04     // 具有參數的建構方法  
05     Test(int initX,int initY) {  
06         x = initX;  
07         y = initY;  
08     }  
09 }  
10
```



# 具有參數的建構方法

```
11 public class WithArgument {  
12  
13     public static void main(String[] argv) {  
14         Test a = new Test(30,40);  
15         System.out.println("成員變數x：" + a.x);  
16         System.out.println("成員變數y：" + a.y);  
17     }  
18 }
```

## 執行結果

成員變數x：30

成員變數y：40

# 具有參數的建構方法

**程式** WrongArgument.java 呼叫建構方法時傳遞錯誤的參數

```
01 class Test {  
02     int x,y;  
03  
04     // 具有參數的建構方法  
05     Test(int initX,int initY) { // 需要 2 個參數  
06         x = initX;  
07         y = initY;  
08     }  
09 }
```

# 具有參數的建構方法

```
10
11 public class WrongArgument {
12
13     public static void main(String[] argv) {
14         Test a = new Test(30); // 少 1 個參數
15         System.out.println("成員變數x：" + a.x);
16         System.out.println("成員變數y：" + a.y);
17     }
18 }
```

# 具有參數的建構方法

## 執行結果

```
WrongArgument.java:14: error: constructor Test in class Test cannot be
```

```
applied to given types;
```

```
        Test a = new Test(30); // 少1個參數
```

```
        ^
```

```
required: int,int
```

```
found: int
```

```
reason: actual and formal argument lists differ in length
```

```
1 error
```

## 9-1-3 建構方法的多重定義 (OVERLOADING)

- 可依據不同的場合劑型最適當的初始設定
- 依據參數個數、資料型別選擇符合的建構方法

## 9-1-3 建構方法的多重定義 (OVERLOADING)

### 程式 Overloading.java 使用多重定義的建構方法

```
01 class Test {
02     int x = 10, y = 20;    // 宣告成員變數時直接指定初值
03
04     // 兩個參數的建構方法
05     Test(int initX,int initY) {
06         x = initX;
07         y = initY;
08     }
09
10     // 一個參數的建構方法
11     Test(int initX) {
12         x = initX;
13     }
14 }
```

# 建構方法的多重定義(OVERLOADING)

```
15    // 不具參數的建構方法
16    Test() {
17    }
18
19    void show() { // 顯示成員變數的方法
20        System.out.println("成員變數x：" + x);
21        System.out.println("成員變數y：" + y);
22    }
23 }
24
25 public class Overloading {
```

# 建構方法的多重定義(OVERLOADING)

```
26  
27 public static void main(String[] argv){  
28     Test a = new Test(30,50);  
29     Test b = new Test(60);  
30     Test c = new Test();  
31  
32     a.show();  
33     b.show();  
34     c.show();  
35 }  
36 }
```

## 執行結果

成員變數x : 30  
成員變數y : 50  
成員變數x : 60  
成員變數y : 20  
成員變數x : 10  
成員變數y : 20



## 9-1-4 THIS 保留字

- 傳入建構方法的參數名稱有時會遮蔽掉成員參數, 可使用 **THIS** 表示執行此方法的物件

## 9-1-4 THIS 保留字

### 程式 Shadowing.java 使用 this 存取成員變數

```
01 class Test {  
02     int x = 10, y = 20;  
03  
04     // 建構方法參數與成員變數同名  
05     Test(int x, int y) {  
06         this.x = x;  
07         this.y = y;  
08     }  
09 }  
10  
11 public class Shadowing {  
12
```

# THIS 保留字

```
13 public static void main(String[] argv) {  
14     Test a = new Test(30,50);  
15     System.out.println("成員變數x：" + a.x);  
16     System.out.println("成員變數y：" + a.y);  
17 }  
18 }
```

## 執行結果

成員變數x：30  
成員變數y：50

# THIS 保留字

## 程式 CallConstructor.java 不正確的呼叫建構方法

```
01 class Test {
02     int x = 10, y = 20;
03
04     // 在建構方法中呼叫另一個建構方法
05     Test(int x, int y) {
06         Test(x); // 錯誤！
07         this.y = y;
08     }
09
10     Test(int x) {
11         this.x = x;
12     }
13 }
```

# THIS 保留字

```
15 public class CallConstructor {  
16  
17     public static void main(String[] argv){  
18         Test a = new Test(30,50);  
19         System.out.println("成員變數x：" + a.x);  
20         System.out.println("成員變數y：" + a.y);  
21     }  
22 }
```

## 執行結果

```
CallConstructor.java:6: cannot find symbol  
        Test(x); // 錯誤!  
        ^  
  
    symbol   : method Test(int)  
    location: class Test  
1 error
```

# THIS 保留字

## 程式 CallByThis.java 透過 this 呼叫建構方法

```
01 class Test {
02     int x = 10, y = 20;
03
04     // 在建構方法中呼叫另一個建構方法
05     Test(int x, int y) {
06         this(x);                // 呼叫另一個建構方法
07         this.y = y;
08     }
09
10     Test(int x) {
11         this.x = x;
12     }
13 }
```

# THIS 保留字

```
14  
15 public class CallByThis {  
16  
17     public static void main(String[] argv) {  
18         Test a = new Test(30,50);  
19         System.out.println("成員變數x：" + a.x);  
20         System.out.println("成員變數y：" + a.y);  
21     }  
22 }
```

## 9-2 封裝與資訊隱藏

- 物件導向程式設計方法，要求『不能』直接修改物件的成員變數。
  - 以術語來說就是資訊隱藏 (INFORMATION HIDING)。
- 將類別的屬性、操作屬性的方法包裝在一起，只對外公開必要的介面，即稱為封裝(ENCAPSULATION)。



## 9-2-1 類別成員的存取控制

- 使用存取控制字符 (**ACCESS MODIFIER**) 來限制外部對類別成員變數的存取。

字符	說明
private	只有在成員所屬的類別之中才能存取此成員
protected	除了類別本身, 在子類別 (Subclass, 參見 11 章) 或同一套件 (Package, 參見 13 章) 中的類別, 也能存取此成員
public	任何類別都可以存取此成員
都不加	只有類別本身, 及同一套件 (參見 13 章) 中的類別才能存取此成員

# 類別成員的存取控制

## 程式 PrivateMember.java 私有成員變數

```
01 class Test {  
02     private int i = 1; // 私有成員變數  
03  
04     void modifyMember(int i) {  
05         this.i = i; // 類別中可以存取 i  
06     }  
07  
08     void show() { // 類別中可以存取 i  
09         System.out.println("成員變數i：" + i);  
10     }  
11 }  
12
```

# 類別成員的存取控制

```
13 public class PrivateMember {  
14  
15     public static void main(String argv[]) {  
16         Test a = new Test();  
17  
18         a.show();  
19         a.modifyMember(20);  
20         a.show();  
21         a.i = 40; // 喔喔, i 是私有成員變數  
22     }  
23 }
```

## 執行結果

```
PrivateMember.java:21: i has private access in Test  
        a.i = 40; // 喔喔, i 是私有成員變數  
            ^  
1 error
```

## 9-2-2 為成員變數撰寫存取方法

- 為了隱藏成員變數，就需適時地為成員變數加上存取限制
  - 除非有公開的必要，  
否則最好所有成員變數都加上 **PRIVATE**
  - 如果使用此類別的程式需要透過成員來完成某件事，就由類別提供方法來完成。

## 9-2-2 為成員變數撰寫存取方法

- 如果需要修改或是取得成員的值，就提供專門存取成員的方法。
- 通常用來取得成員值的方法會命名為 **GETXXX**，其中 **X X X** 就是成員變數的名稱；  
相對的，用來設定成員值的方法就命名為 **SETXXX**。

## 9-2-2 為成員變數撰寫存取方法

- 對於類別中所定義的方法, 加上存取限制的通則如下：
  - 如果是要提供給外界呼叫的方法, 請明確的標示為 **PUBLIC**
  - 如果只是供類別中其他的方法呼叫, 請明確的標示為 **PRIVATE**
  - 建構方法, 除非有特別的用途, 否則應該都標示為 **PUBLIC**

## 9-2-2 為成員變數撰寫存取方法

### 程式 AccessMethod.java 存取控制字符的使用

```
01 class Test {
02     private int x,y; // 成員都是private
03
04     public Test(int x,int y) {
05         this.x = x;
06         this.y = y;
07     }
08
09     // 成員x與y的存取方法
10     public int getX() {return x;}
11     public void setX(int x) {this.x = x;}
12     public int getY() {return y;}
13     public void setY(int y) {this.y = y;}
```

# 為成員變數撰寫存取方法

```
14 }  
15  
16 public class AccessMethod {  
17  
18     public static void main(String[] argv){  
19         Test a = new Test(30,40);  
20  
21         // 透過方法更改成員值  
22         a.setX(80);  
23         a.setY(80);  
24  
25         // 透過方法取得成員值  
26         System.out.println("成員x：" + a.getX());  
27         System.out.println("成員y：" + a.getY());  
28     }  
29 }
```



# 為成員變數撰寫存取方法

## 程式 TestCard.java 符合資訊隱藏的 IC 卡類別

```
01 class IcCard { // 代表 IC 卡的類別
02     private long id;        // 卡號
03     private int money;      // 卡片餘額
04
05     public void showInfo() { // 顯示卡片資訊的方法
06         System.out.print("卡片卡號 " + id);
07         System.out.println(", 餘額 " + money + " 元 ");
08     }
09
10     public Boolean add(int value) { // 加值方法：參數為要加值的金額
11         if (value>0 && value+money <= 10000) { // 儲值上限一萬
12             money += value;
13             return true; // 加值成功
```

# 為成員變數撰寫存取方法

```
14     }  
15     return false;    // 加值失敗  
16 }  
17  
18 public IcCard(long id, int money) {  
19     this.id = id;  
20     this.money =money;  
21 }  
22  
23 public IcCard(long id) {  
24     this(id, 0);    // 呼叫 2 個參數的版本  
25 }  
26 }  
27  
28
```

# 為成員變數撰寫存取方法

```
29 public class TestCard {  
30     public static void main(String[] argv) {  
31         IcCard myCard = new IcCard(0x336789AB, 500); // 建立物件  
32         IcCard hisCard = new IcCard(0x13572468);      // 建立物件  
33  
34         System.out.println("我的卡片加值 500 元" +  
35             (myCard.add(500) ? "成功":"失敗") );  
36         myCard.showInfo();          // 呼叫方法  
37  
38         System.out.println("他的卡片加值 9000 元" +  
39             (hisCard.add(9000) ? "成功":"失敗") );  
40         hisCard.showInfo();         // 呼叫方法  
41     }  
42 }
```

# 為成員變數撰寫存取方法

## 執行結果

我的卡片加值 500 元成功

卡片卡號 862423467, 餘額 1000 元

他的卡片加值 9000 元成功

卡片卡號 324478056, 餘額 9000 元

## 9-2-3 傳回成員物件的資訊

```
class Point {    // 點
    private double x,y;
    public void setX(double x) {
        this.x = x;
    }
    public void setY(double y) {
        this.y = y;
    }
    ...          // 建構方法、其它方法
}
```

# 傳回成員物件的資訊

```
class Circle {           // 圓
    private Point p;      // 圓心
    private double r;     // 半徑
    public Point getP() {
        ...
    }
    ...                  // 建構方法、其它方法
}
```

# 傳回成員物件的資訊

## 程式 SettingPrivateMember.java 修改私有成員物件

```
01 class Point {    // 點
02     private double x,y;
03
04     public void setX(double x) { this.x = x; }
05     public void setY(double y) { this.y = y; }
06
07     public String toString() {    // 將物件資訊轉成字串的方法
08         return "(" + x + "," + y + ")";
09     }
10
11     public Point(double x, double y) {    // 建構方法
12         this.x = x; this.y = y;
13     }
```

# 傳回成員物件的資訊

```
14 }  
15  
16 class Circle {          // 圓  
17     private Point p;     // 圓心  
18     private double r;    // 半徑  
19  
20     public Point getP() { return p; } // 直接傳回成員物件  
21  
22     public String toString() {      // 將物件資訊轉成字串的方法  
23         return "圓心：" + p.toString() + " 半徑：" + r;  
24     }  
25  
26     Circle(double x, double y, double r) {    // 建構方法  
27         p = new Point(x, y);  
28         this.r = r;
```



# 傳回成員物件的資訊

```
29     }  
30 }  
31  
32 public class SettingPrivateMember {  
33     public static void main(String[] argv) {  
34         Circle c = new Circle(3,4,5); // 圓心 (3,4), 半徑 5  
35  
36         Point p = c.getP();           // 取得圓心  
37         p.setX(6);                     // 變更圓心座標  
38         System.out.println(c.toString());  
39     }  
40 }
```

## 執行結果

圓心 : (6.0,4.0) 半徑 : 5.0

# 傳回成員物件的資訊

## 程式 HidePrivateMember.java 隱藏私有的成員

```
01 class Point {    // 點
    .
    .    //同前一程式
    .
15     public Point(Point p) { // 新增一個建構方法
16         x = p.x;           // 以另一物件做為初值
17         y = p.y;
18     }
19 }
20
```

# 傳回成員物件的資訊

```
21 class Circle {           // 圓
22     private Point p;      // 圓心
23     private double r;     // 半徑
24
25     public Point getP() {  // 修改 getP() 方法
26         return new Point(p); // 建立一個新的 Point 物件傳回
27     }
28
29     .
30     . // 同前一程式
31     .
```

## 執行結果

圓心：(3.0,4.0) 半徑：5.0

## 9-3 STATIC 共享成員變數

- 使用 **STATIC** 共享成員變數可以表現這個物件的共用屬性
- 此類別所有物件會共用此屬性

## 9-3-1 STATIC 存取控制

### 程式 StaticMember.java 共享成員

```
01 class Test {  
02     public int x;           // 個別物件擁有一份  
03     public static int y;     // 所有此類別物件共享  
04  
05     public Test(int x,int y) { // 具有參數的建構方法  
06         this.x = x;  
07         this.y = y;  
08     }  
09 }
```

# STATIC 存取控制

```
10 public String toString() { // 轉成字串
11     return "(x,y):(" + x + "," + y + ")";
12 }
13 }
14
15 public class StaticMember {
16
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 }
```

## 執行結果

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

## 程式 AccessByClass.java 透過類別名稱存取 static 成員

```
15 public class AccessByClass {
16
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;           // 透過類別名稱存取 static 成員
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 }
```

### 執行結果

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

# 使用類別名稱存取 **STATIC**成員變數

**程式** ClassVar.java 未建立物件也可使用 static 成員變數

```
01 class Test {  
02     public int x;           // 個別物件擁有一份  
03     public static int y;    // 所有此類別物件共享  
04  
05     public Test(int x) { // 建構方法只設定 x 的值  
06         this.x = x;  
07     }  
08  
09     public String toString() { // 轉成字串  
10         return "(x,y):(" + x + "," + y + ")";  
11     }  
12 }
```



# 使用類別名稱存取 **STATIC**成員變數

```
13
14 public class ClassVar {
15
16     public static void main(String[] argv){
17         Test.y = 100;          // 尚未建立物件即存取 static 成員變數
18         Test a = new Test(100);
19         Test b = new Test(200);
20         Test c = new Test(300);
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 }
```

## 執行結果

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

## 9-3-3 STATIC 初始區塊

- **STATIC** 初始區塊可以在產生物件之前，用程式來設定 **STATIC** 成員

## 9-3-3 STATIC 初始區塊

**程式** StaticInit.java 在 static 初始區塊中設定初值

```
01 class Test {
02     public int x; // 個別物件擁有一份
03     public static int y; // 所有此類別物件共享
04
05     static { // static初始區塊
06         y = 100;
07     }
08
09     // 具有參數的建構方法
10     public Test(int x) {
11         this.x = x;
```

# STATIC 初始區塊

```
12     }  
13  
14     // 轉成字串  
15     public String toString() {  
16         return "(x,y):(" + x + "," + y + ")";  
17     }  
18 }  
19  
20 public class StaticInit {  
21
```

# STATIC 初始區塊

```
22 public static void main(String[] argv){  
23     System.out.println(Test.y);  
24     Test a = new Test(100);  
25     Test b = new Test(200);  
26     Test c = new Test(300);  
27     System.out.println("物件a" + a);  
28     System.out.println("物件b" + b);  
29     System.out.println("物件c" + c);  
30 }  
31 }
```

## 執行結果

100

物件a (x, y) : (100, 100)

物件b (x, y) : (200, 100)

物件c (x, y) : (300, 100)

## 9-3-4 STATIC 方法

- **STATIC** 也可以應用到方法 **METHOD** 上
- 可以在沒有產生物件的情況下呼叫 **STATIC** 方法

## 9-3-4 STATIC 方法

### 程式 StaticMethod.java 呼叫 static 方法

```
01 class Test {
02     public static void print() { // static 方法
03         System.out.println("呼叫static方法");
04     }
05 }
06
07 public class StaticMethod {
08
09     public static void main(String[] argv){
10         Test.print(); // 透過類別名稱呼叫 static 方法
11         Test a = new Test();
12         a.print(); // 透過物件呼叫 static 方法
13     }
14 }
```

### 執行結果

呼叫static方法  
呼叫static方法

## 9-3-5 FINAL 存取控制

- **FINAL** 可限制特定的 **STATIC** 成員變數, 在設定完初值後就不能更動
- 宣告為 **FINAL** 後, 必須同時設定初值



## 9-3-5 FINAL 存取控制

### 程式 StaticFinal.java 不可更動的共享常數

```
01 class Test {  
02     static final int x = 10;  
03 }  
04  
05 public class StaticFinal {  
06  
07     public static void main(String[] argv) {  
08         Test a = new Test();  
09         a.x = 20; // x 是final，不能更改  
10     }  
11 }
```

### 執行結果

```
StaticFinal.java:9: cannot assign a value to final variable x  
    a.x = 20; // x 是final，不能更改  
      ^  
1 error
```

## 9-3-6 成員變數的預設值

- 凡是宣告在方法之外的成員變數，  
除非宣告為 **FINAL** 變數，  
否則都會有預設值

型別	預設值
數值類	0
char	'\u0000'
boolean	false
物件參照型別 ( 類別 )	null

## 9-3-6 成員變數的預設值

- 物件參照型別 (類別) 的預設值為 NULL：

```
public class Test {  
    static String z;                // String 為物件類別  
    public static void main(String[] argv) {  
        System.out.print(Test.z);    // 正確：輸出 null  
        System.out.print(Test.z.length()); // Runtime 錯誤：  
    }  
}
```

↑ Null 參照 (z) 不可使用 . 來存取成員

## 9-3-6 成員變數的預設值

- 程式的最後一行，  
就是因為 **TEST.Z** 的值為 **NULL**，因此後面不可再用句點來存取其內的成員
- 此時也可加一個 **IF** 判斷式來避免此問題：

```
if (Test.z != null)    // 不是 null 才讀取長度  
    System.out.print(Test.z.length());
```

## 9-3-6 成員變數的預設值

- 凡是宣告在方法內的變數, 則不會有預設值, 必須先設定變數的值, 然後才能讀取其內容

```
public class Test {  
    public static void main(String[] argv) {  
        int i;  
        System.out.print(i);           // 編譯錯誤：i 未初始化  
  
        int[] a, b = new int[2];  
        System.out.print(a[0]);        // 編譯錯誤：a 未初始化  
        System.out.print(b[0]);        // 正確：輸出 0  
    }  
}
```

## 9-4 綜合演練

- 9-4-1 提供輔助工具的類別

### 程式 MinMax.java 提供極大極小值功能的類別

```
01 class Utility {
02     public static int min(int[] data) {
03         int min = data[0];
04
05         // 逐一檢查陣列元素，有無比 min 更小的值
06         for(int i = 1; i < data.length; i++) {
07             min = (min <= data[i]) ? min : data[i];
08         }
09         return min;
10     }
```

# 提供輔助工具的類別

```
11
12 public static int max(int[] data) {
13     int max = data[0];
14
15     // 逐一檢查陣列元素，有無比 max 更大的值
16     for(int i = 1; i < data.length; i++) {
17         max = (max >= data[i]) ? max : data[i];
18     }
19     return max;
20 }
21
22 }
```

# 提供輔助工具的類別

```
23  
24 public class MinMax {  
25  
26     public static void main(String[] argv){  
27         int[] data = {9,10,37,3,29,44,9};  
28  
29         System.out.println("最小值：" + Utility.min(data));  
30         System.out.println("最大值：" + Utility.max(data));  
31     }  
32 }
```

## 執行結果

最小值：3

最大值：44



## 9-4-2 善用多重定義

**程式** OverloadConstructor.java 建立多重定義的建構方法

```
01 class Point {  
02     public int x,y;  
03     public Point(int x,int y) {  
04         this.x = x;  
05         this.y = y;  
06     }  
07 }  
08  
09 class Rectangle {  
10     Point upperleft;  
11     Point lowerright;
```

# 善用多重定義

```
12
13 // 完整版建構方法
14 public Rectangle(Point upperleft,Point lowerright) {
15     this.upperleft = upperleft;
16     this.lowerright = lowerright;
17 }
18
19 // 不需參數的建構方法
20 public Rectangle() {
21     this(new Point(0,0),new Point(5,-5));
22 }
23
```

# 善用多重定義

```
24 // 直接指定座標
25 public Rectangle(int x1,int y1,int x2,int y2) {
26     this(new Point(x1,y1),new Point(x2,y2));
27 }
28
29 // 正方形
30 public Rectangle(Point upperleft,int length) {
31     this(upperleft,new Point(upperleft.x + length,
32         upperleft.y - length));
33 }
34
35 // 計算面積
36 public int area() {
37     return (lowerright.x - upperleft.x) *
```

# 善用多重定義

```
38         (upperleft.y - lowerright.y);
39     }
40 }
41
42 public class OverloadConstructor {
43
44     public static void main(String[] argv){
45         Rectangle a = new Rectangle(0,0,5,-5);
46         Rectangle b = new Rectangle(new Point(3,3),4);
47
48         System.out.println("a的面積：" + a.area());
49         System.out.println("b的面積：" + b.area());
50     }
51 }
```

## 執行結果

a的面積：25

b的面積：16