Java程式設計進階
多型

鄭安翔
ansel_cheng@hotmail.com

課程大綱

1) 多型
   - 多型的特性
   - 存取權限與方法覆寫
   - 多型應用
   - 型別轉型
2) Object類別的方法

多型 Polymorphism

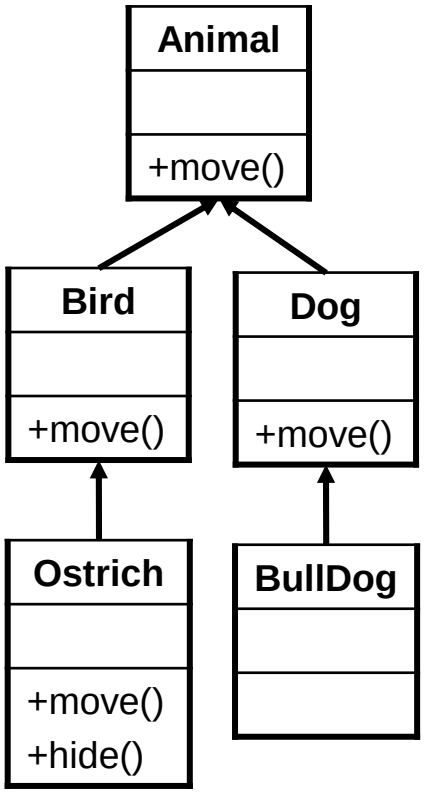
- 多型的意義
  - 一個物件可以用多種形態來看待
  - 型態間需有繼承關係：子類別可以被看待為父類別

- Java技術實作多型
  - 具有繼承關係的架構下,物件實體可以被視為多種型別。
  - 將子類別物件參考指定給父類別變數

父類別 **變數名稱 = new** 子類別建構子**();**

物件多型

- 鬥牛犬是一種狗，所有的狗皆是一種動物。
- 鴕鳥是鳥類，所有鳥類皆是一種動物




```
class Animal {
    void move(){…}
}
class Bird extends Animal {
    void move(){…}
}
class Dog extends Animal {
    void move(){…}
}
class Ostrich extends Bird {
    void move(){…}
    void hide(){…}
}
class Lion extends Cat {
}
```

Bulldog b = new Bulldog( );
用 **Bulldog** 鬥牛犬的眼光來看 **Bulldog**

Dog d = new Bulldog( );
用 **Dog** 狗的眼光來看 **Bulldog**

Animal a = new Bulldog ( );
用 **Animal** 動物的眼光來看 **Bulldog**

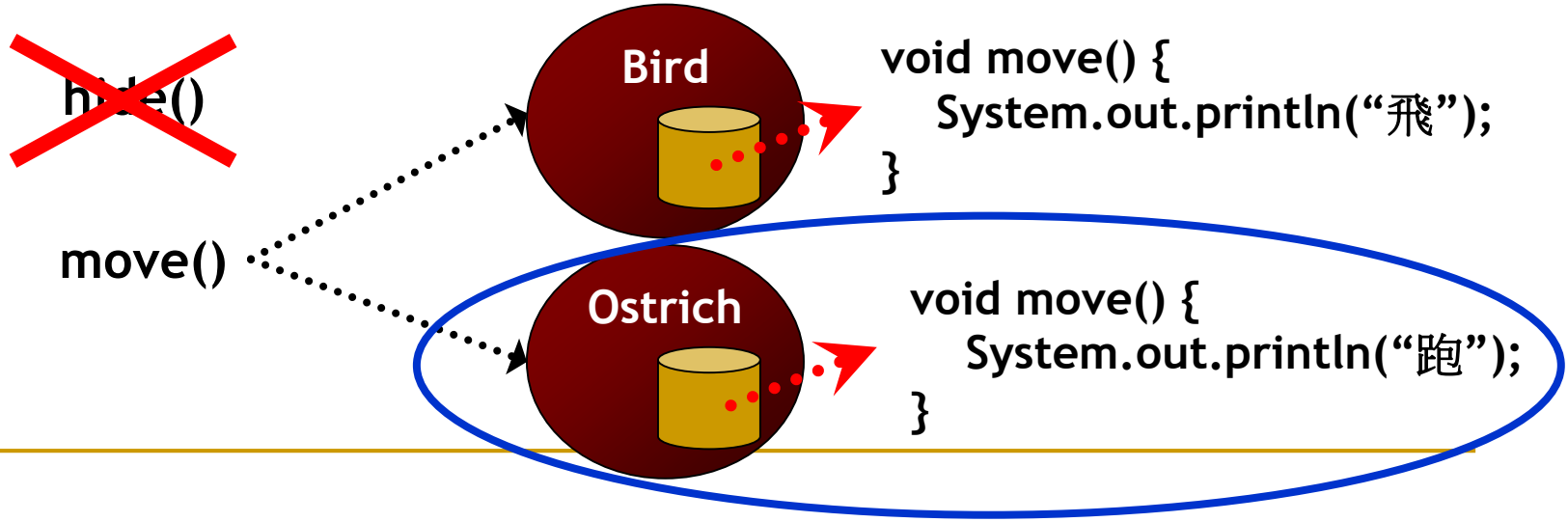
~~Bulldog b1 = new Dog( );~~
~~用 **Bulldog** 鬥牛犬的眼光來看所有狗~~

編譯錯誤

多型的特性

- 多型的特性
  - 不同型態表示並不會改變原來的實體
  - 將物件視為父類別,只能用父類別有定義之屬性及方法
  - 若父類別方法被子類別覆寫,多型時,用父類別的觀點呼叫,仍會執行子類別的方法

```
Bird bird = new Ostrich();
```



範例 - 多型的特性

```
class Animal {
  void move() {
    System.out.println("動");
  }
}
```

```
class Bird extends Animal {
  void move() {
    System.out.println("飛");
  }
}
```

```
class Ostrich extends Bird {
  void move() {
    System.out.println("跑");
  }
  void hide() {
    System.out.println("頭埋在土裡");
  }
}
```

```
Ostrich ostrich = new Ostrich();
ostrich.move();
ostrich.hide();
```

⇒ 跑
頭埋在土裡

```
Bird bird1 = new Bird();
bird1.move();
```

⇒ 飛

```
Bird bird2 = new Ostrich();
bird2.hide();

bird2.move();
```

⇒ 跑

在 Bird 型別中並不知道有 hide() 方法

型別檢查與虛擬方法調用

- Java的型別檢查
  - 編譯時期,compiler會以宣告的型別作型別檢查
    - 確保物件被視為父類別,只能用父類別定義之屬性及方法
  - 執行時期,JVM會以實際的型別作型別檢查
    - 確保多型時,用父類別的觀點呼叫,仍會執行子類別的方法
- 虛擬方法調用(呼叫) Virtual Method Invocation
  - Java程式會調用(呼叫)變數在執行時期所參考之物件的行為,而不是在編譯時期宣告類別的行為

範例

```
public class Employee {
    private String name = “Sean”;
    private double salary = 10000;
    public void getDetails() {
        System.out.println(“Name:” + name);
        System.out.println(“Salary:” + salary);
    }
}
```

```
public class Manager extends Employee {
    private String dept = “EDU”;
    public void getDetails() {
        super.getDetails();
        System.out.println(“Department:” + dept);
    }
    public void getDepartment() {
        System.out.println(“Department:” + dept);
    }
}
```

```
public class Test {
    public static void main(String [] args) {
        Employee e = new Employee();
        e.getDetails();
        Manager m = new Manager();
        m.getDetails();

        Employee p = new Manager();

        p.getDetails();
        p.getDepartment();
    }
}
```

Employee type
Manager instance

範例

```
public class Employee {
    private String name = “Sean”;
    private double salary = 10000;
    public void getDetails() {
        System.out.println(“Name:” + name);
        System.out.println(“Salary:” + salary);
    }
}
```

```
public class Manager extends Employee {
    private String dept = “EDU”;
    public void getDetails() {
        super.getDetails();
        System.out.println(“Department:” + dept);
    }
    public void getDepartment() {
        System.out.println(“Department:” + dept);
    }
}
```

```
public class Test {
    public static void main(String [] args) {
        Employee e = new Employee();
        e.getDetail();
        Manager m = new Manager ();
        m.getDetail();

        Employee p = new Manager();

        p.getDetails();
        p.getDepartment();
    }
}
```

Compile-Time Type
Employee

範例

```
public class Employee {
    private String name = “Sean”;
    private double salary = 10000;
    public void getDetails() {
        System.out.println(“Name:” + name);
        System.out.println(“Salary:” + salary);
    }
}
```

```
public class Manager extends Employee {
    private String dept = “EDU”;
    public void getDetails() {
        super.getDetails();
        System.out.println(“Department:” + dept);
    }
    public void getDepartmentl() {
        System.out.println(“Department:” + dept);
    }
}
```

```
public class Test {
    public static void main(String [] args) {
        Employee e = new Employee();
        e.getDetail();
        Manager m = new Manager ();
        m.getDetail();

        Employee p = new Manager();

        p.getDetails();
    //  p.getDepartment();
    }
}
```

Run-Time Type
Manager

方法覆寫規則

- 覆寫時
  - 方法介面要一模一樣
  - 不可更改 static狀態
  - 不可覆寫 final method
  - 不可以降低可存取範圍
    - **存取權限修飾字只能相同或更寬鬆**
  - 不可以丟出更多的例外

存取權限與方法覆寫

```
public class Employee {
    private String name = “Sean”;
    private double salary = 10000;
    public void getDetails( ) {
        System.out.println(“Name:” + name);
        System.out.println(“Salary:” + salary);
    }
}
```

```
public class Manager extends Employee {
    private String dept = “EDU”;
    public void getDetails( ) {
        super.getDetails( );
        System.out.println(“Department:” + dept);
    }
}
```

```
public class Test {
    public static void main(String [ ] args) {
        Employee p = new Manager( );
        p.getDetails( );
    }
}
```

Run-Time Type
Manager

多型應用

- 在繼承關係架構下，物件實體可被視為多種型別
  - 物件多型

    父類別 **變數名稱 = new** 子類別建構子**( );**

  - 異質集合

    父類別 [ ] 集合名稱 = new 父類別 [3];
    集合名稱[0] = new 子類別建構子**( );**

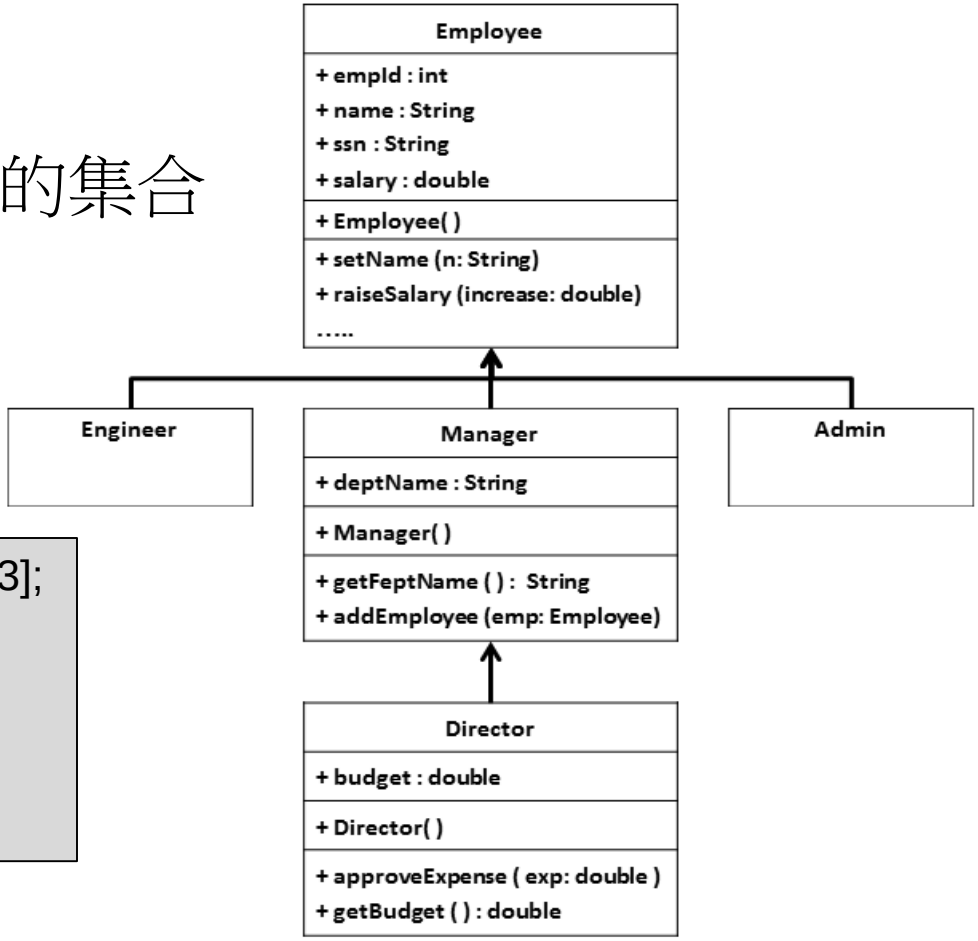
  - 多型參數

```
傳回值 方法名稱(父類別 參數名稱) {
        ....
}
```

物件.方法名稱(子類別參數);

異質集合

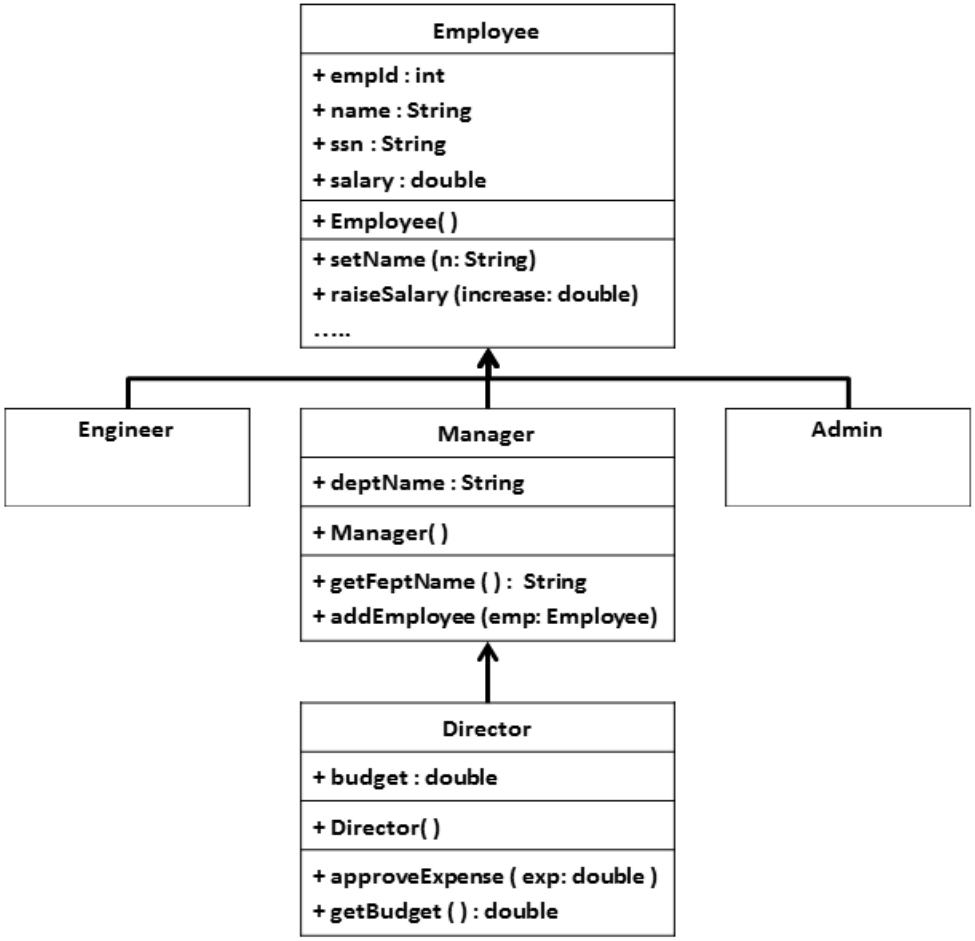
- 異質集合
  - 一群類別不相同物件的集合
  - 需有共同的祖先類別




```
Employee [ ] staff = new Employee[3];
staff[0] = new Manager( );
staff[1] = new Engineer( );
staff[2] = new Admin( );
```

多型參數範例

```
public class EmployeeStockPlan {
    private float stockMultiplier = 1.5;
    public int grantStock(Director d){
        return stockMultiplier * 10000;
    }
    public int grantStock(Manager m){
        return stockMultiplier * 5000;
    }
    public int grantStock(Engineer e){
        return stockMultiplier * 3000;
    }
    public int grantStock(Admin a){
        return stockMultiplier * 1000;
    }
}
```



多型參數範例

```
public class EmployeeStockPlan {
    private float stockMultiplier = 1.5;
    public int grantStock(Employee e){
        return (int)(stockMultiplier * e.calculateStock( ));
    }
}
```

```
public class EmployeeApp {
    public static void main(String[] args) {
        EmployeeStockPlan esp = new EmployeeStockPlan();
        Manager m  = new Manager( );
        Engineer e = new Engineer( );
        Admin a = new Admin( );
        int stockGranted1 = esp.grantStock(m);
        int stockGranted2 = esp.grantStock(e);
        int stockGranted3 = esp.grantStock(a);
    }
}
```

```
public class Employee {
    public int calculateStock(){
        return 1000;
    }
}
```

```
public class Manager extends Employee {
    public int calculateStock( ){
        return 5000;
    }
}
```

```
public class Engineer extends Employee {
    public int calculateStock( ){
        return 3000;
    }
}
```

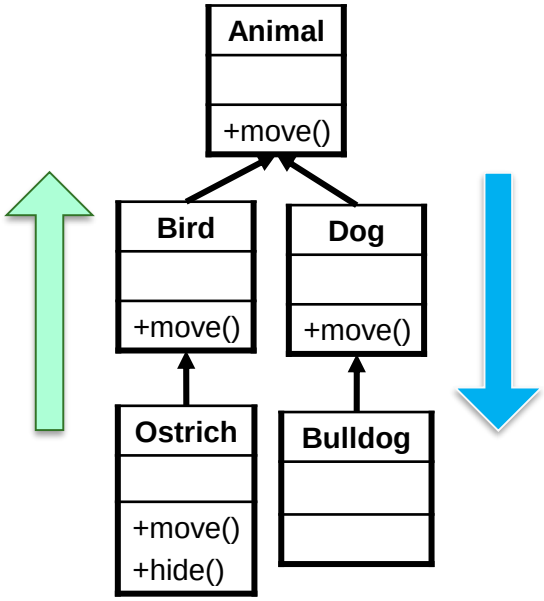
```
public class Admin extends Employee {

}
```

```
public class Director extends Manager {
    public int calculateStock( ){
        return 10000;
    }
}
```

參考型別轉型

- 參考型別轉型
  - 多型操作是一種型別的自動 (隱含) 轉換 (Implicit Casting)
    - 也稱為晉升 (Promotion)
    - 向上(Upward) 轉型：
      - 子類別物件轉型為父類別變數
    - 轉型過程會造成資料或行為隱藏
  - 參考型別強制轉換 (Explicit Casting)
    - 向下 (Downward) 轉型：
      - 父類別變數轉型為子類別物件
    - 還原物件完整資料及功能操作
    - 不當轉型會有編譯錯誤或執行時期例外



強制轉型

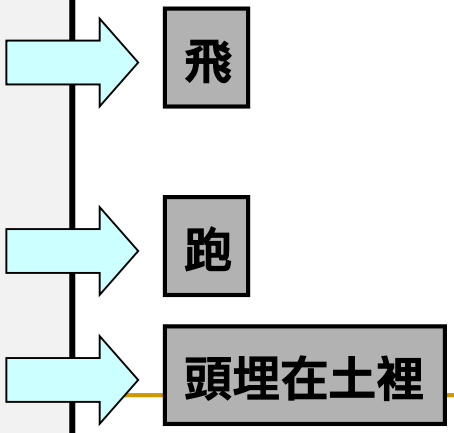
- 強制轉型
  - 將被宣告為父類別的子類別物件轉型回子類別
  - (目標類別名稱)物件名稱;

- 藉由轉型來解決呼叫hide()方法的問題。

```
Bird bird1 = new Bird();
bird1.move();

Bird bird2 = new Ostrich();
bird2.move();

((Ostrich) bird2).hide()
```



instanceof 運算子

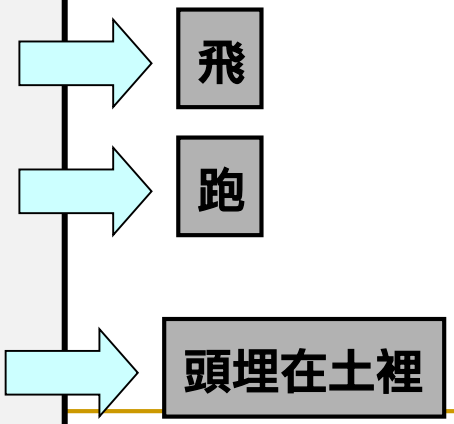
- instanceof 運算子

  <物件名稱> instanceof <類別名稱>

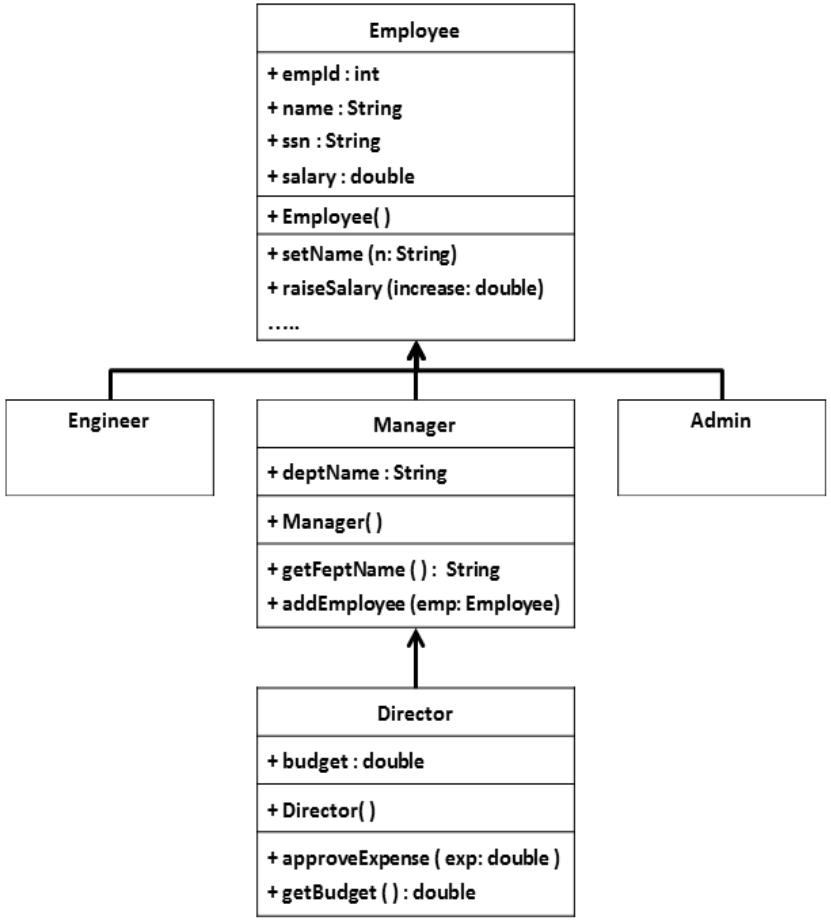
  - 回傳布林值
    - true：該變數所參考的物件可以轉換成特定類別
    - false：反之則否
  - 確認物件是否為某種類別型態

```
Bird bird1 = new Bird();
bird1.move();

Bird bird2 = new Ostrich();
bird2.move();
if(bird2 istanceof Ostrich) {
    ((Ostrich) bird2).hide();
}
```



instanceof 範例




```
Manager m1 = new Manager( );
Director d1 = new Director( );

Employee e1 = m1;
Employee e2 = d1;

Manager m2 = (Manager)e1;
Director d2 = (Director) e2;

Engineer eng1 = (Engineer) m1;
```

編譯失敗

```
Engineer eng2 = (Engineer) e1;
```

編譯成功，執行時轉型失敗

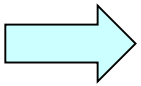
課程大綱

1) 繼承
2) 方法覆寫
3) 多型
4) **Object類別的方法**
   - **equals**
   - **hashcode**
   - **toString**

Java.lang.Object 類別

- Java語言中，Object類別是所有類別的根類別
  - 類別未宣告繼承類別，JVM自動加入extends Object

```
public class Employee {
    ...
}
```



```
public class Employee extends Object {
    ...
}
```

  - 定義所有類別都該有的特性及操作
  - Programmer根據需要改寫
- 常用方法
  - public boolean equals(Object obj) {…}
  - public int hashCode() {...}
  - public String toString() {…}

java.lang

Class Object

java.lang.Object

`public class Object`

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

See Also:

`Class`

Constructor Summary

Constructors

Constructor and Description
`Object()`

Methods

Modifier and Type	Method and Description
`protected Object`	`clone()` Creates and returns a copy of this object.
`boolean`	`equals(Object obj)` Indicates whether some other object is "equal to" this one.
`protected void`	`finalize()` Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
`Class<?>`	`getClass()` Returns the runtime class of this `Object`.
`int`	`hashCode()` Returns a hash code value for the object.
`void`	`notify()` Wakes up a single thread that is waiting on this object's monitor.
`void`	`notifyAll()` Wakes up all threads that are waiting on this object's monitor.
`String`	`toString()` Returns a string representation of the object.

equals() 方法

- equals() 方法
  - 提供的兩個物件實際內容相等的邏輯
  - 若未覆寫此方法，Object類別預設使用”==”比較
    - “==”：兩變數是否參考同一物件(比較stack中變數)
  - 使用者自訂類別通常需依據類別之商業邏輯，覆寫equals()方法
  - 覆寫equals()方法，也需覆寫hashcode()方法

- 方法介面
  - public boolean equals(Object obj) {…}

hashCode()方法

- hashCode規則
  - 相同物件的hashCode，也必須相同。
  - 若兩個物件hashCode不同，其內容必定不同。
  - hashCode相同，不表示內容相同。

hashCode()方法

- hashCode 用途
  - 物件比較前期檢查
    - 呼叫equals()前，先呼叫hashCode()
      - hashCode不同，不必呼叫equals()，兩物件必定不同。
      - hasCode相同，仍需呼叫equals()確認比較結果。
    - hashCode()效能>>>equals()效能
      - 有效的 hashCode，減少呼叫 equals( )，效能可大幅提升
  - hash table是以key.hashCode()取得的hashCode value (雜湊值)

覆寫 hashCode()

- 方法介面
  - public int hashCode() {...}
  - 自行覆寫hashCode的方法
    - 取得用於equals方法內的屬性,將其hashCode傳回
    - 有數個屬性用於equals方法時,將數個屬性的hashCode作XOR(^)傳回
    - 對於基本資料型別的屬性,用其warpper包覆類別來取得hashCode
  - 使用 IDE 工具自動產生

toString() 方法

- toString() 方法
  - 提供物件的字串表示法
  - 需要將物件轉換成字串時呼叫
    - System.out.println(obj);
    - “str” + obj;
  - 若未覆寫此方法，預設傳回類別名稱及hashcode

- 方法介面
  - public String toString() {…}

```java
public class MyDate {
    private int day;
    private int month;
    private int year;
    public MyDate(int day, int month, int year){
        this.day = day;
        this.month = month;
        this.year = year;
    }
    @Override
    public boolean equals(Object o) {
        boolean result = false;
        if((o != null)&& (o instanceof MyDate)) {
            MyDate d = (MyDate) o;
            if((day==d.day)&&(month==d.month)
              &&(year==d.year)) {
                result = true;
            }
        }
        return result;
    }
    @Override
    public int hashCode(){
        return (day<<4 ^ month ^ year);
    }
    @Override
    public String toString(){
        return day + "/" + month + "/" + year;
    }
}
```

```java
public class TestObjMethods {
    public static void main(String[] args) {
        MyDate date1 = new MyDate(14, 3, 1976);
        MyDate date2 = new MyDate(14, 3, 1976);
        System.out.println("date1 : " + date1);
        System.out.println("date2 : " + date2);
        if(date1 == date2){
            System.out.println("date1 is identical to date2");
        } else {
            System.out.println("date1 isn't identical to date2");
        }
        if(date1.equals(date2)){
            System.out.println("date1 is equal to date2");
        } else {
            System.out.println("date1 is not equal to date2");
        }
        System.out.println("set date2 = date1");
        date2 = date1;
        if(date1 == date2){
            System.out.println("date1 is identical to date2");
        } else {
            System.out.println("date1 isn't identical to date2");
        }
    }
}
```