Java程式設計進階
繼承

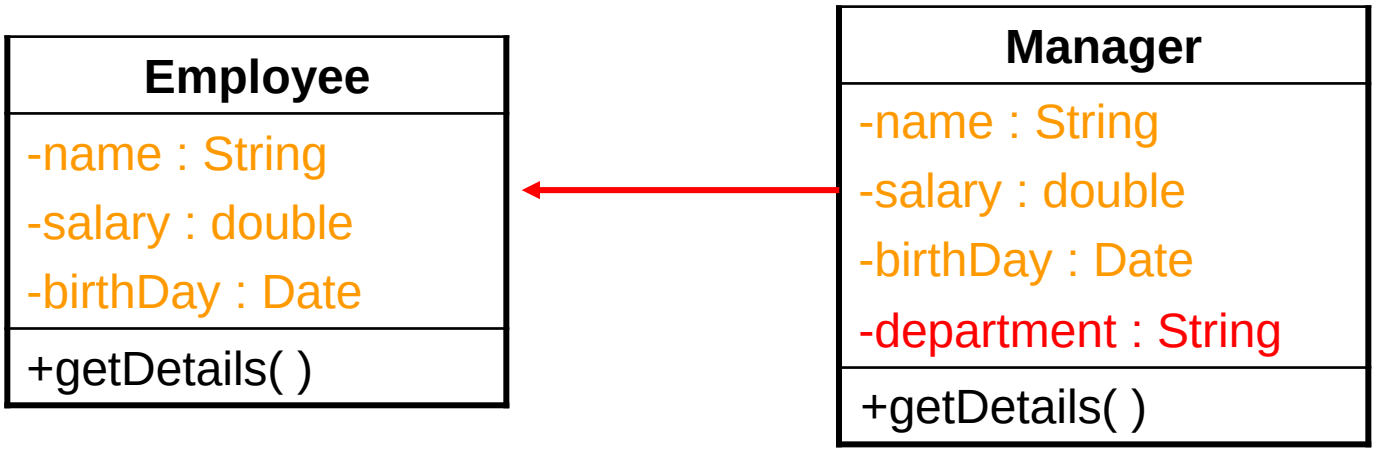
鄭安翔
ansel_cheng@hotmail.com

課程大綱

1) **繼承**
   - **繼承的觀念**
   - **繼承實作**
   - **繼承關係下的物件建構**
2) 方法覆寫

類別的繼承

- 繼承讓類別的程式碼可以延伸及重複使用
  - 父類別 base class/superclass
  - 子類別 derived class/subclass
  - 類別延伸(extends class)的關係是 ”is a” 的概念。



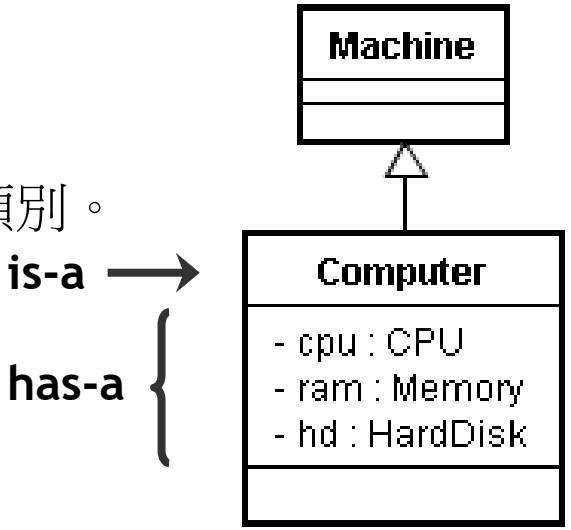
A Manager is an Employee

繼承機制的優點

- 產生更特殊性的型別
- 消除重複的程式碼
- 提高程式的可維護性

is-a 與 has-a

- **is a (是一個)：**
  - 延伸的關係。
  - Java 語言利用 extends 關鍵字來實作
  - EX：電腦是一種電子機械產品
    - ✓ 電腦類別繼承了電子機械產品類別。
    - ✓ 電子機械產品是父類別，而電腦則是子類別。

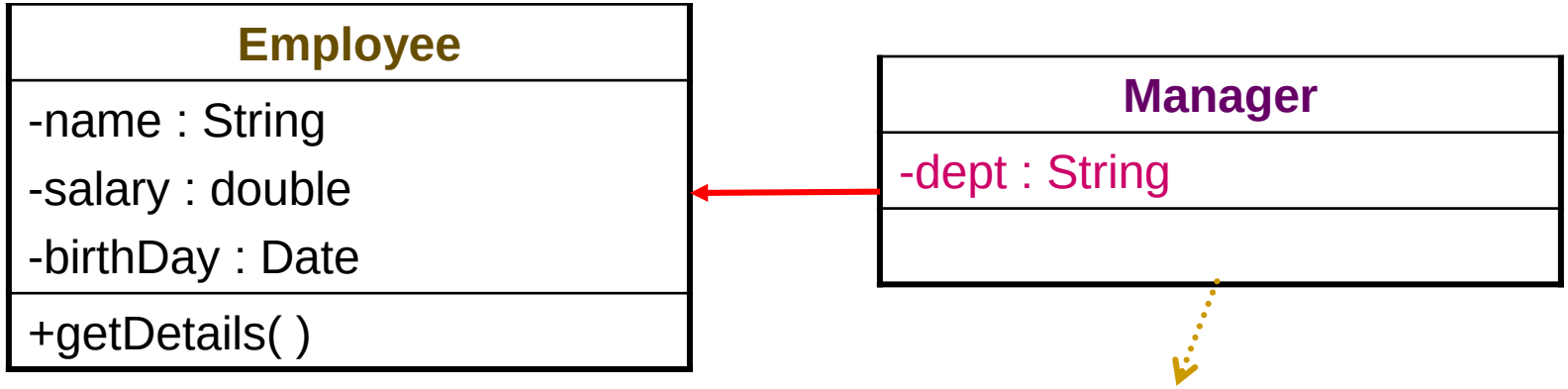


- **has a (有一個) ：**
  - 聚合的關係。
  - 類別中的成員變數(member variable)來表示。
  - EX：電腦中有 CPU、256G RAM、4TB HD
    - ✓ CPU、RAM 與 HD 便成了電腦的成員變數。

類別繼承語法

```
[modifiers] class 子類別名稱 extends 父類別名稱 {
    // 類別內容
}
```

Java 技術實作繼承



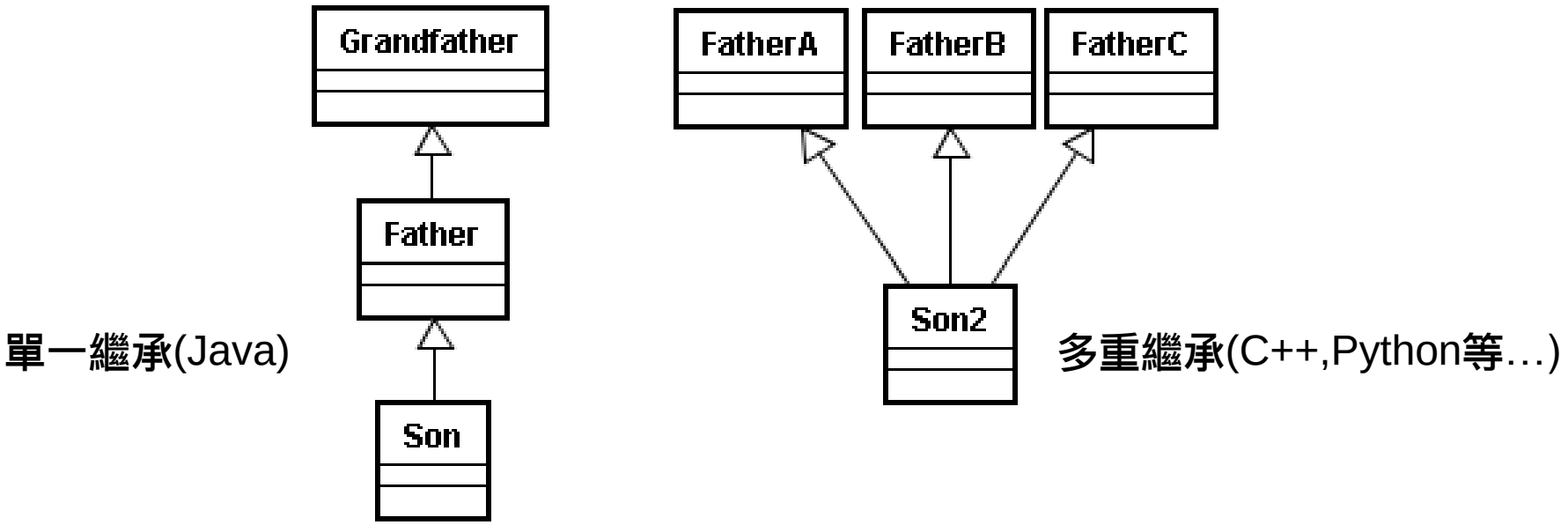

```
01  public class Employee {
02      private String name = “Sean”;
03      private double salary = 10000;
04      public void getDetails( ) {
05          System.out.println(“Name:” + name);
06          System.out.println(“Salary:” + salary);
07      }
08  }
```

```
01  public class Manager extends Employee {
02      private String dept = “EDU”;
03
04  }
05
```

Manager 繼承了 Employee 所有成員;
屬性:name, salary, birthDay
方法:getDetails()方法

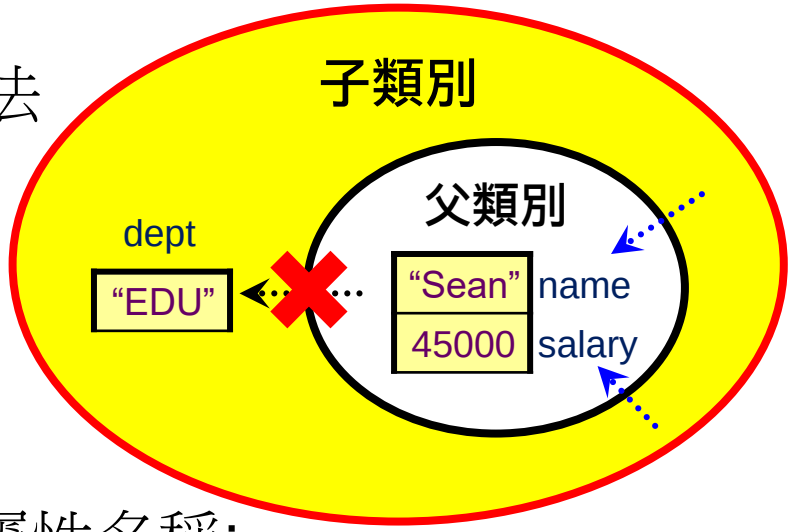
單一繼承

- Java 語言在繼承上只允許單一繼承(Single Inheritance)關係。
  - 子類別在定義繼承的關係時，只能針對單一父類別做延伸，不能同時使用來自多個父類別的資源。



繼承中的資源使用

- 當子類別繼承父類別，擁有父類別中的資源
  - 子類別繼承(擁有)父類別成員(屬性和方法)
    - 實作：子類別中包含父類別
  - 父類別不擁有子類別屬性和方法
  - 建構子不會被繼承



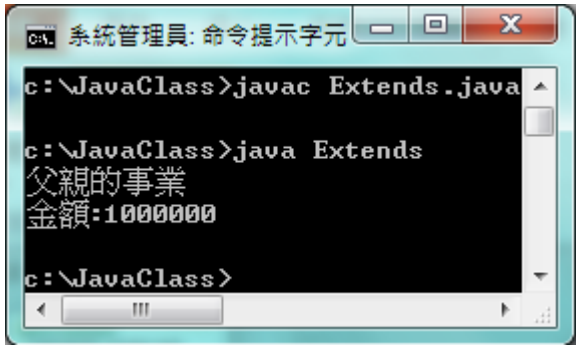
- 子類別存取父類別資源
  - 存取父類別的屬性：直接使用屬性名稱;
  - 呼叫父類別的方法:直接使用方法名稱(參數列);
  - 但存取權限仍受限於父類別的存取修飾字

繼承範例

```
01  class Father {
02      public int money = 1000000;
03      public void undertaking() {
04          System.out.println("父親的事業");
05      }
06  }
```

```
01  class Son extends Father{
02
03  }
```

```
01  public class Extends {
02      public static void main(String[] args) {
03          Son son = new Son();
04          son.undertaking();
05          System.out.println("金額:" + son.money);
06      }
07  }
```



繼承關係下的建構子

- 子類別產生物件時,需先建構父類別物件
  - 先配置父類別所需記憶體，再配置子類別記憶體
  - 先初始化父類別的屬性，才執行子類別初始化
  - 先完成父類別建構式，再完成子類別建構式
- 預設建構子
  - 程式中若沒有定義建構子，在編譯時期會自動加入
  - 沒有參數列 (no arguments)
  - 包含 super( ) 語法，呼叫父類別無參數建構子

Default Constructor

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

```
public class Manager extends Employee {
    private String dept = “EDU”;
    public Manager() {
        super();
    }
}
```

Manager m = new Manager();

繼承關係下的建構子

- 指定建構子
  - super 與 this 關鍵字
    - super → 父類別
    - this → 本類別
  - super(參數列)
    - 子類別建構子中指定呼叫父類別的特定建構子
  - this(參數列)
    - 類別建構子中呼叫本類別其他建構子

繼承關係下的建構子

- 建構子使用 super( ) / this( ) 注意事項
  - 只能用在建構子之中，不能在程式其他位置出現
  - 只能用在建構子程式碼第一行，一次只能使用一種
  - 建構子之中，若沒有使用 super( ) 或 this( )，編譯器自動在第一行加上 super( )

Constructors & super()

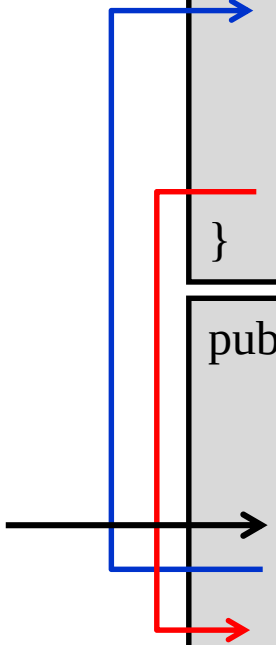
```
public class Employee {
    private String name = "Sean";
    private double salary = 10000;

    public Employee(String n, double s) {
        name  = n;
        salary= s;
    }
}
```

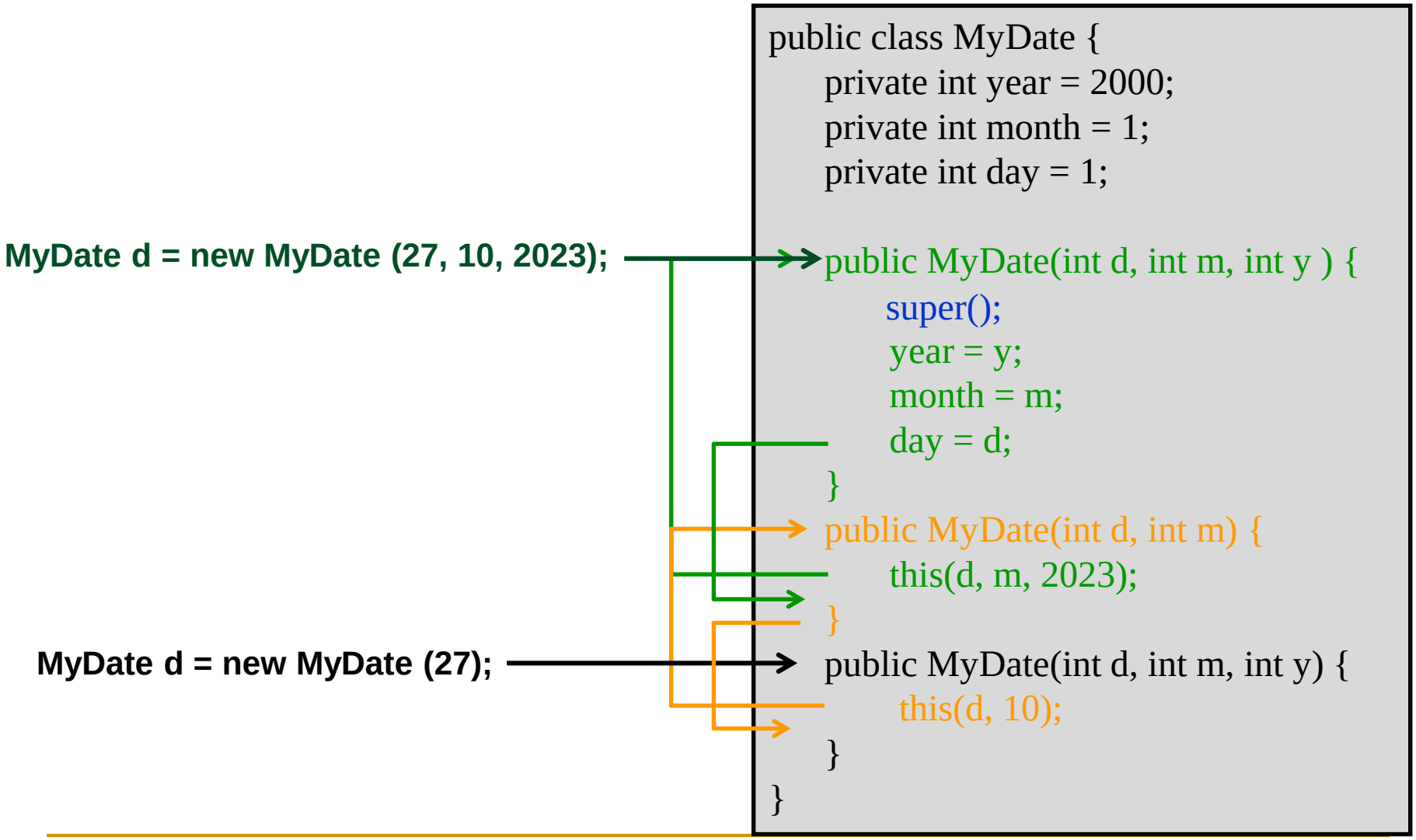
```
public class Manager extends Employee {
    private String dept = "EDU";

    public Manager(String n, double s, String d) {
        super(n, s);
        dept = d;
    }
}
```

Manager m = new Manager ("Sean", 50000.0, "EDU");



Constructors overloading & this()



Default Constructor 編譯錯誤

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

沒有無參數
建構子

Manager m = new Manager( ); →

```
public class Manager extends Employee {
    private String dept = “EDU”;
    public Manager( ) {
        super( );
    }
}
```

Default Constructor 編譯錯誤

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

Manager m = new Manager (“Sean”, 50000.0, “EDU”);

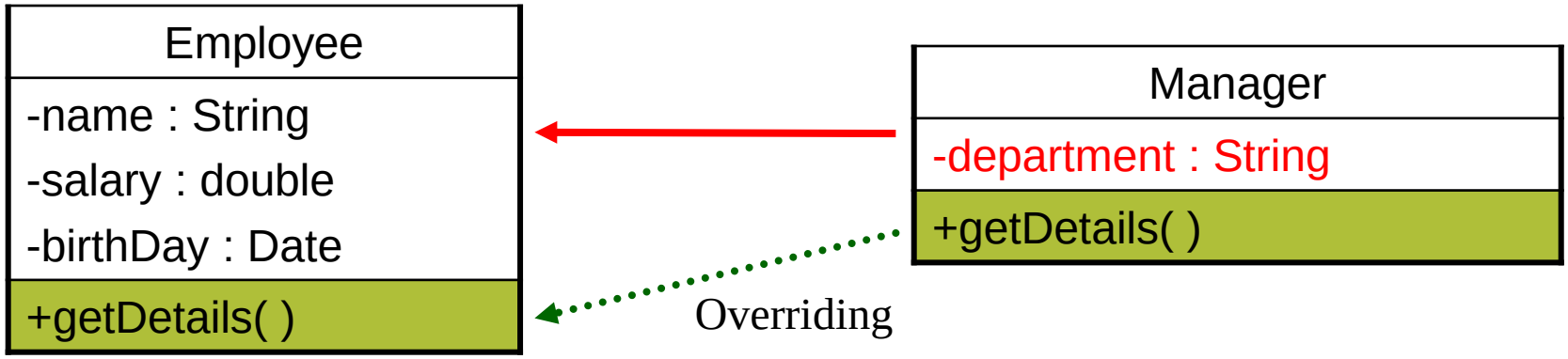
```
public class Manager extends Employee {
    private String dept = “EDU”;
    public Manager(String n, double s, String d) {
        super( );
        name = n;
        salary = s;
        dept = d;
    }
}
```

課程大綱

1) 繼承
2) **方法覆寫**
   - **方法覆寫**
   - **protected 權限**
   - **子類別屬性遮蔽**
   - **final 關鍵字**
3) 多型
4) Object類別的方法

方法覆寫 Overriding Methods

- 方法覆寫 Method Override
  - 子類別改寫父類別中相同的名稱及參數列的方法



方法覆寫規則

- 覆寫時
  - 方法介面要一模一樣
    - 相同的方法名稱
    - 相同的傳入參數列
    - 相同的傳回型態 (同類別或子類別)
    - 可用 @ **Override** 標註(**Annotation**): 防止不正確的覆寫
  - 不可更改 **static**狀態
  - 不可覆寫 final method
  - 不可以降低可存取範圍
  - 不可以丟出更多的例外

方法覆寫 Method Override

```
01  public class Employee {
02   protected  String name = “Sean”;
03   protected  double salary = 10000;
04       public void getDetails( ) {
05              System.out.println(“Name:” + name);
06              System.out.println(“Salary:” + salary);
07      }
08  }
```

```
01  public class Test {
02      public static void main(String [ ] args) {
03              Employee e = new Employee();
04              e.getDetails( );
05              Manager m = new Manager ();
06              m.getDetails( );
07      }
08  }
```

```
01  public class Manager extends Employee {
02       private String dept = “EDU”;
03       @Override
04       public void getDetails( ) {
05            System.out.println(“Name:” + name);
06            System.out.println(“Salary:” + salary);
07            System.out.println(“Department:”+dept);
08     }
09  }
```

protected / default 權限範例

同套件類別可存取或
不同套件的子類別也可存取
僅同套件類別可存取

```
01  package demo;
02  public class Foo {
03      protected int result = 200;
04      int other = 25;
05  }
```

```
01  package test;
02  import demo.Foo;
03  public class Bar extends Foo {
04      private int sum = 10;
05      public void reportSum( ){
06          sum += result;
07          sum += other;
08      }
09  }
```

Compile Error

存取權限修飾字

- 存取權限修飾字(Modifier)是用來宣告類別、屬性與方法(含建構子)可被存取的權限,分成四個等級：

存取權限修飾字	**權限說明**
private	**同一個 class 才可存取**
default **無修飾字**	**同一個 package 的class 才可存取**
protected	**同一個 package 的class才可存取** **不同package 的要有繼承關係才可存取**
public	**皆可存取**

  - 類別 (class) ：只能使用 **public** 與 **default**(無修飾字)
  - 屬性、方法與建構子：四種修飾字皆可以使用

存取權限修飾字

- 存取權限修飾字的可視範圍：

存取權限修飾字	同類別	同套件中 其他類別	不同套件 子類別	不同套件 非子類別
private	Yes	-	-	-
default **無修飾**字	Yes	Yes	-	-
protected	Yes	Yes	Yes	-
public	Yes	Yes	Yes	Yes

呼叫被覆寫的方法 - super

- **super** 關鍵字
  - 子類別物件中欲參考父類別物件的屬性、方法及建構子
    - super.屬性
    - super.方法(參數列)
    - super(參數列)
  - super關鍵字必須在繼承關係的運作下才有意義。

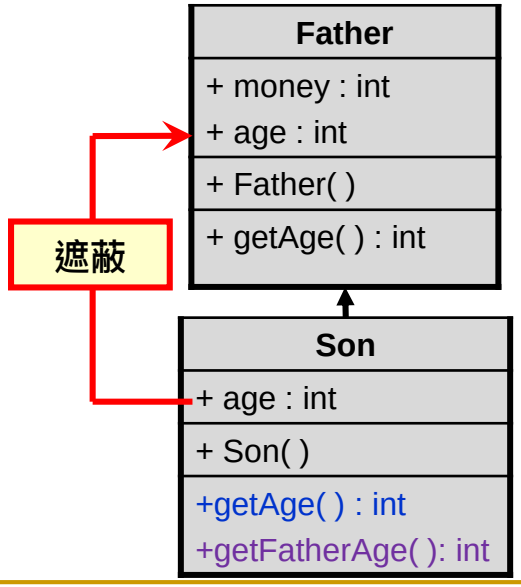
```
public class Employee {
    private String name = “Sean”;
    private double salary = 10000;
    public void getDetails( ) {
        System.out.println(“Name:” + name);
        System.out.println(“Salary:” + salary);
    }
}
```

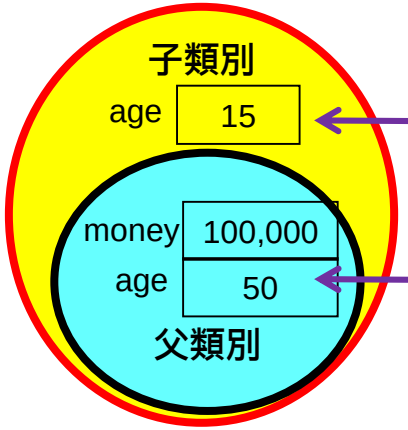
```
public class Manager extends Employee {
    private String dept = “EDU”;
    public void getDetails( ) {
        super.getDetails( );

        System.out.println(“Department:”+dept);
    }
}
```

子類別屬性遮蔽

- 子類別屬性遮蔽
  - 子類別定義與父類別同名的屬性，產生遮蔽現象。
  - 可利用 super 關鍵字來取用父類別的屬性。






```
Son s = new Son( );
 s.getAge( );
 s.getFatherAge( );
public int getAge( ){
   return age;
}
public int getFatherAge( ){
   return super.age;
}
```

範例 - 遮蔽屬性

```
01 package demo;
02 public class Foo2 {
03     protected int result = 20;
04     int other = 25;
05 }
```

```
01 package test;
02 import demo.Foo2;
03 public class Bar2 extends Foo2 {
04     private int sum = 10;
05     private int result = 30;
06     public void reportSum( ){
07         sum += result;  sum = 40
08     }
09 }
```

final 關鍵字

- final 關鍵字
  - 表示被修飾元素是不可變更(immutable)的
  - 可用於修飾類別, 變數及方法
- final classes
  - 不可被繼承
- final method
  - 不可被覆寫 (override)

```
public final class FinalParentClass {
    ......
}
```

```
public class ChildClass extends FinalParentClass {
    ......
}
```

compile time error

```
public class MethodParentClass {
    public final void printMessage() {
        System.out.println("This is a final method");
    }
}
```

```
public class MethodChildClass extends MethodParentClass {
    public void printMessage() {
        System.out.println("Cannot override method");
    }
}
```

compile time error

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
  - 不可以丟出更多的例外

final 屬性與變數

- final variable
  - 基礎資料型別Primitive type 被視為常數,無法更改
  - 參考資料型別Reference type之reference不能被改變(不能在指向其他物件),
    - 但物件內容可以被改變
  - 只能被設定一次
  - 空白常數 blank final variable
    - blank final class variable: 必須在靜態區塊 (static block) 中設定初始值
    - blank final instance variable:必須在建構子中設定初始值
    - blank final local variable：使用前必須先設定初始值

```
import java.util.Date;

public class VariableExampleClass {
    public static final int JAVA_CONSTANT;
    private final int field;
    private final int forgottenField;
    private final Date date = new Date();

    static {
        JAVA_CONSTANT = 10; // allowed
    }

    public VariableExampleClass() {
        field = 100;  // allowed
        // compile time error, variable forgottenField might not have been initialized
    }

    public void changeValues(final int param) {
        param = 1; // compile time error

        final int localVar;
        localVar = 42; // allowed
        localVar = 43; // compile time error

        date.setTime(0); // allowed
        date = new Date(); // compile time error
    }
}
```