Java 的類別種類

一般類別 (class)，抽象類別 (abstract class)， 介面(interface)

介面(interface)

1. 介面主要是用來定義一群行為動作的規格，它並不能直接用來產生物件，但能讓類別去 實作 它
2. 介面是由一些 常數 和 抽象方法 和 預設方法(8.0) 和 靜態方法(8.0) 所組成，介面沒有建構子

   <1>.常數➔public static final ➔若是 public 或 static 不寫 ➔Compiler 會自動補齊，常數必須要在定義時指定它的初始值

   <2>.無 body 的 抽象的方法➔不用寫 abstract，也不用寫 public ➔Compiler 系統會自動補齊➔但寫了也不會錯

```
//public abstract interface Interface1 {
//
//    public static final int x = 4;  //常數
//
//    public abstract void abc(); //抽象的方法
//
//    public abstract int xyz(); //抽象的方法
//
//    public default void sub1() { //預設方法
//        System.out.println("我是預設方法");
//    }
//    public static String fun1() { //工廠函數
//        return "我是工廠函數";
//    }
//}
```

```
public interface Interface1 {

    int x = 4;  //常數

    void abc(); //抽象的方法

    int xyz(); //抽象的方法

    default void sub1() {
        System.out.println("我是預設方法");
    }

    static String fun1() {
        return "我是工廠函數";
    }
}
```

```
class Sub1 implements Interface1 {

    public void abc() {
        System.out.println("abc");
    }

    public int xyz() {
        return 123;
    }
}
```

```
public static void 實作介面(){
    Interface1 p=new Sub1();
    p.abc(); //abc
    System.out.println(p.xyz()); //123
    p.sub1(); //我是預設方法
     System.out.println(Interface1.fun1()); //我是工廠函數
}
```

3. 繼承 (extends) 與實作 (implements) 的語法

普通類別➔有建構函數➔被子類別 super()， new

```
public class InterfaceDemo1P {//我是普通類別

    public int x = 5;

    public InterfaceDemo1P() { //建構函數
        super();
        x = 10;
    }
    public void s1() {

    }
    public void s2() {

    }
}
class InterfaceDemo1C extends InterfaceDemo1P {

    public float x = 4.5F;

    public InterfaceDemo1C() {//建構函數
        super();
        x = 8.5F;
    }
    public void s1() {

    }
}
```

抽象類別➔有建構函數➔被子類別 super()

```
abstract class InterfaceDemo2P {//我是抽象類別

    public int x = 5;

    public InterfaceDemo2P() {//建構函數
        super();
        x = 10;
    }

    public void s1() {

    }

    public abstract void s2(); //我是抽象的方法
}

class InterfaceDemo2C extends InterfaceDemo2P {

    public float x = 4.5F;

    public InterfaceDemo2C() {//建構函數
        super();
        x = 8.5F;
    }

    public void s1() {

    }

    public void s2() {

    }
}
```

```
public static void 普通類別() {
    InterfaceDemo1P p = new InterfaceDemo1P();
}
```

```
public static void 抽象類別() {
    // InterfaceDemo2P p=new InterfaceDemo2P();
    InterfaceDemo2P p = new InterfaceDemo2C();
}
```

介面➔無建構函數➔不需 new 也不需 super()

```
interface InterfaceDemo3P {//我是介面 ,我沒有建構函數

    int x = 4;  //我是常數 public static final int x=4

    void s1(); //我是抽象的方法 public abstract void s1();

    void s2(); //我是抽象的方法 public abstract void s2();
}

class InterfaceDemo3C implements InterfaceDemo3P {

    public float x = 4.5F;

    public InterfaceDemo3C() {//建構函數
        super();
    }
    public void s1() {

    }
    public void s2() {

    }
}
```

```
public static void 介面() {
    //  InterfaceDemo3P p=new InterfaceDemo3P();
    InterfaceDemo3P p = new InterfaceDemo3C();
}
```

三種繼承與實作關係

父類別	父介面	父介面	父類別
extends	extends	implements	無此關係
子類別	子介面	子類別	子介面

父類別與子類別

```
class InterfaceDemo4P1 {
    public int x;
    public void s1() {

    }
}
class InterfaceDemo4P2 {
    public int y;
    public void s2() {

    }
}
//父子類別不能多重繼承
class InterfaceDemo4C extends InterfaceDemo4P1 {
    public int a;
    public void m1() {

    }
}
```

父介面與子介面

```
interface InterfaceDemo5P1 {

    int x = 4;

    void s1();
}
interface InterfaceDemo5P2 {

    int y = 4;

    void s2();
}
//父子介面可以多重繼承
interface InterfaceDemo5C extends InterfaceDemo5P1, InterfaceDemo5P2 {

    void m1();
}
```

父介面與子類別

```
interface InterfaceDemo6P1 {

    int x = 4; //常數

    void s1(); //抽象方法
}
interface InterfaceDemo6P2 {

    int y = 5; //常數

    void s2(); //抽象方法
}
//父介面與子類別==>可以多重實作
class InterfaceDemo6C implements InterfaceDemo6P1, InterfaceDemo6P2 {

    public void s1() {

    }
    public void s2() {

    }
}
```

父類別與子介面➔無此關係

```
class InterfaceDemo7P {

    public void s1() {
    }
}
//父類別與子介面==>無此關係
interface InterfaceDemo7C extends InterfaceDemo7P {

    final int x = 4; //常數

    void s1(); //抽象方法
}
```

繼承➔只能單一繼承，實作➔可以多重實作，可以又繼承又實作

```
interface InterfaceDemo8P {

    void p1();
}
interface InterfaceDemo9P {

    void p2();
}
interface InterfaceDemo10P extends InterfaceDemo8P, InterfaceDemo9P {
// P1() , P2()

    void s1();

    void s2();
}
interface InterfaceDemo11P {

    void s3();
}
class InterfaceDemo12P {

    public void n1() {
    }
}
```

```
201     class InterfaceDemo13P extends InterfaceDemo12P implements InterfaceDemo10P, InterfaceDemo11P {
202
204         public void s1() {
205
206         }
207         public void s2() {
208
209         }
210         public void p1() {
211
212         }
213         public void p2() {
214
215         }
216         public void s3() {
217
218         }
    }
```

繼承 還是 實作?

```java
//錯誤的定義
public abstract class Fish9 {//魚

    public abstract void swim();
}

class Anemonefish9 extends Fish9 {//小丑魚

    public void swim() {
        System.out.println("小丑魚游泳");
    }
}

class Shake9 extends Fish9 {//鯊魚

    public void swim() {
        System.out.println("鯊魚游泳");
    }
}

class Piranha9 extends Fish9 {//食人魚

    public void swim() {
        System.out.println("食人魚游泳");
    }
}

class Human9 extends Fish9 { //人

    public void swim() {
        System.out.println("人游泳");
    }
}

class Submarine9 extends Fish9 {//潛水艇

    public void swim() {
        System.out.println("潛水艇游泳");
    }
}
```

```java
//正確的定義
public interface Swimmer10 {

    public void swim();
}
class Anemonefish10 implements Swimmer10 {//小丑魚

    public void swim() {
        System.out.println("小丑魚游泳");
    }
}
class Shake10 implements Swimmer10 {//鯊魚

    public void swim() {
        System.out.println("鯊魚游泳");
    }
}
class Piranha10 implements Swimmer10 {//食人魚

    public void swim() {
        System.out.println("食人魚游泳");
    }
}
class Human10 implements Swimmer10 { //人

    public void swim() {
        System.out.println("人游泳");
    }
}
class Submarine10 implements Swimmer10 {//潛水艇

    public void swim() {
        System.out.println("潛水艇游泳");
    }
}
```

```java
public static void 游泳1() {//錯誤的定義
    Fish9 swimmer;
    swimmer = new Anemonefish9();
    swimmer.swim();
    swimmer = new Shake9();
    swimmer.swim();
    swimmer = new Piranha9();
    swimmer.swim();
    swimmer = new Human9();
    swimmer.swim();
    swimmer = new Submarine9();
    swimmer.swim();
}
```

```java
public static void 游泳2() {//正確的定義
    Swimmer10 swimmer;
    swimmer = new Anemonefish10();
    swimmer.swim();
    swimmer = new Shake10();
    swimmer.swim();
    swimmer = new Piranha10();
    swimmer.swim();
    swimmer = new Human10();
    swimmer.swim();
    swimmer = new Submarine10();
    swimmer.swim();
}
```