# JAVA Programming Language Homework V: Overall Review

**ID:** **Name:**

1. Given the following Java code: [5 points]

```
public class SimpleCalc {
    public int value;
    public void calculate(){ value = value + 7; }
```

And:

```
public class MultiCalc extends SimpleCalc{
    public void calculate() { value =value – 3;}
    public void calculate( int multiplier) {
        calculate();
        super.calculate();
        value = value* multiplier;
    }
    public static void main(String[] args){
        MultiCalc calculator = new MultiCalc();
        calculator. Calculate(2);
        System.out.println(“Value is: “ + calculator.value);
    }
}
```

What is the result?

(A) Value is: 8

(B) Compilation fails

(C) Value is: 12

(D) Value is: -12

(E) The code runs with no output

**ANS:__ (A)__**

從 MultiCalc 類別的 main() 開始，建構出 MultiCalc()，
無傳參數的建構子，呼叫 calculate(int) 方法
再呼叫本類別的 calculate()，使 value = 0 – 3 = -3
再呼叫父類別的 calculate()，使 value = -3 + 7 = 4
最後將 value x 傳遞的參數值 2 = 4 * 2 = 8
再度返回 main() 方法，將該 value 屬性印出 8。

2. Given the following Java code: [10 points]

```
class Animal { public String noise () { return “peep”} }
   class Dog extends Animal {
      public String noise () { return “back”; }
}
class Cat extends Animal {
   public String noise () { return “move”; }
}
. . .
Animal animal = new Dog();
Cat cat = ( Cat ) animal;
System.out.printIn( cat.noise() ) ;
```

What is the result?

A. peep

B. back

C. move

D. Compilation fails.

E. An exception is thrown at runtime

**ANS:__ (E)__**

- 存在一個 Animal 類別
- Dog 繼承 Animal; Cat 也繼承 Animal
- 宣告一個 Animal 的 animal 物件, 是由 Dog() 建構出來的.
- 再宣告一個 Cat 的 cat 物件，是由 animal 物件強制轉型為 Animal 型別而來的。
- 從語法角度來看，Cat is-a Animal，所以 animal 可以強制轉型成 Cat。
- 但是執行時期，animal 是由 Dog 建構出來的實體，Dog is not a Cat，所以會發生轉型例外(java.lang.ClassCastException)。

3. Given the following Java code: [5 points]

```
      public class Bootchy {
            int botch;
            String snootch;

            public Bootchy() {
                this(“snootchy”);
                System.out.print(“first ”);
        }
        public Bootchy(String snootch) {
                this(420, “snootchy”);
                System.out.print(“second ”);
        }
        public Bootchy(int bootch, String snootch) {
                this.bootch=botch;
                this.snootch = snootch;
                System.out.print(“third ”);
        }
        public static void main(String[] args){
                Bootchy b = new Bootchy();
                System.out.print(b.snootch +” “+ b.bootch);
         }
    }
```

What is the result?

(A) snootchy 420 third second first

(B) snootchy 420 first second third

(C) first second third snootchy 420

(D) third second first snootchy 420

(E) third first second snootchy 420

**ANS:___ (D)___**

- 答案選項中沒有提及是否編譯成功，就可以略過語法檢查，直接跑程式
- 建構出 b，以無傳參數的建構子建構 public Bootchy()
- this( "snootchy" ); 呼叫本類別建構子 public Boothchy( String snootch )
- this( 420, "snootchy" ); 呼叫本類別建構子 public Bootchy(int bootch, String snootch )
- bootch = 420; snootch = "snootchy"，印出 "third"
- 再度返回 public Boothchy( String snootch )，印出 "second "
- 再度返回 public Bootchy()，印出 "first"
- 最後回到 main()，印出 "snootchy" 與 420。

4. Given the following Java code: [10 points]

```
class Test {
        static void alpha() { /* more code here */ }
        void beta(){ /* more code here */ }
}
```

Which two statements are true? (Choose two)

(A) Test.beta() is a valid invocation of beta()

(B) Test.alpha()is a valid invocation of alpha()

(C) Method beta() can directly call method alpha()

(D) Method alpha()can directly call method beta()

**ANS:__ (B), (C)__**

- alpha() 是 static method; 所以直接 Test.alpha() 即可呼叫使用
- beta() 是 non-static method; 則必須建構物件實體才可呼叫使用
- 在 beta() 中可以直接呼叫使用相同類別之 static method
- 但是 static methos 則無法直接呼叫使用，前提必須要有該物件實體。

5. Given the following Java code: [10 points]

```
public abstract class shape {
        private int x;
        private int y;
        public abstract void draw();
        public void setAnchor(int x, int y) {
                this.x=x ;
                this.y=y ;
        }
}
```

Which two classes use the Shape class correctly (choose two)

```
(A)  public class Circle implements Shape {
             private int radius;
     }
(B)  public abstract class Circle extends Shape {
             private int radius;
     }
(C)  public class Circle extends Shape {
```

```
            private int radius;
            public void draw();
        }
(D)    public abstract class Circle implements Shape {
            private int radius;
            public void draw();
        }
(E)    public class Circle extends Shape {
            private int radius;
            public void draw() { /* code here*/ }
        }
```

**ANS:__ (B), (E)__**

**抽象類別, 存在一個抽象方法**

A: implements 就錯誤了，因為是抽象類別而不是介面 interface

B: extends 抽象方法，因為仍未將抽象方法實做，所以繼續抽象，正確.

C: extends 抽象方法，並未宣告抽象，又未將父類別之抽象方法實做，錯誤

D: implements 就錯誤了，因為是抽象類別而不是介面 interface

E: extends 抽象方法，並未宣告抽象，有將父類別之抽象方法實做，所以正確

6. Given the following Java code: [5 points]

```
1.     class Pizza {
2.        java.util.ArrayList toppings;
3.        public final void addTopping(String topping) {
4.           toppings.add(topping);
5.        }
6.     }
7.     public class PepperoniPizza extends Pizza {
8.        public void addTopping(String topping) {
9.           System.out.println("Cannot and Uoppings");
10.       }
11.       public static void main(String[] args) {
12.          Pizza pizza = new PepperoniPizza();
13.          Pizza.addTopping("Mushrooms");
14.       }
15.        }
```

What is the result ?

A. Compilation fails

B. Cannot and Uoppings

C. The code runs with no output

D. A NullPointerException is thrown in Line 4

**ANS:__ A___**

- 答案選項有編譯失敗，所以一定要先檢查語法上的問題
- 很明顯的狀況是，Pizza 中有一個 final 的 addTopping() 方法
- 而子類別 PepperoniPizza 竟然想要 override 其 addTopping() 方法，當然不被允許。

7.        Given the following Java code: [5 points]

```
1.        class One {
2.           void foo() {}
3.        }
4.        class Two extends One {
5.           // insert method here
6.        }
```

Which three methods, inserted individually at line 5 will correctly class Two?

A. int foo(){/*more code here */}

B. void foo(){/*more code here */}

C. public void foo(){/*more code here*/}

D. private void foo(){/*more code here*/}

E. protected void foo(){/*more code here*/}

**ANS:___ B, C, E__**

- 從所有答案選項來看，這題是考 override 相關概念
- 存取修飾字要比原來相等或是更廣，所以 D 不對
    - public
    - protected
    - (none)
    - privte
- 傳回時型態必須一樣，或是原來的子類別，所以 A 不對

8. Given the following Java code: [10 points]

```
class SomeException:
public class SomeException {
}

class A:
public class A {
   public void doSomething() {}
}

class B:
public vlass B extends A {
   public void soSomething() throws SomeException {}
}
```

Which statement is true about the two classes?

A. Compilation of both classes will fail.

B. Compilation of both classes will succeed.

C. Compilation of class A will fail, Compilation of class B will succeed.

D. Compilation of class B will fail, Compilation of class A will succeed.

**ANS:___ (D)___**

- B 繼承 A，並且 override 其方法
- 從存取修飾字、傳回值型態來看都沒有問題
- 問題出在 A 中沒有宣告丟出 Exception，但是 B 中卻宣告丟出 Exception！
- override 的子類別方法，只能丟出比父類別少、小或相等的 Exception！

9. Given the following Java code: [10 points]

```
interface Foo {}
class Alpha implements Foo {}
class Beta extends Alpha {}
class Delta extends Beta {
   public static void main(String[] args) {
      Beta x = new Beta ();
7.          // insert code here
8.      }
9.  }
```

Which code, inserted at line 7 will cause a java.lang.ClassCastException?

A. Alpha a = x;

B. Foo f = (Delta)x;

C. Foo f = (Alpha)x;

D. Beta b = (Beta)(Alpha)x;

**ANS:__ B__**

- **題目是問哪個會造成類別轉換的 Exception.**
- **Alpha is-a Foo**
- **Beta is-a Alpha, Beta is-a Foo**
- **Delta is-a Beta, Delta is-a Alpha, Delta is-a Foo**
- **x is-a Beta**
    - **A : x is-a Beta, so x is-a Alpha, OK**
    - **B : x is-a Beta, but NOT is-a Delta, 錯誤**
    - **C : x is-a Beta, so x is-a Alpha, Alpha is-a Foo, OK**
    - **D : x is-a Beta, so x is-a Alpha, 當然可以再轉回 Beta, OK**

10. 請在底下的選項找出一個適合的配對上面的描述 [30 points]

【問題】

(1) 定義類別的共同標準規範

(2) 物件導向語言的特質中物件間互相溝通是是藉由什麼

(3) 一種將變數型態與程序包裝在一起的集合體

(4) 根據引數的個數或型態，呼叫到對應的函式

(5) 方法在不同的類別中調用卻可以實現的不同結果

(6) 物件的藍圖

(7) 資料和方法的實作程式碼都包裹隱藏起來

(8) 該函式一次只能被一個執行緒所存取

(9) 資料抽象化後所建立的自訂資料型態

(10) 在子類別中改寫繼承自父類別的方法

【選項】

| | | | |
|---|---|---|---|
| (A) Message | (B) State | (C) OOD | (D) Override |
| (E) Interface | (F) Overloading | (G) Inheritance | |
| (H) Identity | (I) Process | (J) this, super | (K) Composition |
| (L) Associations | (M) Class | (N) Object | (O) Module |
| (P) OOA | (Q) Behavior | (R) Encapsulation | (S) View |

|  |  |  |  |
|---|---|---|---|
| (T) Aggregation | (U) Dependency | (V) Polymorphism | (W) Instance |
| (X) Abstract Data Type |  | (Y) Model | (Z) Synchronized |

**ANS:**

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| **(1) E** | **(2) A** | **(3) N** | **(4) F** | **(5) V** | **(6) M** |
| **(7) R** | **(8) Z** | **(9) X** | **(10) D** |  |  |