基礎程式設計（Java）　答案卷

隨堂練習卷

建議作答 50 分鐘 · 總分 118

> 本章為自學延伸、非課程考試範圍，本卷供自修自測用。

§ 1 一、選擇題（每題 6 分，共 30 分）

1. **(B)** — 同一呼叫依物件實際型別跑不同版本是多型；多載是同名不同參數。 2. **(B)** — 動態繫結：執行期看物件的實際型別（Circle），跑 Circle 的 area()。變數型別只決定「能呼叫哪些方法」。 3. **(C)** — 抽象類別不能 `new`、只能被繼承；它仍可有建構子、欄位與一般方法。 4. **(B)** — 類別能 `implements` 多個介面，但只能 `extends` 一個父類別；介面不能 `new`。 5. **(A)** — 向下轉型前用 `instanceof` 確認實際型別，避免 `ClassCastException`。

§ 2 二、概念追蹤（共 16 分）

6. | 行 | 輸出 | |---|---| | 第 1 個 `a.speak()` | 汪 | | 第 2 個 `a.speak()` | 喵 | | `b.speak()` | … | `a` 宣告型別是 Animal，但動態繫結看物件：先指 Dog 跑「汪」，再指 Cat 跑「喵」；`b` 指的是真正的 Animal 物件，跑父類別版「…」。 7. 第 2 行 `Circle c = (Circle) shapes[1];` 會 `ClassCastException`：`shapes[1]` 實際是 `Square`，硬轉成 `Circle` 失敗（兩者只是同為 Shape 的兄弟，不能互轉）。避免法——先 `instanceof` 確認：

```
if (shapes[1] instanceof Circle) {
    Circle c = (Circle) shapes[1];
    System.out.println(c.getRadius());
}   // shapes[1] 是 Square，instanceof Circle 為 false，安全跳過、不轉型
```

§ 3 三、改錯（每題 6 分，共 18 分）

8. 抽象類別不能實例化，`new Shape()` 編譯錯（`Shape is abstract; cannot be instantiated`）。改成 `new` 具象子類別：`Shape s = new Circle(2);`（並確保 `Circle extends Shape` 且實作了 `area()`）。 9. 類別履行介面要用 `implements`，不是 `extends`。改：`class Circle implements Drawable { @Override public void draw(){} }`（`extends` 用於繼承類

別、`implements` 用於介面）。10. `final` 方法子類別不能覆寫，`Saving` 覆寫 `get()` 編譯錯。改：把父類別的 `final` 拿掉（`public double get(){return 0;}`）才允許覆寫；或子類別不要覆寫它。

§ 4 四、程式設計（共 54 分）

11.（18 分）

```
public class ShapeDemo {
    public static void main(String[] args) {
        Shape[] shapes = { new Circle(2), new Square(3) };
        for (Shape s : shapes)
            System.out.printf("%.2f%n", s.area());   // 12.57、9.00
    }
}
abstract class Shape {
    public abstract double area();
}
class Circle extends Shape {
    private double r;
    public Circle(double r) { this.r = r; }
    @Override public double area() { return Math.PI * r * r; }
}
class Square extends Shape {
    private double side;
    public Square(double side) { this.side = side; }
    @Override public double area() { return side * side; }
}
```

（評分：抽象類別＋抽象方法 6 分、兩子類別 `@Override` 實作 area() 8 分、`Shape[]` 多型走訪輸出 4 分。） 12.（18 分）

```
public class PayableDemo {
    public static void main(String[] args) {
        Payable[] items = { new Invoice(3, 19.99), new SalariedEmployee(50000) };
        for (Payable p : items)
            System.out.printf("%.2f%n", p.getPaymentAmount());   // 59.97、50000.00
    }
}
interface Payable { double getPaymentAmount(); }
class Invoice implements Payable {
    private int qty; private double price;
    public Invoice(int qty, double price) { this.qty = qty; this.price = price; }
    @Override public double getPaymentAmount() { return qty * price; }
}
class SalariedEmployee implements Payable {
    private double salary;
    public SalariedEmployee(double salary) { this.salary = salary; }
    @Override public double getPaymentAmount() { return salary; }
}
```

（評分：介面定義 4 分、兩類別 `implements` 並各實作 getPaymentAmount() 8 分、`Payable[]` 多型走訪輸出 6 分。） 13.（18 分）

```
public class DuckDemo {
    public static void main(String[] args) {
        Duck d = new Duck();
        d.fly();
        d.swim();
    }
}
interface Flyable { void fly(); }
interface Swimmable { void swim(); }
class Duck implements Flyable, Swimmable {
    @Override public void fly() { System.out.println("鴨子拍翅飛"); }
    @Override public void swim() { System.out.println("鴨子划水游"); }
}
```

（評分：兩介面定義 4 分、`implements Flyable, Swimmable` 多重實作 8 分、兩方法各實作並呼叫 6 分。關鍵：用 `implements` 逗號分隔、兩個方法都要實作。）