CS23304 – JAVA PROGRAMMING – B

FIRST ASSESSMENT

Date: 18.8.26 Time: 1 hr 30 mins

Max.marks: 50 marks

Part –A (5 X 2 =10 marks)

1. Compare and contrast abstract class and interface.
2. Find the output for the code snippet:

```
interface Shape {
   default void draw() { System.out.println("Drawing shape"); }
}
class Circle implements Shape { }
public class Test {
   public static void main(String[] args) {
      Shape s = new Circle();
      s.draw();
   }
}
```

3. "A class in package a.b can access a protected member of an unrelated class in package x.y simply by importing that class, without extending it." True or False — justify in one line.
4. Why is using '+' to concatenate Strings inside a loop inefficient?
5. What is the output?

```
String s1 = "hello";
String s2 = "hello";
String s3 = new String("hello");
System.out.println(s1 == s2);
System.out.println(s1 == s3);
System.out.println(s1.equals(s3));
```

Part –B (8 x 5 = 40 marks)

Answer any EIGHT QUESTIONS

6. Design an abstract class Vehicle with a concrete method displayType() and an abstract method startEngine(). Define two interfaces, LandCapable (method drive()) and WaterCapable (method sail()). Write class AmphibiousVehicle that extends Vehicle and implements both interfaces, providing all required method bodies, and write test code that calls all four behaviours through a single Vehicle reference
7. Identify the violated OOP principle and fix the design.

```
class Rectangle {
   protected int width, height;
   void setWidth(int w)  { width = w; }
   void setHeight(int h) { height = h; }
   int getArea() { return width * height; }
```

```
}
class Square extends Rectangle {
   @Override void setWidth(int w)  { width = w;  height = w; }
   @Override void setHeight(int h) { width = h;  height = h; }
}
static void resizeAndCheck(Rectangle r) {
   r.setWidth(5);
   r.setHeight(10);
   System.out.println(r.getArea() == 50 ? "OK" : "FAILED: " + r.getArea());
}
resizeAndCheck(new Square());   // caller expected 50
```

8. (a) State the output of obj.level and obj.getLevel(). (b) Modify ONLY the Base class so obj.level prints 2, without changing what obj.getLevel() prints.

```
class Base {
   int level = 1;
   String getLevel() { return "Base level " + level; }
}
class Derived extends Base {
   int level = 2;
   @Override String getLevel() { return "Derived level " + level; }
}
Base obj = new Derived();
System.out.println(obj.level);
System.out.println(obj.getLevel());
```

9. Consider the code:

```
static void modifyArray(int[] arr) {
    arr[0] = 99;
    arr = new int[]{1, 2, 3};
}
int[] data = {5, 6, 7};
modifyArray(data);
System.out.println(data[0]);
```

(a) What does data[0] print, and which of the two statements inside modifyArray actually affects thecaller's array?

(b) Why does reassigning arr inside the method not affect data at all — what exactly is being passed to the method?

(c) If the two statements inside modifyArray were swapped (reassign first, then arr[0] = 99), would data[0] change? Why or why not?

(d) A developer claims "Java passes arrays by reference, so any change inside the method affects the caller." Where exactly does this claim break down?

10. Determine the final printed date and explain why the two method calls appear to have no effect.

```
import java.time.LocalDate;
public class Test {
   public static void main(String[] args) {
```

```
        LocalDate date = LocalDate.of(2026, 1, 15);
        date.plusDays(20);
        date.plusMonths(1);
        System.out.println(date);
    }
}
```

11. This compiles fine but crashes at runtime. Identify the exact exception and line, then rewrite the loop so it safely sums only the Integer elements, skipping any non-Integer values using instanceof. *[4 Marks]*

```
public class Test {
    public static void main(String[] args) {
        Object[] arr = new Object[3];
        arr[0] = "Hello";
        arr[1] = 10;
        arr[2] = 3.14;
        int sum = 0;
        for (Object o : arr) {
            sum += (Integer) o;
        }
        System.out.println(sum);
    }
}
```

12. Consider the code snippet:

```
// package com.store.inventory
class StockItem { protected int quantity; }
// package com.store.orders
class OrderProcessor {
    void checkStock(StockItem item) { System.out.println(item.quantity); }
}
```

(a) Name both access violations here, and explain why protected doesn't work the way it was assumed("protected means accessible if I have a reference").

(b) If StockItem were made public but quantity stayed protected, would checkStock() compile? Why or why not?

(c) What's the minimal access-modifier-only change that makes this compile and behave as intended?

(d) A developer "fixes" this by making quantity public instead of understanding why protected failed. What long-term design cost does that shortcut introduce compared to exposing a public getter?

13. Define a custom checked exception InvalidWithdrawalException. Write a class BankAccount with a private double balance and a method void withdraw(double amount) throws InvalidWithdrawalException that throws the exception (with a meaningful message) if amount is negative, zero, or greater than the balance; otherwise it deducts the amount. Write the test code showing how withdraw() would be safely invoked.

14. Write a Java method public static int secondLargest(int[] arr) that returns the second largest distinct element in an array, without sorting the array. Handle the case where the array has fewer than 2 distinct elements by returning Integer.MIN_VALUE.

15. Write a class Student (fields: name, marks) that implements Comparable<Student>, ordering students by marks in descending order. write Test code that sorts an ArrayList<Student> using Collections.sort() and prints the names in rank order.