DELHI SKILL AND ENTREPRENEURSHIP UNIVERSITY

B.Tech (Computer Science Engineering)

End Semester Examination – Set 3

Subject Name:	Java Programming
Subject Code:	BT-CS-ES501
Time:	3 Hours
Max. Marks:	100

Instructions

1. The question paper contains two sections: **Section A** and **Section B**.
2. **Section A** is compulsory (Short Answer Questions).
3. **Section B** contains descriptive questions.
4. Assume necessary data if not provided.

SECTION A – Short Answer Questions (20 Marks)

Attempt all questions. Each question carries 2 marks.

Q.1 Why is Java described as "Compiled and Interpreted"? [2M]
Answer: Java source code ('.java') is first *compiled* into Bytecode ('.class'), which is platform-independent. This Bytecode is then *interpreted* (or JIT compiled) by the Java Virtual Machine (JVM) into native machine code at runtime.

Q.2 What is the 'instanceof' operator used for? [2M]
Answer: The 'instanceof' operator is a boolean operator used to test if an object is an instance of a specific class, subclass, or an interface. Example: 'if (obj instanceof String)' returns true if 'obj' is a String.

Q.3 Explain the difference between 'break' and 'continue' statements. [2M]
Answer:

- **break:** Terminates the loop or switch statement immediately and transfers control to the statement following the loop.
- **continue:** Skips the current iteration of the loop and jumps to the next iteration (checks condition again).

Q.4 Define a "Marker Interface". Give an example. [2M]
Answer: A Marker Interface is an interface that contains no methods or fields. It provides run-time type information to the JVM so that it can treat the object specially. *Examples:* 'java.io.Serializable', 'java.lang.Cloneable'.

Q.5 What is the default value of the elements in an 'int' array and a 'String' array? [2M]
Answer:

- 'int' array: Default value is '0'.
- 'String' (Object) array: Default value is 'null'.

Q.6 What is the purpose of the ‘volatile‘ keyword? **[2M]**
Answer: The ‘volatile‘ keyword guarantees that the value of a variable is always read from the main memory, not from a thread’s local cache. It prevents memory consistency errors in multi-threaded applications.

Q.7 Differentiate between ‘paint()‘ and ‘repaint()‘ methods in Applets. **[2M]**
Answer:

- ‘paint(Graphics g)‘: Automatically called by the system to draw the component. It contains the actual drawing logic.
- ‘repaint()‘: Called programmatically to request an update. It schedules a call to ‘update()‘ which eventually calls ‘paint()‘.

Q.8 Identify the components of a JDBC URL. **[2M]**
Answer: A JDBC URL structure is: ‘jdbc:¡subprotocol¿:¡subname¿‘.

- ‘jdbc‘: The protocol.
- ‘subprotocol‘: The driver/database type (e.g., ‘mysql‘, ‘oracle‘).
- ‘subname‘: Database location/name (e.g., ‘//localhost:3306/mydb‘).

Q.9 What is a ”Jagged Array”? **[2M]**
Answer: A Jagged Array is a multidimensional array where member arrays are of different sizes. In Java, 2D arrays are technically arrays of arrays, so each row can have a different column length.

Q.10 What is the ‘this‘ keyword? **[2M]**
Answer: The ‘this‘ keyword is a reference variable that refers to the current object. It is used to resolve name conflicts between instance variables and parameters, and to invoke other constructors in the same class.

SECTION B – Descriptive Questions (80 Marks)

Part I – Medium Answer Questions (5 Marks Each)

Q.11 Explain the difference between ‘Call by Value‘ and ‘Call by Reference‘. How does Java handle parameter passing? **[5M]**
Answer:

- **Call by Value:** A copy of the value is passed to the method. Changes to the parameter do not affect the original argument.
- **Call by Reference:** A reference (address) is passed. Changes affect the original data.

Java’s Approach: Java is *strictly* **Pass by Value**.

- For primitives (int, float), the actual value is copied.
- For Objects, the *value of the reference* (memory address) is copied. Thus, while you can modify the object’s internals, you cannot change the reference itself to point to a new object.

Q.12 What are ‘Inner Classes‘? Briefly describe Member Inner Class and Anonymous Inner Class. **[5M]**
Answer: An Inner Class is a class defined within another class. It has access to the members of the enclosing class.

(a) **Member Inner Class:** Defined at the class level (non-static). It requires an instance of the outer class to be instantiated.

(b) **Anonymous Inner Class:** A class without a name, declared and instantiated in a single expression. It is often used for event listeners (e.g., ‘new ActionListener() ... ‘).

Q.13 Explain the role of ‘join()‘ and ‘isAlive()‘ methods in Multithreading. **[5M]**
Answer:

- ‘isAlive()‘: A boolean method that checks if a thread has started and has not yet died. It returns ‘true‘ if the thread is running or runnable.
- ‘join()‘: A synchronization method that allows one thread to wait for the completion of another. If Thread A calls ‘B.join()‘, Thread A halts execution until Thread B finishes.

Q.14 Write a snippet to demonstrate the ‘StringBuffer‘ methods: ‘append()‘, ‘insert()‘, ‘replace()‘, and ‘reverse()‘. **[5M]**
Answer:

```java
public class BufferDemo {
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer("Java");

        sb.append(" Programming");      // "Java Programming"
        sb.insert(5, "Core ");          // "Java Core Programming"
        sb.replace(0, 4, "Advanced");   // "Advanced Core Programming"
        sb.reverse();                   // Reverse the string

        System.out.println(sb);
    }
}

```

Part II – Long Answer Questions (10 Marks Each)

Q.15 Demonstrate ‘Multilevel Inheritance‘ with a Java program. Create a class ‘Student‘ (rollNo), extending to ‘Test‘ (marks), extending to ‘Result‘ (total). Explain the order of constructor execution. **[10M]**
Answer: In Multilevel Inheritance, a class is derived from a class which is also derived from another class.

Constructor Order: When an object of the child class (‘Result‘) is created, constructors are executed from top to bottom (Parent $\rightarrow$ Child): ‘Student‘ $\rightarrow$ ‘Test‘ $\rightarrow$ ‘Result‘.

```java
class Student {
    int rollNo;
    Student(int r) {
        this.rollNo = r;
        System.out.println("Student Constructor");
    }
}
class Test extends Student {
    int m1, m2;
    Test(int r, int m1, int m2) {
        super(r);
        this.m1 = m1; this.m2 = m2;
        System.out.println("Test Constructor");
    }
}
class Result extends Test {
    Result(int r, int m1, int m2) {
        super(r, m1, m2);
        System.out.println("Result Constructor");
    }
    void show() {
        System.out.println("Roll: " + rollNo + " Total: " + (m1 + m2));
    }
}
public class Main {
    public static void main(String[] args) {
        Result r = new Result(101, 85, 90);
        r.show();
    }
}

```

Q.16 What is Object Serialization? Explain the ‘Serializable‘ interface. Write a program to save an object to a file and retrieve it (Deserialization). **[10M]**

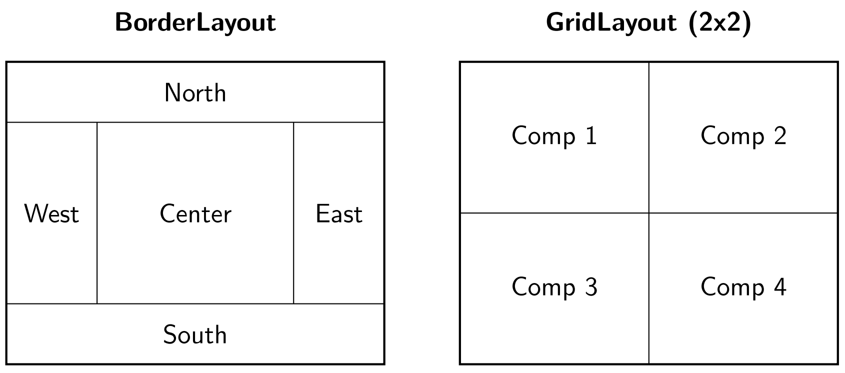
Answer: **Serialization** is the process of converting an object's state into a byte stream to save it to a file or send it over a network. **Deserialization** is the reverse process. The class must implement the ‘java.io.Serializable‘ marker interface.

```java
import java.io.*;

class User implements Serializable {
    String name;
    int id;
    User(String n, int i) { name = n; id = i; }
}

public class SerialDemo {
    public static void main(String[] args) {
        User u1 = new User("Alice", 123);

        // Serialization
        try (ObjectOutputStream out = new ObjectOutputStream(
                new FileOutputStream("user.ser"))) {
            out.writeObject(u1);
            System.out.println("Object Serialized");
        } catch (IOException e) { e.printStackTrace(); }

        // Deserialization
        try (ObjectInputStream in = new ObjectInputStream(
                new FileInputStream("user.ser"))) {
            User u2 = (User) in.readObject();
            System.out.println("Deserialized: " + u2.name);
        } catch (Exception e) { e.printStackTrace(); }
    }
}

```

Q.17 Compare ‘BorderLayout‘ and ‘GridLayout‘ managers in AWT. Draw diagrams to illustrate how they arrange components. **[10M]**

Answer:

- **BorderLayout:** Divides the container into 5 regions: North, South, East, West, and Center. It is the default for Frames. Components stretch to fill the region.
- **GridLayout:** Arranges components in a rectangular grid of equal-sized cells. Components are added left-to-right, top-to-bottom.



Q.18 Discuss ‘PreparedStatement‘ in JDBC. How does it differ from ‘Statement‘? Explain its benefits regarding performance and security (SQL Injection). **[10M]**

Answer: A ‘PreparedStatement‘ is a sub-interface of ‘Statement‘ used to execute parameterized SQL queries.

Differences & Benefits:

(a) **Pre-compilation (Performance):** The database compiles the SQL query structure only once. Subsequent executions with different parameters are faster because the query plan is reused. ‘Statement‘ compiles the SQL every time.

(b) **Security (SQL Injection):** ‘PreparedStatement‘ treats parameters as values, not executable code. This prevents SQL Injection attacks, whereas ‘Statement‘ uses string concatenation which is vulnerable.

Example:

```
// Vulnerable Statement
String query = "SELECT * FROM users WHERE name = '" + userName + "'";

// Secure PreparedStatement
String query = "SELECT * FROM users WHERE name = ?";
PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setString(1, "Alice"); // Parameter index 1
ResultSet rs = pstmt.executeQuery();

```

Q.19 Explain the mechanism of ‘Multiple Inheritance‘ in Java using Interfaces. Solve the ”Diamond Problem” conceptual ambiguity with an example. **[10M]**

Answer: Java does not support multiple inheritance with classes to avoid ambiguity (Diamond Problem). However, a class can implement multiple interfaces.

Since interfaces (prior to Java 8 default methods) only declare methods without body, there is no ambiguity about *which* implementation to use—the implementation is provided solely by the child class.

```
interface Printer {
    void print();
}
interface Scanner {
    void scan();
}

// Multiple Inheritance behavior
class Machine implements Printer, Scanner {
    public void print() {
```

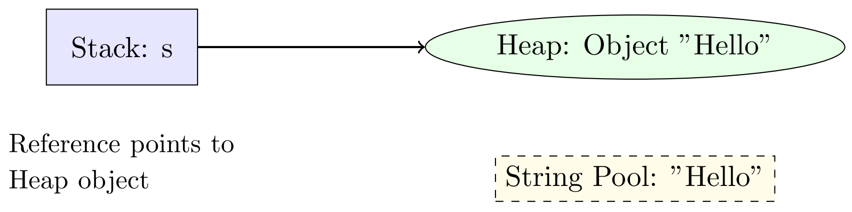
```java
        System.out.println("Printing...");
    }
    public void scan() {
        System.out.println("Scanning...");
    }
}

public class Main {
    public static void main(String[] args) {
        Machine m = new Machine();
        m.print();
        m.scan();
    }
}

```

Note: Even if two interfaces have a method with the same signature, the implementing class provides a single implementation that satisfies both interfaces.

Q.20 Draw the memory map for the statement 'String s = new String("Hello");'. Explain the String Constant Pool and why Strings are immutable in Java. [10M]

Answer: Immutability: Strings are immutable for: 1. **Security:** Strings often store passwords, URLs, etc. 2. **Caching:** HashCodes can be cached. 3. **Thread Safety:** Immutable objects are automatically thread-safe.

Memory Map: When 'String s = new String("Hello");' is executed: 1. Literal "Hello" is placed in the **String Constant Pool** (if not present). 2. A new Object is created in the **Heap** memory (outside the pool). 3. Variable 's' (Stack) points to the Heap object.


Stack: s
Heap: Object "Hello"
Reference points to Heap object
String Pool: "Hello"