DELHI SKILL AND ENTREPRENEURSHIP UNIVERSITY

B.Tech (Computer Science Engineering)

End Semester Examination

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 and contains short answer questions.
3. **Section B** contains descriptive questions including programming problems and theoretical concepts.
4. Assume necessary data if not provided.

SECTION A – Short Answer Questions (20 Marks)

Attempt all questions. Each question carries 2 marks.

Q.1 What is the significance of the 'Bytecode' in Java? How does it contribute to platform independence? **[2M]**
Answer: Bytecode is an intermediate, highly optimized set of instructions generated by the Java compiler ('javac') that is not specific to any processor. It contributes to platform independence because it can be executed on any device that has a Java Virtual Machine (JVM), adhering to the "Write Once, Run Anywhere" (WORA) philosophy.

Q.2 Differentiate between 'StringBuffer' and 'String' classes. **[2M]**
Answer:

- **String:** Immutable; once created, its value cannot be changed. Modifications create new objects.
- **StringBuffer:** Mutable; allows modification of the string content without creating new objects, making it more memory efficient for frequent updates.

Q.3 Explain the use of the 'final' keyword when applied to a method. **[2M]**
Answer: When a method is declared as 'final', it cannot be overridden by subclasses. This is used to prevent the modification of the method's implementation in the inheritance hierarchy, ensuring consistent behavior and security.

Q.4 What is a 'Wrapper Class'? Give two examples. **[2M]**
Answer: A Wrapper Class provides a mechanism to convert primitive data types into objects (Reference types). This is essential for collections like 'ArrayList' that only store objects. *Examples:* 'Integer' (for 'int'), 'Character' (for 'char').

Q.5 Define 'Checked' and 'Unchecked' exceptions. **[2M]**
Answer:

- **Checked Exceptions:** Checked at compile-time (e.g., 'IOException', 'SQLException'). The code must handle them using try-catch or throws.
- **Unchecked Exceptions:** Occur at runtime (e.g., 'NullPointerException', 'ArithmeticException'). The compiler does not enforce handling them.

Q.6 What is the purpose of the 'super' keyword? [2M]
Answer: The 'super' keyword is a reference variable used to refer to the immediate parent class object. It is used to: 1. Call the parent class's constructor. 2. Access parent class methods or variables that have been hidden or overridden by the child class.

Q.7 Briefly explain the role of a 'Layout Manager' in AWT. [2M]
Answer: A Layout Manager automatically arranges GUI components (buttons, labels, etc.) within a container. It manages the size and position of components based on the container's size, ensuring the GUI looks consistent across different platforms. Examples include 'FlowLayout' and 'BorderLayout'.

Q.8 What is a 'Daemon Thread'? [2M]
Answer: A Daemon thread is a low-priority service thread that runs in the background to perform tasks like garbage collection. The JVM terminates automatically when all user threads (non-daemon threads) finish execution, regardless of whether daemon threads are running.

Q.9 List the four JDBC driver types. [2M]
Answer: 1. Type-1: JDBC-ODBC Bridge Driver. 2. Type-2: Native-API Driver (Partially Java). 3. Type-3: Network Protocol Driver (Middleware). 4. Type-4: Thin Driver (Pure Java, Direct to Database).

Q.10 Why does Java not support multiple inheritance with classes? [2M]
Answer: Java avoids multiple inheritance with classes to prevent the "Diamond Problem," where ambiguity arises if two parent classes have methods with the same signature. However, Java achieves multiple inheritance behaviors using **Interfaces**.

SECTION B – Descriptive Questions (80 Marks)

Part I – Medium Answer Questions (5 Marks Each)

Q.11 Explain the concept of 'Dynamic Method Dispatch' with a suitable code example. [5M]
Answer: Dynamic Method Dispatch is a mechanism by which a call to an overridden method is resolved at run-time rather than compile-time. This is how Java implements run-time polymorphism. A reference variable of a superclass can refer to an object of a subclass.

```
class Animal {
    void sound() { System.out.println("Animal makes sound"); }
}
class Dog extends Animal {
    void sound() { System.out.println("Dog Barks"); } // Overriding
}
public class Main {
    public static void main(String args[]) {
        Animal obj = new Dog(); // Upcasting
        obj.sound(); // Output: "Dog Barks" (Resolved at runtime)
    }
}

```

Q.12 Describe the 'Delegation Event Model' used in Java AWT. **[5M]**

Answer: The Delegation Event Model is the standard framework for handling events in Java. It consists of two main parts:

(a) **Source:** The GUI component (e.g., Button) that generates the event when an action occurs.

(b) **Listener:** An object that "listens" for the event and processes it. The listener must implement the appropriate interface (e.g., 'ActionListener').

Process: The Source registers a Listener using methods like 'addActionListener()'. When the event occurs, the Source sends an Event Object to the Listener's handling method. This separates the UI logic from the business logic.

Q.13 Differentiate between 'Abstract Class' and 'Interface'. **[5M]**

Answer:

Abstract Class	**Interface**
Can have both abstract and non-abstract (concrete) methods.	Prior to Java 8, could only have abstract methods. (Java 8+ allows default/static methods).
Supports single inheritance ('extends').	Supports multiple inheritance ('implements').
Variables can be static, non-static, final, or non-final.	Variables are implicitly 'public static final'.
Can have a constructor.	Cannot have a constructor.
Used when classes share a common state/behavior.	Used to define a contract or capability.

Q.14 Write a Java program to read a file named "input.txt" and display its contents on the console using 'FileInputStream'. **[5M]**

Answer:

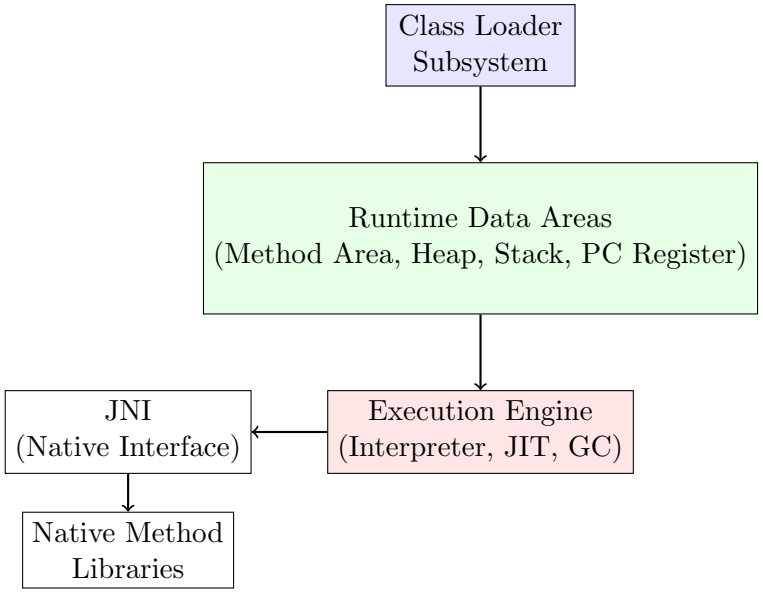
```java
import java.io.*;

public class FileReadExample {
    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream("input.txt");
            int i;
            // Read byte by byte until end of file (-1)
            while ((i = fis.read()) != -1) {
                System.out.print((char) i);
            }
            fis.close();
        } catch (FileNotFoundException e) {
            System.out.println("File not found.");
        } catch (IOException e) {
            System.out.println("Error reading file.");
        }
    }
}

```

Part II – Long Answer Questions (10 Marks Each)

Q.15 Draw the block diagram of the Java Virtual Machine (JVM) architecture and explain its three main subsystems in detail. **[10M]**

Answer:

The JVM acts as an abstract computing machine. Its architecture consists of three main subsystems:



1. Class Loader Subsystem: Responsible for loading class files ('.class') into memory. It performs three main functions:

- **Loading:** Reads the '.class' file and generates binary data.
- **Linking:** Verifies bytecode, prepares static variables, and resolves symbolic references.
- **Initialization:** Executes static blocks and assigns static values.

2. Runtime Data Areas (Memory):

- **Method Area:** Stores class structures (metadata, constants).
- **Heap:** Stores Objects and Instance variables (Garbage collected).
- **Stack:** Stores local variables and method calls.
- **PC Register:** Holds the address of the current instruction.

3. Execution Engine: Executes the bytecode contained in the methods.

- **Interpreter:** Executes instructions line by line (slower).
- **JIT Compiler:** Compiles frequently used bytecode into native code for performance.
- **Garbage Collector:** Reclaims memory from unused objects.

Q.16 Define 'Multithreading'. Explain the Life Cycle of a Thread with a neat diagram. Write a code snippet to create a thread using the 'Runnable' interface. [10M]

Answer: Multithreading is the concurrent execution of two or more parts of a program to maximize CPU utilization.

Thread Life Cycle: A thread goes through five states:

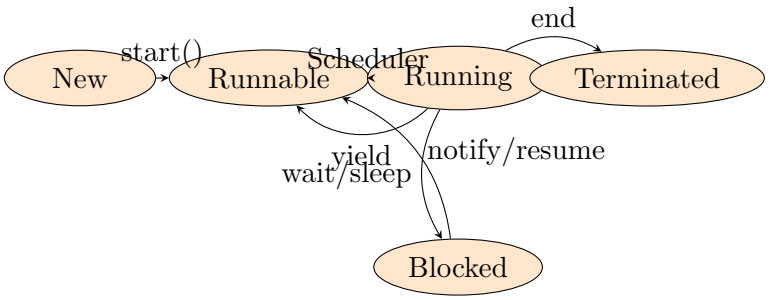
(a) **New:** Thread created but not started.

(b) **Runnable:** Ready to run, waiting for CPU time.

(c) **Running:** Currently executing code.

(d) **Blocked/Waiting:** Waiting for I/O or another thread.

(e) **Terminated (Dead):** Execution completed.



Code Snippet (Runnable Interface):

```
class MyTask implements Runnable {
    public void run() {
        System.out.println("Thread is running...");
    }
}
class Main {
    public static void main(String args[]) {
        MyTask task = new MyTask();
        Thread t1 = new Thread(task);
        t1.start(); // Moves thread to Runnable state
    }
}

```

Q.17 What is Exception Handling? Explain the usage of 'try', 'catch', 'finally', 'throw', and 'throws' keywords with a comprehensive example. [10M]

Answer: Exception Handling is a mechanism to handle runtime errors so that the normal flow of the application can be maintained.

Keywords:

- **try:** Block of code where an exception might occur.
- **catch:** Block that handles the exception thrown by the try block.
- **finally:** Block that always executes (used for cleanup like closing files), regardless of exception occurrence.
- **throw:** Used to explicitly throw an exception.
- **throws:** Declares that a method might throw specific exceptions.

Example:

```
class ExceptionDemo {
    // 'throws' declares that this method may fail
    static void checkAge(int age) throws ArithmeticException {
        if (age < 18) {
            // 'throw' explicitly creates an exception
            throw new ArithmeticException("Access Denied: Under Age");
        } else {
            System.out.println("Access Granted");
        }
    }
```

```
11
12      public static void main(String[] args) {
13          try {
14              // Code that might cause issues
15              checkAge(15);
16          } catch (ArithmeticException e) {
17              // Handling the exception
18              System.out.println("Caught Exception: " + e.getMessage());
19          } finally {
20              // Cleanup code
21              System.out.println("Execution Completed.");
22          }
23      }
24  }
25
```

Q.18 Design a class 'Employee' with attributes 'id' and 'salary'. Create a subclass 'Manager' with an additional attribute 'bonus'. Use constructors to initialize values and a method 'display()' to print details. Demonstrate usage of 'super' to call the parent constructor. [10M]

Answer:

```
1  class Employee {
2      int id;
3      double salary;
4
5      // Parent Constructor
6      Employee(int id, double salary) {
7          this.id = id;
8          this.salary = salary;
9      }
10
11     void display() {
12         System.out.println("ID: " + id);
13         System.out.println("Base Salary: " + salary);
14     }
15 }
16
17 class Manager extends Employee {
18     double bonus;
19
20     // Child Constructor
21     Manager(int id, double salary, double bonus) {
22         // Calling parent constructor using 'super'
23         super(id, salary);
24         this.bonus = bonus;
25     }
26
27     // Overriding display method
28     @Override
29     void display() {
30         super.display(); // Reusing parent display logic
31         System.out.println("Bonus: " + bonus);
32         System.out.println("Total Salary: " + (salary + bonus));
33     }
34 }
35
36 public class Main {
```

```
    public static void main(String[] args) {
        Manager m = new Manager(101, 50000, 10000);
        m.display();
    }
}

```

Q.19 Explain Inter-thread Communication. Write a program to solve the 'Producer-Consumer' problem using 'wait()' and 'notify()' methods. [10M]

Answer: Inter-thread communication allows synchronized threads to communicate with each other using 'wait()', 'notify()', and 'notifyAll()' methods defined in the 'Object' class.

Producer-Consumer Logic: The Producer waits if the buffer is full; otherwise, it produces data and notifies the Consumer. The Consumer waits if the buffer is empty; otherwise, it consumes data and notifies the Producer.

```
class SharedResource {
    int data;
    boolean hasData = false;

    synchronized void produce(int value) {
        while (hasData) { // Wait if data exists
            try { wait(); } catch (InterruptedException e) {}
        }
        this.data = value;
        System.out.println("Produced: " + value);
        hasData = true;
        notify(); // Notify consumer
    }

    synchronized void consume() {
        while (!hasData) { // Wait if no data
            try { wait(); } catch (InterruptedException e) {}
        }
        System.out.println("Consumed: " + data);
        hasData = false;
        notify(); // Notify producer
    }
}
// (Main class would create Producer/Consumer threads calling these
   methods)

```

Q.20 Detail the steps to connect a Java Application to a Database using JDBC. Write the code snippet for establishing a connection and executing a 'SELECT' query. [10M]

Answer: Steps for JDBC Connectivity:

(a) **Import Packages:** Import 'java.sql.*'.

(b) **Load Driver:** Use 'Class.forName("driver$_{c}lass_{n}ame$")'.

(b) **Establish Connection:** Use 'DriverManager.getConnection(url, user, pass)'.

(c) **Create Statement:** Create a 'Statement' or 'PreparedStatement' object.

(d) **Execute Query:** Use 'executeQuery()' for SELECT or 'executeUpdate()' for INSERT/UPDATE.

(e) **Process Results:** Iterate through the 'ResultSet'.

(f) **Close Connection:** Close 'ResultSet', 'Statement', and 'Connection'.

Code Snippet:

```
import java.sql.*;

public class JDBCDemo {
    public static void main(String[] args) {
        try {
            // 1. Load Driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // 2. Connect
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/mydb", "root", "password");

            // 3. Create Statement
            Statement stmt = con.createStatement();

            // 4. Execute Query
            ResultSet rs = stmt.executeQuery("SELECT * FROM students");

            // 5. Process Results
            while(rs.next()) {
                System.out.println(rs.getInt(1) + " " + rs.getString(2));
            }

            con.close();
        } catch (Exception e) { System.out.println(e); }
    }
}

```