DELHI SKILL AND ENTREPRENEURSHIP UNIVERSITY

B.Tech (Computer Science Engineering)

End Semester Examination – Set 2

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 Define Garbage Collection in Java. Which method can be used to request it? [2M]
Answer: Garbage Collection is the process of automatically identifying and reclaiming memory occupied by objects that are no longer reachable or referenced by the program. It prevents memory leaks. The 'System.gc()' or 'Runtime.getRuntime().gc()' methods can be used to request garbage collection.

Q.2 What is the difference between 'Byte Stream' and 'Character Stream'? [2M]
Answer:

- **Byte Stream:** Handles raw binary data (8-bit bytes). Used for images, audio, etc. (e.g., 'FileInputStream').
- **Character Stream:** Handles text data (16-bit Unicode characters). Used for reading/writing text files. (e.g., 'FileReader').

Q.3 Explain the purpose of the 'static' block in a Java class. [2M]
Answer: A 'static' block is used to initialize static variables. It is executed only once when the class is loaded into memory by the ClassLoader, even before the execution of the 'main' method or object creation.

Q.4 Differentiate between Method Overloading and Method Overriding. [2M]
Answer:

- **Overloading:** Same method name, different parameter list (compile-time polymorphism). Occurs within the same class.
- **Overriding:** Same method name, same parameter list (runtime polymorphism). Occurs between superclass and subclass.

Q.5 What is the significance of the 'package' statement? [2M]
Answer: The 'package' statement groups related classes, interfaces, and sub-packages together. It helps in preventing naming conflicts (namespace management) and provides access protection/visibility control. It must be the first statement in a Java source file.

Q.6 Can a 'finally' block exist without a 'catch' block? Justify. [2M]
Answer: Yes, a 'try' block can be followed immediately by a 'finally' block without a 'catch' block. However, in this case, any exception thrown in the try block will propagate up the call stack after the finally block executes.

Q.7 What are 'Adapter Classes' in AWT? [2M]
Answer: Adapter classes are abstract classes that provide empty implementations of listener interfaces. They allow programmers to implement only the specific methods they need (e.g., overriding only 'windowClosing' from 'WindowAdapter') instead of implementing all methods of the interface.

Q.8 What is the purpose of the 'synchronized' keyword? [2M]
Answer: The 'synchronized' keyword is used to control access to shared resources by multiple threads. It prevents thread interference and consistency errors by ensuring that only one thread can execute a synchronized method or block at a time for a given object.

Q.9 Briefly explain the Life Cycle of an Applet. [2M]
Answer: An Applet goes through four main states managed by the browser: 1. 'init()': Initialization. 2. 'start()': Started or resumed. 3. 'stop()': Suspended or minimized. 4. 'destroy()': Terminated and removed from memory.

Q.10 What are Command Line Arguments? How are they accessed? [2M]
Answer: Command Line Arguments are parameters passed to the application at the time of execution. They are stored as strings in the 'String args[]' array passed to the 'main' method. For example, 'args[0]' accesses the first argument.

SECTION B – Descriptive Questions (80 Marks)

Part I – Medium Answer Questions (5 Marks Each)

Q.11 Discuss the four types of Access Specifiers in Java and their visibility scope. [5M]
Answer: Java provides four access levels to determine the visibility of classes, methods, and variables:

(a) **private:** Visible only within the same class. Most restrictive.

(b) **default (no keyword):** Visible within the same class and other classes in the same package.

(c) **protected:** Visible within the same package and in subclasses (even if they are in different packages).

(d) **public:** Visible to all classes in all packages. Least restrictive.

Q.12 Explain 'Constructor Overloading' with an example. How is the 'this' keyword used for constructor chaining? [5M]
Answer: Constructor Overloading allows a class to have multiple constructors with different parameter lists. The 'this()' call allows one constructor to call another constructor within the same class to reduce code duplication.

```
class Box {
    double width, height, depth;
    // Constructor 1
```

```
    Box(double w, double h, double d) {
        this.width = w; this.height = h; this.depth = d;
    }
    // Constructor 2 (Overloaded)
    Box() {
        // Calls Constructor 1 using 'this'
        this(10, 10, 10);
    }
}
```

Q.13 Write a Java program to copy the content of one file to another using 'FileReader' and 'FileWriter'. [5M]
Answer:

```
import java.io.*;
public class FileCopy {
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("source.txt");
            FileWriter fw = new FileWriter("dest.txt");
            int i;
            while ((i = fr.read()) != -1) {
                fw.write(i);
            }
            fr.close();
            fw.close();
            System.out.println("File copied successfully.");
        } catch (IOException e) {
            System.out.println("I/O Error: " + e);
        }
    }
}
```

Q.14 Explain the following String class methods with examples: 'substring()', 'concat()', 'trim()', 'equals()', and 'length()'. [5M]
Answer:

- 'substring(int begin, int end)': Returns part of the string. e.g., '"Hello".substring(0,2)' → '"He"'.
- 'concat(String str)': Appends string. e.g., '"Hi".concat("There")' → '"HiThere"'.
- 'trim()': Removes leading/trailing whitespace. e.g., '" Java ".trim()' → '"Java"'.
- 'equals(Object anObject)': Compares content. e.g., '"A".equals("A")' → 'true'.
- 'length()': Returns number of characters. e.g., '"Hi".length()' → '2'.

Part II – Long Answer Questions (10 Marks Each)

Q.15 Explain the key features of the Java Language that make it a robust and popular programming language. [10M]
Answer: Java is defined by several key buzzwords:

- **Simple:** Removed complex features of C++ like pointers and operator overloading.
- **Object-Oriented:** Everything is an Object. It supports Inheritance, Polymorphism, Encapsulation, and Abstraction.

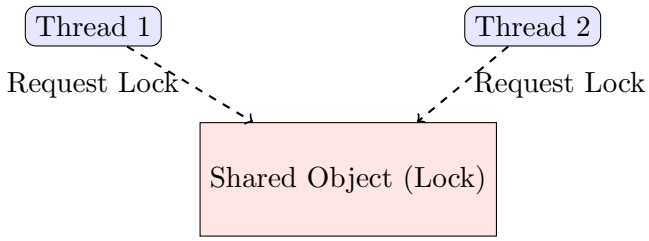
- **Platform Independent:** "Write Once, Run Anywhere". Compiled code (Bytecode) runs on any JVM.
- **Secure:** No explicit pointers, runs inside a virtual machine sandbox, strict compile-time checking.
- **Robust:** Strong memory management (Garbage Collection) and Exception Handling prevent crashes.
- **Multithreaded:** Built-in support for concurrent execution.
- **Dynamic:** Loads classes on demand; supports native methods (C/C++).
- **Distributed:** Designed for the distributed environment of the internet (RMI, EJB).

Q.16 What is Thread Synchronization? Explain the "Race Condition" problem. Demonstrate how to solve it using a synchronized method for a Bank Account transaction. [10M]

Answer: Synchronization coordinates the execution of threads to ensure they don't access shared resources simultaneously in a way that leads to data inconsistency.

Race Condition: Occurs when two threads access a shared variable at the same time, and the final result depends on the timing of their execution.

Visual Representation:



Only one enters at a time

Code Example:

```
class BankAccount {
    private int balance = 1000;

    // Synchronized method prevents Race Condition
    public synchronized void withdraw(int amount) {
        if (balance >= amount) {
            System.out.println(Thread.currentThread().getName() + " is withdrawing...");
            try { Thread.sleep(100); } catch (Exception e) {}
            balance -= amount;
            System.out.println("Remaining: " + balance);
        } else {
            System.out.println("Insufficient funds for " + Thread.currentThread().getName());
        }
    }
}
class Client extends Thread {
    BankAccount account;
    Client(BankAccount acc) { this.account = acc; }
    public void run() { account.withdraw(700); }
}
```

```java
public class SyncDemo {
    public static void main(String args[]) {
        BankAccount acc = new BankAccount();
        Client c1 = new Client(acc);
        Client c2 = new Client(acc);
        c1.start(); c2.start();
    }
}
```

Q.17 How do you create a User-Defined Exception in Java? Create a custom exception class 'InvalidAgeException'. Write a program to validate a voter's age and throw this exception if the age is below 18. **[10M]**

Answer: To create a custom exception, we extend the 'Exception' class (for checked exceptions) or 'RuntimeException' (for unchecked).

```java
// 1. Define the custom exception
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}

// 2. Class using the exception
public class VoterValidator {

    static void validate(int age) throws InvalidAgeException {
        if (age < 18) {
            // Throwing the custom exception
            throw new InvalidAgeException("Age is " + age + ". Minimum age is 18.");
        } else {
            System.out.println("Welcome to vote!");
        }
    }

    public static void main(String[] args) {
        try {
            validate(16); // Testing with invalid age
        } catch (InvalidAgeException e) {
            System.out.println("Exception Caught: " + e.getMessage());
        }
        System.out.println("Process finished.");
    }
}
```

Q.18 Demonstrate Hierarchical Inheritance using a 'Shape' class. Create subclasses 'Rectangle' and 'Circle' that calculate area. Use 'abstract' methods to enforce implementation. **[10M]**

Answer: Hierarchical inheritance involves multiple child classes inheriting from a single parent class. We use an abstract class to define the template.

```java
abstract class Shape {
    abstract void area(); // Abstract method
}

class Rectangle extends Shape {
```

```
    double l, b;
    Rectangle(double l, double b) { this.l = l; this.b = b; }

    @Override
    void area() {
        System.out.println("Area of Rectangle: " + (l * b));
    }
}

class Circle extends Shape {
    double r;
    Circle(double r) { this.r = r; }

    @Override
    void area() {
        System.out.println("Area of Circle: " + (3.14 * r * r));
    }
}

public class Main {
    public static void main(String[] args) {
        Shape s1 = new Rectangle(10, 5);
        Shape s2 = new Circle(7);

        s1.area();
        s2.area();
    }
}

```

Q.19 Write a Java AWT program to create a 'Frame' containing a 'Label', a 'TextField', and a 'Button'. Implement an 'ActionListener' such that clicking the button updates the label with the text entered in the text field. [10M]

Answer:

```
import java.awt.*;
import java.awt.event.*;

public class AWTExample extends Frame implements ActionListener {
    TextField tf;
    Label l;
    Button b;

    AWTExample() {
        // Layout and Components
        tf = new TextField();
        tf.setBounds(50, 50, 150, 20);

        l = new Label("Result here");
        l.setBounds(50, 100, 250, 20);

        b = new Button("Click Me");
        b.setBounds(50, 150, 60, 30);

        // Register Listener
        b.addActionListener(this);

        add(b); add(tf); add(l);
```

```
24
25          setSize(400, 400);
26          setLayout(null);
27          setVisible(true);
28
29          // Close operation
30          addWindowListener(new WindowAdapter() {
31              public void windowClosing(WindowEvent e) { dispose(); }
32          });
33      }
34
35      // Event Handling
36      public void actionPerformed(ActionEvent e) {
37          String text = tf.getText();
38          l.setText("You entered: " + text);
39      }
40
41      public static void main(String[] args) {
42          new AWTExample();
43      }
44  }
45
```

Q.20 Explain the concept of 'Polymorphism' in Java. Describe the two types: Compile-time (Static) and Runtime (Dynamic) with diagrams or examples. [10M]

Answer: Polymorphism means "many forms". It allows an entity (method or object) to behave differently in different contexts.

(a) **Compile-time Polymorphism (Static Binding):** Achieved via *Method Overloading*. The compiler determines which method to call based on the method signature (arguments) at compile time. *Example:* 'add(int a, int b)' vs 'add(double a, double b)'.

(b) **Runtime Polymorphism (Dynamic Binding):** Achieved via *Method Overriding*. The JVM determines which method to call based on the actual object type at runtime, not the reference variable type.