INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, SRI CITY

END TERM EXAMINATION – MONSOON 2025

Object Oriented Programming

CSE:UG2 /PC **Date: 22-11-2025**

Duration: 180 Mins (2.30-5.30 PM) **Max. Marks: 50**

Instructions: **Roll No:________________**

1. All questions are compulsory.
2. Write the answers neatly and clearly without any overwriting or corrections on the answer sheet; otherwise, an appropriate penalty will be imposed.
3. Attach the question paper with the answer sheet
4. Please write your assumptions as appropriate

Answer all the questions

PART-A: 20

1.	**Fill in the blanks.**	
a.	The process of defining multiple methods with the same name but different parameter lists within the same class is called ________, and it is resolved at ________ time. Function overloading/Polymorphis; Compile time	2 Marks
b.	When a class contains references to objects of other classes as its members, it is an example of the concept called ________. Composition/ Class Composition	1 Marks
c.	The ________ keyword is used inside a constructor to invoke another constructor within the same class, whereas the ________ keyword is used to call a constructor from the superclass. this()/this; super()/super	2 Marks
d.	The main method in a Java program must be declared as ________ so that it can be invoked without creating an object. static	1 Marks
e.	____Start________ method used to start a thread execution, and __Run_____ method must be defined by a class implementing the java.lang.Runnable interface?.	2 Marks
f.	_Collection_ is the root interface of the Java Collection framework hierarchy, and _SET_ collection does not allow duplicate elements.	2 Marks
g.	In Java, objects can be printed in a meaningful way when the class overrides ______ method of the Object class. toString	1 Mark

h.	All variables in an interface are ______ and _______ . static and final	1 Mark
2	**What will be the output/possible error of the following code snippets (consider all the required packages have been imported)**.	
a.	See code below	2 marks

a.

```
class Alpha {
   Alpha() {
      System.out.println("Alpha()");
      show();
   }
   void show() {
      System.out.println("Alpha show");
   }
}

class Beta extends Alpha {
   int val = 10;
   Beta() {
      System.out.println("Beta()");
      show();
   }
   void show() {
      System.out.println("Beta show: " + val);
   }
}

public class TestOutput {
   public static void main(String[] args) {
      Beta b = new Beta();
   }
}
```

```
Alpha()
Beta show: 0 // 1 mark
Beta()
Beta show: 10 // 1 mark
```

b.

```
class A {
   private int x = 10;
   class B {
      int x = 20;
      void print() {
```

	`System.out.println(A.this.x + " " + this.x);`
	`}`
	`}`
	`public static void main(String[] args) {`
	`B b = new B();`
	`b.print();`
	`}`
	`}`
	
	Hi //1 Mark
	ERROR! //1 Mark
	Exception in thread "main" java.lang.ArithmeticException: / by zero
	at Test.main(Main.java:7)	

c.	**error: non-static variable this cannot be referenced from a static context**	**2 marks**

d.	**Write the missing code suitable to start a thread ?**
	`class X implements Runnable`
	`{`
	`public static void main(String args[])`
	`{`
	`Thread t = new Thread(new X());`
	`t.start();`
	`}`
	`public void run()`
	`{`
	`}`
	`}`
		2 Marks

PART-B: 30

3(a)	`// Book.java`
	`class Book { // 1.5 mark`
	`String title;`
	`String author;`
	`double price;`
	
	`// Default constructor`
	`Book() {`	

```
        title = "Unknown Title";
        author = "Unknown Author";
        price = 0.0;
    }

    // Parameterized constructor
    Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    // Method to display book details
    void displayDetails() {
        System.out.println("Book Title  : " + title);
        System.out.println("Author      : " + author);
        System.out.println("Price (Rs.) : " + price);
    }
}

// Library.java
class Library { //2 mark
    Book book;  // Each Library has one Book

    // Constructor with a Book object
    Library(Book book) {
        this.book = book;
    }

    // Method to display the library's book details
    void showLibraryBook() {
        System.out.println("Library contains the following book:");
        book.displayDetails();
    }
}

// Main class
public class Main {//1.5 mark
    public static void main(String[] args) {
        // Create a Book using parameterized constructor
        Book b1 = new Book("The Alchemist", "Paulo Coelho", 399.0);

        // Create a Library that contains this book
        Library lib = new Library(b1);

        // Display book details via Library
```

	`lib.showLibraryBook();`
	
	`System.out.println();`
	
	`// Example with default book`
	`Book b2 = new Book();`
	`Library lib2 = new Library(b2);`
	`lib2.showLibraryBook();`
	`    }`
	`}`	

3(b)	Create a Calculator class to compute the area of a circle and a rectangle. The class should accept both integer and decimal inputs, using method overloading where appropriate. When integer values are provided, the computation should internally use the corresponding decimal-based logic. The value of π should be defined as a constant, and all area methods should be declared as static. The actual area calculation must be performed through private helper methods, and the class should handle invalid or zero dimensions gracefully. Write only the class definition (no main() method).	**5 Marks**

```
public class Calculator {

    // Constant value of π
    private static final double PI = 3.14159; // 0.5 mark

    // ----------------------------
    // Circle Area Methods
    // ----------------------------

    // Method for integer radius
    public static double area(int radius) {//0.5 mark
        return area((double) radius);
    }

    // Method for decimal radius
    public static double area(double radius) {//0.5 mark
        if (radius <= 0) {
            System.out.println("Invalid radius. Returning 0.");
            return 0;
        }
        return computeCircle(radius);
    }

    // ----------------------------
    // Rectangle Area Methods
    // ----------------------------

    // Method for integer dimensions//1 mark
    public static double area(int length, int breadth) {
        return area((double) length, (double) breadth);
    }

    // Method for decimal dimensions//1 mark
    public static double area(double length, double breadth) {
        if (length <= 0 || breadth <= 0) {
            System.out.println("Invalid dimensions. Returning 0.");
            return 0;
        }
        return computeRectangle(length, breadth);
    }

    // ----------------------------
    // Private Helper Methods
    // ----------------------------

    private static double computeCircle(double radius) {//0.5 mark
        return PI * radius * radius;
    }

    private static double computeRectangle(double length, double breadth) {//1 mark
        return length * breadth;
    }
}
```

4(a)

I. Explain the core interfaces of the Java Collection Framework and describe the major classes that implement them. Illustrate the complete hierarchy using a neat diagram.

2 Marks

(1 Mark)

1. List Interface: Elements are Ordered, Allows duplicates
   Implementing Classes: ArrayList, LinkedList, Vector, Stack

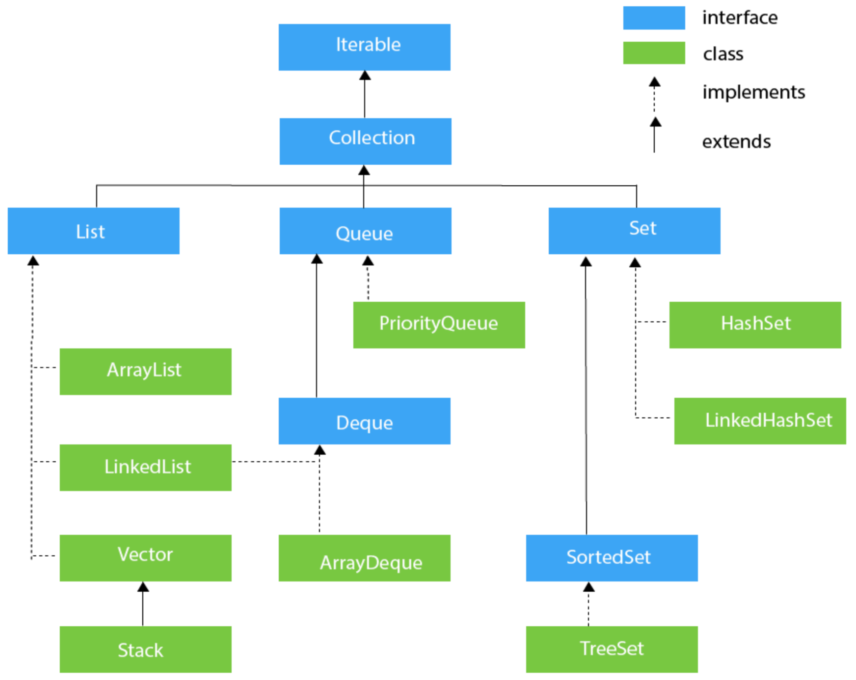
2. Set Interface: No duplicate elements are allowed
   Implementing Classes: HashSet, LinkedHashSet

3. SortedSet Interface (extends Set): Elements in sorted order
   Implementing Class: TreeSet

4. Queue Interface: Typically FIFO ordering
   Implementing Classes: PriorityQueue, LinkedList, ArrayDeque

5. Deque Interface (extends Queue): Double-ended queue
   Implementing Classes: ArrayDeque, LinkedList

(1 mark)



II. Write a Java program using the Collection Framework to find the second-largest element in an ArrayList of integers. Your program should accept a list of values, process them using appropriate Collection methods, and display the second-highest number present in the list. Ensure that the solution handles cases where duplicate values exist and the list contains fewer than two distinct elements. — 3 Marks

```
import java.util.*;
public class SecondLargestArrayList {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter number of elements: ");
      int n = sc.nextInt();
      ArrayList<Integer> list = new ArrayList<>();    ( 1 mark)
```

```
        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            list.add(sc.nextInt());
        }
        if (list.size() < 2) {
            System.out.println("Second largest element does not exist.");
            Return;
    }
       ArrayList<Integer> unique = new ArrayList<>();        ( 1 mark)
        for (Integer num : list) {
            if (!unique.contains(num)) {
                unique.add(num);
             }
        }
        if (unique.size() < 2) {                              ( 1 mark)
                 System.out.println("2nd largest element does not exist (fewer than two
distinct values).");
            return;
       }
        Collections.sort(unique);
        int secondLargest = unique.get(unique.size() - 2);
        System.out.println("Second largest element: " + secondLargest);
    }
}
```

4(b)

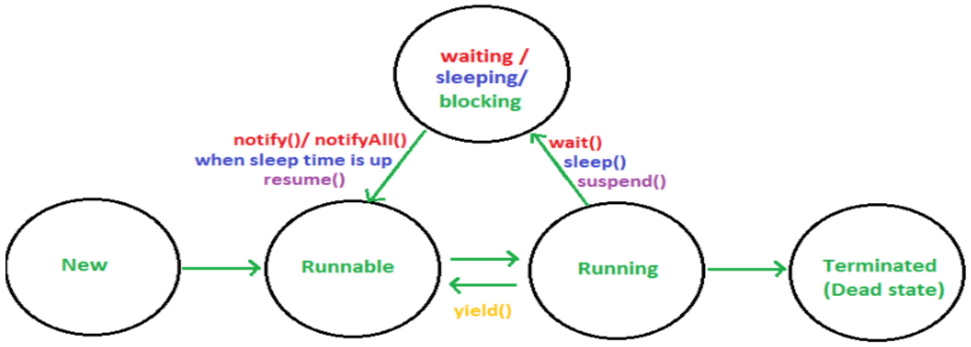
I. Explain various thread states in the life cycle of a thread with a neat diagram.

2 Marks

(1 mark) **New:** A thread is in the New state when an object of the Thread class is created but the start() method has not been called.

Runnable: When start() is called, the thread becomes Runnable.
It means the thread is ready to run and waiting for CPU time.
It may not be running immediately; the scheduler decides when it runs.
Running: A thread is in the Running state when it actually gets CPU time.
JVM moves a thread from Runnable → Running based on the thread scheduler.
Waiting: A thread enters Waiting when it waits indefinitely for another thread to perform a task.
Methods that cause Waiting: wait(), join() (without timeout), park()
Terminated: A thread enters the Terminated state when: run() method finishes OR The thread is stopped due to an exception. After termination, a thread cannot restart.

(1 mark)



3 Marks

II.

```
class Counter {
   private int count = 0;
   public synchronized void increment() {
      count++;
   }

   public int getCount() {
      return count;
   }
}

class MyTask implements Runnable {
   Counter counter;
   MyTask(Counter c) {
      this.counter = c;
   }

   public void run() {
      counter.increment();
   }
}

public class SyncExam {
   public static void main(String[] args) throws InterruptedException {

      Counter c = new Counter();
      Thread t1 = new Thread(new MyTask(c));
      Thread t2 = new Thread(new MyTask(c));
      t1.start();
      t2.start();
      t1.join();
      t2.join();
      System.out.println("Final Count: " + c.getCount());
   }
```

}

a) Explain What is Mutual Exclusion and where do you notice that in the above code? (1 mark)

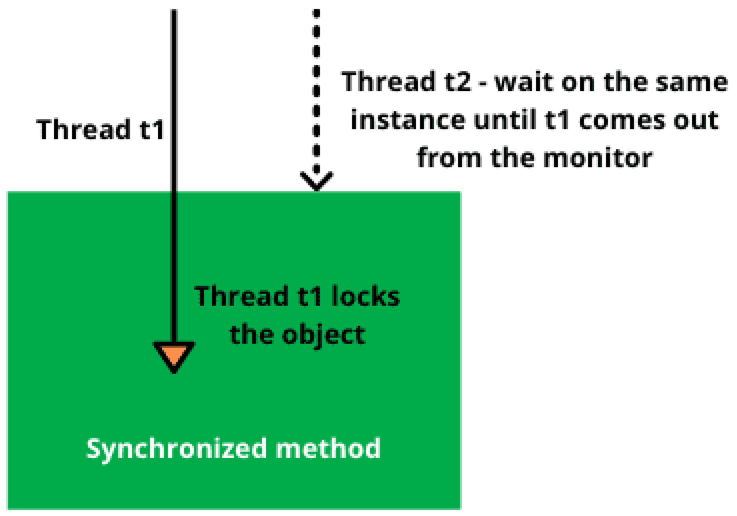
Mutual Exclusion means that only one thread can access a shared resource (critical section) at a time.
This prevents race conditions and ensures data consistency when multiple threads operate on shared data. Mutual exclusion is achieved through the synchronized keyword:

```
public synchronized void increment() {
   count++;
}
```

b) Discuss what might happen if the increment() method is not synchronized? (1 mark)

Two threads may read the same value of count simultaneously and overwrite each other’s update.
Incorrect and unpredictable output of count.
Loss of data consistency.

c) Draw a small diagram showing how both threads try to access the synchronized method but only one is allowed at a time due to the object lock. (1 mark)


Thread t1
Thread t2 - wait on the same instance until t1 comes out from the monitor
Thread t1 locks the object
Synchronized method

5

a. i) **Unreachable catch error** occurs in Java when a catch block can never be executed because an earlier catch block has already caught the same exception type or its parent class.   //1 Mark

Since Java checks exception hierarchy during compilation, the compiler detects that the later catch block will never run and reports:

“exception <type> has already been caught” or **“unreachable catch block.”**

Example: //1 Mark

```
try {
int x = 10 / 0;     // ArithmeticException
}
catch (Exception e) {    // Parent class
System.out.println("Exception caught");
}
catch (ArithmeticException ae) {   // Child class - unreachable!
  System.out.println("ArithmeticException caught");
}
```

ii) User-defined exceptions in Java **can be either checked or unchecked**, depending on the class they extend.
If a user-defined exception extends Exception class, it becomes a checked exception. Such exceptions must be handled using try-catch or declared using throws.
Example: class MyCheckedException extends Exception { }   //1 Mark

If a user-defined exception extends RuntimeException, it becomes an unchecked exception. These exceptions do not need to be explicitly handled or declared.

Example: class MyUncheckedException extends RuntimeException { } //1 Mark

b)
```
interface SmartDevice {   //0.5 Mark

    void turnOn();

    void turnOff();

    int calculatePowerUsage(int hours);

}

interface SpeedControllable {  //0.5 Mark

    void setSpeed(int level);
```

```
}
abstract class AbstractSmartDevice implements SmartDevice {   //1 Mark
    protected String brand;
    public AbstractSmartDevice(String brand) {
        this.brand = brand;
    }
    public void showBrand() {
        System.out.println("Brand: " + brand);
    }
}
class SmartLight extends AbstractSmartDevice {   //1 Mark

    public SmartLight(String brand) {
        super(brand);
    }
    public void turnOn() {
        System.out.println("SmartLight ON");
    }
    public void turnOff() {
        System.out.println("SmartLight OFF");
    }
    public int calculatePowerUsage(int hours) {
        return hours * 10; // Example consumption logic
    }
```

```
}
class SmartFan extends AbstractSmartDevice implements SpeedControllable { //1Mark
    public SmartFan(String brand) {
        super(brand);
    }
    public void turnOn() {
        System.out.println("SmartFan ON");
    }
    public void turnOff() {
        System.out.println("SmartFan OFF");
    }
    public int calculatePowerUsage(int hours) {
        return hours * 30;
    }
    public void setSpeed(int level) {
        System.out.println("SmartFan speed set to " + level);
    }
}
// SmartAC supports speed control
class SmartAC extends AbstractSmartDevice implements SpeedControllable { //1 Mark
    public SmartAC(String brand) {
        super(brand);
    }
    public void turnOn() {
        System.out.println("SmartAC ON");
    }
    public void turnOff() {
        System.out.println("SmartAC OFF");
    }
    public int calculatePowerUsage(int hours) {
        return hours * 100;
    }
    public void setSpeed(int level) {
        System.out.println("SmartAC cooling level set to " + level);
    }
}
```

Justification

Why an Interface?

All smart devices must support turnOn(), turnOff(), and calculatePowerUsage(int hours). These are behavioural requirements that every device must implement differently.

Interfaces are ideal for defining common behaviour without implementation.

→ Therefore, define an interface SmartDevice with:

turnOn(), turnOff(), calculatePowerUsage(int hours).

Why an Abstract Class?

All devices share:

a common field brand

a common method showBrand()

Abstract classes allow fields, constructors, and implemented methods, which interfaces do not (for instance variables).

Also, devices still need to implement the interface methods.

→ Therefore, create an abstract class AbstractSmartDevice that:

Stores brand

Implements showBrand()

Implements the interface SmartDevice

Why Another Interface for Speed Control?

Only some devices (SmartFan, SmartAC) have speed control.

This is an optional capability, so use a separate interface.

	→ Create SpeedControllable interface containing: setSpeed(int level)	