SRN | | | | | | | | | | | | |

PES University, Bengaluru-85
(Established under Karnataka Act No. 16 of 2013)

Jan 2024: ASSESSMENT, CSE, VI SEMESTER

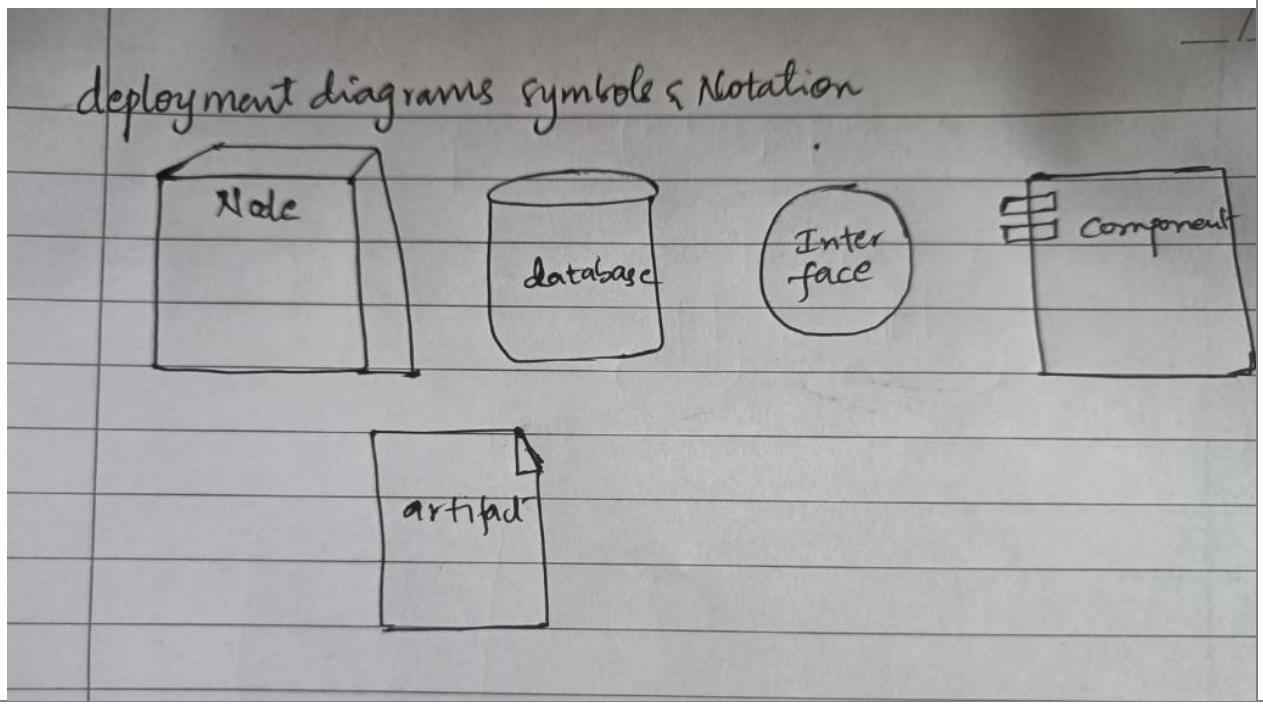
UE21CS352B-OOADJ

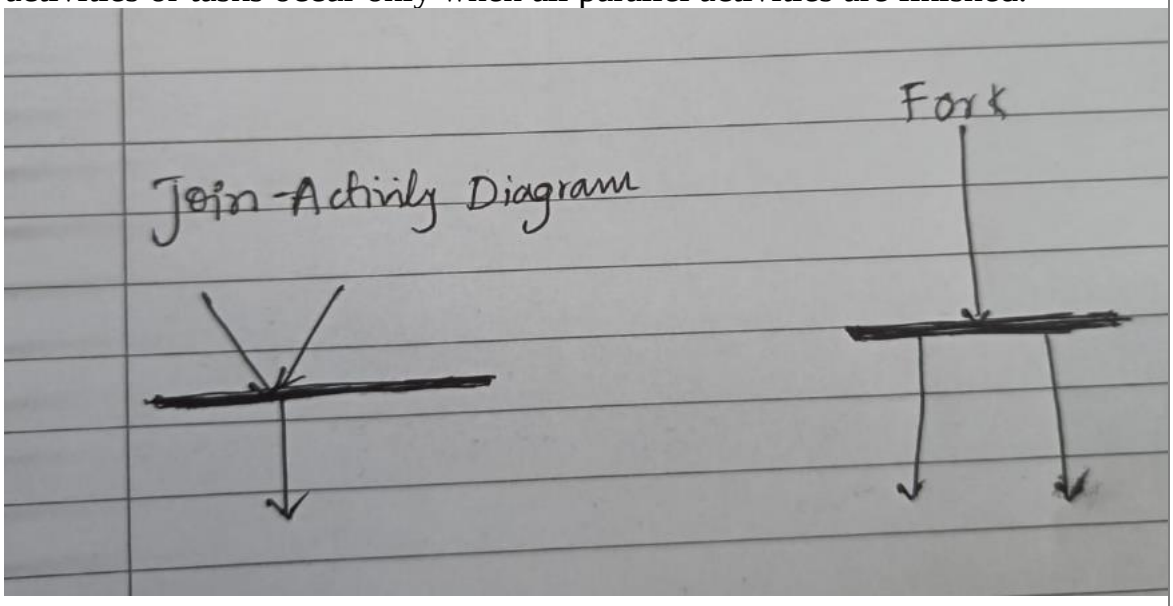
NAME : NAGAVENI L G

SRN : PES2UG21CS315

SEC : F

Time: 1 Hrs. Answer All Questions Max Marks: 20

1	a)	What is a deployment diagram?Draw Some of the deployment diagram symbols and notations A deployment diagram in Unified Modeling Language (UML) is a type of diagram that visualizes the physical deployment of software components and their interactions in a system. It illustrates how software components are deployed on hardware nodes and how they communicate with each other over the network. Deployment diagrams are particularly useful for understanding the distribution and configuration of a system. 	2M

		SRN	
	b)	Differentiate between Fork and Join with proper notations. In which UML diagram are they used and why? Illustrate with a suitable example. Fork: • In UML activity diagrams, Fork represents a control flow element that denotes the point at which a single flow of control is split into multiple parallel flows. • It signifies that multiple activities or tasks can be executed concurrently. • Forks are used to model concurrency in a process, allowing for the simultaneous execution of independent activities. Join: • In UML activity diagrams, Join represents a control flow element that denotes the point at which multiple parallel flows of control converge back into a single flow. • It indicates synchronization, ensuring that the flow of control waits until all parallel activities are completed before proceeding. • Joins are used to model synchronization in a process, ensuring that certain activities or tasks occur only when all parallel activities are finished. •  -used in activity diagram	r

SRN

Reasons:

Fork:

- Indicates a point where a single flow of control splits into multiple parallel flows.
- Models concurrent execution of independent tasks or activities.
- Represents the start of parallel paths in the activity diagram.
- Enhances the readability of the diagram by explicitly showing concurrency.

Join:

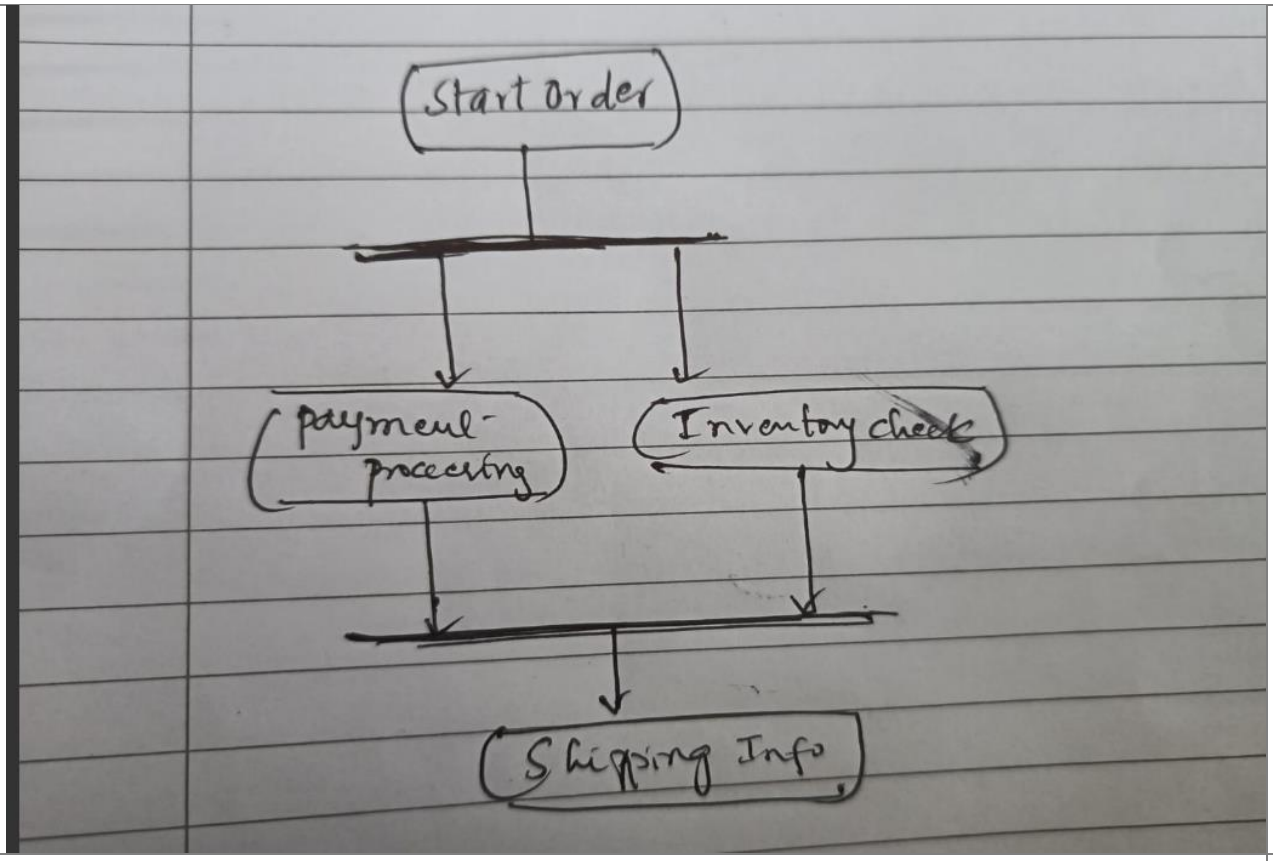
- Represents the point where multiple parallel flows converge into a single flow.
- Ensures synchronization by waiting for all parallel activities to complete.
- Coordinates the flow of control, preventing further progress until all parallel paths are done.
- Enhances the expressiveness of the activity diagram, making it clear where synchronization occurs.

EXAMPLE:

Scenario: Ordering and processing a book online

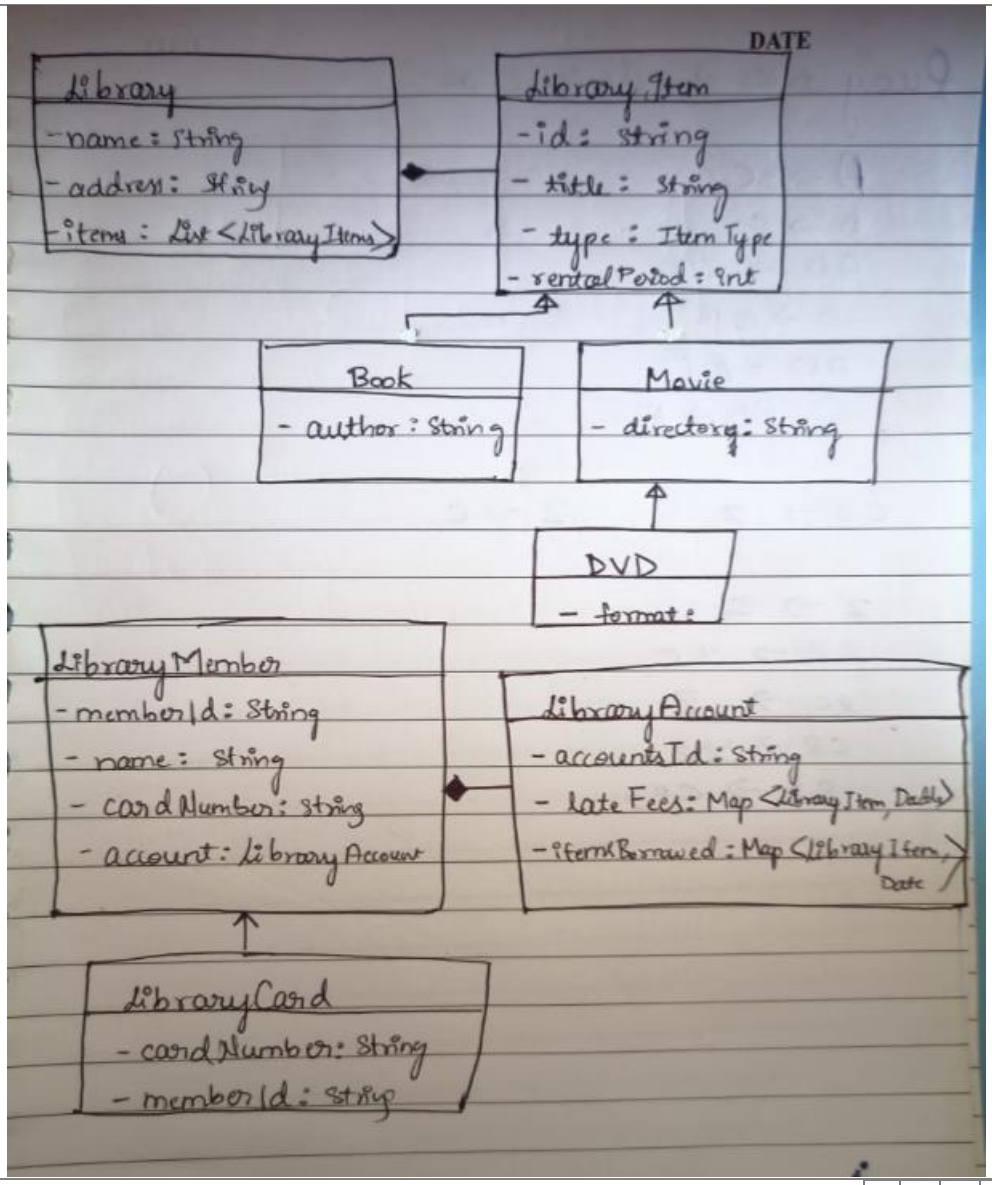
Start: Customer places an order for a book online.

Fork: The order proceeds in two parallel paths:payment processing and inventory check
if both are successful shipping information is collected from the customer.



c) Create a class diagram for a web based public library. A public library typically stores a collection of books, movies, or other library items available to be borrowed by people living in a community. Each library member typically has a library account and a library card with the account's ID number, which she can use to identify herself to the library. A member's library account records which items the member has borrowed and the due date for each borrowed item. Each type of item has a default rental period, which determines the item's due date when the item is borrowed. If a member returns an item after the item's due date, the member owes a late fee specific for that item, an amount of money recorded in the member's library account — 5M

Draw a class diagram which consists of all the classes in your system their attributes and operations, relationships between
the classes, multiplicity specifications, and other model elements that you find appropriate



2	a)	Write a Java program to find the maximum elements in a stack. Implement only the below class	2M

```
import java.util.Stack;

public class MaxStack {
    private Stack<Integer> stack;
    private int max;

    public MaxStack() {
        this.stack = new Stack<>();
        this.max = Integer.MIN_VALUE;
    }
```

```
    public void push(int element) {
        if (element > max) {
            stack.push(max);  // Save the current max before updating
            max = element;
        }
        stack.push(element);
    }

    public int pop() {
        if (stack.isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }

        int poppedElement = stack.pop();
        if (poppedElement == max) {
            max = stack.isEmpty() ? Integer.MIN_VALUE : stack.pop();
        }

        return poppedElement;
    }

    public int get_Max() {
        return max;
    }

    public static void main(String[] args) {
        MaxStack maxStack = new MaxStack();


        maxStack.push(3);
        maxStack.push(5);
        maxStack.push(2);
        maxStack.push(7);

        System.out.println("Maximum element in the stack: " + maxStack.get_Max());

        maxStack.pop();
        maxStack.pop();

        System.out.println("Maximum element in the stack after pops: " +
maxStack.get_Max());
    }
}
```

b) **Write the output of the below program** — 2M

i)

```
public class Code {
    public static void main(String[] args) {
        method(null);
    }
    public static void method(Object o) {
        System.out.println("Object method");
    }
    public static void method(String s) {
        System.out.println("String method");
    }
}
```

Output : String method

ii)

```
public class Code
{
    public static void main(String args[])
    {
        StringBuffer str1 = new StringBuffer("open");
        StringBuffer str2 = str1;
        str1.append("genus");
        System.out.println(str1 + " " + str2 + " " + (str1 == str2));
    }
}
```

Output : opengenus opengenus true

c) What are different types of constructors? Explain each with an example code — 2M

Default Constructor:

A default constructor is provided by Java automatically if no explicit constructor is defined in the class. It initializes the object with default values.

```
public class MyClass {
    // Default constructor is automatically provided
    public MyClass() {
        System.out.println("Default constructor called");
    }

    public static void main(String[] args) {
        MyClass obj = new MyClass();  // Default constructor called when object is created
    }
}
```

Parameterized Constructor:

A parameterized constructor is one that takes parameters and initializes the object with the provided values.

```java
public class Student {
    String name;
    int age;

    // Parameterized constructor
    public Student(String studentName, int studentAge) {
        name = studentName;
        age = studentAge;
    }

    public static void main(String[] args) {
        // Creating an object with a parameterized constructor
        Student student1 = new Student("John", 20);
        System.out.println("Name: " + student1.name + ", Age: " + student1.age);
    }
}
```

Copy Constructor:

A copy constructor creates a new object by copying the values from an existing object of the same type.

```java
public class Point {
    int x, y;

    // Copy constructor
    public Point(Point otherPoint) {
        this.x = otherPoint.x;
        this.y = otherPoint.y;
    }

    public static void main(String[] args) {
        Point point1 = new Point();
        point1.x = 10;
        point1.y = 20;

        // Using copy constructor to create a new object
        Point point2 = new Point(point1);
        System.out.println("Point 2: (" + point2.x + ", " + point2.y + ")");
    }
}
```

Private Constructor:

A private constructor is used to prevent the instantiation of a class from outside the class itself. It is often used in classes that contain only static methods or constants.

```java
public class UtilityClass {
    // Private constructor to prevent instantiation
    private UtilityClass() {
        throw new AssertionError();  // To prevent instantiation even from within the class
    }

    // Static methods and constants can be defined here
}
```

	d)	**What is an interface in java?** SRN
		
		An interface is a collection of abstract methods. It defines a contract that classes must adhere to by providing concrete implementations for all the methods declared in the interface. Interfaces allow for achieving abstraction and multiple inheritances, as a class can implement multiple interfaces
		
		The below code is the interface of car, implement the interface of the car .
		
		interface Car {
		// Change the speed of the car
		void accelerate(int acceleration);
		
		// Turn the car left or right
		void turn(String direction);
		
		// Apply brakes to slow down the car
		void brake(int pressure);
		
		// Turn on the headlights
		void turnOnHeadlights();
		}
		ANSWER:
			5M

```java
// Car interface
interface Car {
    // Change the speed of the car
    void accelerate(int acceleration);

    // Turn the car left or right
    void turn(String direction);

    // Apply brakes to slow down the car
    void brake(int pressure);

    // Turn on the headlights
    void turnOnHeadlights();
}

// Concrete class implementing the Car interface
class SportsCar implements Car {
    private int speed;

    // Constructor
    public SportsCar() {
        this.speed = 0;
    }

    @Override
    public void accelerate(int acceleration) {
        this.speed += acceleration;
```

```
        System.out.println("Accelerating. Current speed: " + this.speed + " km/h");
    }

    @Override
    public void turn(String direction) {
        System.out.println("Turning " + direction);
    }

    @Override
    public void brake(int pressure) {
        this.speed -= pressure;
        System.out.println("Applying brakes. Current speed: " + this.speed + "
km/h");
    }

    @Override
    public void turnOnHeadlights() {
        System.out.println("Headlights turned on");
    }
}

// Main class for testing
public class CarImplementationExample {
    public static void main(String[] args) {
        // Create an instance of SportsCar
        SportsCar sportsCar = new SportsCar();

        // Use methods from the Car interface
        sportsCar.accelerate(50);
        sportsCar.turn("left");
        sportsCar.brake(20);
        sportsCar.turnOnHeadlights();
    }
}
```