



**SCHOOL OF
COMPUTING**

LAB RECORD

**23CSE111- OBJECT ORIENTED
PROGRAMMING**

SUBMITTED BY:

M.CHARULATHA

BACHELOR IN TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

**AMRITA VISHWA VIDYAPEETHAM
AMRITA SCHOOL OF COMPUTING**

CHENNAI

MARCH-2025



SCHOOL OF
COMPUTING

AMRITA VISHWA VIDYAPEETHAM
AMRITA SCHOOL OF COMPUTING, CHENNAI

BONAFIDE CERTIFICATE

This is to certify that the Lab Record work for
23CSE111- Object Oriented Programming
Subject submitted by **CH.SC.U4CSE24029 –**
M.CHARULATHA in “**Computer Science and**
Engineering” is a bonafide record of the work
carried out under my guidance and supervision at
Amrita School of Computing, Chennai.

This Lab examination held on 11/03/2025

Internal Examiner 1

Internal Examiner2

JAVA-LAB MANUAL

UML DIAGRAMS

1.BANKING SYSTEM

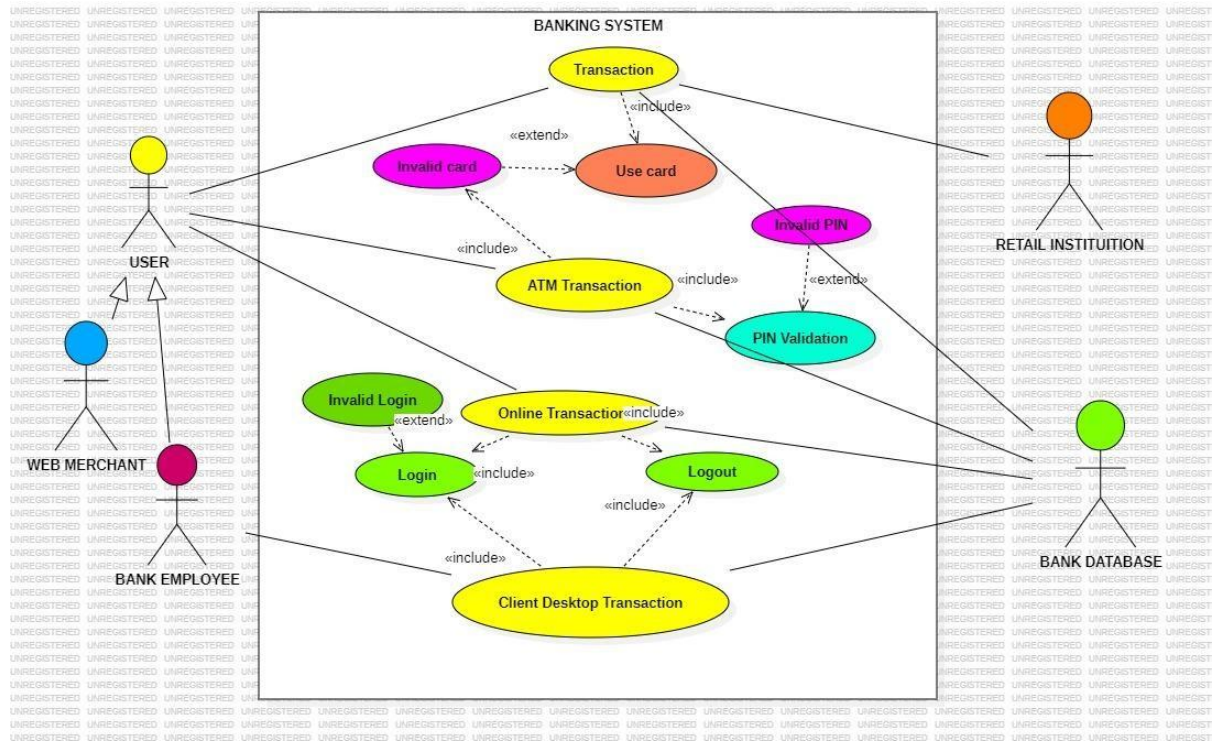
AIM:

To Demonstrate the working of the Banking System using different UML Diagrams.

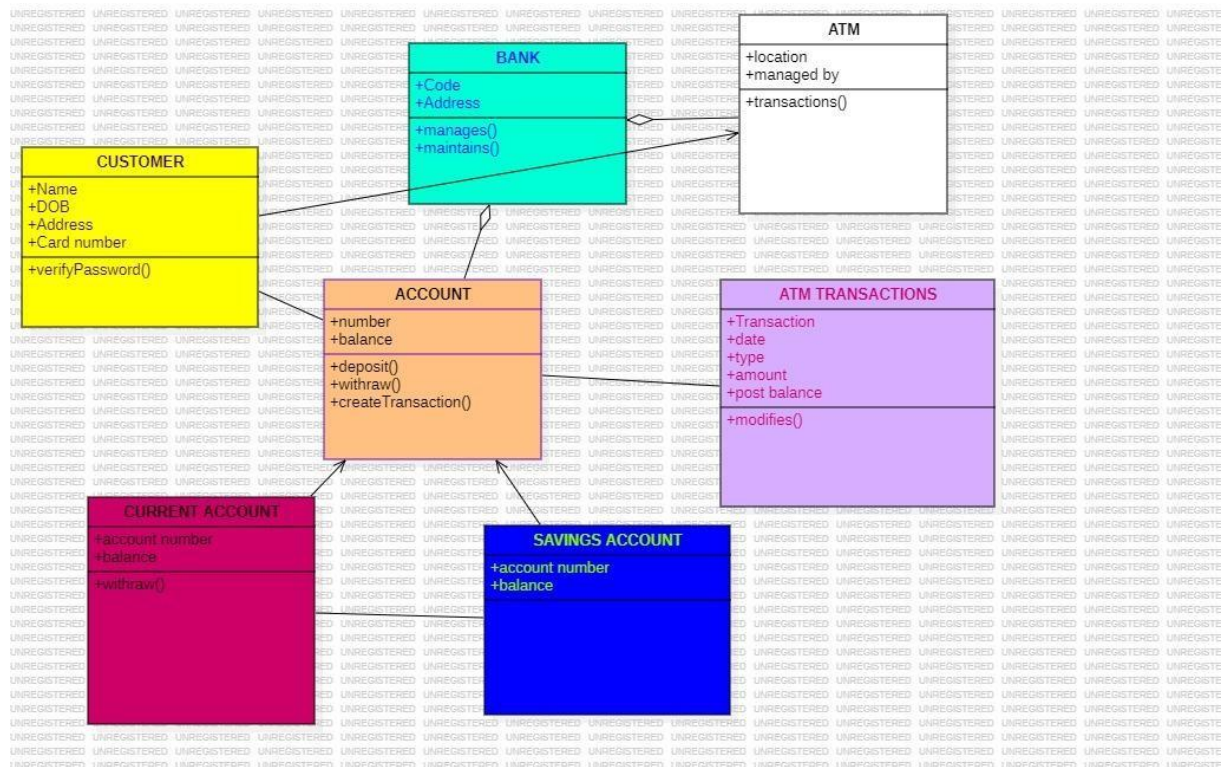
ALGORITHM:

1. Initialize the system
2. User Authentication
3. Main Menu
4. Account Creation
5. Deposit Money
6. Withdraw Money
7. Check Balance
8. Transfer Money
9. Exit System

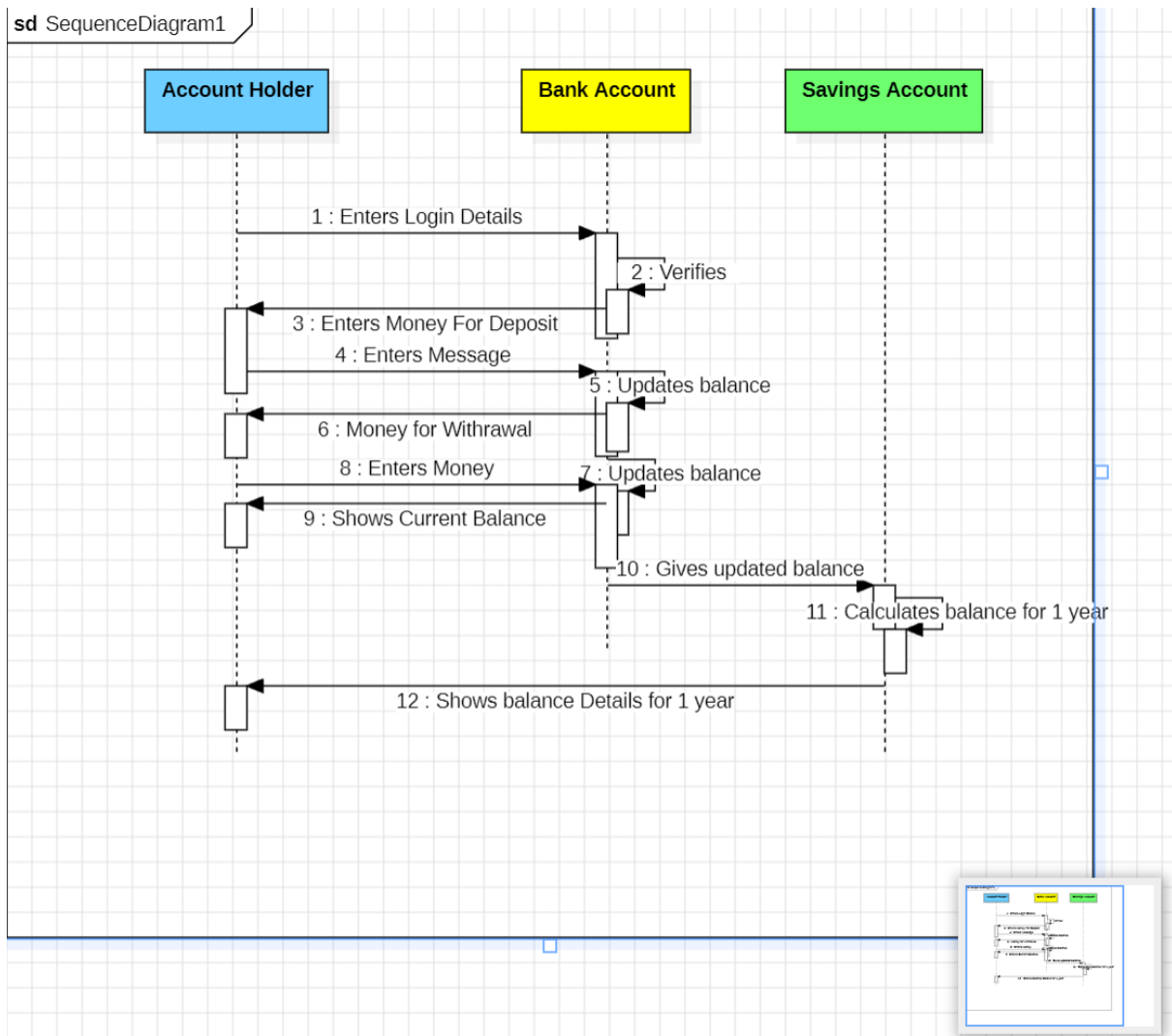
A) Use- Case Diagram



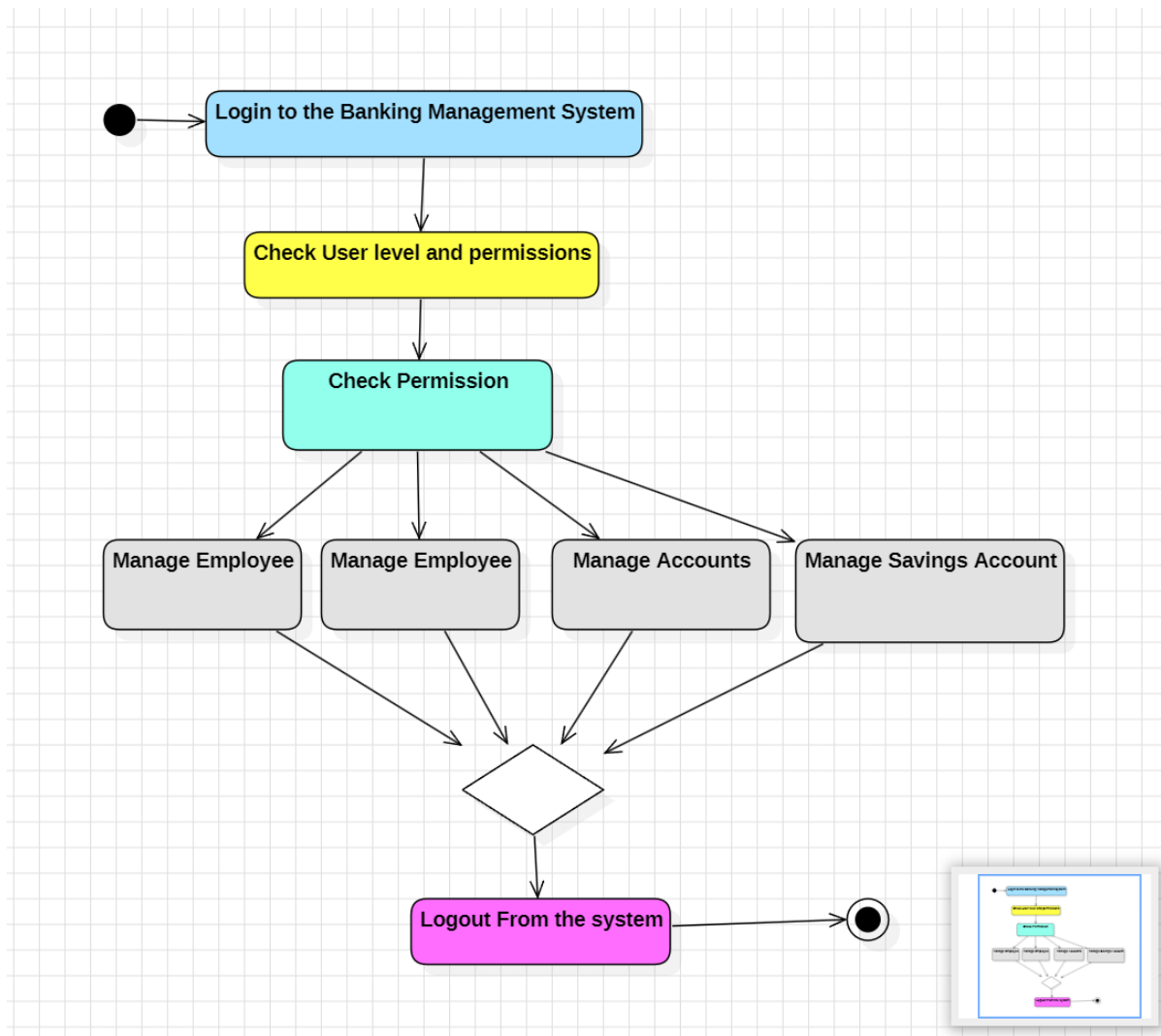
B) Class Diagram



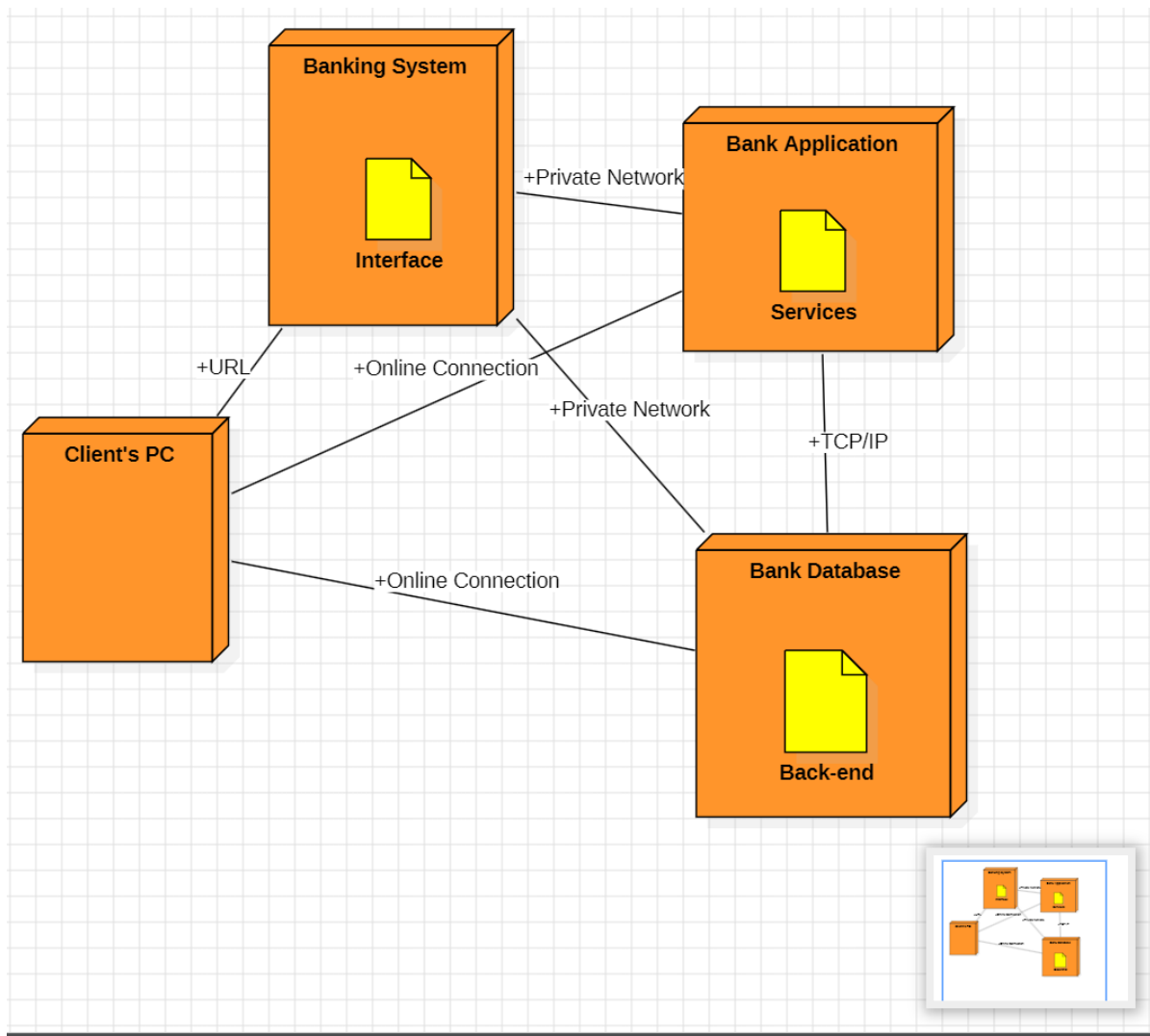
C) Sequence Diagram



D) Activity Diagram



E) Deployment Diagram



2.LIBRARY MANAGEMENT

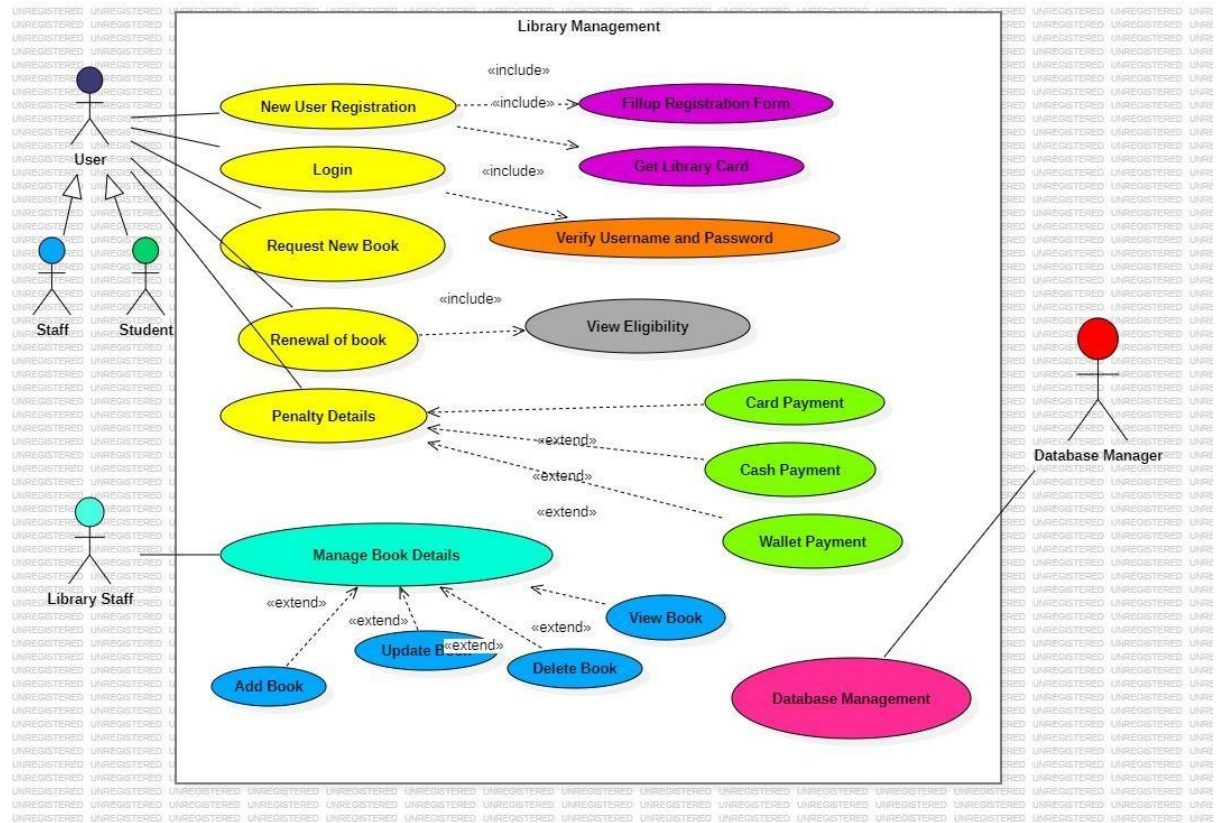
AIM:

To Demonstrate the working of the Library Management System using different UML Diagrams.

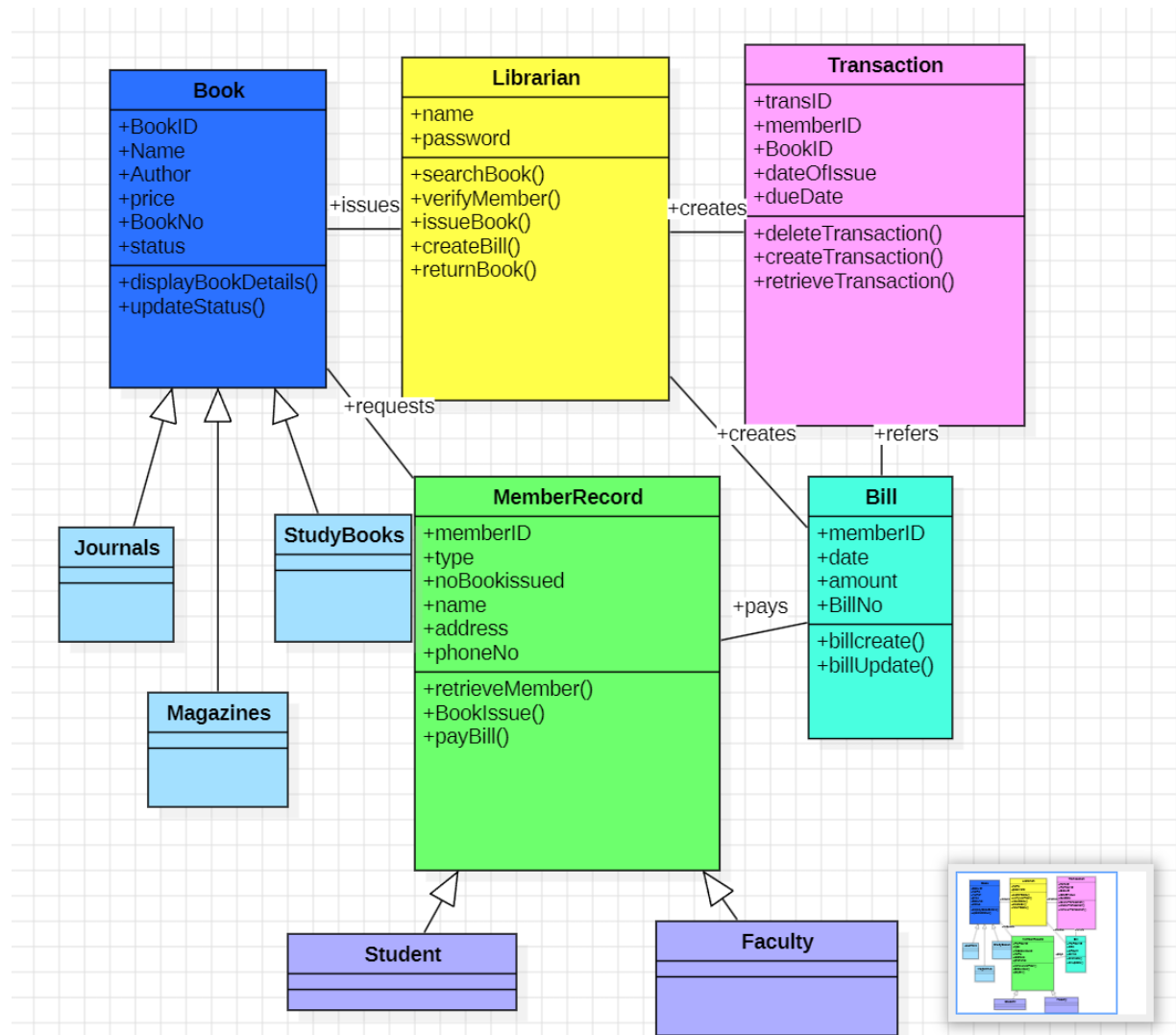
ALGORITHM:

1. Initialize System
2. User Authentication
3. Main Menu
4. Add New Book
5. Issue Book
6. Return Book
7. Search Book
8. Remove Book
9. View Borrowed Books
10. Exit System

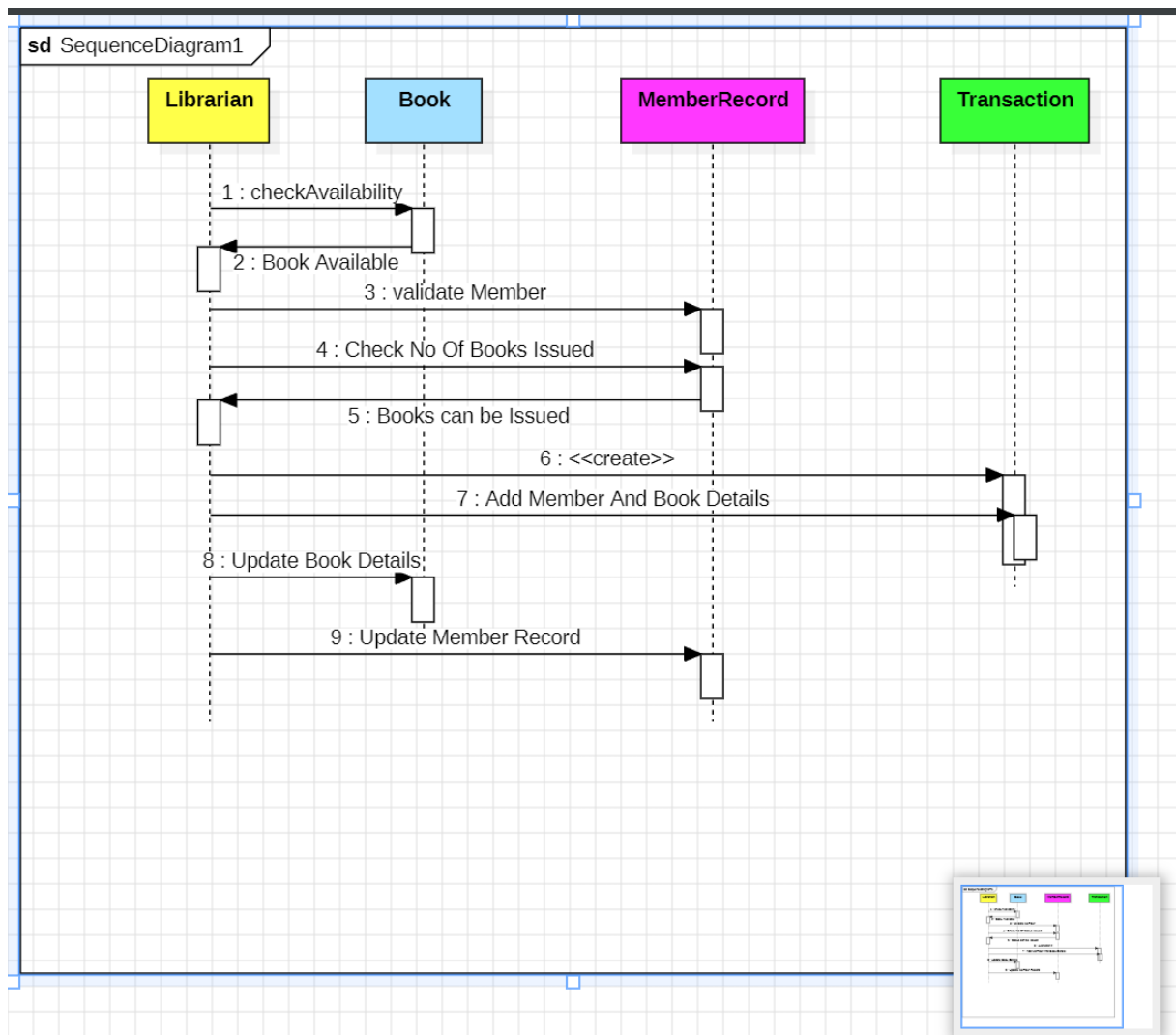
A) Use-Case Diagram



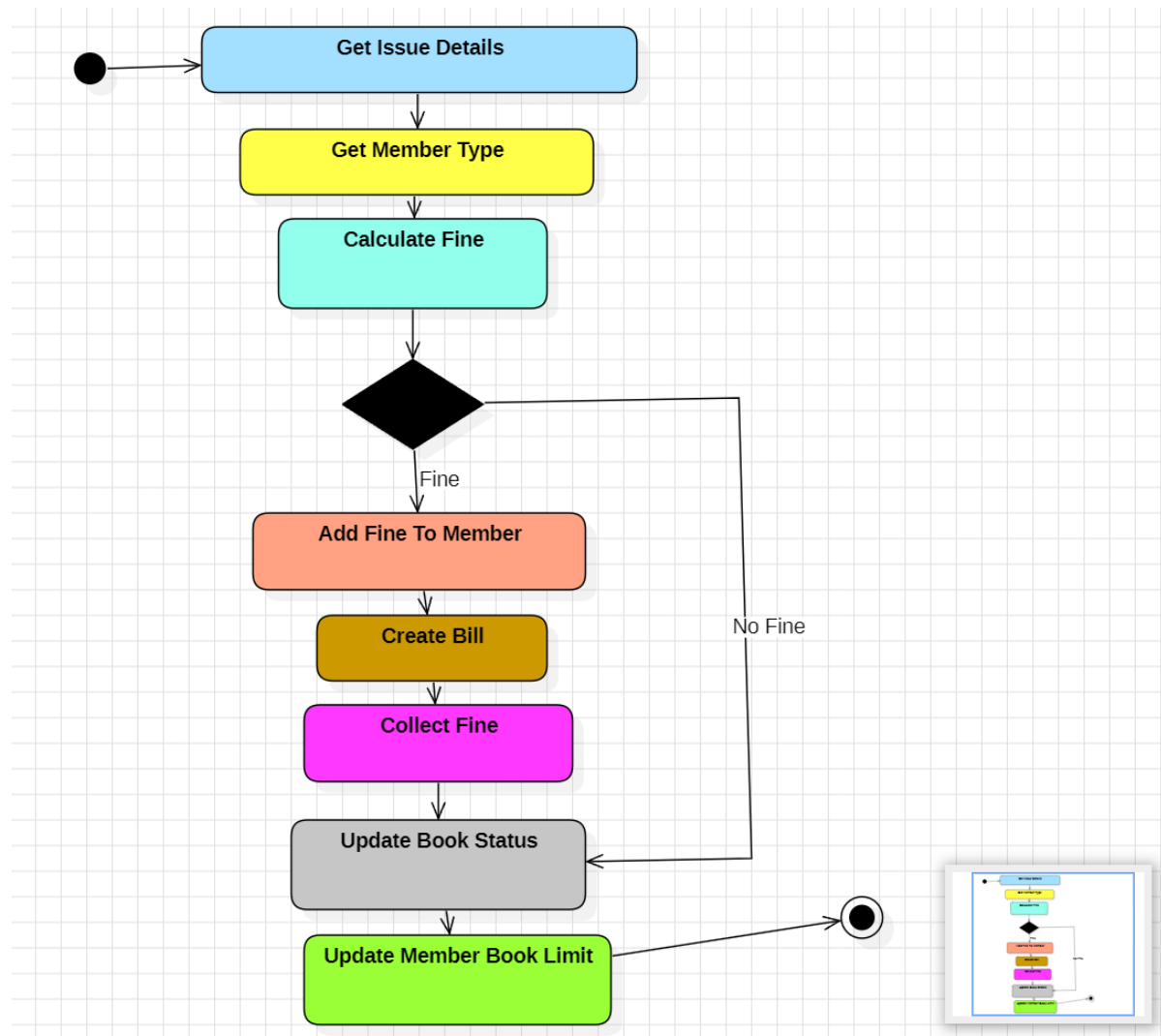
B) Class Diagram



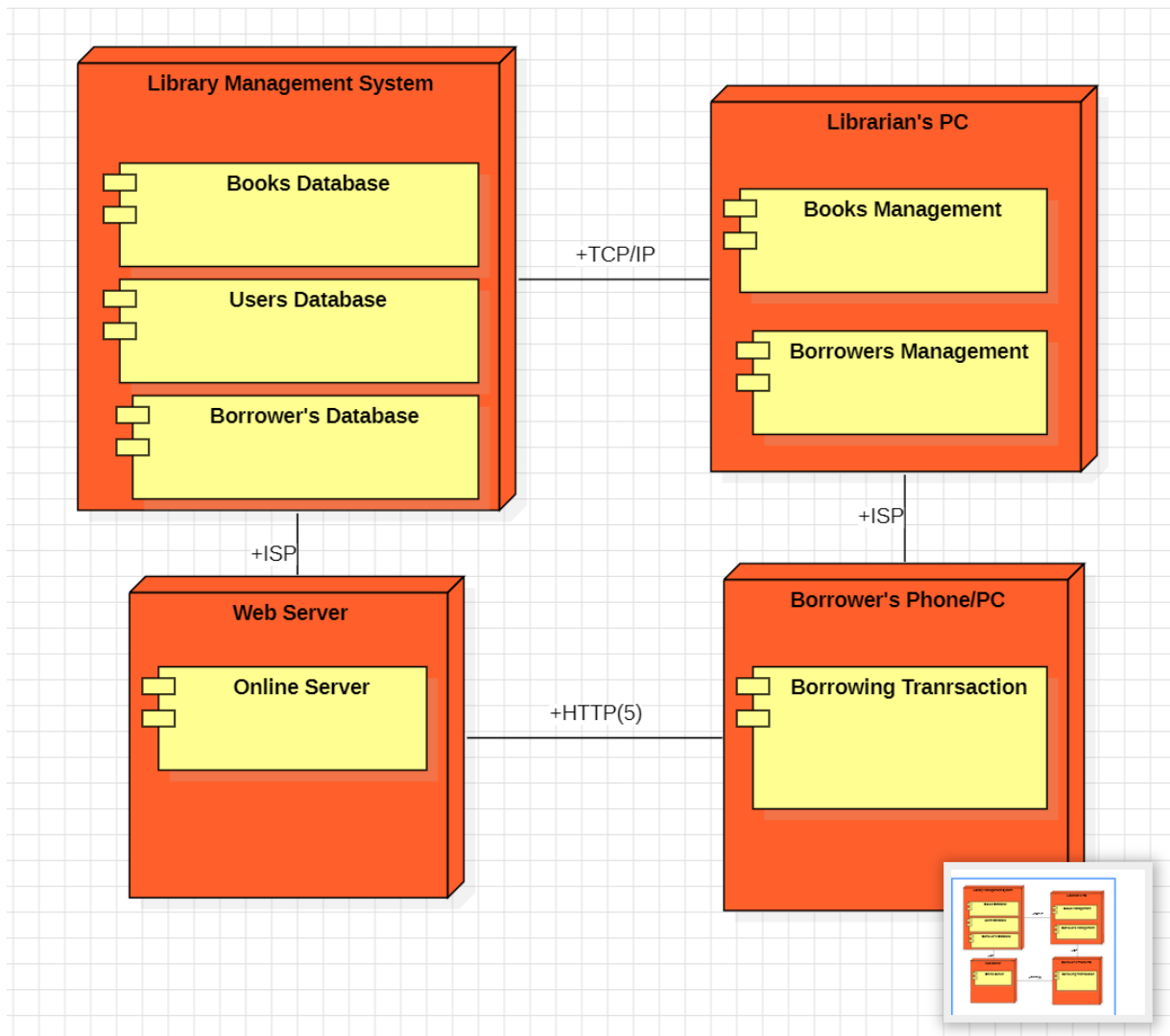
C) Sequence Diagram



D) Activity Diagram



E) DEPLOYMENT DIAGRAM:



BASIC JAVA PROGRAM

1.STUDENT MANAGEMENT SYSTEM

AIM:

- Use inheritance for different types of students (e.g., Undergraduate, Graduate).
- Implement polymorphism for displaying student details.
- Use loops for managing student records.

ALGORITHM:

1. Define base class Student.
2. Create derived classes Undergraduate and Graduate.
3. Override `displayDetails()` method in subclasses.
4. Initialize an array of students.
5. Loop through the array and display details.

CODING:

```
class Student {  
    String name;  
    int id;  
  
    Student(String name, int id) {  
        this.name = name;  
        this.id = id;  
    }  
  
    void displayDetails() {  
        System.out.println("Student ID: " + id + ", Name: " + name);  
    }  
}
```

```
}
```

```
// Derived class for Undergraduate students
```

```
class Undergraduate extends Student {
```

```
    String major;
```

```
    Undergraduate(String name, int id, String major) {
```

```
        super(name, id);
```

```
        this.major = major;
```

```
    }
```

```
    @Override
```

```
    void displayDetails() {
```

```
        System.out.println("Undergraduate Student - ID: " + id + ", Name: " + name + ", Major: " + major);
```

```
    }
```

```
}
```

```
class Graduate extends Student {
```

```
    String researchTopic;
```

```
    Graduate(String name, int id, String researchTopic) {
```

```
        super(name, id);
```

```
        this.researchTopic = researchTopic;
```

```
    }
```

```
    @Override
```

```

    void displayDetails() {
        System.out.println("Graduate Student - ID: " + id + ", Name: " +
name + ", Research Topic: " + researchTopic);
    }
}

public class StudentManagement {
    public static void main(String[] args) {
        Student students[] = new Student[3];

        students[0] = new Undergraduate("Alice", 101, "Computer
Science");
        students[1] = new Graduate("Bob", 102, "Quantum Computing");
        students[2] = new Undergraduate("Charlie", 103, "Mechanical
Engineering");
        for (Student s : students) {
            s.displayDetails();
        }
    }
}

```

OUTPUT:

```

Undergraduate Student - ID: 101, Name: Alice, Major: Computer Science
Graduate Student - ID: 102, Name: Bob, Research Topic: Quantum Computing
Undergraduate Student - ID: 103, Name: Charlie, Major: Mechanical Engineering

```

2.LIBRARY MANAGEMENT SYSTEM

AIM:

- Base class: Book, Derived classes: Ebook, PrintedBook.
- Use polymorphism for book details display.
- Loop through books to manage borrowing and returning.

ALGORITHM:

1. Define base class Book.
2. Create derived classes Ebook and PrintedBook.
3. Override displayInfo() method in subclasses.
4. Initialize an array of books.
5. Loop through the array and display book details.

CODING:

```
class Ebook extends Book {  
    int fileSize;  
  
    Ebook(String title, String author, int fileSize) {  
        super(title, author);  
        this.fileSize = fileSize;  
    }  
  
    @Override  
    void displayInfo() {  
        System.out.println("Ebook: " + title + " by " + author + " (Size: " +  
fileSize + "MB)");  
    }  
}
```

```
}  
  
class PrintedBook extends Book {  
    int pages;  
  
    PrintedBook(String title, String author, int pages) {  
        super(title, author);  
        this.pages = pages;  
    }  
  
    @Override  
    void displayInfo() {  
        System.out.println("Printed Book: " + title + " by " + author + "  
(Pages: " + pages + ")");  
    }  
}  
  
public class LibraryManagement {  
    public static void main(String[] args) {  
        // Creating an array of Book objects (looping used)  
        Book books[] = new Book[3];  
  
        books[0] = new Ebook("Java Basics", "John Doe", 5);  
        books[1] = new PrintedBook("Data Structures", "Jane Smith", 300);  
        books[2] = new Ebook("Machine Learning", "Alan Turing", 10);  
        for (Book b : books) {  
            b.displayInfo();  
        }  
    }  
}
```

```
}
```

OUTPUT:

```
Ebook: Java Basics by John Doe (Size: 5MB)
Printed Book: Data Structures by Jane Smith (Pages: 300)
Ebook: Machine Learning by Alan Turing (Size: 10MB)
```

3.BANKING SYSTEM

AIM:

- Base class: Account, Derived classes: SavingsAccount, CurrentAccount.
- Override methods for withdrawal and deposit behavior.
- Use loops to handle multiple transactions.

ALGORITHM:

1. Define base class Account.
2. Create derived classes SavingsAccount and CheckingAccount.
3. Override withdraw() method in subclasses.
4. Initialize an array of accounts.
5. Loop through accounts and perform transactions.

CODING:

```
class Account {
    String accountHolder;
    double balance;

    Account(String accountHolder, double balance) {
        this.accountHolder = accountHolder;
        this.balance = balance;
    }
}
```

```
void deposit(double amount) {  
    balance += amount;  
    System.out.println(accountHolder + " deposited $" + amount);  
}
```

```
void withdraw(double amount) { // Polymorphic method  
    if (balance >= amount) {  
        balance -= amount;  
        System.out.println(accountHolder + " withdrew $" + amount);  
    } else {  
        System.out.println("Insufficient funds for " + accountHolder);  
    }  
}
```

```
void displayBalance() {  
    System.out.println(accountHolder + "'s Balance: $" + balance);  
}  
}
```

```
class SavingsAccount extends Account {  
    double interestRate;
```

```
    SavingsAccount(String accountHolder, double balance, double  
interestRate) {  
        super(accountHolder, balance);  
        this.interestRate = interestRate;  
    }  
}
```

```

@Override
void withdraw(double amount) {
    if (balance - amount >= 100) { // Maintain minimum balance of $100
        super.withdraw(amount);
    } else {
        System.out.println("Cannot withdraw. Minimum balance must be
$100 for " + accountHolder);
    }
}
}

class CheckingAccount extends Account {
    double overdraftLimit;

    CheckingAccount(String accountHolder, double balance, double
overdraftLimit) {
        super(accountHolder, balance);
        this.overdraftLimit = overdraftLimit;
    }

    @Override
    void withdraw(double amount) {
        if (balance + overdraftLimit >= amount) {
            balance -= amount;

            System.out.println(accountHolder + " withdrew $" + amount + "
(Overdraft allowed)");
        } else {

```

```
        System.out.println("Overdraft limit exceeded for " +
accountHolder);
    }
}
}

public class BankingSystem {
    public static void main(String[] args) {
        // Creating an array of Account objects (looping used)
        Account accounts[] = new Account[3];

        accounts[0] = new SavingsAccount("Alice", 500, 2.5);
        accounts[1] = new CheckingAccount("Bob", 300, 200);
        accounts[2] = new SavingsAccount("Charlie", 150, 3.0);

        for (Account acc : accounts) {
            acc.deposit(100);
            acc.withdraw(400);
            acc.displayBalance();
            System.out.println();
        }
    }
}
```

OUTPUT:

```
Alice deposited $100
Alice withdrew $400
Alice's Balance: $200

Bob deposited $100
Bob withdrew $400 (Overdraft allowed)
Bob's Balance: $0

Charlie deposited $100
Cannot withdraw. Minimum balance must be $100 for Charlie
Charlie's Balance: $250
```

4.EMPLOYEE PAYROLL SYSTEM

AIM:

- Parent class: Employee, Child classes: FullTimeEmployee, PartTimeEmployee.
- Polymorphic method to calculate salary based on type.
- Loop through employees to process salaries.

ALGORITHM:

1. Define base class Employee.
2. Create derived classes FullTimeEmployee and PartTimeEmployee.
3. Override calculateSalary() method in subclasses.
4. Initialize an array of employees.
5. Loop through employees and display salaries.

CODING:

```
class Employee {
    String name;
```

```
int id;
```

```
double baseSalary;
```

```
Employee(String name, int id, double baseSalary) {
```

```
    this.name = name;
```

```
    this.id = id;
```

```
    this.baseSalary = baseSalary;
```

```
}
```

```
double calculateSalary() { // Polymorphic method
```

```
    return baseSalary;
```

```
}
```

```
void displaySalary() {
```

```
    System.out.println("Employee ID: " + id + ", Name: " + name + ",  
Salary: $" + calculateSalary());
```

```
}
```

```
}
```

```
class FullTimeEmployee extends Employee {
```

```
    double bonus;
```

```
FullTimeEmployee(String name, int id, double baseSalary, double  
bonus) {
```

```
    super(name, id, baseSalary);
```

```
    this.bonus = bonus;
```

```
}
```



```
@Override
double calculateSalary() {
    return baseSalary + bonus;
}
}
```

```
class PartTimeEmployee extends Employee {
    int hoursWorked;
    double hourlyRate;
```

```
    PartTimeEmployee(String name, int id, double hourlyRate, int
hoursWorked) {
        super(name, id, 0);
        this.hourlyRate = hourlyRate;
        this.hoursWorked = hoursWorked;
    }
```

```
@Override
double calculateSalary() {
    return hourlyRate * hoursWorked;
}
}
```

```
public class EmployeePayroll {
    public static void main(String[] args) {
        // Creating an array of Employee objects (looping used)
        Employee employees[] = new Employee[3];
```

```
employees[0] = new FullTimeEmployee("Alice", 101, 5000, 1000);
employees[1] = new PartTimeEmployee("Bob", 102, 20, 80);
employees[2] = new FullTimeEmployee("Charlie", 103, 4500, 800);

// Loop through and display employee salaries
for (Employee emp : employees) {
    emp.displaySalary();
}
}
```

OUTPUT:

```
Employee ID: 101, Name: Alice, Salary: $6000.0
Employee ID: 102, Name: Bob, Salary: $1600.0
Employee ID: 103, Name: Charlie, Salary: $5300.0
```

5.VEHICLE RENTAL SYSTEM

AIM:

- Base class: Vehicle, Derived classes: Car, Bike, Truck.
- Polymorphism for rental price calculation.
- Use loops for booking and returning vehicles.

ALGORITHM:

1. Define base class Vehicle.
2. Create derived classes Car, Bike, and Truck.
3. Override calculateRental() method in subclasses.
4. Initialize an array of vehicles.

5. Loop through vehicles and display rental details.

CODING:

```
Vehicle(String model, double rentalPrice) {
    this.model = model;
    this.rentalPrice = rentalPrice;
}

double calculateRental(int days) { // Polymorphic method
    return rentalPrice * days;
}

void displayDetails(int days) {
    System.out.println("Vehicle: " + model + ", Rental
Cost for " + days + " days: $" + calculateRental(days));
}

}

@Override
double calculateRental(int days) {
    return super.calculateRental(days) * 0.9; // 10%
discount for bikes
}

} Truck(String model, double rentalPrice, double
weightLimit) {
    super(model, rentalPrice);
    this.weightLimit = weightLimit;
}

@Override
double calculateRental(int days) {
    return super.calculateRental(days) + (weightLimit * 2
* days);
}
```

```

}

}

{ public static void main(String[] args) {

objects (looping used) Vehicle vehicles[] = new Vehicle[3];

    vehicles[0] = new Car("Sedan", 50);
    vehicles[1] = new Bike("Sports Bike", 30);
    vehicles[2] = new Truck("Heavy Truck", 80, 1000);

    int rentalDays = 5;

    for (Vehicle v : vehicles) {
        v.displayDetails(rentalDays);
    }
}

}

```

OUTPUT:

```

Vehicle: Sedan, Rental Cost for 5 days: $250.0
Vehicle: Sports Bike, Rental Cost for 5 days: $135.0
Vehicle: Heavy Truck, Rental Cost for 5 days: $10050.0

```

6.HOSPITAL MANAGEMENT SYSTEM:

AIM:

- Parent class: Person, Derived classes: Doctor, Patient, Staff.
- Override methods for role-based actions.
- Loop to manage patient check-in and check-out.

ALGORITHM:

1. Define base class Person.
2. Create derived classes Doctor, Patient, and Staff.
3. Override displayDetails() method in subclasses.
4. Initialize an array of hospital personnel.
5. Loop through the array and display details.

CODING:

```
class Person {  
    String name;  
    int id;  
  
    public Person(String name, int id) {  
        this.name = name;  
        this.id = id;  
    }  
  
    public void displayDetails() {  
        System.out.println("ID: " + id + ", Name: " + name);  
    }  
}  
  
class Doctor extends Person {  
    String specialty;  
  
    public Doctor(String name, int id, String specialty) {
```

```
    super(name, id);  
    this.specialty = specialty;  
}
```

```
@Override
```

```
public void displayDetails() {  
    System.out.println("Doctor - ID: " + id + ", Name: " + name + ",  
Specialty: " + specialty);  
}  
}
```

```
class Patient extends Person {
```

```
    String disease;
```

```
    public Patient(String name, int id, String disease) {  
        super(name, id);  
        this.disease = disease;  
    }
```

```
@Override
```

```
public void displayDetails() {  
    System.out.println("Patient - ID: " + id + ", Name: " + name + ",  
Disease: " + disease);  
}
```

```
}
```

```
class Staff extends Person {
```

```
    String role;
```

```
    public Staff(String name, int id, String role) {
```

```
        super(name, id);
```

```
        this.role = role;
```

```
    }
```

```
    @Override
```

```
    public void displayDetails() {
```

```
        System.out.println("Staff - ID: " + id + ", Name: " + name + ", Role: " + role);
```

```
    }
```

```
}
```

```
public class HospitalManagement {
```

```
    public static void main(String[] args) {
```

```
        Person[] people = {
```

```
            new Doctor("Dr. Smith", 101, "Cardiology"),
```

```
            new Patient("Alice", 201, "Flu"),
```

```
            new Staff("John", 301, "Nurse"),
```

```
            new Doctor("Dr. Brown", 102, "Neurology"),
```

```
        new Patient("Bob", 202, "Fracture")
    };

    for (Person p : people) {
        p.displayDetails();
    }
}
}
```

OUTPUT:

```
Doctor - ID: 101, Name: Dr. Smith, Specialty: Cardiology
Patient - ID: 201, Name: Alice, Disease: Flu
Staff - ID: 301, Name: John, Role: Nurse
Doctor - ID: 102, Name: Dr. Brown, Specialty: Neurology
Patient - ID: 202, Name: Bob, Disease: Fracture
```

7.SHOPPING CART SYSTEM:

AIM:

- Base class: Product, Derived classes: Electronics, Clothing.
- Polymorphism for discount calculation.
- Loop through cart items to display the bill.

ALGORITHM:

1. Define base class Product.
2. Create derived classes Electronics and Clothing.
3. Override calculateDiscount() method in subclasses.
4. Initialize an array of cart items.
5. Loop through the cart and display product details.

CODING:

```
class Product {  
    String name;  
    double price;  
  
    public Product(String name, double price) {  
        this.name = name;  
        this.price = price;  
    }  
  
    public double calculateDiscount() {  
        return 0;  
    }  
  
    public void displayProduct() {  
        System.out.println(name + " - Price: $" + price + ", Discount: $" +  
calculateDiscount());  
    }  
}  
  
class Electronics extends Product {  
    public Electronics(String name, double price) {  
        super(name, price);  
    }  
}
```

```
@Override  
  
public double calculateDiscount() {  
    return price * 0.10; // 10% discount  
}  
}  
  
class Clothing extends Product {  
    public Clothing(String name, double price) {  
        super(name, price);  
    }  
}
```

```
@Override  
  
public double calculateDiscount() {  
    return price * 0.20; // 20% discount  
}  
}  
  
public class ShoppingCart {  
    public static void main(String[] args) {  
        Product[] cart = {  
            new Electronics("Laptop", 1000),  
            new Clothing("T-Shirt", 50),  
            new Electronics("Smartphone", 800),  
            new Clothing("Jeans", 70)  
        }  
    }  
}
```

```
};  
  
System.out.println("Shopping Cart:");  
  
for (Product p : cart) {  
    p.displayProduct();  
}  
  
}  
  
}
```

OUTPUT:

```
Shopping Cart:  
Laptop - Price: $1000.0, Discount: $100.0  
T-Shirt - Price: $50.0, Discount: $10.0  
Smartphone - Price: $800.0, Discount: $80.0  
Jeans - Price: $70.0, Discount: $14.0
```

8.ONLINE EXAMINATION SYSTEM

AIM:

- Base class: Question, Derived classes: MCQ, Descriptive Question.
- Override method for evaluating answers.
- Loop to present questions and take user input.

ALGORITHM:

1. Define base class Question.
2. Create derived classes MCQ and DescriptiveQuestion.
3. Override askQuestion() and checkAnswer() methods.
4. Initialize an array of questions.

5. Loop through questions, get user input, and evaluate answers.

CODING:

```
import java.util.Scanner;

class Question {

    String questionText;

    public Question(String questionText) {

        this.questionText = questionText;

    }

    public void askQuestion() {

        System.out.println(questionText);

    }

    public boolean checkAnswer(String answer) {

        return false;

    }

}

class MCQ extends Question {

    String[] options;

    String correctAnswer;
```

```
    public MCQ(String questionText, String[] options, String
correctAnswer) {

        super(questionText);

        this.options = options;

        this.correctAnswer = correctAnswer;

    }
```

@Override

```
    public void askQuestion() {

        System.out.println(questionText);

        for (String option : options) {

            System.out.println(option);

        }

    }
```

@Override

```
    public boolean checkAnswer(String answer) {

        return answer.equalsIgnoreCase(correctAnswer);

    }

}
```

```
class DescriptiveQuestion extends Question {

    String correctAnswer;
```

```
public DescriptiveQuestion(String questionText, String correctAnswer)
{
    super(questionText);
    this.correctAnswer = correctAnswer;
}
```

@Override

```
public boolean checkAnswer(String answer) {
    return answer.equalsIgnoreCase(correctAnswer);
}
}
```

```
public class OnlineExam {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Question[] questions = {
            new MCQ("What is the capital of France?", new String[]{"A) Paris", "B) London", "C) Berlin", "D) Rome"}, "A"),
            new DescriptiveQuestion("Who discovered gravity?", "Newton"),
            new MCQ("Which language is used for Android development?", new String[]{"A) Java", "B) C++", "C) Python", "D) Ruby"}, "A")
        };
        int score = 0;
        for (Question q : questions) {
```

```
        q.askQuestion();

        System.out.print("Your Answer: ");

        String userAnswer = scanner.nextLine();

        if (q.checkAnswer(userAnswer)) {

            System.out.println("Correct!\n");

            score++;

        } else {

            System.out.println("Wrong answer.\n");

        }

    }

    System.out.println("Your total score: " + score + "/" +
questions.length);

    scanner.close();

}

}
```

OUTPUT:

Q1: Capital of France?

Your Answer: A

Correct!

Q2: Who discovered gravity?

Your Answer: Newton

Correct!

Q3: Language for Android development?

Your Answer: C

Wrong.

Score: 2/3

9.ZOO MANAGEMENT SYSTEM

AIM:

- Base class: Animal, Derived classes: Mammal, Bird, Reptile.
- Use polymorphism for different sounds and behaviors.
- Loop to display animal details.

ALGORITHM:

1. Define base class Animal.
2. Create derived classes Mammal, Bird, and Reptile.
3. Override makeSound() method in subclasses.
4. Initialize an array of animals.
5. Loop through the array and display animal sounds.

CODING:


```
class Animal {  
    String name;  
  
    public Animal(String name) {  
        this.name = name;  
    }  
  
    public void makeSound() {  
        System.out.println(name + " makes a sound.");  
    }  
}  
  
class Mammal extends Animal {  
    public Mammal(String name) {  
        super(name);  
    }  
  
    @Override  
    public void makeSound() {  
        System.out.println(name + " growls.");  
    }  
}  
  
class Bird extends Animal {  
    public Bird(String name) {
```

```
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(name + " chirps.");
    }
}

class Reptile extends Animal {
    public Reptile(String name) {
        super(name);
    }

    @Override
    public void makeSound() {
        System.out.println(name + " hisses.");
    }
}

public class ZooManagement {
    public static void main(String[] args) {
        Animal[] zoo = {
            new Mammal("Lion"),
            new Bird("Parrot"),
```

```
        new Reptile("Snake"),  
        new Mammal("Tiger")  
    };  
    for (Animal a : zoo) {  
        a.makeSound();  
    }  
}  
}
```

OUTPUT:

```
Lion growls.  
Parrot chirps.  
Snake hisses.  
Tiger growls.
```

10. GAME LEADERBOARD SYSTEM

AIM:

- Base class: Player, Derived classes: CasualPlayer, ProPlayer.
- Override method for score calculation.
- Loop to rank players based on scores.

ALGORITHM:

1. Define base class Player.
2. Create derived classes CasualPlayer and ProPlayer.
3. Override displayScore() method in subclasses.
4. Initialize an array of players.

5.

Loop through players and display scores.

CODING:

```
class Player {  
    String name;  
    int score;  
  
    public Player(String name, int score) {  
        this.name = name;  
        this.score = score;  
    }  
  
    public void displayScore() {  
        System.out.println(name + " scored " + score + " points.");  
    }  
}  
  
class CasualPlayer extends Player {  
    public CasualPlayer(String name, int score) {  
        super(name, score);  
    }  
  
    @Override  
    public void displayScore() {
```

```
        System.out.println(name + " (Casual) scored " + score + " points.");
    }
}

class ProPlayer extends Player {

    public ProPlayer(String name, int score) {

        super(name, score);
    }

    @Override

    public void displayScore() {

        System.out.println(name + " (Pro) scored " + score + " points!");
    }
}

public class GameLeaderboard {

    public static void main(String[] args) {

        Player[] leaderboard = {

            new CasualPlayer("Alice", 150),

            new ProPlayer("Bob", 300),

            new CasualPlayer("Charlie", 200),

            new ProPlayer("Dave", 500)

        };

        for (Player p : leaderboard) {

            p.displayScore();
        }
    }
}
```

```
    }  
  }  
}
```

OUTPUT:

```
Alice (Casual) scored 150 points.  
Bob (Pro) scored 300 points!  
Charlie (Casual) scored 200 points.  
Dave (Pro) scored 500 points!
```