



SCHOOL OF
COMPUTING

LAB RECORD

23CSE111- Object Oriented Programming

Submitted by

CH.SC.U4CSE24039-SAHIL PAREEK

BACHELOR OF 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.U4CSE24039 – SAHIL PAREEK** 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 08/ 04/ 2025

LAB IN CHARGE:_____

S.NO	TITLE
1.	RAPTOR(3 PROGRAMS)
2.	USE CASE DIAGRAMS(6 DIAGRAMS)
3.	CLASS DIAGRAMS(6 DIAGRAMS)
4.	STATE DIAGRAMS(6 DIAGRAMS)
5.	SEQUENCE DIAGRAMS(6 DIAGRAMS)
6.	OBJECT DIAGRAMS(6 DIAGRAMS)
7.	BASIC JAVA PROGRAMS(10 PROGRAMS)
8.	INHERITANCE(8 PROGRAMS)
9.	POLYMORPHISM(8 PROGRAMS)
10.	ABSTRACTION(8 PROGRAMS)
11.	ENCAPSULATION(4 PROGRAMS)
12.	BUILT-IN PACKAGES(8 PROGRAMS)
13.	USER-DEFINED PACKAGES(2 PROGRAMS)
14.	FILE HANDLING AND EXCEPTION HANDLING(COMBINED) (6 PROGRAMS)

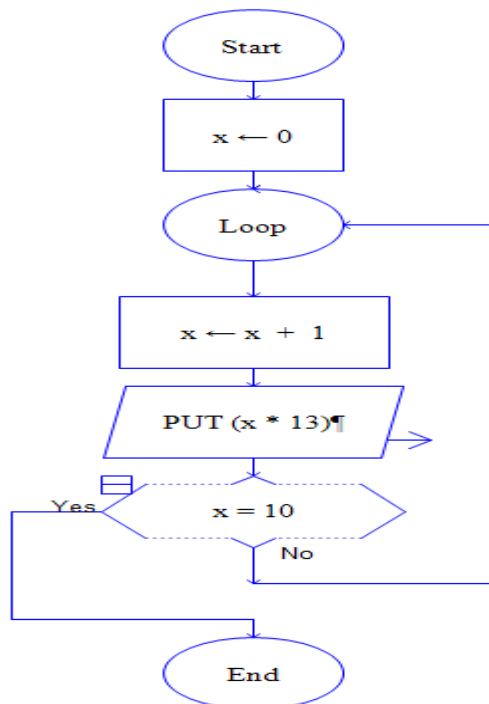
RAPTOR

1) Multiplying a Number Till 10:

Aim: To generate and display the multiplication table of a given number up to 10.

Algorithm:

1. Input a number from the user.
2. Loop from 1 to 10.
 - For each iteration, multiply the input number by the loop index.
 - Print the result in a formatted manner.
3. End the program.

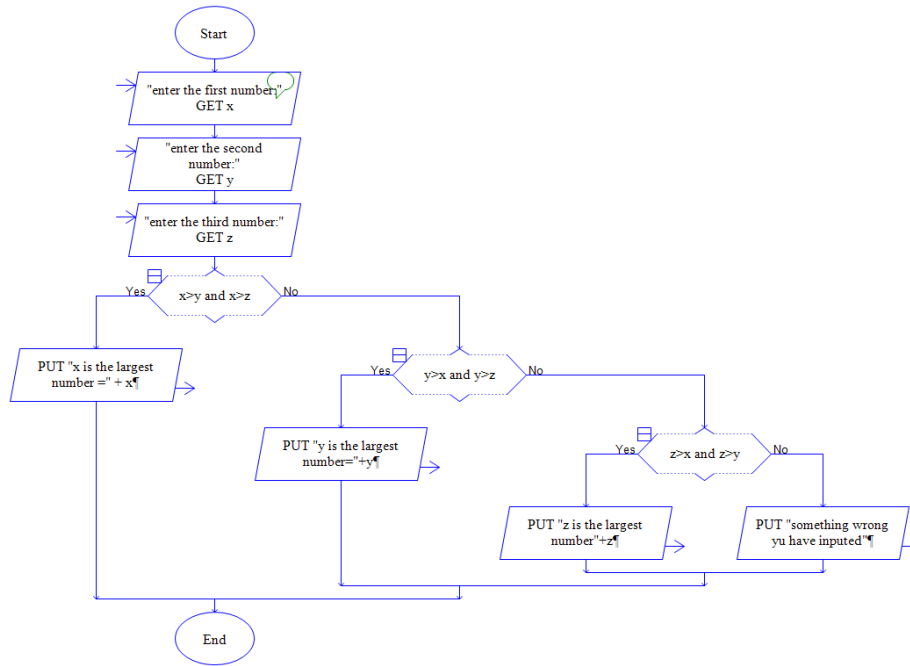


2) Checking Which is the Largest Number from Three:

Aim: To determine and display the largest number among three user-provided numbers.

Algorithm:

- 1. Input three numbers from the user.**
- 2. Initialize a variable to hold the largest number.**
- 3. Compare the first number with the second and third numbers.**
 - If the first number is greater, update the largest number.**
 - If the second number is greater, update the largest number.**
 - If the third number is greater, update the largest number.**
- 4. Print the largest number.**
- 5. End the program.**

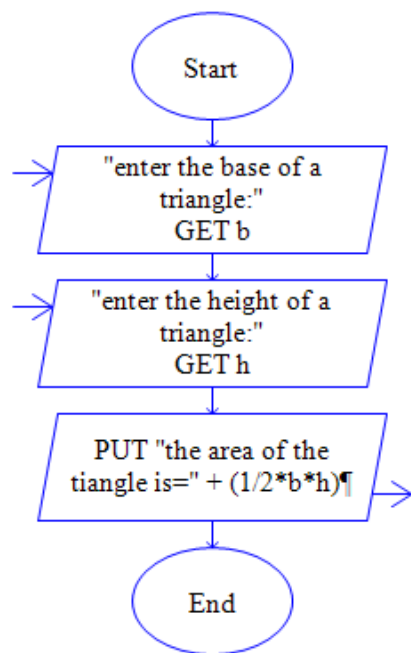


3) Calculating the Area of a Triangle

Aim: To calculate and display the area of a triangle using the base and height provided by the user.

Algorithm:

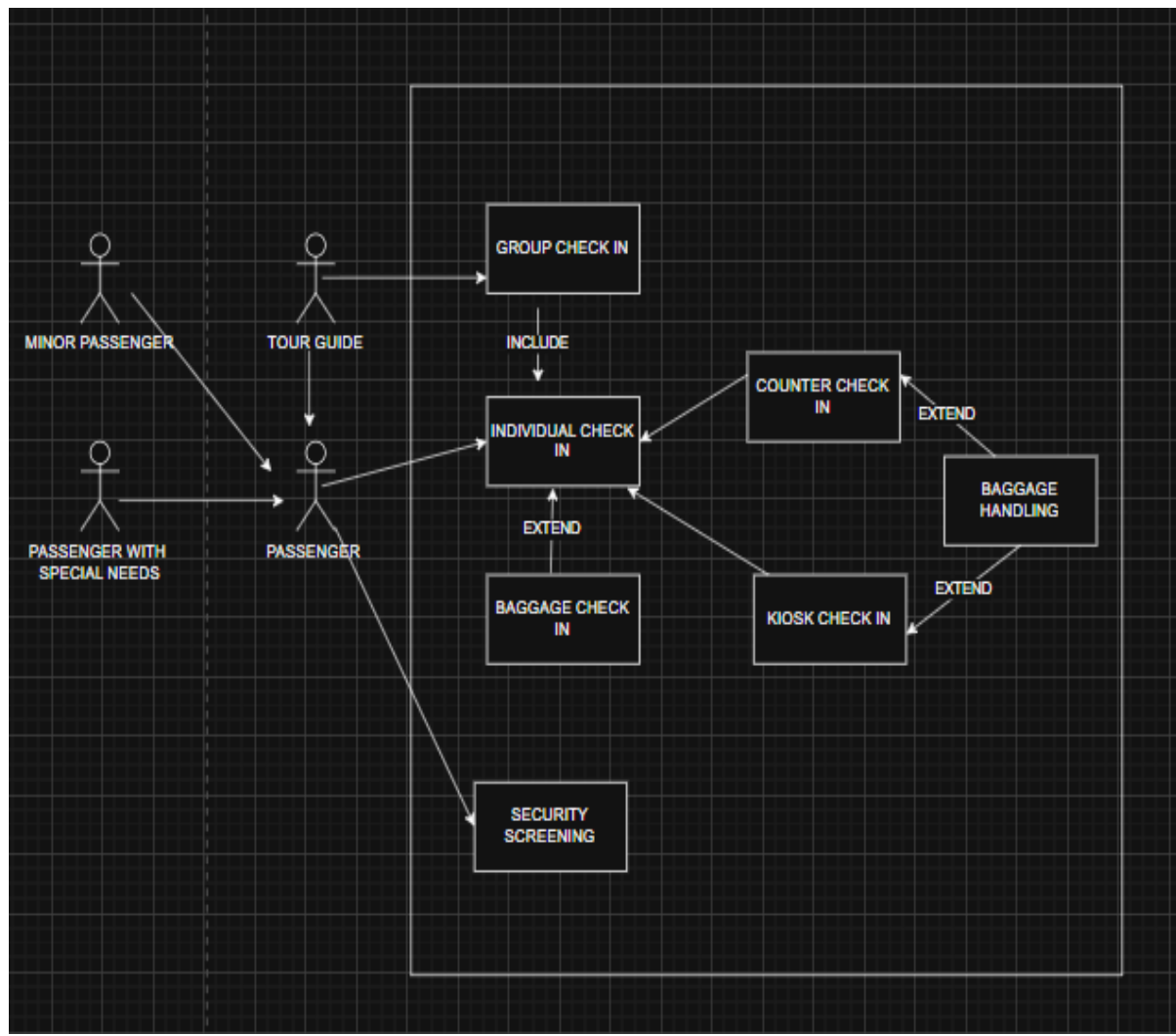
- 1. Input the base and height of the triangle from the user.**
- 2. Use the formula for the area of a triangle: $\text{Area} = (\text{base} * \text{height}) / 2$.**
- 3. Calculate the area using the input values.**
- 4. Print the calculated area.**
- 5. End the program.**



USE CASE DIAGRAMS(UML)

1. Airport Check-In:

- Aim: To illustrate the process of passengers checking in for flights, including interactions with airline staff and kiosks.
- Algorithm:
 1. Identify actors: passengers, airline staff, check-in kiosks.
 2. Define use cases: check-in, baggage drop, seat selection.
 3. Establish relationships between actors and use cases.
 4. Create the diagram with actors and use cases.
 5. Review for clarity.

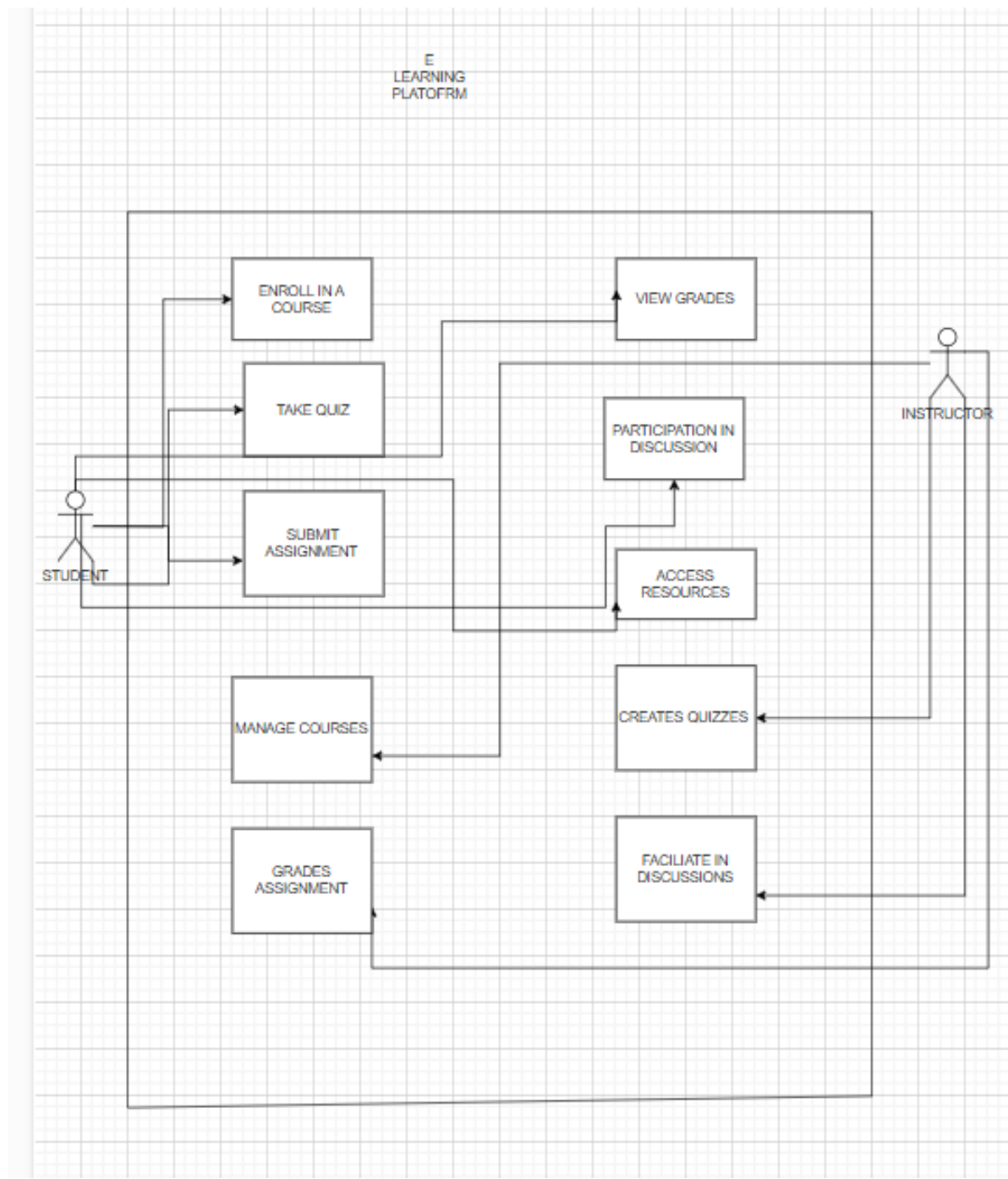


2. E-Learning Platform:

- Aim: To represent user interactions with the platform, including students, instructors, and administrators.
- Algorithm:
 1. Identify actors: students, instructors, administrators.
 2. Define use cases: enroll in courses, submit assignments, grade submissions.
 3. Establish relationships between actors and use cases.

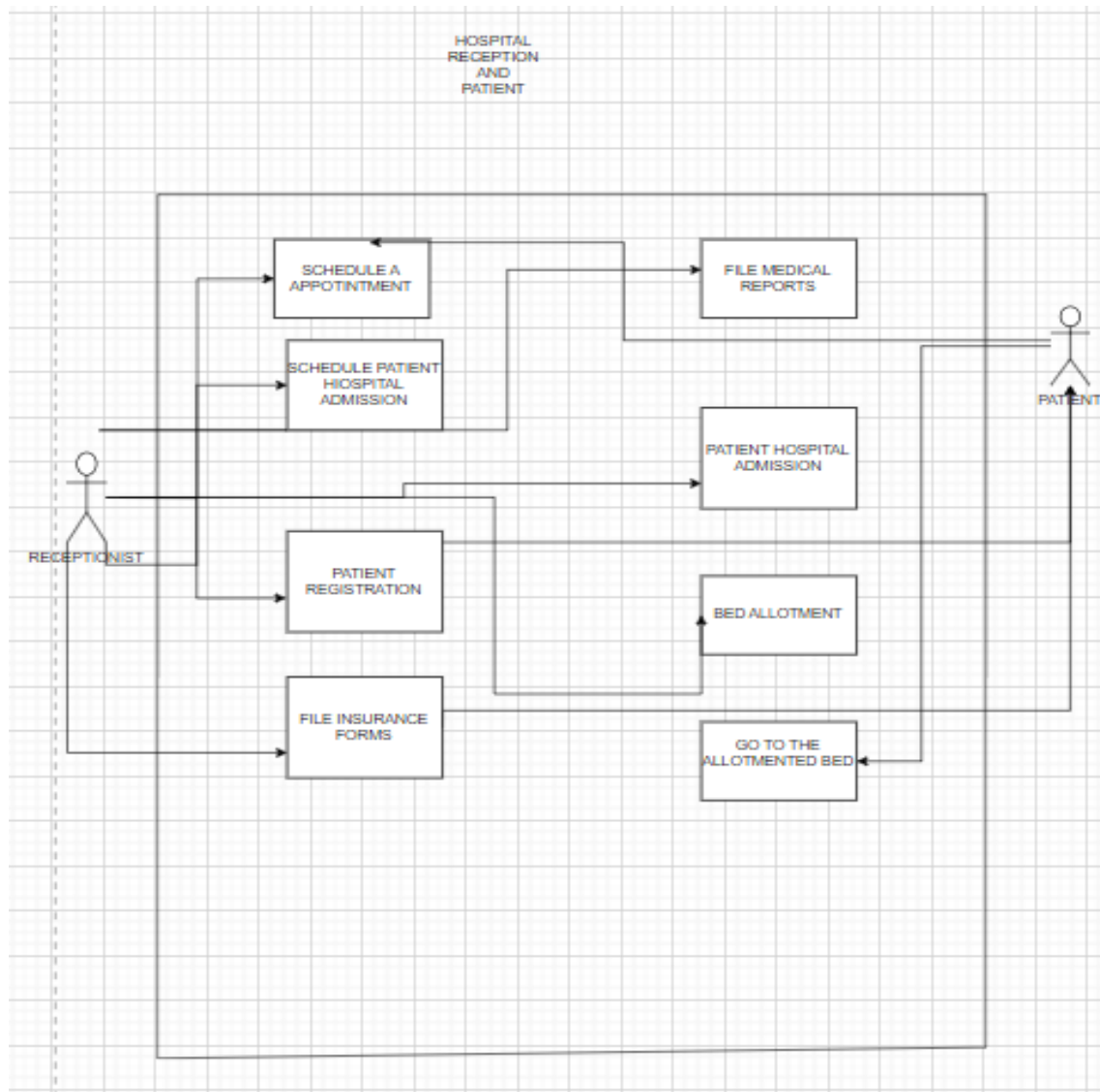
4. Create the diagram with actors and use cases.

5. Review for completeness.



3. Hospital Receptionist and Patient:

- **Aim: To depict the interactions between patients and hospital receptionists during the appointment process.**
- **Algorithm:**
 1. **Identify actors: patients, receptionists.**
 2. **Define use cases: schedule appointment, check-in, update patient information.**
 3. **Establish relationships between actors and use cases.**
 4. **Create the diagram with actors and use cases.**
 5. **Review for accuracy.**



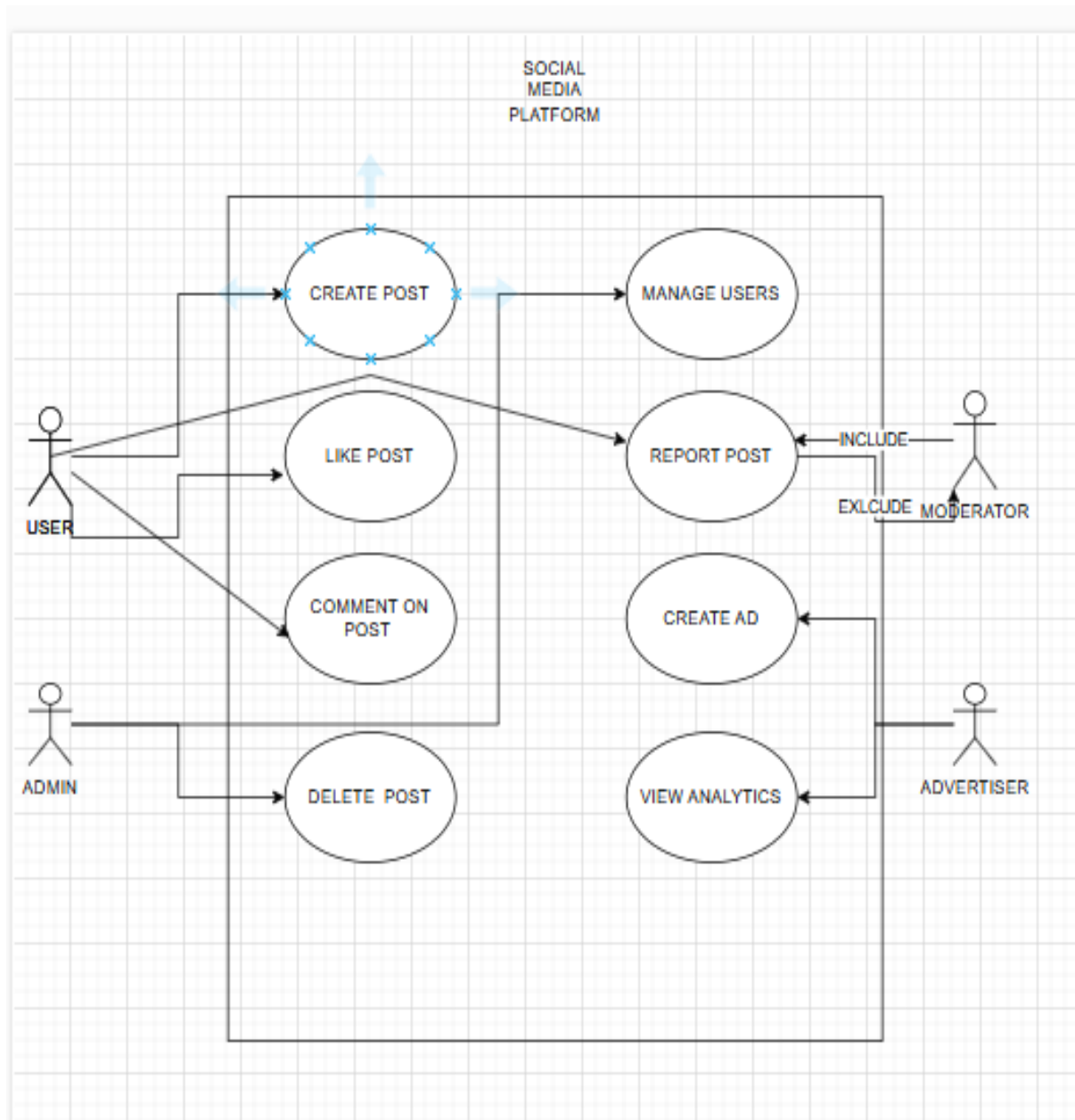
4. Social Media Platform:

- Aim: To illustrate user interactions on a social media platform, including posting content and managing profiles.
- Algorithm:
 1. Identify actors: users, administrators.
 2. Define use cases: create post, like post, follow user.

3. Establish relationships between actors and use cases.

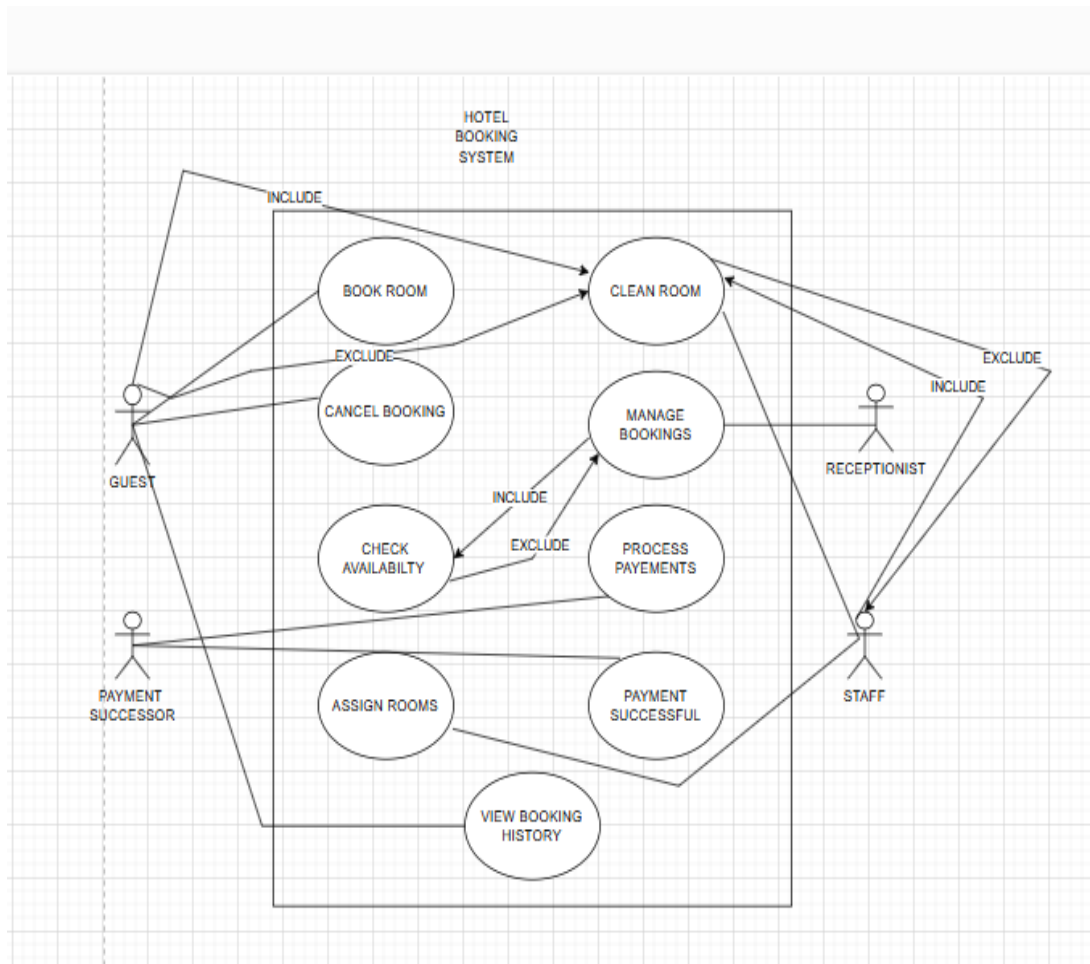
4. Create the diagram with actors and use cases.

5. Review for clarity.



5. Hotel Booking System:

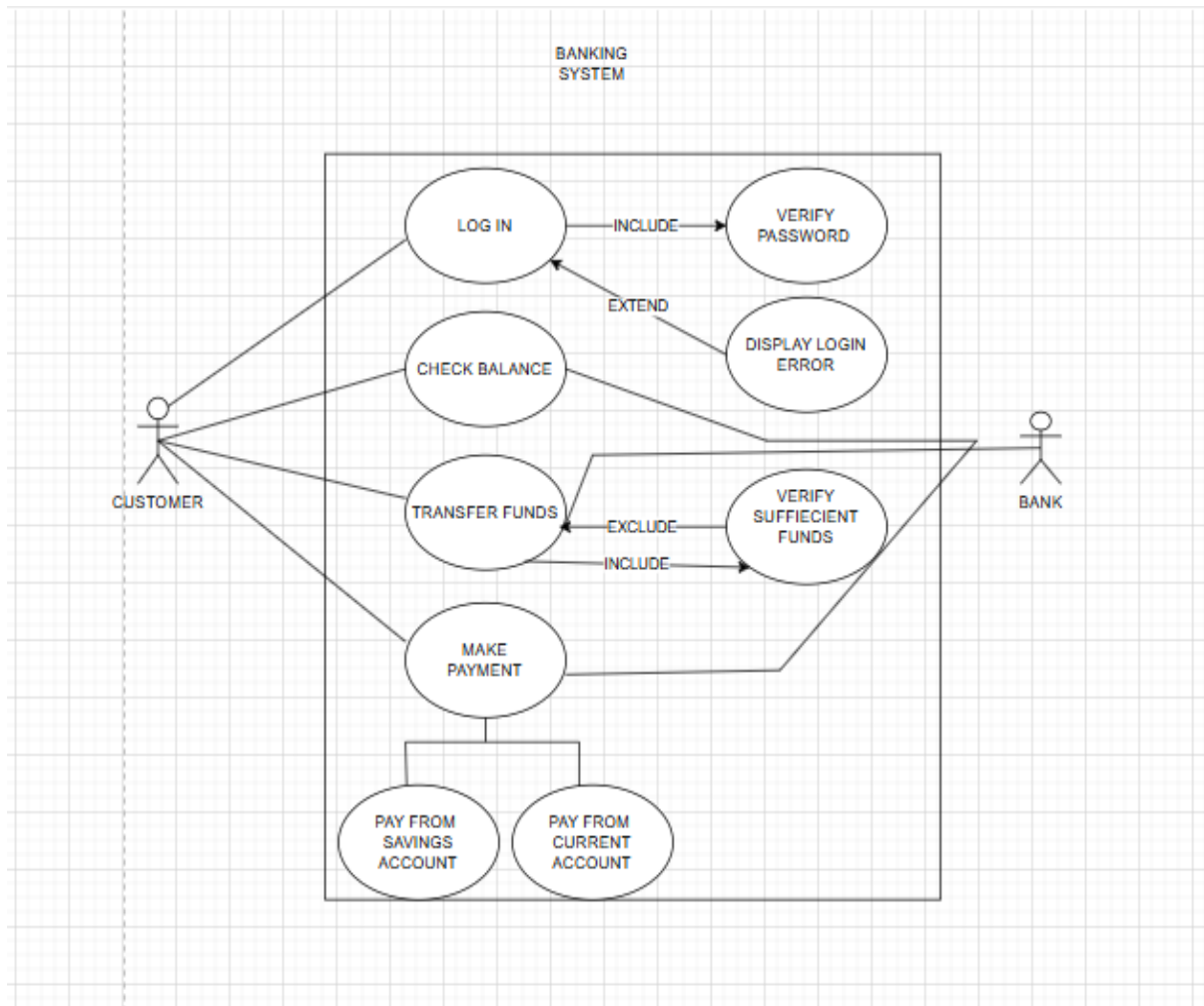
- **Aim: To represent the interactions between customers and the hotel booking system for reservations.**
- **Algorithm:**
 1. **Identify actors: customers, hotel staff.**
 2. **Define use cases: search for hotels, make a reservation, cancel a booking.**
 3. **Establish relationships between actors and use cases.**
 4. **Create the diagram with actors and use cases.**
 5. **Review for completeness.**



6. Banking System:

- Aim: To depict user interactions with a banking system, including account management and transactions.
- Algorithm:
 1. Identify actors: customers, bank staff.
 2. Define use cases: deposit funds, withdraw funds, check balance.
 3. Establish relationships between actors and use cases.
 4. Create the diagram with actors and use cases.

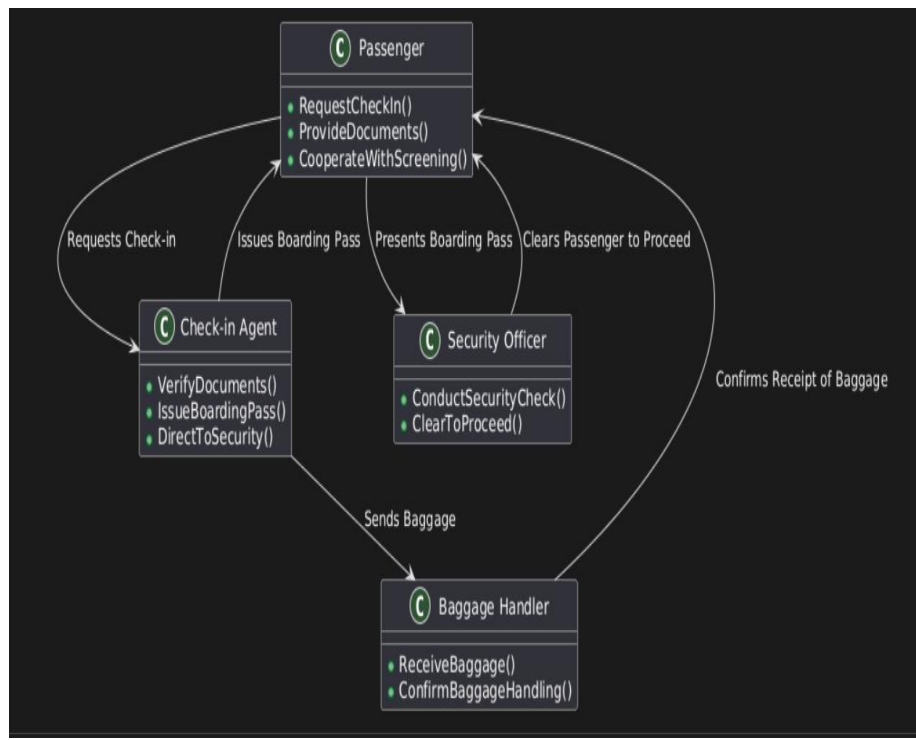
5. Review for accuracy.



CLASS DIAGRAMS(UML)

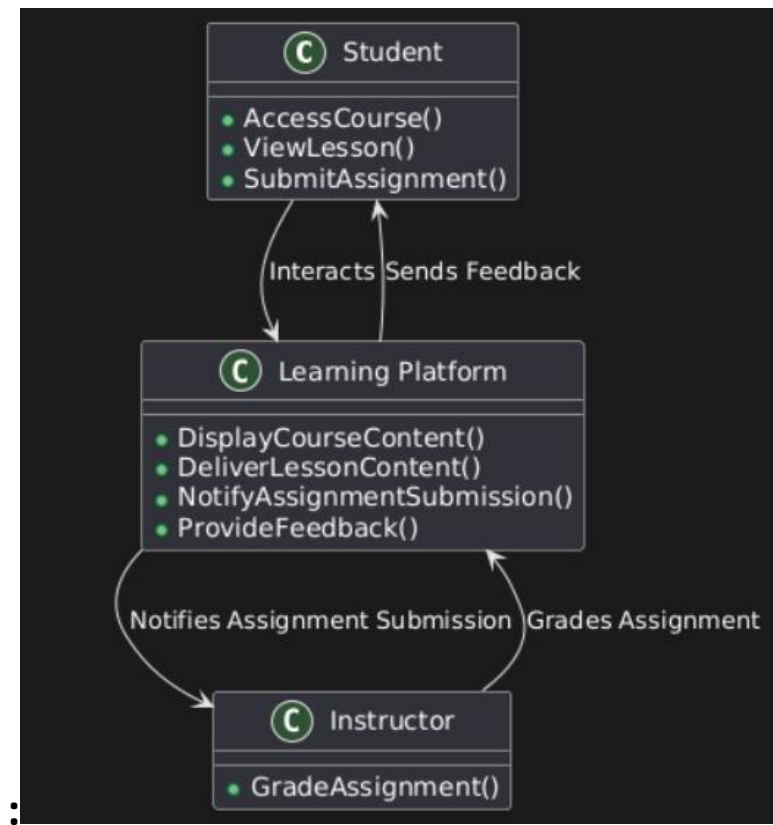
1. Airport Check-In:

- Aim: To represent the structure of the check-in system, including classes for passengers, flights, and check-in kiosks.
- Algorithm:
 1. Identify key classes: Passenger, Flight, CheckInKiosk.
 2. Define attributes and methods for each class.
 3. Determine relationships (e.g., Passenger has a Flight).
 4. Create the diagram with classes and their relationships.
 5. Validate for accuracy.



2. E-Learning Platform:

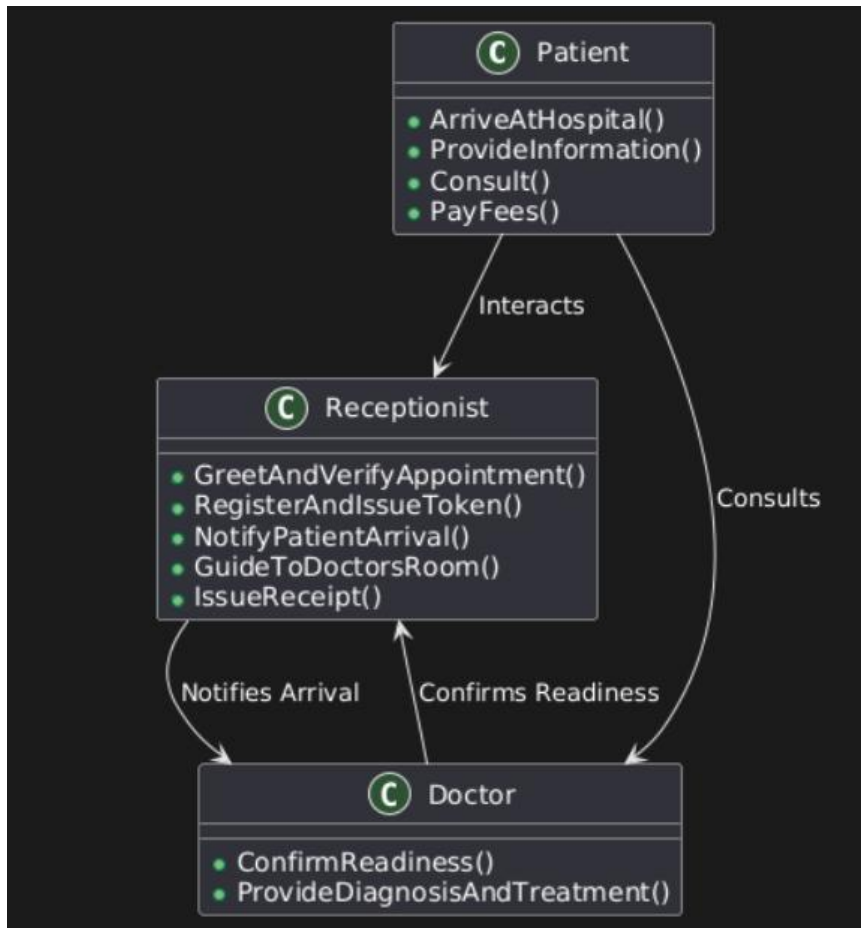
- Aim: To illustrate the structure of the e-learning system, including classes for users, courses, and assignments.
- Algorithm:
 1. Identify key classes: User, Course, Assignment.
 2. Define attributes and methods for each class.
 3. Determine relationships (e.g., User enrolls in Course).
 4. Create the diagram with classes and their relationships.
 5. Validate for completeness.



3. Hospital Receptionist and Patient:

- Aim: To depict the structure of the hospital management system, including classes for patients, appointments, and receptionists.
- Algorithm:
 1. Identify key classes: Patient, Appointment, Receptionist.
 2. Define attributes and methods for each class.
 3. Determine relationships (e.g., Patient has an Appointment).
 4. Create the diagram with classes and their relationships.

5. Validate for accuracy.

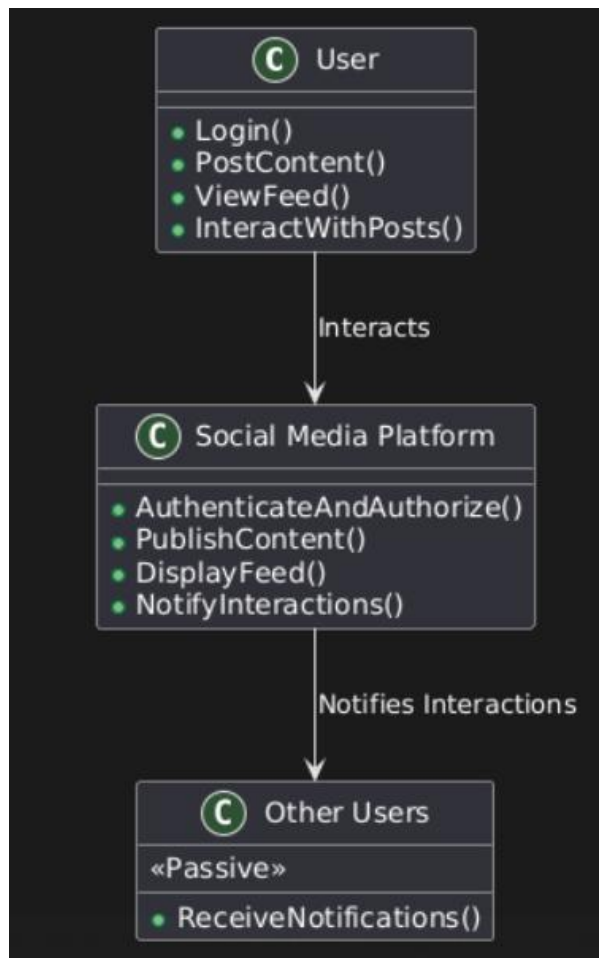


4. Social Media Platform:

- Aim: To represent the structure of the social media system, including classes for users, posts, and comments.
- Algorithm:
 1. Identify key classes: User, Post, Comment.
 2. Define attributes and methods for each class.
 3. Determine relationships (e.g., User creates Post).

4. Create the diagram with classes and their relationships.

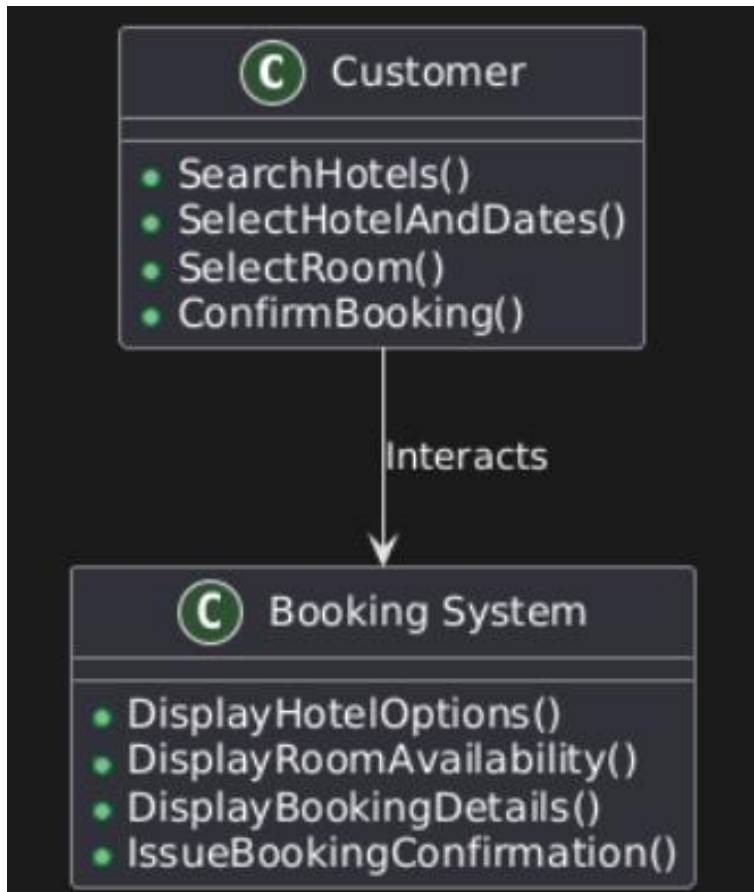
5. Validate for completeness.



5. Hotel Booking System:

- Aim: To illustrate the structure of the hotel booking system, including classes for customers, reservations, and rooms.
- Algorithm:
 1. Identify key classes: Customer, Reservation, Room.
 2. Define attributes and methods for each class.

3. Determine relationships (e.g., Customer makes Reservation).
4. Create the diagram with classes and their relationships.
5. Validate for accuracy.



6. Banking System:

- Aim: To depict the structure of the banking system, including classes for customers, accounts, and transactions.
- Algorithm:
 1. Identify key classes: Customer, Account, Transaction.
 2. Define attributes and methods for each class.

3. Determine relationships (e.g., Customer owns Account).
4. Create the diagram with classes and their relationships.
5. Validate for completeness.

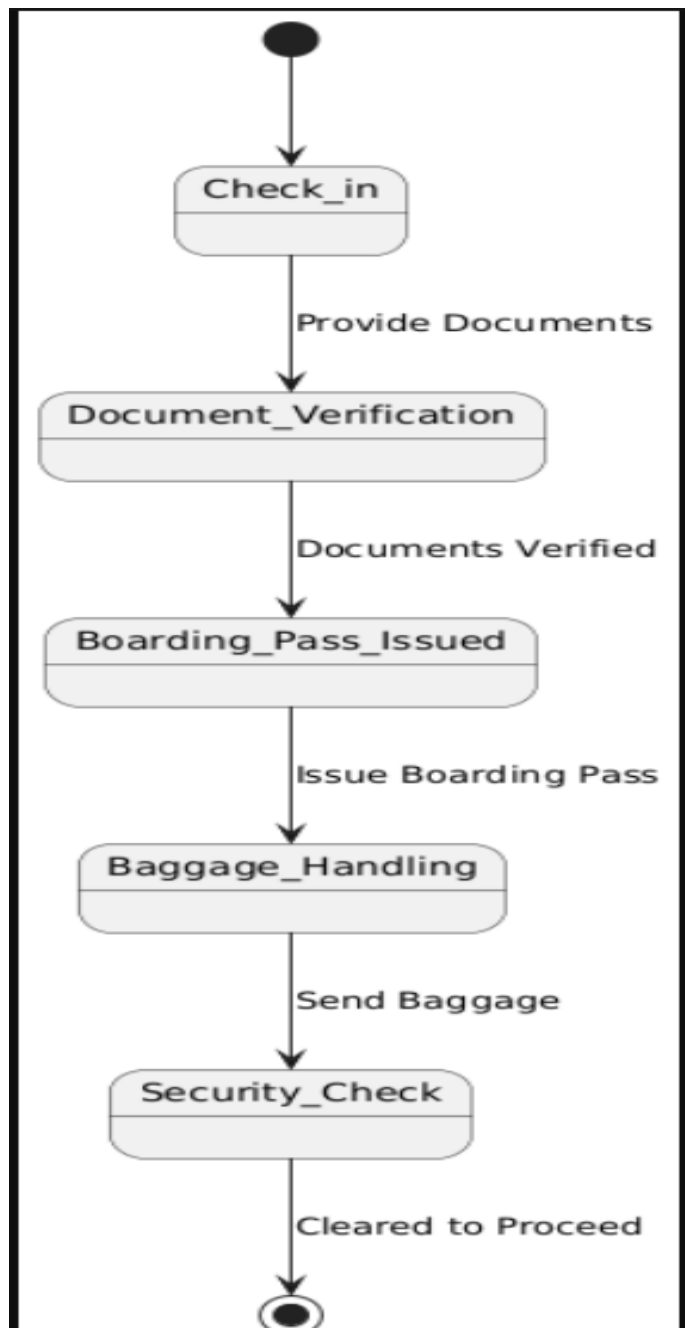


STATE DIAGRAMS(UML)

1. Airport Check-In

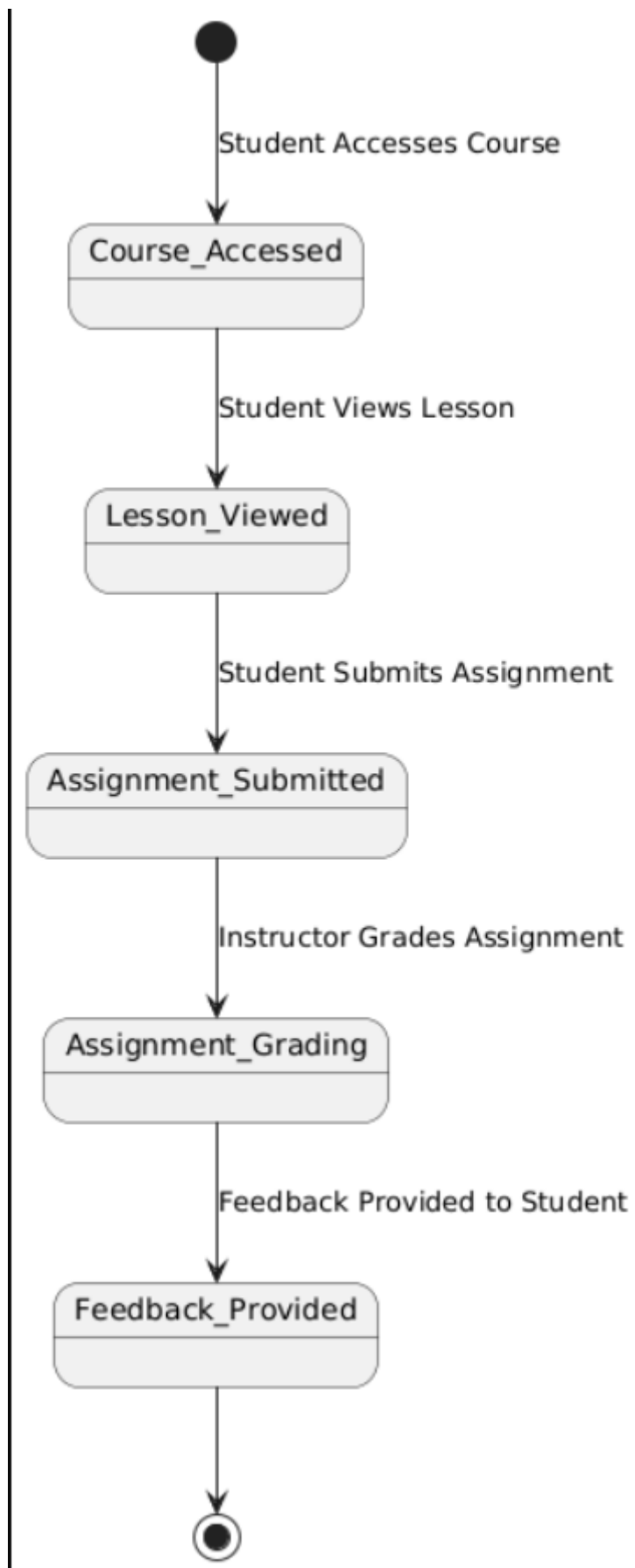
- Aim: To represent the different states of a passenger during the check-in process at the airport.
- Algorithm:

1. Identify the states: At Airport, Checked In, Baggage Dropped, Security Cleared, Boarded.
2. Define events that cause transitions between states (e.g., Check In, Drop Baggage, Pass Security, Board Flight).
3. Create the diagram with states represented as circles or rounded rectangles.
4. Draw arrows to indicate transitions between states, labeling them with the events that trigger the transitions.
5. Validate for completeness and accuracy.



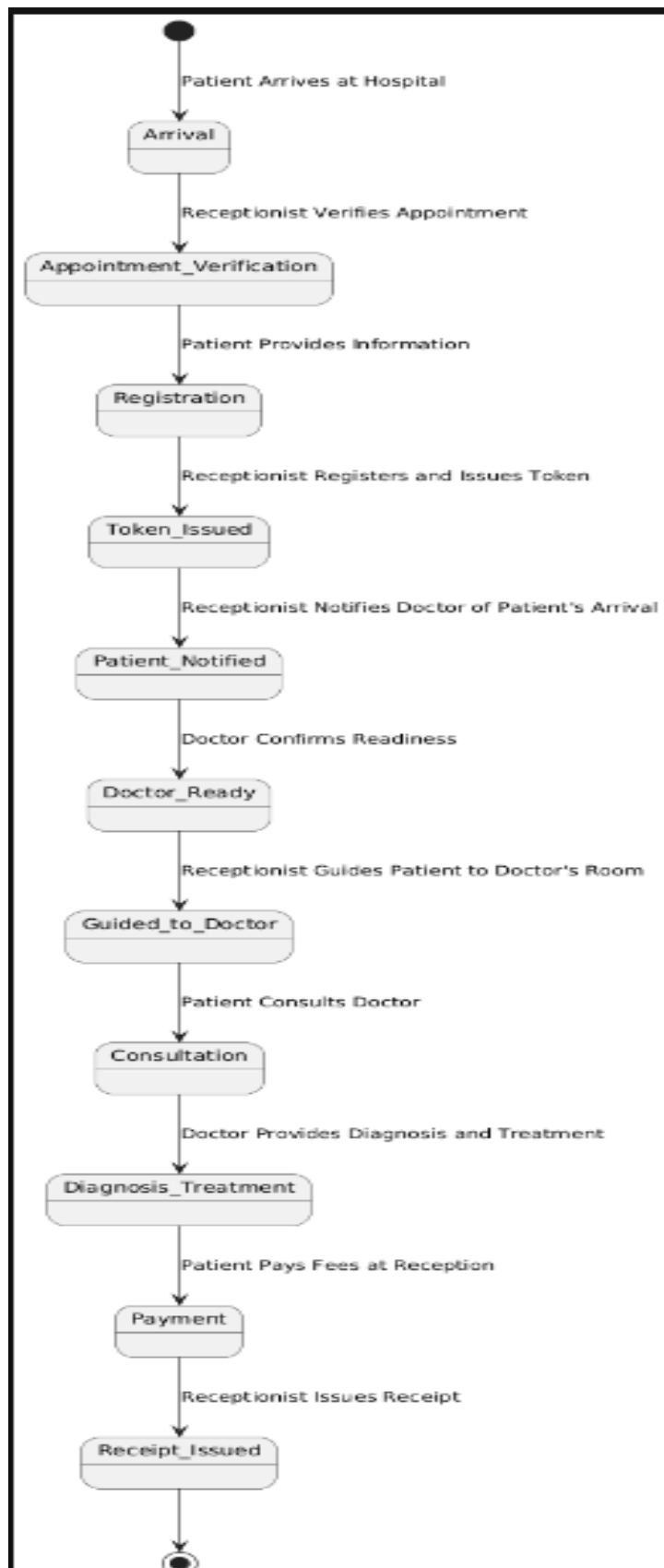
2. E-Learning Platform

- **Aim: To illustrate the states of a course enrollment process in an e-learning platform.**
- **Algorithm:**
 1. **Identify the states: Not Enrolled, Enrolled, In Progress, Completed, Graded.**
 2. **Define events that cause transitions (e.g., Enroll, Start Course, Submit Assignment, Receive Grade).**
 3. **Create the diagram with states represented as circles or rounded rectangles.**
 4. **Draw arrows to indicate transitions between states, labeling them with the events that trigger the transitions.**
 5. **Validate for clarity and completeness.**



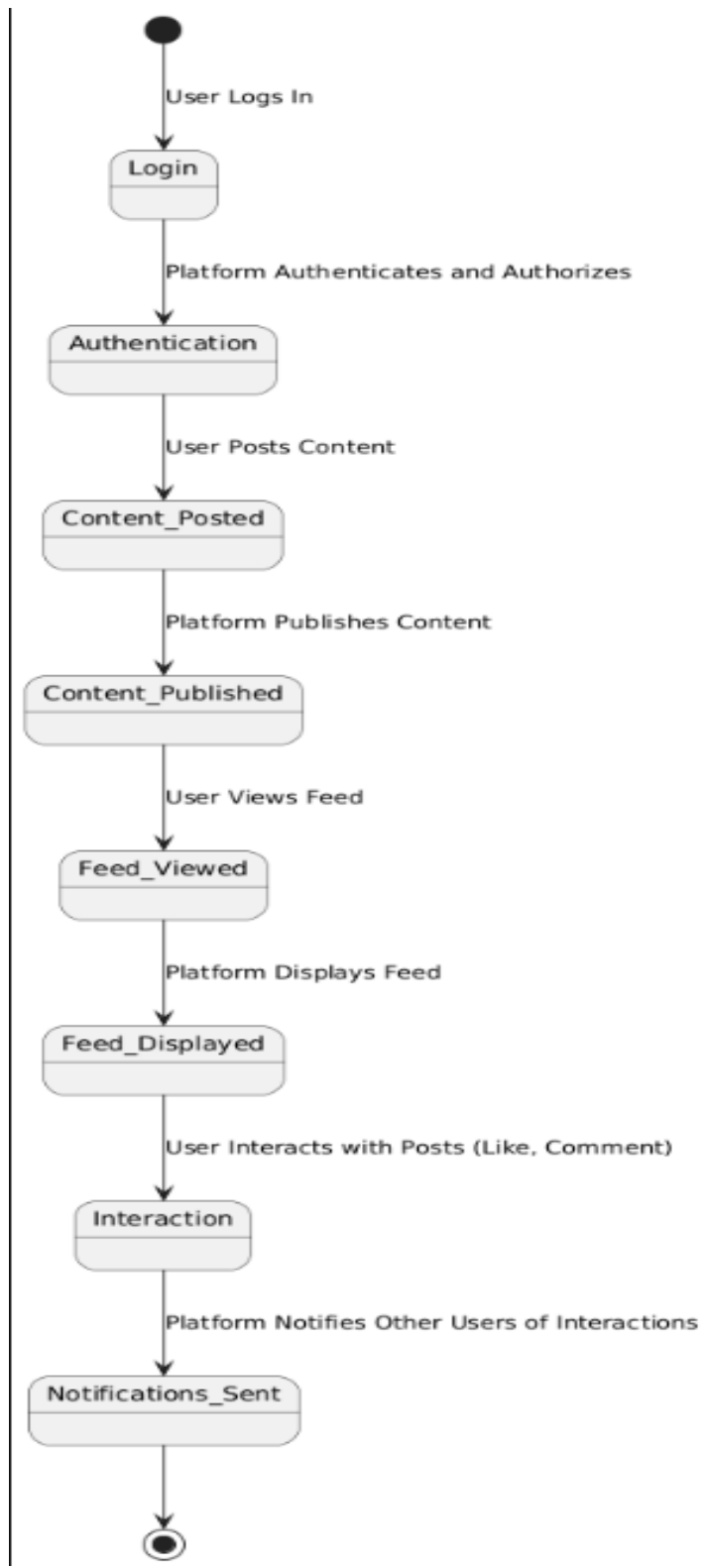
3. Hospital Receptionist and Patient

- **Aim: To depict the states of a patient during the appointment process at a hospital.**
- **Algorithm:**
 1. **Identify the states: Registered, Checked In, In Consultation, Completed Consultation, Discharged.**
 2. **Define events that cause transitions (e.g., Register, Check In, Start Consultation, Complete Consultation).**
 3. **Create the diagram with states represented as circles or rounded rectangles.**
 4. **Draw arrows to indicate transitions between states, labeling them with the events that trigger the transitions.**
 5. **Validate for accuracy and completeness.**



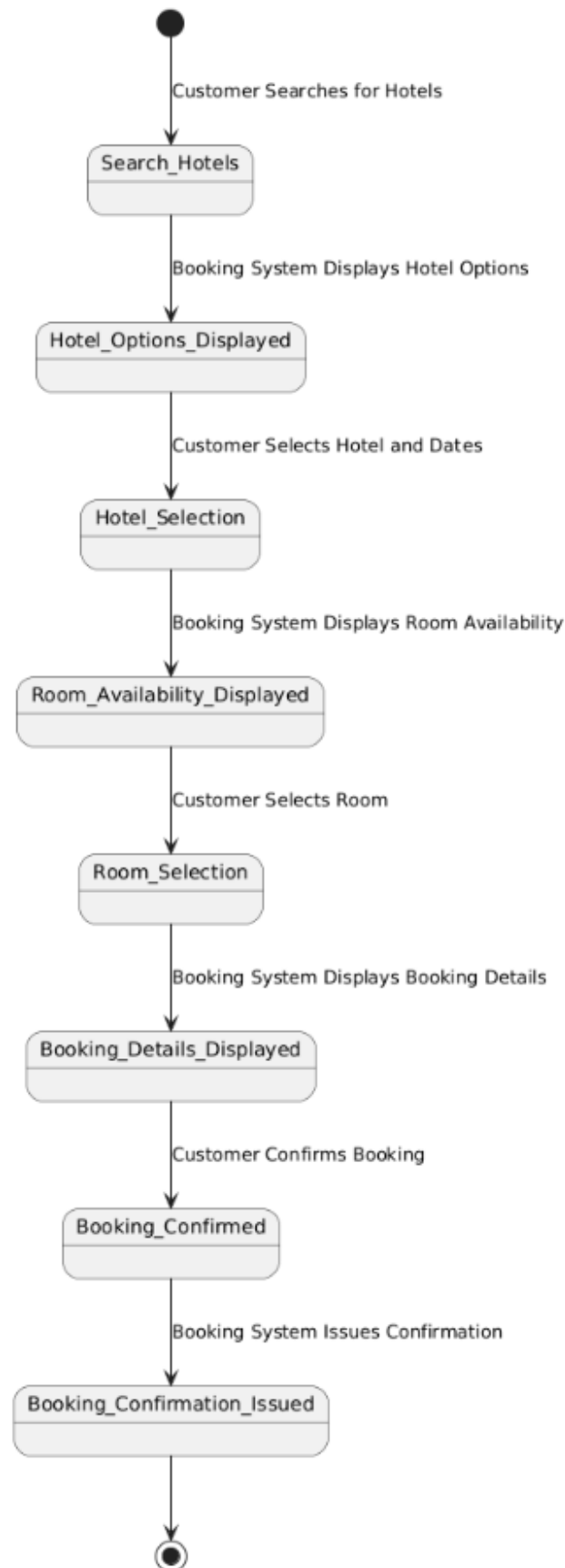
4. Social Media Platform

- **Aim: To represent the states of a user's post on a social media platform.**
- **Algorithm:**
 1. **Identify the states: Draft, Published, Liked, Commented, Archived.**
 2. **Define events that cause transitions (e.g., Publish Post, Like Post, Comment on Post, Archive Post).**
 3. **Create the diagram with states represented as circles or rounded rectangles.**
 4. **Draw arrows to indicate transitions between states, labeling them with the events that trigger the transitions.**
 5. **Validate for clarity and completeness.**



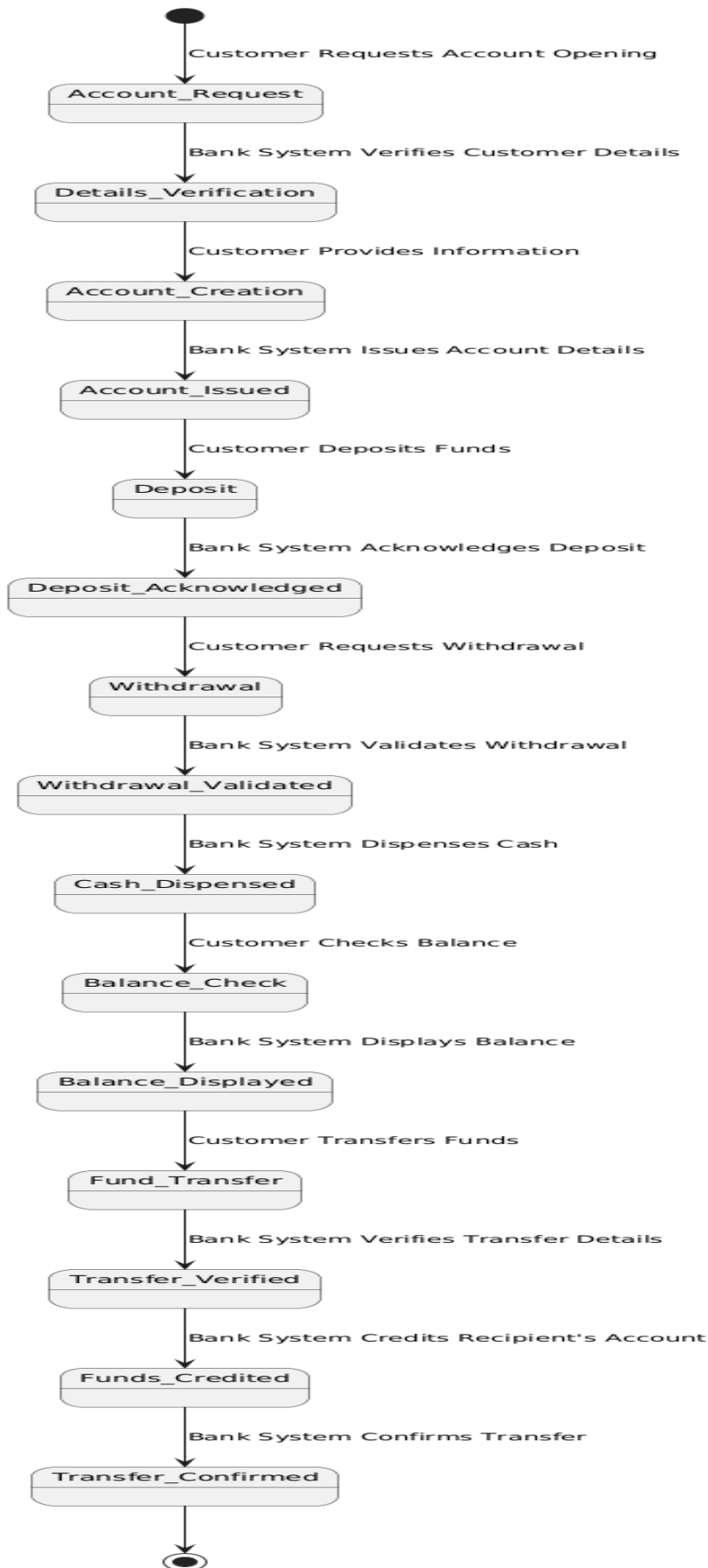
5. Hotel Booking System

- **Aim: To illustrate the states of a hotel booking process.**
- **Algorithm:**
 1. **Identify the states: Searching, Booked, Checked In, Checked Out, Canceled.**
 2. **Define events that cause transitions (e.g., Search Hotels, Confirm Booking, Check In, Check Out, Cancel Booking).**
 3. **Create the diagram with states represented as circles or rounded rectangles.**
 4. **Draw arrows to indicate transitions between states, labeling them with the events that trigger the transitions.**
 5. **Validate for accuracy and completeness.**



6. Banking System

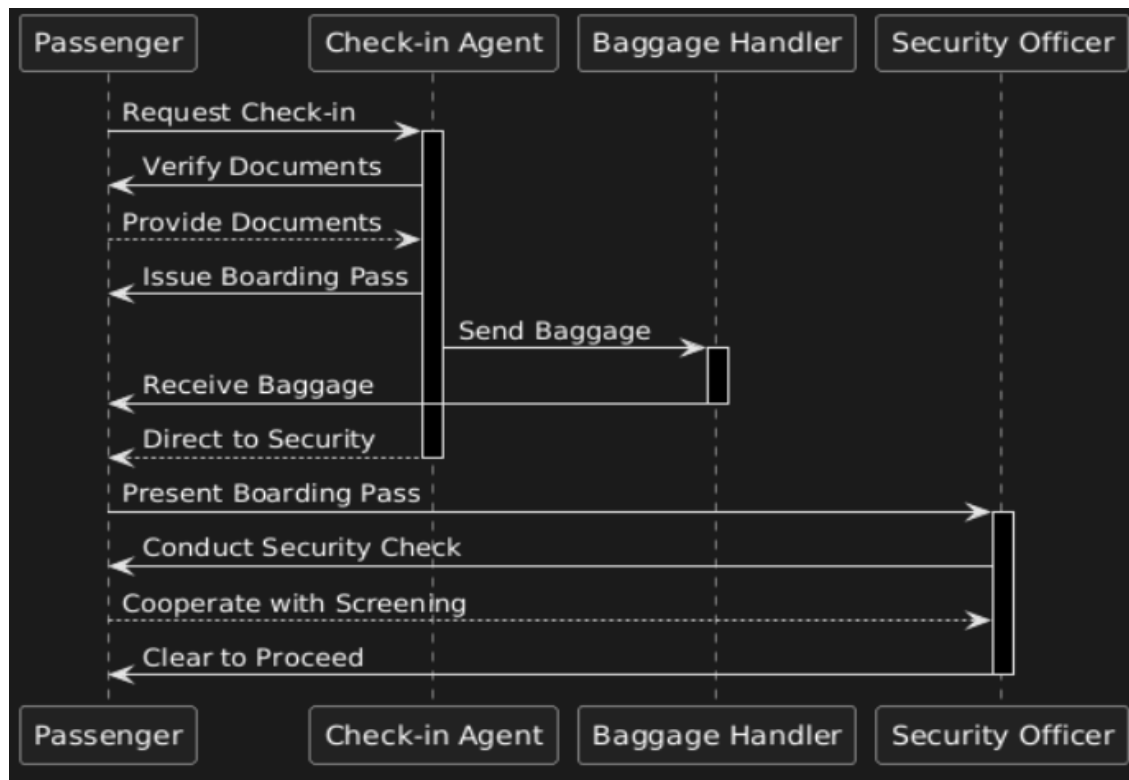
- Aim: To depict the states of a banking transaction process.
- Algorithm:
 1. Identify the states: Logged Out, Logged In, Transaction Initiated, Transaction Completed, Transaction Failed.
 2. Define events that cause transitions (e.g., Log In, Initiate Transaction, Complete Transaction, Fail Transaction).
 3. Create the diagram with states represented as circles or rounded rectangles.
 4. Draw arrows to indicate transitions between states, labeling them with the events that trigger the transitions.
 5. Validate for clarity and completeness.



SEQUENCE DIAGRAMS(UML)

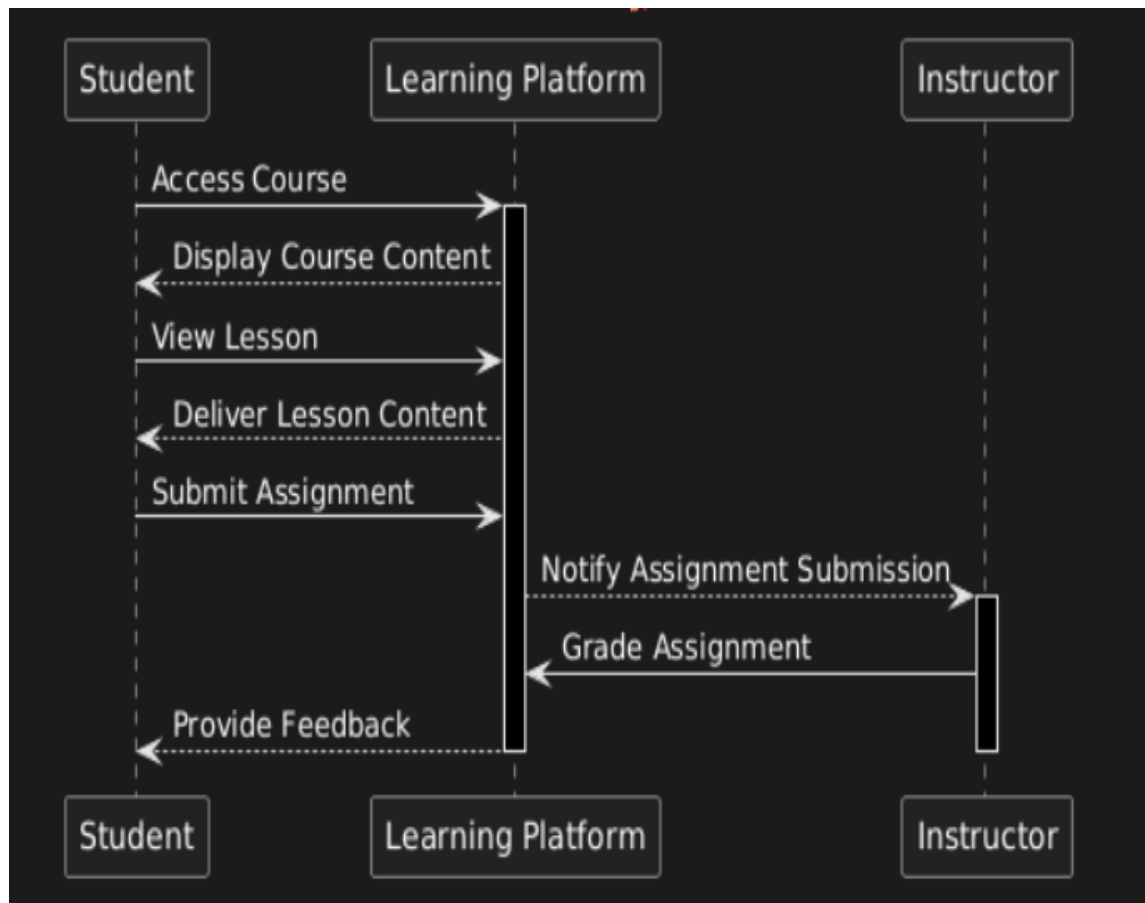
1)Airport Check-In and Security Screening:

- Aim: To illustrate the sequence of interactions during the check-in process.
- Algorithm:
 1. Identify objects: Passenger, CheckInKiosk, AirlineStaff.
 2. Define the sequence of messages exchanged.
 3. Arrange objects horizontally and time sequence vertically.
 4. Draw arrows to represent messages.
 5. Review for accuracy.



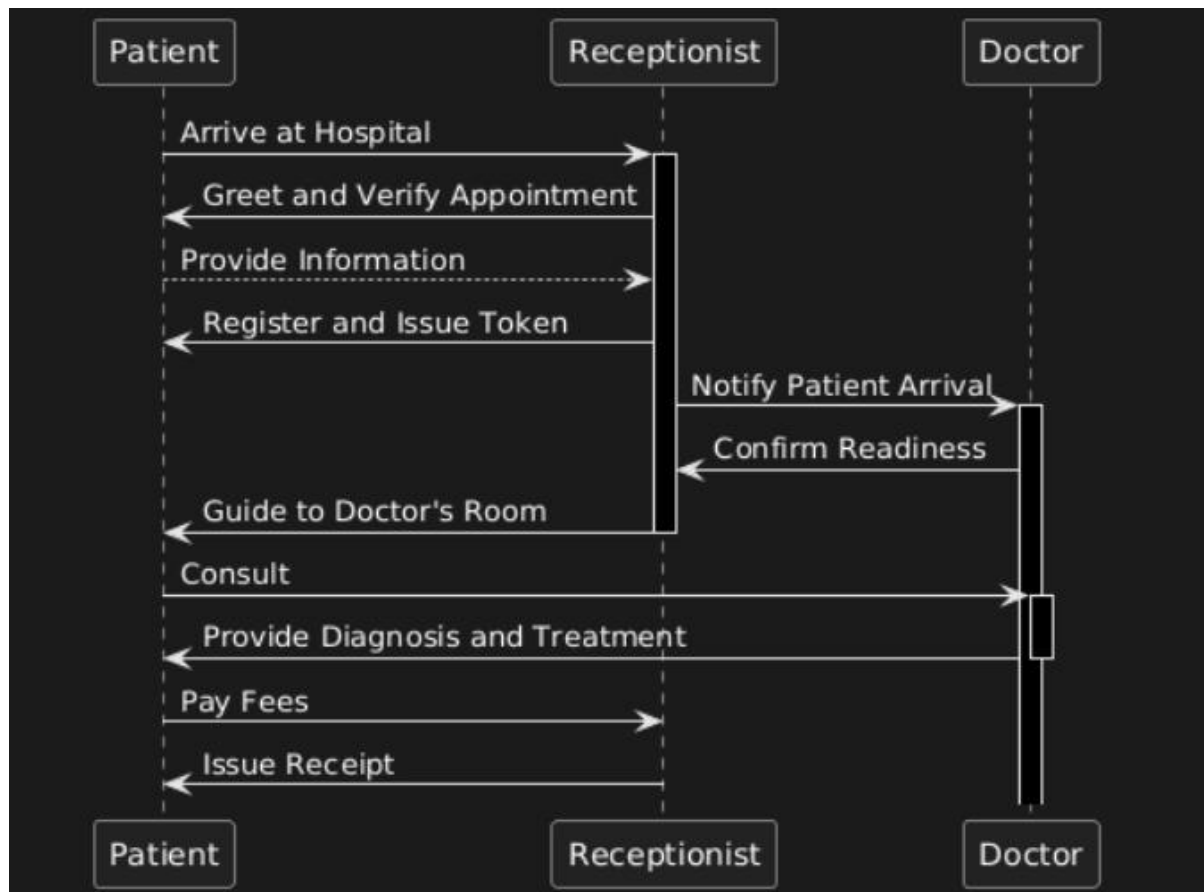
2)E-Learning Platform:

- Aim: To depict the sequence of interactions for course enrollment.
- Algorithm:
 1. Identify objects: Student, Course, Instructor.
 2. Define the sequence of messages exchanged.
 3. Arrange objects horizontally and time sequence vertically.
 4. Draw arrows to represent messages.
 5. Review for completeness.



3)HOSPITAL RECIEPTIONIST AND PATIENT:

- **Aim: To illustrate the sequence of interactions during patient check-in.**
- **Algorithm:**
 1. **Identify objects: Patient, Receptionist, AppointmentSystem.**
 2. **Define the sequence of messages exchanged.**
 3. **Arrange objects horizontally and time sequence vertically.**
 4. **Draw arrows to represent messages.**
 5. **Review for accuracy.**



4)SOCIAL MEDIA PLATFROM:

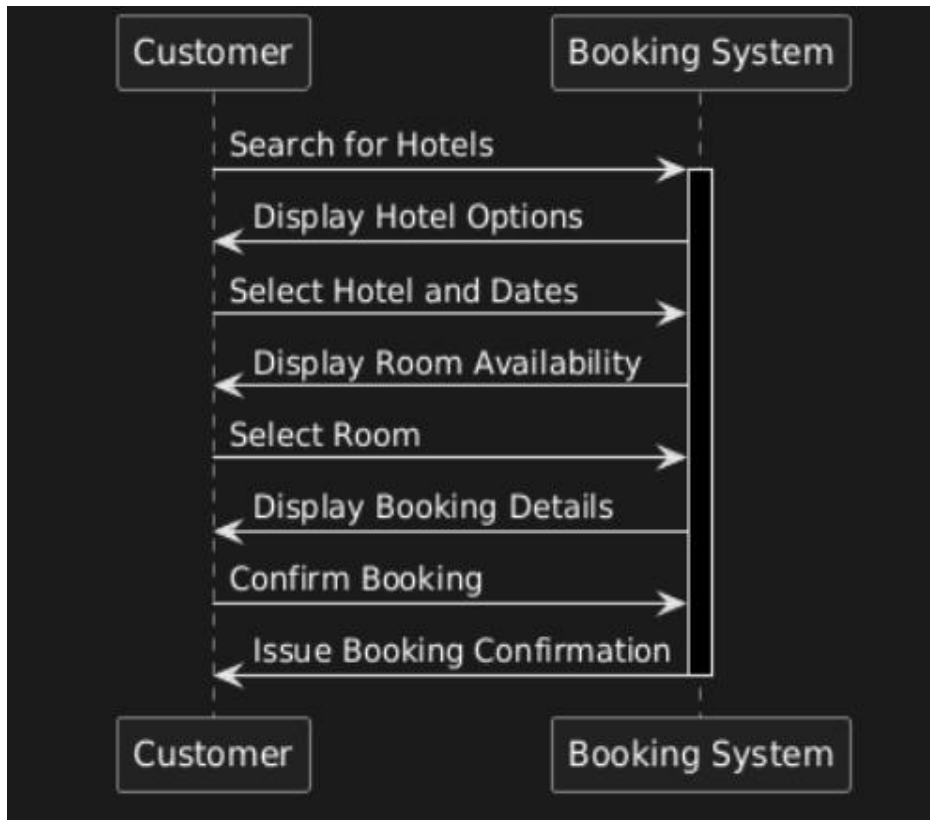
- **Aim: To depict the sequence of interactions for posting content.**
- **Algorithm:**
 1. **Identify objects: User, Post, NotificationSystem.**
 2. **Define the sequence of messages exchanged.**
 3. **Arrange objects horizontally and time sequence vertically.**
 4. **Draw arrows to represent messages.**
 5. **Review for completeness.**



5)HOTEL BOOKING SYSTEM:

- Aim: To illustrate the sequence of interactions during a hotel reservation.
- Algorithm:
 1. Identify objects: Customer, BookingSystem, PaymentGateway.
 2. Define the sequence of messages exchanged (e.g., search hotels, select hotel, make payment).
 3. Arrange objects horizontally and time sequence vertically.

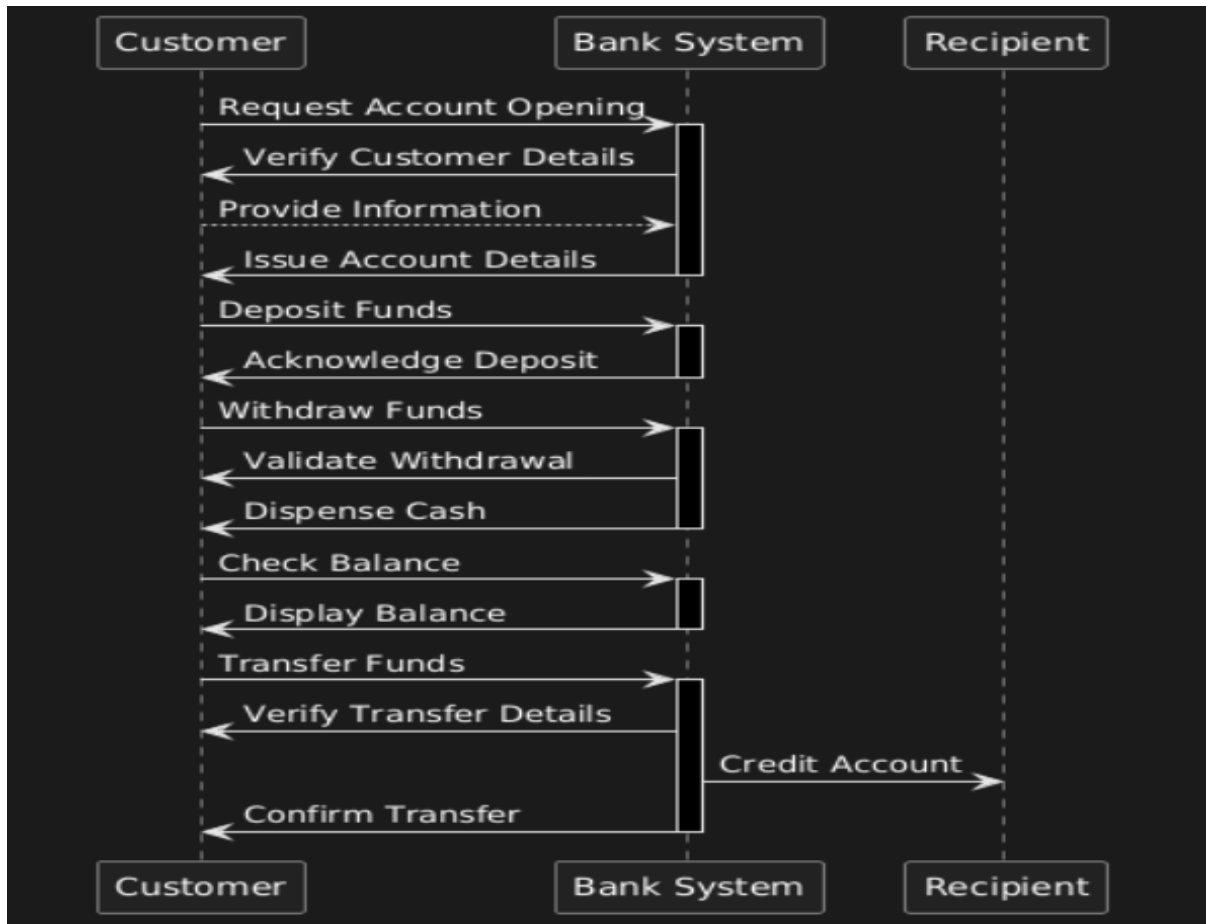
4. Draw arrows to represent messages (e.g., Customer sends a request to BookingSystem).
5. Review for completeness and accuracy.



6) BANKING SYSTEM:

- Aim: To depict the sequence of interactions for a banking transaction.
- Algorithm:
 1. Identify objects: Customer, BankAccount, ATM.
 2. Define the sequence of messages exchanged (e.g., check balance, withdraw funds).
 3. Arrange objects horizontally and time sequence vertically.

4. Draw arrows to represent messages (e.g., Customer requests balance from BankAccount).
5. Review for accuracy.

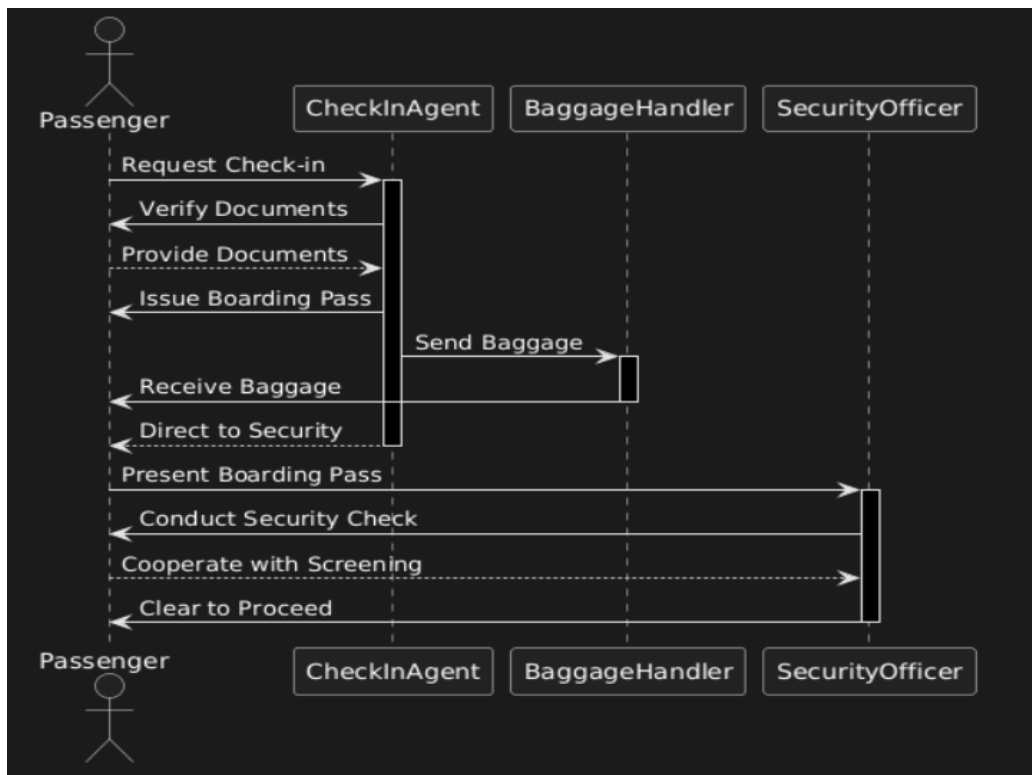


OBJECT DIAGRAMS(UML)

1)Airport Check-In and Security Screening:

- Aim: To model the flow of activities during the airport check-in process.
- Algorithm:
 1. Identify main activities: arrive at airport, check in, drop baggage, go through security.

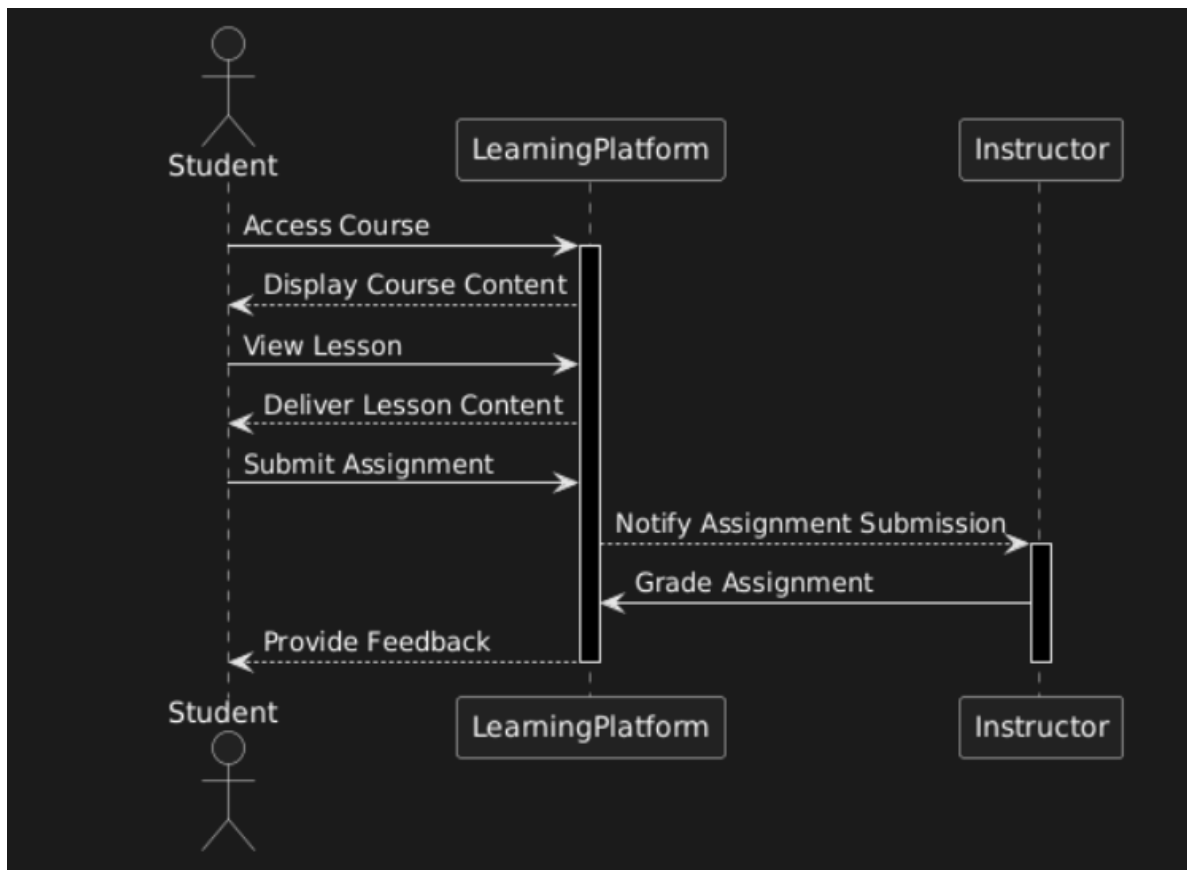
2. Determine flow paths and decision points (e.g., is baggage checked?).
3. Represent activities using rounded rectangles and decisions using diamonds.
4. Draw arrows to indicate the flow of activities.
5. Validate for clarity and completeness.



2)E-Learning Platform:

- Aim: To illustrate the flow of activities in the e-learning platform.
- Algorithm:
 1. Identify main activities: log in, browse courses, enroll, submit assignments.

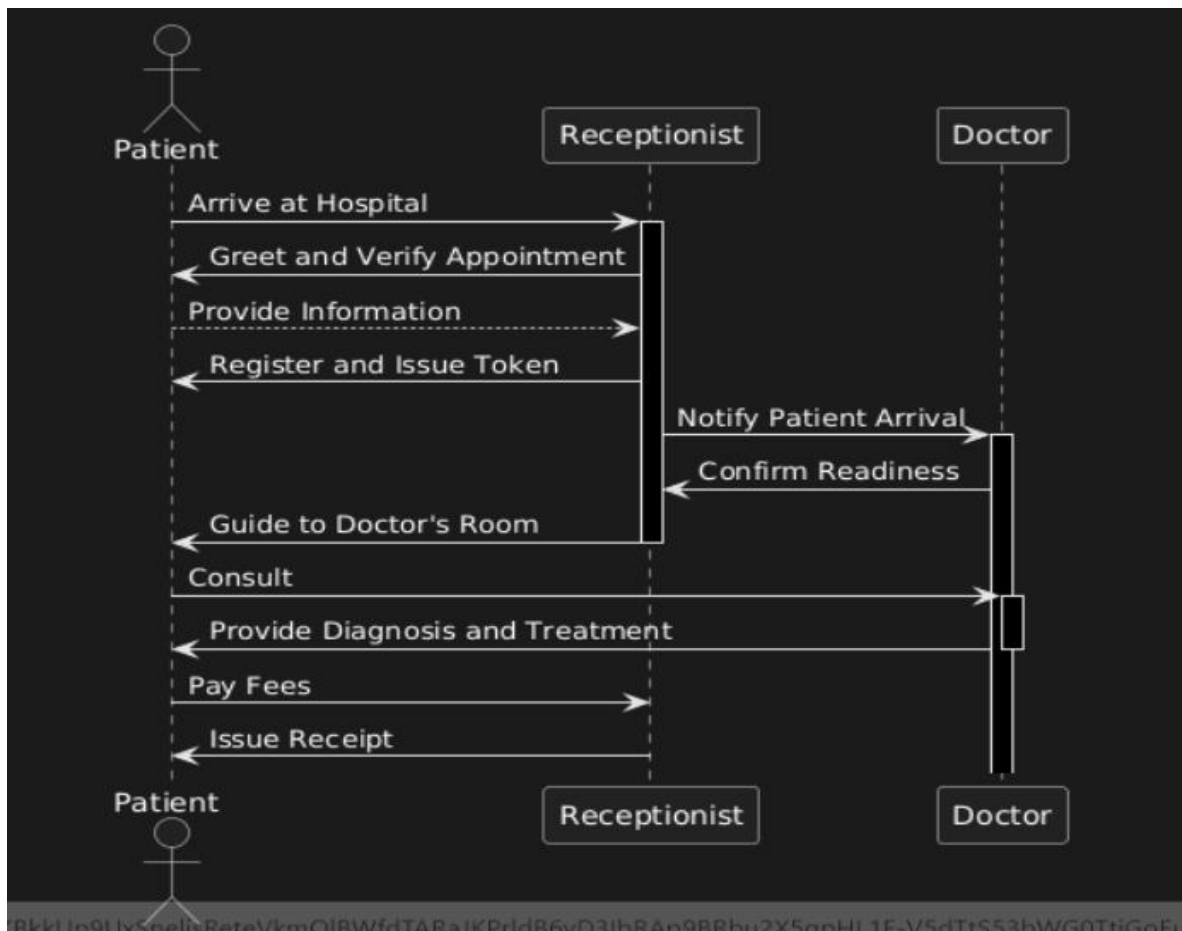
2. Determine flow paths and decision points (e.g., is the assignment submitted on time?).
3. Represent activities using rounded rectangles and decisions using diamonds.
4. Draw arrows to indicate the flow of activities.
5. Validate for accuracy.



3)HOSPITAL RECIEPTIONIST AND PATIENT:

- Aim: To model the flow of activities during patient check-in at a hospital.
- Algorithm:
 1. Identify main activities: patient arrives, check-in, verify information, schedule appointment.

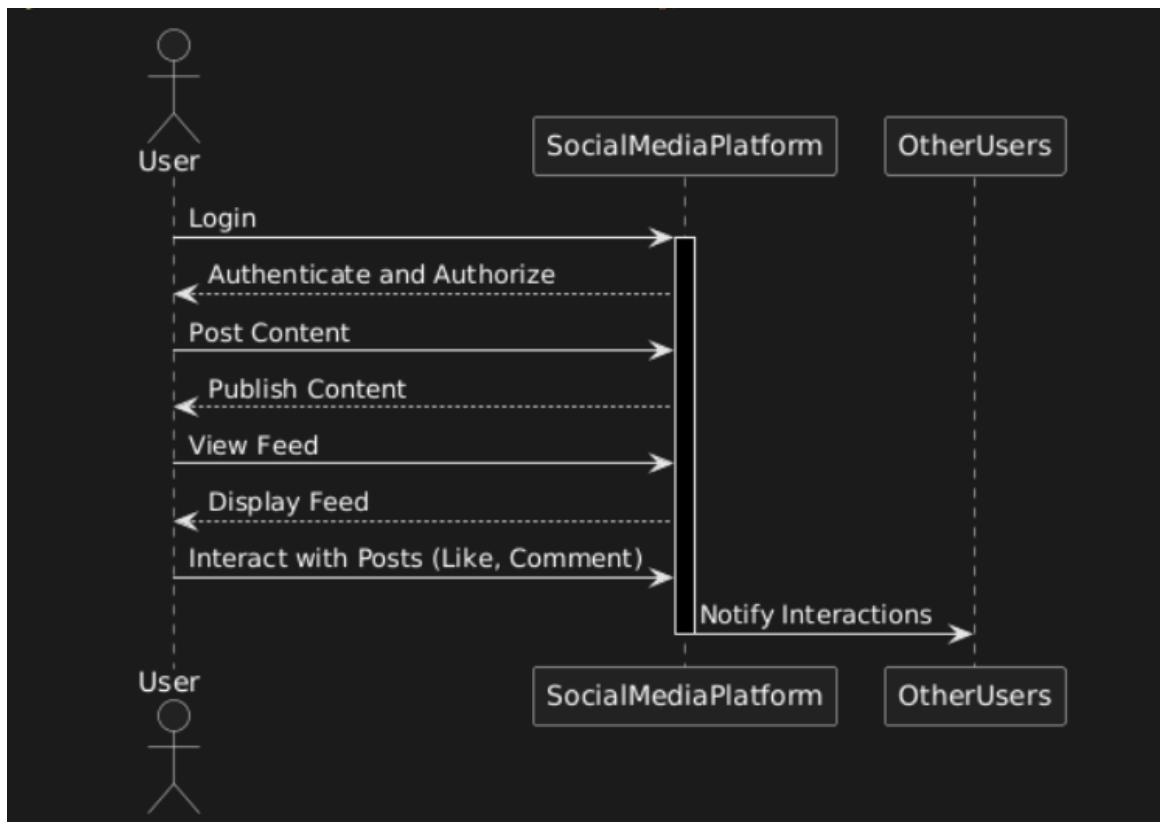
2. Determine flow paths and decision points (e.g., is the patient a new or returning patient?).
3. Represent activities using rounded rectangles and decisions using diamonds.
4. Draw arrows to indicate the flow of activities.
5. Validate for clarity.



4)SOCIAL MEDIA PLATFROM:

- Aim: To illustrate the flow of activities on a social media platform.
- Algorithm:

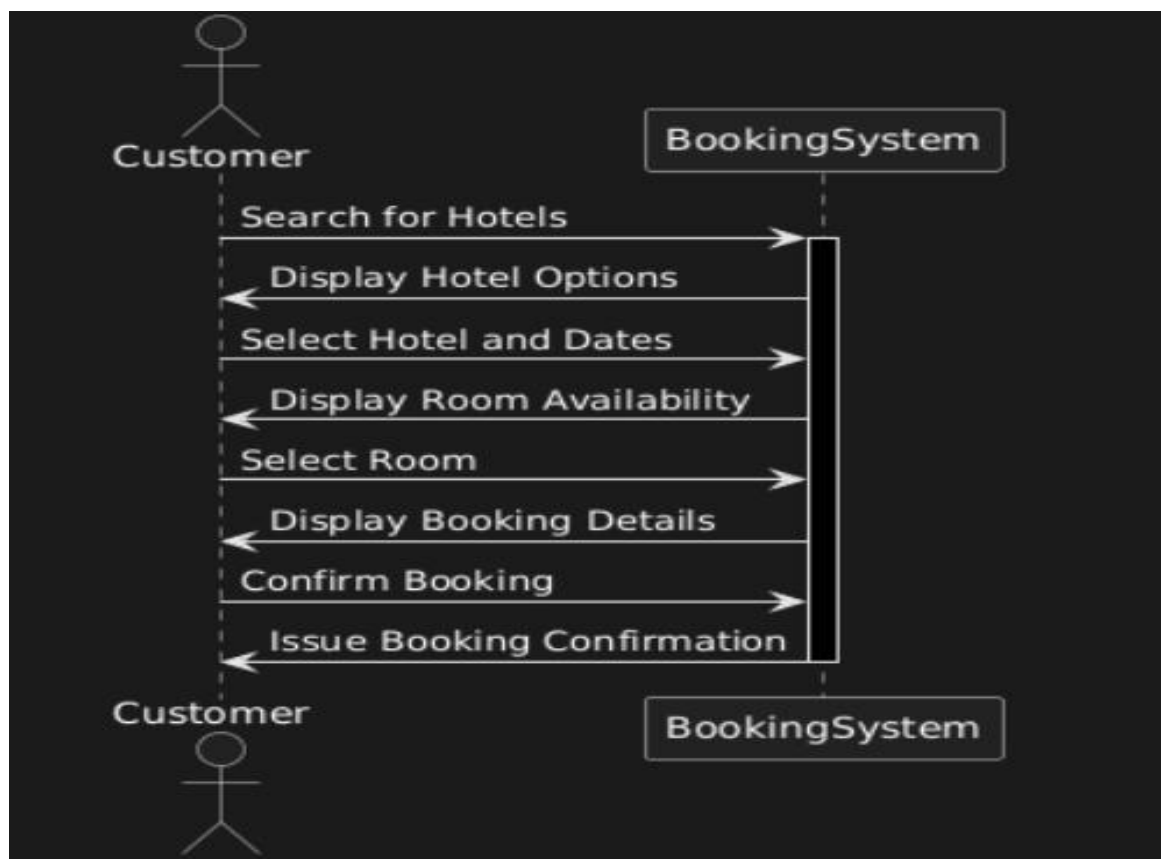
1. Identify main activities: log in, create post, like post, comment on post.
2. Determine flow paths and decision points (e.g., is the post public or private?).
3. Represent activities using rounded rectangles and decisions using diamonds.
4. Draw arrows to indicate the flow of activities.
5. Validate for completeness.



5)HOTEL BOOKING SYSTEM:

- Aim: To model the flow of activities during the hotel booking process.
- Algorithm:

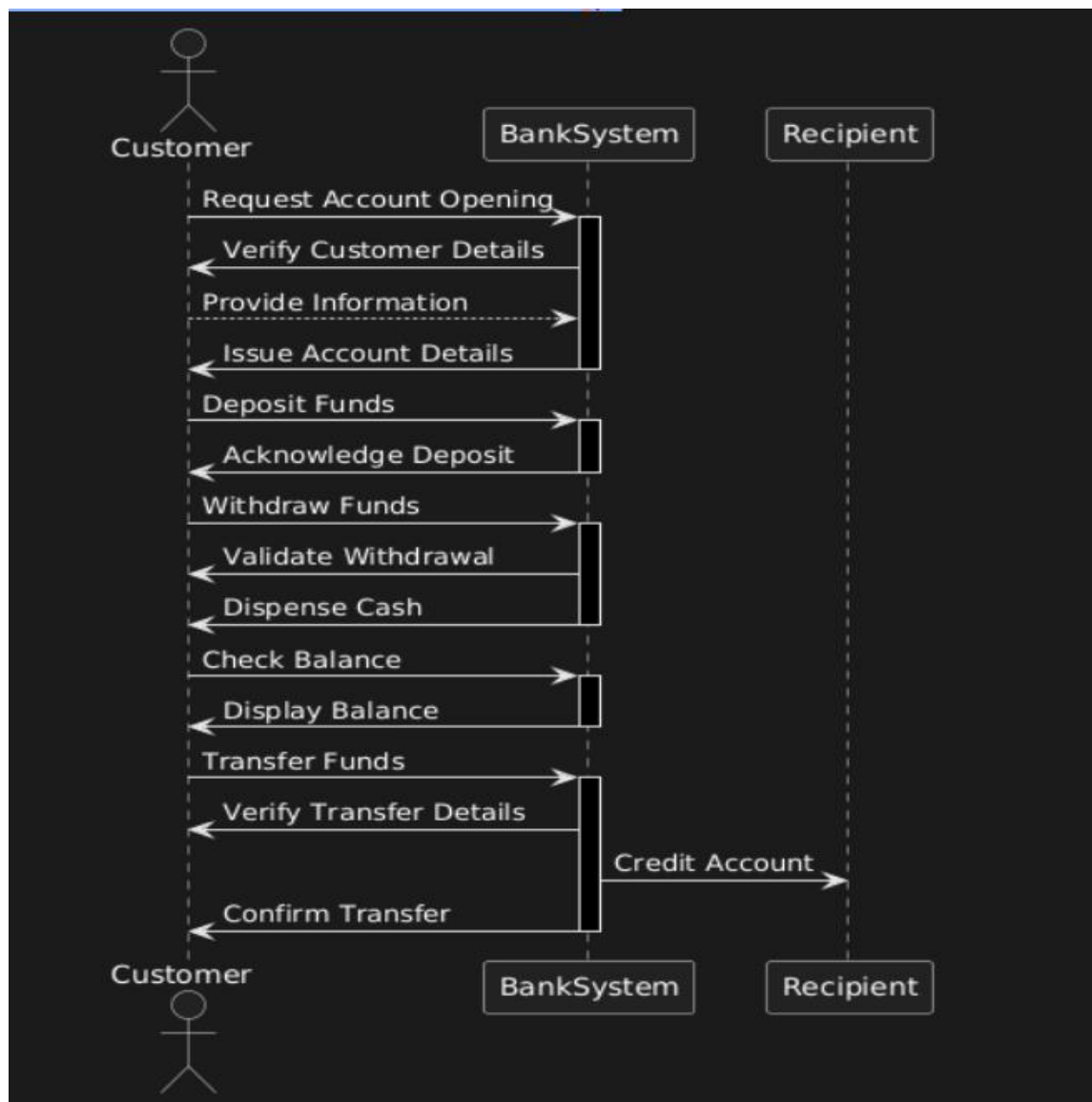
1. Identify main activities: search for hotels, select hotel, enter guest information, confirm booking.
2. Determine flow paths and decision points (e.g., is payment successful?).
3. Represent activities using rounded rectangles and decisions using diamonds.
4. Draw arrows to indicate the flow of activities.
5. Validate for accuracy.



6)BANKING SYSTEM:

- Aim: To illustrate the flow of activities in a banking transaction.
- Algorithm:

1. Identify main activities: log in, check balance, withdraw funds, confirm transaction.
2. Determine flow paths and decision points (e.g., is there sufficient balance?).
3. Represent activities using rounded rectangles and decisions using diamonds.
4. Draw arrows to indicate the flow of activities.
5. Validate for clarity and completeness.



JAVA BASICS PROGRAMS:

1)Multiplying a number till 10:

PROGRAM:

```
import java.util.*;

public class basics {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int a = sc.nextInt();

        int i = 1;

        while (i <= a) {

            System.out.println(i * a);

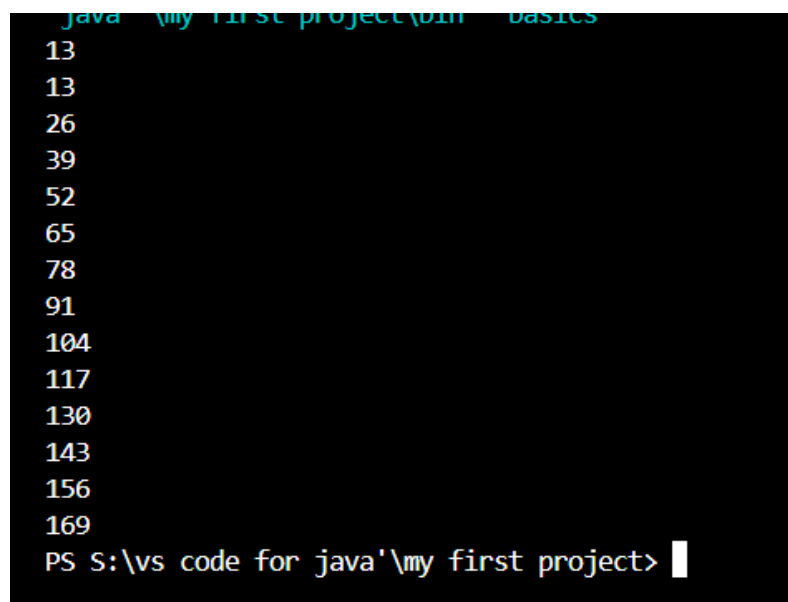
            i++;

        }

    }

}
```

OUTPUT:



```
java -my first project\bin -basics
13
13
26
39
52
65
78
91
104
117
130
143
156
169
PS S:\vs code for java'\my first project> |
```

2)Checking which the largest number from three:

PROGRAM:

```

import java.util.*;

public class basics {

    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);

        int a = sc.nextInt();

        int b = sc.nextInt();

        int c = sc.nextInt();

        if(a > b && a > c){

            System.out.println(a + " is the greatest among the three");

        } else if(b > a && b > c){

            System.out.println(b + " is the greatest among the three ");

        } else {

            System.out.println(c + " is the greatest among the three");

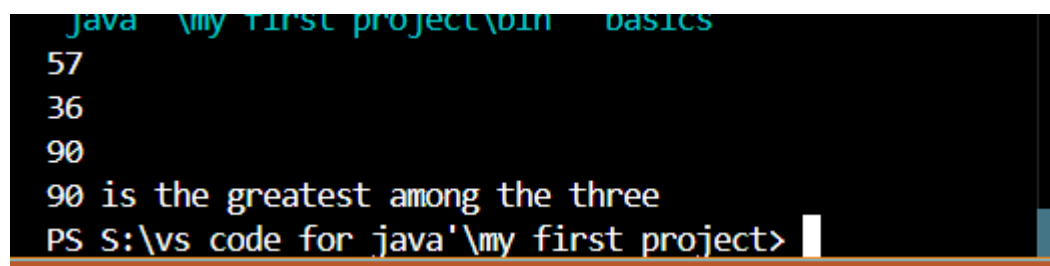
        }

    }

}

```

OUTPUT:



The screenshot shows a terminal window with a black background and green text. The prompt is 'java -\my first project\bin - basics'. The user has entered three integers: 57, 36, and 90. The program output is '90 is the greatest among the three'. The prompt at the bottom is 'PS S:\vs code for java'\my first project>'. There is a white cursor bar after the prompt.

```

java -\my first project\bin - basics
57
36
90
90 is the greatest among the three
PS S:\vs code for java'\my first project>

```

3)Calculating the area of a triangle:

PROGRAM:

```

import java.util.*;

```

```

public class basics{

    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);

        int b = sc.nextInt();

        int h = sc.nextInt();

        double c = (0.5 * b * h);

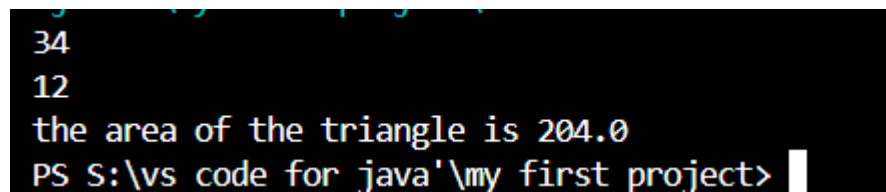
        System.out.println("the area of the triangle is " + c);

    }

}

```

OUTPUT:



```

34
12
the area of the triangle is 204.0
PS S:\vs code for java'\my first project>

```

4)Calculating the area of a circle:

PROGRAM:

```

import java.util.*;

public class basics{

    public static Double getCircumference(Double r){

        return 2 * 3.14 * r;

    }

    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);

        Double r = sc.nextDouble();

        System.out.println(getCircumference(r));

    }

}

```

```
}  
}
```

OUTPUT:

```
4  
25.12  
PS S:\vs code for java'\my first project> |
```

5)Checking whether the person is eligible for voting or not:

PROGRAM:

```
import java.util.*;  
public class basics{  
    public static boolean isEligible(int age){  
        if(age >= 18){  
            return true;  
        } else {  
            return false;  
        }  
    }  
}  
  
public static void main(String[] args){  
    Scanner sc = new Scanner(System.in);  
    int age = sc.nextInt();  
    System.out.println(isEligible(age));  
  
}  
}
```

OUTPUT:

```
19
true
PS S:\vs code for java'\my first project> |
```

6)Fibonacci series:

PROGRAM:

```
import java.util.*;

public class basics{

    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int a = 0;

        int b = 1;

        System.out.println(a + " ");

        if(n > 1){

            for(int i = 2; i <= n; i++){

                System.out.println(b + " ");

                int temp = b;

                b = a + b;

                a = temp;

            }

            System.out.println();

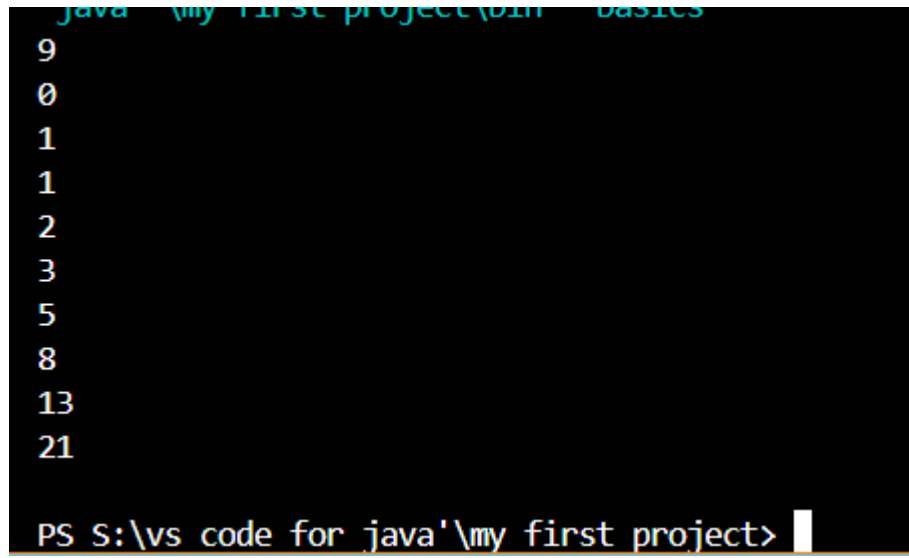
        }

    }

}
```

```
}  
}
```

OUTPUT:



```
java -my first project\bin -basics  
9  
0  
1  
1  
2  
3  
5  
8  
13  
21  
PS S:\vs code for java'\my first project>
```

7)Area and Volume of a Room (Single Inheritance)

PROGRAM:

```
class Room {  
    double length;  
    double width;  
    double height;  
  
    double calculateArea() {  
        return length * width;  
    }  
}  
  
class VolumeRoom extends Room {
```

```

double calculateVolume() {
    return length * width * height;
}
}

```

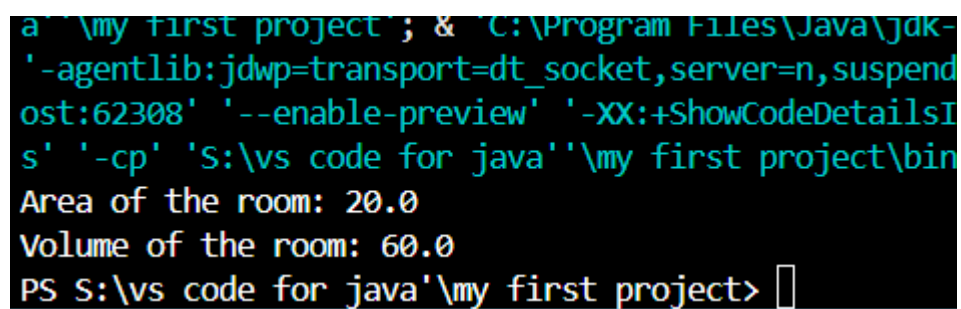
```

public class college {
    public static void main(String[] args) {
        VolumeRoom room = new VolumeRoom();
        room.length = 5;
        room.width = 4;
        room.height = 3;

        System.out.println("Area of the room: " + room.calculateArea());
        System.out.println("Volume of the room: " + room.calculateVolume());
    }
}

```

OUTPUT:



```

a ^\my first project ; & ^C:\Program Files\Java\jdk-
'-agentlib:jdwp=transport=dt_socket,server=n,suspend
ost:62308' '--enable-preview' '-XX:+ShowCodeDetailsI
s' '-cp' 'S:\vs code for java'\my first project\bin
Area of the room: 20.0
Volume of the room: 60.0
PS S:\vs code for java\my first project>

```

8) Hierarchical Inheritance (Shape, Rectangle, Circle)

PROGRAM:

```

class Shape {

```

```
double area;

void calculateArea() {
    // Default implementation
}

}

class Rectangle extends Shape {
    double length;
    double width;

    @Override
    void calculateArea() {
        area = length * width;
        System.out.println("Area of Rectangle: " + area);
    }
}

class Circle extends Shape {
    double radius;

    @Override
    void calculateArea() {
        area = Math.PI * radius * radius;
        System.out.println("Area of Circle: " + area);
    }
}
```



```

public class college {

    public static void main(String[] args) {

        Rectangle rectangle = new Rectangle();

        rectangle.length = 5;

        rectangle.width = 4;

        rectangle.calculateArea();


        Circle circle = new Circle();

        circle.radius = 3;

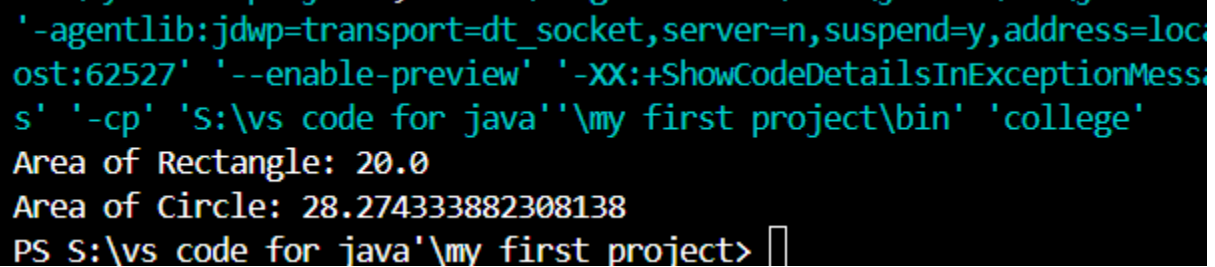
        circle.calculateArea();

    }

}

```

OUTPUT:



```

'-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=loc
ost:62527' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMess
s' '-cp' 'S:\vs code for java\my first project\bin' 'college'
Area of Rectangle: 20.0
Area of Circle: 28.274333882308138
PS S:\vs code for java\my first project>

```

9)Multilevel Inheritance Example

PROGRAM:

```

class Animal {

    void eat() {

        System.out.println("Animal is eating.");

    }

}

```

```

}

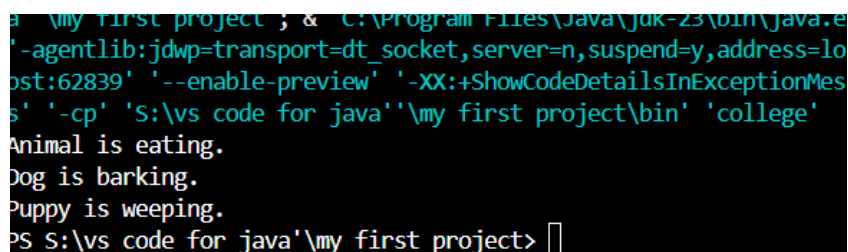
class Dog extends Animal {
    void bark() {
        System.out.println("Dog is barking.");
    }
}

class Puppy extends Dog {
    void weep() {
        System.out.println("Puppy is weeping.");
    }
}

public class Main {
    public static void main(String[] args) {
        Puppy puppy = new Puppy();
        puppy.eat();
        puppy.bark();
        puppy.weep();
    }
}

```

OUTPUT:



```

PS S:\vs code for java\my first project> java -cp S:\vs code for java\my first project\bin college
Animal is eating.
Dog is barking.
Puppy is weeping.
PS S:\vs code for java\my first project>

```

10)Shape Class with CalculateArea Method

PROGRAM:

```
abstract class shape{
    abstract double calculatearea();
}

class circle extends shape{
    double radius;
    circle(double radius){
        this.radius = radius;
    }
    @Override
    double calculatearea(){
        return 3.14 * radius * radius;
    }
}

class rectangle extends shape{
    double length;
    double breadth;
    rectangle(double length,double breadth){
        this.length = length;
        this.breadth = breadth;
    }
    @Override
    double calculatearea(){
        return length * breadth;
    }
}
```

```

    }
}

public class college{

    public static void main(String[] args){

        shape mycircle = new circle(5);

        shape myrectangle = new rectangle(4,6);

        System.out.println(mycircle.calculatearea());

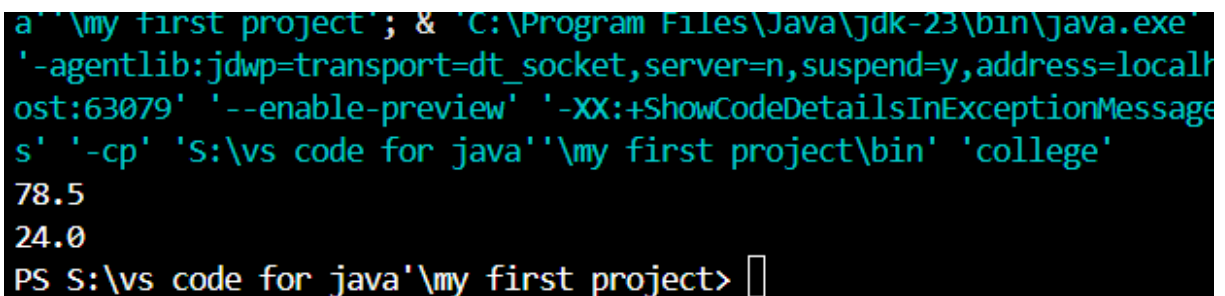
        System.out.println(myrectangle.calculatearea());

    }

}

```

OUTPUT:



```

a '\my first project'; & 'C:\Program Files\Java\jdk-23\bin\java.exe'
'-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:63079' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'S:\vs code for java'\my first project\bin' 'college'
78.5
24.0
PS S:\vs code for java\my first project>

```

INHERITANCE PROGRAMS:

Single Inheritance:

1)**Question:** Create a class Animal with a method sound() and a method calculateAge(int years). Create a subclass Dog that overrides the sound() method and uses the calculateAge() method.

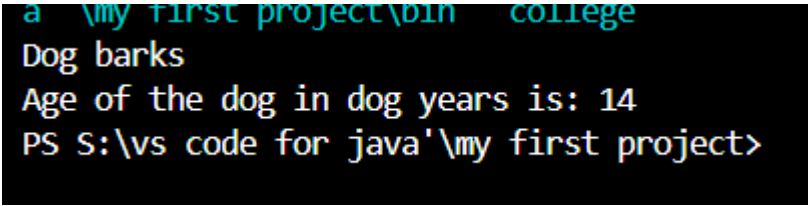
Code:

```
class Animal {  
    void sound() {  
        System.out.println("Animal makes a sound");  
    }  
  
    int calculateAge(int years) {  
        return years * 7;  
    }  
}  
  
class Dog extends Animal {  
    void sound() {  
        System.out.println("Dog barks");  
    }  
}  
  
class Main {  
    public static void main(String[] args) {  
        Dog d = new Dog();  
        d.sound();  
    }  
}
```

```
        int age = d.calculateAge(2);

        System.out.println("Age of the dog in dog years is: " +
age);
    }
}
```

Output:



```
a \my first project\bin college
Dog barks
Age of the dog in dog years is: 14
PS S:\vs code for java'\my first project>
```

2)**Question:** Create a class Vehicle with a method type() and a method calculateSpeed(int distance, int time). Create a subclass Car that overrides the type() method.

Code:

```
class Vehicle {
    void type() {
        System.out.println("This is a vehicle");
    }

    double calculateSpeed(int distance, int time) {
        return (double) distance / time;
    }
}
```

```
}
```

```
class Car extends Vehicle {
```

```
    void type() {
```

```
        System.out.println("This is a car");
```

```
    }
```

```
}
```

```
class Main {
```

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

```
        Car c = new Car();
```

```
        c.type();
```

```
        double speed = c.calculateSpeed(100, 2);
```

```
        System.out.println("The speed of the car is " + speed + " km/h");
```

```
    }
```

```
}
```

Output:

```
This is a car
```

```
The speed of the car is 50.0 km/h
```

```
PS S:\vs code for java\my first project> █
```

MULTILEVEL INHERITANCE:

3)**Question:** Create a class Person, a subclass Employee, and a subclass Manager. Each class should have a method to display its role and a method to calculate salary.

Code:

```
class Person {  
    void role() {  
        System.out.println("I am a person");  
    }  
  
    double calculateSalary(double baseSalary) {  
        return baseSalary;  
    }  
}
```

```
class Employee extends Person {  
    void role() {  
        System.out.println("I am an employee");  
    }  
  
    double calculateSalary(double baseSalary) {  
        return baseSalary * 1.2;  
    }  
}
```



```
}
```

```
class Manager extends Employee {  
    void role() {  
        System.out.println("I am a manager");  
    }  
  
    double calculateSalary(double baseSalary) {  
        return baseSalary * 1.5;  
    }  
}
```

```
class Main {  
    public static void main(String[] args) {  
        Manager m = new Manager();  
        m.role();  
        double salary = m.calculateSalary(1000);  
        System.out.println("Manager's salary is: " + salary);  
    }  
}
```

Output:

```
03E05014 enable preview 0X.F3HOWE0A0E0E
s' '-cp' 'S:\vs code for java'\my first projec
I am a manager
Manager's salary is: 1500.0
PS S:\vs code for java'\my first project> █
```

4)**Question:** Create a class Animal, a subclass Mammal, and a subclass Dog. Each class should have a method to display its characteristics and a method to calculate the average weight.

Code:

```
class Animal {
    void character() {
        System.out.println("I am an animal");
    }

    double calculateAverageWeight(double weight1, double
weight2) {
        return (weight1 + weight2) / 2;
    }
}

class Mammal extends Animal {
    void character() {
        System.out.println("I am a mammal");
    }
}
```

```

    }
}

class Dog extends Mammal {
    void character() {
        System.out.println("I am a dog");
    }
}

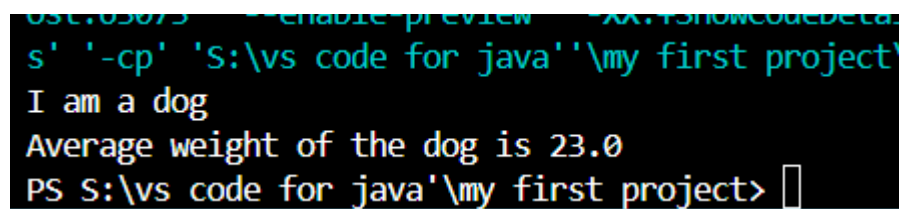
```

```

class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.character();
        double avgWeight = d.calculateAverageWeight(20.5,
        25.5);
        System.out.println("Average weight of the dog is " +
        avgWeight);
    }
}

```

Output:



```

PS S:\vs code for java\my first project> java -cp 'S:\vs code for java\my first project'
I am a dog
Average weight of the dog is 23.0
PS S:\vs code for java\my first project>

```

HIERARICAL INHERITANCE:

5)Question: Create a class Shape with a method draw() and a method calculateArea(double radius) for circles and calculateArea(double length, double width) for rectangles. Create subclasses Circle and Rectangle that override the draw() method.

Code:

```
class Shape {  
    void draw() {  
        System.out.println("Drawing a shape");  
    }  
  
    double calculateArea(double radius) {  
        return 3.14 * radius * radius;  
    }  
  
    double calculateArea(double length, double breadth) {  
        return length * breadth;  
    }  
}
```

```
class Circle extends Shape {  
    void draw() {  
        System.out.println("Drawing a circle");  
    }  
}
```

```
class Rectangle extends Shape {  
    void draw() {  
        System.out.println("Drawing a rectangle");  
    }  
}
```

```
class Main {  
    public static void main(String[] args) {  
        Circle c = new Circle();  
        c.draw();  
        double cArea = c.calculateArea(5);  
        System.out.println("The area of the circle is: " + cArea);  
  
        Rectangle r = new Rectangle();  
        r.draw();  
    }  
}
```

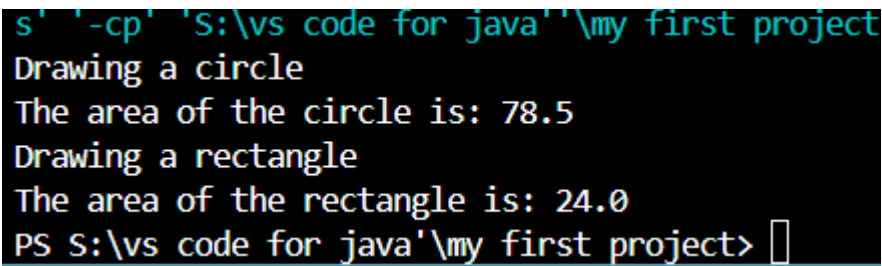
```
double rArea = r.calculateArea(4, 6);

System.out.println("The area of the rectangle is: " +
rArea);

}

}
```

Output:



```
s' '-cp' 'S:\vs code for java'\my first project
Drawing a circle
The area of the circle is: 78.5
Drawing a rectangle
The area of the rectangle is: 24.0
PS S:\vs code for java\my first project> █
```

6)Question: Create a class Bird with a method fly() and a method calculateWingspan(double wingLength). Create subclasses Sparrow and Eagle that override the fly() method.

Code:

```
class Bird {

    void fly() {

        System.out.println("Birds can fly");

    }

    double calculateWingspan(double wingLength) {

        return wingLength * 2;

    }

}
```

```
class Sparrow extends Bird {  
    void fly() {  
        System.out.println("Sparrow can fly");  
    }  
}
```

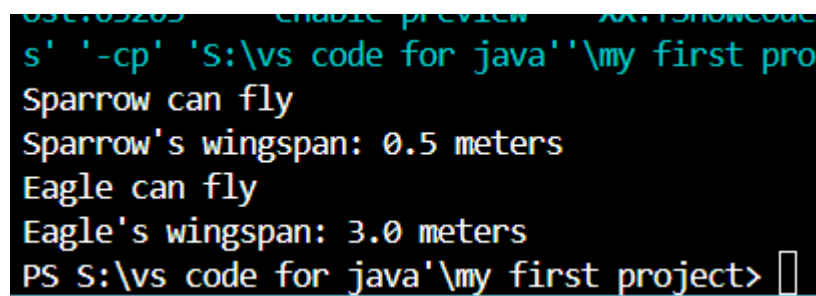
```
class Eagle extends Bird {  
    void fly() {  
        System.out.println("Eagle can fly");  
    }  
}
```

```
class Main {  
    public static void main(String[] args) {  
        Sparrow s = new Sparrow();  
        s.fly();  
        double sWingspan = s.calculateWingspan(0.25);  
        System.out.println("Sparrow's wingspan: " + sWingspan +  
" meters");  
  
        Eagle e = new Eagle();  
        e.fly();  
    }  
}
```

```
        double eWingspan = e.calculateWingspan(1.5);

        System.out.println("Eagle's wingspan: " + eWingspan + "
meters");
    }
}
```

Output:



```
03c703203 - Enable preview - xX.f3h0wcode
s' '-cp' 'S:\vs code for java'\my first pro
Sparrow can fly
Sparrow's wingspan: 0.5 meters
Eagle can fly
Eagle's wingspan: 3.0 meters
PS S:\vs code for java\my first project> █
```

HYBRID INHERITANCE:

7)**Question:** Create a class Person with a method displayInfo(). Create subclasses Employee and Student, and then create a subclass Intern that inherits from Employee and implements an interface Internship.

Code:

```
class Person {

    void displayInfo() {
```



```
        System.out.println("I am a person");
    }
}

class Employee extends Person {
    void displayInfo() {
        System.out.println("I am an employee");
    }
}

class Student extends Person {
    void displayInfo() {
        System.out.println("I am a student");
    }
}

class Intern extends Employee {
    void displayInfo() {
        System.out.println("I am an intern");
    }

    void internshipDetails() {
```

```
        System.out.println("Internship duration: 3 months");
    }
}
```

```
class GraduateIntern extends Intern {
    void displayInfo() {
        System.out.println("I am a graduate intern");
    }

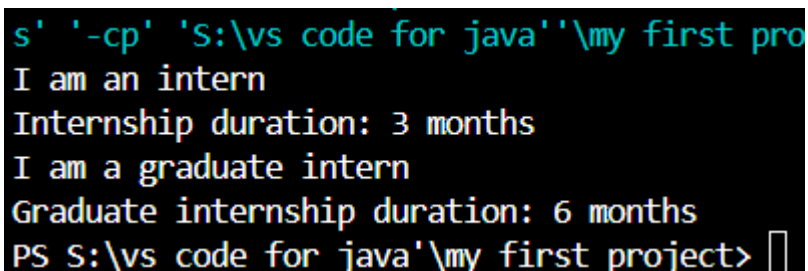
    void internshipDetails() {
        System.out.println("Graduate internship duration: 6
months");
    }
}
```

```
class Main {
    public static void main(String[] args) {
        Intern i = new Intern();
        i.displayInfo();
        i.internshipDetails();

        GraduateIntern g = new GraduateIntern();
        g.displayInfo();
    }
}
```

```
        g.internshipDetails();  
    }  
}
```

Output:



```
s' '-cp' 'S:\vs code for java'\my first pro  
I am an intern  
Internship duration: 3 months  
I am a graduate intern  
Graduate internship duration: 6 months  
PS S:\vs code for java'\my first project> █
```

8)Question: Create a class Animal with a method speak(). Create subclasses Mammal and Bird, and then create a subclass Bat that inherits from Mammal. Additionally, create a subclass Eagle that inherits from Bird.

Code:

```
class Animal {  
    void speak() {  
        System.out.println("Animal makes a sound");  
    }  
}
```

```
class Mammal extends Animal {  
    void speak() {  
        System.out.println("Mammal roars");  
    }  
}
```

```
class Bird extends Animal {  
    void speak() {  
        System.out.println("Bird chirps");  
    }  
}
```

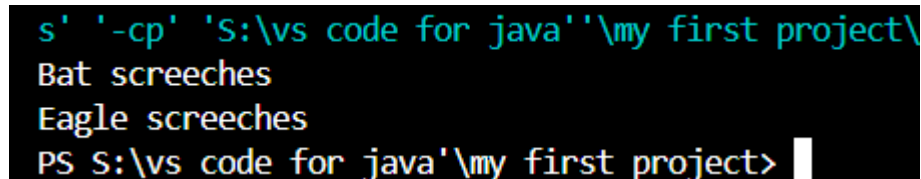
```
class Bat extends Mammal {  
    void speak() {  
        System.out.println("Bat screeches");  
    }  
}
```

```
class Eagle extends Bird {  
    void speak() {  
        System.out.println("Eagle screeches");  
    }  
}
```

```
}
```

```
class Main {  
    public static void main(String[] args) {  
        Bat bat = new Bat();  
        bat.speak();  
  
        Eagle eagle = new Eagle();  
        eagle.speak();  
    }  
}
```

Output:



```
s' '-cp' 'S:\vs code for java'\my first project\  
Bat screeches  
Eagle screeches  
PS S:\vs code for java\my first project>
```

POLYMORPHISM PROGRAMS:

OVERRIDING PROGRAMS:

1. Create a base class Animal with a method sound(). Create subclasses Dog and Cat that override the sound() method.

Code:

```
class Animal {  
    void sound() {  
        System.out.println("Animal makes a sound");  
    }  
}
```

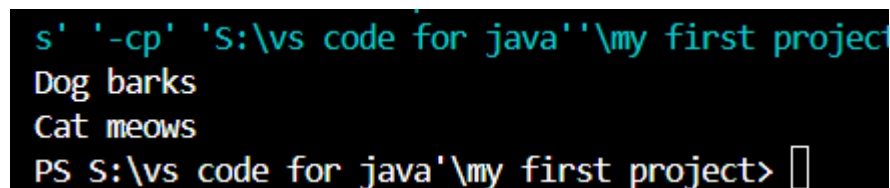
```
class Dog extends Animal {  
    @Override  
    void sound() {  
        System.out.println("Dog barks");  
    }  
}
```

```
class Cat extends Animal {  
    @Override  
    void sound() {  
        System.out.println("Cat meows");  
    }  
}
```

```
public class Main {
```

```
public static void main(String[] args) {  
    Animal myDog = new Dog();  
    Animal myCat = new Cat();  
  
    myDog.sound();  
    myCat.sound();  
}  
}
```

Output:



```
s' '-cp' 'S:\vs code for java'\my first project'  
Dog barks  
Cat meows  
PS S:\vs code for java'\my first project>
```

2. Create a base class Shape with a method area(). Create subclasses Circle and Rectangle that override the area() method.

Code:

```
class Shape {  
    double area() {  
        return 0;  
    }  
}
```

```
}
```

```
class Circle extends Shape {
```

```
    double radius;
```

```
    Circle(double radius) {
```

```
        this.radius = radius;
```

```
    }
```

```
    @Override
```

```
    double area() {
```

```
        return Math.PI * radius * radius;
```

```
    }
```

```
}
```

```
class Rectangle extends Shape {
```

```
    double length, width;
```

```
    Rectangle(double length, double width) {
```

```
        this.length = length;
```

```
        this.width = width;
```

```
    }
```



```

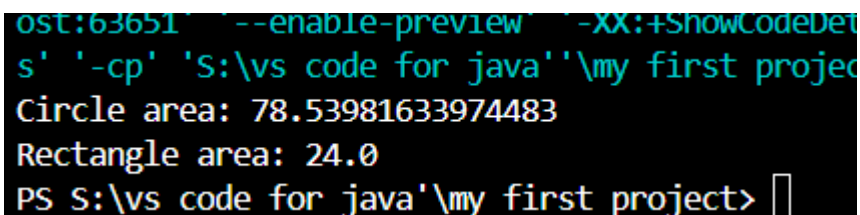
@Override
double area() {
    return length * width;
}
}

public class Main {
    public static void main(String[] args) {
        Shape circle = new Circle(5);
        Shape rectangle = new Rectangle(4, 6);

        System.out.println("Circle area: " + circle.area());
        System.out.println("Rectangle area: " + rectangle.area());
    }
}

```

Output:



```

ost:63651' --enable-preview' -XX:+ShowCodeDet
s' '-cp' 'S:\vs code for java'\my first projec
Circle area: 78.53981633974483
Rectangle area: 24.0
PS S:\vs code for java\my first project>

```

3. Create a base class Vehicle with a method speed(). Create subclasses Car and Bike that override the speed() method.

Code:

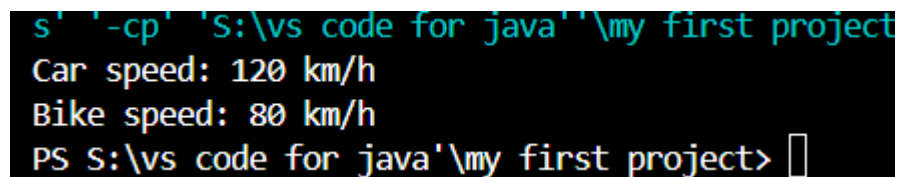
```
class Vehicle {  
    void speed() {  
        System.out.println("Vehicle speed");  
    }  
}
```

```
class Car extends Vehicle {  
    @Override  
    void speed() {  
        System.out.println("Car speed: 120 km/h");  
    }  
}
```

```
class Bike extends Vehicle {  
    @Override  
    void speed() {  
        System.out.println("Bike speed: 80 km/h");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Vehicle myCar = new Car();  
        Vehicle myBike = new Bike();  
  
        myCar.speed();  
        myBike.speed();  
    }  
}
```

Output:



```
s' '-cp' 'S:\vs code for java'\my first project  
Car speed: 120 km/h  
Bike speed: 80 km/h  
PS S:\vs code for java\my first project> █
```

4. Create a base class Employee with a method calculateSalary(). Create subclasses FullTimeEmployee and PartTimeEmployee that override the calculateSalary() method.

Code:

```
class Employee {  
    double calculateSalary() {
```

```
        return 0;
    }
}
```

```
class FullTimeEmployee extends Employee {
    double salary;
```

```
    FullTimeEmployee(double salary) {
        this.salary = salary;
    }
```

```
    @Override
    double calculateSalary() {
        return salary;
    }
}
```

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

```
    PartTimeEmployee(double hourlyRate, int hoursWorked) {
```

```
        this.hourlyRate = hourlyRate;
        this.hoursWorked = hoursWorked;
    }

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

public class Main {
    public static void main(String[] args) {
        Employee fullTime = new FullTimeEmployee(3000);
        Employee partTime = new PartTimeEmployee(20, 100);

        System.out.println("Full-time salary: " +
            fullTime.calculateSalary());

        System.out.println("Part-time salary: " +
            partTime.calculateSalary());
    }
}
```

Output:

```
lsInExceptionMessages' '-cp' 'S:\vs code for  
bin' 'Main'  
Full-time salary: 3000.0  
Part-time salary: 2000.0  
PS S:\vs code for java\my first project>
```

OVERLOADING PROGRAMS:

1. Create a class Calculator with overloaded methods add(). Implement different versions of the add() method to handle different numbers of parameters.

Code:

```
class Calculator {  
    int add(int a, int b) {  
        return a + b;  
    }  
  
    int add(int a, int b, int c) {
```

```

        return a + b + c;
    }

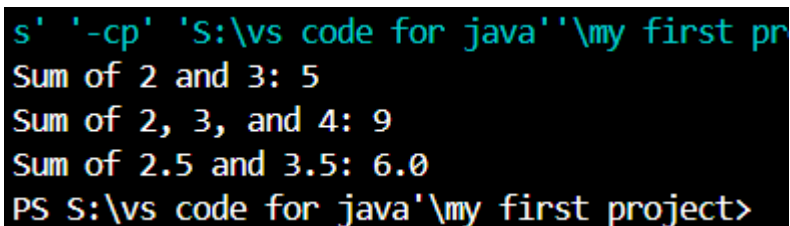
    double add(double a, double b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        System.out.println("Sum of 2 and 3: " + calc.add(2, 3));
        System.out.println("Sum of 2, 3, and 4: " + calc.add(2, 3,
4));
        System.out.println("Sum of 2.5 and 3.5: " + calc.add(2.5,
3.5));
    }
}

```

Output:



```

s' '-cp' 'S:\vs code for java'\my first pr
Sum of 2 and 3: 5
Sum of 2, 3, and 4: 9
Sum of 2.5 and 3.5: 6.0
PS S:\vs code for java\my first project>

```

2. Create a class Printer with overloaded methods print(). Implement different versions of the print() method to handle different data types.

.

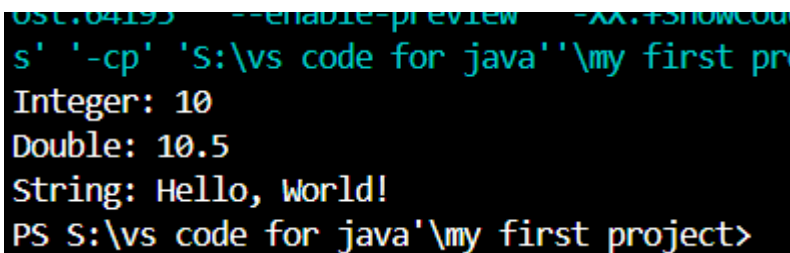
Code:

```
class Printer {  
    void print(int a) {  
        System.out.println("Integer: " + a);  
    }  
  
    void print(double a) {  
        System.out.println("Double: " + a);  
    }  
  
    void print(String a) {  
        System.out.println("String: " + a);  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Printer printer = new Printer();
```



```
    printer.print(10);  
    printer.print(10.5);  
    printer.print("Hello, World!");  
}  
}
```

Output:



```
OST:04193 --enable-preview -XX:+ShowCode  
s' '-cp' 'S:\vs code for java'\my first pr  
Integer: 10  
Double: 10.5  
String: Hello, World!  
PS S:\vs code for java\my first project>
```

3. Create a class Multiplier with overloaded methods multiply(). Implement different versions of the multiply() method to handle different numbers of parameters.

.

Code:

```
class Multiplier {  
    int multiply(int a, int b) {  
        return a * b;  
    }  
}
```

```
int multiply(int a, int b, int c) {  
    return a * b * c;  
}
```

```
double multiply(double a, double b) {  
    return a * b;  
}  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Multiplier multiplier = new Multiplier();  
  
        System.out.println("Product of 2 and 3: " +  
multiplier.multiply(2, 3));  
  
        System.out.println("Product of 2, 3, and 4: " +  
multiplier.multiply(2, 3, 4));  
  
        System.out.println("Product of 2.5 and 3.5: " +  
multiplier.multiply(2.5, 3.5));  
    }  
}
```

Output:

```
s' -cp 'S:\vs code for java'\my first pro
Product of 2 and 3: 6
Product of 2, 3, and 4: 24
Product of 2.5 and 3.5: 8.75
PS S:\vs code for java\my first project>
```

4. Create a class MaxFinder with overloaded methods findMax(). Implement different versions of the findMax() method to handle different data types.

.

Code:

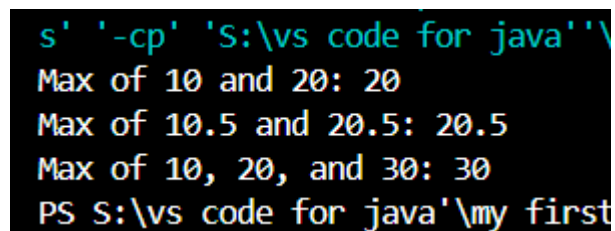
```
class MaxFinder {
    int findMax(int a, int b) {
        return (a > b) ? a : b;
    }

    double findMax(double a, double b) {
        return (a > b) ? a : b;
    }

    int findMax(int a, int b, int c) {
        return Math.max(a, Math.max(b, c));
    }
}
```

```
public class Main {  
    public static void main(String[] args) {  
        MaxFinder maxFinder = new MaxFinder();  
  
        System.out.println("Max of 10 and 20: " +  
maxFinder.findMax(10, 20));  
  
        System.out.println("Max of 10.5 and 20.5: " +  
maxFinder.findMax(10.5, 20.5));  
  
        System.out.println("Max of 10, 20, and 30: " +  
maxFinder.findMax(10, 20, 30));  
    }  
}
```

Output:



```
s' '-cp' 'S:\vs code for java'\  
Max of 10 and 20: 20  
Max of 10.5 and 20.5: 20.5  
Max of 10, 20, and 30: 30  
PS S:\vs code for java\my first
```

ABSTRACTION PROGRAMS:

Abstraction:

1)**Question:** Create an abstract class Shape with an abstract method area(). Create subclasses Circle and Rectangle that implement the area() method.

Code:

```
abstract class Shape {  
    abstract double area();  
}  
  
class Circle extends Shape {  
    double radius;  
  
    Circle(double radius) {  
        this.radius = radius;  
    }  
  
    @Override  
    double area() {  
        return Math.PI * radius * radius;  
    }  
}
```

```
}  
}  
  
class Rectangle extends Shape {  
    double length, width;  
  
    Rectangle(double length, double width) {  
        this.length = length;  
        this.width = width;  
    }  
  
    @Override  
    double area() {  
        return length * width;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Shape circle = new Circle(5);  
        Shape rectangle = new Rectangle(4, 6);  
    }  
}
```

```
        System.out.println("Circle area: " + circle.area());  
        System.out.println("Rectangle area: " + rectangle.area());  
    }  
}
```

Output:

```
-cp' 'S:\vs code for java'\my first project\bi  
Circle area: 78.53981633974483  
Rectangle area: 24.0  
PS S:\vs code for java'\my first project>
```

2)**Question:** Create an abstract class Employee with an abstract method calculateSalary(). Create subclasses FullTimeEmployee and PartTimeEmployee tht implement the calculateSalary() method.

Code:

```
abstract class Employee {  
    abstract double calculateSalary();  
}  
  
class FullTimeEmployee extends Employee {  
    double salary;  
  
    FullTimeEmployee(double salary) {  
        this.salary = salary;
```

```
}
```

```
@Override
```

```
double calculateSalary() {
```

```
    return salary;
```

```
}
```

```
}
```

```
class PartTimeEmployee extends Employee {
```

```
    double hourlyRate;
```

```
    int hoursWorked;
```

```
    PartTimeEmployee(double hourlyRate, int hoursWorked) {
```

```
        this.hourlyRate = hourlyRate;
```

```
        this.hoursWorked = hoursWorked;
```

```
}
```

```
@Override
```

```
double calculateSalary() {
```

```
    return hourlyRate * hoursWorked;
```

```
}
```

```
}
```



```

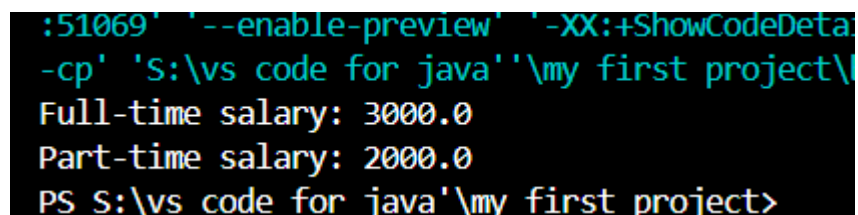
public class Main {
    public static void main(String[] args) {
        Employee fullTime = new FullTimeEmployee(3000);
        Employee partTime = new PartTimeEmployee(20, 100);

        System.out.println("Full-time salary: " +
fullTime.calculateSalary());

        System.out.println("Part-time salary: " +
partTime.calculateSalary());
    }
}

```

Output:



```

:51069' '--enable-preview' '-XX:+ShowCodeDeta
-cp' 'S:\vs code for java'\my first project\
Full-time salary: 3000.0
Part-time salary: 2000.0
PS S:\vs code for java'\my first project>

```

3)Question: Create an abstract class Vehicle with an abstract method calculateFuelEfficiency(). Create subclasses Car and Truck that implement the calculateFuelEfficiency() method.

Code:

```

abstract class Vehicle {

```

```
    abstract double calculateFuelEfficiency(double distance,  
    double fuelUsed);  
}
```

```
class Car extends Vehicle {  
    @Override  
    double calculateFuelEfficiency(double distance, double  
    fuelUsed) {  
        return distance / fuelUsed; // km/l  
    }  
}
```

```
class Truck extends Vehicle {  
    @Override  
    double calculateFuelEfficiency(double distance, double  
    fuelUsed) {  
        return distance / fuelUsed; // km/l  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Vehicle myCar = new Car();  
    }  
}
```

```
Vehicle myTruck = new Truck();
```

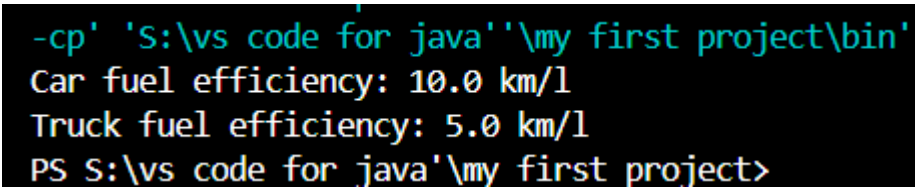
```
System.out.println("Car fuel efficiency: " +  
myCar.calculateFuelEfficiency(500, 50) + " km/l");
```

```
System.out.println("Truck fuel efficiency: " +  
myTruck.calculateFuelEfficiency(300, 60) + " km/l");
```

```
}
```

```
}
```

Output:



```
-cp' 'S:\vs code for java'\my first project\bin'  
Car fuel efficiency: 10.0 km/l  
Truck fuel efficiency: 5.0 km/l  
PS S:\vs code for java\my first project>
```

4)Question: Create an abstract class BankAccount with an abstract method calculateInterest(). Create subclasses SavingsAccount and CurrentAccount that implement the calculateInterest() method.

Code:

```
abstract class BankAccount {
```

```
    double balance;
```

```
    BankAccount(double balance) {
```

```
        this.balance = balance;
```

```
    }
```

```
    abstract double calculateInterest();
}

class SavingsAccount extends BankAccount {
    double interestRate;

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

    @Override
    double calculateInterest() {
        return balance * interestRate / 100;
    }
}

class CurrentAccount extends BankAccount {
    double overdraftLimit;

    CurrentAccount(double balance, double overdraftLimit) {
```

```

        super(balance);
        this.overdraftLimit = overdraftLimit;
    }

    @Override
    double calculateInterest() {
        return 0; // No interest for current accounts
    }
}

public class Main {
    public static void main(String[] args) {
        BankAccount savings = new SavingsAccount(1000, 5);
        BankAccount current = new CurrentAccount(2000, 500);

        System.out.println("Savings account interest: " +
            savings.calculateInterest());

        System.out.println("Current account interest: " +
            current.calculateInterest());
    }
}

```

Output:

```
agentlib:jdwp=transport=dt_socket,server=n,susp
:51223' '--enable-preview' '-XX:+ShowCodeDetail
-cp' 'S:\vs code for java'\my first project\bi
Savings account interest: 50.0
Current account interest: 0.0
PS S:\vs code for java'\my first project>
```

INTERFACE PROGRAMS:

1)Question: Create an interface **Calculable** with methods **add()** and **subtract()**. Implement this interface in a class **Calculator**.

Code:

```
interface Calculable {
    double add(double a, double b);
    double subtract(double a, double b);
}

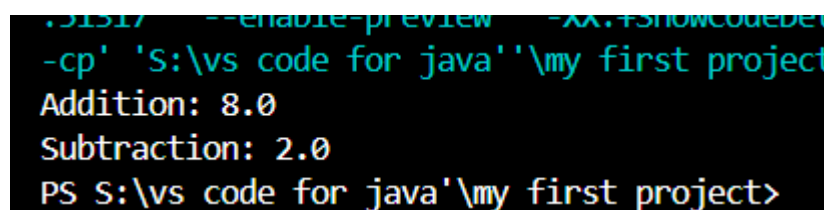
class Calculator implements Calculable {
    @Override
    public double add(double a, double b) {
        return a + b;
    }
}
```

@Override

```
public double subtract(double a, double b) {  
    return a - b;  
}  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Calculator calc = new Calculator();  
  
        System.out.println("Addition: " + calc.add(5, 3));  
        System.out.println("Subtraction: " + calc.subtract(5, 3));  
    }  
}
```

Output:



```
.JIS17 --enable-preview -XX:+ShowCodeDetails  
-cp' 'S:\vs code for java'\my first project  
Addition: 8.0  
Subtraction: 2.0  
PS S:\vs code for java\my first project>
```

2)Question: Create an interface Drawable with a method draw(). Implement this interface in classes Circle and Rectangle.

Code:

```
interface Drawable {  
    void draw();  
}
```

```
class Circle implements Drawable {  
    @Override  
    public void draw() {  
        System.out.println("Drawing a Circle");  
    }  
}
```

```
class Rectangle implements Drawable {  
    @Override  
    public void draw() {  
        System.out.println("Drawing a Rectangle");  
    }  
}
```

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



```

    Drawable circle = new Circle();

    Drawable rectangle = new Rectangle();

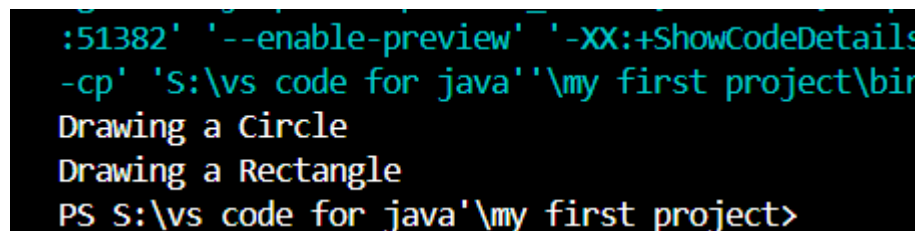
    circle.draw();

    rectangle.draw();

}
}

```

Output:



```

:51382' '--enable-preview' '-XX:+ShowCodeDetails
-cp' 'S:\vs code for java'\my first project\bin
Drawing a Circle
Drawing a Rectangle
PS S:\vs code for java\my first project>

```

3)Question: Create an interface Payable with a method calculatePayment(). Implement this interface in classes Contractor and Employee.

Code:

```

interface Payable {

    double calculatePayment();

}

class Contractor implements Payable {

    double hourlyRate;

    int hoursWorked;

```

```
Contractor(double hourlyRate, int hoursWorked) {  
    this.hourlyRate = hourlyRate;  
    this.hoursWorked = hoursWorked;  
}
```

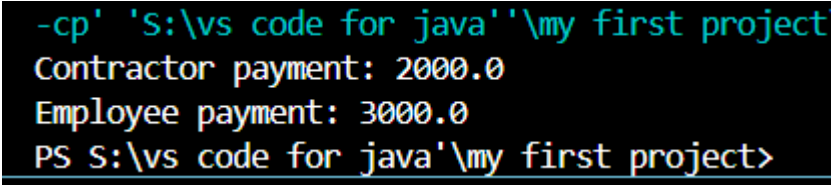
```
@Override  
public double calculatePayment() {  
    return hourlyRate * hoursWorked;  
}  
}
```

```
class Employee implements Payable {  
    double salary;  
  
    Employee(double salary) {  
        this.salary = salary;  
    }  
}
```

```
@Override  
public double calculatePayment() {  
    return salary;  
}
```

```
}  
  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Payable contractor = new Contractor(50, 40);  
        Payable employee = new Employee(3000);  
  
        System.out.println("Contractor payment: " +  
contractor.calculatePayment());  
  
        System.out.println("Employee payment: " +  
employee.calculatePayment());  
    }  
}
```

Output:



```
-cp' 'S:\vs code for java'\my first project  
Contractor payment: 2000.0  
Employee payment: 3000.0  
PS S:\vs code for java'\my first project>
```

4)Question: Create an interface Measurable with methods `getArea()` and `getPerimeter()`. Implement this interface in classes `Square` and `Triangle`.

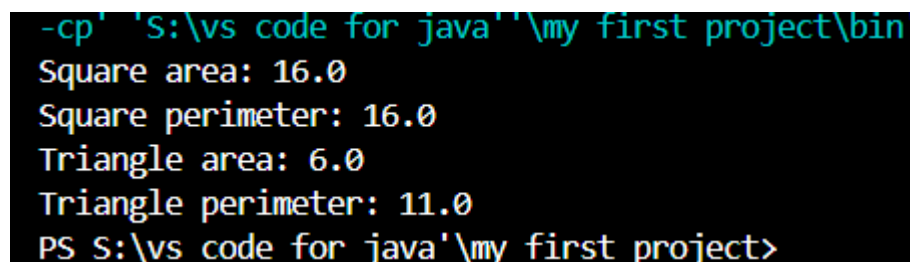
Code:

```
interface Measurable {  
    double getArea();  
    double getPerimeter();  
}  
  
class Square implements Measurable {  
    double side;  
  
    Square(double side) {  
        this.side = side;  
    }  
  
    @Override  
    public double getArea() {  
        return side * side;  
    }  
  
    @Override  
    public double getPerimeter() {  
        return 4 * side;  
    }  
}
```

```
class Triangle implements Measurable {  
    double base, height, side1, side2;  
  
    Triangle(double base, double height, double side1, double  
side2) {  
        this.base = base;  
        this.height = height;  
        this.side1 = side1;  
        this.side2 = side2;  
    }  
  
    @Override  
    public double getArea() {  
        return 0.5 * base * height;  
    }  
  
    @Override  
    public double getPerimeter() {  
        return base + side1 + side2;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Measurable square = new Square(4);  
        Measurable triangle = new Triangle(3, 4, 3, 5);  
  
        System.out.println("Square area: " + square.getArea());  
        System.out.println("Square perimeter: " +  
square.getPerimeter());  
        System.out.println("Triangle area: " + triangle.getArea());  
        System.out.println("Triangle perimeter: " +  
triangle.getPerimeter());  
    }  
}
```

Output:



```
-cp' 'S:\vs code for java'\my first project\bin  
Square area: 16.0  
Square perimeter: 16.0  
Triangle area: 6.0  
Triangle perimeter: 11.0  
PS S:\vs code for java\my first project>
```

ENCAPSULATION PROGRAMS:

Encapsulation:

1)**Question:** Create a class BankAccount that encapsulates the account balance. Provide methods to deposit and withdraw money, ensuring that the balance cannot be negative.

Code:

```
class BankAccount {  
    private double balance;  
  
    public BankAccount(double initialBalance) {  
        if (initialBalance >= 0) {  
            this.balance = initialBalance;  
        } else {  
            this.balance = 0;  
            System.out.println("Initial balance cannot be negative.  
Setting balance to 0.");  
        }  
    }  
  
    public void deposit(double amount) {  
        if (amount > 0) {  
            balance += amount;  
            System.out.println("Deposited: " + amount);  
        } else {
```

```
        System.out.println("Deposit amount must be  
positive.");
```

```
    }  
}
```

```
public void withdraw(double amount) {  
    if (amount > 0 && amount <= balance) {  
        balance -= amount;  
        System.out.println("Withdrawn: " + amount);  
    } else {  
        System.out.println("Invalid withdrawal amount.");  
    }  
}
```

```
public double getBalance() {  
    return balance;  
}  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        BankAccount account = new BankAccount(1000);  
        account.deposit(500);
```



```
        account.withdraw(200);

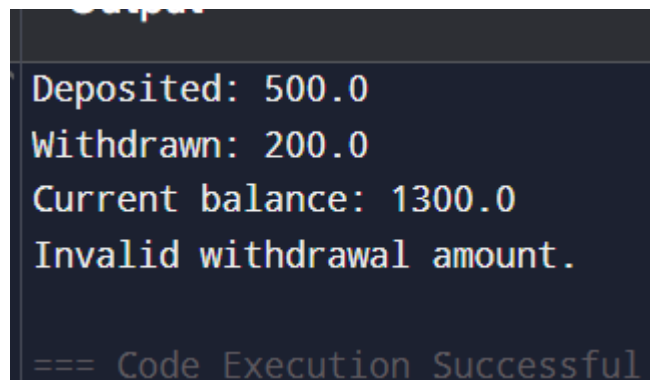
        System.out.println("Current balance: " +
account.getBalance());

        account.withdraw(1500);

    }

}
```

Output:



```
Deposited: 500.0
Withdrawn: 200.0
Current balance: 1300.0
Invalid withdrawal amount.

=== Code Execution Successful
```

2)**Question:** Create a class Student that encapsulates the student's name and grade. Provide methods to set and get the name and grade, ensuring that the grade is between 0 and 100.

Code:

```
class Student {

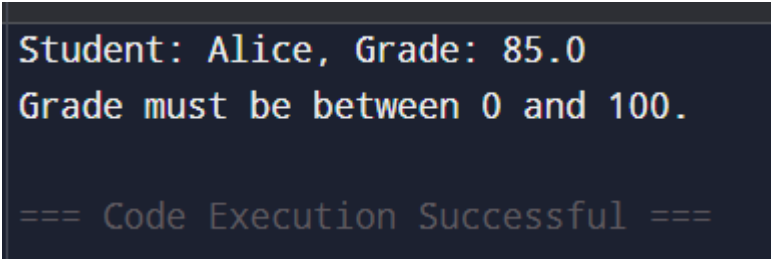
    private String name;

    private double grade;
```

```
public Student(String name) {  
    this.name = name;  
    this.grade = 0; // Default grade  
}  
  
public String getName() {  
    return name;  
}  
  
public double getGrade() {  
    return grade;  
}  
  
public void setGrade(double grade) {  
    if (grade >= 0 && grade <= 100) {  
        this.grade = grade;  
    } else {  
        System.out.println("Grade must be between 0 and  
100.");  
    }  
}  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Student student = new Student("Alice");  
        student.setGrade(85);  
        System.out.println("Student: " + student.getName() + ",  
Grade: " + student.getGrade());  
        student.setGrade(110); // Invalid grade  
    }  
}
```

Output:

A screenshot of a code execution environment with a dark background. It shows the output of the Java program: "Student: Alice, Grade: 85.0" followed by "Grade must be between 0 and 100." on the next line. At the bottom, it says "=== Code Execution Successful ===".

```
Student: Alice, Grade: 85.0  
Grade must be between 0 and 100.  
  
=== Code Execution Successful ===
```

3)**Question:** Create a class Product that encapsulates the product's name, price, and quantity. Provide methods to set and get these values, ensuring that the price and quantity are non-negative.

Code:

```
class Product {  
    private String name;  
    private double price;  
    private int quantity;
```

```
public Product(String name, double price, int quantity) {  
    this.name = name;  
    setPrice(price);  
    setQuantity(quantity);  
}
```

```
public String getName() {  
    return name;  
}
```

```
public double getPrice() {  
    return price;  
}
```

```
public int getQuantity() {  
    return quantity;  
}
```

```
public void setPrice(double price) {  
    if (price >= 0) {  
        this.price = price;  
    }  
}
```

```
        } else {  
            System.out.println("Price cannot be negative.");  
        }  
    }  
}  
  
public void setQuantity(int quantity) {  
    if (quantity >= 0) {  
        this.quantity = quantity;  
    } else {  
        System.out.println("Quantity cannot be negative.");  
    }  
}  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Product product = new Product("Laptop", 1200.00, 10);  
        System.out.println("Product: " + product.getName() + ",  
Price: " + product.getPrice() + ", Quantity: " +  
product.getQuantity());  
        product.setPrice(-500); // Invalid price  
        product.setQuantity(5);  
    }  
}
```

```
        System.out.println("Updated Quantity: " +  
product.getQuantity());  
    }  
}
```

Output:

```
Product: Laptop, Price: 1200.0, Quantity: 10  
Price cannot be negative.  
Updated Quantity: 5  
  
=== Code Execution Successful ===
```

4)**Question:** Create a class Employee that encapsulates the employee's ID, name, and salary. Provide methods to set and get these values, ensuring that the salary is non-negative.

Code:

```
class Employee {  
    private int id;  
    private String name;  
    private double salary;  
  
    public Employee(int id, String name) {  
        this.id = id;  
        this.name = name;
```

```
        this.salary = 0; // Default salary
    }
```

```
    public int getId() {
        return id;
    }
```

```
    public String getName() {
        return name;
    }
```

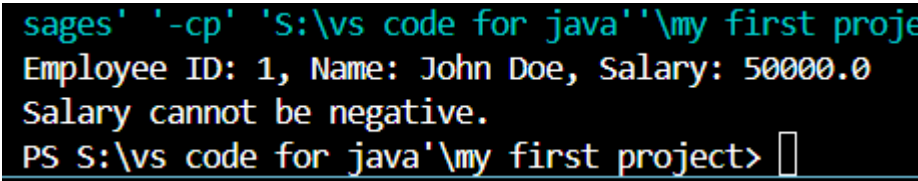
```
    public double getSalary() {
        return salary;
    }
```

```
    public void setSalary(double salary) {
        if (salary >= 0) {
            this.salary = salary;
        } else {
            System.out.println("Salary cannot be negative.");
        }
    }
```

```
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Employee employee = new Employee(1, "John Doe");  
        employee.setSalary(50000);  
        System.out.println("Employee ID: " + employee.getId() +  
            ", Name: " + employee.getName() + ", Salary: " +  
            employee.getSalary());  
        employee.setSalary(-1000); // Invalid salary  
    }  
}
```

Output:



```
sages' '-cp' 'S:\vs code for java'\my first proje  
Employee ID: 1, Name: John Doe, Salary: 50000.0  
Salary cannot be negative.  
PS S:\vs code for java\my first project> |
```

PROGRAMS ON PACKAGES:

BUILT-IN PACKAGES:

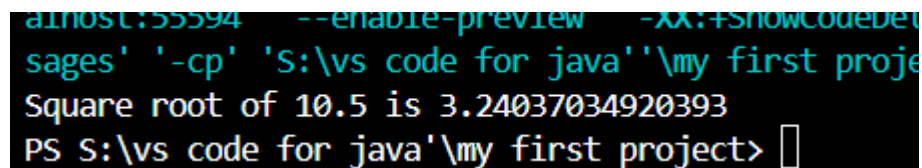
1) 1: Using the Math class

Code:

```
import java.lang.Math;

public class Main {
    public static void main(String[] args) {
        double num = 10.5;
        double sqrt = Math.sqrt(num);
        System.out.println("Square root of " + num + " is " +
sqrt);
    }
}
```

Output:



```
almost:55594 --enable-preview -XX:+ShowCodeDetails
sages' '-cp' 'S:\vs code for java'\my first proje
Square root of 10.5 is 3.24037034920393
PS S:\vs code for java\my first project> █
```

2) 2: Using the String class

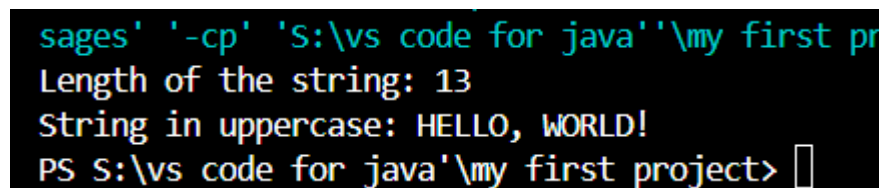
Code:

```
import java.lang.String;

public class Main {
```

```
public static void main(String[] args) {  
    String str = "Hello, World!";  
    System.out.println("Length of the string: " + str.length());  
    System.out.println("String in uppercase: " +  
str.toUpperCase());  
}  
}
```

Output:



```
sages' '-cp' 'S:\vs code for java'\my first pr  
Length of the string: 13  
String in uppercase: HELLO, WORLD!  
PS S:\vs code for java\my first project> █
```

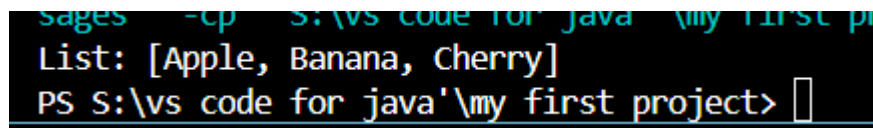
3) 3: Using the ArrayList class

Code:

```
import java.util.ArrayList;  
  
public class Main {  
    public static void main(String[] args) {  
        ArrayList<String> list = new ArrayList<>();  
        list.add("Apple");  
        list.add("Banana");  
        list.add("Cherry");  
        System.out.println("List: " + list);  
    }  
}
```

```
}  
}
```

Output:



```
sages' '-cp' 'S:\vs code for java'\my first pr  
List: [Apple, Banana, Cherry]  
PS S:\vs code for java'\my first project> |
```

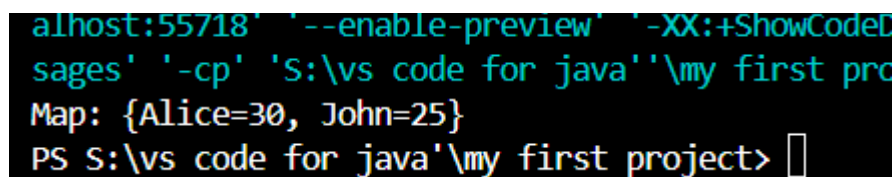
4) 4: Using the HashMap class

Code:

```
import java.util.HashMap;
```

```
public class Main {  
    public static void main(String[] args) {  
        HashMap<String, Integer> map = new HashMap<>();  
        map.put("John", 25);  
        map.put("Alice", 30);  
        System.out.println("Map: " + map);  
    }  
}
```

Output:



```
alhost:55718' '--enable-preview' '-XX:+ShowCodeD  
sages' '-cp' 'S:\vs code for java'\my first pro  
Map: {Alice=30, John=25}  
PS S:\vs code for java'\my first project> |
```

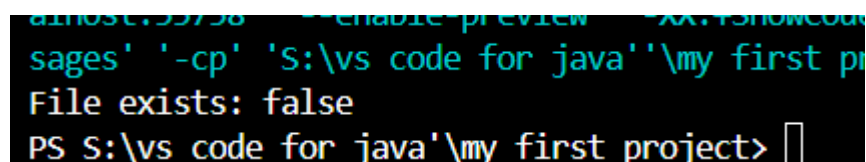
5)5: Using the File class

Code:

```
import java.io.File;

public class Main {
    public static void main(String[] args) {
        File file = new File("example.txt");
        System.out.println("File exists: " + file.exists());
    }
}
```

Output:



```
almost.55758 --enable-preview --xx:showcode
sages' '-cp' 'S:\vs code for java'\my first pr
File exists: false
PS S:\vs code for java\my first project> |
```

7)7. Networking Program (Using java.net)

Code:

```
import java.net.HttpURLConnection;
import java.net.URL;

public class SimpleHttpClient {
```

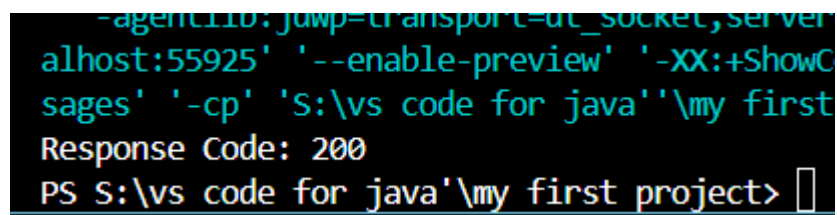
```

public static void main(String[] args) {
    try {
        URL url = new
URL("https://jsonplaceholder.typicode.com/posts/1");
        HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
        connection.setRequestMethod("GET");

        int responseCode = connection.getResponseCode();
        System.out.println("Response Code: " +
responseCode);
    } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
    }
}
}

```

Output:



```

-agent110:jump-transport=dt_socket,server
alhost:55925' '--enable-preview' '-XX:+ShowC
sages' '-cp' 'S:\vs code for java'\my first
Response Code: 200
PS S:\vs code for java'\my first project> 

```

8)8. SQL Program (Using java.sql)

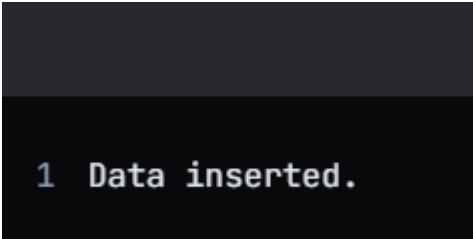
Code:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class SimpleDatabaseExample {
    public static void main(String[] args) {
        try (Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/t
estdb", "your_username", "your_password")) {
            String insertSQL = "INSERT INTO Users (name) VALUES
('Alice')";
            try (PreparedStatement insertStatement =
connection.prepareStatement(insertSQL)) {
                insertStatement.executeUpdate();
                System.out.println("Data inserted.");
            }
        } catch (Exception e) {
            System.out.println("Database error: " +
e.getMessage());
        }
    }
}
```

```
}
```

Output:



```
1 Data inserted.
```

USER-DEFINED PACKAGES:

- 1) **1:** You are tasked with creating a simple shape calculator in Java that can compute the area and perimeter of different shapes, specifically circles and rectangles. The functionality is encapsulated in a user-defined package named shapes. The package contains a class called shapecalculator, which provides methods to calculate the area and perimeter of the specified shapes. The main program, located in a separate file, interacts with the user to gather input and display results.

Code:

shapecalculator.java

```
package shapes;
```

```
public class shapecalculator {  
    private double radius;  
    private double length;  
    private double width;  
  
    public shapecalculator(double radius) {  
        this.radius = radius;  
    }  
  
    public shapecalculator(double length, double width) {  
        this.length = length;  
        this.width = width;  
    }  
  
    public double circlearea() {  
        return 3.14 * radius * radius;  
    }  
  
    public double circleperimeter() {  
        return 2 * 3.14 * radius;  
    }  
}
```



```
public double rectanglearea() {  
    return length * width;  
}  
  
public double rectangleperimeter() {  
    return 2 * (length + width);  
}  
}
```

Main.java

```
import shapes.shapecalculator;  
import java.util.*;  
  
public class Main {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        int choice;  
        do {  
            System.out.println("Choose a shape to calculate");  
            System.out.println("1. Circle");  
            System.out.println("2. Rectangle");  
            System.out.println("3. Exit");  
        } while (choice != 3);  
    }  
}
```

```
System.out.print("Enter a choice between (1 - 3): ");
choice = sc.nextInt();
switch (choice) {
    case 1:
        System.out.print("Enter the radius of the circle: ");
        double circleradius = sc.nextDouble();
        shapecalculator ci = new
shapecalculator(circleradius);

        System.out.printf("Circle Area: %.2f\n",
ci.circlearea()); // Added format specifier

        System.out.printf("Circle Perimeter: %.2f\n",
ci.circleperimeter()); // Added format specifier

        break;

    case 2:
        System.out.print("Enter the length of the
rectangle: ");
        double rectanglelength = sc.nextDouble();
        System.out.print("Enter the width of the
rectangle: ");
        double rectanglewidth = sc.nextDouble();
        shapecalculator re = new
shapecalculator(rectanglelength, rectanglewidth);
```

```
        System.out.printf("Rectangle Area: %.2f\n",
re.rectanglearea()); // Added format specifier

        System.out.printf("Rectangle Perimeter: %.2f\n",
re.rectangleperimeter()); // Added format specifier

        break;

    case 3:

        System.out.println("Exiting the program.");
        break;

    default:

        System.out.println("Invalid choice. Please choose a
number between 1 and 3."); // Updated to match choices

        break;

    }

    System.out.println();
} while (choice != 3);

    sc.close(); // Close the scanner to avoid resource leaks
}
}
```

Output:

```

PS C:\Users\Sahil\Desktop\ShapeCalculatorProject> javac -d . Main.java shape
s/shapecalculator.java
PS C:\Users\Sahil\Desktop\ShapeCalculatorProject> java Main
Choose a shape to calculate
1. Circle
2. Rectangle
3. Exit
Enter a choice between (1 - 3): 1
Enter the radius of the circle: 3
Circle Area: 28.26
Circle Perimeter: 18.84

Choose a shape to calculate
1. Circle
2. Rectangle
3. Exit
Enter a choice between (1 - 3): 2
Enter the length of the rectangle: 3
Enter the width of the rectangle: 4
Rectangle Area: 12.00
Rectangle Perimeter: 14.00

Choose a shape to calculate
1. Circle
2. Rectangle
3. Exit
Enter a choice between (1 - 3): 3
Exiting the program.

PS C:\Users\Sahil\Desktop\ShapeCalculatorProject> |

```

1) 1: Using the Math class

Code:

shapes.java

```
package geometry;
```

```
// Abstract class to define the structure for shapes
```

```
abstract class Shape {
    abstract double area();
    abstract double perimeter();
}
```

```
// Class for Square that extends Shape
class Square extends Shape {
    private double side;

    // Constructor
    public Square(double side) {
        this.side = side;
    }

    // Calculate area of the square
    @Override
    double area() {
        return side * side;
    }

    // Calculate perimeter of the square
    @Override
    double perimeter() {
        return 4 * side;
    }
}
```

```
// Class for Triangle that extends Shape
class Triangle extends Shape {
    private double base;
    private double height;
    private double side1;
    private double side2;
    private double side3;

    // Constructor
    public Triangle(double base, double height, double side1,
double side2, double side3) {
        this.base = base;
        this.height = height;
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    // Calculate area of the triangle
    @Override
    double area() {
        return 0.5 * base * height;
    }
}
```

```
// Calculate perimeter of the triangle
@Override
double perimeter() {
    return side1 + side2 + side3;
}
}
```

Main.java

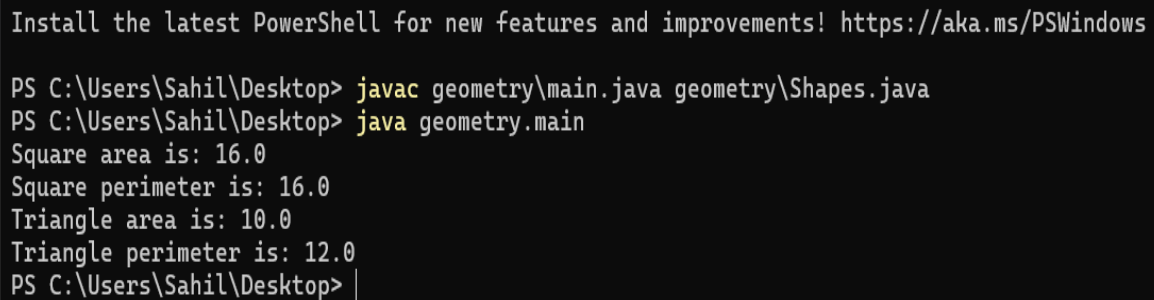
```
package geometry;

// Importing the Shape class and its subclasses
import geometry.Shape;
import geometry.Square;
import geometry.Triangle;

public class main {
    public static void main(String[] args) {
        // Create a square with side length 4
        Shape square = new Square(4);
        System.out.println("Square area is: " + square.area());
        System.out.println("Square perimeter is: " +
square.perimeter());
    }
}
```

```
// Create a triangle with base 5, height 4, and sides 3, 4, 5
Shape triangle = new Triangle(5, 4, 3, 4, 5);
System.out.println("Triangle area is: " + triangle.area());
System.out.println("Triangle perimeter is: " +
triangle.perimeter());
}
}
```

Output:



```
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\Users\Sahil\Desktop> javac geometry\main.java geometry\Shapes.java
PS C:\Users\Sahil\Desktop> java geometry.main
Square area is: 16.0
Square perimeter is: 16.0
Triangle area is: 10.0
Triangle perimeter is: 12.0
PS C:\Users\Sahil\Desktop> |
```

PROGRAMS ON FILE HANDLING AND EXCEPTION HANDLING(COMBINED):

1) Creating a file

```
import java.io.*;

class file {

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

        File f = new File("S:\\vs code for java\\my first
project\\lc.txt");

        if (f.createNewFile()) {

            System.out.println("File successfully created.");

        } else {

            System.out.println("File already exists.");

        }

    }

}
```

// Output:

// File successfully created. (if the file didn't exist)

// OR

// File already exists. (if the file already existed)

2) Different functions of a file

```

class file{
    public static void main(String[] args) throws IOException{
        File f = new File("S:\\vs code for java\\my first
project\\lc.txt");
        if(f.exists()){
            System.out.println("file name." + f.getName());
            System.out.println("file Location." +
f.getAbsolutePath());
            System.out.println("file Writable." + f.canWrite());
            System.out.println("file Readable." + f.canRead());
            System.out.println("file size." + f.length());
            System.out.println("file removed." + f.delete());
        } else {
            System.out.println("file does not exist");
        }
    }
}

```

// Output:

```

// file name.lc.txt
// file Location.S:\vs code for java\my first project\lc.txt
// file Writable.true (or false depending on permissions)
// file Readable.true (or false depending on permissions)
// file size.0 (or the size of the file in bytes)

```

// file removed.true (or false if deletion failed)

// OR

// file does not exist (if the file doesn't exist)

3) Write a file

```
import java.io.*;
```

```
class FileWriterExample {
```

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

```
        try {
```

```
            File f = new File("S:\\vs code for java\\my first  
project\\lc.txt");
```

```
            FileWriter writer = new FileWriter(f);
```

```
            try {
```

```
                writer.write("java programming is the best  
language.");
```

```
            } finally {
```

```
                writer.close();
```

```
            }
```

```
            System.out.println("Successfully wrote data to the  
file.");
```

```
        } catch (IOException i) {
```

```
        System.out.println(i);
    }
}
}
```

// Output:

// Successfully wrote data to the file.

// OR

// (IOException details if an error occurred)

4) Read data from a file

```
import java.io.*;
```

```
class FileReaderExample {
```

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

```
        try {
```

```
            FileReader r = new
            FileReader("C:\\Users\\lenovo\\Desktop\\LC.txt");
```

```
            try {
```

```
                int i;
```

```
        while ((i = r.read()) != -1) {  
            System.out.print((char) i);  
        }  
    } finally {  
        r.close();  
        System.out.println("file closed");  
    }  
  
    } catch (IOException e) {  
        System.out.println("Exception Handled...!");  
    }  
}  
}
```

// Output:

// (Contents of the file LC.txt)

// file closed

// OR

// Exception Handled...! (if an error occurred)

5) Renaming a file

```
import java.io.*;
```

```
class renamefile{  
    public static void main(String[] args){  
        File f = new File("C:\\Users\\lenovo\\Desktop\\LC.txt");  
        File r = new File("C:\\Users\\lenovo\\Desktop\\FC.txt");  
        if(f.exists()){  
            System.out.println(f.renameTo(r));  
        } else {  
            System.out.println("File does not exist");  
        }  
    }  
}
```

// Output:

// true (if the file was successfully renamed)

// OR

// false (if renaming failed)

// OR

// File does not exist (if the original file didn't exist)

6) Copying data from one file to another file

```
import java.io.*;
```

```
class copydata{  
    public static void main(String[] args){  
        FileInputStream r = new  
FileInputStream("C:\\Users\\lenovo\\Desktop\\LC.txt");  
        FileOutputStream w = new  
FileOutputStream("C:\\Users\\lenovo\\Desktop\\FC.txt");  
  
        int i;  
        while((i = r.read()) != -1) {  
            w.write((char) i);  
        }  
        System.out.println("Data copied succesfully");  
    }  
}
```

// Output:

// Data copied succesfully

CH.SC.U4CSE24039

SAHIL PAREEK