



**SCHOOL OF
COMPUTING**

LAB RECORD

23CSE111- Object Oriented Programming

Submitted by

CH.SC.U4CSE24142 - Geetesh P

**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.U4CSE24142 – Geetesh P** 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 8/4/2025

Internal Examiner 1

Internal Examiner 2

INDEX

S.NO	TITLE	PAGE.NO
	UML DIAGRAM	
1.	TITLE OF UML DIAGRAM -1	
	1.a) Use Case Diagram	4
	1.b) Class Diagram	5
	1.c) Sequence Diagram	6
	1.d) State Diagram	7
	1.e) Activity Diagram	8
2.	TITLE OF UML DIAGRAM -2	
	2.a) Use Case Diagram	9
	2.b) Class Diagram	10
	2.c) Sequence Diagram	11
	2.d) State Diagram	12
	2.e) Activity Diagram	13
3.	BASIC JAVA PROGRAMS	
	3.a) Even Or Odd	14
	3.b) Factorial	15
	3.c) Number Guess	16
	3.d) Palindrome Check	17
	3.e) Print Numbers Check	18
	3.f) Reverse Number	19
	3.g) Reverse String	20
	3.h) Sum of Digits	21
	3.i) Table Printer	22
	3.j) Vowel Counter	23
	INHERITANCE	
4.	SINGLE INHERITANCE PROGRAMS	
	4.a) Employee	24

	4.b) Vehicle	25
5.	MULTILEVEL INHERITANCE PROGRAMS	
	5.a) Speaker Writer	26
	5.b) Student Employee	27
6.	HIERARCHICAL INHERITANCE PROGRAMS	
	6.a) Animal	28
	6.b) Product	29
7.	HYBRID INHERITANCE PROGRAMS	
	7.a) Chef Waiter	30
	7.b) Printer Scanner	31
	POLYMORPHISM	
8.	CONSTRUCTOR PROGRAMS	
	8.a) Laptop	32
9.	CONSTRUCTOR OVERLOADING PROGRAMS	
	9.a) Employee	33
10.	METHOD OVERLOADING PROGRAMS	
	10.a) Calculator	34
	10.b) Display	35
11.	METHOD OVERRIDING PROGRAMS	
	11.a) Shape	36
	11.b) Vehicle	37
	ABSTRACTION	
12.	INTERFACE PROGRAMS	
	12.a) Animal	38
	12.b) Bank	38
	12.c) Multiple	39
	12.d) Printable	39
13.	ABSTRACT CLASS PROGRAMS	
	13.a) Animal	40
	13.b) Employee	40
	13.c) Shape	41
	13.d) Vehicle	41
	ENCAPSULATION	
14.	ENCAPSULATION PROGRAMS	
	14.a) Bank Account	42
	14.c) Car	42
	14.d) Employee	43
	14.g) Person	43

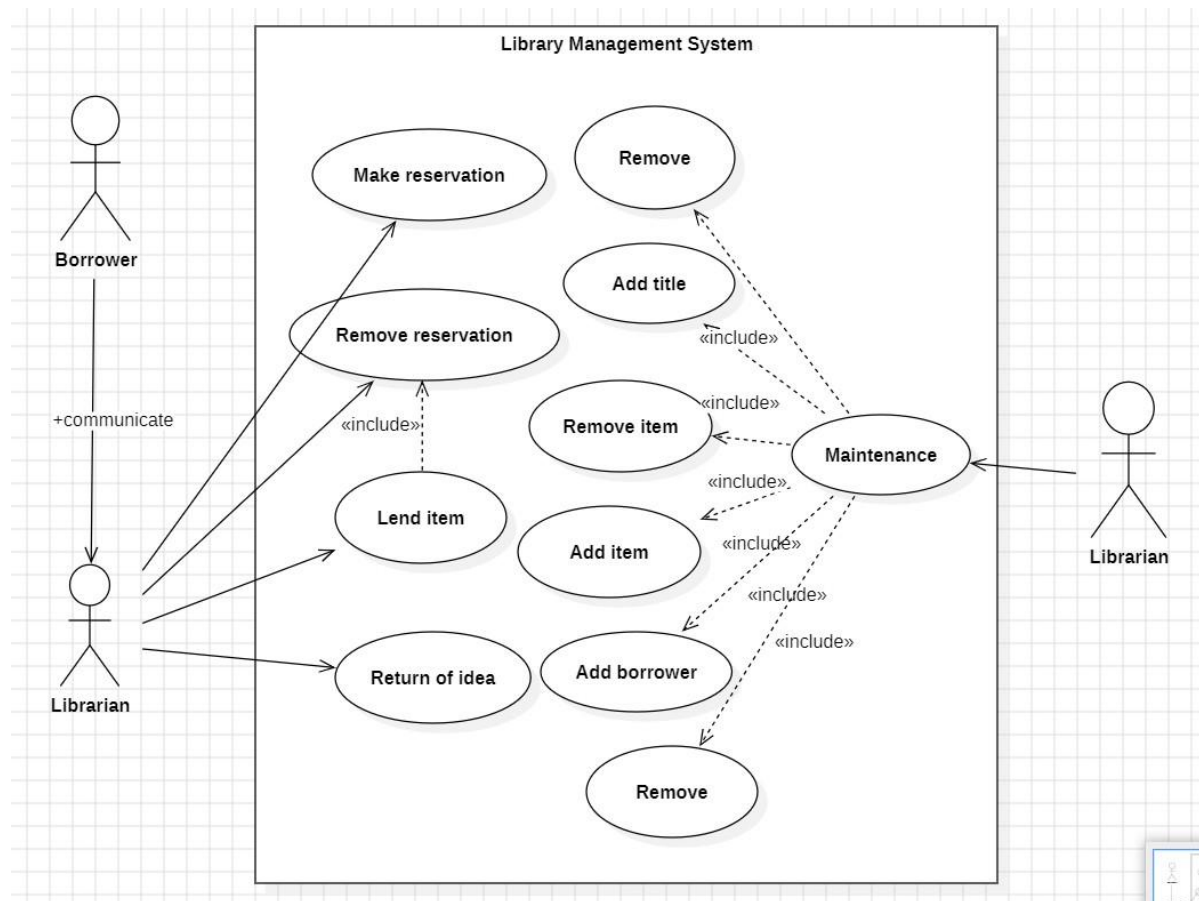
15.	PACKAGES PROGRAMS	
	15.a) B1	44
	15.b) B2	44
	15.c) Banking	45
	15.d) Library	48
16.	EXCEPTION HANDLING PROGRAMS	
	16.a) Custom Exception	50
	16.b) Multiple Catch	52
	16.c) Throw Throws	53
	16.d) Try Catch	55
17.	FILE HANDLING PROGRAMS	
	17.a) Append File	57
	17.b) Create File	57
	17.c) Read File	59
	17.d) Write File	60

1.UML Diagrams (Library Management System)

a)

Aim: To demonstrate Use case diagram of Library Management System

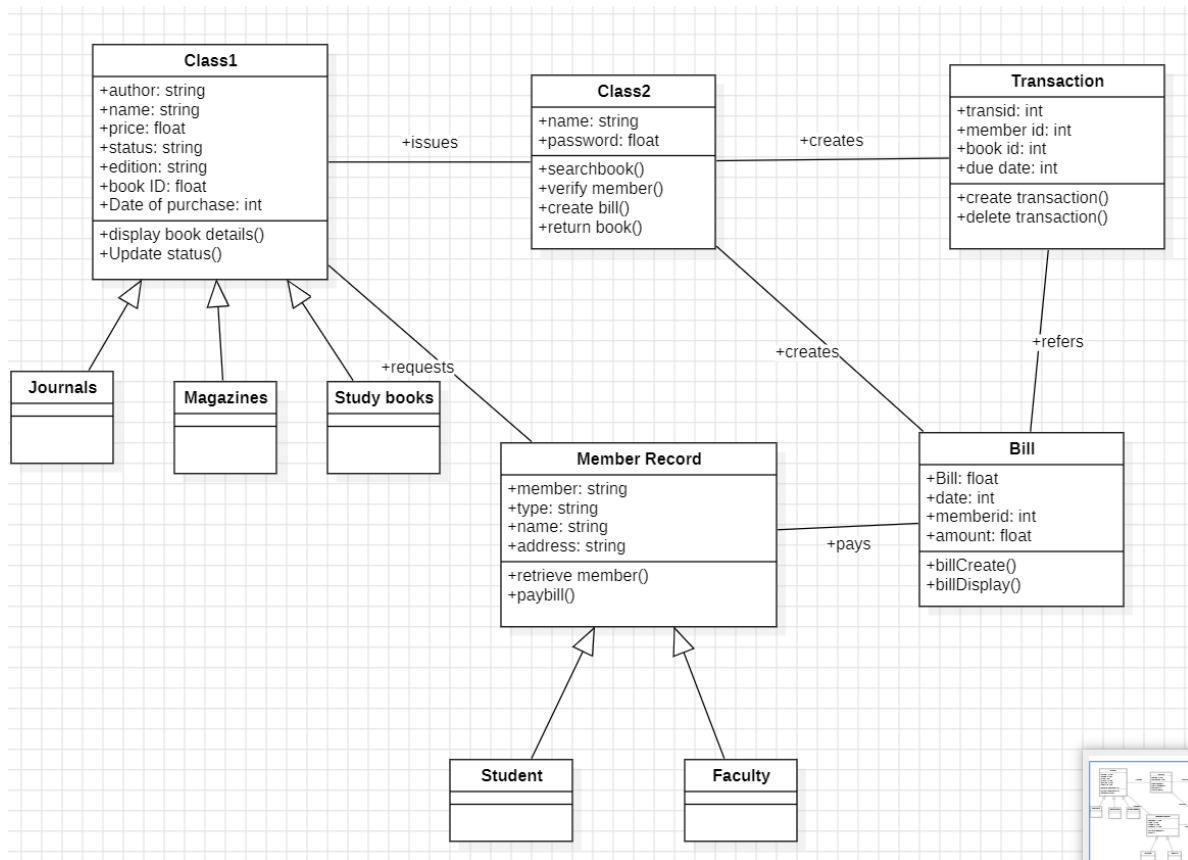
Diagram:



b)

Aim: To demonstrate Class diagram of Library Management System

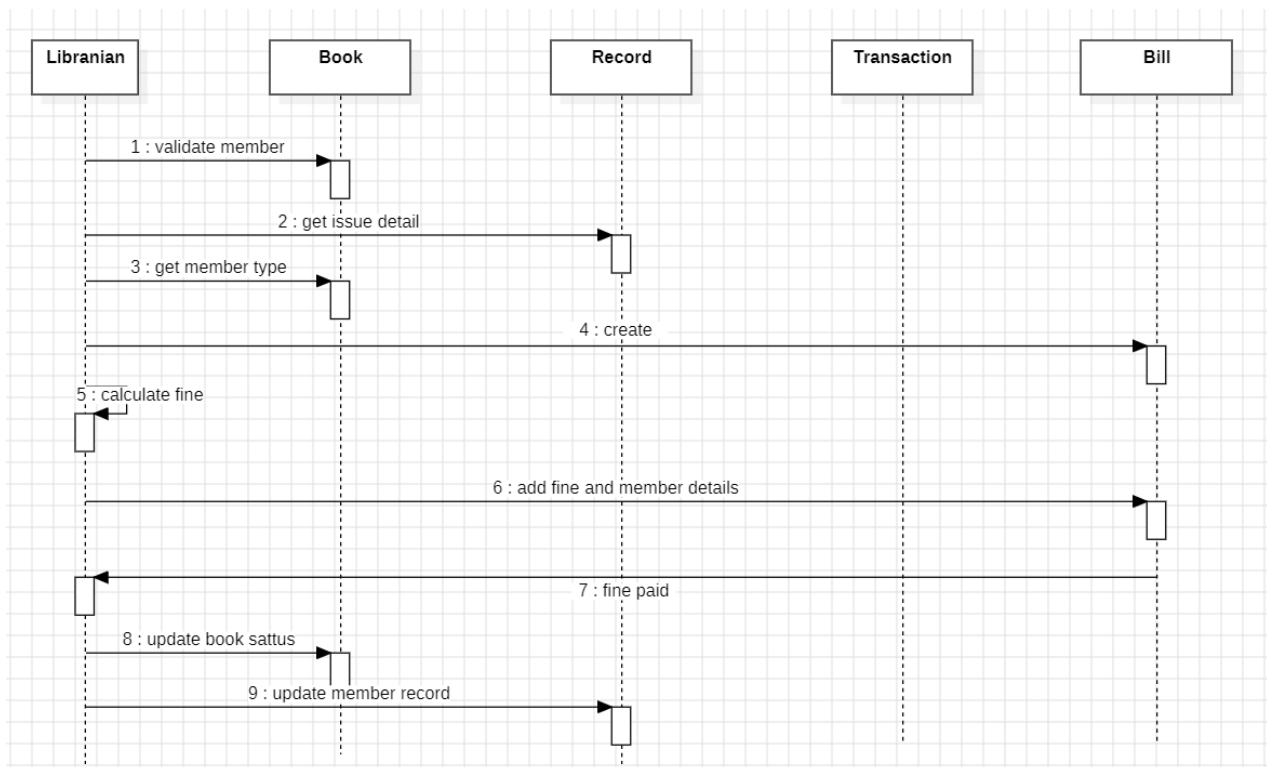
Diagram:



c)

Aim: To demonstrate Sequence diagram of Library Management System

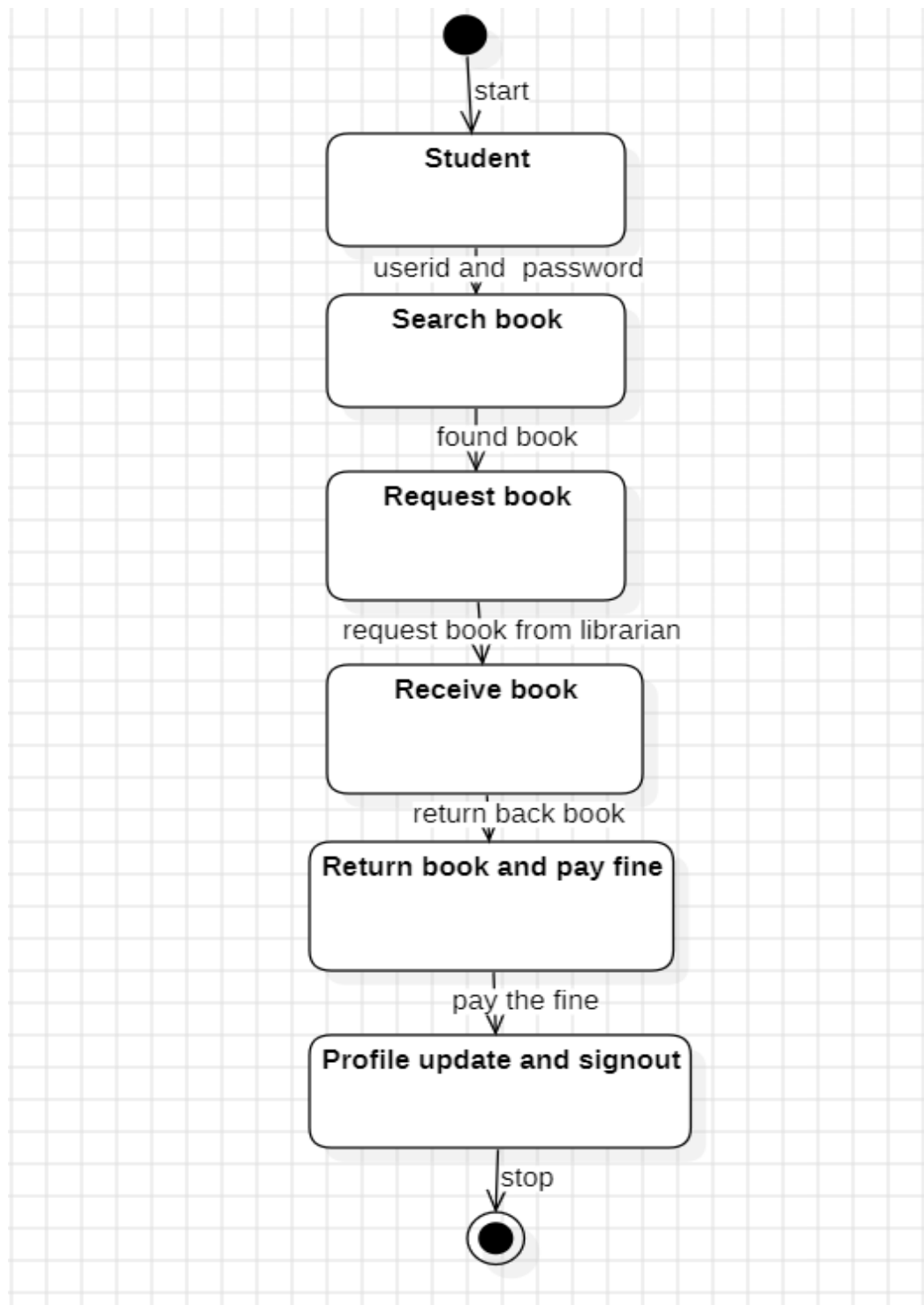
Diagram:



d)

Aim: To demonstrate State diagram of Library Management System

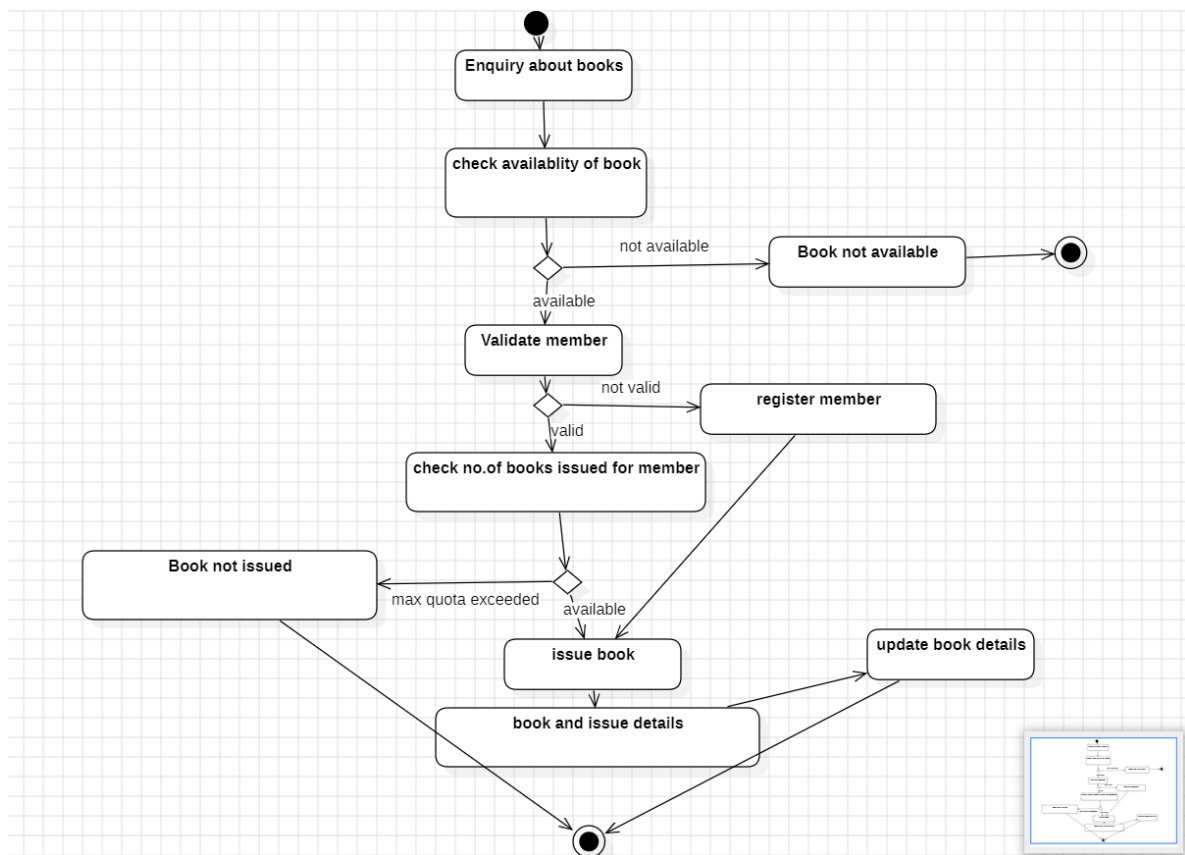
Diagram:



e)

Aim: To demonstrate Activity diagram of Library Management System

Diagram:

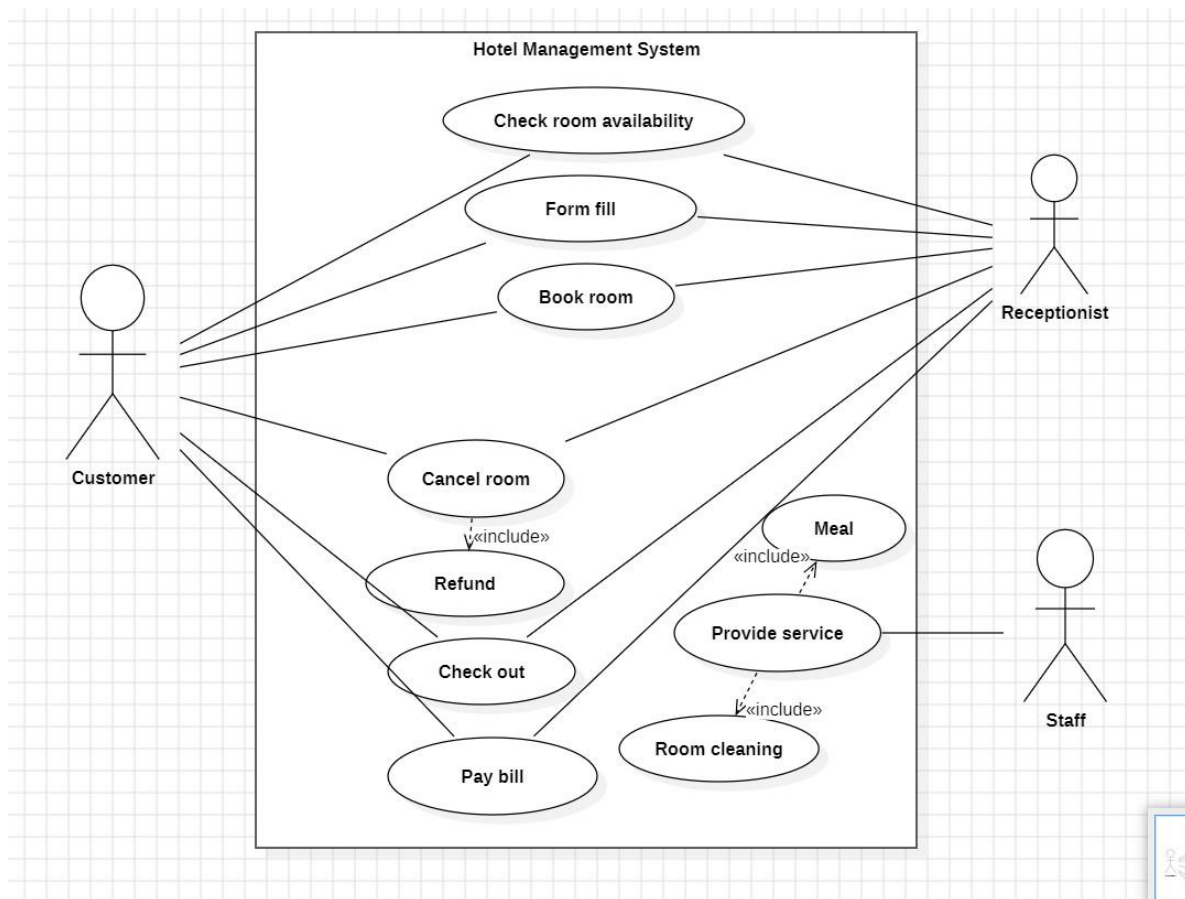


2.UML Diagrams (Hotel Management System)

a)

Aim: To demonstrate Use Case diagram of Hotel Management System

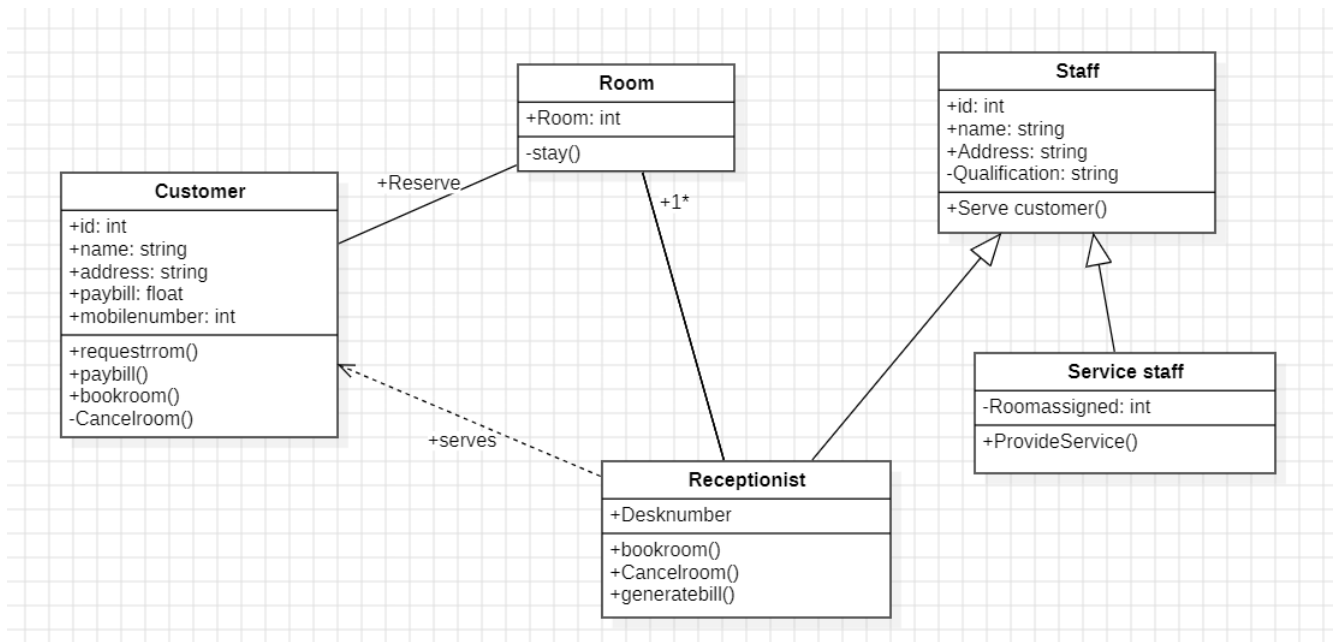
Diagram:



b)

Aim: To demonstrate Class diagram of Hotel Management System

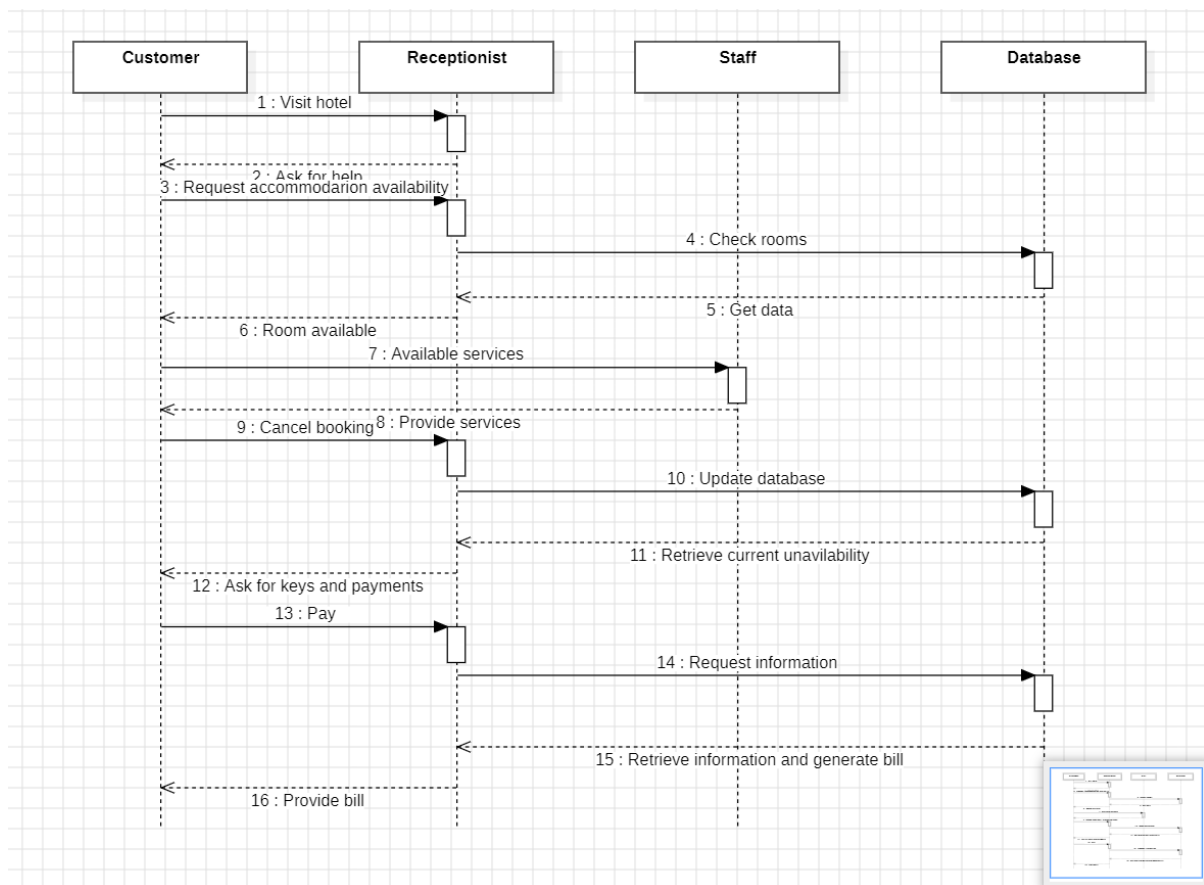
Diagram:



c)

Aim: To demonstrate Sequence diagram of Hotel Management System

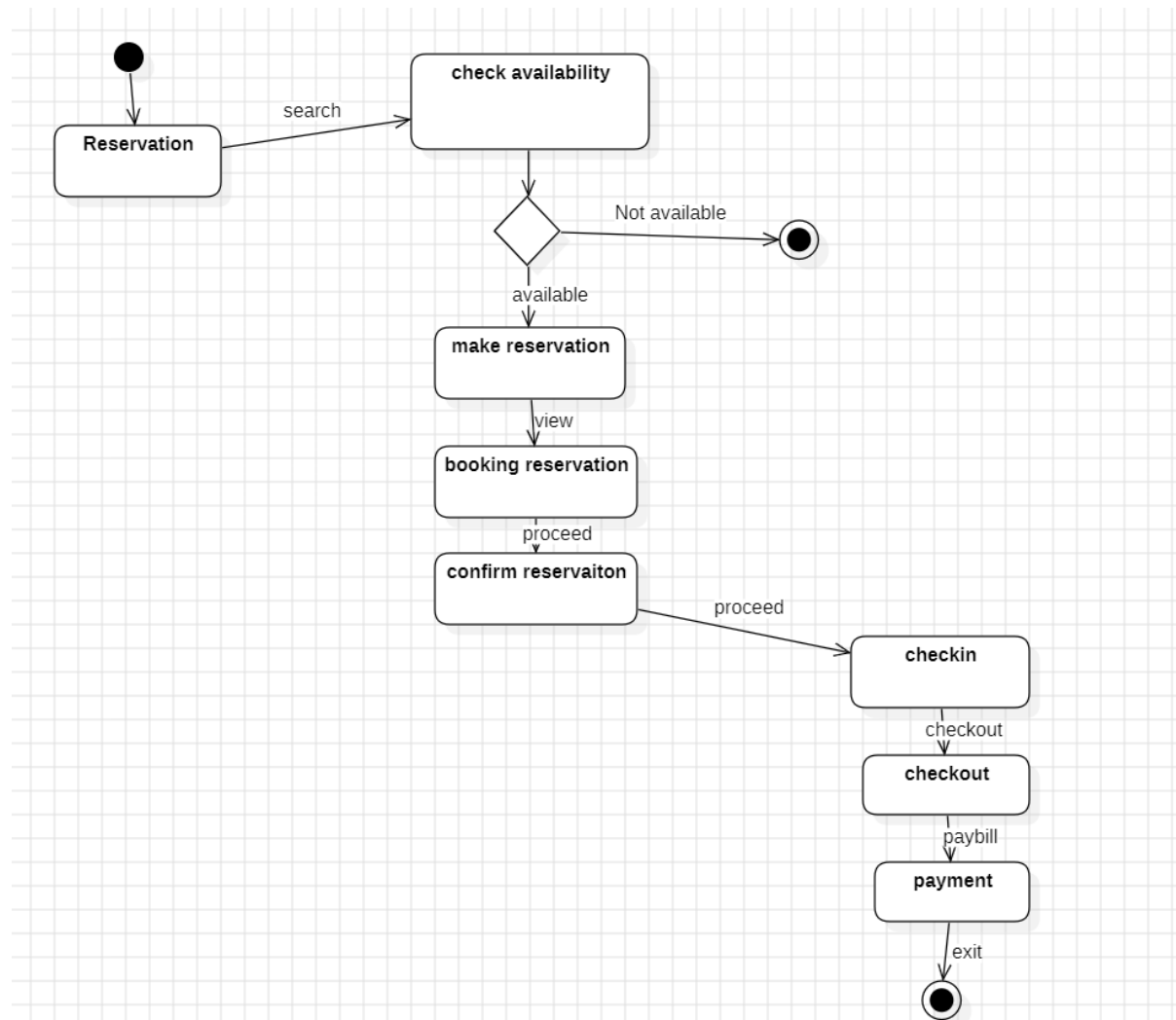
Diagram:



d)

Aim: To demonstrate State diagram of Hotel Management System

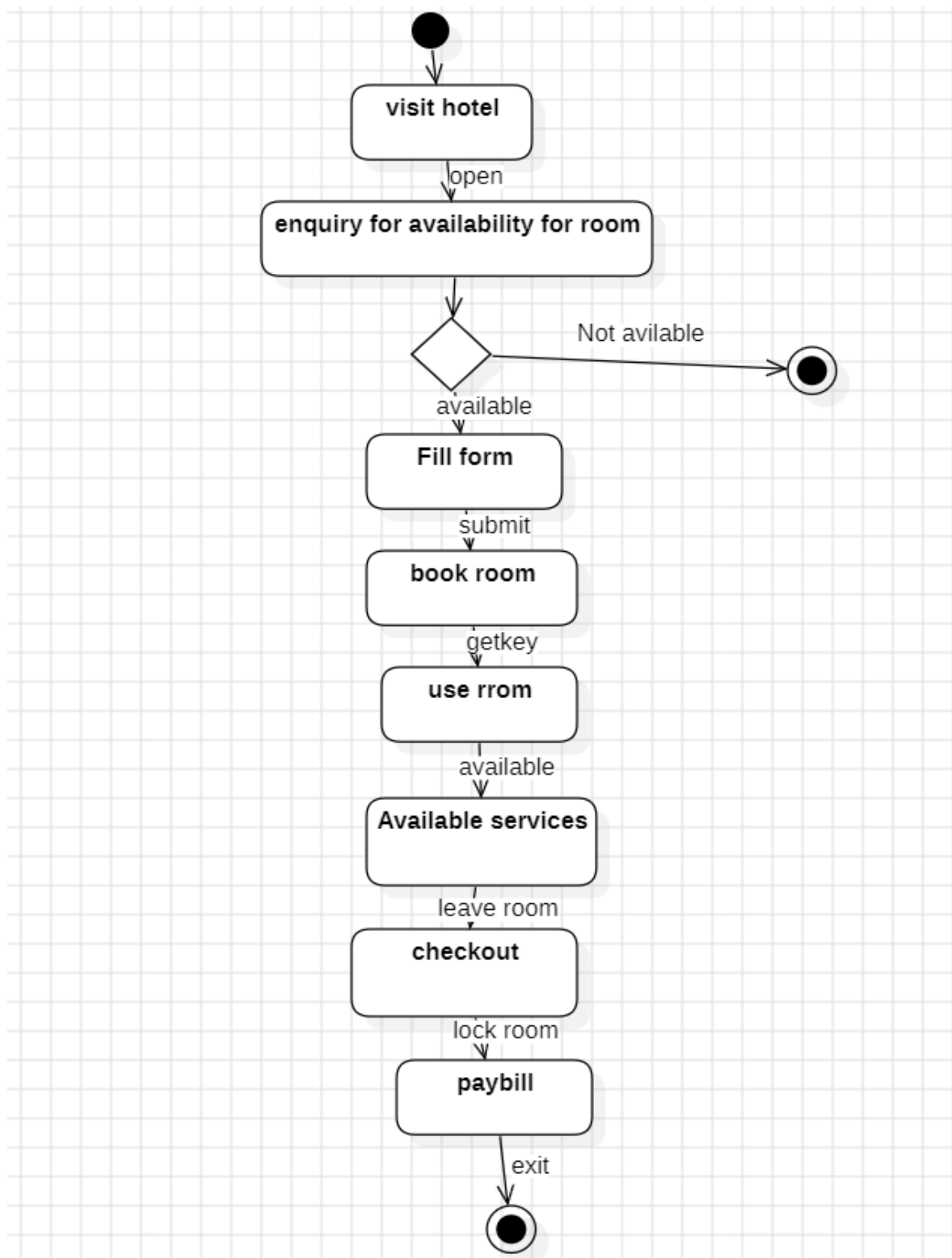
Diagram:



e)

Aim: To demonstrate Activity diagram of Hotel Management System

Diagram:



3. Basic Java Programs

a)

Code:

```
import java.util.Scanner;

public class EvenOrOdd {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");

        int num = scanner.nextInt();

        if (num % 2 == 0) {

            System.out.println(num + " is Even");

        } else {

            System.out.println(num + " is Odd");

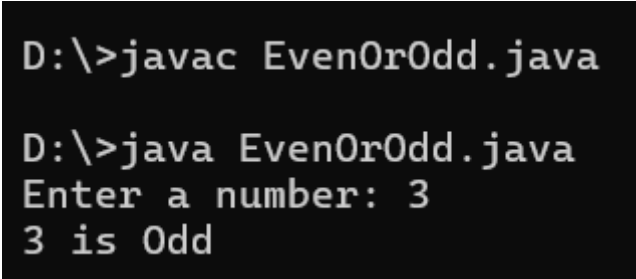
        }

        scanner.close();

    }

}
```

Output:



```
D:\>javac EvenOrOdd.java

D:\>java EvenOrOdd.java
Enter a number: 3
3 is Odd
```


b)

Code:

```
import java.util.Scanner;

public class Factorial {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");

        int num = scanner.nextInt();

        int fact = 1;

        for (int i = 1; i <= num; i++) {

            fact *= i;

        }

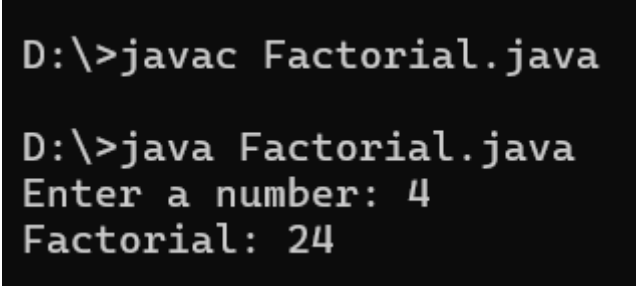
        System.out.println("Factorial: " + fact);

        scanner.close();

    }

}
```

Output:



```
D:\>javac Factorial.java

D:\>java Factorial.java
Enter a number: 4
Factorial: 24
```

c)

Code:

```
import java.util.Scanner;

public class NumberGuess {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int secret = 7, guess;

        do {

            System.out.print("Guess the number: ");

            guess = scanner.nextInt();

            if (guess > secret) {

                System.out.println("Too high!");

            } else if (guess < secret) {

                System.out.println("Too low!");

            }

        } while (guess != secret);

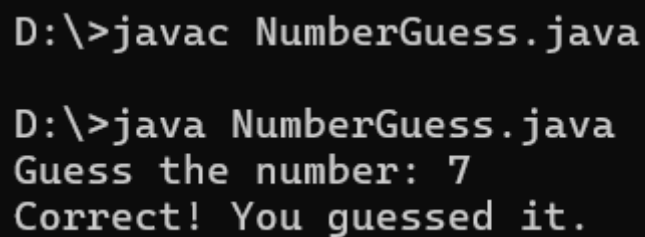
        System.out.println("Correct! You guessed it.");

        scanner.close();

    }

}
```

Output:



```
D:\>javac NumberGuess.java

D:\>java NumberGuess.java
Guess the number: 7
Correct! You guessed it.
```

d)

Code:

```
import java.util.Scanner;

public class Palindrome {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a string: ");

        String str = scanner.nextLine();

        String reversed = "";

        for (int i = str.length() - 1; i >= 0; i--) {

            reversed += str.charAt(i);

        }

        if (str.equals(reversed)) {

            System.out.println("It's a palindrome!");

        } else {

            System.out.println("Not a palindrome.");

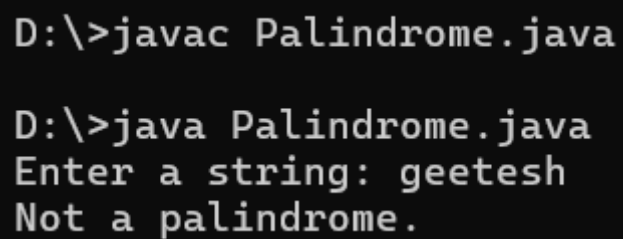
        }

        scanner.close();

    }

}
```

Output:



```
D:\>javac Palindrome.java

D:\>java Palindrome.java
Enter a string: geetesh
Not a palindrome.
```

e)

Code:

```
import java.util.Scanner;

public class PrintNumbers {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");

        int n = scanner.nextInt();

        for (int i = 1; i <= n; i++) {

            System.out.println(i);

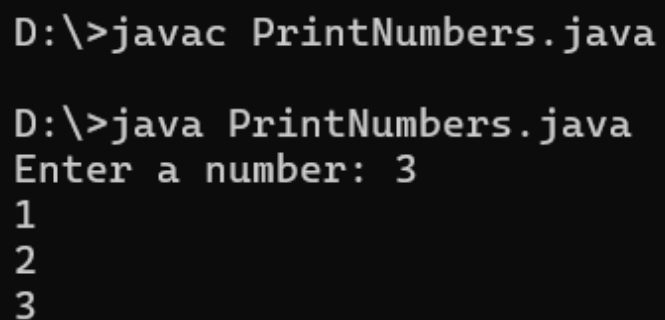
        }

        scanner.close();

    }

}
```

Output:



```
D:\>javac PrintNumbers.java

D:\>java PrintNumbers.java
Enter a number: 3
1
2
3
```

f)

Code:

```
import java.util.Scanner;

public class ReverseNumber {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");

        int num = scanner.nextInt();

        int reversed = 0;

        while (num != 0) {

            reversed = reversed * 10 + num % 10;

            num /= 10;

        }

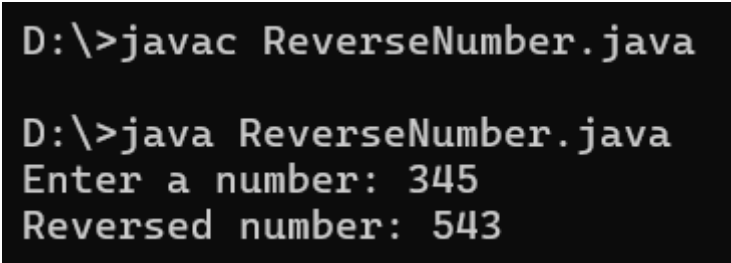
        System.out.println("Reversed number: " + reversed);

        scanner.close();

    }

}
```

Output:



```
D:\>javac ReverseNumber.java

D:\>java ReverseNumber.java
Enter a number: 345
Reversed number: 543
```

g)

Code:

```
import java.util.Scanner;

public class ReverseString {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a string: ");

        String str = scanner.nextLine();

        String reversed = "";

        for (int i = str.length() - 1; i >= 0; i--) {

            reversed += str.charAt(i);

        }

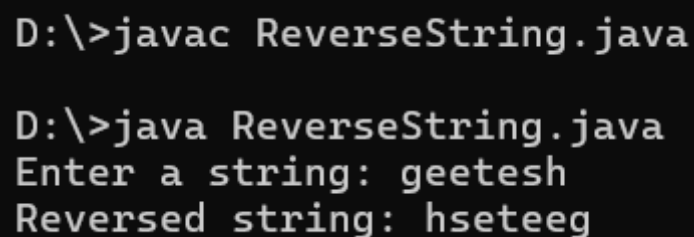
        System.out.println("Reversed string: " + reversed);

        scanner.close();

    }

}
```

Output:



```
D:\>javac ReverseString.java

D:\>java ReverseString.java
Enter a string: geetesh
Reversed string: hseteeg
```

h)

Code:

```
import java.util.Scanner;

public class SumOfDigits {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");

        int num = scanner.nextInt();

        int sum = 0;

        while (num != 0) {

            sum += num % 10;

            num /= 10;

        }

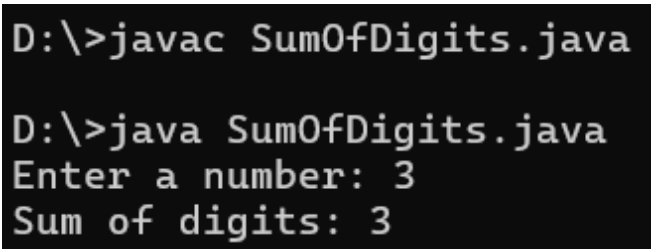
        System.out.println("Sum of digits: " + sum);

        scanner.close();

    }

}
```

Output:



```
D:\>javac SumOfDigits.java

D:\>java SumOfDigits.java
Enter a number: 3
Sum of digits: 3
```

i)

Code:

```
import java.util.Scanner;

public class TablePrinter {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");

        int num = scanner.nextInt();

        for (int i = 1; i <= 10; i++) {

            System.out.println(num + " x " + i + " = " + (num * i));

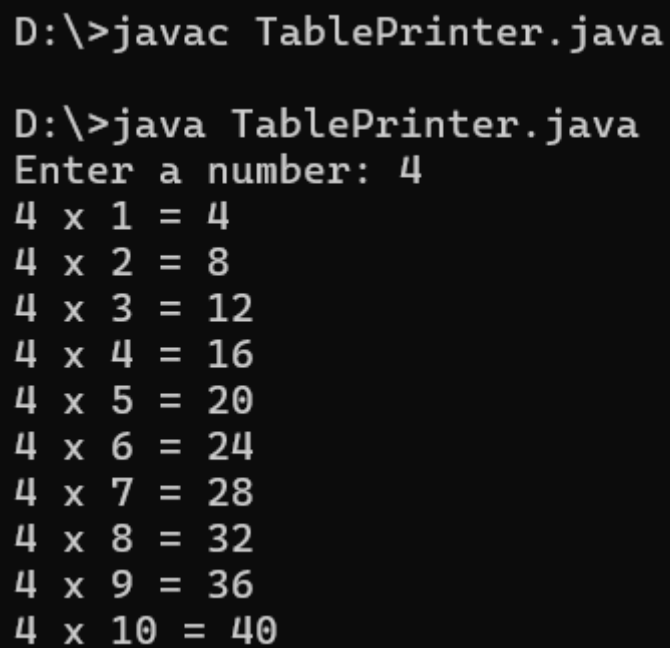
        }

        scanner.close();

    }

}
```

Output:



```
D:\>javac TablePrinter.java

D:\>java TablePrinter.java
Enter a number: 4
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
```


j)

Code:

```
import java.util.Scanner;

public class VowelCounter {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a string: ");

        String str = scanner.nextLine().toLowerCase();

        int count = 0;

        for (char c : str.toCharArray()) {

            if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {

                count++;

            }

        }

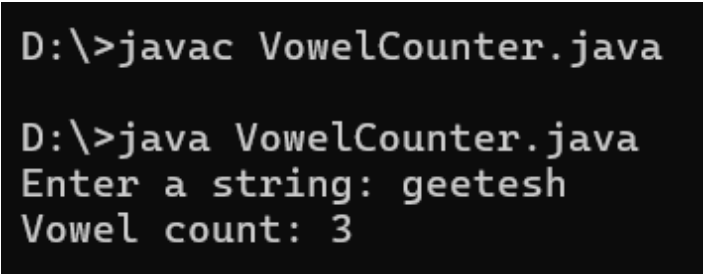
        System.out.println("Vowel count: " + count);

        scanner.close();

    }

}
```

Output:



```
D:\>javac VowelCounter.java

D:\>java VowelCounter.java
Enter a string: geetesh
Vowel count: 3
```

Inheritance

4.Single Inheritance

4 a)

Code:

```
// Single Inheritance - Employee and Manager
class Employee {
    String name;
    String id;

    public Employee(String name, String id) {
        this.name = name;
        this.id = id;
    }

    public void displayEmployee() {
        System.out.println("Employee Name: " + name + ", ID: " + id);
    }
}

class Manager extends Employee {
    String department;

    public Manager(String name, String id, String department) {
        super(name, id);
        this.department = department;
    }

    public void displayManager() {
        displayEmployee();
        System.out.println("Department: " + department);
    }
}

public class SingleInheritance_Employee {
    public static void main(String[] args) {
        Manager manager = new Manager("John", "M123", "HR");
        manager.displayManager();
    }
}
```

```
}
```

Output:

```
Employee Name: John, ID: M123  
Department: HR
```

4b)

Code:

// Single Inheritance - Vehicle and Car

```
class Vehicle {  
    String make;  
    String model;  
  
    public Vehicle(String make, String model) {  
        this.make = make;  
        this.model = model;  
    }  
  
    public void displayInfo() {  
        System.out.println("Vehicle Make: " + make + ", Model: " + model);  
    }  
}  
  
class Car extends Vehicle {  
    String fuelType;  
  
    public Car(String make, String model, String fuelType) {  
        super(make, model);  
        this.fuelType = fuelType;  
    }  
  
    public void carDetails() {  
        displayInfo();  
        System.out.println("Fuel Type: " + fuelType);  
    }  
}
```

```

public class SingleInheritance_Vehicle {
    public static void main(String[] args) {
        Car car = new Car("Toyota", "Camry", "Petrol");
        car.carDetails();
    }
}

```

Output:

```

Vehicle Make: Toyota, Model: Camry
Fuel Type: Petrol

```

5. Multiple Inheritance

5a)

Code:

```

// Multiple Inheritance (via Interfaces) - Speaker and Writer
interface Speaker {
    void speak();
}

interface Writer {
    void write();
}

class SpeakerWriter implements Speaker, Writer {
    String name;
    String topic;
    String genre;

    public SpeakerWriter(String name, String topic, String genre) {
        this.name = name;
        this.topic = topic;
        this.genre = genre;
    }

    @Override
    public void speak() {

```

```

        System.out.println(name + " is speaking about " + topic);
    }

    @Override
    public void write() {
        System.out.println(name + " is writing in the " + genre + " genre");
    }

    public void displayInfo() {
        speak();
        write();
    }
}

public class MultipleInheritance_SpeakerWriter {
    public static void main(String[] args) {
        SpeakerWriter speakerWriter = new SpeakerWriter("David", "Technology",
"Science Fiction");
        speakerWriter.displayInfo();
    }
}

```

Output:

```

David is speaking about Technology
David is writing in the Science Fiction genre

```

5b)

Code:

```

// Multiple Inheritance (via Interfaces) - Student and Employee
interface Student {
    void studentInfo();
}

interface Employee {
    void employeeInfo();
}

```

```

}

class StudentEmployee implements Student, Employee {
    String studentName;
    String studentId;
    String empId;
    String department;

    public StudentEmployee(String studentName, String studentId, String
empId, String department) {
        this.studentName = studentName;
        this.studentId = studentId;
        this.empId = empId;
        this.department = department;
    }

    @Override
    public void studentInfo() {
        System.out.println("Student Name: " + studentName + ", ID: " + studentId);
    }

    @Override
    public void employeeInfo() {
        System.out.println("Employee ID: " + empId + ", Department: " +
department);
    }

    public void displayInfo() {
        studentInfo();
        employeeInfo();
    }
}

public class MultipleInheritance_StudentEmployee {
    public static void main(String[] args) {
        StudentEmployee studentEmployee = new StudentEmployee("Alice",
"S001", "E101", "Finance");
        studentEmployee.displayInfo();
    }
}

```

Output:

```
Student Name: Alice, ID: S001
Employee ID: E101, Department: Finance
```

6.Hierarchical Inheritance

6a)

Code:

```
// Hierarchical Inheritance - Animal, Bird, and Mammal
class Animal {
    String name;
    String species;

    public Animal(String name, String species) {
        this.name = name;
        this.species = species;
    }

    public void animalInfo() {
        System.out.println("Animal Name: " + name + ", Species: " + species);
    }
}

class Bird extends Animal {
    double wingSpan;

    public Bird(String name, String species, double wingSpan) {
        super(name, species);
        this.wingSpan = wingSpan;
    }

    public void birdInfo() {
        animalInfo();
        System.out.println("Wing Span: " + wingSpan + " meters");
    }
}

class Mammal extends Animal {
```

```

String furColor;

public Mammal(String name, String species, String furColor) {
    super(name, species);
    this.furColor = furColor;
}

public void mammalInfo() {
    animalInfo();
    System.out.println("Fur Color: " + furColor);
}
}

public class HierarchicalInheritance_Animal {
    public static void main(String[] args) {
        Bird bird1 = new Bird("Parrot", "Psittaciformes", 0.25);
        bird1.birdInfo();

        Mammal mammal1 = new Mammal("Tiger", "Panthera tigris", "Orange
with black stripes");
        mammal1.mammalInfo();
    }
}

```

Output:

```

Animal Name: Parrot, Species: Psittaciformes
Wing Span: 0.25 meters
Animal Name: Tiger, Species: Panthera tigris
Fur Color: Orange with black stripes

```

6b)

Code:

```

// Hierarchical Inheritance - Product, Electronics, and Clothing
class Product {
    String name;
    double price;
}

```



```

public Product(String name, double price) {
    this.name = name;
    this.price = price;
}

public void productInfo() {
    System.out.println("Product Name: " + name + ", Price: $" + price);
}
}

class Electronics extends Product {
    String brand;

    public Electronics(String name, double price, String brand) {
        super(name, price);
        this.brand = brand;
    }

    public void electronicsInfo() {
        productInfo();
        System.out.println("Brand: " + brand);
    }
}

class Clothing extends Product {
    String size;

    public Clothing(String name, double price, String size) {
        super(name, price);
        this.size = size;
    }

    public void clothingInfo() {
        productInfo();
        System.out.println("Size: " + size);
    }
}

public class HierarchicalInheritance_Product {
    public static void main(String[] args) {
        Electronics electronic1 = new Electronics("Smartphone", 799, "Samsung");
    }
}

```

```
    electronic1.electronicsInfo();

    Clothing clothing1 = new Clothing("T-shirt", 29, "L");
    clothing1.clothingInfo();
}
}
```

Output:

```
Product Name: Smartphone, Price: $799.0
Brand: Samsung
Product Name: T-shirt, Price: $29.0
Size: L
```

7. Hybrid Inheritance

7a)

Code:

```
// Hybrid Inheritance - Chef, Waiter, and Restaurant
interface Chef {
    void prepareFood(String foodItem);
}

interface Waiter {
    void serveFood(String foodItem);
}

class Restaurant implements Chef, Waiter {
    @Override
    public void prepareFood(String foodItem) {
        System.out.println("Chef is preparing " + foodItem);
    }

    @Override
    public void serveFood(String foodItem) {
        System.out.println("Waiter is serving " + foodItem);
    }
}
```

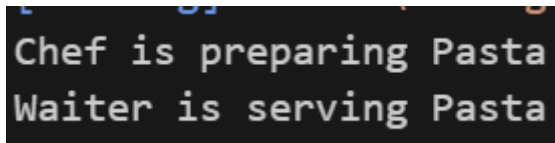
```

    public void manageOrder(String foodItem) {
        prepareFood(foodItem);
        serveFood(foodItem);
    }
}

public class HybridInheritance_ChefWaiter {
    public static void main(String[] args) {
        Restaurant restaurant = new Restaurant();
        restaurant.manageOrder("Pasta");
    }
}

```

Output:



```

Chef is preparing Pasta
Waiter is serving Pasta

```

7b)

Code:

```

// Hybrid Inheritance - Printer, Scanner, and MultiFunction
interface Printer {
    void printDocument(String document);
}

interface Scanner {
    void scanDocument(String document);
}

class MultiFunction implements Printer, Scanner {
    @Override
    public void printDocument(String document) {
        System.out.println("Printing document: " + document);
    }

    @Override
    public void scanDocument(String document) {

```

```

        System.out.println("Scanning document: " + document);
    }

    public void copyDocument(String document) {
        System.out.println("Copying document: " + document);
    }
}

public class HybridInheritance_PrinterScanner {
    public static void main(String[] args) {
        MultiFunction mf = new MultiFunction();
        mf.printDocument("Report");
        mf.scanDocument("Invoice");
        mf.copyDocument("Contract");
    }
}

```

Output:

```

Printing document: Report
Scanning document: Invoice
Copying document: Contract

```

Polymorphism

8) Constructor

Code:

```

// Constructor Polymorphism - Example 2
class Laptop {
    String brand;
    int price;

    Laptop(String brand) {
        this.brand = brand;
        System.out.println("Laptop brand: " + brand);
    }
}

```

```

    }

    Laptop(String brand, int price) {
        this.brand = brand;
        this.price = price;
        System.out.println("Laptop brand: " + brand + ", Price: " + price);
    }

    public static void main(String[] args) {
        Laptop laptop1 = new Laptop("Dell");
        Laptop laptop2 = new Laptop("HP", 800);
    }
}

```

Output:

```

Laptop brand: Dell
Laptop brand: HP, Price: 800

```

9) Constructor Overloading

Code:

```

// Constructor Overloading - Example 1
class Employee {
    String name;
    int age;

    Employee(String name) {
        this.name = name;
        System.out.println("Employee Name: " + name);
    }

    Employee(String name, int age) {
        this.name = name;
        this.age = age;
        System.out.println("Employee Name: " + name + ", Age: " + age);
    }

    public static void main(String[] args) {

```

```
Employee emp1 = new Employee("Alice");
Employee emp2 = new Employee("Bob", 30);
}
}
```

Output:

```
Employee Name: Alice
Employee Name: Bob, Age: 30
```

10) Method Overloading

10.a)

Code:

```
// Method Overloading - Example 1
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

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

    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println("Sum of integers: " + calc.add(10, 20));
        System.out.println("Sum of doubles: " + calc.add(10.5, 20.5));
    }
}
```

Output:

```
Sum of integers: 30
Sum of doubles: 31.0
```

10.b)

Code:

```
// Method Overloading - Example 2
class Display {
    void show(String message) {
        System.out.println("Message: " + message);
    }

    void show(int number) {
        System.out.println("Number: " + number);
    }

    public static void main(String[] args) {
        Display display = new Display();
        display.show("Hello, World!");
        display.show(123);
    }
}
```

Output:

```
Message: Hello, World!
Number: 123
```

11. Method Overriding

11.a)

Code:

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

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

```

public class Main {
    public static void main(String[] args) {
        Shape shape = new Shape();
        shape.draw();// Calls Shape's draw() method

        Circle circle = new Circle();
        circle.draw();// Calls Circle's overridden draw() method
    }
}

```

Output:

```

Drawing a shape
Drawing a circle

```

11.b)

Code:

```

class Vehicle {
    void start() {
        System.out.println("Vehicle is starting");
    }
}

class Bike extends Vehicle {
    @Override
    void start() {
        System.out.println("Bike is starting");
    }
}

public class MethodOverridingExample {
    public static void main(String[] args) {
        Vehicle vehicle = new Vehicle();
        vehicle.start();// Calls Vehicle's start() method

        Bike bike = new Bike();
        bike.start();// Calls Bike's overridden start() method
    }
}

```


Output:

```
Vehicle is starting  
Bike is starting
```

Abstraction

12. Interface programs

12.a)

Code:

```
interface AnimalInterface {  
    void sound();  
}  
  
class Cat implements AnimalInterface {  
    public void sound() {  
        System.out.println("Meowing");  
    }  
}  
  
public class AnimalInterfaceDemo {  
    public static void main(String args[]) {  
        AnimalInterface obj = new Cat();// Polymorphism  
        obj.sound();  
    }  
}
```

Output:

```
Meowing
```

12b)

Code:

```
interface Bank {  
    float getRateOfInterest();  
}
```

```

}

class SBI implements Bank {
    public float getRateOfInterest() {
        return 5.5f;
    }
}

public class BankDemo {
    public static void main(String args[]) {
        Bank obj = new SBI();// Polymorphism: Interface reference holding SBI
        object
        System.out.println("Rate of Interest: " + obj.getRateOfInterest() + "%");
    }
}

```

Output:

```

Rate of Interest: 5.5%

```

12c)

Code:

```

interface A {
    void methodA();
}

interface B {
    void methodB();
}

class MultipleInheritanceExample implements A, B {
    public void methodA() {
        System.out.println("Method A");
    }

    public void methodB() {
        System.out.println("Method B");
    }
}

```

```

}

public class MultipleInheritanceDemo {
    public static void main(String args[]) {
        A objA = new MultipleInheritanceExample();
        objA.methodA();

        B objB = new MultipleInheritanceExample();
        objB.methodB();
    }
}

```

Output:

```

Method A
Method B

```

12d)

Code:

```

interface Printable {
    void print();
}

class Document implements Printable {
    public void print() {
        System.out.println("Printing Document");
    }
}

public class PrintableDemo {
    public static void main(String args[]) {
        Printable obj = new Document();// Using interface reference
        obj.print();
    }
}

```

Output:

Abstract Classes Programs

13a)

Code:

```
abstract class Animal {  
    abstract void makeSound();  
}  
  
class Dog extends Animal {  
    void makeSound() {  
        System.out.println("Barking");  
    }  
}  
  
public class AnimalSound {  
    public static void main(String args[]) {  
        Animal obj = new Dog();  
        obj.makeSound();  
    }  
}
```

Output:

Barking

13b)

Code:

```
abstract class Employee {  
    abstract void showDetails();  
}  
  
class Manager extends Employee {  
    void showDetails() {  
        System.out.println("Manager Details");  
    }  
}
```

```

    }
}

public class EmployeeDemo {
    public static void main(String args[]) {
        Employee obj = new Manager();
        obj.showDetails();
    }
}

```

Output:

```

Manager Details

```

13.c)

Code:

```

abstract class Shape {
    abstract void draw();
}

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

public class ShapeDemo {
    public static void main(String args[]) {
        Shape obj = new Circle();
        obj.draw();
    }
}

```

Output:

```

Drawing Circle

```

13d)

Code:

```

abstract class Vehicle {
    abstract void start();
}

class Car extends Vehicle {
    void start() {
        System.out.println("Car is starting");
    }
}

public class VehicleDemo {
    public static void main(String args[]) {
        Vehicle obj = new Car(); // Polymorphism
        obj.start();
    }
}

```

Output:

```
Car is starting
```

Encapsulation

14a)

Code:

```

// Encapsulation Example 2: Encapsulation with Constructor
class BankAccount {
    private String accountHolder;
    private double balance;

    public BankAccount(String accountHolder, double balance) {
        this.accountHolder = accountHolder;
        this.balance = (balance >= 0) ? balance : 0.0;
        if (balance < 0) { System.out.println("Initial balance cannot be negative.
Setting balance to $0.0"); }
    }

    public String getAccountHolder() { return accountHolder; }
    public double getBalance() { return balance; }
}

```

```

public void deposit(double amount) {
    if (amount > 0) { balance += amount; }
    else { System.out.println("Deposit amount must be positive"); }
}

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

public static void main(String[] args) {
    BankAccount account = new BankAccount("Alice", 1000.0);
    System.out.println("Account Holder: " + account.getAccountHolder());
    System.out.println("Balance: $" + account.getBalance());

    account.deposit(500.0);
    System.out.println("Balance after deposit: $" + account.getBalance());

    account.withdraw(300.0);
    System.out.println("Balance after withdrawal: $" + account.getBalance());
}
}

```

Output:

```

Account Holder: Alice
Balance: $1000.0
Balance after deposit: $1500.0
Balance after withdrawal: $1200.0

```

14c)

Code:

```

// Encapsulation Example 4: Encapsulation with Access Control
class Car {
    private String brand;
    private String model;
    private int year;
}

```

```

public void setCarDetails(String brand, String model, int year) {
    this.brand = brand;
    this.model = model;
    this.year = year;
}

public void displayCarDetails() {
    System.out.println("Car Brand: " + brand);
    System.out.println("Car Model: " + model);
    System.out.println("Car Year: " + year);
}

public static void main(String[] args) {
    Car car = new Car();
    car.setCarDetails("Toyota", "Camry", 2022);
    car.displayCarDetails();
}
}

```

Output:

```

Car Brand: Toyota
Car Model: Camry
Car Year: 2022

```

14d)

Code:

```

class Book {
    private String title;
    private String author;

    // Constructor to initialize the book
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }
}

```



```

// Getter for title
public String getTitle() {
    return title;
}

// Setter for title
public void setTitle(String title) {
    this.title = title;
}

// Getter for author
public String getAuthor() {
    return author;
}

// Setter for author
public void setAuthor(String author) {
    this.author = author;
}
}

public class EncapsulationExample {
    public static void main(String[] args) {
        // Creating an object using the constructor
        Book book = new Book("The Alchemist", "Paulo Coelho");

        // Displaying book details
        System.out.println("Book Title: " + book.getTitle());
        System.out.println("Book Author: " + book.getAuthor());

        // Modifying book details using setters
        book.setTitle("Atomic Habits");
        book.setAuthor("James Clear");

        // Displaying updated book details
        System.out.println("\nUpdated Book Details:");
        System.out.println("Book Title: " + book.getTitle());
        System.out.println("Book Author: " + book.getAuthor());
    }
}

```

```
}
```

Output:

```
Book Title: The Alchemist  
Book Author: Paulo Coelho
```

```
Updated Book Details:  
Book Title: Atomic Habits  
Book Author: James Clear
```

14g)

Code:

```
// Encapsulation Example 1: Basic Encapsulation  
class Person {  
    private String name;  
    private int age;  
  
    public String getName() { return name; }  
    public void setName(String name) { this.name = name; }  
  
    public int getAge() { return age; }  
    public void setAge(int age) {  
        if (age > 0) { this.age = age; }  
        else { System.out.println("Age must be positive"); }  
    }  
  
    public static void main(String[] args) {  
        Person person = new Person();  
        person.setName("John");  
        person.setAge(25);  
  
        System.out.println("Name: " + person.getName());  
        System.out.println("Age: " + person.getAge());  
    }  
}
```

Output:

```
Name: John  
Age: 25
```

Packages

15a)

Code:

```
import java.util.ArrayList;  
import java.util.List;  
  
public class B1 {  
    public static void main(String[] args) {  
        List<String> names = new ArrayList<>();  
        names.add("John");  
        names.add("Alice");  
        names.add("Bob");  
  
        System.out.println("Names List: " + names);  
    }  
}
```

Output:

```
Names List: [John, Alice, Bob]
```

15b)

Code:

```
import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
  
public class B2 {  
    public static void main(String[] args) {  
        try {  
            File file = new File("output.txt");  
            if (file.createNewFile()) {
```

```

        System.out.println("File created: " + file.getName());
    } else {
        System.out.println("File already exists.");
    }

    FileWriter writer = new FileWriter(file);
    writer.write("Hello, World!");
    writer.close();
    System.out.println("Successfully wrote to the file.");
} catch (IOException e) {
    System.out.println("An error occurred.");
    e.printStackTrace();
}
}
}

```

Output:

```

File created: output.txt
Successfully wrote to the file.

```

15c)

Code:

```

import pkg.Account;
import pkg.Transaction;

public class BankExample {
    public static void main(String[] args) {
        Account acc1 = new Account("1001", 1000.0);
        Account acc2 = new Account("1002", 500.0);
        Transaction trans = new Transaction();

        System.out.println("Before: Acc1: " + acc1.getBalance() + ", Acc2: " +
acc2.getBalance());
        trans.transfer(acc1, acc2, 300.0);
        System.out.println("After: Acc1: " + acc1.getBalance() + ", Acc2: " +
acc2.getBalance());
    }
}

```

```
}
```

```
package pkg;
```

```
public class Account {  
    private double balance;  
    private String accountNumber;  
  
    public Account(String accountNumber, double initialBalance) {  
        this.accountNumber = accountNumber;  
        this.balance = initialBalance;  
    }  
  
    public double getBalance() {  
        return balance;  
    }  
  
    public void deposit(double amount) {  
        balance += amount;  
    }  
}
```

```
package pkg;
```

```
public class Transaction {  
    public void transfer(Account from, Account to, double amount) {  
        if (from.getBalance() >= amount) {  
            from.deposit(-amount);  
            to.deposit(amount);  
        }  
    }  
}
```

Output:

```
Before: Acc1: 1000.0, Acc2: 500.0
After:  Acc1: 700.0, Acc2: 800.0
```

15d)

Code:

```
import pkg.Book;
import pkg.Member;

public class LibraryExample {
    public static void main(String[] args) {
        Book book = new Book("Java Basics", "John Smith");
        Member member = new Member("Sarah");

        System.out.println("Book: " + book.getTitle() + " by " + book.getAuthor());
        member.borrowBook(book);
    }
}
```

```
package pkg;
```

```
public class Book {
    private String title;
    private String author;

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }
}
```

```

}

package pkg;

public class Member {
    private String name;

    public Member(String name) {
        this.name = name;
    }

    public void borrowBook(Book book) {
        System.out.println(name + " borrowed " + book.getTitle());
    }
}

```

Output:

```

Book: Java Basics by John Smith
Sarah borrowed Java Basics

```

Exception Handling

16a)

Code:

```

class CustomException extends Exception {
    CustomException(String message) {
        super(message);
    }
}

class CustomExceptionExample {
    static void checkNumber(int num) throws CustomException {
        if (num < 0) {
            throw new CustomException("Negative number not allowed");
        } else {
            System.out.println("Valid number");
        }
    }
}

```

```

public static void main(String args[]) {
    try {
        checkNumber(-5);
    } catch (CustomException e) {
        System.out.println("Caught Custom Exception: " + e.getMessage());
    }
}
}

```

Output:

```

Caught Custom Exception: Negative number not allowed

```

16b)

Code:

```

class MultipleCatchExample {
    public static void main(String args[]) {
        try {
            int arr[] = new int[5];
            arr[10] = 30 / 0; // ArrayIndexOutOfBoundsException &
ArithmeticException
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Exception: " + e);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array Index Out Of Bounds: " + e);
        }
    }
}

```

Output:

```

Arithmetic Exception: java.lang.ArithmeticException: / by zero

```

16c)

Code:

```

class ThrowThrowsExample {
    static void validate(int age) throws IllegalArgumentException {

```



```

    if (age < 18) {
        throw new IllegalArgumentException("Not eligible to vote");
    } else {
        System.out.println("Eligible to vote");
    }
}
public static void main(String args[]) {
    try {
        validate(16);
    } catch (Exception e) {
        System.out.println("Exception caught: " + e);
    }
}
}

```

Output:

```
Exception caught: java.lang.IllegalArgumentException: Not eligible to vote
```

16d)

Code:

```

class TryCatchExample {
    public static void main(String args[]) {
        try {
            int data = 50 / 0; // ArithmeticException
        } catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero!");
        }
    }
}

```

Output:

```
Cannot divide by zero!
```

File Handling

17a)

Code:

```
import java.io.FileWriter;
import java.io.IOException;
class AppendFileExample {
    public static void main(String args[]) {
        try {
            FileWriter writer = new FileWriter("example.txt", true);
            writer.write("\nAppending new data.");
            writer.close();
            System.out.println("Successfully appended data.");
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

Output:

```
Successfully appended data.
```

17b)

Code:

```
import java.io.File;
import java.io.IOException;
class CreateFileExample {
    public static void main(String args[]) {
        try {
            File myFile = new File("example.txt");
            if (myFile.createNewFile()) {
                System.out.println("File created: " + myFile.getName());
            } else {
                System.out.println("File already exists.");
            }
        } catch (IOException e) {
            System.out.println("An error occurred.");
        }
    }
}
```

```

        e.printStackTrace();
    }
}

```

Output:

File already exists.

17c)

Code:

```

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
class ReadFileExample {
    public static void main(String args[]) {
        try {
            File myFile = new File("example.txt");
            Scanner reader = new Scanner(myFile);
            while (reader.hasNextLine()) {
                String data = reader.nextLine();
                System.out.println(data);
            }
            reader.close();
        } catch (FileNotFoundException e) {
            System.out.println("File not found.");
            e.printStackTrace();
        }
    }
}

```

Output:

Appending new data.

17d)

Code:

```

import java.io.FileWriter;
import java.io.IOException;

```

```
class WriteFileExample {  
    public static void main(String args[]) {  
        try {  
            FileWriter writer = new FileWriter("example.txt");  
            writer.write("Hello, this is a test file.");  
            writer.close();  
            System.out.println("Successfully wrote to the file.");  
        } catch (IOException e) {  
            System.out.println("An error occurred.");  
            e.printStackTrace();  
        }  
    }  
}
```

Code:

```
Successfully wrote to the file.
```