



SCHOOL OF
COMPUTING

LAB RECORD

23CSE111- Object Oriented Programming

Submitted by

CH.SC.U4CSE24112 -Deepak SN

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.U4CSE24112 – Deepak SN** 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 / /2025

Internal Examiner 1

Internal Examiner 2

INDEX

S.NO	TITLE	PAGE.NO
	UML DIAGRAM	
1.	LIBRARY MANAGEMENT SYSTEM	
	1.a) Use Case Diagram	
	1.b) Class Diagram	
	1.c) Sequence Diagram	
	1.d) Activity Diagram	
	1.e) State Diagram	
2.	ONLINE FOOD ORDERING SYSTEM	
	2.a) Use Case Diagram	
	2.b) Class Diagram	
	2.c) Sequence Diagram	
	2.d) Activity Diagram	
	2.e) State Diagram	
3.	BASIC JAVA PROGRAMS	
	3.a) Armstrong Number	
	3.b) Checking even or odd	
	3.c) Factorial	
	3.d) Fibonacci	
	3.e) Finding the largest number	
	3.f) Finding the leap year	
	3.g) Number check	
	3.h) Palindrome Checker	
	3.i) Prime	
	3.j) Multiplication Table	
	INHERITANCE	
4.	SINGLE INHERITANCE PROGRAMS	
	4.a) Single inheritance 1	
	4.b) Single inheritance 2	

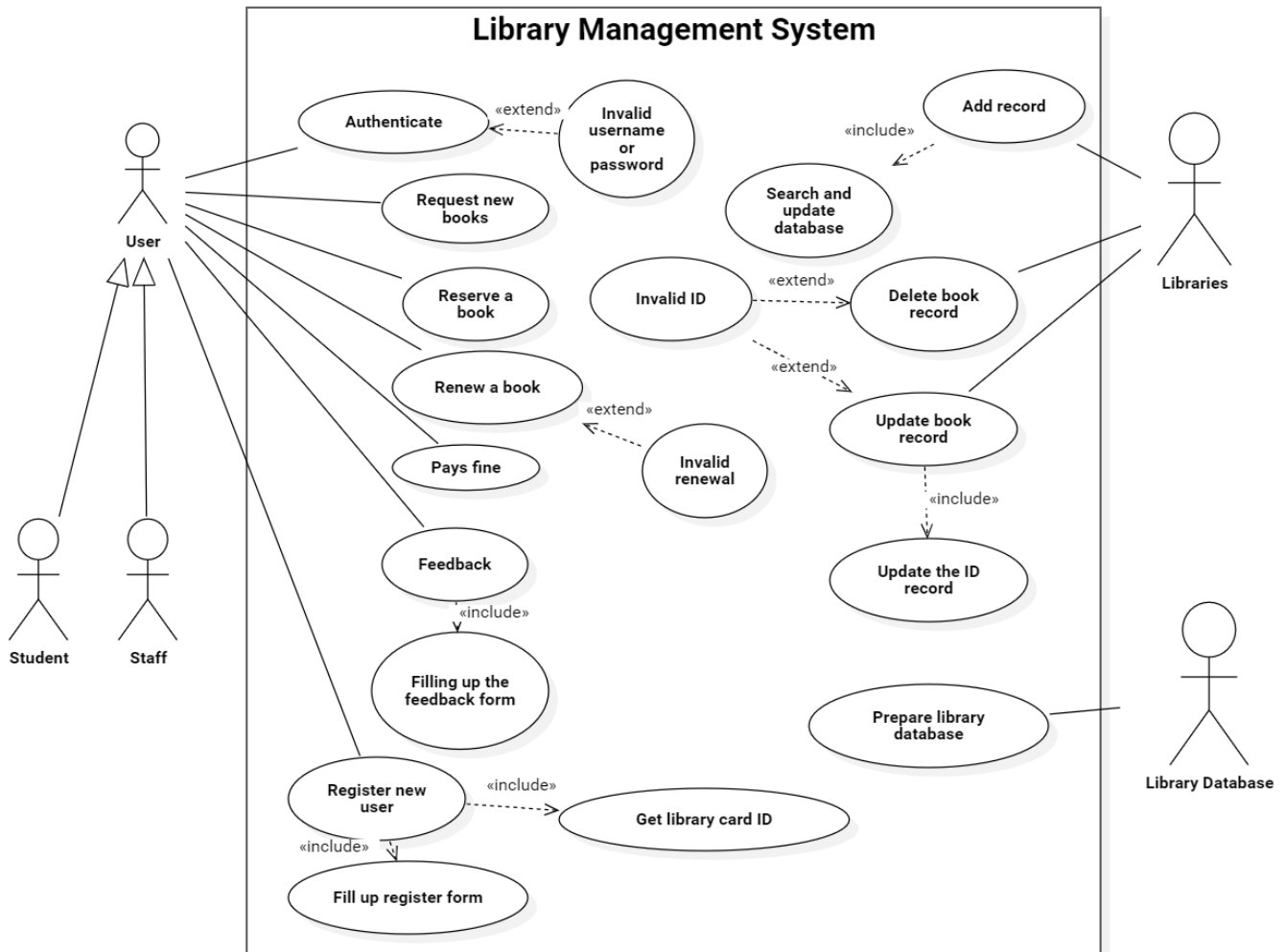
5.	MULTILEVEL INHERITANCE PROGRAMS	
	5.a) Multilevel 1	
	5.b) Multilevel 2	
6.	HIERARCHICAL INHERITANCE PROGRAMS	
	6.a) Hierarchical 1	
	6.b) Hierarchical 2	
7.	HYBRID INHERITANCE PROGRAMS	
	7.a) Hybrid 1	
	7.b) Hybrid 2	
	POLYMORPHISM	
8.	CONSTRUCTOR PROGRAMS	
	8.a) Constructor 1	
	8.b) Constructor 2	
9.	CONSTRUCTOR OVERLOADING PROGRAMS	
	9.a) Constructor overloading 1	
	9.b) Constructor overloading 2	
10.	METHOD OVERLOADING PROGRAMS	
	10.a) Overloading 1	
	10.b) Overloading 2	
11.	METHOD OVERRIDING PROGRAMS	
	11.a) Overriding 1	
	11.b) Overriding 2	
	ABSTRACTION	
12.	INTERFACE PROGRAMS	
	12.a) Interface 1	
	12.b) Interface 2	
	12.c) Interface 3	
	12.d) Interface 4	
13.	ABSTRACT CLASS PROGRAMS	
	13.a) Abstract class 1	
	13.b) Abstract class 2	
	13.c) Abstract class 3	
	13.d) Abstract class 4	
	ENCAPSULATION	
14.	ENCAPSULATION PROGRAMS	
	14.a) Encapsulation 1	
	14.b) Encapsulation 2	
	14.c) Encapsulation 3	
	14.d) Encapsulation 4	
15.	PACKAGES PROGRAMS	
	15.a) User-Defined 1	
	15.b) User-Defined 2	

	15.c) Built-in 1	
	15.d) Built-in 2	
16.	EXCEPTION HANDLING PROGRAMS	
	16.a) Exception 1	
	16.b) Exception 2	
	16.c) Exception 3	
	16.d) Exception 4	
17.	FILE HANDLING PROGRAMS	
	17.a) File 1	
	17.b) File 2	
	17.c) File 3	
	17.d) File 4	

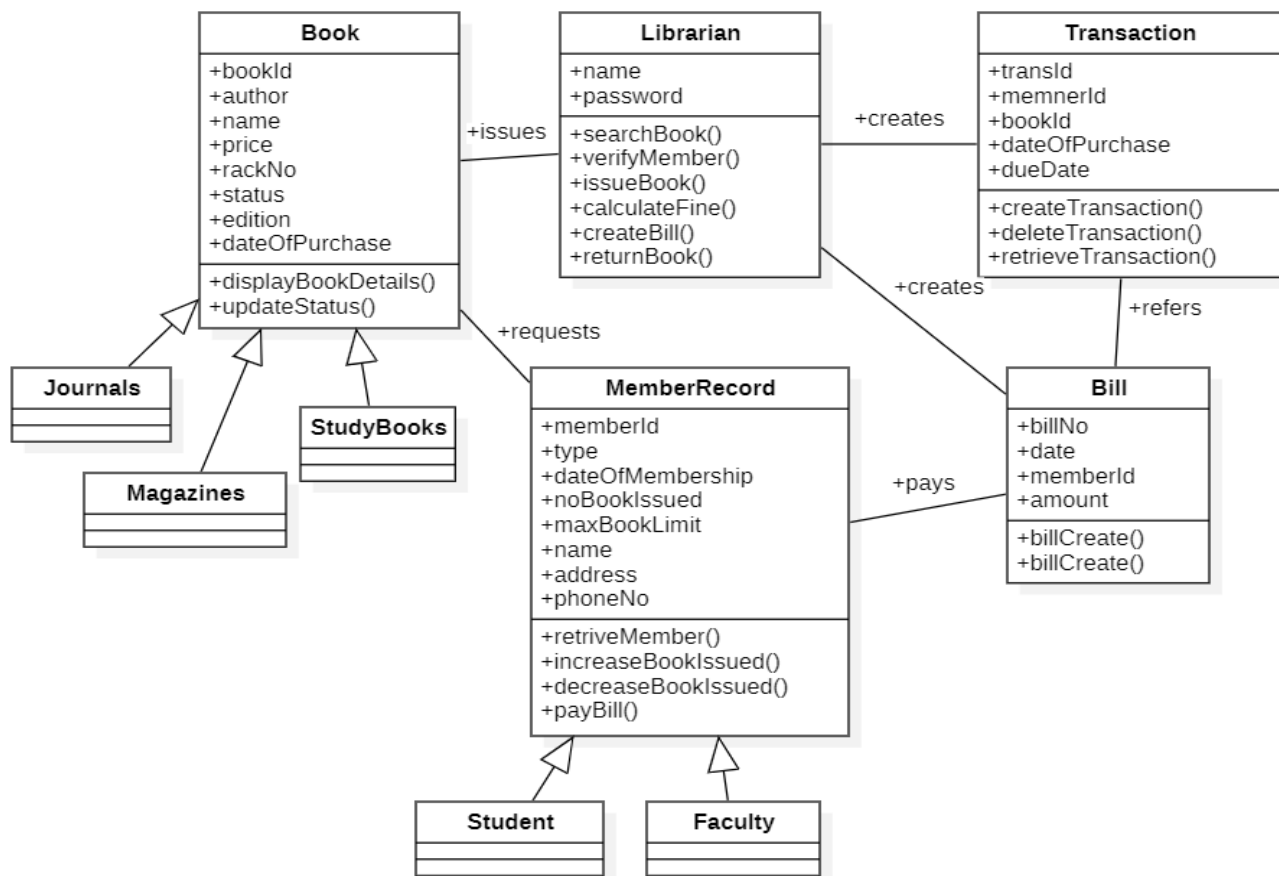
UML DIAGRAMS

1.LIBRARY MANAGEMENT SYSTEM

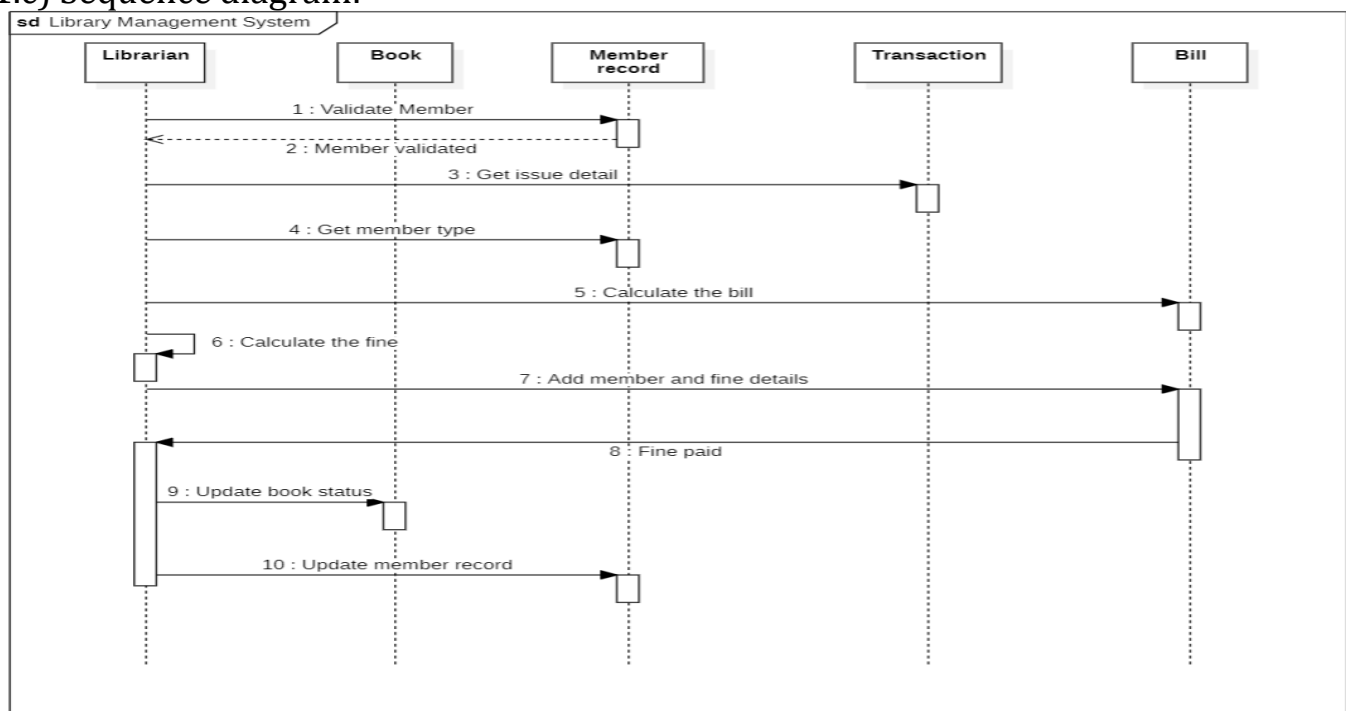
1.a) Use case diagram:



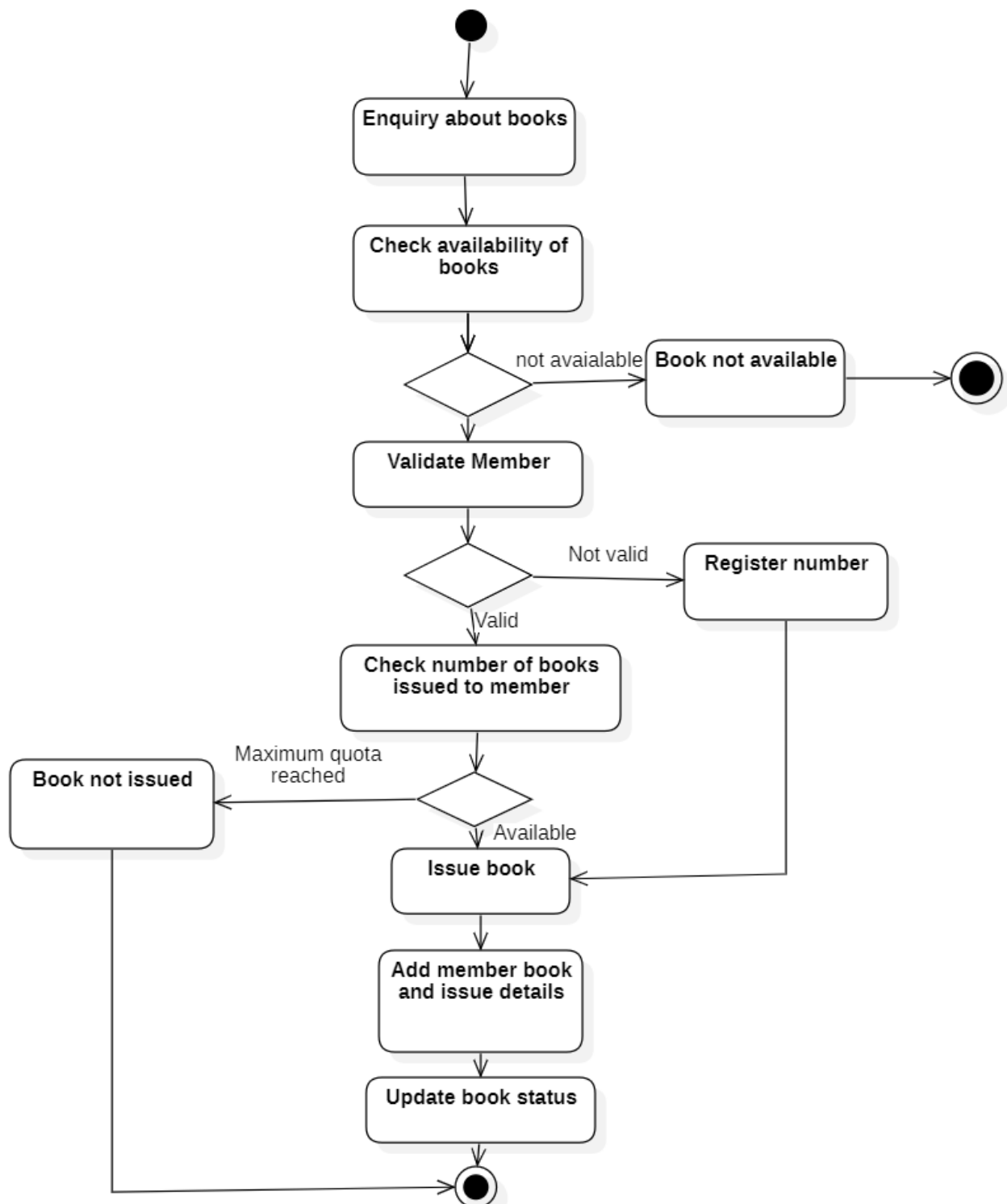
1.b) Class diagram:



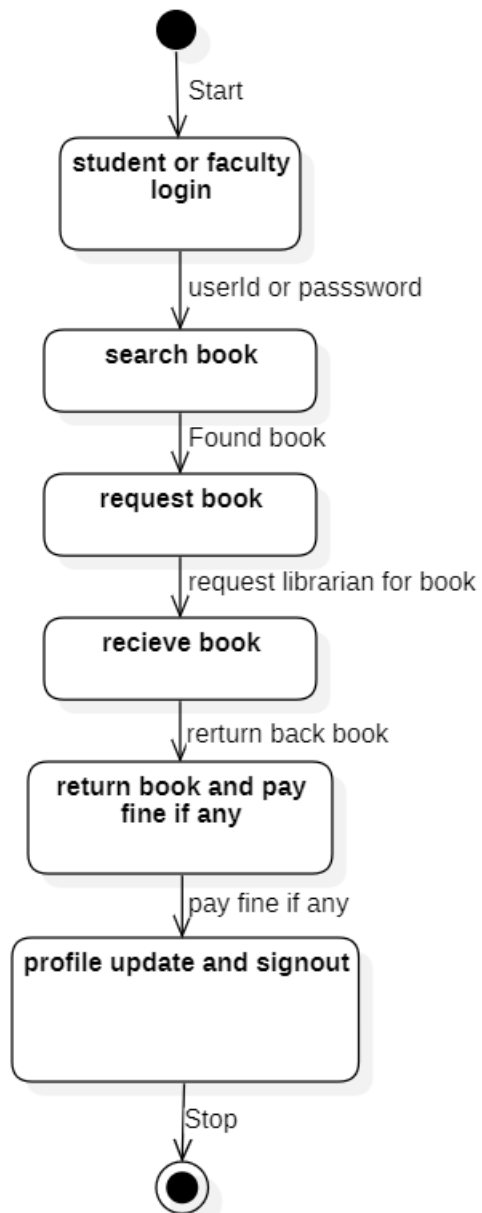
1.c) Sequence diagram:



1.d) Activity diagram:

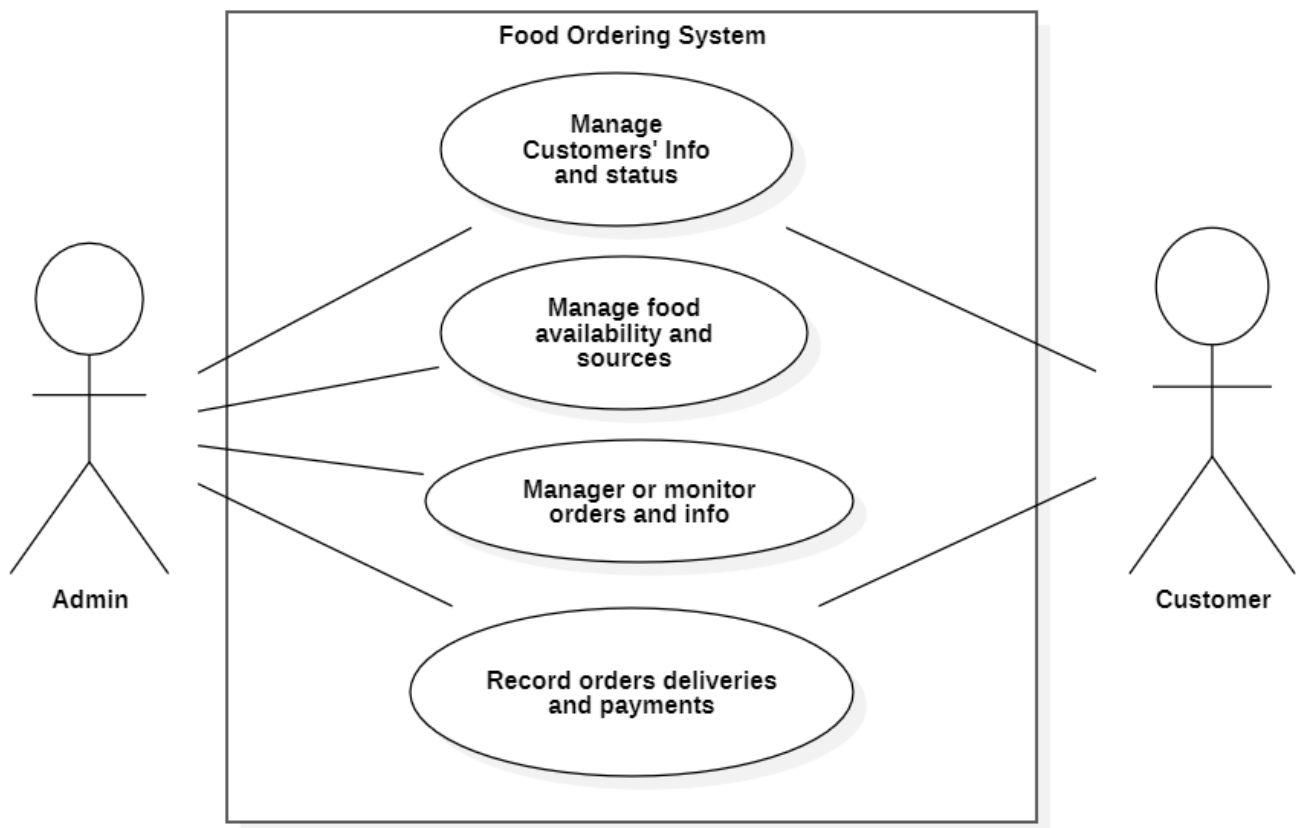


1.e) State diagram:

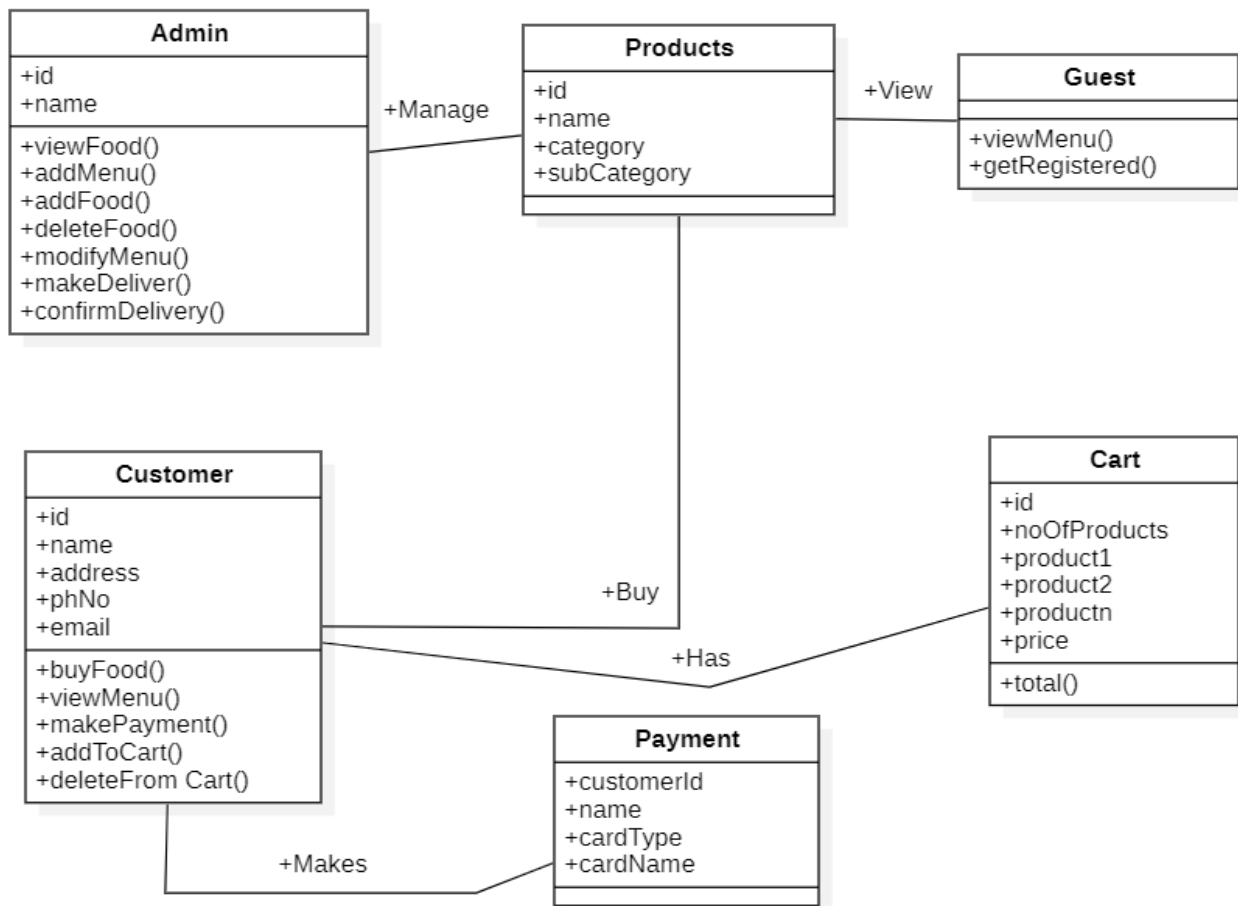


2. ONLINE FOOD ORDERING SYSTEM

2.a) Use case diagram:

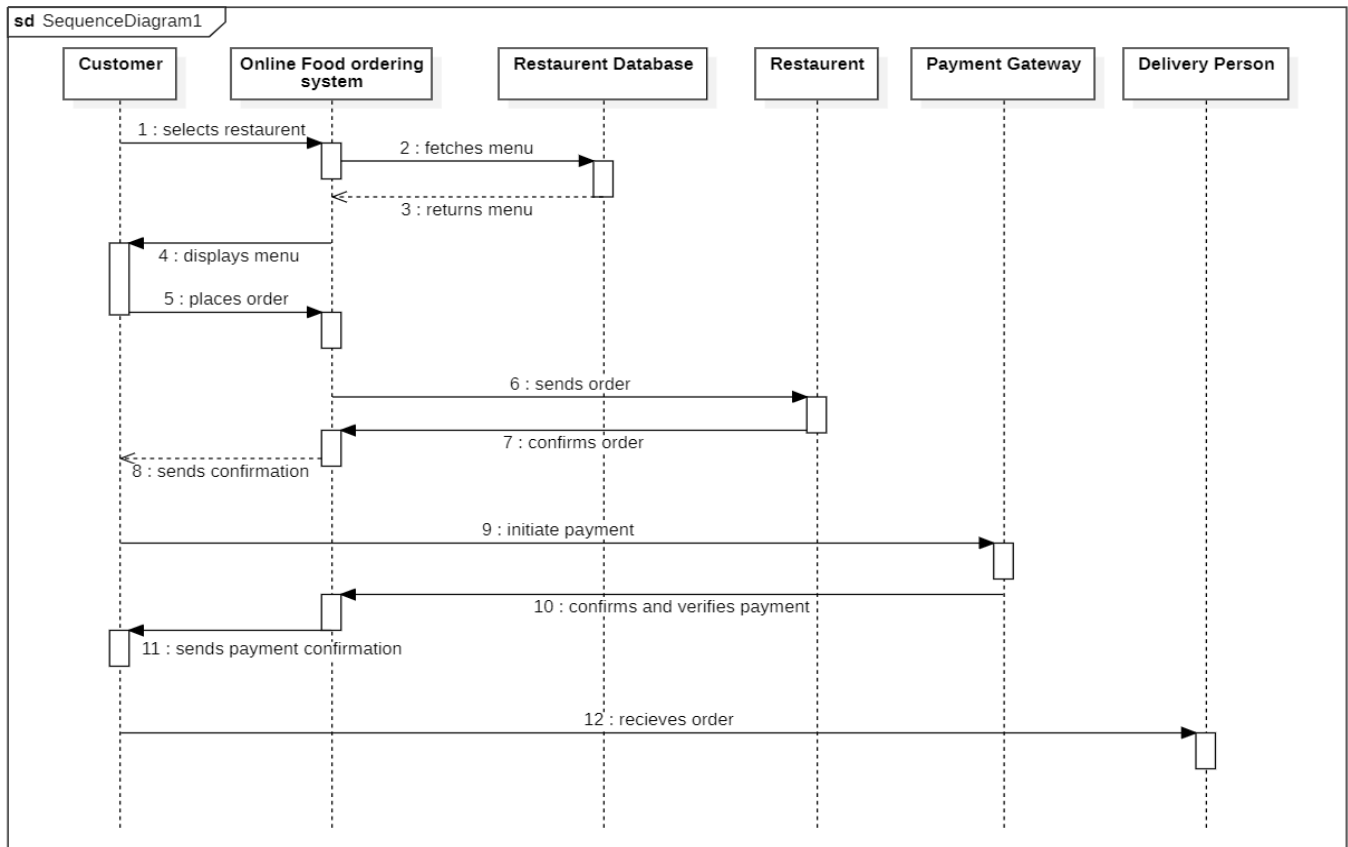


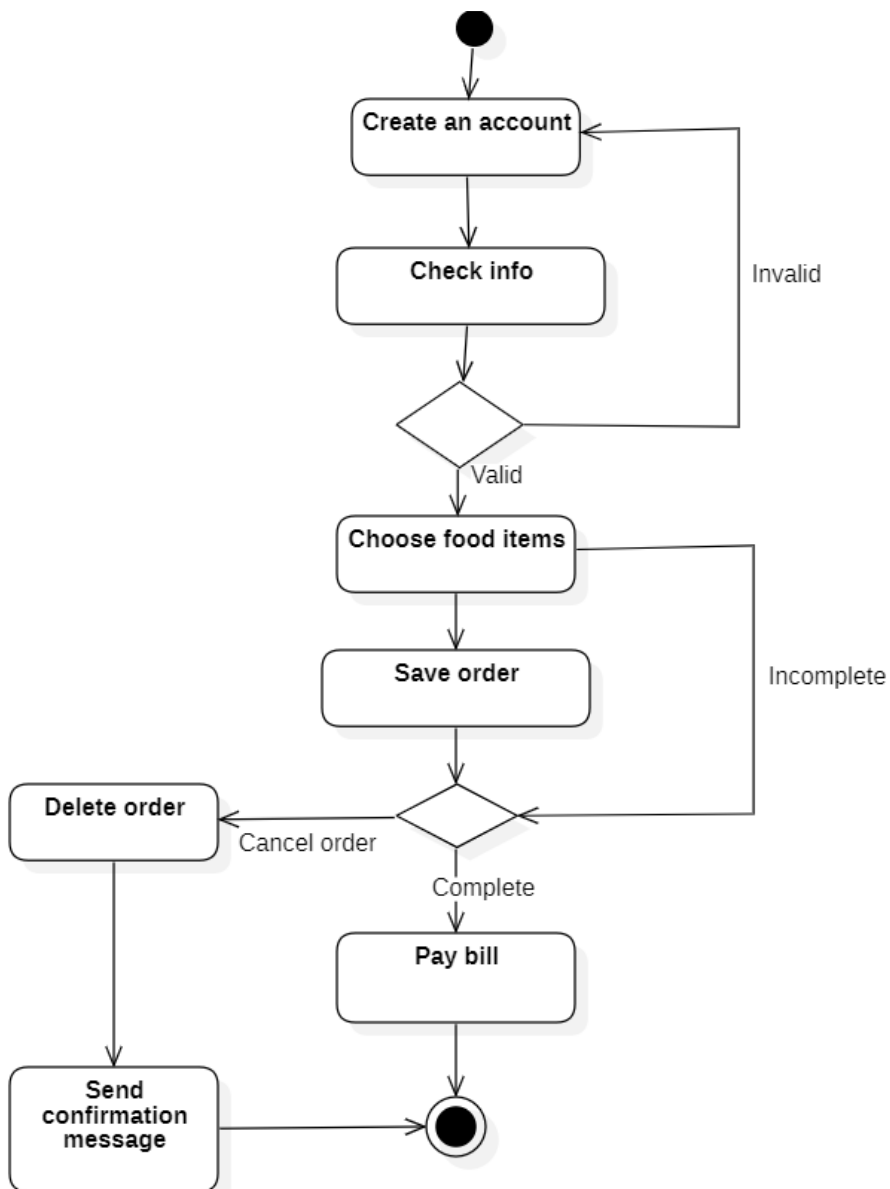
2.b) Class diagram:



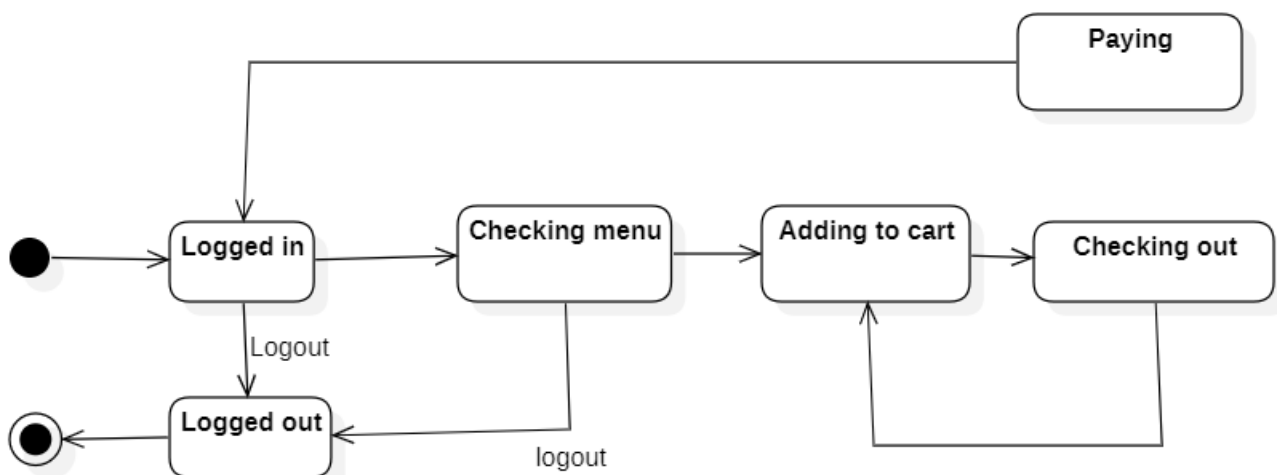
2.c) Sequence diagram:

2.d) Activity diagram:





2.e) State diagram:



3. BASIC JAVA PROGRAMS

3.a) Armstrong number:

```
import java.util.Scanner;
public class Armstrong {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        int original = num, sum = 0, digits = 0, temp = num;
        while (temp > 0) {
            temp /= 10;
            digits++;
        }
        temp = num;
        while (temp > 0) {
            int digit = temp % 10;
            sum += Math.pow(digit, digits);
            temp /= 10;
        }
        if (sum == original) {
            System.out.println(original + " is an Armstrong number.");
        } else {
            System.out.println(original + " is not an Armstrong number.");
        }
        sc.close();
    }
}
```

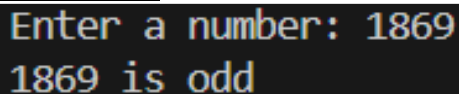
OUTPUT:

```
Enter a number: 789
789 is not an Armstrong number.
```

3.b) Checking of even or odd:

```
import java.util.Scanner;
public class EvenOdd {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        if (num % 2 == 0) {
            System.out.println(num + " is even");
        } else {
            System.out.println(num + " is odd");
        }
        sc.close();
    }
}
```

OUTPUT:



```
Enter a number: 1869
1869 is odd
```

3.c) Factorial:

```
import java.util.Scanner;
public class Factorial {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();
        long fact = 1;
        for (int i = 1; i <= n; i++) {
            fact *= i;
        }
        System.out.println("Factorial of " + n + " is: " + fact);
        sc.close();
    }
}
```

OUTPUT:

```
Enter a number: 6
Factorial of 6 is: 720
```

3.d) Fibonacci series:

```
import java.util.Scanner;
public class Fibonacci {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of terms: ");
        int n = sc.nextInt();
        int a = 0, b = 1, c;
        if (n <= 0) {
            System.out.println("Please enter a positive number.");
        } else if (n == 1) {
            System.out.println(a);
        } else {
            System.out.print(a + " " + b + " ");
            for (int i = 2; i < n; i++) {
                c = a + b;
                System.out.print(c + " ");
                a = b;
                b = c;
            }
        }
        sc.close();
    }
}
```

OUTPUT:

```
Enter number of terms: 8
0 1 1 2 3 5 8 13
```

3.e) Finding the largest number:

```
import java.util.Scanner;
public class LargestNumber {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter three numbers: ");
        int a = sc.nextInt();
        int b = sc.nextInt();
        int c = sc.nextInt();
        if (a >= b && a >= c) {
            System.out.println(a + " is the largest");
        } else if (b >= a && b >= c) {
            System.out.println(b + " is the largest");
        } else {
            System.out.println(c + " is the largest");
        }
        sc.close();
    }
}
```

OUTPUT:

```
Enter three numbers: 2323 343 948437
948437 is the largest
```


3.f) Finding the leap year:

```
import java.util.Scanner;
public class Leap {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a year: ");
        int year = sc.nextInt();
        if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
            System.out.println(year + " is a leap year.");
        } else {
            System.out.println(year + " is not a leap year.");
        }
        sc.close();
    }
}
```

OUTPUT:

```
Enter a year: 2025
2025 is not a leap year.
```

3.g) Number check:

```
import java.util.Scanner;
public class NumberCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        if (num > 0) {
            System.out.println("The number is positive");
        } else if (num < 0) {
            System.out.println("The number is negative");
        } else {
            System.out.println("The number is zero");
        }
        sc.close();
    }
}
```

OUTPUT:

```
Enter a number: -10
The number is negative
```

3.h) Palindrome checker:

```
import java.util.Scanner;
public class Palindrome {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        int original = num, rev = 0;
        while (num > 0) {
            int digit = num % 10;
            rev = rev * 10 + digit;
            num /= 10;
        }
        if (original == rev)
            System.out.println(original + " is a palindrome.");
        else
            System.out.println(original + " is not a palindrome.");
        sc.close();
    }
}
```

OUTPUT:

```
Enter a number: 767
767 is a palindrome.
```

3.i) Prime number:

```
import java.util.Scanner;
public class Prime {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        boolean isPrime = true;
        if (num <= 1) {
            isPrime = false;
        } else {
            for (int i = 2; i * i <= num; i++) {
                if (num % i == 0) {
                    isPrime = false;
                    break;
                }
            }
        }
        if (isPrime)
            System.out.println(num + " is a prime number.");
        else
            System.out.println(num + " is not a prime number.");

        sc.close();
    }
}
```

OUTPUT:

```
Enter a number: 8
8 is not a prime number.
```

3.j) Multiplication table:

```
import java.util.Scanner;
public class Table {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        for (int i = 1; i <= 10; i++) {
            System.out.println(num + " x " + i + " = " + (num * i));
        }
        sc.close();
    }
}
```

OUTPUT:

```
Enter a number: 8
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80
```

INHERITANCE

4. SINGLE LEVEL INHERITANCE

4.a) Parent class:

```
class Parent {  
    void show() {  
        System.out.println("This is the Parent class.");  
    }  
}
```

```
class Child extends Parent {  
    void display() {  
        System.out.println("This is the Child class.");  
    }  
}
```

```
public class SI1 {  
    public static void main(String[] args) {  
        Child obj = new Child();  
        obj.show();  
        obj.display();  
    }  
}
```

OUTPUT:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> javac SI1.java  
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> java SI1  
This is the Parent class.  
This is the Child class.
```

4.b) Employee class:

```
class Employee {  
    float salary = 50000;  
}
```

```
class Programmer extends Employee {  
    int bonus = 10000;  
}
```

```
public class SI2 {  
    public static void main(String[] args) {  
        Programmer p = new Programmer();  
        System.out.println("Programmer salary: " + p.salary);  
        System.out.println("Bonus: " + p.bonus);  
    }  
}
```

```
}
}
```

OUTPUT:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> javac SI2.java
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> java SI2
Programmer salary: 50000.0
Bonus: 10000
```

5. MULTILEVEL INHERITANCE

5.a) Vehicle class:

```
class Vehicle {
    void run() {
        System.out.println("Vehicle is running");
    }
}
```

```
class Car extends Vehicle {
    void speed() {
        System.out.println("Car is moving at 80 km/h.");
    }
}
```

```
class ElectricCar extends Car {
    void charge() {
        System.out.println("Electric car's maximum speed is 120 km/h.");
    }
}
```

```
public class MLI1 {
    public static void main(String[] args) {
        ElectricCar eCar = new ElectricCar();
        eCar.run();
        eCar.speed();
        eCar.charge();
    }
}
```

OUTPUT:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> javac MLI1.java
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> java MLI1
Vehicle is running
Car is moving at 80 km/h.
Electric car's maximum speed is 120 km/h.
```

5.b) Animal class:

```
class Animal {  
    void eat() {  
        System.out.println("Animals eat food");  
    }  
}  
  
class Mammal extends Animal {  
    void breathe() {  
        System.out.println("Mammals dont lay eggs");  
    }  
}  
  
class Dog extends Mammal {  
    void bark() {  
        System.out.println("Dog barks");  
    }  
}  
  
public class MLI2 {  
    public static void main(String[] args) {  
        Dog d = new Dog();  
        d.eat();  
        d.breathe();  
        d.bark();  
    }  
}
```

OUTPUT:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> javac MLI2.java  
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> java MLI2  
Animals eat food  
Mammals dont lay eggs  
Dog barks
```

6. HIERARCHICAL INHERITANCE

6.a) Shape class:

```
class Shape {  
    void display() {  
        System.out.println("Shapes are from mathematics");  
    }  
}
```

```
class Circle extends Shape {
    void area() {
        System.out.println("Circle area =  $\pi*r*r$ ");
    }
}

class Rectangle extends Shape {
    void area() {
        System.out.println("Rectangle area = length*width");
    }
}

public class HI1 {
    public static void main(String[] args) {
        Circle c = new Circle();
        Rectangle r = new Rectangle();

        c.display();
        c.area();

        r.display();
        r.area();
    }
}
```

OUTPUT:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> javac HI1.java
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> java HI1
Shapes are from mathematics
Circle area =  $\pi*r*r$ 
Shapes are from mathematics
Rectangle area = length*width
```

6.b) Animal Hierarchal class:

```
class Animal {
    void sound() {
        System.out.println("Every animal has different sound");
    }
}

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



```
}  
  
class Cat extends Animal {  
    void meow() {  
        System.out.println("Cat meows");  
    }  
}  
  
public class HI2 {  
    public static void main(String[] args) {  
        Dog d = new Dog();  
        Cat c = new Cat();  
  
        d.sound();  
        d.bark();  
  
        c.sound();  
        c.meow();  
    }  
}
```

OUTPUT:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> javac HI2.java  
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> java HI2  
Every animal has different sound  
Dog barks  
Every animal has different sound  
Cat meows
```

7. HYBRID INHERITANCE

7.a) Person class:

```
class Person {  
    String name;  
    int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public void displayPerson() {  
        System.out.println("Name: " + name + ", Age: " + age);  
    }  
}
```

```
}  
  
class Employee extends Person {  
    double salary;  
  
    public Employee(String name, int age, double salary) {  
        super(name, age);  
        this.salary = salary;  
    }  
  
    public void displayEmployee() {  
        displayPerson();  
        System.out.println("Salary: $" + salary);  
    }  
}  
  
class Student extends Person {  
    String grade;  
  
    public Student(String name, int age, String grade) {  
        super(name, age);  
        this.grade = grade;  
    }  
  
    public void displayStudent() {  
        displayPerson();  
        System.out.println("Grade: " + grade);  
    }  
}  
  
class Intern extends Employee {  
    String internshipField;  
  
    public Intern(String name, int age, double salary, String internshipField) {  
        super(name, age, salary);  
        this.internshipField = internshipField;  
    }  
  
    public void displayIntern() {  
        displayEmployee();  
        System.out.println("Internship Field: " + internshipField);  
    }  
}
```

```
public class HY1 {  
    public static void main(String[] args) {  
        Intern intern = new Intern("Alice", 22, 3000, "Software Development");  
        intern.displayIntern();  
    }  
}
```

OUTPUT:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> javac HY1.java  
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> java HY1  
Name: Alice, Age: 22  
Salary: $3000.0  
Internship Field: Software Development
```

```
7.b) Grandparent class:  
class Grandparent {  
    void grandparentMethod() {  
        System.out.println("This is a grandparent");  
    }  
}  
  
class Parent1 extends Grandparent {  
    void parent1Method() {  
        System.out.println("This is from Parent1");  
    }  
}  
  
class Parent2 extends Grandparent {  
    void parent2Method() {  
        System.out.println("This is from Parent2");  
    }  
}  
  
class Child extends Parent1 {  
    Parent2 p2 = new Parent2();  
  
    void childMethod() {  
        System.out.println("This is the child class method");  
    }  
  
    void useParent2Methods() {  
        p2.parent2Method();  
    }  
}
```

```
}  
  
public class HY2 {  
    public static void main(String[] args) {  
        Child obj = new Child();  
        obj.grandparentMethod();  
        obj.parent1Method();  
        obj.useParent2Methods();  
        obj.childMethod();  
    }  
}
```

OUTPUT:

```
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> javac HY2.java  
PS C:\Users\Rocki\OneDrive\Documents\College\SEM-2\JAVA\Java\Inheritance> java HY2  
This is a grandparent  
This is from Parent1  
This is from Parent2  
This is the child class method
```

POLYMORPHISM

8. CONSTRUCTOR

8.a) Constructor 1:

```
class Student {  
    String name;  
    int rollNumber;  
  
    public Student(String name, int rollNumber) {  
        this.name = name;  
        this.rollNumber = rollNumber;  
    }  
  
    public void display() {  
        System.out.println("Student Name: " + name);  
        System.out.println("Roll Number: " + rollNumber);  
    }  
  
    public static void main(String[] args) {  
        Student s1 = new Student("Alice", 101);  
        s1.display();  
    }  
}
```

OUTPUT:

```
Student Name: Alice  
Roll Number: 101
```

8.b) Constructor 2:

```
class Rectangle {  
    double length, breadth;  
  
    // Constructor  
    public Rectangle(double length, double breadth) {  
        this.length = length;  
        this.breadth = breadth;  
    }  
  
    // Method to calculate and return area  
    public double getArea() {  
        return length * breadth;  
    }  
  
    public static void main(String[] args) {  
        Rectangle rect = new Rectangle(5.0, 3.0);  
        System.out.println("Area of Rectangle: " + rect.getArea());  
    }  
}
```

OUTPUT:

```
Area of Rectangle: 15.0
```

9. CONSTRUCTOR OVERLOADING

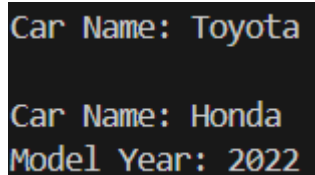
9.a) Constructor overloading 1:

```
class Car {  
    String name;  
    int modelYear;  
  
    public Car(String name) {  
        this.name = name;  
        this.modelYear = 0;  
    }  
  
    public Car(String name, int modelYear) {  
        this.name = name;  
        this.modelYear = modelYear;  
    }  
}
```

```
public void display() {
    System.out.println("Car Name: " + name);
    if (modelYear != 0)
        System.out.println("Model Year: " + modelYear);
}

public static void main(String[] args) {
    Car car1 = new Car("Toyota");
    Car car2 = new Car("Honda", 2022);

    car1.display();
    System.out.println();
    car2.display();
}
}
```

OUTPUT:A screenshot of a terminal window showing the output of the Java program. The text is displayed in a monospaced font with syntax highlighting: 'Car Name: Toyota' on the first line, 'Car Name: Honda' on the second line, and 'Model Year: 2022' on the third line.

```
Car Name: Toyota
Car Name: Honda
Model Year: 2022
```

9.b) Constructor overloading 2:

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

    public BankAccount(String accountHolder) {
        this.accountHolder = accountHolder;
        this.balance = 0.0;
    }

    public BankAccount(String accountHolder, double balance) {
        this.accountHolder = accountHolder;
        this.balance = balance;
    }

    public void display() {
        System.out.println("Account Holder: " + accountHolder);
        System.out.println("Balance: $" + balance);
    }
}
```

```
public static void main(String[] args) {  
    BankAccount acc1 = new BankAccount("John Doe");  
    BankAccount acc2 = new BankAccount("Jane Doe", 5000.0);  
  
    acc1.display();  
    System.out.println();  
    acc2.display();  
}  
}
```

OUTPUT:

```
Account Holder: John Doe  
Balance: $0.0  
  
Account Holder: Jane Doe  
Balance: $5000.0
```

10. METHOD OVERLOADING

10.a) Method overloading 1:

```
class Calculator {  
    public int add(int a, int b) {  
        return a + b;  
    }  
  
    public double add(double a, double b, double c) {  
        return a + b + c;  
    }  
  
    public static void main(String[] args) {  
        Calculator calc = new Calculator();  
        System.out.println("Sum of 5 and 10: " + calc.add(5, 10));  
        System.out.println("Sum of 2.5, 3.5, and 4.0: " + calc.add(2.5, 3.5, 4.0));  
    }  
}
```

OUTPUT:

```
Sum of 5 and 10: 15  
Sum of 2.5, 3.5, and 4.0: 10.0
```

10.b) Method overloading 2:

```
class Display {  
    public void show(int number) {
```

```
    System.out.println("Integer: " + number);
}

public void show(String message) {
    System.out.println("Message: " + message);
}

public static void main(String[] args) {
    Display obj = new Display();
    obj.show(42);
    obj.show("Polymorphism");
}
}
```

OUTPUT:

```
Integer: 42
Message: Polymorphism
```

11. METHOD OVERRIDING

11.a) Method overriding 1:

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

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

    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        myAnimal.makeSound();

        Dog myDog = new Dog();
        myDog.makeSound();
    }
}
```


OUTPUT:

```
Animal makes a sound
Dog barks
```

11.b) Method overriding 2:

```
class Bank {
    public double getInterestRate() {
        return 5.0;
    }
}
```

```
class SBI extends Bank {
    @Override
    public double getInterestRate() {
        return 6.5;
    }
}
```

```
class HDFC extends Bank {
    @Override
    public double getInterestRate() {
        return 7.0;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Bank b = new Bank();
        SBI sbi = new SBI();
        HDFC hdfc = new HDFC();

        System.out.println("Bank Interest Rate: " + b.getInterestRate() + "%");
        System.out.println("SBI Interest Rate: " + sbi.getInterestRate() + "%");
        System.out.println("HDFC Interest Rate: " + hdfc.getInterestRate() + "%");
    }
}
```

OUTPUT:

```
Bank Interest Rate: 5.0%
SBI Interest Rate: 6.5%
HDFC Interest Rate: 7.0%
```

ABSTRACTION

12. INTERFACE

12.a) Interface 1:

```
interface Printable {  
    void print();  
}
```

```
class Document implements Printable {  
    public void print() {  
        System.out.println("Printing the document");  
    }  
  
    public static void main(String[] args) {  
        Document doc = new Document();  
        doc.print();  
    }  
}
```

OUTPUT:

```
Printing the document
```

12.b) Interface 2:

```
interface Vehicle {  
    void drive();  
}
```

```
interface ElectricVehicle {  
    void chargeBattery();  
}
```

```
class Tesla implements Vehicle, ElectricVehicle {  
    public void drive() {  
        System.out.println("Tesla is driving silently...");  
    }  
  
    public void chargeBattery() {  
        System.out.println("Tesla battery is charging...");  
    }  
  
    public static void main(String[] args) {  
        Tesla myTesla = new Tesla();  
        myTesla.drive();  
    }  
}
```

```
    myTesla.chargeBattery();  
  }  
}
```

OUTPUT:

```
Tesla is driving silently...  
Tesla battery is charging...
```

12.c) Interface 3:

```
interface Sports {  
    void play();  
}  
  
class Football implements Sports {  
    public void play() {  
        System.out.println("Playing Football...");  
    }  
}  
  
class Cricket implements Sports {  
    public void play() {  
        System.out.println("Playing Cricket...");  
    }  
}  
  
public class SportsTest {  
    public static void main(String[] args) {  
        Sports football = new Football();  
        Sports cricket = new Cricket();  
  
        football.play();  
        cricket.play();  
    }  
}
```

OUTPUT:

```
Playing Football...  
Playing Cricket...
```

12.d) Interface 4:

```
interface BankAccount {
    void deposit(double amount);
    void withdraw(double amount);
    double getBalance();
}

class SavingsAccount implements BankAccount {
    private double balance;

    public SavingsAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: $" + amount);
    }

    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawn: $" + amount);
        } else {
            System.out.println("Insufficient balance!");
        }
    }

    public double getBalance() {
        return balance;
    }

    public static void main(String[] args) {
        SavingsAccount account = new SavingsAccount(1000);
        account.deposit(500);
        account.withdraw(300);
        System.out.println("Current Balance: $" + account.getBalance());
    }
}
```

OUTPUT:

```
Deposited: $500.0  
Withdrawn: $300.0  
Current Balance: $1200.0
```

13. ABSTRACT CLASS

13.a) Abstract class 1:

```
abstract class Animal {  
    abstract void makeSound();  
}  
  
class Dog extends Animal {  
    public void makeSound() {  
        System.out.println("Dog barks");  
    }  
}  
  
class Cat extends Animal {  
    public void makeSound() {  
        System.out.println("Cat meows");  
    }  
}  
  
public class AnimalTest {  
    public static void main(String[] args) {  
        Animal dog = new Dog();  
        Animal cat = new Cat();  
  
        dog.makeSound();  
        cat.makeSound();  
    }  
}
```

OUTPUT:

```
Dog barks  
Cat meows
```

13.b) Abstract class 2:

```
abstract class Shape {  
    abstract double calculateArea();  
}
```

```
class Circle extends Shape {
    double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public double calculateArea() {
        return Math.PI * radius * radius;
    }
}

class Rectangle extends Shape {
    double length, breadth;

    public Rectangle(double length, double breadth) {
        this.length = length;
        this.breadth = breadth;
    }

    public double calculateArea() {
        return length * breadth;
    }
}

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

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

OUTPUT:

```
Circle Area: 78.53981633974483
Rectangle Area: 24.0
```

13.c) Abstract class 3:

```
abstract class Employee {
    abstract double calculateSalary();
}
```

```
class FullTimeEmployee extends Employee {
    private double monthlySalary;

    public FullTimeEmployee(double salary) {
        this.monthlySalary = salary;
    }

    public double calculateSalary() {
        return monthlySalary;
    }
}

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

    public PartTimeEmployee(int hoursWorked, double hourlyRate) {
        this.hoursWorked = hoursWorked;
        this.hourlyRate = hourlyRate;
    }

    public double calculateSalary() {
        return hoursWorked * hourlyRate;
    }
}

public class EmployeeTest {
    public static void main(String[] args) {
        Employee fullTime = new FullTimeEmployee(5000);
        Employee partTime = new PartTimeEmployee(20, 15);

        System.out.println("Full-time Salary: $" + fullTime.calculateSalary());
        System.out.println("Part-time Salary: $" + partTime.calculateSalary());
    }
}
```

OUTPUT:

```
Full-time Salary: $5000.0
Part-time Salary: $300.0
```

13.d) Abstract class 4:
abstract class Vehicle {

```
    abstract void start();
}

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

class Bike extends Vehicle {
    public void start() {
        System.out.println("Bike is starting with a self-start button...");
    }
}

public class VehicleTest {
    public static void main(String[] args) {
        Vehicle car = new Car();
        Vehicle bike = new Bike();

        car.start();
        bike.start();
    }
}
```

OUTPUT:

```
Car is starting with a key...
Bike is starting with a self-start button...
```


ENCAPSULATION

14. ENCAPSULATION PROGRAMS

14.a) Encapsulation 1:

```
class BankAccount {
    private String accountNumber;
    private double balance;

    public BankAccount(String accountNumber, double balance) {
        this.accountNumber = accountNumber;
        if (balance >= 0) {
            this.balance = balance;
        } else {
            this.balance = 0;
            System.out.println("Balance cannot be negative. Setting to 0.");
        }
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: $" + amount);
        } else {
            System.out.println("Invalid deposit amount.");
        }
    }

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

    public double getBalance() {
        return balance;
    }

    public String getAccountNumber() {
        return accountNumber;
    }
}
```

```
public static void main(String[] args) {  
    BankAccount account = new BankAccount("123456789", 500);  
    account.deposit(200);  
    account.withdraw(100);  
    System.out.println("Final Balance: $" + account.getBalance());  
}  
}
```

OUTPUT:

```
Deposited: $200.0  
Withdrawn: $100.0  
Final Balance: $600.0
```

14.b) Encapsulation 2:

```
class Student {  
    private String name;  
    private int rollNumber;  
    private int marks;  
  
    public Student(String name, int rollNumber, int marks) {  
        this.name = name;  
        this.rollNumber = rollNumber;  
        setMarks(marks);  
    }  
  
    public void setMarks(int marks) {  
        if (marks >= 0 && marks <= 100) {  
            this.marks = marks;  
        } else {  
            System.out.println("Invalid marks! Setting marks to 0.");  
            this.marks = 0;  
        }  
    }  
  
    public int getMarks() {  
        return marks;  
    }  
  
    public void display() {  
        System.out.println("Name: " + name + ", Roll Number: " + rollNumber + ", Marks: "  
+ marks);  
    }  
}
```

```
public static void main(String[] args) {  
    Student student = new Student("Alice", 101, 95);  
    student.display();  
}  
}
```

OUTPUT:

```
Name: Alice, Roll Number: 101, Marks: 95
```

14.c) Encapsulation 3:

```
class Employee {  
    private String name;  
    private int id;  
    private double salary;  
  
    public Employee(String name, int id, double salary) {  
        this.name = name;  
        this.id = id;  
        setSalary(salary);  
    }  
  
    public void setSalary(double salary) {  
        if (salary >= 0) {  
            this.salary = salary;  
        } else {  
            System.out.println("Salary cannot be negative. Setting to 0.");  
            this.salary = 0;  
        }  
    }  
  
    public double getSalary() {  
        return salary;  
    }  
  
    public void display() {  
        System.out.println("Employee Name: " + name + ", ID: " + id + ", Salary: $" +  
salary);  
    }  
  
    public static void main(String[] args) {  
        Employee emp = new Employee("John", 1001, 5000);  
        emp.display();  
    }  
}
```

```
}  
}
```

OUTPUT:

```
Employee Name: John, ID: 1001, Salary: $5000.0
```

14.d) Encapsulation 4:

```
class Car {  
    private String brand;  
    private String model;  
    private double price;  
  
    public Car(String brand, String model, double price) {  
        this.brand = brand;  
        this.model = model;  
        setPrice(price);  
    }  
  
    public void setPrice(double price) {  
        if (price > 0) {  
            this.price = price;  
        } else {  
            System.out.println("Price cannot be zero or negative. Setting to default  
$10,000.");  
            this.price = 10000;  
        }  
    }  
  
    public double getPrice() {  
        return price;  
    }  
  
    public void display() {  
        System.out.println("Car Brand: " + brand + ", Model: " + model + ", Price: $" +  
price);  
    }  
  
    public static void main(String[] args) {  
        Car car = new Car("Tesla", "Model S", 79999);  
        car.display();  
    }  
}
```

OUTPUT:

```
Car Brand: Tesla, Model: Model S, Price: $79999.0
```

15. PACKAGES [USER-DEFINED AND BUILT-IN]

15.a) User-defined 1:

CODE-1:

```
package package1;
```

```
public class Calculator {  
    public int add(int a, int b) {  
        return a + b;  
    }  
  
    public int subtract(int a, int b) {  
        return a - b;  
    }  
  
    public int multiply(int a, int b) {  
        return a * b;  
    }  
}
```

CODE-2:

```
import package1.Calculator;
```

```
public class PackageTest {  
    public static void main(String[] args) {  
        Calculator calc = new Calculator();  
        System.out.println("Addition: " + calc.add(5, 3));  
        System.out.println("Subtraction: " + calc.subtract(8, 2));  
        System.out.println("Multiplication: " + calc.multiply(4, 7));  
    }  
}
```

OUTPUT:

```
Addition: 8  
Subtraction: 6  
Multiplication: 28
```

15.b) User-defined 2:

CODE-1:

```
package package2;
```

```
public class EmployeeDetails {  
    private String name;  
    private int id;  
    private double salary;  
  
    public EmployeeDetails(String name, int id, double salary) {  
        this.name = name;  
        this.id = id;  
        this.salary = salary;  
    }  
  
    public void display() {  
        System.out.println("Employee: " + name + ", ID: " + id + ", Salary: $" + salary);  
    }  
}
```

CODE-2:

```
import package2.EmployeeDetails;
```

```
public class EmployeeTest {  
    public static void main(String[] args) {  
        EmployeeDetails emp = new EmployeeDetails("John Doe", 101, 6000);  
        emp.display();  
    }  
}
```

OUTPUT:

```
Employee: John Doe, ID: 101, Salary: $6000.0
```

15.c) Built-in 1:

```
import java.util.Scanner;  
import java.io.FileWriter;  
import java.io.IOException;
```

```
public class FileWriteExample {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Enter some text to save to a file:");  
        String text = scanner.nextLine();  
    }  
}
```

```
try {
    FileWriter writer = new FileWriter("output.txt");
    writer.write(text);
    writer.close();
    System.out.println("Text written to file successfully!");
} catch (IOException e) {
    System.out.println("Error occurred while writing to file.");
}

scanner.close();
}
```

OUTPUT:

```
Enter some text to save to a file:
Hi i am god speaking
Text written to file successfully!
```

15.d) Built-in 2:

```
import java.util.Scanner;
import java.net.URL;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class URLReaderExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a website URL (e.g., https://www.example.com): ");
        String urlString = scanner.nextLine();

        try {
            URL url = new URL(urlString);
            BufferedReader reader = new BufferedReader(new
InputStreamReader(url.openStream()));

            String line;
            System.out.println("Fetching data from " + urlString + "...");
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
            reader.close();
        }
```

```

    } catch (Exception e) {
        System.out.println("Error fetching data from the URL.");
    }

    scanner.close();
}
}

```

OUTPUT:

```

Enter a website URL (e.g., https://www.example.com):
https://www.google.com
Fetching data from https://www.google.com...
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta content=
"/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><title>Google</title><script nonce="-LnY0j1ZgSjtelAF60bC9w">(function(){var _g={kEI:'uMLvZ9
OMFvq11sQPmujp-AQ',kEXP1:'0,202791,3497503,655,435,538661,94918,344796,247320,42724,11106,5230575,58,15,34,36812536,25275060,91889,8182,5928,22918,42258,6749,23878,9139,46

```

16. EXCEPTION HANDLING

16.a) Exception 1:

```

public class ArrayException{
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3};

        try {
            System.out.println(numbers[5]); // Accessing invalid index
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Index out of bounds!");
        }

        System.out.println("Program continues...");
    }
}

```

OUTPUT:

```

Error: Index out of bounds!
Program continues...

```

16.b) Exception 2:

```

class AgeException extends Exception {
    public AgeException(String message) {
        super(message);
    }
}

public class CustomExceptionExample {
    static void checkAge(int age) throws AgeException {

```



```
if (age < 18) {
    throw new AgeException("Not eligible to vote!");
} else {
    System.out.println("Eligible to vote.");
}
}

public static void main(String[] args) {
    try {
        checkAge(16); // This will throw an exception
    } catch (AgeException e) {
        System.out.println("Exception: " + e.getMessage());
    }
}
}
```

OUTPUT:

Exception: Not eligible to vote!

16.c) Exception 3:

```
public class DivideByZero {
    public static void main(String[] args) {
        try {
            int result = 10 / 0; // This will cause ArithmeticException
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero!");
        }
        System.out.println("Program continues...");
    }
}
```

OUTPUT:

Error: Cannot divide by zero!
Program continues...

16.d) Exception 4:

```
public class FinallyExample {
    public static void main(String[] args) {
        try {
            int num = 10 / 2;
            System.out.println("Result: " + num);
        } catch (ArithmeticException e) {
```

```

    System.out.println("Error: Division by zero!");
} finally {
    System.out.println("This will always execute.");
}

System.out.println("Program continues...");
}
}

```

OUTPUT:

```

Result: 5
This will always execute.
Program continues...

```

17. FILE HANDLING

17.a) File 1:

```

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

```

```

public class AppendFile {
    public static void main(String[] args) {
        try (FileWriter writer = new FileWriter("test.txt", true)) {
            writer.write("\nAppending new text!");
            System.out.println("Successfully appended to the file.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

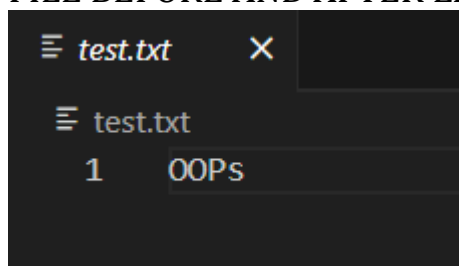
OUTPUT:

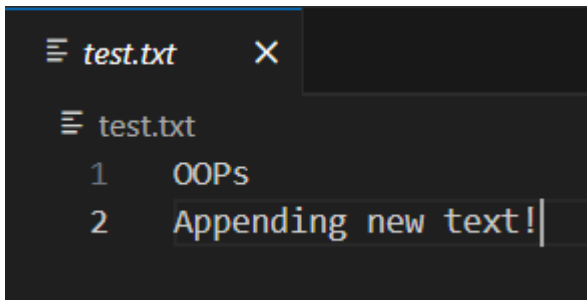
```

Successfully appended to the file.

```

FILE BEFORE AND AFTER EXECUTING:





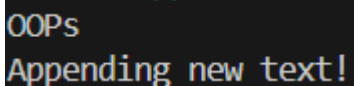
```
test.txt
1 OOPs
2 Appending new text!
```

17.b) File 2:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ReadFile {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(new File("test.txt"))) {
            while (scanner.hasNextLine()) {
                System.out.println(scanner.nextLine());
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:



```
OOPs
Appending new text!
```

17.c) File 3:

```
import java.io.*;
import java.nio.file.*;

public class ReplaceWord {
    public static void main(String[] args) {
        String filePath = "test1.txt";
        String oldWord = "Java";
        String newWord = "Python";

        try {
            String content = new String(Files.readAllBytes(Paths.get(filePath)));
            content = content.replaceAll(oldWord, newWord);
            Files.write(Paths.get(filePath), content.getBytes());
        }
    }
}
```

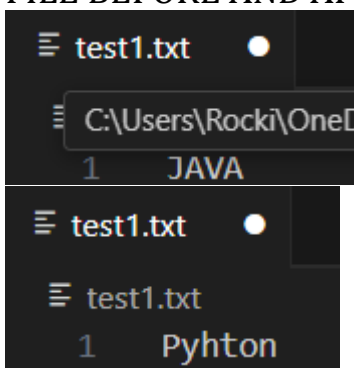
```

        System.out.println("Replaced all occurrences of '" + oldWord + "' with '" +
newWord + "'");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```

OUTPUT:

Replaced all occurrences of 'Java' with 'Python'.

FILE BEFORE AND AFTER EXECUTING:**17.d) File 4:**

```

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

```

```


public class WriteFile {
    public static void main(String[] args) {
        try (FileWriter writer = new FileWriter("test.txt")) {
            writer.write("Hello, Java File Handling!");
            System.out.println("Successfully wrote to the file.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

OUTPUT:


Successfully wrote to the file.

FILE BEFORE AND AFTER EXECUTING:

 test.txt

1 OOPs

2 Appending new text!

 test.txt

1 Hello, Java File Handling!