



**SCHOOL OF  
COMPUTING**

Susendran M  
CH.SC.U4CSE24154  
OBJECT ORIENTED PROGRAMMING  
(23CSE111)  
LAB RECORD



**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.U4CSE24154 – Susendran M** 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

Internal Examiner 1

Internal Examiner 2

# INDEX

S.NO	TITLE	PAGE.NO
<b>UML DIAGRAM</b>		
<b>1.</b>	<b>ONLINE SHOPPING</b>	
	1.a) Use Case Diagram	6
	1.b) Class Diagram	7
	1.c) Sequence Diagram	8
	1.d) State Diagram	9
	1.e) Activity Diagram	10
<b>2.</b>	<b>LIBRARY MANAGEMENT SYSTEM</b>	
	2.a) Use Case Diagram	11
	2.b) Class Diagram	12
	2.c) Sequence Diagram	13
	2.d) State Diagram	14
	2.e) Activity Diagram	15
<b>3.</b>	<b>BASIC JAVA PROGRAMS</b>	
	3.a) Sum of Digits	16
	3.b) Reverse Number	17
	3.c) Prime Number	18
	3.d) Palindrome Number	20
	3.e) Lower Triangle	22
	3.f) LCM Numbers	23
	3.g) Fibonacci Series	25
	3.h) Factorial Number	27
	3.i) Sum of Even, Odd Digits	28
	3.j) Armstrong Number	30
<b>INHERITANCE</b>		
<b>4.</b>	<b>SINGLE INHERITANCE PROGRAMS</b>	
	4.a) Class Manager extends Employee	32

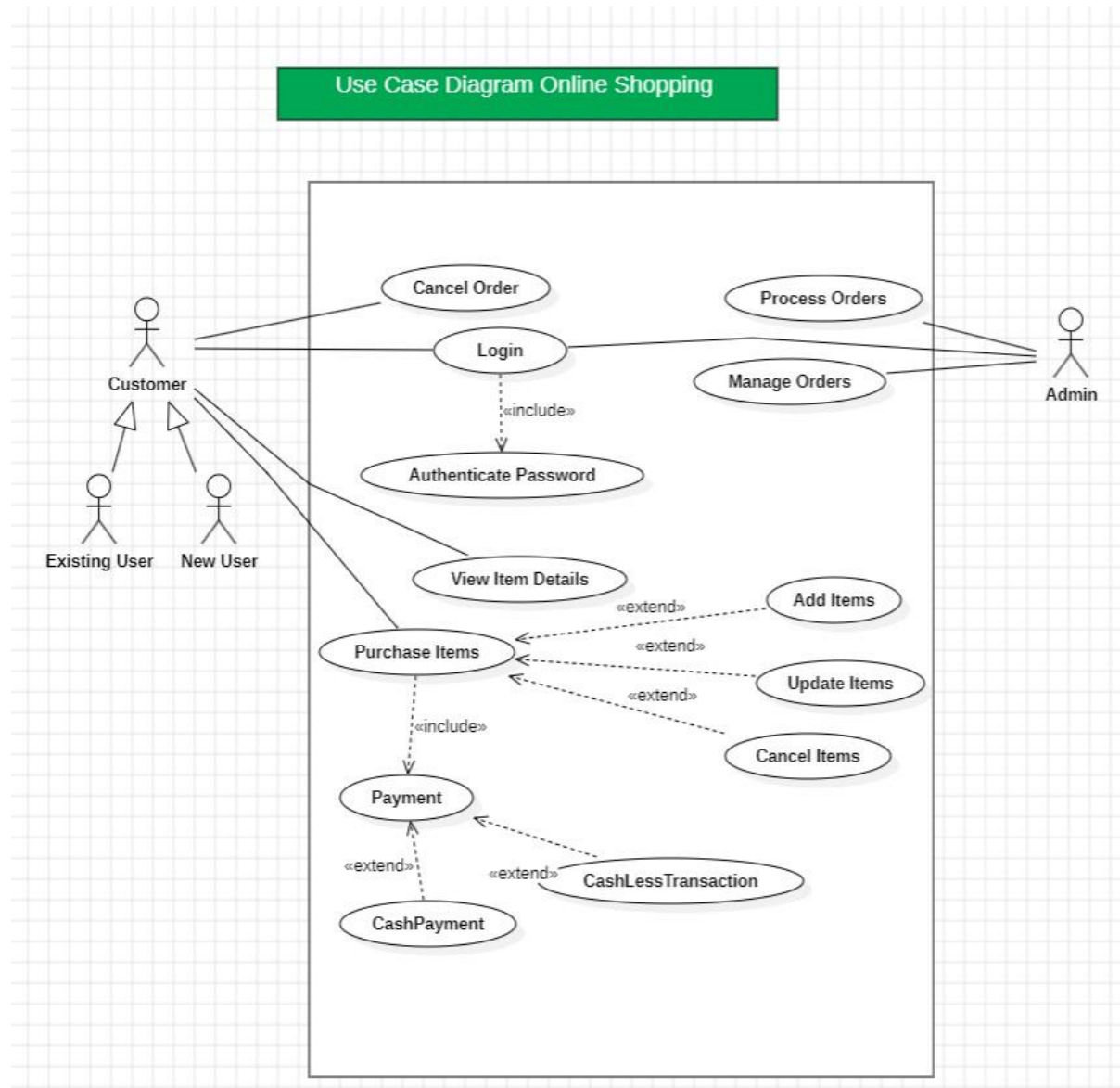
	4.b) Class Batsman extends Cricket	33
<b>5.</b>	<b>MULTILEVEL INHERITANCE PROGRAMS</b>	
	5.a) Class CEO extends Manager	34
	5.b) Class ALLRounder extends Batsman	36
<b>6.</b>	<b>HEIRARCHIAL INHERITANCE PROGRAMS</b>	
	6.a) Class Manager and Developer extends Employee	37
	6.b) Class Batsman and Bowler extends Cricket	39
<b>7.</b>	<b>HYBRID INHERITANCE PROGRAMS</b>	
	7.a) Class TechLead extends Developer	41
	7.b) Class WicketKeeper extends Batsman	43
<b>POLYMORPHISM</b>		
<b>8.</b>	<b>CONSTRUCTOR OVERLOADING PROGRAMS</b>	
	8.a) Constructor constructHouse	46
<b>9.</b>	<b>CONSTRUCTOR OVERRIDING PROGRAMS</b>	
	9.a) Constructor ManufacturingMachine	48
<b>10.</b>	<b>METHOD OVERLOADING PROGRAMS</b>	
	10.a) Method constructHouse	51
	10.b) Method ManufacturingMachine	53
<b>11.</b>	<b>METHOD OVERRIDING PROGRAMS</b>	
	11.a) Method construct	56
	11.b) Method operate	58
<b>ABSTRACTION</b>		
<b>12.</b>	<b>INTERFACE PROGRAMS</b>	
	12.a) Class ZeptoDelivery implements DeliveryService	61
	12.b) Class shapeG implements shapeY	62
	12.c) Class Cat implements Animal	63
	12.d) Class SmartTV implements Television	65
<b>13.</b>	<b>ABSTRACT CLASS PROGRAMS</b>	
	13.a) Class ZeptoDelivery extends DeliveryService	67

	13.b) Class ShapeG extends ShapeY	68
	13.c) Class Cat extends Animal	69
	13.d) Class SmartTV implements Television	70
<b>ENCAPSULATION</b>		
<b>14.</b>	<b>ENCAPSULATION PROGRAMS</b>	
	14.a) Class Student private variables SimpleGradebook	72
	14.b) Class Vehicle variables OdometerDemo	73
	14.c) Class TempConverter variables TempConverterDemo	75
	14.d) Class HealthMonitor variables HealthMonitorDemo	76
<b>15.</b>	<b>PACKAGE PROGRAMS</b>	
	15.a) Import Package vehicles, electronics Class VehicleSystem	78
	15.b) Import Package banking, transactions Class BankApplication	80
	15.c) Import io, lang, util Class UserInputFile	82
	15.d) Import awt, net, sql Class Packed	82
<b>16.</b>	<b>EXCEPTION HANDLING PROGRAMS</b>	
	16.a) NumberFormat, Arithmetic, Exception with try, catch	85
	16.b) IllegalArgumentException, InputMismatch, Arithmetic, Exception with try and catch	86
	16.c) InvalidAge, Exception with throw	87
	16.d)	88
<b>17.</b>	<b>FILE HANDLING PROGRAMS</b>	
	17.a) FileWriter example.txt	90
	17.b) BufferedWriter output.txt	90
	17.c) FileReader example.txt	91
	17.d) BufferedReader example.txt	92

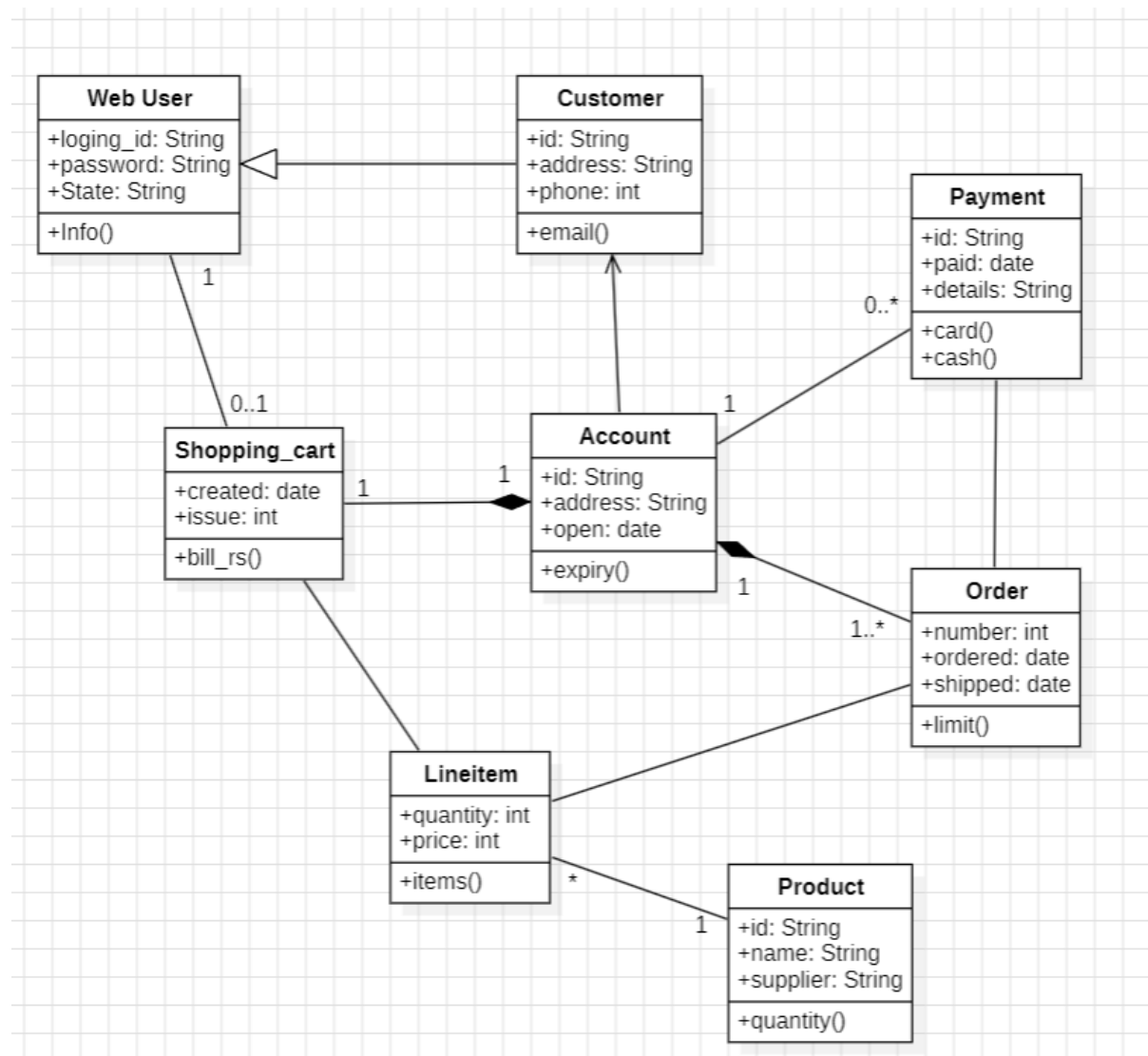
# UML DIAGRAMS

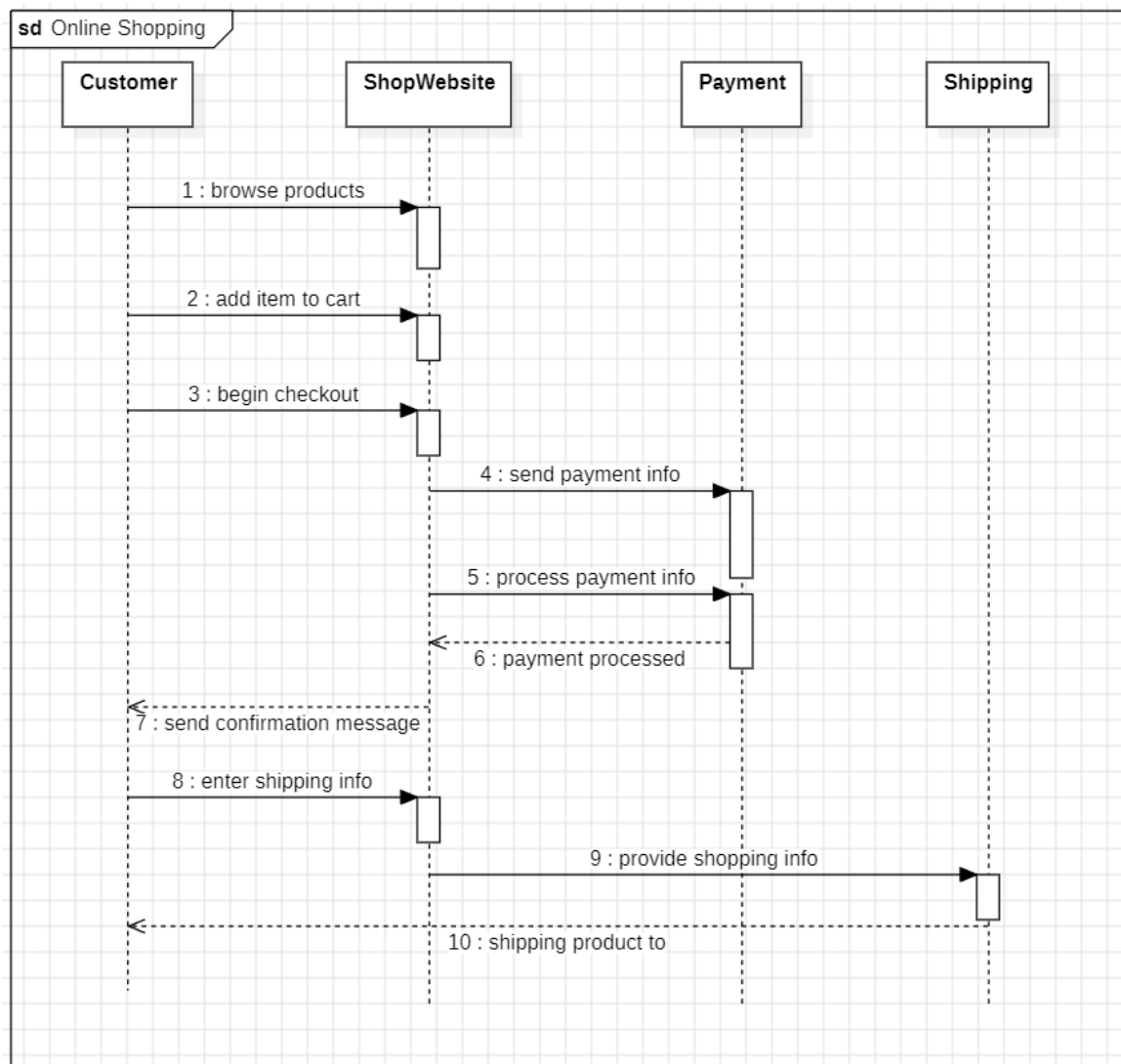
## 1. ONLINE SHOPPING

### 1.a) Use Case Diagram:

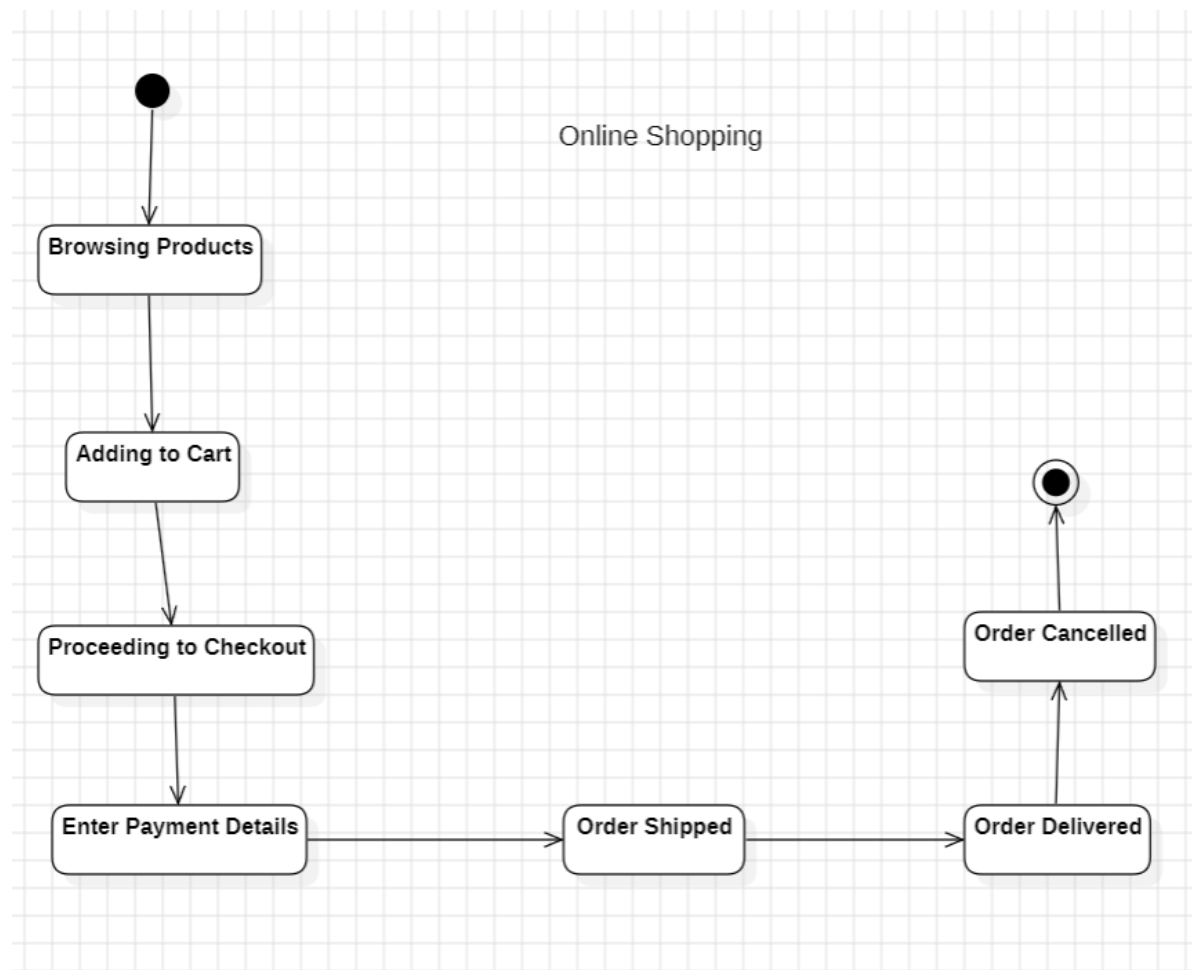


## 1.b) Class Diagram:

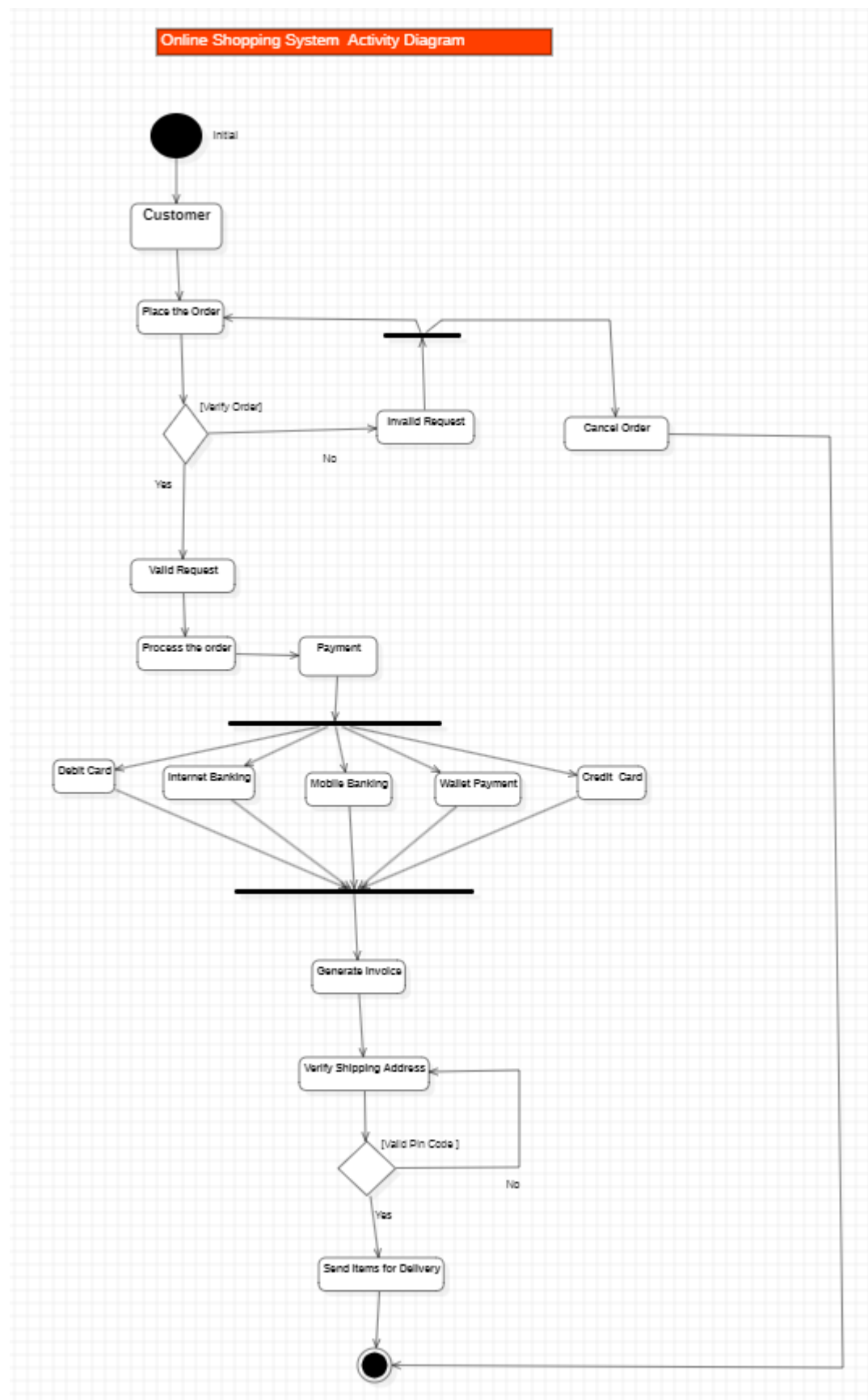


**1.c) Sequence Diagram:**



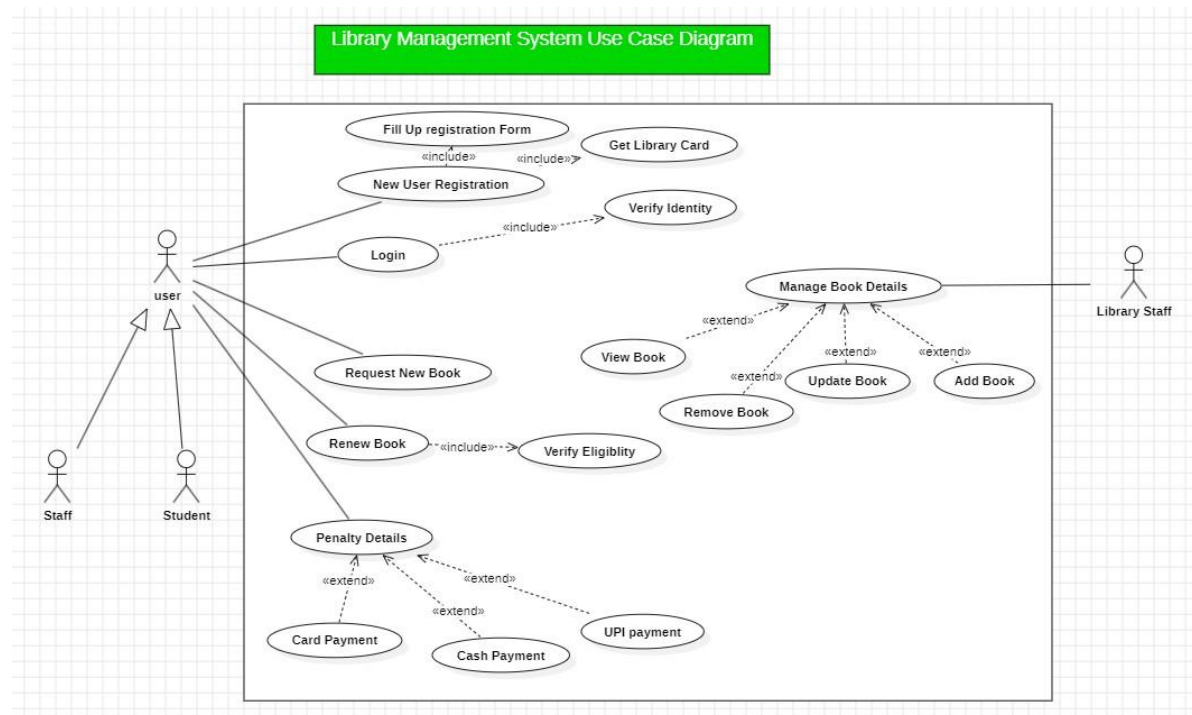
**1.d) State Diagram:**

### 1.e) Activity Diagram:

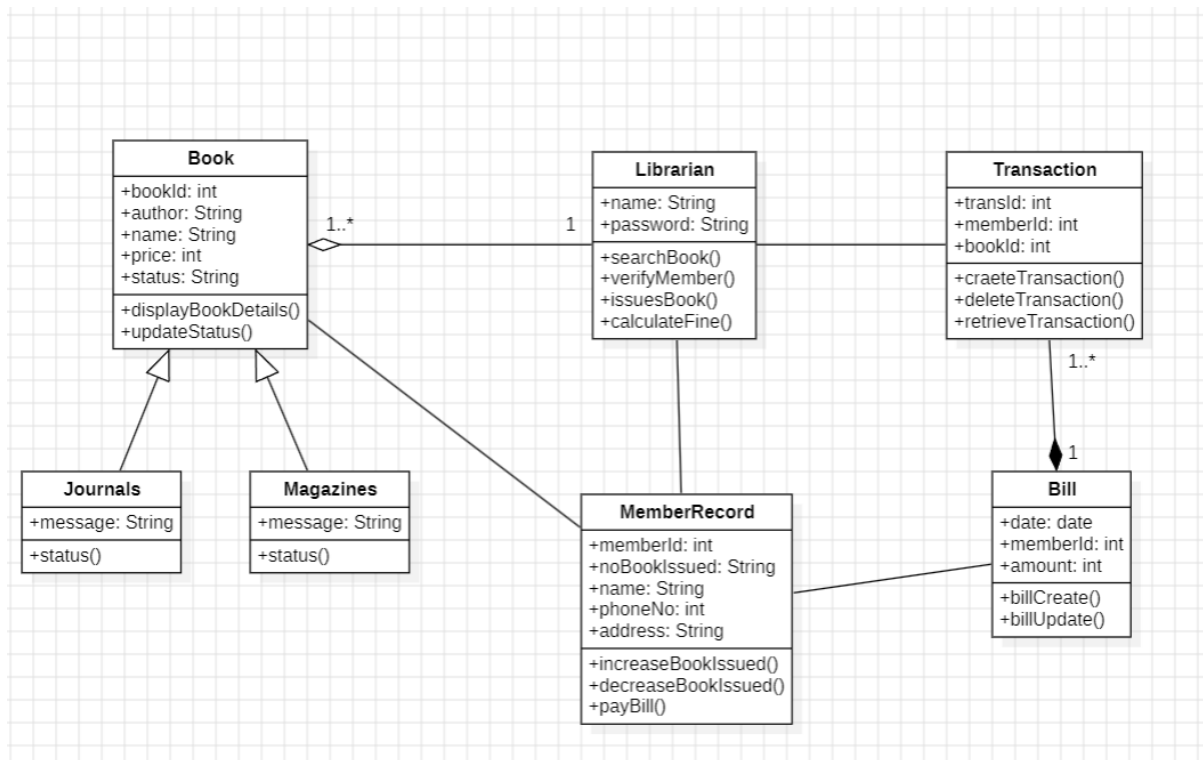


## 2. LIBRARY MANAGEMENT SYSTEM

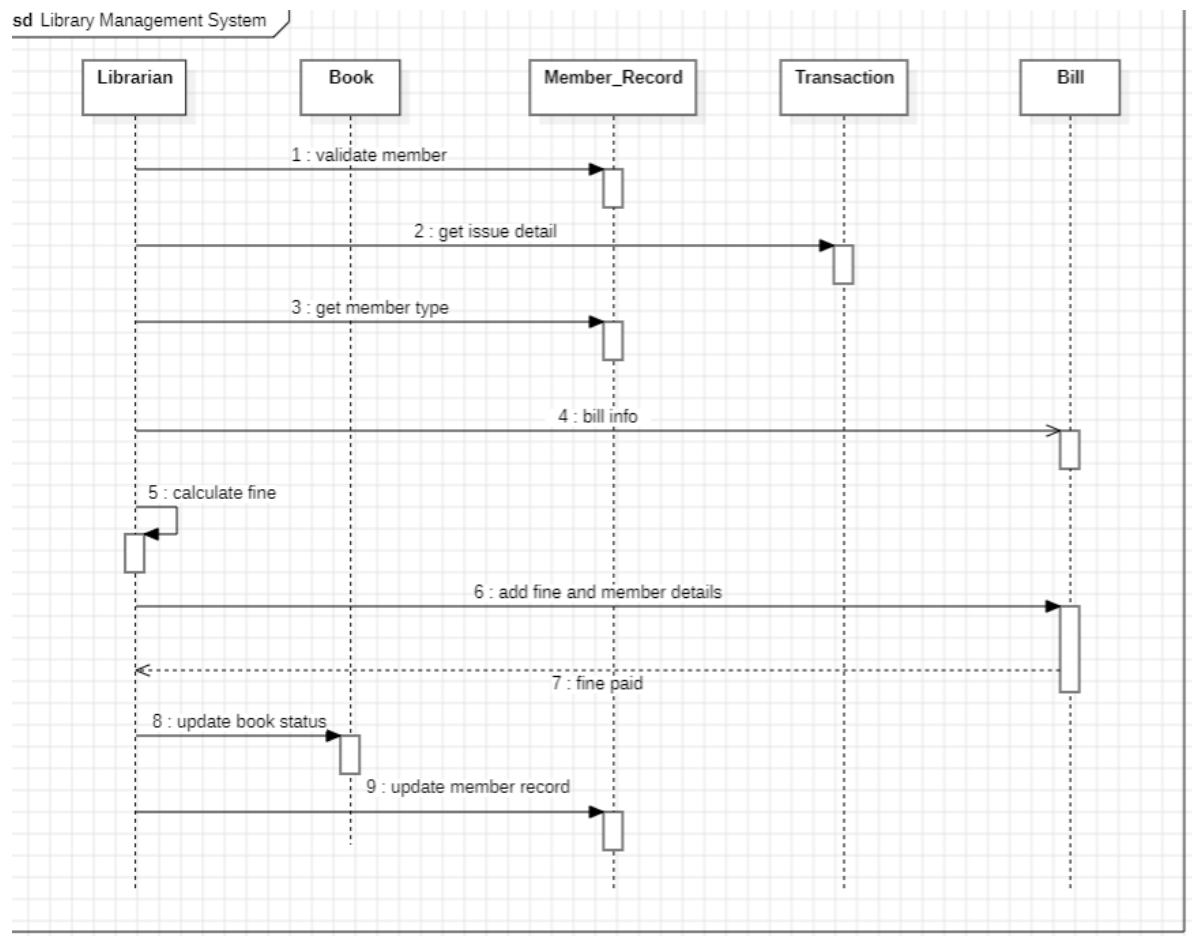
### 2.a) Use Case Diagram:



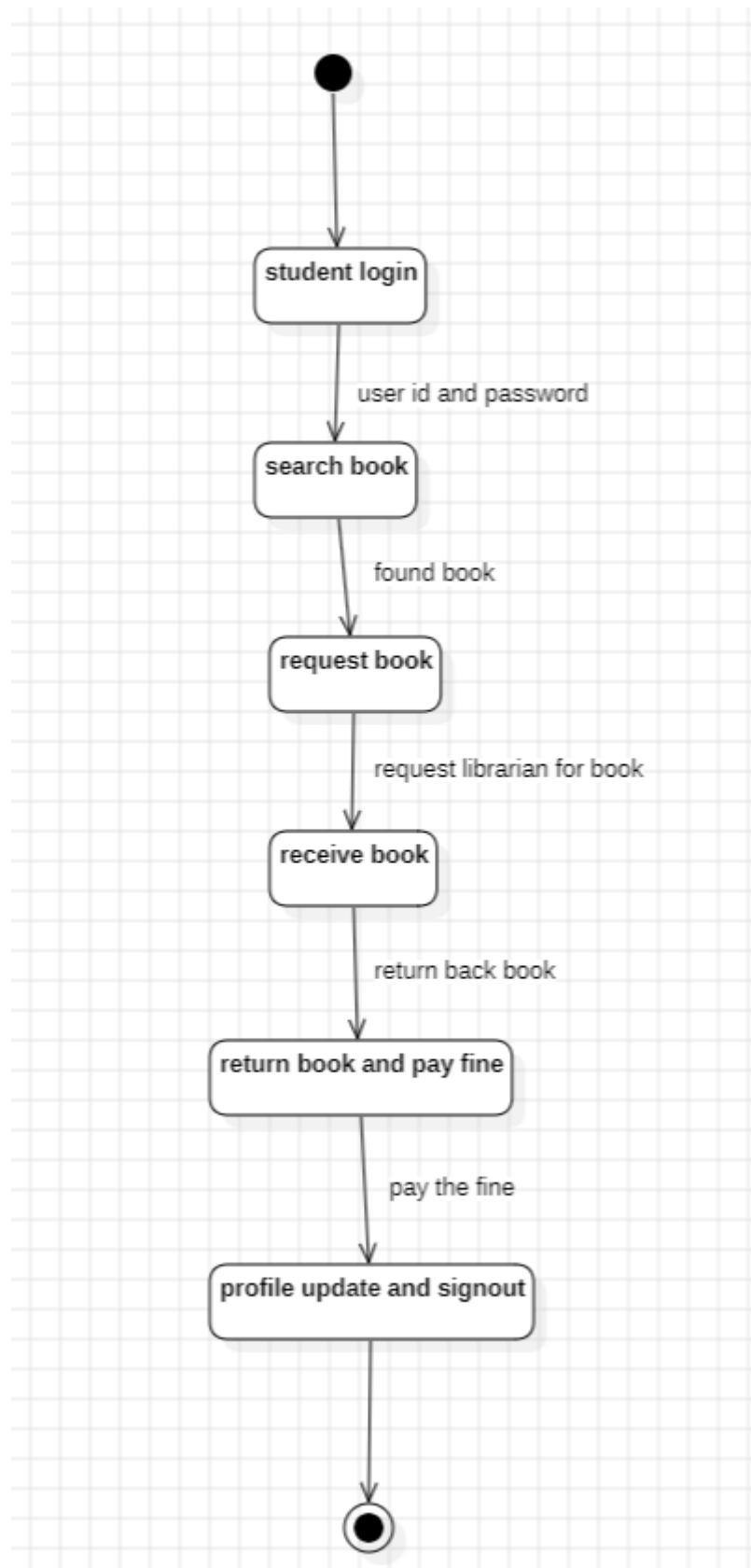
## 2.b) Class Diagram:



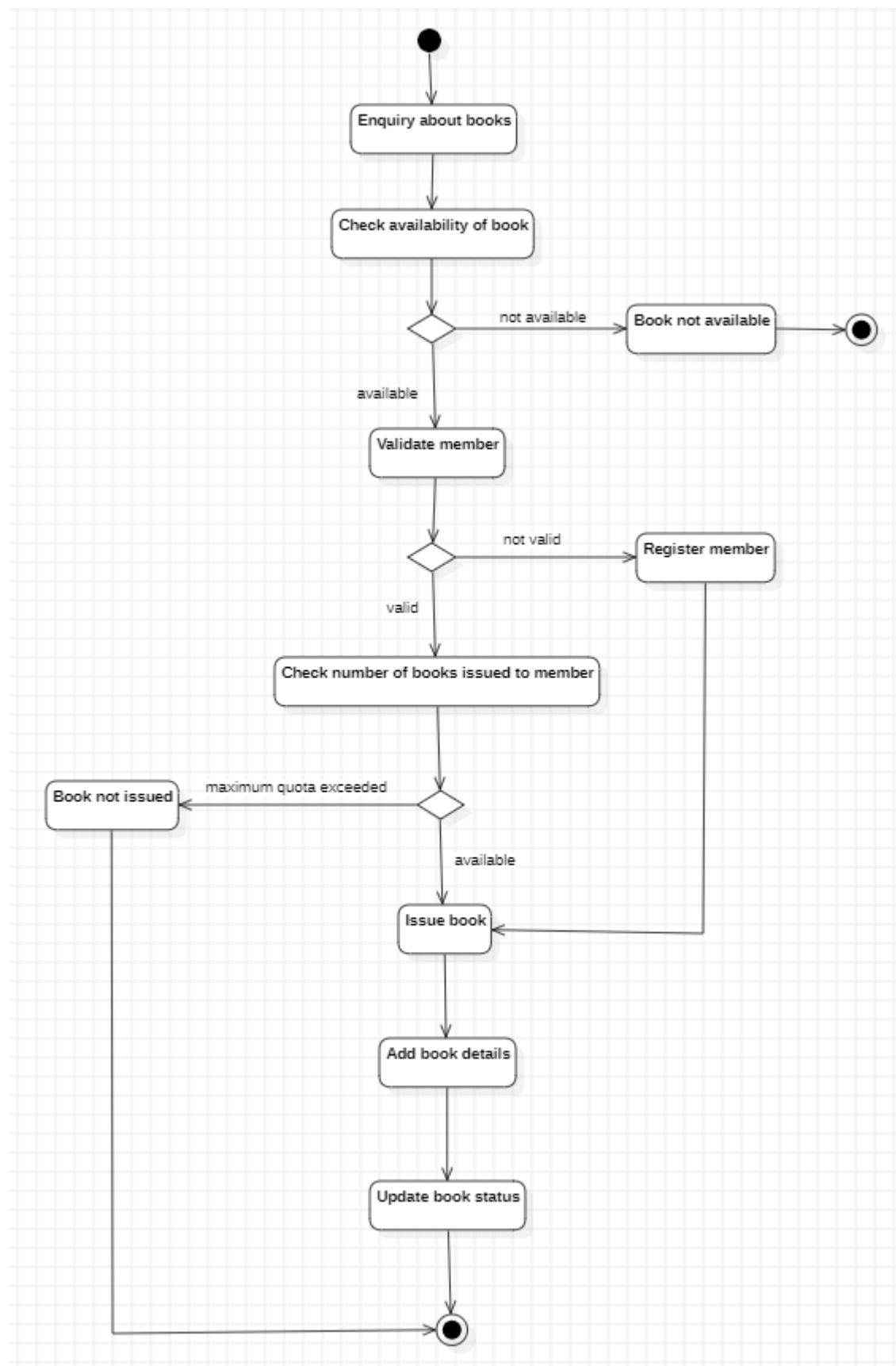
## 2.c) Sequence Diagram:



## 2.d) State Diagram:



## 2.e) Activity Diagram:



## 3. Basic Java Programs

### 3.a) Sum of Digits :

**Code:**

```
import java.util.Scanner;

public class sum_of_digits
{
    int num,sum=0;
    public sum_of_digits(int num)
    {
        this.num=num;
    }
    public void operation()
    {
        while (num != 0)
        {
            sum=sum+num%10;
            num=num/10;
        }
        System.out.println("Sum of digits : " + sum);
    }
    public static void main (String[] args)
    {
        Scanner inp= new Scanner(System.in);
        System.out.println("Enter the number : ");
        int inputNum=inp.nextInt();
        sum_of_digits obj=new sum_of_digits(inputNum);
        obj.operation();
    }
}
```



```
}  
}
```

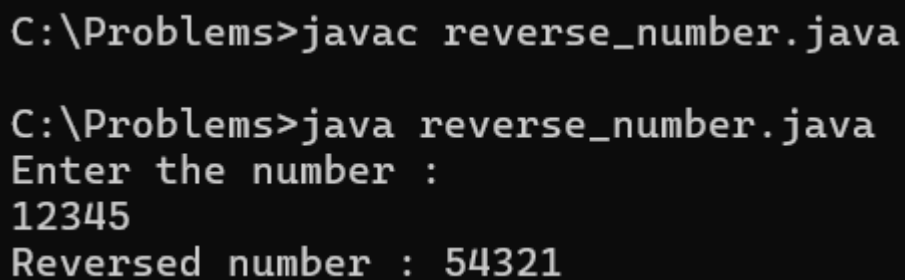
**Output:**

```
C:\Problems>javac sum_of_digits.java  
  
C:\Problems>java sum_of_digits.java  
Enter the number :  
30  
Sum of digits : 3
```

**3.b) Reverse Number :****Code:**

```
import java.util.Scanner;  
  
public class reverse_number  
{  
    int num,rev=0,digit;  
    public reverse_number(int num)  
    {  
        this.num=num;  
    }  
    public void operation()  
    {  
        while (num != 0)  
        {  
            digit=num%10;  
            rev=rev*10+digit;  
            num=num/10;  
        }  
    }  
}
```

```
System.out.println("Reversed number : " + rev);
}
public static void main (String[] args)
{
Scanner inp= new Scanner(System.in);
System.out.println("Enter the number : ");
int inputNum=inp.nextInt();
reverse_number obj=new reverse_number(inputNum);
obj.operation();
}
}
```

**Output:**

```
C:\Problems>javac reverse_number.java

C:\Problems>java reverse_number.java
Enter the number :
12345
Reversed number : 54321
```

**3.c) Prime Number :****Code:**

```
import java.util.Scanner;
public class prime_number
{
int num;
Boolean isPrime=true;
public prime_number(int num)
```

```
{
this.num=num;
}
public void operation()
{
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");
}
}
public static void main (String[] args)
```

```
{
Scanner inp= new Scanner(System.in);
System.out.println("Enter the number : ");
int inputNum=inp.nextInt();
prime_number obj=new prime_number(inputNum);
obj.operation();
}
}
```

**Output:**

```
C:\Problems>javac prime_number.java

C:\Problems>java prime_number.java
Enter the number :
23
23 is a prime number
```

**3.d) Palindrome Number :****Code:**

```
import java.util.Scanner;
public class armstrong_number
{
int num;
public armstrong_number(int num)
{
this.num = num;
}
public void operation()
```

```
{
int originalNum = num;
int sum = 0;
int digits = String.valueOf(num).length();
while (num != 0)
{
int digit = num % 10;
sum += Math.pow(digit, digits);
num /= 10;
}
if (sum == originalNum)
{
System.out.println(originalNum + " is an Armstrong number.");
}
else
{
System.out.println(originalNum + " is not an Armstrong number.");
}
}

public static void main(String[] args)
{
Scanner inp = new Scanner(System.in);
System.out.println("Enter the number: ");
int inputNum = inp.nextInt();
armstrong_number obj = new armstrong_number(inputNum);
obj.operation();
}
}
```

**Output;**

```
C:\Problems>javac palindrome_number.java

C:\Problems>java palindrome_number.java
Enter the number:
123
123 is not a palindrome number.
```

**3.e) Lower Triangle :****Code:**

```
import java.util.Scanner;

public class lower_triangle
{
    int rows;

    public lower_triangle(int rows)
    {
        this.rows = rows;
    }

    public void operation()
    {
        for (int i = 1; i <= rows; i++)
        {
            for (int j = 1; j <= i; j++)
            {
                System.out.print(j + " ");
            }
            System.out.println();
        }
    }
}
```

```
}  
public static void main(String[] args)  
{  
    Scanner inp = new Scanner(System.in);  
    System.out.println("Enter the number of rows: ");  
    int inputRows = inp.nextInt();  
    lower_triangle obj = new lower_triangle(inputRows);  
    obj.operation();  
}  
}
```

**Output:**

```
C:\Problems>javac lower_triangle.java  
  
C:\Problems>java lower_triangle.java  
Enter the number of rows:  
3  
1  
1 2  
1 2 3
```

**3.f) LCM Numbers :****Code:**

```
import java.util.Scanner;  
public class lcm_numbers  
{  
    int num1, num2;  
    public lcm_numbers(int num1, int num2)  
    {
```

```
this.num1 = num1;
this.num2 = num2;
}
public void operation()
{
    int lcm = (num1 > num2) ? num1 : num2;
    while (true)
    {
        if (lcm % num1 == 0 && lcm % num2 == 0)
        {
            System.out.println("LCM of " + num1 + " and " + num2 + " is: " +
                                lcm);
            break;
        }
        lcm++;
    }
}

public static void main(String[] args)
{
    Scanner inp = new Scanner(System.in);
    System.out.println("Enter the first number: ");
    int inputNum1 = inp.nextInt();
    System.out.println("Enter the second number: ");
    int inputNum2 = inp.nextInt();
    lcm_numbers obj = new lcm_numbers(inputNum1, inputNum2);
    obj.operation();
}
}
```



**Output:**

```
C:\Problems>javac lcm_numbers.java

C:\Problems>java lcm_numbers.java
Enter the first number:
3
Enter the second number:
4
LCM of 3 and 4 is: 12
```

**3.g) Fibonacci Series :****Code:**

```
import java.util.Scanner;

public class armstrong_number
{
    int num;

    public armstrong_number(int num)
    {
        this.num = num;
    }

    public void operation()
    {
        int originalNum = num;
        int sum = 0;
        int digits = String.valueOf(num).length();
        while (num != 0)
        {
            int digit = num % 10;
            sum += Math.pow(digit, digits);
        }
    }
}
```

```
num /= 10;
}
if (sum == originalNum)
{
    System.out.println(originalNum + " is an Armstrong number.");
}
else
{
    System.out.println(originalNum + " is not an Armstrong number.");
}
}

public static void main(String[] args)
{
    Scanner inp = new Scanner(System.in);
    System.out.println("Enter the number: ");
    int inputNum = inp.nextInt();
    armstrong_number obj = new armstrong_number(inputNum);
    obj.operation();
}
}
```

**Output:**

```
C:\Problems>javac fibonacci_series.java

C:\Problems>java fibonacci_series.java
Enter the number of terms for the Fibonacci series:
4
Fibonacci Series up to 4 terms:
0 1 1 2
```

### 3.h) Factorial Number :

**Code:**

```
import java.util.Scanner;

public class armstrong_number
{
    int num;
    public armstrong_number(int num)
    {
        this.num = num;
    }
    public void operation()
    {
        int originalNum = num;
        int sum = 0;
        int digits = String.valueOf(num).length();
        while (num != 0)
        {
            int digit = num % 10;
            sum += Math.pow(digit, digits);
            num /= 10;
        }
        if (sum == originalNum)
        {
            System.out.println(originalNum + " is an Armstrong number.");
        }
        else
        {
            System.out.println(originalNum + " is not an Armstrong number.");
        }
    }
}
```

```
}  
public static void main(String[] args)  
{  
    Scanner inp = new Scanner(System.in);  
    System.out.println("Enter the number: ");  
    int inputNum = inp.nextInt();  
    armstrong_number obj = new armstrong_number(inputNum);  
    obj.operation();  
}  
}
```

**Output:**

```
C:\Problems>javac factorial_number.java  
  
C:\Problems>java factorial_number.java  
Enter the number:  
3  
Factorial of 3 is: 6
```

**3.i) Sum of Even, Odd Digits :****Code:**

```
import java.util.Scanner;  
public class even_odd_sum  
{  
    int limit;  
    public even_odd_sum(int limit)  
    {  
        this.limit = limit;  
    }  
}
```

```
}  
public void operation()  
{  
    int evenSum = 0, oddSum = 0;  
    for (int i = 1; i <= limit; i++)  
    {  
        if (i % 2 == 0)  
        {  
            evenSum += i;  
        }  
        else  
        {  
            oddSum += i;  
        }  
    }  
    System.out.println("Sum of even numbers up to " + limit + " is: " +  
        evenSum);  
    System.out.println("Sum of odd numbers up to " + limit + " is: " +  
        oddSum);  
}  
public static void main(String[] args)  
{  
    Scanner inp = new Scanner(System.in);  
    System.out.println("Enter the limit: ");  
    int inputLimit = inp.nextInt();  
    even_odd_sum obj = new even_odd_sum(inputLimit);  
    obj.operation();  
}  
}
```

**Output:**

```
C:\Problems>javac even_odd_sum.java
C:\Problems>java even_odd_sum.java
Enter the limit:
5
Sum of even numbers up to 5 is: 6
Sum of odd numbers up to 5 is: 9
```

**3.j) Armstrong Number :****Code:**

```
import java.util.Scanner;
public class armstrong_number
{
    int num;
    public armstrong_number(int num)
    {
        this.num = num;
    }
    public void operation()
    {
        int originalNum = num;
        int sum = 0;
        int digits = String.valueOf(num).length();
        while (num != 0)
        {
            int digit = num % 10;
            sum += Math.pow(digit, digits);
        }
    }
}
```

```
num /= 10;
}
if (sum == originalNum)
{
    System.out.println(originalNum + " is an Armstrong number.");
}
else
{
    System.out.println(originalNum + " is not an Armstrong number.");
}
}

public static void main(String[] args)
{
    Scanner inp = new Scanner(System.in);
    System.out.println("Enter the number: ");
    int inputNum = inp.nextInt();
    armstrong_number obj = new armstrong_number(inputNum);
    obj.operation();
}
}
```

**Output:**

```
C:\Problems>javac armstrong_number.java

C:\Problems>java armstrong_number.java
Enter the number:
2345
2345 is not an Armstrong number.
```

# INHERITANCE

## 4. SINGLE INHERITANCE PROBLEMS:

### 4.a) Class Manager extends Employee

#### Code:

```
class Employee
{
    String name;
    double salary;
    Employee(String name, double salary)
    {
        this.name = name;
        this.salary = salary;
    }
    void display1()
    {
        System.out.println("Name: " + name + ", Salary: " + salary);
    }
}
class Manager extends Employee
{
    String department;
    Manager(String name, double salary, String department)
    {
        super(name, salary);
        this.department = department;
    }
    void display2()
    {
        System.out.println("Department: " + department);
    }
}
public class Company
{
    public static void main(String[] args)
    {
        Manager mgr = new Manager("John", 50000, "HR");
        mgr.display1();
        mgr.display2();
    }
}
```



```
}  
}
```

**Output:**

```
C:\Problems>javac Company.java  
  
C:\Problems>java Company.java  
Name: John, Salary: 50000.0  
Department: HR
```

**4.b) Class Batsman extends Cricket****Code:**

```
class CricketPlayer  
{  
    String name;  
    int runs;  
    CricketPlayer(String name, int runs)  
    {  
        this.name = name;  
        this.runs = runs;  
    }  
    void displayPlayer()  
    {  
        System.out.println("Player: " + name + ", Runs: " + runs);  
    }  
}  
class Batsman extends CricketPlayer  
{  
    double strikeRate;  
    Batsman(String name, int runs, double strikeRate)  
    {  
        super(name, runs);  
        this.strikeRate = strikeRate;  
    }  
    void displayBatsman()  
    {  
        System.out.println("Strike Rate: " + strikeRate);  
    }  
}  
public class Cricket  
{
```

```
public static void main(String[] args)
{
    Batsman batsman = new Batsman("Virat Kohli", 12000, 92.5);
    batsman.displayPlayer();
    batsman.displayBatsman();
}
}
```

**Output:**

```
C:\Problems>javac Cricket.java

C:\Problems>java Cricket.java
Player: Virat Kohli, Runs: 12000
Strike Rate: 92.5
```

## 5. MULTILEVEL INHERITANCE PROBLEMS:

### 5.a) Class CEO extends Manager

**Code:**

```
class Employee
{
    String name;
    double salary;
    Employee(String name, double salary)
    {
        this.name = name;
        this.salary = salary;
    }
    void displayEmployee()
    {
        System.out.println("Name: " + name + ", Salary: " + salary);
    }
}
class Manager extends Employee
{
    String department;
    Manager(String name, double salary, String department)
```

```
{
super(name, salary);
this.department = department;
}
void displayManager()
{
System.out.println("Department: " + department);
}
}
class CEO extends Manager
{
String company;
CEO(String name, double salary, String department, String company)
{
super(name, salary, department);
this.company = company;
}
void displayCEO()
{
System.out.println("Company: " + company);
}
}
public class Company1
{
public static void main(String[] args)
{
CEO ceo = new CEO("John", 500000, "Executive", "Tech Corp");
ceo.displayEmployee();
ceo.displayManager();
ceo.displayCEO();
}
}
```

**Output:**

```
C:\Problems>javac Company1.java

C:\Problems>java Company1.java
Name: John, Salary: 500000.0
Department: Executive
Company: Tech Corp
```

## 5.b) Class ALLRounder extends Batsman

### Code:

```
class CricketPlayer
{
    String name;
    int runs;
    CricketPlayer(String name, int runs)
    {
        this.name = name;
        this.runs = runs;
    }
    void displayPlayer()
    {
        System.out.println("Player: " + name + ", Runs: " + runs);
    }
}
class Batsman extends CricketPlayer
{
    double strikeRate;
    Batsman(String name, int runs, double strikeRate)
    {
        super(name, runs);
        this.strikeRate = strikeRate;
    }
    void displayBatsman()
    {
        System.out.println("Strike Rate: " + strikeRate);
    }
}
class AllRounder extends Batsman
{
    int wickets;
    AllRounder(String name, int runs, double strikeRate, int wickets)
    {
        super(name, runs, strikeRate);
        this.wickets = wickets;
    }
    void displayAllRounder()
    {
        System.out.println("Wickets Taken: " + wickets);
    }
}
public class Cricket1
{
    public static void main(String[] args)
    {
        AllRounder ar = new AllRounder("Shakib Al Hasan", 6000, 85.2, 300);
```

```
ar.displayPlayer();
ar.displayBatsman();
ar.displayAllRounder();
}
}
```

**Output:**

```
C:\Problems>javac Cricket1.java

C:\Problems>java Cricket1.java
Player: Shakib Al Hasan, Runs: 6000
Strike Rate: 85.2
Wickets Taken: 300
```

## 6. HEIRARCHIAL INHERITANCE PROBLEMS:

### 6.a) Class Manager and Developer extends Employee

**Code:**

```
class Employee
{
    String name;
    double salary;
    Employee(String name, double salary)
    {
        this.name = name;
        this.salary = salary;
    }
    void displayEmployee()
    {
        System.out.println("Name: " + name + ", Salary: " + salary);
    }
}
class Manager extends Employee
{
    String department;
    Manager(String name, double salary, String department)
    {
        super(name, salary);
```

```
this.department = department;
}
void displayManager()
{
System.out.println("Department: " + department);
}
}
class Developer extends Employee
{
String programmingLanguage;
Developer(String name, double salary, String programmingLanguage)
{
super(name, salary);
this.programmingLanguage = programmingLanguage;
}
void displayDeveloper()
{
System.out.println("Programming Language: " + programmingLanguage);
}
}
public class Company2
{
public static void main(String[] args)
{
Manager mgr = new Manager("Sarah", 80000, "IT");
Developer dev = new Developer("Alex", 70000, "Java");
System.out.println("Manager Details:");
mgr.displayEmployee();
mgr.displayManager();
System.out.println("\nDeveloper Details:");
dev.displayEmployee();
dev.displayDeveloper();
}
}
```

**Output:**

```
C:\Problems>javac Company2.java

C:\Problems>java Company2.java
Manager Details:
Name: Sarah, Salary: 80000.0
Department: IT

Developer Details:
Name: Alex, Salary: 70000.0
Programming Language: Java
```

## 6.b) Class Batsman and Bowler extends Cricket

### Code:

```
class CricketPlayer
{
    String name;
    int runs;
    CricketPlayer(String name, int runs)
    {
        this.name = name;
        this.runs = runs;
    }
    void displayPlayer()
    {
        System.out.println("Player: " + name + ", Runs: " + runs);
    }
}
class Batsman extends CricketPlayer
{
    double strikeRate;
    Batsman(String name, int runs, double strikeRate)
    {
        super(name, runs);
        this.strikeRate = strikeRate;
    }
    void displayBatsman()
    {
        System.out.println("Strike Rate: " + strikeRate);
    }
}
class Bowler extends CricketPlayer
{
    int wickets;
    double economy;
    Bowler(String name, int runs, int wickets, double economy)
    {
        super(name, runs);
        this.wickets = wickets;
        this.economy = economy;
    }
    void displayBowler()
    {
        System.out.println("Wickets: " + wickets + ", Economy: " + economy);
    }
}
class AllRounder extends Batsman
{
    int wickets;
```

```
AllRounder(String name, int runs, double strikeRate, int wickets)
{
    super(name, runs, strikeRate);
    this.wickets = wickets;
}
void displayAllRounder()
{
    System.out.println("Wickets Taken: " + wickets);
}
}
public class Cricket2
{
    public static void main(String[] args)
    {
        System.out.println("Batsman Details:");
        Batsman bat = new Batsman("Virat Kohli", 12000, 92.5);
        bat.displayPlayer();
        bat.displayBatsman();
        System.out.println("\nBowler Details:");
        Bowler bow = new Bowler("Jasprit Bumrah", 500, 300, 4.5);
        bow.displayPlayer();
        bow.displayBowler();
        System.out.println("\nAll-Rounder Details:");
        AllRounder ar = new AllRounder("Shakib Al Hasan", 6000, 85.2, 300);
        ar.displayPlayer();
        ar.displayBatsman();
        ar.displayAllRounder();
    }
}
```

**Output:**

```
C:\Problems>javac Cricket2.java

C:\Problems>java Cricket2.java
Batsman Details:
Player: Virat Kohli, Runs: 12000
Strike Rate: 92.5

Bowler Details:
Player: Jasprit Bumrah, Runs: 500
Wickets: 300, Economy: 4.5

All-Rounder Details:
Player: Shakib Al Hasan, Runs: 6000
Strike Rate: 85.2
Wickets Taken: 300
```



## 7. HYBRID INHERITANCE PROBLEMS:

### 7.a) Class TechLead extends Developer

#### Code:

```
class Employee
{
    String name;
    double salary;
    Employee(String name, double salary)
    {
        this.name = name;
        this.salary = salary;
    }
    void displayEmployee()
    {
        System.out.println("Name: " + name + ", Salary: " + salary);
    }
}
class Manager extends Employee
{
    String department;
    Manager(String name, double salary, String department)
    {
        super(name, salary);
        this.department = department;
    }
    void displayManager()
    {
        System.out.println("Department: " + department);
    }
}
class Developer extends Employee
{
    String programmingLanguage;
    Developer(String name, double salary, String programmingLanguage)
    {
        super(name, salary);
        this.programmingLanguage = programmingLanguage;
    }
    void displayDeveloper()
    {
        System.out.println("Programming Language: " + programmingLanguage);
    }
}
```

```
class TechLead extends Developer
{
String team;
TechLead(String name, double salary, String programmingLanguage, String team)
{
super(name, salary, programmingLanguage);
this.team = team;
}
void displayTechLead()
{
System.out.println("Team: " + team);
}
}
public class Company3
{
public static void main(String[] args)
{
Manager mgr = new Manager("Sarah", 80000, "IT");
Developer dev = new Developer("Alex", 70000, "Java");
TechLead tl = new TechLead("Mark", 90000, "Python", "AI Team");
System.out.println("Manager Details:");
mgr.displayEmployee();
mgr.displayManager();
System.out.println("\nDeveloper Details:");
dev.displayEmployee();
dev.displayDeveloper();
System.out.println("\nTechLead Details:");
tl.displayEmployee();
tl.displayDeveloper();
tl.displayTechLead();
}
}
```

**Output:**

```
C:\Problems>javac Company3.java

C:\Problems>java Company3.java
Manager Details:
Name: Sarah, Salary: 80000.0
Department: IT

Developer Details:
Name: Alex, Salary: 70000.0
Programming Language: Java

TechLead Details:
Name: Mark, Salary: 90000.0
Programming Language: Python
Team: AI Team
```

## 7.b) Class WicketKeeper extends Batsman

### Code:

```
class CricketPlayer
{
    String name;
    int runs;
    CricketPlayer(String name, int runs)
    {
        this.name = name;
        this.runs = runs;
    }
    void displayPlayer()
    {
        System.out.println("Player: " + name + ", Runs: " + runs);
    }
}
class Batsman extends CricketPlayer
{
    double strikeRate;
    Batsman(String name, int runs, double strikeRate)
    {
        super(name, runs);
        this.strikeRate = strikeRate;
    }
    void displayBatsman()
    {
        System.out.println("Strike Rate: " + strikeRate);
    }
}
class Bowler extends CricketPlayer
{
    int wickets;
    double economy;
    Bowler(String name, int runs, int wickets, double economy)
    {
        super(name, runs);
        this.wickets = wickets;
        this.economy = economy;
    }
    void displayBowler()
    {
        System.out.println("Wickets: " + wickets + ", Economy: " + economy);
    }
}
class AllRounder extends Batsman
{

```

```
int wickets;
double bowlingAvg;
AllRounder(String name, int runs, double strikeRate, int wickets, double bowlingAvg)
{
    super(name, runs, strikeRate);
    this.wickets = wickets;
    this.bowlingAvg = bowlingAvg;
}
void displayAllRounder()
{
    System.out.println("Wickets: " + wickets + ", Bowling Avg: " + bowlingAvg);
}
}
class WicketKeeper extends Batsman
{
    int catches;
    int stumpings;
    WicketKeeper(String name, int runs, double strikeRate, int catches, int stumpings)
    {
        super(name, runs, strikeRate);
        this.catches = catches;
        this.stumpings = stumpings;
    }
    void displayWicketKeeper()
    {
        System.out.println("Catches: " + catches + ", Stumpings: " + stumpings);
    }
}
public class Cricket3
{
    public static void main(String[] args)
    {
        System.out.println("Batsman Details:");
        Batsman bat = new Batsman("Virat Kohli", 12000, 92.5);
        bat.displayPlayer();
        bat.displayBatsman();
        System.out.println("\nBowler Details:");
        Bowler bow = new Bowler("Jasprit Bumrah", 500, 300, 4.5);
        bow.displayPlayer();
        bow.displayBowler();
        System.out.println("\nAll-Rounder Details:");
        AllRounder ar = new AllRounder("Shakib Al Hasan", 6000, 85.2, 300, 28.5);
        ar.displayPlayer();
        ar.displayBatsman();
        ar.displayAllRounder();
        System.out.println("\nWicket-Keeper Details:");
        WicketKeeper wk = new WicketKeeper("MS Dhoni", 8000, 87.6, 250, 100);
        wk.displayPlayer();
        wk.displayBatsman();
        wk.displayWicketKeeper();
    }
}
```

```
}  
}
```

**Output:**

```
C:\Problems>javac Cricket3.java  
  
C:\Problems>java Cricket3.java  
Batsman Details:  
Player: Virat Kohli, Runs: 12000  
Strike Rate: 92.5  
  
Bowler Details:  
Player: Jasprit Bumrah, Runs: 500  
Wickets: 300, Economy: 4.5  
  
All-Rounder Details:  
Player: Shakib Al Hasan, Runs: 6000  
Strike Rate: 85.2  
Wickets: 300, Bowling Avg: 28.5  
  
Wicket-Keeper Details:  
Player: MS Dhoni, Runs: 8000  
Strike Rate: 87.6  
Catches: 250, Stumpings: 100
```

# POLYMORPHISM

## 8. CONSTRUCTOR OVERLOADING PROGRAMS:

### 8.a) Constructor constructHouse

#### Code:

```
class ConstructionTeam
{
    public void constructHouse()
    {
        System.out.println("Constructing standard house:");
        System.out.println("- Concrete foundation");
        System.out.println("- Wooden structure");
        System.out.println("- Shingle roof");
        System.out.println("- 3 rooms");
    }
    public void constructHouse(int rooms)
    {
        System.out.println("Constructing house with "+rooms+" rooms");
        System.out.println("- Concrete foundation");
        System.out.println("- Wooden structure");
        System.out.println("- Shingle roof");
    }
    public void constructHouse(String f,String s,String r)
    {
        System.out.println("Constructing custom house:");
        System.out.println("- "+f+" foundation");
        System.out.println("- "+s+" structure");
        System.out.println("- "+r+" roof");
        System.out.println("- 3 rooms (default)");
    }
    public void constructHouse(String f,String s,String r,int rm)
    {
        System.out.println("Constructing full custom house:");
        System.out.println("- "+f+" foundation");
        System.out.println("- "+s+" structure");
        System.out.println("- "+r+" roof");
        System.out.println("- "+rm+" rooms");
    }
    public void constructHouse(double budget)
    {
        if(budget<100000)
```

```
{
constructHouse(2);
}
else if(budget<250000)
{
constructHouse(3);
}
else
{
constructHouse("Reinforced","Steel","Tile",4);
}
}
}
class HouseBuilder
{
public static void main(String[] args)
{
ConstructionTeam t=new ConstructionTeam();
System.out.println("=== CONSTRUCTION DEMO ===");
System.out.println("\n1. Default house:");
t.constructHouse();
System.out.println("\n2. 5-room house:");
t.constructHouse(5);
System.out.println("\n3. Stone house:");
t.constructHouse("Stone","Wood","Tile");
}
}
```

**Output:**

```
C:\Problems>javac HouseBuilder.java

C:\Problems>java HouseBuilder.java
=== CONSTRUCTION DEMO ===

1. Default house:
Constructing standard house:
- Concrete foundation
- Wooden structure
- Shingle roof
- 3 rooms

2. 5-room house:
Constructing house with 5 rooms
- Concrete foundation
- Wooden structure
- Shingle roof

3. Stone house:
Constructing custom house:
- Stone foundation
- Wood structure
- Tile roof
- 3 rooms (default)
```

## 9. CONSTRUCTOR OVERRIDING PROGRAMS:

### 9.b) Constructor ManufacturingMachine

#### Code:

```
class ManufacturingMachine
{
    String machineType;
    int productionRate;
    ManufacturingMachine()
    {
        this.machineType = "General Purpose";
        this.productionRate = 100;
        System.out.println("Basic manufacturing machine created");
    }
    ManufacturingMachine(String type, int rate)
    {
        this.machineType = type;
        this.productionRate = rate;
        System.out.println("Custom " + type + " machine created with
rate: " + rate);
    }
    void operate()
    {
        System.out.println(machineType + " machine operating at " +
productionRate + " units/hour");
    }
}
class InjectionMolder extends ManufacturingMachine
{
    String moldType;
    int temperature;
    InjectionMolder()
    {
        super();
        this.moldType = "Standard";
        this.temperature = 200;
    }
    InjectionMolder(String type, int rate, String mold, int temp)
    {
        super(type, rate);
        this.moldType = mold;
        this.temperature = temp;
    }
}
```



```
}
@Override
void operate()
{
    System.out.println("Injection molding with " + moldType + " mold
    at " + temperature + "°C");
    super.operate();
}
}
class AssemblyRobot extends ManufacturingMachine
{
    int precisionLevel;
    AssemblyRobot()
    {
        super("Assembly Robot", 50);
        this.precisionLevel = 5;
    }
    AssemblyRobot(int precision)
    {
        super("High-Precision Robot", 30);
        this.precisionLevel = precision;
    }
    @Override
    void operate()
    {
        System.out.println("Robot assembling with precision level " +
        precisionLevel);
        super.operate();
    }
}
public class ManufacturingDemo
{
    public static void main(String[] args)
    {
        System.out.println("=== MANUFACTURING SYSTEM ===");
        ManufacturingMachine machine1 = new ManufacturingMachine();
        machine1.operate();
        ManufacturingMachine machine2 = new ManufacturingMachine("CNC",
        75);
        machine2.operate();
        System.out.println("\nSpecialized Machines:");
        InjectionMolder molder1 = new InjectionMolder();
        InjectionMolder molder2 = new InjectionMolder("Plastic Molder",
        120, "Complex", 250);
        molder1.operate();
        molder2.operate();
        AssemblyRobot robot1 = new AssemblyRobot();
        AssemblyRobot robot2 = new AssemblyRobot(8);
        robot1.operate();
        robot2.operate();
    }
}
```

```
}  
}
```

**Output:**

```
C:\Problems>javac ManufacturingDemo.java  
  
C:\Problems>java ManufacturingDemo.java  
=== MANUFACTURING SYSTEM ===  
Basic manufacturing machine created  
General Purpose machine operating at 100 units/hour  
Custom CNC machine created with rate: 75  
CNC machine operating at 75 units/hour  
  
Specialized Machines:  
Basic manufacturing machine created  
Custom Plastic Molder machine created with rate: 120  
Injection molding with Standard mold at 200°C  
General Purpose machine operating at 100 units/hour  
Injection molding with Complex mold at 250°C  
Plastic Molder machine operating at 120 units/hour  
Custom Assembly Robot machine created with rate: 50  
Custom High-Precision Robot machine created with rate: 30  
Robot assembling with precision level 5  
Assembly Robot machine operating at 50 units/hour  
Robot assembling with precision level 8  
High-Precision Robot machine operating at 30 units/hour
```

## 10. METHOD OVERLOADING PROGRAMS:

### 10.a) Method constructHouse

#### Code:

```
class ConstructionTeam
{
    public void constructHouse()
    {
        System.out.println("Constructing standard house:");
        System.out.println("- Concrete foundation");
        System.out.println("- Wooden structure");
        System.out.println("- Shingle roof");
        System.out.println("- 3 rooms");
    }
    public void constructHouse(int rooms)
    {
        System.out.println("Constructing house with "+rooms+" rooms");
        System.out.println("- Concrete foundation");
        System.out.println("- Wooden structure");
        System.out.println("- Shingle roof");
    }
    public void constructHouse(String f,String s,String r)
    {
        System.out.println("Constructing custom house:");
        System.out.println("- "+f+" foundation");
        System.out.println("- "+s+" structure");
        System.out.println("- "+r+" roof");
        System.out.println("- 3 rooms (default)");
    }
    public void constructHouse(String f,String s,String r,int rm)
    {
        System.out.println("Constructing full custom house:");
        System.out.println("- "+f+" foundation");
        System.out.println("- "+s+" structure");
        System.out.println("- "+r+" roof");
        System.out.println("- "+rm+" rooms");
    }
    public void constructHouse(double budget)
    {
        if(budget<100000)
        {
            constructHouse(2);
        }
    }
}
```

```
else if(budget<250000)
{
constructHouse(3);
}
else
{
constructHouse("Reinforced","Steel","Tile",4);
}
}
}
class HouseBuilder1
{
public static void main(String[] args)
{
ConstructionTeam t=new ConstructionTeam();
System.out.println("=== CONSTRUCTION DEMO ===");
System.out.println("\n1. Default house:");
t.constructHouse();
System.out.println("\n2. 5-room house:");
t.constructHouse(5);
System.out.println("\n3. Stone house:");
t.constructHouse("Stone","Wood","Tile");
}
}
```

**Output:**

```
C:\Problems>javac HouseBuilder1.java

C:\Problems>java HouseBuilder1.java
=== CONSTRUCTION DEMO ===

1. Default house:
Constructing standard house:
- Concrete foundation
- Wooden structure
- Shingle roof
- 3 rooms

2. 5-room house:
Constructing house with 5 rooms
- Concrete foundation
- Wooden structure
- Shingle roof

3. Stone house:
Constructing custom house:
- Stone foundation
- Wood structure
- Tile roof
- 3 rooms (default)
```

## 10.b) Method ManufacturingMachine

### Code:

```
class ManufacturingMachine
{
    String machineType;
    int productionRate;
    ManufacturingMachine()
    {
        this.machineType = "General Purpose";
        this.productionRate = 100;
        System.out.println("Basic manufacturing machine created");
    }
    ManufacturingMachine(String type, int rate)
    {
        this.machineType = type;
        this.productionRate = rate;
        System.out.println("Custom " + type + " machine created with
rate: " + rate);
    }
    void operate()
    {
        System.out.println(machineType + " machine operating at " +
productionRate + " units/hour");
    }
    void operate(int duration)
    {
        System.out.println(machineType + " machine operating at " +
productionRate + " units/hour for " + duration + " hours");
    }
    void operate(String shift)
    {
        System.out.println(machineType + " machine operating at " +
productionRate + " units/hour during " + shift + " shift");
    }
}
class InjectionMolder extends ManufacturingMachine
{
    String moldType;
    int temperature;
    InjectionMolder()
    {
        super();
        this.moldType = "Standard";
        this.temperature = 200;
    }
    InjectionMolder(String type, int rate, String mold, int temp)
    {
```

```
super(type, rate);
this.moldType = mold;
this.temperature = temp;
}
void operateWithMold()
{
System.out.println("Operating with " + moldType + " mold at " +
temperature + "°C");
}
void operateWithMold(int cycles)
{
System.out.println("Operating " + cycles + " cycles with " +
moldType + " mold at " + temperature + "°C");
}
}
class AssemblyRobot extends ManufacturingMachine
{
int precisionLevel;
AssemblyRobot()
{
super("Assembly Robot", 50);
this.precisionLevel = 5;
}
AssemblyRobot(int precision)
{
super("High-Precision Robot", 30);
this.precisionLevel = precision;
}
void assemble()
{
System.out.println("Assembling with precision level " +
precisionLevel);
}
void assemble(String component)
{
System.out.println("Assembling " + component + " with precision
level " + precisionLevel);
}
}
public class ManufacturingDemo1
{
public static void main(String[] args)
{
System.out.println("=== MANUFACTURING SYSTEM ===");
ManufacturingMachine machine1 = new ManufacturingMachine();
machine1.operate();
machine1.operate(8);
machine1.operate("night");
ManufacturingMachine machine2 = new ManufacturingMachine("CNC",
75);
```

```
machine2.operate();
machine2.operate(12);
System.out.println("\nSpecialized Machines:");
InjectionMolder molder1 = new InjectionMolder();
molder1.operate();
molder1.operateWithMold();
molder1.operateWithMold(50);
AssemblyRobot robot1 = new AssemblyRobot();
robot1.operate();
robot1.assemble();
robot1.assemble("engine block");
}
}
```

**Output:**

```
C:\Problems>javac ManufacturingDemo1.java

C:\Problems>java ManufacturingDemo1.java
=== MANUFACTURING SYSTEM ===
Basic manufacturing machine created
General Purpose machine operating at 100 units/hour
General Purpose machine operating at 100 units/hour for 8 hours
General Purpose machine operating at 100 units/hour during night shift
Custom CNC machine created with rate: 75
CNC machine operating at 75 units/hour
CNC machine operating at 75 units/hour for 12 hours

Specialized Machines:
Basic manufacturing machine created
General Purpose machine operating at 100 units/hour
Operating with Standard mold at 200°C
Operating 50 cycles with Standard mold at 200°C
Custom Assembly Robot machine created with rate: 50
Assembly Robot machine operating at 50 units/hour
Assembling with precision level 5
Assembling engine block with precision level 5
```

# 11. METHOD OVERRIDING PROGRAMS:

## 11.a) Method construct

### Code:

```
class ConstructionTeam
{
    public void construct()
    {
        System.out.println("Constructing standard house:");
        System.out.println("- Concrete foundation");
        System.out.println("- Wooden structure");
        System.out.println("- Shingle roof");
        System.out.println("- 3 rooms");
    }
}
class LuxuryConstructionTeam extends ConstructionTeam
{
    @Override
    public void construct()
    {
        System.out.println("Constructing luxury house:");
        System.out.println("- Reinforced foundation");
        System.out.println("- Steel structure");
        System.out.println("- Tile roof");
        System.out.println("- 5 rooms with amenities");
    }
}
class EcoConstructionTeam extends ConstructionTeam
{
    @Override
    public void construct()
    {
        System.out.println("Constructing eco-friendly house:");
        System.out.println("- Green foundation");
        System.out.println("- Bamboo structure");
        System.out.println("- Solar roof");
        System.out.println("- 4 rooms with energy saving");
    }
}
class HouseBuilder2
{
    public static void main(String[] args)
    {
        ConstructionTeam t1=new ConstructionTeam();
    }
}
```



```
LuxuryConstructionTeam t2=new LuxuryConstructionTeam();
EcoConstructionTeam t3=new EcoConstructionTeam();
System.out.println("=== CONSTRUCTION DEMO ===");
System.out.println("\n1. Standard house:");
t1.construct();
System.out.println("\n2. Luxury house:");
t2.construct();
System.out.println("\n3. Eco-friendly house:");
t3.construct();
}
}
```

**Output:**

```
C:\Problems>javac HouseBuilder2.java
```

```
C:\Problems>java HouseBuilder2.java
```

```
=== CONSTRUCTION DEMO ===
```

```
1. Standard house:
```

```
Constructing standard house:
```

- Concrete foundation
- Wooden structure
- Shingle roof
- 3 rooms

```
2. Luxury house:
```

```
Constructing luxury house:
```

- Reinforced foundation
- Steel structure
- Tile roof
- 5 rooms with amenities

```
3. Eco-friendly house:
```

```
Constructing eco-friendly house:
```

- Green foundation
- Bamboo structure
- Solar roof
- 4 rooms with energy saving

## 11.b) Method operate

### Code:

```
class ManufacturingMachine
{
    String machineType;
    int productionRate;
    ManufacturingMachine()
    {
        this.machineType = "General Purpose";
        this.productionRate = 100;
        System.out.println("Basic manufacturing machine created");
    }
    ManufacturingMachine(String type, int rate)
    {
        this.machineType = type;
        this.productionRate = rate;
        System.out.println("Custom " + type + " machine created with
rate: " + rate);
    }
    void operate()
    {
        System.out.println(machineType + " machine operating at " +
productionRate + " units/hour");
    }
}
class InjectionMolder extends ManufacturingMachine
{
    String moldType;
    int temperature;
    InjectionMolder()
    {
        super();
        this.moldType = "Standard";
        this.temperature = 200;
    }
    InjectionMolder(String type, int rate, String mold, int temp)
    {
        super(type, rate);
        this.moldType = mold;
        this.temperature = temp;
    }
    @Override
    void operate()
    {
        System.out.println("Injection molding machine operating with " +
moldType + " mold at " + temperature + "°C");
        System.out.println("Production rate: " + productionRate + "
```

```
units/hour");
}
}
class AssemblyRobot extends ManufacturingMachine
{
int precisionLevel;
AssemblyRobot()
{
super("Assembly Robot", 50);
this.precisionLevel = 5;
}
AssemblyRobot(int precision)
{
super("High-Precision Robot", 30);
this.precisionLevel = precision;
}
@Override
void operate()
{
System.out.println("Robot assembling with precision level " +
precisionLevel);
System.out.println("Production rate: " + productionRate + "
units/hour");
}
}
public class ManufacturingDemo2
{
public static void main(String[] args)
{
System.out.println("=== MANUFACTURING SYSTEM ===");
ManufacturingMachine machine1 = new ManufacturingMachine();
machine1.operate();
ManufacturingMachine machine2 = new ManufacturingMachine("CNC",
75);
machine2.operate();
System.out.println("\nSpecialized Machines:");
InjectionMolder molder1 = new InjectionMolder();
molder1.operate();
AssemblyRobot robot1 = new AssemblyRobot();
robot1.operate();
AssemblyRobot robot2 = new AssemblyRobot(8);
robot2.operate();
}
}
```

**Output:**

```
C:\Problems>javac ManufacturingDemo2.java

C:\Problems>java ManufacturingDemo2.java
=== MANUFACTURING SYSTEM ===
Basic manufacturing machine created
General Purpose machine operating at 100 units/hour
Custom CNC machine created with rate: 75
CNC machine operating at 75 units/hour

Specialized Machines:
Basic manufacturing machine created
Injection molding machine operating with Standard mold at 200°C
Production rate: 100 units/hour
Custom Assembly Robot machine created with rate: 50
Robot assembling with precision level 5
Production rate: 50 units/hour
Custom High-Precision Robot machine created with rate: 30
Robot assembling with precision level 8
Production rate: 30 units/hour
```

# ABSTRACTION

## 12. INTERFACE PROGRAMS:

### 12.a) Class ZeptoDelivery implements DeliveryService

#### Code:

```
interface DeliveryService
{
    void assignDelivery(String orderId, String address);
    void trackDelivery(String orderId);
    void cancelDelivery(String orderId);
    double calculateFee(double distance, boolean isPriority);
}
class ZeptoDelivery implements DeliveryService
{
    public void assignDelivery(String orderId, String address)
    {
        System.out.println("Zepto: Delivery assigned for order " +
            orderId);
        System.out.println("Delivery address: " + address);
        System.out.println("10-min delivery promise activated!");
    }
    public void trackDelivery(String orderId)
    {
        System.out.println("Zepto: Tracking order " + orderId);
        System.out.println("Current status: Out for delivery");
        System.out.println("Estimated arrival: 8 minutes");
    }
    public void cancelDelivery(String orderId)
    {
        System.out.println("Zepto: Cancelling order " + orderId);
        System.out.println("Full refund will be processed");
    }
    public double calculateFee(double distance, boolean isPriority)
    {
        double baseFee = 25.0;
        double distanceCharge = distance * 5.0;
        double priorityFee = isPriority ? 15.0 : 0.0;
        return baseFee + distanceCharge + priorityFee;
    }
}
public class ZeptoDeliveryDemo
```

```
{
public static void main(String[] args)
{
DeliveryService zepto = new ZeptoDelivery();
System.out.println("=== ZEPTO DELIVERY DEMO ===");
zepto.assignDelivery("ORD12345", "123 MG Road, Bangalore");
System.out.println("Delivery fee: ₹" + zepto.calculateFee(3.5,
true));
zepto.trackDelivery("ORD12345");
zepto.cancelDelivery("ORD12345");
}
}
```

**Output:**

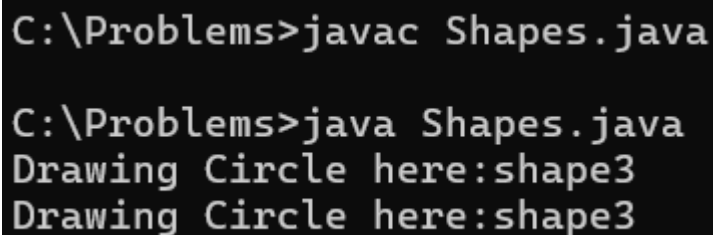
```
C:\Problems>javac ZeptoDeliveryDemo.java

C:\Problems>java ZeptoDeliveryDemo.java
=== ZEPTO DELIVERY DEMO ===
Zepto: Delivery assigned for order ORD12345
Delivery address: 123 MG Road, Bangalore
10-min delivery promise activated!
Delivery fee: ₹57.5
Zepto: Tracking order ORD12345
Current status: Out for delivery
Estimated arrival: 8 minutes
Zepto: Cancelling order ORD12345
Full refund will be processed
```

**12.b) Class shapeG implements shapeY****Code:**

```
interface shapeX
{
public String base = "shape1";
public void Draw();
}
interface shapeY extends shapeX
{
public String base = "shape2";
public void Draw2();
}
```

```
class shapeG implements shapeY
{
public String base = "shape3";
public void Draw()
{
System.out.println("Drawing Circle here:" + base);
}
@Override
public void Draw2()
{
System.out.println("Drawing Circle here:" + base);
}
}
public class Shapes
{
public static void main(String[] args)
{
shapeG circlesshape = new shapeG();
circleshape.Draw();
circleshape.Draw();
}
}
```

**Output:**

```
C:\Problems>javac Shapes.java

C:\Problems>java Shapes.java
Drawing Circle here:shape3
Drawing Circle here:shape3
```

**12.c) Class Cat implements Animal****Code:**

```
interface Animal
{
void makeSound();
void eat();
}
class Dog implements Animal
{
@Override
public void makeSound()
```

```
{
System.out.println("Woof! Woof!");
}
@Override
public void eat()
{
System.out.println("Dog is eating kibble");
}
public void fetch()
{
System.out.println("Dog fetches the ball");
}
}
class Cat implements Animal
{
@Override
public void makeSound()
{
System.out.println("Meow!");
}
@Override
public void eat()
{
System.out.println("Cat is eating fish");
}
public void scratch()
{
System.out.println("Cat scratches the furniture");
}
}
public class Animals
{
public static void main(String[] args)
{
Animal dog = new Dog();
Animal cat = new Cat();
System.out.println("--- Dog Activities ---");
dog.makeSound();
dog.eat();
((Dog)dog).fetch();
System.out.println("\n--- Cat Activities ---");
cat.makeSound();
cat.eat();
((Cat)cat).scratch();
}
}
```



**Output:**

```
C:\Problems>javac Animals.java

C:\Problems>java Animals.java
--- Dog Activities ---
Woof! Woof!
Dog is eating kibble
Dog fetches the ball

--- Cat Activities ---
Meow!
Cat is eating fish
Cat scratches the furniture
```

**12.d) Class SmartTV implements Television****Code:**

```
interface Television {
    void turnOn();
    void turnOff();
}
class SmartTV implements Television {
    @Override
    public void turnOn() {
        System.out.println("Smart TV is turning on...");
    }
    @Override
    public void turnOff() {
        System.out.println("Smart TV is turning off...");
    }
    public void browseInternet() {
        System.out.println("Smart TV is browsing the internet");
    }
}
class LEDTV implements Television {
    @Override
    public void turnOn() {
        System.out.println("LED TV is turning on...");
    }
    @Override
    public void turnOff() {
```

```
System.out.println("LED TV is turning off...");
}
public void adjustBrightness() {
System.out.println("LED TV brightness is being adjusted");
}
}
public class Viewers {
public static void main(String[] args) {
Television smartTv = new SmartTV();
Television ledTv = new LEDTV();
System.out.println("--- Smart TV Activities ---");
smartTv.turnOn();
((SmartTV) smartTv).browseInternet();
smartTv.turnOff();
System.out.println("\n--- LED TV Activities ---");
ledTv.turnOn();
((LEDTV) ledTv).adjustBrightness();
ledTv.turnOff();
}
}
```

**Output:**

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac Viewers.java

C:\Users\susendran\OneDrive\Desktop\Problems>java Viewers.java
--- Smart TV Activities ---
Smart TV is turning on...
Smart TV is browsing the internet
Smart TV is turning off...

--- LED TV Activities ---
LED TV is turning on...
LED TV brightness is being adjusted
LED TV is turning off...
```

## 13. ABSTRACT CLASS PROGRAMS:

### 13.a) Class ZeptoDelivery extends DeliveryService

#### Code:

```
abstract class DeliveryService {
    abstract void assignDelivery(String orderId, String address);
    abstract void trackDelivery(String orderId);
    abstract void cancelDelivery(String orderId);
    abstract double calculateFee(double distance, boolean isPriority);
}
class ZeptoDelivery extends DeliveryService {
    public void assignDelivery(String orderId, String address) {
        System.out.println("Zepto: Delivery assigned for order " + orderId);
        System.out.println("Delivery address: " + address);
        System.out.println("10-min delivery promise activated!");
    }
    public void trackDelivery(String orderId) {
        System.out.println("Zepto: Tracking order " + orderId);
        System.out.println("Current status: Out for delivery");
        System.out.println("Estimated arrival: 8 minutes");
    }
    public void cancelDelivery(String orderId) {
        System.out.println("Zepto: Cancelling order " + orderId);
        System.out.println("Full refund will be processed");
    }
    public double calculateFee(double distance, boolean isPriority) {
        double baseFee = 25.0;
        double distanceCharge = distance * 5.0;
        double priorityFee = isPriority ? 15.0 : 0.0;
        return baseFee + distanceCharge + priorityFee;
    }
}
public class ZeptoDeliveryDemo1 {
    public static void main(String[] args) {
        DeliveryService zepto = new ZeptoDelivery();
        System.out.println("=== ZEPTO DELIVERY DEMO ===");
        zepto.assignDelivery("ORD12345", "123 MG Road, Bangalore");
        System.out.println("Delivery fee: ₹" + zepto.calculateFee(3.5, true));
        zepto.trackDelivery("ORD12345");
        zepto.cancelDelivery("ORD12345");
    }
}
```

**Output:**

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac ZeptoDeliveryDemo1.java

C:\Users\susendran\OneDrive\Desktop\Problems>java ZeptoDeliveryDemo1.java
=== ZEPTO DELIVERY DEMO ===
Zepto: Delivery assigned for order ORD12345
Delivery address: 123 MG Road, Bangalore
10-min delivery promise activated!
Delivery fee: ₹57.5
Zepto: Tracking order ORD12345
Current status: Out for delivery
Estimated arrival: 8 minutes
Zepto: Cancelling order ORD12345
Full refund will be processed
```

**13.b) Class ShapeG extends ShapeY****Code:**

```
abstract class ShapeX {
    public String base = "shape1";
    public abstract void Draw();
}
abstract class ShapeY extends ShapeX {
    public String base = "shape2";
    public abstract void Draw2();
}
class ShapeG extends ShapeY {
    public String base = "shape3";
    @Override
    public void Draw() {
        System.out.println("Drawing Circle here: " + base);
    }
    @Override
    public void Draw2() {
        System.out.println("Drawing Circle here: " + base);
    }
}
public class Shapes1 {
    public static void main(String[] args) {
        ShapeG circleshape = new ShapeG();
        circleshape.Draw();
        circleshape.Draw2();
    }
}
```

**Output:**

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac Shapes1.java  
C:\Users\susendran\OneDrive\Desktop\Problems>java Shapes1.java  
Drawing Circle here: shape3  
Drawing Circle here: shape3
```

**13.c) Class Cat extends Animal****Code:**

```
abstract class Animal {  
    abstract void makeSound();  
    abstract void eat();  
}  
class Dog extends Animal {  
    @Override  
    public void makeSound() {  
        System.out.println("Woof! Woof!");  
    }  
    @Override  
    public void eat() {  
        System.out.println("Dog is eating kibble");  
    }  
    public void fetch() {  
        System.out.println("Dog fetches the ball");  
    }  
}  
class Cat extends Animal {  
    @Override  
    public void makeSound() {  
        System.out.println("Meow!");  
    }  
    @Override  
    public void eat() {  
        System.out.println("Cat is eating fish");  
    }  
    public void scratch() {  
        System.out.println("Cat scratches the furniture");  
    }  
}  
public class Animals1 {  
    public static void main(String[] args) {  
        Animal dog = new Dog();
```

```
Animal cat = new Cat();
System.out.println("--- Dog Activities ---");
dog.makeSound();
dog.eat();
((Dog)dog).fetch();
System.out.println("\n--- Cat Activities ---");
cat.makeSound();
cat.eat();
((Cat)cat).scratch();
}
}
```

**Output:**

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac Animals1.java

C:\Users\susendran\OneDrive\Desktop\Problems>java Animals1.java
--- Dog Activities ---
Woof! Woof!
Dog is eating kibble
Dog fetches the ball

--- Cat Activities ---
Meow!
Cat is eating fish
Cat scratches the furniture
```

**13.d) Class SmartTV implements Television****Code:**

```
abstract class Television {
    abstract void turnOn();
    abstract void turnOff();
}
class SmartTV extends Television {
    @Override
    public void turnOn() {
        System.out.println("Smart TV is turning on...");
    }
    @Override
    public void turnOff() {
        System.out.println("Smart TV is turning off...");
    }
    public void browseInternet() {
```

```
System.out.println("Smart TV is browsing the internet");
}
}
class LEDTV extends Television {
@Override
public void turnOn() {
System.out.println("LED TV is turning on...");
}
@Override
public void turnOff() {
System.out.println("LED TV is turning off...");
}
public void adjustBrightness() {
System.out.println("LED TV brightness is being adjusted");
}
}
public class Viewers1 {
public static void main(String[] args) {
Television smartTv = new SmartTV();
Television ledTv = new LEDTV();
System.out.println("--- Smart TV Activities ---");
smartTv.turnOn();
((SmartTV) smartTv).browseInternet();
smartTv.turnOff();
System.out.println("\n--- LED TV Activities ---");
ledTv.turnOn();
((LEDTV) ledTv).adjustBrightness();
ledTv.turnOff();
}
}
```

**Output:**

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac Viewers.java

C:\Users\susendran\OneDrive\Desktop\Problems>java Viewers.java
--- Smart TV Activities ---
Smart TV is turning on...
Smart TV is browsing the internet
Smart TV is turning off...

--- LED TV Activities ---
LED TV is turning on...
LED TV brightness is being adjusted
LED TV is turning off...
```

## 14. ENCAPSULATION PROGRAMS:

### 14.a) Class Student private variables SimpleGradebook

#### Code:

```
class Student {
private String name;
private int grade1;
private int grade2;
private int grade3;
public Student(String name) {
this.name = name;
this.grade1 = -1;
this.grade2 = -1;
this.grade3 = -1;
}
public void setGrade(int whichGrade, int score) {
if (score < 0 || score > 100) {
System.out.println("Invalid grade! Use 0-100");
return;
}
switch (whichGrade) {
case 1: grade1 = score; break;
case 2: grade2 = score; break;
case 3: grade3 = score; break;
default: System.out.println("Use 1, 2 or 3 for grade slots");
}
}
public double getAverage() {
int count = 0;
int sum = 0;
if (grade1 != -1) { sum += grade1; count++; }
if (grade2 != -1) { sum += grade2; count++; }
if (grade3 != -1) { sum += grade3; count++; }
if (count == 0) return 0;
return (double)sum / count;
}
public String getName() {
return name;
}
public void showGrades() {
System.out.println(name + "'s grades:");
if (grade1 != -1) System.out.println("Grade 1: " + grade1);
```



```
if (grade2 != -1) System.out.println("Grade 2: " + grade2);
if (grade3 != -1) System.out.println("Grade 3: " + grade3);
}
}
public class SimpleGradebook {
    public static void main(String[] args) {
        Student s = new Student("Alex");
        s.setGrade(1, 85);
        s.setGrade(2, 92);
        s.setGrade(3, 78);
        s.setGrade(1, 105);
        s.showGrades();
        System.out.println("Average: " + s.getAverage());
    }
}
```

**Output:**

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac SimpleGradebook.java

C:\Users\susendran\OneDrive\Desktop\Problems>java SimpleGradebook.java
Invalid grade! Use 0-100
Alex's grades:
Grade 1: 85
Grade 2: 92
Grade 3: 78
Average: 85.0
```

**14.b) Class Vehicle variables OdometerDemo****Code:**

```
class Vehicle {
    private String model;
    private int currentMileage;
    private int lastServiceMileage;
    public Vehicle(String model) {
        this.model = model;
        this.currentMileage = 0;
        this.lastServiceMileage = 0;
    }
    public void addMiles(int miles) {
        if (miles > 0) {
            currentMileage += miles;
        } else {
```

```
System.out.println("Error: Miles must be positive");
}
}
public boolean isServiceDue() {
return (currentMileage - lastServiceMileage) >= 5000;
}
public void servicePerformed() {
lastServiceMileage = currentMileage;
System.out.println("Service recorded at " + currentMileage + "
miles");
}
public int getCurrentMileage() {
return currentMileage;
}
public String getModel() {
return model;
}
}
public class OdometerDemo {
public static void main(String[] args) {
Vehicle car = new Vehicle("Toyota Corolla");
car.addMiles(3000);
System.out.println(car.getModel() + " mileage: " +
car.getCurrentMileage());
System.out.println("Service due? " + car.isServiceDue());
car.addMiles(2500);
System.out.println("New mileage: " + car.getCurrentMileage());
System.out.println("Service due? " + car.isServiceDue());
car.servicePerformed();
}
}
```

**Output:**

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac OdometerDemo.java

C:\Users\susendran\OneDrive\Desktop\Problems>java OdometerDemo.java
Toyota Corolla mileage: 3000
Service due? false
New mileage: 5500
Service due? true
Service recorded at 5500 miles
```

## 14.c) Class TempConverter variables TempConverterDemo

### Code:

```
class TempConverter {
    private double celsius;
    public TempConverter() {
        this.celsius = 0;
    }
    public void setCelsius(double celsius) {
        this.celsius = celsius;
    }
    public void setFahrenheit(double fahrenheit) {
        this.celsius = (fahrenheit - 32) * 5 / 9;
    }
    public double getCelsius() {
        return celsius;
    }
    public double getFahrenheit() {
        return celsius * 9 / 5 + 32;
    }
}

public class TempConverterDemo {
    public static void main(String[] args) {
        TempConverter temp = new TempConverter();
        temp.setCelsius(25);
        System.out.println("25°C is " + temp.getFahrenheit() + "°F");
        temp.setFahrenheit(77);
        System.out.println("77°F is " + temp.getCelsius() + "°C");
    }
}
```

### Output:

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac TempConverterDemo.java

C:\Users\susendran\OneDrive\Desktop\Problems>java TempConverterDemo.java
25°C is 77.0°F
77°F is 25.0°C
```

## 14.d) Class HealthMonitor variables HealthMonitorDemo

### Code:

```
class HealthMonitor {
    private String patientName;
    private int heartRate;
    private int systolicBP;
    private int diastolicBP;
    public HealthMonitor(String name) {
        this.patientName = name;
        this.heartRate = 0;
        this.systolicBP = 0;
        this.diastolicBP = 0;
    }
    public void setHeartRate(int rate) {
        if (rate >= 30 && rate <= 200) {
            this.heartRate = rate;
        } else {
            System.out.println("Invalid heart rate! Must be 30-200 bpm");
        }
    }
    public void setBloodPressure(int systolic, int diastolic) {
        if (systolic > diastolic && systolic > 0 && diastolic > 0) {
            this.systolicBP = systolic;
            this.diastolicBP = diastolic;
        } else {
            System.out.println("Invalid blood pressure readings!");
        }
    }
    public String checkHealthStatus() {
        if (heartRate == 0 || systolicBP == 0) {
            return "No data recorded";
        }
        String status = "Normal";
        if (heartRate > 100) status = "Elevated Heart Rate";
        if (systolicBP > 140 || diastolicBP > 90) status = "High Blood Pressure";
        if (heartRate > 100 && (systolicBP > 140 || diastolicBP > 90))
            status = "Critical Condition";
        return status;
    }
    public String getPatientName() {
        return patientName;
    }
    public String getVitals() {
        return "HR: " + heartRate + " bpm, BP: " + systolicBP + "/" +
            diastolicBP + " mmHg";
    }
}
```

```
}  
}  
public class HealthMonitorDemo {  
    public static void main(String[] args) {  
        HealthMonitor patient = new HealthMonitor("John Doe");  
        patient.setHeartRate(85);  
        patient.setBloodPressure(120, 80);  
        System.out.println("Patient: " + patient.getPatientName());  
        System.out.println("Vitals: " + patient.getVitals());  
        System.out.println("Status: " + patient.checkHealthStatus());  
        patient.setHeartRate(110);  
        patient.setBloodPressure(150, 95);  
        System.out.println("\nNew Vitals: " + patient.getVitals());  
        System.out.println("Status: " + patient.checkHealthStatus());  
    }  
}
```

**Output:**

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac HealthMonitorDemo.java  
  
C:\Users\susendran\OneDrive\Desktop\Problems>java HealthMonitorDemo.java  
Patient: John Doe  
Vitals: HR: 85 bpm, BP: 120/80 mmHg  
Status: Normal  
  
New Vitals: HR: 110 bpm, BP: 150/95 mmHg  
Status: Critical Condition
```

## 15. PACKAGE PROGRAMS:

### 15.a) Import Package vehicles, electronics Class VehicleSystem

#### Code:

##### Engine.java

```
package vehicles;
public class Engine {
    private String type;
    private int horsepower;
    public Engine(String type, int horsepower) {
        this.type = type;
        this.horsepower = horsepower;
    }
    public String getType() {
        return type;
    }
    public int getHorsepower() {
        return horsepower;
    }
}
```

##### GPS.java

```
package electronics;
public class GPS {
    private String model;
    public GPS(String model) {
        this.model = model;
    }
    public void navigate(String destination) {
        System.out.println(model + " GPS navigating to " + destination);
    }
}
```

##### VehicleSystem.java

```
import vehicles.Engine;
import electronics.GPS;
public class VehicleSystem {
```

```
public static void main(String[] args) {
    Engine v6 = new Engine("V6", 300);
    GPS navSystem = new GPS("Garmin");
    Car myCar = new Car("Toyota Camry", v6, navSystem);
    myCar.displaySpecs();
    myCar.driveTo("New York");
}
}
class Car
{
    private String model;
    private Engine engine;
    private GPS gps;
    public Car(String model, Engine engine, GPS gps) {
        this.model = model;
        this.engine = engine;
        this.gps = gps;
    }
    public void displaySpecs() {
        System.out.println("\n=== VEHICLE SPECS ===");
        System.out.println("Model: " + model);
        System.out.println("Engine: " + engine.getType());
        System.out.println("Horsepower: " + engine.getHorsepower() + "
        HP");
    }
    public void driveTo(String destination) {
        System.out.println("\nStarting journey...");
        gps.navigate(destination);
        System.out.println("Arrived at " + destination);
    }
}
}
```

## Output:

```
C:\Users\susendran\OneDrive\Desktop\user1>javac -d . Engine.java
C:\Users\susendran\OneDrive\Desktop\user1>javac -d . GPS.java
C:\Users\susendran\OneDrive\Desktop\user1>javac VehicleSystem.java
C:\Users\susendran\OneDrive\Desktop\user1>java VehicleSystem

=== VEHICLE SPECS ===
Model: Toyota Camry
Engine: V6
Horsepower: 300 HP

Starting journey...
Garmin GPS navigating to New York
Arrived at New York
```

## 15.b) Import Package banking, transactions Class BankApplication

### Code:

#### Account.java

```
package banking;
public class Account {
    private String accountNumber;
    private double balance;
    public Account(String accountNumber, double initialBalance) {
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }
    public boolean withdraw(double amount) {
        if (amount > 0 && balance >= amount) {
            balance -= amount;
            return true;
        }
        return false;
    }
    public double getBalance() {
        return balance;
    }
    public String getAccountNumber() {
        return accountNumber;
    }
}
```

#### Transaction.java

```
package transactions;
import banking.Account;
public class Transaction {
    public static void transfer(Account from, Account to, double amount) {
        if (from.withdraw(amount)) {
            to.deposit(amount);
            System.out.println("Transfer successful: $" + amount);
        }
        else {
            System.out.println("Transfer failed: Insufficient funds");
        }
    }
}
```



```
}  
}  
}
```

### BankApplication.java

```
import banking.Account;  
import transactions.Transaction;  
public class BankApplication {  
    public static void main(String[] args) {  
        Account alice = new Account("ACC1001", 1000);  
        Account bob = new Account("ACC1002", 500);  
        System.out.println("Initial Balances:");  
        System.out.println("Alice: $" + alice.getBalance());  
        System.out.println("Bob: $" + bob.getBalance());  
        Transaction.transfer(alice, bob, 300);  
        System.out.println("\nAfter Transfer:");  
        System.out.println("Alice: $" + alice.getBalance());  
        System.out.println("Bob: $" + bob.getBalance());  
    }  
}
```

### Output:

```
C:\Users\susendran\OneDrive\Desktop\user2>javac -d . Account.java  
C:\Users\susendran\OneDrive\Desktop\user2>javac -d . Transaction.java  
C:\Users\susendran\OneDrive\Desktop\user2>javac BankApplication.java  
C:\Users\susendran\OneDrive\Desktop\user2>java BankApplication.java  
Initial Balances:  
Alice: $1000.0  
Bob: $500.0  
Transfer successful: $300.0  
  
After Transfer:  
Alice: $700.0  
Bob: $800.0
```

## 15.c) Import io, lang, util Class UserInputFile

### Code:

```
import java.util.Scanner;
import java.io.FileWriter;
import java.io.IOException;
public class UserInputToFile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = scanner.nextLine();
        try {
            FileWriter writer = new FileWriter("output.txt");
            writer.write("Hello, " + name + "!");
            writer.close();
            System.out.println("Data saved to 'output.txt'!");
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
        scanner.close();
    }
}
```

### Output:

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac UserInputToFile.java

C:\Users\susendran\OneDrive\Desktop\Problems>java UserInputToFile.java
Enter your name: susendran
Data saved to 'output.txt'!
```

## 15.d) Import awt, net, sql Class Packed

### Code:

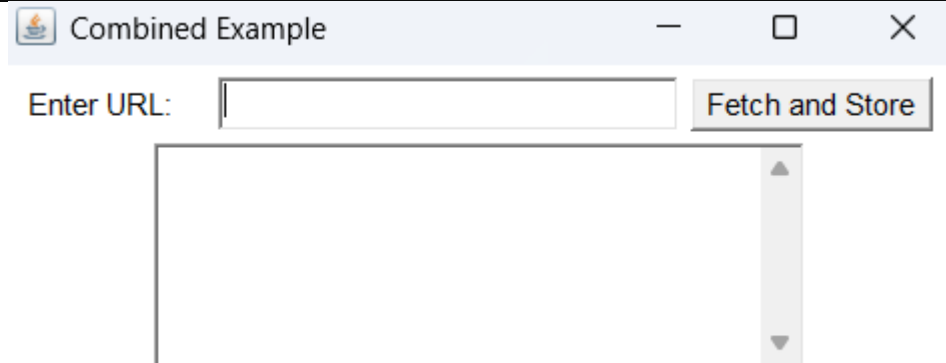
```
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.sql.*;
public class Packed {
    public static void main(String[] args) {
```

```
Frame frame = new Frame("Combined Example");
frame.setLayout(new FlowLayout());
frame.setSize(400, 200);
Label label = new Label("Enter URL:");
TextField urlField = new TextField(20);
Button fetchButton = new Button("Fetch and Store");
TextArea resultArea = new TextArea(5, 30);
frame.add(label);
frame.add(urlField);
frame.add(fetchButton);
frame.add(resultArea);
fetchButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String url = urlField.getText();
        try {
            URLConnection conn = new URL(url).openConnection();
            BufferedReader reader = new BufferedReader(
                new InputStreamReader(conn.getInputStream()));
            StringBuilder content = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                content.append(line).append("\n");
            }
            reader.close();
            Connection dbConn = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root", "password");
            PreparedStatement stmt = dbConn.prepareStatement(
                "INSERT INTO web_content (url, content) VALUES (?, ?)");
            stmt.setString(1, url);
            stmt.setString(2, content.toString());
            stmt.executeUpdate();
            stmt.close();
            dbConn.close();
            resultArea.setText("Successfully fetched and stored content
from:\n" + url);
        } catch (Exception ex) {
            resultArea.setText("Error: " + ex.getMessage());
        }
    }
});
frame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
frame.setVisible(true);
}
```

**Output:**

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac Packed.java
Note: Packed.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

```
C:\Users\susendran\OneDrive\Desktop\Problems>java Packed
```



The screenshot shows a Java Swing window titled "Combined Example". It features a text input field labeled "Enter URL:" and a button labeled "Fetch and Store". Below the input field is a large, empty text area with a vertical scrollbar on the right side.

## 16. EXCEPTION HANDLING PROGRAMS

### 16.a) NumberFormat, Arithmetic, Exception with try, catch

#### Code:

```
import java.util.Scanner;
public class ExceptionHandling1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            System.out.print("Enter a number: ");
            int num1 = Integer.parseInt(scanner.nextLine());
            System.out.print("Enter another number: ");
            int num2 = Integer.parseInt(scanner.nextLine());
            int result = num1 / num2;
            System.out.println("Result: " + result);
        } catch (NumberFormatException e) {
            System.out.println("Error: Please enter valid numbers!");
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero!");
        } catch (Exception e) {
            System.out.println("An unexpected error occurred: " +
                e.getMessage());
        } finally {
            scanner.close();
            System.out.println("Program completed.");
        }
    }
}
```

#### Output:

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac ExceptionHandling1.java
C:\Users\susendran\OneDrive\Desktop\Problems>java ExceptionHandling1.java
Enter a number: 3
Enter another number: 5
Result: 0
Program completed.
```

## 16.b) IllegalArgumentException, InputMismatch, Arithmetic, Exception with try and catch

### Code:

```
import java.util.InputMismatchException;
import java.util.Scanner;
public class ExceptionHandling2 {
    public static void main(String[] args) {
        final double MIN_BALANCE = 100.0;
        Scanner scanner = new Scanner(System.in);
        double balance = 500.0;
        try {
            System.out.print("Enter withdrawal amount: ");
            double amount = scanner.nextDouble();
            if (amount <= 0) {
                throw new IllegalArgumentException("Amount must be positive");
            }
            if ((balance - amount) < MIN_BALANCE) {
                throw new ArithmeticException("Insufficient funds - cannot go below minimum balance");
            }
            balance -= amount;
            System.out.printf("Withdrawal successful. New balance: %.2f\n", balance);
        } catch (InputMismatchException e) {
            System.out.println("Error: Please enter a valid number");
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (ArithmeticException e) {
            System.out.println("Transaction failed: " + e.getMessage());
        } finally {
            scanner.close();
            System.out.println("Thank you for banking with us!");
            final String TRANSACTION_COMPLETE = "Transaction processed";
            System.out.println(TRANSACTION_COMPLETE);
        }
    }
}
```

### Output:

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac ExceptionHandling2.java

C:\Users\susendran\OneDrive\Desktop\Problems>java ExceptionHandling2.java
Enter withdrawal amount: 2000
Transaction failed: Insufficient funds - cannot go below minimum balance
Thank you for banking with us!
Transaction processed
```

## 16.c) InvalidAge, Exception with throw

### Code:

```
import java.util.Scanner;
class InvalidAgeException extends Exception {
public InvalidAgeException(String message) {
super(message);
}
}
public class ExceptionHandling3 {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
try {
System.out.print("Enter your age: ");
int age = scanner.nextInt();
if (age < 0) {
throw new InvalidAgeException("Age cannot be negative");
} else if (age > 120) {
throw new InvalidAgeException("Age cannot be more than 120");
}
System.out.println("Valid age entered: " + age);
} catch (InvalidAgeException e) {
System.out.println("Error: " + e.getMessage());
} catch (Exception e) {
System.out.println("Invalid input");
} finally {
scanner.close();
System.out.println("Program completed");
}
}
}
```

### Output:

```
C:\Users\susendran\OneDrive\Desktop\Problems>javac ExceptionHandling3.java
C:\Users\susendran\OneDrive\Desktop\Problems>java ExceptionHandling3.java
Enter your age: 19
Valid age entered: 19
Program completed
```

## 16.d) IllegalArgumentException, InsufficientFunds, Exception with throw

### Code:

```
import java.util.Scanner;
class InsufficientFundsException extends Exception {
public InsufficientFundsException(String message) {
super(message);
}
}
public class ExceptionHandling4 {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
double balance = 1000.0;
try {
System.out.print("Enter withdrawal amount: ");
double amount = scanner.nextDouble();
if (amount <= 0) {
throw new IllegalArgumentException("Amount must be positive");
}
if (amount > balance) {
throw new InsufficientFundsException("Not enough money in
account");
}
balance -= amount;
System.out.println("Remaining balance: " + balance);
} catch (IllegalArgumentException e) {
System.out.println("Error: " + e.getMessage());
} catch (InsufficientFundsException e) {
System.out.println("Transaction failed: " + e.getMessage());
} catch (Exception e) {
System.out.println("Invalid input");
} finally {
scanner.close();
System.out.println("Transaction completed");
}
}
```

### Output:



```
C:\Users\susendran\OneDrive\Desktop\Problems>javac ExceptionHandling4.java  
C:\Users\susendran\OneDrive\Desktop\Problems>java ExceptionHandling4.java  
Enter withdrawal amount: 1000  
Remaining balance: 0.0  
Transaction completed
```

## 17. FILE HANDLING PROGRAMS

### 17.a) FileWriter example.txt

#### Code:

```
import java.io.FileWriter;
import java.io.IOException;
public class SimpleFileWrite {
    public static void main(String[] args) {
        try (FileWriter writer = new FileWriter("example.txt")) {
            writer.write("Hello, this is even simpler!");
            System.out.println("File written successfully!");
        } catch (IOException e) {
            System.out.println("Error writing file: " + e.getMessage());
        }
    }
}
```

#### Output:

```
C:\Users\susendran\OneDrive\Desktop\FileHandling>javac SimpleFileWrite.java
C:\Users\susendran\OneDrive\Desktop\FileHandling>java SimpleFileWrite.java
File written successfully!
```

### 17.b) BufferedWriter output.txt

#### Code:

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
public class BufferedFileWrite {
    public static void main(String[] args) {
        try (BufferedWriter writer = new BufferedWriter(new
            FileWriter("output.txt"))) {
            writer.write("Using BufferedWriter is more efficient");
            writer.newLine();
            writer.write("Especially when writing multiple lines");
            writer.newLine();
        }
    }
}
```

```
writer.write("Or large amounts of data");
System.out.println("File written successfully!");
} catch (IOException e) {
System.out.println("An error occurred: " + e.getMessage());
}
}
}
```

### Output:

```
C:\Users\susendran\OneDrive\Desktop\FileHandling>javac BufferedFileWrite.java
C:\Users\susendran\OneDrive\Desktop\FileHandling>java BufferedFileWrite.java
File written successfully!
```

## 17.c) FileReader example.txt

### Code:

```
import java.io.FileReader;
import java.io.IOException;
public class SimpleFileRead {
public static void main(String[] args) {
try (FileReader reader = new FileReader("example.txt")) {
int character;
System.out.println("File content:");
while ((character = reader.read()) != -1) {
System.out.print((char) character);
}
} catch (IOException e) {
System.out.println("Error reading file: " + e.getMessage());
}
}
}
```

### Output:

```
C:\Users\susendran\OneDrive\Desktop\FileHandling>javac SimpleFileRead.java
C:\Users\susendran\OneDrive\Desktop\FileHandling>java SimpleFileRead.java
File content:
Hello, this is even simpler!
```

## 17.d) BufferedReader example.txt

### Code:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class BufferedFileRead {
    public static void main(String[] args) {
        try (BufferedReader reader = new BufferedReader(new
            FileReader("example.txt"))) {
            String line;
            System.out.println("File content:");
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + e.getMessage());
        }
    }
}
```

### Output:

```
C:\Users\susendran\OneDrive\Desktop\FileHandling>javac BufferedFileRead.java
C:\Users\susendran\OneDrive\Desktop\FileHandling>java BufferedFileRead.java
File content:
Hello, this is even simpler!
```