ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF COMPUTING

CHENNAI

March - 2025

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.U4CSE24031– Nishitha penagaluru*** 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

	3.h) Swapping numbers	
	3.i) Leap year	
	3.j) Amstrong number	
	INHERITANCE	
4.	**SINGLE INHERITANCE PROGRAMS**	
	4.a) Doctor details	
	4.b) Student marks	
5.	**MULTILEVEL INHERITANCE PROGRAMS**	
	5.a) Login check	
	5.b) Student details	
6.	**HIERARCHICAL INHERITANCE PROGRAMS**	
	6.a) Course	
	6.b) Payment	
7.	**HYBRID INHERITANCE PROGRAMS**	
	7.a) Sportsman	
	7.b) Appliances	
	POLYMORPHISM	
8.	**CONSTRUCTOR PROGRAMS**	
	8.a) Weather cast	
9.	**CONSTRUCTOR OVERLOADING PROGRAMS**	
	9.a) Students age	
10.	**METHOD OVERLOADING PROGRAMS**	
	10.a) Book details	
	10.b) temperature convertion	
11.	**METHOD OVERRIDING PROGRAMS**	
	11.a) Hospital	
	11.b) Appiliances	
	ABSTRACTION	
12.	**INTERFACE PROGRAMS**	
	12.a) mediaplayer	
	12.b) Bank	
	12.c) Payment	
	12.d) Taxcaluculator	
13.	**ABSTRACT CLASS PROGRAMS**	
	13.a) Order food	
	13.b) Report	
	13.c) Transport	
	13.d) Account type	
	ENCAPSULATION	
14.	**ENCAPSULATION PROGRAMS**	

	14.a) Patient details	
	14.b) Student info	
	14.c) Quiz	
	14.d) Product stock	
15.	**PACKAGES PROGRAMS**	
	15.a)User Defined Packages	
	15.b)User Defined Packages	
	15.c)Built – in Package(3 Packages)	
	15.d)Built – in Package(3 Packages)	
16.	**EXCEPTION HANDLING PROGRAMS**	
	16.a) Divided by zero	
	16.b) Marks	
	16.c) File not found	
	16.d) null point	
17.	**FILE HANDLING PROGRAMS**	
	17.a) create a file	
	17.b) Read a file	
	17.c) counting words	
	17.d) Listing all files	

UML DIAGRAM

1.a) Use Case Diagram

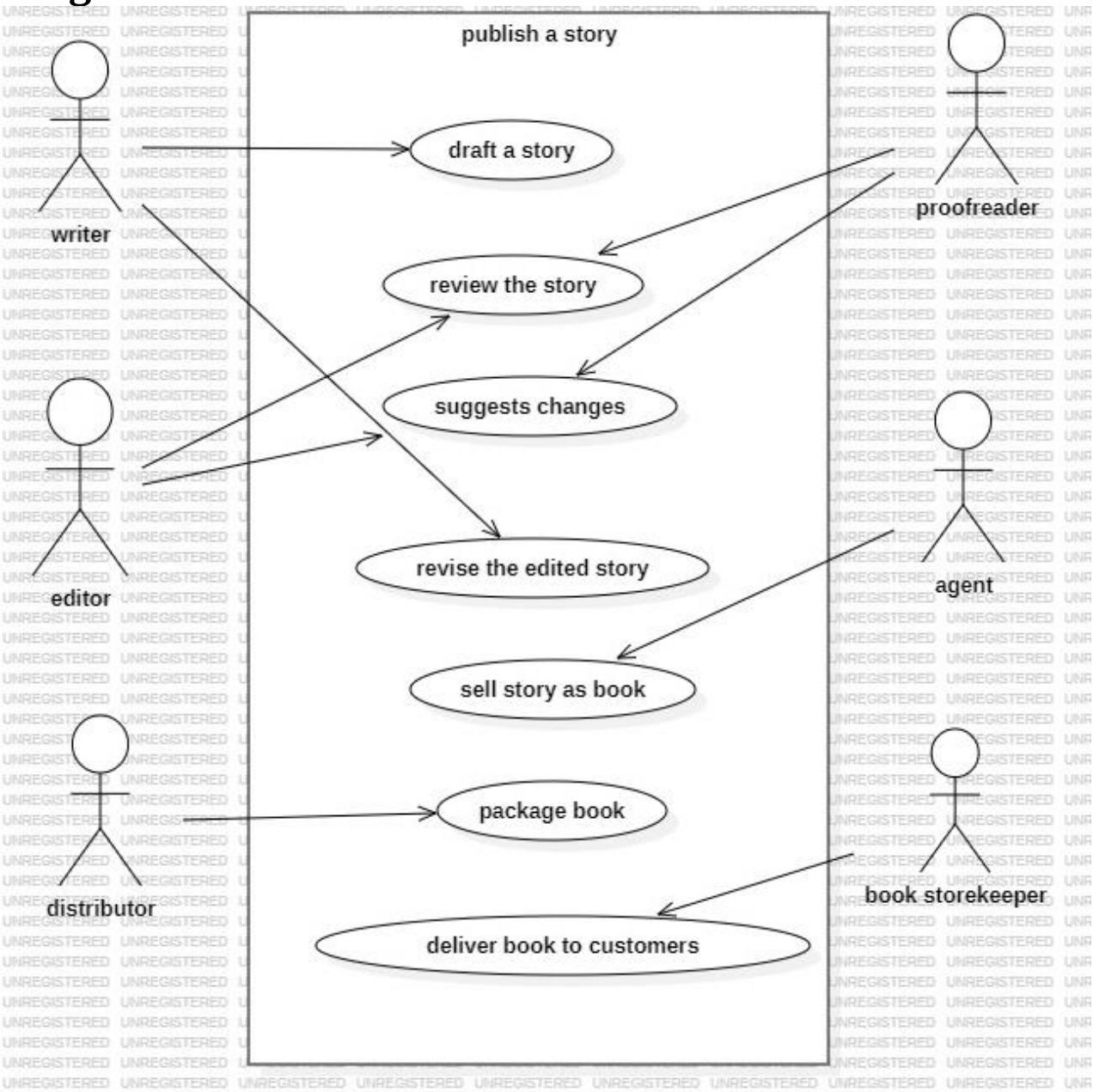
AIM: publishing a story

Software required: starUML

Algorithm:

1. **Start**
2. **Writer drafts a story**
3. **Editor reviews the story**
4. **Proofreader reviews the story**
5. **If changes are needed:**
    a. Editor suggests changes
    b. Writer revises the story based on suggestions
    c. Go back to **Step 3 (Review the story)**
6. **Once the story is approved:**
    a. Agent sells the story as a book
7. **Package the book**
8. **Distribute the book:**
    a. Distributor packages the book
    b. Book storekeeper delivers the book to customers
9. **End**

Diagram:



1.a)Use Case Diagram

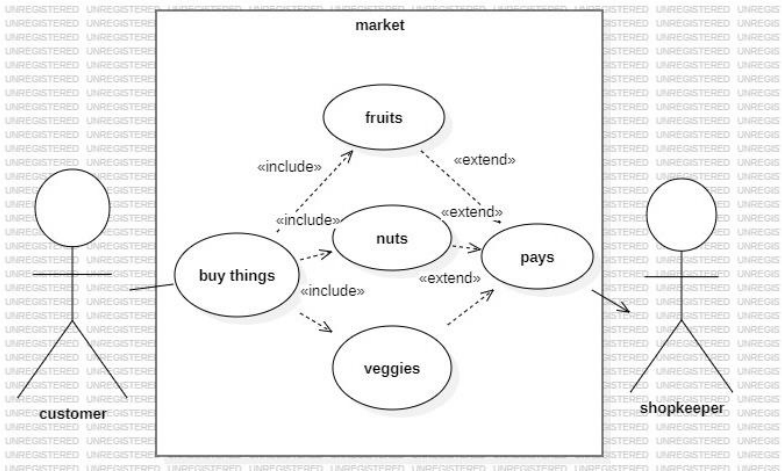
AIM: Market

SOFTWARE USED: STARUML

Algorithm: Customer Shopping in a Market

1. **Start**
2. **Customer enters the market**
3. **Customer decides to buy things**
    a. Includes:
        i. **Buy fruits**
        ii. **Buy veggies**
        iii. **Buy nuts**
4. **If nuts are bought**, the action may **extend to paying** the shopkeeper
5. **Customer pays the shopkeeper**
6. **End**

DIAGARM:

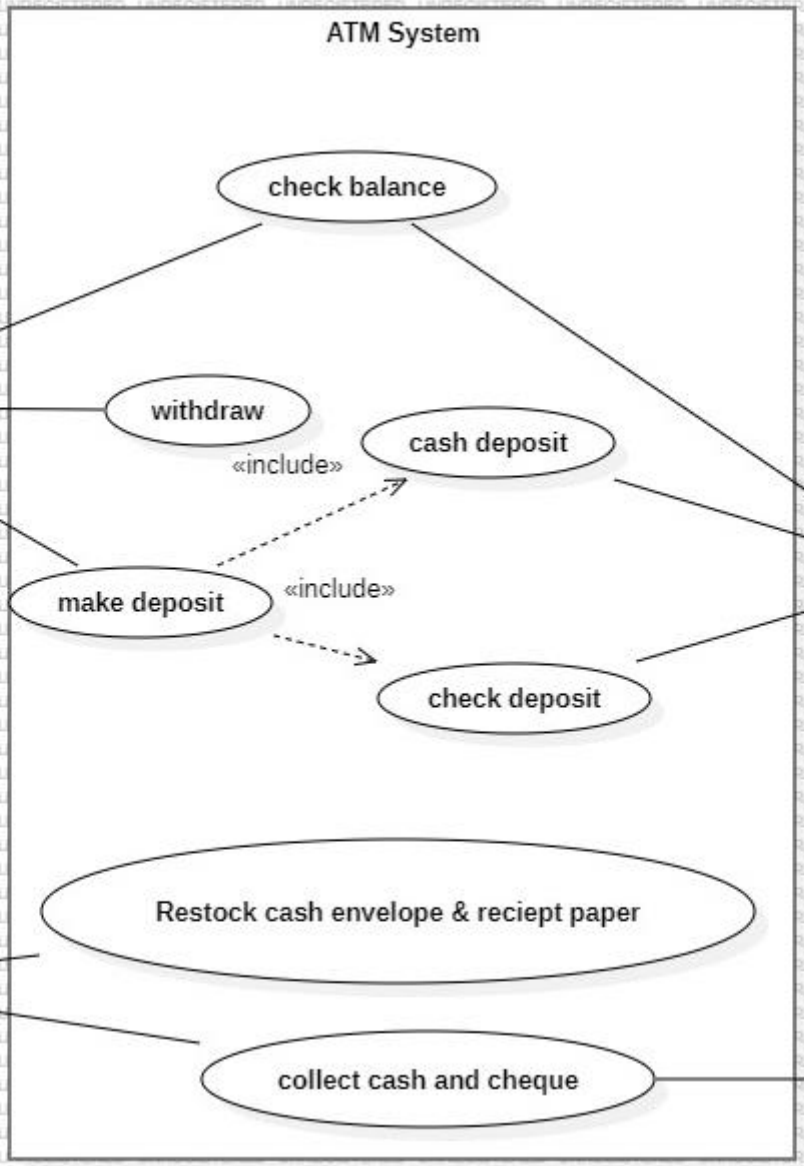


1.a)Use Case Diagram

AIM: ATM

SOFTWARE USED: STARUML

DIAGARM:

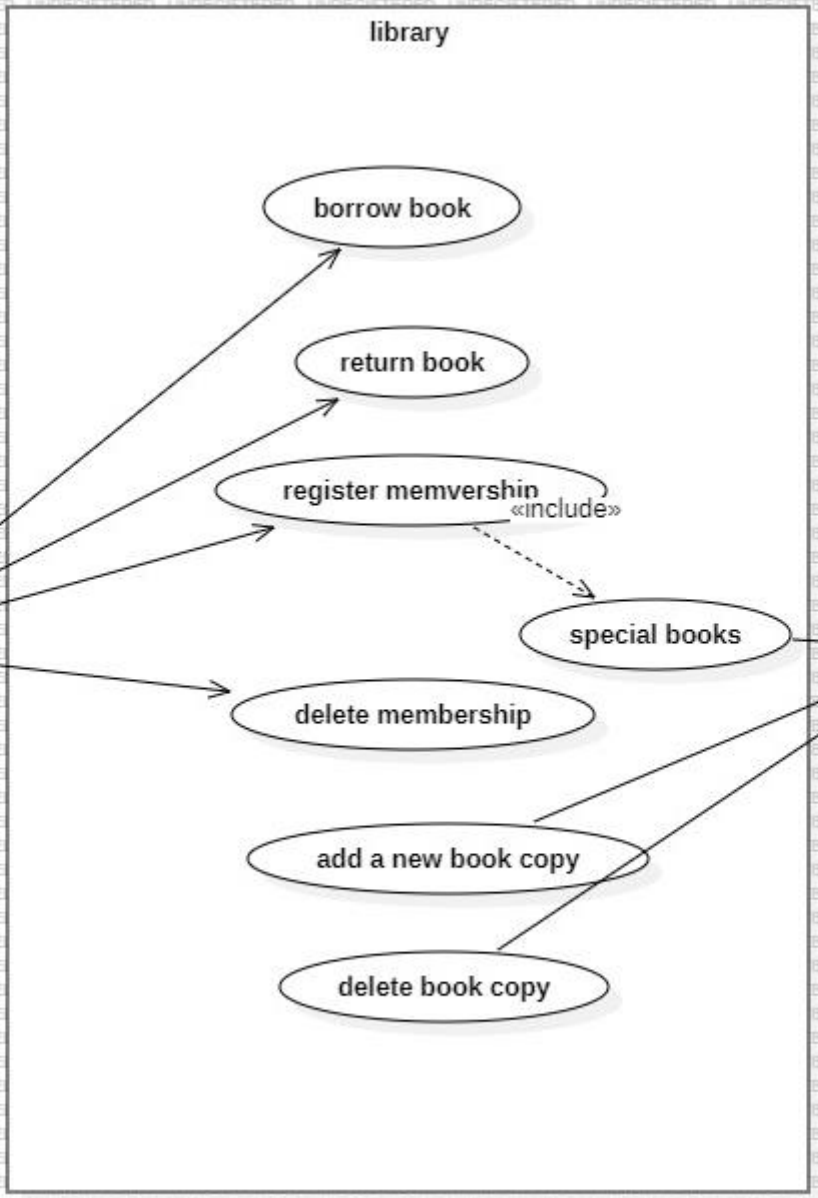


1.a)Use Case Diagram

AIM: Library

SOFTWARE USED: STARUML

DIAGARM:

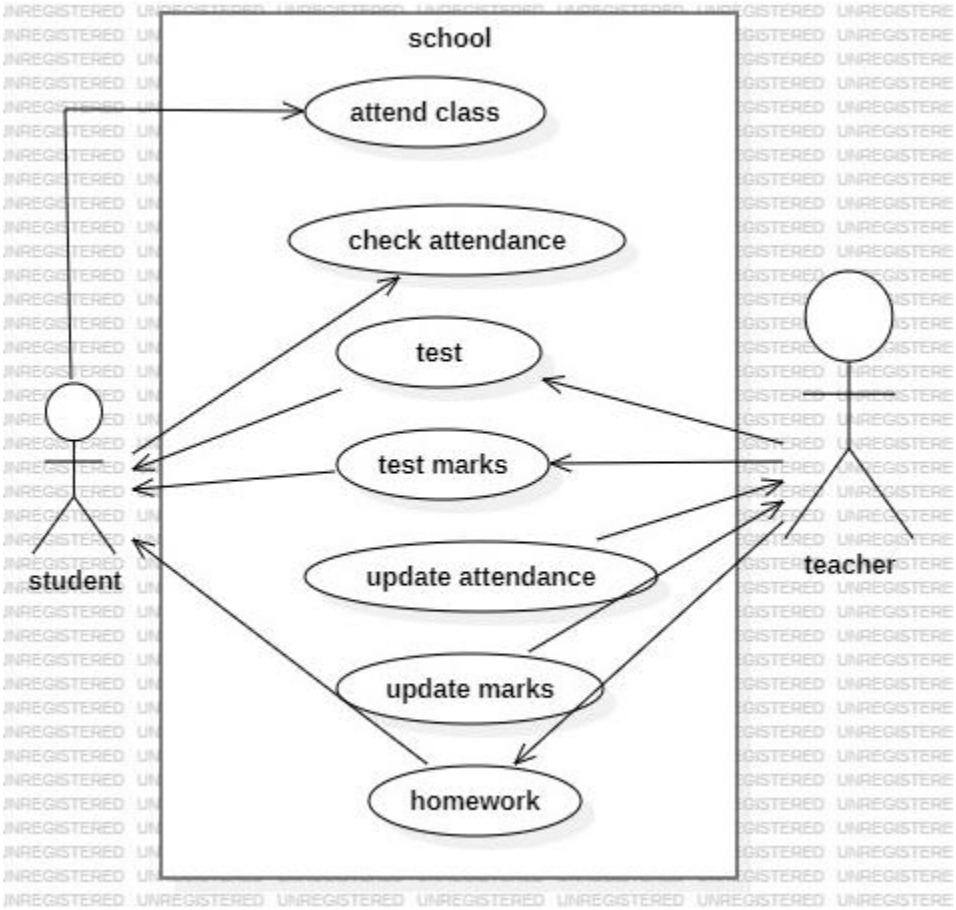


1.a)Use Case Diagram

AIM: school

SOFTWARE USED: STARUML

DIAGARM:

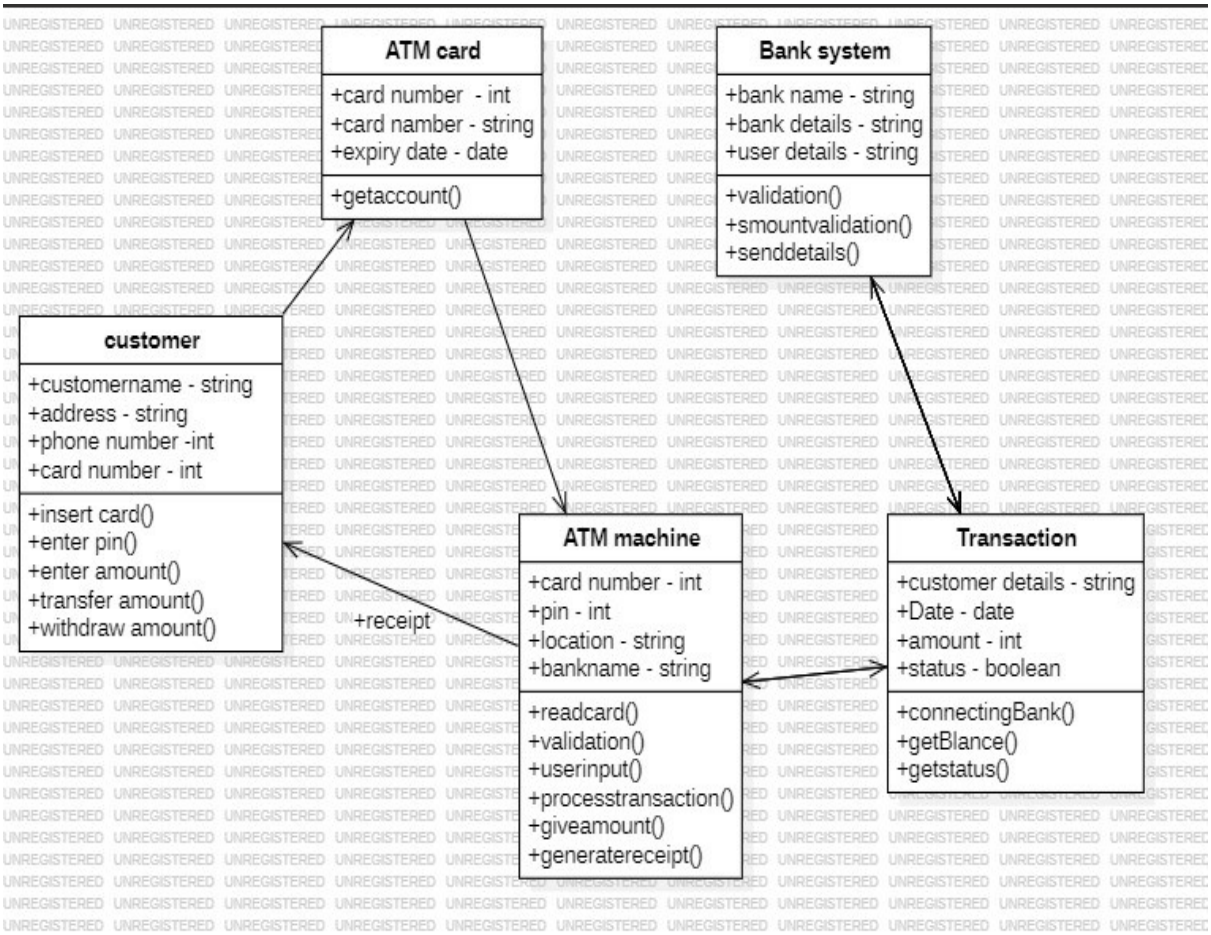


1.b)Class Diagram

AIM: Atm

Software required: starUML

Diagram:

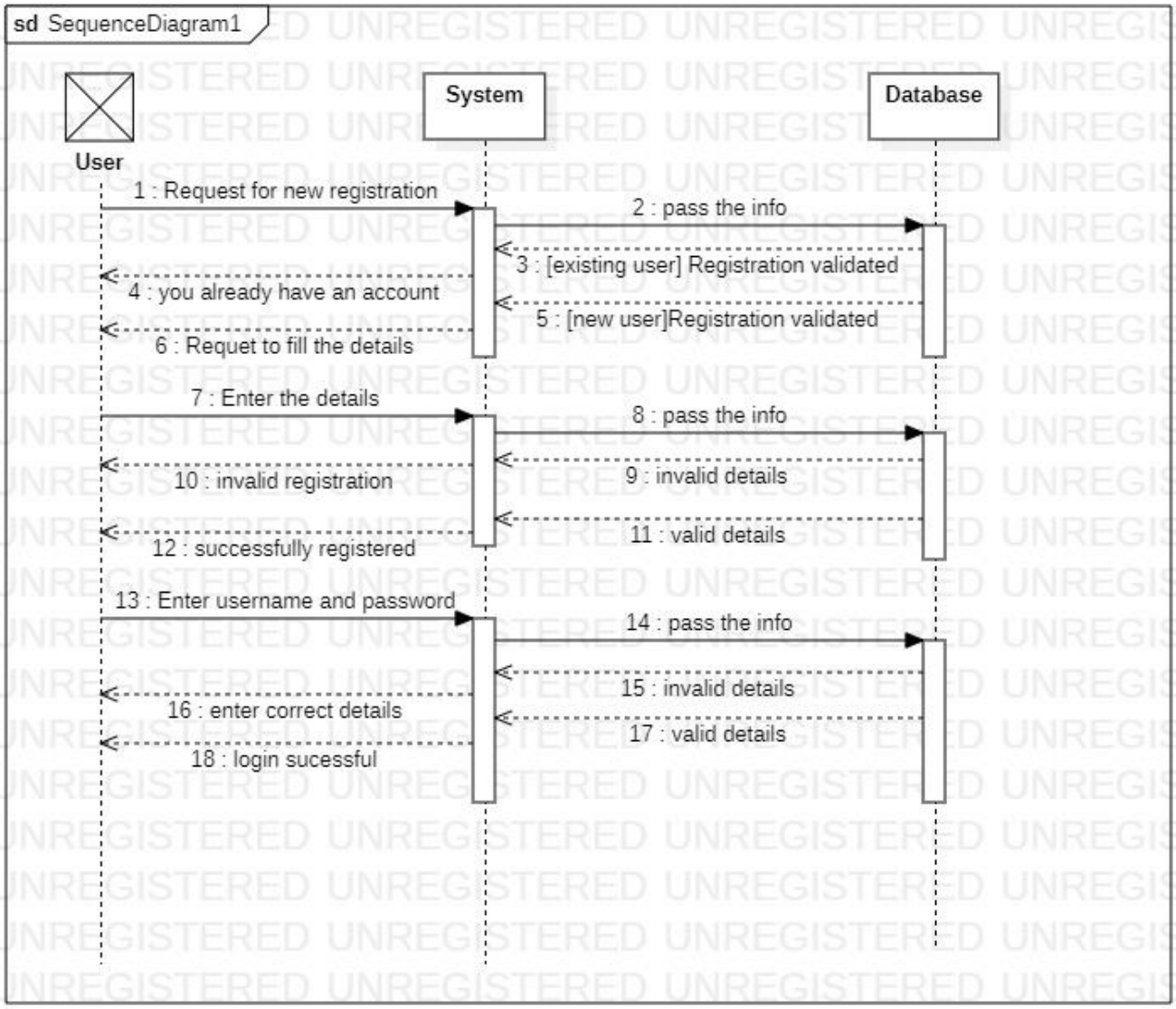


1.c) Sequence Diagram

AIM: Login page

Software required: starUML

Diagram:

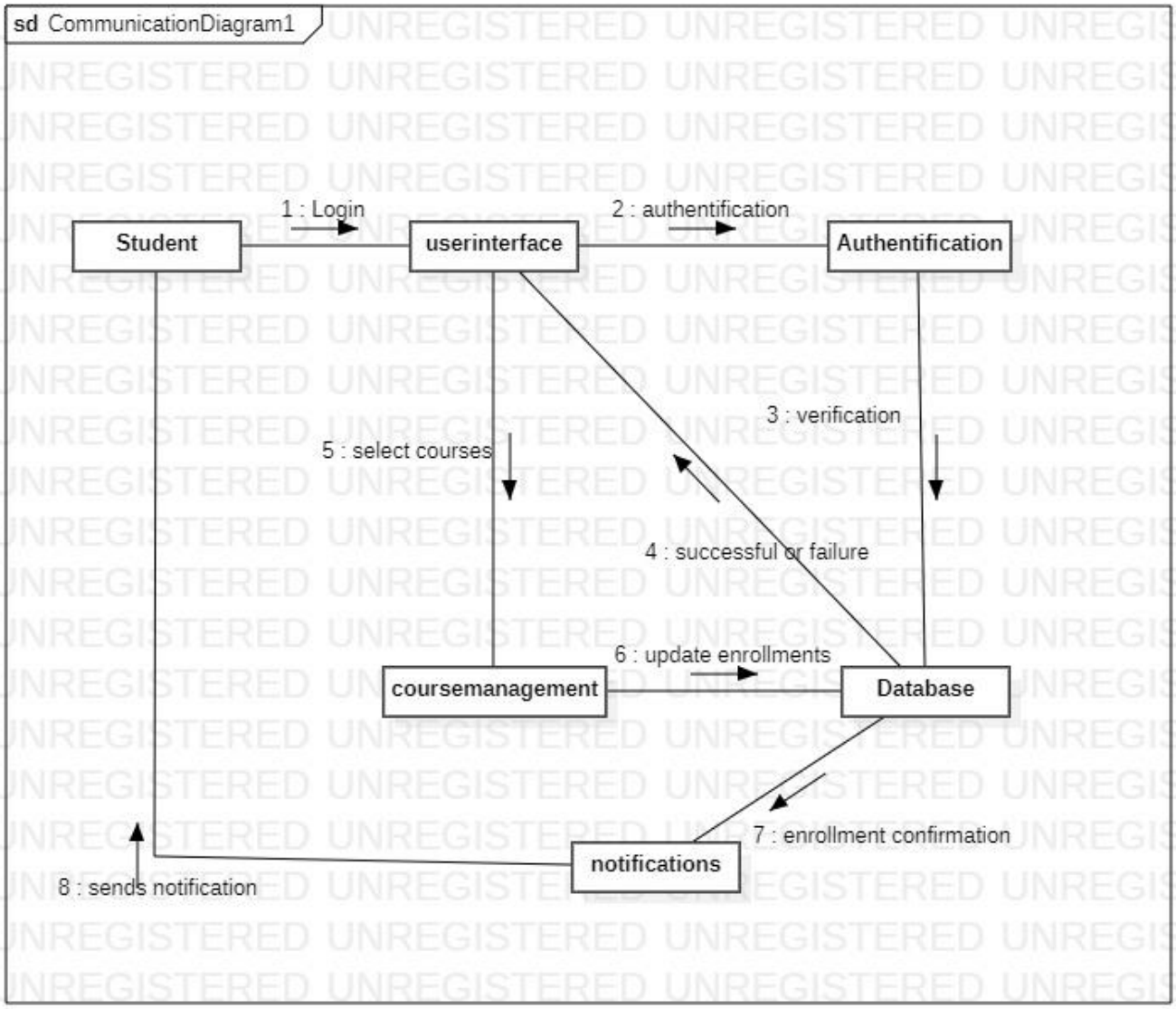


1.d)collaboration Diagram

AIM: student course enrollment

Software required: starUML

Diagram:

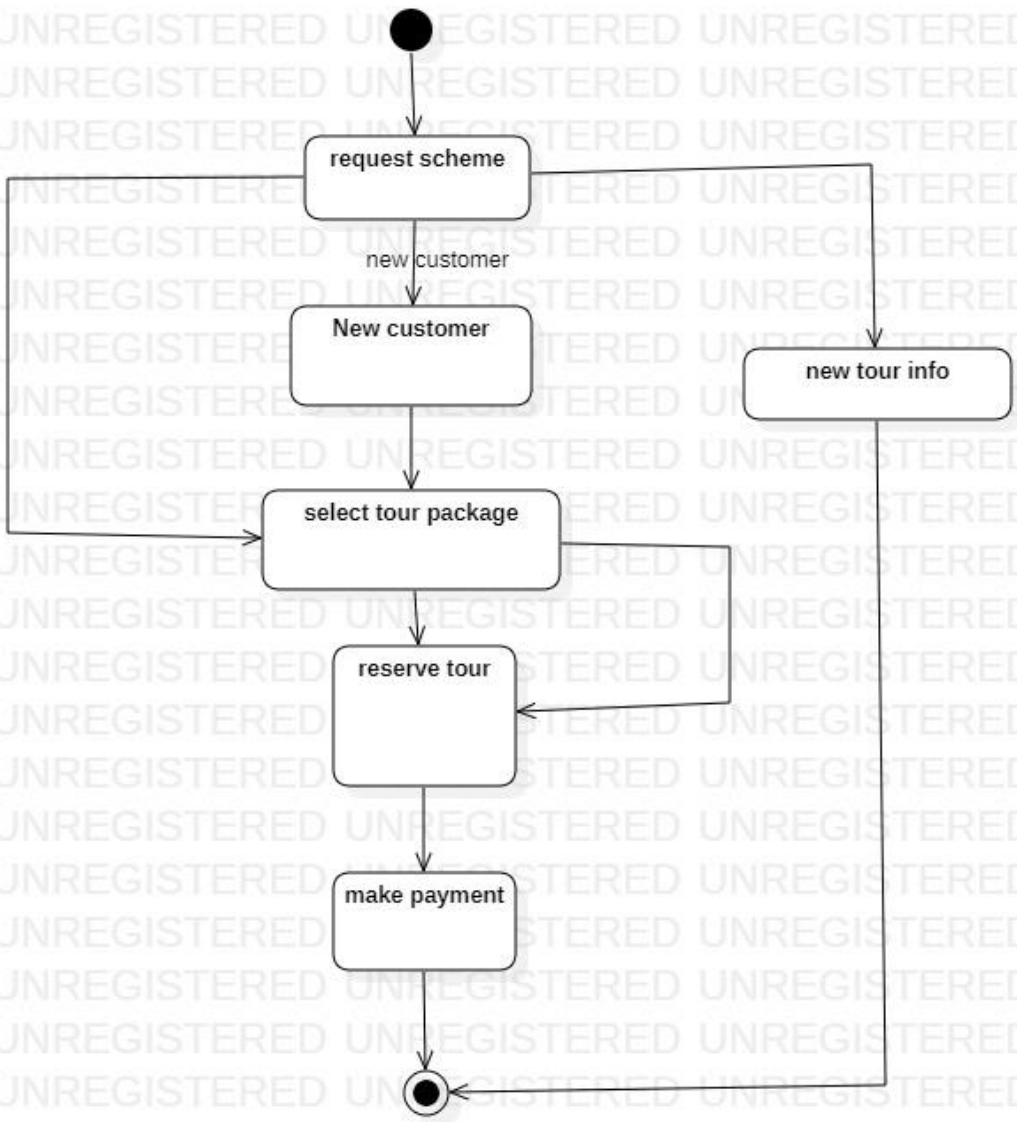


1.e)state Diagram

AIM: tour reservation

Software required: starUML

Diagram:



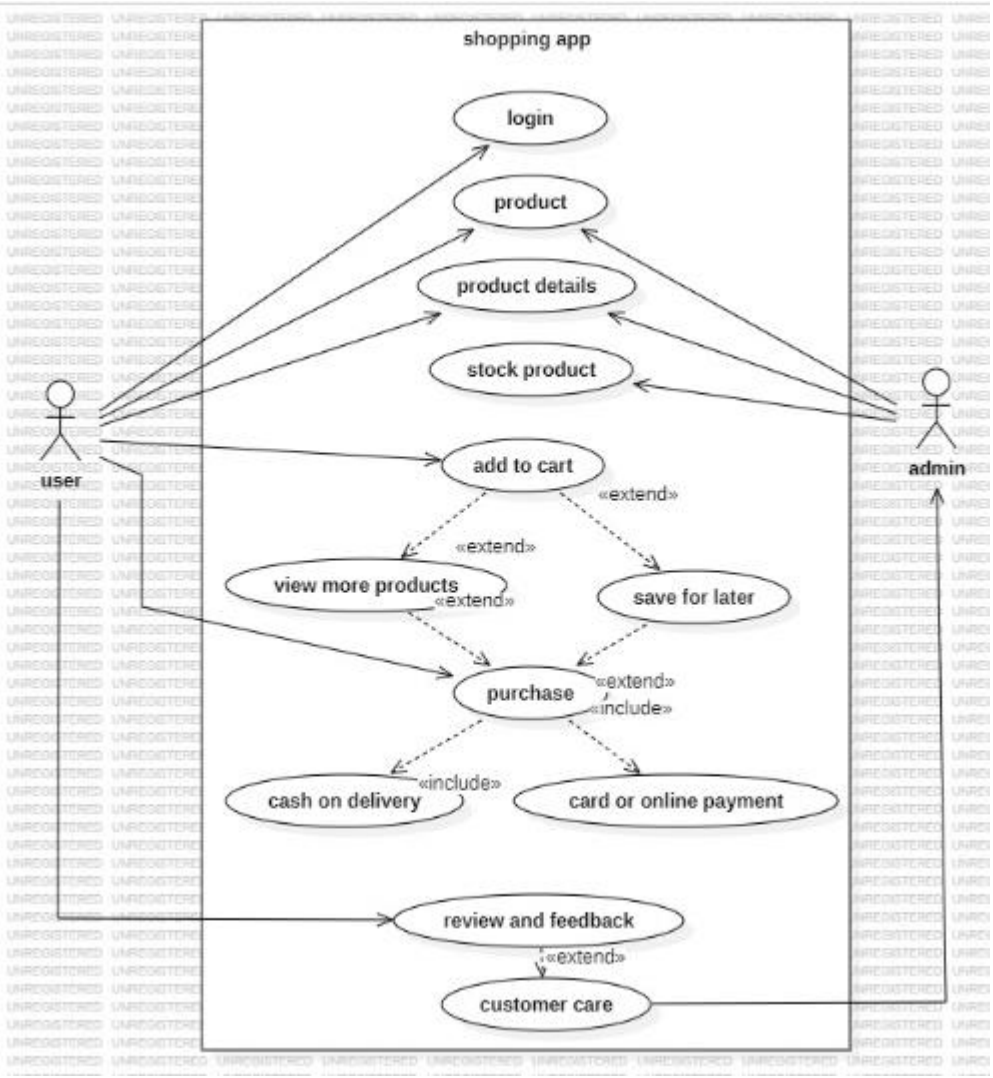
UML DIAGRAM-2

2.a) Use Case Diagram

AIM: shopping

Software required: starUML

Diagram:

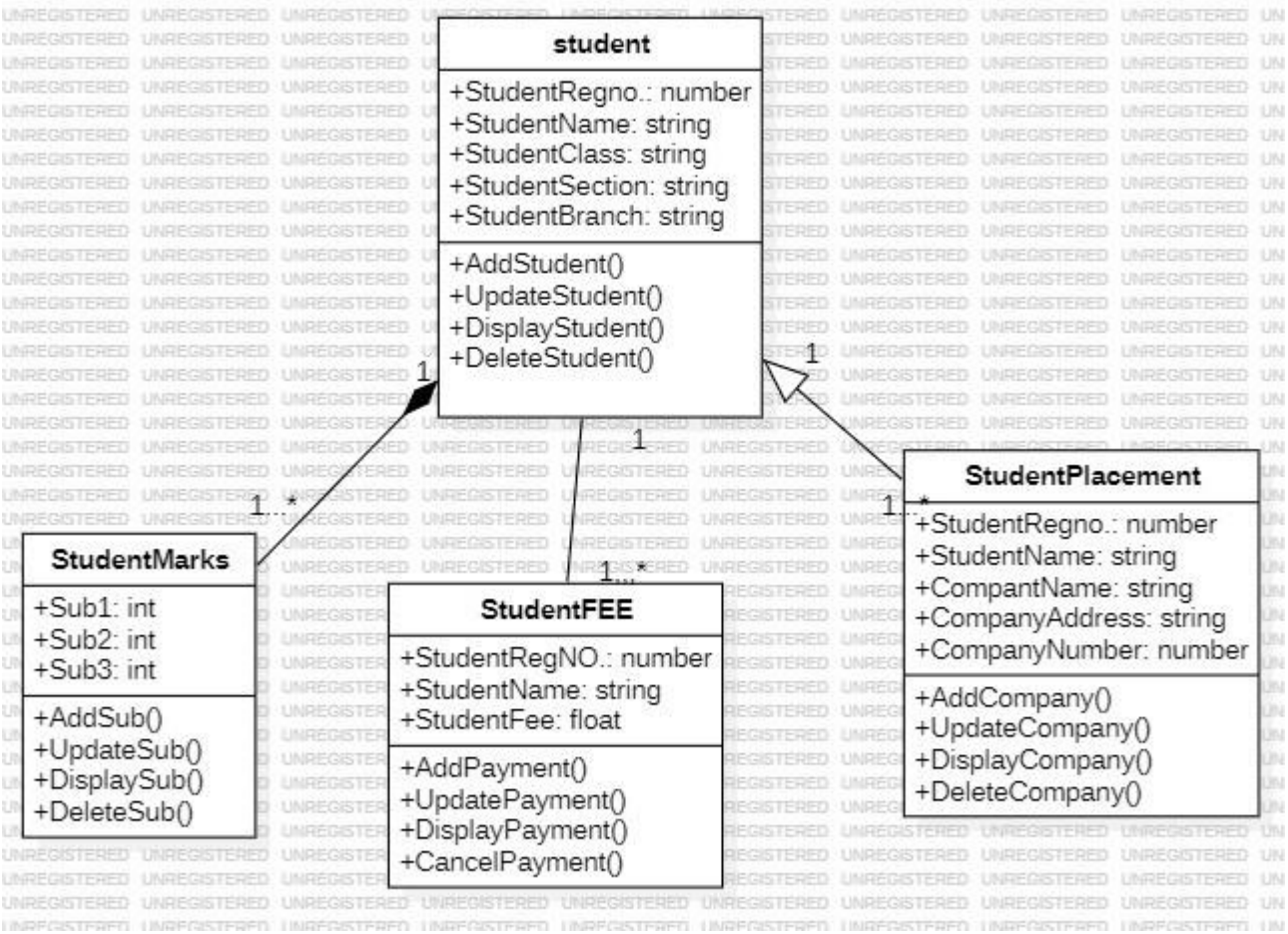


2.b) Class Diagram

AIM: student management

Software required: starUML

Diagram:

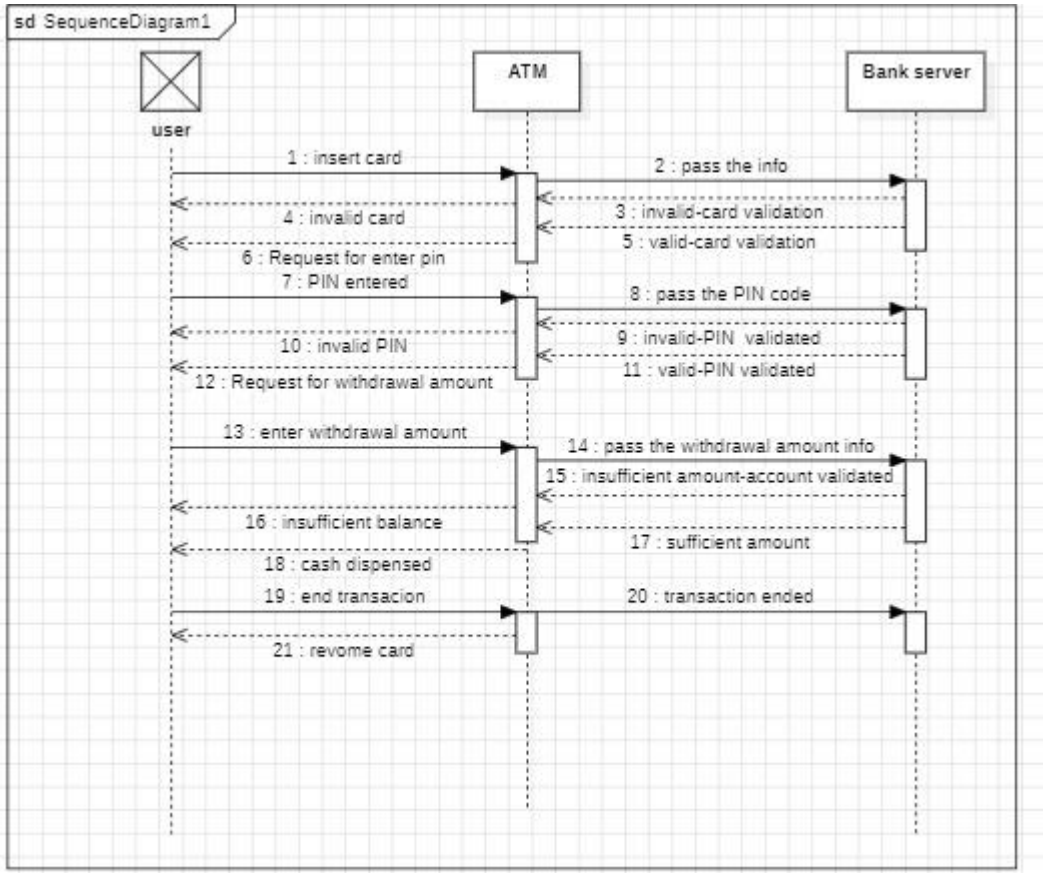


2.c) Sequence Diagram

AIM: ATM

Software required: starUML

Diagram:

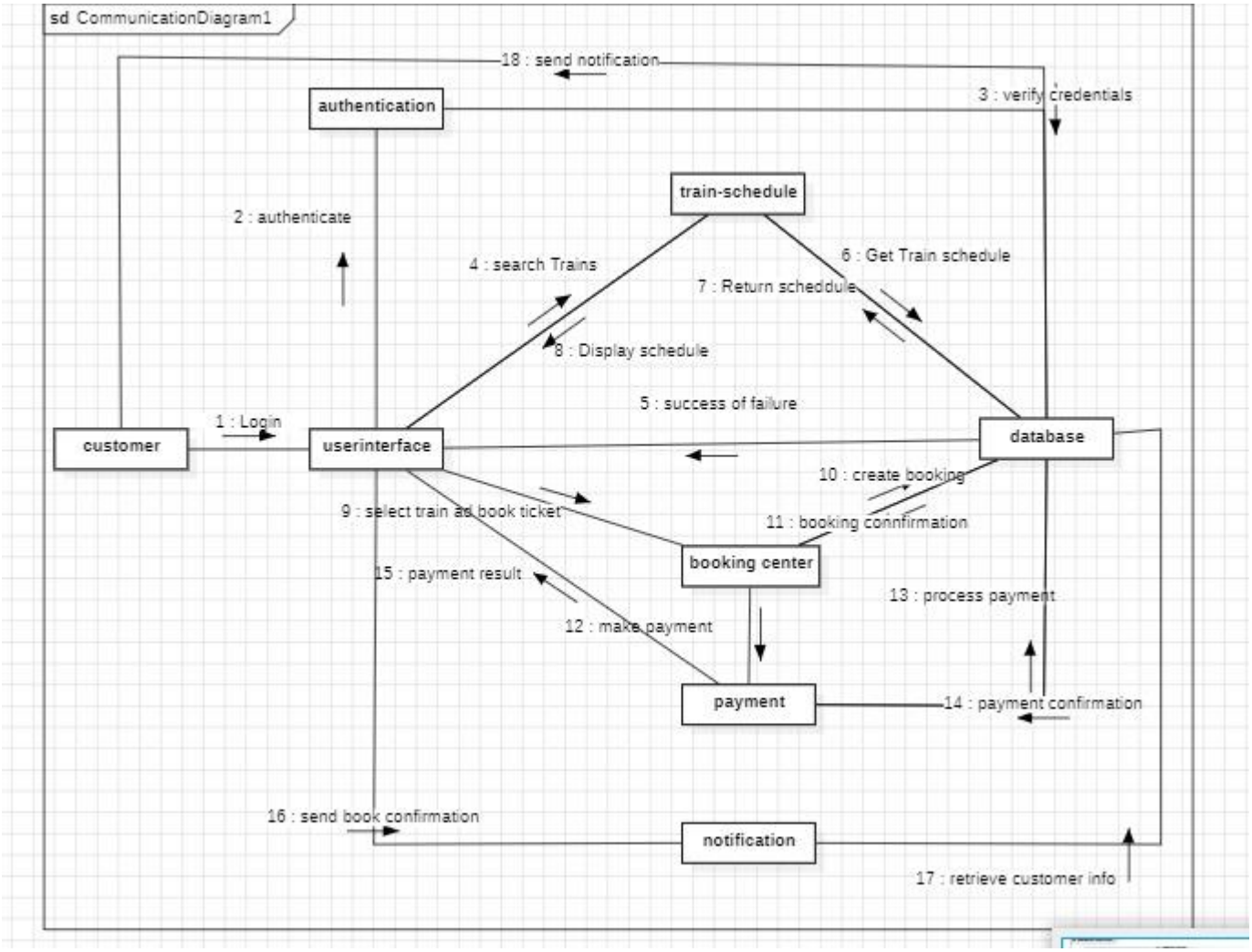


2.D) Collabaration Diagram

AIM: Train ticket booking

Software required: starUML

Diagram:

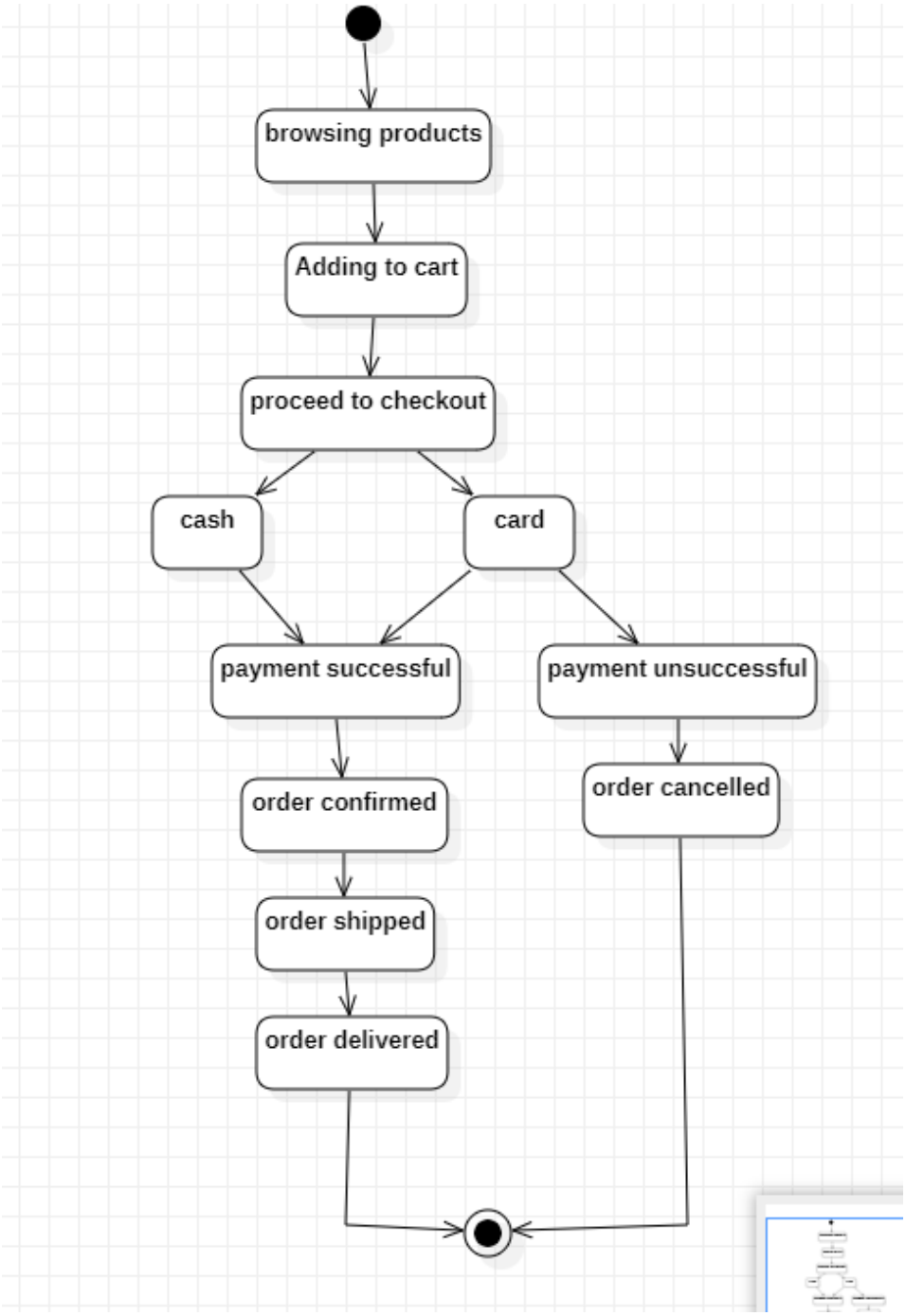


2.e) state Diagram

AIM: shopping

Software required: starUML

Diagram:



BASIC JAVA PROGRAMS

3.a)

AIM: Finding largest number

Software required: notepad

Algorithm:

- Declare and initialize three numbers: a = 40, b = 78, c = 19

- Compare:
- If a >= b and a >= c → a is largest
- Else if b >= a and b >= c → b is largest
- Else → c is largest

Code:

```
public class LargestNumber{
public static void main(String[] args)
 { int a=40, b=78, c=19;
 if(a>=b && a>=c)
 System.out.println(a+" is the largest Number");
 else if (b>=a && b>=c)
 System.out.println(b+" is the largest Number");
 else
 System.out.println(c+" is the largest number");
} }
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac LargestNumber.java

C:\Users\nishi\Desktop>java LargestNumber
78 is the largest Number
```

3b)

AIM: NUMBERS DIVISIBLE BY 5

Software required: notepad

Algorithm:

- Initialize an integer n = 464565625

- Check:
- If `n % 5 == 0` → print `"Yes"`
- Else → print `"No"`

Code:

```
public class GFG {

 public static void main(String[] args)
   { int n = 464565625;
      if (n % 5 == 0)
         System.out.println("Yes");
      else
         System.out.println("No");
   }
}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac GFG.java

C:\Users\nishi\Desktop>java GFG
Yes
```

3c)

AIM: EVEN NUMBERS FROM 1 TO 10

Software required: notepad

Algorithm:

- Initialize `number = 10`
- Loop from `i = 1` to `i <= number`:
- If `i % 2 == 0` (i is even), print `i`

code:

```
public class even{
public static void main(String args[]){
int number=10;
System.out.print("List of even numbers from 1 to "+number+": ");
 for (int i=1; i<=number; i++) {
 if (i%2==0)
{ System.out.print(i + " "); }
 }
 }
}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac even.java

C:\Users\nishi\Desktop>java even
List of even numbers from 1 to 10: 2 4 6 8 10
C:\Users\nishi\Desktop>
```

3d)

AIM: REVERSING A NUMBER

Software required: notepad

Algorithm:

- **nitialize** `number = 995`, `reverse = 0`
- **Repeat while** `number != 0`
- Get the last digit: `remainder = number % 10`
- Append it to `reverse`: `reverse = reverse * 10 + remainder`
- Remove last digit from number: `number = number / 10`
- **Print** the reversed number

Code:

```
public class ReverseNumber
{
public static void main(String[] args)
{
int number = 995, reverse = 0;
while(number != 0)
```

```
{
int remainder = number % 10;
reverse = reverse * 10 + remainder;
number = number/10;
}
System.out.println("The reverse of the given number is: " + reverse);  }
}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac ReverseNumber.java

C:\Users\nishi\Desktop>java ReverseNumber
The reverse of the given number is: 599
```

3e)

AIM: Palindrome

Software required: notepad

Algorithm:

- Initialize a number `n` (e.g., `454`)
- Store the original number in `temp`
- Initialize `sum = 0`
- Repeat while `n > 0`:

- Get the last digit: `r = n % 10`
- Add it to the reversed number: `sum = (sum * 10) + r`
- Remove the last digit: `n = n / 10`
- Compare `temp` with `sum`:
- If equal → it's a palindrome
- Else → not a palindrome

Code:

```
class Palindrome{
public static void main(String args[]){
  int r,sum=0,temp;
  int n=454;//It is the number variable to be checked for palindrome

  temp=n;
  while(n>0){
   r=n%10;  //getting remainder
   sum=(sum*10)+r;
   n=n/10;
  }
  if(temp==sum)
   System.out.println("palindrome number ");
  else
   System.out.println("not palindrome");
}
}
```

OUTPUT:

```
C:\Users\ch.sc.u4cse24031\Desktop>javac Palindrome.java

C:\Users\ch.sc.u4cse24031\Desktop>java Palindrome
palindrome number
```

3f)

AIM: temperature change

Software required: notepad

Algorithm:

- Declare two variables: `Celsius` and `Fahrenheit`
- Assign a value to `Celsius` (in this case, `13`)
- Calculate Fahrenheit using the formula:

Fahrenheit=(Celsius×9/5)+32

- Print the Fahrenheit value

Code:

```
public class temperature
{
  public static void main (String args[])
   { float Fahrenheit, Celsius;
       Celsius= 13;
       Fahrenheit =((Celsius*9)/5)+32;
       System.out.println("Temperature in Fahrenheit is: "+Fahrenheit);
   }}
```

OUTPUT:

```
C:\Users\ch.sc.u4cse24031\Desktop>javac temperature.java

C:\Users\ch.sc.u4cse24031\Desktop>java temperature
Temperature in Fahrenheit is: 55.4
```

3g)

AIM: factorial

Software required: notepad

Algorithm:

- Input a number `num` from the user
- Initialize `sum = 0` and `temp = num`
- Find number of digits in `num` and store in `digits`
- Repeat while `temp ≠ 0`:
- Get last digit: `digit = temp % 10`
- Add `digit^digits` to `sum`
- Remove last digit: `temp = temp / 10`
- Compare `sum` with `num`
- If equal, print "Armstrong Number", else print "Not Armstrong"

Code:

```
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(), sum = 0, temp = num;
        int digits = String.valueOf(num).length();
```

```
        while (temp != 0) {
            sum += Math.pow(temp % 10, digits);
            temp /= 10;
        }
        System.out.println(num == sum ? "Armstrong Number" : "Not Armstrong");
        sc.close();
    }
}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Factorial.java

C:\Users\nishi\Desktop>java Factorial
Enter a number: 10
Factorial: 3628800
```

3h)

AIM: Swapping numbers

Software required: notepad

Algorithm:

- **Input** two numbers a and b from the user
- Add both numbers and store in a → a = a + b
- Subtract new b from new a and store in b → b = a - b
- Subtract new b from new a and store in a → a = a - b
- **Print** the swapped values of a and b

Code:

```
import java.util.Scanner;
public class Swap {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter two numbers: ");
        int a = sc.nextInt(), b = sc.nextInt();
        a = a + b;
        b = a - b;
        a = a - b;
        System.out.println("After Swap: a = " + a + ", b = " + b);
        sc.close();
    }
}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Swap.java

C:\Users\nishi\Desktop>java Swap
Enter two numbers: 2 5
After Swap: a = 5, b = 2

C:\Users\nishi\Desktop>
```

3i)

AIM: Leap year

Software required: notepad

Algorithm:

- Read a year from the user
- Check if the year is divisible by 4 and not divisible by 100
- OR divisible by 400
- If condition is true, it's a leap year
- Else, it's not a leap year
- Display the result

Code:

```
import java.util.Scanner;
public class LeapYear {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a year: ");
        int year = sc.nextInt();
        boolean isLeap = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
        System.out.println(isLeap ? "Leap Year" : "Not a Leap Year");
        sc.close();
    }
}
```

Output:

```
C:\Users\nishi\Desktop>javac LeapYear.java

C:\Users\nishi\Desktop>java LeapYear
Enter a year: 2028
Leap Year

C:\Users\nishi\Desktop>
```

3j)

AIM: Armstrong numbers

Software required: notepad

Algorithm:

- Read a number from the user
- Count number of digits in the number
- Initialize `sum = 0` and `temp = number`
- Repeat while `temp != 0`
- Get last digit using `temp % 10`
- Raise it to the power of number of digits and add to `sum`
- Remove last digit using `temp / 10`
- Compare `sum` with original number
- If equal, it's an Armstrong number
- Else, it's not Display the result

Code:

```
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(), sum = 0, temp = num;
        int digits = String.valueOf(num).length();
        while (temp != 0) {
            sum += Math.pow(temp % 10, digits);
```

```
        temp /= 10;
    }
    System.out.println(num == sum ? "Armstrong Number" : "Not Armstrong");
    sc.close();
  }
}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Armstrong.java

C:\Users\nishi\Desktop>java Armstrong
Enter a number: 20
Not Armstrong

C:\Users\nishi\Desktop>
```

INHERITANCE

SINGLE INHERITANCE PROGRAMS

4a)

AIM: doctor details

Software required: notepad

Algoritm:

- Create `Hospital` class with a method to show hospital info
- Create `Doctor` class that extends `Hospital`
- Add a method in `Doctor` class to show doctor info
- In `main()` method:

a. Create object of `Doctor`
b. Call hospital info method
c. Call doctor info method

Code:

```
class Hospital {
void hospitalInfo() {
System.out.println("Welcome to City Hospital.");}}
class Doctor extends Hospital {
void doctorInfo() {
System.out.println("Doctor: Dr. Smith, Specialization: Cardiology.");}
public static void main(String[] args) {
Doctor d = new Doctor();
d.hospitalInfo();
d.doctorInfo();  }}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Doctor.java

C:\Users\nishi\Desktop>java Doctor
Welcome to City Hospital.
Doctor: Dr. Smith, Specialization: Cardiology.

C:\Users\nishi\Desktop>
```

4B)

AIM: students marks

Software required: notepad

ALgorithm:

- Create `Student` class with name and roll number
- Create `Marks` class that extends `Student`
- Add marks variable in `Marks` class
- In `main()` method:
  a. Create `Marks` object
  b. Set name, roll number, and marks
  c. Display name, roll number, and marks

Code:

```
class Student {
String name;
 int rollNo
void setStudent(String n, int r) {
name = n;
rollNo = r;  }
void showStudent() {
System.out.println("Name: " + name + ", Roll No: " + rollNo);}}
class Marks extends Student {
int marks;
void setMarks(int m) {
marks = m; }
 void showMarks() {
showStudent();
System.out.println("Marks: " + marks);    }
 public static void main(String[] args) {
Marks s = new Marks();
s.setStudent("John", 5);
s.setMarks(85);
s.showMarks(); }}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Marks.java

C:\Users\nishi\Desktop>java Marks
Name: John, Roll No: 5
Marks: 85

C:\Users\nishi\Desktop>
```

EXPERIMENT-5

MULTILEVEL INHERITANCE PROGRAMS

4B)

AIM: Login check

Software required: notepad

ALGORITHM:

Define class SystemBase
- Create method startSystem()
  Print "System is starting..."

: Define class LoginSystem (inherits SystemBase)
- Create method login(username, password)
  If username is "admin" and password is "1234"
    Print "Login successful."
  Else
    Print "Login failed."

Define class AdminDashboard (inherits LoginSystem)
- Create method accessDashboard()
  Print "Accessing admin dashboard..."

: In main() method of AdminDashboard
- Create object 'admin' of AdminDashboard
- Call startSystem() using 'admin' object
- Call login("admin", "1234")
- Call accessDashboard()

Code:

```
class SystemBase {
void startSystem() {
System.out.println("System is starting...");}}
class LoginSystem extends SystemBase {
void login(String username, String password) {
if (username.equals("admin") && password.equals("1234")) {
System.out.println("Login successful.");}
else {
System.out.println("Login failed."); }}}
class AdminDashboard extends LoginSystem {
void accessDashboard() {
System.out.println("Accessing admin dashboard...");}
public static void main(String[] args) {
AdminDashboard admin = new AdminDashboard();
admin.startSystem();
admin.login("admin", "1234");
admin.accessDashboard();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac AdminDashboard.java

C:\Users\nishi\Desktop>java AdminDashboard
System is starting...
Login successful.
Accessing admin dashboard...

C:\Users\nishi\Desktop>
```

5b)

AIM: students Details

Software required: notepad

Algorithm

: Define class 'University'
Create method uniInfo()
- Print "Amrita Vishwa Vidyapeetham University"

: Define class 'Department' which extends 'University'
Create method deptInfo()
- Print "Department of Computer Science"

: Define class 'Student' which extends 'Department'
Create method studentInfo()
- Print "Student: Nishitha Penagaluru"

: In the main() method of 'Student' class
Create object 's' of class Student
Call s.uniInfo() // Inherited from University
Call s.deptInfo() // Inherited from Department
Call s.studentInfo() // Own method

Code:
```
class University {
void uniInfo() {
System.out.println("Amrita Vishwa Vidyapeetham University");}}
class Department extends University {
void deptInfo() {
System.out.println("Department of Computer Science");}}
class Student extends Department {
void studentInfo() {
System.out.println("Student: Nishitha Penagaluru");}
 public static void main(String[] args) {
Student s = new Student();
s.uniInfo();     // University
s.deptInfo();    // Department
 s.studentInfo(); // Student }}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Student.java

C:\Users\nishi\Desktop>java Student
Amrita Vishwa Vidyapeetham University
Department of Computer Science
Student: Nishitha Penagaluru

C:\Users\nishi\Desktop>
```

EXPERIMENT-6

HIERARCHICAL INHERITANCE PROGRAMS

6a)

AIM: Course

Software required: notepad

ALgorithm:

Define a base class 'Course'
- Method: syllabus()
  → Prints "Common syllabus structure."

Define class 'OnlineCourse' extending 'Course'
- Method: platform()

Prints "Accessed via Zoom or Teams."

Define class 'OfflineCourse' extending 'Course'
- Method: location()
  Prints "Conducted in physical classrooms."

: In the main() method of OfflineCourse:

a. Create object 'online' of OnlineCourse
   - Call online.syllabus()
   - Call online.platform()

b. Create object 'offline' of OfflineCourse
   - Call offline.syllabus()
   - Call offline.location()

Code:

```
class Course {
void syllabus() {
System.out.println("Common syllabus structure.");}}
class OnlineCourse extends Course {
void platform() {
System.out.println("Accessed via Zoom or Teams.");}}
 class OfflineCourse extends Course {
void location() {
System.out.println("Conducted in physical classrooms.");}
  public static void main(String[] args) {
OnlineCourse online = new OnlineCourse();
online.syllabus();
online.platform();

OfflineCourse offline = new OfflineCourse();
offline.syllabus();
offline.location(); }}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac OfflineCourse.java

C:\Users\nishi\Desktop>java OfflineCourse
Common syllabus structure.
Accessed via Zoom or Teams.
Common syllabus structure.
Conducted in physical classrooms.

C:\Users\nishi\Desktop>
```

6b)

AIM:

Software required: notepad

Algorithm:

: Define a parent class 'Payment'
- Method: makePayment()
  → Print "Payment initiated..."

: Define subclass 'CreditCard' extending Payment
- Method: swipe()
  → Print "Payment via credit card."

: Define subclass 'UPI' extending Payment
- Method: scanQR()
  → Print "Payment via UPI QR code."

: In the main() method (inside UPI class):
a. Create an object 'cc' of CreditCard
   - Call cc.makePayment()
   - Call cc.swipe()

b. Create an object 'upi' of UPI
   - Call upi.makePayment()
   - Call upi.scanQR()

Code:

```
class Payment {
void makePayment() {
System.out.println("Payment initiated..."); }}
class CreditCard extends Payment {
void swipe() {
System.out.println("Payment via credit card."); }}
 class UPI extends Payment {
void scanQR() {
System.out.println("Payment via UPI QR code.");}
public static void main(String[] args) {
CreditCard cc = new CreditCard();
cc.makePayment();
cc.swipe();

UPI upi = new UPI();
upi.makePayment();
upi.scanQR();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac UPI.java

C:\Users\nishi\Desktop>java UPI
Payment initiated...
Payment via credit card.
Payment initiated...
Payment via UPI QR code.

C:\Users\nishi\Desktop>
```

EXPERIMENT-7

HYBRID INHERITANCE PROGRAMS

7a)

AIM:

Software required: notepad

ALGORITHM:

: Create an interface 'Sports'
- Declare method play()

: Create a base class 'Person'
- Method walk(): Prints "Person is walking."

: Create a class 'Student' that:
- Inherits from 'Person'
- Implements 'Sports'
- Defines method play(): Print "Student is playing cricket."
- Adds method study(): Print "Student is studying."

: In main():

a. Create an object 's' of class Student
b. Call s.walk()
c. Call s.play()
d. Call s.study()

Code:

```
interface Sports {
void play();}
class Person {
void walk() {
System.out.println("Person is walking.");}}
class Student extends Person implements Sports {
public void play() {
System.out.println("Student is playing cricket.");}
void study() {
System.out.println("Student is studying.");}
public static void main(String[] args) {
Student s = new Student();
s.walk();
s.play();
s.study();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Student.java

C:\Users\nishi\Desktop>java Student
Person is walking.
Student is playing cricket.
Student is studying.

C:\Users\nishi\Desktop>
```

7b)

AIM: Appliances

Software required: notepad

Algorithm:

: Create interface 'Scanner'
- Abstract method: scan()

: Create base class 'Machine'
- Method: start()
  → Print "Machine starting..."

: Create class 'Printer'
- Inherit from 'Machine'
- Implement 'Scanner'
- Define scan() → Print "Scanning document."
- Define print() → Print "Printing document."

: In main():
- Create object p of Printer
- Call p.start()
- Call p.scan()
- Call p.print()

Code:

```
interface Scanner {
void scan();}
class Machine {
void start() {
System.out.println("Machine starting...");}}
class Printer extends Machine implements Scanner {
public void scan() {
System.out.println("Scanning document.");}
void print() {
System.out.println("Printing document.");}
public static void main(String[] args) {
Printer p = new Printer();
p.start();
```

```
 p.scan();
p.print();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Printer.java

C:\Users\nishi\Desktop>java Printer
Machine starting...
Scanning document.
Printing document.
```

POLYMORPHISM

EXPERIMENT-8

CONSTRUCTOR PROGRAMS

7a)

AIM: weather

Software required: notepad

Algorithm:

: Define class 'Weather' with:
- Instance variables: city (String), temperature (int)

: Define a default constructor
- Set city = "Unknown", temperature = 25

Step 4: Define a parameterized constructor
- Accept city and temperature as parameters
- Set the instance variables

: Create a method 'displayWeather()' to print weather details

: In main():
- Create object w1 using default constructor
- Create object w2 using parameterized constructor ("Chennai", 35)
- Call displayWeather() for both objects

Code:

```
class Weather {
 String city;
int temperature;
Weather() {
city = "Unknown";
temperature = 25;}
Weather(String c, int t) {
city = c;
temperature = t;}
 void displayWeather() {
System.out.println("Weather in " + city + ": " + temperature + "°C");}
public static void main(String[] args) {
Weather w1 = new Weather();
Weather w2 = new Weather("Chennai", 35);
w1.displayWeather();
w2.displayWeather();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Weather.java

C:\Users\nishi\Desktop>java Weather
Weather in Unknown: 25°C
Weather in Chennai: 35°C
```

EXPERIMENT-9

CONSTRUCTOR OVERLOADING PROGRAMS

9a)

AIM: student age

Software required: notepad

Algorithm:

: Define a class named 'Student' with:
- Two instance variables: name (String), age (int)

: Create a default constructor:
- Set name to "sana"
- Set age to 18

Step 4: Create a parameterized constructor:
- Accept name and age as arguments
- Assign them to the instance variables

: Create a method display() to print student details

: In the main() method:
- Create object s1 using default constructor
- Create object s2 using parameterized constructor ("Nishitha", 20)

- Call display() on both objects

Code:

```
class Student {
String name;
int age;
 Student() {
name = "sana";
 age = 18;}
Student(String n, int a) {
name = n;
age = a;}
 void display() {
System.out.println(name + " is " + age + " years old.");  }
public static void main(String[] args) {
Student s1 = new Student();
Student s2 = new Student("Nishitha", 20);
s1.display();
s2.display();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Student.java

C:\Users\nishi\Desktop>java Student
sana is 18 years old.
Nishitha is 20 years old.

C:\Users\nishi\Desktop>
```

EXPERIMENT-10

METHOD OVERLOADING PROGRAMS

10a)

AIM: Book details

Software required: notepad

Algorithm:

: Create a class 'Book'

: Define method 'display' that takes one parameter:
- Print the book title

: Overload the 'display' method to take two parameters:
- Print the book title and author

: In the main method:
- Create object 'b' of class Book
- Call 'display' with one argument (title only)
- Call 'display' with two arguments (title + author)

Code:

```
class Book {
void display(String title) {
System.out.println("Title: " + title);}
void display(String title, String author) {
System.out.println("Title: " + title + ", Author: " + author); }
public static void main(String[] args) {
Book b = new Book();
b.display("Wings of Fire");
b.display("Wings of Fire", "APJ Abdul Kalam"); }}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Book.java

C:\Users\nishi\Desktop>java Book
Title: Wings of Fire
Title: Wings of Fire, Author: APJ Abdul Kalam
```

10b)

AIM: temperature converter

Software required: notepad

Algorithm:

Step 1: Start

Step 2: Create a class named 'Temperature'

Step 3: Define method 'convertToFahrenheit' that:
- Takes a Celsius temperature
- Applies formula: (C × 9/5) + 32
- Returns the result

Step 4: Define another method 'convertToCelsius' that:
- Takes a Fahrenheit temperature
- Applies formula: (F − 32) × 5/9
- Returns the result

Step 5: In the main method:
- Create an object 't' of Temperature
- Call both methods with example values (30, 86)
- Print results

Step 6: End

Code:

```
class Temperature {
double convertToFahrenheit(double celsius) {
return (celsius * 9 / 5) + 32;}
double convertToCelsius(double fahrenheit) {
 return (fahrenheit - 32) * 5 / 9;}
public static void main(String[] args) {
Temperature t = new Temperature();
System.out.println("30°C to °F: " + t.convertToFahrenheit(30));
 System.out.println("86°F to °C: " + t.convertToCelsius(86)); }}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Temperature.java

C:\Users\nishi\Desktop>java Temperature
30°C to °F: 86.0
86°F to °C: 30.0
```

EXPERIMENT-11

METHOD OVERRIDING PROGRAMS

11a)

AIM: Hospital

Software required: notepad

Algorithm:

- Define a base class `Hospital` with a method `service()` that prints `"General hospital service."`
- Create a subclass `EyeHospital` that overrides the `service()` method to print `"Eye checkup service."`
- In the `main()` method:Create a reference of type `Hospital` but point it to an object of `EyeHospital`
- Call the `service()` methodJava uses dynamic method dispatch, so it runs the overridden method in `EyeHospital`, not the one in `Hospital`

Code:

```
class Hospital {
void service() {
System.out.println("General hospital service.");}}
class EyeHospital extends Hospital {
 void service() {
 System.out.println("Eye checkup service.");}
 public static void main(String[] args) {
Hospital h = new EyeHospital();
h.service();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac  Hospital.java

C:\Users\nishi\Desktop>java EyeHospital
Eye checkup service.

C:\Users\nishi\Desktop>
```

11b)

AIM: Appliances

Software required: notepad

Algorithm:

- Define a base class `Appliance` with method `powerOn()`
  Prints "Appliance is on."
- Create a subclass `WashingMachine` that overrides `powerOn()`
  Prints "Washing machine is running."
- In the `main()` method:Declare a reference `a` of type `Appliance`
- Call the `powerOn()` method
- Java determines at runtime which version of the method to execute
  Since object is of `WashingMachine`, it calls the overridden version

Code:

```
class Appliance {

void powerOn() {

System.out.println("Appliance is on.");}}

 class WashingMachine extends Appliance {

void powerOn()

{System.out.println("Washing machine is running.");}

public static void main(String[] args) {

Appliance a = new WashingMachine();

a.powerOn();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Appliance.java

C:\Users\nishi\Desktop>java WashingMachine
Washing machine is running.
```

ABSTRACTION
EXPERIMENT-12
INTERFACE PROGRAMS

12a)

AIM: mediaplayer

Software required: notepad

Algorithm:

- Define an Interface `MediaPlayer` with an abstract method `play()`.
- Create a class `MP3Player` that implements the `MediaPlayer` interface.
- Inside `MP3Player`, override the `play()` method to provide a concrete implementation.
- In the `main()` method:Create an object of `MP3Player` using a `MediaPlayer` interface reference.Call the `play()` method — it will execute the overridden version in `MP3Player`.

Code:

```
interface MediaPlayer {
void play();}
class MP3Player implements MediaPlayer {
public void play() {
System.out.println("Playing MP3 song...");}
public static void main(String[] args) {
MediaPlayer m = new MP3Player();
m.play(); }}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac MP3Player.java

C:\Users\nishi\Desktop>java MP3Player
Playing MP3 song...

C:\Users\nishi\Desktop>
```

12b)

AIM: Appiliance

Software required: notepad

Algorithm:

- Create an interface `Bank` with an abstract method `rateOfInterest()`.
- Implement the interface in two separate classes:
- `SBI` returns interest as `7.5f`
- `ICICI` returns interest as `8.0f`
- In the `main()` method:Create two interface references `b1` and `b2`, assigned to `SBI` and `ICICI` objects respectively.Call the overridden `rateOfInterest()` method via these references.Print the interest rates.

Code:

```
interface Bank {
float rateOfInterest();}
class SBI implements Bank {
public float rateOfInterest() {
return 7.5f;}}
class ICICI implements Bank {
public float rateOfInterest() {
return 8.0f;}
 public static void main(String[] args) {
Bank b1 = new SBI();
Bank b2 = new ICICI();
System.out.println(b1.rateOfInterest());
System.out.println(b2.rateOfInterest());}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac ICICI.java

C:\Users\nishi\Desktop>java ICICI
7.5
8.0
```

12c)

AIM: payment

Software required: notepad

Algorithm:

- Define an interface `PaymentMethod` with an abstract method `pay()`.
- Create class `PayPal` that implements `PaymentMethod` and overrides `pay()` to print "Paying via PayPal."
- Create class `UPI` that also implements `PaymentMethod` and overrides `pay()` to print "Paying via UPI."
- Inside `main method` Declare a reference of type `PaymentMethod`.Assign it with objects of `PayPal` and `UPI` classes.

Code:

```
interface PaymentMethod {
void pay();}
class PayPal implements PaymentMethod {
public void pay() {
System.out.println("Paying via PayPal.");}}
class UPI implements PaymentMethod {
public void pay() {
System.out.println("Paying via UPI.");}
 public static void main(String[] args) {
PaymentMethod p1 = new PayPal();
PaymentMethod p2 = new UPI();
p1.pay();
 p2.pay();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac UPI.java

C:\Users\nishi\Desktop>java UPI
Paying via PayPal.
Paying via UPI.
```

12d)

AIM: taxcaLculator

Software required: notepad

Algorithm:

- Define an interface `TaxCalculator` with a method `calculateTax(double income)`.
- Create a class `IncomeTax` that implements `TaxCalculator`.
- Override the method `calculateTax()` to return 10% of the income.
- In the `main()` method:Create a reference of type `TaxCalculator` and assign it an `IncomeTax` object.Call `calculateTax(50000)` and print the result.

Code:

```
interface TaxCalculator {
```

```
double calculateTax(double income);}
class IncomeTax implements TaxCalculator {
public double calculateTax(double income) {
return income * 0.1;}
public static void main(String[] args) {
TaxCalculator tc = new IncomeTax();
System.out.println("Tax: " + tc.calculateTax(50000)); }}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac IncomeTax.java

C:\Users\nishi\Desktop>java IncomeTax.java
error: can't find main(String[]) method in class: TaxCalculator

C:\Users\nishi\Desktop>java IncomeTax
Tax: 5000.0
```

EXPERIMENT-13

ABSTRACT CLASS PROGRAMS

13a)

AIM: food order

Software required: notepad

Algorithm:

- Create an abstract class `Order` with an abstract method `placeOrder()`.
- Create a subclass `FoodOrder` that extends `Order` and implements the `placeOrder()` method.
- Inside `main()`, create an object of `FoodOrder`.
  Use the object to call `placeOrder()` which prints `"Food order placed successfully."`

Code:

```
abstract class Order {
abstract void placeOrder();}
class FoodOrder extends Order {
void placeOrder() {
System.out.println("Food order placed successfully.");   }
public static void main(String[] args) {
FoodOrder f = new FoodOrder();
f.placeOrder();   }}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac FoodOrder.java

C:\Users\nishi\Desktop>java FoodOrder
Food order placed successfully.

C:\Users\nishi\Desktop>
```

13b)

AIM: report

Software required: notepad

Algorithm:

- Abstract Class Definition
  Define an abstract class `Report` with an abstract method `generate()`.
- Subclass Implementation
  Create a subclass `PDFReport` that extends `Report` and implements the `generate()` method.
- Main MethodCreate a reference of type `Report`.Instantiate the subclass `PDFReport` and assign it to the reference (`Report r = new PDFReport();`).
- Call the overridden method `generate()` on the object `r`.

Code:

```
abstract class Report {
abstract void generate();}
class PDFReport extends Report {
void generate() {
System.out.println("PDF Report generated.");}
public static void main(String[] args) {
Report r = new PDFReport();
r.generate();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac PDFReport.java

C:\Users\nishi\Desktop>java PDFReport
PDF Report generated.
```

13c)

AIM: transport

Software required: notepad

Algorithm:

- **Create an Abstract Class**
- Define an abstract class called `Transport`.
- Declare an abstract method `move()` with no body**s**
- Define a sub class `Bike` that **extends** the `Transport` class.
- Implement the `move()` method in `Bike` to print "Bike is moving."
- Main Method Execution Create an object of `Bike`.Call the `move()` method using the object.

Code:

```
abstract class Transport {
abstract void move();}
class Bike extends Transport {
void move() {
System.out.println("Bike is moving.");}
public static void main(String[] args) {
Bike b = new Bike();
b.move();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac  Bike.java

C:\Users\nishi\Desktop>java Bike
Bike is moving.

C:\Users\nishi\Desktop>
```

13d)

AIM: account type

Software required: notepad

Algorithm:

- Create an abstract class named `Account`.
- Inside it, define an abstract method `accountType()` without implementation.
- Extend the Abstract Class:
- Create a class `Savings` that extends `Account`.
- Provide implementation for the abstract method `accountType()` in `Savings`.
- Object Creation and Method Call:
- Inside the `main` method, create an object of the `Savings` class.
- Call `accountType()` method using that object, which prints "Savings Account".

Code:

```
abstract class Account {
abstract void accountType();}
class Savings extends Account {
void accountType() {
System.out.println("Savings Account"); }
public static void main(String[] args) {
Savings s = new Savings();
 s.accountType(); }}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac  Savings.java

C:\Users\nishi\Desktop>java Savings.
Error: Could not find or load main class Savings.
Caused by: java.lang.ClassNotFoundException: Savings.

C:\Users\nishi\Desktop>java Savings
Savings Account
```

ENCAPSULATION
EXPERIMENT-14
ENCAPSULATION PROGRAMS

14a)

AIM: patient details

Software required: notepad

Algorithm:

- Define a class Patient with two private variables: name, age.
- Create a method setDetails() that assigns values to name and age.
- Create another method showDetails() to print those values.
- In main() Create a Patient object.Use setDetails() to set values.Use showDetails() to display patient information.

Code:

```
class Patient {
private String name;
private int age;
public void setDetails(String n, int a) {
 name = n;
age = a;}
public void showDetails() {
System.out.println("Patient: " + name + ", Age: " + age);}
public static void main(String[] args) {
Patient p = new Patient();
p.setDetails("Nishitha", 25);
p.showDetails();}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Patient.java

C:\Users\nishi\Desktop>java Patient
Patient: Nishitha, Age: 25
```

14b)

AIM: student info

Software required: notepad

Algorithm:

1. Define class `Student` with private variables `name` and `rollNo`.
2. Create `setStudent()` method to assign values.
3. Create `getName()` and `getRollNo()` to retrieve values.
4. In `main()` method:Create a `Student` object.Use `setStudent()` to assign data.Use `getName()` and `getRollNo()` to access data and print.

Code:

```
class Student {
private String name;
private int rollNo;
public void setStudent(String n, int r) {
name = n;
rollNo = r;}
public String getName() {
 return name;}
 public int getRollNo() {
return rollNo;}
public static void main(String[] args) {
Student s = new Student();
s.setStudent("Nishitha", 101);
System.out.println("Name: " + s.getName());
System.out.println("Roll No: " + s.getRollNo());}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Student.java

C:\Users\nishi\Desktop>java Student
Name: Nishitha
Roll No: 101
```

14c)

AIM: quiz score

Software required: notepad

Algorithm:

- Define a class `Quiz` with a private variable `score`.
- Create method `updateScore(int s)` to:
- Add `s` to score if `s` is non-negative.
- Create `getScore()` to return the current score.
- In the `main()` method:Create an object of `Quiz`.
- Call `updateScore()` with values like 10 and 15.Call `getScore()`

Code:

```
class Quiz {
private int score;
public void updateScore(int s){
 if (s >= 0)
 score += s;}
 public int getScore(){
return score;}
public static void main(String[] args) {
Quiz q = new Quiz();
q.updateScore(10);
q.updateScore(15);
 System.out.println("Final Score: " + q.getScore());}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Quiz.java

C:\Users\nishi\Desktop>java Quiz
Final Score: 25

C:\Users\nishi\Desktop>
```

14d)

AIM: stock

Software required: notepad

Algorithm:

- Define a class `Product` with a private variable `stock`.
- Create method `addStock(int s)`:
- If `s` is greater than 0, add `s` to `stock`.

- Create method `getStock()` to return the current value of `stock`.
- In the `main()` method:Call `addStock(50)` and `addStock(30)`.and print the totals stock

Code:

```
class Product {
private int stock;
public void addStock(int s){
if (s > 0) stock += s;}
 public int getStock() {
return stock;}
public static void main(String[] args) {
Product p = new Product();
p.addStock(50);
p.addStock(30);
System.out.println("Current Stock: " + p.getStock());
}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac Product.java

C:\Users\nishi\Desktop>java Product
Current Stock: 80
```

PACKAGES PROGRAMS

EXPERIMENT-15

User Defined Packages

15a)

AIM:

Software required: notepad

Algorithm:

- Create a utility class with static methods for:
- Maximum of two numbers

- Minimum of two numbers
- Square of a number
- Use `static import` to call methods directly (without class name).
- Also show traditional usage using class name.
- Print all results.

Code:

```
package utilities;
 public class MathUtils {
public static int max(int a, int b) {
 return (a > b) ? a : b; }
public static int min(int a, int b) {
  return (a < b) ? a : b; }
public static int square(int a) {
 return a * a; } }
```

Main code:

```
 import utilities.MathUtils;
 import static utilities.MathUtils.max;
 import static utilities.MathUtils.min;
import static utilities.MathUtils.square;
public class M { public static void main(String[] args) {
 int maxValue = max(5, 8);
int minValue = min(5, 8);
int squareValue = square(4);
 System.out.println("Max value: " + maxValue);
 System.out.println("Min value: " + minValue);
System.out.println("Square value: " + squareValue);
 int maxValueNormalImport = MathUtils.max(23, 48);
System.out.println("Max value (normal import): " + maxValueNormalImport); } }
```

OUTPUT:

```
C:\javaa>javac -d . MathUtils.java

C:\javaa>javac M.java

C:\javaa>java M
Max value: 8
Min value: 5
Square value: 16
Max value (normal import): 48
```

15b)

AIM:

Software required: notepad

Algorithm:

Step 1: Create a package named `mathoperations`.

Step 2: Inside the package, define a class called `Addition`.

Step 3: In the `Addition` class, define a method `add(int n1, int n2)` that returns the sum of `n1` and `n2`.

Step 4: Create a main class `Math` in a different file (outside the package).

Step 5: Import the `Addition` class from the `mathoperations` package.

Step 6: In the `main()` method, create an object of the `Addition` class.

Step 7: Call the `add()` method using the object and store the result.

Step 8: Print the result.

code:

```
package mathoperations;
public class Addition{ public int add(int n1,int n2){
```

```
return n1+n2;
}
}
```

Main Code:

```
import mathoperations.Addition;
public class Math
{
public static void main(String[] args)
{ Addition a=new Addition();
int sum=a.add(2,3);
System.out.println("the sum of the numbers is:"+sum);
}
}
```

OUTPUT:

```
C:\>cd javaa

C:\javaa>javac -d . Addition.java

C:\javaa>javac Math.java

C:\javaa>java Math
the sum of the numbers is:5
```

15c)

AIM:

Software required: notepad

Algorithm:

Step 1: Create the package `company.hr`

- Define a class named `Employee`.
- Declare private fields: `name`, `baseSalary`, and `hoursWorked`.
- Create a constructor to initialize these fields.
- Provide public getter methods: `getName()`, `getBaseSalary()`, and `getHoursWorked()`.

Step 2: Create the package `company.finance`

- Import `company.hr.Employee`.
- Define a class named `Payroll`.
- Define a method `calculateSalary(Employee employee)`:
  - Get base salary and hours worked using getters.
  - Calculate hourly rate: `hourlyRate = baseSalary / 160`.
  - Return total salary: `hourlyRate * hoursWorked`.

Step 3: Create the `Main` class (outside packages or in a suitable one)

- Import both `company.hr.Employee` and `company.finance.Payroll`.
- In the `main()` method:
  - Create an `Employee` object with name, base salary, and hours worked.
  - Create a `Payroll` object.
  - Call `calculateSalary()` using the `Payroll` object and pass the `Employee`.

Code:

```
Package company.hr;
public class Employee {
private String name; private double baseSalary;
 private int hoursWorked; public Employee(String name, double baseSalary, int hoursWorked)
{ this.name = name; this.baseSalary = baseSalary;
this.hoursWorked = hoursWorked; }
public String getName() {
return name; }
 public double getBaseSalary() {
return baseSalary; }
 public int getHoursWorked() {
return hoursWorked; } }
 Package code:
 package company.finance;
 import company.hr.Employee;
public class Payroll {
 public double calculateSalary(Employee employee) {
 double baseSalary = employee.getBaseSalary();
 int hoursWorked = employee.getHoursWorked();
double hourlyRate = baseSalary / 160;
return hourlyRate * hoursWorked; } }
 Main Code: import company.hr.Employee;
import company.finance.Payroll;
public class Main {
public static void main(String[] args) {
Employee employee = new Employee("mahi", 4000.0, 160);
 Payroll payroll = new Payroll();
 double salary = payroll.calculateSalary(employee);
System.out.println("Salary of " + employee.getName() + " is: $" + salary);} }
```

OUTPUT:

```
C:\javaa>javac -d . Employee.java

C:\javaa>javac -d . Payroll.java

C:\javaa>javac Main.java

C:\javaa>java Main
Salary of mahi is: $4000.0
```

15d)

AIM:

Software required: notepad

Algorithm:

Step 1: Create the package `exceptions`

- Define a custom exception class `InvalidAgeException` that extends `Exception`.
- Create a constructor that takes a `String` message and passes it to the `super()` constructor.

Step 2: Create the package `validation`

- Import the `InvalidAgeException` class.
- Define a class `AgeValidator`.
- Create a method `validateAge(int age)`:
  - If `age < 18`, throw `InvalidAgeException` with message `"Age must be 18 or above."`.
  - Else, print `"Age is valid."`.

Step 3: Create the `Main` class (in default package or another one)

- Import `AgeValidator` and `InvalidAgeException`.
- In the `main()` method:
  - Create an object of `AgeValidator`.
  - Use a `try-catch` block to:
    - Call `validateAge(15)`, which throws an exception. Catch it and print the error message.

- Call `validateAge(20)`, which is valid. Print confirmation message.

Code:

package code:

```
package exceptions;
public class InvalidAgeException extends Exception {
public InvalidAgeException(String message) {
super(message);
}
}
```

Package code:

```
package validation;
import exceptions.InvalidAgeException;
public class AgeValidator {
public void validateAge(int age) throws InvalidAgeException {
if (age < 18) {
throw new InvalidAgeException("Age must be 18 or above.");
}
System.out.println("Age is valid.");
}
}
```

Main code:

```
import validation.AgeValidator;
import exceptions.InvalidAgeException;
public class Main {
public static void main(String[] args) {
AgeValidator validator = new AgeValidator();
try {
```

validator.validateAge(15); // This will throw an exception

} catch (InvalidAgeException e) {

System.out.println("Error: " + e.getMessage());

} try { validator.validateAge(20);

} catch (InvalidAgeException e) { System.out.println("Error: " + e.getMessage()); } }

OUTPUT:

```
C:\javaa>javac -d . InvalidAgeException.java

C:\javaa>javac -d . AgeValidator.java

C:\javaa>javac Main.java

C:\javaa>java Main
Error: Age must be 18 or above.
Age is valid.
```

EXPERIMENT-16

EXCEPTION HANDLING PROGRAMS

16a)

AIM: divided by zero

Software required: notepad

Algorithm:

- Declare and initialize two integers: `a = 10`, `b = 0`
- Attempt to divide `a` by `b` and store the result in `result`
- Since dividing by zero is not allowed, an `ArithmeticException` is thrown
- Catch the exception and print: `"Cannot divide by zero."`

Code:

```
public class DivideExample {
public static void main(String[] args) {
try {
int a = 10, b = 0;
int result = a / b;
System.out.println("Result: " + result);}
catch (ArithmeticException e){
System.out.println("Cannot divide by zero.");}}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac DivideExample.java

C:\Users\nishi\Desktop>java DivideExample
Cannot divide by zero.
```

16b)

AIM:

Software required: notepad

Algorithm:

- Define a **custom exception class** `MyException` extending `Exception`.
- In `CustomExceptionDemo`, define a method `test(int marks)`:
- If `marks < 50`, **throw** a `MyException` with the message `"Failed!"`.
- Else, print `"Passed"`.
- In the `main()` method:
- Call `test(45)` inside a **try block**.
- If exception is thrown, the **catch block** catches it and prints the error message.

Code:

```
class MyException extends Exception {
MyException(String s) {
super(s); }}
public class CustomExceptionDemo {
static void test(int marks) throws MyException {
if (marks < 50)
throw new MyException("Failed!");
else
 System.out.println("Passed");}
 public static void main(String[] args) {
try {  test(45); }
catch (MyException e){
System.out.println(e.getMessage());}}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac  CustomExceptionDemo.java

C:\Users\nishi\Desktop>java CustomExceptionDemo
Failed!
```

16c)

AIM: file not found

Software required: notepad

Algorithm:

- Import the `java.io.*` package to handle file input/output operations.
- Inside the `main()` method:
- Use a `try` block to attempt reading a file named `"missing.txt"` using `FileReader`.
- If the file does **not exist**, a `FileNotFoundException` is thrown.
- The `catch` block handles this exception and prints:
  "File not found."

Code:

```
import java.io.*;

public class FileDemo {
public static void main(String[] args) {
try {
FileReader fr = new FileReader("missing.txt");
}catch (FileNotFoundException e){
System.out.println("File not found.");}}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac FileDemo.java

C:\Users\nishi\Desktop>java FileDemo
File not found.
```

16d)

AIM: null point

Software required: notepad

Algorithm:

Code:

```
public class NullPointerDemo {
public static void main(String[] args) {
try{
String s = null;
System.out.println(s.length());}
catch (NullPointerException e){
System.out.println("Null object cannot be used.");}}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac NullPointerDemo.java

C:\Users\nishi\Desktop>java NullPointerDemo
Null object cannot be used.
```

FILE HANDLING PROGRAMS
EXPERIMENT-17

17a)

AIM: creating a file

Software required: notepad

Algorithm:

- **Import File Class:** Import the `java.io.File` class to access file-related methods.
- **Create a File Object:** Use `new File("data.txt")` to represent the file you want to check.
- **Check Existence:** Use `file.exists()` to determine if the file actually exists in the current directory.

Code:

```
 Import java.io.file:
Public class checkFile{
Public static void main(string[]args){
File file = new File("data.txt");
System.out.println("FILE EXISTS");}
Else{
System.out.println("File could not be found")
}}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac CheckFile.java

C:\Users\nishi\Desktop>java CheckFile
File not found.
```

17b)

AIM: Read a file

Software required: notepad

Algorithm:

- Import Required Classes:

- `File` to access files.
- `Scanner` to read the file.
- `FileNotFoundException` for error handling.
- Create a File Object:
- `File f = new File("data.txt");` – represents the file.
- Read File Using Scanner:
- `Scanner sc = new Scanner(f);` opens the file for reading.
- Use `while (sc.hasNextLine())` to loop through each line.
- `System.out.println(sc.nextLine());` prints each line.
- Close Scanner:
- `sc.close();` releases the file resource.
- Exception Handling:
- If the file is missing, the catch block prints "File not found."

Code:

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

OUTPUT:

```
C:\Users\nishi\Desktop>javac ReadFile.java

C:\Users\nishi\Desktop>java ReadFile
hello iam nishitha hehe
```

17c)

AIM: counting words

Software required: notepad

Algorithm:

- Import required classes: `File` and `Scanner`.
- Create a `File` object pointing to `"data.txt"`.
- Initialize a counter variable `count = 0`.
- Try to open the file using `Scanner`.
- While the file has more words,Read the next word.
- Increment the counter.
- Print the total word count.

Catch any exceptions and print `"Error reading file."`

Code:

```
import java.io.File;
import java.util.Scanner;
public class WordCount {
public static void main(String[] args) {
Try{
File file = new File("data.txt");
Scanner sc = new Scanner(file);
int count = 0;
while (sc.hasNext()){
```

```
sc.next();
count++;}
System.out.println("Word count: " + count);
sc.close();}
catch (Exception e){
System.out.println("Error reading file.");}}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac WordCount.java

C:\Users\nishi\Desktop>java WordCount
Word count: 4
```

17d)

AIM: listing files

Software required: notepad

Algorithm:

- Import the `File` class.
- Create a `File` object pointing to the current directory: `File dir = new File(".");`
- Get the list of files: `String[] files = dir.list();`
- Loop through each file name in `files`. And then print all teh files

Code:

```
import java.io.File;
public class ListFiles{
public static void main(String[] args){
File dir = new File(".");
String[] files = dir.list();
for (String name : files){
System.out.println(name);}}}
```

OUTPUT:

```
C:\Users\nishi\Desktop>javac ListFiles.java

C:\Users\nishi\Desktop>java ListFiles
0001-1526047220763767296.jpeg
0001-1526047220763767296.jpg
19CUL111[1].pdf
1a.html
1b.html
1c.html
2a.html
2b.html
6249025_Person_People_3840x2160.mp4
7a.html
8.html
8b.html
abc.html
about.html
Account.class
AdminDashboard.class
AdminDashboard.java
ai_for_pharmaZoPwq3nKXzUa0TzeyCkzlmd84N2ms5sSMtoie8Mz.webp
AMRITA
Animal.class
Animal.java
Appliance.class
Appliance.java
asdf@Nishi ~zphisher.txt
Autodesk Fusion.lnk
Bank.class
Bike.class
Bike.java
Book.class
Book.java
bored.html
bored.text
bored.txt
brownie.png
cake.png
```