LAB RECORD

23CSE111- Object Oriented Programming

Submitted by:

Vivek Patel

CH.SC.U4CSE24165

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND 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.U4CSE24165 – Vivek Patel in **"Computer Science and Engineering"** is a Bonafide record of the work carried out under my guidance and supervision at Amrita School of Computing, Chennai.

This Lab examination held on

2025

Internal Examiner 1 Internal Examiner 2

INDEX

S.NO	TITLE	PAGE.NO
	UML DIAGRAM	
1.	**TITLE OF UML DIAGRAM -1**	
	1.a)Use Case Diagram	
	1.b)Class Diagram	
	1.c) Sequence Diagram	
	1.d)	
	1.e)	
2.	**TITLE OF UML DIAGRAM -2**	
	2.a) Use Case Diagram	
	2.b) Class Diagram	
	2.c) Sequence Diagram	
	2.d)	
	2.e)	
3.	**BASIC JAVA PROGRAMS**	
	3.a)	
	3.b)	
	3.c)	
	3.d)	
	3.e)	
	3.f)	
	3.g)	
	3.h)	
	3.i)	
	3.j)	
	INHERITANCE	
4.	**SINGLE INHERITANCE PROGRAMS**	
	4.a)	
	4.b)	

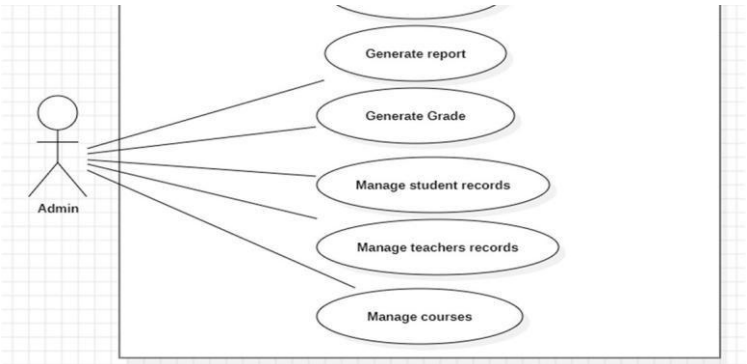
5.	**MULTILEVEL INHERITANCE PROGRAMS**	
	5.a)	
	5.b)	
6.	**HIERARCHICAL INHERITANCE PROGRAMS**	
	6.a)	
	6.b)	
7.	**HYBRID INHERITANCE PROGRAMS**	
	7.a)	
	7.b)	
	POLYMORPHISM	
8.	**CONSTRUCTOR PROGRAMS**	
	8.a)	
9.	**CONSTRUCTOR OVERLOADING PROGRAMS**	
	9.a)	
10.	**METHOD OVERLOADING PROGRAMS**	
	10.a)	

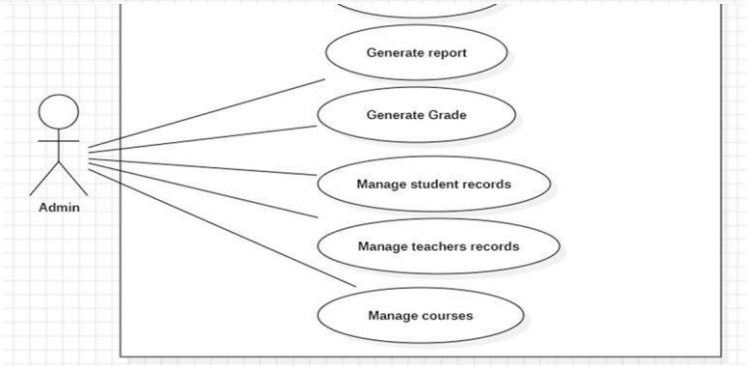
	10.b)	
11.	**METHOD OVERRIDING PROGRAMS**	
	11.a)	
	11.b)	
	ABSTRACTION	
12.	**INTERFACE PROGRAMS**	
	12.a)	
	12.b)	
	12.c)	
	12.d)	
13.	**ABSTRACT CLASS PROGRAMS**	
	13.a)	
	13.b)	
	13.c)	
	13.d)	
	ENCAPSULATION	
14.	**ENCAPSULATION PROGRAMS**	

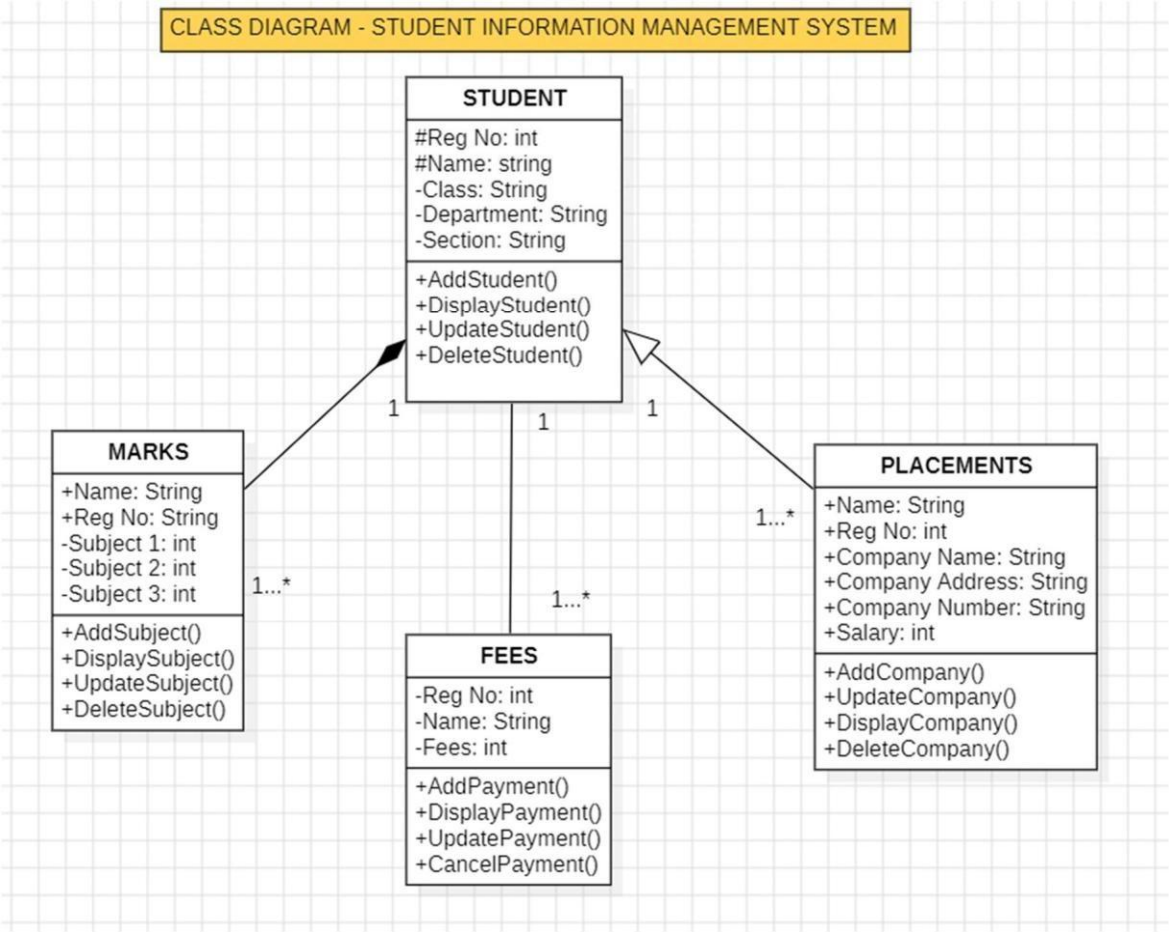
	14.a)	
	14.b)	
	14.c)	
	14.d)	
15.	**PACKAGES PROGRAMS**	
	15.a)User Defined Packages	
	15.b)User Defined Packages	
	15.c)Built – in Package(3 Packages)	
	15.d)Built – in Package(3 Packages)	
16.	**EXCEPTION HANDLING PROGRAMS**	
	16.a)	
	16.b)	
	16.c)	
	16.d)	
17.	**FILE HANDLING PROGRAMS**	
	17.a)	
	17.b)	

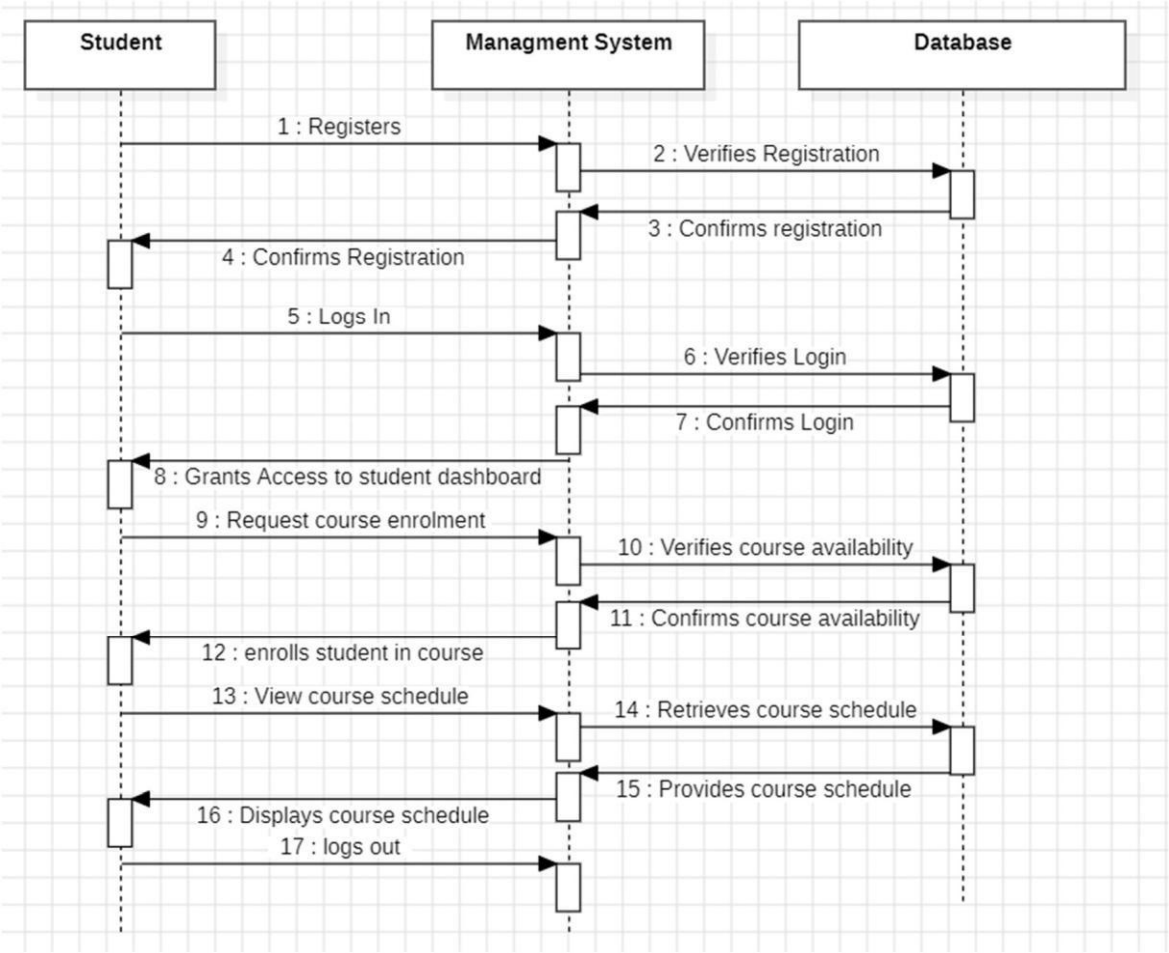
	17.c)	
	17.d)	

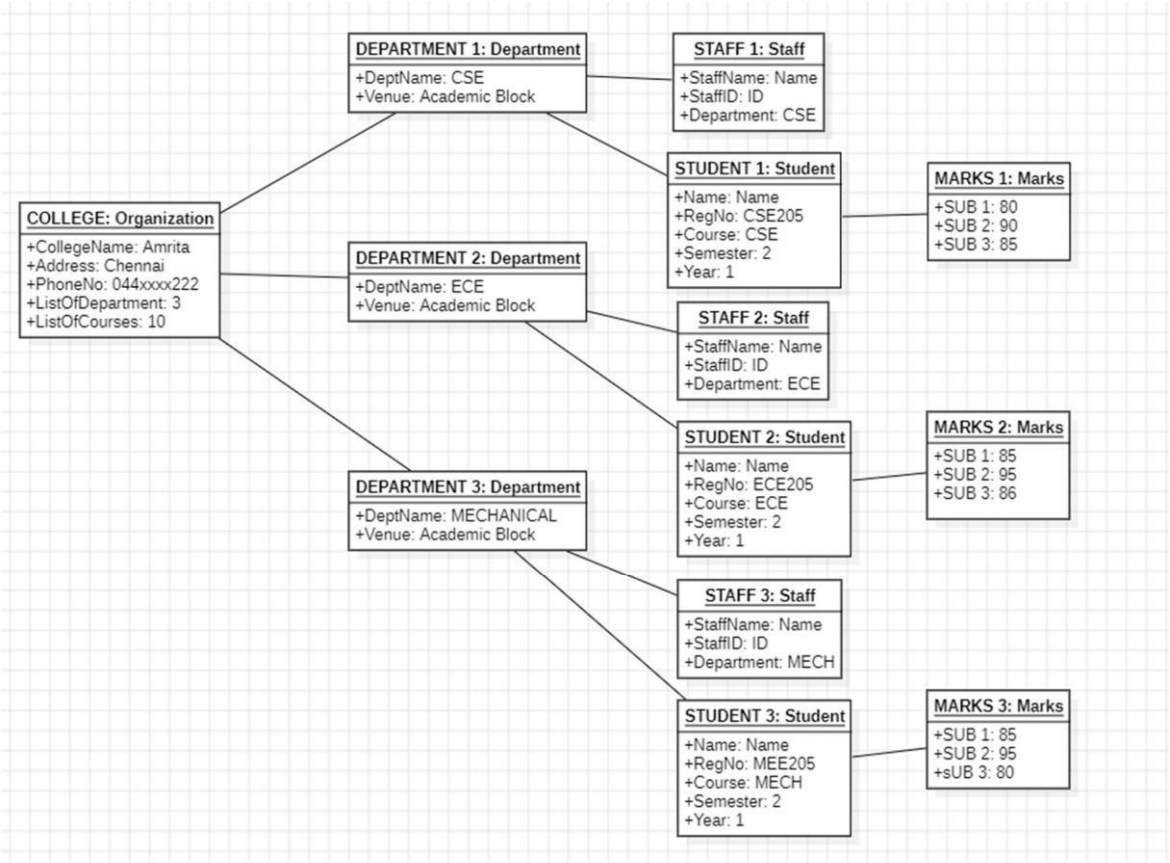
UML Diagram

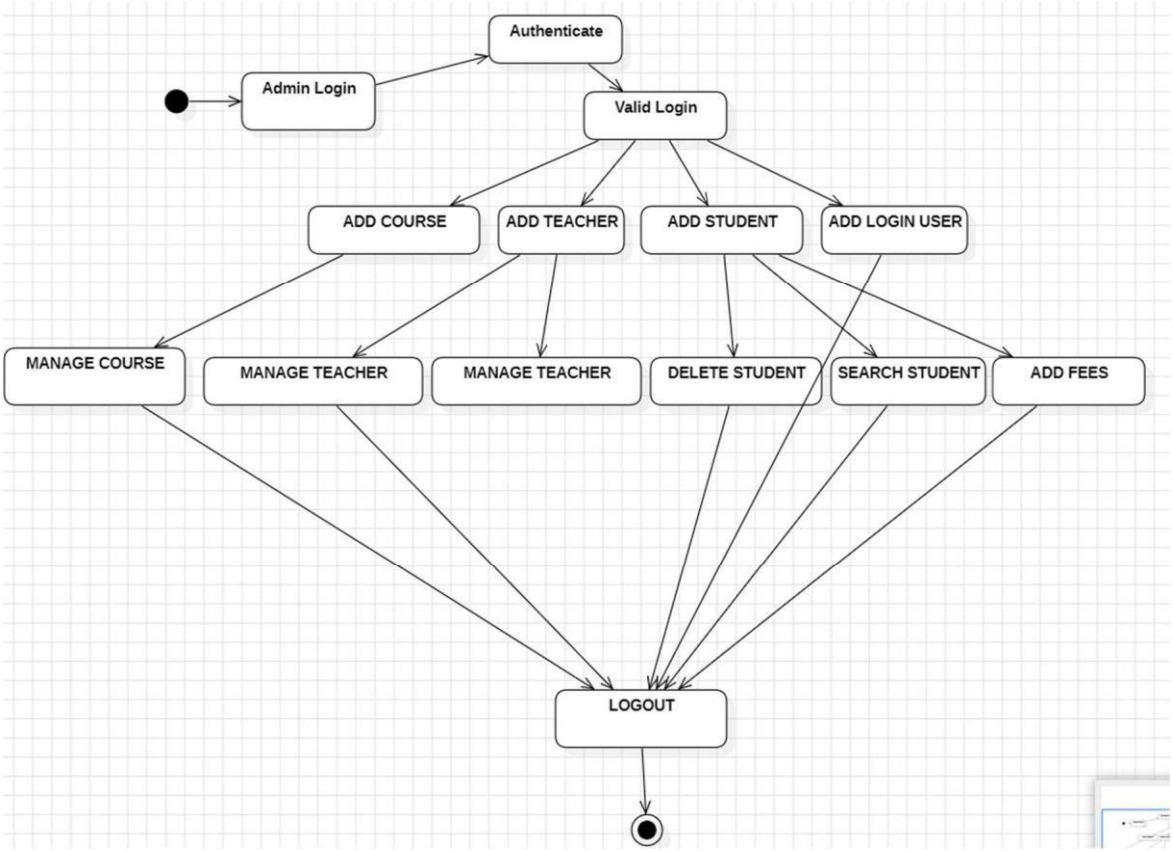


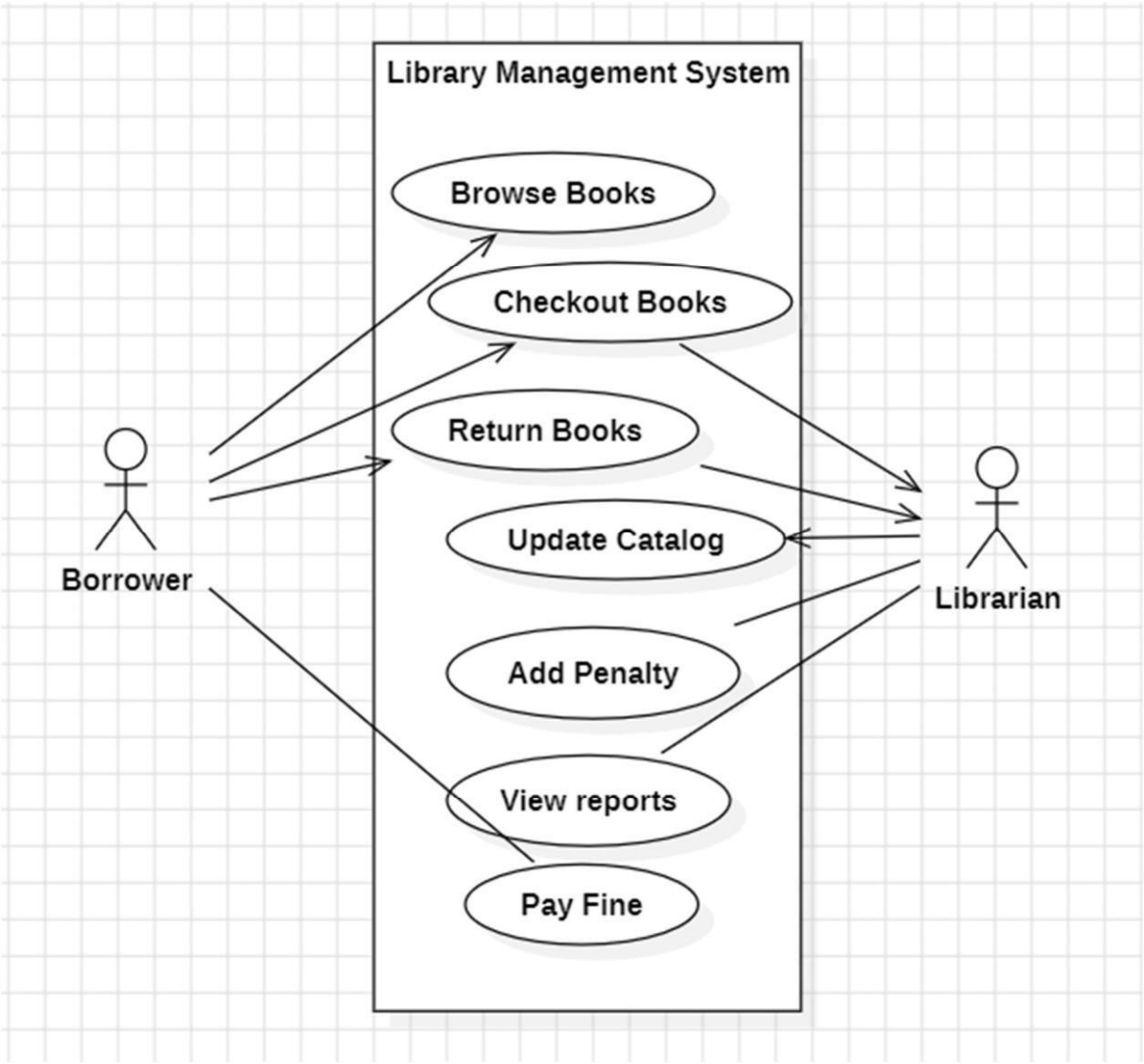


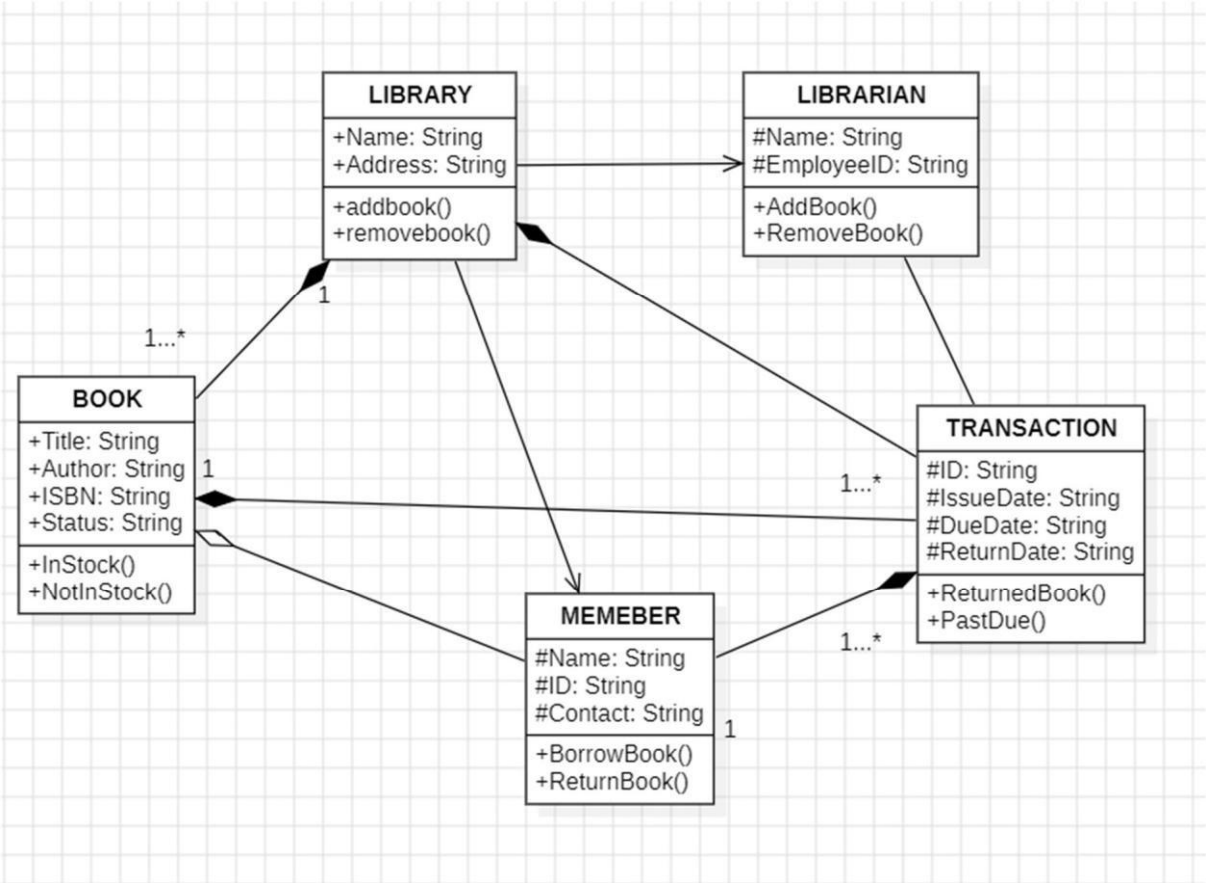

CLASS DIAGRAM - STUDENT INFORMATION MANAGEMENT SYSTEM
STUDENT
#Reg No: int
#Name: string
-Class: String
-Department: String
-Section: String
+AddStudent()
+DisplayStudent()
+UpdateStudent()
+DeleteStudent()
MARKS
+Name: String
+Reg No: String
-Subject 1: int
-Subject 2: int
-Subject 3: int
+AddSubject()
+DisplaySubject()
+UpdateSubject()
+DeleteSubject()
FEES
-Reg No: int
-Name: String
-Fees: int
+AddPayment()
+DisplayPayment()
+UpdatePayment()
+CancelPayment()
PLACEMENTS
+Name: String
+Reg No: int
+Company Name: String
+Company Address: String
+Company Number: String
+Salary: int
+AddCompany()
+UpdateCompany()
+DisplayCompany()
+DeleteCompany()
1
1
1
1...*
1...*
1...*

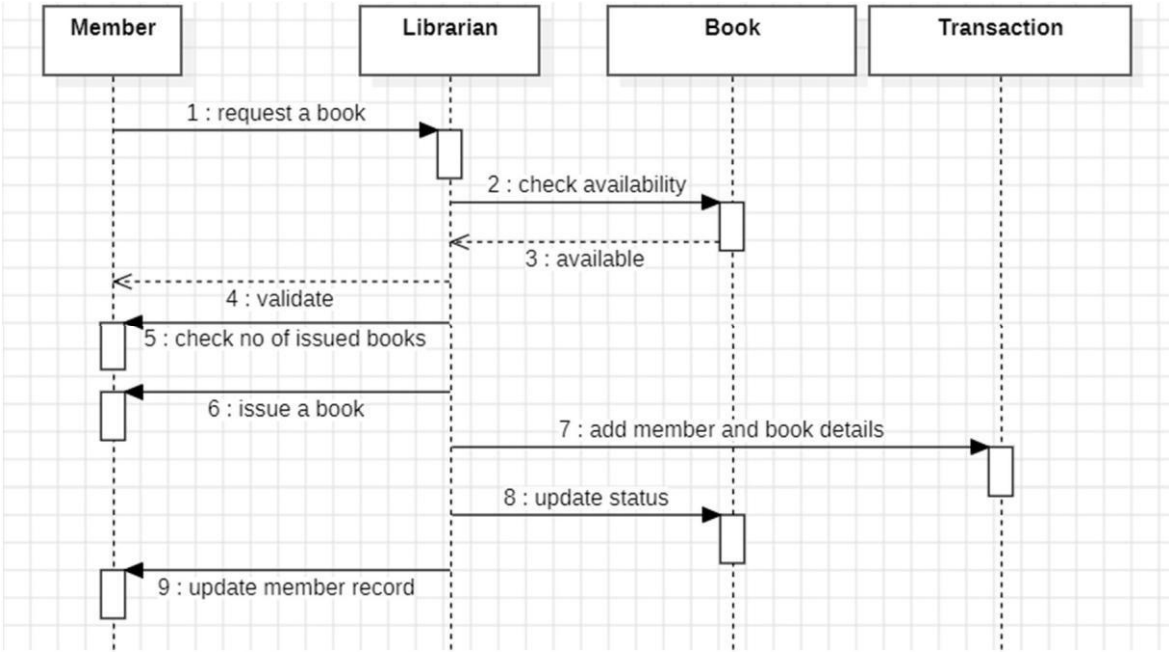

Student
Managment System
Database
1 : Registers
2 : Verifies Registration
3 : Confirms registration
4 : Confirms Registration
5 : Logs In
6 : Verifies Login
7 : Confirms Login
8 : Grants Access to student dashboard
9 : Request course enrolment
10 : Verifies course availability
11 : Confirms course availability
12 : enrolls student in course
13 : View course schedule
14 : Retrieves course schedule
15 : Provides course schedule
16 : Displays course schedule
17 : logs out

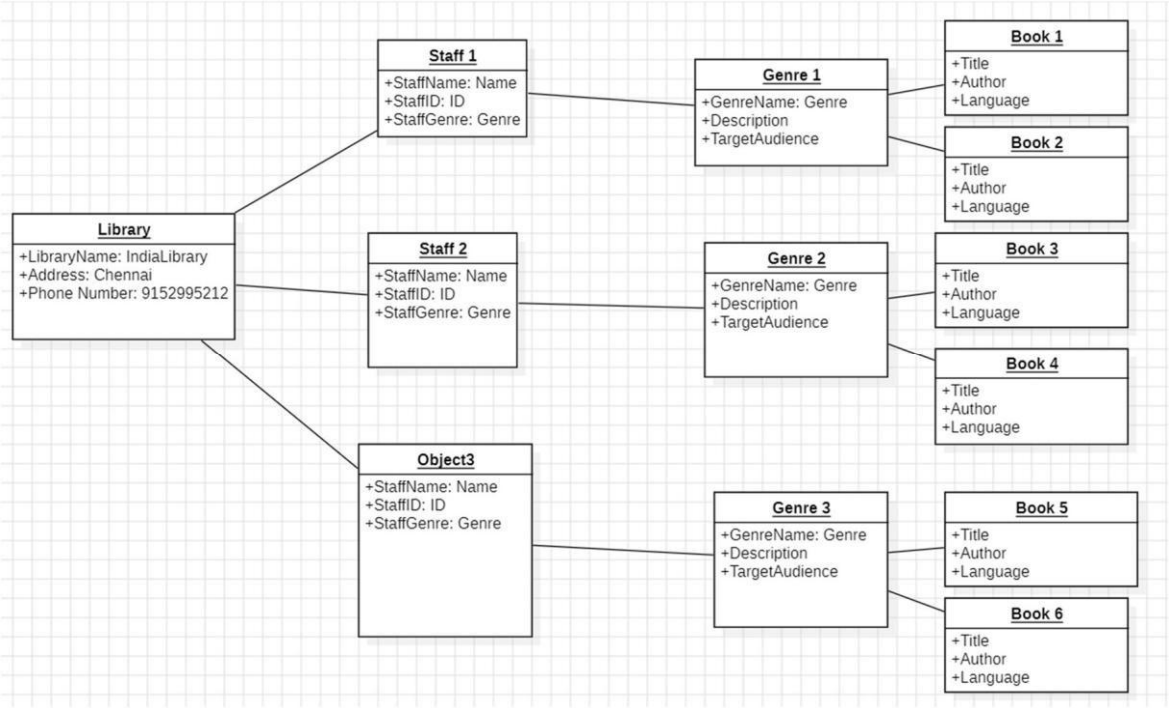

COLLEGE: Organization
+CollegeName: Amrita
+Address: Chennai
+PhoneNo: 044xxxx222
+ListOfDepartment: 3
+ListOfCourses: 10
DEPARTMENT 1: Department
+DeptName: CSE
+Venue: Academic Block
DEPARTMENT 2: Department
+DeptName: ECE
+Venue: Academic Block
DEPARTMENT 3: Department
+DeptName: MECHANICAL
+Venue: Academic Block
STAFF 1: Staff
+StaffName: Name
+StaffID: ID
+Department: CSE
STUDENT 1: Student
+Name: Name
+RegNo: CSE205
+Course: CSE
+Semester: 2
+Year: 1
MARKS 1: Marks
+SUB 1: 80
+SUB 2: 90
+SUB 3: 85
STAFF 2: Staff
+StaffName: Name
+StaffID: ID
+Department: ECE
STUDENT 2: Student
+Name: Name
+RegNo: ECE205
+Course: ECE
+Semester: 2
+Year: 1
MARKS 2: Marks
+SUB 1: 85
+SUB 2: 95
+SUB 3: 86
STAFF 3: Staff
+StaffName: Name
+StaffID: ID
+Department: MECH
STUDENT 3: Student
+Name: Name
+RegNo: MEE205
+Course: MECH
+Semester: 2
+Year: 1
MARKS 3: Marks
+SUB 1: 85
+SUB 2: 95
+sUB 3: 80

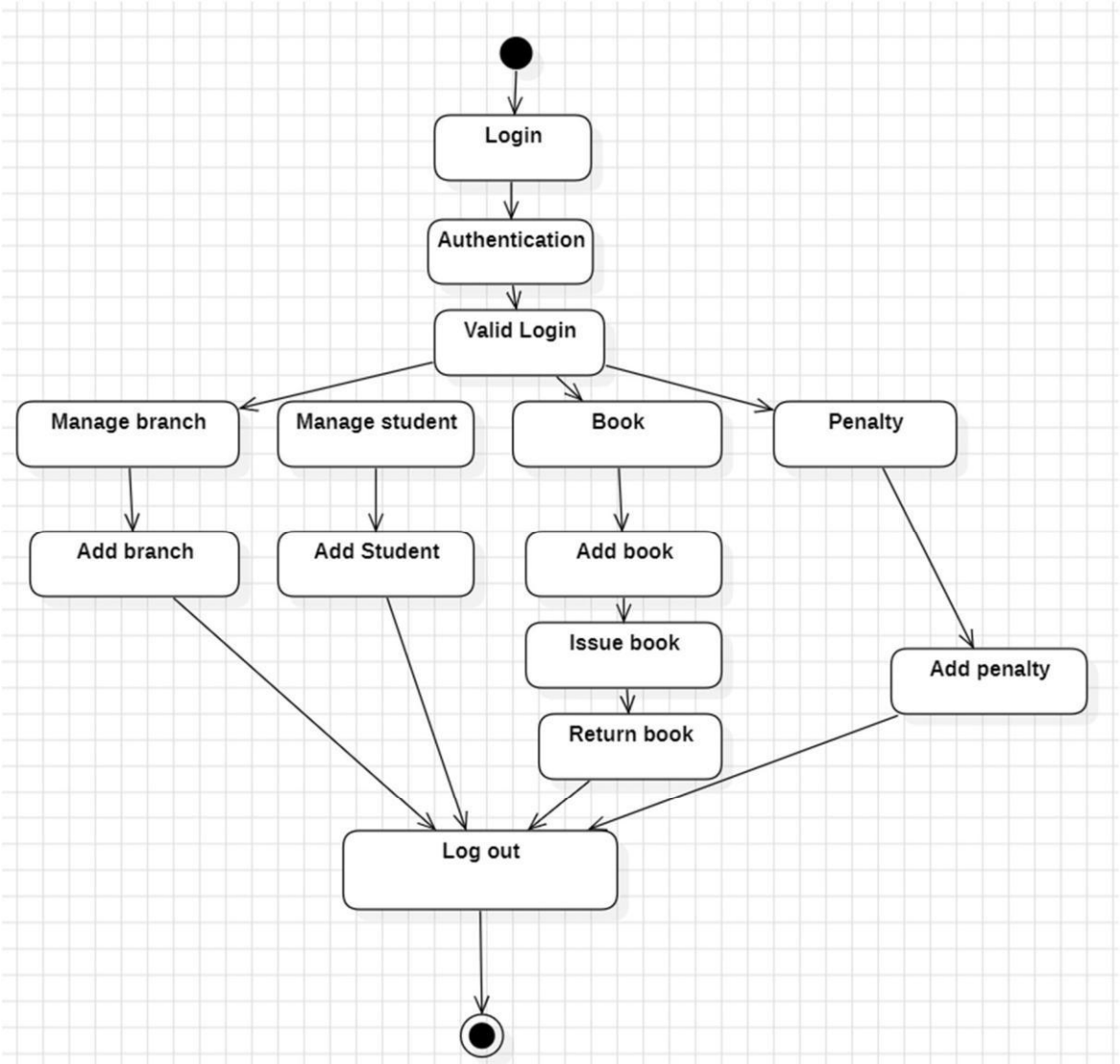

Authenticate
Admin Login
Valid Login
ADD COURSE
ADD TEACHER
ADD STUDENT
ADD LOGIN USER
MANAGE COURSE
MANAGE TEACHER
MANAGE TEACHER
DELETE STUDENT
SEARCH STUDENT
ADD FEES
LOGOUT


Library Management System
Browse Books
Checkout Books
Return Books
Update Catalog
Add Penalty
View reports
Pay Fine
Borrower
Librarian


LIBRARY
+Name: String
+Address: String
+addbook()
+removebook()
LIBRARIAN
#Name: String
#EmployeeID: String
+AddBook()
+RemoveBook()
BOOK
+Title: String
+Author: String
+ISBN: String
+Status: String
+InStock()
+NotInStock()
TRANSACTION
#ID: String
#IssueDate: String
#DueDate: String
#ReturnDate: String
+ReturnedBook()
+PastDue()
MEMEBER
#Name: String
#ID: String
#Contact: String
+BorrowBook()
+ReturnBook()
1
1...*
1
1...*
1...*
1


Member
Librarian
Book
Transaction
1 : request a book
2 : check availability
3 : available
4 : validate
5 : check no of issued books
6 : issue a book
7 : add member and book details
8 : update status
9 : update member record


Library
+LibraryName: IndiaLibrary
+Address: Chennai
+Phone Number: 9152995212
Staff 1
+StaffName: Name
+StaffID: ID
+StaffGenre: Genre
Staff 2
+StaffName: Name
+StaffID: ID
+StaffGenre: Genre
Object3
+StaffName: Name
+StaffID: ID
+StaffGenre: Genre
Genre 1
+GenreName: Genre
+Description
+TargetAudience
Genre 2
+GenreName: Genre
+Description
+TargetAudience
Genre 3
+GenreName: Genre
+Description
+TargetAudience
Book 1
+Title
+Author
+Language
Book 2
+Title
+Author
+Language
Book 3
+Title
+Author
+Language
Book 4
+Title
+Author
+Language
Book 5
+Title
+Author
+Language
Book 6
+Title
+Author
+Language



Java Programs

1) Hello World Program:

```
public class HelloWorld {

 public static void main(String[] args) {

 System.out.println("Hello, World!");

 }

}
```

Output:

Hello, World!

2) Addition of Two Numbers:

```
public class AddNumbers {

 public static void main(String[] args) {

 int num1 = 5, num2 = 10, sum;

 sum = num1 + num2;

 System.out.println("Sum of " + num1 + " and " + num2 + " is: " + sum);

 }

}
```

Output:

Sum of 5 and 10 is: 15

3) Find Maximum of Three Numbers:

```
public class MaxOfThreeNumbers {

 public static void main(String[] args) {

 int num1 = 10, num2 = 20, num3 = 15, max;

 max = (num1 > num2) ? (num1 > num3 ? num1 : num3) : (num2 > num3 ? num2 : num3);

 System.out.println("Maximum of " + num1 + ", " + num2 + ", and " + num3 + " is: " + max);

 }

}
```

Output:

Maximum of 10, 20, and 15 is: 20

4) Check Even or Odd Number:

```
public class EvenOdd {

 public static void main(String[] args) {

 int num = 5;

 if(num % 2 == 0)

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

 else

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

 }

}
```

Output:

5 is odd.

5) Factorial of a Number:

```
public class Factorial {

 public static void main(String[] args) {

 int num = 5, factorial = 1;

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

 factorial *= i;

 }

 System.out.println("Factorial of " + num + " is: " + factorial);

 }

}
```

Output:

Factorial of 5 is: 120

6) Print Pattern in Java:

```
public class PrintPattern {
 public static void main(String[] args) {
 int rows = 5;
 for (int i = 1; i <= rows; ++i) {
 for (int j = 1; j <= i; ++j) {
 System.out.print("* ");
 }
 System.out.println();
 }
 }
}
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

7) Add Two Binary Numbers in Java:

```
public class AddBinaryNumbers {

 public static void main(String[] args) {

 long binary1 = 1010, binary2 = 1101;

 int i = 0, remainder = 0;

 long[] sum = new long[20];

 while (binary1 != 0 || binary2 != 0) {

 sum[i++] = (binary1 % 10 + binary2 % 10 + remainder) % 2;

 remainder = (int) (binary1 % 10 + binary2 % 10 + remainder) / 2;

 binary1 = binary1 / 10;

 binary2 = binary2 / 10;

 }

 if (remainder != 0) {

 sum[i++] = remainder;

 }

 –i;

 System.out.print("Sum of two binary numbers: ");

 while (i >= 0) {

 System.out.print(sum[i–]);

 }

 }

}
```

Output:

Sum of two binary numbers: 11011

8) Add Two Complex Numbers in Java:

```
class Complex {
 double real, imaginary;
 Complex(double r, double i) {
 this.real = r;
 this.imaginary = i;
 }
 public static Complex add(Complex c1, Complex c2) {
 Complex temp = new Complex(0, 0);
 temp.real = c1.real + c2.real;
 temp.imaginary = c1.imaginary + c2.imaginary;
 return temp;
 }
}
public class AddComplexNumbers {
 public static void main(String[] args) {
 Complex c1 = new Complex(4.5, 5);
 Complex c2 = new Complex(2.5, 3.5);
 Complex temp = Complex.add(c1, c2);
 System.out.println("Sum = " + temp.real + " + " + temp.imaginary + "i");
 }
}
```

Output:

```
Sum = 7.0 + 8.5i
```

9) Multiply Two Numbers in Java:

```
public class MultiplyTwoNumbers {

 public static void main(String[] args) {

 double first = 2.5, second = 4.5;

 double product = first * second;

 System.out.println("The product is: " + product);

 }

}
```

Output:

The product is: 11.25

10) Check Leap Year in Java:

```
public class LeapYear {

 public static void main(String[] args) {

 int year = 2024;

 if (((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)) {

 System.out.println(year + " is a leap year.");

 } else {

 System.out.println(year + " is not a leap year.");

 }

 }

}
```

Output:

2024 is a leap year.

Single Inheritance Program

Java

```
class Employee {
   void salary() {
      System.out.println("Salary= 200000");
   }
}

class Programmer extends Employee {
// Programmer class inherits from Employee class
   void bonus() {
      System.out.println("Bonus=50000");
   }
}

class single_inheritance {
   public static void main(String args[]) {
      Programmer p = new Programmer();
      p.salary(); // calls method of super class
      p.bonus(); // calls method of sub class
   }
}
```

```
Salary= 200000
Bonus=50000
```

Java

```
class Animal {
    void eat() {
        System.out.println("Animal is eating");
    }
}

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

class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.eat(); // Inherited from Animal
        myDog.bark();
    }
}
```

```
Animal is eating
Dog is barking
```

Abstract Programs(Abstract Class)

abstract class Shape {

```
// Abstract method (must be implemented
by subclasses)
abstract double calculateArea();

// Concrete method (default
implementation)    void
display() {
System.out.println("This is a shape.");
}
}

class Circle extends Shape {
double radius;

Circle(double r) {
radius = r;
}
double calculateArea() {
return Math.PI * radius * radius;
```

```
}
}

class Square extends Shape {
double side;

Square(double s) {
side = s;
}
double calculateArea() {
return side * side;
}
}

public class ShapeDemo {
public static void main(String[] args) {
Shape circle = new Circle(5);
Shape square = new Square(4);
```

```
System.out.println("Circle Area: " +
circle.calculateArea());
System.out.println("Square Area: " +
square.calculateArea());
circle.display();
}
}
```

```
Circle Area: 78.53981633974483
Square Area: 16.0
This is a shape.
```

```
abstract class Animal {
abstract void makeSound();
abstract void move();
}

class Dog extends Animal {
void makeSound() {
System.out.println("Bark!");
}
```

```
void
move() {
System.out.println("Running on four legs.");
}
}

class Bird extends Animal {
void makeSound() {
System.out.println("Chirp!");
}
void
move() {
System.out.println("Flying in the sky.");
}
}

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

```
Animal dog = new Dog();
Animal bird = new Bird();

dog.makeSound();
dog.move();
bird.makeSound();
bird.move();
}
}
```

```
Bark!
Running on four legs.
Chirp!
Flying in the sky.
```

```
abstract class BankAccount {
double balance;

BankAccount(double initialBalance) {
balance = initialBalance;
}
```

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

// Abstract method (must be implemented)
abstract void withdraw(double amount);

void checkBalance() {
System.out.println("Current Balance: $" +
balance);
}
}

class SavingsAccount extends BankAccount {
SavingsAccount(double initialBalance) {
super(initialBalance);
```

```
}

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

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

```
account.withdraw(200);
account.checkBalance();
}
}
```

```
Deposited: $500.0
Withdrawn: $200.0
Current Balance: $1300.0
```

```
abstract class Employee {
String name;
double salary;

Employee(String n, double s) {
name = n;
salary = s;
}

// Abstract method abstract
double calculateBonus();
```

```
// Concrete method
void display() {
System.out.println("Employee: " + name);
System.out.println("Salary: $" + salary);
System.out.println("Bonus: $" +
calculateBonus());
}
}

class Manager extends Employee {
Manager(String n, double s) {
super(n, s);
}
double calculateBonus() {
return salary * 0.20; // 20% bonus
}
}

class Developer extends Employee {
```

```
Developer(String n, double s) {
super(n, s);
}
double calculateBonus() {
return salary * 0.15; // 15% bonus
}
}

public class EmployeeDemo {
public static void main(String[] args) {
Employee emp1 = new Manager("John",
80000);
Employee emp2 = new Developer("Alice",
60000);

emp1.display();
emp2.display();
}
}
```

```
Employee: John
Salary: $80000.0
Bonus: $16000.0
Employee: Alice
Salary: $60000.0
Bonus: $9000.0
```

Interface Programs interface PaymentGateway {
void processPayment(double amount);
void refund(double amount);
}

class CreditCardPayment implements PaymentGateway {
@Override
public void processPayment(double amount) {
System.out.println("Processing Credit Card Payment: $" + amount);
}

```
@Override
public void refund(double amount) {
    System.out.println("Refunding $" +
    amount + " to Credit Card");
}
}

class UPIPayment implements
PaymentGateway {
@Override
public void processPayment(double
amount) {
    System.out.println("Processing UPI
    Payment: ₹" + amount);
}

@Override
public void refund(double amount) {
```

```
        System.out.println("Refunding ₹" +
            amount + " via UPI");
        }
    }

public class PaymentSystem {
    public static void main(String[] args) {
        PaymentGateway creditCard = new
            CreditCardPayment();
        PaymentGateway upi = new
            UPIPayment();

        creditCard.processPayment(100.50);
        upi.processPayment(500);

        creditCard.refund(20.25);
        }
    }
```

```
Processing Credit Card Payment: $100.50
Processing UPI Payment: ₹500.00
Refunding $20.25 to Credit Card
Refunding ₹150.75 via UPI
```

```
interface NotificationService {
void sendNotification(String message);
}

class EmailService implements
NotificationService {
public void sendNotification(String
message) {
System.out.println("Email Sent: " +
message);
}
}

class SMSService implements
```

```
NotificationService {
public void sendNotification(String
message) {
        System.out.println("SMS Sent: " +
                message);
                }
                }

class PushNotification implements
NotificationService {   public void
sendNotification(String message) {
System.out.println("Push Notification: " +
                message);
                }
                }

public class NotificationSystem {
public static void main(String[] args) {
        NotificationService email = new
```

```
EmailService();
NotificationService sms = new SMSService();

email.sendNotification("Your order is confirmed!");
sms.sendNotification("OTP: 123456");
}
}
```

```
Email Sent: Your order is confirmed!
SMS Sent: OTP: 123456
```

```
interface Database {
void connect();
void disconnect();
}
```

```
abstract class AbstractDatabase implements
Database {
abstract void logConnection(String
message);

public void connect() {
logConnection("Database connected.");
}
public void disconnect() {
logConnection("Database disconnected.");
}
}

class MySQLDatabase extends
AbstractDatabase {
void logConnection(String message) {
System.out.println("MySQL Log: " +
message);
```

```
    }
}

class MongoDB extends AbstractDatabase {
    void logConnection(String message) {
        System.out.println("MongoDB Log: " +
message);
    }
}

public class DatabaseSystem {
    public static void main(String[] args) {
        Database mysql = new MySQLDatabase();
        Database mongo = new MongoDB();

        mysql.connect();
        mongo.disconnect();
    }
}
```

```
MySQL Log: Database connected.
MongoDB Log: Database disconnected.
```

Built in Package import
java.util.ArrayList;
import java.util.Scanner; import
java.io.File;
import java.io.IOException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class BuiltInPackageDemo {
public static void main(String[] args) {
System.out.println("\n=== java.util
Package Examples ===");
Scanner scanner = new
Scanner(System.in);
ArrayList<String> names = new
ArrayList<>();

```
System.out.print("Enter 3 names: ");
for (int i = 0; i < 3; i++) {
names.add(scanner.next());
}

System.out.println("Names you entered: " + names);

System.out.println("\n=== java.io Package Examples ===");
try {
File file = new File("test.txt");
if (file.createNewFile()) {
System.out.println("File created: " + file.getName());
} else {
System.out.println("File already exists.");
```

```
}
System.out.println("File path: " +
file.getAbsolutePath());
} catch (IOException e) {
System.out.println("An error occurred:
" + e.getMessage());
}

System.out.println("\n=== java.time
Package Examples ===");
LocalDate today = LocalDate.now();
System.out.println("Current date: " +
today);

LocalDate nextWeek =
today.plusWeeks(1);
System.out.println("Date after one week: "
+ nextWeek);
```

```
DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd-
MMMyyyy");
System.out.println("Formatted date: " +
today.format(formatter));

scanner.close();
}
}
```

```
=== java.util Package Examples ===
Enter 3 names: Alice Bob Charlie
Names you entered: [Alice, Bob, Charlie]

=== java.io Package Examples ===
File created: test.txt
File path: /your/path/to/project/test.txt

=== java.time Package Examples ===
Current date: 2025-04-05
Date after one week: 2025-04-12
Formatted date: 05-Apr-2025
```

```
// Source code is decompiled from a .class file
using FernFlower decompiler.
import java.io.PrintStream;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.text.DecimalFormat; import
java.text.NumberFormat; import
java.util.HashMap;
import java.util.Random;

public class BuiltInPackageDemo2 {
public BuiltInPackageDemo2() {
}

public static void main(String[] var0) {
System.out.println("=== java.util Package
===");
```

```
HashMap var1 = new HashMap();
var1.put("Alice", 85);
var1.put("Bob", 92);
var1.put("Charlie", 78);
System.out.println("Student Grades: " +
String.valueOf(var1));
Random var2 = new Random();
int var3 = var2.nextInt(100);
System.out.println("Random number: " +
var3);
System.out.println("\n=== java.text
Package ===");
double var4 = 1234.5678;
NumberFormat var6 =
NumberFormat.getCurrencyInstance();
PrintStream var10000 = System.out;
String var10001 = var6.format(var4);
var10000.println("Formatted price: " +
var10001);
```

```
DecimalFormat var7 = new
DecimalFormat("#.##");
var10000 = System.out;
var10001 = var7.format(var4);
var10000.println("Rounded price: " +
var10001);
System.out.println("\n=== java.math
Package ===");
BigDecimal var8 = new
BigDecimal("1000.00");
BigDecimal var9 = new
BigDecimal("0.05");
BigDecimal var10 =
var8.multiply(var9).setScale(2,
RoundingMode.HALF_UP);
System.out.println("Principal: $" +
String.valueOf(var8));
System.out.println("Interest (5%): $" +
```

```
String.valueOf(var10));
var10000 = System.out;
BigDecimal var11 = var8.add(var10);
var10000.println("Total after interest: $" +
String.valueOf(var11));
}
}
```

```
=== java.util Package ===
Student Grades: {Alice=85, Bob=92, Charlie=78}
Random number: 47

=== java.text Package ===
Formatted price: ₹1,234.57
Rounded price: 1234.57

=== java.math Package ===
Principal: $1000.00
Interest (5%): $50.00
Total after interest: $1050.00
```

Encapsulation Program public class BankAccount {
private String accountNumber;
private double balance;

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

// Getter for balance (read-only access)
public double getBalance() {
return balance;
}

// Setter for deposit (controlled modification)

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

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

```
System.out.println("Invalid withdrawal amount!");
}
}
}

public class BankDemo {
public static void main(String[] args) {
BankAccount account = new BankAccount("123456", 1000);
account.deposit(500);
account.withdraw(200);
System.out.println("Current Balance: $" + account.getBalance());
}
}
```

```
Deposited: $500.0
Withdrawn: $200.0
Current Balance: $1300.0
```

```
public class Student {
    private String name;
    private int age;
    private double gpa;

public Student(String name, int age, double gpa) {
    setName(name); // Use setter for validation
    setAge(age);
    setGpa(gpa);
}

// Getter methods
public String getName() { return name; }
public int getAge() { return age; }     public double getGpa() { return gpa; }
```

```
// Setter methods with validation
public void setName(String name) {
if (name != null && !name.isEmpty()) {
this.name = name;
} else {
System.out.println("Invalid name!");
}
}
public void setAge(int age) {
if (age >= 0 && age <= 120) {
this.age = age;
} else {
System.out.println("Invalid age!");
}
}
public void setGpa(double gpa) {
if (gpa >= 0.0 && gpa <= 4.0) {
this.gpa = gpa;
} else {
```

```
System.out.println("Invalid GPA!");
}
}
}

public class StudentDemo {
public static void main(String[] args) {
Student student = new Student("Alice",
20, 3.8);
System.out.println("Name: " +
student.getName());
System.out.println("Age: " +
student.getAge());
System.out.println("GPA: " +
student.getGpa());

student.setGpa(4.5); // Invalid input
}
}
```

```
Name: Alice
Age: 20
GPA: 3.8
Invalid GPA!
```

```
public class Temperature {
    private double celsius;

public Temperature(double celsius) {
        this.celsius = celsius;
                }

            // Getter for Celsius
    public double getCelsius() {
    return celsius;
                }

            // Setter for Celsius
public void setCelsius(double celsius) {
        this.celsius = celsius;
```

```
}

// Read-only Fahrenheit conversion
public double getFahrenheit() {
    return (celsius * 9/5) + 32;
}

// Read-only Kelvin conversion
public double getKelvin() {
    return celsius + 273.15;
}
}

public class TemperatureDemo {
public static void main(String[] args) {
Temperature temp = new
    Temperature(25);
    System.out.println("Celsius: " +
    temp.getCelsius());
```

```
System.out.println("Fahrenheit: " +
temp.getFahrenheit());
System.out.println("Kelvin: " +
temp.getKelvin());

temp.setCelsius(0);
System.out.println("\nNew Celsius: " +
temp.getCelsius());
}
}
```

```
Celsius: 25.0
Fahrenheit: 77.0
Kelvin: 298.15

New Celsius: 0.0
```

```
public class Employee {
private String name;
private String id;
private double salary;
public Employee(String
```

```
name, String id, double
salary) {
        this.name = name;
        this.id = id;
setSalary(salary); // Use setter for
validation
}

// Getter methods
public String getName() { return name; }
public String getId() { return id; }    public
double getSalary() { return salary; }

// Setter with validation
public void setSalary(double salary) {
if (salary >= 0) {
        this.salary = salary;
} else {
```

```
System.out.println("Salary cannot be negative!");
}
}
//
Meth
od to
appl
y
raise
(enc
apsul
ated
logic
)
public void applyRaise(double percentage) {
if (percentage > 0) {
salary += salary * (percentage / 100);
System.out.println("Raise applied. New salary: $" + salary);
```

```
} else {
System.out.println("Invalid raise percentage!");
}
}
}

public class EmployeeDemo {
public static void main(String[] args) {
Employee emp = new Employee("John", "E1001", 50000);
System.out.println("Name: " + emp.getName());
System.out.println("ID: " + emp.getId());
System.out.println("Salary: $" + emp.getSalary());

emp.applyRaise(10);
emp.setSalary(-1000); // Invalid input
```

```
    }
}
```

```
Name: John
ID: E1001
Salary: $50000.0
Raise applied. New salary: $55000.0
Salary cannot be negative!
```

Exception Handling Programs

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

        try {
            // This will throw ArrayIndexOutOfBoundsException
            System.out.println("Element at index 3: " + numbers[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
```

```
            System.out.println("Caught
ArrayIndexOutOfBoundsException:");
        System.out.println("Message: " +
            e.getMessage());
        System.out.println("Stack Trace:");
            e.printStackTrace();
            }

                try {
    // This will throw ArithmeticException
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("\nCaught
            ArithmeticException:");
        System.out.println("Message: " +
            e.getMessage());
            }
```

```
System.out.println("\nProgram continues after exceptions...");
}
}
```

```
Caught ArrayIndexOutOfBoundsException:
Message: Index 3 out of bounds for length 3
Stack Trace:
java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
    at BasicExceptionHandling.main(BasicExceptionHandling.java:6)

Caught ArithmeticException:
Message: / by zero

Program continues after exceptions...
```

```
import java.util.Scanner;

public class MultipleCatchBlocks {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);

try {
```

```
System.out.print("Enter numerator: ");
int numerator = scanner.nextInt();

System.out.print("Enter denominator: ");
int denominator = scanner.nextInt();

int result = numerator / denominator;
System.out.println("Result: " + result);

String str = null;
System.out.println("Length: " + str.length()); // Will throw NullPointerException

} catch (ArithmeticException e) {
System.out.println("Error: Division by zero is not allowed.");
} catch (NullPointerException e) {
```

```
System.out.println("Error: Null reference encountered.");
} catch (Exception e) {
System.out.println("An unexpected error occurred: " + e.getMessage());
} finally {
System.out.println("This block always executes, regardless of exceptions.");
scanner.close();
}

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

```
Enter numerator: 10
Enter denominator: 0
```

```
// Custom exception class  class InsufficientFundsException extends
```

```
Exception {  private
double amount;
public InsufficientFundsException(double
amount) {
super("Insufficient funds: " + amount);
this.amount = amount;
}
public double getAmount() {  return
amount;
}
}
class BankAccount {
private double balance;
public BankAccount(double initialBalance) {
this.balance = initialBalance;
}
public void withdraw(double amount) throws
```

```
InsufficientFundsException {  if
(amount > balance) {  throw
new
InsufficientFundsException(amount -
balance);
}
balance -= amount;
System.out.println("Withdrawal successful.
Remaining balance: " + balance);
}
}
public class CustomExceptionDemo {
public static void main(String[] args) {
BankAccount account = new
BankAccount(1000);
try {
System.out.println("Current balance: 1000");
System.out.println("Withdrawing 600...");
account.withdraw(600);
```

```
System.out.println("\nWithdrawing 500...");
account.withdraw(500);
System.out.println("\nWithdrawing 200...");
 account.withdraw(200); // This will throw
 the custom exception
   } catch (InsufficientFundsException e) {
        System.out.println("Error: " +
              e.getMessage());
        System.out.println("You need $" +
   e.getAmount() + " more to complete this
              transaction.");
                    }
                    }
                    }
```

```
Current balance: 1000
Withdrawing 600...
Withdrawal successful. Remaining balance: 400.0

Withdrawing 500...
Error: Insufficient funds: 100.0
You need $100.0 more to complete this transaction.
```

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class ExceptionPropagation {
public static void main(String[] args) {
try {
readFile("nonexistent.txt");
} catch (FileNotFoundException e) {
System.out.println("File not found error
caught in main():");
System.out.println(e.getMessage());
}

try {
processData("123");
processData("abc"); // This will cause
NumberFormatException
} catch (NumberFormatException e) {
```

```
        System.out.println("\nNumber format
            error caught in main():");
        System.out.println(e.getMessage());
                }
            }

        // Method declares it throws
            FileNotFoundException
    public static void readFile(String filename)
        throws FileNotFoundException {
            File file = new File(filename);
        Scanner scanner = new Scanner(file);
        System.out.println("File content: " +
                scanner.nextLine());
                scanner.close();
                }

    // Exception propagates up the call stack
public static void processData(String input)
```

```
{
int number = Integer.parseInt(input);
System.out.println("Processed number: "
+ (number * 2));
}
}
```

```
File not found error caught in main():
nonexistent.txt (No such file or directory)
Processed number: 246

Number format error caught in main():
For input string: "abc"
```

File Handling Programs

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

public class BasicFileOperations {
```

```
public static void main(String[] args) {
    String fileName = "example.txt";

    // Create a new file
    try {
        File file = new File(fileName);
        if (file.createNewFile()) {
            System.out.println("File created: " +
                file.getName());
        } else {
            System.out.println("File already
                exists.");
        }
    } catch (IOException e) {
        System.out.println("An error occurred while
            creating file.");
        e.printStackTrace();
    }
```

```
// Write to file
try {
FileWriter writer = new FileWriter(fileName);
writer.write("Hello, this is line 1.\nThis is line 2.\nLine 3.");
writer.close();
System.out.println("Successfully wrote to the file.");
} catch (IOException e) {
System.out.println("An error occurred while writing to file.");
e.printStackTrace();
}

// Read from file
try {
File file = new File(fileName);
Scanner reader = new Scanner(file);
```

```
System.out.println("\nFile content:");
while (reader.hasNextLine()) {
    String data = reader.nextLine();
    System.out.println(data);
}
reader.close();
} catch (IOException e) {
System.out.println("An error occurred
while reading file.");
    e.printStackTrace();
}

// Delete file
File file = new File(fileName);
if (file.delete()) {
System.out.println("\nDeleted the file: "
+ file.getName());
} else {
```

```
System.out.println("Failed to delete the file.");
        }
    }
}
```

```
File created: example.txt
Successfully wrote to the file.

File content:
Hello, this is line 1.
This is line 2.
Line 3.

Deleted the file: example.txt
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FileCopy {
public static void main(String[] args) {
    String sourceFile = "source.txt";
    String destFile = "destination.txt";
```

```
try {
// Create source file with some content
FileOutputStream fosSource = new
FileOutputStream(sourceFile);
String content = "This is the source file
content.\nSecond line of text.";
fosSource.write(content.getBytes());
fosSource.close();

// Copy file
FileInputStream fis = new
FileInputStream(sourceFile);
FileOutputStream fosDest = new
FileOutputStream(destFile);

int byteData;
while ((byteData = fis.read()) != -1) {
fosDest.write(byteData);
```

```
                }

            fis.close();
                fosDest.close();

            System.out.println("File copied
successfully from " + sourceFile + " to " +
destFile);
            } catch (IOException e) {
        System.out.println("An error occurred
during file copy.");
                e.printStackTrace();
                }
            }
        }
```

```
CSV file created successfully.

Reading CSV file:
Name: John Doe   | Age: 32  | Department: IT
Name: Jane Smith | Age: 28  | Department: HR
Name: Bob Johnson| Age: 45  | Department: Finance
```

```
Name,Age,Department
John Doe,32,IT
Jane Smith,28,HR
Bob Johnson,45,Finance
```

```
import java.io.*;

class Student implements Serializable {
private static final long serialVersionUID =
1L;
private String name;
private int rollNumber;
private transient String password; //
transient fields are not serialized

public Student(String name, int rollNumber,
```

```
String password) {
this.name = name;
this.rollNumber = rollNumber;
this.password = password;
}

@Override public
String toString() { return "Student
[name=" + name + ", rollNumber=" +
rollNumber +
", password=" + password + "]";
}
}

public class ObjectSerialization {
public static void main(String[] args) {
String fileName = "student.ser";
Student student = new Student("Alice",
101, "secure123");
```

```
// Serialization
try (ObjectOutputStream oos = new
ObjectOutputStream(new
FileOutputStream(fileName))) {
oos.writeObject(student);
System.out.println("Object serialized
and saved to " + fileName);
} catch (IOException e) {
System.out.println("Error during
serialization.");
e.printStackTrace();
}

// Deserialization try
(ObjectInputStream ois = new
ObjectInputStream(new
FileInputStream(fileName))) {
```

```
Student deserializedStudent = (Student)
ois.readObject();
System.out.println("\nDeserialized
Student:");
System.out.println(deserializedStudent)
;
System.out.println("Note: Password
field was transient and not serialized");
} catch (IOException |
ClassNotFoundException e) {
System.out.println("Error during
deserialization.");
e.printStackTrace();
}
}
}
```

```
Object serialized and saved to student.ser

Deserialized Student:
Student [name=Alice, rollNumber=101, password=null]
Note: Password field was transient and not serialized
```

Inbuilt Programs

// Source code is decompiled from a .class file using FernFlower decompiler. package citymanagement;

import java.util.Scanner;

public class City {
private String cityname;
private int population;
private double budget;
private int powersupply;

```
private int watersupply;

public City(String var1) {
this.cityname = var1;
  this.population = 10000;
this.budget = 500000.0;
 this.powersupply = 1000;
  this.watersupply = 800;
}
public void increasepopulation(int var1) {
  this.population += var1;      this.budget -=
  (double)(var1 * 50);
}

public void decreasepopulation(int var1) {
this.population -= var1;
if (this.population < 0) {
  this.population = 0;
}
```

```
}
public void adjustbudget(double var1) {
this.budget += var1;
}
public void buildpowerplant(int var1) {
this.powersupply += var1;      this.budget -= (double)(var1 * 1000);
}
public void buildwaterfacility(int var1) {
this.watersupply += var1;      this.budget -= (double)(var1 * 800);
}
public String getcitystats() {
return this.cityname + " - Pop: " + this.population + ", Budget: $" + this.budget + ", Power: " + this.powersupply + ", Water: " + this.watersupply;
}
```

```
public void collecttaxes(double var1) {
this.budget += (double)this.population *
var1;
}
public void repairinfrastructure(int var1) {
this.budget -= (double)var1;
this.powersupply += 50;
this.watersupply += 40;
}
public void simulateday(Scanner var1) {
System.out.println("Enter population
growth: ");
this.increasepopulation(var1.nextInt());
System.out.println("Enter tax rate: ");
this.collecttaxes(var1.nextDouble());  }
public boolean issustainable() {
return this.budget > 0.0 &&
this.powersupply > this.population / 10 &&
this.watersupply > this.population / 15;
```

```
}
public void expandcity(int var1, int var2, int var3) {
this.population += var1;
this.powersupply += var2;
this.watersupply += var3;      this.budget -= (double)(var1 * 50 + var2 * 1000 + var3 * 800);
}
public static void main(String[] var0) {
Scanner var1 = new Scanner(System.in);
System.out.println("Enter city name: ");
String var2 = var1.nextLine();
City var3 = new City(var2);
var3.simulateday(var1);
System.out.println(var3.getcitystats());
var1.close();
}
}
```

```
Enter city name:
EcoVille
Enter population growth:
500
Enter tax rate:
12.5
EcoVille - Pop: 10500, Budget: $531250.0, Power: 1000, Water: 800
```

```
package citymanagement;

import java.util.Scanner;

public class City {
private String cityname;
private int population;
private double budget;
private int powersupply;
private int watersupply;    public
```

```
City(String cityname) {
this.cityname = cityname;
this.population = 10000;
this.budget = 500000.0;
this.powersupply = 1000;
this.watersupply = 800;
}
public void increasepopulation(int amount)
{
population += amount;
budget -= amount * 50;
}
public void decreasepopulation(int amount)
{
population -= amount;        if
(population < 0) population = 0;
}
public void adjustbudget(double amount) {
budget += amount;
```

```
}
public void buildpowerplant(int capacity) {
powersupply += capacity;
budget -= capacity * 1000;
}
public void buildwaterfacility(int capacity) {
watersupply += capacity;
budget -= capacity * 800;
}
public String getcitystats() {
return cityname + " - Pop: " + population +
", Budget: $" + budget + ", Power: " +
powersupply + ", Water: " + watersupply;
}
public void collecttaxes(double rate) {
budget += population * rate;
}
public void repairinfrastructure(int cost) {
budget -= cost;
```

```
powersupply += 50;
watersupply += 40;
}
public void simulateday(Scanner scanner) {
System.out.println("Enter population
growth: ");
increasepopulation(scanner.nextInt());
System.out.println("Enter tax rate: ");
collecttaxes(scanner.nextDouble());
}
public boolean issustainable() {
return budget > 0 && powersupply >
population / 10 && watersupply > population
/ 15;
}
public void expandcity(int newpop, int
newpower, int newwater) {
population += newpop;
```

```
powersupply += newpower;
watersupply += newwater;
budget -= (newpop * 50 + newpower * 1000 + newwater * 800);
}

// Added main method to test the City class
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.println("Enter city name: ");
String name = scanner.nextLine();
City city = new City(name);
city.simulateday(scanner);
System.out.println(city.getcitystats());
scanner.close();
}
}
```

```
AmritaVille - Pop: 11000, Budget: $537500.00, Power: 1000, Water: 800
Is the city sustainable? Yes
```

Inheritance Programs

```
// Interface for transactions interface
Transaction {
void deposit(double amount);
void withdraw(double amount);
}

// Parent class for all accounts class
Account {
protected double balance;
protected String accountNumber;

Account(String accountNumber, double
initialBalance) {
```

```
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }
    void displayBalance() {
        System.out.println("Account " + accountNumber + " Balance: $" + balance);
    }
}

// Child class inheriting from Account and implementing Transaction
class SavingsAccount extends Account implements Transaction {    private double interestRate;

    SavingsAccount(String accountNumber, double initialBalance, double interestRate) {
        super(accountNumber, initialBalance);
        this.interestRate = interestRate;
```

```
}
public void deposit(double amount) {
balance += amount;
System.out.println("Deposited $" +
amount + " to Savings Account");
}
public void withdraw(double amount) {
if (balance >= amount) {
balance -= amount;
System.out.println("Withdrawn $" +
amount + " from Savings Account");
} else {
System.out.println("Insufficient balance
in Savings Account");
}
}
void addInterest() {
double interest = balance * interestRate /
100;
```

```
balance += interest;
System.out.println("Interest added: $" +
interest);
}
}

// Another child class inheriting from Account
class CurrentAccount extends Account {
private double overdraftLimit;

CurrentAccount(String accountNumber,
double initialBalance, double overdraftLimit) {
super(accountNumber, initialBalance);
this.overdraftLimit = overdraftLimit;
}
void checkOverdraft() {
System.out.println("Overdraft Limit for "
+ accountNumber + ": $" + overdraftLimit);
}
```

```
}

// Main class to test the banking system
public class BankingSystem {
public static void main(String[] args) {
// Create a savings account
SavingsAccount savings = new
SavingsAccount("SAV123", 1000.0, 5.0);
savings.deposit(500.0);
savings.addInterest();
savings.withdraw(200.0);
savings.displayBalance();

System.out.println();

// Create a current account
CurrentAccount current = new
```

```
CurrentAccount("CUR456", 2000.0, 1000.0);
current.displayBalance();
current.checkOverdraft();
}
}
```

```
Deposited $500.0 to Savings Account
Interest added: $75.0
Withdrawn $200.0 from Savings Account
Account SAV123 Balance: $1375.0

Account CUR456 Balance: $2000.0
Overdraft Limit for CUR456: $1000.0
```

```
// Interface for booking operations
interface Bookable {
void bookTicket();
}

// Interface for payment operations
interface Payable {
void makePayment(double amount);
```

```
}

// Parent class for flights
class Flight {
protected String flightNumber;
protected String destination;

Flight(String flightNumber, String
destination) {
 this.flightNumber = flightNumber;
 this.destination = destination;
}
void displayFlightDetails() {
 System.out.println("Flight " +
flightNumber + " to " + destination);
}
}
```

```
// Child class inheriting from Flight and
implementing Bookable and Payable
class PassengerFlight extends Flight
implements Bookable, Payable {
private int seatsAvailable;     private
double ticketPrice;
PassengerFlight(String
flightNumber, String destination, int
seatsAvailable, double ticketPrice) {
super(flightNumber, destination);
this.seatsAvailable = seatsAvailable;
this.ticketPrice = ticketPrice;
}
public void bookTicket() {
if (seatsAvailable > 0) {
seatsAvailable--;
System.out.println("Ticket booked for
Flight " + flightNumber);
} else {
```

```
System.out.println("No seats available
on Flight " + flightNumber);
}
}
public void makePayment(double amount) {
if (amount >= ticketPrice) {
System.out.println("Payment of $" +
amount + " successful for Flight " +
flightNumber);
} else {
System.out.println("Insufficient
payment for Flight " + flightNumber);
}
}
void checkAvailability() {
System.out.println("Seats available: " +
seatsAvailable);
}
}
```

```
// Main class to test the flight booking system
public class FlightBookingSystem {
    public static void main(String[] args) {
        PassengerFlight flight = new
PassengerFlight("FL123", "New York", 2,
300.0);

        flight.displayFlightDetails();
        flight.checkAvailability();
        flight.bookTicket();
        flight.makePayment(350.0);
        flight.checkAvailability();
        flight.bookTicket();
    flight.bookTicket(); // Should show no
seats available
    }
}
```

```
Flight FL123 to New York
Seats available: 2
Ticket booked for Flight FL123
Payment of $350.0 successful for Flight FL123
Seats available: 1
Ticket booked for Flight FL123
No seats available on Flight FL123
```

```
// Interface for transactions interface
Transaction {
void deposit(double amount);
void withdraw(double amount);
}

// Parent class for all accounts class
Account {
protected double balance;
protected String accountNumber;

Account(String accountNumber, double
initialBalance) {
```

```
this.accountNumber = accountNumber;
this.balance = initialBalance;
}
void displayBalance() {
System.out.println("Account " +
accountNumber + " Balance: $" + balance);
}
}

// Child class inheriting from Account and
implementing Transaction
class SavingsAccount extends Account
implements Transaction {    private
double interestRate;

SavingsAccount(String accountNumber,
double initialBalance, double interestRate) {
super(accountNumber, initialBalance);
this.interestRate = interestRate;
```

```
}
public void deposit(double amount) {
balance += amount;
System.out.println("Deposited $" +
amount + " to Savings Account");
}
public void withdraw(double amount) {
if (balance >= amount) {
balance -= amount;
System.out.println("Withdrawn $" +
amount + " from Savings Account");
} else {
System.out.println("Insufficient balance
in Savings Account");
}
}
void addInterest() {
double interest = balance * interestRate /
100;
```

```
        balance += interest;
        System.out.println("Interest added: $" +
interest);
    }
}

// Another child class inheriting from Account
class CurrentAccount extends Account {
private double overdraftLimit;

    CurrentAccount(String accountNumber,
double initialBalance, double overdraftLimit) {
        super(accountNumber, initialBalance);
        this.overdraftLimit = overdraftLimit;
    }
    void checkOverdraft() {
        System.out.println("Overdraft Limit for "
+ accountNumber + ": $" + overdraftLimit);
    }
```

```
}

// Main class to test the banking system
public class BankingSystem {
public static void main(String[] args) {
// Create a savings account
SavingsAccount savings = new
SavingsAccount("SAV123", 1000.0, 5.0);
savings.deposit(500.0);
savings.addInterest();
savings.withdraw(200.0);
savings.displayBalance();

System.out.println();

// Create a current account
CurrentAccount current = new
```

```
CurrentAccount("CUR456", 2000.0, 1000.0);
current.displayBalance();
current.checkOverdraft();
}
}
```

```
Deposited $500.0 to Savings Account
Interest added: $75.0
Withdrawn $200.0 from Savings Account
Account SAV123 Balance: $1375.0

Account CUR456 Balance: $2000.0
Overdraft Limit for CUR456: $1000.0
```

```
// Interface for booking operations
interface Bookable {
void bookTicket();
}
```

```
// Interface for payment operations
interface Payable {    void
makePayment(double amount);
}

// Parent class for flights class
Flight {
protected String flightNumber;
protected String destination;

Flight(String flightNumber, String
destination) {
this.flightNumber = flightNumber;
this.destination = destination;
}
void displayFlightDetails() {
System.out.println("Flight " +
flightNumber + " to " + destination);
}
```

```
}

// Child class inheriting from Flight and
implementing Bookable and Payable
class PassengerFlight extends Flight
implements Bookable, Payable {
    private int seatsAvailable;     private
    double ticketPrice;

PassengerFlight(String flightNumber, String
destination, int seatsAvailable, double
ticketPrice) {
    super(flightNumber, destination);
this.seatsAvailable = seatsAvailable;
    this.ticketPrice = ticketPrice;
}
public void bookTicket() {
    if (seatsAvailable > 0) {
seatsAvailable--;
```

```
System.out.println("Ticket booked for
Flight " + flightNumber);
} else {
System.out.println("No seats available
on Flight " + flightNumber);
}
}
public void makePayment(double amount) {
if (amount >= ticketPrice) {
System.out.println("Payment of $" +
amount + " successful for Flight " +
flightNumber);
} else {
System.out.println("Insufficient
payment for Flight " + flightNumber);
}
}
void checkAvailability() {
```

```
System.out.println("Seats available: " +
seatsAvailable);
}
}

// Main class to test the flight booking system
public class FlightBookingSystem {
public static void main(String[] args) {
PassengerFlight flight = new
PassengerFlight("FL123", "New York", 2,
300.0);

flight.displayFlightDetails();
flight.checkAvailability();
flight.bookTicket();
flight.makePayment(350.0);
flight.checkAvailability();
flight.bookTicket();
flight.bookTicket(); // Should show no
```

seats available
}
}

```
Flight FL123 to New York
Seats available: 2
Ticket booked for Flight FL123
Payment of $350.0 successful for Flight FL123
Seats available: 1
Ticket booked for Flight FL123
No seats available on Flight FL123
```

// Source code is decompiled from a .class file using FernFlower decompiler.

```
class Animal {
    String name;

    Animal(String var1) {
        this.name = var1;
    }
```

```
String speak() {
return "Animal sound";
}
}
```

```
Dog
Animal sound
```

```
// Source code is decompiled from a .class file
using FernFlower decompiler.
public class AnimalTest {
public AnimalTest() {
}
public static void main(String[] var0) {
Dog var1 = new Dog("Buddy", "Brown",
"Golden Retriever");
String var10001 = var1.name;
System.out.println("Name: " + var10001 + ",
Fur Color: " + var1.getFurColor() + ", Breed:
```

```
" + var1.getBreed());
System.out.println(var1.speak());
}
}
```

```
Name: Buddy, Fur Color: Brown, Breed: Golden Retriever
Woof! Woof!
```

Polymorphism:

```
class Student {
    String name;
    int age;
    String grade;
```

```
Student() {
name = "Unknown";
age = 0;
grade = "Not Assigned";
System.out.println("Default constructor called");
}

Student(String n) {
name = n;
age = 0;
grade = "Not Assigned";
System.out.println("Constructor with name: " + n + " called");
}

Student(String n, int a) {
name = n;
age = a;
```

```
grade = "Not Assigned";
System.out.println("Constructor with name: " + n + " and age: " + a + " called");
}

Student(String n, int a, String g) {
name = n;        age = a;
grade = g;
System.out.println("Constructor with name: " + n + ", age: " + a + ", and grade: " + g + " called");
}

void display() {
System.out.println("Name: " + name + ", Age: " + age + ", Grade: " + grade);
}
}
```

```
public class ConstructorOverloadingDemo {
    public static void main(String[] args) {
        Student s1 = new Student();
        Student s2 = new Student("Alice");
        Student s3 = new Student("Bob", 15);
        Student s4 = new Student("Charlie", 16,
        "A");

    System.out.println("\nStudent Details:");
        s1.display();       s2.display();
        s3.display();       s4.display();
                }
            }
```

```
Default constructor called
Constructor with name: Alice called
Constructor with name: Bob and age: 15 called
Constructor with name: Charlie, age: 16, and grade: A called

Student Details:
Name: Unknown, Age: 0, Grade: Not Assigned
Name: Alice, Age: 0, Grade: Not Assigned
Name: Bob, Age: 15, Grade: Not Assigned
Name: Charlie, Age: 16, Grade: A
```

```
class Animal {
    String name;

    Animal() {
        name = "Unknown";
        System.out.println("Default Animal constructor called");
    }

    Animal(String name) {
        this.name = name;
```

```
System.out.println("Animal constructor
with name: " + name + " called");
}

void sound() {
System.out.println(name + " makes a
sound");
}
}

class Dog extends Animal {
Dog() {
super();
System.out.println("Default Dog
constructor called");
}

Dog(String name) {
super(name);
```

```
System.out.println("Dog constructor with
name: " + name + " called");
}

void sound() {
System.out.println(name + " barks");
}
}

public class SimplePolymorphism {
public static void main(String[] args) {
Animal a1 = new Animal();
Animal a2 = new Animal("Cat");
Dog d1 = new Dog();
Dog d2 = new Dog("Rex");

System.out.println("\nTesting sounds:");
a1.sound();        a2.sound();
d1.sound();        d2.sound();
```

```
System.out.println("\nPolymorphic reference:");
Animal ref = new Dog("Max");
ref.sound();
}
}
```

Method Overloading:

```
public class AreaCalculator {

// Calculate area of a square
public double calculateArea(double side) {
return side * side;
}
```

```
// Calculate area of a rectangle
public double calculateArea(double length,
double width) {        return length * width;
}

// Calculate area of a circle     public double
calculateArea(double radius,
String shape) {
if (shape.equalsIgnoreCase("circle")) {
return Math.PI * radius * radius;
}
return 0;
}

// Calculate area of a triangle
public double calculateArea(double base,
double height, String shape) {        if
(shape.equalsIgnoreCase("triangle")) {
return 0.5 * base * height;
```

```
}
return 0;
}
public static void main(String[] args) {
AreaCalculator calculator = new
AreaCalculator();

System.out.println("Area of square (side
5): " + calculator.calculateArea(5));
System.out.println("Area of rectangle
(6x4): " + calculator.calculateArea(6, 4));
System.out.println("Area of circle (radius
3): " + calculator.calculateArea(3, "circle"));
System.out.println("Area of triangle (base
4, height 7): " +
calculator.calculateArea(4, 7,
"triangle"));
}
}
```

```
Area of square (side 5): 25.0
Area of rectangle (6x4): 24.0
Area of circle (radius 3): 28.274333882308138
Area of triangle (base 4, height 7): 14.0
```

```
public class DatabaseQuery {

    // 1. Search by ID
    public String buildQuery(int id) {
        return "SELECT * FROM users WHERE id = " + id;
    }

    // 2. Search by name (overloaded)
    public String buildQuery(String name) {
        return "SELECT * FROM users WHERE name = '" + name + "'";
    }

    // 3. Search by name and age (overloaded)
    public String buildQuery(String name, int
```

```
age) {
    return "SELECT * FROM users WHERE name = '" + name + "' AND age = " + age;
}
public static void main(String[] args) {
    DatabaseQuery db = new DatabaseQuery();

    System.out.println(db.buildQuery(101)); // Search by ID
    System.out.println(db.buildQuery("Alice")); // Search by name
    System.out.println(db.buildQuery("Bob", 30)); // Search by name & age
}
}
```

```
SELECT * FROM users WHERE id = 101
SELECT * FROM users WHERE name = 'Alice'
SELECT * FROM users WHERE name = 'Bob' AND age = 30
```

Method Overriding: class
BankAccount {
protected double
balance;

```
public BankAccount(double balance) {
        this.balance = balance;
                }

  // Default interest calculation (5%)
public double calculateInterest() {
return balance * 0.05;
                }
              }

class SavingsAccount extends BankAccount {
```

```
public SavingsAccount(double balance) {
        super(balance);
        }

// Override: Higher interest for savings
(7%)
        @Override
    public double calculateInterest() {
        return balance * 0.07;
        }
        }

class FixedDepositAccount extends
        BankAccount {
    public FixedDepositAccount(double
        balance) {
            super(balance);
        }
```

```
// Override: Highest interest for fixed deposits (9%)
@Override
public double calculateInterest() {
    return balance * 0.09;
}
}

public class BankingSystem {
public static void main(String[] args) {
    BankAccount regular = new BankAccount(10000);
    BankAccount savings = new SavingsAccount(10000);
    BankAccount fixedDeposit = new FixedDepositAccount(10000);

    System.out.println("Regular Account Interest: $" + regular.calculateInterest());
```

```
System.out.println("Savings Account Interest: $" + savings.calculateInterest());
System.out.println("Fixed Deposit Interest: $" + fixedDeposit.calculateInterest());
}
}
```

```
Regular Account Interest: $500.0
Savings Account Interest: $700.0
Fixed Deposit Interest: $900.0
```

```
class Product {
protected String name;
protected double price;

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

```
// Default discount (10%)
public double calculateDiscount() {
return price * 0.10;
}
public void display() {
System.out.println("Product: " + name + ",
Price: $" + price);
}
}

class Electronics extends Product {
public Electronics(String name, double
price) {
super(name, price);
}

// Override: Higher discount for electronics
(15%)
@Override
```

```
public double calculateDiscount() {
return price * 0.15;
}
}

class Groceries extends Product {
public Groceries(String name, double price)
{
super(name, price);
}

// Override: Lower discount for groceries
(5%)
@Override
public double calculateDiscount() {
return price * 0.05;
}
}
```

```
public class ECommerceSystem {
    public static void main(String[] args) {
        Product laptop = new
        Electronics("Laptop", 1000);
        Product bread = new Groceries("Bread",
5);
        Product book = new Product("Book", 20);

        laptop.display();
        System.out.println("Discount: $" +
        laptop.calculateDiscount());

        bread.display();
        System.out.println("Discount: $" +
        bread.calculateDiscount());

        book.display();
        System.out.println("Discount: $" +
        book.calculateDiscount());
```

```
    }
}
```

```
Product: Laptop, Price: $1000.0
Discount: $150.0
Product: Bread, Price: $5.0
Discount: $0.25
Product: Book, Price: $20.0
Discount: $2.0
```

User defined Package:

```
// File: GeometryPackageDemo.java

// Package declaration package
geometry;

// Circle class in geometry package class
Circle {
private double radius;
```

```
public Circle(double radius) {
    this.radius = radius;
}
public double getArea() {
return Math.PI * radius * radius;
}
public double getCircumference() {
    return 2 * Math.PI * radius;
}
public void displayInfo() {
System.out.println("Circle with radius: " +
radius);
System.out.println("Area: " + getArea());
System.out.println("Circumference: " +
getCircumference());
}
}
```

```
// Rectangle class in geometry package
class Rectangle {
private double length;
private double width;

public Rectangle(double length, double
width) {
this.length = length;
this.width = width;
}
public double getArea() {
return length * width;
}

public double getPerimeter() {
return 2 * (length + width);
}
public void displayInfo() {
```

```
        System.out.println("Rectangle with length: " + length + " and width: " + width);
        System.out.println("Area: " + getArea());
        System.out.println("Perimeter: " + getPerimeter());
    }
}

// ShapeCalculator utility class in geometry package
class ShapeCalculator {
    public static void printCircleDetails(double radius) {
        Circle circle = new Circle(radius);
        circle.displayInfo();
    }
    public static void printRectangleDetails(double length, double width) {
        Rectangle rectangle = new
```

```
Rectangle(length, width);
rectangle.displayInfo();
}
}

// Main class to demonstrate the package
(outside the geometry package) public
class GeometryPackageDemo {
public static void main(String[] args) {
// Using the geometry package classes
System.out.println("=== Geometry
Package Demo ===");

// Method 1: Using fully qualified names
geometry.Circle myCircle = new
geometry.Circle(5.0);
myCircle.displayInfo();
```

```
        // Method 2: Using import (not needed since we're in same file)
            Rectangle myRectangle = new Rectangle(4.0, 6.0);
        myRectangle.displayInfo();

        // Using the ShapeCalculator utility
            System.out.println("\nUsing ShapeCalculator:");
        ShapeCalculator.printCircleDetails(3.0);
        ShapeCalculator.printRectangleDetails(5.0, 7.0);
    }
}
```

```
=== Geometry Package Demo ===
Circle with radius: 5.0
Area: 78.53981633974483
Circumference: 31.41592653589793
Rectangle with length: 4.0 and width: 6.0
Area: 24.0
Perimeter: 20.0

Using ShapeCalculator:
Circle with radius: 3.0
Area: 28.274333882308138
Circumference: 18.84955592153876
Rectangle with length: 5.0 and width: 7.0
Area: 35.0
Perimeter: 24.0
```

```
Default Animal constructor called
Animal constructor with name: Cat called
Default Animal constructor called
Default Dog constructor called
Animal constructor with name: Rex called
Dog constructor with name: Rex called

Testing sounds:
Unknown makes a sound
Cat makes a sound
Unknown barks
Rex barks

Polymorphic reference:
Animal constructor with name: Max called
Dog constructor with name: Max called
Max barks
```