



SCHOOL OF
COMPUTING

LAB RECORD

23CSE111- Object Oriented Programming

Submitted by

CH.SC.U4CSE24102 - Agneay B Nair

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.U4CSE24102 Agneay B Nair** 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.	ATM	
	1.a) Sequence Diagram	6
	1.b) Use Case Diagram	7
	1.c) Class	8
	1.d) State Chart Diagram	8
2.	HOSPITAL	
	2.a) Use Case Diagram	9
	2.d) State Chart Diagram	9
	2.b) Class Diagram	10
	2.c) Sequence Diagram	11
3.	BASIC JAVA PROGRAMS	
	3.a) Calculator	11
	3.h) Largest Of Three NumS	12
	3.i) Fibonacci	12
	3.j) Even Odd	12
	3.g) Reverse String	13
	3.b) Prime Checker	13
	3.c) Palindrome Checker	14
	3.d) Simple Interest Calculator	14
	3.f) Temperature Converter	15
	3.e) Sum Of Digits	15
	INHERITANCE	
4.	SINGLE INHERITANCE PROGRAMS	
	4.a) Who let the dog bark ?	17
	4.b) Doors of Inheritance: When Cars Speak	19
5.	MULTILEVEL INHERITANCE PROGRAMS	
	5.a) Class of Classes – Person to PhD	21

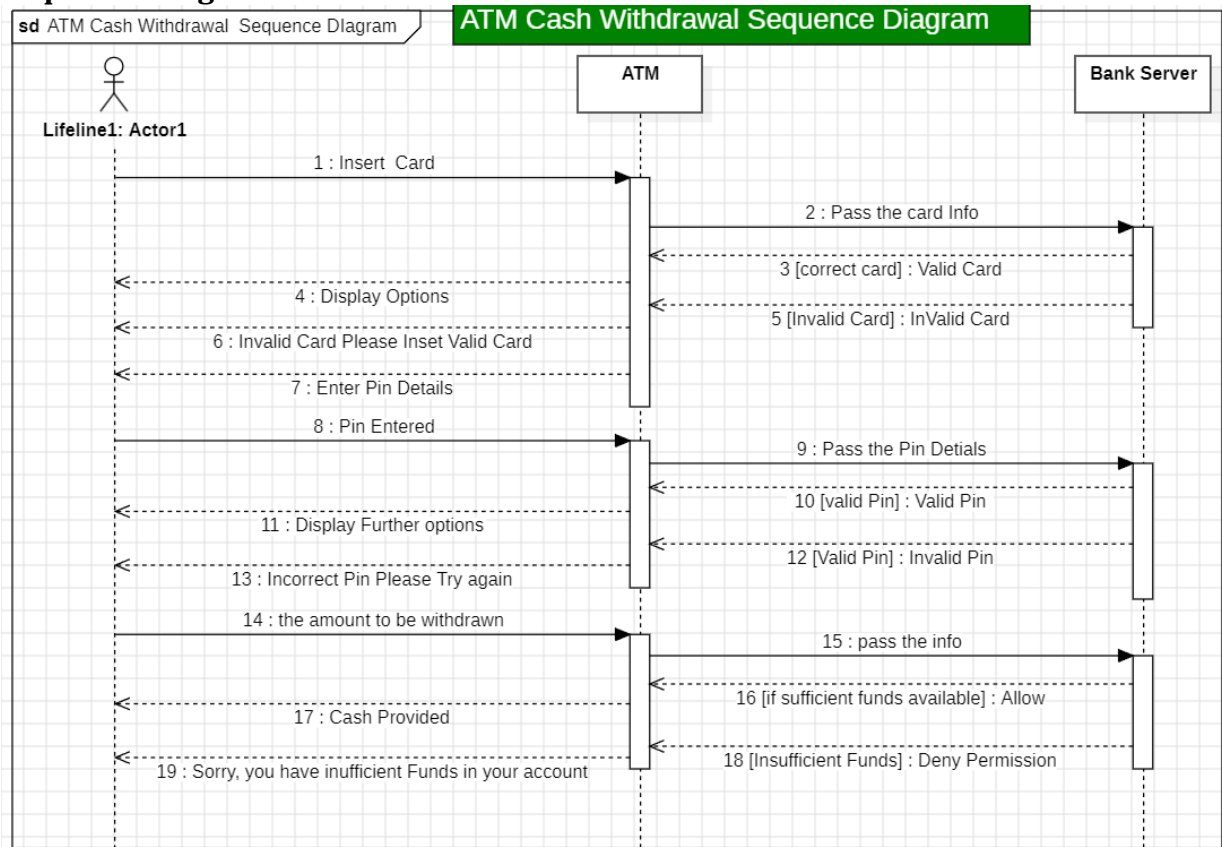
	5.b) The Org Chart Challenge	25
6.	HIERARCHICAL INHERITANCE PROGRAMS	
	6.a) Doggo Details – When Breeds Inherit	30
	6.b) Class Roads – Car or Cargo?	33
7.	HYBRID INHERITANCE PROGRAMS	
	7.a) Hybrid Havoc	38
	7.b) Two Roads Converged	42
	POLYMORPHISM	
8.	CONSTRUCTOR PROGRAMS	
	8.a) Constructor Chronicles	45
9.	CONSTRUCTOR OVERLOADING PROGRAMS	
	9.a) Constructor Cascade	47
10.	METHOD OVERLOADING PROGRAMS	
	10.a) Overload Overdrive	50
	10.b) Shape Shifter – Overloading Area Calculation	53
11.	METHOD OVERRIDING PROGRAMS	
	11.a) Woof and Roar	56
	11.b) Bank Interest	57
	ABSTRACTION	
12.	INTERFACE PROGRAMS	
	12.a) Animal Analyzer	67
	12.b) Shape Factory	68
	12.c) Vehicle startup	69
	12.d) Drivable Bikes	71
13.	ABSTRACT CLASS PROGRAMS	
	13.a) Person Mania	60
	13.b) Employee Rush	61
	13.c) Bank Account	63
	13.d) Student	65
	ENCAPSULATION	
14.	ENCAPSULATION PROGRAMS	
	14.a) Data Vault	74
	14.b) Age Validator	75
	14.c) Bank Secure	76
	14.d) Employee Records	77
15.	PACKAGES PROGRAMS	
	15.a) User Defined Packages	92
	15.b) User Defined Packages	95
	15.c) Built – in Package(3 Packages)	96
	15.d) Built – in Package(3 Packages)	97
16.	EXCEPTION HANDLING PROGRAMS	

	16.a) Zero Trouble	82
	16.b) Out of Bounds	83
	16.c) Finally Finished	84
	16.d) Age Gate Keeper	85
17.	FILE HANDLING PROGRAMS	
	17.a) File Forge	87
	17.b) ReadMe Master	89
	17.c) Append Attack	90
	17.d) File Vanisher	91

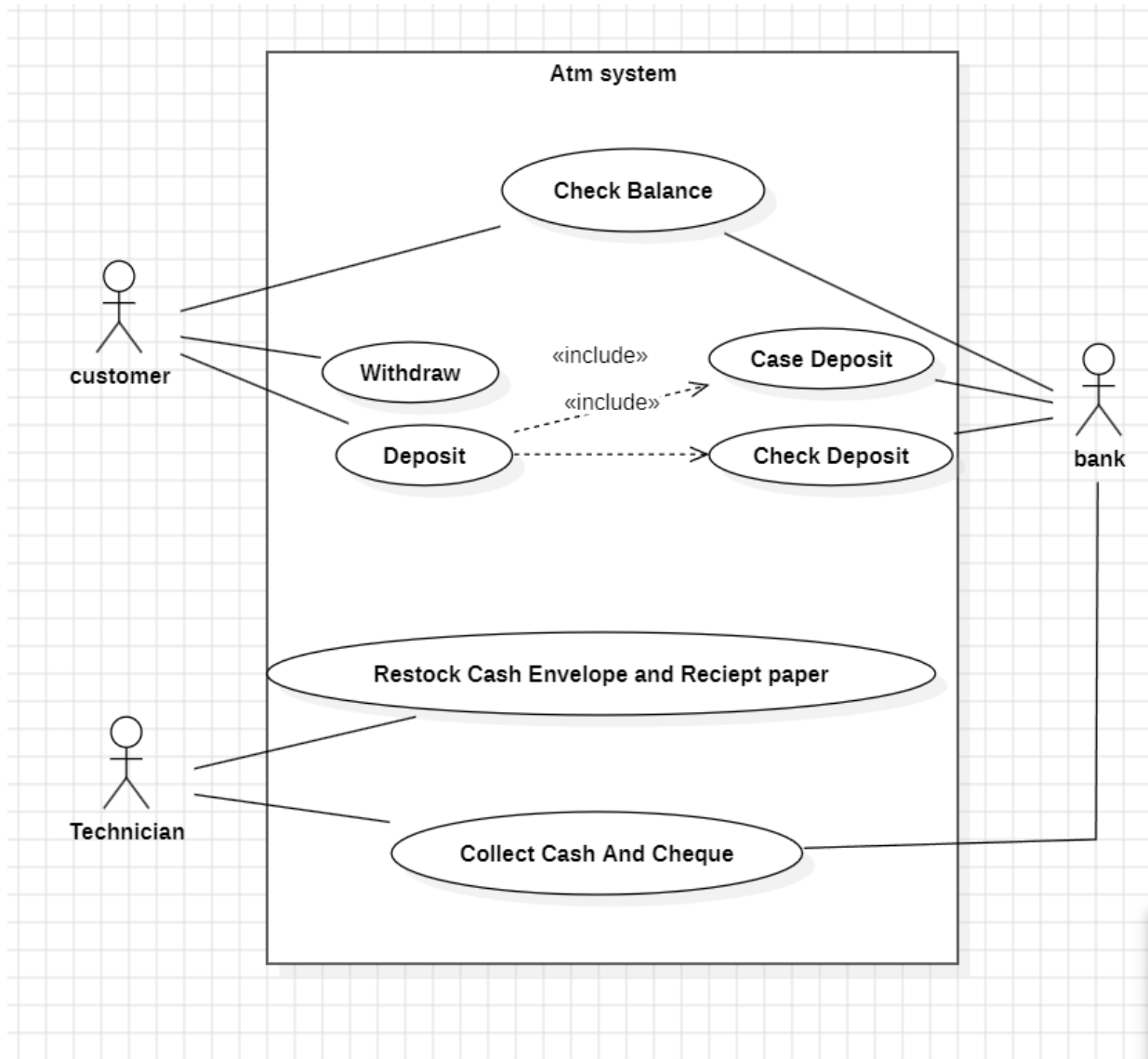
UML Diagrams

1. ATM

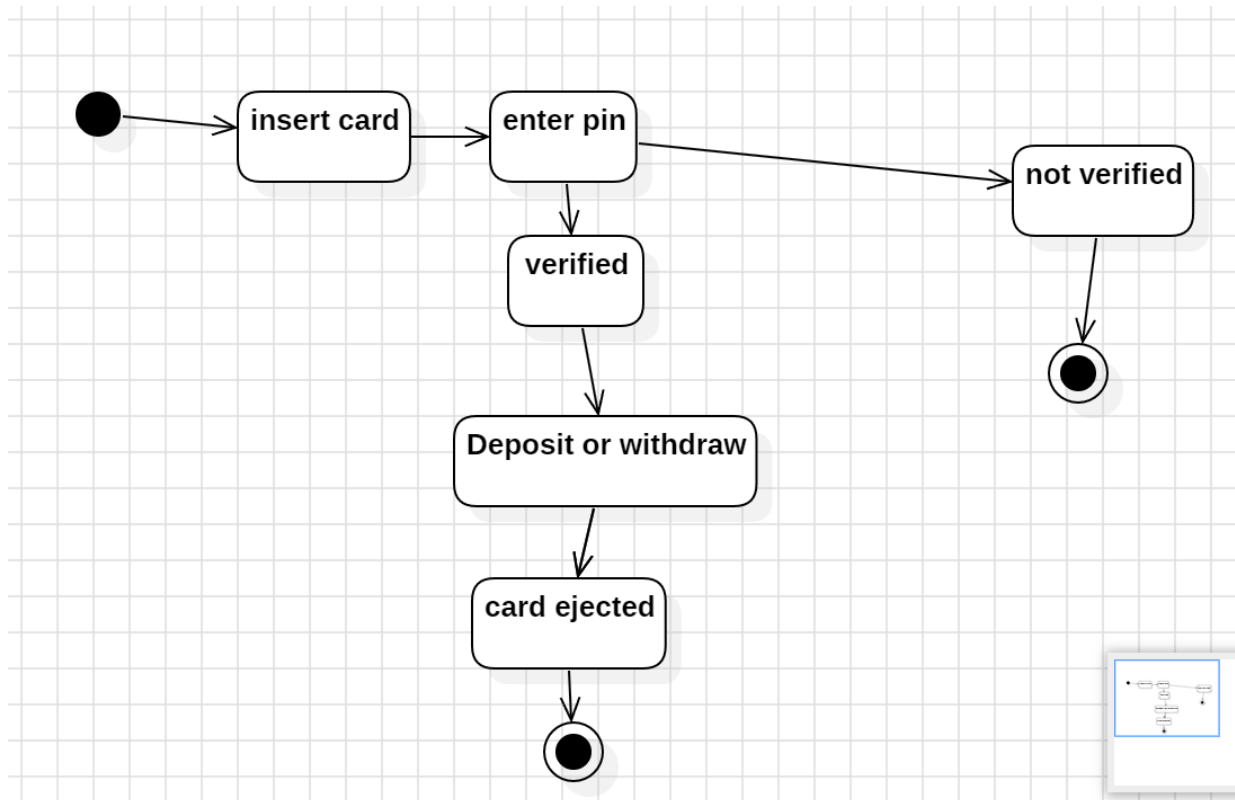
Sequence Diagram



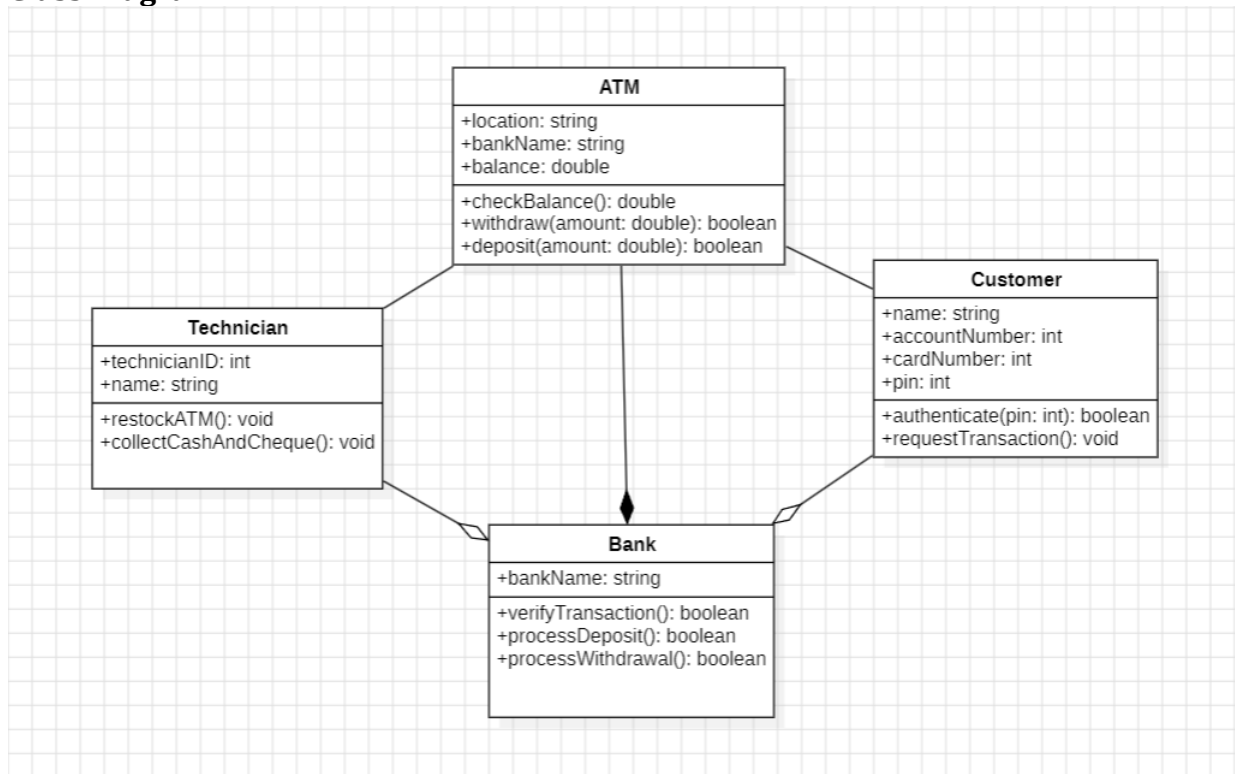
Use Case Diagram



State Chart Diagram

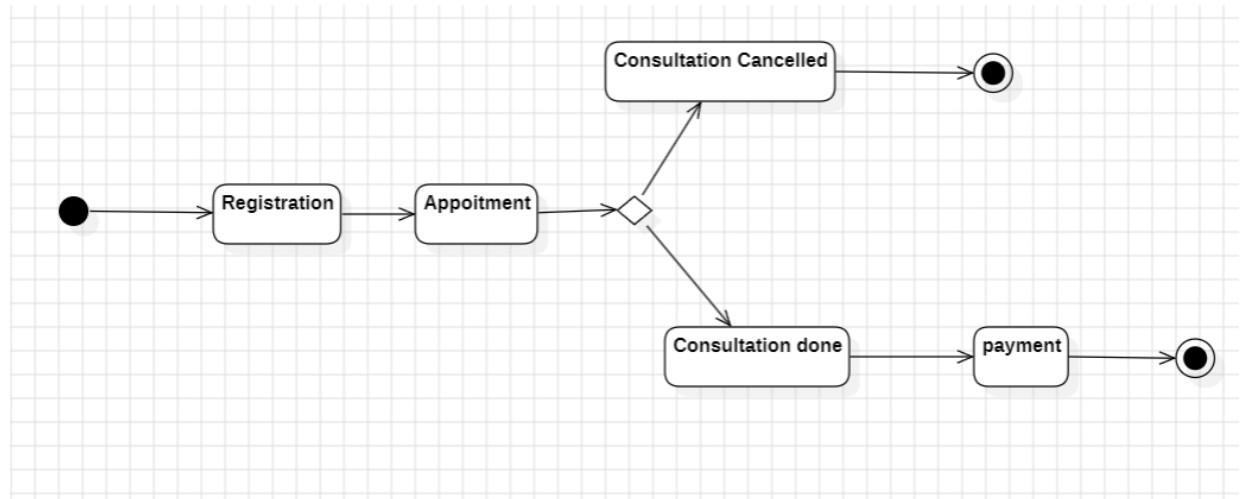


Class Diagram

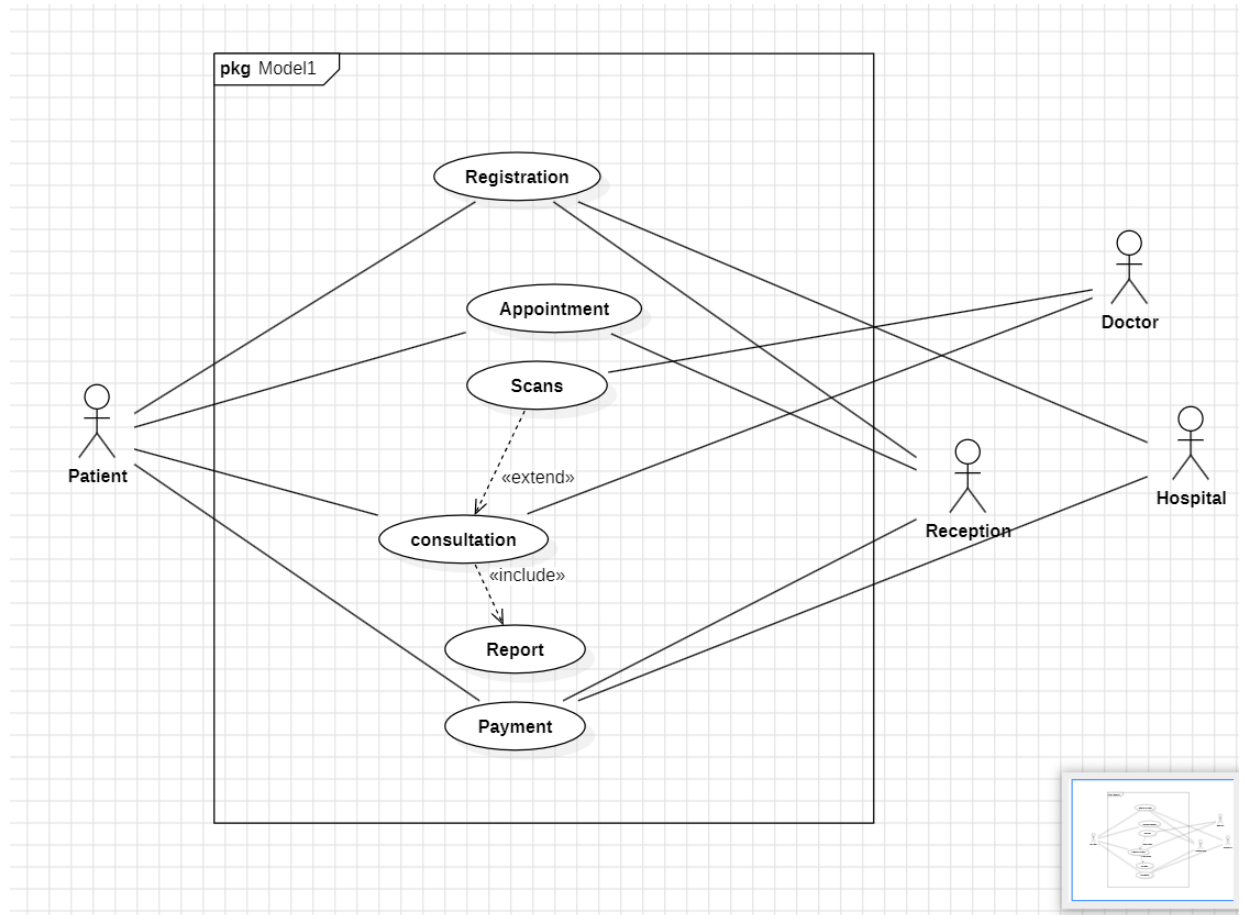


Hospital

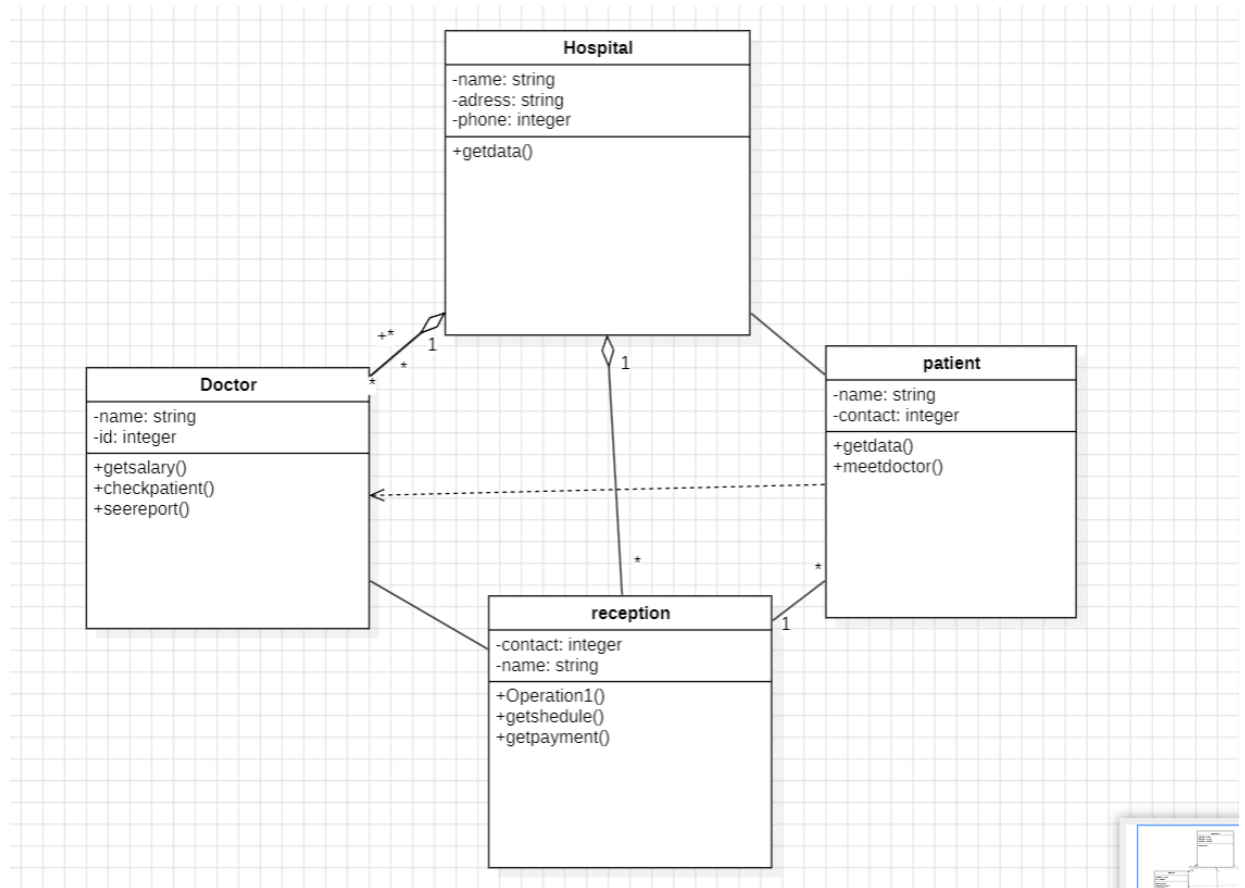
1. StateChart



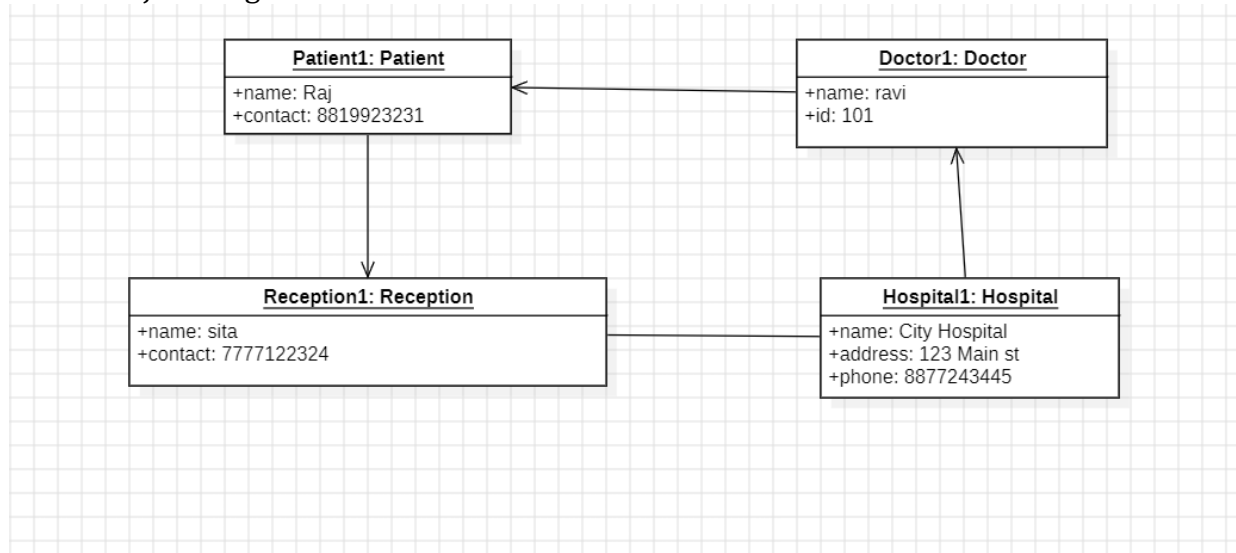
2. UseCase



3. Class Diagram



4. Object Diagram



Java Codes

List of Programs

1. Calculator.java
2. EvenOdd.java
3. Fibonacci.java
4. LargestOfThreeNums.java
5. ReverseString.java
6. PrimeCheck.java
7. PalindromeCheck.java
8. SimpleInterestCalculator.java
9. SumOfDigits.java
10. TemperatureConverter.java

Calculator.java

```
import java.util.Scanner;

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

        System.out.print("Enter two numbers: ");
        double num1 = sc.nextDouble(), num2 = sc.nextDouble();

        System.out.print("Choose an operation (+, -, *, /): ");
        char op = sc.next().charAt(0);

        switch (op) {
            case '+':
                System.out.println("Result: " + (num1 + num2));
                break;
            case '-':
                System.out.println("Result: " + (num1 - num2));
                break;
            case '*':
                System.out.println("Result: " + (num1 * num2));
                break;
            case '/':
                if (num2 != 0)
                    System.out.println("Result: " + (num1 / num2));
                else
                    System.out.println("Division by zero is not
allowed.");
                break;
            default:
                System.out.println("Invalid operation.");
        }
    }
}
```

```

        sc.close();
    }
}

```

EvenOdd.java

```

import java.util.Scanner;

public class EvenOdd {
    public static void main(String[] args) {
        System.out.println("Input Enter the number:");
        Scanner myScannerObj = new Scanner(System.in);
        int num = myScannerObj.nextInt();
        if (num % 2 == 0) {
            System.out.println("Even");
        } else {
            System.out.println("Odd");
        }
        myScannerObj.close();
    }
}

```

Fibonacci.java

```

import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of terms: ");
        int n = sc.nextInt();

        int a = 0, b = 1, count = 0;
        System.out.print("Fibonacci Series: ");
        while (count < n) {
            System.out.print(a + " ");
            int temp = a + b;
            a = b;
            b = temp;
            count++;
        }
        sc.close();
    }
}

```

LargestOfThreeNums

```

import java.util.Scanner;

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

```

```

        System.out.print("Enter three numbers: ");
        int a = sc.nextInt(), b = sc.nextInt(), c = sc.nextInt();

        if (a > b && a > c) {
            System.out.println(a + " is the largest.");
        } else if (b > c) {
            System.out.println(b + " is the largest.");
        } else {
            System.out.println(c + " is the largest.");
        }
        sc.close();
    }
}

```

ReverseString.java

```

// Description: A program that reverses a string.
import java.util.Scanner;

public class ReverseString {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = sc.nextLine();

        String reversed = "";
        for (int i = str.length() - 1; i >= 0; i--) {
            reversed += str.charAt(i);
        }

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

```

PrimeCheck.java

```

import java.util.Scanner;

public class PrimeCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int n = sc.nextInt();

        boolean isPrime = true;
        if (n <= 1)
            isPrime = false;

        for (int i = 2; i <= Math.sqrt(n); i++) {

```

```

        if (n % i == 0) {
            isPrime = false;
            break;
        }
    }

    System.out.println(n + " is " + (isPrime ? "Prime" : "Not
Prime"));
    sc.close();
}
}

```

PalindromeCheck.java

```

import java.util.Scanner;

public class PalindromeCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        int original = num, reversed = 0;
        while (num != 0) {
            int digit = num % 10;
            reversed = reversed * 10 + digit;
            num /= 10;
        }

        System.out.println(original + (original == reversed ? " is "
: " is not ") + "a palindrome.");
        sc.close();
    }
}

```

SimpleInterestCalculator.java

```

import java.util.Scanner;

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

        // Input
        System.out.print("Enter Principal amount: ");
        double principal = sc.nextDouble();

        System.out.print("Enter Rate of Interest (%): ");
        double rate = sc.nextDouble();
    }
}

```

```

    System.out.print("Enter Time (in years): ");
    double time = sc.nextDouble();

    // Calculation
    double simpleInterest = (principal * rate * time) / 100;

    // Output
    System.out.println("Simple Interest: " + simpleInterest);
    System.out.println("Total Amount: " + (principal +
simpleInterest));
    sc.close();
}
}

```

SumOfDigits.java

```

import java.util.Scanner;

public class SumOfDigits {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        int sum = 0;
        while (num != 0) {
            sum += num % 10;
            num /= 10;
        }

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

```

TemperatureConverter.java

```

import java.util.Scanner;

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

        // Display menu
        System.out.println("Choose conversion type:");
        System.out.println("1. Celsius to Fahrenheit");
        System.out.println("2. Fahrenheit to Celsius");

        // Choice input
        System.out.print("Enter your choice (1 or 2): ");
        int choice = sc.nextInt();
    }
}

```

```
    if (choice == 1) {
        // Celsius to Fahrenheit
        System.out.print("Enter temperature in Celsius: ");
        double celsius = sc.nextDouble();
        double fahrenheit = (9.0 / 5.0) * celsius + 32;
        System.out.println("Temperature in Fahrenheit: " +
fahrenheit);
    } else if (choice == 2) {
        // Fahrenheit to Celsius
        System.out.print("Enter temperature in Fahrenheit: ");
        double fahrenheit = sc.nextDouble();
        double celsius = (5.0 / 9.0) * (fahrenheit - 32);
        System.out.println("Temperature in Celsius: " + celsius);
    } else {
        System.out.println("Invalid choice. Please enter 1 or
2.");
    }
    sc.close();
}
```


Inheritance

a. Single Inheritance Program

First program

```
// Superclass (Parent class)
class Animal {
    // Property of the superclass
    String name;

    // Constructor
    public Animal(String name) {
        this.name = name;
    }

    // Method of the superclass
    public void makeSound() {
        System.out.println("The
animal makes a sound.");
    }
}

// Subclass (Child class) inherits
from Animal
class Dog extends Animal {
    // Constructor of subclass
    public Dog(String name) {
        super(name); // Calling the
superclass constructor
    }
}
```

```
    }

    // Overriding the makeSound
method
    @Override
    public void makeSound() {
        System.out.println("The dog
barks.");
    }
}

// Main class to run the code
public class Main {
    public static void main(String[]
args) {
        Animal myAnimal = new
Animal("Generic Animal");
        myAnimal.makeSound();

        Dog myDog = new Dog("Buddy");
        myDog.makeSound(); // This
will use the overridden method
    }
}
```

Output:

The animal makes a sound.
The dog barks.

Second

```
// Superclass (Parent class)
class Vehicle {
    // Property of superclass
    String brand;

    // Constructor of superclass
    public Vehicle(String brand) {
        this.brand = brand;
    }

    // Method of superclass
    public void start() {
        System.out.println(brand + "
is starting.");
    }
}

// Subclass (Child class) inherits
from Vehicle
class Car extends Vehicle {
    // Property specific to Car
```

```
int doors;

// Constructor of subclass
public Car(String brand, int
doors) {
    super(brand); // Calling the
superclass constructor
    this.doors = doors;
}

// Overriding the start method
@Override
public void start() {
    System.out.println(brand + "
car with " + doors + " doors is
starting.");
}
}

// Main class to run the code
public class Main {
    public static void main(String[]
args) {
        Vehicle myVehicle = new
Vehicle("Toyota");
        myVehicle.start();
    }
}
```

```
        Car myCar = new Car("Honda",  
4);  
        myCar.start(); // This will  
use the overridden method  
    }  
}
```

Output:

```
Toyota is starting.  
Honda car with 4 doors is starting.
```

b. MultiLevel Inheritance

First

```
// Base class - Person  
class Person {  
    String name;  
    int age;  
  
    // Constructor  
    public Person(String name, int  
age) {  
        this.name = name;  
        this.age = age;  
    }  
}
```

```
// Method to display person
details
public void
displayPersonDetails() {
    System.out.println("Name: " +
name);
    System.out.println("Age: " +
age);
}
}

// Derived class 1 - Student
(inherits from Person)
class Student extends Person {
    String course;

    // Constructor
    public Student(String name, int
age, String course) {
        super(name, age); // Call
the constructor of Person
        this.course = course;
    }

    // Method to display student
details
```

```
        public void
displayStudentDetails() {
            displayPersonDetails();    //
Call method from Person class
            System.out.println("Course: "
+ course);
        }
    }

// Derived class 2 - GraduateStudent
(inherits from Student)
class GraduateStudent extends Student
{
    String thesisTitle;

    // Constructor
    public GraduateStudent(String
name, int age, String course, String
thesisTitle) {
        super(name, age, course);    //
Call the constructor of Student
        this.thesisTitle =
thesisTitle;
    }
}
```

```
// Method to display graduate
student details
public void
displayGraduateStudentDetails() {
    displayStudentDetails(); //
Call method from Student class
    System.out.println("Thesis
Title: " + thesisTitle);
}
}

// Main class to run the program
public class Main {
    public static void main(String[]
args) {
        // Create an object of
GraduateStudent (three-level
inheritance)
        GraduateStudent gradStudent =
new GraduateStudent("Alice", 25,
"Computer Science", "AI in
Healthcare");

        // Call method to display
details from all classes
```



```
        gradStudent.displayGraduateSt  
udentDetails(); // Displays all  
details  
    }  
}
```

Output:

```
Name: Alice  
Age: 25  
Course: Computer Science  
Thesis Title: AI in Healthcare
```

Second

```
// Base class - Person  
class Person {  
    String name;  
    int age;  
  
    // Constructor  
    public Person(String name, int  
age) {  
        this.name = name;  
        this.age = age;  
    }  
}
```

```
// Method to display person
details
public void
displayPersonDetails() {
    System.out.println("Name: " +
name);
    System.out.println("Age: " +
age);
}
}

// Derived class 1 - Employee
(inherits from Person)
class Employee extends Person {
    String position;

    // Constructor
    public Employee(String name, int
age, String position) {
        super(name, age); // Call
the constructor of Person
        this.position = position;
    }

    // Method to display employee
details
```

```
        public void
displayEmployeeDetails() {
            displayPersonDetails(); //
Call method from Person class
            System.out.println("Position:
" + position);
        }
    }

// Derived class 2 - Manager
(inherits from Employee)
class Manager extends Employee {
    String department;

    // Constructor
    public Manager(String name, int
age, String position, String
department) {
        super(name, age,
position); // Call the constructor
of Employee
        this.department = department;
    }

    // Method to display manager
details
```

```
        public void
displayManagerDetails() {
            displayEmployeeDetails(); //
Call method from Employee class
            System.out.println("Departmen
t: " + department);
        }
    }

// Main class to run the program
public class Main {
    public static void main(String[]
args) {
        // Create an object of
Manager (three-level inheritance)
        Manager manager = new
Manager("John", 35, "Senior Manager",
"Sales");

        // Call method to display
details from all classes
        manager.displayManagerDetails
(); // Displays all details
    }
}
```

Output:

```
Name: John  
Age: 35  
Position: Senior Manager  
Department: Sales
```

c. Hierarchical Inheritance

First

```
// Base class - Animal
class Animal {
    String name;
    int age;

    // Constructor
    public Animal(String name, int
age) {
        this.name = name;
        this.age = age;
    }

    // Method to display animal
details
    public void displayDetails() {
        System.out.println("Name: " +
name);
        System.out.println("Age: " +
age);
    }
}

// Derived class 1 - Dog (inherits
from Animal)
class Dog extends Animal {
```

```
String breed;

// Constructor
public Dog(String name, int age,
String breed) {
    super(name, age); // Call
the constructor of Animal
    this.breed = breed;
}

// Method to display dog details
public void displayDogDetails() {
    displayDetails(); // Call
method from Animal class
    System.out.println("Breed: "
+ breed);
}
}

// Derived class 2 - Cat (inherits
from Animal)
class Cat extends Animal {
    String color;

    // Constructor
```

```
    public Cat(String name, int age,
String color) {
        super(name, age); // Call
the constructor of Animal
        this.color = color;
    }

    // Method to display cat details
    public void displayCatDetails() {
        displayDetails(); // Call
method from Animal class
        System.out.println("Color: "
+ color);
    }
}

// Main class to run the program
public class Main {
    public static void main(String[]
args) {
        // Create an object of Dog
        Dog dog = new Dog("Buddy", 3,
"Golden Retriever");
        // Create an object of Cat
        Cat cat = new Cat("Whiskers",
2, "Black");
    }
}
```



```
        // Call methods to display
details
        System.out.println("Dog
Details:");
        dog.displayDogDetails(); //
Displays dog details

        System.out.println("\nCat
Details:");
        cat.displayCatDetails(); //
Displays cat details
    }
}
```

Output:

```
Cat Details:
Name: Whiskers
Age: 2
Color: Black
```

Second

```
// Base class - Vehicle
class Vehicle {
    String brand;
    int year;
```

```
// Constructor
public Vehicle(String brand, int
year) {
    this.brand = brand;
    this.year = year;
}

// Method to display vehicle
details
public void displayDetails() {
    System.out.println("Brand: "
+ brand);
    System.out.println("Year: " +
year);
}
}

// Derived class 1 - Car (inherits
from Vehicle)
class Car extends Vehicle {
    int numberOfDoors;

    // Constructor
    public Car(String brand, int
year, int numberOfDoors) {
```

```
        super(brand, year); // Call
the constructor of Vehicle
        this.numberOfDoors =
numberOfDoors;
    }

    // Method to display car details
    public void displayCarDetails() {
        displayDetails(); // Call
method from Vehicle class
        System.out.println("Number of
Doors: " + numberOfDoors);
    }
}

// Derived class 2 - Truck (inherits
from Vehicle)
class Truck extends Vehicle {
    int loadCapacity;

    // Constructor
    public Truck(String brand, int
year, int loadCapacity) {
        super(brand, year); // Call
the constructor of Vehicle
```

```
        this.loadCapacity =  
loadCapacity;  
    }  
  
    // Method to display truck  
details  
    public void displayTruckDetails()  
{  
        displayDetails(); // Call  
method from Vehicle class  
        System.out.println("Load  
Capacity: " + loadCapacity + "  
tons");  
    }  
}  
  
// Main class to run the program  
public class Main {  
    public static void main(String[]  
args) {  
        // Create an object of Car  
        Car car = new Car("Toyota",  
2021, 4);  
        // Create an object of Truck  
        Truck truck = new  
Truck("Ford", 2018, 10);
```

```
        // Call methods to display
details
        System.out.println("Car
Details:");
        car.displayCarDetails(); //
Displays car details

        System.out.println("\nTruck
Details:");
        truck.displayTruckDetails();
// Displays truck details
    }
}
```

Output:

Car Details:

Brand: Toyota

Year: 2021

Number of Doors: 4

Truck Details:

Brand: Ford

Year: 2018

Load Capacity: 10 tons

d. Hybrid Inheritance

First

```
// Base class - Person
class Person {
    String name;
    int age;

    // Constructor
    public Person(String name, int
age) {
        this.name = name;
        this.age = age;
    }
}
```

```
// Method to display person
details
public void
displayPersonDetails() {
    System.out.println("Name: " +
name);
    System.out.println("Age: " +
age);
}
}

// Interface 1 - Employee (related to
employee duties)
interface Employee {
    void work(); // Abstract method
}

// Interface 2 - Manager (related to
managerial duties)
interface Manager {
    void manage(); // Abstract
method
}
```

```
// Derived class - ManagerEmployee
(inherits from Person, implements
Employee and Manager interfaces)
class ManagerEmployee extends Person
implements Employee, Manager {
    String department;

    // Constructor
    public ManagerEmployee(String
name, int age, String department) {
        super(name, age); // Call
the constructor of Person
        this.department = department;
    }

    // Implement method from Employee
interface
    public void work() {
        System.out.println(name + "
is working in the department: " +
department);
    }

    // Implement method from Manager
interface
    public void manage() {
```



```
        System.out.println(name + "
is managing the team in the " +
department + " department.");
    }

    // Method to display manager-
employee details
    public void
displayManagerEmployeeDetails() {
        displayPersonDetails(); //
Call method from Person class
        System.out.println("Departmen
t: " + department);
        work(); // Call work method
from Employee interface
        manage(); // Call manage
method from Manager interface
    }
}

// Main class to run the program
public class Main {
    public static void main(String[]
args) {
        // Create an object of
ManagerEmployee (hybrid inheritance)
```

```
        ManagerEmployee manager = new
ManagerEmployee("John", 35, "Sales");

        // Call method to display
details
        manager.displayManagerEmployeeDetails(); // Displays all details
    }
}
```

Output:

```
Name: John
Age: 35
Department: Sales
John is working in the department: Sales
John is managing the team in the Sales department.
```

Second

```
// Base class
class A {
    void displayA() {
        System.out.println("Class A:
Base Class");
    }
}

// Derived class B inheriting from A
(Single Inheritance)
```

```
class B extends A {
    void displayB() {
        System.out.println("Class B:
Derived from A");
    }
}

// Interface C
interface C {
    void displayC();
}

// Class D inheriting from B and
implementing interface C (Hybrid
Inheritance)
class D extends B implements C {
    public void displayC() {
        System.out.println("Class C:
Implemented in D");
    }

    void displayD() {
        System.out.println("Class D:
Derived from B and implements C");
    }
}
```

```
// Main class
public class Main {
    public static void main(String[]
args) {
        D obj = new D();
        obj.displayA(); // From Class
A
        obj.displayB(); // From Class
B
        obj.displayC(); // From
Interface C (Implemented in D)
        obj.displayD(); // From Class
D
    }
}
```

Output:

```
Class A: Base Class
Class B: Derived from A
Class C: Implemented in D
Class D: Derived from B and implements C
```

Polymorphism

a. Constructor Programs

First

```
// Class with Constructors
class Student {
```

```
String name;
int age;

// Default Constructor (No
parameters)
Student() {
    System.out.println("Default
Constructor called.");
    name = "Unknown";
    age = 0;
}

// Parameterized Constructor
Student(String n, int a) {
    System.out.println("Parameter
ized Constructor called.");
    name = n;
    age = a;
}

// Copy Constructor
Student(Student s) {
    System.out.println("Copy
Constructor called.");
    name = s.name;
    age = s.age;
```

```
    }

    // Display Method
    void display() {
        System.out.println("Name: " +
name + ", Age: " + age);
    }
}

// Main Class
public class Main {
    public static void main(String[]
args) {
        // Using Default Constructor
        Student s1 = new Student();
        s1.display();

        // Using Parameterized
Constructor
        Student s2 = new
Student("Alice", 20);
        s2.display();

        // Using Copy Constructor
        Student s3 = new Student(s2);
        s3.display();
    }
}
```

```
}  
}
```

Output:

```
Default Constructor called.  
Name: Unknown, Age: 0  
Parameterized Constructor called.  
Name: Alice, Age: 20  
Copy Constructor called.  
Name: Alice, Age: 20
```

b. Constructor Overloading Programs

First

```
// Class with Constructor Overloading  
class Person {  
    String name;  
    int age;  
    String city;  
  
    // Constructor 1: No parameters  
    (Default Constructor)  
    Person() {  
        System.out.println("Default  
Constructor called.");  
        name = "Unknown";  
        age = 0;  
        city = "Not Specified";  
    }  
}
```

```
// Constructor 2: Parameterized
Constructor (name & age)
Person(String n, int a) {
    System.out.println("Construct
or with Name & Age called.");
    name = n;
    age = a;
    city = "Not Specified";
}

// Constructor 3: Parameterized
Constructor (name, age & city)
Person(String n, int a, String c)
{
    System.out.println("Construct
or with Name, Age & City called.");
    name = n;
    age = a;
    city = c;
}

// Display Method
void display() {
    System.out.println("Name: " +
name + ", Age: " + age + ", City: " +
city);
}
```



```
    }  
}  
  
// Main Class  
public class  
ConstructorOverloadingExample {  
    public static void main(String[]  
args) {  
        // Using Default Constructor  
        Person p1 = new Person();  
        p1.display();  
  
        // Using Constructor with  
Name & Age  
        Person p2 = new  
Person("Alice", 25);  
        p2.display();  
  
        // Using Constructor with  
Name, Age & City  
        Person p3 = new Person("Bob",  
30, "New York");  
        p3.display();  
    }  
}
```

Output:

```
Default Constructor called.  
Name: Unknown, Age: 0, City: Not Specified  
Constructor with Name & Age called.  
Name: Alice, Age: 25, City: Not Specified  
Constructor with Name, Age & City called.  
Name: Bob, Age: 30, City: New York
```

c. Method Overloading Programs

First

```
// Calculator class demonstrating  
method overloading  
class Calculator {  
  
    // Method 1: Addition of two  
integers  
    int add(int a, int b) {  
        System.out.println("Adding  
two integers:");  
        return a + b;  
    }  
  
    // Method 2: Addition of three  
integers  
    int add(int a, int b, int c) {  
        System.out.println("Adding  
three integers:");  
        return a + b + c;  
    }  
}
```

```
// Method 3: Addition of two
double values
    double add(double a, double b) {
        System.out.println("Adding
two double values:");
        return a + b;
    }

// Method 4: Concatenation of two
strings
    String add(String a, String b) {
        System.out.println("Concatena
ting two strings:");
        return a + b;
    }
}

// Main class
public class MethodOverloadingDemo {
    public static void main(String[]
args) {
        Calculator calc = new
Calculator();
```

```
        // Calling different
overloaded methods
        System.out.println(calc.add(5
, 10));           // Two integers
        System.out.println(calc.add(3
, 7, 2));         // Three integers
        System.out.println(calc.add(4
.5, 2.3));        // Two doubles
        System.out.println(calc.add("
Hello, ", "World!")); // Two Strings
    }
}
```

Output:

```
Adding two integers:
15
Adding three integers:
12
Adding two double values:
6.8
Concatenating two strings:
Hello, World!
```

```
// Shape class demonstrating method
overloading
class Shape {

    // Method 1: Area of a square
    int area(int side) {
        System.out.println("Calculati
ng area of square:");
        return side * side;
    }

    // Method 2: Area of a rectangle
    int area(int length, int width) {
        System.out.println("Calculati
ng area of rectangle:");
        return length * width;
    }

    // Method 3: Area of a circle
    (double parameter)
    double area(double radius) {
        System.out.println("Calculati
ng area of circle:");
        return Math.PI * radius *
radius;
    }
}
```

```
// Method 4: Area of a triangle
double area(double base, double
height) {
    System.out.println("Calculati
ng area of triangle:");
    return 0.5 * base * height;
}

// Main class
public class Main {
    public static void main(String[]
args) {
        Shape shape = new Shape();

        // Calling different
overloaded methods
        System.out.println("Area: " +
shape.area(5)); // Square
        System.out.println("Area: " +
shape.area(4, 6)); // Rectangle
        System.out.println("Area: " +
shape.area(3.5)); // Circle
        System.out.println("Area: " +
shape.area(5.0, 8.0)); // Triangle
```

```
    }  
}
```

Output:

```
Area: 25  
Calculating area of rectangle:  
Area: 24  
Calculating area of circle:  
Area: 38.48451000647496  
Calculating area of triangle:  
Area: 20.0
```

d. Method Overriding

First

```
// Parent class  
class Animal {  
    void makeSound() {  
        System.out.println("Animals  
make sounds");  
    }  
}  
  
// Child class overriding makeSound()  
method  
class Dog extends Animal {  
    @Override  
    void makeSound() {
```

```
        System.out.println("Dog
barks");
    }
}

// Main class
public class Main {
    public static void main(String[]
args) {
        Animal myAnimal = new
Animal(); // Parent class object
        myAnimal.makeSound();

        Dog myDog = new Dog(); //
Child class object
        myDog.makeSound();
    }
}
```

Output:

```
Animals make sounds
Dog barks
```

Second

```
// Parent class: Bank
class Bank {
```



```
    double getInterestRate() {
        return 5.0; // Default
interest rate
    }
}

// Child class 1: SBI overriding
getInterestRate()
class SBI extends Bank {
    @Override
    double getInterestRate() {
        return 6.5; // SBI specific
interest rate
    }
}

// Child class 2: HDFC overriding
getInterestRate()
class HDFC extends Bank {
    @Override
    double getInterestRate() {
        return 7.2; // HDFC specific
interest rate
    }
}
```

```
// Main class
public class Main {
    public static void main(String[]
args) {
        Bank b1 = new Bank();
        System.out.println("Bank
Interest Rate: " +
b1.getInterestRate() + "%");

        Bank b2 = new SBI(); //
Dynamic Method Dispatch
        System.out.println("SBI
Interest Rate: " +
b2.getInterestRate() + "%");

        Bank b3 = new HDFC(); //
Dynamic Method Dispatch
        System.out.println("HDFC
Interest Rate: " +
b3.getInterestRate() + "%");
    }
}
```

Output:

Bank Interest Rate: 5.0%
SBI Interest Rate: 6.5%
HDFC Interest Rate: 7.2%

Abstraction

a. Abstract Class Programs

First

```
// Abstract class
abstract class Animal {
    // Abstract method (without
    implementation)
    abstract void makeSound();

    // Concrete method (with
    implementation)
    void sleep() {
        System.out.println("Animal is
        sleeping...");
    }
}

// Subclass implementing the abstract
method
class Dog extends Animal {
    @Override
```

```
void makeSound() {  
    System.out.println("Dog  
barks: Woof! Woof!");  
}  
}  
  
// Main class  
public class Main {  
    public static void main(String[]  
args) {  
        Dog d = new Dog();  
        d.makeSound(); // Calls the  
implemented method  
        d.sleep(); // Calls the  
inherited concrete method  
    }  
}
```

Output:

```
Dog barks: Woof! Woof!  
Animal is sleeping...
```

Second

```
// Abstract class  
abstract class Shape {  
    // Abstract method
```

```
        abstract void draw();
    }

    // Subclass 1
    class Circle extends Shape {
        @Override
        void draw() {
            System.out.println("Drawing a
Circle");
        }
    }

    // Subclass 2
    class Rectangle extends Shape {
        @Override
        void draw() {
            System.out.println("Drawing a
Rectangle");
        }
    }

    // Main class
    public class Main {
        public static void main(String[]
args) {
            Shape s1 = new Circle();
```

```
        s1.draw(); // Calls the
Circle's implementation

        Shape s2 = new Rectangle();
        s2.draw(); // Calls the
Rectangle's implementation
    }
}
```

Output:

```
Drawing a Circle
Drawing a Rectangle
```

Third

```
// Abstract class
abstract class Vehicle {
    String brand;

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

    // Abstract method
    abstract void start();
}
```

```
}

// Subclass implementing the abstract
method
class Car extends Vehicle {
    Car(String brand) {
        super(brand); // Calling the
parent class constructor
    }

    @Override
    void start() {
        System.out.println("Car is
starting...");
    }
}

// Main class
public class Main {
    public static void main(String[]
args) {
        Car c = new Car("Toyota");
        c.start();
    }
}
```

Output:

Vehicle brand: Toyota
Car is starting...

Fourth

```
// Interface
interface Drivable {
    void drive();
}

// Abstract class implementing the
// interface
abstract class Bike implements
Drivable {
    abstract void fuelType();
}

// Subclass providing implementations
class ElectricBike extends Bike {
    @Override
    void fuelType() {
        System.out.println("Electric
Bike runs on battery.");
    }

    @Override
    public void drive() {
```



```
        System.out.println("Electric  
Bike is being driven.");  
    }  
}  
  
// Main class  
public class Main {  
    public static void main(String[]  
args) {  
        Bike myBike = new  
ElectricBike();  
        myBike.fuelType();  
        myBike.drive();  
    }  
}
```

Output:

```
Electric Bike runs on battery.  
Electric Bike is being driven.
```

b. Interface

First

```
// Define an interface
interface Animal {
    void makeSound(); // Abstract
method
}

// Implementing class
class Dog implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Dog
barks: Woof! Woof!");
    }
}

// Main class
public class Main {
    public static void main(String[]
args) {
        Dog d = new Dog();
        d.makeSound();
    }
}
```

Output:

```
Dog barks: Woof! Woof!
```

Second

```
// First interface
interface Flyable {
    void fly();
}

// Second interface
interface Swimable {
    void swim();
}

// Class implementing both interfaces
class Bird implements Flyable,
Swimable {
    @Override
    public void fly() {
        System.out.println("Bird is
flying.");
    }

    @Override
    public void swim() {
        System.out.println("Bird is
swimming.");
    }
}
```

```
// Main class
public class Main {
    public static void main(String[]
args) {
        Bird b = new Bird();
        b.fly();
        b.swim();
    }
}
```

Output:

```
Bird is flying.
Bird is swimming.
```

Third

```
// Define an interface
interface Vehicle {
    void start();

    // Default method
    default void honk() {
        System.out.println("Vehicle
is honking.");
    }
}
```

```
// Static method
static void show() {
    System.out.println("This is a
Vehicle interface.");
}
}

// Implementing class
class Car implements Vehicle {
    @Override
    public void start() {
        System.out.println("Car is
starting.");
    }
}

// Main class
public class Main {
    public static void main(String[]
args) {
        Car c = new Car();
        c.start();
        c.honk(); // Calling default
method
}
```

```
        Vehicle.show(); // Calling  
static method  
    }  
}
```

Output:

```
Car is starting.  
Vehicle is honking.  
This is a Vehicle interface.
```

Fourth

```
// Define an interface  
interface Payment {  
    void processPayment(int amount);  
}  
  
// Implementing class 1  
class CreditCardPayment implements  
Payment {  
    @Override  
    public void processPayment(int  
amount) {  
        System.out.println("Credit  
Card Payment of $" + amount + "  
processed.");  
    }  
}
```

```
// Implementing class 2
class PayPalPayment implements
Payment {
    @Override
    public void processPayment(int
amount) {
        System.out.println("PayPal
Payment of $" + amount + "
processed.");
    }
}

// Main class
public class Main {
    public static void main(String[]
args) {
        Payment p1 = new
CreditCardPayment();
        p1.processPayment(100);

        Payment p2 = new
PayPalPayment();
        p2.processPayment(200);
    }
}
```



Output:

```
Credit Card Payment of $100 processed.  
PayPal Payment of $200 processed.
```


Encapsulation

a. Encapsulation Programs

First

```
// Encapsulated class
class Person {
    private String name; // Private
variable

    // Setter method
    public void setName(String name)
{
    this.name = name;
}

    // Getter method
    public String getName() {
        return name;
    }
}

// Main class
public class EncapsulationExample1 {
    public static void main(String[]
args) {
        Person p = new Person();
        p.setName("John Doe");
        System.out.println("Person's
Name: " + p.getName());
    }
}
```

```
}  
}
```

Output:

```
Person's Name: John Doe
```

Second

```
// Encapsulated class  
class Student {  
    private int age; // Private  
variable  
  
    // Setter method with validation  
    public void setAge(int age) {  
        if (age > 0) {  
            this.age = age;  
        } else {  
            System.out.println("Invalid Age. Age must be positive.");  
        }  
    }  
  
    // Getter method  
    public int getAge() {  
        return age;  
    }  
}
```

```
}

// Main class
public class EncapsulationExample2 {
    public static void main(String[]
args) {
        Student s = new Student();
        s.setAge(20);
        System.out.println("Student's
Age: " + s.getAge());

        s.setAge(-5); // Invalid age
input
    }
}
```

Output:

```
Student's Age: 20
Invalid Age. Age must be positive.
```

Third

```
// Encapsulated class
class BankAccount {
    private double balance; //
Private variable
```

```
// Constructor to initialize
balance
public BankAccount(double
initialBalance) {
    if (initialBalance > 0) {
        this.balance =
initialBalance;
    }
}

// Getter method
public double getBalance() {
    return balance;
}

// Deposit method
public void deposit(double
amount) {
    if (amount > 0) {
        balance += amount;
        System.out.println("Depos
ited: $" + amount);
    } else {
        System.out.println("Inval
id deposit amount.");
    }
}
```

```
    }

    // Withdraw method with
validation
    public void withdraw(double
amount) {
        if (amount > 0 && amount <=
balance) {
            balance -= amount;
            System.out.println("Withd
rawn: $" + amount);
        } else {
            System.out.println("Insuf
ficient balance or invalid amount.");
        }
    }
}

// Main class
public class EncapsulationExample3 {
    public static void main(String[]
args) {
        BankAccount account = new
BankAccount(500);
        System.out.println("Initial
Balance: $" + account.getBalance());
    }
}
```

```
        account.deposit(200);  
        account.withdraw(100);  
        System.out.println("Final  
Balance: $" + account.getBalance());  
    }  
}
```

Output:

```
Initial Balance: $500.0  
Deposited: $200.0  
Withdrawn: $100.0  
Final Balance: $600.0
```

Fourth

```
// Encapsulated class  
class Employee {  
    private String name;  
    private int empID;  
    private double salary;  
  
    // Constructor  
    public Employee(String name, int  
empID, double salary) {  
        this.name = name;  
        this.empID = empID;  
    }  
}
```

```
        this.salary = salary;
    }

    // Getter methods
    public String getName() {
        return name;
    }

    public int getEmpID() {
        return empID;
    }

    public double getSalary() {
        return salary;
    }

    // Setter method for salary with
validation
    public void setSalary(double
salary) {
        if (salary > 0) {
            this.salary = salary;
        } else {
            System.out.println("Inval
id salary amount.");
        }
    }
}
```

```
    }  
}  
  
// Main class  
public class EncapsulationExample4 {  
    public static void main(String[]  
args) {  
        Employee emp = new  
Employee("Alice", 101, 55000);  
  
        // Accessing details via  
getter methods  
        System.out.println("Employee  
Name: " + emp.getName());  
        System.out.println("Employee  
ID: " + emp.getEmpID());  
        System.out.println("Employee  
Salary: $" + emp.getSalary());  
  
        // Updating salary using  
setter  
        emp.setSalary(60000);  
        System.out.println("Updated  
Salary: $" + emp.getSalary());  
    }  
}
```


Output:

```
Employee Name: Alice
Employee ID: 101
Employee Salary: $55000.0
Updated Salary: $60000.0
```

b. Exception Handling

First

```
public class
ExceptionHandlingExample1 {
    public static void main(String[]
args) {
        try {
            int a = 10, b = 0;
            int result = a / b; //
This will throw ArithmeticException
            System.out.println("Resul
t: " + result);
        } catch (ArithmeticException
e) {
            System.out.println("Error
: Division by zero is not allowed.");
        }
    }
}
```

```
}
```

Output:

```
Error: Division by zero is not allowed.
```

Second

```
public class
ExceptionHandlingExample2 {
    public static void main(String[]
args) {
        try {
            int[] numbers = {1, 2,
3};
                System.out.println(number
s[5]); // Invalid index
        } catch
(ArrayIndexOutOfBoundsException e) {
            System.out.println("Error
: Array index is out of bounds.");
        }
    }
}
```

Output:

```
Error: Array index is out of bounds.
```

Third

```
public class
ExceptionHandlingExample3 {
```

```
public static void main(String[]
args) {
    try {
        System.out.println("Tryin
g to divide...");
        int result = 10 / 2;
        System.out.println("Resul
t: " + result);
    } catch (ArithmeticException
e) {
        System.out.println("Error
: Cannot divide by zero.");
    } finally {
        System.out.println("Execu
tion completed. Cleaning up
resources...");
    }
}
```

Output:

```
Trying to divide...
Result: 5
Execution completed. Cleaning up resources...
```

Fourth

```
// Custom exception class
```

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

// Main class
public class
ExceptionHandlingExample4 {
    // Method that throws an
exception
    static void checkAge(int age)
throws InvalidAgeException {
        if (age < 18) {
            throw new
InvalidAgeException("Age must be 18
or above.");
        } else {
            System.out.println("Acces
s granted.");
        }
    }
}
```

```
public static void main(String[]
args) {
    try {
        checkAge(16); // Invalid
age
    } catch (InvalidAgeException
e) {
        System.out.println("Excep
tion: " + e.getMessage());
    }
}
```

Output:

```
Exception: Age must be 18 or above.
```

c. File Handling

First

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

public class FileHandlingExample1 {
    public static void main(String[]
args) {
        try {
            File file = new
File("example1.txt"); // File object
```


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


        // Writing to the file
        FileWriter writer = new
FileWriter("example1.txt");
        writer.write("Hello, this
is a file handling example in
Java.");

        writer.close();
        System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

Output:

```
File created: example1.txt  
Successfully wrote to the file.
```

 example1.txt ×

Encapsulation > File Handling >  example1.txt

```
1 Hello, this is a file handling example in Java.
```

Second

```
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Scanner;  
  
public class FileHandlingExample2 {  
    public static void main(String[]  
args) {  
        try {  
            File file = new  
File("example1.txt"); // Read the  
same file from Program 1  
            Scanner reader = new  
Scanner(file);  
  
            while  
(reader.hasNextLine()) {  
                String data =  
reader.nextLine();
```

```
        System.out.println(data);
    }
    reader.close();
} catch
(FileNotFoundException e) {
    System.out.println("An
error occurred. File not found.");
    e.printStackTrace();
}
}
}
```

Output:

```
Hello, this is a file handling example in Java.
```

Third

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

public class FileHandlingExample3 {
    public static void main(String[]
args) {
        try {
            FileWriter writer = new
FileWriter("example1.txt", true); //
Append mode
```



```
        writer.write("\nAppending
new content to the file.");
        writer.close();
        System.out.println("Succe
ssfully appended to the file.");
    } catch (IOException e) {
        System.out.println("An
error occurred.");
        e.printStackTrace();
    }
}
}
```

Output:

Successfully appended to the file.

FileHandlingExample2.java × example1.txt ×

Encapsulation > File Handling > example1.txt

```
1 Hello, this is a file handling example in Java.
2 Appending new content to the file.
```

Fourth

```
import java.io.File;

public class FileHandlingExample4 {
    public static void main(String[]
args) {
```

```
File file = new
File("example1.txt"); // File to be
deleted
    if (file.delete()) {
        System.out.println("Delet
ed the file: " + file.getName());
    } else {
        System.out.println("Faile
d to delete the file.");
    }
}
```

Output:

```
Deleted the file: example1.txt
```

Packages Problem

User Defined Packages

First

Folder Structure

```
ProjectFolder/
├── mathutils/
│   ├── Calculator.java
└── MainApp.java
```

Calculator.java

```
package mathutils;

public class Calculator {
```

```
public int add(int a, int b) {  
    return a + b;  
}  
  
public int subtract(int a, int b)  
{  
    return a - b;  
}  
  
public int multiply(int a, int b)  
{  
    return a * b;  
}  
  
public double divide(int a, int  
b) {  
    if (b == 0) {  
        System.out.println("Error  
: Cannot divide by zero.");  
        return 0;  
    }  
    return (double) a / b;  
}
```

```
}
```

MainApp.java

```
import mathutils.Calculator;

public class MainApp {
    public static void main(String[]
args) {
        Calculator calc = new
Calculator();

        System.out.println("Addition:
" + calc.add(10, 5));
        System.out.println("Subtracti
on: " + calc.subtract(10, 5));
        System.out.println("Multiplic
ation: " + calc.multiply(10, 5));
        System.out.println("Division:
" + calc.divide(10, 5));
    }
}
```

Output:

Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0

Second

Folder Structure

```
ProjectFolder/  
├── geometry/  
│   ├── AreaCalculator.java  
│   └── GeometryApp.java
```

GeometryApp.java

```
import geometry.AreaCalculator;  
  
public class GeometryApp {  
    public static void main(String[]  
args) {  
        AreaCalculator ac = new  
AreaCalculator();  
  
        double circle =  
ac.circleArea(7.5);  
        double rectangle =  
ac.rectangleArea(10, 5);
```

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

AreaCalculator.java

```
package geometry;  
  
public class AreaCalculator {  
    public double circleArea(double  
radius) {  
        return Math.PI * radius *  
radius;  
    }  
  
    public double  
rectangleArea(double length, double  
width) {  
        return length * width;  
    }  
}
```

Output:

Area of Circle: 176.71458676442586

Area of Rectangle: 50.0

Basic Built In code

FileExample.java

```
import java.io.FileWriter;

import java.io.FileReader;

import java.io.IOException;

public class FileExample {

    public static void main(String[] args) {

        try {

            // Writing to file

            FileWriter writer = new FileWriter("sample.txt");

            writer.write("Hello, Agneay!");

            writer.close();

            // Reading from file

            FileReader reader = new FileReader("sample.txt");

            int ch;

            while ((ch = reader.read()) != -1) {

                System.out.print((char) ch);

            }

            reader.close();

        }

    }

}
```

```
    } catch (IOException e) {  
  
        System.out.println("An error occurred.");  
  
        e.printStackTrace();  
  
    }  
  
}  
  
}
```

MathExample.java

```
public class MathExample {  
  
    public static void main(String[] args) {  
  
        double result = Math.pow(2, 5); // 2 to the power 5  
  
        System.out.println("2^5 = " + result);  
  
    }  
  
}
```

Input Example

```
import java.io.FileWriter;  
  
import java.io.FileReader;  
  
import java.io.IOException;  
  
public class FileExample {  
  
    public static void main(String[] args) {  
  
        try {  
  
            // Writing to file  
  
            FileWriter writer = new FileWriter("sample.txt");
```



```
        writer.write("Hello, Agneay!");

        writer.close();

        // Reading from file

        FileReader reader = new FileReader("sample.txt");

        int ch;

        while ((ch = reader.read()) != -1) {

            System.out.print((char) ch);

        }

        reader.close();

    } catch (IOException e) {

        System.out.println("An error occurred.");

        e.printStackTrace();

    }

}
```

Advanced Built In Packages

DatabaseExample.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
public class DatabaseExample {  
    public static void main(String[]  
args) {  
        String url =  
"jdbc:sqlite:sample.db"; // Path to  
SQLite database file  
        try (Connection conn =  
DriverManager.getConnection(url)) {  
            if (conn != null) {  
                System.out.println("C  
onnected to the database!");  
                String query =  
"SELECT id, name FROM users";  
                Statement stmt =  
conn.createStatement();  
                ResultSet rs =  
stmt.executeQuery(query);  
  
                while (rs.next()) {  
                    int id =  
rs.getInt("id");
```

```
                String name =  
rs.getString("name");  
                System.out.println  
n("User ID: " + id + ", Name: " +  
name);  
            }  
        }  
    } catch (SQLException e) {  
        System.out.println(e.getM  
essage());  
    }  
}
```

SwingExample.java

```
import javax.swing.*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class SwingExample {  
    public static void main(String[]  
args) {
```

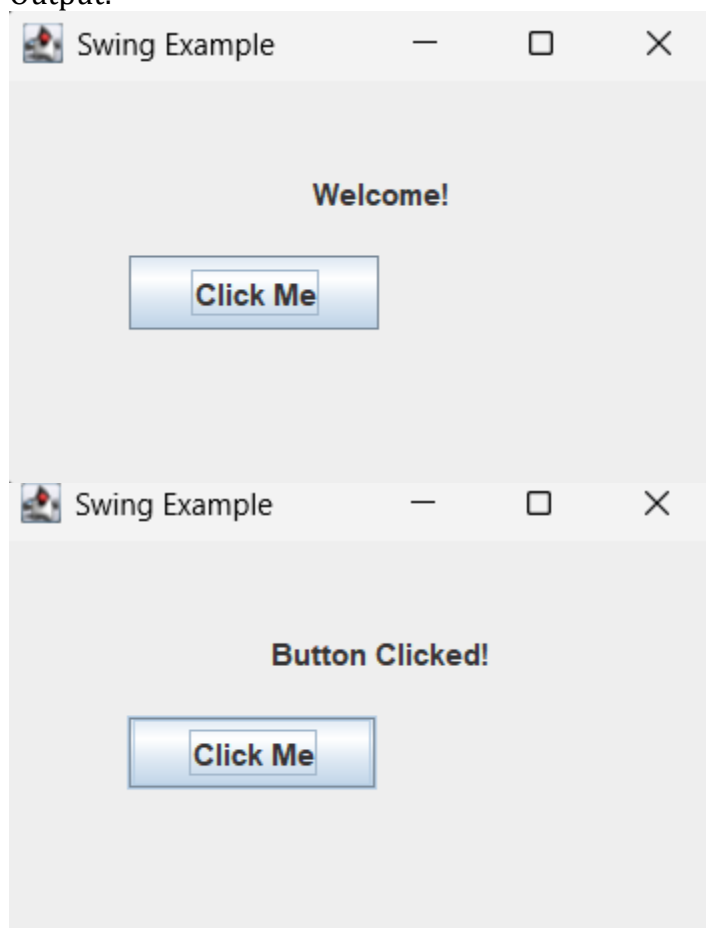
```
        JFrame frame = new
JFrame("Swing Example");
        JButton button = new
JButton("Click Me");
        JLabel label = new
JLabel("Welcome!", JLabel.CENTER);

        button.addActionListener(new
ActionListener() {
            @Override
            public void
actionPerformed(ActionEvent e) {
                label.setText("Button
Clicked!");
            }
        });

        frame.setLayout(null);
        label.setBounds(50, 30, 200,
30);
        button.setBounds(50, 70, 100,
30);
```

```
frame.add(label);
frame.add(button);
frame.setSize(300, 200);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
}
}
```

Output:



TCPServer.java

```
import java.net.*;
```

```
import java.io.*;

public class TCPServer {
    public static void main(String[]
args) throws IOException {
        ServerSocket serverSocket =
new ServerSocket(12345); // Listening
on port 12345
        System.out.println("Server is
running...");

        Socket clientSocket =
serverSocket.accept(); // Accept
client connection
        System.out.println("Client
connected");

        BufferedReader input = new
BufferedReader(new
InputStreamReader(clientSocket.getInp
utStream()));
        String message =
input.readLine();
```

```
        System.out.println("Client  
says: " + message);  
  
        clientSocket.close();  
        serverSocket.close();  
    }  
}
```