**FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERIG**

**Department of Computer Science and Engineering**

|  |  |  |  |
| --- | --- | --- | --- |
| **Academic Year** | **2025-2026** | **Estimated Time** | **Experiment No. 2 – 02 Hours** |
| **Course & Semester** | **S.E. CSE** | **Subject Name** | **Object Oriented**  **Programming with Java Lab** |
| **Module No.** | **02** | **Chapter Title** |  |
| **Experiment Type** | **Software Performance** | **Subject Code** | 25PCC12CS07 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of Student** | Atharva Dharmendra Jagtap | **Roll No.** |  |
| **Date of Performance.:** |  | **Date of Submission.:** |  |
| **CO Mapping** | **CO2. Apply Object-Oriented Programming Principles to given problem.** | | |

**Objective of Experiment:**

To explore and demonstrate the fundamental concepts of Object-Oriented Programming (OOP),

**Pre-Requisite:** Any programming language like C, C++

**Tools:** Java IDLE

**Theory:**

**A. Classes and Objects**

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of objects, which contain data in the form of fields (often known as attributes or properties) and code in the form of methods (functions associated with the object). OOP allows for modular, reusable, and maintainable code, making it a fundamental approach in software engineering.

* **Classes:** A class is a blueprint for creating objects. It defines a type by bundling data and methods that work on the data into one single unit. Classes encapsulate data for the object and the methods to manipulate that data. For example, a Car class might have fields like model, color, and year, and methods like startEngine and stopEngine.
* **Objects:** An object is an instance of a class. When a class is defined, no memory is allocated until an object of that class is created. An object is created using the new keyword, which allocates memory for the object and initializes its fields.

Car myCar = new Car();

myCar.model = "Toyota";

myCar.color = "Red";

myCar.year = 2022;

myCar.startEngine();

public class Car

{

// Fields

String model;

String color;

int year;

// Methods

void start Engine()

{

System.out.println("Engine started.");

}

}

public class Car {

// Fields

String model;

String color;

int year;

// Methods

void startEngine() {

System.out.println("Engine started.");

}

}

* **Creating Objects:** Objects are instances of a class. You create an object using the new keyword.

#### Fields and Methods

#### Instance Variables and Class Variables:

#### Instance variables are unique to each object instance.

#### Class variables (declared with the static keyword) are shared among all instances of a class.

**Types of Variables:**

|  |  |
| --- | --- |
| **Description** | **Example** |
| **Instance Variables:**  Instance variables (non-static fields) are unique to each instance of a class. Every object has its own copy of these variables. | public class Student  {  String name; // Instance variable static int total Students; // Class variable  Student (String name)  {this.name = name;  total Students++;  } } |
| **Class Variables**:  Class variables (static fields) are shared among all instances of a class. They are declared with the static keyword, meaning they are associated with the class rather than any particular object. |

**B. Types of Methods**

|  |  |
| --- | --- |
| **Description** | **Example** |
| **Instance Method:**  Instance methods are associated with an object and can access the object’s fields and methods. | public class Calculator  {  // Instance method  int add(int a, int b)  {  return a + b;  }  // Class method  static int multiply(int a, int b)  {  return a \* b;  }  } |
| **Class Methods:**  Class methods, also known as static methods, belong to the class itself rather than any object. They can be called without creating an instance of the class and can only access static data. |
| **Accessor and Mutator Methods:** Accessor (getter) methods are used to retrieve the value of private fields, while Mutator (setter) methods are used to modify the value of private fields. This practice is part of encapsulation, which restricts direct access to an object's data and allows controlled access through methods. | public class Person {  private String name;  // Getter method  public String getName() {  return name;  }  // Setter method  public void setName(String name) {  this.name = name;  }  } |

### C Constructors:

### Multiple constructors can be defined with different parameter lists.

|  |  |
| --- | --- |
| **Description** | **Example** |
| **Default Constructor:**  A constructor is a special method that is called when an object is instantiated. The default constructor is provided by the compiler if no constructors are defined in the class. It initializes object fields with default values. | public class Dog {  String breed;  Dog() {  breed = "Unknown";  }  } |
| **Parameterized Constructors**:  Parameterized constructors allow the initialization of objects with specific values at the time of creation. This is useful for setting up an object in a known state. | public class Dog {  String breed;  Dog(String breed) {  this.breed = breed;  }  } |
| **Constructor Overloading:**  Constructor overloading is the ability to have multiple constructors in the same class with different parameter lists. This allows objects to be initialized in different ways. | public class Book {  String title;  String author;  // Default constructor  Book() {  title = "Unknown";  author = "Unknown";  }  // Parameterized constructor  Book(String title, String author) {  this.title = title;  this.author = author;  }  } |

### D. Encapsulation

|  |  |  |
| --- | --- | --- |
| **Description** | | **Example** |
| **Encapsulation:**  Encapsulation is the bundling of data (fields) and methods that operate on that data within a single unit or class. It restricts direct access to some of an object’s components, which is a means of preventing unintended interference and misuse. Encapsulation is typically achieved using access modifiers like private, protected, public, and package-private (default).  **Private Access Modifier:**  The private modifier restricts access to fields and methods within the same class only. It is the most restrictive level of access control. | public class Account{private double balance;public double getBalance() {return balance;}public void deposit(double amount) {if(amount > 0) {balance += amount;}}} | |
| **Public, Protected, and Package-private Access Modifiers:**   * **Public**: Accessible from any other class. * **Protected:** Accessible within the same package and by subclasses. * **Package-private** (default): Accessible only within the same package. | public class Example{public int publicVar;protected int protectedVar;int packagePrivateVar;private int privateVar;} | |
| The “**this**” Keyword:  The **this** keyword is used to refer to the current instance of the class. It is often used to distinguish between class fields and parameters with the same name. | public class Employee {  private String name;  public Employee(String name) {  this.name = name;  }  public void printName() {  System.out.println(this.name);  }  } | |

D. Static Members:

|  |  |
| --- | --- |
|  |  |
| **Static Variables:**  Static variables, also known as class variables, are shared among all instances of a class. They are used for values that should be common to all objects of that class. | public class Company  {  static String companyName = "Tech Corp";  } |
| **Static Methods:** Static methods belong to the class rather than any particular object of the class. They can be invoked without creating an instance of the class. Static methods can only access static variables and cannot call instance methods directly. | public class MathUtils  {  public static int square(int number)  {  return number \* number;  }  } |
| **The Static Block:** A static block is a block of code that runs when the class is first loaded. It is used for initializing static variables or performing operations that are necessary before the class is used | public class Database  {  static {  System.out.println("Connecting to the database...");  }  } |
| **Static Nested Classes:** A static nested class is a class declared inside another class with the static modifier. Unlike inner classes, static nested classes do not have access to the instance variables and methods of the outer class. | public class OuterClass  {  static class StaticNestedClass  {  void display()  {  System.out.println("Static nested class.");  }  }  } |
| **Inner Classes** | |
| **Member Inner Classes:**  A member inner class is a non-static class that is defined within another class. It can access the outer class’s members**,** including private ones. | public class Outer {  class Inner {  void display()  {  System.out.println("Inner class method.");  }  }  } |
| **Local Inner Classes:** A local inner class is defined within a method of the outer class. It can access the local variables of the method it is defined in, provided they are final or effectively final**.** | public class Outer  {  void myMethod()  {  class LocalInner  {  void display()  {  System.out.println("Local inner class.");  }  }  LocalInner local = new LocalInner();  local.display();  }  } |
| **Anonymous Inner Classes:** An anonymous inner class is a class without a name that is declared and instantiated all in one statement. It is often used to provide a simple implementation of an interface or a subclass with a few methods overridden. | public class Main  {  public static void main(String[] args)  {  MyInterface obj = new MyInterface()  {  public void display()  {  System.out.println("Anonymous inner class.");  }  };  obj.display();  }  }  interface MyInterface  {  void display();  } |
| **Static Nested Classes:** A static nested class is a static class that is defined within another class. It cannot access non-static members of the outer class directly, as it does not have a reference to an instance of the outer class. | public class Outer {  static class StaticNested  {  void display()  {  System.out.println("Static nested class.");  }  }  } |
|  | |

**F. Arrays of Objects**:

|  |  |
| --- | --- |
| **Defining Methods**  A method in Java is defined within a class. The basic syntax for defining a method includes the access modifier, return type, method name, and parameters (if any). | public class Main {  public static void main(String[] args) {  // Creating an array of Car objects  Car[] cars = new Car[3];  cars[0] = new Car("Toyota", "Red", 2022);  cars[1] = new Car("Honda", "Blue", 2021);  cars[2] = new Car("Ford", "Black", 2023);  // Accessing elements  for (Car car : cars) {  System.out.println(car.model + " " + car.color + " " + car.year);  }  }  } |

**Problem Description:**

|  |  |
| --- | --- |
| **Banking Problem:** Banking Application Design a simple banking application that allows users to deposit, withdraw, and check their account balance. | |
| Input:   * **Deposit Amount**: The amount of money the user wants to deposit into their account. * **Withdraw Amount**: The amount of money the user wants to withdraw from their account. * **Check Balance**: A request to check the current balance of the user's account. * **Input:** PIN or Password | Expected Output:  **Deposit Action:** Updated account balance after the deposit.  **Withdraw Action:** Updated account balance after the withdrawal or an error message if funds are insufficient.  **Check Balance Action:** The current account balance.  **Authentication:** Access granted to proceed with transactions or an error message if the authentication fails. |

**Code:**

import java.util.Scanner;

// 1. Account Class to encapsulate account data

class Account {

    private String accountNumber;

    private double balance;

    private String pin; // Storing PIN as String for simplicity; in real app, it would be hashed.

    public Account(String accountNumber, String pin, double initialBalance) {

        this.accountNumber = accountNumber;

        this.pin = pin;

        this.balance = initialBalance;

    }

    public String getAccountNumber() {

        return accountNumber;

    }

    public double getBalance() {

        return balance;

    }

    // Authenticates the user with the provided PIN

    public boolean authenticate(String enteredPin) {

        return this.pin.equals(enteredPin);

    }

    // Deposits money into the account

    public void deposit(double amount) {

        if (amount > 0) {

            balance += amount;

            System.out.printf("Successfully deposited $%.2f.%n", amount);

        } else {

            System.out.println("Deposit amount must be positive.");

        }

    }

    // Withdraws money from the account

    public boolean withdraw(double amount) {

        if (amount <= 0) {

            System.out.println("Withdrawal amount must be positive.");

            return false;

        }

        if (balance >= amount) {

            balance -= amount;

            System.out.printf("Successfully withdrew $%.2f.%n", amount);

            return true;

        } else {

            System.out.println("Insufficient funds. Withdrawal failed.");

            return false;

        }

    }

}

// Main BankingApplication class

public class BankingApplication {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Create a single account for this simple simulation

        // Account Number: 12345

        // PIN: 1234

        // Initial Balance: $500.00

        Account userAccount = new Account("12345", "1234", 500.00);

        System.out.println("--- Welcome to Your Bank ---");

        System.out.println("Please insert your card (Account: " + userAccount.getAccountNumber() + ")");

        // Authentication Loop

        int attempts = 0;

        final int MAX\_ATTEMPTS = 3;

        boolean isAuthenticated = false;

        while (attempts < MAX\_ATTEMPTS && !isAuthenticated) {

            System.out.print("Enter your 4-digit PIN: ");

            String enteredPin = scanner.nextLine();

            if (userAccount.authenticate(enteredPin)) {

                System.out.println("PIN accepted. Access granted!");

                isAuthenticated = true;

            } else {

                attempts++;

                System.out.println("Incorrect PIN. " + (MAX\_ATTEMPTS - attempts) + " attempts remaining.");

            }

        }

        if (!isAuthenticated) {

            System.out.println("Too many incorrect PIN attempts. Card blocked. Please contact your bank.");

            scanner.close();

            return; // Exit the program if authentication fails

        }

        // Main Transaction Loop

        int choice;

        do {

            System.out.println("\n--- Main Menu ---");

            System.out.println("1. Deposit");

            System.out.println("2. Withdraw");

            System.out.println("3. Check Balance");

            System.out.println("4. Exit");

            System.out.print("Enter your choice: ");

            if (scanner.hasNextInt()) {

                choice = scanner.nextInt();

                scanner.nextLine(); // Consume newline

                switch (choice) {

                    case 1:

                        System.out.print("Enter amount to deposit: $");

                        if (scanner.hasNextDouble()) {

                            double depositAmount = scanner.nextDouble();

                            scanner.nextLine(); // Consume newline

                            userAccount.deposit(depositAmount);

                            System.out.printf("Current Balance: $%.2f%n", userAccount.getBalance());

                        } else {

                            System.out.println("Invalid input. Please enter a numeric amount.");

                            scanner.nextLine(); // Consume invalid input

                        }

                        break;

                    case 2:

                        System.out.print("Enter amount to withdraw: $");

                        if (scanner.hasNextDouble()) {

                            double withdrawAmount = scanner.nextDouble();

                            scanner.nextLine(); // Consume newline

                            userAccount.withdraw(withdrawAmount);

                            System.out.printf("Current Balance: $%.2f%n", userAccount.getBalance());

                        } else {

                            System.out.println("Invalid input. Please enter a numeric amount.");

                            scanner.nextLine(); // Consume invalid input

                        }

                        break;

                    case 3:

                        System.out.printf("Your current balance is: $%.2f%n", userAccount.getBalance());

                        break;

                    case 4:

                        System.out.println("Thank you for banking with us. Goodbye!");

                        break;

                    default:

                        System.out.println("Invalid choice. Please select a number between 1 and 4.");

                }

            } else {

                System.out.println("Invalid input. Please enter a number for your choice.");

                scanner.nextLine(); // Consume invalid input

                choice = 0; // Set to 0 to keep the loop running

            }

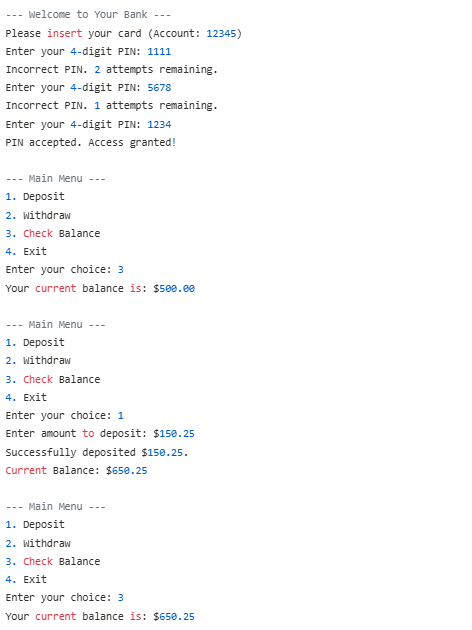
        } while (choice != 4);

        scanner.close();

    }

}

**Output:**

****

**Post Lab Questions:**

|  |
| --- |
|  |
|  |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| **On time Completion and Submission (2)** | **Knowledge of the topic (4)** | **Implementation and Output (4)** | **Total (10)** |
|  |  |  |  |

**References:**

|  |  |
| --- | --- |
| **Study Materials**  [**https://www.w3schools.com/java/**](https://www.w3schools.com/java/)  [**https://www.geeksforgeeks.org/java/**](https://www.geeksforgeeks.org/java/)  https://www.codecademy.com/learn/learn-java | **Video Channels**:  [**https://www.youtube.com/user/programmingwithmosh**](https://www.youtube.com/user/programmingwithmosh)  [**https://www.youtube.com/c/TheNetNinja**](https://www.youtube.com/c/TheNetNinja)  [**https://www.youtube.com/c/Freecodecamp**](https://www.youtube.com/c/Freecodecamp)  [**https://www.youtube.com/user/Simplilearn**](https://www.youtube.com/user/Simplilearn) |
| **Study Materials used for Demo**  <Add links here> | |

**Note:-students are expected to paste screenshot of the program with output**