Experiment - 4

Student Name: Ankush Thakur
UID: 22BCS11815
Branch: BE- CSE
Section/Group: 22BCS_IOT-637 (A)
Semester: 6th
Date of Performance: 14/2/2025
Subject Name: Project Based Learning in Java
Subject Code: 22CSH-359

Problem - 1

1. **Aim:** Write a Java program to implement an ArrayList that stores employee details (ID, Name, and Salary). Allow users to add, update, remove, and search employees.

2. **Objective:** To implement an ArrayList in Java for managing employee details (ID, Name, Salary) with functionalities to add, update, remove, search, and display employee records.

3. **Implementation/Code:**

```
import java.util.ArrayList;
import java.util.Scanner;

class Employee {
  int ID;
  String Name;
  double Salary;

  Employee(int id, String name, double salary) {
    ID = id;
    Name = name;
    Salary = salary;
  }

  @Override
  public String toString() {
    return "ID: " + ID + "\tName: " + Name + "\tSalary: " + Salary;
  }

  void changeValue(String newName) { Name = newName; }
  void changeValue(double newSalary) { Salary = newSalary; }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
```

```
        return true;
      if (obj == null || getClass() != obj.getClass())
        return false;
      Employee employee = (Employee)obj;
      return ID == employee.ID || Name.equals(employee.Name);
    }
}

class Operations {
  int Finder(ArrayList<Employee> arr, String name) {
    Employee searchByName = new Employee(0, name, 0);
    if (arr.contains(searchByName)) {
      int index = arr.indexOf(searchByName);
      Employee foundEmployee = arr.get(index);
      System.out.println(foundEmployee.toString());
      return index;
    } else {
      System.out.println("Employee not found by name.");
      return -1;
    }
  }

  int Finder(ArrayList<Employee> arr, int id) {
    Employee searchById = new Employee(id, "", 0);
    if (arr.contains(searchById)) {
      int index = arr.indexOf(searchById);
      Employee foundEmployee = arr.get(index);
      System.out.println(foundEmployee.toString());
      return index;
    } else {
      System.out.println("Employee not found by ID.");
      return -1;
    }
  }

  void addEmployee(ArrayList<Employee> arr, Scanner obj) {
    System.out.print("ID: ");
    int id = obj.nextInt();
    obj.nextLine();
    System.out.print("Name: ");
    String name = obj.nextLine();
    System.out.print("Salary: ");
    double salary = obj.nextDouble();

    Employee temp = new Employee(id, name, salary);
    arr.add(temp);
  }

  void removeEmployee(ArrayList<Employee> arr, int id) {
    int idx = Finder(arr, id);
    if (idx != -1) {
      Employee foundEmployee = arr.get(idx);
      System.out.println("Removing employee\n" + foundEmployee.toString());
      arr.remove(foundEmployee);
    }
```

```
    }

    void removeEmployee(ArrayList<Employee> arr, String name) {
      int idx = Finder(arr, name);
      if (idx != -1) {
        Employee foundEmployee = arr.get(idx);
        System.out.println("Removing employee\n" + foundEmployee.toString());
        arr.remove(foundEmployee);
      }
    }

    void updateEmployee(ArrayList<Employee> arr, int id, Scanner obj) {
      int idx = Finder(arr, id);
      if (idx != -1) {
        Employee foundEmployee = arr.get(idx);
        System.out.println("Current details: " + foundEmployee.toString());
        System.out.println("What do you want to update?");
        System.out.println("1. Name");
        System.out.println("2. Salary");
        System.out.print("Choose: ");
        int choice = obj.nextInt();
        obj.nextLine();

        switch (choice) {
        case 1:
          System.out.print("Enter new name: ");
          String newName = obj.nextLine();
          foundEmployee.changeValue(newName);
          System.out.println("Name updated successfully.");
          break;

        case 2:
          System.out.print("Enter new salary: ");
          double newSalary = obj.nextDouble();
          foundEmployee.changeValue(newSalary);
          System.out.println("Salary updated successfully.");
          break;

        default:
          System.out.println("Invalid choice.");
        }
      }
    }
}

class exp1 {
    public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      ArrayList<Employee> employees = new ArrayList<>();
      Operations operations = new Operations();
      boolean flag = true;

      System.out.println("Choose Operations-");
      System.out.println("1. Add employee\n2. Find Employee\n3. Remove "
                         + "Employee\n4. Update Employee\n5. Exit");
```

```
        while (flag) {
          int choice;
          System.out.print("Choose: ");
          choice = input.nextInt();
          input.nextLine();

          switch (choice) {
          case 1:
            operations.addEmployee(employees, input);
            break;

          case 2:
            System.out.println("Enter name or ID: ");
            String key = input.nextLine();
            try {
              int id = Integer.parseInt(key);
              operations.Finder(employees, id);
            } catch (NumberFormatException e) {
              operations.Finder(employees, key);
            }
            break;

          case 3:
            System.out.println("Enter name or ID: ");
            String removeKey = input.nextLine();
            try {
              int removeId = Integer.parseInt(removeKey);
              operations.removeEmployee(employees, removeId);
            } catch (NumberFormatException e) {
              operations.removeEmployee(employees, removeKey);
            }
            break;

          case 4:
            System.out.print("Enter employee ID to update: ");
            int updateId = input.nextInt();
            input.nextLine();
            operations.updateEmployee(employees, updateId, input);
            break;

          case 5:
            flag = false;
            System.out.println("Exiting...");
            break;

          default:
            System.out.println("Invalid choice. Try again.");
          }
        }
        input.close();
      }
    }
```

4. Output:

```
Choose Operations-
1. Add employee
2. Find Employee
3. Remove Employee
4. Update Employee
5. Exit
Choose: 1
ID: 132
Name: Ankush
Salary: 69000
Choose: 1
ID: 143
Name: Vishal
Salary: 5400
```

```
Choose: 2
Enter name or ID:
Ankush
ID: 132 Name: Ankush    Salary: 69000.0
Choose: 4
Enter employee ID to update: 143
ID: 143 Name: Vishal    Salary: 5400.0
Current details: ID: 143        Name: Vishal    Salary: 5400.0
What do you want to update?
1. Name
2. Salary
Choose: 2
Enter new salary: 68000
Salary updated successfully.
```

```
Choose: 2
Enter name or ID:
Vishal
ID: 143 Name: Vishal    Salary: 68000.0
Choose: 5
Exiting...
```

Problem – 2

1. **Aim:** Create a program to collect and store all the cards to assist the users in finding all the cards in each symbol using Collection interface.

2. **Objective:** To develop a Java program using the Collection interface to store and manage a set of playing cards. The program will allow users to add, retrieve, and search for all cards of a given symbol efficiently. This ensures organized card management and easy lookup based on user input.

3. **Implementation/Code:**

```
import java.util.*;
class Card {
    String rank;
    String suit;
    public Card(String rank, String suit) {
        this.rank = rank;
        this.suit = suit;
    }
    public void displayCard() {
        System.out.println(rank + " of " + suit);
    }
}

public class CardCollection {
    private static List < Card > deck = new ArrayList < > ();
    public static void initializeDeck() {
        String[] ranks = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J",
"Q", "K", "A"};
        String[] suits = {
            "Hearts",
            "Diamonds",
            "Clubs",
            "Spades"
        };
        for (String suit: suits) {
            for (String rank: ranks) {
                deck.add(new Card(rank, suit));
            }
        }
    }

    public static void findCardsBySuit(String suit) {
        boolean found = false;
        for (Card card: deck) {
            if (card.suit.equalsIgnoreCase(suit)) {
                card.displayCard();
                found = true;
```

```
                }
            }
            if (!found) {
                System.out.println("No cards found for the suit: " + suit);
            }
        }
        public static void displayAllCards() {
            if (deck.isEmpty()) {
                System.out.println("The deck is empty.");
            } else {
                for (Card card: deck) {
                    card.displayCard();
                }
            }
        }

        public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            initializeDeck();
            while (true) {
                System.out.println("\nCard Collection System");
                System.out.println("1. Display All Cards");
                System.out.println("2. Find Cards by Suit");
                System.out.println("3. Exit");
                System.out.print("Enter your choice: ");
                int choice = scanner.nextInt();
                scanner.nextLine();
                switch (choice) {
                    case 1:
                        displayAllCards();
                        break;
                    case 2:
                        System.out.print("Enter the suit (Hearts, Diamonds, Clubs,
Spades) to find: ");
                        String suit = scanner.nextLine();
                        findCardsBySuit(suit);
                        break;
                    case 3:
                        System.out.println("Exiting... Goodbye!");
                        scanner.close();
                        return;
                    default:
                        System.out.println("Invalid choice. Please try again.");
                }
            }
        }
}
```

4. Output:

```
Card Collection System
1. Display All Cards
2. Find Cards by Suit
3. Exit
Enter your choice: 2
Enter the suit (Hearts, Diamonds, Clubs, Spades) to find: Hearts
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
J of Hearts
Q of Hearts
K of Hearts
A of Hearts
```

Problem – 3

1. **Aim:** Develop a ticket booking system with synchronized threads to ensure no double booking of seats. Use thread priorities to simulate VIP bookings being processed first.

2. **Objective:** To develop a multi-threaded Ticket Booking System in Java that ensures synchronized seat booking, preventing double bookings. The system will use thread priorities to simulate VIP bookings being processed first, demonstrating effective thread management, synchronization, and priority handling in concurrent programming.

3. **Implementation/Code:**

```
class TicketBookingSystem {
    private int totalSeats;
    private int bookedSeats;
    public TicketBookingSystem(int totalSeats) {
        this.totalSeats = totalSeats;
        this.bookedSeats = 0;
    }
    public synchronized boolean bookTicket(String customerName, boolean isVIP)
{
        if (bookedSeats < totalSeats) {
            bookedSeats++;
            System.out.println(customerName + " successfully booked a seat.");
            return true;
        } else {
            System.out.println("Sorry  "  +  customerName  +  ",  no  seats
available.");
            return false;
        }
    }
    public int getAvailableSeats() {
        return totalSeats - bookedSeats;
    }
}
class Customer extends Thread {
    private String customerName;
    private TicketBookingSystem bookingSystem;
    private boolean isVIP;

    public  Customer(String  customerName,  TicketBookingSystem  bookingSystem,
boolean isVIP) {
        this.customerName = customerName;
        this.bookingSystem = bookingSystem;
        this.isVIP = isVIP;
    }
    @Override
    public void run() {
        if (isVIP) {
```

```
            System.out.println(customerName + " is a VIP. Processing VIP
booking...");
        } else {
            System.out.println(customerName + " is a regular customer.
Processing booking...");
        }
        boolean booked = bookingSystem.bookTicket(customerName, isVIP);
        if (booked) {
            System.out.println(customerName + " has successfully booked the
ticket.");
        } else {
            System.out.println(customerName + " failed to book a ticket due to
no available seats.");
        }
    }
}
public class Main {
    public static void main(String[] args) {
        TicketBookingSystem bookingSystem = new TicketBookingSystem(5);

        Customer customer1 = new Customer("Ankush", bookingSystem, true);
        Customer customer2 = new Customer("Akshu", bookingSystem, false);
        Customer customer3 = new Customer("Tika", bookingSystem, false);
        Customer customer4 = new Customer("Mia", bookingSystem, true);
        Customer customer5 = new Customer("Alex", bookingSystem, false);
        Customer customer6 = new Customer("Oliver", bookingSystem, true);

        customer1.setPriority(Thread.MAX_PRIORITY);
        customer2.setPriority(Thread.NORM_PRIORITY);
        customer3.setPriority(Thread.NORM_PRIORITY);
        customer4.setPriority(Thread.MAX_PRIORITY);
        customer5.setPriority(Thread.NORM_PRIORITY);
        customer6.setPriority(Thread.MAX_PRIORITY);
        customer1.start();
        customer2.start();
        customer3.start();
        customer4.start();
        customer5.start();
        customer6.start();
    }
}
```

4. Output:

```
Akshu has successfully booked the ticket.
Tika successfully booked a seat.
Mia successfully booked a seat.
Mia has successfully booked the ticket.
Alex successfully booked a seat.
Alex has successfully booked the ticket.
Tika has successfully booked the ticket.
Sorry Oliver, no seats available.
Oliver failed to book a ticket due to no available seats.
```

Learning Outcomes:

1. **I was able to implement and manipulate an ArrayList** to store and manage complex data, such as employee details (ID, Name, and Salary). I gained hands-on experience in performing CRUD operations (Create, Read, Update, Delete) on collections, which enhanced my understanding of dynamic data storage and retrieval.
2. **I mastered the Java Collections Framework** and learned to use interfaces like List, Set, and Map effectively. I understood how to organize and retrieve data (e.g., cards grouped by symbols) using appropriate collection types, which improved my ability to design efficient data management systems.
3. **I developed a deep understanding of multithreading in Java**, including the use of synchronized methods to prevent race conditions. I successfully implemented a thread-safe application, such as a ticket booking system, ensuring no double booking of seats.
4. **I explored thread prioritization and its role in managing concurrent processes**. By simulating VIP bookings with higher thread priorities, I understood how to control the order of thread execution and applied this knowledge to real-world scenarios requiring prioritized task handling.
5. **I enhanced my problem-solving skills** by designing and implementing real-world applications, such as an employee management system, a card collection system, and a ticket booking system. I learned to translate user requirements into functional Java programs, integrating concepts like collections, multithreading, and synchronization.