Experiment 4

Student Name: Diksha
UID: 23BCS10994
Branch: CSE
Section/Group: KRG_2B
Semester: 5th
Date of Performance: 20/9/25
Subject Name: PBLJ
Subject Code: 23CSH-304

1. Aim:

Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

A) Easy Level:

- 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.

B) Medium Level:

- Create a program to collect and store all the cards to assist the users in finding all the cards in a given symbol using Collection interface.

C) Hard Level:

- 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. Objectives:

- To understand how to use Java Collections, specifically ArrayList, to manage dynamic data efficiently.
- To understand collection interfaces like Map, List, and how to store and retrieve grouped data.
- To understand multithreading, thread synchronization, and thread priorities in Java.
- To illustrate basic thread handling, synchronization, and concurrency concepts.
- To simulate a real-world scenario of priority-based resource allocation.

3. JAVA script and output:

EASY-LEVEL PROBLEM

```
import java.util.*;

class Employee {
    int id;
    String name;
    double salary;

    Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}
public class EmployeeList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<Employee> list = new ArrayList<>();

        while (true) {
            System.out.println("\n1. Add Employee");
            System.out.println("2. Update Employee");
            System.out.println("3. Remove Employee");
            System.out.println("4. Search Employee");
            System.out.println("5. Display All Employees");
            System.out.println("6. Exit");
            System.out.print("Enter choice: ");
            int choice = sc.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter ID: ");
                    int id = sc.nextInt();
```

```
            sc.nextLine();
            System.out.print("Enter Name: ");
            String name = sc.nextLine();
            System.out.print("Enter Salary: ");
            double salary = sc.nextDouble();
            list.add(new Employee(id, name, salary));
            System.out.println("Employee added.");
            break;

        case 2:
            System.out.print("Enter ID to update: ");
            int uid = sc.nextInt();
            boolean foundUpdate = false;
            for (Employee e : list) {
                if (e.id == uid) {
                    sc.nextLine();
                    System.out.print("Enter new Name: ");
                    e.name = sc.nextLine();
                    System.out.print("Enter new Salary: ");
                    e.salary = sc.nextDouble();
                    System.out.println("Employee updated.");
                    foundUpdate = true;
                    break;
                }
            }
            if (!foundUpdate) System.out.println("Employee not found.");
            break;

        case 3:
            System.out.print("Enter ID to remove: ");
            int rid = sc.nextInt();
            boolean removed = list.removeIf(e -> e.id == rid);
            if (removed) System.out.println("Employee removed.");
            else System.out.println("Employee not found.");
            break;

        case 4:
            System.out.print("Enter ID to search: ");
            int sid = sc.nextInt();
            boolean found = false;
```

```
                for (Employee e : list) {
                    if (e.id == sid) {
                        System.out.println(e);
                        found = true;
                        break;
                    }
                }
                if (!found) System.out.println("Employee not found.");
                break;

            case 5:
                if (list.isEmpty()) System.out.println("No employees.");
                else for (Employee e : list) System.out.println(e);
                break;

            case 6:
                System.out.println("Exiting...");
                sc.close();
                return;

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

Output:

Output

```
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter choice: 1
Enter ID: 11
Enter Name: Diksha
Enter Salary: 20000
Employee added.

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter choice: 2
Enter ID to update: 13
Employee not found.

1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Enter choice: 6
Exiting...

=== Code Execution Successful ===
```

MEDIUM LEVEL PROBLEM:

```
import java.util.*;
class Card {
    String symbol;
    String value;
    Card(String symbol, String value) {
        this.symbol = symbol;
        this.value = value;
    }
    public String toString() {
        return value + " of " + symbol;
    }
}
public class CardCollection {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Collection<Card> cards = new ArrayList<>();
        cards.add(new Card("Hearts", "A"));
        cards.add(new Card("Hearts", "2"));
        cards.add(new Card("Diamonds", "K"));
        cards.add(new Card("Spades", "Q"));
        cards.add(new Card("Hearts", "10"));
        cards.add(new Card("Clubs", "J"));

        System.out.print("Enter symbol to search (Hearts/Diamonds/Clubs/Spades): ");
        String symbol = sc.nextLine();
```

```
        boolean found = false;
        for (Card c : cards) {
            if (c.symbol.equalsIgnoreCase(symbol)) {
                System.out.println(c);
                found = true;
            }
        }
        if (!found) {
            System.out.println("No cards found with symbol: " + symbol);
        }
        sc.close();
    }
}
```

Output:

```
Output
Enter symbol to search (Hearts/Diamonds/Clubs/Spades): Hearts
A of Hearts
2 of Hearts
10 of Hearts

=== Code Execution Successful ===
```

HARD LEVEL PROBLEM

```
class TicketBookingSystem {
    private int availableSeats;
    TicketBookingSystem(int seats) {
        this.availableSeats = seats;
    }
    public synchronized void bookSeat(String customer) {
        if (availableSeats > 0) {
            System.out.println(customer + " booked a seat. Seats left: " + (availableSeats - 1));
            availableSeats--;
        } else {
            System.out.println("No seats available for " + customer);
        }
    }
}
class Customer extends Thread {
    private TicketBookingSystem system;

    Customer(String name, TicketBookingSystem system, int priority) {
        super(name);
        this.system = system;
        setPriority(priority);
    }
    public void run() {
        system.bookSeat(getName());
    }
}

public class Main {
    public static void main(String[] args) {
        TicketBookingSystem system = new TicketBookingSystem(5);

        Customer c1 = new Customer("VIP1", system, Thread.MAX_PRIORITY);
        Customer c2 = new Customer("VIP2", system, Thread.MAX_PRIORITY);
        Customer c3 = new Customer("Normal1", system, Thread.NORM_PRIORITY);
        Customer c4 = new Customer("Normal2", system, Thread.NORM_PRIORITY);
        Customer c5 = new Customer("Normal3", system, Thread.NORM_PRIORITY);
        Customer c6 = new Customer("Normal4", system, Thread.NORM_PRIORITY);
```

```
        c1.start();
        c2.start();
        c3.start();
        c4.start();
        c5.start();
        c6.start();

        try {
            c1.join(); c2.join(); c3.join(); c4.join(); c5.join(); c6.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT: