Experiment 4

Student Name: Alok Verma **UID:** 22BCS10396

Branch: CSE **Section/Group:** 22BCS_IOT-638/B

Semester: 6th **Date of Performance:** 11/02/2025

Subject Name: PBLJ Lab **Subject Code:** 22CSH-359

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

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.

Programming Code:

```
import java.util.ArrayList;

import java.util.Scanner;


class Employee {

    int id;

    String name;

    double salary;


    public Employee(int id, String name, double salary) {

        this.id = id;

        this.name = name;

        this.salary = salary;

    }
```

```
    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Salary: " + salary;
    }
}

public class EmployeeManagement {
    static ArrayList<Employee> employees = new ArrayList<>();
    static Scanner sc = new Scanner(System.in);

    public static void addEmployee() {
        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();
        employees.add(new Employee(id, name, salary));
        System.out.println("Employee Added Successfully!\n");
    }

    public static void updateEmployee() {
        System.out.print("Enter ID to update: ");
        int id = sc.nextInt();
```

```
    for (Employee e : employees) {
        if (e.id == id) {
            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 Successfully!\n");
            return;
        }
    }
    System.out.println("Employee not found!\n");
}

public static void removeEmployee() {
    System.out.print("Enter ID to remove: ");
    int id = sc.nextInt();
    employees.removeIf(e -> e.id == id);
    System.out.println("Employee Removed Successfully!\n");
}

public static void searchEmployee() {
    System.out.print("Enter ID to search: ");
    int id = sc.nextInt();
    for (Employee e : employees) {
        if (e.id == id) {
```

```
            System.out.println("Employee Found: " + e + "\n");
            return;
        }
    }
    System.out.println("Employee not found!\n");
}

public static void displayEmployees() {
    System.out.println("Employee List:");
    for (Employee e : employees) {
        System.out.println(e);
    }
    System.out.println();
}

public static void main(String[] args) {
    while (true) {
        System.out.println("1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit");
        System.out.print("Enter choice: ");
        int choice = sc.nextInt();
        switch (choice) {
            case 1 -> addEmployee();
            case 2 -> updateEmployee();
            case 3 -> removeEmployee();
            case 4 -> searchEmployee();
            case 5 -> displayEmployees();
```

```
            case 6 -> {
                System.out.println("Exiting...");
                System.exit(0);
            }
            default -> System.out.println("Invalid choice!\n");
        }
    }
  }
}
```

Output:

```
input
1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit
Enter choice: 1
Enter ID: 01
Enter Name: Alok
Enter Salary: 99999
Employee Added Successfully!

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit
Enter choice: 2
Enter ID to update: 01
Enter new Name: 22BCS10396_Alok verma
Enter new Salary: 989898
Employee Updated Successfully!

1. Add 2. Update 3. Remove 4. Search 5. Display 6. Exit
Enter choice: 5
Employee List:
ID: 1, Name: 22BCS10396_Alok verma, Salary: 989898.0

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


...Program finished with exit code 0
Press ENTER to exit console.
```

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.

Programming Code:

```
import java.util.*;


public class CardCollection {
    public static void main(String[] args) {
        HashMap<String, ArrayList<String>> cards = new HashMap<>();
        Scanner sc = new Scanner(System.in);


        while (true) {
            System.out.println("1. Add Card  2. Find Cards by Symbol  3. Display All  4. Exit");
            System.out.print("Enter choice: ");
            int choice = sc.nextInt();
            sc.nextLine();


            switch (choice) {
                case 1 -> {
                    System.out.print("Enter Card Symbol (e.g., Hearts, Spades): ");
                    String symbol = sc.nextLine();
                    System.out.print("Enter Card Name (e.g., Ace, King): ");
                    String name = sc.nextLine();
                    cards.putIfAbsent(symbol, new ArrayList<>());
                    cards.get(symbol).add(name);
                    System.out.println("Card Added Successfully!\n");
                }
                case 2 -> {
                    System.out.print("Enter Symbol to Search: ");
```

```
                String symbol = sc.nextLine();
                System.out.println(cards.getOrDefault(symbol, new ArrayList<>()) + "\n");
            }
            case 3 -> {
                System.out.println("Card Collection:");

                cards.forEach((symbol, list) -> System.out.println(symbol + " -> " + list));
                System.out.println();
            }
            case 4 -> {
                System.out.println("Exiting...");
                System.exit(0);
            }
            default -> System.out.println("Invalid Choice!\n");
        }
    }
  }
}
```

Output:

```
input
1. Add Card  2. Find Cards by Symbol  3. Display All  4. Exit
Enter choice: 1
Enter Card Symbol (e.g., Hearts, Spades): Hearts
Enter Card Name (e.g., Ace, King): King
Card Added Successfully!

1. Add Card  2. Find Cards by Symbol  3. Display All  4. Exit
Enter choice: 1
Enter Card Symbol (e.g., Hearts, Spades): Spades
Enter Card Name (e.g., Ace, King): Queen
Card Added Successfully!

1. Add Card  2. Find Cards by Symbol  3. Display All  4. Exit
Enter choice: 3
Card Collection:
Spades -> [Queen]
Hearts -> [King]

1. Add Card  2. Find Cards by Symbol  3. Display All  4. Exit
Enter choice: 4
Exiting...


...Program finished with exit code 0
Press ENTER to exit console.
```

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.

Programming Code:

```
import java.util.concurrent.locks.*;


class TicketBookingSystem {

    private int availableSeats = 3;

    private final Lock lock = new ReentrantLock(true);
```

```
    public void bookTicket(String name) {
        lock.lock();
        try {
            if (availableSeats > 0) {
                System.out.println(name + " booked a ticket. Seats left: " + (--availableSeats));
            } else {
                System.out.println(name + " failed to book. No seats available.");
            }
        } finally {
            lock.unlock();
        }
    }
}

class Passenger extends Thread {
    private final TicketBookingSystem system;
    private final String name;

    public Passenger(TicketBookingSystem system, String name, int priority) {
        this.system = system;
        this.name = name;
        setPriority(priority);
    }

    public void run() {
        system.bookTicket(name);
```

```
    }
}

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

        Passenger vip1 = new Passenger(system, "VIP 1", Thread.MAX_PRIORITY);
        Passenger vip2 = new Passenger(system, "VIP 2", Thread.MAX_PRIORITY);
        Passenger user1 = new Passenger(system, "User 1", Thread.NORM_PRIORITY);
        Passenger user2 = new Passenger(system, "User 2", Thread.NORM_PRIORITY);
        Passenger user3 = new Passenger(system, "User 3", Thread.MIN_PRIORITY);

        vip1.start();
        vip2.start();
        user1.start();
        user2.start();
        user3.start();
    }
}
```

Output:

```
VIP 1 booked a ticket. Seats left: 2
VIP 2 booked a ticket. Seats left: 1
User 2 booked a ticket. Seats left: 0
User 3 failed to book. No seats available.
User 1 failed to book. No seats available.


...Program finished with exit code 0
Press ENTER to exit console.
```

Learning Outcomes:

Easy Level: Learn to manage employee records using ArrayList with add, update, remove, and search operations.

Medium Level: Understand how to use HashMap and ArrayList to store and retrieve categorized data efficiently.

Hard Level: Implement synchronized multithreading with Lock to ensure secure ticket booking with thread priority.

- Understand the use of the **Collection Framework** in Java for efficient data management.
- Learn to implement **ArrayList** for dynamic storage and manipulation of employee records.
- Explore **HashMap** for categorizing and retrieving data using key-value pairs.
- Implement **synchronized multithreading** to prevent race conditions in ticket booking.
- Use **thread priorities** to simulate VIP and normal user processing in real-time applications.
- Gain hands-on experience with **thread lifecycle, synchronization, and data structures** in Java.