Experiment 2.1

Student Name: Manyata **UID: 22BCS10802**
Branch: CSE **Section: 22KPIT-901/A**
Semester: 6th **Date of Performance:21/02/2025**
Subject: Project Based Learning in Java **Subject Code: 22CSH-359**

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

2. **Objective 1: 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.

3. **Code/Implementation:**

```
import java.util.*;
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;
    }
}

public class EmployeeManagement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Employee> employees = new ArrayList<>();

        while (true) {
            System.out.println("1. Add Employee\n2. Remove Employee\n3.
Search Employee\n4. Exit");
            int choice = scanner.nextInt();
            if (choice == 1) {
                System.out.println("Enter ID, Name, Salary:");
                employees.add(new Employee(scanner.nextInt(),
scanner.next(), scanner.nextDouble()));
            } else if (choice == 2) {
                System.out.println("Enter Employee ID to Remove:");
```

```
                    int id = scanner.nextInt();
                    employees.removeIf(emp -> emp.id == id);
                } else if (choice == 3) {
                    System.out.println("Enter Employee ID to Search:");
                    int id = scanner.nextInt();
                    employees.stream().filter(emp -> emp.id == id).findFirst()
                            .ifPresent(emp -> System.out.println(emp.id + " " +
emp.name + " " + emp.salary));
                } else {
                    break;
                }
            }
            scanner.close();
        }
}
```

Output:

```
1. Add Employee
2. Remove Employee
3. Search Employee
4. Exit
1
Enter ID, Name, Salary:
1025
Tanishka
200000
1. Add Employee
2. Remove Employee
3. Search Employee
4. Exit
3
Enter Employee ID to Search:
1024
1024 Manyata 1000000.0
```

4. **Objective 2: 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.

5. **Code/Implementation:**

```
import java.util.*;
class Card {
    char symbol;
    int number;

    public Card(char symbol, int number) {
        this.symbol = symbol;
```

```
            this.number = number;
        }
    }

    public class CollectAndGroupCards {
        public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            Map<Character, List<Card>> cardMap = new TreeMap<>();

            System.out.println("Enter Number of Cards :");
            int n = scanner.nextInt();

            for (int i = 1; i <= n; i++) {
                System.out.println("Enter card " + i + ":");
                char symbol = scanner.next().charAt(0);
                int number = scanner.nextInt();

                cardMap.putIfAbsent(symbol, new ArrayList<>());
                cardMap.get(symbol).add(new Card(symbol, number));
            }

            System.out.println("Distinct Symbols are :");
            for (char symbol : cardMap.keySet()) {
                System.out.print(symbol + " ");
            }
            System.out.println();

            for (char symbol : cardMap.keySet()) {
                System.out.println("Cards in " + symbol + " Symbol");
                int sum = 0;
                for (Card card : cardMap.get(symbol)) {
                    System.out.println(card.symbol + " " + card.number);
                    sum += card.number;
                }
                System.out.println("Number of cards : " +
    cardMap.get(symbol).size());
                System.out.println("Sum of Numbers : " + sum);
            }
            scanner.close();
        }
    }
```

Output:

```
Enter Number of Cards :
5
Enter card 1:
s 5
Enter card 2:
s 12
Enter card 3:
h 5
Enter card 4:
h 6
Enter card 5:
c 1
```

6. **Objective 3 : 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.

7. **Code/Implementation:**

```
import java.util.*;

class TicketBookingSystem {
    private static final int TOTAL_SEATS = 100;
    private final boolean[] seats = new boolean[TOTAL_SEATS];

    public synchronized boolean bookSeat(int seatNumber) {
        if (seatNumber < 0 || seatNumber >= TOTAL_SEATS ||
seats[seatNumber]) {
            return false; // Invalid or already booked seat
        }
        seats[seatNumber] = true;
        return true;
    }
}

public class TicketBooking {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        TicketBookingSystem bookingSystem = new TicketBookingSystem();

        // Collect VIP seat inputs first
```

```
            System.out.println("Enter number of VIP seats to book:");
            int vipSeatsCount = scanner.nextInt();
            List<Integer> vipSeats = new ArrayList<>();
            for (int i = 0; i < vipSeatsCount; i++) {
                System.out.println("Enter VIP seat number:");
                vipSeats.add(scanner.nextInt());
            }

            // Collect Normal seat inputs
            System.out.println("Enter number of Normal seats to book:");
            int normalSeatsCount = scanner.nextInt();
            List<Integer> normalSeats = new ArrayList<>();
            for (int i = 0; i < normalSeatsCount; i++) {
                System.out.println("Enter Normal seat number:");
                normalSeats.add(scanner.nextInt());
            }

            // Create and run threads for booking seats
            Thread vipThread = new Thread(() -> {
                for (int seat : vipSeats) {
                    System.out.println("VIP booked seat " + seat + ": " +
bookingSystem.bookSeat(seat));
                }
            });

            Thread normalThread = new Thread(() -> {
                for (int seat : normalSeats) {
                    System.out.println("Normal booked seat " + seat + ": " +
bookingSystem.bookSeat(seat));
                }
            });

            vipThread.setPriority(Thread.MAX_PRIORITY);
            normalThread.setPriority(Thread.MIN_PRIORITY);

            vipThread.start();
            normalThread.start();

            try {
                vipThread.join();
                normalThread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            scanner.close();
        }
    }
```

Output:

```
Enter number of VIP seats to book:
3
Enter VIP seat number:
25
Enter VIP seat number:
26
Enter VIP seat number:
27
Enter number of Normal seats to book:
3
Enter Normal seat number:
55
Enter Normal seat number:
56
Enter Normal seat number:
58
```

```
VIP booked seat 25: true
VIP booked seat 26: true
VIP booked seat 27: true
Normal booked seat 55: true
Normal booked seat 56: true
Normal booked seat 58: true
```

8. Learning Outcomes:

- Learned how to use ArrayList to store and manage employee details.
- Understood how to group and retrieve cards using the Map interface.
- Practiced synchronized threads to prevent double booking in a ticket system.
- Used thread priorities to ensure VIP bookings happen first.
- Fixed input handling issues to avoid errors in multithreaded programs.