DEPARTMENT OF
Discover. Learn. Empower.
COMPUTER SCIENCE & ENGINEERING

Experiment 4

Name:Kaustubh Umrao
UID: 22BCS16008
Branch: BE-CSE
Section/Group: 22BCS_IOT_643/A
Semester: 6th
Date of Performance: 11-02-25
Subject Name: Project based Learning in Java
Subject Code: 22ITH-352

Problem 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.

Objective:

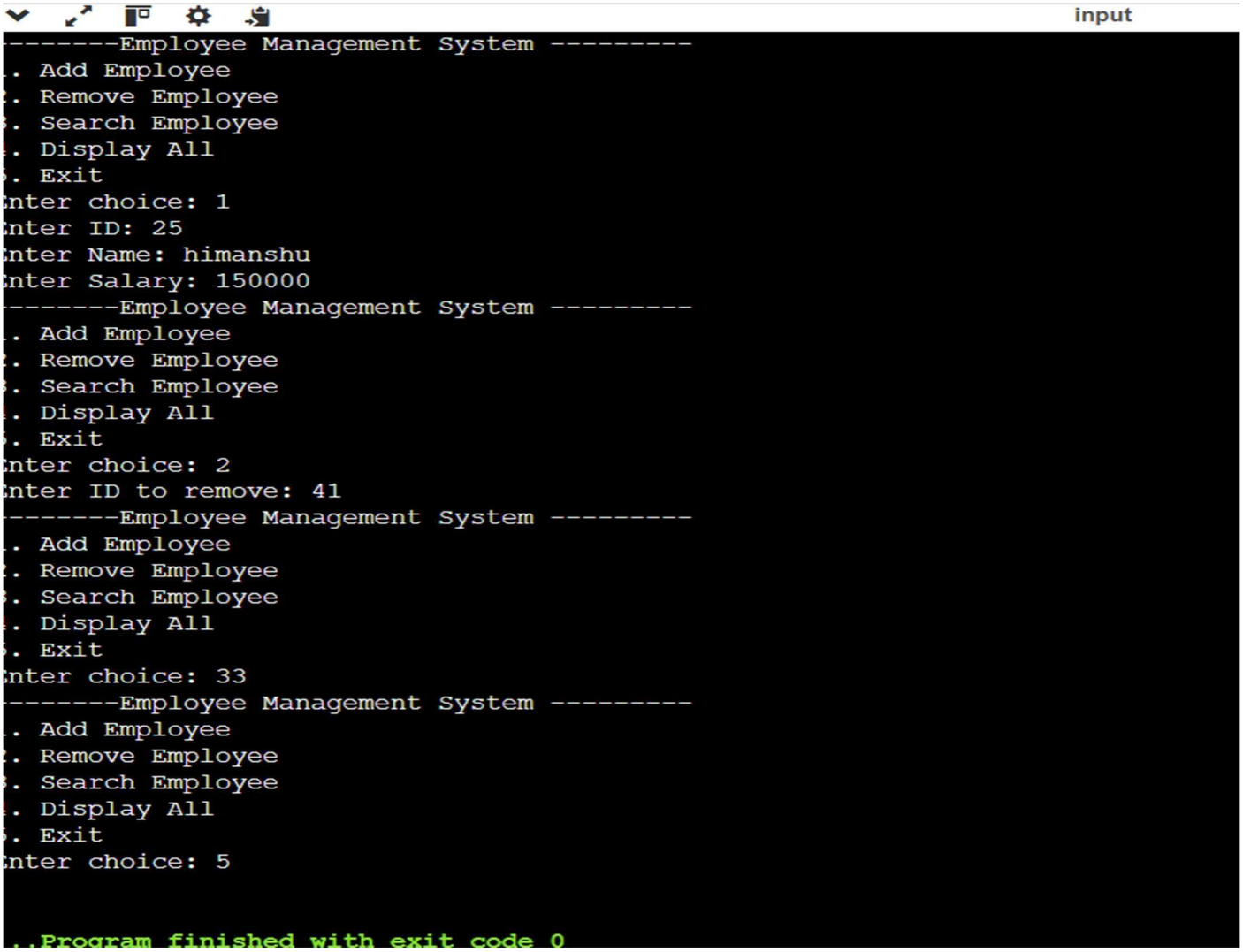
- To create a Java program to manage employee information (ID, Name, Salary) using an ArrayList.
- To enable users to add, update, delete, and search for employee records.
- To ensure efficient access and modification of employee details.

Code:

```
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 EmployeeManagement { public static
    void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<>();
        Scanner sc = new Scanner(System.in); int choice;
do {
                System.out.println("--------Employee Management System --------- \n1. Add Employee\n2. Remove
Employee\n3. Search Employee\n4. Display All\n5. Exit");
          System.out.print("Enter choice: "); choice
```

```
            = sc.nextInt();

            switch (choice) {
               case 1:
                  System.out.print("Enter ID: "); int id    =          sc.nextInt();
                  System.out.print("Enter Name: ");
                  String name = sc.next(); System.out.print("Enter Salary: "); double salary = sc.nextDouble();
                  employees.add(new     Employee(id,   name, salary)); break;
case 2:
                  System.out.print("Enter ID to remove: "); int removeId = sc.nextInt();
                  employees.removeIf(emp -> emp.id == removeId); break;
case 3:
                  System.out.print("Enter ID to search: "); int     searchId          =
                           sc.nextInt();     for
                  (Employee emp : employees) {  if (emp.id == searchId) {
                        System.out.println(emp);
                     }
                  } break;
case 4:
                  for (Employee emp : employees) { System.out.println(emp);  } break;
               }
         } while (choice != 5);

         sc.close();
      }
   }
```

Output :



Learning Outcomes:

- Gained knowledge on utilizing ArrayList for dynamically storing and managing employee records.
- Learned the methods for adding, updating, deleting, and searching elements in an ArrayList.
- Learnt implementing search functionality using switch-case statements, loops, and conditions..

Problem 2 :

Aim: 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.

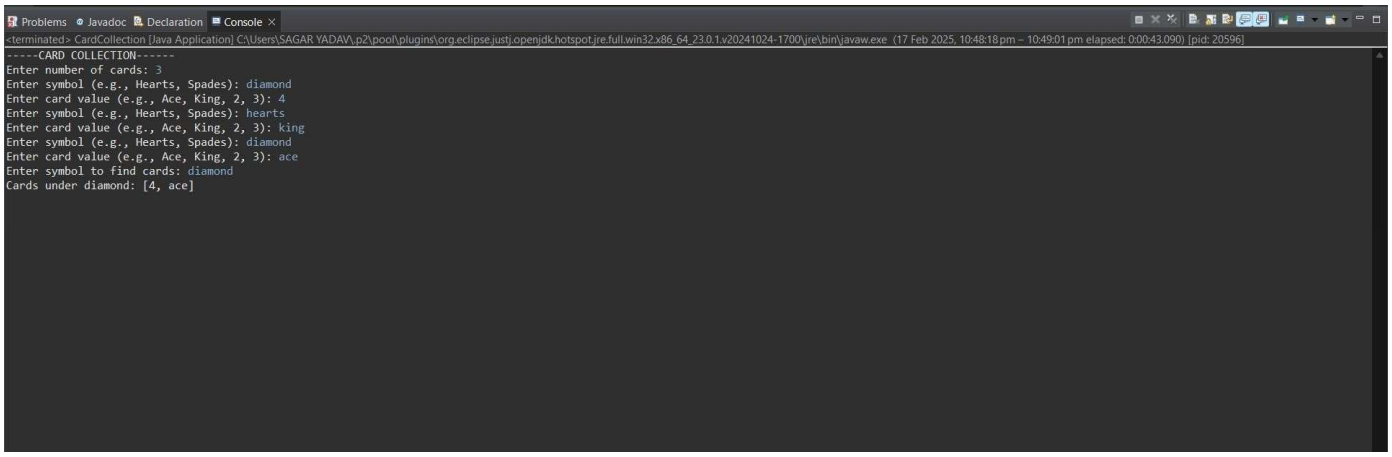
Objective:

- To use the Java Collection Interface to effectively store and manage card information.
- To implement symbol-based searching to allow users to find all cards linked to a specific symbol.
- To ensure organized storage and retrieval by using suitable data structures such as HashSet or HashMap.

Code: import java.util.ArrayList; import java.util.HashMap; import java.util.List; import java.util.Scanner;

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

        System.out.print("-----CARD COLLECTION ----- \nEnter number of cards: "); int
        n = sc.nextInt(); sc.nextLine(); // newline
for (int i = 0; i < n; i++) {
            System.out.print("Enter symbol (e.g., Hearts, Spades): ");
            String symbol = sc.nextLine();
            System.out.print("Enter card value (e.g., Ace, King, 2, 3): ");  String
            value = sc.nextLine();

            cards.putIfAbsent(symbol, new ArrayList<>()); // It will store the values in the HashMap
        cards.get(symbol).add(value); }

        System.out.print("Enter symbol to find cards: ");
        String findSymbol = sc.nextLine();

       List<String> cardList = cards.getOrDefault(findSymbol, new ArrayList<>()); //we will get the number of
  cards and which card is stored under particular symbol
        System.out.println("Cards under " + findSymbol + ": " + cardList);

sc.close();
    }
}
```

Output:



Learning Outcomes:

- Understand the Collection Interface and how to implement it for managing card data.
- Explored different Collection types like List, Set, or Map based on the use case.
- Learned how to choose the appropriate Collection implementation for different scenarios.

Problem 3:

Aim: To 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.

Objective:

- To use synchronized threads to avoid multiple users booking the same seat at the same time.
- To implement locks or synchronized methods to ensure thread safety.
- To assign higher thread priority to VIP bookings to ensure they are processed first.

Code:

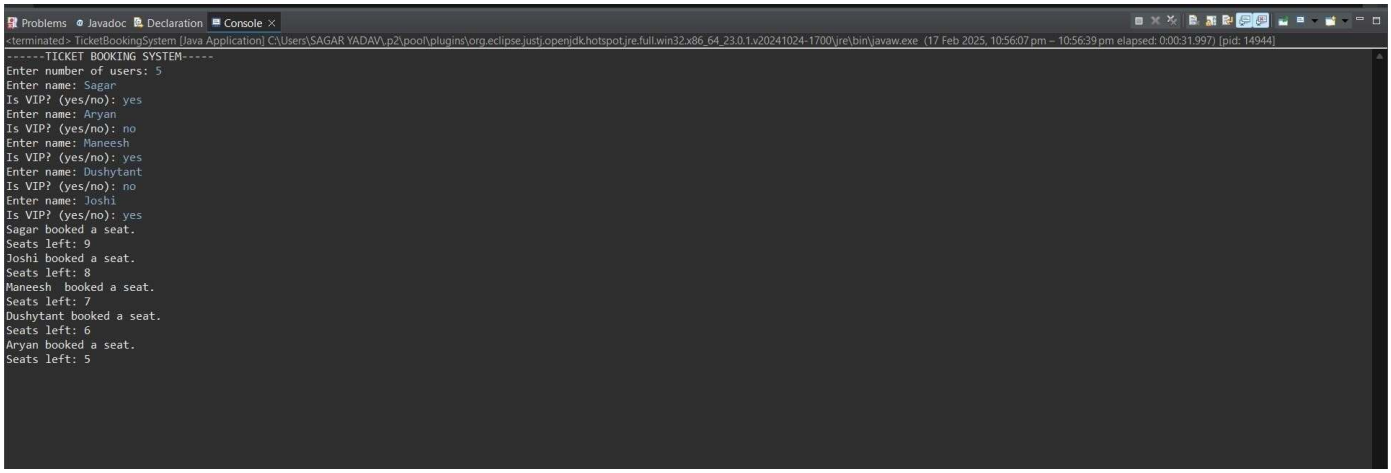
```
import java.util.*; import
java.util.concurrent.*;

class TicketBooking implements Runnable {
   private static int availableSeats = 10;
   private final String name; private final
   boolean isVIP;
```

```
    TicketBooking(String        name,   boolean        isVIP)  { this.name =
       name;
       this.isVIP = isVIP;
    }

    public        boolean        isVIP()
          { return isVIP;
    }

    @Override public void run() { if
    (availableSeats
    > 0) {
          System.out.println(name + " booked a seat. Seats left: " + (--availableSeats));
       } else {
          System.out.println(name + " booking failed. No seats available.");
       }
    }
}

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

       PriorityQueue<TicketBooking> queue = new PriorityQueue<>(
          Comparator.comparing(TicketBooking::isVIP).reversed() // VIP Should come first
       );

       System.out.print("------TICKET BOOKING SYSTEM ----- \nEnter number of users:
       "); int n = sc.nextInt(); sc.nextLine();

       for (int i = 0; i < n; i++) {
          System.out.print("Enter name: ");
          String name = sc.nextLine();
          System.out.print("Is VIP? (yes/no): ");  boolean isVIP =
          sc.nextLine().equalsIgnoreCase("yes");

          queue.add(new TicketBooking(name, isVIP));
       }
```

ExecutorService executor = Executors.newSingleThreadExecutor(); // Make sure VIP get the Priority while (!queue.isEmpty()) { executor.execute(queue.poll()); } executor.shutdown();

```
    sc.close();
} }
```

Output:



Learning Outcomes:

- Gained knowledge on creating and managing multiple threads by understanding the thread lifecycle and its various states.
- Learnt how to set and manage thread priorities.
- Understood how to set thread priorities to control the order of execution.