Experiment-4

Student Name: Rakesh Reddy **UID: 22BCS12118**
Branch: BE-CSE **Section/Group: 641-B**
Semester: 6th **Date of Performance: 05/03/2025**
Subject Name: Project Based Learning In Java with Lab **Subject Code: 22CSH-359**

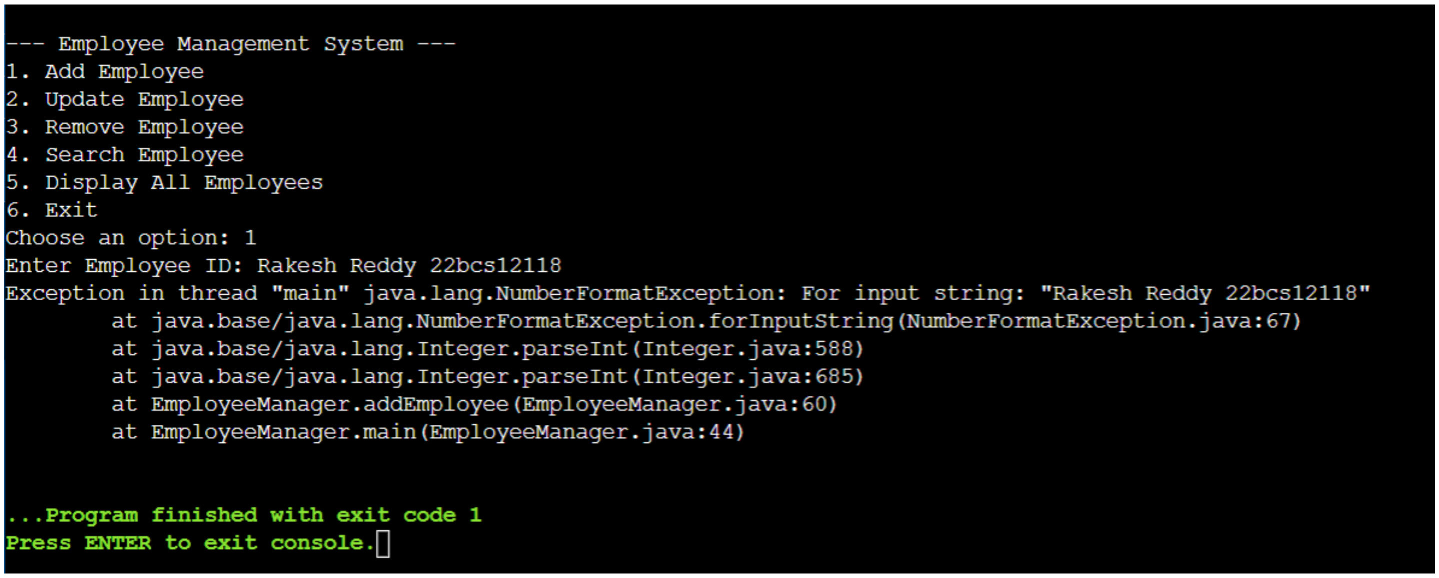
1. Write a Java program to implement an **ArrayList** that stores employee details (**ID**, **Name**, and **Salary**). The program should allow users to **add**, **update**, **remove**, and **search** employee records.

2. **Implementation/Code:**

```
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;
        }

        @Override
        public String toString() { return String.format("ID: %d, Name: %s, Salary:
                %.2f", id, name, salary);
        }
}

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

        public static void main(String[] args) { while (true) {
                    System.out.print("""
                            \n--- Employee Management System --- 1.
                            Add Employee
                            2. Update Employee
                            3. Remove Employee
                            4. Search Employee
                            5. Display All Employees
                            6. Exit
```

```
                                    Choose an option: """);


           switch (sc.nextInt()) { case 1 ->              addEmployee();
           case 2 -> updateEmployee(); case 3 ->     removeEmployee(); case 4 ->
           searchEmployee(); case 5 ->       displayEmployees();
                                       case 6 -> { System.out.println("Exiting..."); return; } default -> System.out.println("Invalid
                                        choice! Try
          again.");
                              }
                         }
                    }
static    void    addEmployee()    {    System.out.print("Enter    ID,    Name,    Salary:    ");
                    employees.add(new    Employee(sc.nextInt(),    sc.next(),    sc.nextDouble()));
                    System.out.println("Employee added successfully!"); }

                    static  void  updateEmployee()  {  System.out.print("Enter  Employee  ID  to
                         update: "); int id = sc.nextInt();
                         employees.stream().filter(e -> e.id == id).findFirst().ifPresentOrElse(e -> {
                              System.out.print("Enter New Name and Salary: "); e.name = sc.next();
                              e.salary  =  sc.nextDouble();  System.out.println("Employee  updated
                              successfully!");
                         }, () -> System.out.println("Employee not found!")); }

                    static  void  removeEmployee()  {  System.out.print("Enter  Employee  ID  to
                         remove: ");
                         System.out.println(employees.removeIf(e -> e.id == sc.nextInt())
                              ? "Employee removed successfully!"
                              : "Employee not found!"); }

                    static void searchEmployee() { System.out.print("Enter Employee ID to search:
                         ");
                         employees.stream().filter(e -> e.id == sc.nextInt()).findFirst()
                                    .ifPresentOrElse(System.out::println, () -> System.out.println("Employee not found!"));
                    }

                    static      void      displayEmployees()      {      if      (employees.isEmpty())
                         System.out.println("No          employees          found.");          else
                         employees.forEach(System.out::println);
                    }
          }
```

3. OUTPUT:


```
--- Employee Management System ---
1. Add Employee
2. Update Employee
3. Remove Employee
4. Search Employee
5. Display All Employees
6. Exit
Choose an option: 1
Enter Employee ID: Rakesh Reddy 22bcs12118
Exception in thread "main" java.lang.NumberFormatException: For input string: "Rakesh Reddy 22bcs12118"
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
        at java.base/java.lang.Integer.parseInt(Integer.java:588)
        at java.base/java.lang.Integer.parseInt(Integer.java:685)
        at EmployeeManager.addEmployee(EmployeeManager.java:60)
        at EmployeeManager.main(EmployeeManager.java:44)


...Program finished with exit code 1
Press ENTER to exit console.
```

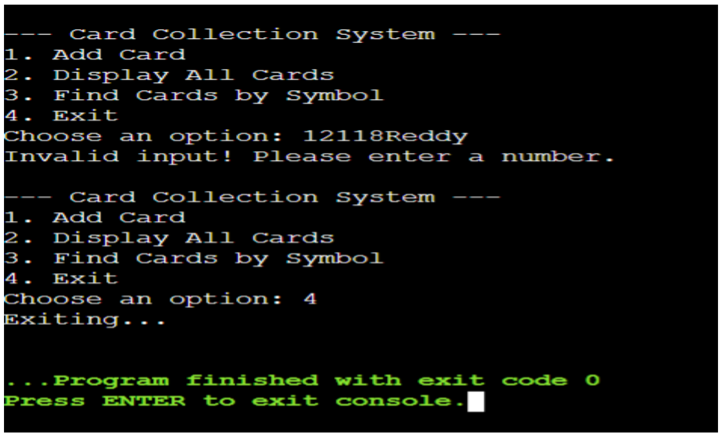
4. Create a Java program to **collect** and **store** all playing cards to help users **find** all cards of a given **symbol** (e.g., **Hearts**, **Diamonds**) using the **Collection** interface.

5. **CODE:**

```
import java.util.*;

// Class representing a Card class Card {
      private String symbol; private int
      value;

      public Card(String symbol, int value) { this.symbol = symbol; this.value
            = value;
      }

      public String getSymbol() { return symbol;
      }

      public int getValue() { return value; }

      @Override
      public String toString() { return String.format("Card { Symbol: '%s', Value: %d
            }", symbol, value);
      }
}
```

```
// Class for Card Collection Management public class CardCollection
{
        private Collection<Card> cards = new ArrayList<>(); private Scanner scanner = new
        Scanner(System.in);

        // Method to add a card public void
        addCard() {
                System.out.print("Enter Card Symbol: "); String symbol =
                scanner.next(); System.out.print("Enter Card Value: "); int
                value = scanner.nextInt(); cards.add(new Card(symbol,
                value));
                System.out.println("Card added successfully!");
        }

        // Method to display all cards public void
        displayCards() { if (cards.isEmpty()) {
                        System.out.println("No cards in the collection."); return;
                }
                System.out.println("\n--- All Cards ---"); cards.forEach(System.out::println);
        }

        // Method to find all cards of a given symbol public void findCardsBySymbol()
        {
                System.out.print("Enter Symbol to search: "); String symbol
                = scanner.next(); boolean found = false;
                System.out.println("\nCards with Symbol '" + symbol +
"':");
                for (Card card : cards) { if (card.getSymbol().equalsIgnoreCase(symbol)) {
                        System.out.println(card); found = true;
                        }
                }
                if (!found) {
                        System.out.println("No cards found with symbol '" + symbol + "'.");
                }
        }

        // Menu-driven interface public void start() {
                while (true) {
                        System.out.println("\n--- Card Collection System ---

                    ");        System.out.println("1. Add Card"); System.out.println("2. Display All Cards");
System.out.println("3. Find Cards by Symbol"); System.out.println("4. Exit"); System.out.print("Choose an
                                                                                        option: ");
                    int choice = scanner.nextInt(); switch (choice) {
                        case 1 -> addCard(); case 2 ->
                        displayCards();
                        case 3 -> findCardsBySymbol(); case 4 -> {
                            System.out.println("Exiting..."); return;
                        }
                        default -> System.out.println("Invalid choice!
            Try again.");
                    }
                }
            }

            // Main method
            public static void main(String[] args) { CardCollection system =
            new CardCollection(); system.start(); }
        }
```

6. OUTPUT:



```
--- Card Collection System ---
1. Add Card
2. Display All Cards
3. Find Cards by Symbol
4. Exit
Choose an option: 12118Reddy
Invalid input! Please enter a number.

--- Card Collection System ---
1. Add Card
2. Display All Cards
3. Find Cards by Symbol
4. Exit
Choose an option: 4
Exiting...


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

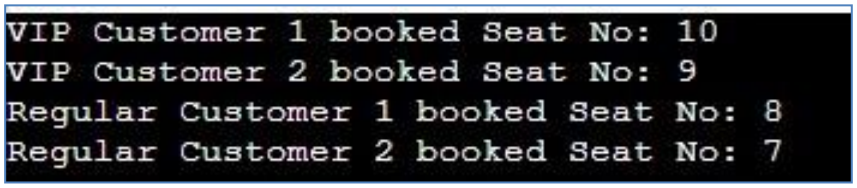
7. Develop a **ticket booking system** in **Java** using **synchronized threads** to ensure **no double booking** of seats. Implement **thread priorities** to simulate **VIP bookings** being **processed first**.

8. CODE:

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

```
// TicketBooking class handles seat reservations class TicketBooking
implements Runnable { private static int availableSeats = 10; // Total
seats
        private static final ReentrantLock lock = new ReentrantLock(); // Lock to prevent
double booking private final String customerType; // VIP
        or Regular

        public TicketBooking(String customerType) { this.customerType = customerType;
        }

        @Override
        public void run() { bookTicket(); }

        // Method to handle ticket booking private void bookTicket() { lock.lock(); //
        Ensure only one thread modifies availableSeats at a time try { if
        (availableSeats > 0) {
                                System.out.println(customerType + " booked Seat No: " + availableSeats);
                                availableSeats--; // Reduce seat count
                        } else {
                                System.out.println(customerType + " tried to book, but no seats
left!");
                        }
                } finally { lock.unlock(); // Release
                the lock }
        }

}

// Main class for Ticket Booking System public class
TicketBookingSystem { public static void
        main(String[] args) {
                // Create ticket booking threads for VIP and Regular customers
                Thread vip1 = new Thread(new TicketBooking("VIP Customer
1"));
                Thread vip2 = new Thread(new TicketBooking("VIP Customer Thread reg1 = new
2"));
                Thread(new TicketBooking("Regular
Customer 1"));
                Thread reg2 = new Thread(new TicketBooking("Regular Customer 2"));

                // Set VIP bookings to higher priority vip1.setPriority(Thread.MAX_PRIORITY); //
                Priority 10 vip2.setPriority(Thread.MAX_PRIORITY); // Priority 10
                reg1.setPriority(Thread.MIN_PRIORITY); // Priority 1
                reg2.setPriority(Thread.MIN_PRIORITY); // Priority 1

                // Start threads
                vip1.start(); vip2.start();
                reg1.start(); reg2.start();
            }
        }
```

9. OUTPUT:



```
VIP Customer 1 booked Seat No: 10
VIP Customer 2 booked Seat No: 9
Regular Customer 1 booked Seat No: 8
Regular Customer 2 booked Seat No: 7
```