Experiment 4

Student Name: Akshit Dutt **UID: 22BCS16465**
Branch: B.E CSE **Section: IOT-643-A**
Semester: 6th **DOP:24/02/25**
Subject: PBLJ **Subject Code: 22CSH-359**

Aim:

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

Problem Statement :

1) 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.

2) 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.

3) 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.

Algorithm:

1. Employee Management (Using ArrayList)

- ➢ Initialize an ArrayList to store employees.
- ➢ Display a menu with options: Add, Update, Remove, Search, and Exit.
- ➢ **Add Employee**:
  - Take user input for ID, Name, and Salary.
  - Create an Employee object and add it to the list.
- ➢ **Update Employee**:
  - Ask for the Employee ID.
  - If found, update Name and Salary.
- ➢ **Remove Employee**:
  - Ask for the Employee ID.
  - Remove matching employee from the list.
- ➢ **Search Employee**:
  - Ask for the Employee ID.
  - If found, display details.
- ➢ Repeat until the user chooses to exit.

2. Card Collection (Using Collections)

- Initialize an ArrayList to store Card objects.
- Display a menu with options: Add Card, Find Cards by Symbol, and Exit.
- **Add Card**:
  - Ask for card symbol (e.g., Hearts, Diamonds).
  - Ask for card value (A, 2, 3, ... J, Q, K).
  - Create a Card object and store it in the list.
- **Find Cards by Symbol**:
  - Ask for a symbol.
  - Search and display all cards with that symbol.
- Repeat until the user chooses to exit.

3. Ticket Booking System (Multithreading)

- Create a TicketBookingSystem with a limited number of seats.
- Implement synchronized booking to prevent double booking.
- Create Customer threads with different priorities (VIP first).
- **Each Customer thread**:
  - Tries to book a ticket.
  - If seats are available, booking is confirmed, and the seat count decreases.
  - If not, booking fails.
- Start all customer threads and process bookings.
- Stop when all threads have completed execution.

Program :

1. Employee Management:

```
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;
    }

    public void display() {
        System.out.println("ID: " + id + ", Name: " + name + ", Salary: $" + salary);
    }
}

public class EmployeeManager {

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

        while (true) {
            System.out.println("\n----- Employee Management System -----");
            System.out.println("1. Add Employee");
            System.out.println("2. Update Employee");
            System.out.println("3. Remove Employee");
            System.out.println("4. Search Employee");
            System.out.println("5. View All Employees");
            System.out.println("6. Exit");
            System.out.print("Enter your choice (1-6): ");

            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter Employee ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Enter Employee Name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter Employee Salary: ");
                    double salary = scanner.nextDouble();
                    employees.add(new Employee(id, name, salary));
                    System.out.println("Employee added successfully!");
                    break;

                case 2:
                    System.out.print("Enter the Employee ID to update: ");
                    int updateId = scanner.nextInt();
                    Employee employeeToUpdate = null;
                    for (Employee emp : employees) {
                        if (emp.id == updateId) {
                            employeeToUpdate = emp;
                            break;
                        }
                    }
                    if (employeeToUpdate != null) {
                        scanner.nextLine();
                        System.out.print("Enter new name: ");
                        employeeToUpdate.name = scanner.nextLine();
                        System.out.print("Enter new salary: ");
                        employeeToUpdate.salary = scanner.nextDouble();
                        System.out.println("Employee updated successfully!");
                    } else {
                        System.out.println("Employee with ID " + updateId + " not found.");
                    }
                    break;

                case 3:
                    System.out.print("Enter the Employee ID to remove: ");
                    int removeId = scanner.nextInt();
                    Employee employeeToRemove = null;
                    for (Employee emp : employees) {
                        if (emp.id == removeId) {
                            employeeToRemove = emp;
                            break;
                        }
                    }
                    if (employeeToRemove != null) {
                        employees.remove(employeeToRemove);
                        System.out.println("Employee removed successfully!");
                    } else {
                        System.out.println("Employee with ID " + removeId + " not found.");
                    }
                    break;

                case 4:
                    System.out.print("Enter Employee ID to search: ");
                    int searchId = scanner.nextInt();
                    Employee employeeToSearch = null;
                    for (Employee emp : employees) {
                        if (emp.id == searchId) {
                            employeeToSearch = emp;
                            break;
                        }
                    }
                    if (employeeToSearch != null) {
                        System.out.println("Employee found:");
```

```
                        employeeToSearch.display();
                    } else {
                        System.out.println("Employee with ID " + searchId + " not found.");
                    }
                    break;

                case 5:
                    if (employees.isEmpty()) {
                        System.out.println("No employees to display.");
                    } else {
                        System.out.println("\nList of all Employees:");
                        for (Employee emp : employees) {
                            emp.display();
                        }
                    }
                    break;

                case 6:
                    System.out.println("Exiting the program. Goodbye!");
                    scanner.close();
                    return;

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

2. Card Collection :

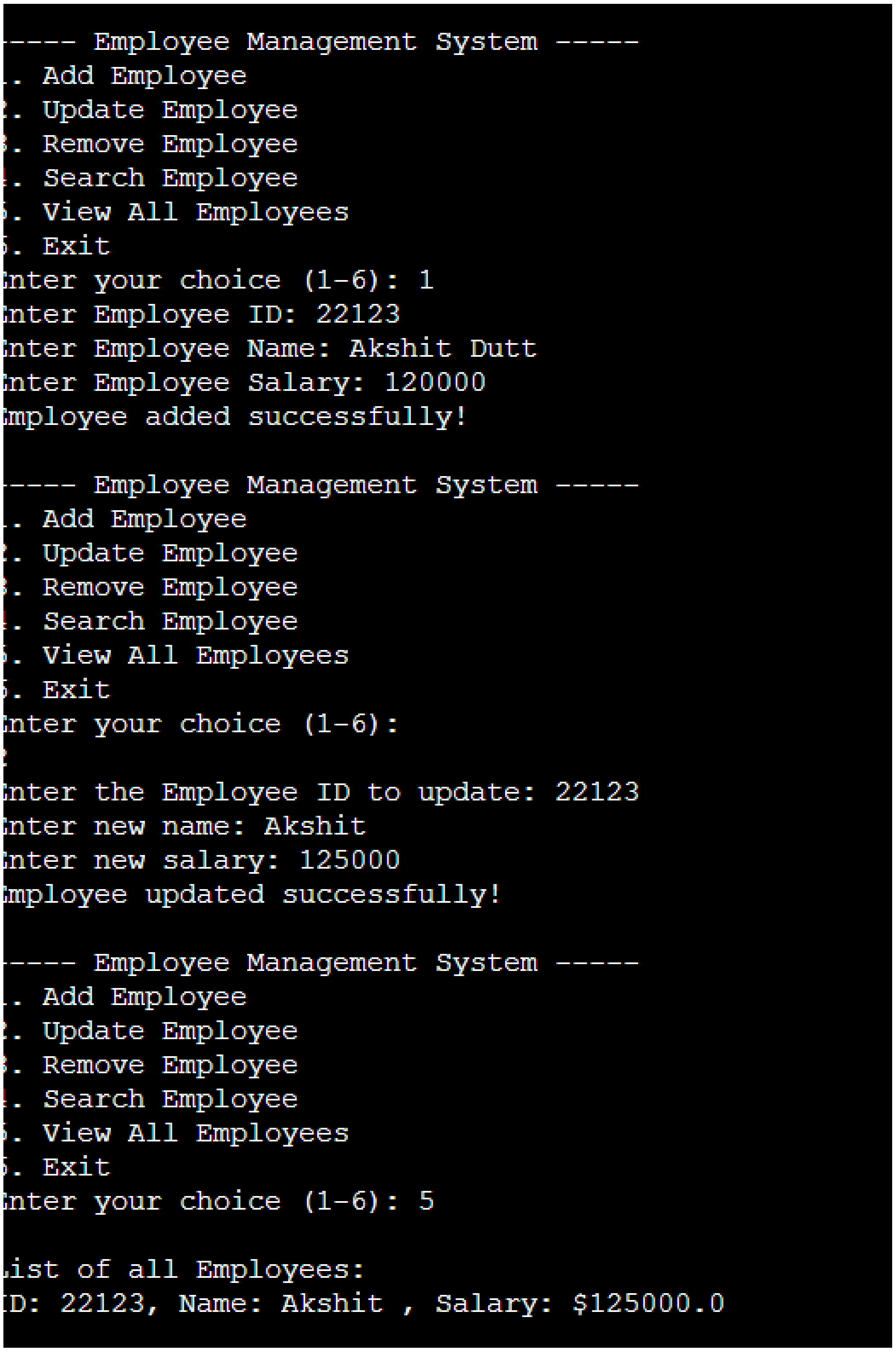
```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

class Card {
    String rank;
    String suit;

    public Card(String rank, String suit) {
        this.rank = rank;
        this.suit = suit;
    }

    @Override
    public String toString() {
        return rank + " of " + suit;
    }
}

public class CardCollection {

    public static void main(String[] args) {
        List<Card> deck = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);

        // Add cards to the deck
        String[] suits = {"Hearts", "Diamonds", "Clubs", "Spades"};
        String[] ranks = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King",
"Ace"};

        for (String suit : suits) {
            for (String rank : ranks) {
                deck.add(new Card(rank, suit));
            }
        }

        // User interaction to find cards by suit
        while (true) {
            System.out.println("\n----- Card Finder -----");
            System.out.println("1. Find cards by suit");
            System.out.println("2. Exit");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume the newline character

            if (choice == 1) {
                System.out.print("Enter the suit (Hearts, Diamonds, Clubs, Spades): ");
                String suit = scanner.nextLine();

                // Search for cards of the given suit
                System.out.println("Cards of suit " + suit + ":");
                boolean found = false;
                for (Card card : deck) {
                    if (card.suit.equalsIgnoreCase(suit)) {
                        System.out.println(card);
                        found = true;
                    }
                }
                if (!found) {
                    System.out.println("No cards found for the suit " + suit);
                }
            } else if (choice == 2) {
                System.out.println("Exiting the program. Goodbye!");
                break;
            } else {
                System.out.println("Invalid choice, please try again.");
            }
        }

        scanner.close();
    }
}
```

3. Ticket Booking System:

```
class TicketBooking {
    private boolean[] seats; // Array to store seat availability (true = booked, false = available)

    public TicketBooking(int totalSeats) {
        seats = new boolean[totalSeats];
    }

    // Synchronized method to book a seat
    public synchronized boolean bookSeat(int seatNumber, String customerType) {
        if (seatNumber < 0 || seatNumber >= seats.length) {
            System.out.println("Invalid seat number: " + seatNumber);
            return false;
        }

        if (seats[seatNumber]) {
            System.out.println(customerType + " failed to book seat " + seatNumber + " (Already 
booked)");
            return false;
        } else {
            seats[seatNumber] = true;
            System.out.println(customerType + " successfully booked seat " + seatNumber);
            return true;
        }
    }
}

class VIPBookingThread extends Thread {
    private TicketBooking ticketBooking;
    private int seatNumber;

    public VIPBookingThread(TicketBooking ticketBooking, int seatNumber) {
        this.ticketBooking = ticketBooking;
        this.seatNumber = seatNumber;
        setPriority(Thread.MAX_PRIORITY);  // Set VIP thread priority to maximum
    }

    @Override
    public void run() {
        ticketBooking.bookSeat(seatNumber, "VIP");
    }
}

class RegularBookingThread extends Thread {
    private TicketBooking ticketBooking;
    private int seatNumber;

    public RegularBookingThread(TicketBooking ticketBooking, int seatNumber) {
        this.ticketBooking = ticketBooking;
        this.seatNumber = seatNumber;
        setPriority(Thread.NORM_PRIORITY);  // Set regular thread priority to normal
    }

    @Override
    public void run() {
        ticketBooking.bookSeat(seatNumber, "Regular");
    }
}

public class TicketBookingSystem {
    public static void main(String[] args) {
        TicketBooking ticketBooking = new TicketBooking(5);  // Assume we have 5 seats
available

        // Create and start VIP threads (VIP bookings with high priority)
        VIPBookingThread vip1 = new VIPBookingThread(ticketBooking, 0);
        VIPBookingThread vip2 = new VIPBookingThread(ticketBooking, 1);

        // Create and start Regular threads (Regular bookings with normal priority)
        RegularBookingThread regular1 = new RegularBookingThread(ticketBooking, 1);
        RegularBookingThread regular2 = new RegularBookingThread(ticketBooking, 2);
        RegularBookingThread regular3 = new RegularBookingThread(ticketBooking, 3);

        // Start threads
        vip1.start();
        vip2.start();
        regular1.start();
        regular2.start();
        regular3.start();
    }
}
```

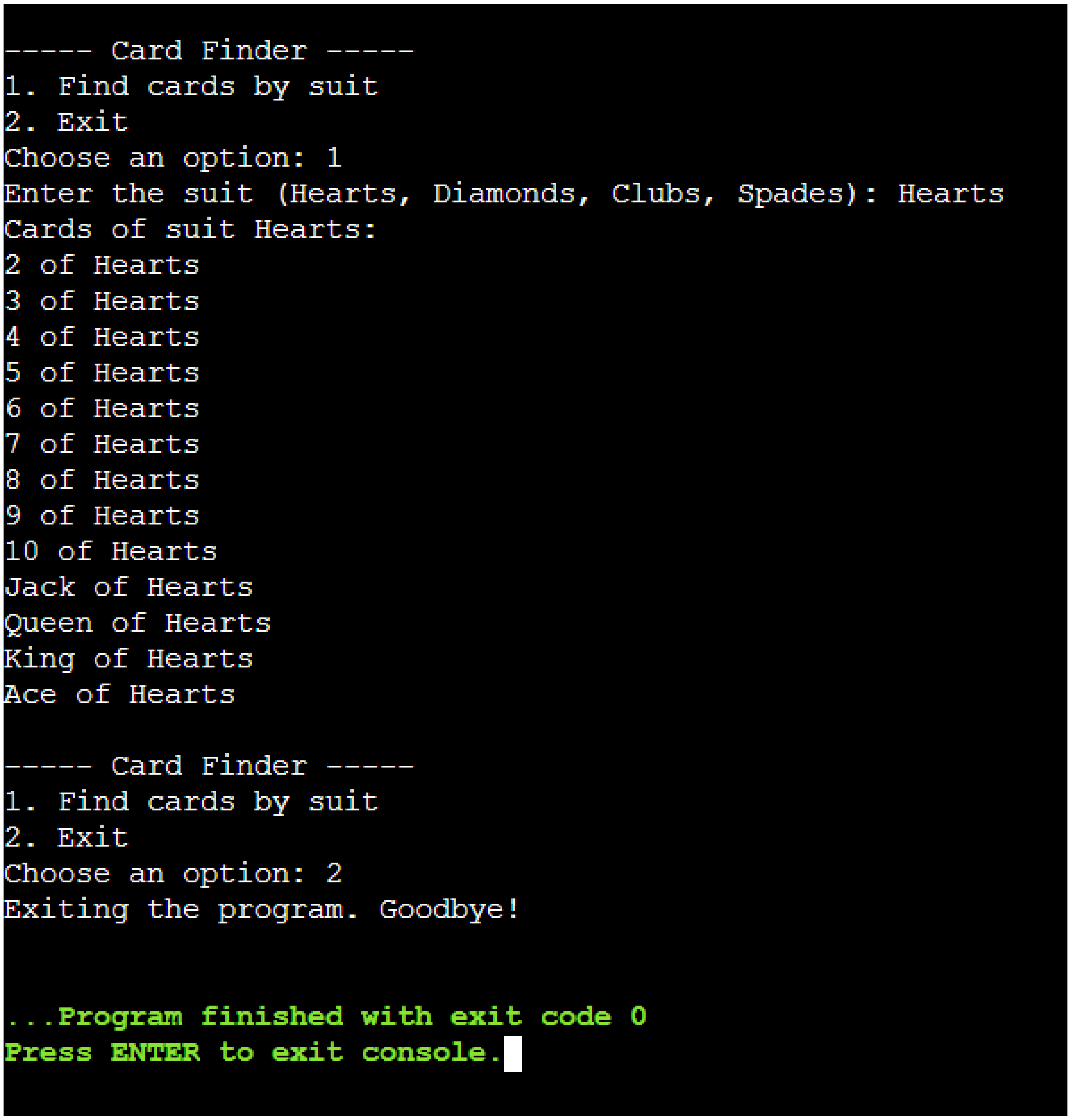
OUTPUT :

1. **Employee Management:**



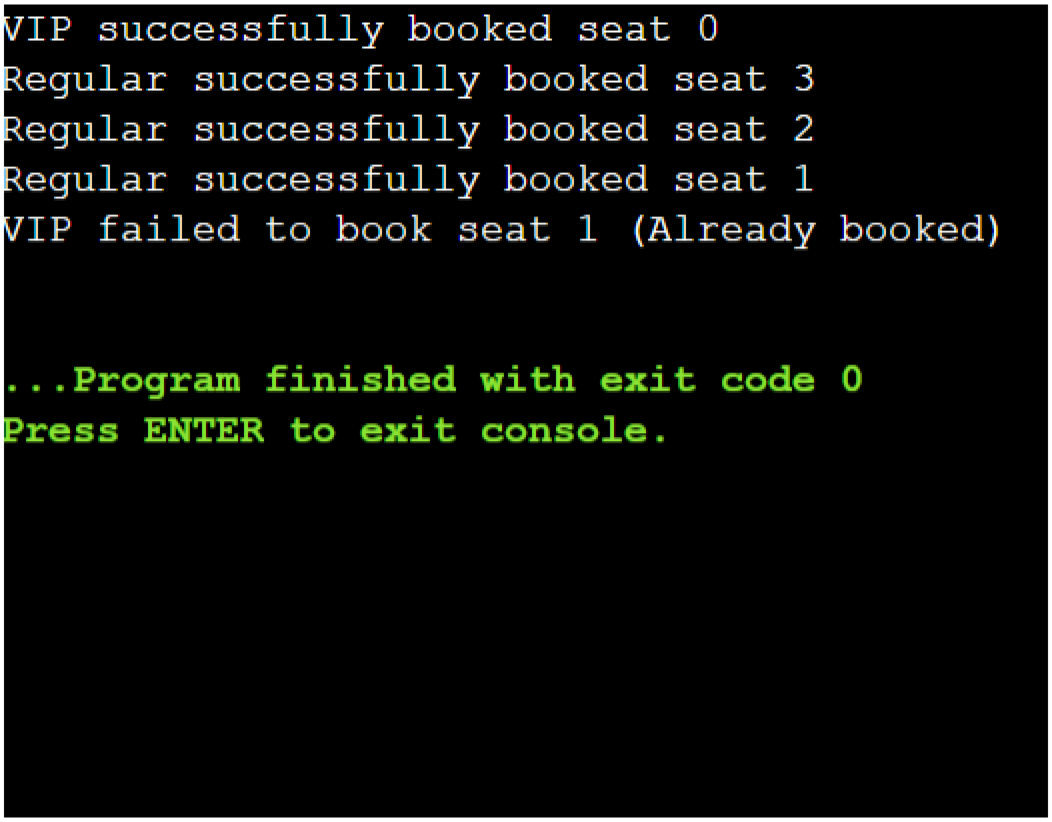
```
----- Employee Management System -----
. Add Employee
. Update Employee
. Remove Employee
. Search Employee
. View All Employees
. Exit
nter your choice (1-6): 1
nter Employee ID: 22123
nter Employee Name: Akshit Dutt
nter Employee Salary: 120000
mployee added successfully!

----- Employee Management System -----
. Add Employee
. Update Employee
. Remove Employee
. Search Employee
. View All Employees
. Exit
nter your choice (1-6):


nter the Employee ID to update: 22123
nter new name: Akshit
nter new salary: 125000
mployee updated successfully!

----- Employee Management System -----
. Add Employee
. Update Employee
. Remove Employee
. Search Employee
. View All Employees
. Exit
nter your choice (1-6): 5

ist of all Employees:
D: 22123, Name: Akshit , Salary: $125000.0
```

2. **Card Collection :**



```
----- Card Finder -----
1. Find cards by suit
2. Exit
Choose an option: 1
Enter the suit (Hearts, Diamonds, Clubs, Spades): Hearts
Cards of suit Hearts:
2 of Hearts
3 of Hearts
4 of Hearts
5 of Hearts
6 of Hearts
7 of Hearts
8 of Hearts
9 of Hearts
10 of Hearts
Jack of Hearts
Queen of Hearts
King of Hearts
Ace of Hearts

----- Card Finder -----
1. Find cards by suit
2. Exit
Choose an option: 2
Exiting the program. Goodbye!


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

3. **Ticket Booking System:**



```
VIP successfully booked seat 0
Regular successfully booked seat 3
Regular successfully booked seat 2
Regular successfully booked seat 1
VIP failed to book seat 1 (Already booked)


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

Learning Outcomes:

- **Object-Oriented Design** (Classes for real-world entities)
- **Core Programming Skills** (Loops, conditionals, methods for inventory operations)
- **Data Structure Usage** (ArrayList for dynamic data management)
- **User-Friendly Systems** (Intuitive interface with error handling)