Experiment 4

Student Name: Manvendra Singh UID: 22BCS16432
Branch: BE-CSE Section/Group: DL_903_B
Semester: 6th Date of Performance: 11-02-2025
Subject Name: Program Based Learning in Java with Lab Subject Code: 22CSH-359

1. Aim: Use of Collections in Java. ArrayList, LinkedList, HashMap, TreeMap, HashSet in Java. Multithreading in Java. Thread Synchronization. Thread Priority, Thread LifeCycle.Develop Java programs using core concepts such as data structures, collections, and multithreading to manage and manipulate data.

2. Implementation/Code:

1.) Easy: 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.

Code:

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

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 {
    static ArrayList<Employee> employees = new ArrayList<>();
    static Scanner scanner = new Scanner(System.in);
          public static void addEmployee() {
          System.out.print("Enter Employee ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();  // Consume newline
        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!");
    }

    public static void updateEmployee() {
        System.out.print("Enter Employee ID to update: ");
        int id = scanner.nextInt();
        for (Employee emp : employees) {
            if (emp.id == id) {
                scanner.nextLine(); // Consume newline
                System.out.print("Enter New Name: ");
                emp.name = scanner.nextLine();
                System.out.print("Enter New Salary: ");
                emp.salary = scanner.nextDouble();
                System.out.println("Employee updated successfully!");
                return;
            }
        }
        System.out.println("Employee not found!");
    }

    public static void removeEmployee() {
        System.out.print("Enter Employee ID to remove: ");
        int id = scanner.nextInt();
        boolean removed = employees.removeIf(emp -> emp.id == id);
        if (removed) {
            System.out.println("Employee removed successfully!");
        } else {
```

```
        System.out.println("Employee not found!");
    }
  }

  public static void searchEmployee() {


 System.out.print("Enter Employee ID to search: ");
    int id = scanner.nextInt();
    for (Employee emp : employees) {
      if (emp.id == id) {
        System.out.println("Employee Found: " + emp);
        return;
      }
    }
    System.out.println("Employee not found!");
  }


      public static void displayAllEmployees() {
          if (employees.isEmpty()) {
            System.out.println("No employees in the system.");
          } else {
            System.out.println("Employee List:");
            for (Employee emp : employees) {
              System.out.println(emp);
            }
    }
  }

  public static void main(String[] args) {
    while (true) {
      System.out.println("\n1. Add Employee  2. Update Employee  3. Remove Employee  4. Search
Employee  5. Display All Employees  6. Exit");
      System.out.print("Enter your choice: ");
      int choice = scanner.nextInt();
      switch (choice) {
        case 1 -> addEmployee();
        case 2 -> updateEmployee();
        case 3 -> removeEmployee();
        case 4 -> searchEmployee();
        case 5 -> displayAllEmployees();
        case 6 -> {
          System.out.println("Exiting...");
          System.exit(0);
        }
        default -> System.out.println("Invalid choice! Try again.");
      }
    }
```

```
   }
```

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.

Code:

```
import java.util.*;

class CardCollection {
    private Map<String, Collection<String>> cardMap = new HashMap<>();
    private Scanner scanner = new Scanner(System.in);

    // Add a card
    public void addCard() {
        System.out.print("Enter Card Symbol (e.g., Spades, Hearts): ");
        String symbol = scanner.nextLine().trim();
        System.out.print("Enter Card Name (e.g., Ace, King, Queen): ");
        String cardName = scanner.nextLine().trim();

        cardMap.putIfAbsent(symbol, new ArrayList<>()); // Ensure symbol exists
        cardMap.get(symbol).add(cardName);
        System.out.println("Card added successfully!");
    }

    // Find cards by symbol
    public void findCards() {
        System.out.print("Enter Symbol to search: ");
        String symbol = scanner.nextLine().trim();

        if (cardMap.containsKey(symbol)) {
            System.out.println("Cards with symbol " + symbol + ": " + cardMap.get(symbol));
        } else {
            System.out.println("No cards found for this symbol.");
        }
    }

    // Remove a specific card
    public void removeCard() {
        System.out.print("Enter Symbol of the card to remove: ");
        String symbol = scanner.nextLine().trim();
```

```
            if (cardMap.containsKey(symbol)) {
                System.out.print("Enter Card Name to remove: ");
                String cardName = scanner.nextLine().trim();
                Collection<String> cards = cardMap.get(symbol);

                if (cards.remove(cardName)) {
                    System.out.println("Card removed successfully!");
                    if (cards.isEmpty()) {
                        cardMap.remove(symbol); // Remove symbol if no cards left


         }
                } else {

System.out.println("Card not found!");
  }
   } else {
                System.out.println("No such symbol exists!");
            }
        }

        // Display all cards
        public void displayAllCards() {
            if (cardMap.isEmpty()) {
                System.out.println("No cards in the collection.");
            } else {
                System.out.println("Card Collection:");
                for (Map.Entry<String, Collection<String>> entry : cardMap.entrySet()) {
                    System.out.println(entry.getKey() + " -> " + entry.getValue());
                }
            }
        }

        // Start the interactive menu
        public void start() {
            while (true) {
                System.out.println("\n1. Add Card  2. Find Cards  3. Remove Card  4. Display All Cards  5. Exit");
                System.out.print("Enter choice: ");
                int choice = scanner.nextInt();
                scanner.nextLine();  // Consume newline
```

```
 switch (choice) {
             case 1 -> addCard();
             case 2 -> findCards();
             case 3 -> removeCard();
             case 4 -> displayAllCards();
             case 5 -> {
                 System.out.println("Exiting...");
                 return;
             }
             default -> System.out.println("Invalid choice! Try again.");
         }

}
   }

   public static void main(String[] args) {
       CardCollection collection = new CardCollection();
       collection.start();
   }
}
```

3.) Hard: 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.

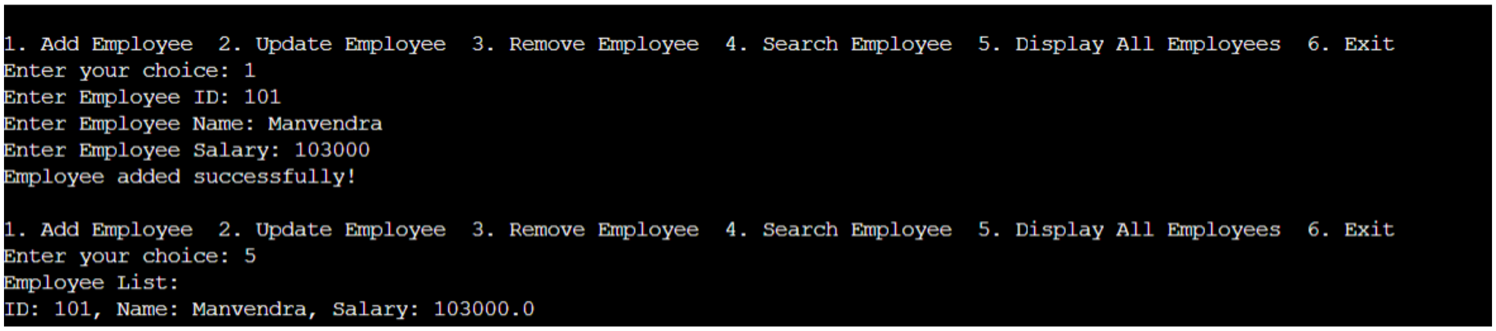
Code:

```
import java.util.*;

class TicketBookingSystem {
    private final int totalSeats = 10; // Number of available seats
    private final boolean[] seats = new boolean[totalSeats]; // Seat availability
    private final Object lock = new Object(); // Lock for synchronization

    public void bookTicket(String user, int seatNumber) {
        synchronized (lock) {
            if (seatNumber < 0 || seatNumber >= totalSeats) {
                System.out.println(user + " tried to book an invalid seat: " + seatNumber);
                return;
            }
            if (!seats[seatNumber]) { // If seat is available
                seats[seatNumber] = true; // Mark seat as booked
                System.out.println(user + " successfully booked Seat " + seatNumber);
        } else {
                System.out.println(user + " failed to book Seat " + seatNumber + " (Already booked!)");
            }
        }
    }

    public void displayAvailableSeats() {
        synchronized (lock) {
            System.out.print("Available Seats: ");
            for (int i = 0; i < totalSeats; i++) {
                if (!seats[i]) System.out.print(i + " ");
            }
            System.out.println();
        }
    }
}

class BookingThread extends Thread {
    private final TicketBookingSystem system;
    private final String user;
    private final int seatNumber;

    public BookingThread(TicketBookingSystem system, String user, int seatNumber, int priority) {
        this.system = system;
        this.user = user;
        this.seatNumber = seatNumber;
        this.setPriority(priority); // Set thread priority
    }

    @Override
    public void run() {
        system.bookTicket(user, seatNumber);
    }
}

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

        // Display initial available seats
        system.displayAvailableSeats();

        // Creating VIP and Regular users
```

```
  BookingThread vip1 = new BookingThread(system, "VIP User 1", random.nextInt(10),
Thread.MAX_PRIORITY);
      BookingThread vip2 = new BookingThread(system, "VIP User 2", random.nextInt(10),
Thread.MAX_PRIORITY);
      BookingThread user1 = new BookingThread(system, "Regular User 1", random.nextInt(10),
Thread.NORM_PRIORITY);
      BookingThread user2 = new BookingThread(system, "Regular User 2", random.nextInt(10),
Thread.NORM_PRIORITY);
      BookingThread user3 = new BookingThread(system, "Regular User 3", random.nextInt(10),
Thread.MIN_PRIORITY);

      // Start booking threads
      vip1.start();
      vip2.start();
      user1.start();
      user2.start();
      user3.start();

      // Wait for all threads to complete
      try {
         vip1.join();
         vip2.join();
         user1.join();
         user2.join();
         user3.join();
      } catch (InterruptedException e) {
         e.printStackTrace();
      }

      // Display final available seats
      system.displayAvailableSeats();
   }
}
```

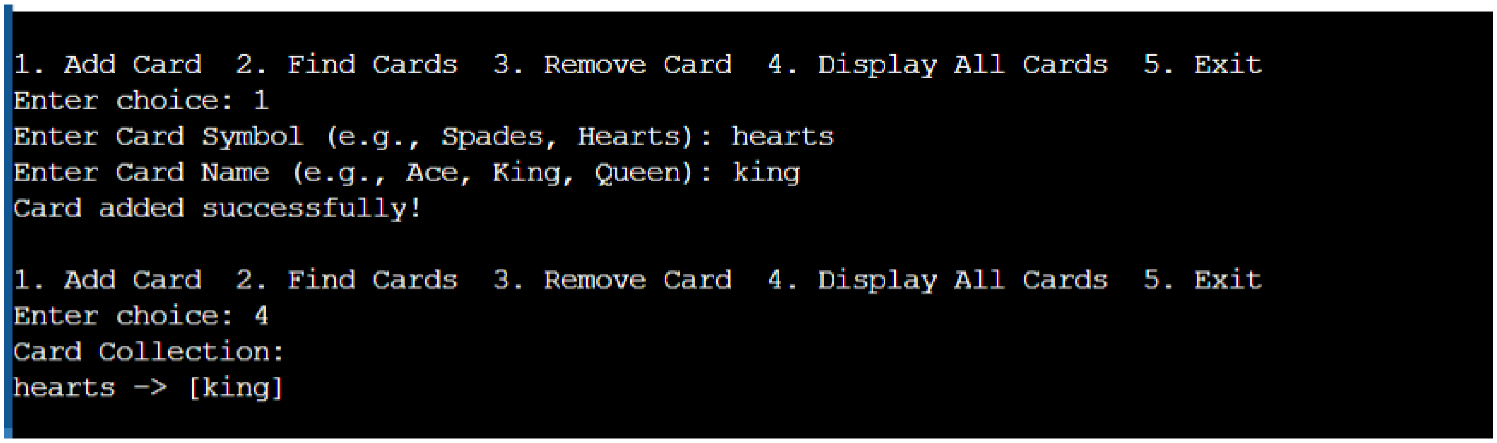
5. Output

1.) Easy problem output



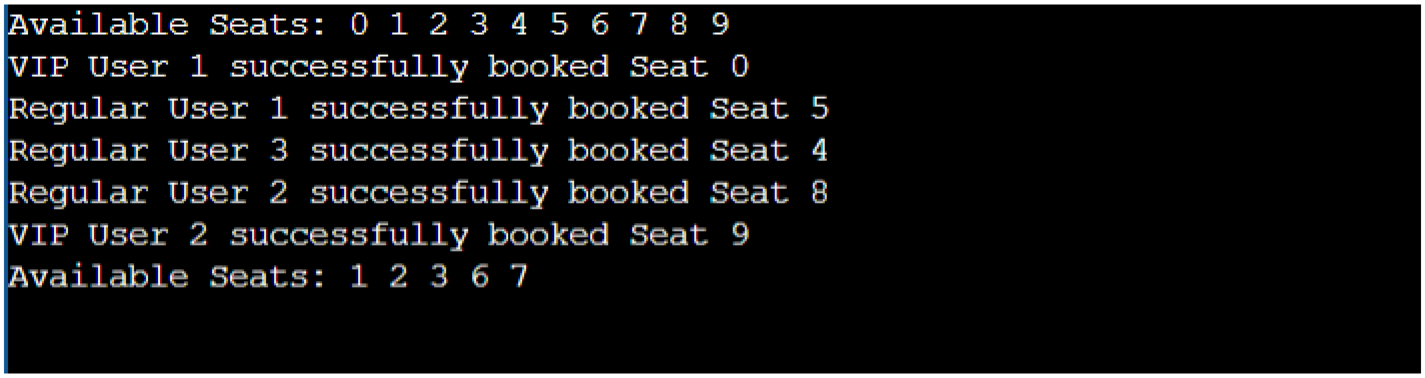
```
1. Add Employee  2. Update Employee  3. Remove Employee  4. Search Employee  5. Display All Employees  6. Exit
Enter your choice: 1
Enter Employee ID: 101
Enter Employee Name: Manvendra
Enter Employee Salary: 103000
Employee added successfully!

1. Add Employee  2. Update Employee  3. Remove Employee  4. Search Employee  5. Display All Employees  6. Exit
Enter your choice: 5
Employee List:
ID: 101, Name: Manvendra, Salary: 103000.0
```

2.) Medium problem output



```
1. Add Card  2. Find Cards  3. Remove Card  4. Display All Cards  5. Exit
Enter choice: 1
Enter Card Symbol (e.g., Spades, Hearts): hearts
Enter Card Name (e.g., Ace, King, Queen): king
Card added successfully!

1. Add Card  2. Find Cards  3. Remove Card  4. Display All Cards  5. Exit
Enter choice: 4
Card Collection:
hearts -> [king]
```

3.) Hard problem output



```
Available Seats: 0 1 2 3 4 5 6 7 8 9
VIP User 1 successfully booked Seat 0
Regular User 1 successfully booked Seat 5
Regular User 3 successfully booked Seat 4
Regular User 2 successfully booked Seat 8
VIP User 2 successfully booked Seat 9
Available Seats: 1 2 3 6 7
```

3.Learning Outcomes

1. Learned to use classes and objects for organizing employee and designation data in Java.
2. Implemented salary calculations using switch-case and array data handling.
3. Practiced input handling with the Scanner class and validating user input.
4. Gained experience in searching arrays and structuring conditional logic.

Displayed formatted output for real-world applications like employee management systems