Experiment 5.3

1. **Aim:** Create a menu-based Java application with the following options.
   1. Add an Employee
   2. Display All
   3. Exit If option 1 is selected, the application should gather details of the employee like employee name, employee id, designation and salary and store it in a file. If option 2 is selected, the application should display all the employee details. If option 3 is selected the application should exit.
2. **Objective:** The objective is to develop a menu-based Java application that allows users to **add employee details**, **store them in a file**, and **display all stored employee records**, with an option to exit the program.

3. **Algorithm:**

Step 1: Initialize the Program

1. Start the program.
2. Import java.util.* and java.util.concurrent.* for thread handling.
3. Define a class TicketBookingSystem with:
   - A List<Boolean> representing seat availability (true for available, false for booked).
   - A synchronized method bookSeat(int seatNumber, String passengerName) to ensure thread safety.

Step 2: Implement Seat Booking Logic

1. Define bookSeat(int seatNumber, String passengerName):
   - If the seat is available (true), mark it as booked (false).
   - Print confirmation: "Seat X booked successfully by Y".
   - If already booked, print: "Seat X is already booked."

Step 3: Define Booking Threads

1. Create a class PassengerThread extending Thread:
   - Store passenger name, seat number, and booking system reference.
   - Implement run() method to call bookSeat().

Step 4: Assign Thread Priorities

1. Create VIP and Regular passenger threads.
2. Set higher priority for VIP passengers using setPriority(Thread.MAX_PRIORITY).
3. Set default priority for regular passengers.

Step 5: Handle User Input & Simulate Booking

1. In main(), create an instance of TicketBookingSystem.
2. Accept number of seats and bookings from the user.
3. Create multiple PassengerThread instances for VIP and regular passengers.
4. Start all threads using start().

Step 6: Synchronization & Preventing Double Booking

1. Use the synchronized keyword in bookSeat() to ensure only one thread accesses it at a time.
2. Ensure thread execution order by assigning higher priority to VIP threads.

Step 7: Display Final Booking Status

1. After all threads finish execution, display the list of booked seats.
2. End the program with a message: "All bookings completed successfully."

4. Implementation Code:

```
import java.io.*;
import java.util.*;
```

```
// Employee class implementing Serializable
class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String name;
    private String designation;
    private double salary;

    public Employee(int id, String name, String designation, double salary) {
        this.id = id;
        this.name = name;
        this.designation = designation;
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Employee ID: " + id + ", Name: " + name + ", Designation: " + designation + ", Salary: " +
salary;
    }
}

// Employee Management System
public class EmployeeManagementSystem {
    private static final String FILE_NAME = "employees.ser";
    private static List<Employee> employees = new ArrayList<>();

    // Method to add an employee
    public static void addEmployee() {
        Scanner scanner = new Scanner(System.in);

        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 Designation: ");
        String designation = scanner.nextLine();

        System.out.print("Enter Salary: ");
        double salary = scanner.nextDouble();

        Employee employee = new Employee(id, name, designation, salary);
        employees.add(employee);
        saveEmployees();

        System.out.println("Employee added successfully!");
    }

    // Method to display all employees
    public static void displayAllEmployees() {
```

```
        loadEmployees();

        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            System.out.println("\nEmployee List:");
            for (Employee employee : employees) {
                System.out.println(employee);
            }
        }
    }

    // Method to save employees to a file
    private static void saveEmployees() {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE_NAME))) {
            oos.writeObject(employees);
        } catch (IOException e) {
            System.err.println("Error saving employees: " + e.getMessage());
        }
    }

    // Method to load employees from a file
    @SuppressWarnings("unchecked")
    private static void loadEmployees() {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_NAME))) {
            employees = (List<Employee>) ois.readObject();
        } catch (FileNotFoundException e) {
            employees = new ArrayList<>();
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Error loading employees: " + e.getMessage());
        }
    }

    // Main method for menu-driven program
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nEmployee Management System");
            System.out.println("1. Add an Employee");
            System.out.println("2. Display All Employees");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    addEmployee();
                    break;
                case 2:
                    displayAllEmployees();
```

```
                break;
            case 3:
                System.out.println("Exiting...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice! Please try again.");
        }
    }
}
```

5. Output:

```
******* WELCOME TO Employee Management System *******

Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 1
Enter Employee ID: 18
Enter Employee Name: Anshu Kumar
Enter Designation: Senior developer
Enter Salary: 400000
Employee added successfully!
******* WELCOME TO Employee Management System *******

Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 2

Employee List:
Employee ID: 18, Name: Anshu Kumar, Designation: Senior developer, Salary: 400000.0
******* WELCOME TO Employee Management System *******

Employee Management System
1. Add an Employee
2. Display All Employees
3. Exit
Enter your choice: 3
Exiting...


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

6. Learning Outcomes:

- Understand file handling and serialization in Java to store and retrieve objects persistently.
- Learn how to implement a menu-driven console application using loops and conditional statements.
- Gain experience in object-oriented programming (OOP) by defining and managing Employee objects.
- Practice exception handling to manage file-related errors like FileNotFoundException and IOException.
- Develop skills in list manipulation and user input handling using ArrayList and Scanner.