RA2411026010738

Abhishek kumar A

Q1. Hospital Appointment Booking System

Requirements:

- Book, reschedule, cancel appointments
- View doctor schedules

<u>Procedural Approach (Imperative)</u>:

```
struct Appointment {
   int patientId;
   int doctorId;
   char date[20];
   char time[10];
};

void bookAppointment(struct Appointment a);
void rescheduleAppointment(int appointmentId, char* newDate, char* newTime);
void cancelAppointment(int appointmentId);
void viewDoctorSchedule(int doctorId);
```

<u>Object-Oriented Approach (OOP):</u>

```
class Appointment {
   int id;
   int patientId;
   int doctorId;
   String date;
   String time;

   void book();
```

```
  void reschedule(String newDate, String newTime);
  void cancel();
}


class Doctor {
  int doctorId;
  List<Appointment> schedule;


  void viewSchedule();
}
```

Memory Overhead Comparison:

- **Procedural:** Subroutines use the **stack**, low overhead.
- **OOP:** Methods are tied to objects, adding **heap memory usage** due to object instantiation.
- OOP may have slightly more overhead due to **virtual tables**, **objects**, and **method resolution**, but is more maintainable.

Q2. Food Delivery App – Live Tracking with Parallel Processing

Key Focus: Show how parallel processing helps handle multiple live updates.

Example (Sequential vs Parallel):

Sequential (Procedural in Python):

```
for order in orders:
  update_order_status(order)
```

Parallel (Using threading):

```
import threading
for order in orders:
  threading.Thread(target=update_order_status, args=(order,)).start()
```

Comparison:

- **Sequential**: Updates are handled one by one – slow and blocking.

- **Parallel**: Real-time updates, faster, responsive for multiple users.

Q3. Library Book Search – Declarative vs Procedural

Declarative Approach (e.g., SQL or Functional):

```
SELECT * FROM books WHERE title = 'Data Structures';
```

- High-level, tells **what** to do, not **how**.

Procedural Approach:

```
for book in books:
  if book['title'] == "Data Structures":
    print(book)
```

- Low-level control, explicitly defines steps.

Comparison:

- Declarative: More concise, easier to read.
- Procedural: More flexible, but verbose.

Q4. Course Registration System

Requirements:

- Enroll in courses
- Drop courses
- View timetable

Procedural Approach:

```
void enrollCourse(int studentId, int courseId);
void dropCourse(int studentId, int courseId);
void viewTimetable(int studentId);
```

Object-Oriented Approach:

```
class Course {
  int id;
  String name;
}

class Student {
  int id;
  List<Course> enrolledCourses;

  void enroll(Course course);
  void drop(Course course);
  void viewTimetable();
}
```