**Smart Traffic Signal Optimization**

Maithreyee. J

192325058

CSA0959

Programming in Java for Scientific Applications

Question:

You are part of a team working on an initiative to optimize traffic signal management in a busy city to reduce congestion and improve traffic flow efficiency using smart technologies.

**1. Data Collection and Modeling**

To collect and manage real-time traffic data from sensors, we'll define data structures that can store this information efficiently. We'll have two main entities: **TrafficSensorData** and **Intersection**. Here’s how we can structure the data:

**TrafficSensorData**

This class will represent the data collected from traffic sensors at various intersections.

**Attributes:**

* **sensorID**: Unique identifier for the sensor.
* **intersectionID**: Foreign key linking to the intersection where the sensor is located.
* **timestamp**: The time at which the data is collected.
* **vehicleCount**: Number of vehicles detected by the sensor.
* **averageSpeed**: Average speed of the vehicles detected.
* **trafficDensity**: Calculated density of the traffic (vehicles per unit area).
* **queueLength**: Length of the vehicle queue at the intersection.
* **pedestrianCrossingCount**: Number of pedestrians waiting to cross.

public class TrafficSensorData {

private String sensorID;

private String intersectionID;

private long timestamp;

private int vehicleCount;

private double averageSpeed;

private double trafficDensity;

private int queueLength;

private int pedestrianCrossingCount;

}

2. Algorithm Design

Optimization Algorithm:

The core of the system is the algorithm that analyzes the collected data and adjusts the signal timings to optimize traffic flow. The algorithm considers various factors such as:

Traffic Density: High traffic density might indicate the need for longer green lights.

Queue Length: Long vehicle queues might require an extension of green lights to clear the congestion.

Pedestrian Crossings: If there are many pedestrians waiting, the system should prioritize pedestrian signals to ensure safety.

3. Implementation

Real-time Integration:

The implementation involves integrating with traffic sensors to collect data and adjusting traffic signal timings in real-time. This requires:

TrafficSensorData and Intersection classes to model the data.

TrafficSignalOptimizer class to implement the optimization algorithm.

TrafficSignalController manages the intersections and periodically updates the signal timings based on the collected data.

4. Visualization and Reporting

Data Flow Diagram: A diagram illustrating how data flows from sensors to the optimization algorithm and then to the traffic signal controllers helps in understanding the overall system architecture.

DATA COLLECTION

TRAFFIC SENSORS

DATA ANALYSIS

SIGNAL OPTIMIZATION

VISUALIZATION

REAL- TIME ADJUSTMENT

5. User Interaction

User Interface:

A user-friendly interface is crucial for traffic managers to monitor real-time traffic conditions and manually adjust signal timings if needed. The interface should include:

Dashboards: Visualize traffic data, signal timings, and historical data.

Control Panels: Allow manual adjustments to signal timings.

Documentation

Design Decisions:

Data Structures: Chosen to effectively represent traffic sensor data and intersections.

Algorithms: Designed to dynamically optimize traffic signals based on real-time data.

Assumptions: Assumed high reliability of traffic sensors and accurate data.

Potential Improvements:

Incorporate machine learning for predictive traffic flow optimization.

Improve sensor reliability and data accuracy.

Integrate with other smart city infrastructure (e.g., public transport schedules).

Testing

Test Cases:

Normal Traffic Conditions: Ensure the system optimizes signal timings correctly.

High Traffic Density: Verify the system extends green light duration for high-density directions.

Long Vehicle Queues: Confirm the system handles long queues by adjusting signal timings.

Pedestrian Crossings: Check the system activates pedestrian signals when needed.

**java final codes:**

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

// Step 1: Define the IntersectionData class to hold sensor data

class IntersectionData {

private String intersectionID;

private String signalID;

private int queueLength;

private int averageSpeed;

public IntersectionData(String intersectionID, String signalID, int queueLength, int averageSpeed) {

this.intersectionID = intersectionID;

this.signalID = signalID;

this.queueLength = queueLength;

this.averageSpeed = averageSpeed;

}

public String getIntersectionID() {

return intersectionID;

}

public String getSignalID() {

return signalID;

}

public int getQueueLength() {

return queueLength;

}

public int getAverageSpeed() {

return averageSpeed;

}

@Override

public String toString() {

return "Intersection " + intersectionID + " - Signal " + signalID + " | Queue Length: " + queueLength + ", Average Speed: " + averageSpeed + " km/h";

}

}

// Step 2: Create a service to fetch real-time sensor data

class SensorDataService {

private Map<String, IntersectionData> dataMap;

public SensorDataService() {

this.dataMap = new HashMap<>();

}

public void addData(String intersectionID, IntersectionData data) {

dataMap.put(intersectionID, data);

}

public Map<String, IntersectionData> getRealTimeData() {

return dataMap;

}

}

// Step 3: Create a service to control signal timings

class SignalControlService {

public void updateSignalTiming(String signalID, int phaseDuration) {

System.out.println("Updating signal " + signalID + " with duration: " + phaseDuration + " seconds.");

}

}

// Step 4: Define the main TrafficSignalController class to optimize signals

class TrafficSignalController {

private SensorDataService sensorDataService;

private SignalControlService signalControlService;

public TrafficSignalController(SensorDataService sensorService, SignalControlService controlService) {

this.sensorDataService = sensorService;

this.signalControlService = controlService;

}

public void optimizeTrafficSignals() {

Map<String, IntersectionData> intersections = sensorDataService.getRealTimeData();

for (IntersectionData data : intersections.values()) {

int phaseDuration = calculateOptimalPhaseDuration(data);

signalControlService.updateSignalTiming(data.getSignalID(), phaseDuration);

}

}

private int calculateOptimalPhaseDuration(IntersectionData data) {

int queueLength = data.getQueueLength();

// Simple formula: phase duration is queue length divided by 2, within a range

return Math.max(30, Math.min(120, queueLength \* 2));

}

public void displayTrafficConditions() {

for (IntersectionData data : sensorDataService.getRealTimeData().values()) {

System.out.println(data.toString());

}

}

public String generateReport() {

int totalQueueLength = 0;

int totalAverageSpeed = 0;

int count = 0;

for (IntersectionData data : sensorDataService.getRealTimeData().values()) {

totalQueueLength += data.getQueueLength();

totalAverageSpeed += data.getAverageSpeed();

count++;

}

double avgQueueLength = (double) totalQueueLength / count;

double avgSpeed = (double) totalAverageSpeed / count;

StringBuilder report = new StringBuilder();

report.append("Traffic Flow Report\n");

report.append("===================\n");

report.append("Average Queue Length: ").append(avgQueueLength).append("\n");

report.append("Average Speed: ").append(avgSpeed).append(" km/h\n");

return report.toString();

}

}

// Main class to run the program

public class TrafficManagementSystem {

public static void main(String[] args) {

// Instantiate services

SensorDataService sensorService = new SensorDataService();

SignalControlService controlService = new SignalControlService();

TrafficSignalController controller = new TrafficSignalController(sensorService, controlService);

// Get user input

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of intersections:");

int numberOfIntersections = scanner.nextInt();

scanner.nextLine(); // Consume newline

for (int i = 0; i < numberOfIntersections; i++) {

System.out.println("Enter details for intersection " + (i + 1) + ":");

System.out.print("Intersection ID: ");

String intersectionID = scanner.nextLine();

System.out.print("Signal ID: ");

String signalID = scanner.nextLine();

System.out.print("Queue Length: ");

int queueLength = scanner.nextInt();

System.out.print("Average Speed: ");

int averageSpeed = scanner.nextInt();

scanner.nextLine(); // Consume newline

IntersectionData data = new IntersectionData(intersectionID, signalID, queueLength, averageSpeed);

sensorService.addData(intersectionID, data);

}

// Optimize traffic signals and display conditions

controller.optimizeTrafficSignals();

controller.displayTrafficConditions();

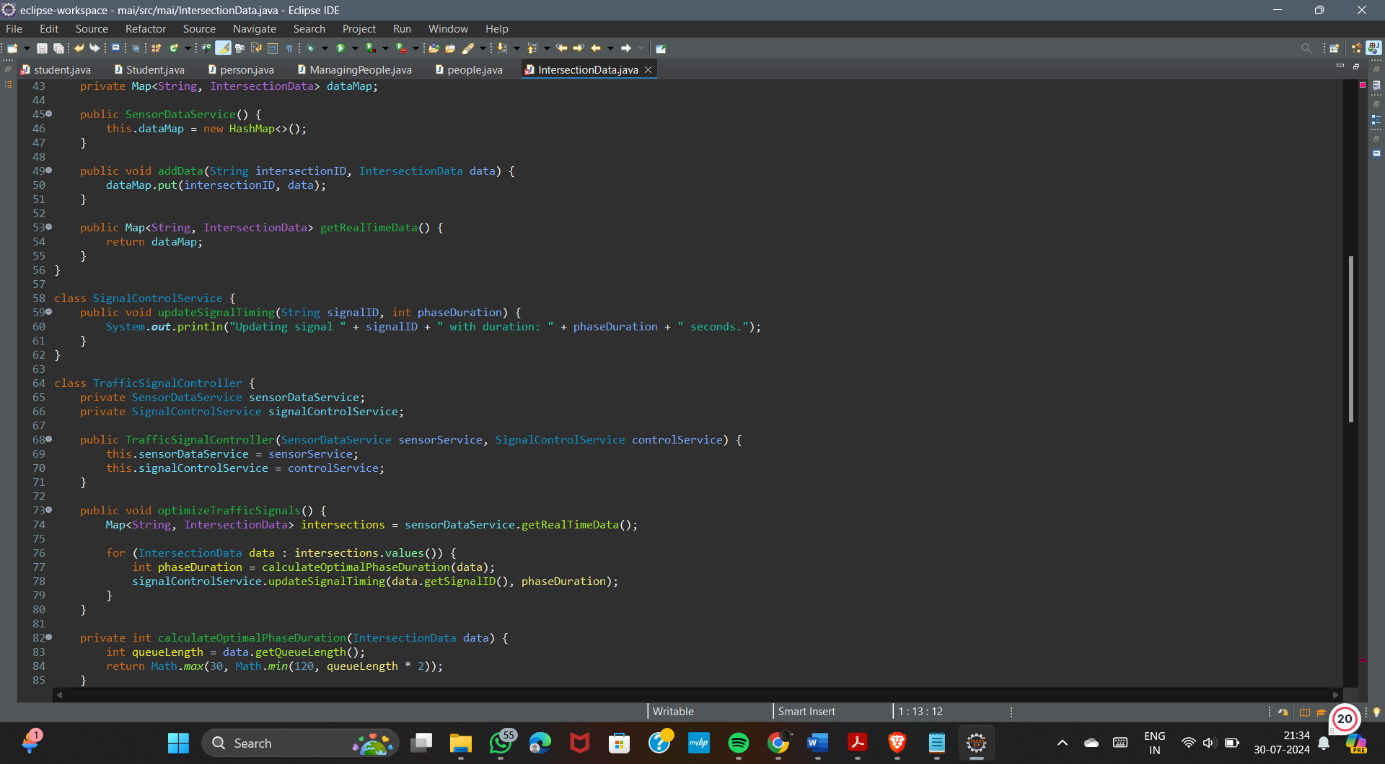
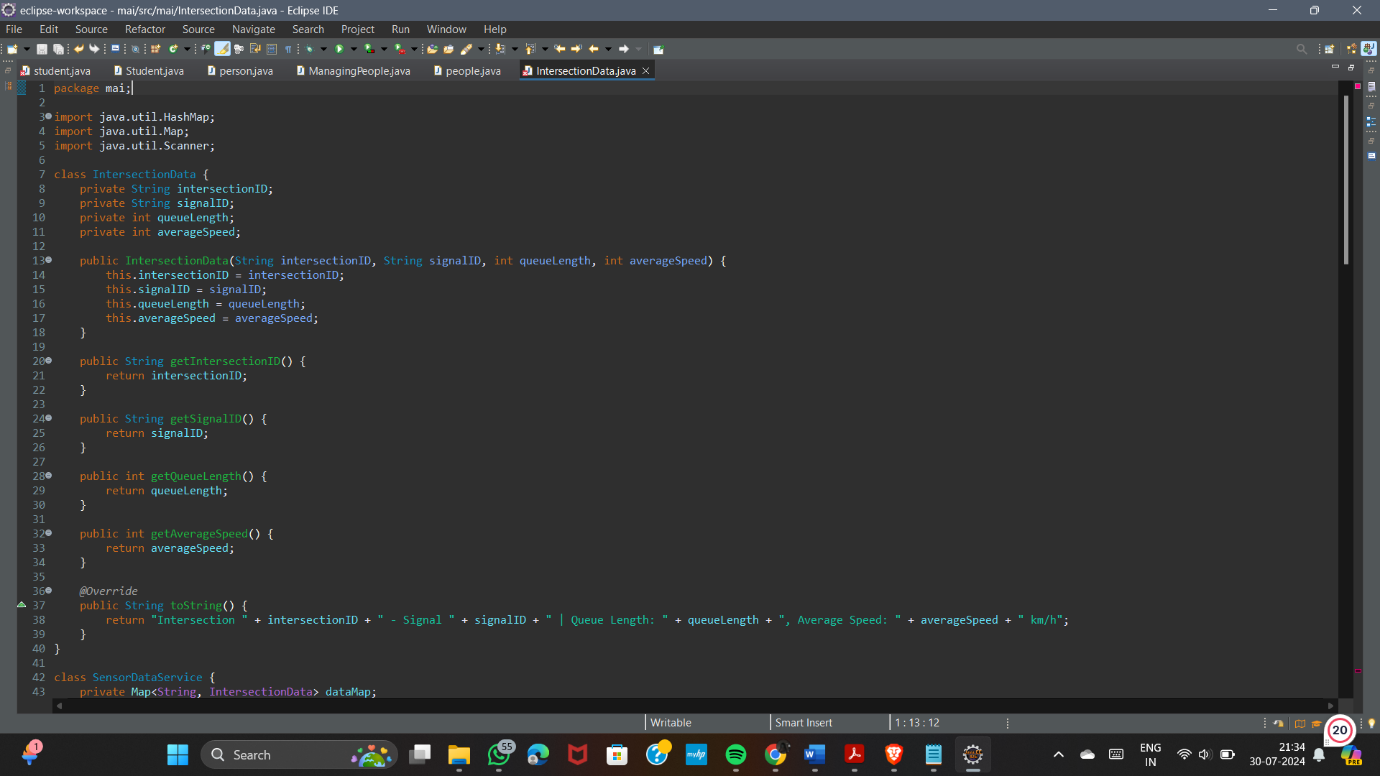
// Generate and display report

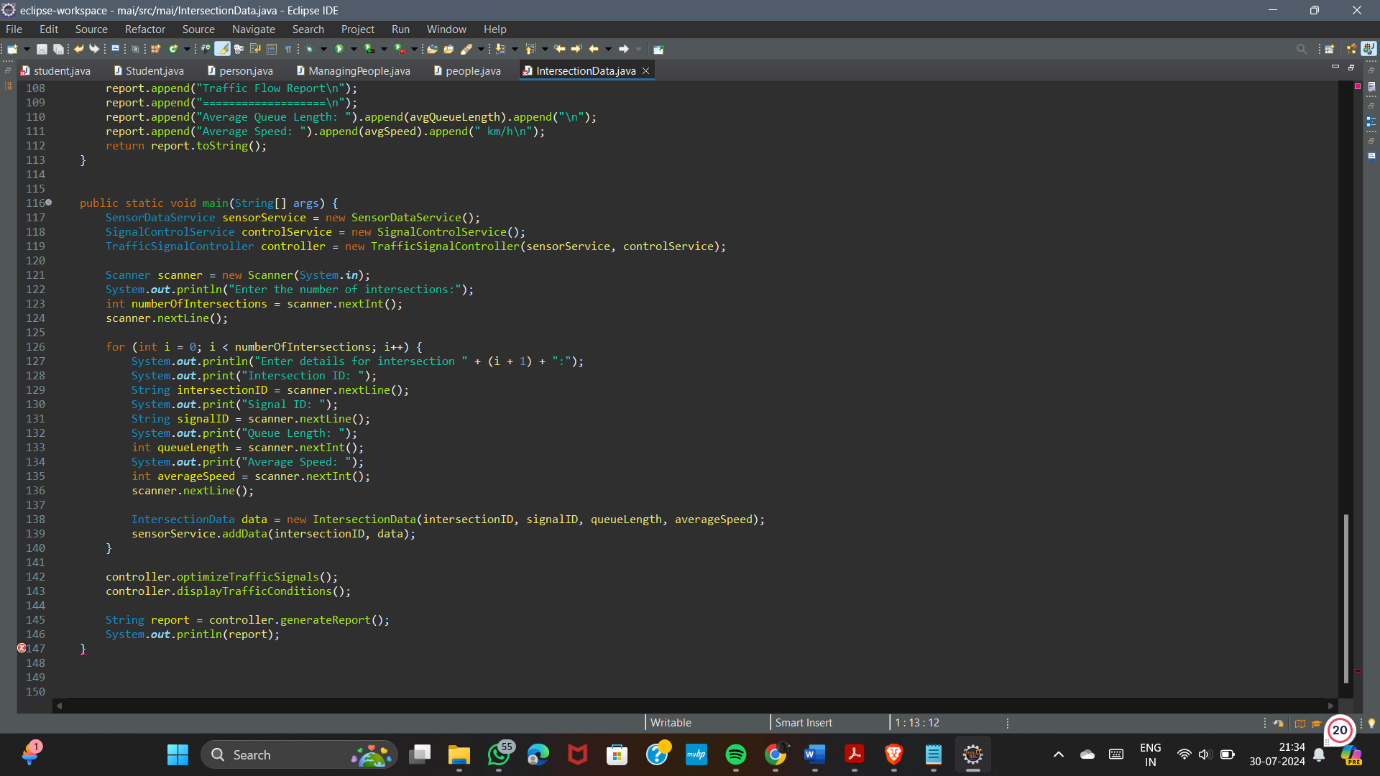
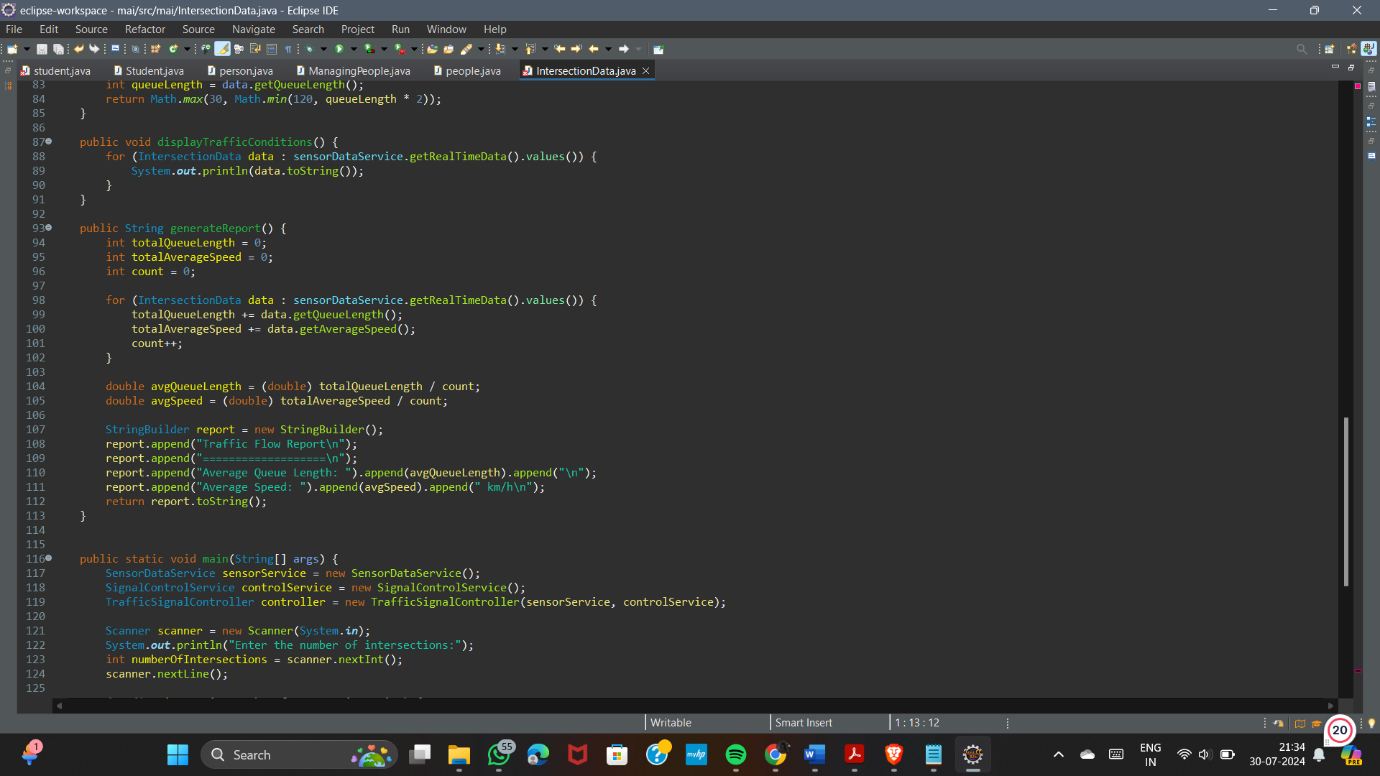
String report = controller.generateReport();

System.out.println(report);

}

}





Output:

