CSA 0959: Programming in Java for Scientific Applications

"Smart Traffic Signal Optimization"

BY: PRIYAL T

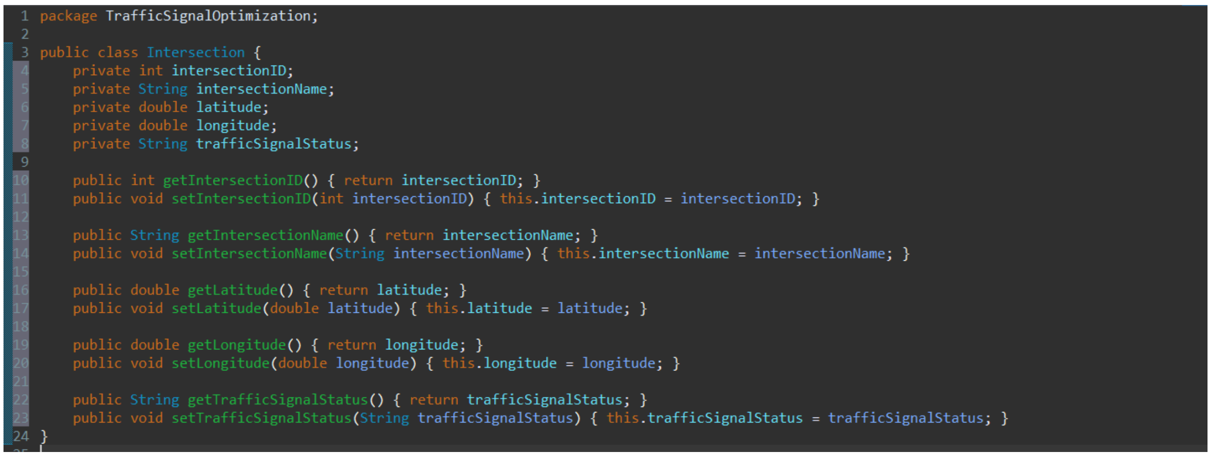
192324132

1. Define Data Models

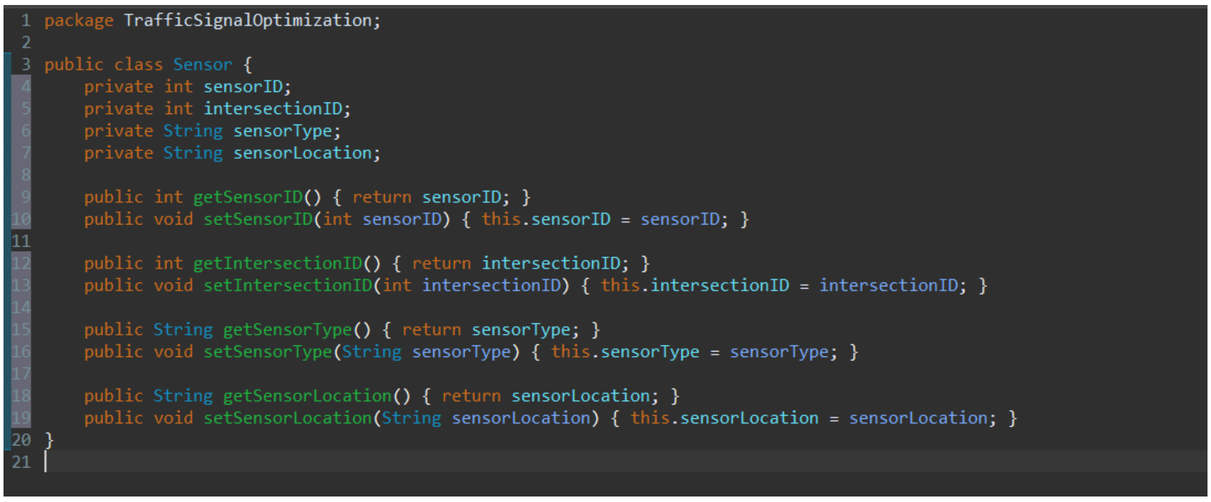
You will create Java classes that correspond to the database tables. These classes will be used to interact with the database and manage traffic data.

Data Structure Design:

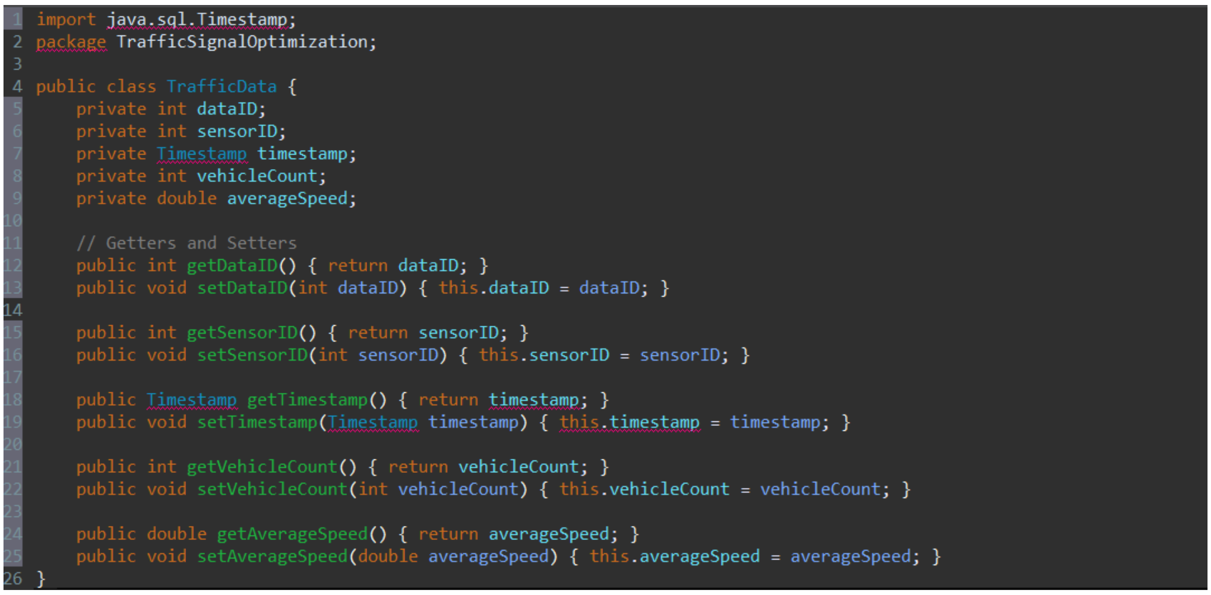
- Intersections Class
- Sensors Class
- Traffic Data Class
- Traffic Signal Timing Table



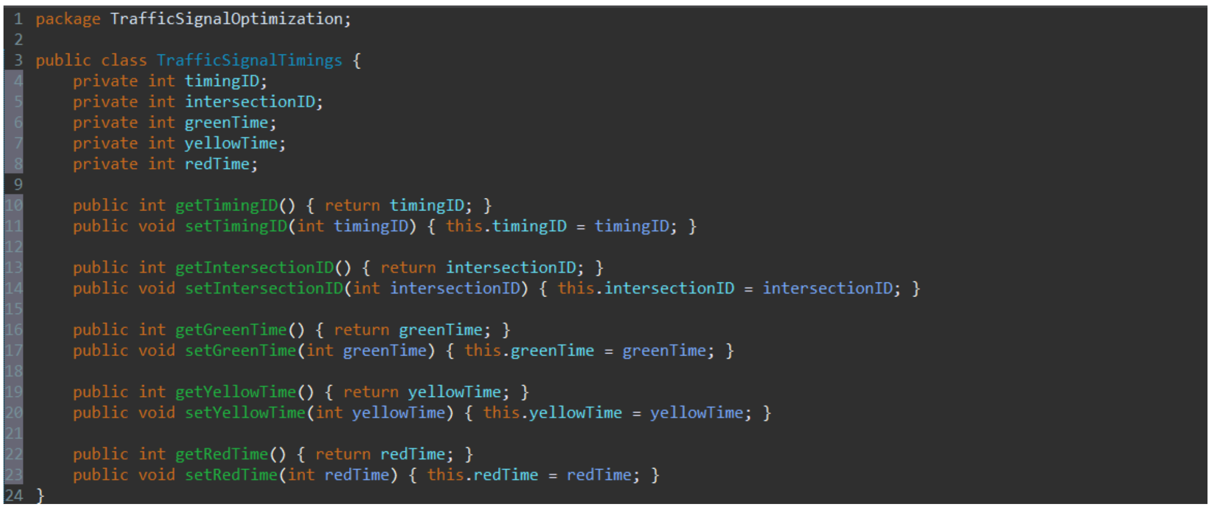
```java
package TrafficSignalOptimization;

public class Intersection {
    private int intersectionID;
    private String intersectionName;
    private double latitude;
    private double longitude;
    private String trafficSignalStatus;

    public int getIntersectionID() { return intersectionID; }
    public void setIntersectionID(int intersectionID) { this.intersectionID = intersectionID; }

    public String getIntersectionName() { return intersectionName; }
    public void setIntersectionName(String intersectionName) { this.intersectionName = intersectionName; }

    public double getLatitude() { return latitude; }
    public void setLatitude(double latitude) { this.latitude = latitude; }

    public double getLongitude() { return longitude; }
    public void setLongitude(double longitude) { this.longitude = longitude; }

    public String getTrafficSignalStatus() { return trafficSignalStatus; }
    public void setTrafficSignalStatus(String trafficSignalStatus) { this.trafficSignalStatus = trafficSignalStatus; }
}
```



```java
package TrafficSignalOptimization;

public class Sensor {
    private int sensorID;
    private int intersectionID;
    private String sensorType;
    private String sensorLocation;

    public int getSensorID() { return sensorID; }
    public void setSensorID(int sensorID) { this.sensorID = sensorID; }

    public int getIntersectionID() { return intersectionID; }
    public void setIntersectionID(int intersectionID) { this.intersectionID = intersectionID; }

    public String getSensorType() { return sensorType; }
    public void setSensorType(String sensorType) { this.sensorType = sensorType; }

    public String getSensorLocation() { return sensorLocation; }
    public void setSensorLocation(String sensorLocation) { this.sensorLocation = sensorLocation; }
}
```



```java
import java.sql.Timestamp;
package TrafficSignalOptimization;

public class TrafficData {
    private int dataID;
    private int sensorID;
    private Timestamp timestamp;
    private int vehicleCount;
    private double averageSpeed;

    // Getters and Setters
    public int getDataID() { return dataID; }
    public void setDataID(int dataID) { this.dataID = dataID; }

    public int getSensorID() { return sensorID; }
    public void setSensorID(int sensorID) { this.sensorID = sensorID; }

    public Timestamp getTimestamp() { return timestamp; }
    public void setTimestamp(Timestamp timestamp) { this.timestamp = timestamp; }

    public int getVehicleCount() { return vehicleCount; }
    public void setVehicleCount(int vehicleCount) { this.vehicleCount = vehicleCount; }

    public double getAverageSpeed() { return averageSpeed; }
    public void setAverageSpeed(double averageSpeed) { this.averageSpeed = averageSpeed; }
}
```



```java
package TrafficSignalOptimization;

public class TrafficSignalTimings {
    private int timingID;
    private int intersectionID;
    private int greenTime;
    private int yellowTime;
    private int redTime;

    public int getTimingID() { return timingID; }
    public void setTimingID(int timingID) { this.timingID = timingID; }

    public int getIntersectionID() { return intersectionID; }
    public void setIntersectionID(int intersectionID) { this.intersectionID = intersectionID; }

    public int getGreenTime() { return greenTime; }
    public void setGreenTime(int greenTime) { this.greenTime = greenTime; }

    public int getYellowTime() { return yellowTime; }
    public void setYellowTime(int yellowTime) { this.yellowTime = yellowTime; }

    public int getRedTime() { return redTime; }
    public void setRedTime(int redTime) { this.redTime = redTime; }
}
```

2. Algorithm Design

Algorithm for Dynamic Traffic Signal Optimization:

1. **Input:**

- Real-time data (vehicle counts, speeds)
    - Historical traffic patterns
    - Time of day, peak hours

2. **Process:**

Analyse Traffic Data:

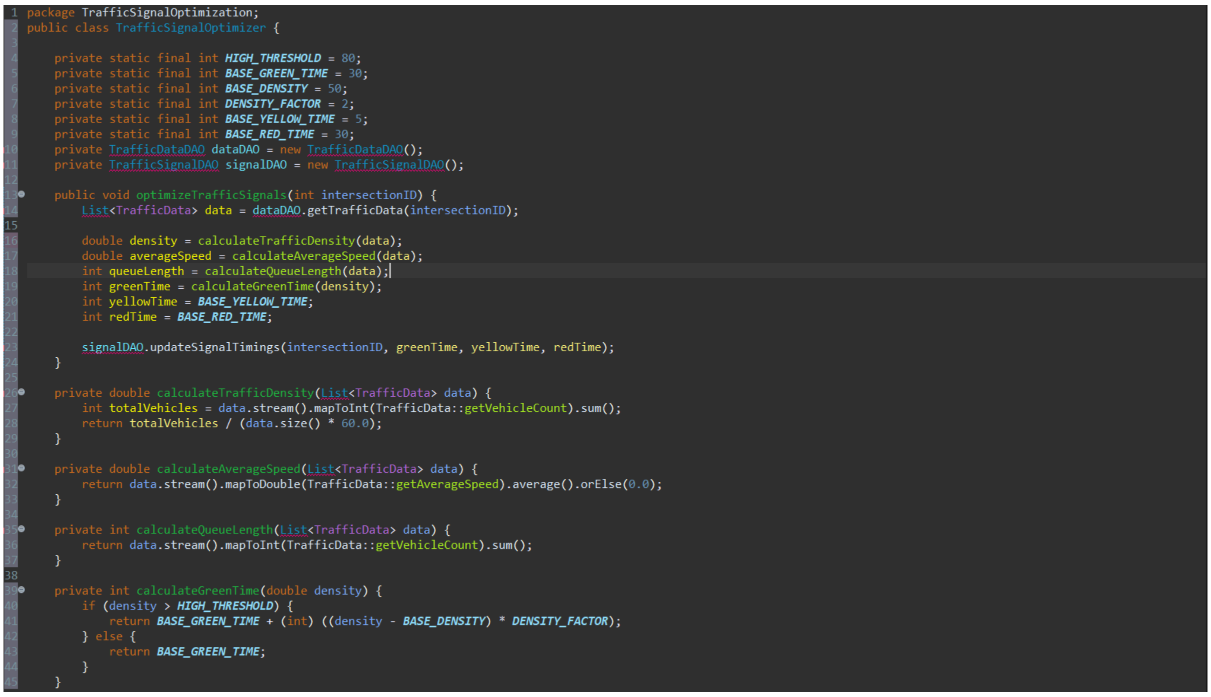
- Calculate traffic density (number of vehicles per unit length of road).
- Calculate average speed and vehicle queue length.

Adjust Signal Timings:

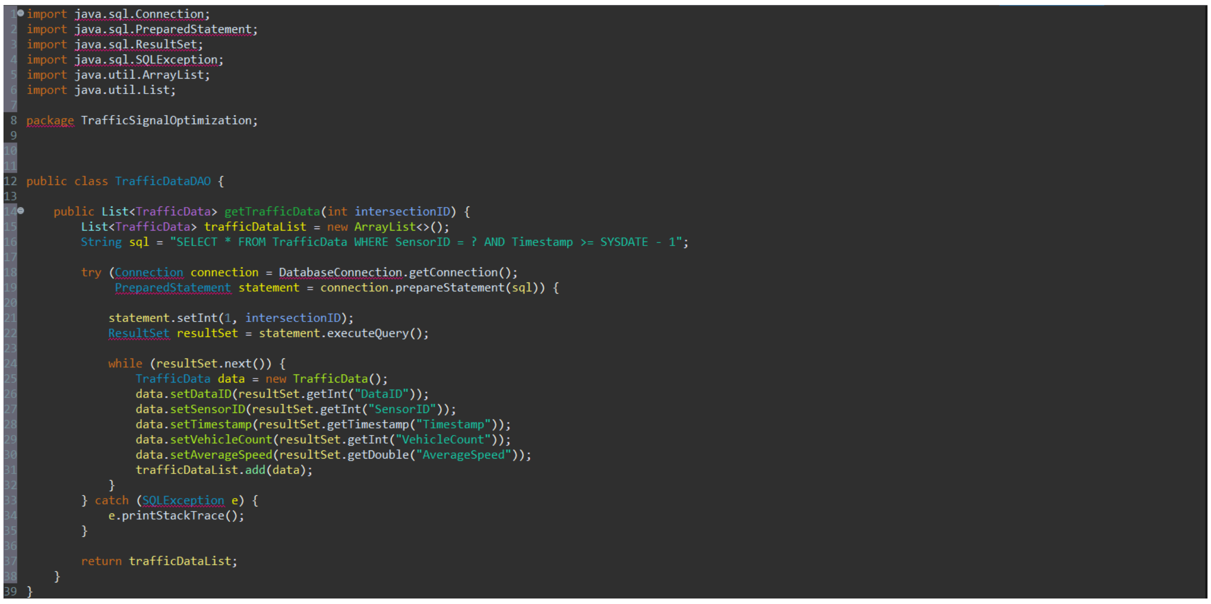
- **Green Time Calculation:**
    - Increase green time if traffic density is high.
    - Use historical data to predict peak periods and adjust timings accordingly.
- **Yellow Time:**
    - Maintain a fixed or slightly adjustable value based on traffic safety requirements.
- **Red Time:**
    - Ensure sufficient time for vehicles to clear the intersection.

3. **Output:**

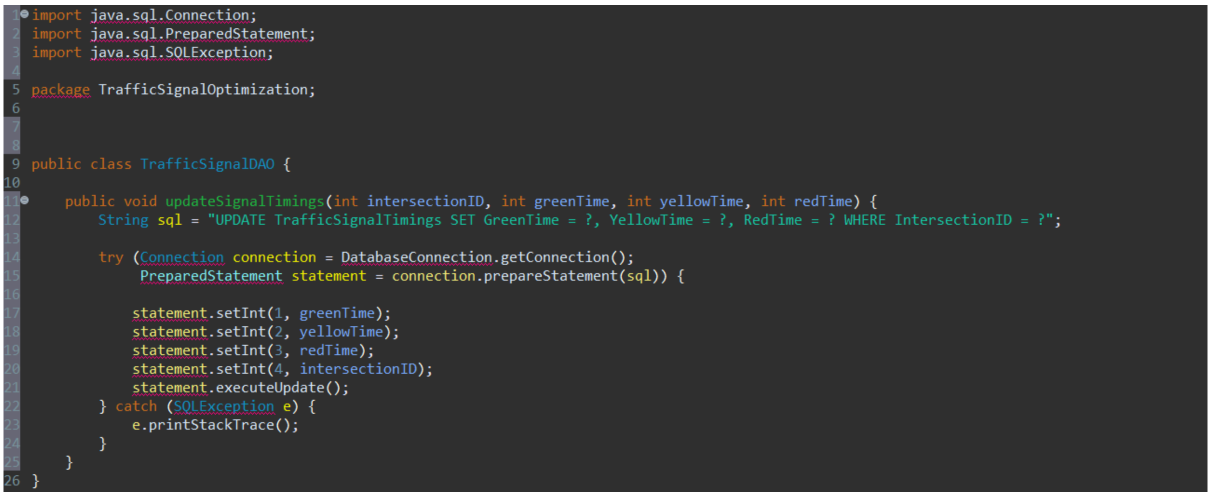
- Updated traffic signal timings (Green, Yellow, Red).



```java
package TrafficSignalOptimization;
public class TrafficSignalOptimizer {

    private static final int HIGH_THRESHOLD = 80;
    private static final int BASE_GREEN_TIME = 30;
    private static final int BASE_DENSITY = 50;
    private static final int DENSITY_FACTOR = 2;
    private static final int BASE_YELLOW_TIME = 5;
    private static final int BASE_RED_TIME = 30;
    private TrafficDataDAO dataDAO = new TrafficDataDAO();
    private TrafficSignalDAO signalDAO = new TrafficSignalDAO();

    public void optimizeTrafficSignals(int intersectionID) {
        List<TrafficData> data = dataDAO.getTrafficData(intersectionID);

        double density = calculateTrafficDensity(data);
        double averageSpeed = calculateAverageSpeed(data);
        int queueLength = calculateQueueLength(data);
        int greenTime = calculateGreenTime(density);
        int yellowTime = BASE_YELLOW_TIME;
        int redTime = BASE_RED_TIME;

        signalDAO.updateSignalTimings(intersectionID, greenTime, yellowTime, redTime);
    }

    private double calculateTrafficDensity(List<TrafficData> data) {
        int totalVehicles = data.stream().mapToInt(TrafficData::getVehicleCount).sum();
        return totalVehicles / (data.size() * 60.0);
    }

    private double calculateAverageSpeed(List<TrafficData> data) {
        return data.stream().mapToDouble(TrafficData::getAverageSpeed).average().orElse(0.0);
    }

    private int calculateQueueLength(List<TrafficData> data) {
        return data.stream().mapToInt(TrafficData::getVehicleCount).sum();
    }

    private int calculateGreenTime(double density) {
        if (density > HIGH_THRESHOLD) {
            return BASE_GREEN_TIME + (int) ((density - BASE_DENSITY) * DENSITY_FACTOR);
        } else {
            return BASE_GREEN_TIME;
        }
    }
```



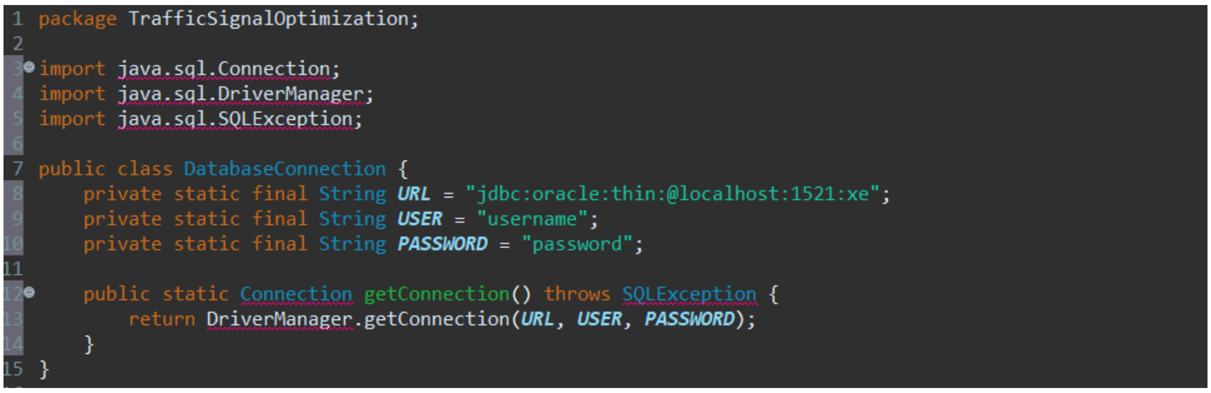
```java
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

package TrafficSignalOptimization;



public class TrafficDataDAO {

    public List<TrafficData> getTrafficData(int intersectionID) {
        List<TrafficData> trafficDataList = new ArrayList<>();
        String sql = "SELECT * FROM TrafficData WHERE SensorID = ? AND Timestamp >= SYSDATE - 1";

        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {

            statement.setInt(1, intersectionID);
            ResultSet resultSet = statement.executeQuery();

            while (resultSet.next()) {
                TrafficData data = new TrafficData();
                data.setDataID(resultSet.getInt("DataID"));
                data.setSensorID(resultSet.getInt("SensorID"));
                data.setTimestamp(resultSet.getTimestamp("Timestamp"));
                data.setVehicleCount(resultSet.getInt("VehicleCount"));
                data.setAverageSpeed(resultSet.getDouble("AverageSpeed"));
                trafficDataList.add(data);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return trafficDataList;
    }
}
```



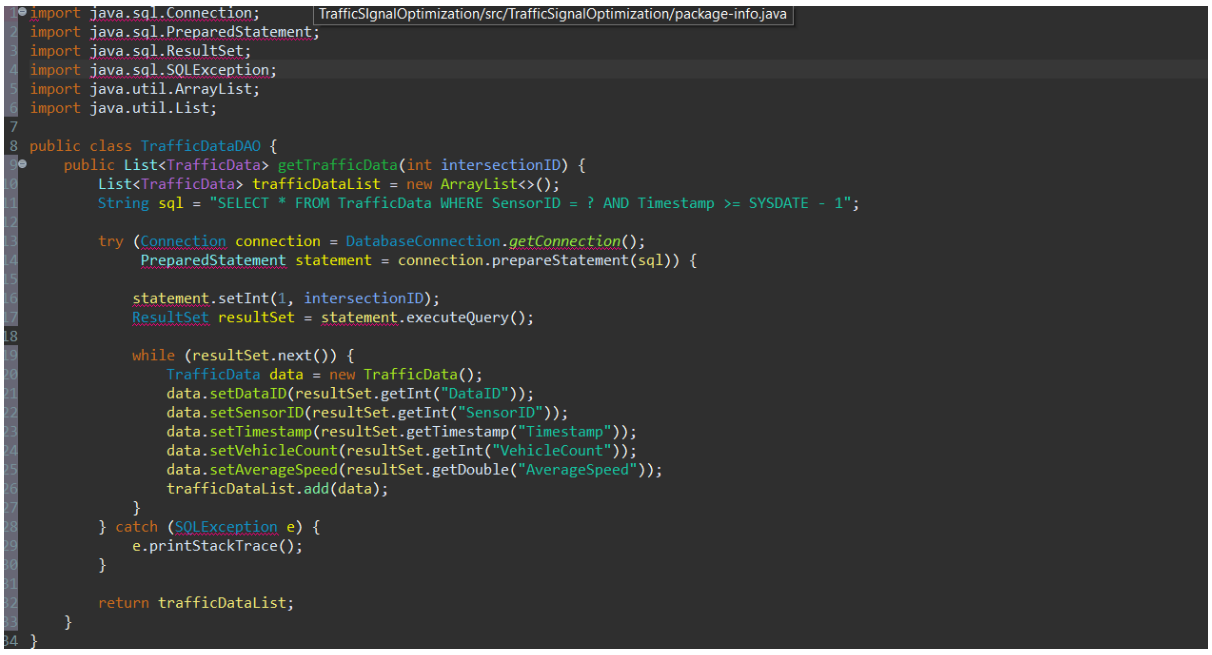
```java
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

package TrafficSignalOptimization;



public class TrafficSignalDAO {

    public void updateSignalTimings(int intersectionID, int greenTime, int yellowTime, int redTime) {
        String sql = "UPDATE TrafficSignalTimings SET GreenTime = ?, YellowTime = ?, RedTime = ? WHERE IntersectionID = ?";

        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {

            statement.setInt(1, greenTime);
            statement.setInt(2, yellowTime);
            statement.setInt(3, redTime);
            statement.setInt(4, intersectionID);
            statement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

3. Implementation:

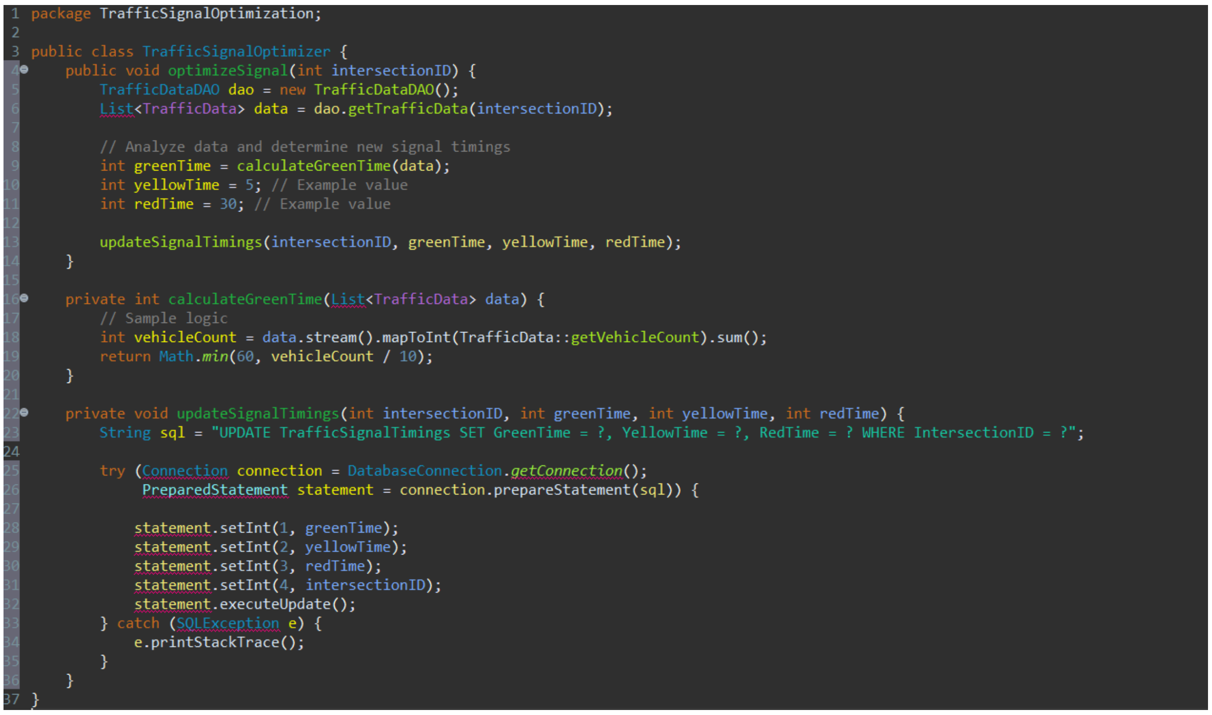
- Implement of Java application that integrates with traffic sensors and controls traffic signals at selected intersections.



```java
package TrafficSignalOptimization;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    private static final String URL = "jdbc:oracle:thin:@localhost:1521:xe";
    private static final String USER = "username";
    private static final String PASSWORD = "password";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```


TrafficSignalOptimization/src/TrafficSignalOptimization/package-info.java

```java
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class TrafficDataDAO {
    public List<TrafficData> getTrafficData(int intersectionID) {
        List<TrafficData> trafficDataList = new ArrayList<>();
        String sql = "SELECT * FROM TrafficData WHERE SensorID = ? AND Timestamp >= SYSDATE - 1";

        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {

            statement.setInt(1, intersectionID);
            ResultSet resultSet = statement.executeQuery();

            while (resultSet.next()) {
                TrafficData data = new TrafficData();
                data.setDataID(resultSet.getInt("DataID"));
                data.setSensorID(resultSet.getInt("SensorID"));
                data.setTimestamp(resultSet.getTimestamp("Timestamp"));
                data.setVehicleCount(resultSet.getInt("VehicleCount"));
                data.setAverageSpeed(resultSet.getDouble("AverageSpeed"));
                trafficDataList.add(data);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return trafficDataList;
    }
}
```



```java
package TrafficSignalOptimization;

public class TrafficSignalOptimizer {
    public void optimizeSignal(int intersectionID) {
        TrafficDataDAO dao = new TrafficDataDAO();
        List<TrafficData> data = dao.getTrafficData(intersectionID);

        // Analyze data and determine new signal timings
        int greenTime = calculateGreenTime(data);
        int yellowTime = 5; // Example value
        int redTime = 30; // Example value

        updateSignalTimings(intersectionID, greenTime, yellowTime, redTime);
    }

    private int calculateGreenTime(List<TrafficData> data) {
        // Sample logic
        int vehicleCount = data.stream().mapToInt(TrafficData::getVehicleCount).sum();
        return Math.min(60, vehicleCount / 10);
    }

    private void updateSignalTimings(int intersectionID, int greenTime, int yellowTime, int redTime) {
        String sql = "UPDATE TrafficSignalTimings SET GreenTime = ?, YellowTime = ?, RedTime = ? WHERE IntersectionID = ?";

        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {

            statement.setInt(1, greenTime);
            statement.setInt(2, yellowTime);
            statement.setInt(3, redTime);
            statement.setInt(4, intersectionID);
            statement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

4. Visualization and Reporting

Visualization:

- Use a web framework (e.g., JavaFX for desktop or Spring Boot with Thymeleaf for web) to create a dashboard.
- Visualize traffic flow, signal timings, and congestion using charts and maps.

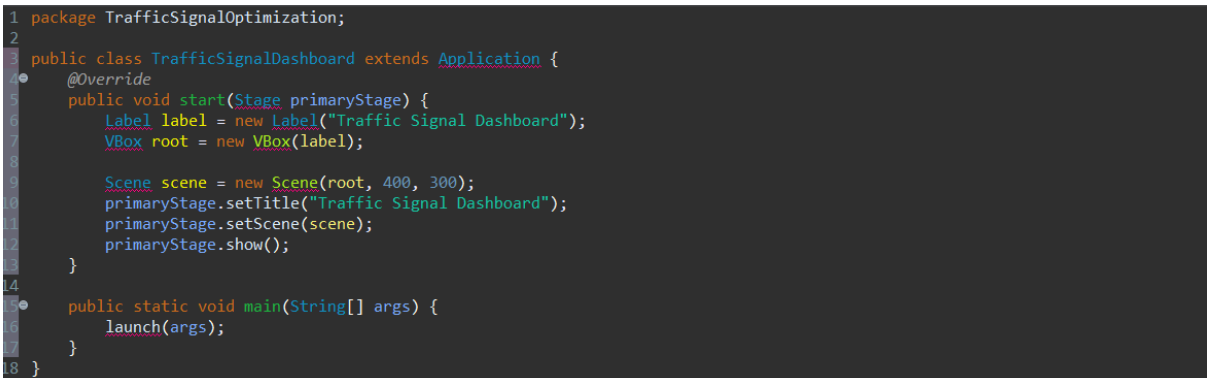
Reporting:

- Generate PDF or Excel reports using libraries like Apache POI or iText.
- Include metrics such as average wait times, traffic density, and congestion reduction.

5. User Interaction

User Interface:

- **For Traffic Managers:**
    - Implement a UI for real-time monitoring and manual adjustment of signal timings.
    - Provide controls to override automatic adjustments if needed.
- **For City Officials:**
    - Develop a dashboard displaying performance metrics and historical data.
    - Include visualizations of traffic flow, congestion, and optimization results.



```java
package TrafficSignalOptimization;

public class TrafficSignalDashboard extends Application {
    @Override
    public void start(Stage primaryStage) {
        Label label = new Label("Traffic Signal Dashboard");
        VBox root = new VBox(label);

        Scene scene = new Scene(root, 400, 300);
        primaryStage.setTitle("Traffic Signal Dashboard");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Documentation

- **Design Decisions:**
    - Use of historical and real-time data for dynamic signal optimization.
    - Assumptions about sensor accuracy and data reliability.
- **Data Structures:**
    - Explanation of table schemas and how they relate to real-time data collection and signal management.
- **Potential Improvements:**
    - Integration with machine learning for advanced prediction models.
    - Expansion of the system to include more intersections and sensors.

Testing

- **Unit Tests:**

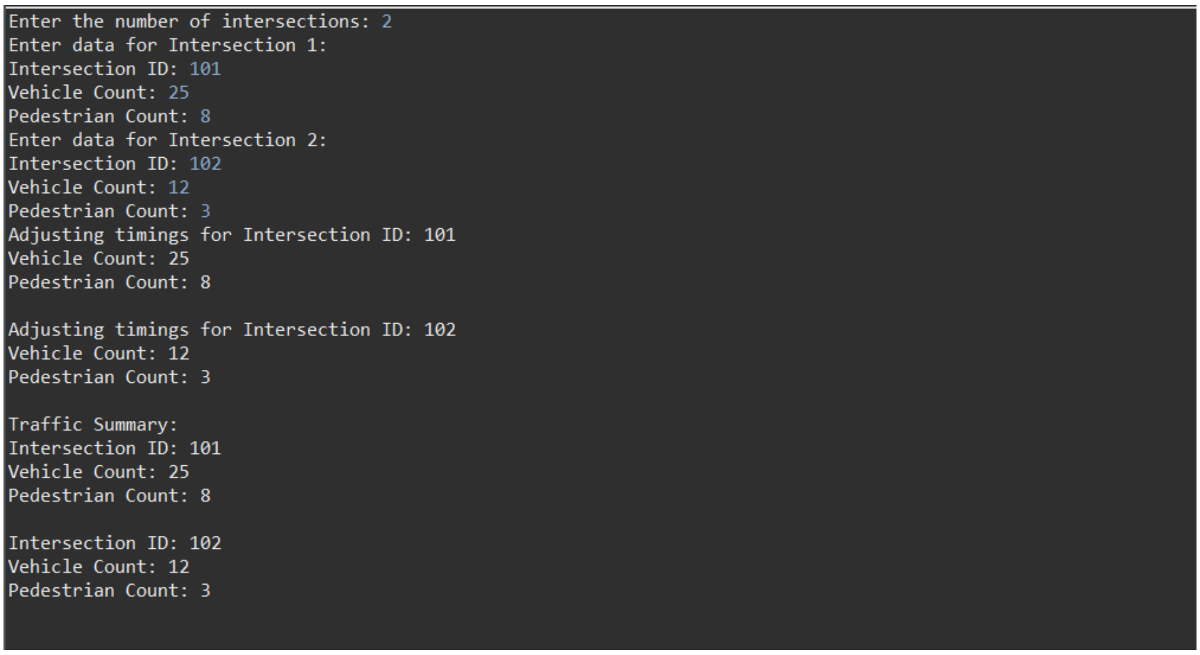
    - Test individual components (data retrieval, signal optimization) using JUnit.

- **Integration Tests:**
    - Verify that the system integrates well with the database and traffic sensors.

- **Performance Tests:**
    - Test the system under different traffic scenarios to ensure it can handle high traffic volumes efficiently.

By following this structured approach, we are able to build a robust smart traffic signal optimization system that improves traffic flow and reduces congestion.

Output:



```
Enter the number of intersections: 2
Enter data for Intersection 1:
Intersection ID: 101
Vehicle Count: 25
Pedestrian Count: 8
Enter data for Intersection 2:
Intersection ID: 102
Vehicle Count: 12
Pedestrian Count: 3
Adjusting timings for Intersection ID: 101
Vehicle Count: 25
Pedestrian Count: 8

Adjusting timings for Intersection ID: 102
Vehicle Count: 12
Pedestrian Count: 3

Traffic Summary:
Intersection ID: 101
Vehicle Count: 25
Pedestrian Count: 8

Intersection ID: 102
Vehicle Count: 12
Pedestrian Count: 3
```