**General Assignment Policies for Java Programming**

1. **Submission Deadlines:**
   * Assignments must be submitted by the specified deadline. Late submissions will incur a penalty of 10% per day, up to a maximum of 2 days. Submissions beyond 2 days will not be accepted unless prior arrangements have been made.
2. **Plagiarism:**
   * All work must be original and completed individually. Collaboration on concepts is allowed, but each student must submit their own implementation. Plagiarism will result in a score of zero for the assignment and possible disciplinary action.
3. **Code Quality:**
   * Code should be clean, well-documented, and follow proper naming conventions and coding standards. Include meaningful comments explaining the logic and purpose of each section of your code.
4. **Testing:**
   * Provide comprehensive test cases demonstrating the correct functionality of your program. Include test inputs and expected outputs to validate your code thoroughly.
5. **Documentation:**
   * Each assignment must include clear and comprehensive documentation. This should cover your approach, pseudocode, detailed explanation of the actual code, assumptions made, and any limitations. Additionally, include a user manual if necessary, detailing how to run and interact with the application.

**Grading Rubrics**

| **Criterion** | **Excellent (90-100%)** | **Good (75-89%)** | **Satisfactory (60-74%)** | **Needs Improvement (0-59%)** | **Weight** |
| --- | --- | --- | --- | --- | --- |
| Code Functionality | Code meets all requirements, works correctly, handles all edge cases and errors gracefully. | Code meets most requirements, works correctly for most cases, handles some edge cases/errors. | Code meets basic requirements, works for some cases, handles few edge cases/errors. | Code does not meet requirements, has significant errors, handles few or no edge cases/errors. | 30% |
| Code Quality | Code is clean, well-organized, follows naming conventions and coding standards. | Code is mostly clean and organized, minor deviations from naming conventions and standards. | Code is somewhat organized, several deviations from naming conventions and standards. | Code is disorganized, does not follow naming conventions or standards. | 20% |
| Documentation | Documentation is comprehensive, clear, covers approach, pseudocode, code explanation, assumptions. | Documentation is clear, covers most aspects of approach, pseudocode, code explanation, assumptions. | Documentation is basic, covers some aspects of approach, pseudocode, code explanation, assumptions. | Documentation is incomplete or unclear, missing significant aspects. | 20% |
| Testing | Includes comprehensive test cases demonstrating correct functionality and covering a wide range of inputs. | Includes several test cases demonstrating correct functionality, covers some range of inputs. | Includes basic test cases demonstrating correct functionality for limited inputs. | Includes few or no test cases, limited or no demonstration of correct functionality. | 20% |
| User Interface (if applicable) | Interface is intuitive, user-friendly, fully functional. | Interface is mostly intuitive, functional, minor usability issues. | Interface is functional but has significant usability issues. | Interface is difficult to use or non-functional. | 10% |

**Due Date : 31st July**

**Smart Traffic Signal Optimization**

**Scenario:** 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.

**Tasks:**

1. **Data Collection and Modeling:**
   * Define the data structure to collect real-time traffic data from sensors (e.g., vehicle counts, speeds) at various intersections across the city.
2. **Algorithm Design:**
   * Develop algorithms to analyze the collected data and optimize traffic signal timings dynamically based on current traffic conditions.
   * Consider factors such as traffic density, vehicle queues, peak hours, and pedestrian crossings in your algorithm.
3. **Implementation:**
   * Implement a Java application that integrates with traffic sensors and controls traffic signals at selected intersections.
   * Ensure the application can adjust signal timings in real-time to respond to changing traffic patterns and optimize flow.
4. **Visualization and Reporting:**
   * Develop visualizations to monitor traffic conditions and signal timings in real-time.
   * Generate reports on traffic flow improvements, average wait times, and overall congestion reduction achieved.
5. **User Interaction:**
   * Design a user interface for traffic managers to monitor and manually adjust signal timings if needed.
   * Provide a dashboard for city officials to view performance metrics and historical data.

**Deliverables:**

* **Data Flow Diagram:** Illustrate how real-time traffic data is collected, analyzed, and used to optimize traffic signal timings.
* **Pseudocode and Implementation:** Provide detailed pseudocode and Java code for the algorithms used to optimize traffic signals and manage intersections.
* **Documentation:** Explain the design decisions behind the algorithms, data structures used for efficient processing, assumptions made (e.g., sensor reliability), and potential improvements for further optimization.
* **User Interface:** Develop intuitive and informative interfaces for traffic managers and city officials to interact with the system, monitor traffic conditions, and manage signal timings.
* **Testing:** Include comprehensive test cases to validate the functionality and effectiveness of the traffic signal optimization system under various traffic scenarios and conditions.

This problem leverages modern technologies and data-driven approaches to address the challenge of optimizing traffic flow in urban settings, aligning with current trends in smart city initiatives and traffic management solutions. Adjustments can be made based on specific requirements or additional features desired for the assignment.