Use Case: Automotive Diagnostic System Simulator

Background

In the automotive industry, diagnostic systems play a crucial role in identifying and troubleshooting issues within a vehicle. These systems typically interface with various sensors and control units to retrieve real-time data and fault codes.

Objective

Design and implement a simplified **Automotive Diagnostic System Simulator** in C++. The simulator should mimic the interaction between a diagnostic tool and a vehicle's electronic control units (ECUs) to retrieve data and diagnose simulated faults.

The simulator should additionally include a comprehensive fault simulation across various ECUs (Engine, Transmission, Brake), enhanced user interaction via a detailed menu-driven interface, and simulated communication methods for notifying users about diagnostic results.

Requirements

1. **Core Components**:
    - Implement classes for **DiagnosticTool**, **Vehicle**, and multiple **ECUs** (e.g., EngineControlUnit, TransmissionControlUnit, BrakeControlUnit). Use inheritance and polymorphism where appropriate.
    - Each ECU should have a simulated data model containing parameters relevant to its domain (e.g., engine RPM, transmission gear, brake fluid level).
    - An abstract class **Faults** should be introduced to provide a unified interface for simulating faults across different types of ECUs."
    - Specialized classes derived from Faults, namely **EngineFaults**, **TransmissionFaults**, and **BrakeFaults**, should be implemented to simulate specific faults relevant to each ECU.
    - Each ECU class contains an instance of a Faults object, enabling the simulation of faults and the retrieval of fault codes for diagnostic purposes.
2. **Data Retrieval**:
    - Implement functionality in the **DiagnosticTool** class to connect to and communicate with the ECUs. This could involve sending simulated "diagnostic requests" and receiving data.
    - The system should simulate the detailed diagnostic capabilities, allowing for the identification and description of specific faults within each ECU, further enhancing the realism of the simulation.

3. **Fault Simulation and Detection**:

    - Allow ECUs to simulate faults (e.g., high engine temperature, low brake fluid). These faults can be predefined or triggered randomly.

    - The **DiagnosticTool** should be able to diagnose these faults by interpreting the data or fault codes retrieved from the ECUs.

    - The simulator should now be allowed for a more dynamic fault simulation process. ECUs should now simulate a variety of faults, which can be reset post-diagnosis. This process involves the generation of random faults, the retrieval of specific fault codes, and detailed fault descriptions.

4. **User Interface (Console-based)**:

    - Implement a simple console-based interface that allows the user to interact with the **DiagnosticTool**, initiate a diagnostic session, and display the results.

    - A comprehensive menu-driven user interface should be implemented in the main function. It enables user interaction for vehicle addition, ECU overview, fault simulation, fault code clearing, and sending fault details to users. This interface significantly enhances the user experience by providing a realistic simulation of diagnostic system operations.

5. User Communication Simulation

    - To mimic real-world diagnostic systems, simulated communication methods should be integrated. This includes the simulation of sending diagnostic details to users via messages, representing email or SMS communication. It demonstrates the system's capability to notify vehicle owners of diagnostic results, enhancing the simulation's applicability for real-world scenarios.

6. **Documentation**:

    - Provide comments and documentation within the code to explain the design decisions, the structure of the classes, and the flow of the program.

    - Extensive comments and documentation should be added to cover the newly introduced functionalities, including the abstract Faults class and its derivatives, the enhanced user interface, and the simulation of user communication. These additions aim to clarify the purpose, functionality, and interaction between the various components of the simulator.