Inheritance Problems

1. Single Inheritance: `Car` and `ElectricCar`

Problem Statement: Design a new class, **`ElectricCar`**, that extends a base class `Car`. The `Car` class should have protected members for `make`, `model`, and `year`. The `ElectricCar` class should add a private member, `batteryRange` (in miles). Implement a constructor for `ElectricCar` that initializes all members (both inherited and its own) and an overridden method `displayInfo()` that prints all the car's details, including its battery range. In `main`, create an `ElectricCar` object and call `displayInfo()` to show its complete specifications.

---

2. Multi-level Inheritance: `Appliance`, `SmartAppliance`, and `SmartTV`

Problem Statement: Create a multi-level inheritance structure with `Appliance` as the base class, `SmartAppliance` inheriting from `Appliance`, and `SmartTV` inheriting from `SmartAppliance`. The `Appliance` class should have protected members for `brand` and `price`. `SmartAppliance` should add a private member `networkProtocol` (e.g., "Wi-Fi", "Bluetooth"). The `SmartTV` class should add a private member `screenSize` (in inches). Each class should have a constructor that passes relevant data up the hierarchy. The `SmartTV` class must implement a `displayDetails()` method that prints all its inherited and unique attributes. In `main`, instantiate a `SmartTV` object and call its `displayDetails()` method.

---

3. Multiple Inheritance: `Artist`, `Programmer`, and `GameDeveloper`

Problem Statement: Implement a multiple inheritance model for a **`GameDeveloper`** class. This class should publicly inherit from a `Artist` class and a `Programmer` class. The `Artist` class should have a method `createArt()` that prints "Creating game art," and the `Programmer` class should have a method `writeCode()` that prints "Writing game code." The `GameDeveloper` class should have its own method, `designGame()`, that prints "Designing game." In `main`, instantiate a `GameDeveloper` object and call `createArt()`, `writeCode()`, and `designGame()` to demonstrate that it inherits behaviors from both parent classes.

---

4. Hierarchical Inheritance: `Vehicle`, `Car`, and `Motorcycle`

Problem Statement: Create a hierarchical inheritance structure with `Vehicle` as the base class. The `Vehicle` class should have protected members for `manufacturer` and `year`. Then, create two derived classes: **`Car`** and **`Motorcycle`**. The `Car` class should add a private member `numDoors`, and the `Motorcycle` class should add a private member `hasSidecar` (a boolean). Both `Car` and `Motorcycle` must have a constructor that initializes all their members and a `displayDetails()` method that prints all of the vehicle's information. In `main`, create objects for both a `Car` and a `Motorcycle` and call `displayDetails()` on each to demonstrate the hierarchy.

---

5. Hybrid Inheritance: `Student`, `TeachingAssistant`, `Instructor`, and `ResearchAssistant`

Problem Statement: Design a hybrid inheritance structure that involves the diamond problem, specifically for a **`ResearchAssistant`**. The `ResearchAssistant` class should inherit from two classes: `TeachingAssistant` and `Instructor`. Both `TeachingAssistant` and `Instructor` should virtually inherit from a common base class, **`Student`**. The `Student` class should have a private member `studentID`. The `TeachingAssistant` class should add a private member `department`, and the `Instructor` class should add a private member `officeHours`. The `ResearchAssistant` class should add a private member `researchTopic`. Ensure the `Student` class constructor is called only once for a `ResearchAssistant` object by using `virtual` inheritance. Implement a `displayInfo()` method in `ResearchAssistant` that prints all the inherited and unique attributes. In `main`, create a `ResearchAssistant` object and call `displayInfo()` to demonstrate that the `studentID` is correctly managed without ambiguity.