Scenario 1: **University Management System (Inheritance and Aggregation)**

Description:

A system to manage a university's employees and departments, where inheritance is used to model different types of employees and aggregation models the relationship between departments and employees.

Entities and Attributes:

1. **Base Class:** `Employee`
   - **Attributes**:
     - `string name`
     - `int employeeId`
     - `double salary`
   - **Methods**:
     - `void displayInfo()`: Displays the employee's details.
     - `void calculateAnnualSalary()`: Calculates the annual salary.
2. **Derived Class:** `Professor` **(Inheritance from** `Employee`**)**
   - **Attributes**:
     - `string specialization`
     - `int numPublications`
   - **Methods**:
     - `void teachCourse(string courseName)`: Simulates teaching a course.
     - `void addPublication(string title)`: Adds a publication to the professor's record.
3. **Derived Class:** `AdministrativeStaff` **(Inheritance from** `Employee`**)**
   - **Attributes**:
     - `string department`
     - `string role`
   - **Methods**:
     - `void assignTask(string task)`: Assigns a task to the staff member.
     - `void manageFiles()`: Simulates file management work.
4. **Class:** `Department` **(Aggregation)**
   - **Attributes**:
     - `string departmentName`
     - `vector<Employee*> employees`

- **Methods**:
  - `void addEmployee(Employee* employee)`: Adds an employee to the department.
  - `void displayDepartmentInfo()`: Displays the department's details, including its employees.

Example Relationships:

1. `Professor` and `AdministrativeStaff` inherit from `Employee`.
2. `Department` aggregates `Employee` objects, which can be either `Professor` or `AdministrativeStaff`.

Use Case:

- A department has multiple employees, such as professors and administrative staff, each with unique attributes and behaviors. By using inheritance, you can efficiently share common properties between employee types. Aggregation is used to represent that a department "has-a" list of employees.

Scenario 2: **E-commerce System (Inheritance and Aggregation)**

Description:

An e-commerce system to manage products, customers, and orders. Inheritance models different product categories, and aggregation models the relationship between customers and their orders.

Entities and Attributes:

1. **Base Class:** `Product`
   - **Attributes**:
     - `int productId`
     - `string name`
     - `double price`
   - **Methods**:
     - `void displayProductDetails()`: Displays the product's details.
     - `double calculateDiscount(double percentage)`: Calculates the discounted price.
2. **Derived Class:** `Electronics` **(Inheritance from** `Product`**)**
   - **Attributes**:
     - `string brand`
     - `int warrantyYears`
   - **Methods**:

        - `void showWarrantyInfo()`: Displays warranty information.
        - `void checkCompatibility(string device)`: Checks compatibility with other devices.

3. **Derived Class: `Clothing` (Inheritance from `Product`)**
    - **Attributes**:
        - `string size`
        - `string material`
    - **Methods**:
        - `void suggestCareInstructions()`: Provides care instructions for the clothing item.
        - `void matchWithAccessories()`: Suggests matching accessories.
4. **Class: `Customer`**
    - **Attributes**:
        - `int customerId`
        - `string name`
        - `vector<Order*> orders`
    - **Methods**:
        - `void placeOrder(Order* order)`: Adds a new order for the customer.
        - `void displayOrderHistory()`: Displays the customer's order history.
5. **Class: `Order` (Aggregation)**
    - **Attributes**:
        - `int orderId`
        - `vector<Product*> products`
        - `double totalPrice`
    - **Methods**:
        - `void addProduct(Product* product)`: Adds a product to the order.
        - `void calculateTotal()`: Calculates the total price of the order.

Example Relationships:

1. `Electronics` and `Clothing` inherit from `Product`.
2. `Order` aggregates `Product` objects, and `Customer` aggregates `Order` objects.

Use Case:

- Customers place orders for products from various categories like electronics and clothing. Inheritance simplifies managing product attributes and behaviors, while aggregation models the "has-a" relationship between customers and their orders.