Emergency Response Simulation - OOP Concepts Report

OOP Concepts Applied:

This project demonstrates the use of several fundamental Object-Oriented Programming (OOP) concepts:

1. Inheritance:

Inheritance is used to share common functionality between different emergency unit types. The base class `Emergency Unit` defines common properties (`Name`, `Speed`) and methods (`Can Handle()`, `RespondToIncident()`). Each subclass, such as `Police`, `Firefighter`, and `Ambulance`, inherits from this base class and provides specific implementations of these methods based on the type of incident it can handle.

2. Polymorphism:

Polymorphism is employed in this project through method overriding. Each subclass overrides the `Can Handle()` and `RespondToIncident()` methods from the `Emergency Unit` class. For instance, the `Police` class can only handle "Crime" incidents, while the `Firefighter` class can only handle "Fire" incidents. This allows the program to dynamically choose the correct behavior based on the type of incident.

3. Encapsulation:

Encapsulation is used to keep the internal details of each class hidden. The attributes `Name`, `Speed`, and `Difficulty` are set as private or protected, and are accessed or modified only via public methods (getters and setters). This ensures that the internal data of each class is protected and can only be modified in controlled ways.

4. Abstraction:

Abstraction is used with the `Emergency Unit` abstract class. This class defines an abstract interface (`Can Handle()`, `RespondToIncident()`), but leaves the implementation details to its subclasses. This allows us to treat all emergency units uniformly, even though their behavior can vary.

Class Diagram or Text-based Structure:

+------------------+ | Emergency Unit | +------------------+ | - Name: string | | - Speed: in | +------------------+ | + Can Handle(type) | | + RespondToIncident() | +------------------+ | +------+-------+ | | | | +-------+ +-------+ +--------+ +--------------+ | Police | | Firefighter| | Ambulance| | SearchAndRescue | +-------+ +----------+ +--------+ +--------------+ | +Can Handle() | +Can Handle()| +Can Handle()| +Can Handle()| | +RespondToIncident()| +RespondToIncident()| +RespondToIncident()| +RespondToIncident()| +------------------+ +------------------+ +-----------------+ +---------------------+

Lessons Learned or Challenges Faced:

1. Designing the Inheritance Structure:

Designing the inheritance structure was one of the main challenges. Initially, I struggled with determining which attributes and methods should belong to the base `EmergencyUnit` class and which ones should be specific to subclasses. Eventually, the concept of "responding to incidents" and "handling specific types of incidents" was abstracted in a way that allowed for minimal code duplication and greater extensibility.

2. Polymorphism and Dynamic Dispatch:

I had to ensure that polymorphism was used effectively so that each emergency unit responded differently to the same `RespondToIncident()` method. By using method overriding and dynamic dispatch, the correct behavior was executed based on the selected unit and the incident type.

3. Handling User Input:

Another challenge was handling user input and ensuring that the right unit was selected. I implemented both manual and automatic selection mechanisms, but dealing with edge cases (e.g., invalid input or no available unit for a specific incident) required extra handling, which helped me better understand how to handle unexpected inputs in a user interface.

4. Improving Code Flexibility:

While the current implementation works well for basic incidents, I learned that adding more complex scenarios (such as multiple units responding to a single incident or handling concurrent incidents) would require extending the current design. I plan to explore more advanced concepts, such as design patterns (e.g., Strategy or Factory), to improve scalability and maintainability.