Emergency Response System Simulation

Code Description This C# program simulates an emergency response system where various emergency units (e.g., Police, Firefighters, Ambulance, SWAT, Search and Rescue, Hazmat) respond to incidents. The program employs object-oriented programming (OOP) concepts such as abstraction, inheritance, and polymorphism to model the different emergency units and their specific behaviors.

Key Components

1. Emergency Unit (Abstract Class) Name: Represents the name of the emergency unit (e.g., Police Unit, Firefighter Unit, etc.).

Speed: Represents the speed of the unit (used to simulate the response time to incidents).

Methods:

CanHandle (string incident Type): An abstract method that checks if a unit can handle a specific type of incident.

RespondToIncident (Incident incident): An abstract method that defines how the unit responds to the incident.

2. Incident Class Type: The type of the incident (e.g., Crime, Fire, Medical, etc.).

Location: The location where the incident takes place.

Difficulty: A numeric value (1 to 10) that represents the difficulty level of handling the incident.

3. Specialized Units The following classes inherit from the Emergency Unit base class and implement their own versions of CanHandle () and RespondToIncident () methods:

Police

Firefighter

Ambulance

SWAT

SearchAndRescue

Hazmat

Each of these classes has a specific CanHandle () method that checks if the unit can handle a specific incident type and a RespondToIncident () method that defines how each unit responds to the incident.

4. Program (Main Simulation Logic) Unit Selection: The user can choose whether to manually select a unit or let the system automatically select one.

Incident Generation: Each turn, a random incident is generated with a type, location, and difficulty level.

Score Calculation: Points are awarded based on the unit's ability to handle the incident. If the unit can handle the incident, it responds, and the score is updated based on the difficulty and response time. If the unit cannot handle the incident, a penalty is applied.

The simulation runs for 5 turns, and at the end, the program displays the total score.

OOP Concepts Applied

1. Abstraction The Emergency Unit class defines the common properties (such as Name and Speed) and abstract methods (CanHandle () and RespondToIncident ()) that all emergency units share.

Specialized classes such as Police, Firefighter, and Ambulance inherit from the Emergency Unit class and provide their own specific implementations of the abstract methods. This abstraction helps to hide the complexity and present a simplified interface.

2. Inheritance Classes like Police, Firefighter, Ambulance, etc., inherit from the Emergency Unit class, which allows them to share common properties and methods. This promotes code reuse and makes the design more maintainable.

Each class can modify or extend the base behavior by overriding the abstract methods like CanHandle () and RespondToIncident () to implement unit-specific logic.

3. Polymorphism The RespondToIncident () method is polymorphic because each derived class overrides it to provide unit-specific behavior. For example, the Police class responds to crimes, while the Firefighter class handles fires.

This allows the same method (RespondToIncident ()) to be used across different types of Emergency Unit objects, and it behaves differently depending on the specific type of the unit.

simple class diagram

```
+-------------------+              +-----------------+

|   Emergency Unit   | <------------> |     Incident    |

|-------------------|              |-----------------|

| - Name: string    |              | - Type: string |

| - Speed: int      |              | - Location: string |

|-------------------|              | - Difficulty: int |

| + CanHandle ()     |              |-----------------|

| + RespondToIncident ()|

+-------------------+

       ^

       | (Inheritance)

+-------+--------+------------+-------+-----------+---------------------+

|   Police      | Firefighter | Ambulance |   SWAT    | SearchAndRescue | Hazmat     |

|--------------|--------------|-----------|-----------|-----------------|-----------|

| + CanHandle () | + CanHandle () | + CanHandle () | + CanHandle () | + CanHandle () | +
CanHandle () |

| + RespondToIncident () | + RespondToIncident () | + RespondToIncident () | +
RespondToIncident () | + RespondToIncident () | + RespondToIncident () |

+---------------------+-----------------------+------------------------+------------------+------------------
+---------------------+
```

Key Relationships:

- **Emergency Unit** is an abstract base class with common properties (`Name`, `Speed`) and methods (`CanHandle ()`, `RespondToIncident ()`), which are implemented by specialized units.
- **Incident** is a class containing information about the incident (Type, Location, Difficulty).
- Each specialized class (**Police**, **Firefighter**, **Ambulance**, **SWAT**, **SearchAndRescue**, **Hazmat**) inherits from **Emergency Unit** and provides specific implementations of the `CanHandle ()` and `RespondToIncident ()` methods

Specialized units (e.g., Police, Firefighter) inherit from Emergency Unit and implement the abstract methods.

Each specialized unit class provides specific functionality for handling and responding to different incident types.

Lessons Learned

1. Abstraction Helps Simplify Code by using an abstract base class (Emergency Unit), the code is easier to maintain and extend. New emergency units can be added with minimal changes to existing code.
2. Importance of Inheritance for Code Reusability Inheritance allowed for shared logic (such as having Name and Speed properties) to be reused in specialized classes, reducing code duplication and enhancing readability.
3. Polymorphism for Flexible Method Implementation Polymorphism allowed each emergency unit to implement its own version of the RespondToIncident () method. This enables easy expansion of the system with new units that may require different handling of incidents.

Challenges Faced

1. Handling User Input A challenge was managing user input for manual selection of emergency units. Ensuring that users could select a valid unit while providing meaningful feedback for incorrect inputs required implementing input validation.
2. Randomizing Incidents The simulation involves generating random incidents with types and difficulty levels. Handling this random generation required ensuring that all units could handle their designated incident types and giving the user an intuitive way to interact with the system.

3. Score Calculation Logic Designing the scoring system that accounts for response time, unit speed, and incident difficulty was tricky. Fine-tuning the formula to provide balanced points and penalties required careful consideration to make the game enjoyable.

Conclusion This program demonstrates core OOP principles and how they can be applied to model real-world scenarios. It introduces concepts such as abstraction, inheritance, and polymorphism while providing a simple yet engaging simulation. The key takeaway is how OOP promotes clean, maintainable code and facilitates system expansion with minimal effort.