C# Emergency Response Simulation - Code Explanation

1. Overview

This is a summary of a C# console application that implements the response of emergency units (Police, Firefighters, Ambulance) to various incidents (Crime, Fire, Medical) in a city. It simulates for a specified number of rounds and provides the user with a choice of choosing how incidents happen in a round, tracks the deployment of the appropriate units, and maintains a score of successful responses.

2. Core Classes

The simulation revolves around a series of core classes:

Incident Class:

One emergency incident.

Properties:

Type (string): Incident type (e.g., "Fire", "Crime", "Medical").

Location (string): Location where the incident occurred (e.g., "Downtown", "City Park").

Has a ToString() method for easy display of incident data.

EmergencyUnit (Abstract Class):

The base class template for all types of emergency responders.

Properties:

Name (string): The identifier of the unit (e.g., "Police Unit").

Speed (int): How fast the unit moves (presently for informational purposes, but could be used for timing extensions).

Abstract Methods (must be overridden by subclasses):

CanHandle(string incidentType): Returns true if the unit can handle a specific type of incident, false otherwise.

RespondToIncident(Incident incident): Declares the specific actions the unit takes when responding to an incident it can handle.

Concrete EmergencyUnit Subclasses:

Police: Inherits from EmergencyUnit.

CanHandle: Returns true only for "Crime" incidents (case-insensitive).

RespondToIncident: Prints a message that the police unit is securing the area.

Firefighter: Extends EmergencyUnit.

CanHandle: Returns true only for "Fire" incidents (case-insensitive).

RespondToIncident: Prints a message that the fire engine is extinguishing the fire.

Ambulance: Extends EmergencyUnit.

CanHandle: Returns true only for "Medical" incidents (case-insensitive).

RespondToIncident: Prints a message that the ambulance is administering medical aid.

3. Simulation Logic (Program Class)

Initialization:

Initializes the predefined lists of possible incidentTypes and locations for random selection.

Instantiates each concrete emergency unit (Police, Firefighter, Ambulance) and adds them to the availableUnits list.

Initializes the score to 0 and sets the totalRounds (currently 5).

Uses one Random instance (random) for random number generation.

Main Loop:

The simulation is run for the specified totalRounds.

User Choice: For each round, the user is prompted to choose:

Option 1: Create 5 random incidents.

Option 2: Enter information manually for 1 custom incident.

Invalid input reverts to Option 1.

Incident Generation:

GenerateRandomIncident(): Selects a random type and location from the pre-defined lists.

GetCustomIncident(): Prompts the user for the incident type (validating against "Fire", "Crime", "Medical") and location.

Incident Processing:

A list (incidentsThisRound) holds the incident(s) for the current round (either 5 random or 1 custom).

The program iterates through each currentIncident in the list.

FindHandlerUnit(Incident incident): This helper method iterates over the availableUnits list. It invokes CanHandle() on each unit to find the first one that can handle the currentIncident's type. It returns the proper unit or null if none.

Response & Scoring:

In case a respondingUnit is found:

The unit's RespondToIncident() method is invoked, and the response message is printed.

The score is increased by 10 points.

If there is no proper unit (e.g., wrong custom type input):

A message indicates the incident could not be processed.

The score decreases by 5 points.

There are brief pauses (Thread.Sleep) inserted to allow for readability, especially between incidents in one round.

End of Round: The score for the current round is displayed.

Final Score: After all rounds have been processed, the final score is displayed.

4. Key Features

Object-Oriented Design: It uses abstraction (EmergencyUnit) and inheritance (Police, Firefighter, Ambulance) for an explicit and extensible design.

Polymorphism: The RespondToIncident method varies according to the actual type of the EmergencyUnit object.

User Interaction: Allows the user to have influence on the simulation by choosing the incident generation method.

Randomness: Offers random incident generation for unpredictability.

Basic Scoring: Provides basic feedback about the success of the responses.

Modularity: Helper functions (GenerateRandomIncident, GetCustomIncident, FindHandlerUnit) improve code structure.