Emergency Response Simulation — Program Report

Program Description

This program implements a simple **Emergency Response Simulation** using fundamental Object-Oriented Programming (OOP) principles.
The system models three types of emergency units:

- **Police**
- **Firefighter**
- **Ambulance**

The user inputs an **incident type** and **location** during each round of the simulation. The program then selects the appropriate emergency unit to respond.
The player's **score** is updated based on the correctness of their actions.

The program emphasizes clean class structure, code reusability, and modular design — all essential for scalable software development.

Program Functionality

- **User Interaction**: The player enters the incident type and location.
- **Incident Handling**:
  - The system checks if the incident type is valid.
  - It finds the first available emergency unit capable of handling the incident.
  - If successful, points are awarded; otherwise, points are deducted.
- **Score Keeping**:
  - Correctly handled incidents: **+10 points**.
  - Invalid incidents or unhandled incidents: **-5 points**.
- **Program Termination**:
  - After five rounds, the simulation ends and the final score is displayed.

Object-Oriented Programming (OOP) Concepts Applied

Concept	How it Appears in the Code
Abstraction	`Emergency Unit` is an abstract class defining a general blueprint for all units. Specific units implement the response behavior.

Concept	How it Appears in the Code
Inheritance	`Police`, `Firefighter`, and `Ambulance` inherit from `Emergency Unit`, acquiring common properties like `Name` and `Speed`.
Polymorphism	The program uses an abstract reference (`Emergency Unit`) to call methods like `Can Handle()` and `RespondToIncident()`, allowing different behaviors dynamically at runtime.
Encapsulation	Emergency units manage their internal state through properties (`Name`, `Speed`) and hide the specific handling logic behind method interfaces.

□ Simple Class Diagram (Text-based)

```
lua
Copyedit
    +----------------------+
    |   Emergency Unit       | (abstract)
    +----------------------+
    | - Name: string        |
    | - Speed: int          |
    +----------------------+
    | + Can Handle(string) | (abstract)
    | + RespondToIncident (string, string) | (abstract)
    +----------------------+
              ▲
              |
    ------------------------------
    |             |               |
+----------+  +-------------+  +------------+
| Police | | Firefighter | | Ambulance |
+----------+  +-------------+  +------------+
| + Can Handle ()               |
| + RespondToIncident ()       |
+------------------------------+

+----------------------+
|      Program         |
+----------------------+
| + Main (string [] args) |
| + IsValidIncidentType(string) |
+----------------------+
```

Lessons Learned and Challenges Faced

Lesson/Challenge	Description
Modeling Real-world Behavior	Abstracting the behavior of different emergency units into a common parent class (`Emergency Unit`) made the code logical, clean, and extensible.
Input Validation Importance	Verifying user input prevented invalid operations and crashes, highlighting the necessity of defensive programming.

Lesson/Challenge	Description
Benefits of Polymorphism	Enabled handling different units in a uniform way, reducing complexity and avoiding the need for multiple type-checks (`if`/`else if`).
Extendibility of the System	The design allows easy future expansion, such as adding new unit types (e.g., "Rescue Team") without modifying the existing logic heavily.
Challenge - Handling Multiple Units	In a more complex version, multiple units might be able to respond, needing prioritization based on criteria like speed, distance, or severity.

Summary

This project successfully demonstrates key Object-Oriented Programming practices, combining abstraction, inheritance, polymorphism, and encapsulation to create a maintainable, extendable, and robust C# simulation.