1. OOP Concepts Applied

1.1 Inheritance

- `EmergencyUnit` is an **abstract base class** that defines common properties (`Name`, `Speed`) and methods (`CanHandle()`, `RespondToIncident()`).
- **Derived classes** (`Police`, `Firefighter`, `Ambulance`, etc.) **inherit** from `EmergencyUnit` and implement their own versions of the abstract methods.

1.2 Polymorphism

- Each derived class **overrides** `CanHandle()` and `RespondToIncident()` to provide **specialized behavior**.
- The program treats all emergency units **uniformly** through the base class, but their **actual behavior differs** at runtime.

1.3 Abstraction

- `EmergencyUnit` is **abstract**, meaning it cannot be instantiated directly.
- It **hides implementation details** and forces derived classes to define their own logic.

1.4 Encapsulation

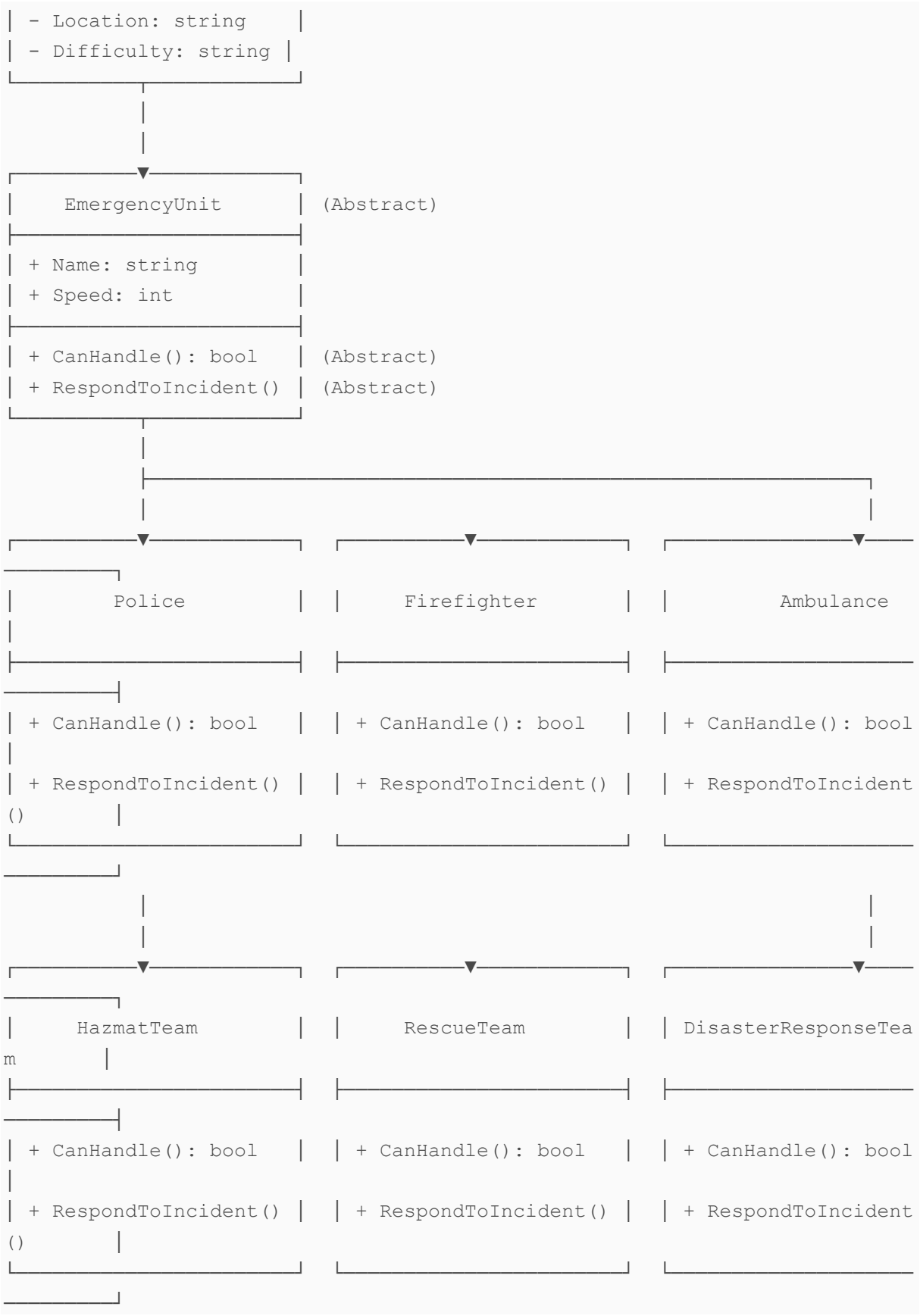
- Each class **encapsulates** its own data (e.g., `Incident` holds `Type`, `Location`, `Difficulty`).
- Properties are **protected** within their respective classes.

---

2. Class Diagram (Text-Based Structure)

Copy
Download

```
┌──────────────────────────┐
│       Incident           │
├──────────────────────────┤
│ - Type: string           │
```


- Location: string
- Difficulty: string
EmergencyUnit
(Abstract)
+ Name: string
+ Speed: int
+ CanHandle(): bool
(Abstract)
+ RespondToIncident()
(Abstract)
Police
Firefighter
Ambulance
+ CanHandle(): bool
+ RespondToIncident()
+ CanHandle(): bool
+ RespondToIncident()
+ CanHandle(): bool
+ RespondToIncident()
HazmatTeam
RescueTeam
DisasterResponseTeam
+ CanHandle(): bool
+ RespondToIncident()
+ CanHandle(): bool
+ RespondToIncident()
+ CanHandle(): bool
+ RespondToIncident()

3. Lessons Learned & Challenges Faced

3.1 Design Challenges

- **Balancing Abstraction vs. Specificity**:
  - Making `EmergencyUnit` too generic would lose important details.
  - Making it too specific would reduce flexibility for new unit types.
- **Handling Incorrect User Input**:
  - Manual mode required extra validation to prevent wrong unit assignments.

3.2 Implementation Insights

- **Polymorphism Simplified Logic**:
  - The same method (`RespondToIncident()`) works differently for each unit type.
- **Easy to Extend**:
  - Adding a new emergency unit (e.g., `CyberSecurityTeam`) would require minimal changes.

3.3 Potential Improvements

- **Factory Pattern**:
  - Could be used to **create units dynamically** instead of hardcoding them.
- **Better Scoring System**:
  - Reward faster response times and penalize wrong unit assignments more.
- **Unit Specializations**:
  - Some incidents could require **multiple units** (e.g., Fire + Ambulance).

Conclusion

This project effectively demonstrates **inheritance, polymorphism, abstraction, and encapsulation** in a practical emergency simulation. The design allows for **easy expansion** while keeping the code **clean and maintainable**