Object-Oriented Programming
Semester 2025-III
Class Workshop — Pokemon Battle System Implementation

Eng. Carlos Andrés Sierra, M.Sc.
Lecturer
Computer Engineering
School of Engineering
Universidad Distrital Francisco José de Caldas

Welcome to the class workshop of the *Object-Oriented Programming* course! In this session, you will implement a **Pokemon Battle System** that demonstrates the four fundamental pillars of object-oriented programming: *Inheritance*, *Polymorphism*, *Composition*, and *Aggregation*. This practical exercise uses the well-known Pokemon universe to make these concepts more accessible and engaging.

Workshop Scope and Objectives:

- **Inheritance:** Implement Pokemon type hierarchies where different Pokemon categories share common behaviors but have specialized characteristics.
- **Polymorphism:** Demonstrate how the same method calls (attack, defense) behave differently depending on the Pokemon type.
- **Composition:** Show "has-a" relationships where objects are essential components of other objects and cannot exist independently.
- **Aggregation:** Implement "uses" relationships where objects can exist independently but are temporarily grouped together.
- **Encapsulation:** Protect internal object state and provide controlled access through appropriate methods.

Carlos Andrés Sierra, Computer Engineer, M.Sc. in Computer Engineering, Lecturer at Universidad Nacional de Colombia.
Any comment or concern regarding this workshop can be sent to Carlos A. Sierra at: *casierrav@unal.edu.co*.

System Requirements and Implementation Guidelines:

1. **Player System (Aggregation & Encapsulation):**
   - Each player must have: name, collection of badges, and a pokebag (Pokemon collection).
   - *Encapsulation Hint:* Based on the badges collection, determine if a player qualifies as a "Pokemon Master". This logic should be internal to the Player class.
   - *Aggregation Hint:* The pokebag contains Pokemon, but Pokemon can exist independently of the player.
2. **Badge System and Gym Battles:**
   - Players earn badges by defeating gym leaders in Pokemon battles.
   - Each victory against a gym leader awards the player a badge specific to that gym.
   - *Design Hint:* Consider how badges relate to gyms and how this affects the player's status.
3. **Wild Pokemon Encounters (Aggregation):**
   - Wild Pokemon can be found in various locations outside of gyms.
   - When a player successfully catches a wild Pokemon, it should be added to their pokebag.
   - *Aggregation Hint:* Pokemon exist independently in the wild before being caught.
4. **Pokemon Type System (Inheritance & Polymorphism):**
   - Pokemon is a base character type with common attributes and behaviors.
   - Create Pokemon categories (Fire, Water, Electric, Grass, etc.) that inherit from the base Pokemon class.
   - Each category should have similar capabilities but specialized implementations.
   - *Polymorphism Hint:* The same attack or defense method should behave differently based on the Pokemon's type.
   - Individual Pokemon are instances of these categories.
5. **Gym System (Composition):**
   - Each gym has a specific type (Fire, Water, Electric, etc.) and a gym leader.
   - Each gym leader has a main Pokemon that represents the gym's specialty.
   - *Composition Hint:* The gym is composed of the leader and the main Pokemon. Without these components, the gym cannot function.
   - Players can choose any Pokemon from their pokebag to battle against the gym leader's main Pokemon.

6. **Enhanced Combat System (Polymorphism):**
   - Every Pokemon must have at least one attack method and one defense method.
   - Some Pokemon types may have multiple attack options available during battle.
   - *Polymorphism Hint:* These methods should be implemented differently for each Pokemon type, demonstrating how the same interface can have varied behaviors.
   - Players should be able to choose which attack to use during battle scenarios.
   - Consider type advantages/disadvantages in your implementation.
7. **Pokemon Recovery System (Composition & Polymorphism):**
   - Injured Pokemon can be healed at Pokemon clinics scattered throughout the world.
   - Small clinics are composed of only a nurse, while big clinics are composed of both a medical doctor and a nurse.
   - Big clinics can recover Pokemon faster and heal more severe injuries compared to small clinics.
   - *Composition Hint:* Clinics cannot function without their essential staff components.
   - *Polymorphism Hint:* The healing process should behave differently based on clinic type and Pokemon condition.
8. **Advanced Badge System (Encapsulation):**
   - Each badge must have a specific type (Fire, Water, Electric, etc.) and a level ranging from 1 to 5.
   - Badge level represents the difficulty and prestige of the victory achieved.
   - To qualify as a Pokemon Master, a player must have at least one badge per gym type AND each badge must be level 3 or higher.
   - *Encapsulation Hint:* This complex master status determination logic should remain internal to the Player class.

Enhanced Implementation Phases:

Phase 1: Design and Plan

- Create UML class diagrams showing inheritance hierarchies.
- Identify composition vs. aggregation relationships.
- Plan which methods should be abstract/overridden.
- Design clinic hierarchy and badge leveling system.

Phase 2: Base Classes Implementation

- Implement the base Pokemon class with common attributes and methods.
- Create the Player class with enhanced encapsulation for master status.
- Implement basic Gym structure.
- Create base Clinic class and Badge class with type and level attributes.

Phase 3: Inheritance and Polymorphism

- Create Pokemon type subclasses (FirePokemon, WaterPokemon, etc.).
- Override attack and defense methods for each type.
- Implement multiple attack options for selected Pokemon types.
- Demonstrate polymorphic behavior in battles and healing processes.
- Create SmallClinic and BigClinic subclasses with different healing capabilities.

Phase 4: Composition and Aggregation

- Implement gym composition with leaders and main Pokemon.
- Create pokebag aggregation system with battle selection capability.
- Implement wild Pokemon encounter and capture mechanics.
- Compose clinics with appropriate staff (nurse vs. medical doctor + nurse).
- Implement Pokemon injury and recovery system.

Phase 5: Integration and Testing

- Create scenarios that demonstrate all four OOP concepts working together.
- Test battle system with different Pokemon types and multiple attack options.
- Verify enhanced encapsulation by testing complex master status determination.
- Test clinic healing system with different clinic types and injury levels.
- Validate badge leveling system and master qualification logic.

Additional Implementation Hints:

- **Clinic System:** Consider how healing time and effectiveness vary between clinic types. Small clinics might only heal minor injuries, while big clinics can handle severe damage.
- **Multiple Attacks:** Use method overloading or strategy pattern for different attack combinations. Some Pokemon might have basic and special attacks.
- **Battle Selection:** Implement a user interface for choosing Pokemon from the poke-bag during gym battles. Consider Pokemon health status in selection.
- **Badge Complexity:** The master status now requires both breadth (one badge per type) and depth (level 3+ badges). This creates a more challenging achievement system.
- **Recovery Mechanics:** Design injury levels that match with clinic capabilities for realistic healing scenarios. Consider Pokemon type compatibility with healing methods.

Notes:

- Focus on demonstrating OOP concepts clearly rather than creating a complex game.
- Use meaningful class and method names that reflect the Pokemon domain.
- Include comments explaining how each OOP concept is being demonstrated.
- Test your polymorphic methods with different Pokemon types to verify correct behavior.
- Pay special attention to the enhanced encapsulation logic for master status determination.

This enhanced exercise will help you understand how object-oriented principles work together in a practical, engaging context. The Pokemon domain provides an excellent framework for demonstrating these fundamental programming concepts with added complexity that mirrors real-world software development challenges!