15 - Polymorphism II

March 13, 2023

COMP2404

Darryl Hill

Contents

Polymorphism and Design Patterns

Polymorphism is the mechanism behind many design patterns.

- At the heart of many design patterns are abstract interfaces.
- Polymorphism allows these interfaces to hide different implementations

We will look at an example of this with the ***Strategy Design Pattern***

- In our example we will use ***Strategy*** to implement ***Behaviour classes***
- Polymorphism will provide the common interface

Encapsulating Behaviour

Behaviour class

- a class that implements a behaviour or algorithm

Purpose

- allows the client to dynamically change which code executes
- behaviour object can be switched at runtime
- new behaviour can be added without breaking existing code

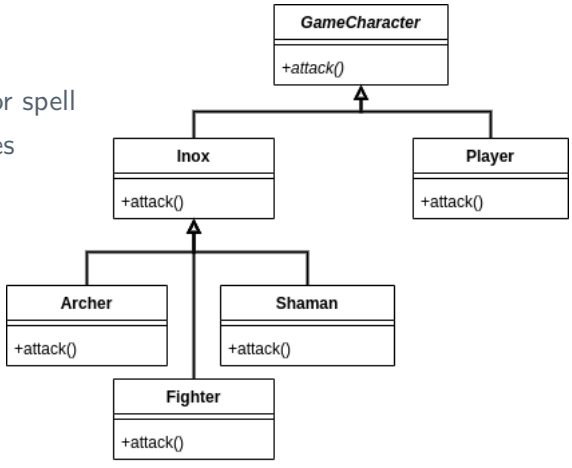
Why?

- Agility to make changes
  - client changes their mind
  - improvements / enhancements
  - etc

Encapsulating Behaviour with Entity Classes

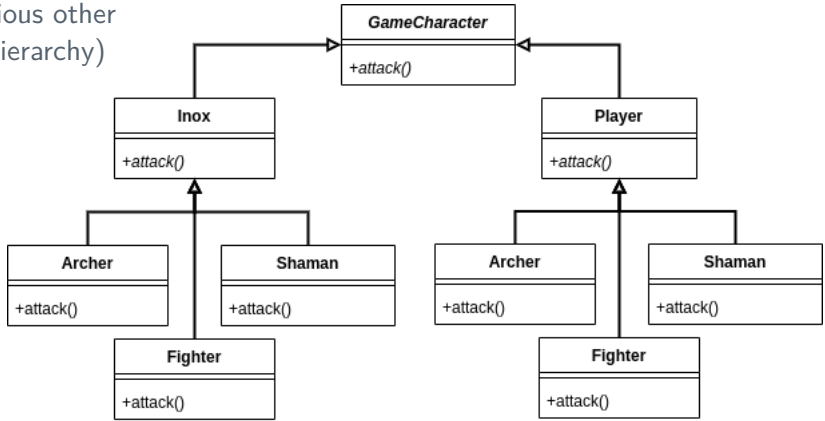
Example:

- Inox comes in three variants
- Player picks up a new weapon or spell
- They need similar attack abilities



Encapsulating Behaviour with Entity Classes

(There are various other options for a hierarchy)



Using Entity Classes

Traditional approach

- promotes data abstraction, encapsulation
- uses polymorphism

Problem

- how do we change the behaviour at runtime?
- e.g. player picks up a new weapon or spell?
  - deleting the old object and making a new object is expensive
  - allocating and deallocating is slow

Using Behaviour Classes

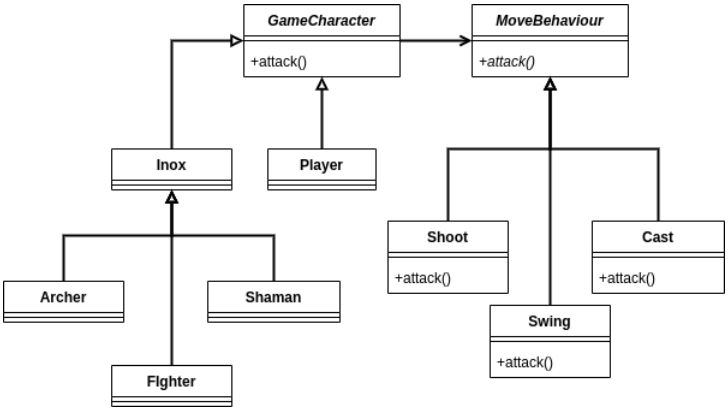
New solution

- Encapsulate behaviour inside a behaviour class
- Entity object **delegates** behaviour to a different object
- Entity can point to a different behaviour object.
  - Same function call leads to different code being run.
  - Entity behaviour is changed.

Using Behaviour Classes

The `attack()` function is in the `GameCharacter` class

- no longer virtual
- `GameCharacter::attack()` calls `MoveBehaviour::attack()`
- `MoveBehaviour::attack()` is virtual
- The derived class has its `attack()` called at runtime



Behaviour Classes

Advantages

- We can introduce new behaviours without changing existing code.
  - Introducing a new weapon or weapon type is a breeze!
  - Entity objects are unchanged.
- We can reuse and share behaviours
  - Any humanoid can use `Swing` or `Cast`
- Behaviour changes easily at runtime
  - Pick up a different weapon, change the behaviour object to correspond.

Problem

- We still must allocate and deallocate behaviour objects to change behaviour.
- Potential solution - Can have a shared "behaviour pool"

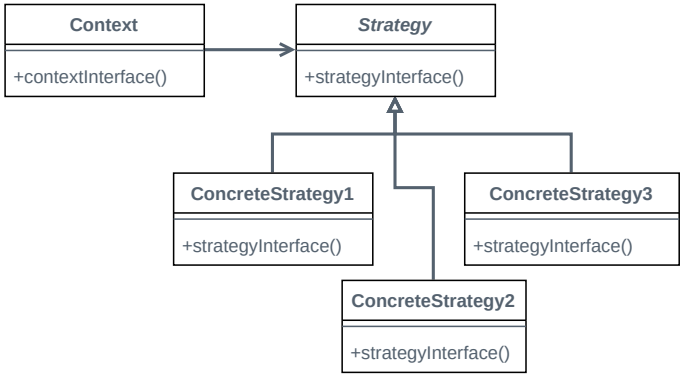
coding <p8>

Strategy Design Pattern

Behaviour classes are an example of the **Strategy Design Pattern**

- Strategy is a **behavioural** design pattern
- A Strategy provides a family of algorithms.
  - We define an abstract interface for the algorithms.
  - Derived classes implement an algorithm.
  - Concrete implementations are interchangeable at runtime.

Strategy Design Pattern



Strategy Design Pattern