Chap. 15
Design Patterns

Object Oriented Systems Analysis and Design Using UML, (4th Edition), McGraw Hill

Topics Covered

- Software Development Patterns
- Documenting Patterns – Pattern Templates
- Design Patterns
- How to Use Design Patterns
- Benefits and Dangers of Using Patterns

Patterns

- A pattern is proven solution to a problem that recurs in a particular context
- Are *Discovered*, not *invented* - they already exist
- Capture knowledge about problems and successful solutions
- Experience that has been gained in the past can be reused in similar situations

Pattern Template

- *Name*
  - meaningful that reflects the knowledge embodied by the pattern
- *Problem*
  - description of the problem that the pattern addresses (the intent of the pattern).
- *Context*
  - represents the circumstances or preconditions under which it can occur.
- *Forces*
  - embodied in a pattern are the constraints or issues that must be addressed by the solution
- *Solution*
  - description of the static and dynamic relationships among the components of the pattern

Other aspects of Templates

- The rationale that justifies the chosen solution
- Known uses of the pattern that validate it (some suggest that until the problem and its solution have been used successfully at least three times—the rule of three—they should not be considered as a pattern)
- A list of aliases for the pattern ('also known as' or AKA)
- Sample program code and implementation details (commonly used languages include C++, Java and Smalltalk)
- Related patterns

GOF Design Patterns

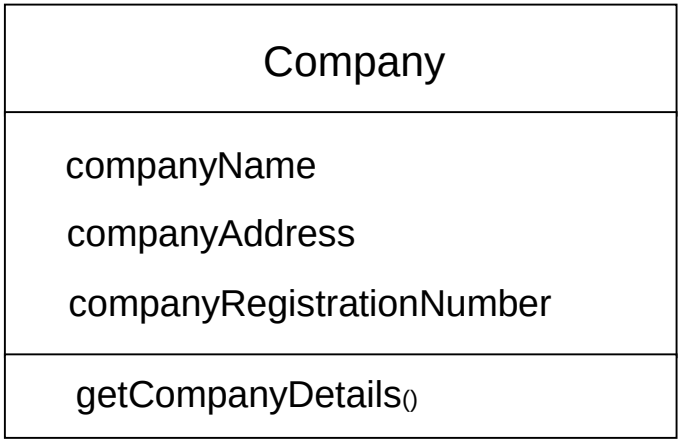
- Catalogue of 23 design patterns presented by Gamma et al. (1995) patterns
- Known as Gang of Four – hence GOF Patterns
- Classified as creational, structural or behavioural

Creational Patterns

- Concerned with the construction of object instances
- Separate the operation of an application from how its objects are created
- Gives the designer considerable flexibility in configuring all aspects of object creation

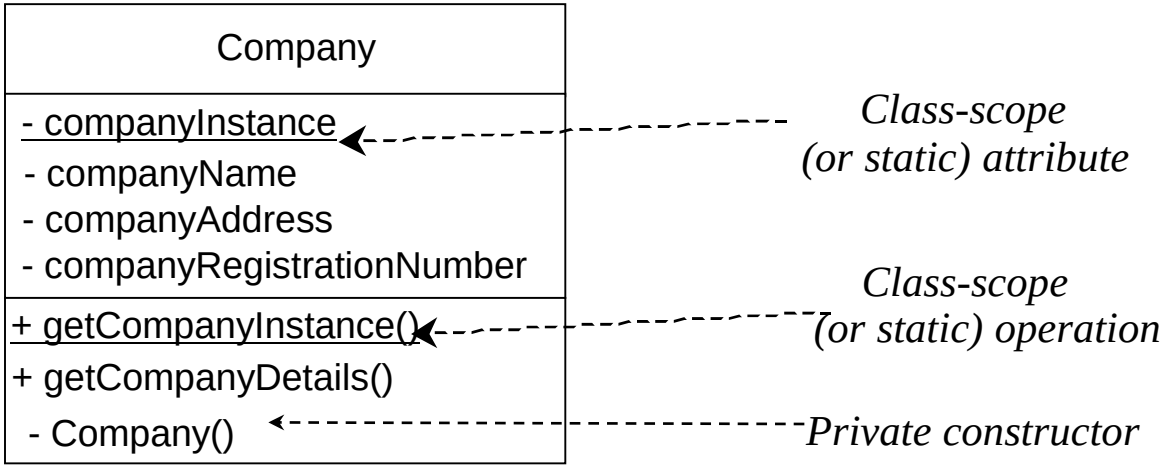
Creational Patterns: Singleton

- How does one ensure that only one instance of the company class is created?



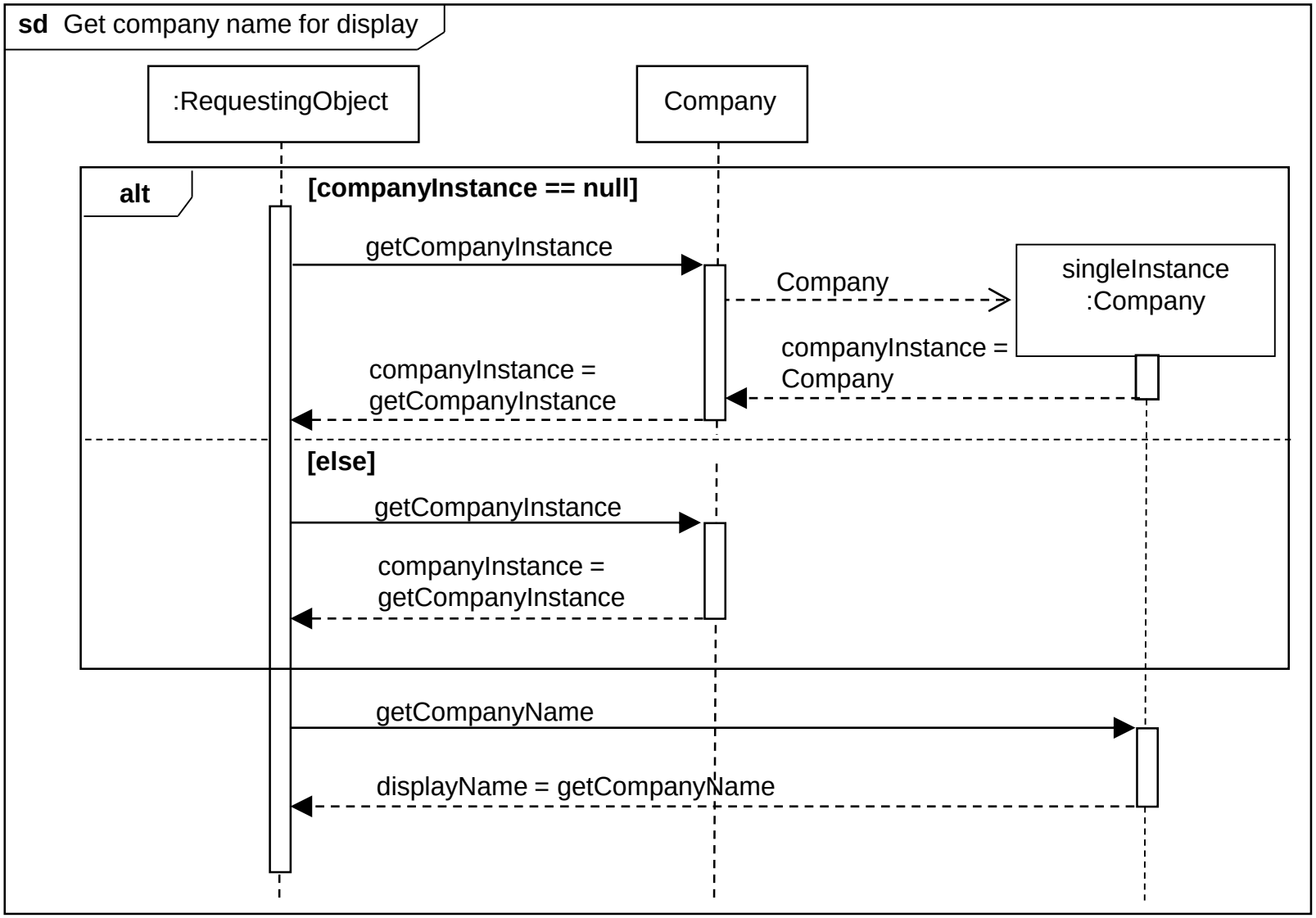
Creational Patterns: Singleton

- Solution – restrict access to the constructor!

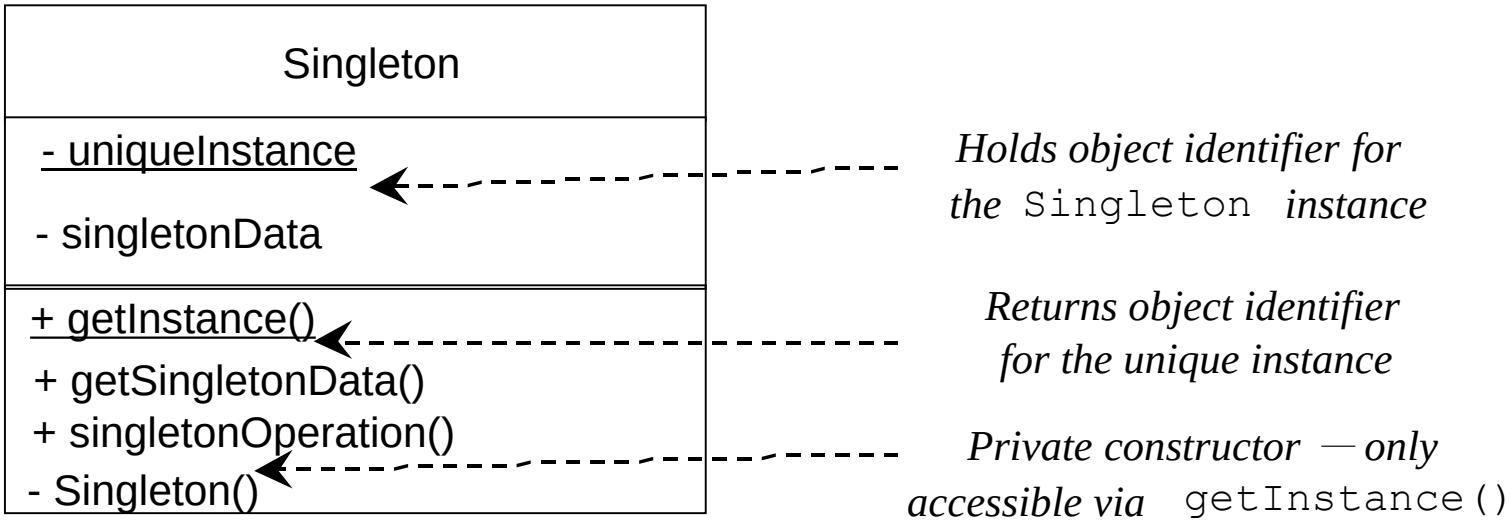


The use of class-scope operations allows global access

Singleton: Sequence Diagram



Creational Patterns: Singleton



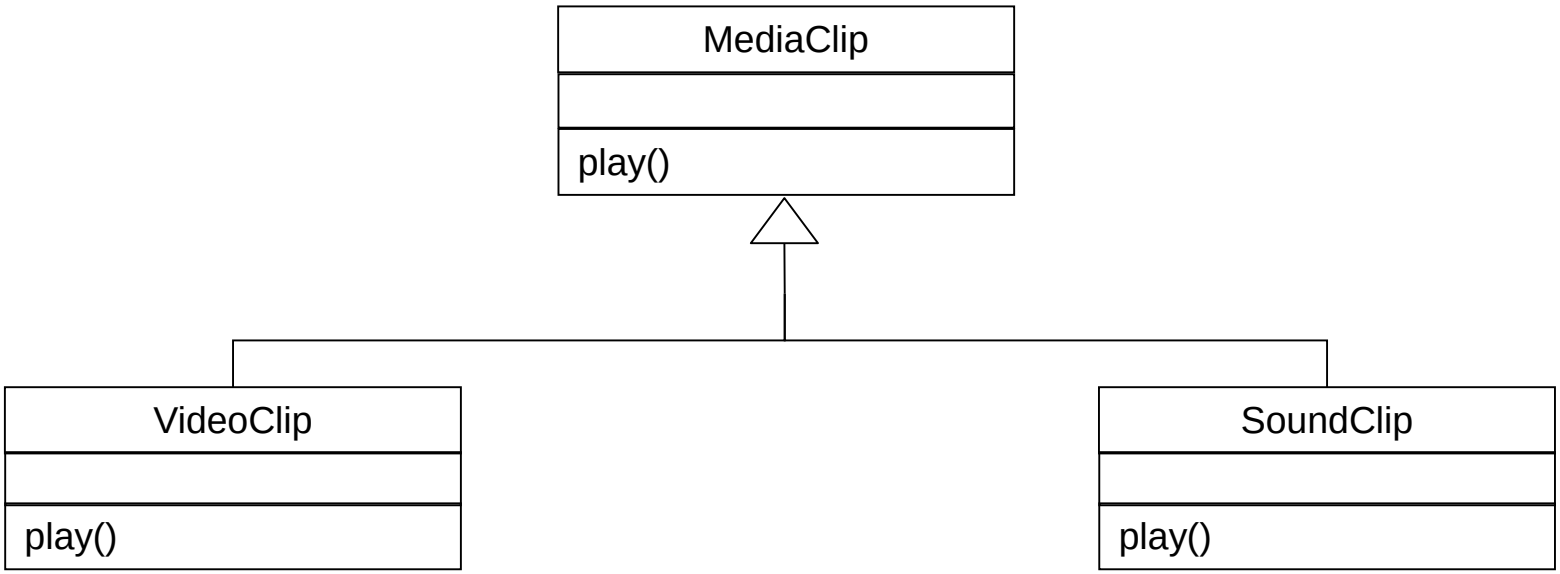
General form of Singleton pattern

Structural Patterns

- Concerned with the way in which classes and objects are organized
- Offer effective ways of using object-oriented constructs such as inheritance, aggregation and composition to satisfy particular requirements

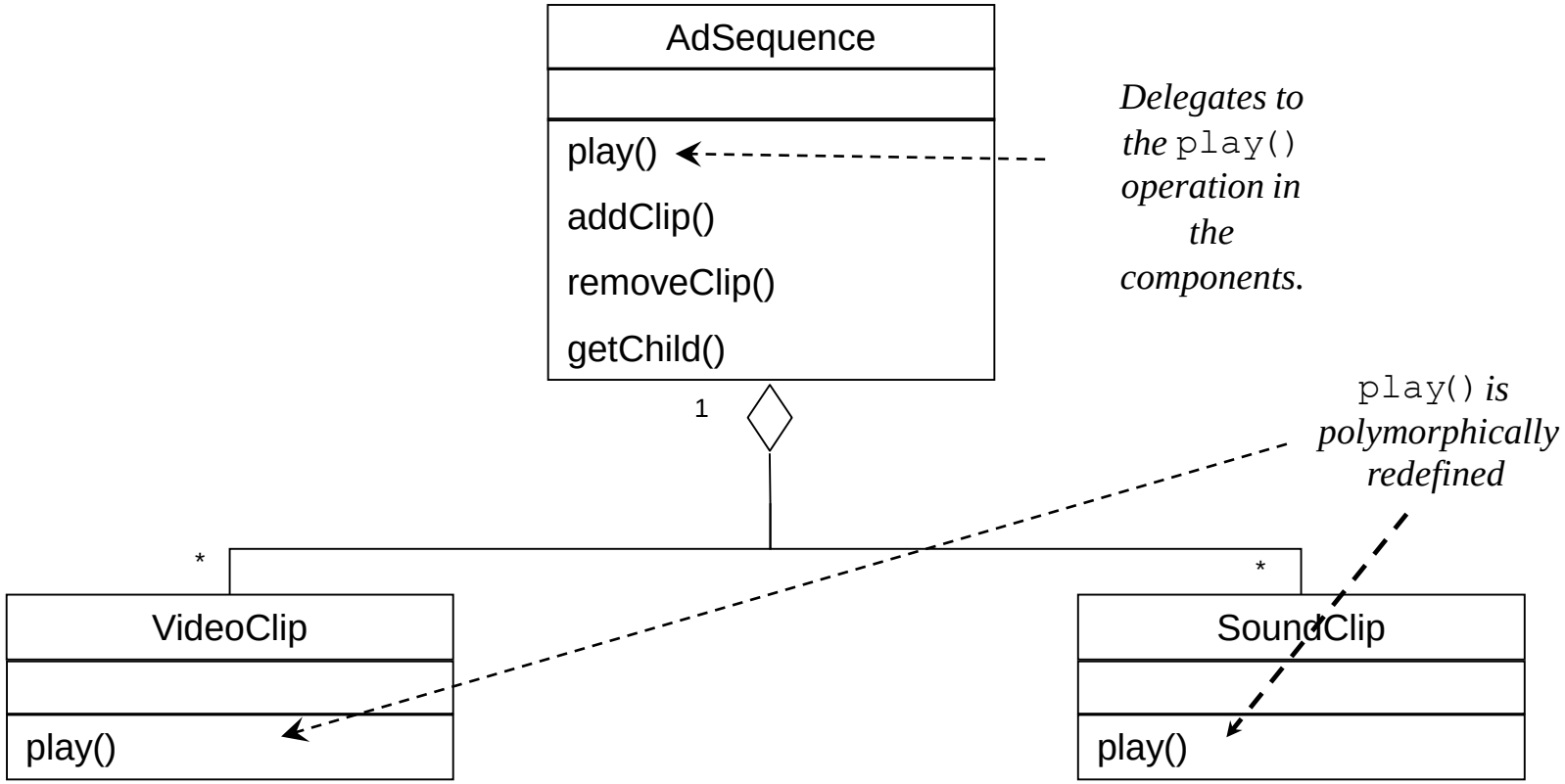
Structural Patterns: Composite

- How can we present the same interface for a media clip whether it is composite or not?

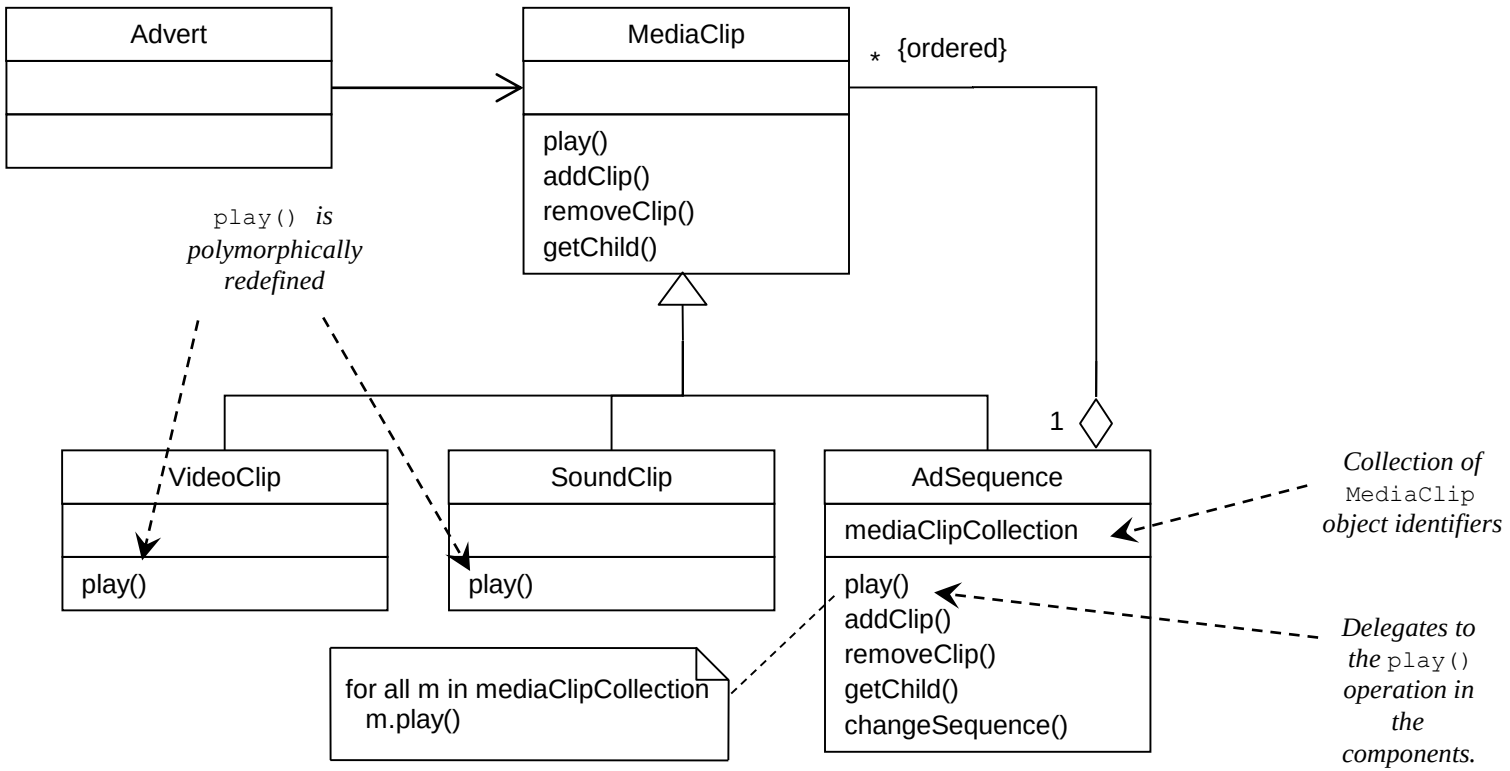


Structural Patterns: Composite

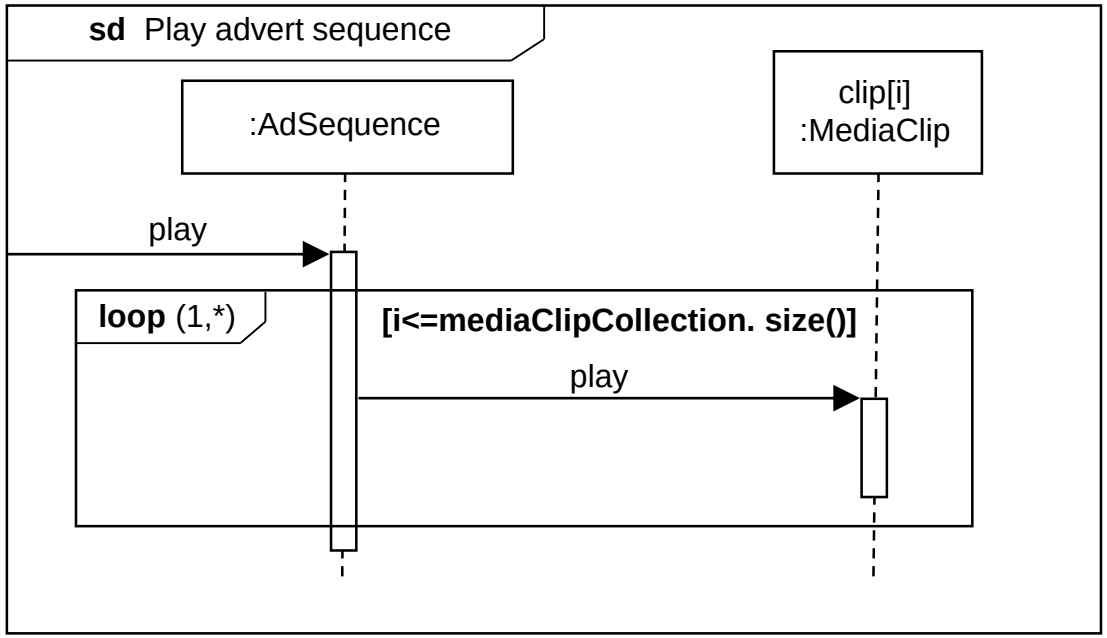
How can we incorporate composite structures?



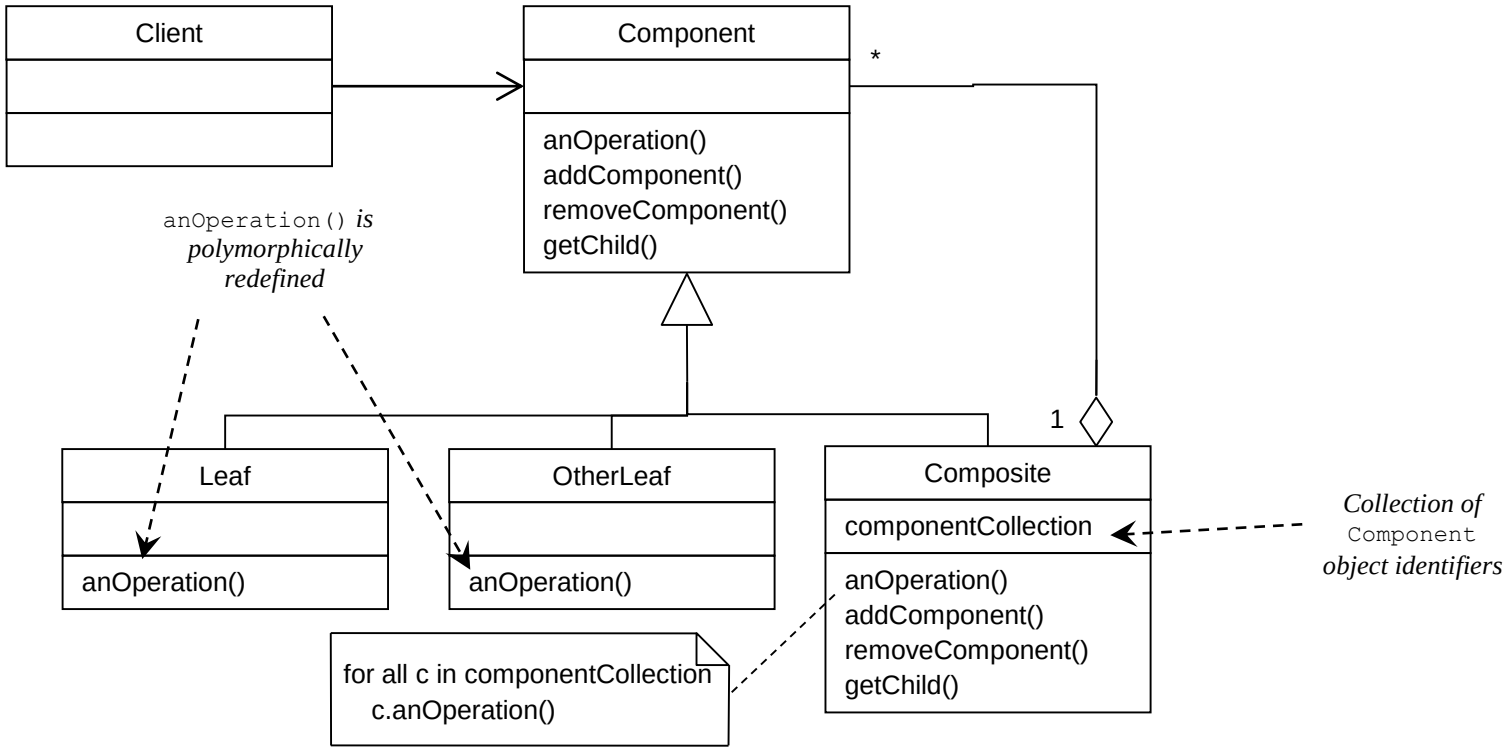
Composite applied to Agate



Composite Pattern: Sequence Diagram



Composite Pattern
General Form



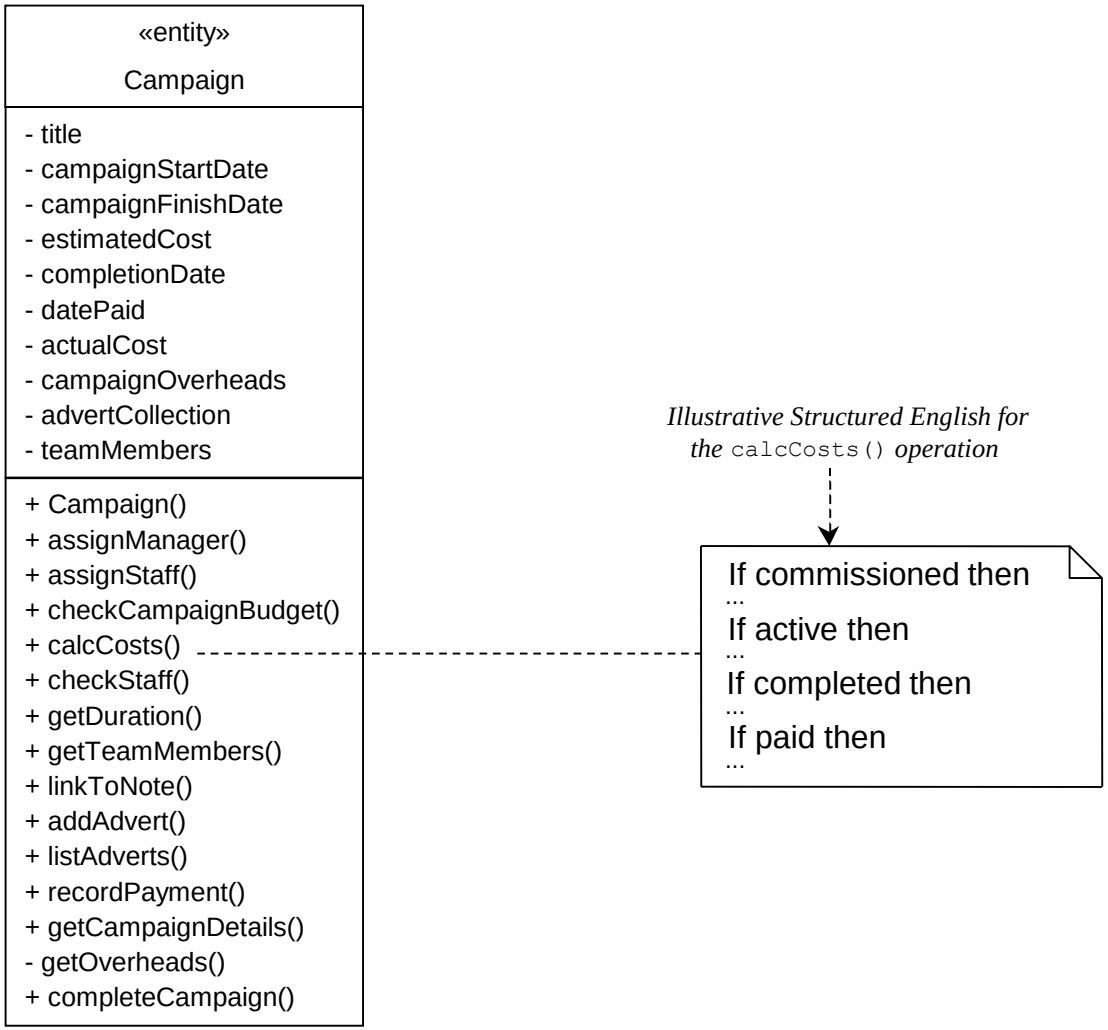
Behavioural Patterns

- Address the problems that arise when assigning responsibilities to classes and when designing algorithms
- Suggest particular static relationships between objects and classes and also describe how the objects communicate

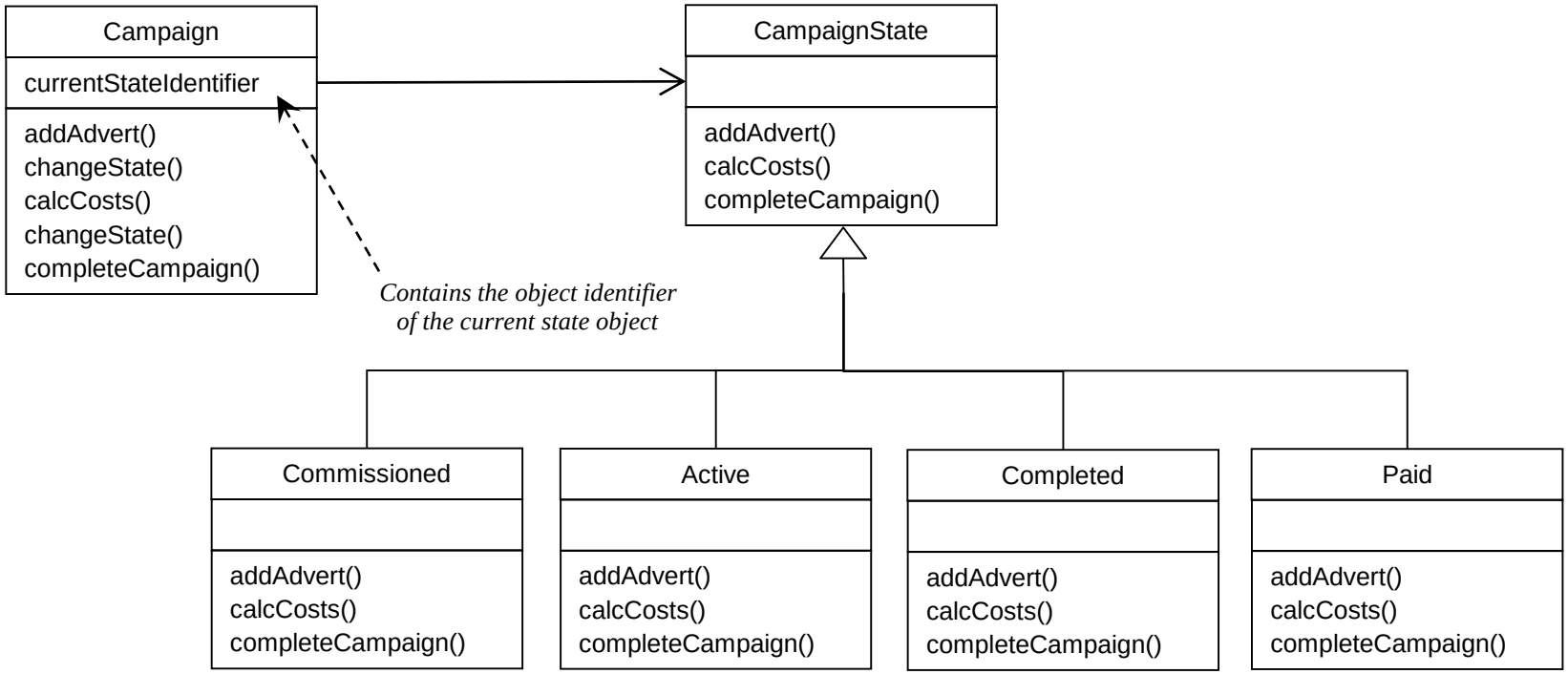
Behavioural Patterns: State

- Consider the class `Campaign`.
- It has four states – Commissioned, Active, Completed and Paid
- A Campaign object has different behaviour depending upon which state it occupies.
- Operations have case statements giving this alternative behaviour
- The class factored into separate components – one for each of its states

Campaign class: could have state pattern applied

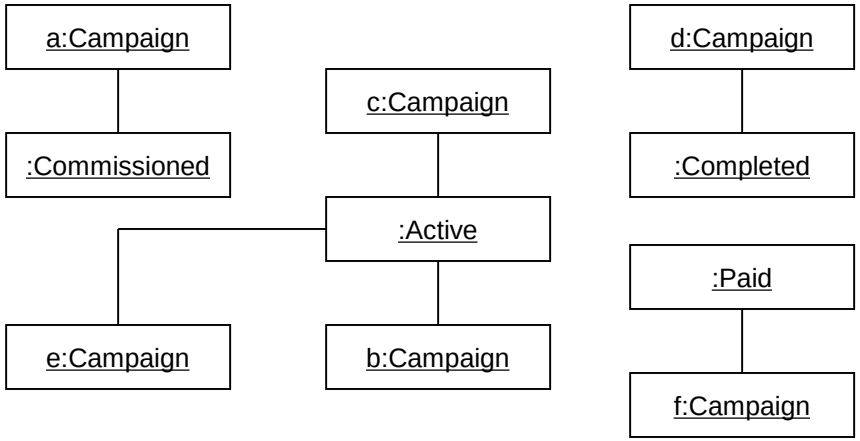


Behavioural Patterns: State



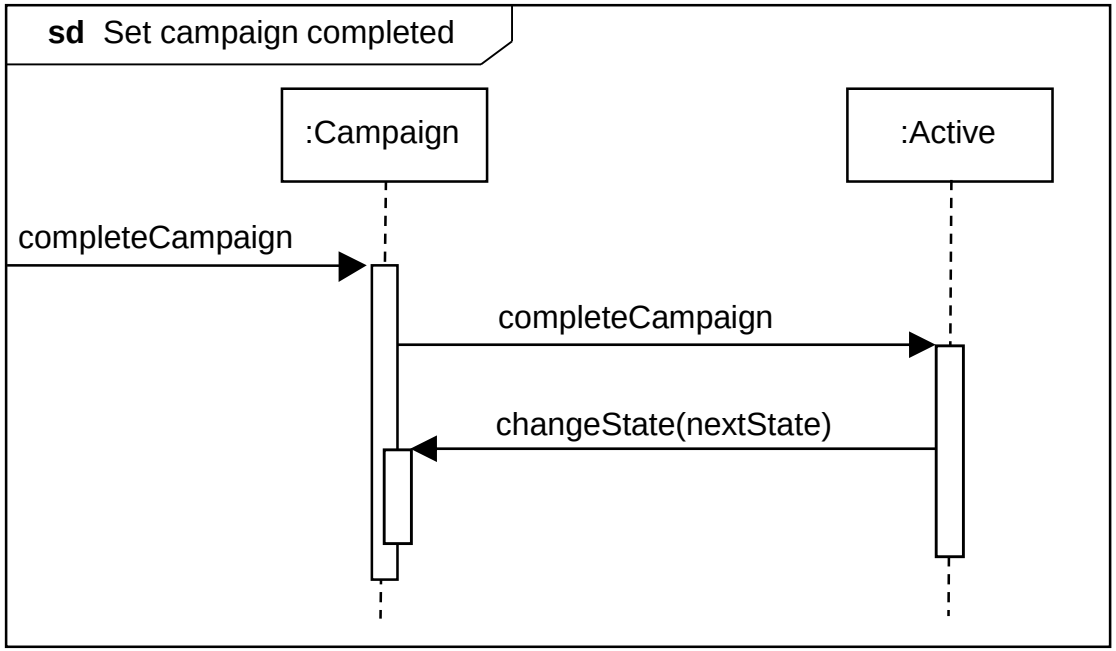
State pattern applied to the class **`Campaign`**

Behavioural Patterns: State

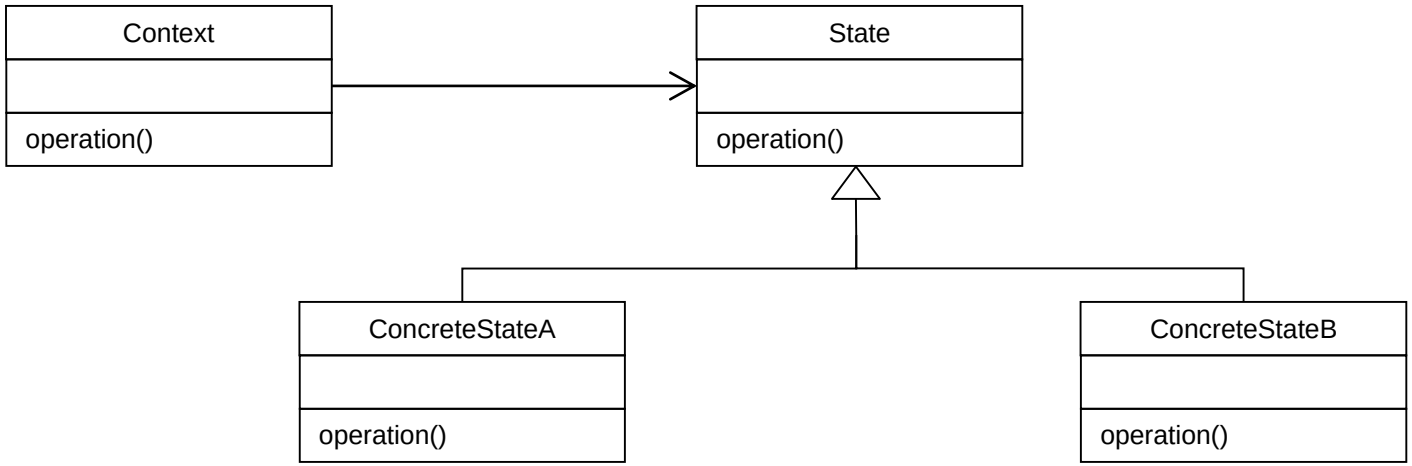


Some State pattern objects for Agate – note that there are 6 **`Campaign`** objects sharing the four State objects.

State Pattern: Sequence Diagram



General form of State Pattern



Before Using Patterns

- Before using a pattern to resolve the problem, ask
  - Is there a pattern that addresses a similar problem?
  - Does the pattern trigger an alternative solution that may be more acceptable?
  - Is there a simpler solution? Patterns should not be used just for the sake of it
  - Is the context of the pattern consistent with that of the problem?
  - Are the consequences of using the pattern acceptable?

Using Patterns

- After selecting a suitable pattern
  1. Read the pattern to get a complete overview
  2. Study the Structure, Participants and Collaborations of the pattern in detail
  3. Examine the Sample Code to see an example of the pattern in use

Using Patterns

4. Choose names for the pattern's participants (i.e. classes) that are meaningful to the application
5. Define the classes
6. Choose application specific names for the operations
7. Implement operations that perform the responsibilities and collaborations in the pattern