CS4125
SYSTEMS ANALYSIS
SPRING SEMESTER 2010-2011

J.J. Collins
Dept of CSIS
University of Limerick

Lecture Q
W11L1 / W11L2

Introduction to Design Patterns

1. Introduction to Patterns

- Patterns provide a means for capturing knowledge about a problem and successful solutions in software development.
- Experience that has been gained in the past can now be reused in similar situations. Economic benefits.
- The architect Christopher Alexander first used the term *pattern*. Addressed many structural issues such as best way to design a waiting room.
- *Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of a solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice* - Alexander (1977).

1. Introduction to Patterns

- The Layer, Broker and MVC architectures are examples of the application of patterns from Buschmann et al. (1996) to the activity of system design.
- Coad et al. (1997) describe a pattern as a template that embodies an example worth emulating.
- Anti-patterns capture practice that is demonstrably bad.
- An anti-pattern can also include reworked solutions that proved effective ((Brown et al., 1998). i.e. Mushroom Management and the use of DSDM.

1: Introduction to Patterns

Design Patterns

- Description of communicating objects and classes that are customized to solve a general design problem in a particular context.

  (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns – Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994 (22nd printing July 2001))

- Each pattern focuses in a particular object-oriented design problem or issue

1. Introduction to Patterns

- In the catalogue of 23 design patterns presented by the Gang of Four (GOF) in their book "Design Patterns" (1995), patterns are classified by their purpose: creational, structural or behavioural.
- Scope may be primarily at either the class level or object level.
- Patterns that are scoped at object level describe relationships that may change at run-time and hence are more dynamic.

1. Introduction to Patterns

Categories of Design Patterns

- Creational
  - Deal with the best way to create objects
- Structural
  - Ways to bring together groups of objects
- Behavioral
  - Ways for objects to communicate & interact

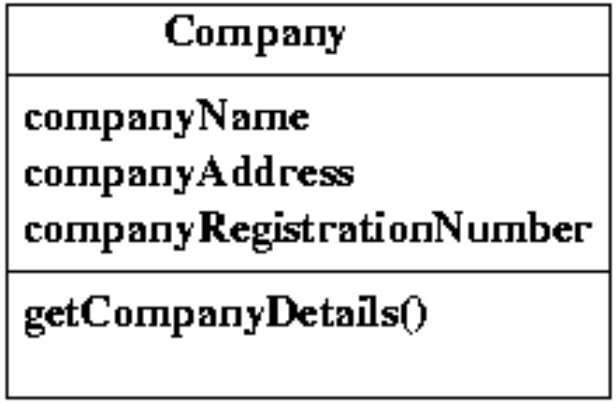
1. Introduction to Patterns

GOF Design Patterns		Purpose		
		Creational	**Structural**	**Behavioural**
Scope	**Class**	Factory Method	Adapter (Class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (Object) Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Not Examinable!

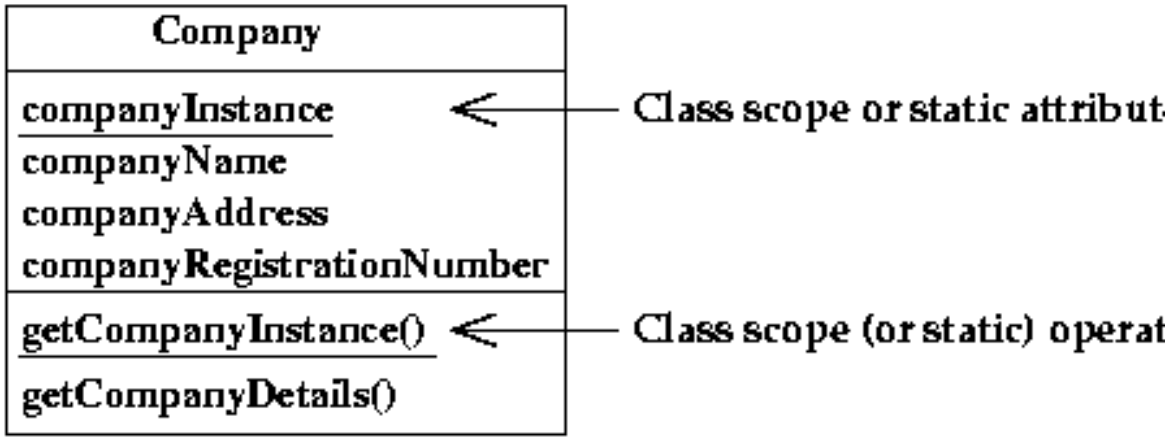
2. Example: The GoF - The Singleton Creational Design Pattern

- Examples of Creational design patterns are Abstract Factory, Builder, Factory Method, Prototype, and Singleton.
- A creational design pattern is concerned with the construction of object instances.
- Separation of operation of an application from how its objects are created.
- Facilitates flexibility in configuring all aspects of object creation.
- Configuration may be static or dynamic.
- An example - the Singleton pattern, used to ensure that only one instance of a class is created.
- The business system needs to hold information about the company, which should be stored in one place only in the system, and accessible by all other objects.



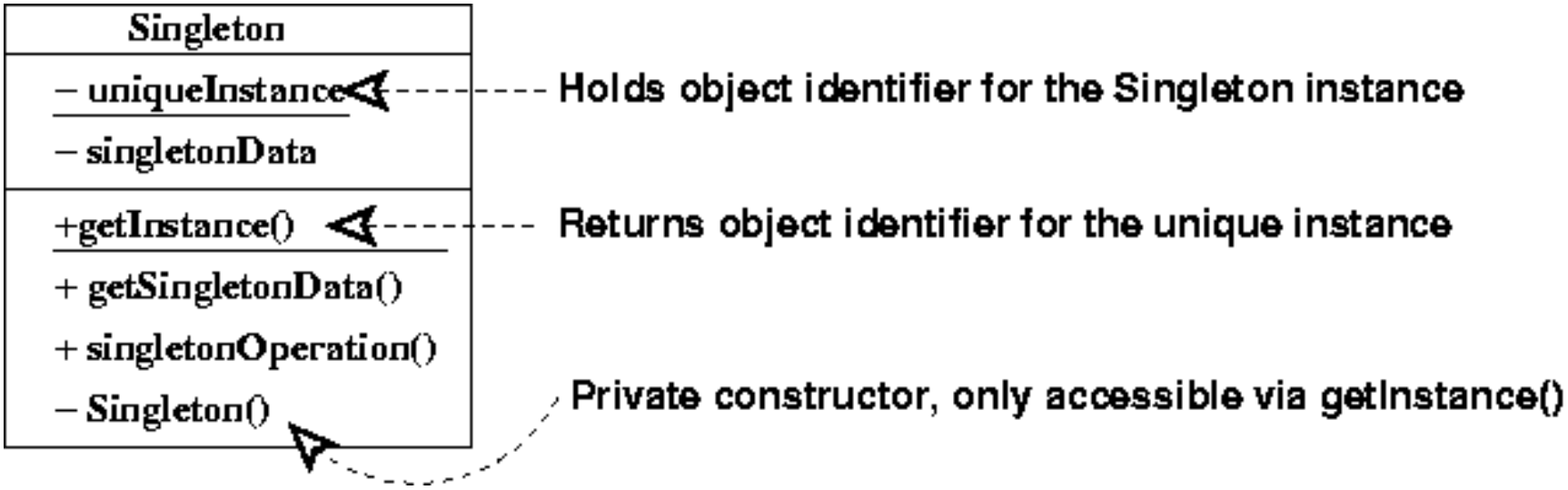
2. Example: The GoF - The Singleton Creational Design Pattern

- Creation of a company class. Problem: all objects that want to use the Company class must know its identifier.
- May make the Company object identifier globally accessible - violates encapsulation.
- Some object-oriented languages provide a mechanism that enables certain types of operations to be accessed without reference to a specified object, called *class or static operations*.
- Example: a static operation getCompanyInstance() can be defined in such a way that it will provide a client object with the identifier for the Company instance:
- Company.getCompanyInstance()



2. Example: The GoF - The Singleton Creational Design Pattern

- The Singleton pattern offers three advantages:
  - Provides controlled access to the sole object instance.
  - The name space is not extended with global variables.
  - The Singleton class may be subclassed.



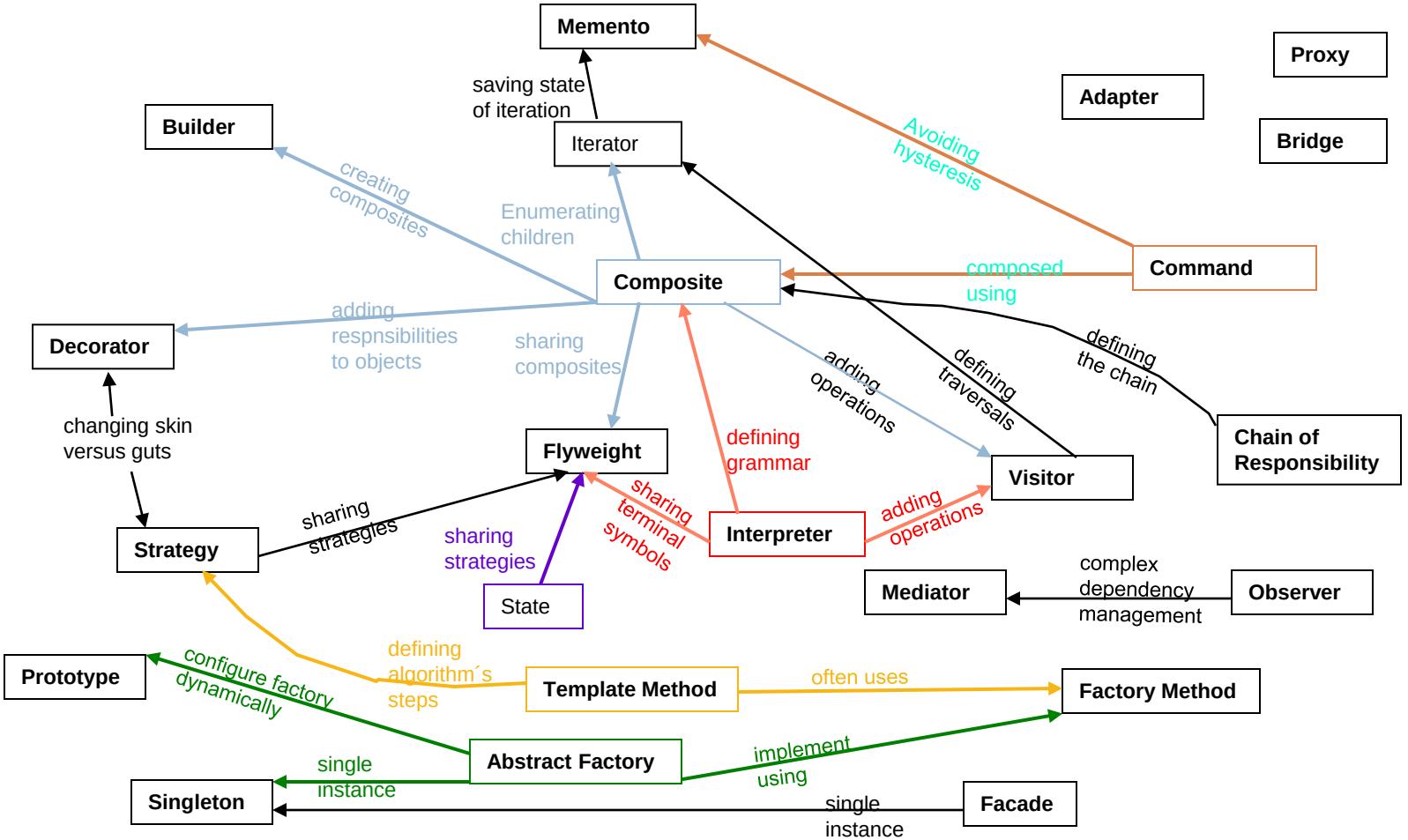
3. Patterns

- Coplien (1992) catalogued a set of patterns specifically for use in C++ programming. Patterns that are related to constructs in a programming language are known as idioms.
- Design patterns were popularised by Gamma et al. (1995) in their book *Design Patterns: Elements of Reusable Object Oriented Software*.
- Other authors have identified patterns that are concerned with Analysis (Fowler, 1997), organisational issues (Coplien, 1996) and systems architecture using CORBA (OMG, 1995).
- Kerievsky – refactoring to patterns published in 2005.
- Buschmann et al. (1996) suggests the following categories:
  - Architectural Patterns: identifies subsystems, their responsibilities, and their inter-relationships.
  - Design patterns.
  - Idioms.
  - Analysis patterns are defined as describing groups of concepts that represent common constructions in domain modelling.

3. Pattern Catalogues

- Patterns are grouped into catalogues and languages.
- A pattern catalogue is a group of patterns that are related and may be used together or independently of each other.
- Cunningham (1995) documented the *Check Pattern Language of Information Integrity*, which consists of eleven patterns that address issues of data validation.
- One of these patterns, Echo, describes how data should be echoed back to the user after it has been modified and validated by the information system.
- Patterns can address the issues that are raised by non-functional requirements.
- Buschmann (1996) identifies the following important non-functional properties of system architecture: changeability, interoperability, efficiency, reliability, testability and reusability.

3. Pattern Catalogues: Relations among Design Patterns



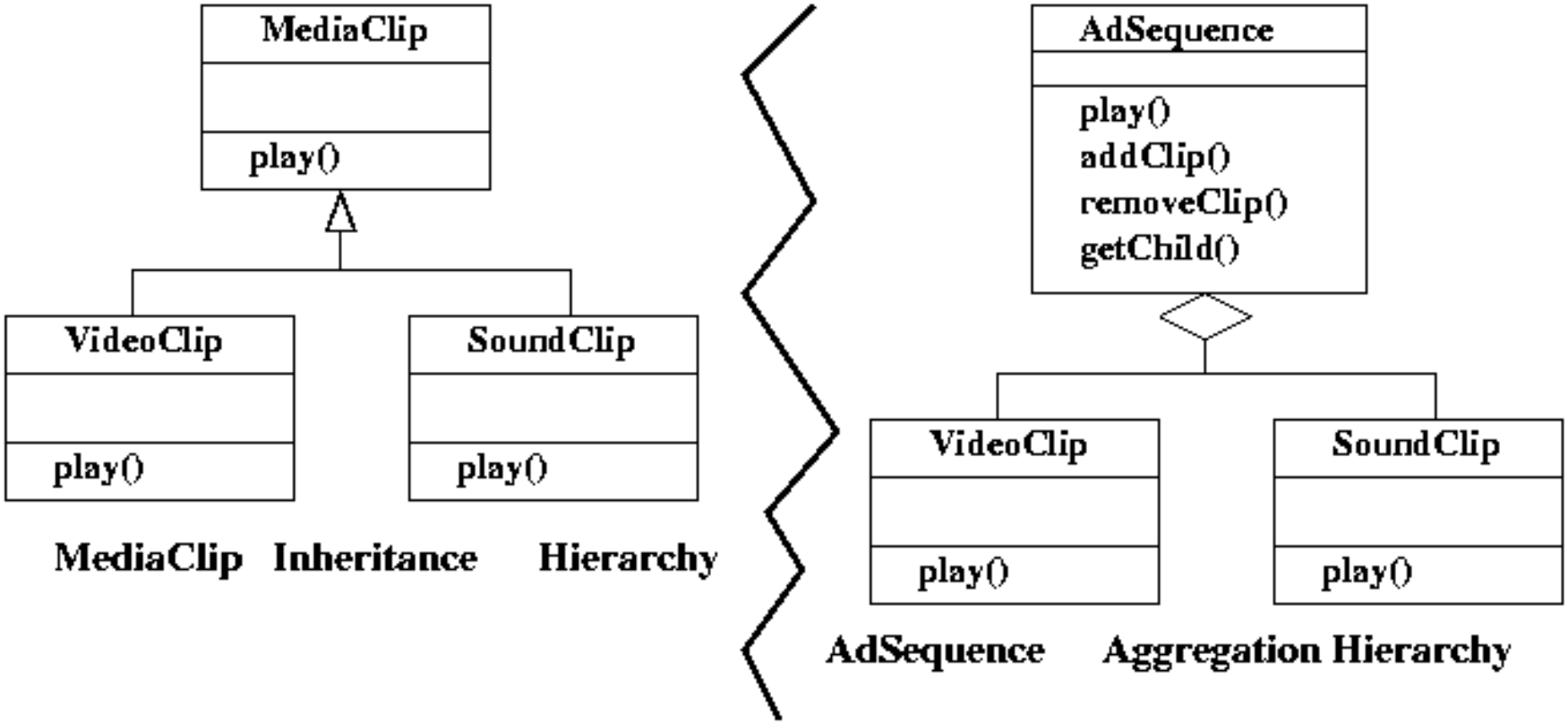
Not Examinable!

3. Frameworks

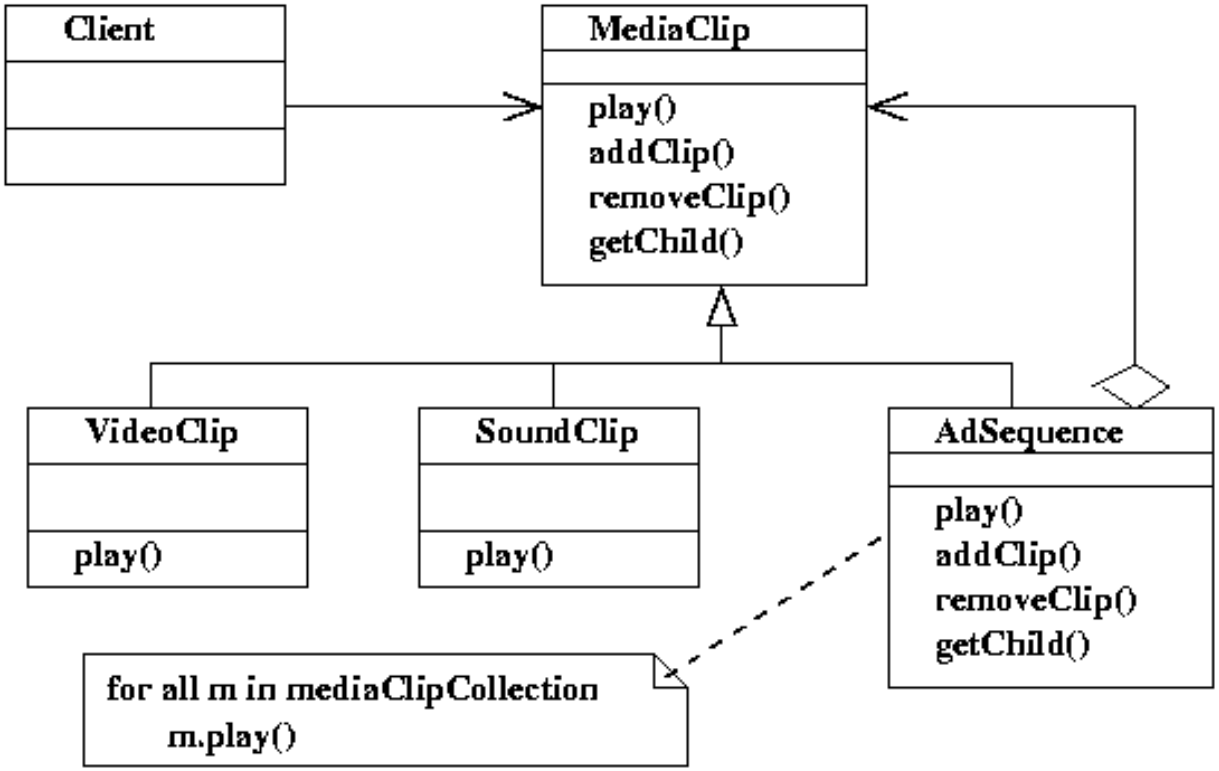
- Frameworks are partially completed software systems that may be targeted at a specific type of application, i.e. IBM's Webshere (formerly San Francisco) sales order processing framework.
- The framework is a mini-architecture that provides structure and behaviour common to all applications of this type.
- Difference between patterns and frameworks:
  - Patterns are more abstract and general than frameworks. A pattern is a description of the way that a type of problem can be solved, but the pattern itself is not a solution.
  - Patterns are more primitive than frameworks. A framework can employ several patterns, but a pattern cannot incorporate a framework.

4. The GoF: The Composite Structural Design Pattern

- Examples include Adapter, Bridge, Composite,



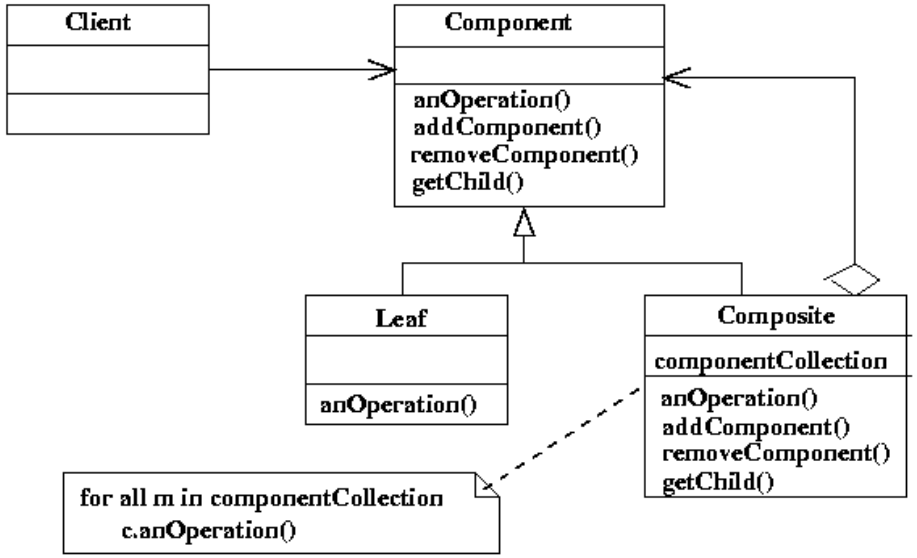
4. The GoF: The Composite Structural Design Pattern



Integrating the two hierarchies for MediaClip

- 2 orthogonal hierarchies can be integrated by treating AddSequence both as a subclass of MediaClip and also as an aggregation of MediaClip objects.

4. The GoF: The Composite Structural Design Pattern

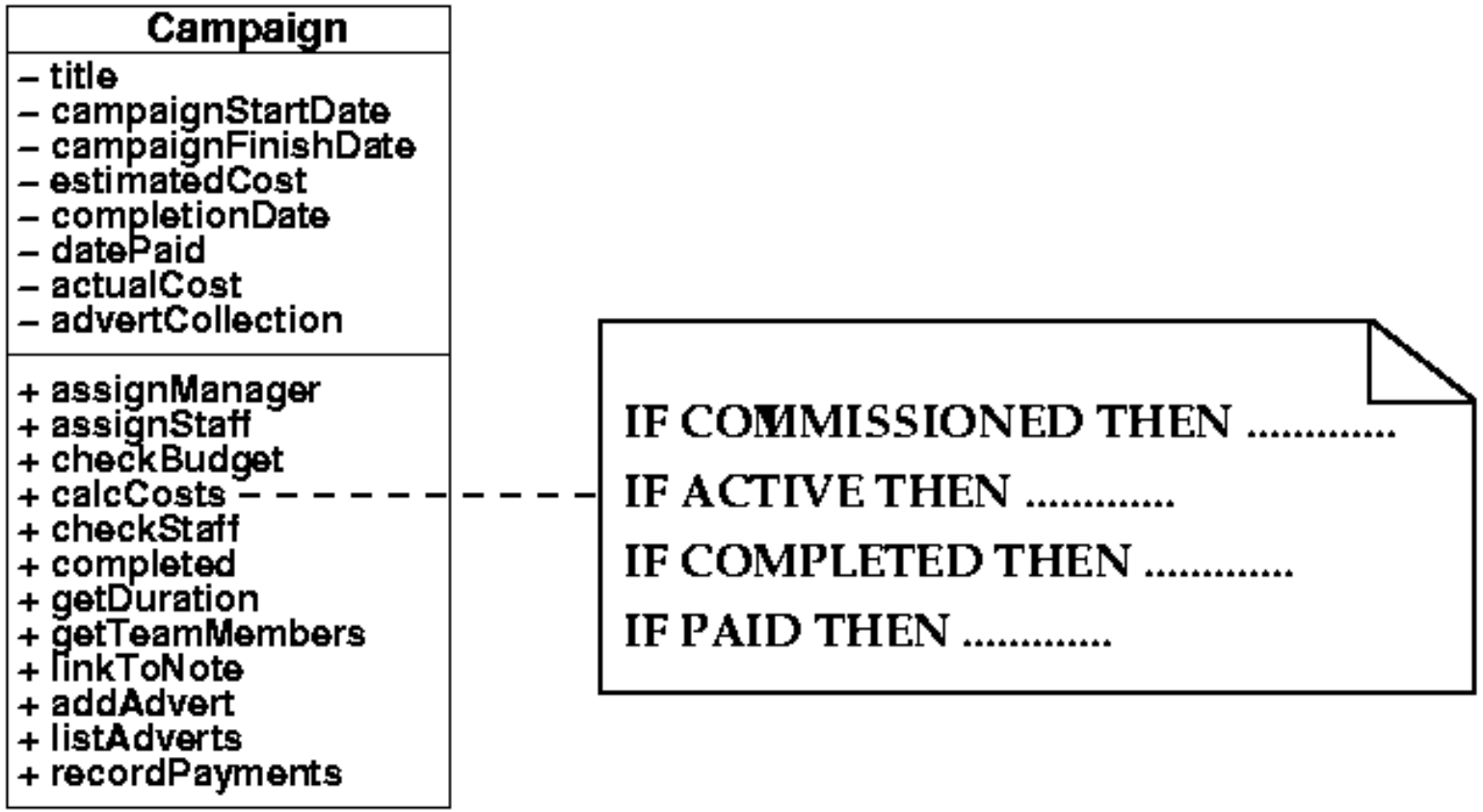


Structural Patterns – Composite

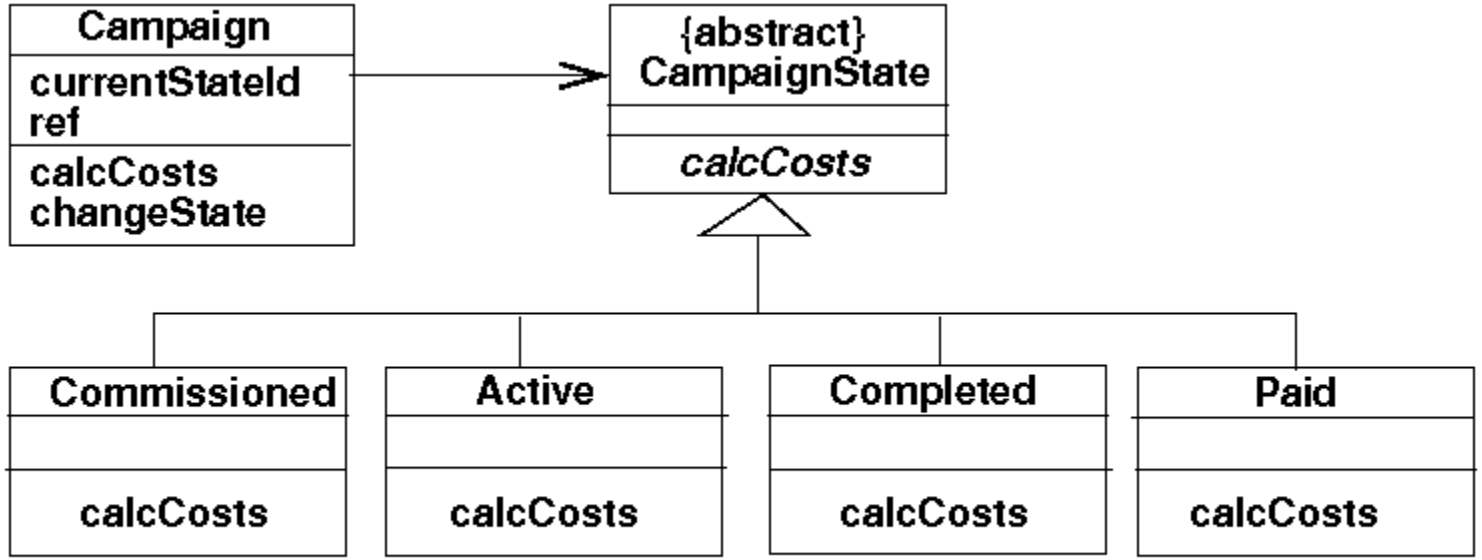
1	**Name**	**Composite**
2	**Problem**	**Requirement to represent aggregation so that whole and part offer the same interface to clients.**
3	**Context**	**In an application, both composite and component objects are required to present the same interface. A commonly used example is a graphical package. A user can create atomic objects such as circles or squares, and also create composite objects from the atomic objects.**
4	**Forces**	**The requirement that the objects present the same interface suggests that they belong to the same inheritance hierarchy.**
5	**Solution**	**The solution combines inheritance and aggregation hierarchies.**

5. The GoF: The State Behavioural Design Pattern

- Examples include Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, and Visitor.



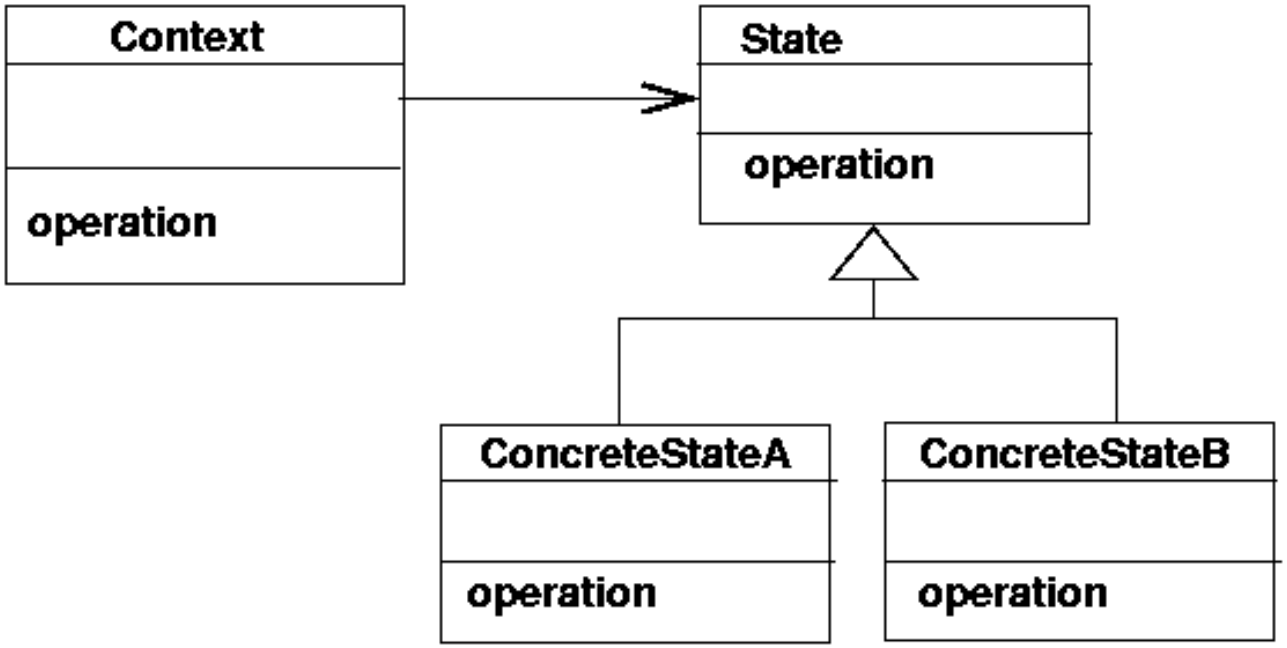
5. The GoF: The State Behavioural Design Pattern



State pattern for Campaign

1	**Name**	**State**
2	**Problem**	**An object's exhibits different behaviour when its internal state changes, making it appear that the object changes class at runtime.**
3	**Context**	**An object may have complex behaviour that is heavily state dependent. An example is calcCost() in the Campaign class.**
4	**Forces**	**The object's complex behaviour should be factored into classes that are abstractions of internal state.**
5	**Solution**	**Separate the state dependent behaviour from the original class.**

5. The GoF: The State Behavioural Design Pattern



Behavioural patterns – State

6. GoF Principles

- Favour delegation over inheritance
  - White box (inheritance) versus black box (delegation) reuse
  - Black box reuse is more flexible
  - Cannot change the implementation being inherited at runtime with white box reuse
  - Object composition defined dynamically at runtime through objects acquiring references to other objects.
- Program to interfaces, not implementation

7. Documenting Patterns

- Employ pattern templates.
- Pattern templates determine the style and structure of pattern description,
- A pattern description should include the following:
  - Name.
  - Problem description - identify objectives to be achieved, within a specified context and constraining forces.
  - Context: circumstances or preconditions under which problem can occur.
  - Forces - constraints or issues that must be addressed by solution.
  - Solution - description of static and dynamic relationships among the pattern model elements.

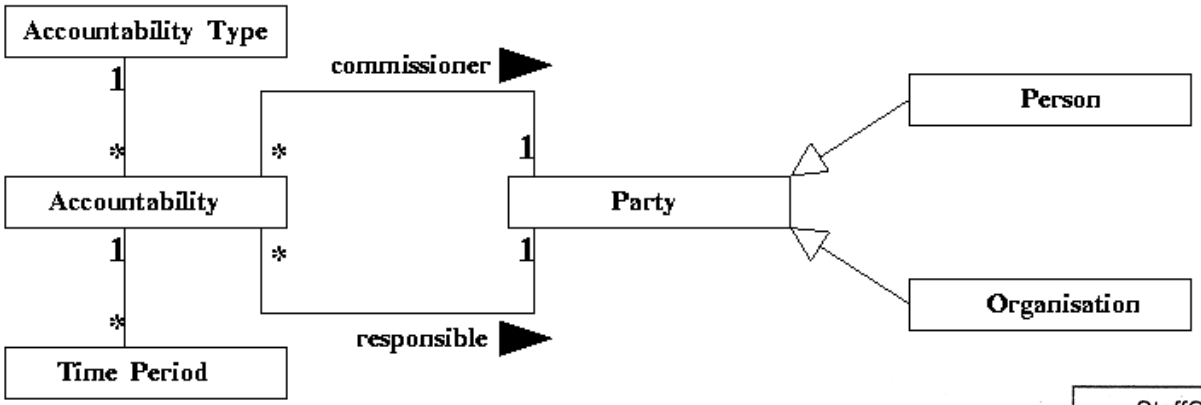
8. Software Development Principles and Patterns

- Patterns are based on good design principles.
- Buschmann (1996) suggests that the following are the key principles that drive the evolution of good patterns:
  1. Abstraction
  2. Encapsulation
  3. Information hiding
  4. Modularisation
  5. Separation of concerns
  6. Coupling and cohesion
  7. Sufficiency, completeness and primitiveness
  8. Separation of policy and implementation.
  9. Separation of interface and implementation.
  10. Single point of reference
  11. Divide and conquer.

9. Reading

- Bennett, McRobb, and Farmer:
  - Section 5 from chapter 8, excluding 8.5.3
  - Chapter 15.

Appendix: Fowler - The Accountability Analysis Patterns



In the Agate case study, accountability exists between

- A manager and a member of staff.
- A client and a campaign manager.
- A client and the staff contact.
- Will not be examined!