Design Patterns

Software Development Patterns (1)

- Patterns: a means for capturing knowledge about problems and successful solutions
- Framework: partially completed software system targeted at a particular type of application
  - Reusable mini-architecture
  - Class extension and operation implementation
- Patterns versus frameworks
  - Patterns are more abstract and general
  - Patterns cannot be directly implemented in a particular software environment
  - Patterns are more primitive

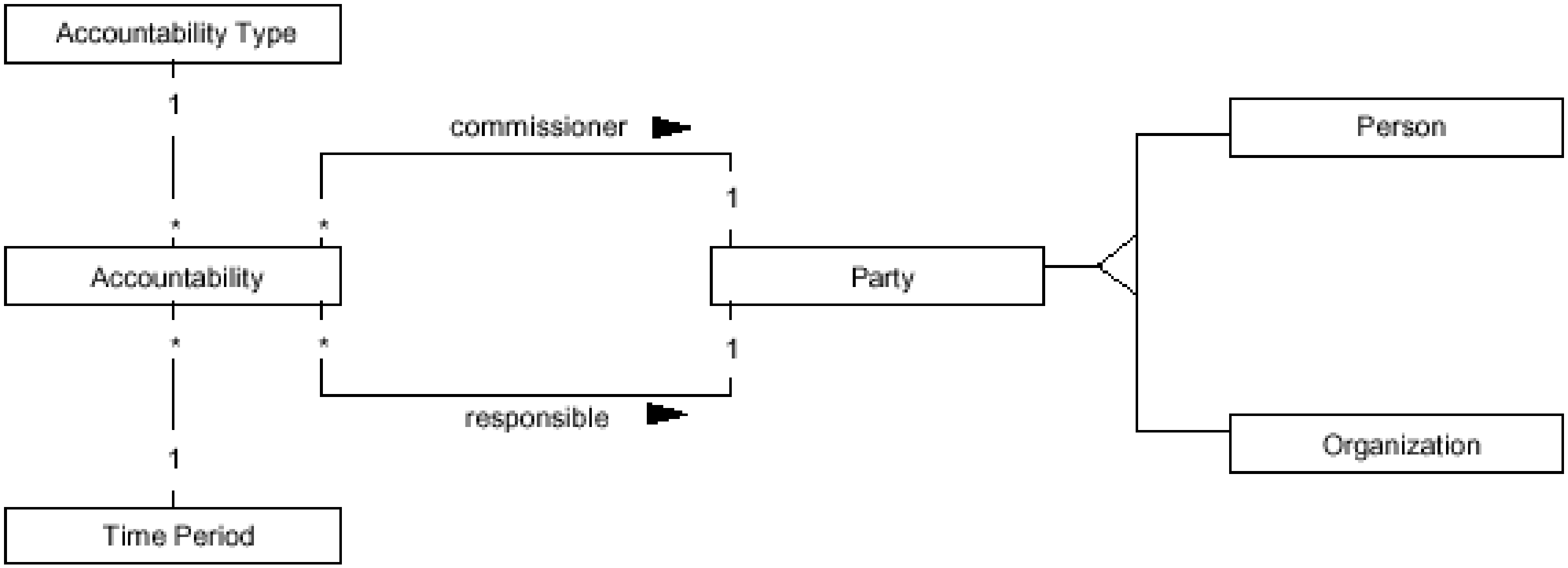
Software Development Patterns (2)

- Collection of patterns: pattern catalogue, pattern language (specific domain, completeness), pattern system (classification scheme and relationships)
- Key principles underlying patterns
  - Abstraction, encapsulation, information hiding, modularisation, separation of concerns, coupling and cohesion, sufficiency-completeness and primitiveness, separation of policy and implementation, separation of interface and implementation, single point of reference, divide and conquer
- Patterns and non-functional requirements
  - Changeability, interoperability, efficiency, reliability, testability and reusability

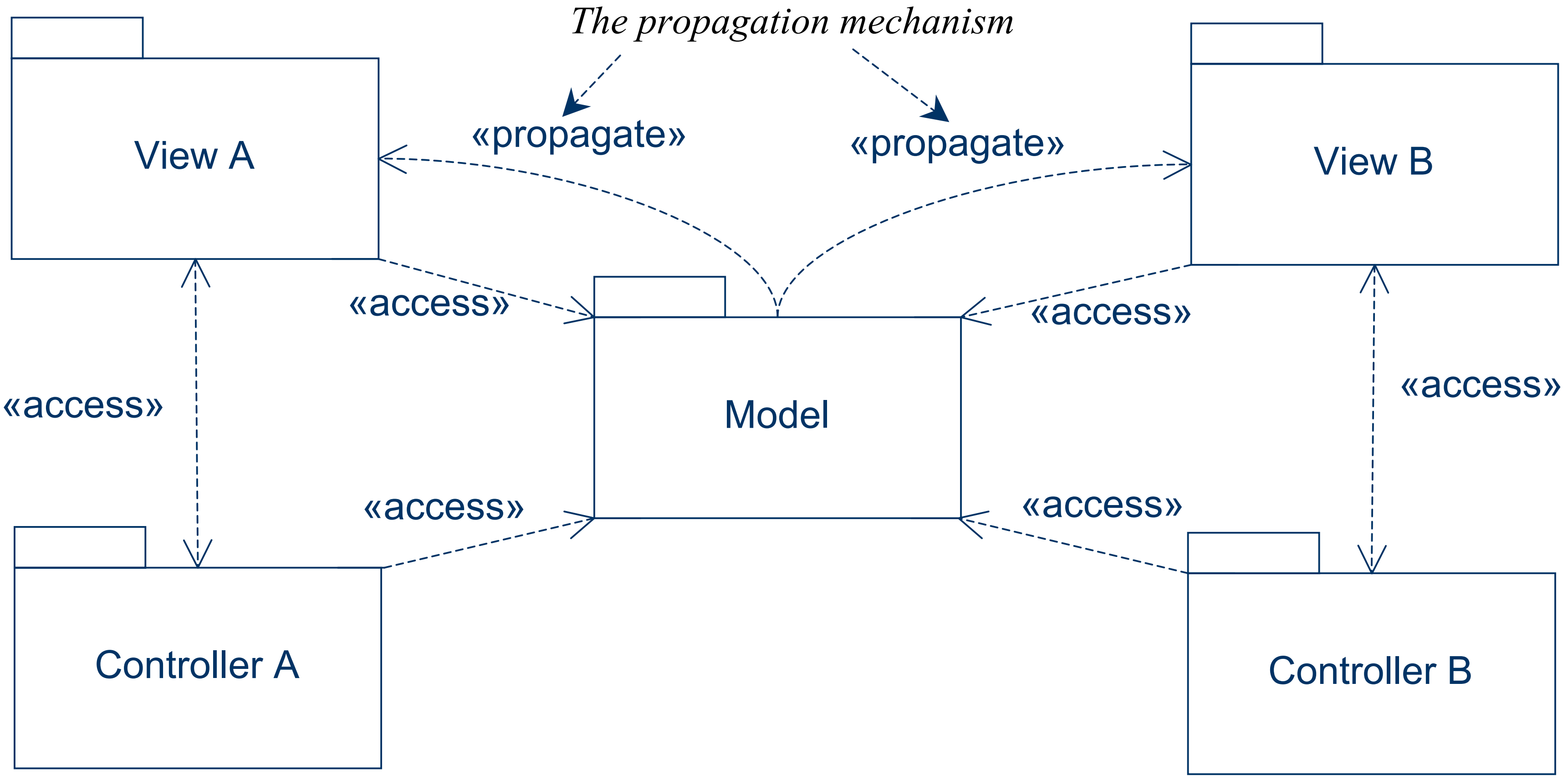
Software Development Patterns (3)

- Different kinds of patterns
  - Analysis patterns: groups of concepts useful in modelling requirements
    - Example: Accountability
  - Architectural patterns: describe the structure and relationships of major components of a software system
    - Example: Model-View-Controller
  - Design patterns: describe the structure and interaction of smaller software components
    - Example: Singleton
  - Idioms: patterns that are related to constructs in a specific programming language
    - Example: Counted pointer in C++

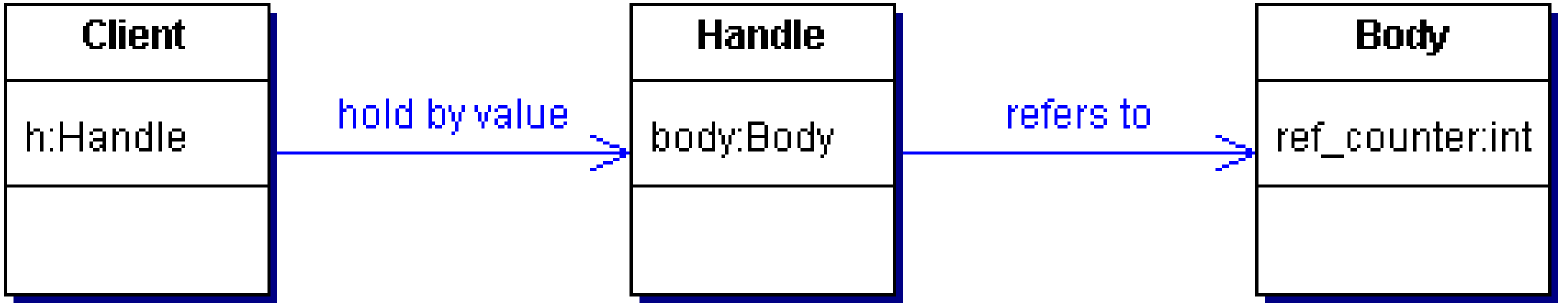
Software Development Patterns (4)



Software Development Patterns (5)



Software Development Patterns (6)



Software Development Patterns (7)

- More patterns
  - Beyond good practice – Anti-patterns: practice that is demonstrably bad including possibly reworked solutions
    - Example: Mushroom Management
      - Isolate developers from users to limit requirement drift
      - Solutions: spiral process development model or involvement of domain experts in the development team
  - Beyond software development
    - Architecture – Alexander
    - Organisational patterns
    - Pedagogical patterns

Pattern Templates

- Style and structure of pattern description
  - Name – meaningful
  - Problem – intent
  - Context – preconditions
  - Forces – constraints
  - Solution – static and dynamic relationships among the components
  - Other aspects: an example of use, resulting context, rationale of the chosen solution, related patterns, known uses of the pattern (rule of three), aliases, sample code and implementation details

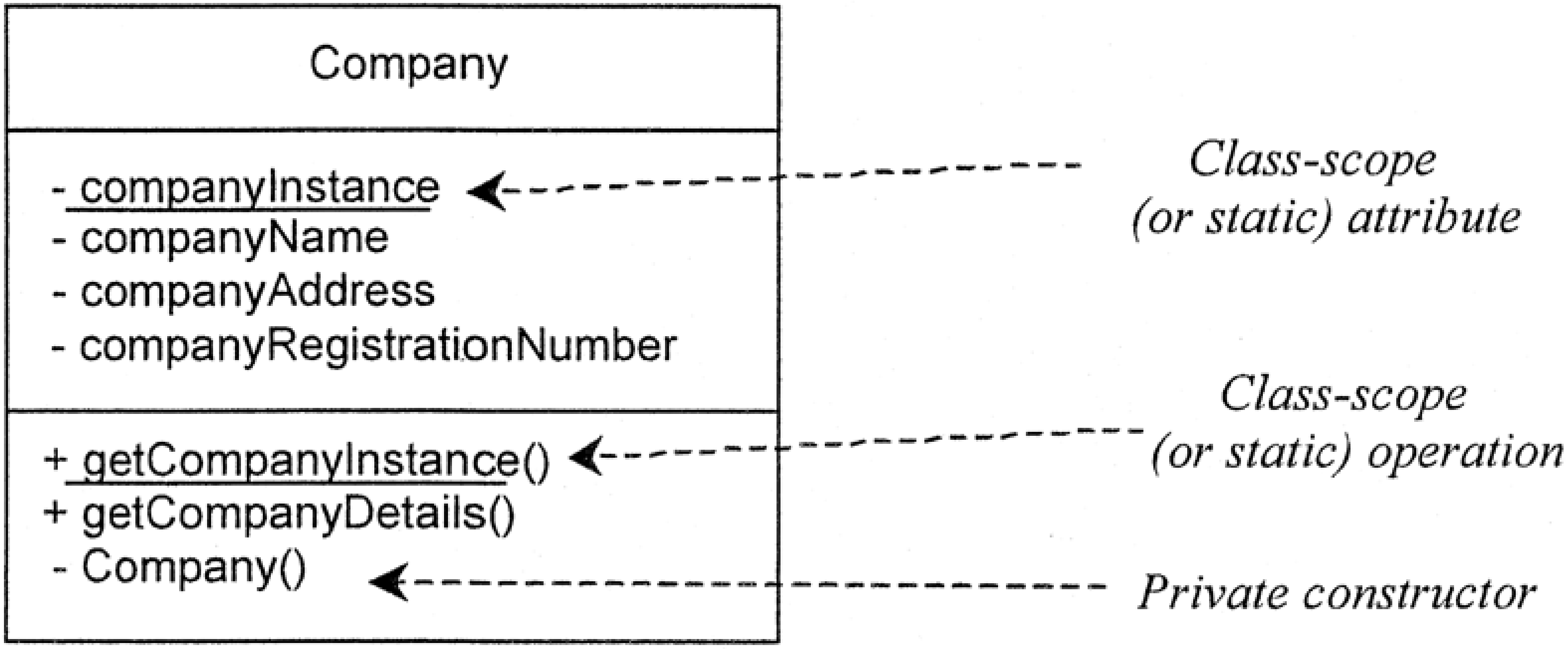
Design Patterns (1)

- Gang of Four catalogue classification
  - Scope: class (compile time, static) or object level (runtime, dynamic)
  - Purpose: creational, structural, behavioural
  - Ease of changes by reducing coupling and maximising cohesion
    - Maintainability – correcting errors
    - Extensibility – inclusion of new features, removal of unwanted features
    - Restructuring – increase flexibility
    - Portability – executing in different environments (OS, hardware, etc.)

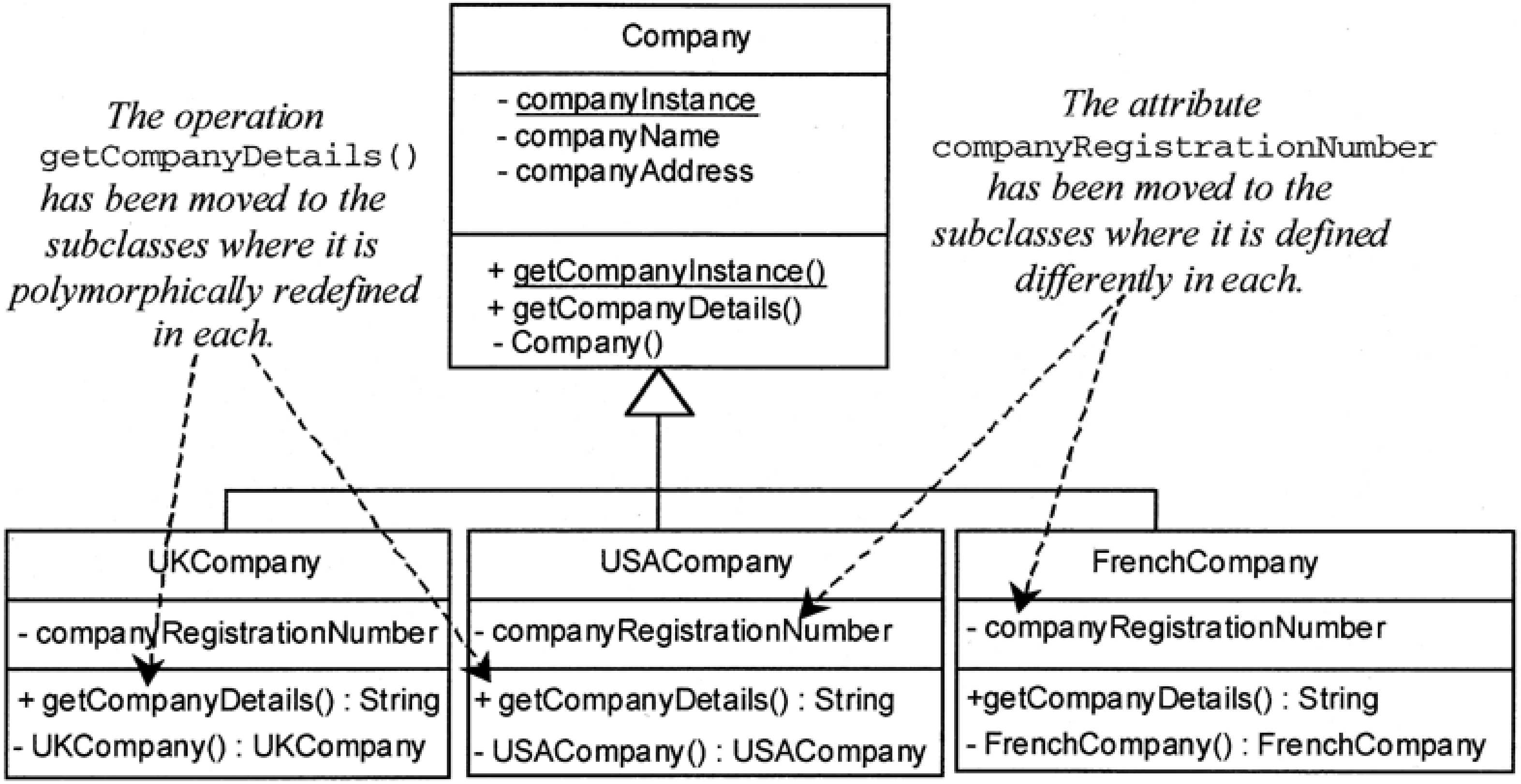
Design Patterns (2)

- Creational patterns (construction of instances)
  - Separate object construction from object use
    - Dynamic or static
- Singleton pattern
  - Ensures only one instance of a class is created!
    - Instead of global data, encapsulate the data into an object!
    - Use static operation getInstance()
    - Private constructor
    - Creation on demand!
  - Extension to accommodate variations

Design Patterns (3)



Design Patterns (4)



Company
- companyInstance
- companyName
- companyAddress
+ getCompanyInstance()
+ getCompanyDetails()
- Company()
UKCompany
- companyRegistrationNumber
+ getCompanyDetails() : String
- UKCompany() : UKCompany
USACompany
- companyRegistrationNumber
+ getCompanyDetails() : String
- USACompany() : USACompany
FrenchCompany
- companyRegistrationNumber
+getCompanyDetails() : String
- FrenchCompany() : FrenchCompany

The operation `getCompanyDetails()` *has been moved to the subclasses where it is polymorphically redefined in each.*

The attribute `companyRegistrationNumber` *has been moved to the subclasses where it is defined differently in each.*

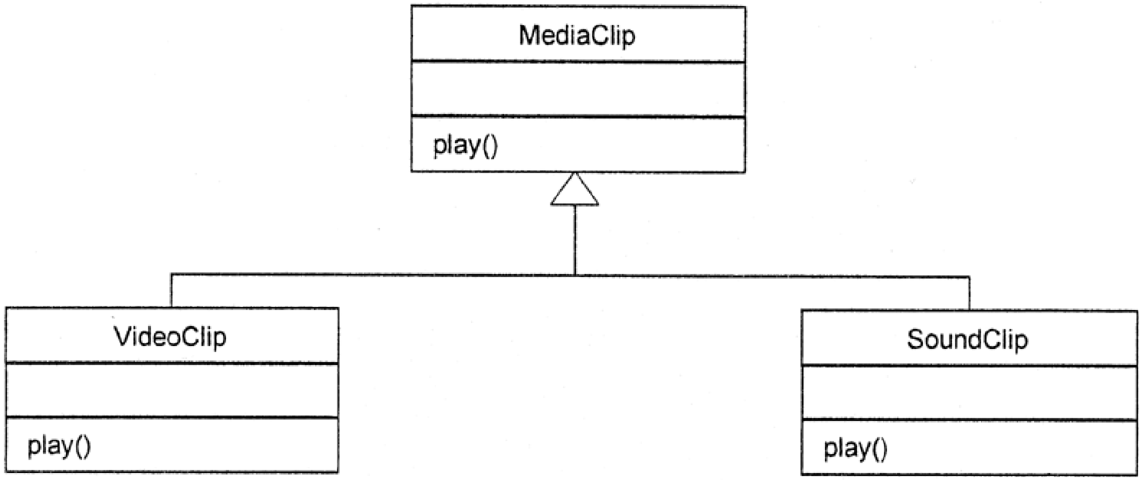
Design Patterns (5)

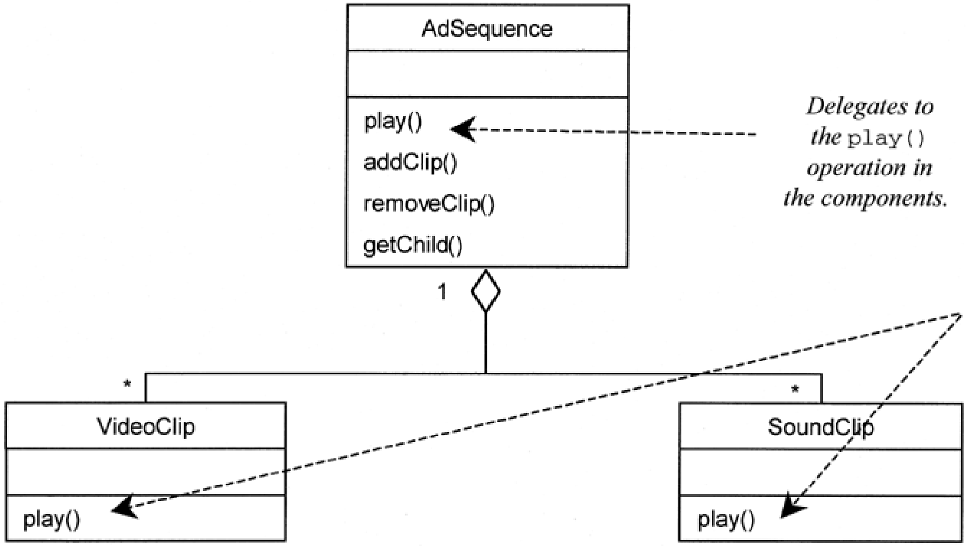
- Advantages
  - Controlled access to the sole instance
  - No global variables
  - The Singleton class may be subclassed
  - A variation can create a specified number of instances
- Disadvantages
  - Pattern introduces additional message passing
  - Limits the application flexibility
  - Developers are tempted to use even when inappropriate

Design Patterns (6)

- Structural patterns (organisation of classes and objects)
  - Inheritance, aggregation, composition
- Composite pattern
  - Represent whole-part hierarchies so that both whole and part objects offer the same interface to client objects
  - Same interface suggests same inheritance hierarchy – polymorphic definition of operations

Design Patterns (7)

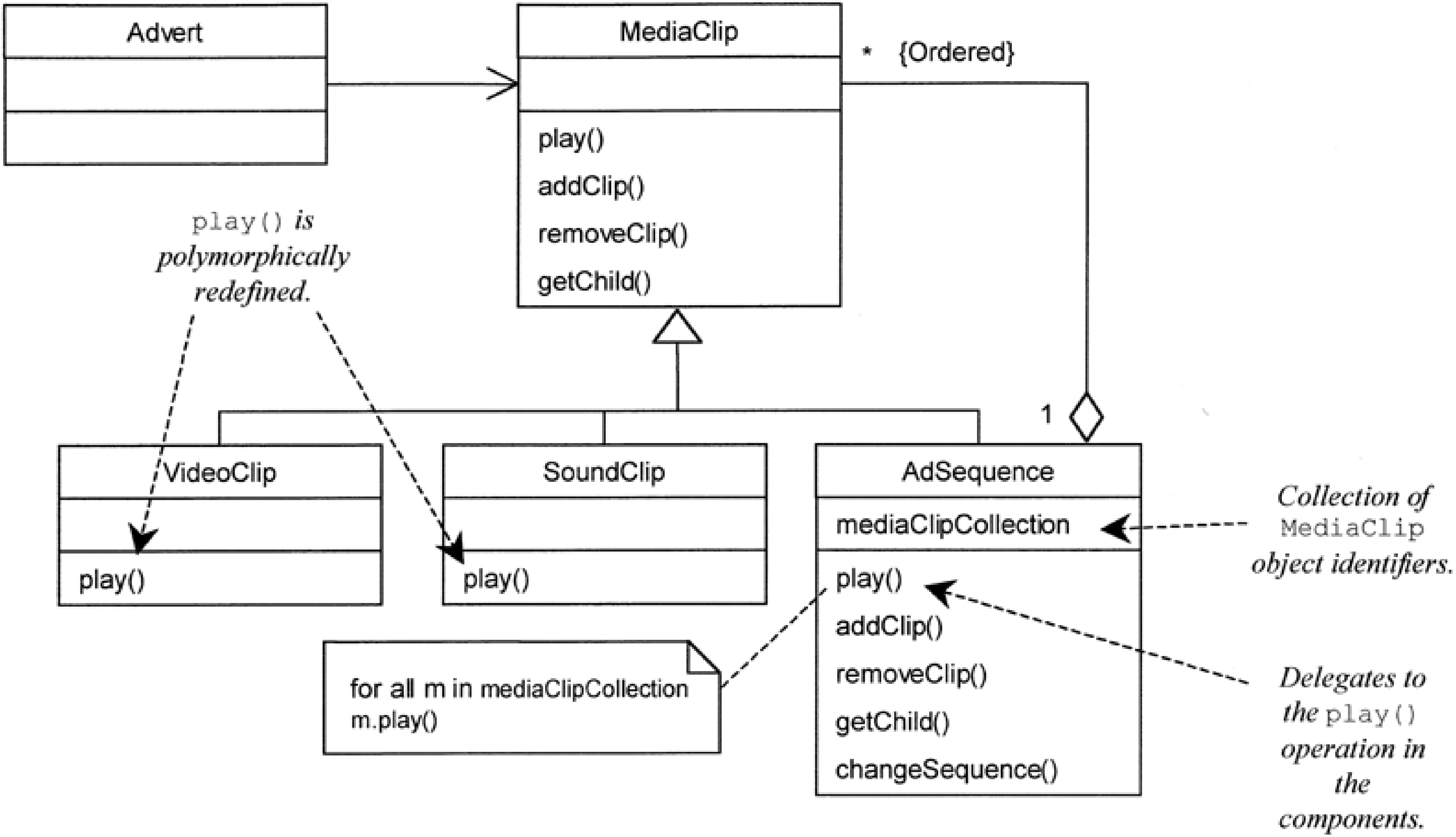




Delegates to the `play()` *operation in the components.*

`play()` *is polymorphically redefined.*

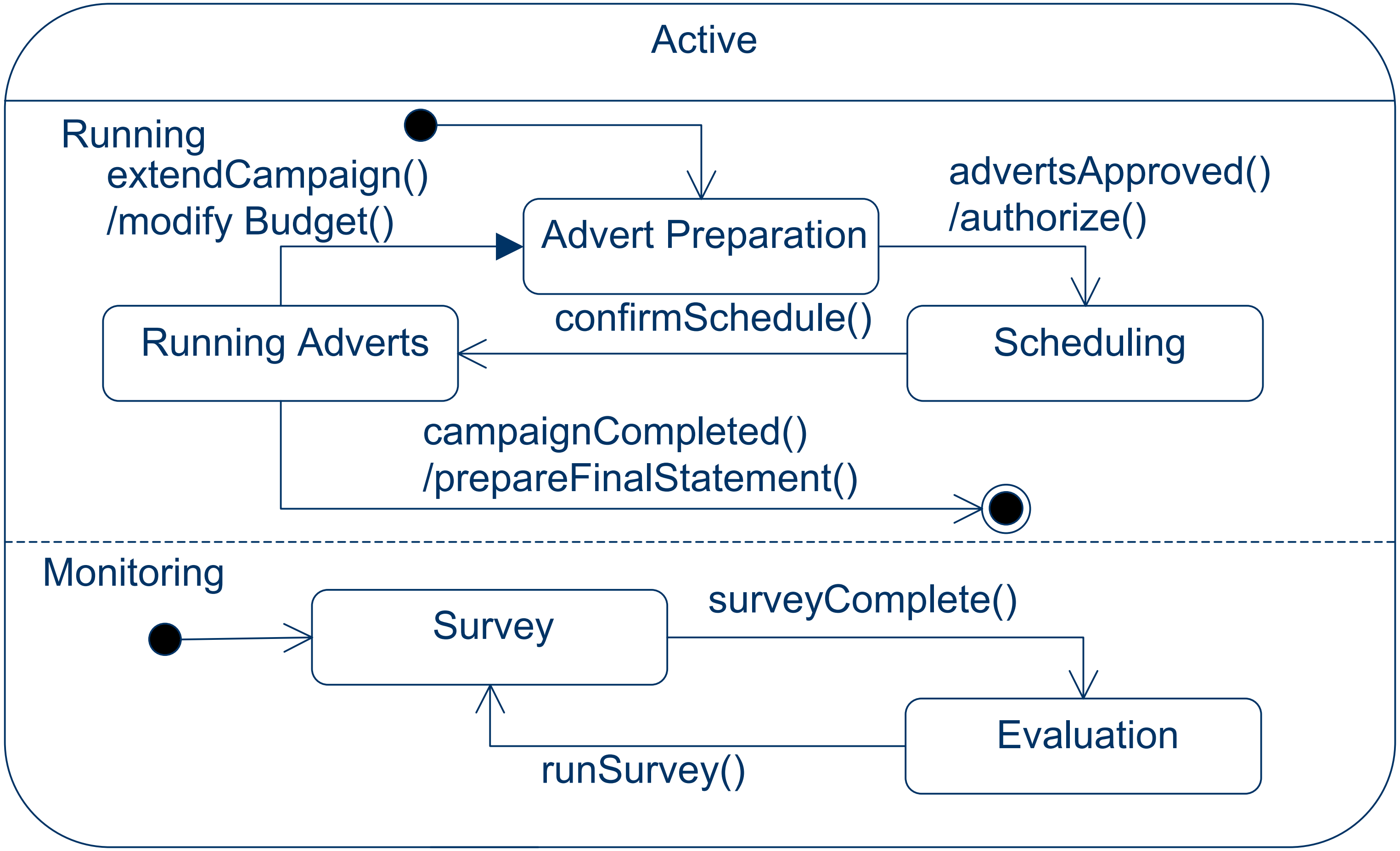
Design Patterns (8)



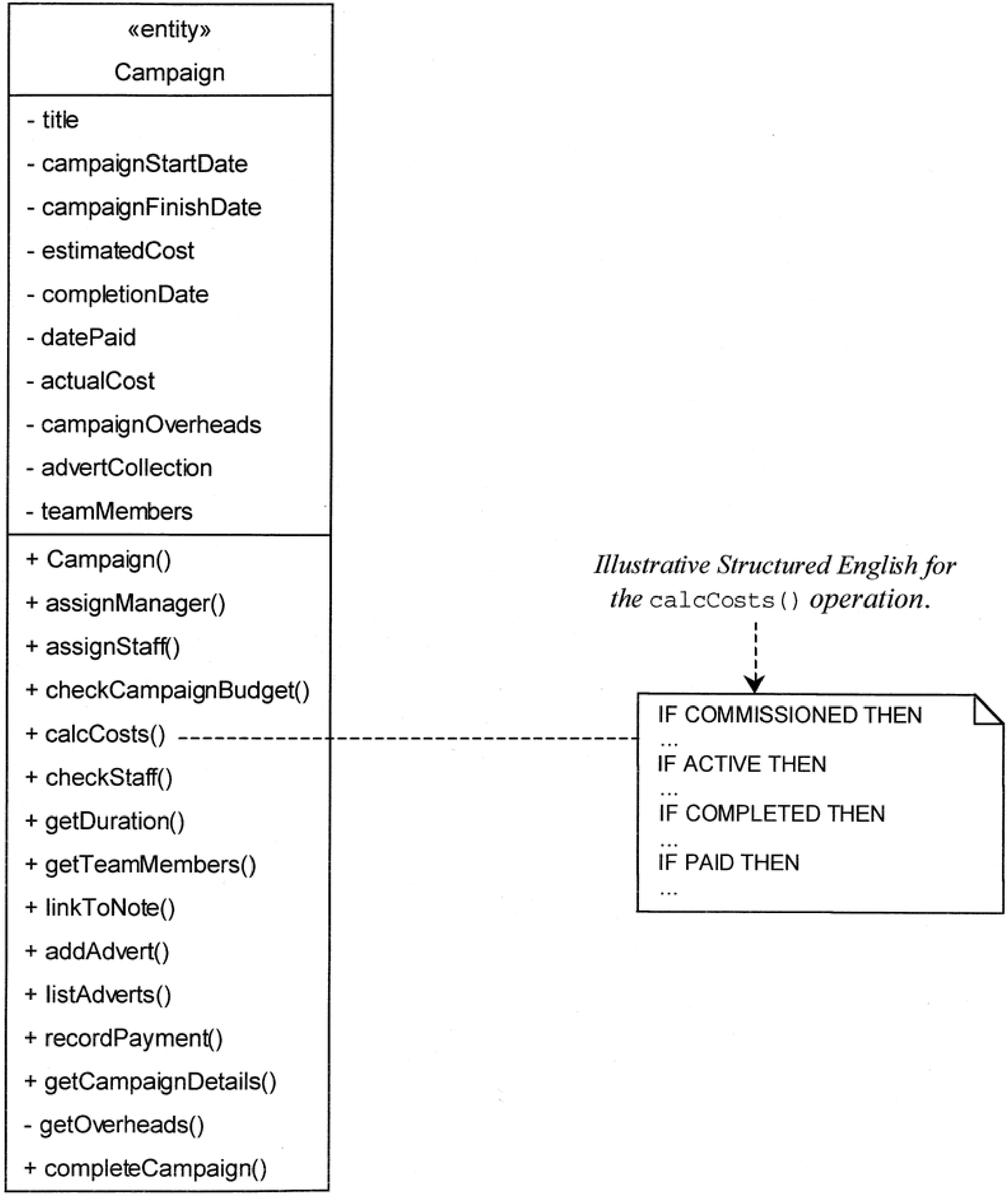
Design Patterns (9)

- Behavioural patterns (problems of assigning responsibilities to class and designing algorithms)
  - Inheritance structures to spread behaviour
  - Aggregation structures to build complex behaviour
- State pattern
  - Objects exhibit different behaviour when their internal stage changes appearing as if the change class at run-time
  - Complex behaviour is broken down into simpler operation which are allocated to different objects one for each state, and the original object delegates responsibility to the appropriate state object
  - State transition responsibility either on context or shared

Design Patterns (10)



(11)



«entity»
Campaign

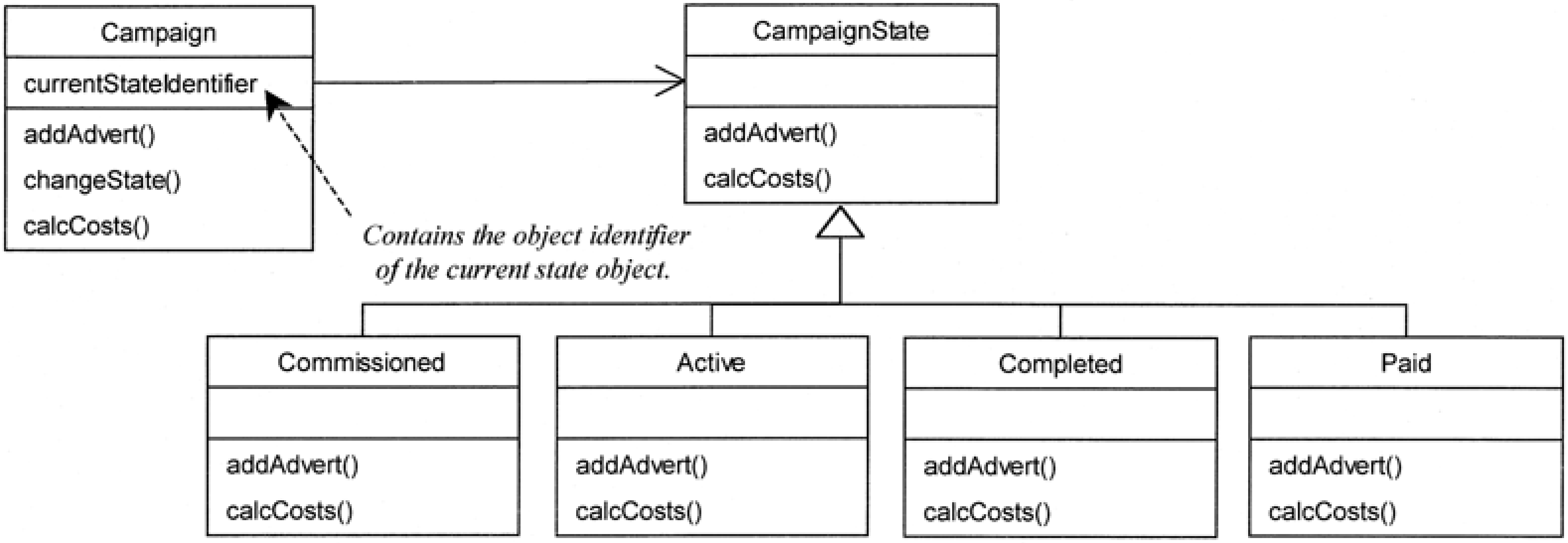
- title
- campaignStartDate
- campaignFinishDate
- estimatedCost
- completionDate
- datePaid
- actualCost
- campaignOverheads
- advertCollection
- teamMembers

+ Campaign()
+ assignManager()
+ assignStaff()
+ checkCampaignBudget()
+ calcCosts()
+ checkStaff()
+ getDuration()
+ getTeamMembers()
+ linkToNote()
+ addAdvert()
+ listAdverts()
+ recordPayment()
+ getCampaignDetails()
- getOverheads()
+ completeCampaign()

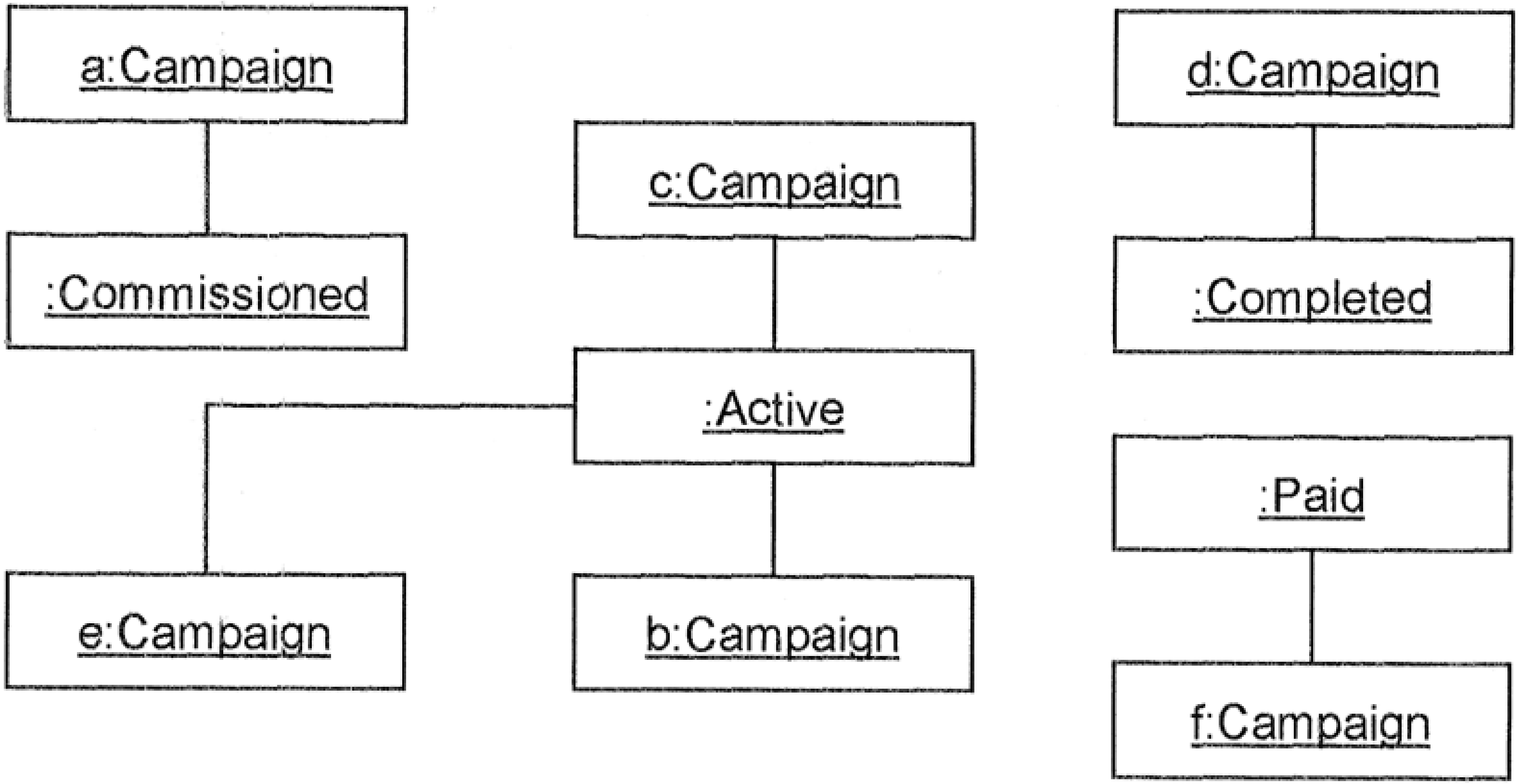
Illustrative Structured English for the `calcCosts()` *operation.*

```
IF COMMISSIONED THEN
...
IF ACTIVE THEN
...
IF COMPLETED THEN
...
IF PAID THEN
...
```

Design Patterns (12)



Design Patterns (13)



Design Patterns (14)

- Advantages
  - State behaviour is localised
  - State transitions are made explicit
  - State object can be shared among Context objects
- Disadvantages
  - If state objects cannot be share among Context objects there is an explosion of objects
  - Processing overheads for the creation and deletion of state objects
  - Processing overhead from the additional message

How to use design patterns

- Patterns require training
- Issues to consider
  - Is there a pattern for the problem?
  - Does the pattern trigger a more acceptable solution?
  - Is there a simpler solution?
  - Is the context of the pattern consistent with that of the problem?
  - Are the consequences of using the pattern acceptable?
  - Are any constraints of the software environment in conflict with the use of the pattern?
- Pattern application procedure
  - Read the pattern to get a complete overview
  - Study the Structure, Participants and Collaborations in detail
  - Examine the Sample Code
  - Choose names for the participants that are meaningful for the application
  - Define the classes
  - Implement operations that perform the responsibilities and collaboration in the pattern
- Pattern mining (pattern writer’s workshop)

Benefits and Dangers of using patterns

- Reuse of generic solutions
  - Reusable design elements
- A vocabulary for discussing the problem domain
- Patterns can limit creativity
- Patterns may lead to over-design
- Introduction of patterns has cost for the organisation
- Introduction of patterns requires a reuse culture
  - More acceptable than code reuse
- Use with care and planning