Creational Pattern :

abstract factory :

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

Kit

motif presentation manager, window,scrollbar

abstract factory, concrete factory, abstract product A and product B

It isolates concrete classes
It makes exchanging product families easy
It promotes consistency among products
Supporting new kinds of products is difficult

Factories as singletons
Creating the products
Defining extensible factories

builder :

Separate the construction of a complex object from its representation so that the same construction process can create different representations.

RTF reader, text converter, Asciiconverter, texconverter, textwidget converter.

Code, director, builder, concrete builder, product.

Client, director, concrete builder,

It lets you vary a product's internal representation.
It isolates code for construction and representation.
It gives you finer control over the construction process.

Assembly and construction interface.
Why no abstract class for products?
Empty methods as default in Builder

factory method :

Define an interface for creating an object, but let subclasses decide which class to instantiate.
Factory Method lets a class defer instantiation to subclasses.

Virtual cluster

code, myapplication, application, document, Mydocument.

Code, concretecreater, creater, concreteproduct, product.

Provides hooks for subclasses.
Connects parallel class hierarchies.

Two major varieties.
Parameterized factory methods
Language-specific variants and issues

prototype :

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

Music score

Client, prototype, concrete prototype1 and concrete prototype2

Adding and removing products at run-time.
Specifying new objects by varying values.
Specifying new objects by varying structure.

Using a prototype manager.
Implementing the Clone operation.
Initializing clones.

singleton

Ensure a class only has one instance, and provide a global point of access to it.

Printer, document

singleton

Controlled access to sole instance
Reduced name space
Permits refinement of operations and representation
Permits a variable number of instances
More flexible than class operations

Ensuring a unique instance
Subclassing the Singleton class

structural pattern :

adapter :

Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

Wrapper

drawing window, rectangle, polygon

client, target, adapter, adaptee

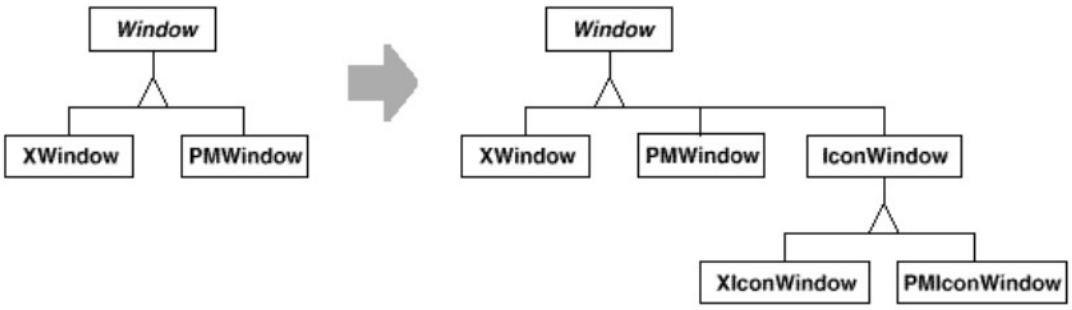
How much adapting does Adapter do
Pluggable adapters
Using two-way adapters to provide transparency

Implementing class adapters in C++
Pluggable adapters

Bridge:

Decouple an abstraction from its implementation so that the two can vary independently.

Handle, body



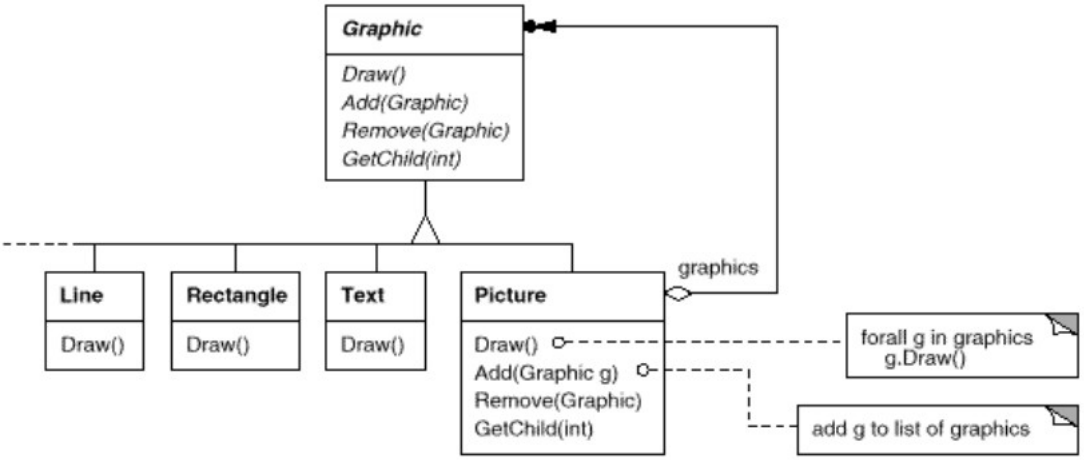
abstract, implementer, concreteimplementer A and concreteimplementer B

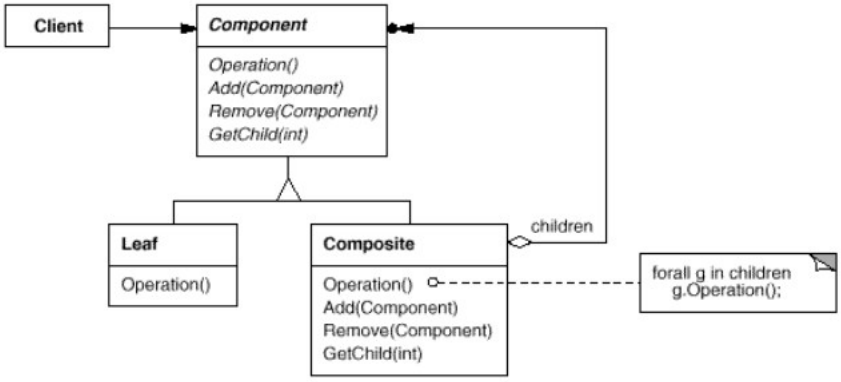
Decoupling interface and implementation.
Improved extensibility
Hiding implementation details from clients

Only one Implementor.
Creating the right Implementor object.
Sharing implementors.

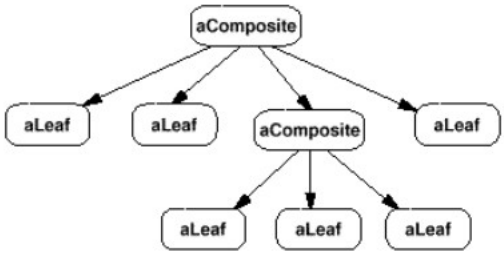
Composite :

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.





A typical Composite object structure might look like this:



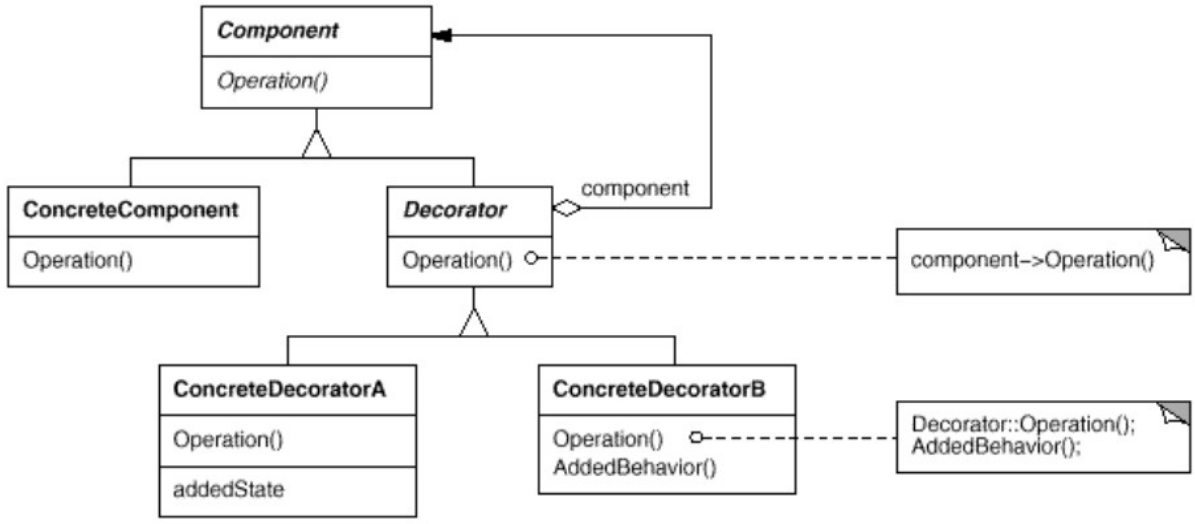
Explicit parent references
Sharing components
Maximizing the Component interface
Declaring the child management operations
Should Component implement a list of Components
Child ordering
Caching to improve performance

Decorator :

Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

Wrapper

scrollbar, border decorator, textview



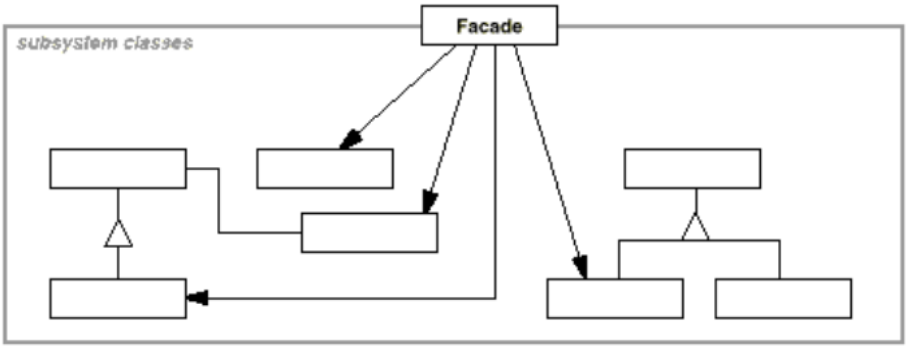
More flexibility than static inheritance
Avoids feature-laden classes high up in the hierarchy
A decorator and its component aren't identical
Lots of little objects

Interface conformance
Omitting the abstract Decorator class
Keeping Component classes lightweight
Changing the skin of an object versus changing its guts

Facade :

Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.

.

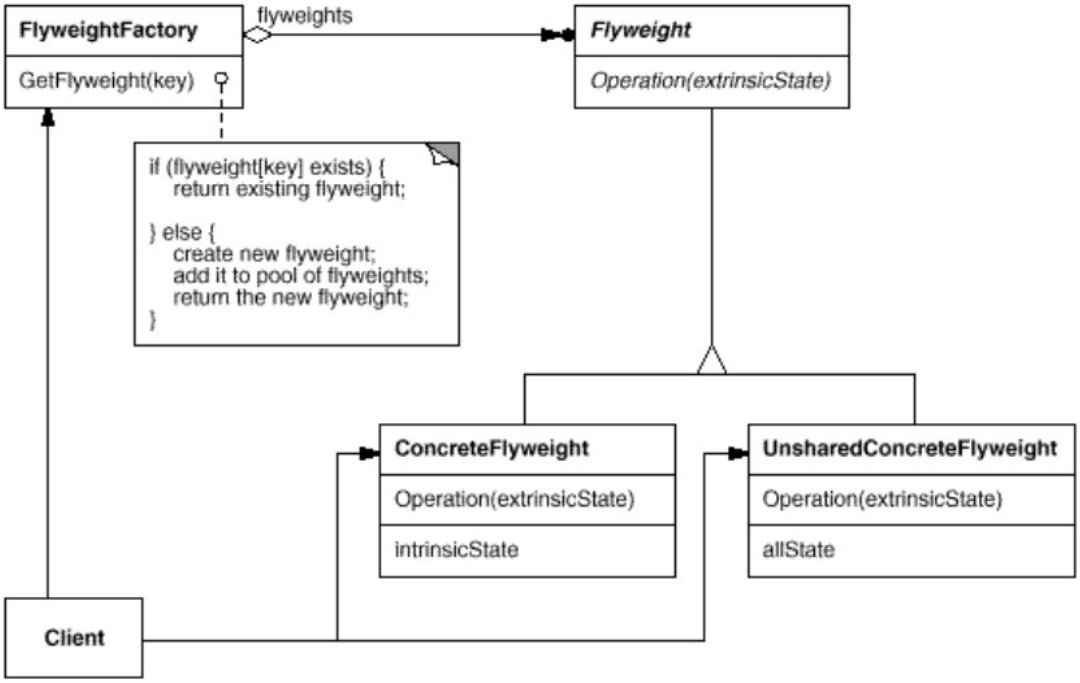


Reducing client-subsystem coupling.
Public versus private subsystem classes

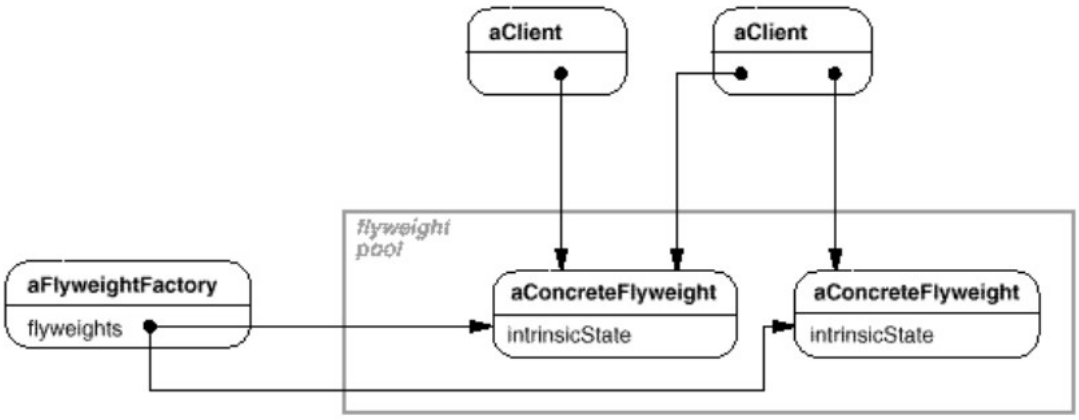
Flyweight :

Use sharing to support large numbers of fine-grained objects efficiently.

text formatting and editing



.

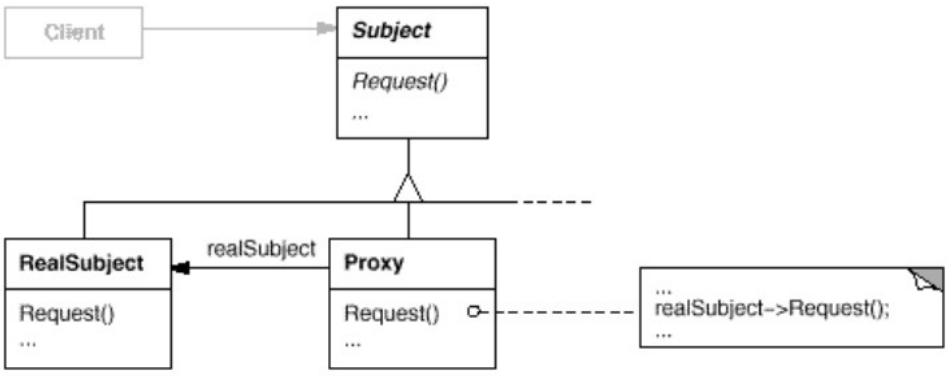


.
Removing extrinsic state
Managing shared objects

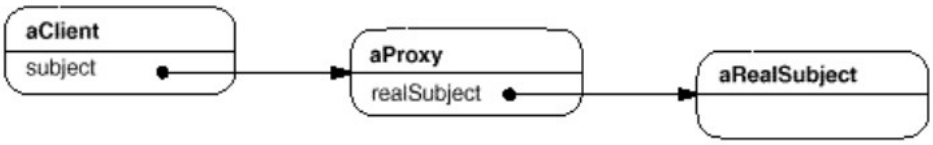
Proxy :

Provide a surrogate or placeholder for another object to control access to it

surrogate



Here's a possible object diagram of a proxy structure at run-time:



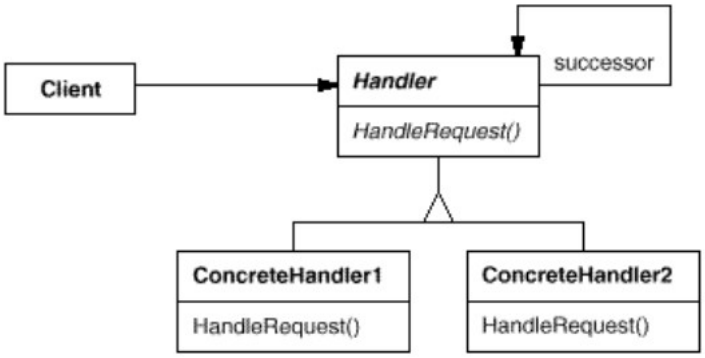
.
Overloading the member access operator in C++
Using doesNotUnderstand in Smalltalk
Proxy doesn't always have to know the type of real subject

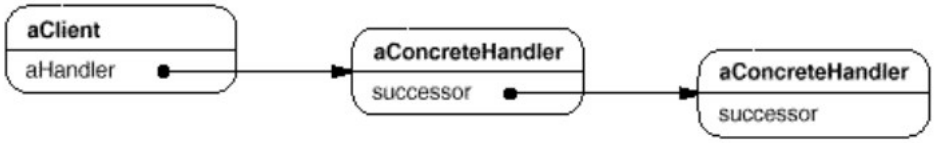
Behavioural Patterns :

Chain of Responsibility :

Avoid coupling the sender of a request to its receiver by giving morethan one object a chance to handle the request. Chain the receivingobjects and pass the request along the chain until an objecthandles it.

.





.

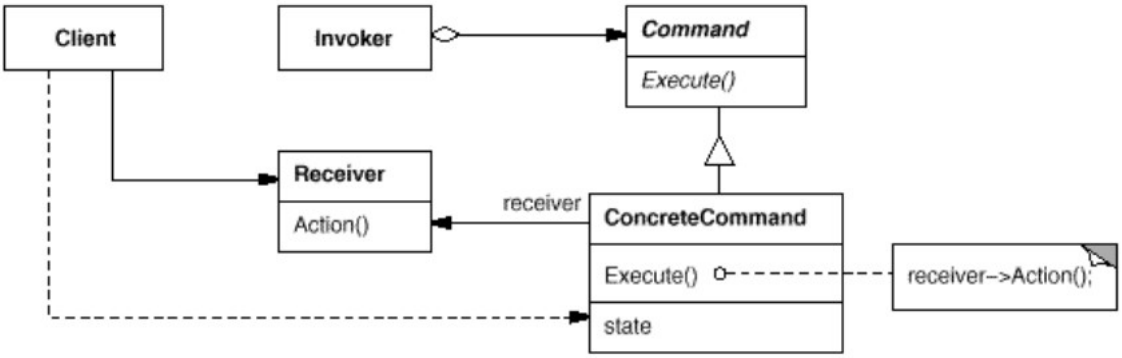
Reduced coupling
Added flexibility in assigning responsibilities to objects.
Receipt isn't guaranteed

Implementing the successor chain
Connecting successors
Representing requests
Automatic forwarding in Smalltalk

Command :

Encapsulate a request as an object, thereby letting you parameterizeclients with different requests, queue or log requests, and supportundoable operations.

Action, Transaction

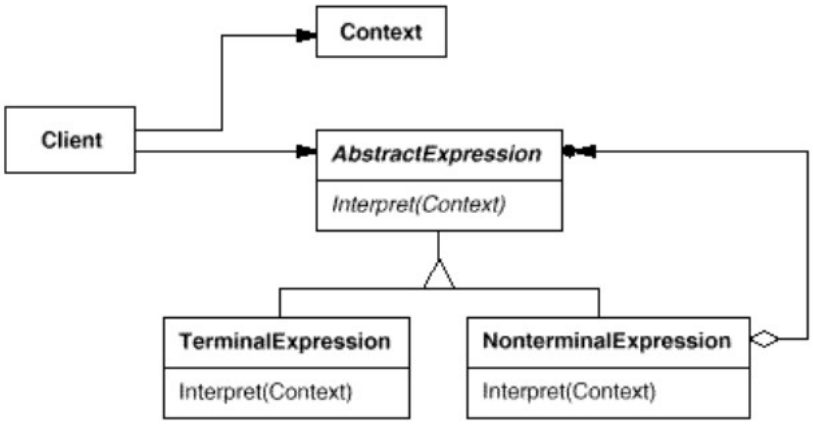


.
How intelligent should a command be
Supporting undo and redo
Avoiding error accumulation in the undo process
Using C++ templates.

Interpreter :

Given a language, define a represention for its grammar along with aninterpreter that uses the representation
to interpret
sentences in
thelanguage.

.

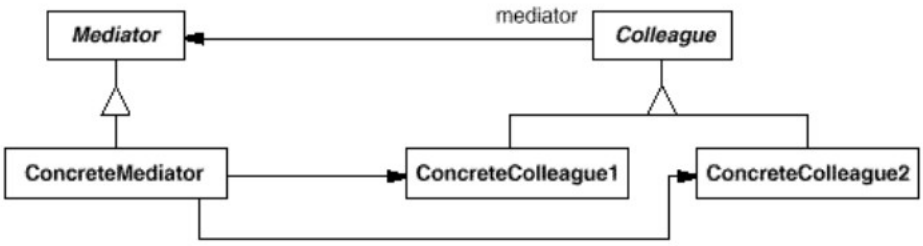


It's easy to change and extend the grammar
Implementing the grammar is easy, too
Complex grammars are hard to maintain
Adding new ways to interpret expressions

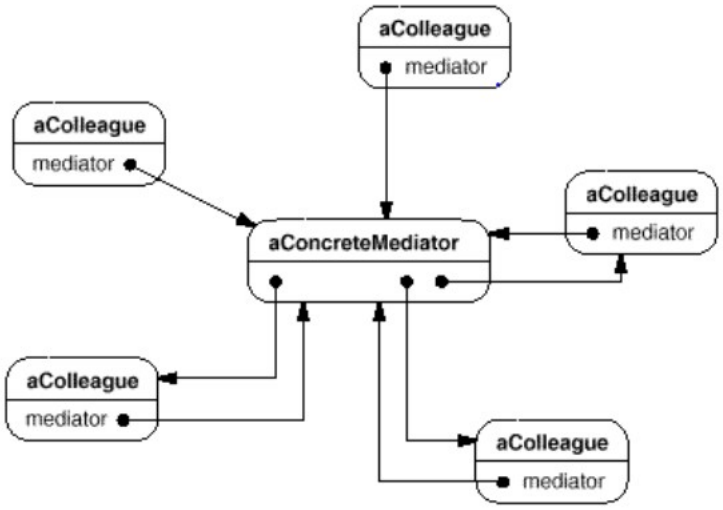
Creating the abstract syntax tree
Defining the Interpret operation
Sharing terminal symbols with the Flyweight pattern.

Mediator :

Define an object that encapsulates how a set of objects interact.Mediator promotes loose coupling by keeping objects from referring toeach other explicitly, and it lets you vary their interactionindependently.



A typical object structure might look like this:



.
It limits subclassing
It decouples colleagues
It simplifies object protocols
It abstracts how objects cooperate
It centralizes control

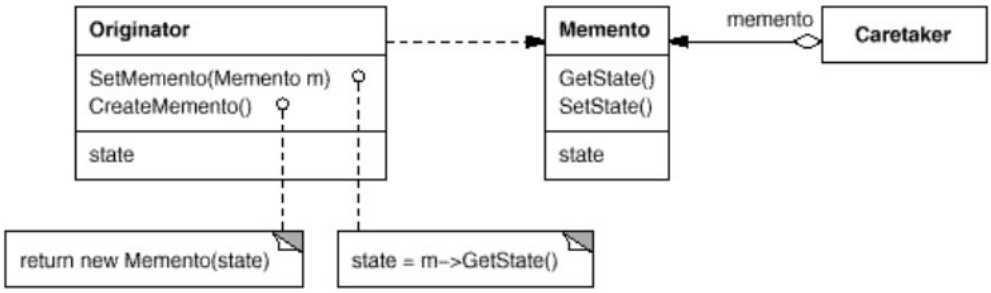
Omitting the abstract Mediator class

Colleague-Mediator communication

Memento :

Without violating encapsulation, capture and externalize an object'sinternal state so that the object can be restored to this state later.

Token



Preserving encapsulation boundaries
It simplifies Originator
Using mementos might be expensive
Defining narrow and wide interfaces
Hidden costs in caring for mementos

Language support
Storing incremental changes