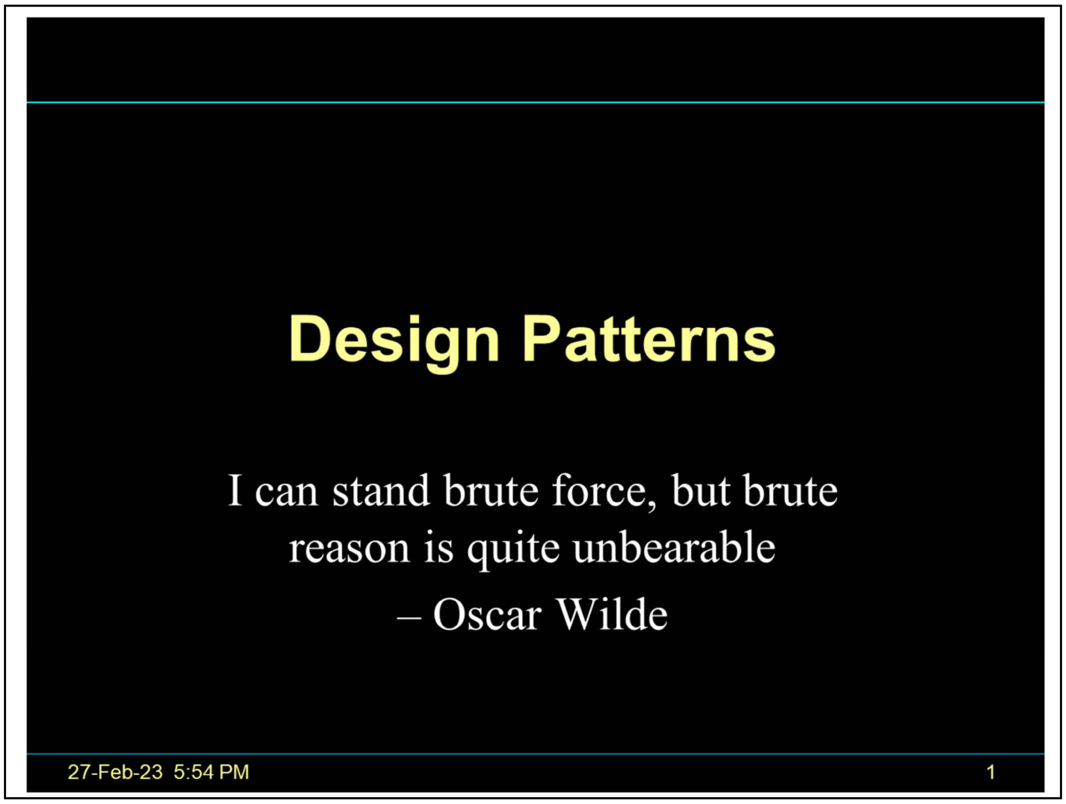
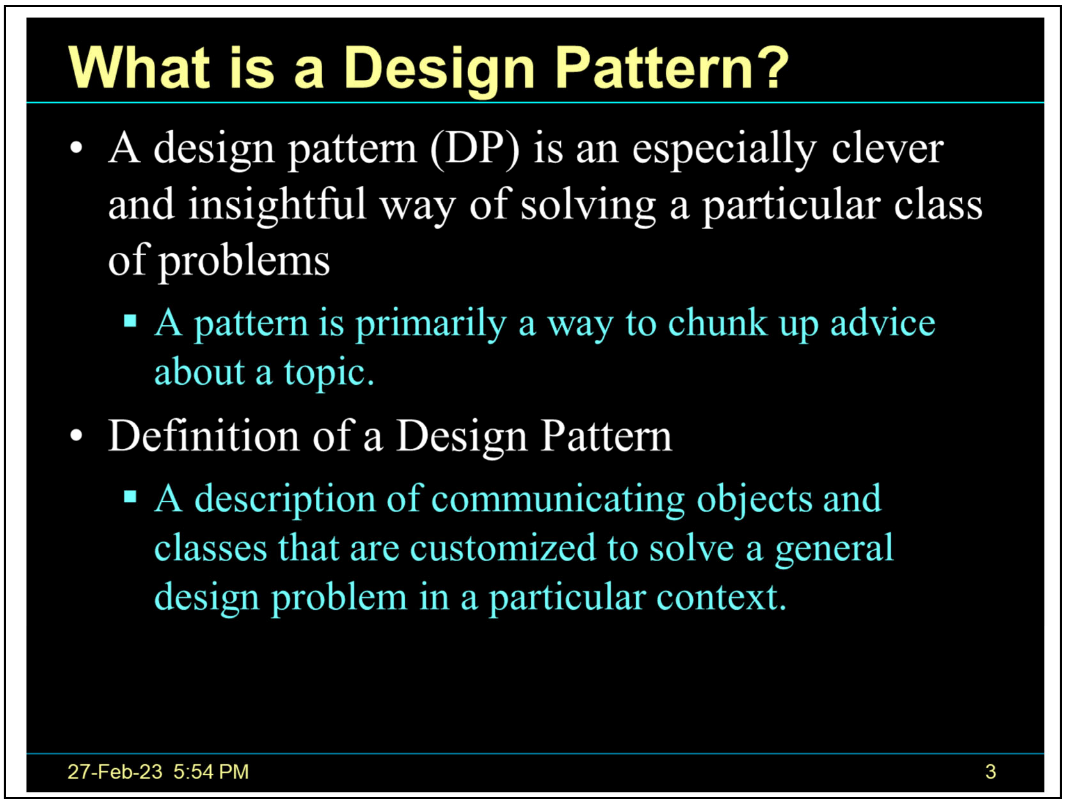


Design Patterns

I can stand brute force, but brute reason is quite unbearable
– Oscar Wilde

Introduction

Experience is a hard teacher, because she give the test first, the lesson afterwards. – Chinese Proverb



What is a Design Pattern?

- A design pattern (DP) is an especially clever and insightful way of solving a particular class of problems
  - A pattern is primarily a way to chunk up advice about a topic.
- Definition of a Design Pattern
  - A description of communicating objects and classes that are customized to solve a general design problem in a particular context.

27-Feb-23 5:54 PM 3

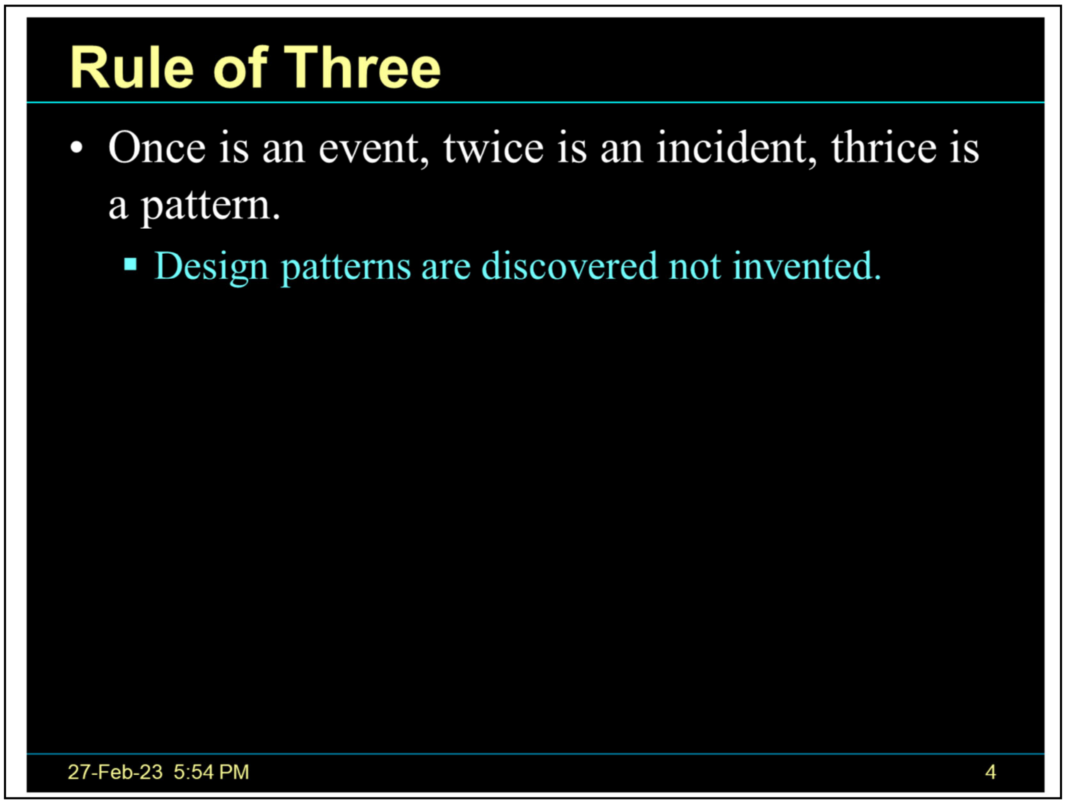
DPs creates a design that is more reusable, flexible and maintainable.

We start thinking in a higher level of abstraction.

- DPs can document the architecture of a system and enhance its understanding.
- We can communicate better with fewer words.

Improvement in productivity.

- Someone has already solved a similar problem.
- Design patterns help us learn from others' successes instead of our own failures



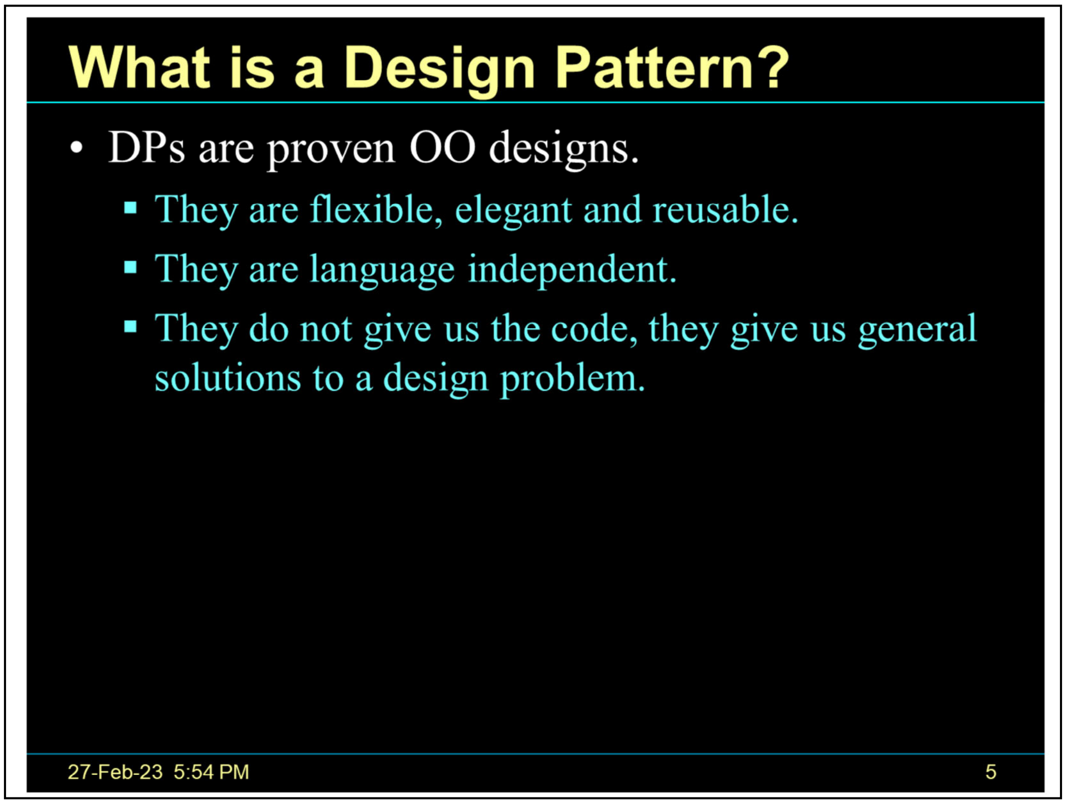
Rule of Three

- Once is an event, twice is an incident, thrice is a pattern.
  - Design patterns are discovered not invented.

27-Feb-23 5:54 PM 4

Design patterns are recurring solutions to design problems we see over and over again.

Design patterns emerge from practice, not from theory.



What is a Design Pattern?

- DPs are proven OO designs.
  - They are flexible, elegant and reusable.
  - They are language independent.
  - They do not give us the code, they give us general solutions to a design problem.

27-Feb-23 5:54 PM 5

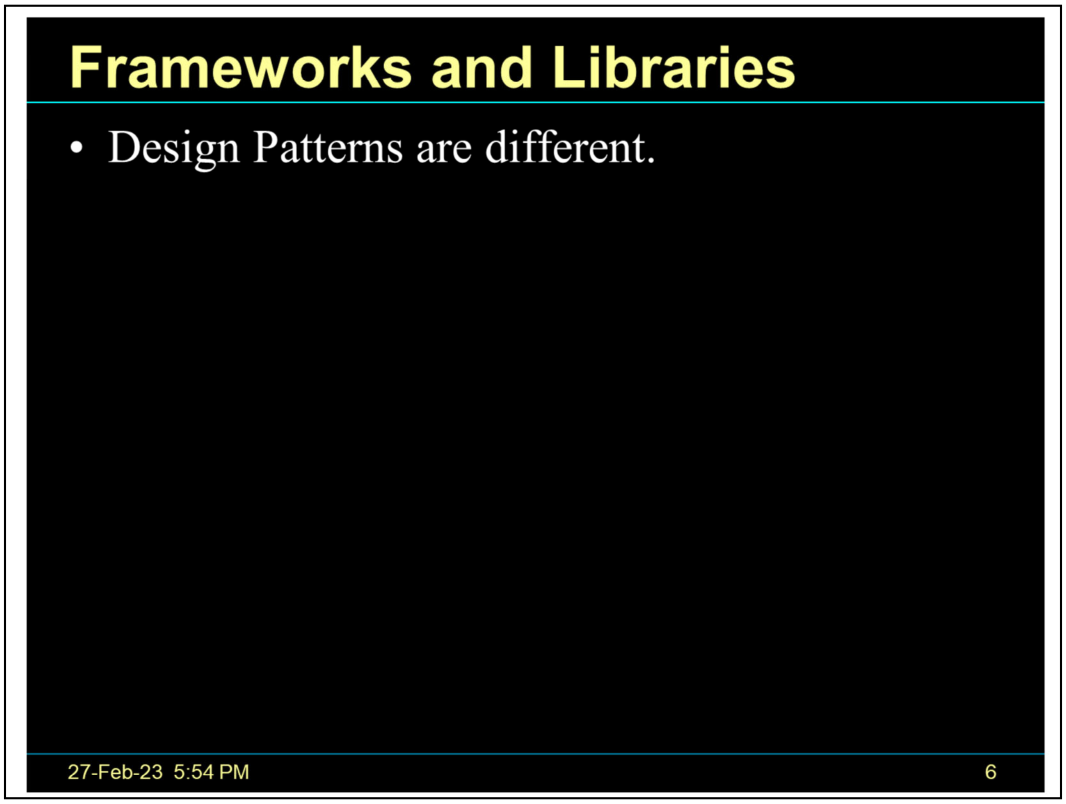
OOP gives us the <u>tools</u> to cope with change:

Encapsulation

Inheritance

Polymorphism

Having tools is good. Using them well is better. Welcome to Design Patterns.



Frameworks and Libraries

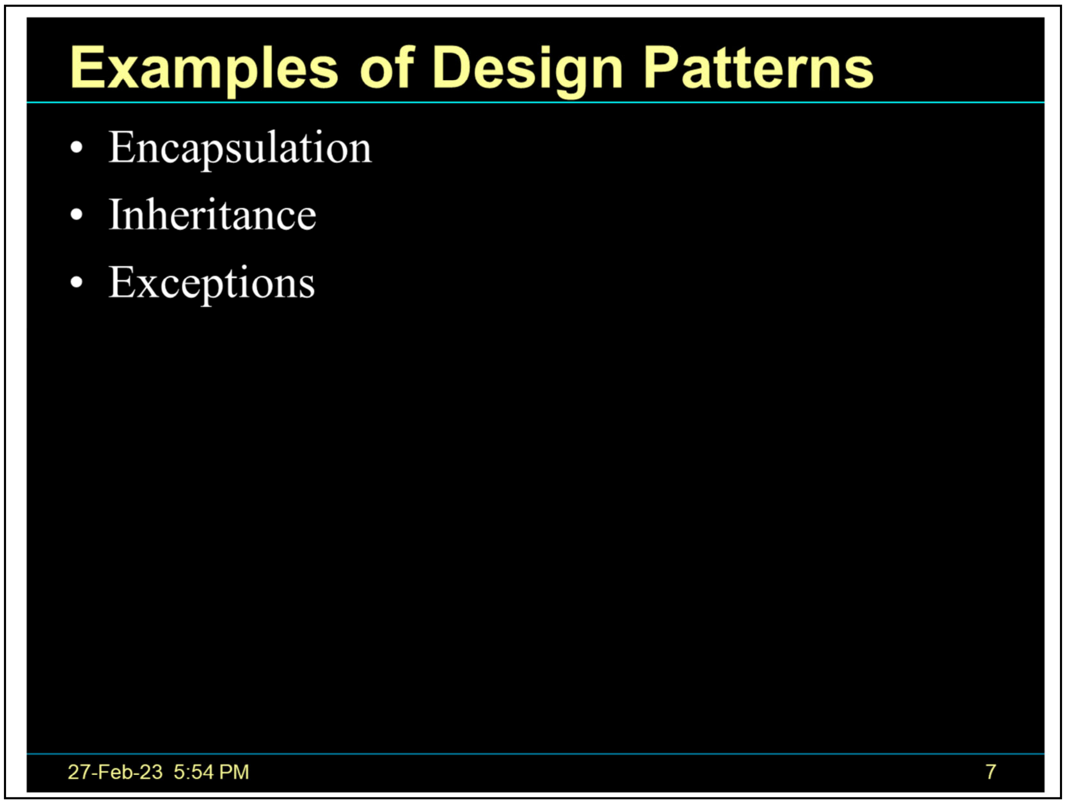
- Design Patterns are different.

27-Feb-23 5:54 PM 6

Framework is a set of co-operating classes, which together make up reusable design, for a specific kind of software. The Framework provides an architectural guidance by structuring the system in abstract classes, as well their responsibilities and interactions. The framework user adapts the framework to his specific needs by deriving concrete classes or by assembling classes provided by the framework.

How are design patterns different from frameworks or libraries?

- Design patterns are more abstract than frameworks or libraries.
- Design patterns are smaller architectural elements than frameworks / libraries.
- Frameworks and libraries usually use design patterns.
  - We generally call the library code. A framework generally calls our code.
- Design patterns focus more on reuse of recurring architectural design themes, while frameworks focus on detailed design and implementation.



Examples of Design Patterns

- Encapsulation
- Inheritance
- Exceptions

27-Feb-23 5:54 PM 7

Encapsulation: Problem description:

Exposed fields can be directly manipulated from outside, leading to violations of the representation invariant or undesirable dependencies that prevent changing the implementation.

Solution: Hide some components, permitting only stylized access to object.

Disadvantages: Indirection may reduce performance.

Sub-Classing: Problem: Similar abstractions have similar members (fields and methods). Repeating these is tedious, error-prone and a maintenance headache.

Solution: Inherit default members from a super class; select the correct implementation via run-time dispatching.

Disadvantages: Code for a class is spread out, potentially reducing understandability. Virtual functions introduce runtime overheads.

Exceptions: Problem: Errors occurring in one part of code should be handled elsewhere. Code should not be cluttered up with error-handling code nor return values preempted by error-codes.

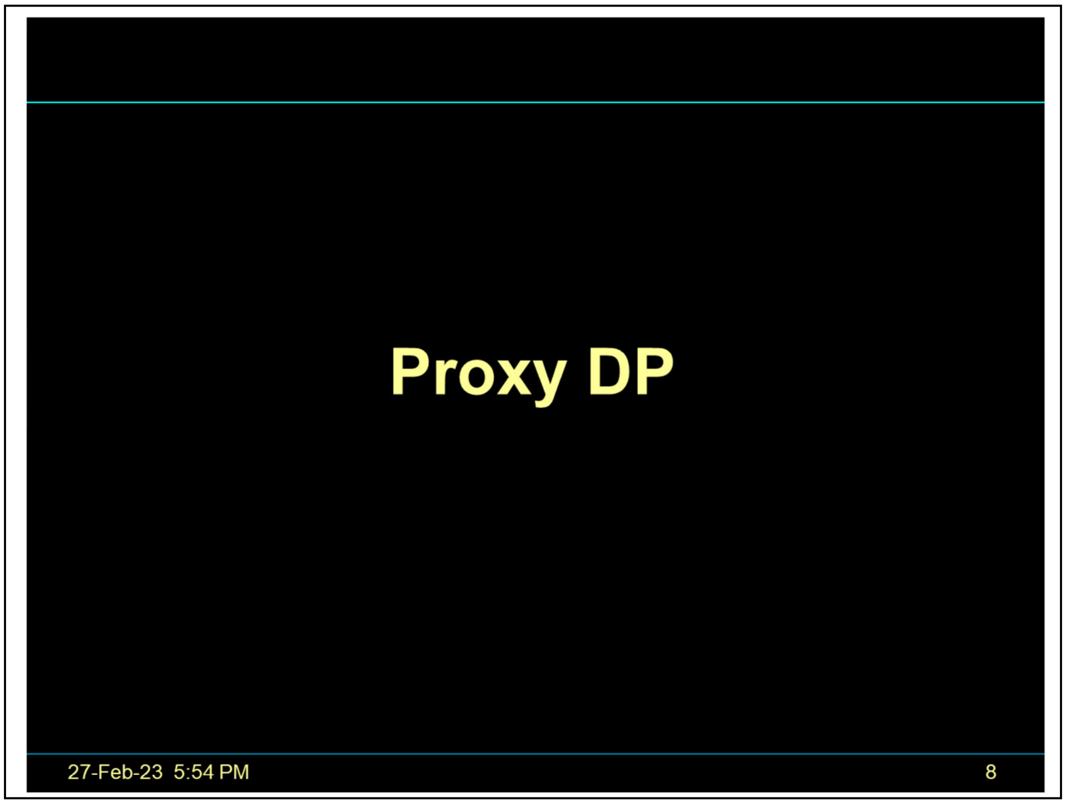
Solution: Introduce language structures for throwing and catching exceptions.

Disadvantages:

Code may still be cluttered (for checked exceptions).

It is hard to know where an exception will be handled.

Programmers may misuse the feature for jumping from location to another. This will be confusing and inefficient.



Proxy DP

27-Feb-23 5:54 PM

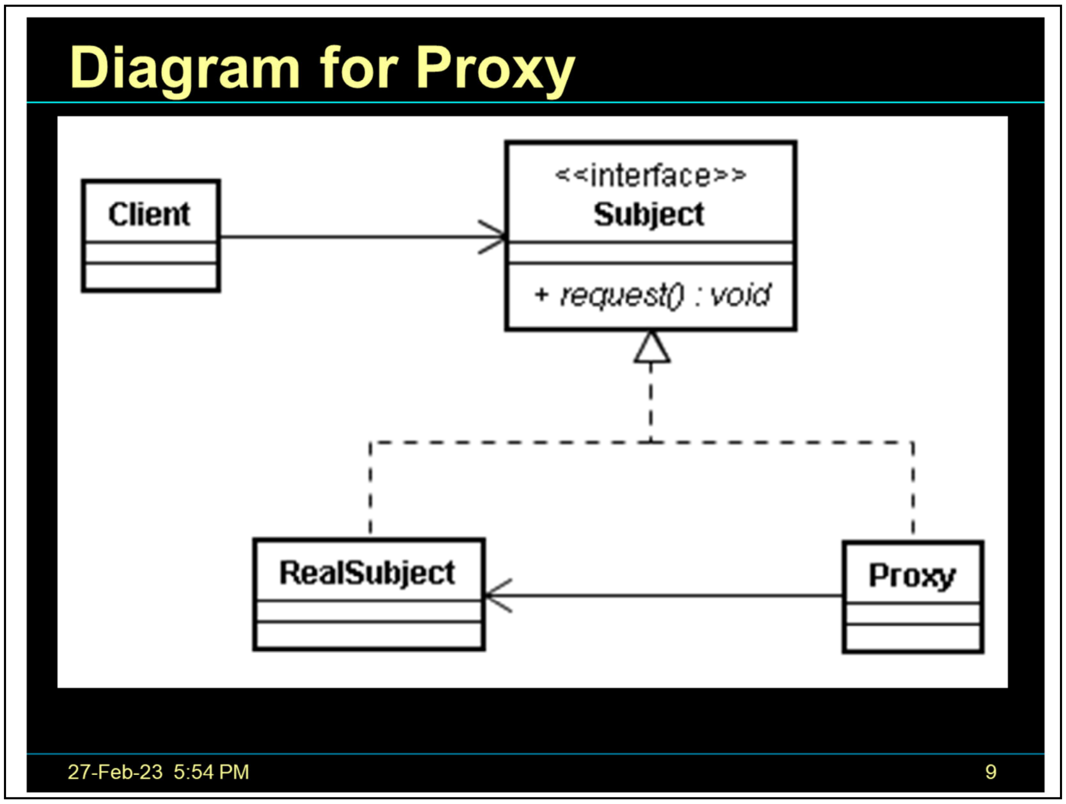
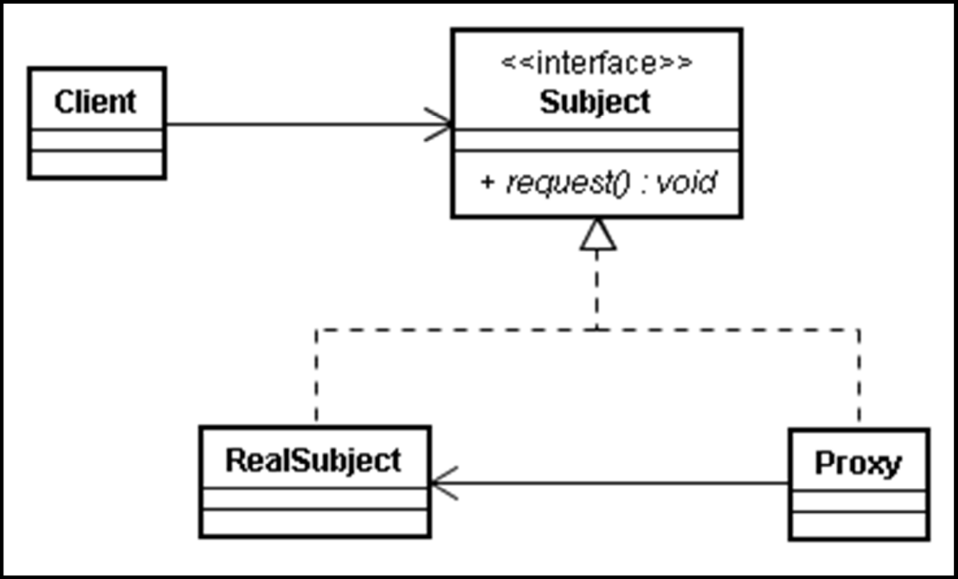


Diagram for Proxy



27-Feb-23 5:54 PM 9

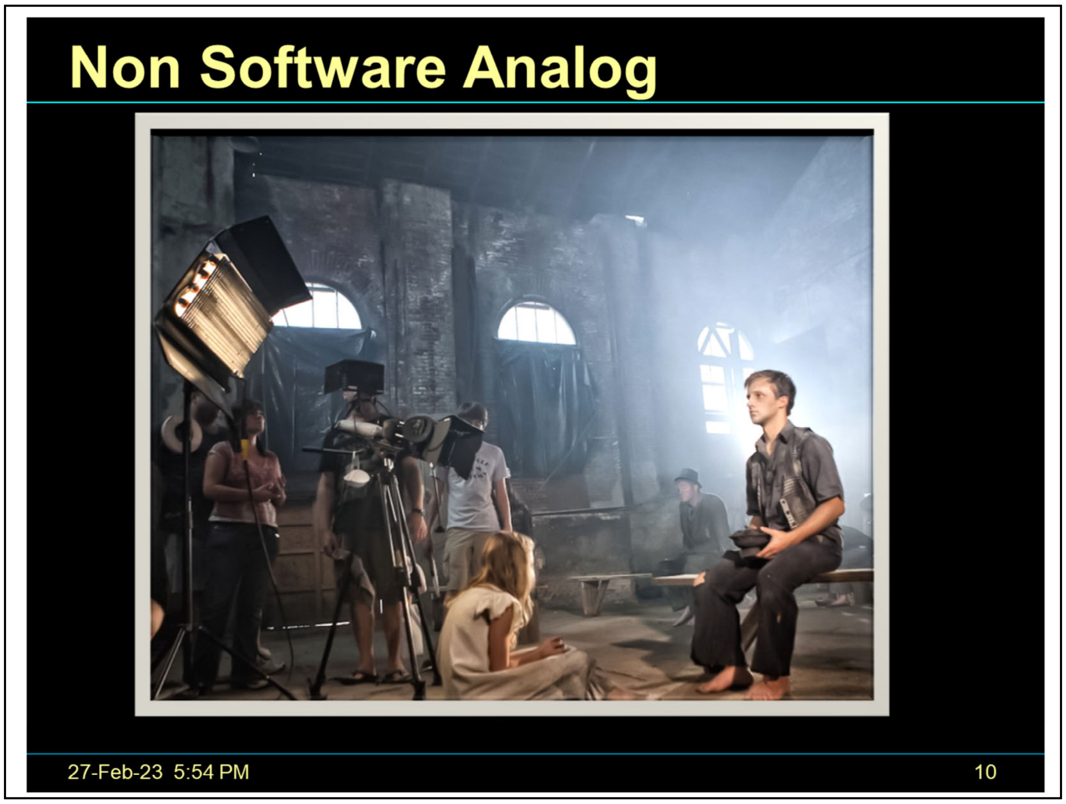
The Subject provides same interface for RealSubject and the Proxy.

By implementing this interface, the Proxy can be substituted for the RealSubject anywhere it occurs.

The RealSubject is the object that does the real work.

The Proxy holds a reference to the RealSubject.

Clients interact with the RealSubject via Proxy



Non Software Analog

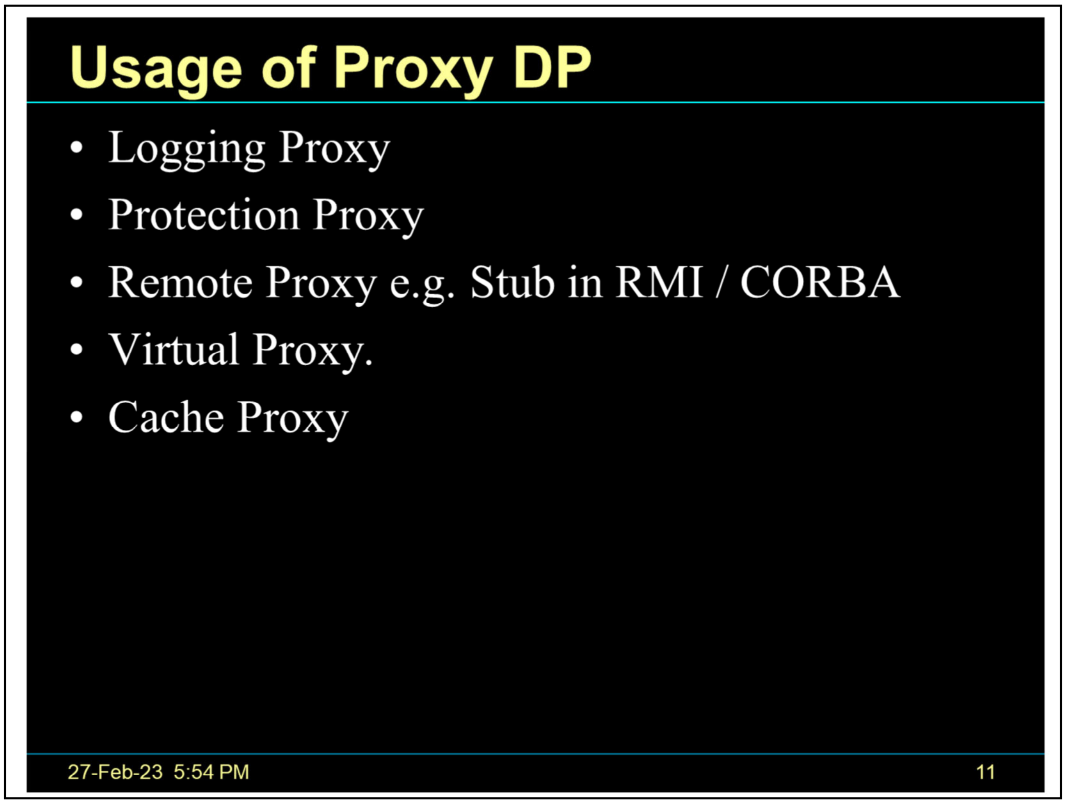
27-Feb-23 5:54 PM 10

Movie double replaces a real hero during light setup

Another examples: Garden hose with and without attachment.

I use a hand-held hose to water my garden. A few times a year, I add a container between the hose fitting and the watering head, which puts plant food into the water as it passes through.

I take no different action when watering in this way, and yet there is additional behavior; feeding the plants. Also, various different plant foods could be introduced in this way (although only one at any given time), and again I would take no different action at watering time.



Usage of Proxy DP

- Logging Proxy
- Protection Proxy
- Remote Proxy e.g. Stub in RMI / CORBA
- Virtual Proxy.
- Cache Proxy

27-Feb-23 5:54 PM 11

Virtual Proxy: When the actual object is heavyweight

A remote proxy provides a local representative for an object that resides in a different address space.

This is what the "stub" code in RPC and CORBA provides.

A protective proxy controls access to a sensitive master object. The "surrogate" object checks that the caller has the access permissions required prior to forwarding the request.

A smart proxy interposes additional actions when an object is accessed. Typical uses include:

Counting the number of references to the real object so that it can be freed automatically when there are no more references (like a smart pointer),

Loading a persistent object into memory when it's first referenced,

Checking that the real object is locked before it is accessed to ensure that no other object can change it.

======================

There are a number of different proxies, named for the behavior they add to class being proxied. Only some of them are possible with the class proxy form, but all of them are possible with the form that uses delegation.

Some examples:

Logging Proxy: Logs all calls to the method of the original class, either before or after the base behavior, or both.

Protection Proxy: Block access to one or more methods in the original class. When those methods are called, the Protection Proxy may return a null, or a default value, or throw an exception, etc…

Remote Proxy: The Proxy resides in the same process space as the client object, but the original class does not. Hence, the Proxy contains the networking, piping, or other logic required to access the original object across the barrier. This cannot be accomplished with a class proxy.

Virtual Proxy: The Proxy delays the instantiation of the original object until its behavior is called for. If its behavior is never called for, then the original class is never instantiated. This is useful when the original class is heavyweight and/or there are a large number of instances needed. The Virtual Proxy is very lightweight. This cannot be accomplished with a class proxy.

Cache Proxy: The Proxy adds caching behavior to an object that represents a data source.

Usage of Proxy

- Mocking in TDD
- Java: Spring @Transactional