

SOFTWARE DESIGN

COSC 4353/6353

Dr. Raj Singh





What are Design Patterns?



Why Design Patterns?



Example



Design Pattern Types

OUTLINE

TOOLKIT, FRAMEWORK, AND DESIGN PATTERN



A toolkit is a library of reusable classes designed to provide useful, general-purpose functionality

E.g., Java APIs (awt, util, io, net, etc)



A framework is a specific set of classes that cooperate closely with each other and together embody a reusable design for a category of problems

E.g., Struts, JSF, WCF, WPF, etc.



A design pattern describes a general recurring problem in different domains, a solution, when to apply the solution, and its consequences

E.g., Factory, Façade, Singleton etc.



A framework embodies a complete design of an application



A pattern is an outline of a solution to a class of problems



A framework dictates the architecture of an application and can be customized (e.g. Entity)



When one uses a framework, one reuses the main body of the framework and writes the code it calls.



When one uses a toolkit, one writes the main body of the application that calls the code in the toolkit.

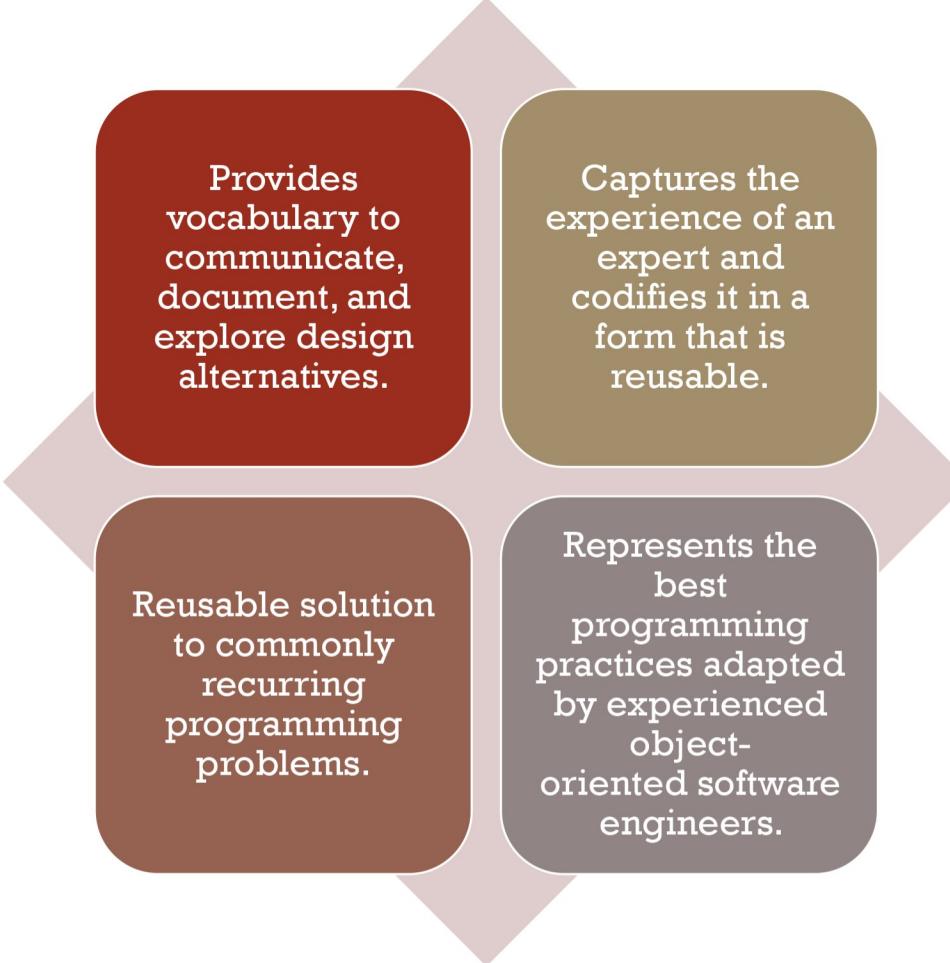


Design patterns are integral parts of frameworks and toolkits

TOOLKIT, FRAMEWORK, AND DESIGN PATTERN

WHAT ARE DESIGN PATTERNS?

- 💡 A reusable solution for common occurring problems
- 🛠 A description or template for how to solve a problem
- ✓ Formalized best practices to speed up the development process
- ✍ Provide tested, proven development paradigms
- 👤 OOP/OOD compatible
- 📄 Documented in a platform independent format



WHY DESIGN PATTERNS?

Reusable solution to commonly recurring programming problems.

Represents the best programming practices adapted by experienced object-oriented software engineers.

Provides vocabulary to communicate, document, and explore design alternatives.

Captures the experience of an expert and codifies it in a form that is reusable.

WHY DESIGN PATTERNS?



Effective software design requires consideration of:

short term and long term issues
improved code readability
ease of implementation and reproducible results



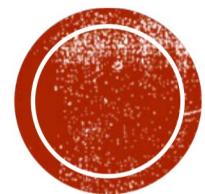
DP facilitates achieve reliable and flexible code



Patterns turn into components



Resolves known issues and can capture unknowns



EXAMPLES



THE INTERMEDIARY PATTERN



A client interacts with an intermediary



The requested services are carried out by the server/worker.

PROXY



Intermediary acts like a transmission agent



A proxy, in its most general form, is a class functioning as an interface to something else.



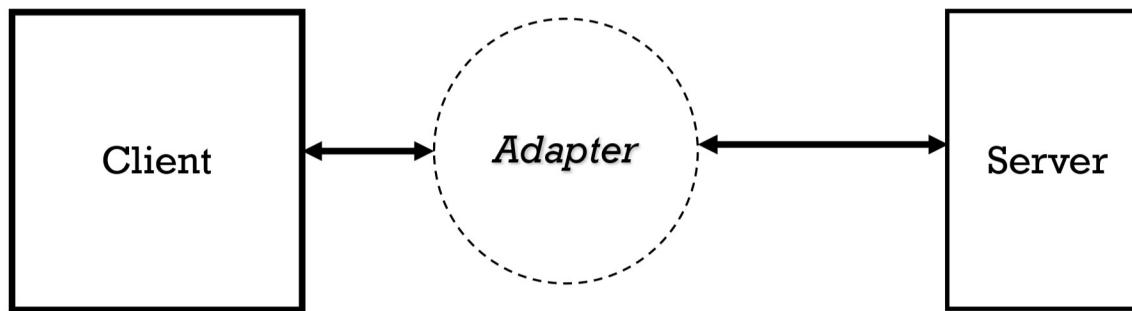
TRANSLATOR / ADAPTER



Intermediary acts like a translator between the client and the server.



E.g., Format/protocol conversions.



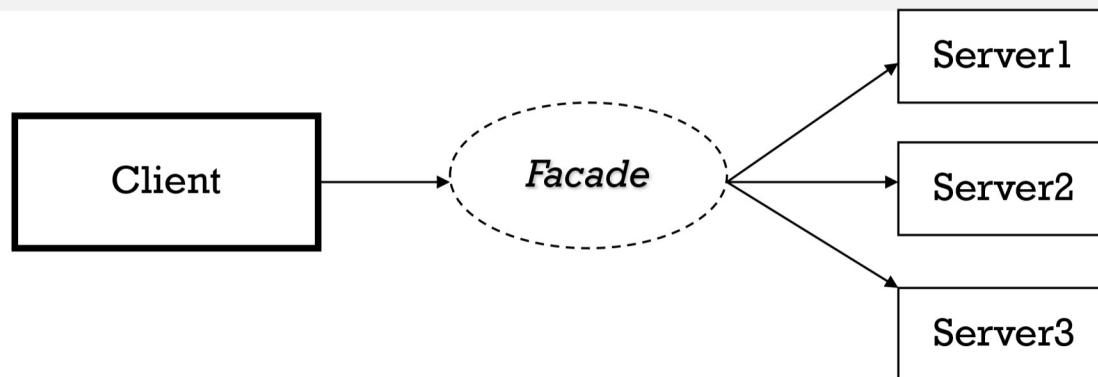
FACADE



Intermediary acts like a focal point distributing work to other agents.

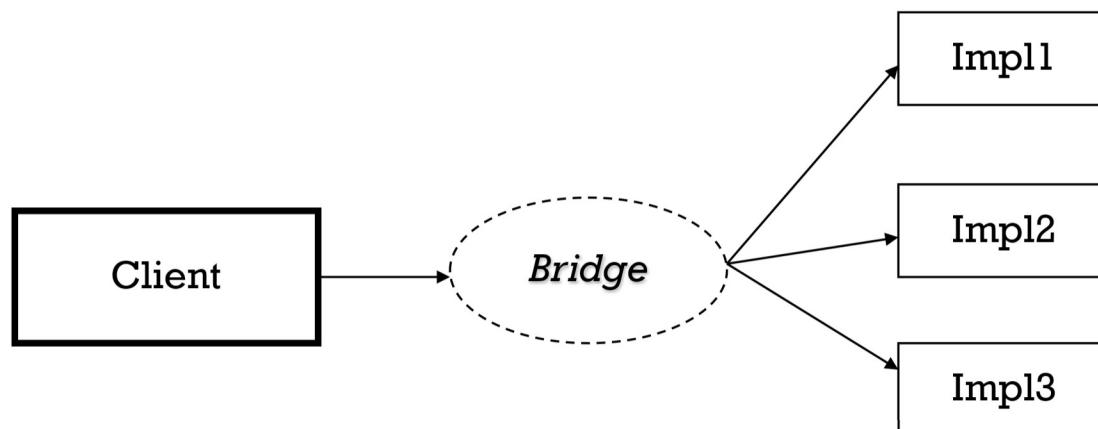


E.g. telnet, ftp, ... --> web-browser



BRIDGE/ABSTRACT FACTORY/HANDLE

- ✳️ Intermediary defines the interface but not the implementation.
- ✓ E.g., Motif/Mac/Windows look and feel





Several sections defining:



a prototypical micro-architecture (classes and objects)



developers copy and adapt to their particular designs



solution to the recurrent problem described by the design pattern

DP STRUCTURE



Must explain why a particular situation causes problems



Why the proposed solution is considered a good one



Must define the boundaries and environments it is applicable in



Must be a general approach with options

PATTERNS – MUST HAVE



Based on the problem scope there are different types



Creational



Structural



Behavioral



Architectural

DP TYPES

CREATIONAL

Creates object for you, rather than having you instantiate objects directly.

More flexibility in deciding which objects need to be created for a given case.

Abstract Factory

- groups object factories that have a common theme.

Builder

- constructs complex objects by separating construction and representation.

Factory

- method creates objects without specifying the exact class to create.

Prototype

- creates objects by cloning an existing object.

Singleton

- restricts object creation for a class to only one instance.

STRUCTURAL

These concern class and object composition

Defines ways to compose objects to obtain new functionality

Adapter

- allows classes with incompatible interfaces to work together by wrapping its own interface around that of an already existing class.

Bridge

- decouples an abstraction from its implementation so that the two can vary independently

Façade

- provides a simplified interface to a large body of code

Composite

- composes zero-or-more similar objects so that they can be manipulated as one object.

Flyweight

- reduces the cost of creating and manipulating similar objects

BEHAVIORAL

These concern how objects communicate with each other

Identifies common communication pattern

Chain of responsibility

- delegates commands to a chain of processing objects.

Command

- creates objects which encapsulate actions and parameters.

Interpreter

- implements a specialized language

Iterator

- accesses the elements of an object sequentially without exposing its underlying representation

State

- allows an object to alter its behavior when its internal state changes

ARCHITECTURAL

These address various issues in software engineering

Reusable solution to recurring problem in software architecture

Application

- create the composite architecture scalable, reliable, available and manageable

Data

- rules or standards that govern which data is collected, and how it is stored, arranged

HOMEWORK



Review class notes.



Additional reading:
Examples of Design Patterns



Start a discussion on Google Groups to clarify your doubts.