Advanced Object-Oriented Design
Lecture 10

Design patterns
Part I

Bartosz Walter
<Bartek.Walter@man.poznan.pl>

Agenda

- Motivation for patterns
- Systematics of patterns
- Patterns by Gang of Four
- Catalog of design patterns

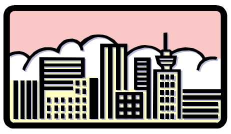
Motivation

- Can the common problems we encounter while development be solved in a similar manner?
- Can these problems be abstracted so that they could help in creating concrete solutions?

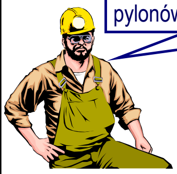
Motivation

"The pattern describes a problem, which occurs over and over again in our environment, and describes the core of the solution to that problem, in such a way that you can use the solution a million times over, without ever doing it the same way twice"

Christopher Alexander et al.:"A Pattern Language", 1977



Patterns for civil architecture

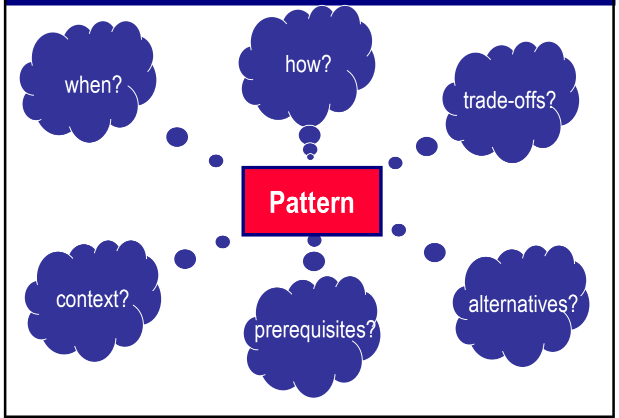

Czy zbudujemy most, opierając przęsło na kolejnych filarach połączonych łukiem, tak aby łuk usztywniał i naciągał przęsło stanowiąc jego podparcie na całej długości przęsła,
czy też mocując płyty z obu stron za pomocą kilku lin stalowych o kolejno coraz krótszych długościach do pylonów umieszczonych pośrodku długości mostu?

inspired by example by Ralph Johnson

Patterns for civil architecture

inspired by example by Ralph Johnson

Attributes of a pattern



Patterns in software

"Wzorzec projektowy identyfikuje i specyfikuje pewną abstrakcję, której poziom znajduje się powyżej poziomu abstrakcji pojedynczej klasy, instancji lub komponentu"

E. Gamma et al., 1993

Design pattern template (by the Gang of Four)

A pattern is described by:

- ***Name*** – *a part of designer's vocabulary*
- *Classification – category it belongs to*
- ***Intent*** – *what it does, what issue it addresses*
- *Also Known As – aliases*
- ***Motivation*** – *a scenario that illustrates a design problem and how the pattern solves that problem*
- *Applicability – situations in which it can be applied*
- ***Structure*** – *graphical representation of the classes involved in the pattern (class & interactions)*

Design pattern template (cont.)

- ***Participants*** – *classes and objects participating in the pattern and their responsibilities*
- ***Collaborations*** – *how the participants interact*
- ***Consequences*** – *trade-offs, variables and alternatives*
- *Implementation – pitfalls & hints for implementation, language-specific issues*
- *Sample Code*
- *Known Uses – examples of application in real systems*
- *Related Patterns – patterns closely related to this pattern*

Design patterns

Catalog of Design Patterns

Design patterns systematics

Creational

- abstract the instantiation process
- make the system independent of how the objects are created

Structural

- how the classes are composed
- use inheritance or composition appropriately

Behavioral

- deal with algorithms and assignment of responsibilities
- characterize control flow & interaction

Design patterns systematics

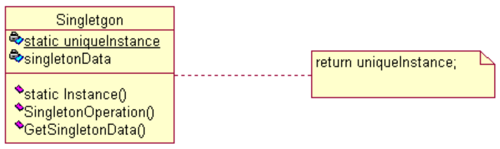
Creational	Structural	Behavioral
Abstract Factory	Adapter	Chain of responsibility
Builder	Bridge	Command
Factory Method	Composite	Interpreter
Prototype	Decorator	Iterator
Singleton	Facade	Mediator
	Proxy	Memento
		Observer
		State
		Strategy
		Template method
		Visitor

Singleton: Intent

- Ensure **the class has a single instance** throughout the application
- Provide **an access point** for it

by the Gang of Four

Singleton: Structure



by the Gang of Four

Singleton: Participants

Singleton

- defines statically available *getInstance()* operation for accessing the object
- restricts access to its constructor only to itself and its subclasses
- is responsible for creating its own (or its subclass') unique instance

by the Gang of Four

Singleton: Consequences

Singleton:

- takes care of the creating its instance
- separates Clients from managing its instance; they expect the instance to exist when requested
- allows for refinement of the instance by subclassing
- can be extended to a pool of instances
- is usually stateless (due to multithreading issues)
- acts like a global instance
- may increase coupling

Apply only when needed!

by the Gang of Four

Singleton: Example of use

There are several tax systems for small companies:

- *flat tax*
- *income-based tax*
- *ordinary PIT*

The company can choose one of them.

by the Gang of Four

Singleton: 2PL version

```
static public Tax getInstance() {
   if (taxInstance == null) {
      synchronize (this) {
         if (taxInstance == null) {
            taxInstance == new TaxA();
         }
      }
   }
   return instance;
}
```

But:

- Java compiler can optimize the code and allow two instances
- use *volatile* keyword to mark synchronization method

by A. Shalloway and J. Trott

Singleton: Inner classes

```
public class TaxA extends Tax {
   private static class Instance {
      static final Tax instance = new TaxA();
   }

   private TaxA() {}
   public static Taxt getInstance() {
      return Instance.instance;
   }
}
```

A class loader loads a single instance of TaxA.Instance class, which holds a single instance of Tax object.

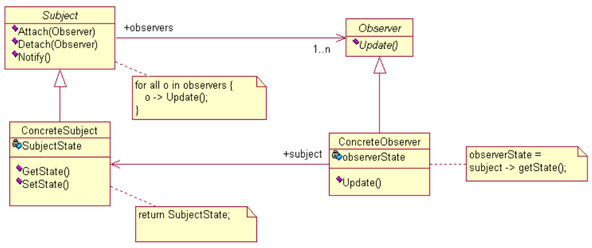
by A. Shalloway and J. Trott

Observer: Intent

Define a **one-to-many dependency** between objects so that when **one object changes state**, **all its depending objects are notified** and updated automatically

by the Gang of Four

Observer: Structure



by the Gang of Four

Observer: Participants

- **Subject**
  - knows its Observers
  - provides interface for attaching and detaching Observers
- **Observer**
  - defines an updating interface
- **Concrete Subject**
  - stores state of interest to Concrete Observers
  - sends notification ot its Observers
- **Concrete Observer**
  - maintains reference to Concrete Subject
  - updates its state with Subject

by the Gang of Four

Observer: Consequences

Observer permits for:

- abstract coupling between Subject and Observer
  - Subject knows hardly anything of its Observers
  - Subject and Observers may belong to different abstraction layers
- message broadcasting
- scalable updates (pull vs. push models)

  push: Observers get full information about change

  pull: Observers get plain notification, needs to query Subject for details
- keeping system's state consistent
- dangling references at Subjects to be removed (solution: weak references)

by the Gang of Four

Observer: Example of use

To stay in touch with your favourite news service, you can subscribe to a newsletter.

Any urgent news will be broadcasted to all subscribers until they cancel subscription.

News are broadcasted either as full data or plain links

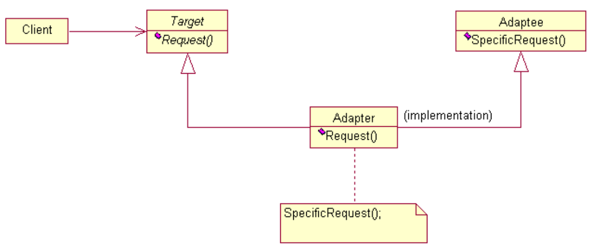
by the Gang of Four

Adapter (Wrapper): Motivation

- **Allow classes to work together** that could not happen otherwise because of incompatible interfaces
- **Convert the interface of a class** into another interface a client expects

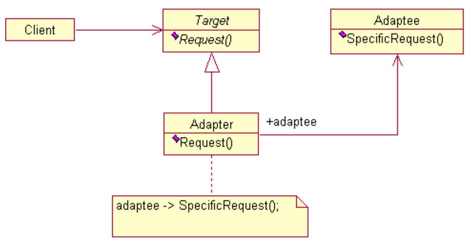
by the Gang of Four

Adapter (Wrapper): Structure for inheritance



by the Gang of Four

Adapter (Wrapper): Structure for delegation



by the Gang of Four

Adapter (Wrapper): Participants

- **Target**
  defines the domain-specific interface
- **Client**
  collaborates with objects conforming to the Target interface
- **Adaptee**
  defines an existing interface that needs adapting
- **Adapter**
  adapts the interface of Adaptee to Target

by the Gang of Four

Adapter (Wrapper): Consequences for object adapter

- **high flexibility**
  a single Adapter may work with many Adaptees
  Adapter can add functionality to all Adaptees at once (Decorator)
- **difficult behavior overriding**
  it requires subclassing Adaptee and making Adapter to refer directly to the subclasses

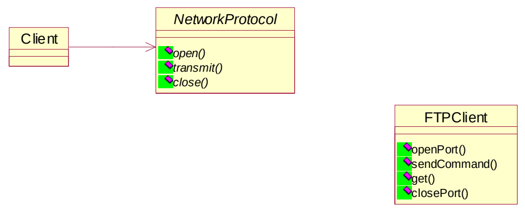
by the Gang of Four

Adapter (Wrapper): Consequences for class adapter

- **low flexibility**
  - Adapter adapts only to a concrete Adaptee class, not its subclasses
  - Adaptees cannot be changed at runtime
- **potential behavior change**
  - Adapter may override some of Adaptee behaviour (Decorator)
- **low overhead**
  - introduces only a single object
  - no additional reference is needed to get to the Adaptee

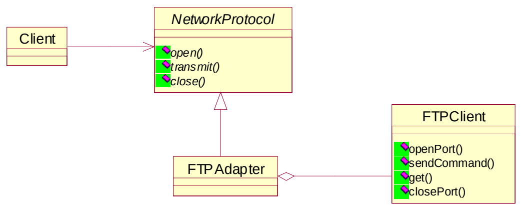
by the Gang of Four

Adapter (Wrapper): Example of use



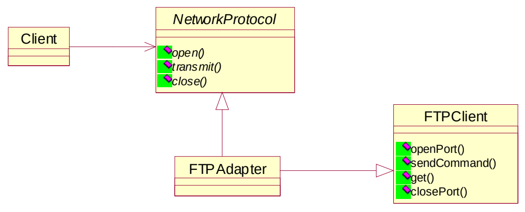
by the Gang of Four

Adapter (Wrapper): Example of use



by the Gang of Four

Adapter (Wrapper): Example of use



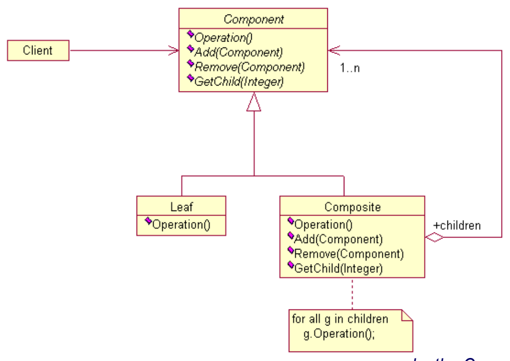
by the Gang of Four

Composite: Motivation

- Compose objects into tree-like structures to **represent part-whole hierarchies**
- Allow for **uniform handling of both single objects and compound structures**

by the Gang of Four

Composite: Structure



by the Gang of Four

Composite: Participants

- **Component**
  - declares an interface for objects in composition
  - implements common behaviour to all classes
- **Leaf**
  - represents a node without children
- **Composite**
  - represents a node with children
  - stores references to children

by the Gang of Four

Composite: Consequences

- **defines class hierarchies**
- **makes the client application simple**
- **makes adding new kind of components easier**
- **permits a variable number of instances to exist**
- **restricting components is difficult**

by the Gang of Four

Composite: Example of use

Organizations usually have a tree-like form. Departments make tree branches, and employees are the leaves.

Both departments and single employees can be managed in a common way.

Regardless from the actual organization's size, it is managed in a uniform way.

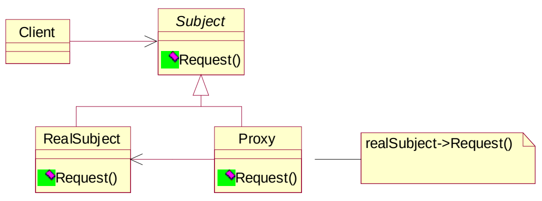
by the Gang of Four

Proxy: Motivation

- **Provide a surrogate** for another object to controll access to it
- Smoothly **defer initialization after the object is needed**
- **Initialize actual objects on demands**

by the Gang of Four

Proxy: Structure



by the Gang of Four

Proxy: Participants

- **Proxy**
  - maintains reference to *RealSubject*
  - implements same interface as *Subject*
  - controls access to the *RealSubject*
- **Subject**
  - defines a common interface for *Proxy* and *RealSubject*
- **Real Subject**
  - defines the real object represented by *Proxy*

by the Gang of Four

Proxy: Consequences

- **confusing indirections**
  object may reside in a different address space
- **optimizations**
  virtual proxies may provide optimizations
- **high level of protection for the actual object**
  may require additional attention and care

by the Gang of Four

Proxy: Applicability

- A **remote proxy** provides a local representative for an object in a different address space
- A **virtual proxy** creates expensive objects on demand
- A **protection proxy** controls access to the original object

by the Gang of Four

Proxy: Example of use

Huge memory arrays can be expensive in handling.

Instead of creating the array at startup, a proxy can be used to avoid the initial overhead. The proxy stands for the actual array, since it implements the same interface.

Remote objects are difficult to handle and expensive in creation.

Remote proxy hides the complexity of network communication and acts like a local stub for the remote object

by the Gang of Four

Factory Method: Motivation

- Define **an interface for creating an object**
- Let a class **defer instatiation to subclasses**
- Let the **subclass decide the class and method of creating the product object**

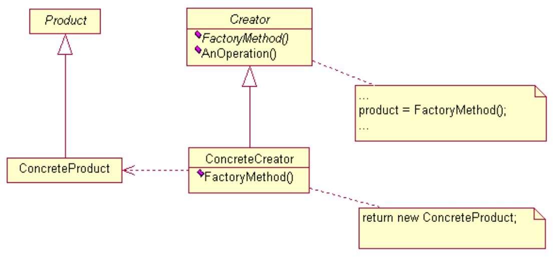
by the Gang of Four

Factory Method: Applicability

- Client can't anticipate the class of objects it must create
- A class wants its subclasses to specify the objects to create
- Classes delegate responsibility to one of several helper subclasses

by the Gang of Four

Factory Method: Structure



```
...
product = FactoryMethod();
...
```

```
return new ConcreteProduct;
```

by the Gang of Four

Factory Method: Participants

- **Product**
  - declares an interface for objects created by *Factory Method*
- **ConcreteProduct**
  - implements the *Product* interface
- **Creator**
  - declares an interface for a type of *Product* object
  - may provide a default implementation of *Factory Method*
- **ConcreteCreator**
  - overrides the factory method to return *Concrete Product*

by the Gang of Four

Factory Method: Consequences

- **provided hooks for subclasses**

  gives subclasses a hook for providing an extended version of an object
- **Parametrized FM vs. Polymorphic FM**

  Creator can select the object to create or it is the responsibility of the subclasses

by the Gang of Four

Factory Method: Example of use

A word processor creates documents of different formats. Regardless from the format, they are handled in similar way.

Factory Method allows to hide the decision about the type of the document. The type of document depends on the factory implementation, not the document itself.

Collection.iterator() creates appropriate object for the underlying collection.

by the Gang of Four

Abstract Factory: Motivation

Provide **an interface for creating families of related or dependent objects** without specyfying concrete classes

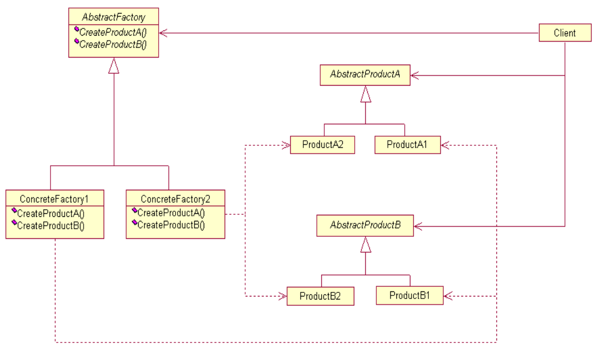
by the Gang of Four

Abstract Factory: Applicability

- System should be independent of how its products are created, composed and represented
- A family of related *Product* objects is designed to be used together
- A library of products should not reveal their implementation

by the Gang of Four

Abstract Factory: Structure



by the Gang of Four

Abstract Factory: Participants

- **Abstract Factory**
  declares an interface for operations that create *Abstract Products*
- **Concrete Factory**
  implements operations to create *Concrete Products*
- **Abstract Product**
  declares an interface for a type of *Product* object
- **Concrete Product**
  defines a product object to be created by appropriate *Concrete Factory*
  implements the Abstract Product interface

by the Gang of Four

Abstract Factory: Consequences

- **isolation of concrete classes**
  Factory encapsulates responsibility for object creation
  clients manipulate instances through their abstract interfaces
- **ease of *Product* families exchange**
  only the *Concrete Factory* needs to replaced
- **promotion of consistency among *Products***
  declares an interface for a type of *Product* object
- **difficult support for new *Products***
  Abstract Factory fixes the set of delivered *Products*

by the Gang of Four

Abstract Factory: Implementation

- **Factories as singletons**
  only a single *ConcreteFactory* per product family is needed
- **Creating the products via *Factory Methods***
  every product has its *Factory Method* in an *Abstract Factory*
- **Extensible *Factories***
  the type of *Product* is determined by a parameter

by the Gang of Four

Abstract Factory: Example of use

Publishing application can produce both screen and paper output. The widgets used for assembling the publications are make families of corresponding objects.

In order to reduce the configuration needed to get the required format, the Abstract Factory produces an appropriate set of widgets, which are referenced as abstract objects.

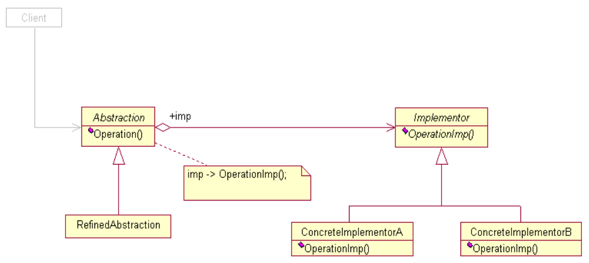
by the Gang of Four

Bridge: Motivation

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

by the Gang of Four

Bridge: Structure



by the Gang of Four

Bridge: Participants

- **Abstraction**
  - defines the abstraction interface
  - maintains a reference to the Implementor object
- **Refined Abstraction**
  - extends the interface defined by Abstraction
- **Implementor**
  - defines the interface for implementations
  - may be different from Abstraction
- **Concrete Implementor**
  - implements Implementor interface

by the Gang of Four

Bridge: Consequences

Bridge:

- decouples the interface from implementation
- improves extensibility of Abstraction and Implementor
- hides implementation details from clients

by the Gang of Four

Bridge: Implementation

- **Single Implementor only**
  - Bridge is useful even if there is only one Implementor
- **Strategies for choosing right Implementor**
  - Abstraction instantiates appropriate Implementor in constructor
  - use default Implementor initially and change it later
  - delegate the decision to another object (a Factory)

by the Gang of Four

Bridge: Example of use

Graphical objects represented in application can defer their functions to specialized libraries, which will manage them.

Objects should only know the library interface, not specific implementation

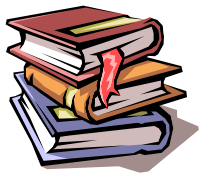
Libraries can be switched as long as they share common interface.

by the Gang of Four

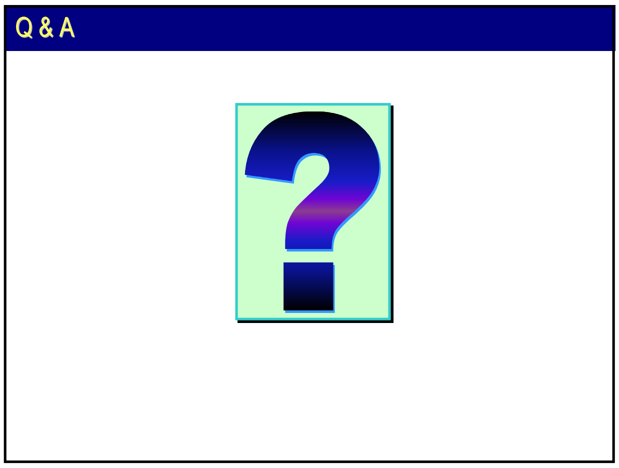
Design patterns

... to be continued

Readings



1. Gamma E. et al., ***Design Patterns. Elements of Reuseable Object-Oriented Software.*** **PWN 2005**
2. Eckel B., ***Thinking in patterns.*** **http://www.bruceeckel.com**
3. Cooper J., ***Java. Wzorce Projektowe.*** **Helion, 2001**
4. Shalloway A., Trott J., ***Projektowanie zorientowane obiektowo. Wzorce projektowe.*** **Helion, 2001**



Q & A