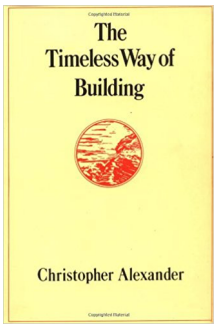
Design Patterns



Oregon State University

Christopher Alexander

Architect, who asked the question ***"What makes good architectural design?"***

By studying high quality structures that solve similar patterns, he saw that ***patterns*** would appear the solutions to the problems.

He identified over 200 pattern for city planning, building design, gardens etc.

Oregon State University

Patterns

*"Each pattern is a three-part rule, which expresses a relation between a certain **context**, a **problem**, and a **solution**."*

-Christopher Alexander

Oregon State University

Patterns

Four elements describe the pattern:

The ***name***

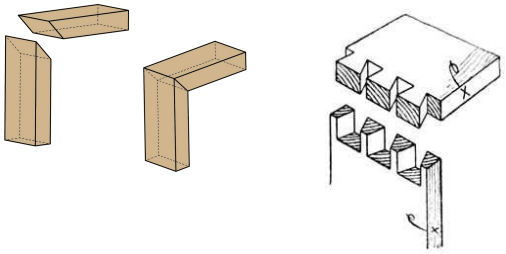
The ***purpose***; what problem is solves

How to solve the problem; the ***solution***

The ***constraints*** we have to consider in our solution

Oregon State University

A higher level perspective



Oregon State University

A higher level perspective

Patterns also describe a ***shared vocabulary.***

Which one is better?

"Should we use a dovetail or miter joint?"

or

"Should I make the joint by cutting down into the wood and then going back up 45 degrees and..."

Oregon State University

A higher level perspective

The former avoids getting bogged down in details

The former relies on the carpenter's ***shared knowledge***

[Design patterns] distill and provide a means to reuse the design knowledge gained by experienced practitioners.

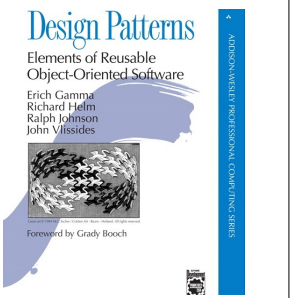
-G.O.F.

Oregon State University

Software Design Patterns

The seminal book published by the "Gang of Four."

They propose 23 patterns, organized in 3 categories.



Oregon State University

Key Features of a Pattern

Name

Intent: The purpose of a pattern.

Problem: What problem does it solve?

Solution: The approach taken.

Participants: The entities involved in the pattern.

Consequences: The effect the pattern has on your code.

Implementation: Example ways to implement it

Structure: a class diagram

Oregon State University

Software Design Patterns

3 Categories:

Creational: they abstract away the object instantiation (creation)

Structural: are concerned with how classes and objects are composed to form larger structures

Behavioral: are concerned with algorithms and the assignment of responsibilities between objects.

Oregon State University

Creational patterns

Oregon State University

Singleton

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

Motivation: having a single instance of a class is sometimes important; e.g. There can be only one file system or event thread.

Oregon State University

Singleton

We want to restrict access such that this is no longer possible:

```
Singleton s = new Singleton();
```

Instead, we want to do this:

```
Singleton s = Singleton.getInstance();
```

We want to ensure that only a ***unique instance*** exists.

Oregon State University

```
public class Singleton {
    private static Singleton s = null;

    private Singleton() {}

    public static Singleton getInstance() {
        if (s == null)
            s = new Singleton();
        return s;
    }
}
```


Oregon State University

```
public class Singleton {
    private static Singleton s = null;

    private Singleton() {}

tInstance() {
);
}
```

Declaring the constructor ***private*** means we cannot create instances outside of the class.

Therefore, we control where an object can be instantiated.

Oregon State University

```
public class Singleton {
    private static Singleton s = null;

    private Singleton() {}

    public static Singleton getInstance() {
        if (s == nu
            s = new
        return s;
    }
}
```

The ***static*** keyword allows us to access fields and methods without an instance:

```
Singleton.getInstance();
```


University

```
public cl
    priva
    priva

    public static Singleton getInstance() {
        if (s == null)
            s = new Singleton();
        return s;
    }
}
```

This is called ***lazy initialization.*** We only create the object when we need them.

Oregon State University

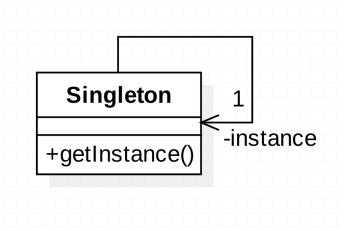
```
public class Singleton {
    private static Singleton s = null;

    private Singleton() {}

    public static Singleton getInstance() {
        if (s == null)
            s = new Singleton();
        return s;
    }
}
```


Oregon State University

Structure



Oregon State University

Pros & Cons

Pros:

Easy instance management

Cons:

It acts like a global variable and shares all the problems associated with them

Breaks SRP, as the objects now has to control it's own lifetime cycle

Oregon State University

Structural Patterns

Oregon State University

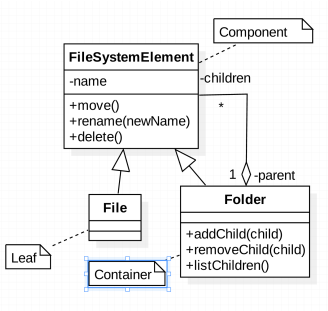
Composite

Intent: compose objects into tree structures to represent part-whole hierarchies. Clients can treat individual objects and compositions **uniformly**.

Motivation: A file system has files and folders. Users want to manipulate files and folders the same way (e.g. move, rename, delete etc).

Oregon State University

Composite



Oregon State University

```
public abstract class FileSystemElement {
    public void move(){}
    public void rename(String newName){}
    public void delete(){}
}

public class Folder extends FileSystemElement {
    private List<FSE> children = new ArrayList<>();

    public void addChild(FileSystemElement child) {
        children.add(child);
    }

    public void removeChild(FileSystemElement child) {
        children.remove(child);
    }
}

public class File extends FileSystemElement {
    // do file stuff
}
```


Oregon State University

```
public abstract class FileSystemElement {
    public void move(){}
    public void rename(String newName){}
    public void delete(){}
}

public cla
    privat

    public
        ch
    }

    public
        ch
    }
}

public cla
    // do
}
```

The base class contains the ***common operations*** for a "File System Element"

`abstract` means that it cannot be instantiated.

Oregon State University

```
public abstract class FileSystemElement {
    public void move(){}
    public void rename(String newName){}
    public void delete(){}
}

public class Folder extends FileSystemElement {
    private List<FSE> children = new ArrayList<>();

    public void addChild(FileSystemElement child) {
    }

    p
    }
}

publi
    /
}
```

The `Folder` class is the ***container***.

It's responsibility is managing and accessing the children.

It may override some common operations (e.g. `delete`)

Oregon State University

```
public abstract class FileSystemElement {
    public void move(){}
    public void rename(String newName){}
    public void delete(){}
}

public class Folder extends FileSystemElement {
    private List<FSE> children = new ArrayList<>();

    public void addChild(FileSystemElement child) {
    }

    pu
    }
    pu                                    d) {
    }
}

public class File extends FileSystemElement {
    // do file stuff
}
```

The `File` class is the ***leaf,*** as it had no children.

Oregon State University

Pros & Cons

Pros:

It's easy to add new types of components

Clients can manipulate both types homogeneously.

Cons:

It's hard to restrict the types of a component (design is too general)

Oregon State University

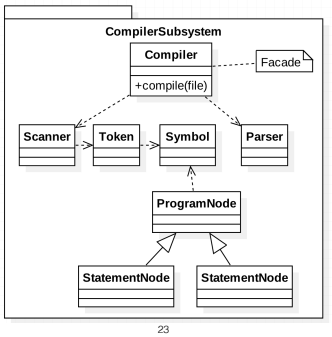
Facade

Intent: provide a unified interface to a set of interfaces in a subsystem. It defines a higher-level interface that makes the subsystem easier to sue.

Motivation: Structuring a system into subsystems reduces complexity. Facade provides a single, simplified interface to the subsystem.

Oregon State University

Facade



Oregon State University

Pros & Cons

Pros:

Isolates clients from subsystem components

Minimizes the dependency of the client code on the subsystems

Cons:

The Facade class risks accumulating a lot of responsibility because it is linked to all the classes in the application

Oregon State University

Behavioral Patterns

Oregon State University

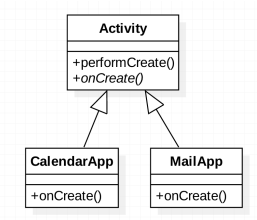
Template Method

Intent: define a skeleton of an algorithm and defer some steps to subclasses.

Motivation: The Android OS must support multiple types of app. These apps all have a common lifecycle and need to handled uniformly by the OS.

Oregon State University

Template Method



Oregon State University

Template Method

The base class provides the ***basic steps*** of the algorithm.

The subclasses provide the ***details.***

Oregon State University

```
public abstract class Activity {
    final void performCreate(Bundle icicle) {
        restoreHasCurrentPermissionRequest(icicle);
        onCreate(icicle);
        mActivityTransitionState.readState(icicle);
        performCreateCommon();
    }

    public abstract void onCreate(Bundle bundle);
}

public class MyApp extends Activity {
    @Override
    public void onCreate(Bundle bundle) {
        // app specific stuff goes here
    }
}
```


Oregon State University

```
public abstract class Activity {
    final void performCreate(Bundle icicle) {
        restoreHasCurrentPermissionRequest(icicle);
        onCreate(icicle);
        mActivityTransitionState.readState(icicle);
        performCreateCommon();
    }

    public abstract void onCreate(Bundle bundle);
}

public class MyApp e
    @Override
    public void onCreate(Bundle bundle) {
        // app specific stuff goes here
    }
}
```

(Part of) the algorithm for starting an `Activity`.


```
public abstract class Activity {
    final void performCreate(
        restoreHasCurrentPerm
        onCreate(icicle);
        mActivityTransitionSt
        performCreateCommon()
    }

    public abstract void onCreate(Bundle bundle);
}

public class MyApp extends Activity {
    @Override
    public void onCreate(Bundle bundle) {
        // app specific stuff goes here
    }
}
```

The specific details are handled by the subclasses.

Pros & Cons

Pros:

Helps eliminate code duplication

Easy to customize the algorithm

Cons:

Your options are limited by the existing skeleton

Observer

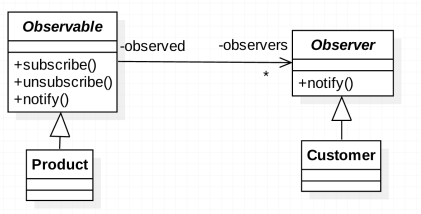
Observer

Intent: Define a one-to-many relationship between objects, so that when one object changes it's state, the dependents are notified and updated automatically.

Motivation: An online store is about to receive a large shipment of a high demand product. The store wants to notify the customers when the product is in stock.

Oregon State University

Observer



Oregon State University

```
public abstract class Observable {

    private List<Observer> observers = new ArrayList<>();

    public void subscribe(Observer o) {
        observers.add(o);
    }

    public void unsubscribe(Observer o) {
        observers.remove(o);
    }

    public void notify(Object data) {
        for(Observer o : observers) {
            o.notify(data);
        }
    }

}

public abstract class Observer {
    public abstract void notify(Object data);
}
```


Oregon State University

```
public abstract class Observable {

    private List<Observer> observers = new ArrayList<>();

    public void subscribe(Observer o) {
        observers.add(o);
    }

    public void unsubscribe(Observer o) {
        observers.remove(o);
    }

    publi
        f
        }
    }
}

public ab
    public abstract void notify(Object data);
}
```

The `Observable` keeps track of *observers* and provides methods to subscribe and unsubscribe

Oregon State University

```
public abstract class Observable {

    private List<Observer> observers = new ArrayList<>();

    public void subscribe(Observer o) {
        observers.a
    }

    public void uns
        observers.r
    }

    public void notify(Object data) {
        for(Observer o : observers) {
            o.notify(data);
        }
    }
}

public abstract class Observer {
    public abstract void notify(Object data);
}
```

It also handles notifying the observers

Oregon State University

```
public class Product extends Observable{
    public void updateStock(int units) {
        // .....
        this.notify(units);
    }
}

public class Customer extends Observer {
    public void notify(Object data) {
        // react to the notification
    }
}
```


Oregon State University

```
public class Product extends Observable{
    public void updateStock(int units) {
        // .....
        this.notify(units);
    }
}

public class Customer e
    public void notify(
        // react to the
    }
}
```

The client can notify all the observers of the change.

Oregon State University

```
public class Product extends Observable{
    publi
        /
        t
    }
}

public class Customer extends Observer {
    public void notify(Object data) {
        // react to the notification
    }
}
```

The observer can deal with the notification in any way it sees fit.

Oregon State University

Pros & Cons

Pros:

Observers are isolated from Observables

You can dynamically subscribe and unsubscribe

Cons:

The order in which Observers are called might not be deterministic.