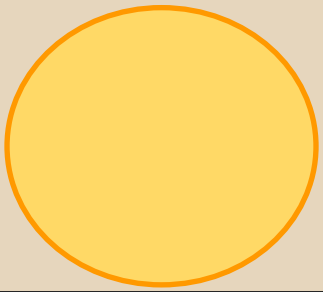
Design Patterns



How the Problem Determines the Pattern

by Team Birdfeedr

Note that this presentation focuses on only object oriented design patterns.
Note on note: An object oriented design pattern can work for non object oriented languages, but the transition isn't very smooth and some design patterns are made obsolete by the language.

But what's a design pattern?

"A design pattern is a repeatable solution to a software engineering problem."

- Templates for multiple problems

Wei

SOURCE: http://code.tutsplus.com/articles/a-beginners-guide-to-design-patterns--net-12752

Design patterns are optimized, reusable solutions to the programming problems that we encounter every day.

It is a template.

Design patterns are well researched and battle tried solutions.

Distinguishable -- there are characteristics of a single design pattern that make it recognizable.

Elements of a Design Pattern

1. The name
2. The problem
3. The solution.
4. Consequences.

Wei

Source: “Design Patterns - Elements of Reusable Object-Oriented Software”

Every design pattern has a name.

The problem that the design pattern fits with.

The way the design pattern solves the problem.

The tradeoffs or pros and cons of using the design pattern.

The 3 Basic Kinds of Patterns

1. Creational
2. Structural
3. Behavioral

Wei

SOURCE: http://www.gofpatterns.com/design-patterns/module2/three-types-design-patterns.php

Creational design patterns control creating an object. They prevent outside classes from creating the object as in the case of a singleton. They can also use polymorphism and help the program choose between classes at runtime.

Structural design patterns use structures having beneficial properties. For example, they may help classes connect to each other, increase code reuse, and allow teams to work together better.

Behavioral patterns focus on interactions between objects and how they communicate with each other.

Transition:

Can any design pattern work for me?

No.

Thomas

Why not?

Know your problem before choosing a design pattern.

Thomas

SOURCE: http://programmers.stackexchange.com/questions/227868/choosing-the-right-design-pattern

Choose a design pattern when the tradeoff meets your needs.

Design patterns and Math formulas

- Design patterns to design are are like math formulas to math.
- By understanding the problem you have a better understanding of what formula to use, if one exists.

Nathan

Thank you Thomas, Hello Im Nathan and i'm going to offer a comparison to design patterns to help better understand their nature and how knowing the problem helps.

To better understand how knowing a problem aids in the application of a design pattern we can look to another similar field that depends on this same concept, math.
More specifically how understanding a problem helps in picking, or developing, the right formula.

Note: Used http://sourcemaking.com/design_patterns to gain understanding for this comparison.

Swimming Pool Allegory

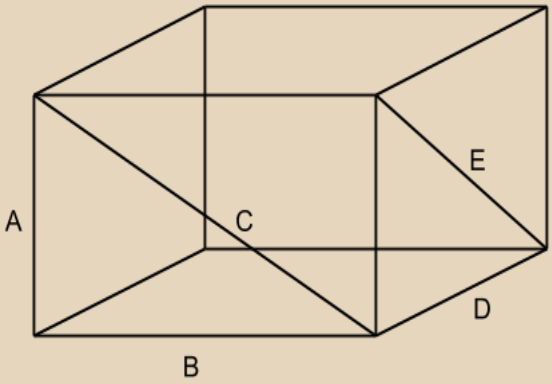
- We want to know the volume of a rectangular pool.
- Choose Pythagorean theorem.

Nathan

For example: lets say we want to know the volume of a rectangular pool? While there are many formulas to pick from, only a few will help us directly in making the right answer.

Let's say we, not knowing our problem or the potential solutions, just picked the formula we knew best, say the Pythagorean theorem, to solve our quarry.

Swimming Pool Allegory Cont.



- Starting with ***Pythagorean theorem***:
  - Pick C, B, E as lengths to measure
  - Volume = Sqrt(C^2 - B^2) * B * Sqrt(E^2 - C^2 + B^2)
- Starting with ***Volume of rectangle***:
  - Pick A, B, D as lengths to measure
  - Volume = A * B * D

- Ended up making volume formula with extra calculations.

Nathan

While it is possible to find the volume of a pool using this formula, seen here <Read first bullet>.

Pythagorean theorem:
A^2 + B^2 = C^2 ---> A = Sqrt(C^2 - B^2)
B = B
A^2 + D^2 = E^2 ---->D = Sqrt(E^2 - A^2) = Sqrt(E^2 - (C^2 - B^2)) = Sqrt(E^2 - C^2 + B^2)

All we really did was derive the real formula for a rectangular pool which is the width * length * height with some extra overhead measuring the diagonals. Shown in simpler form here<Read second bullet>.

Volume of rectangle:
*V = A * B * D*

By knowing our problem better, and what we have to use, we can see that all we needed from the beginning was the standard formula for the area of a rectangle without the need for the Pythagorean theorem at all.

Math problems and Design problems

- Both have a wide variety of patterns or formulas to select from.
- Both are easier to solve with understanding of the problem.
- When a good pick is made the outcome can be much smoother and less error prone.

Nathan

Of course it can be hard to see what formulas or patterns fit what problems best in both these fields, especially for a novice.
But just like with math, with research and understanding of the nature of your problem and the understanding of the formulas or patterns.
you will be able to see that:

Both have a wide variety of patterns or formulas to select from.
Both are easier to solve with understanding of the problem
And when a good pick is made the outcome is much smoother and less error prone.

And with that i'll hand things off to Josh for a more Computer science tuned example of why knowing your problem is important to picking a design pattern.

Application to CS

- The math examples shows the concept.
- What about an example for CS?

Josh

So Here's a CS Example

- We are tasked to create a program.
- The program accesses a bank account.

Josh

You're in charge of a program that connects to a bank account and performs various functions similar to an ATM.

So Here's a CS Example

- Program functions.
  - Make withdrawals and deposits.
  - Display balance.

Josh

So Here's a CS Example

- Problem analysis
  - No duplicate withdrawals.
  - Accurate balance summaries.
- What pattern could we use?
  - First thought: Singleton pattern

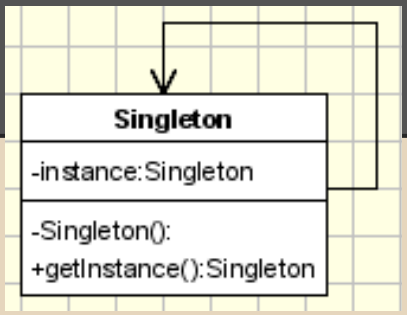
Josh

If multiple connections are made, then the customer could withdraw twice.

Also, if balance summary occurs before the withdraw then the customer could have the wrong information.

This sounds like you can use the singleton pattern! But what is it?

The Singleton Pattern



- Controls creation.
- A maximum of one object.
- Makes the constructor private.
- Provides a getInstance() method.

Josh

A singleton pattern controls the creation of a object such that only one may exist during the lifetime of the application.

The object makes the constructor private so outside objects can't create the object and the object provides
a method for creating instances that limits number of instances to one.

How Does It Apply?

- Prevents duplicate connections.
- Shared data.

Josh

The singleton pattern seems like it could coordinate the actions taken on the bank account by allowing all class to have access to only that one instance.

Any method may call the static getInstance() method to access the connection. Thus, methods need not have additional parameters.

That wasn't a good idea.

"Most importantly, any design pattern can be a double-edged sword— if implemented in the wrong place, it can be disastrous and create many problems for you."

- Concurrency issues?
- Global access really needed?
- Lesson: analyze the problem better.

Josh

The problem is that the singleton pattern does not control the bank connection with multiple threads.

For example, if the program reads a summary of the account before making a withdraw, the summary will be out of date.
This can happen because we could two threads one for the display and another for the business logic.
These could be separate classes and they could both hold an instance of the singleton.

We also don't really need the benefit that global access gives us.

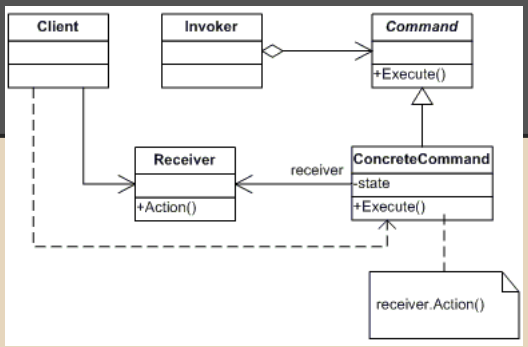
Clarifying the Problem

- We don’t need global access.
- Prevent multiple withdraw requests
- Order is important
- How about the command pattern?

Di

Thanks, Josh. So we’re going to clarify our problem to make a different pattern. In our problem, we don’t need to provide global access to our variables. We simply need to handle transaction requests as they come, as accurately as possible. And there is a design pattern that can fit this scenario much better than the singleton pattern: the command pattern!

Command Pattern



- Has three key classes:
  - Command
  - Invoker
  - Receiver
- The receiver sends commands to invoker.
- Invoker queues and processes commands.

Di

SOURCE: http://www.dofactory.com/net/command-design-pattern

In this pattern, a request becomes an object. The command class is an interface that concrete commands can be created from. A client can create commands, send it to the receiver - the receiver can then send these commands to the invoker, which will process the commands through a queue.

How does this work with our bank pattern? Well, if our client needs to withdraw some money and view a summary, the client class can just make a receiver object, and pass withdraw and summary command objects to it. The invoker then queues up these commands and does them in that order! Seems more useful than a singleton pattern.

This demonstrates how this pattern was able to suit our problem better. Now I'll hand things off to Miguel to wrap this up. Miguel?

How to Pick a Good Pattern

- Define your problem.

- The more you understand it, the more appropriate your choice of design pattern will be.

Miguel