Design Patterns

By David Cun

What are Design Patterns?

Ever heard of the term “why reinvent the wheel?” That’s Design Patterns in a nutshell. Someone else figured it out, everyone started to use it, and now it works in most cases. “But wait, isn’t that copying?” Unless it’s a replica of the first wheel and you claimed it as your own, nah you good.

Another way to think of it is cooking an Egg. You can boil it, scramble it, fry it, or if you’re insane, eat it raw. But you’re not discovering a new Egg cooking technique (at least I hope not), you’re applying what was discovered since the first time someone cooked an Egg.

In programmer speak, you’re not discovering new coding techniques no one has ever heard (not yet at least), you’re using techniques that have been learned and sharpened to figure out problems that seem similar.

Elements of a Design Problem

Names - Design Patterns have names, allowing you to recognize them. It helps you differentiate between boiling in a pot and frying in a pan. .

Problems - What types of problems can be solved with this? Are you trying to make soup or fry an egg?

Solution - Describes how things work to solve the problem.

Consequences - The tradeoff for using this Design Pattern. Sure you can boil soup in a pan, and fry in a pot, but come on.

Creational Patterns

Design patterns that deal with object creation mechanisms. It abstracts the instantiation process.

Singleton Pattern

Description - A single instance of a class that provides Global Access.

Cool Feature - Prevents the program from having duplicate instances of the class.

Example uses - Databases (such as user profiles).

Factory Pattern

Description - Creating a class that aids in the creation of other classes.

Cool Feature - Instead of having to type “Subclasstype name = new Subclasstype” every time, you can have it placed inside a class as a method. This also hides the specific subclass as long as you know it’s Abstract Class.

Example uses - AnimalFactory, CarFactory, RocketFactory

Builder Class

Description - Creating a class that allows you to construct complex objects step by step.

Cool Feature - When performing “Subclasstype name = new Subclasstype(params)”, it can be confusing to keep track of multiple parameters and its entries. Builder classes will have these methods fill out the details before outputting the new object.

Example uses - When creating objects such as License, Cars, or anything with multiple details that must be declared inside a constructor.

Structural Patterns

Patterns that describe relationships of classes to other classes.

Facade

Description - Hides the complexities of the system behind a

Cool Feature - If your code has methods that you're trying to hide to simplify the code/privacy concerns, you can create this pattern class to hide the complexities behind its method. I.e, methods in a method.

Example uses - When computers start up, a lot of things happen in the background. To simplify it for the user to read, the information is hidden/translated to something more legible.

Decorator

Description - Adding new abilities to its “user class” without altering the user class directly. Think of it as “getting into a vehicle without becoming a vehicle”

Cool Feature - Allows the “user class” to stay intact while allowing access to the methods of the “vehicle class”.

Example uses - In a casino game where the “patron” goes to a gameTable and becomes a “player”. As a player, it can now do things a player of the game can.

Adapter Pattern

Description - Works as a bridge between two incompatible interfaces.

Cool Feature - Helps reduce the amount of code that needs to be written and allows access of other interface methods via the bridge.

Example uses - When designing a game, there are three parts. The Engine, Game, and Player class. The Engine class can access the interfaces used by both the Game and Player class without having to write it in itself.

Behavioral Patterns

Patterns that are used to describe how other classes INTERACT with other classes.

Observer Pattern

Description - For objects that rely on the state of others, it is notified by this design pattern and can act accordingly.

Cool Feature - Objects can subscribe/unsubscribe to the state change of other objects and act accordingly.

Example uses - Example would be like a thermostat to a heater. If the thermostat reads the temperature and stopHeating at a certain temp, the heater will be notified and turnsOff.

Strategy Pattern

Description - A class behavior that can be changed at runtime.

Cool Feature - Allows the code to be flexible without having to rewrite anything or modify it.

Example uses - For example, if discounts are applied at a certain period, say during christmas, the christmasDiscount method will be selected based on the date entered during runtime.

Template Pattern

Description - Abstract classes are created

Cool Feature - If the core concepts are the same between similar classes, its methods can be established at the beginning and be modified by its subclasses as needed.

Example uses - A gameEngine class that has core methods such as “Starts the game, plays the game, ends the game” can be extended to all its subclasses.

Command Pattern

Description - Encapsulates in an object all the data required for performing a command.

Cool Feature - Decouples the objects that produce the commands from the consumers, simplifying the “behind the scenes”.

Example uses - Say you want to deposit money at a bank. A method is called to input the deposit, but within that method are a bunch of other methods to get the money from your “deposit” into your “account”.

Bloopers

Creational Patterns

Singleton - A single instance of a class that provides Global Access. Example is a Database.

Factory - A Class that aids in the creation of other classes. This class would have methods that perform the “new Object”, that way you don’t have to keep typing “new Object every time, you use a Factory.createObject instead.

Builder - Lets you construct complex objects step by step. If you need to set the instance variables of a license for example. Makes it easier so that you don’t have to declare information in a certain order, you’ll have methods to call with details of what to expect, and then it spits out the new Object.