[The 3 Types of Design Patterns All Developers Should Know (with code examples of each) (freecodecamp.org)](https://www.freecodecamp.org/news/the-basic-design-patterns-all-developers-need-to-know/)

**What are design patterns?**

Design patterns are design level solutions for recurring problems that we software engineers come across often. It’s not code - I repeat, not CODE. It’s like a description no how to tackle these problems and design a solution.

Using these patterns is considered good practice, as the design of the solution is quite tired and tested, resulting in higher readability of the final code. Design patterns are quite often created for and used by OOP languages, like Java, in which most of the examples from here on will be written.

**Types of design patterns**

1. Creational: These patterns are designed for class instantiation. They can be either class-creation patterns or object-creational patterns.
2. Structural: These patterns are designed with regard to a class’s structure and composition. The main goal of most of these patterns is to increase the functionality of the class(es) involved, without changing much of its composition.
3. Behavioral: These patterns are designed depending on how one class communicates with others.

In this post, we will go through one basic design pattern for each classified type.

Type 1: Creational – The singleton design Pattern

The singleton design pattern is a creational pattern, whose objective is to create only one instance of a class and to provide only one global access point to that object. One commonly used example of such a class in Java is Calendar, where you cannot make an instance of that class. It also uses its own getInstance() method to get the object to be used.

A class using the singleton design pattern will include.

Type2: Structural – The Decorator Design Pattern

I’m gonna give you a small scenario to give a better context to why and where you should use the Decorator Pattern.

Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class.

This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

We are demonstrating the use of decorator pattern via following example in which we will decorate a shape with some color without alter shape class.

Type3: Behavioral – The Command Design Pattern

A behavioral design pattern focuses on how classes and objects communicate with each other. The main focus of the command pattern is to inculcate a higher degree of loose coupling between involved parties.

Coupling is the way that two ( or more ) classes that interact with each other, well interact. The ideal scenario when these classes interact is that they do not depend heavily on each other. That’s loose coupling. So a better definition for loose coupling would be, classes that are interconnected, making the least use of each other.

The need for this pattern arose when requests needed to be sent without consciously knowing what you are asking for or who the receiver is.

In this pattern, the invoking class is decoupled from the class that actually performs an action. The invoker class only has the callable method execute, which runs the necessary command, when the client requests it.