DESIGN PATTERNS

Nusera Neha

WHAT ARE DESIGN PATTERNS?

When you start writing code, you may start to notice certain *"patterns"* in the style of the code you write.

STUDY HARD!

THERE ARE THREE TYPES OF DESIGN PATTERNS!

Each type corresponds to a different part of the structure of your code:

- CREATIONAL
- STRUCTURAL
- BEHAVIORAL

Creational patterns defer parts of object creation to another class. They are used to separate the system from how its objects are created

CREATIONAL PATTERNS

01

SINGLETON

Designs a class as non-static but accessing it gives the appearance of static operations

02

FACTORY

Methods that are responsible for creating and instantiating an object

03

BUILDER

Separates the construction of an object and allows the same construction to create different representations

SINGLETON

WHAT

Instantiation of a class object that is created once and accessed globally

WHY

Allows you to create a separate class and access it in a different class, allowing you access to all of its methods and fields

WHEN

Used in instances where more than one instance of a class is not required, but access to its methods is

FACTORY

WHAT

Methods & classes that instantiate and return an object. This can be described as a "factory"' that creates families of objects.

WHY

You can create new objects by calling the methods of a factory class rather than creating a new object all over again.

WHEN

When you want to create more types of an object but still retain the source code that has other types.

BUILDER

WHAT

Builds an object using its specified fields and allowing for customization based on needs.

WHY

Allows you to use only the features you want to use, but has the options for all features.

WHEN

When there's multiple uses for an object that doesn't require all the fields.

Structural patterns help reduce redundancies in code by identifying the relationships between structures.

STRUCTURAL PATTERNS

01
FACADE
Uses interfaces to limit rather than expand functionality of an encapsulated object

02
DECORATOR
Allows you to add behaviors across related objects at runtime

03
ADAPTER
Converts interfaces to another for its intended use

FACADE

WHAT

Creates a simple interface for a larger body of code to reduce user-error

WHY

Limits the access to methods so that only desired methods can be used and source code won't be affected

WHEN

When you want one interface to "manage" the rest without the user having to go into each specific class

DECORATOR

WHAT

Extends functionality of a class by adding to the object through extension and implementation of classes

WHY

You can have a base class that inherits methods from a parent class, giving it all the features of the inherited class

WHEN

When multiple objects have similar base functions that you want them to inherit from a parent class

ADAPTER

WHAT

Converts an interface of a class into another interface

WHY

Allows your to morph your source class to a specified need

WHEN

Moments when you need to convert your class type to fit a different need.

Behavioral patterns are concerned with algorithms and interactions between objects and classes.

BEHAVIORAL PATTERNS

01

OBSERVER

Registration and notification of a behavior

02

STRATEGY

A class that encapsulates an algorithm

03

TEMPLATE

Defers the steps of an algorithm to a subclass

04

COMMAND

Encapsulates a command request as a pattern

OBSERVER

WHAT

Observer class with an Observable class that interact with each other

WHY

Updates the observer class on any changes made to the observable class

WHEN

When you want to register any changes to a specific class to an observable class

STRATEGY

WHAT

Creates a concrete class per strategy and externalizes algorithms

WHY

It's cool because it eliminates conditional statements from your code.

WHEN

When you have specific recurring algorithms you want to be able to call without the clutter

TEMPLATE

WHAT

Creates a method of high freedome to be used by a method of a lower freedom

WHY

You have an encapsulating method that can be used at more restricted levels with less arguments than the original method

WHEN

When you have one with with arguments that have multiple uses

COMMAND

WHAT

Encapsulates all the details of a request in an object and passes it to another object to be executed

WHY

Decouples what is done from when it is done

WHEN

When you have multiple operations to execute and the object that holds the commands can be called to execute them

FINAL SUMMARY

01

There are three types of Design Patterns: Creational, Structural, Behavioral

02

Creational patterns deal with how classes and objects are created and accessed.

03

Structural patterns deal with classes and objects are composed from larger structures.

04

Behavioral patterns look at algorithms and how they are used and implemented .