#PYTHONTIPS

# EXPLORING DESIGN PATTERNS

## UNLOCKING THE SECRETS OF SOFTWARE REUSABILITY AND MAINTAINABILITY

@VEDANTSOLANKI

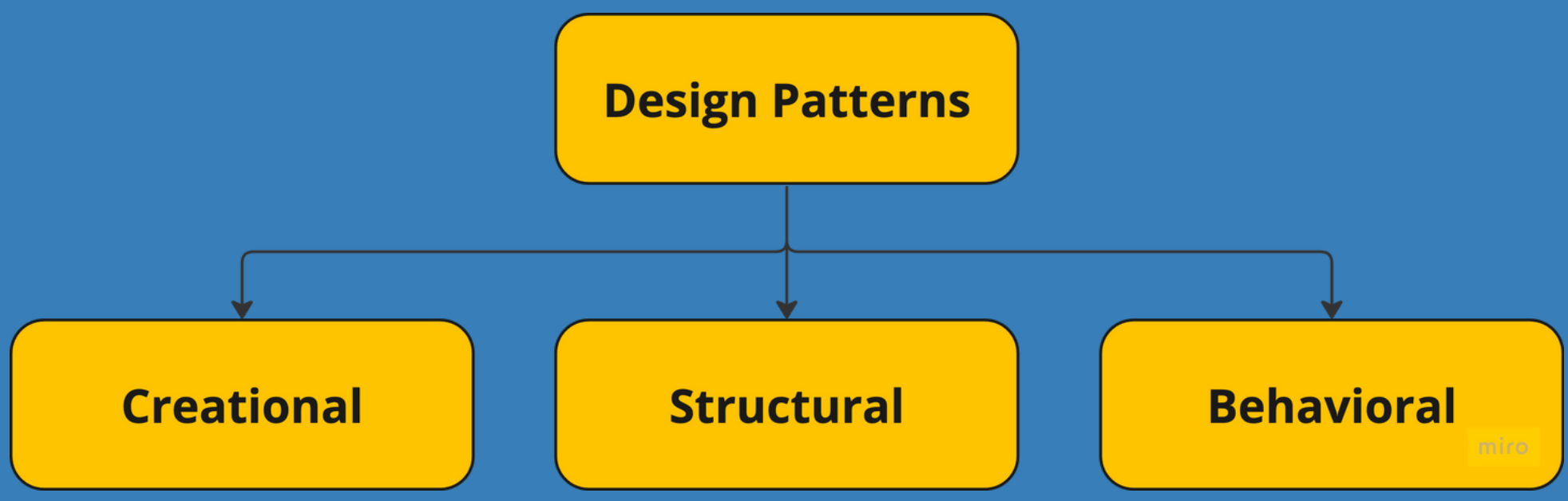
# WHAT ARE DESIGN PATTERNS ?

- Design patterns are proven solutions to recurring problems in software design.
- They provide a blueprint for structuring code to achieve certain goals.
- Why they matter : They improve code organization, maintainability, and scalability.



#PYTHONTIPS

# CATEGORIES




@VEDANTSOLANKI

# CREATIONAL PATTERNS

- This patterns focuses on object creation mechanisms.
- Example: Making sure you have only one copy of something, or building complex objects step by step
- Patterns :
  - Singleton:
    - Ensures a class has only one instance.
  - Factory:
    - Defines an interface for creating objects.



- Abstract Factory:
  - Provides an interface for creating families of related objects.
- Builder:
  - Separates the construction of a complex object from its representation.
- Prototype:
  - Creates new objects by copying an existing object.



#PYTHONTIPS

# STRUCTURAL PATTERNS

- This pattern helps to organize how different parts of code work together.
- Example: Making different systems talk to each other or adding extra features to objects.
- Patterns :
  - Adapter:
    - Allows the interface of an existing class to be used as another interface.
  - Bridge:
    - Separates an object's abstraction from its implementation.



- Composite:
  - Composes objects into tree structures to represent part-whole hierarchies.
- Decorator:
  - Attaches additional responsibilities to an object dynamically.
- Facade:
  - Provides a unified interface to a set of interfaces in a subsystem.
- Flyweight:
  - Minimizes memory usage or computational expenses of an object.



#PYTHONTIPS

# BEHAVIORAL PATTERNS

- This pattern define different ways objects  can interact and communicate.
- Example: Keeping track of changes in a system or letting objects change their behavior.
- Patterns :
  - Observer:
    - Defines a one-to-many dependency between objects.
  - Chain of Responsibility:
    - Passes the request along a chain of handlers.



- Strategy:
  - Defines a family of algorithms, encapsulates each, and makes them interchangeable.
- Command:
  - Encapsulates a request as an object, thereby allowing for parameterization of clients with queues, requests, and operations.
- Visitor:
  - Represents an operation to be performed on elements of an object structure.



- State:
  - Allows an object to alter its behavior when its internal state changes.
- Memento:
  - Captures and externalizes an object's internal state.
- Mediator:
  - Represents an operation to be performed on elements of an object structure.
- Iterator:
  - Provides a way to access the elements of an aggregate object sequentially.



In the next post, we'll embark on a journey to explore each design pattern.

With hands-on examples and practical scenarios, you'll gain the insights to use them effectively.



@TAG

# SOMEONE WHO WILL FIND THIS HELPFUL

## FOLLOW FOR MORE !

**Vedant Solanki**

**@vedantsolanki**