Design Patterns

Singleton Pattern

- Ensure that a class has just a **single instance**.
- Provide a global access point and **protects** the instance from being **overwritten** by other code

Builder Pattern

- Constructor with **optional parameters**.
- class Pizza {

Pizza(int size) { ... }

Pizza(int size, boolean cheese) { ... }

Pizza(int size, boolean cheese, boolean pepperoni) { ... }

Factory Pattern

- Use the Factory Method when you don’t know beforehand the **exact types** and dependencies of the objects your code should work with.
- Use the Factory Method when you want to provide users of your library or framework with a way to **extend** its **internal components**.

Abstract Factory Pattern

- Use the Abstract Factory when your code needs to work with various families of **related** products.
- Consider implementing the Abstract Factory when you have a class with a set of **Factory Methods**.

Adapter Pattern

- Use the Adapter class when you want to use some existing class, but its interface isn’t **compatible** with the rest of your code
- The Adapter pattern lets you create a **middle-layer** class that serves as a translator between your code and a legacy class

Bridge Pattern

- Use the Bridge pattern when you want to **divide and organize** a monolithic class that has several variants of some functionality.
- Use the pattern when you need to extend a class in several **independent dimensions**.

Decorator Pattern

- Use the Decorator pattern when you need to be able to assign extra **behaviors** to objects **at runtime** without breaking the code that uses these objects.
- Use the pattern when it’s **not** possible to **extend** an object’s behavior using inheritance.

Proxy Pattern

- Access control (protection proxy). This is when you want only **specific clients** to be able to use the service object
- **Lazy initialization** (virtual proxy). This is when you have a heavyweight service object that wastes system resources by being always up, even though you only need it from time to time.
- Caching request results

Observer Pattern

- Use the Observer pattern when changes to the **state of one** object may require **changing other** objects.
- Use the pattern when some objects in your app must **observe** others, but only for a limited time or in specific cases.

Mamento Pattern

- Use the Memento pattern when you want to produce snapshots of the object's **state** to be able to restore a **previous state** of the object.
- Use the pattern when direct access to the object's fields/getters/setters violates its encapsulation.

Iterator Pattern

- Use the Iterator pattern when your collection has a complex data structure under the hood, but you want to **hide** its **complexity** from clients.
- Use the pattern to reduce **duplication** of the traversal code across your app.

- https://refactoring.guru/
- https://www.youtube.com/@geekific
- https://www.youtube.com/@programmingwithmosh
- https://www.youtube.com/@KeertiPurswani
- Design Patterns: Elements of Reusable Object-Oriented Software