**Patterns -** *Need to grasp the classes and interfaces used for them, enough to argue for when or not to use them.*

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

* **Singleton**:
  + The Singleton pattern ensures that a class has only one instance and provides a global point of access to it. This is useful when exactly one object is needed to coordinate actions across the system, such as a configuration manager or a connection pool.
  + **Classes/Interfaces involved**: A single class that is responsible for creating its own unique instance and ensuring no other instance can be created.
  + **When to Use:** When there must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
* **Factory method:** 
  + The Factory Method pattern defines an interface for creating an object, but lets subclasses alter the type of objects that will be created. This pattern is particularly useful when a class cannot anticipate the class of objects it needs to create.
  + **Classes/Interfaces Involved:** Creator (an abstract class with a method that returns objects of a specific type). ConcreteCreator (subclasses of Creator that implement the factory method to create specific products)
  + **When to Use:** When a class wants its subclasses to specify the objects it creates. When a class can't foresee what kind of class of objects it needs to create.
* **Strategy:** 
  + The Strategy pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
  + **Classes/Interfaces Involved:** Context (uses a Strategy instance). Strategy (interface defining an action). ConcreteStrategy (implements the Strategy interface with specific algorithms)
  + **When to Use:** When you have multiple classes that differ only in their behavior. Strategies provide a way to configure a class with one of many behaviors. When you need to dynamically switch algorithms used within an object at runtime.
* **Observer:**
* **Iteration:**
* **Memento:**
* **Composite:**
* **Prototype:**

**Bra sida:**

<https://refactoring.guru/design-patterns/catalog>