Creational Patterns

* [**Abstract Factory Pattern**](https://csharpdesignpatterns.codeplex.com/wikipage?title=Abstract%20Factory%20Pattern&referringTitle=Home)**:** Create instances of classes belonging to different families
* [**Builder Pattern**](https://csharpdesignpatterns.codeplex.com/wikipage?title=Builder%20Pattern&referringTitle=Home)**:** Separate representation and object construction
* [**Factory Method Pattern**](https://csharpdesignpatterns.codeplex.com/wikipage?title=Factory%20Method%20Pattern&referringTitle=Home)**:** Create instances of derived classes
* [**Prototype Pattern**](https://csharpdesignpatterns.codeplex.com/wikipage?title=Prototype%20Pattern&referringTitle=Home)**:** Clone or copy initialized instances
* [**Singleton Pattern**](https://csharpdesignpatterns.codeplex.com/wikipage?title=Singleton%20Pattern&referringTitle=Home)**:** Class with only one single possible instance

Structural Patterns

* [**Adapter Pattern**](https://csharpdesignpatterns.codeplex.com/wikipage?title=Adapter%20Pattern&referringTitle=Home)**:** Match interfaces of classes with different interfaces
* [**Bridge Pattern**](https://csharpdesignpatterns.codeplex.com/wikipage?title=Bridge%20Pattern&referringTitle=Home)**::** Separate implementation and object interfaces
* **Composite:** Simple and composite objects tree
* **Decorator:** Dynamically add responsibilities to objects
* **Facade:** Class that represents subclasses and subsystems
* **Flyweight:** Minimize memory usage by sharing as much data as possible with similar objects
* **Proxy:** Object that represents another object

Behavioral Patterns

* **Chain of Responsibility:** Pass requests between command and processing objects within a chain of objects
* **Command:** Encapsulate a method call as an object containing all necessary information
* **Interpreter:** Include language elements and evaluate sentences in a given language
* **Iterator:** Give sequential access to elements in a collection
* **Mediator:** Encapsulates and simplifies communication between objects
* **Memento:** Undo modifications and restore an object to its initial state
* **Observer:** Notify dependent objects of state changes
* **State:** Change object behavior depending on its state
* **Strategy:** Encapsulate algorithms within a class and make them interchangeable
* **Template Method:** Define an algorithm skeleton and delegate algorithm steps to subclasses so that they may be overridden
* **Visitor:** Add new operations to classes without modifying them