**Design Patterns in C#**

**What is design Pattern in c#?**

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Design Patterns are categorize into three types and they are: **CSB**

1. Creational Design Patterns. 🡪 Deal with the process of objects creation
2. Structural Design Patterns. 🡪 Deal with the composition of objects structures. Composition= Configuration/Alignment
3. Behavioral Design Patterns. 🡪 Deal with the process of communication, managing relationships, and responsibilities between objects.

**What are Creational Design Patterns?**  
These patterns deal with the process of objects creation. The flowing are the different types of Creational Design patterns.

1. Abstract Factory Pattern: - Create instances of several classes belonging to different families.
2. Factory Pattern: - Create instances of derived classes.
3. Builder Pattern: - Separates an object construction from its representation.
4. Lazy Pattern: - Create a duplicate object or clone of the object.
5. Prototype Pattern: - Specify the kind of objects to create using a prototypical instance, and create new objects by copying this prototype.
6. Singleton Pattern: - Ensures that a class can have only one instance.

**What are Structural Design Patterns?**  
These patterns deal with the composition of objects structures. The flowing are the different types of Structural Design patterns.

1. Adapter Pattern: - Interfaces of classes vary depending on the requirement.
2. Bridge Pattern: - Class level abstraction is separated from its implementation.
3. Composite Pattern: - Individual objects & a group of objects are treated similarly in this approach.
4. Decorator Pattern: - Functionality is assigned to an object.
5. Facade Pattern: - A common interface is created for a group of interfaces sharing a similarity.
6. Flyweight Pattern: - The concept of sharing a group of small sized objects.
7. Proxy Pattern: - When an object is complex and needs to be shared, its copies are made. These copies are called the proxy objects.

**What are Behavioral Design Patterns?**  
These patterns deal with the process of communication, managing relationships, and responsibilities between objects. The flowing are the different types of Behavioral Design patterns.

1. Chain Or Responsibilities Pattern: - In this pattern, objects communicate with each other depending on logical decisions made by a class.
2. Command Pattern: - In this pattern, objects encapsulate methods and the parameters passed to them.
3. Observer Pattern: - Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
4. Interpreter Pattern: - A way to include language elements in a program.
5. Iterator Pattern: - Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
6. Mediator Pattern: - Define an object that encapsulates how a set of objects interact. In other words, it defines simplified communication between classes.
7. Memento Pattern: - Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.
8. State Pattern: - Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
9. Strategy Pattern: - Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
10. Visitor Pattern: - Defines a new operation to a class without change.
11. Template Method Pattern: - Defer the exact steps of an algorithm to a subclass.

**Total: 6+7+11= 24**

Creational Design Patterns🡪 6

Structural Design Patterns🡪 7

Behavioral Design Patterns🡪 11

**From sourcemaking.com**

[**Creational design patterns**](https://sourcemaking.com/design_patterns/creational_patterns)

These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

[](https://sourcemaking.com/design_patterns/abstract_factory)

1. [**Abstract Factory**](https://sourcemaking.com/design_patterns/abstract_factory)  
   Creates an instance of several families of classes
2. [**Builder**](https://sourcemaking.com/design_patterns/builder)  
   Separates object construction from its representation
3. [**Factory Method**](https://sourcemaking.com/design_patterns/factory_method)  
   Creates an instance of several derived classes
4. [**Object Pool**](https://sourcemaking.com/design_patterns/object_pool)  
   Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
5. [**Prototype**](https://sourcemaking.com/design_patterns/prototype)  
   A fully initialized instance to be copied or cloned
6. [**Singleton**](https://sourcemaking.com/design_patterns/singleton)  
   A class of which only a single instance can exist

### [Structural design patterns](https://sourcemaking.com/design_patterns/structural_patterns)

These design patterns are all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

[](https://sourcemaking.com/design_patterns/decorator)

**[](https://sourcemaking.com/design_patterns/proxy)**

1. [**Adapter**](https://sourcemaking.com/design_patterns/adapter)  
   Match interfaces of different classes
2. [**Bridge**](https://sourcemaking.com/design_patterns/bridge)  
   Separates an object’s interface from its implementation
3. [**Composite**](https://sourcemaking.com/design_patterns/composite)  
   A tree structure of simple and composite objects
4. [**Decorator**](https://sourcemaking.com/design_patterns/decorator)  
   Add responsibilities to objects dynamically
5. [**Facade**](https://sourcemaking.com/design_patterns/facade)  
   A single class that represents an entire subsystem
6. [**Flyweight**](https://sourcemaking.com/design_patterns/flyweight)  
   A fine-grained instance used for efficient sharing
7. [**Private Class Data**](https://sourcemaking.com/design_patterns/private_class_data)  
   Restricts accessor/mutator access
8. [**Proxy**](https://sourcemaking.com/design_patterns/proxy)  
   An object representing another object

### [Behavioral design patterns](https://sourcemaking.com/design_patterns/behavioral_patterns)

These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

**[](https://sourcemaking.com/design_patterns/state)**

[](https://sourcemaking.com/design_patterns/interpreter)

1. [**Chain of responsibility**](https://sourcemaking.com/design_patterns/chain_of_responsibility)  
   A way of passing a request between a chain of objects
2. [**Command**](https://sourcemaking.com/design_patterns/command)  
   Encapsulate a command request as an object
3. [**Interpreter**](https://sourcemaking.com/design_patterns/interpreter)  
   A way to include language elements in a program
4. [**Iterator**](https://sourcemaking.com/design_patterns/iterator)  
   Sequentially access the elements of a collection
5. [**Mediator**](https://sourcemaking.com/design_patterns/mediator)  
   Defines simplified communication between classes
6. [**Memento**](https://sourcemaking.com/design_patterns/memento)  
   Capture and restore an object's internal state
7. [**Null Object**](https://sourcemaking.com/design_patterns/null_object)  
   Designed to act as a default value of an object
8. [**Observer**](https://sourcemaking.com/design_patterns/observer)  
   A way of notifying change to a number of classes
9. [**State**](https://sourcemaking.com/design_patterns/state)  
   Alter an object's behavior when its state changes
10. [**Strategy**](https://sourcemaking.com/design_patterns/strategy)  
    Encapsulates an algorithm inside a class
11. [**Template method**](https://sourcemaking.com/design_patterns/template_method)  
    Defer the exact steps of an algorithm to a subclass
12. [**Visitor**](https://sourcemaking.com/design_patterns/visitor)  
    Defines a new operation to a class without change

Ref:  
<http://www.c-sharpcorner.com/blogs/design-patterns-in-c-sharp1>  
<https://sourcemaking.com/design_patterns>