**Design Patterns C# .NET - Summary**

Creational Patterns

**Builder**: (piece wise construction of an object)

-Making a separate component when object construction gets too complicated

-Can have multiple cooperating sub-builders through a base class

-Often has a fluent interface

**Factories**: (wholesale construction of objects)

-Like a constructor but more expressive (can have a name that is different from the containing class)

-Can be a inner class or outside class (of the object that is being constructed – if its inner it has the benefits of accessing private members)

**Prototype:** (creation of an object from an existing object)

-Deep copy or copy through serialization

**Singleton:** (only a single instance of a particular object exists)

-If you use thread safe and Lazy<T> - it has less problems

-Extract singleton´s interface or dependency injection instead of hardcoding it(cause with hardcoding it is more difficult to remove later if needed)

Structural Patterns

**Adapter:** (Converts the interface you get to the interface you need)

-If your adapter generates objects or data more than once, maybe you should use cache to store this information

**Bridge:** (Decouple abstraction from implementation)

-Take you a part of the hierarchy in a separate interface and refer to it via an ordinary field

**Composite:** (allows you to work with individual and composite objects uniformly)

**Decorator:** (Allows you to attach additional responsibilities to objects)

-Useful when inheritance is not available (like when using sealed classes), you could build an object that aggregates the object from sealed classes, copying its functionality, and then you can add additional functionality on top of that.

-You can also have decorators of decorators (decoratorception)

**Façade:** (Provides a single unified interface over a set of classes/systems

**Flyweight:** (efficiently supports many similar objects, it’s a space optimization technique)

-links that point to the same object (shown in various places)

**Proxy:** (The creation of an object that forwards the calls to the real object while performing additional functions)

-Common uses: protection proxy (checking if the user has access rights), remoting proxies (object does not live on the system), communication, logging

-Dynamic proxy: constructing it at runtime, as opposed to compile time. Upside: you don’t have to manually replicate the target.

Behavioral Patterns

**Chain of Responsibility:** (allow components to process information/events in a chain)

-Each element in the chain refers to the next element or make a list and go through it (foreach statement)

**Command** (encapsulates a request into a separate object)

-Good for audit, replay, undo/redo

-Part of Command Query Separation (CQS) – QUERYS can be commands also

**Interpreter** (Transforms textual input into object-oriented structures

-Used by interpreters, compilers, static analysis tools

**Iterator** (Provides an interface for accessing elements of an aggregate object)

-IEnumerable<T> should be used in 99% of cases

**Mediator** (Provides mediation services between two objects)

-Examples: Message passing, chat room

-With dependency injection, it is possible to insert the mediator as a constructor argument of (for example) a base class of the entire hierarchy and every member that inherits it will initialize the mediator

**Memento** (amazing movie and also yields tokens that represent states of the system)

-Tokens are typically read only (don’t allow manipulation)

**Observer** (enables a subscriber to register with and receive notifications from a provider)

-Built in into C# with the event keyword

-Additional support provided for properties, collections and observable streams

**State** (Modeling systems by having one of a possible number of states and transitions between this states)

-This system in the description is usually called a state machine

-Special frameworks exists to orchestrate state machines

**Strategy and Template** (They define an algorithm blueprint/placeholder)

-Strategy uses composition, template uses inheritance

**Visitor** (Adds functionality to existing classes through double dispatch)

- separate algorithms from the objects on which they operate.

-double dispatch helps to execute the proper method on an object without cumbersome conditionals