Introduction to creational patterns

One group of such design patterns is called creational design patterns.

The goal of these patterns is to make you write more flexible and reusable code using simple and fast solutions for common problems.

What is a creational pattern?

Creational design patterns are design patterns that focus on making objects through a mechanism for a required application.

The result of using a mechanism could be the simplification of your code with a quick solution to the most common problems.

This pattern separates your system's main functions from the process of object creation.

The concept of creational design patterns allows us to develop more flexible systems in terms of object creation.

Using creational patterns

Creational patterns are one of the ways to get rid of hard coding.

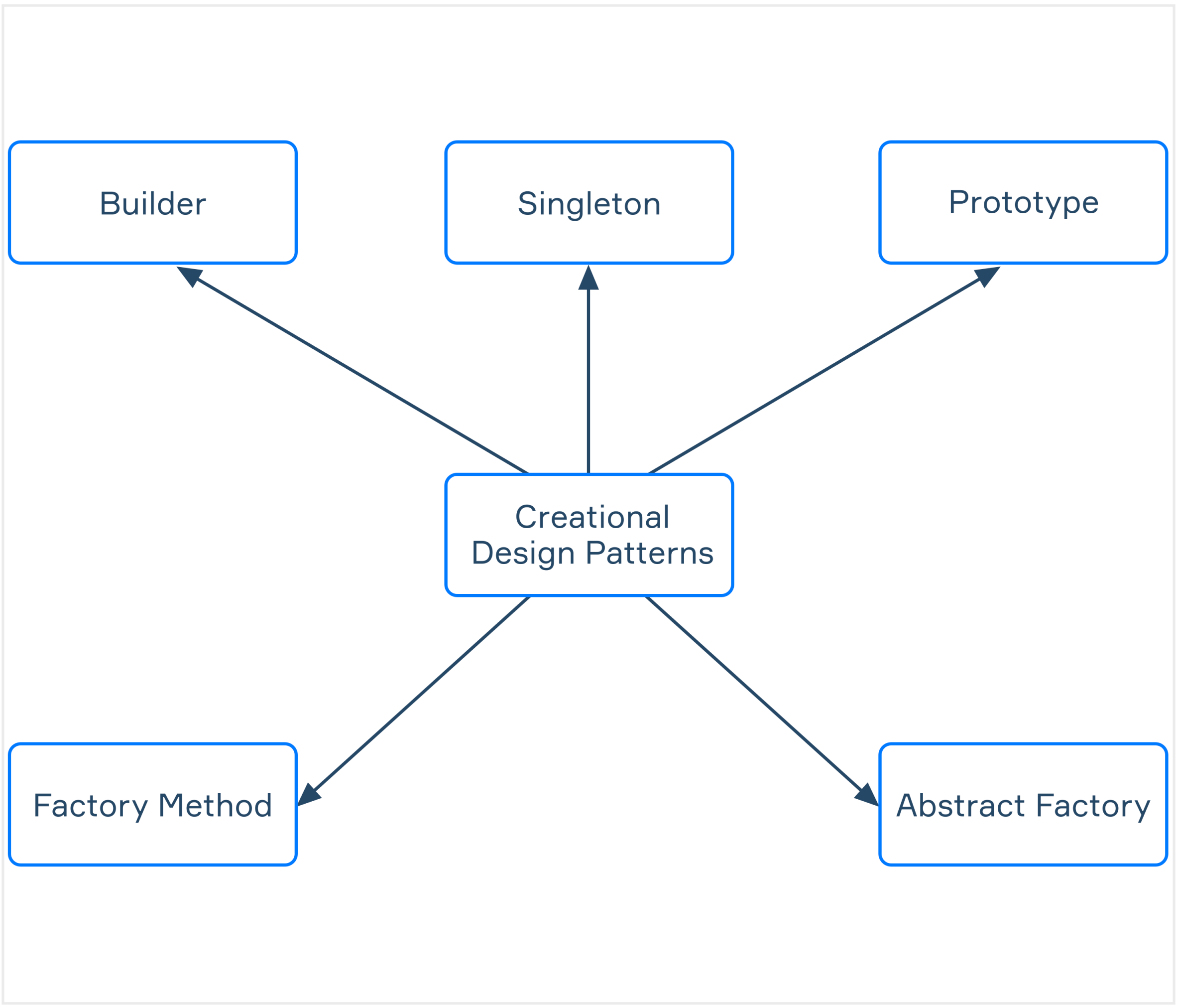
It allows us to make a system that does not depend on the way its objects are created.

Here are a few situations when you should apply creational patterns:

- Designing a set of related objects that are to be used together.
- Hiding the implementation of a class library, revealing only its interfaces.
- Constructing different representations of independent complex objects.
- A class delegates its subclass to implement the object it creates.
- There must be a single instance and the client can access this instance at all times.

Mainly it is their unjustified usage and their occasional inefficiency. So you should keep in mind the situations described above to get the most benefits out of creational design patterns.

But these design patterns could also limit your creativity and make you think that they are multi-tools that can solve any problem. So you shouldn't always solely rely on them.



Types of creational patterns

- **Factory method** is a pattern that creates an interface or abstract class for object creation that also allows for a modification of object creation in subclasses. You could use this pattern when you don't know beforehand the exact types and dependencies of the objects your code should work with.

- **Singleton** is a pattern that creates only one instance of an object while providing a point of global access to this instance. This pattern should be implemented when the class in your program should have just a single instance available to all clients.

- **Prototype** is a pattern based on the concept of copying an existing class for the creation of a new one. Use this to reduce the number of subclasses that only differ in the way they initialize their objects.
- **Builder** is a pattern that lets you design complex objects using simple objects using a step-by-step approach. This allows you to create different modified variations of the same object.

- **Abstract factory** is a pattern that works with a method that creates other factories. Its factories could further be used for a factory method.