Creational Design Patterns

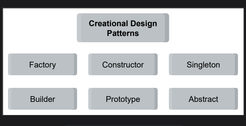
Get introduced to creational design patterns and learn when to use them.

We'll cover the following

Introduction to creational design patterns

In this lesson, we will discuss creational design patterns. **Creational design patterns** deal with object creation mechanisms. As the name implies, these patterns provide optimized object creation techniques. They help cater to the design and complexity problems that might occur when using the basic approach. They also help control the creation of objects.

The chart below shows the patterns that fall under this category:



Creational design patterns

Factory pattern

The **Factory pattern** is a creational pattern that provides a template that can be used to create objects. It is used in complex situations where the type of the object required varies and needs to be specified in each case.

It does not use the `new` keyword directly to instantiate objects. This means that it does not explicitly require the use of a constructor to create objects. Instead, it provides a generic interface that delegates the object creation responsibility to the corresponding subclass.

Constructor pattern

The **Constructor pattern**, as the name defines, is a class-based pattern that uses the constructors present in the class to create specific types of objects.

Singleton pattern

The **Singleton pattern** is a type of design pattern that restricts the instantiation of a class to a single object. This allows the class to create its instance the first time it is instantiated. However, on the next try, the existing instance of the class is returned. No new instance is created.

Builder pattern

The Builder pattern is a type of a creational design pattern that helps in building complex objects using simpler objects. It provides a flexible and step-by-step approach towards making these objects. It also shields the representation and process of creation.

Prototype pattern

The **Prototype pattern** is used to instantiate objects with some default values using an existing object. It clones the object and provides the existing properties to the cloned object using prototypal inheritance.

In **prototypal inheritance**, a prototype object acts as a blueprint from which other objects inherit when the constructor instantiates them. Hence, any properties defined on the prototype of a constructor function will also be present in the cloned object it creates.

Abstract pattern

We use the Factory pattern to create multiple objects from the same family without having to deal with the creation process. The **Abstract pattern** is similar. The difference is that it provides a constructor to create families of related objects. It is abstract, which means that it does not specify concrete classes or constructors.

When to use creational design patterns?

Creational Design Patterns	When to use
Factory pattern	• When the type of objects required cannot be anticipated beforehand. • When multiple objects that share similar characteristics need to be crea • When you want to generalize the object instantiation process, since the complex in nature.
Constructor pattern	• You can use it when you want to create multiple instances of the same t instances can share methods but can still be different. • It can be useful in the Libraries and Plugins design.
Singleton pattern	• The Singleton pattern is mostly used in cases where we want a single ob actions across a system. • **Services** can be singleton since they store the state, configuration, and resources. Therefore, it makes sense to have a single instance of a servic • **Databases** such as MongoDB utilize the Singleton pattern when it come connections. • **Configurations** are used if there is an object with a specific configuratio a new instance every time that configuration object is needed.
Builder pattern	• You can use this design pattern when building apps that require you to objects. It can help you hide the construction process of building these • A good example would be a DOM, where we might need to create plenty attributes. The construction process can get quite messy if we are buildi object. In cases like these, the Builder pattern can be used.
Prototype pattern	• To eliminate the overhead of initializing an object. • When we want the system to be independent about how the products ir • When creating objects from a database whose values are copied to the c
Abstract pattern	• Applications requiring the reuse or sharing of objects. • Applications with complex logic because they have multiple families of need to be used together. • When we require object caching. • When the object creation process is to be shielded from the client.

Let's look at the structural design patterns in the next lesson.

Back

Classification of Design Patterns

Complete

Next

Structural Design Patterns