Creational Patterns

Wednesday, November 23, 2022 9:13 AM

The Creational Design Patterns category are the design patterns which deal with **creating objects**.

Builder

Helps to **separate the construction from the representation** of a complex object. By utilizing the same construction process, the program can create different representations.
Builder is a creational design pattern that lets you **construct complex object step by step**. The pattern allows you to produce different types and representations of an object using the same construction code.

When to implement?

When you need to **create a complex object which has different representations**, you would need to implement a solid Builder Design Pattern.

Factory

One of the most used in real world applications. By using it, **objects are created without exposing the creation logic** to the client by **using a common interface**.
Factory defines an **interface for creating an object**, but **let subclasses decide which class to instantiate**. The Factory methods lets a class defer instantiation it uses to subclasses.

When to implement?

Is used when you have classes which **don't know what exact sub-classes it has to create**. Example: it happens in real world application when the Product implementation tends to change over time but the Client needs to remain unchanged. Vehicle example

Prototype

Programs are able to **create a new instance** of a class **from an existing object**. By doing so, it's **easier to create an instance of a complex object** that maybe costly to create a new one.
Prototype specifies the kinds of objects to create using a prototype instance, and **create new objects by copying this prototype**.

When to implement?

For example, in a car manufacturing industry, it would be difficult to produce a car from scratch. Rather than that, car companies usually make a **clone from a previous car model** and **enhance it** to produce a **new and better car model** every year. The same is apply to your programs if there is a complex class which is costly (in term of time and space) to be newly initiated.

Singleton

Is a way to **control the number of instance of a class to be only one**. By doing so, the application is able to provide a **global access point to that single instance**.
Singleton lets you **ensure** that a **class has only one instance**, while **providing** a **global access point** to

this instance.

When to implement?

When your applications have some components which only make sense to have **one instance in the system**, such as a Database Repository or an Factory Object.