Prototype Design Pattern

Difficulty Level : Medium • Last Updated : 25 Dec, 2022

Prototype allows us to hide the complexity of making new instances from the client. The concept is to copy an existing object rather than creating a new instance from scratch, something that may include costly operations. The existing object acts as a prototype and contains the state of the object. The newly copied object may change same properties only if required. This approach saves costly resources and time, especially when object creation is a heavy process.
The prototype pattern is a creational design pattern. Prototype patterns are required, when object creation is time consuming, and costly operation, so we create objects with the existing object itself. One of the best available ways to create an object from existing objects is the **clone() method**. Clone is the simplest approach to implement a prototype pattern. However, it is your call to decide how to copy existing object based on your business model.

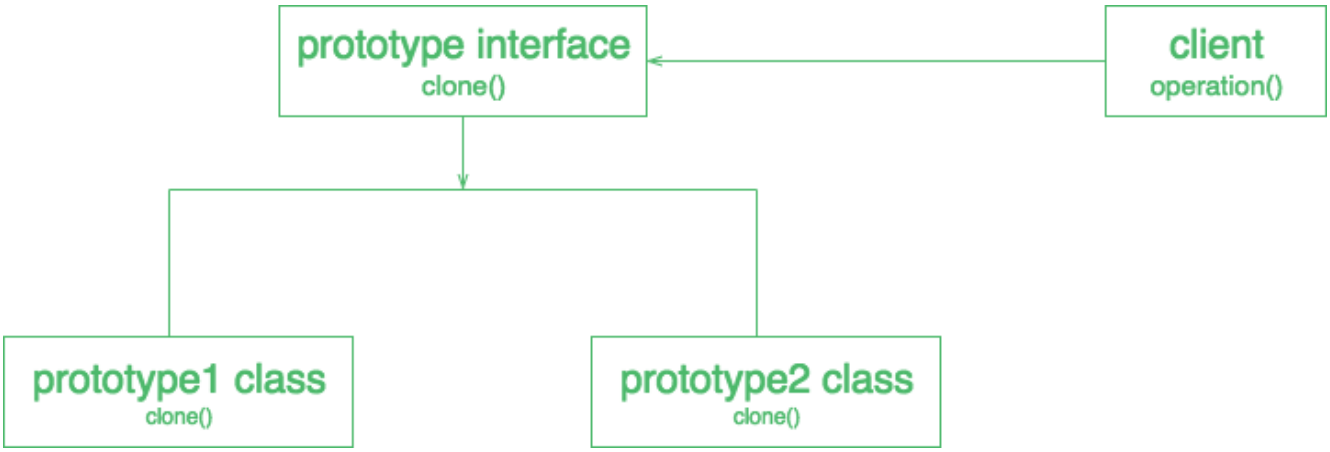
Prototype Design Participants

1) **Prototype** : This is the prototype of an actual object.
2) **Prototype registry** : This is used as a registry service to have all prototypes accessible using simple string parameters.
3) **Client** : Client will be responsible for using registry service to access prototype instances.

When to use the Prototype Design Pattern

When a system should be independent of how its products are created, composed, and represented and
When the classes to instantiate are specified at run-time.
For example,
1) By dynamic loading or To avoid building a class hierarchy of factories that parallels the class hierarchy of products or
2) When instances of a class can have one of only a few different combinations of state. It may be more convenient to install a corresponding number of prototypes and clone them rather than instantiating the class manually, each time with the appropriate state.

The UML Diagram of the Prototype Design Pattern



Java

```
// A Java program to demonstrate working of
// Prototype Design Pattern with example
// of a ColorStore class to store existing objects.
```

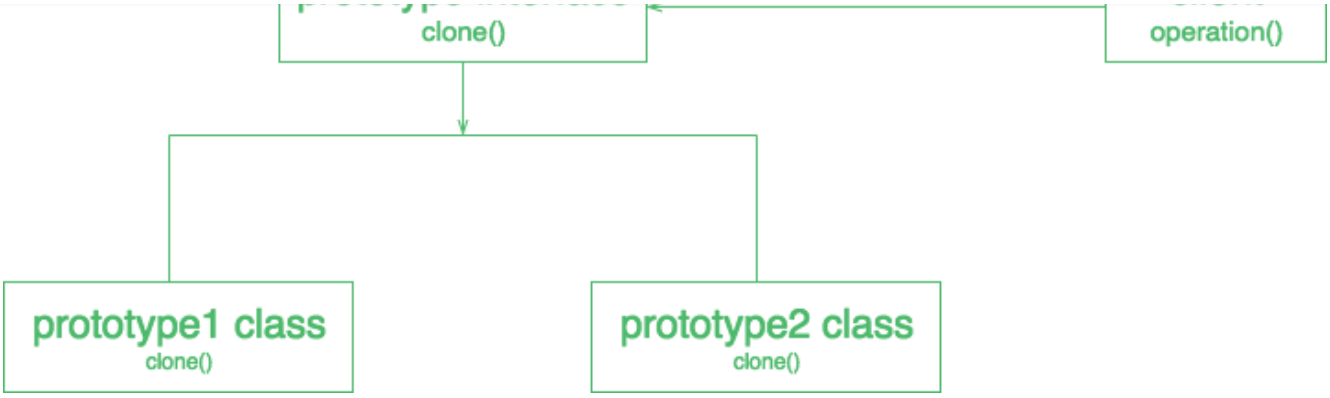
```
abstract class Color implements Cloneable
{

    protected String colorName;

    abstract void addColor();

    public Object clone()
    {
        Object clone = null;
        try
        {
            clone = super.clone();
        }
        catch (CloneNotSupportedException e)
        {
            e.printStackTrace();
        }
        return clone;
    }
}

class blueColor extends Color
{
    public blueColor()
    {
        this.colorName = "blue";
    }

    @Override
    void addColor()
    {
        System.out.println("Blue color added");
    }

}

class blackColor extends Color{

    public blackColor()
    {
        this.colorName = "black";
    }

    @Override
    void addColor()
    {
        System.out.println("Black color added");
    }
}
```

```
    {
        colorMap.put("blue", new blueColor());
        colorMap.put("black", new blackColor());
    }

    public static Color getColor(String colorName)
    {
        return (Color) colorMap.get(colorName).clone();
    }
}


// Driver class
class Prototype
{
    public static void main (String[] args)
    {
        ColorStore.getColor("blue").addColor();
        ColorStore.getColor("black").addColor();
        ColorStore.getColor("black").addColor();
        ColorStore.getColor("blue").addColor();
    }
}
```

Output:

```
Blue color added
Black color added
Black color added
Blue color added
```

UML diagram of example:



Advantages of Prototype Design Pattern

- **Adding and removing products at run-time –** Prototypes let you incorporate a new concrete product class into a system simply by registering a prototypical instance with the client. That's a bit more flexible than other creational patterns, because a client can install and remove prototypes at run-time.

- **Specifying new objects by varying values –** Highly dynamic systems let you define new behavior through object composition by specifying values for an object's variables and not by defining new classes.

- **Specifying new objects by varying structure –** Many applications build objects from parts and subparts. For convenience, such applications often let you instantiate complex, user-defined structures to use a specific subcircuit again and again.

- **Reduced subclassing –** Factory Method often produces a hierarchy of Creator classes that parallels the product class hierarchy. The Prototype pattern lets you clone a prototype instead of asking a factory method to make a new object. Hence you don't need a Creator class hierarchy at all.

Disadvantages of Prototype Design Pattern

- Overkill for a project that uses very few objects and/or does not have an underlying emphasis on the extension of prototype chains.

can be difficult when their internals include objects that don't support copying or have circular references.

Further Read: Prototype Method in Python

This article is contributed by **Saket Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

15

Related Articles

1. Singleton Design Pattern | Implementation
2. The Decorator Pattern | Set 2 (Introduction and Design)
3. Decorator Pattern | Set 3 (Coding the Design)
4. Flyweight Design Pattern
5. Singleton Design Pattern | Introduction
6. Java Singleton Design Pattern Practices with Examples
7. Proxy Design Pattern
8. Composite Design Pattern
9. Mediator design pattern
10. Template Method Design Pattern

Start Your Coding Journey Now!

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : Pushpender007, twinshu_parmar

Article Tags : Design Pattern, System Design

Practice Tags : System Design

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org