Prototype Design Pattern

Last Updated : 03 Jan, 2025

The Prototype Design Pattern is a creational pattern that enables the creation of new objects by copying an existing object. Prototype allows us to hide the complexity of making new instances from the client. The existing object acts as a prototype and contains the state of the object.

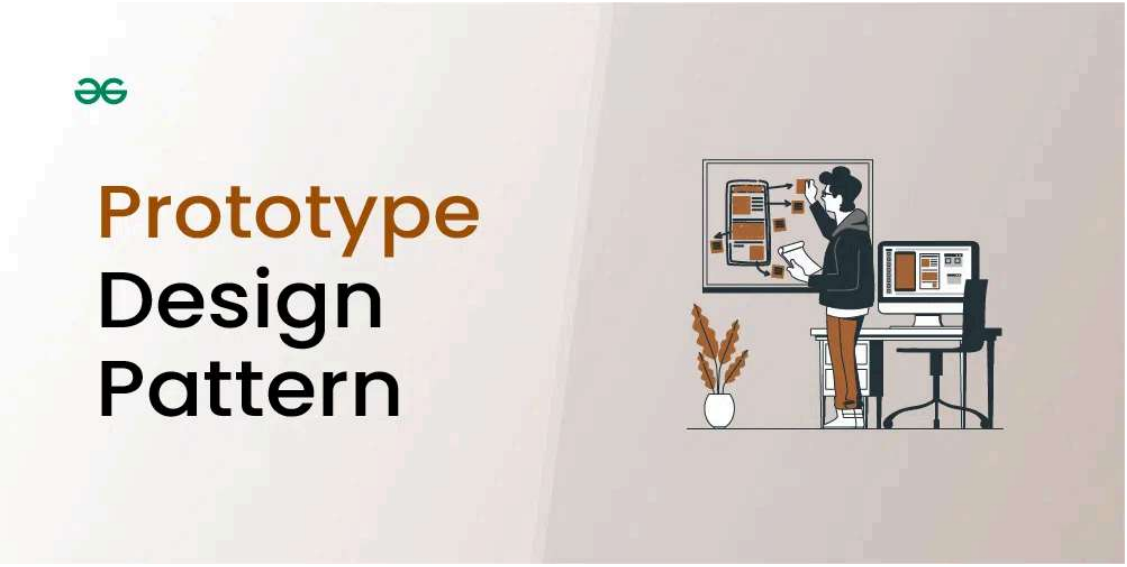


Table of Content

What is Prototype Design Pattern?

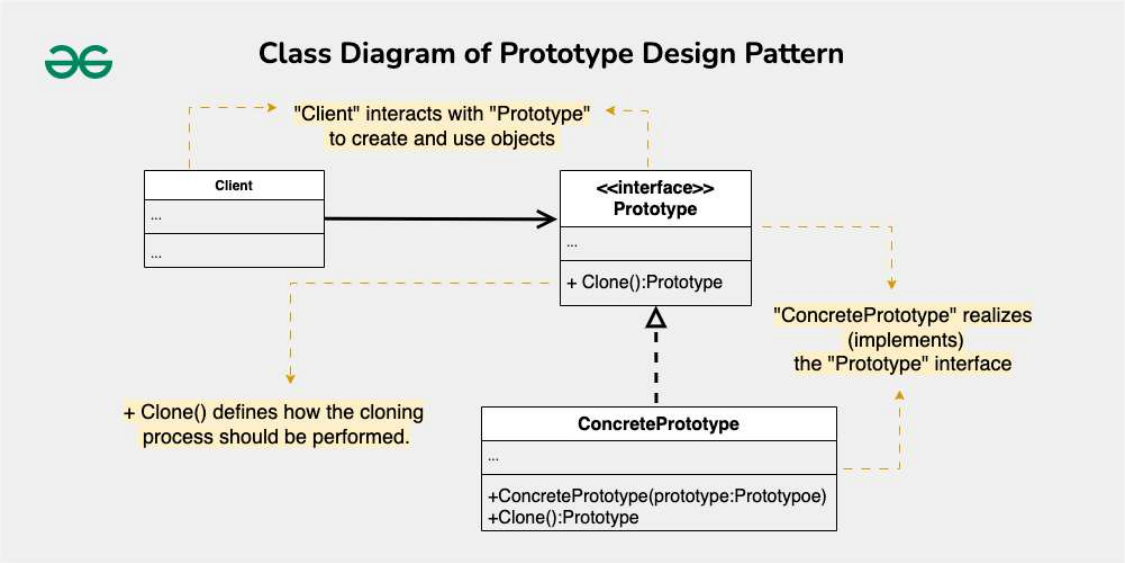
The prototype pattern is a creational design pattern which is required when object creation is a time-consuming, and costly operation, so we create objects with the existing object itself to by copying the existing ones.

- The newly copied object may change the same properties only if required. This approach saves costly resources and time, especially when object creation is a heavy process.
- One of the best available ways to create an object from existing objects is the **clone() method**. Clone is the simplest approach to implementing a prototype pattern. However, it is your call to decide how to copy existing objects based on your business model.

Suppose a user creates a document with a specific layout, fonts, and styling, and wishes to create similar documents with slight modifications.

Components of Prototype Design Pattern

The Prototype Design Pattern's components include the prototype interface or abstract class, concrete prototypes and the client code, and the `clone` method specifying cloning behavior. These components work together to enable the creation of new objects by copying existing ones.



- **Prototype Interface or Abstract Class**
    - This defines the method for cloning objects and sets a standard that all concrete prototypes must follow. Its main purpose is to serve as a blueprint for creating new objects by outlining the cloning contract.
    - It includes a clone method that concrete prototypes will implement to create copies of themselves.

- **Concrete Prototype**
    - This class implements the prototype interface or extends the abstract class. It represents a specific type of object that can be cloned.
    - The Concrete Prototype details how the cloning process should work for instances of that class and provides the specific logic for the clone method.
- **Client**
    - The Client is the code or module that requests new object creation by interacting with the prototype.
    - It initiates the cloning process without needing to know the specifics of the concrete classes involved.
- **Clone Method**
    - This method is declared in the prototype interface or abstract class and outlines how an object should be copied.
    - Concrete prototypes implement this method to define their specific cloning behavior, detailing how to duplicate the object's internal state to create a new, independent instance.

Prototype Design Pattern example in Java

Let's understand how prototype design pattern work with the help of an example:

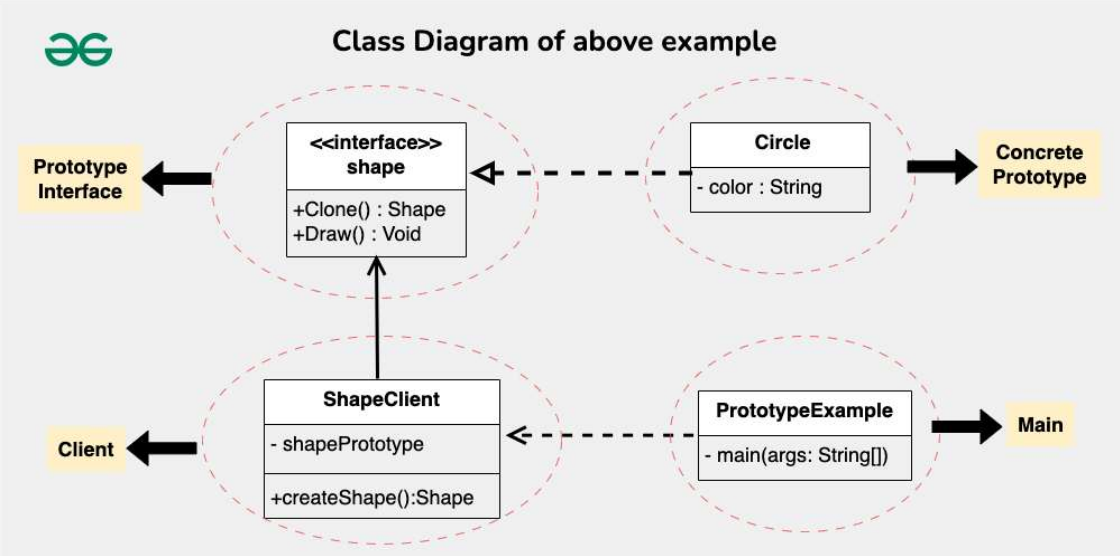
> *Imagine you're working on a drawing application, and you need to create and manipulate various shapes. Each shape might have different attributes like color or size. Creating a new shape class for every variation becomes tedious. Also, dynamically adding or removing shapes during runtime can be challenging.*

Let's understand how Prototype Design Pattern will help to solve this problem:

- The Prototype Design Pattern helps in managing variations of shapes efficiently, promoting flexibility in shape creation, and simplifying the

process of adding or removing shapes at runtime.

- The Prototype Design Pattern addresses this by introducing a prototype interface (`Shape`) that declares common methods for cloning and drawing shapes.
- Concrete prototypes like `Circle` implement this interface, providing their unique cloning logic.
- The `ShapeClient` acts as a user, utilizing the prototype to create new shapes.



1. Prototype Interface (`Shape`):

We define an interface called `Shape` that acts as the prototype.It declares two methods: `clone()` for making a copy of itself and `draw()` for drawing the shape.

```
// This is like a blueprint for creating shapes.
// It says every shape should be able to clone itself and draw.
public interface Shape {
    Shape clone();  // Make a copy of itself
    void draw();    // Draw the shape
}
```

2. Concrete Prototype (`Circle`):

We implement the `Shape` interface with a concrete class `Circle`. The `Circle` class has a private field `color` and a constructor to set the color when creating a circle. It implements the `clone()` method to create a copy of itself (a new `Circle` with the same color). The `draw()` method is implemented to print a message indicating how the circle is drawn.

```
// This is a specific shape, a circle, implementing the Shape interface.
// It can create a copy of itself (clone) and draw in its own way.
public class Circle implements Shape {
    private String color;

    // When you create a circle, you give it a color.
    public Circle(String color) {
        this.color = color;
    }

    // This creates a copy of the circle.
    @Override
    public Shape clone() {
        return new Circle(this.color);
    }

    // This is how a circle draws itself.
    @Override
    public void draw() {
        System.out.println("Drawing a " + color + " circle.");
    }
}
```

3. Client (`ShapeClient`):

We create a client class, `ShapeClient`, which will use the prototype to create new shapes. The client has a field `shapePrototype` representing the prototype it will use. The constructor takes a `Shape` prototype, and there's a method `createShape()` that creates a new shape using the prototype's `clone()` method.

```
// This is like a user of shapes.
// It uses a prototype (a shape) to create new shapes.
public class ShapeClient {
    private Shape shapePrototype;

    // When you create a client, you give it a prototype (a shape).
    public ShapeClient(Shape shapePrototype) {
        this.shapePrototype = shapePrototype;
    }

    // This method creates a new shape using the prototype.
    public Shape createShape() {
        return shapePrototype.clone();
    }
}
```

4. Complete code for the above example:

In the main class, `PrototypeExample`, we create a concrete prototype (`circlePrototype`) of a red circle. We then create a `ShapeClient` and provide it with the red circle prototype. The client uses the prototype to create a new shape (`redCircle`) using the `createShape()` method. Finally, we draw the newly created red circle using its `draw()` method.

```
// Prototype interface
interface Shape {
    Shape clone();  // Make a copy of itself
    void draw();    // Draw the shape
}

// Concrete prototype
class Circle implements Shape {
    private String color;

    // When you create a circle, you give it a color.
    public Circle(String color) {
```

```
        this.color = color;
    }

    // This creates a copy of the circle.
    @Override
    public Shape clone() {
        return new Circle(this.color);
    }

    // This is how a circle draws itself.
    @Override
    public void draw() {
        System.out.println("Drawing a " + color + " circle.");
    }
}

// Client code
class ShapeClient {
    private Shape shapePrototype;

    // When you create a client, you give it a prototype (a shape).
    public ShapeClient(Shape shapePrototype) {
        this.shapePrototype = shapePrototype;
    }

    // This method creates a new shape using the prototype.
    public Shape createShape() {
        return shapePrototype.clone();
    }
}

// Main class
public class PrototypeExample {
    public static void main(String[] args) {
        // Create a concrete prototype (a red circle).
        Shape circlePrototype = new Circle("red");

        // Create a client and give it the prototype.
        ShapeClient client = new ShapeClient(circlePrototype);

```

```
        // Use the prototype to create a new shape (a red circle).
        Shape redCircle = client.createShape();

        // Draw the newly created red circle.
        redCircle.draw();
    }
}
```

```
Drawing a red circle.
```

When to use the Prototype Design Pattern

Below is when to use prototype design pattern:

- Use the Prototype pattern when creating new objects is more complex or costly than copying existing ones. Cloning can be more efficient if significant resources are needed.
- The Prototype pattern is helpful for managing various objects with minor differences. Instead of creating multiple classes, you can clone and modify prototypes.
- Consider the Prototype pattern for dynamic configurations where you need to create objects at runtime. You can clone a base configuration and adjust it as necessary.
- The Prototype pattern can lower initialization costs, as cloning is often faster than building a new object from scratch, especially if initialization is resource-intensive.

When not to use the Prototype Design Pattern

Below is when not to use Prototype design pattern:

- Avoid using the Prototype pattern when your application predominantly deals with unique object instances, and the overhead of implementing the pattern outweighs its benefits.
- If object creation is simple and does not involve significant resource consumption, and there are no variations of objects, using the Prototype

pattern might be unnecessary complexity.

- If your objects are immutable (unchangeable) and do not need variations, the benefits of cloning may not be significant.
- If your system has a clear and straightforward object creation process that is easy to understand and manage, introducing the Prototype pattern may add unnecessary complexity.

Next Article

Builder Design Pattern

Similar Reads

Software Design Patterns Tutorial

Software design patterns are important tools developers, providing proven solutions to common problems encountered during software development. This article will act as tutorial to help you understand the concep...

9 min read

Complete Guide to Design Patterns

Design patterns help in addressing the recurring issues in software design and provide a shared vocabulary for developers to communicate and collaborate effectively. They have been documented and refined over tim...

11 min read

Types of Software Design Patterns

Designing object-oriented software is hard, and designing reusable object-oriented software is even harder. Christopher Alexander says, "Each pattern describes a problem which occurs over and over again in our...

9 min read

1. Creational Design Patterns

Creational Design Patterns

Creational Design Patterns focus on the process of object creation or problems related to object creation. They help in making a system independent of how its objects are created, composed, and represented. Creational...

4 min read

Types of Creational Patterns

Factory method Design Pattern

The Factory Method Design Pattern is a creational design pattern that provides an interface for creating objects in a superclass, allowing subclasses to alter the type of objects that will be created. This pattern is...

8 min read

Abstract Factory Pattern

The Abstract Factory Pattern is one of the creational design patterns that provides an interface for creating families of related or dependent objects without specifying their concrete classes and implementation, in...

8 min read

Singleton Method Design Pattern in JavaScript

Singleton Method or Singleton Design Pattern is a part of the Gang of Four design pattern and it is categorized under creational design patterns. It is one of the most simple design patterns in terms of...

10 min read

Singleton Method Design Pattern

The Singleton Method Design Pattern ensures a class has only one instance and provides a global access point to it. It's ideal for scenarios requiring centralized control, like managing database connections or...

11 min read

Prototype Design Pattern

The Prototype Design Pattern is a creational pattern that enables the creation of new objects by copying an existing object. Prototype allows us to hide the complexity of making new instances from the client. The...

8 min read

Builder Design Pattern

The Builder Design Pattern is a creational pattern used in software design to construct a complex object step by step. It allows the construction of a product in a step-by-step manner, where the construction process ca...

7 min read

2. Structural Design Patterns

Structural Design Patterns

Structural Design Patterns are solutions in software design that focus on how classes and objects are organized to form larger, functional structures. These patterns help developers simplify relationships betwee...

7 min read

Types of Structural Patterns

3. Behvioural Design Patterns

Behavioral Design Patterns

Behavioral design patterns are a category of design patterns that focus on the interactions and communication between objects. They help define how objects collaborate and distribute responsibility among them, makin...

5 min read

3. Types of Behvioural Patterns

Top Design Patterns Interview Questions [2024]

A design pattern is basically a reusable and generalized solution to a common problem that arises during software design and development. Design patterns are not specific to a particular programming language or...

9 min read

Software Design Pattern in Different Programming Languages

Java Design Patterns Tutorial

Design patterns in Java refer to structured approaches involving objects and classes that aim to solve recurring design issues within specific contexts. These patterns offer reusable, general solutions to common problems...

8 min read

Python Design Patterns Tutorial

Design patterns in Python are communicating objects and classes that are customized to solve a general design problem in a particular context. Software design patterns are general, reusable solutions to common...

8 min read

Modern C++ Design Patterns Tutorial

Design patterns in C++ help developers create maintainable, flexible, and understandable code. They encapsulate the expertise and experience of seasoned software architects and developers, making it easier f...

7 min read

JavaScript Design Patterns Tutorial

Design patterns in Javascipt are communicating objects and classes that are customized to solve a general design problem in a particular context. Software design patterns are general, reusable solutions to common...

8 min read

Software Design Pattern Books

Software Design Pattern in Development

Some other Popular Design Patterns

Design Patterns in Different Languages

Commerce

World GK

CS Subjects