Prototype Design Pattern

Contents

[Intent of the DP 2](#_Toc516313072)

[Explanation 2](#_Toc516313073)

[Common situations of use 3](#_Toc516313074)

[Related patterns 3](#_Toc516313075)

[Useful Links 3](#_Toc516313076)

# Intent of the DP

The **prototype pattern** is a creational design pattern, used when the type of objects to create is determined by a prototypical instance, which is cloned to produce new objects. This pattern is used to:

* Avoid subclasses of an object creator in the client application, like the factory method pattern does.
* Avoid the inherent cost of creating a new object in the standard way (e.g., using the 'new' keyword) when it is prohibitively expensive for a given application (i.e. database access etc.).

# Explanation

The diagram below exemplifies the implementation of the prototype design pattern. This pattern respects the SOLID principles.

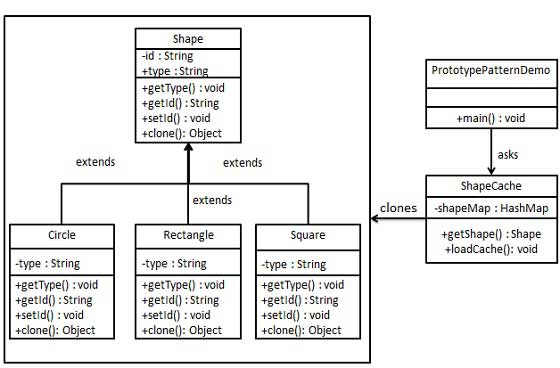
**S:** ShapeCache has the single responsibility of storing prototypes and retrieving the clones.

**O:** The shape class is extended by the three subclasses, enforcing the open/closed principle of having a class open for extension, but closed for modification.

**L:** The Liskov substitution principle stands, as Circle, Rectangle and Square can be substituted for their parent, Shape.

I: The interface segregation principle doesn’t necessarily apply here but we could use an interface to represent a prototype class, which would be unique for all prototypes.

D: The dependency inversion principle doesn’t apply.



# Common situations of use

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:

* By dynamic loading or to avoid building a class hierarchy of factories that parallels the class hierarchy of products.
* When instances of a class can have one of only a few different combinations of states. 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.

# Related patterns

This pattern is related to the Abstract Factory and Builder patterns, all being creational design patterns.

While Prototype returns a copy of a predefined object, Abstract Factory creates new instances of several related abstract classes without specifying their concrete subclasses.

The Builder pattern implies writing a specialized builder for a given class, to be used instead of the constructor. It also creates new instances of the class.

The mentioned patterns seem to get more use than Prototype, because:

* Object creation itself is very lightweight in Java and usually objects hold different data from each other, not the same data.
* The prototype pattern is inherently less flexible than the factory/abstract factory patterns.
* The Java clone method is considered confusing because you have to remember that it only does a shallow copy (reference copy, not object), therefore you might have to create your own method.

# Useful Links

<https://www.gofpatterns.com/design-patterns/module4/abstract-builder-prototype.php>

<https://www.tutorialspoint.com/design_pattern/prototype_pattern.htm>

https://www.geeksforgeeks.org/prototype-design-pattern/

https://hackernoon.com/solid-principles-made-easy-67b1246bcdf