ACCOLITE UNIVERSITY, JULY 2020

Design Principles / Patterns Foundation

Assignment Submission

By: Ayush Malik

Prototype Design Pattern

(Creational Pattern)

Creational design patterns deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or it could add complexity to the design. Creational design patterns are used to solve this problem by somehow controlling this object creation. Prototype Pattern is one of the many types of Creational design pattern.

The first question which comes to mind is when and why would you use a Prototype design pattern. The following section addresses this question.

When and Why?

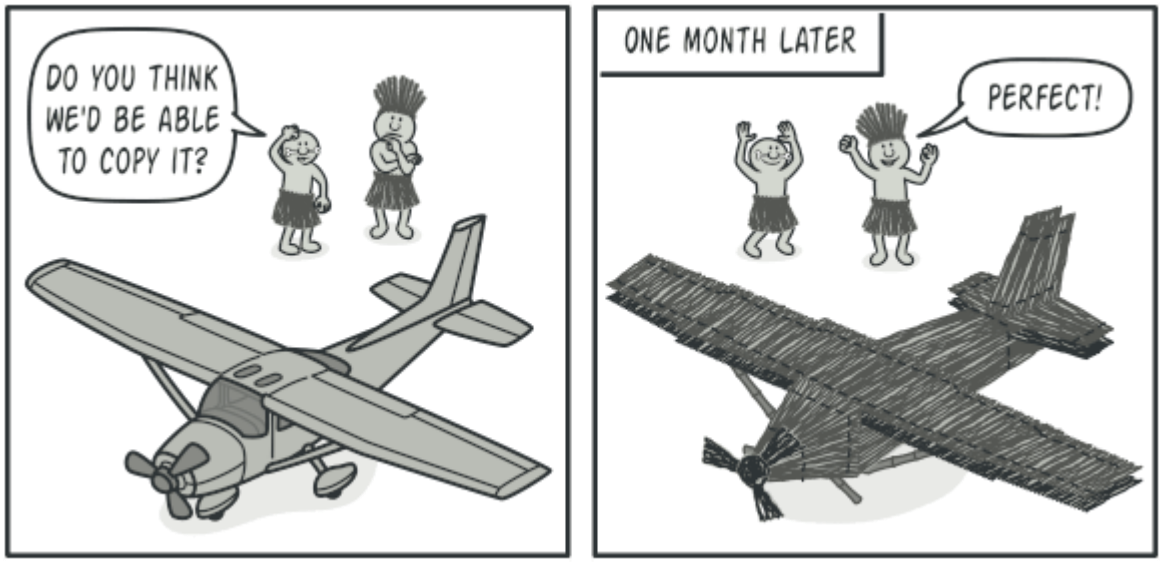
Prototype is a Creational design pattern that lets you copy existing objects without making your code dependent on their classes. Consider the following scenario:

Let us say you have an object, and you want to create an exact copy of it. How would you do it? First, you have to create a new object of the same class. Then you have to go through all the fields of the original object and copy their values over to the new object. There are many problems in this direct approach.

First, not all objects can be copied that way because some of the object's fields may be private and not visible from outside of the object itself.

Secondly, since you have to know the object's class to create a duplicate, your code becomes dependent on that class.

Third, sometimes you only know the interface that the object follows, but not its concrete class, when, for example, a parameter in a method accepts any objects that follow some interface.



Prototype design pattern provides a solution to the above problem.

How Prototype Design Pattern solves this problem?

The Prototype pattern delegates the cloning process to the actual objects that are being cloned. The pattern declares a common interface for all objects that support cloning. This interface lets you clone an object without coupling your code to the class of that object. Usually, such an interface contains just a single clone method.

The implementation of the clone method is very similar in all classes. The method creates an object of the current class and carries over all of the field values of the old object into the new one. You can even copy private fields because most programming languages let objects access private fields of other objects that belong to the same class.

An object that supports cloning is called a *prototype*. When your objects have dozens of fields and hundreds of possible configurations, cloning them might serve as an alternative to sub-classing.

Here's how it works: you create a set of objects, configured in various ways. When you need an object like the one you've configured, you just clone a prototype instead of constructing a new object from scratch.

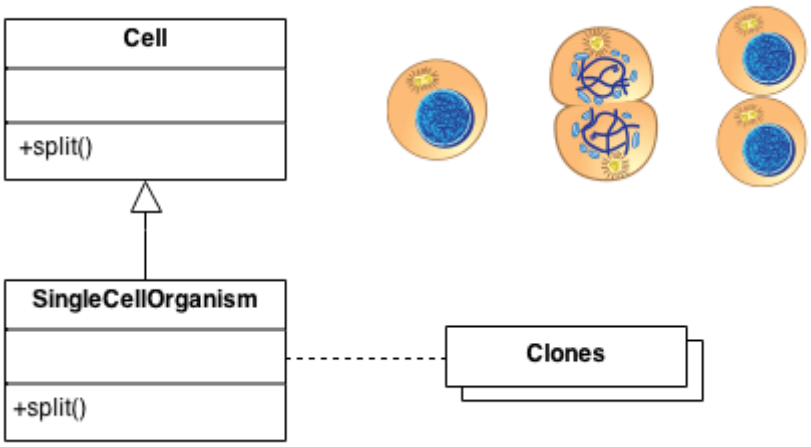
A Concrete Definition

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 the object creation is a heavy process.

Prototype patterns is required, when object creation is time consuming, and costly operation, so we create object with existing object itself. One of the best available way to create object from existing objects are clone() method. Clone is the simplest approach to implement the Prototype pattern. However, it is your call to decide how to copy existing object based on your business model.

A Real-Life Analogy

The mitotic division of a cell - resulting in two identical cells - is an example of a prototype that plays an active role in copying itself and thus, demonstrates the Prototype pattern. When a cell splits, two cells of identical genotype result. In other words, the cell clones itself.

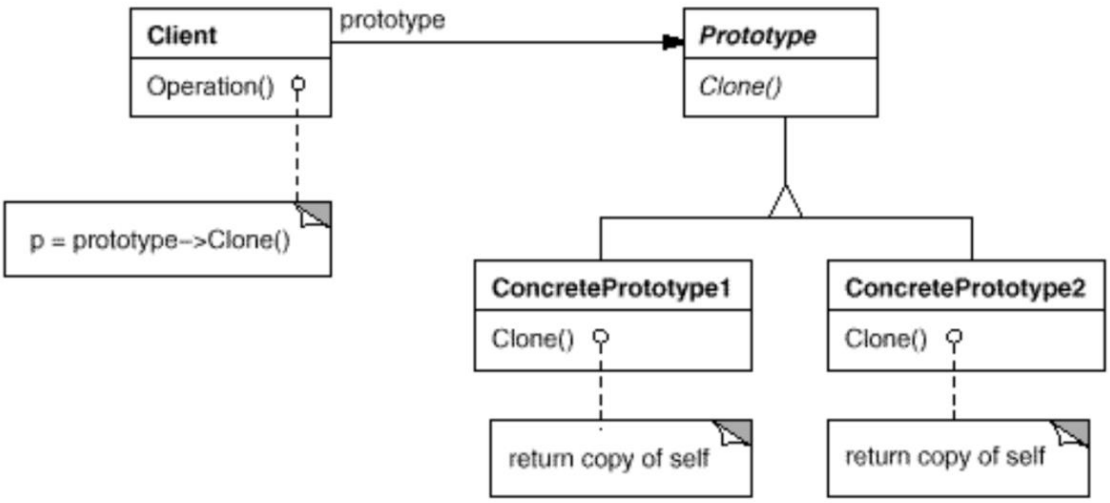


Now that we have a basic understanding of the concept, let's have a look at the structure for implementing this design pattern.

Code Structure

There are 3 key components of a prototype design pattern:

- Prototype: This is the prototype of actual object.
- Prototype registry: This is used as registry service to have all prototypes accessible using simple string parameters.
- Client: Client will be responsible for using registry service to access prototype instances.



Coding Example

Suppose we have an Object that loads data from database. Now we need to modify this data in our program multiple times, so it's not a good idea to create the Object using new keyword and load all the data again from database.

The better approach would be to clone the existing object into a new object and then do the data manipulation. Prototype design pattern mandates that the Object which you are copying should provide the copying feature. It should not be done by any other class. Consider the following code:

```
package com.journaldev.design.prototype;
import java.util.ArrayList;
import java.util.List;

public class Employees implements Cloneable{
	private List<String> empList;

	public Employees(){
		empList = new ArrayList<String>();
	}

	public Employees(List<String> list){
		this.empList=list;
	}
	public void loadData(){
		//read all employees from database and put into the list
		empList.add("Pankaj");
		empList.add("Raj");
		empList.add("David");
		empList.add("Lisa");
	}

	public List<String> getEmpList() {
		return empList;
	}

	@Override
	public Object clone() throws CloneNotSupportedException{
			List<String> temp = new ArrayList<String>();
			for(String s : this.getEmpList()){
				temp.add(s);
			}
			return new Employees(temp);
	}

}
```

```
package com.journaldev.design.test;
import java.util.List;

import com.journaldev.design.prototype.Employees;

public class PrototypePatternTest {

	public static void main(String[] args) throws CloneNotSupportedException {
		Employees emps = new Employees();
		emps.loadData();

		//Use the clone method to get the Employee object
		Employees empsNew = (Employees) emps.clone();
		Employees empsNew1 = (Employees) emps.clone();
		List<String> list = empsNew.getEmpList();
		list.add("John");
		List<String> list1 = empsNew1.getEmpList();
		list1.remove("Pankaj");

		System.out.println("emps List: "+emps.getEmpList());
		System.out.println("empsNew List: "+list);
		System.out.println("empsNew1 List: "+list1);
	}

}
```

If the object cloning was not provided, we will have to make database call to fetch the employee list every time. Then do the manipulations that would have been resource and time consuming.

We conclude our discussion by discussing the advantages and disadvantages of using Prototype Design Pattern.

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 behaviour 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 sub circuit again and again.
- Reduced sub classing – 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.
- It also hides concrete product classes from the client
- Each subclass of Prototype must implement the clone() operation which may be difficult, when the classes under consideration already exist. Also, implementing clone() can be difficult when their internals include objects that don't support copying or have circular references.

References

- https://www.geeksforgeeks.org/prototype-design-pattern/
- https://refactoring.guru/design-patterns/prototype
- https://sourcemaking.com/design_patterns/prototype
- https://www.javatpoint.com/prototype-design-pattern
- https://www.journaldev.com/1440/prototype-design-pattern-in-java