Prototype Pattern

Michal Moravik, SD20w2

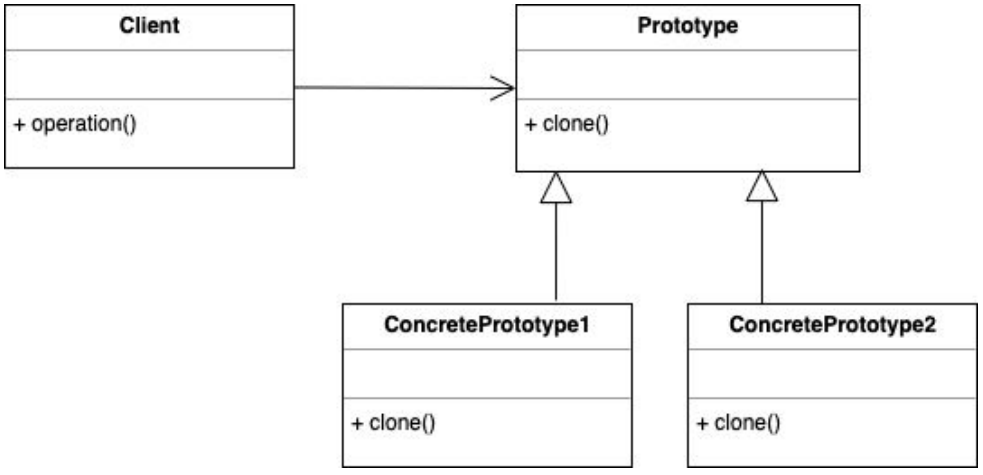
Name: Prototype Pattern

Category: Creational pattern

Intent: Used when we want to create new instances of a class by copying the original ones **to save an expensive instance creation**. It allows us to create any subclass instance of a known superclass at runtime.

Motivation: We use the pattern when there are classes that we want to only use if needed at runtime. It will create a copy of an instance instead of creating a new instance. **This is especially useful when the object creation is too expensive, heavy, and time-consuming** (for example when an object is created with values taken from a database, which takes quite some time).

UML:



Implementation:

First, we need to create an abstract class that will define the important “clone” method and implement the Clonable interface.

```
public abstract class Shape implements Cloneable {

    protected String type;

    abstract void draw();

    public String getType(){
        return type;
    }

    public Object clone() {
        Object clone = null;

        try {
            clone = super.clone();

        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }

        return clone;
    }
}
```

Define the implementation of the abstract class.

```
public class Rectangle extends Shape {

    public Rectangle(){
        type = "Rectangle";
    }

    @Override
    public void draw() {
        System.out.println("This is a draw function of a rectangle :)");
    }
}
```

Make clones during the runtime. We can use the factory method pattern to instantiate clones of some instances. In that case, this would be much more useful.

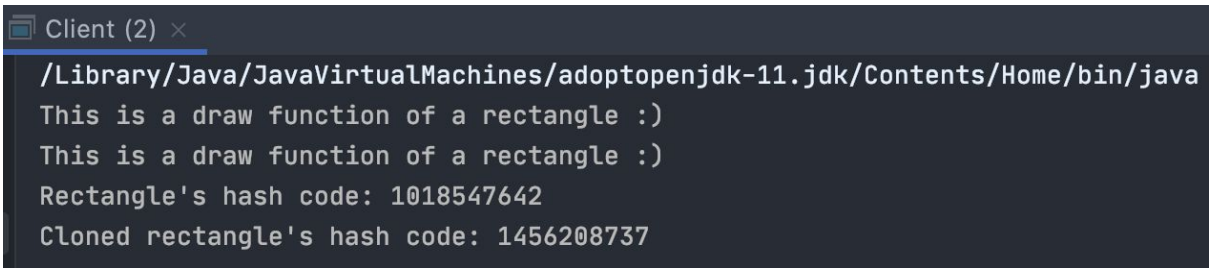
```
public class Client {
    public static void main(String[] args) {

        Rectangle rectangle = new Rectangle();
        Shape clonedRectangle = (Shape) rectangle.clone();

        rectangle.draw();
        clonedRectangle.draw();

        System.out.println("Rectangle's hash code: " + System.identityHashCode(rectangle));
        System.out.println("Cloned rectangle's hash code: " + System.identityHashCode(clonedRectangle));
    }
}
```

We can see that the clone() function will create two totally independent objects with different hash codes. The clone() function creates a deep copy of an instance.



Consequences:

Pros:

- We can create objects without the need of creating a new instance.
- We can produce more expensive object easily without wasting time and resources
- Adding/removing objects at the runtime.

Cons:

- Can be an overkill for simple applications without expensive instance creations

Known uses:

- A good example can be a very expensive object with data taken from various sources (database or heavy API). Instead, we can just copy the one we created with already set properties and proceed from there.

Related patterns:

- Factory Method Pattern - we can control cloning in the factory method.