Prototype

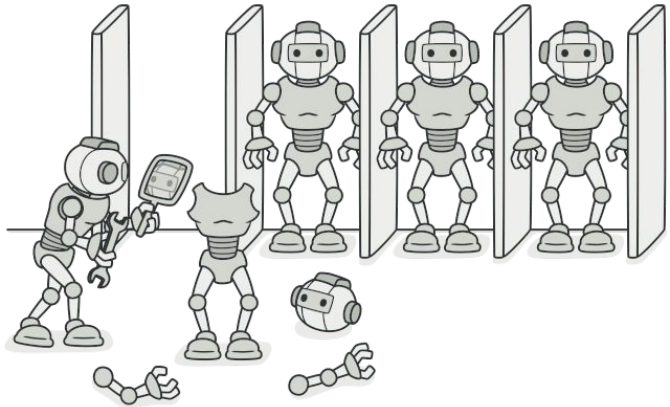
Gabriel Silva - 113786

April 24th, 2024

When should we use this pattern?

Use the Prototype pattern when your code shouldn’t depend on the concrete classes of objects that you need to copy.

Use the pattern when you want to reduce the number of subclasses that only differ in the way they initialize their respective objects.



How to implement this pattern?

Steps:

1. Create the prototype interface and declare the `clone` method in it.

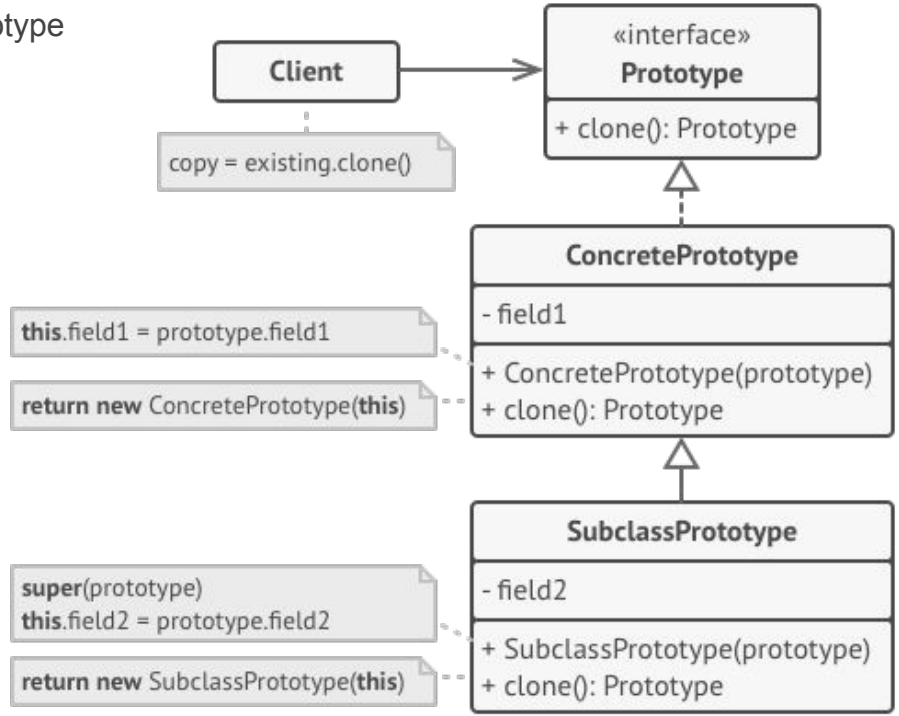
2. A prototype class must define the alternative constructor that accepts an object of that class as an argument. The constructor must copy the values of all fields defined in the class from the passed object into the newly created instance. If you're changing a subclass, you must call the parent constructor to let the superclass handle the cloning of its private fields.

3. The cloning method usually consists of just one line: running a `new` operator with the prototypical version of the constructor. Note, that every class must explicitly override the cloning method and use its own class name along with the `new` operator. Otherwise, the cloning method may produce an object of a parent class.

4. Optionally, create a centralized prototype registry to store a catalog of frequently used prototypes.

(Base) Class Structure

Classes: Client, ConcretePrototype, SubclassPrototype
Interfaces: Prototype



Code Example(s) Prototype Interface

shapes/Shape.java: Common shape interface

```java
package refactoring_guru.prototype.example.shapes;

import java.util.Objects;

public abstract class Shape {
    public int x;
    public int y;
    public String color;

    public Shape() {
    }

    public Shape(Shape target) {
        if (target != null) {
            this.x = target.x;
            this.y = target.y;
            this.color = target.color;
        }
    }

    public abstract Shape clone();

    @Override
    public boolean equals(Object object2) {
        if (!(object2 instanceof Shape)) return false;
        Shape shape2 = (Shape) object2;
        return shape2.x == x && shape2.y == y && Objects.equals(shape2.color, color);
    }
}
```

Code Example(s)
Concrete Prototype

shapes/Circle.java: Simple shape

```
package refactoring_guru.prototype.example.shapes;

public class Circle extends Shape {
    public int radius;

    public Circle() {
    }

    public Circle(Circle target) {
        super(target);
        if (target != null) {
            this.radius = target.radius;
        }
    }

    @Override
    public Shape clone() {
        return new Circle(this);
    }

    @Override
    public boolean equals(Object object2) {
        if (!(object2 instanceof Circle) || !super.equals(object2)) return false;
        Circle shape2 = (Circle) object2;
        return shape2.radius == radius;
    }
}
```

shapes/Rectangle.java: Another shape

```
package refactoring_guru.prototype.example.shapes;

public class Rectangle extends Shape {
    public int width;
    public int height;

    public Rectangle() {
    }

    public Rectangle(Rectangle target) {
        super(target);
        if (target != null) {
            this.width = target.width;
            this.height = target.height;
        }
    }

    @Override
    public Shape clone() {
        return new Rectangle(this);
    }

    @Override
    public boolean equals(Object object2) {
        if (!(object2 instanceof Rectangle) || !super.equals(object2)) return false;
        Rectangle shape2 = (Rectangle) object2;
        return shape2.width == width && shape2.height == height;
    }
}
```

Code Example(s) Demo

OutputDemo.txt: Execution result

```
0: Shapes are different objects (yay!)
0: And they are identical (yay!)
1: Shapes are different objects (yay!)
1: And they are identical (yay!)
2: Shapes are different objects (yay!)
2: And they are identical (yay!)
```

Demo.java: Cloning example

```
package refactoring_guru.prototype.example;

import refactoring_guru.prototype.example.shapes.Circle;
import refactoring_guru.prototype.example.shapes.Rectangle;
import refactoring_guru.prototype.example.shapes.Shape;

import java.util.ArrayList;
import java.util.List;

public class Demo {
    public static void main(String[] args) {
        List<Shape> shapes = new ArrayList<>();
        List<Shape> shapesCopy = new ArrayList<>();

        Circle circle = new Circle();
        circle.x = 10;
        circle.y = 20;
        circle.radius = 15;
        circle.color = "red";
        shapes.add(circle);

        Circle anotherCircle = (Circle) circle.clone();
        shapes.add(anotherCircle);

        Rectangle rectangle = new Rectangle();
        rectangle.width = 10;
        rectangle.height = 20;
        rectangle.color = "blue";
        shapes.add(rectangle);

        cloneAndCompare(shapes, shapesCopy);
    }

    private static void cloneAndCompare(List<Shape> shapes, List<Shape> shapesCopy) {
        for (Shape shape : shapes) {
            shapesCopy.add(shape.clone());
        }

        for (int i = 0; i < shapes.size(); i++) {
            if (shapes.get(i) != shapesCopy.get(i)) {
                System.out.println(i + ": Shapes are different objects (yay!)");
                if (shapes.get(i).equals(shapesCopy.get(i))) {
                    System.out.println(i + ": And they are identical (yay!)");
                } else {
                    System.out.println(i + ": But they are not identical (booo!)");
                }
            } else {
                System.out.println(i + ": Shape objects are the same (booo!)");
            }
        }
    }
}
```