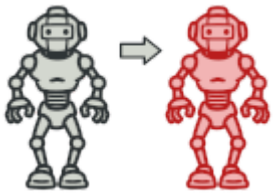

Prototype in Java

Prototype is a creational design pattern that allows cloning objects, even complex ones, without coupling to their specific classes.

All prototype classes should have a common interface that makes it possible to copy objects even if their concrete classes are unknown. Prototype objects can produce full copies since objects of the same class can access each other's private fields.

Learn more about Prototype →

Navigation

Complexity: ★☆☆

Popularity: ★★☆

interface.

Any class can implement this interface to become cloneable.

`java.lang.Object#clone()` (class should implement the `java.lang.Cloneable` interface)

Identification: The prototype can be easily recognized by a `clone` or `copy` methods, etc.

Copying graphical shapes

Let's take a look at how the Prototype can be implemented without the standard `Cloneable` interface.

shapes: Shape list

shapes/Shape.java: Common shape interface

```
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);
```

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) {
```

```
            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;
    }
}
```

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;
```

```
        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!)");
            }
        }
    }
}
```

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!)
```

Prototype registry

You could implement a centralized prototype registry (or factory), which would contain a set of pre-defined prototype objects. This way you could retrieve new objects from the factory by passing its name or other parameters. The factory would search for an appropriate prototype, clone it and return you a copy.

cache

```
package refactoring_guru.prototype.caching.cache;

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

import java.util.HashMap;
import java.util.Map;

public class BundledShapeCache {
    private Map<String, Shape> cache = new HashMap<>();

    public BundledShapeCache() {
        Circle circle = new Circle();
        circle.x = 5;
        circle.y = 7;
        circle.radius = 45;
        circle.color = "Green";

        Rectangle rectangle = new Rectangle();
        rectangle.x = 6;
        rectangle.y = 9;
        rectangle.width = 8;
        rectangle.height = 10;
        rectangle.color = "Blue";

        cache.put("Big green circle", circle);
        cache.put("Medium blue rectangle", rectangle);
    }

    public Shape put(String key, Shape shape) {
        cache.put(key, shape);
        return shape;
    }

    public Shape get(String key) {
        return cache.get(key).clone();
    }
}
```

Demo.java: Cloning example

```
package refactoring_guru.prototype.caching;

import refactoring_guru.prototype.caching.cache.BundledShapeCache;
```

```
public class Demo {
    public static void main(String[] args) {
        BundledShapeCache cache = new BundledShapeCache();

        Shape shape1 = cache.get("Big green circle");
        Shape shape2 = cache.get("Medium blue rectangle");
        Shape shape3 = cache.get("Medium blue rectangle");

        if (shape1 != shape2 && !shape1.equals(shape2)) {
            System.out.println("Big green circle != Medium blue rectangle (yay!)");
        } else {
            System.out.println("Big green circle == Medium blue rectangle (booo!)");
        }

        if (shape2 != shape3) {
            System.out.println("Medium blue rectangles are two different objects (yay!)");
            if (shape2.equals(shape3)) {
                System.out.println("And they are identical (yay!)");
            } else {
                System.out.println("But they are not identical (booo!)");
            }
        } else {
            System.out.println("Rectangle objects are the same (booo!)");
        }
    }
}
```

OutputDemo.txt: Execution result

```
Big green circle != Medium blue rectangle (yay!)
Medium blue rectangles are two different objects (yay!)
And they are identical (yay!)
```

READ NEXT