Prototype

The Prototype Pattern is a creational design pattern used in software development to create new objects based on a prototype of an existing object. The main idea behind this pattern is to create new objects by copying/cloning an existing object, rather than creating new instances from scratch.

Here's an overview of the Prototype Pattern:

1. **Prototype Interface/Abstract Class**: This is an interface or an abstract class that declares the methods for cloning itself. In some cases, it may also define a method to retrieve a prototype instance.

2. **Concrete Prototype Classes**: These are the classes that implement the Prototype interface/abstract class. They provide the implementation for cloning themselves.

3. **Client**: The client is responsible for creating new objects by requesting the prototype to clone itself. The client typically does not have to know the specific classes of the prototypes, as long as they adhere to the Prototype interface.

Here's a simplified example of how the Prototype Pattern can be implemented in Java:

```java
// Prototype interface
interface Prototype {
    Prototype clone();
}

// Concrete prototype class
class ConcretePrototype implements Prototype {
    private String data;

    public ConcretePrototype(String data) {
        this.data = data;
    }

    // Clone method to create a new instance by copying the data
    @Override
    public Prototype clone() {
        return new ConcretePrototype(this.data);
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}

// Client class
public class Client {
    public static void main(String[] args) {
        // Create a prototype instance
        ConcretePrototype prototype = new ConcretePrototype("Initial data");

        // Clone the prototype to create new instances
        ConcretePrototype clonedPrototype1 = (ConcretePrototype)
prototype.clone();
        ConcretePrototype clonedPrototype2 = (ConcretePrototype)
prototype.clone();

        // Modify data in the cloned instances
        clonedPrototype1.setData("Modified data 1");
        clonedPrototype2.setData("Modified data 2");

        // Output data from the prototypes and their clones
        System.out.println("Original Prototype Data: " + prototype.getData());
        System.out.println("Cloned Prototype 1 Data: " +
clonedPrototype1.getData());
        System.out.println("Cloned Prototype 2 Data: " +
clonedPrototype2.getData());
    }
}
```

In this example:

- We have a `Prototype` interface defining the `clone()` method.
- `ConcretePrototype` implements the `Prototype` interface and provides its own implementation of the `clone()` method to create a new instance by copying itself.
- The `Client` class creates a prototype instance and clones it to create new instances. Modifications made to the cloned instances do not affect the original prototype.