



Design Patterns Tutorial

- ▣ Design Patterns - Home
- ▣ Design Patterns - Overview
- ▣ Design Patterns - Factory Pattern
- ▣ Abstract Factory Pattern
- ▣ Design Patterns - Singleton Pattern
- ▣ Design Patterns - Builder Pattern
- ▣ Design Patterns - Prototype Pattern
- ▣ Design Patterns - Adapter Pattern
- ▣ Design Patterns - Bridge Pattern
- ▣ Design Patterns - Filter Pattern
- ▣ Design Patterns - Composite Pattern
- ▣ Design Patterns - Decorator Pattern
- ▣ Design Patterns - Facade Pattern
- ▣ Design Patterns - Flyweight Pattern
- ▣ Design Patterns - Proxy Pattern
- ▣ Chain of Responsibility Pattern
- ▣ Design Patterns - Command Pattern
- ▣ Design Patterns - Interpreter Pattern
- ▣ Design Patterns - Iterator Pattern



▣ [Design Patterns - Observer Pattern](#)

▣ [Design Patterns - State Pattern](#)

▣ [Design Patterns - Null Object Pattern](#)

▣ [Design Patterns - Strategy Pattern](#)

▣ [Design Patterns - Template Pattern](#)

▣ [Design Patterns - Visitor Pattern](#)

▣ [Design Patterns - MVC Pattern](#)

▣ [Business Delegate Pattern](#)

▣ [Composite Entity Pattern](#)

▣ [Data Access Object Pattern](#)

▣ [Front Controller Pattern](#)

▣ [Intercepting Filter Pattern](#)

▣ [Service Locator Pattern](#)

▣ [Transfer Object Pattern](#)

Design Patterns Resources

▣ [Design Patterns - Questions/Answers](#)

▣ [Design Patterns - Quick Guide](#)

▣ [Design Patterns - Useful Resources](#)

▣ [Design Patterns - Discussion](#)

Design Patterns - Proxy Pattern

[⬅ Previous Page](#)[Next Page ➡](#)

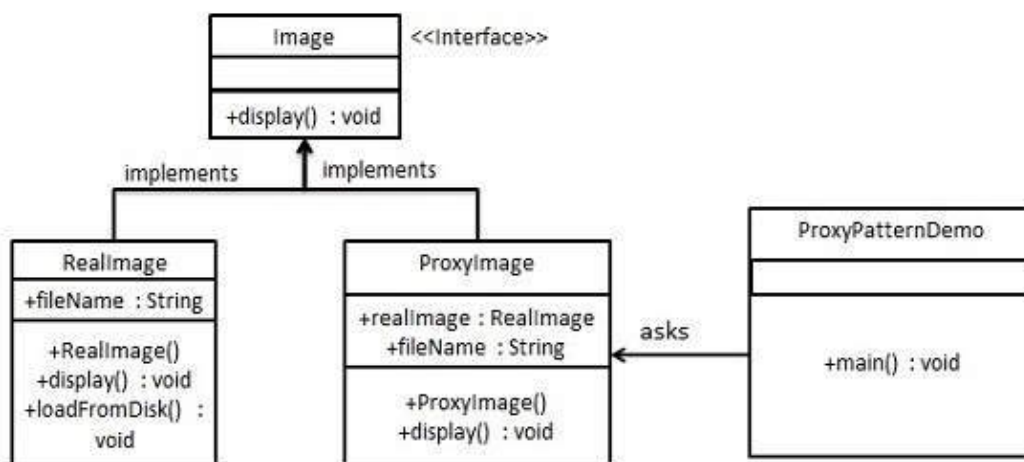
In proxy pattern, a class represents functionality of another class. This type of design pattern comes under structural pattern.

In proxy pattern, we create object having original object to interface its functionality to outer world.

Implementation

We are going to create an *Image* interface and concrete classes implementing the *Image* interface. *ProxyImage* is a proxy class to reduce memory footprint of *RealImage* object loading.

ProxyPatternDemo, our demo class, will use *ProxyImage* to get an *Image* object to load and display as it needs.



Step 1

Create an interface.

Image.java

```
public interface Image {
    void display();
}
```

Step 2



```
public class RealImage implements Image {  
  
    private String fileName;  
  
    public RealImage(String fileName){  
        this.fileName = fileName;  
        loadFromDisk(fileName);  
    }  
  
    @Override  
    public void display() {  
        System.out.println("Displaying " + fileName);  
    }  
  
    private void loadFromDisk(String fileName){  
        System.out.println("Loading " + fileName);  
    }  
}
```

ProxyImage.java

```
public class ProxyImage implements Image{  
  
    private RealImage realImage;  
    private String fileName;  
  
    public ProxyImage(String fileName){  
        this.fileName = fileName;  
    }  
  
    @Override  
    public void display() {  
        if(realImage == null){  
            realImage = new RealImage(fileName);  
        }  
        realImage.display();  
    }  
}
```

Step 3

Use the *ProxyImage* to get object of *RealImage* class when required.

ProxyPatternDemo.java

```
public class ProxyPatternDemo {  
  
    public static void main(String[] args) {  
        Image image = new ProxyImage("test_10mb.jpg");  
  
        //image will be loaded from disk  
        image.display();  
        System.out.println("");  
    }  
}
```



Step 4

Verify the output.

```
Loading test_10mb.jpg
Displaying test_10mb.jpg

Displaying test_10mb.jpg
```

⬅ Previous Page

Next Page ➡

Advertisements

OPENSIFT MARKETPLACE

marketplace.openshift.com

Add Databases, Monitoring, Search, Messaging, Scheduling, and More.



© Copyright 2015. All Rights Reserved.

Enter email for newsletter

go