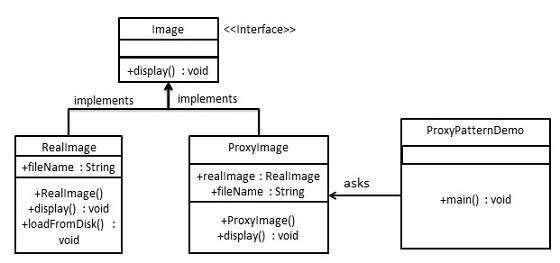
In proxy pattern, a class represents functionality of another class. This type of design pattern comes under structural pattern.

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 *Real Image* object loading.

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



/\*

\* step 1

\* create an interface

\* Image.java

\*/

interface Image{

void display();

}

/\*

\* step 2

\* create concrete classes implementing the same interface

\* RealImage.java

\*/

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

\*/

class ProxyImage implements Image{

private RealImage realImage;

private String fileName;

public ProxyImage(String fileName) {

this.fileName = fileName;

}

@Override

public void display() {

// TODO Auto-generated method stub

if(realImage == null) {

realImage = new RealImage(fileName);

}

realImage.display();

}

}

/\*

\* TestProxyDemo.java

\*/

public class TestProxyDemo {

public static void main(String[] args) {

Image image = new ProxyImage("test\_mb.jpg");

//image will be loaded from disk

image.display();

System.out.println("");

//image will not be loaded from disk

image.display();

}

}