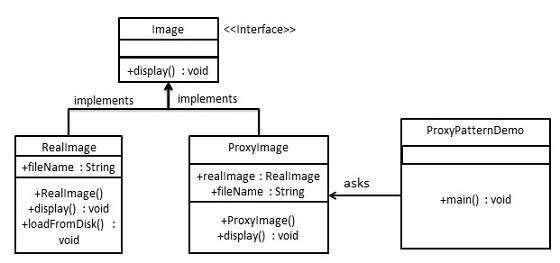
In proxy pattern, a class represents functionality of another class. This type of design pattern comes under structural pattern.

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 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

Create concrete classes implementing the same interface.

*RealImage.java*

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("");

//image will not be loaded from disk

image.display();

}

}

Step 4

Verify the output.

Loading test\_10mb.jpg

Displaying test\_10mb.jpg

Displaying test\_10mb.jpg

///////////////////

**When to use this pattern?**

Proxy pattern is used when we need to create a wrapper to cover the main object’s complexity from the client.

**Types of proxies**

**Remote proxy:**  
They are responsible for representing the object located remotely. Talking to the real object might involve marshalling and unmarshalling of data and talking to the remote object. All that logic is encapsulated in these proxies and the client application need not worry about them.

**Virtual proxy:**

These proxies will provide some default and instant results if the real object is supposed to take some time to produce results. These proxies initiate the operation on real objects and provide a default result to the application. Once the real object is done, these proxies push the actual data to the client where it has provided dummy data earlier.

**Protection proxy:**

If an application does not have access to some resource then such proxies will talk to the objects in applications that have access to that resource and then get the result back.

**Smart Proxy:**

A smart proxy provides additional layer of security by interposing specific actions when the object is accessed. An example can be to check if the real object is locked before it is accessed to ensure that no other object can change it.

**Some Examples**

A very simple real life scenario is our college internet, which restricts few site access. The proxy first checks the host you are connecting to, if it is not part of restricted site list, then it connects to the real internet. This example is based on Protection proxies.

Lets see how it works :

**Interface of Internet**

|  |
| --- |
| package com.saket.demo.proxy;    public interface Internet  {      public void connectTo(String serverhost) throws Exception;  } |

**RealInternet.java**

|  |
| --- |
| package com.saket.demo.proxy;    public class RealInternet implements Internet  {      @Override      public void connectTo(String serverhost)      {          System.out.println("Connecting to "+ serverhost);      }  } |

**ProxyInternet.java**

|  |
| --- |
| package com.saket.demo.proxy;    import java.util.ArrayList;  import java.util.List;      public class ProxyInternet implements Internet  {      private Internet internet = new RealInternet();      private static List<String> bannedSites;        static      {          bannedSites = new ArrayList<String>();          bannedSites.add("abc.com");          bannedSites.add("def.com");          bannedSites.add("ijk.com");          bannedSites.add("lnm.com");      }        @Override      public void connectTo(String serverhost) throws Exception      {          if(bannedSites.contains(serverhost.toLowerCase()))          {              throw new Exception("Access Denied");          }            internet.connectTo(serverhost);      }    } |

**Client.java**

|  |
| --- |
| package com.saket.demo.proxy;    public class Client  {      public static void main (String[] args)      {          Internet internet = new ProxyInternet();          try          {              internet.connectTo("geeksforgeeks.org");              internet.connectTo("abc.com");          }          catch (Exception e)          {              System.out.println(e.getMessage());          }      }  } |

//////////////////////////////////////

Types of proxies

Remote proxy:

They are responsible for representing the object located remotely. Talking to the real object might involve marshalling and unmarshalling of data and talking to the remote object. All that logic is encapsulated in these proxies and the client application need not worry about them.

Virtual proxy:

These proxies will provide some default and instant results if the real object is supposed to take some time to produce results. These proxies initiate the operation on real objects and provide a default result to the application. Once the real object is done, these proxies push the actual data to the client where it has provided dummy data earlier.

Protection proxy:

If an application does not have access to some resource then such proxies will talk to the objects in applications that have access to that resource and then get the result back.

Smart Proxy:

A smart proxy provides additional layer of security by interposing specific actions when the object is accessed. An example can be to check if the real object is locked before it is accessed to ensure that no other object can change it.

Some Examples

A very simple real life scenario is our college internet, which restricts few site access. The proxy first checks the host you are connecting to, if it is not part of restricted site list, then it connects to the real internet. This example is based on Protection proxies.

Lets see how it works :

Interface of Internet

package com.saket.demo.proxy;

public interface Internet

{

public void connectTo(String serverhost) throws Exception;

}

RealInternet.java

package com.saket.demo.proxy;

public class RealInternet implements Internet

{

@Override

public void connectTo(String serverhost)

{

System.out.println("Connecting to "+ serverhost);

}

}

ProxyInternet.java

package com.saket.demo.proxy;

import java.util.ArrayList;

import java.util.List;

public class ProxyInternet implements Internet

{

private Internet internet = new RealInternet();

private static List<String> bannedSites;

static

{

bannedSites = new ArrayList<String>();

bannedSites.add("abc.com");

bannedSites.add("def.com");

bannedSites.add("ijk.com");

bannedSites.add("lnm.com");

}

@Override

public void connectTo(String serverhost) throws Exception

{

if(bannedSites.contains(serverhost.toLowerCase()))

{

throw new Exception("Access Denied");

}

internet.connectTo(serverhost);

}

}

Client.java

package com.saket.demo.proxy;

public class Client

{

public static void main (String[] args)

{

Internet internet = new ProxyInternet();

try

{

internet.connectTo("geeksforgeeks.org");

internet.connectTo("abc.com");

}

catch (Exception e)

{

System.out.println(e.getMessage());

}

}

}

As one of the site is mentioned in the banned sites, So

Running the program will give the output :

Connecting to geeksforgeeks.org

Access Denied