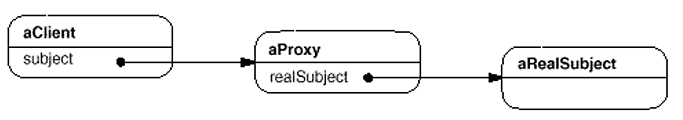
Proxy Design Pattern

GOF : **Provide a surrogate or placeholder for another object to control access to it.**

The main benefit behind proxy design pattern is minimize the usage of very large or complex object or impossible object.

# **Object Diagram**



Let us consider a basic example.

interface Image {

public abstract void displayImage();

}

public class RealImage implements Image {

private String filename = null;

/\*\*

\* Constructor

\* @param FILENAME

\*/

public RealImage(final String FILENAME) {

filename = FILENAME;

loadImageFromDisk();

}

/\*\*

\* Loads the image from the disk

\*/

private void loadImageFromDisk() {

System.out.println("Loading " + filename);

}

/\*\*

\* Displays the image

\*/

public void displayImage() {

System.out.println("Displaying " + filename);

}

}

public class ProxyImage implements Image {

private RealImage image = null;

private String filename = null;

/\*\*

\* Constructor

\* @param FILENAME

\*/

public ProxyImage(final String FILENAME) {

filename = FILENAME;

}

/\*\*

\* Displays the image

\*/

public void displayImage() {

if (image == null) {

image = new RealImage(filename);

}

image.displayImage();

}

}

class TestProxyDesign {

/\*\*

\* Test method

\*/

public static void main(String[] args) {

final Image IMAGE1 = new ProxyImage("HiRes\_10MB\_Photo1");

final Image IMAGE2 = new ProxyImage("HiRes\_10MB\_Photo2");

IMAGE1.displayImage(); // loading necessary

IMAGE1.displayImage(); // loading unnecessary

IMAGE2.displayImage(); // loading necessary

IMAGE2.displayImage(); // loading unnecessary

IMAGE1.displayImage(); // loading unnecessary

}

}

In the above case, there is no need to create the Image object once again.

Let us corelate with the real world objects and entities. As we know that vendors like Dell, Lenovo, HP can not be reached directly. If you want to get technical solution from them, you have to contact their Call Centres or Customer Care. In this case Customer Care or Call Centres are the proxies on behalf of the respective vendors.

Let us consider another good example. As you know that it is very difficult to meet the President of a Nation directly. We have to go through the channel or we have to meet the PA to President. In this PA to President is the Proxy on behalf of President of the Nation.

Let us consider the following piece of code.

public interface SolutionProvider {

public String provideSolution();

}

Public final class President implements SolutionProvider {

@Override

public String provideSolution() {

System.out.println("Solution Provided by President of the Nation");

return "Solution Provided";

}

}

public class ProxyPresident implements SolutionProvider {

private String problemType;

private President president = new President();

public ProxyPresident( String problemType ) {

this.problemType = problemType;

}

@Override

public String provideSolution() {

if( problemType.equals("Political") || problemType.equals("Economic") )

return president.provideSolution();

else

{

System.out.println("Solution Provided by Proxy which is a PA to President");

return "Solution Provided";

}

}

}

The test harnes class is given below.

import com.ddlab.design.proxy.type2.ProxyPresident;

import com.ddlab.design.proxy.type2.SolutionProvider;

public class TestProxyPattern {

public static void main(String[] args) {

String problemType = "Love Issue";

SolutionProvider provider = new ProxyPresident(problemType);

String solution = provider.provideSolution();

System.out.println("Solution : "+solution);

problemType = "Political";

provider = new ProxyPresident(problemType);

solution = provider.provideSolution();

System.out.println("Solution : "+solution);

}

}

From the above design of the class, it is evident that you can not directly meet the president as the class is

package private, ofcourse you can not extend it. The most import point here is that if you have any issue related to Love or relationship, the solution will be provided by Proxy which is the PA to President other political and economic issues will be resolved only by President.

There is another good example of Proxy design suggested by Joshua Bloch is SerializationProxy.

package org.effectivejava.examples.chapter11.item78;

import java.io.InvalidObjectException;

import java.io.ObjectInputStream;

import java.io.Serializable;

import java.util.Date;

public final class Period implements Serializable {

private final Date start;

private final Date end;

public Period(Date start, Date end) {

this.start = new Date(start.getTime());

this.end = new Date(end.getTime());

if (this.start.compareTo(this.end) > 0)

throw new IllegalArgumentException(start + " after " + end);

}

public Date start() {

return new Date(start.getTime());

}

public Date end() {

return new Date(end.getTime());

}

public String toString() {

return start + " - " + end;

}

// Serialization proxy for Period class - page 312

private static class SerializationProxy implements Serializable {

private final Date start;

private final Date end;

SerializationProxy(Period p) {

this.start = p.start;

this.end = p.end;

}

//use Any Number

private static final long serialVersionUID = 234098243823485285L;

// readResolve method for Period.SerializationProxy - Page 313

private Object readResolve() {

return new Period(start, end); // Uses public constructor

}

}

// writeReplace method for the serialization proxy pattern - page 312

private Object writeReplace() {

return new SerializationProxy(this);

}

// readObject method for the serialization proxy pattern - Page 313

private void readObject(ObjectInputStream stream)

throws InvalidObjectException {

throw new InvalidObjectException("Proxy required");

}

}