Proxy Design Pattern - An Industry Perspective

❖ **Publisher :**
Harshwardhan Publication Pvt.Ltd.
Limbaganesh, Dist. Beed (Maharashtra)
Pin-431126, vidyawarta@gmail.com

❖ **Printed by :**
Harshwardhan Publication Pvt.Ltd.
Limbaganesh, Dist. Beed, Pin-431126

❖ **Page design & Cover :**
Shaikh Jahurodden

❖ **Edition: July 2017**

ISBN 978-93-86778-20-8

❖ **Price : 500/ -**

Acknowledgements

Many individuals share credit for this book's preparation. I extend my sincere thanks to Late Prof. A.D.Shinde, the Founder Director and Managing Trustee who has been a constant source of inspiration for me throughout my career. His support is really a driving force for me. Also, I would like to thank Dr.R.A.Shinde, Hon'ble Secretary, SIBER for his whole hearted support and continuous encouragement. I am grateful to Dr. M.M.Ali, Director SIBER for his invaluable guidance. I take this opportunity to thank Dr.V.M.Hilage, Former Director and Academic Advisor, SIBER, Dr. R.V.Kulkarni, H.O.D., Department of Computer Studies, for showing a keen interest in the matter of this book and extending all support facilities for the in-timely completion of this book. I cannot refrain myself from acknowledging my heartfelt gratitude towards CSIBER alumnus Mr. Kedar Kulkarni who motivated me to come up with this book and my discussion with him was the driving force for the preparation of this stuff. The material covered in this book is a systematic effort taken towards solving the various queries which I received from my students time to time, during my 15+ years tenure of teaching at PG level. Last but not the least I thank all faculty members and non-teaching staff of department of computer studies, CSIBER, Kolphpur and department of Computer Science, Shivaji University, Kolhapur who have made contribution to this book either directly or indirectly.

Dr. Poornima G. Naik

Preface

It gives us an immense pleasure to bring out a book entitled '**Proxy Design Pattern - An Industry Perspective**'. The book is intended to serve those who have just started their long journey in software development. The book deals with some of the best practices in software design and development.

Design patterns are based on some of the best practices adapted by experienced object-oriented software developers. Design Patterns are very popular among software developers. Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system. A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems in a language independent manner. It describes the problem, the solution, when to apply the solution, and its consequences. This book is intended to serve those who have just stepped in to software development by providing them issues related to design and solutions for the same. Design Patterns offer best practices for solving commonly occurring problems in software development and are proven solution approaches to problems belonging to a specific category. They provide a guideline for an inexperienced developer on issues related to software development in real world application development and facilitate in robust software development. Some such design patterns are singleton, observer, façade, adapter to name a few. Design patterns are independent of languages and platforms and offer developers a common language for software development independent of technology and language and are based on basic principles of object oriented analysis and design techniques conforming to abstraction, inheritance, polymorphism, delegation, composition, aggregation. Design patterns have their origin in the book entitled "**Design Patterns - Elements of Reusable Object-Oriented Software**" by Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides, commonly referred to as Gang of

Four (GoF).The 23 Gang of Four (GoF) patterns are generally considered the foundation for all other patterns. They are categorized in three groups: Creational, Structural, and Behavioral.

There are many issues to be meticulously observed and followed in the case of large software designs. Overlooking any of these aspects results into problematic a software. Programming by contract assumes both sides in a transaction understand what actions generate what behavior and will abide by that contract. The second principle in object-oriented programming is that, it is beneficial to assign responsibilities to classes in a way that keeps cohesion high. Further the principle of least knowledge communication with only immediate neighbours which address software maintainability issues. Open Closed principle dictates that software entities like classes, modules and functions should be open for extension but closed for modifications.

The book is thought provoking and stimulates the reader to think beyond basics. Author through her academic experience of more than 15 years found that there is no book in market which has a pin point focus only on practical concepts with needed theory. There is a need for book which can give hands on approach to students for brushing up their software design basics and handy reference to the candidates to prepare for their internship projects in industry. The book will play a key role for the students during their academics and even during their job hunting. Language used is very simple with necessary comments wherever required.

How this book is organized

The series is designed in a manner that tries to manage a logical progression through implementation of Proxy Design Pattern – starting with the most fundamental model 1to the most Generic Model 7. Each chapter deals with the refinement of the model presented in the previous chapter addressing its drawbacks one at a time. **Chapter 1** gives a formal introduction to Software Design Pattern, their categorization. The purpose of each design pattern is briefly mentioned. **Chapter 2** provides implementation details of Model 1 which offers a console

based interface to an end user and simulates user interaction. The purpose of the module, module requirement specification, system requirements, application architecture is provided along with a complete source code in each case. **Chapter 3** is the refinement of the model described in Chapter 2 which provides a GUI to an end user for displaying a single proxy object. On clicking the proxy, the real object is instantiated which displays a detailed information about the product. **Chapter 4** through **Chapter 7** successively refine the model to instantiate multiple proxy objects, storing the configuration information pertaining to proxy and real image dimensions, no of proxy products to be accommodated in a single row persistently in an XML and JSON file. Retrieving the persistent product information stored in back end database management system. Finally, I have ended with the most generic version of the model where the end user can select between two different options available for storing configuration in formation in XML file format or JSON file format and retrieve the product information from either MS-Access, MySQL or Oracle database. In the subsequent books I intend to explore few more software design patterns.

Dr. Poornima G. Naik

Contents

Chapter 1
Introduction to Design Pattern

Design pattern is a template or a reusable solution to a commonly occurring problem in software design. Design patterns are employed for representing some of the best practices normally adopted in object oriented software design. A design pattern systematically addresses the recurring problems occurring in object-oriented software designs. The goals of design patterns are:

- To address solution to a recurring problem and matching it with a pattern.
- To address consequences of a solution
- To provide implementation hints and examples.
- To render the code reusable.

A design patterns is a well-proved solution for solving the specific problem/task. 23 design patterns were listed by Gang of Four in their book "Design Patterns: Elements of Reusable Object-Oriented Software".

1.1 Types of Design Patterns

Design patterns can be broadly classified into 3 categories:

- Creational
- Structural
- Behavioural

1.1.1. Creational Design Patterns

This category deals with object instantiation. These design patterns can further be classified into the following sub categories.

- Class-creational patterns
- Object-creational patters

1.1.1.1 Class-creational patterns

These type of patterns use inheritance in instantiation process.

1.1.1.2 Object-creational patters

These type of patterns use delegation in instantiation process.

Creational design patterns are

- Factory Method
- Abstract Factory
- Builder
- Singleton
- Object Pool
- Prototype and Singleton.

1.1.2. Structural Design Patterns

Structural design patterns deal with organization of different classes and objects to form a larger structure thereby providing a new functionality.

Structural design patterns are

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Private Class Data and Proxy.

1.1.3. Behavioural Design Patterns

These type of patterns are concerned with the identification of common communication patterns between objects are realization of these patterns.

Behavioral patterns are

- Chain of responsibility
- Command
- Interpreter
- Iterator

- Mediator
- Memento
- Null Object
- Observer
- State Object Behavioural
- Strategy Object Behavioural
- Template method Class Behavioural
- Visitor Object Behavioural

The purpose of each design pattern is depicted in the following table:

Design Pattern	Purpose
Chain of responsibility	To give more than one object an opportunity to handle the request by linking the receiving objects together
Command	To encapsulate a request allowing it to be treated as an object which allows the request to be handled in traditionally object based relationships such as queuing and callbacks.
Interpreter	To define a representation for a grammar as well as a mechanism to understand and act upon the grammar.
Iterator	To allow for access to the elements of an object collection without allowing access to its underlying representation.
Mediator	To allow loose coupling by encapsulating the way disparate sets of objects interact and communicate with each other, thereby enabling actions of each object set to vary independently of one another.
Memento	To allow for capturing and externalizing an object's internal state so that it can be restored later.
Observer	To notify one or more objects of state changes in other

	objects within the system.
State Object Behavioural	To associate object circumstances to its behavior, thereby allowing the object to behave in different ways based upon its internal state.
Strategy Object Behavioural	To define a set of encapsulated algorithms that can be swapped to carry out a specific behavior.
Template method Class Behavioural	To identify the framework of an algorithm, allowing implementing classes to define the actual behavior.
Visitor Object Behavioural	To allow for one or more operations to be applied to a set of objects at runtime thereby decoupling the operations from the object structure.

Chapter 2
Proxy Design Pattern Implementation Model 1

Purpose : Model 1 generates a CLI interface to an end user simulating the display of a single proxy object of dimension 10 x 10 displaying a short description. On simulating a clicking of the proxy object, the corresponding real object of dimension 100 x 100 is generated which displays a detailed information.

2.1 Module Requirement Specification :

The proposed module should be capable of performing the following tasks:

i) Module should generate a console based interface for an end user.

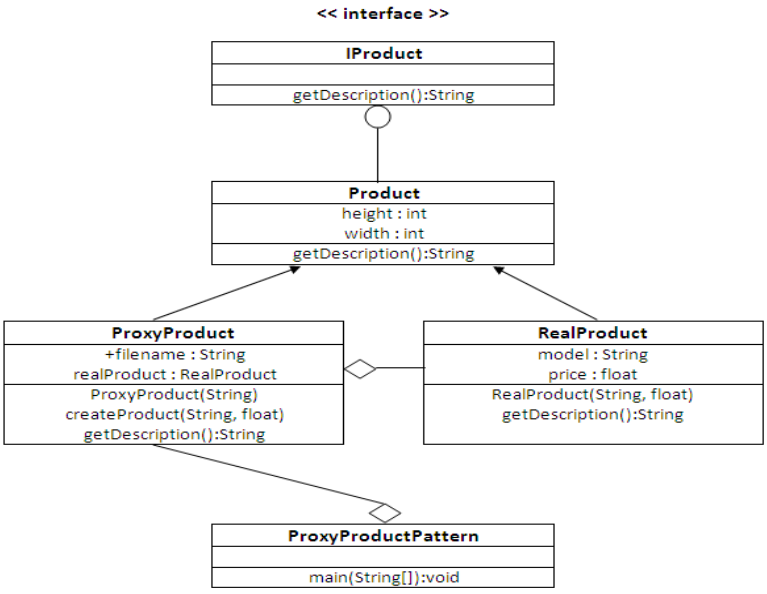
ii) Module should allow the user to instantiate a proxy object.

iii) Module should display a short description to an end user on creating a proxy object

iv) Module should allow the user to simulate clicking on proxy object to instantiate a real object

v) Module should display a detailed description to an end user on creating a real object

2.2 Application Architecture



2.3 Complete Source Code

IProduct.java

```
interface IProduct
{
   public String getDescription();
}
```

Product.java

```
class Product implements IProduct
{
 /* Attributes of a Product */
 int height;
 int width;
 public String getDescription()
 {
  return null;
 }
}
```


ProxyProduct.java

```
class ProxyProduct extends Product
{
 /* Attributes of Proxy Product */
 private String filename;
 private RealProduct realProduct;

 public ProxyProduct(String filename)
 {
    this.filename=filename;
    height=10;
    width=10;
 }
```

```
public String getDescription()
 {
  /* Invokes getDescription() method of realProduct, if valid */
   if (realProduct==null)
   {
     StringBuffer sb=new StringBuffer();
     sb.append("File Name : "+filename);
     sb.append("\n");
     sb.append("Image Size : "+width+" x "+height);
     return sb.toString();
   }
   else
     return realProduct.getDescription();
 }

 public void createProduct(String model,float price)
 {
  realProduct=new RealProduct(model,price);
 }
}
```


RealProduct.java

```
class RealProduct extends Product
{
/* Attributes of  Real Product */
  String model
  float price;

  public RealProduct(String model,float price)
  {
      height=100;
      width=100;
```

```
      this.price=price;
      this.model=model;
   }

  public String getDescription()
  {
     StringBuffer sb=new StringBuffer();
     sb.append("Model : "+model);
     sb.append("\n");
     sb.append("Price : "+price);
     sb.append("\n");
     sb.append("Image Size : "+width+" x "+height);
     return sb.toString();
  }
}
```

ProxyProductPattern.java

```
import java.io.IOException;


public class ProxyProductPattern
{
  public static void main(String[] args)
  {
    ProxyProduct proxyProduct=null;
    char option='y';
    while (option != 'E' && option != 'e' )
    {
     try
     {
       switch(option)
       {
          case 'P':
          case 'p':
```

```
            System.out.println("\n Proxy Object Created .....");
            proxyProduct=new ProxyProduct("mobile1.jpg");
            System.out.println(proxyProduct.getDescription());
            System.in.skip(2);
            break;
        case 'C':
        case 'c':
           System.out.println("\n Proxy Object Clicked .....");
           if (proxyProduct == null)
           {
                proxyProduct=new ProxyProduct("mobile1.jpg");
           }
           proxyProduct.createProduct("Nokia",8000.0f);
           System.out.println(proxyProduct.getDescription());
            System.in.skip(2);
            break;
        default:
             break;
      }
    /* Display Main Menu */
      System.out.println("* * * * * * * * * * * * * * * * * * *\n");
      System.out.println("              Main Menu\n");
      System.out.println("* * * * * * * * * ** * * * * * * * * *");
      System.out.println(" 1. Create [P/p]roxy Object ");
      System.out.println(" 2. [C/c]lick Proxy Object ");
      System.out.println(" 3. [E/e]xit");
      System.out.println("* * * * *  * * * * ** * * * * * * * *");
      System.out.print("Enter Your Choice : ");
      option=(char)System.in.read();
    }
    catch (IOException e)
    {
       System.out.println("Error in reading...");
    }
```

```
  }
 }
}
```

Executable Batch File

run.bat

```
javac IProduct.java
pause
javac Product.java
pause
javac RealProduct.java
pause
javac ProxyProduct.java
pause
javac ProxyProductPattern.java
pause
```

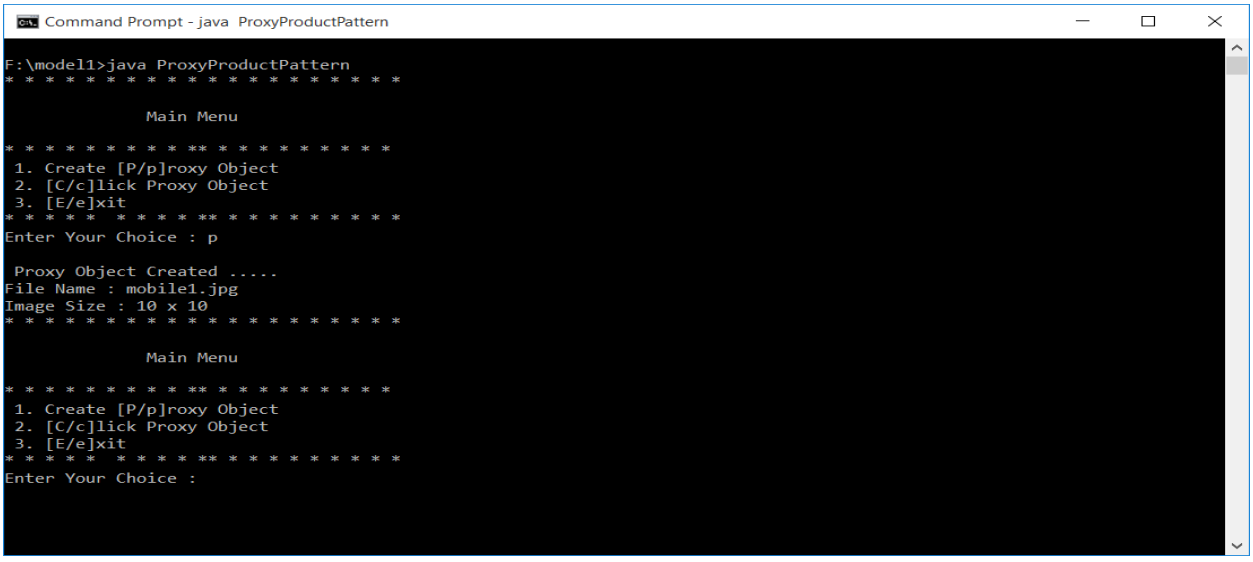

2.4 Test Cases and Screen Shots

Test Cases :

Test Case 1 : Select Option ‘p’ or ‘P’

Desired Output : Proxy object created and short description displayed.

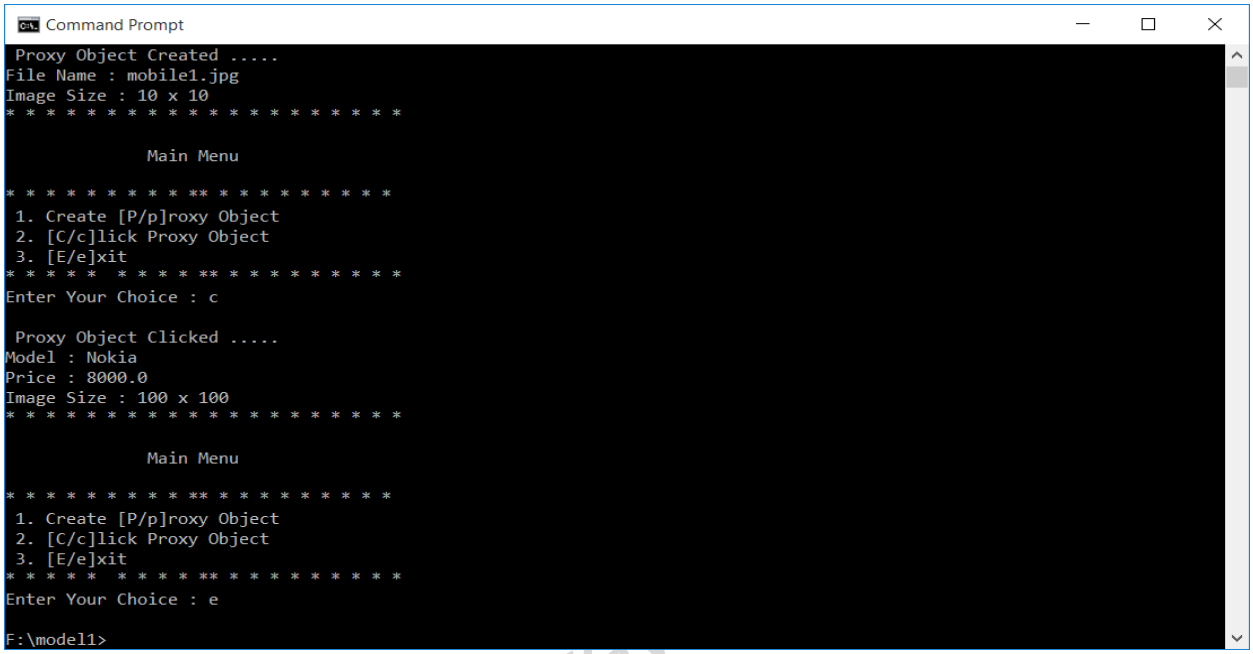
Actual Output :



Test Case 2 : Select Option ‘c’ or ‘C’

Desired Output : Proxy object should be created if not already created which in turn should create a Real object and display a detailed description.

Actual Output :



Test Case 3 : Select Option ‘e’ or ‘E’

Desired Output : Application should terminate.

2.5 Module Limitations :

1. Module does not offer a GUI-based interface to an end user.
2. Module does not create multiple proxy objects.
3. Module data is non-persistent.
4. Dimensions of proxy object and real object are not configurable.

The first limitation is overcome in Model 2 discussed in next chapter.

Chapter 3
Proxy Design Pattern Implementation Model 2

Purpose : Model 2 is an enhancement over model 1 for generating GUI interface to an end user for displaying a single proxy object of dimension 100 x 100 displaying a short description. On clicking the proxy object, the corresponding real object of dimension 300 x 400 is generated which displays a detailed information.

3.1 Module Requirement Specification:

The proposed module should be capable of performing the following tasks:

i) Module should generate a Graphical User Interface (GUI) to an end user.

ii) Module should instantiate a proxy object of size 100 x 100.

iii) Module should display a short description to an end user on creating a proxy object

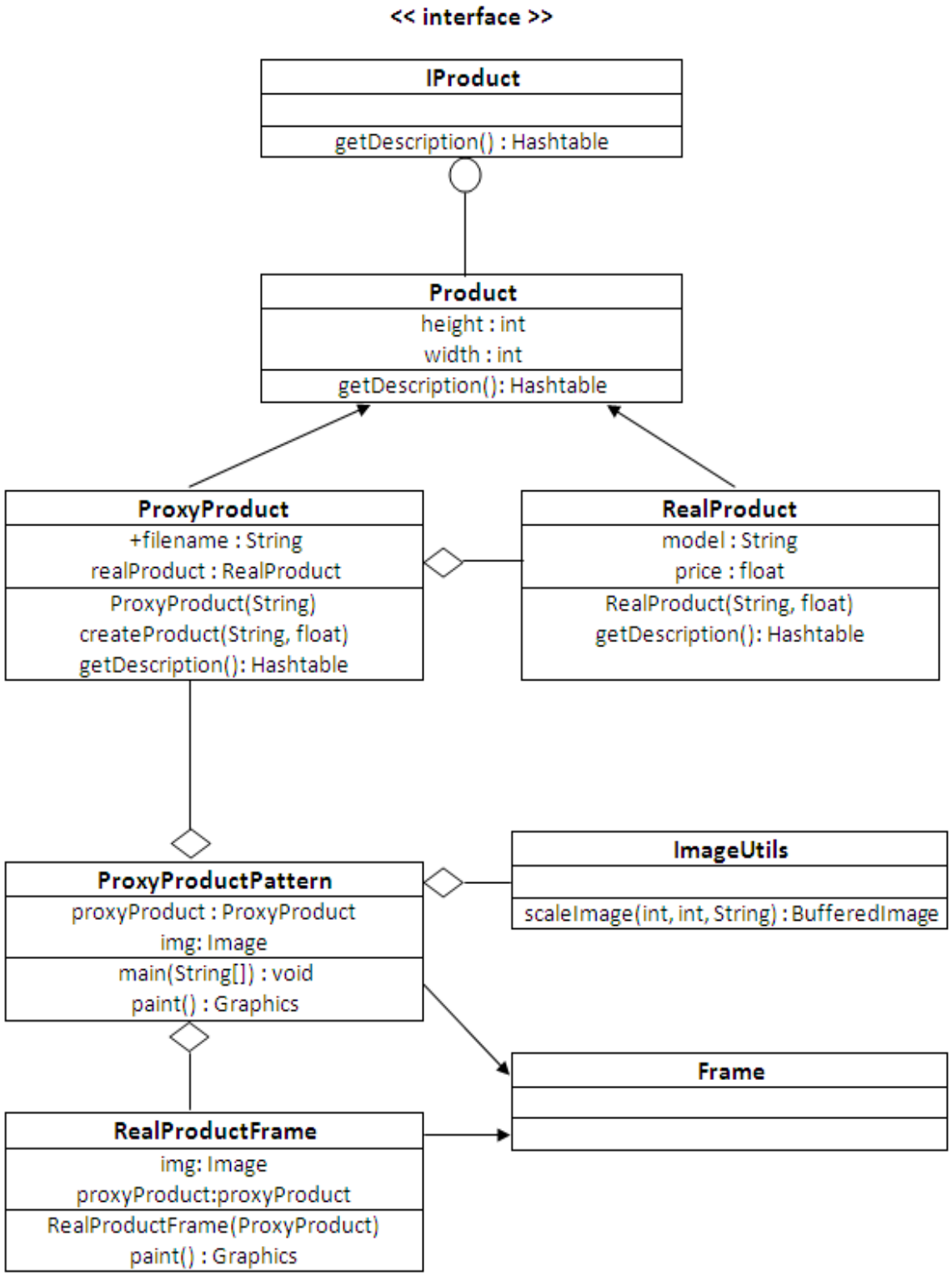
iv) Module should allow the user to click on proxy object to instantiate a real object of size 300 x 400.

v) Module should display a detailed description to an end user on creating a real object

Implementation Details:

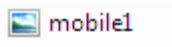
1. The module employs hash table for storing the product features in the form of a name/value pair.
2. The module employs Graphics2D class for scaling the image appropriately for generating proxy and real objects.
3. The module employs java.awt package for creating Frame and employs anonymous inner classes for handling mouse-click event and window event.

3.2 Application Architecture



System Requirements

Resources – Image



3.3 Complete Source Code

IProduct.java

```
import java.util.Hashtable;
interface IProduct
{
   public Hashtable getDescription();
}
```

Product.java

```
import java.util.Hashtable;
class Product implements IProduct
{
/* Attributes of a Product */
  int height;
  int width;
  public Hashtable getDescription()
  {
    return null;
  }
}
```

ProxyProduct.java

```
import java.util.Hashtable;
import java.util.Map;


class ProxyProduct extends Product
{
 /* Attributes of Proxy Product */
  String filename;
  RealProduct realProduct;
```

```
 public ProxyProduct(String filename)
 {
   this.filename=filename;
   height=100;
   width=100;
 }

 public Hashtable getDescription()
 {
   Hashtable features=new Hashtable();
   if (realProduct==null)
   {
     features.put("filename",filename);
     return features;
   }
   else
     return realProduct.getDescription();
 }

 public void createProduct(String model,float price)
 {
  realProduct=new RealProduct(model,price);
 }
}
```


RealProduct.java

```
import java.util.Hashtable;
import java.util.Map;

class RealProduct extends Product
{
```

```
/* Attributes of  Real Product */
   String model;
   float price;

   public RealProduct(String model,float price)
   {
    height=400;
    width=300;
    this.model=model;
    this.price=price;
   }

  public Hashtable getDescription()
  {
/* Populate Hashtable with features of Real Product */
     Hashtable features = new Hashtable();
     features.put("Model",model);
     features.put("Price",price);
     return features;
  }
}
```


ProxyProductPattern.java

```
import javax.swing.ImageIcon;
import java.awt.image.BufferedImage;
import java.awt.Image;
import java.awt.Color;
import java.awt.Graphics2D;
import java.io.File;
import javax.imageio.ImageIO;
import java.awt.RenderingHints;
import java.awt.Frame;
```

```
import java.awt.Graphics;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Font;
import java.awt.TextArea;
import java.util.Hashtable;
/* Class for Scaling Images using 2D Graphics Library */
class ImgUtils
{
public BufferedImage scaleImage(int WIDTH, int HEIGHT, String filename)
  {
/* Scale image with custom width and height attributes */
    BufferedImage bi = null;
    try
    {
       ImageIcon ii = new ImageIcon(filename);//path to image
       bi = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
       Graphics2D g2d = (Graphics2D) bi.createGraphics();
       g2d.addRenderingHints(new
RenderingHints(RenderingHints.KEY_RENDERING,RenderingHints.VALUE_RENDER_QUALITY));
       g2d.drawImage(ii.getImage(), 0, 0, WIDTH, HEIGHT, null);
    }
    catch (Exception e)
    {
       e.printStackTrace();
       return null;
    }
    return bi;
  }
}
```

```
public class ProxyProductPattern extends Frame
{
 ProxyProduct proxyProduct;
 Image img;
 public ProxyProductPattern()
 {
  setTitle("Proxy Pattern");
  setSize(400,400);
  setVisible(true);
  proxyProduct=new ProxyProduct("mobile1.jpg");
  addMouseListener(new MouseAdapter()
  {
   public void mousePressed(MouseEvent e)
   {
    /* Instantiate Real Product on clicking Proxy Product */
     new RealProductFrame(proxyProduct);
   }
  });

  addWindowListener(new WindowAdapter()
  {
   public void windowClosing(WindowEvent e)
   {
    System.exit(0);
   }
  });
 }

 public void paint(Graphics g)
 {
  /* To draw scaled version of image */
   BufferedImage img=new  ImgUtils().scaleImage(
                                proxyProduct.height,proxyProduct.width,proxyProduct.filename);
   g.drawImage(img, 50, 50, this);
```

```
    g.setFont(new Font("Arial",Font.BOLD,12));
    g.drawString(proxyProduct.filename,70,70+proxyProduct.height);
  }

  public static void main(String[] args)
  {
    ProxyProductPattern pattern=new ProxyProductPattern();
  }
}
/* Frame class for diaplying real product */
class RealProductFrame extends Frame
{
  ProxyProduct proxyProduct;
  public RealProductFrame(ProxyProduct proxyProduct)
  {
   setTitle("Image Scaling");
   setSize(600,600);
   setVisible(true);
   this.proxyProduct=proxyProduct;
   proxyProduct.createProduct("Nokia",5000.0f);

   addWindowListener(new WindowAdapter()
   {
     public void windowClosing(WindowEvent e)
     {
       dispose();
     }
   });
  }

  public void paint(Graphics g)
  {
/* To draw scaled version of image */
    BufferedImage img=new ImgUtils().scaleImage(
```

```
                    proxyProduct.realProduct.width,proxyProduct.realProduct.height,"mobile1.jpg");
    g.drawImage(img, 50, 50, this);
    g.setFont(new Font("Arial",Font.BOLD,16));
    Hashtable features=proxyProduct.getDescription();
    int y=70;
    for(Object key: features.keySet())
    {
        String skey=(String)key;
        g.drawString(key + " : " +features.get(key).toString(),150,y+proxyProduct.realProduct.height);
        y+=30;
    }
  }
}
```

Executable Batch File

run.bat

```
javac IProduct.java
pause
javac Product.java
pause
javac RealProduct.java
pause
javac ProxyProduct.java
pause
javac ProxyProductPattern.java
pause
```

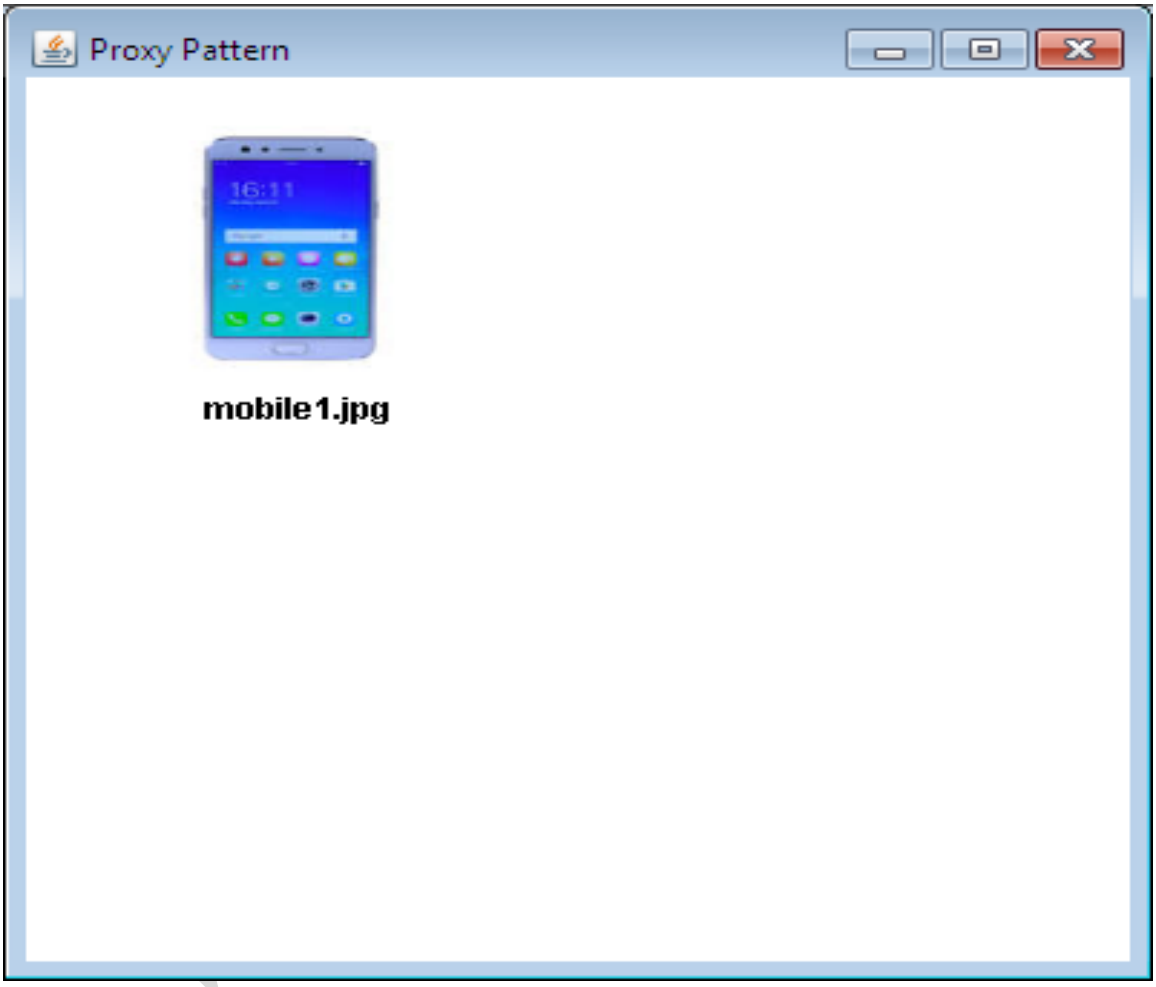

3.4 Test Cases and Screen Shots

Test Cases :

Test Case 1: Execute the application

Desired Output: Proxy object of dimension 100 x 100 is created and its short description displayed below it on a frame window.

Actual Output :



Test Case 2: Click on the proxy product displayed on the Frame window.

Desired Output: Real object of dimension 400 x 300 is created and its detailed description displayed below it on a frame window.

Actual Output :

3.5 Module Limitations:

1. Module does not create multiple proxy objects.
2. Module data is non-persistent.
3. Dimensions of proxy object and real object are not configurable.

The first limitation is overcome in Model 3 discussed in next chapter.

Chapter 4
Proxy Design Pattern Implementation Model 3

Purpose : Model 3 is an enhancement over model 2 for generating GUI interface to an end user for generating multiple proxy objects of dimension 100 x 100 displaying short description of each. On clicking any particular proxy object, the corresponding real object of dimension 300 x 400 is generated which displays detailed description of the features specific to it. The module keeps track of the proxy object clicked by an end user.

4.1 Module Requirement Specification:

The proposed module should be capable of performing the following tasks:

i) Module should generate a Graphical User Interface (GUI) to an end user.

ii) Module should instantiate multiple proxy object of size 100 x 100.

iii) Each row should display exactly four proxy objects.

iv) On creating a proxy object, module should display a short description of features specific to that proxy object to an end user

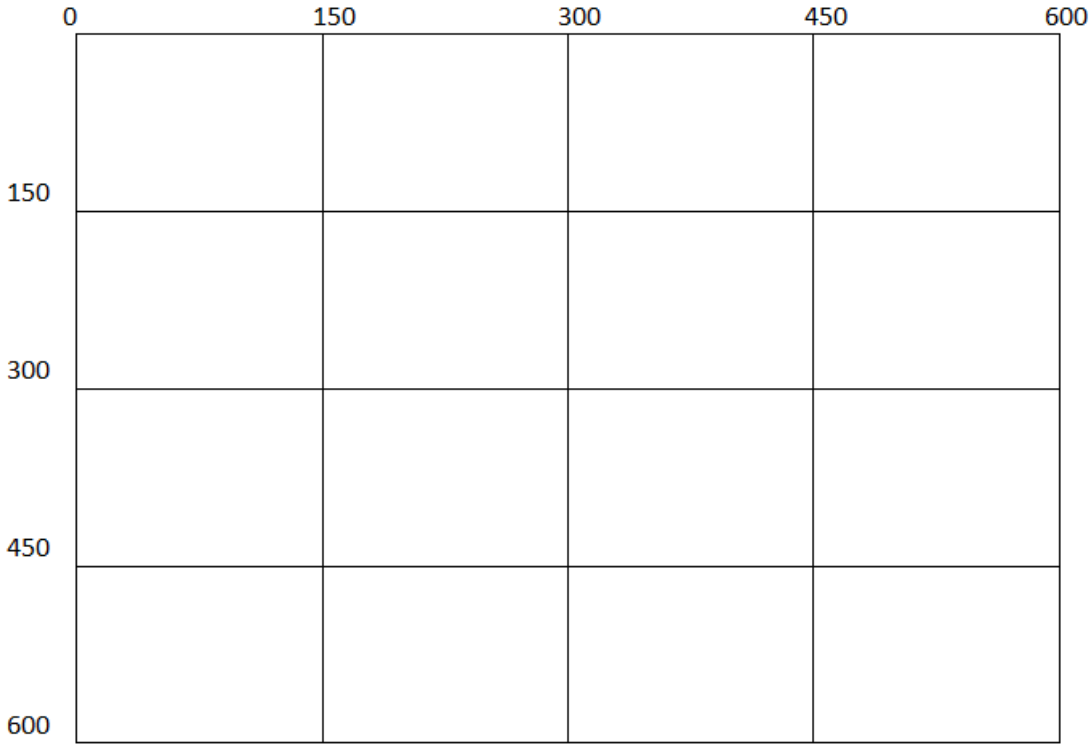
v) Module should allow the user to click on proxy object to instantiate a corresponding real object of size 300 x 400. For this the module should keep track of the proxy object clicked by the user.

vi) Module should display a detailed description to an end user on creating a real object corresponding to the proxy object clicked by the user.

Implementation Details:

1. The module employs hash table for storing the product features in the form of a name/value pair.
2. The module employs Graphics2D class for scaling the image appropriately for generating proxy and real objects.
3. The module employs java.awt package for creating Frame and employs anonymous inner classes for handling mouse-click event and window event.

The following Figure depicts the coordinate system where proxy objects are displayed.



Let x and y represent the coordinates where the user clicks the left mouse button. The following functions are employed to map them to the row index and column index of a grid where the images are displayed.

row_index=[x/150]+1;
column_index=[y/150]+1;

(1)

where [] returns integer part of a number.

Finally, the row_index and the column_index are used for generating the ID of the proxy object employing the formula,

ID = 4 x (row_index – 1) * column_index

(2)

Example :

Suppose the x and y coordinates of the point where the user clicks the mouse are 350, 100, respectively.

Employing the above equ^{n} (1),

row_index=[350/150]+1

= 2 + 1

= 3

column_index=[100/150]+1

= 0 + 1

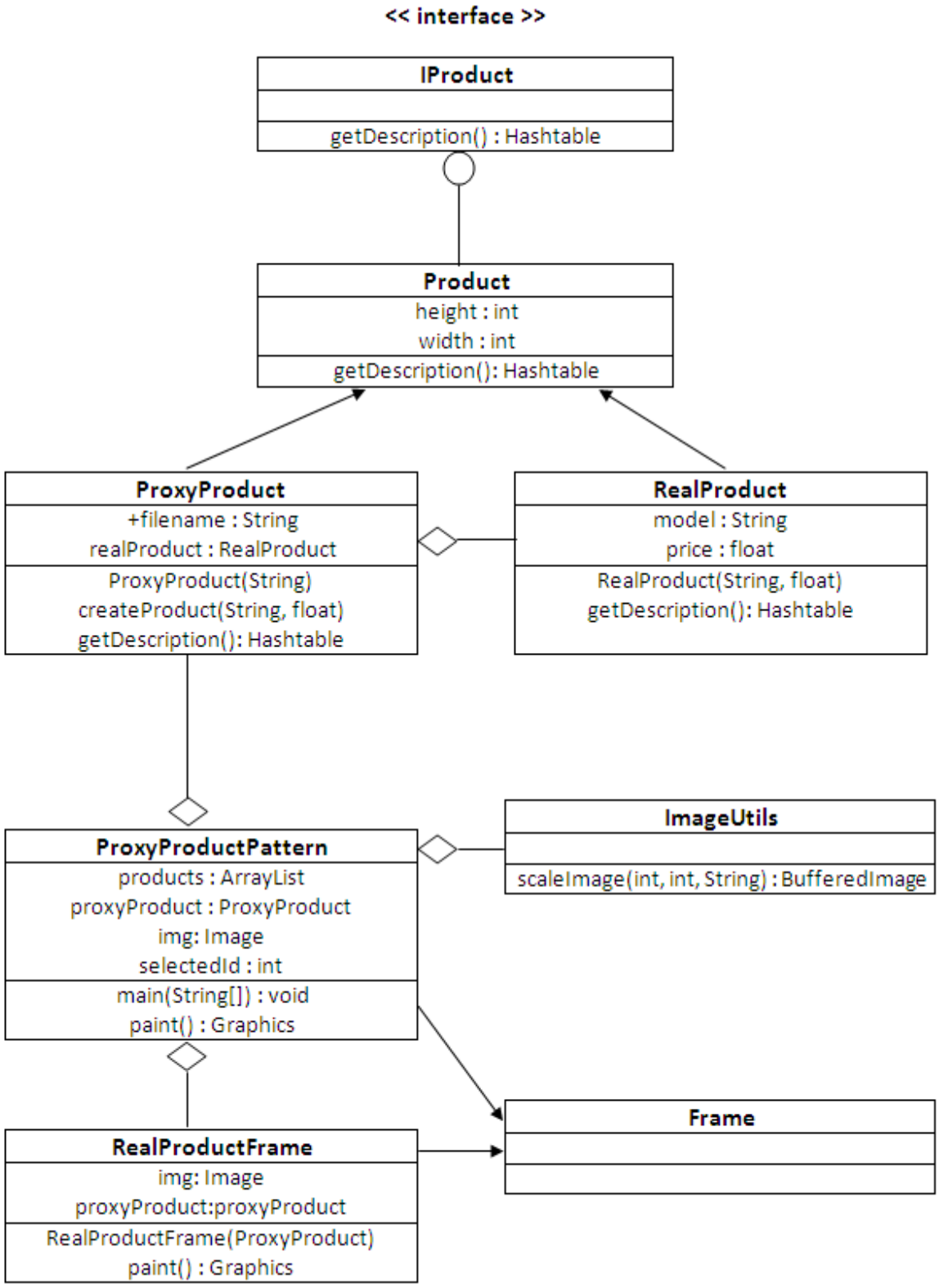
=1

Hence using equ^{n} (2) ID is given by,

ID = 3 + 1 = 4

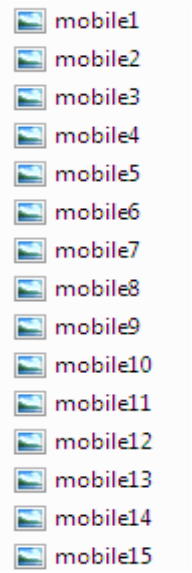
Hence the user has clicked in image whose ID is 4 as depicted in the above Figure.

4.2 Application Architecture



System Requirements:

Resources – Images



4.3 Complete Source Code

IProduct.java

```
import java.util.Hashtable;
interface IProduct
{
  public Hashtable getDescription();
}
```

Product.java

```
import java.util.Hashtable;
class Product implements IProduct
{
/* Attributes of a Product */
  int height;
  int width;
  public Hashtable getDescription()
  {
    return null;
  }
```

```
}
```

ProxyProduct.java

```
import java.util.Hashtable;
import java.util.Map;

class ProxyProduct extends Product
{
/* Attributes of Proxy Product *
  String filename;
  RealProduct realProduct;

  public ProxyProduct(String filename)
  {
    this.filename=filename;
    height=100;
    width=100;
  }

  public Hashtable getDescription()
  {
    Hashtable features=new Hashtable();
/* Create and return features if realProduct is null, otherwise invoke getDescription() method of real
product */
    if (realProduct==null)
    {
      features.put("filename",filename);
      return features;
    }
    else
      return realProduct.getDescription();
  }

  public void createProduct(String model,float price)
```

```
  {
    realProduct=new RealProduct(model,price);
  }
}
```

RealProduct.java

```
import java.util.Hashtable;
import java.util.Map;

class RealProduct extends Product
{
/* Attributes of  Real Product */   String model;
   float price;

   public RealProduct(String model,float price)
   {
    height=700;
    width=700;
    this.model=model;
    this.price=price;
   }

  public Hashtable getDescription()
  {
     Hashtable features = new Hashtable();
     features.put("Model",model);
     features.put("Price",price);
     return features;
  }
}
```

ProxyProductPattern.java

```
import javax.swing.ImageIcon;
import java.awt.image.BufferedImage;
import java.awt.Image;
import java.awt.Color;
import java.awt.Graphics2D;
import java.io.File;
import javax.imageio.ImageIO;
import java.awt.RenderingHints;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Font;
import java.awt.TextArea;
import java.util.Hashtable;
import java.util.ArrayList;

/* Class for Scaling Images using 2D Graphics Library */
class ImgUtils
{
  public BufferedImage scaleImage(int WIDTH, int HEIGHT, String filename)
  {
    BufferedImage bi = null;
    try
    {
/* Scale image with custom width and height attributes */
        ImageIcon ii = new ImageIcon(filename);//path to image
        bi = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = (Graphics2D) bi.createGraphics();
        g2d.addRenderingHints(new RenderingHints(RenderingHints.KEY_RENDERING,
```

```
                                        RenderingHints.VALUE_RENDER_QUALITY));
        g2d.drawImage(ii.getImage(), 0, 0, WIDTH, HEIGHT, null);
    }
    catch (Exception e)
    {
        e.printStackTrace();
        return null;
    }
    return bi;
  }
}

public class ProxyProductPattern extends Frame
{
  ArrayList products;
  ProxyProduct proxyProduct;
  Image img;
  int selectedId=0;
  public ProxyProductPattern()
  {
    setTitle("Proxy Pattern");
    setSize(1000,1000);
    setVisible(true);
    products=new ArrayList<ProxyProduct>();
/* Populate array list with 12 proxy products */
    products.add(new ProxyProduct("mobile1.jpg"));
    products.add(new ProxyProduct("mobile2.jpg"));
    products.add(new ProxyProduct("mobile3.jpg"));
    products.add(new ProxyProduct("mobile4.jpg"));
    products.add(new ProxyProduct("mobile5.jpg"));
    products.add(new ProxyProduct("mobile6.jpg"));
    products.add(new ProxyProduct("mobile7.jpg"));
    products.add(new ProxyProduct("mobile8.jpg"));
    products.add(new ProxyProduct("mobile9.jpg"));
```

```
  products.add(new ProxyProduct("mobile10.jpg"));
  products.add(new ProxyProduct("mobile11.jpg"));
  products.add(new ProxyProduct("mobile12.jpg"));

  addMouseListener(new MouseAdapter()
  {
    public void mousePressed(MouseEvent e)
    {
      if (e.getButton() == MouseEvent.BUTTON1)
      {
          int x=e.getX();
          int y=e.getY();
          x=(x)/150+1;
          y=(y)/150+1;
          /* Use mapping function to map x and y coordinates to id of the proxy product  and instantiate
          appropriate real product  */
          selectedId=4*(y-1)+x;
          new RealProductFrame(proxyProduct,selectedId);
     }
    }
  });
  addWindowListener(new WindowAdapter()
  {
    public void windowClosing(WindowEvent e)
    {
      System.exit(0);
    }
  });
}

public void paint(Graphics g)
{
  int x=50,y=50;
  int count=1;
```

```
    for (Object obj:products)
    {
     proxyProduct=(ProxyProduct)obj;
/* Scale image with custom width and height attributes */
     BufferedImage img=new ImgUtils().scaleImage(proxyProduct.height,
                                    proxyProduct.width,proxyProduct.filename);
     g.drawImage(img, x, y, this);
     g.setFont(new Font("Arial",Font.BOLD,12));
     g.drawString(proxyProduct.filename,x+20,y+20+proxyProduct.height);
     x+=150;
/* Display four proxy products in a single row */
     if ((count % 4)==0)
     {
       y+=150;
       x=50;
     }
     count++;
    }
  }

  public static void main(String[] args)
  {
     ProxyProductPattern pattern=new ProxyProductPattern();
  }
}
/* Frame class for diaplying real product */
class RealProductFrame extends Frame
{
   ProxyProduct proxyProduct;
   int selectedId;

   public RealProductFrame(ProxyProduct proxyProduct, int selectedId)
   {
    setTitle("Image Scaling");
```

```
    setSize(1000,1000);
    setVisible(true);
    this.proxyProduct=proxyProduct;
    this.selectedId=selectedId;
    proxyProduct.createProduct("Nokia"+selectedId,selectedId*1000.0f);

    addWindowListener(new WindowAdapter()
    {
      public void windowClosing(WindowEvent e)
      {
        dispose();
      }
    });
  }

  public void paint(Graphics g)
  {
/* Scale image with custom width and height attributes */
     String filename="Mobile"+selectedId+".jpg";
     BufferedImage img=new ImgUtils().scaleImage(proxyProduct.realProduct.width,
                                          proxyProduct.realProduct.height,filename);
      g.drawImage(img, 50, 50, this);
     g.setFont(new Font("Arial",Font.BOLD,24));
     Hashtable features=proxyProduct.getDescription();
     int y=70;
/* Display detailed description of real product by reading hashtable, features */
     for(Object key: features.keySet())
     {
        String skey=(String)key;
        g.drawString(key + " : " +features.get(key).toString(),300,y+proxyProduct.realProduct.height);
        y+=30;
     }
  }
}
```

run.bat

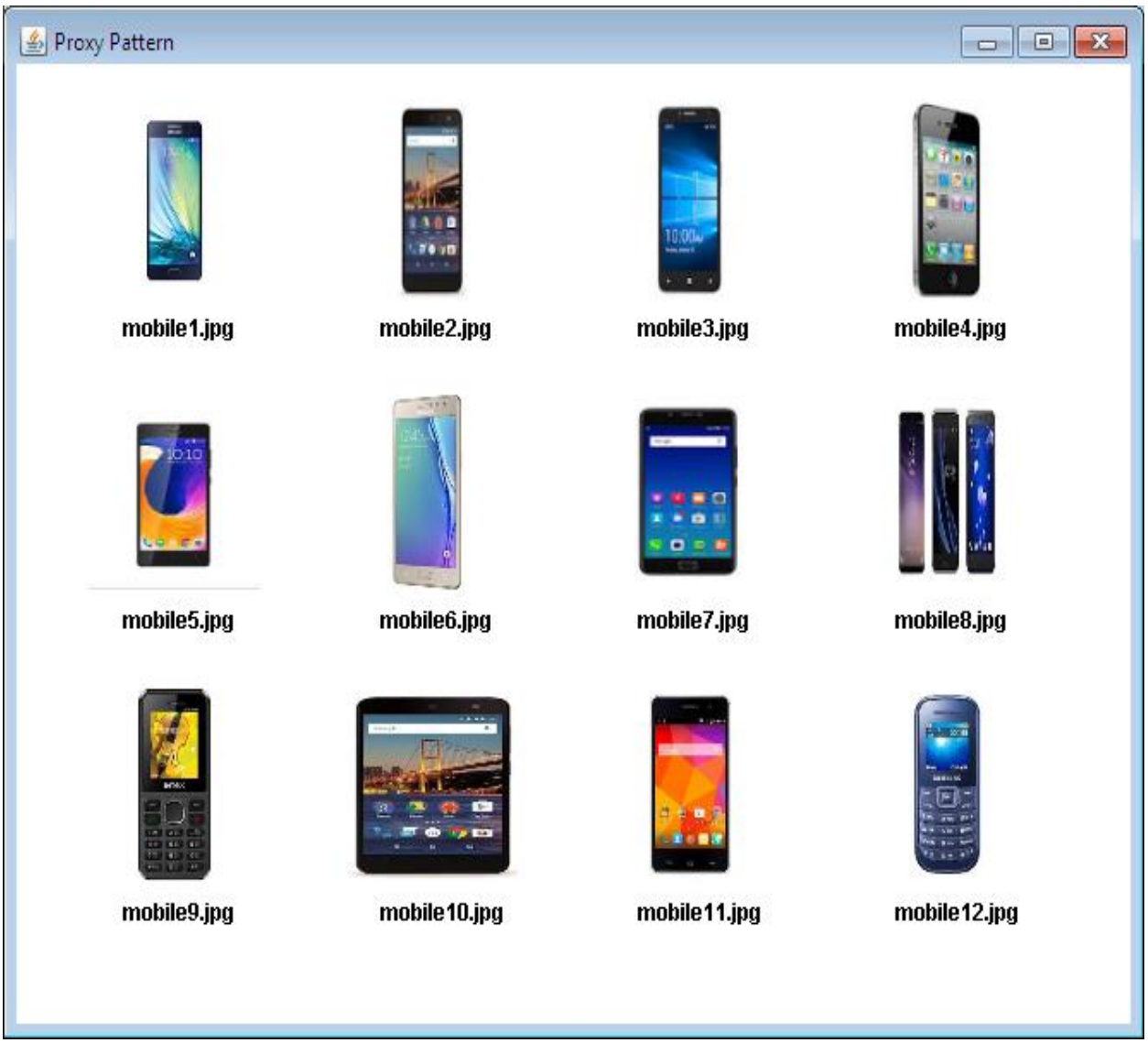
```
javac IProduct.java
pause
javac Product.java
pause
javac RealProduct.java
pause
javac ProxyProduct.java
pause
javac ProxyProductPattern.java
pause
```

4.4 Test Cases and Screen Shots

Test Case 1: Execute the application

Desired Output:16 proxy object of dimension 100 x 100 are created and their short description displayed below each on a frame window. Four proxy objects are displayed in a single row.

Actual Output:



Test Case 2: Click on any proxy product displayed on the Frame window.

Desired Output: Corresponding real object of dimension 400 x 300 is created and its detailed description displayed below it on a frame window.

4.5 Module Limitations :

1. Module data is non-persistent.
2. Dimensions of proxy object and real object are not configurable.
3. No. of proxy objects displayed in a row is now configurable. (Exactly four proxy objects are accommodated in a single row).

The first limitation is overcome in Model4 discussed in next chapter.

Chapter 5
Proxy Design Pattern Implementation Model 4

Purpose : Model4 is an enhancement over model 3 for generating GUI interface to an end user for generating multiple proxy objects of dimension 100 x 100 displaying short description of each. The product information is persistently stored in a MySQL database management system. On clicking any particular proxy object, the corresponding real object of dimension 300 x 400 is generated by loading the requisite information from the back end database management system and detailed description of the features specific to it is displayed to an end user.

5.1 Module Requirement Specification:

The proposed module should be capable of performing the following tasks:

i) Module should be capable of connecting the MySQL database employing Type-IV JDBC driver and retrieving the requisite information from a product table.

ii) Module should generate a Graphical User Interface (GUI) to an end user.

iii) Module should instantiate multiple proxy object of size 100 x 100 by retrieving the current information from the Product table.

iv) The module should employ collection classes for storing product information and feature information specific to the selected product.

v) Each row should display exactly four proxy objects.

vi) On creating a proxy object, module should display a short description of features specific to that proxy object to an end user

vii) Module should allow the user to click on proxy object to instantiate a corresponding real object of size 300 x 400. For this the module should keep track of the proxy object clicked by the user.

viii) Module should display a detailed description to an end user on creating a real object corresponding to the proxy object clicked by the user by retrieving the feature-related information corresponding to the selected product from the back end.

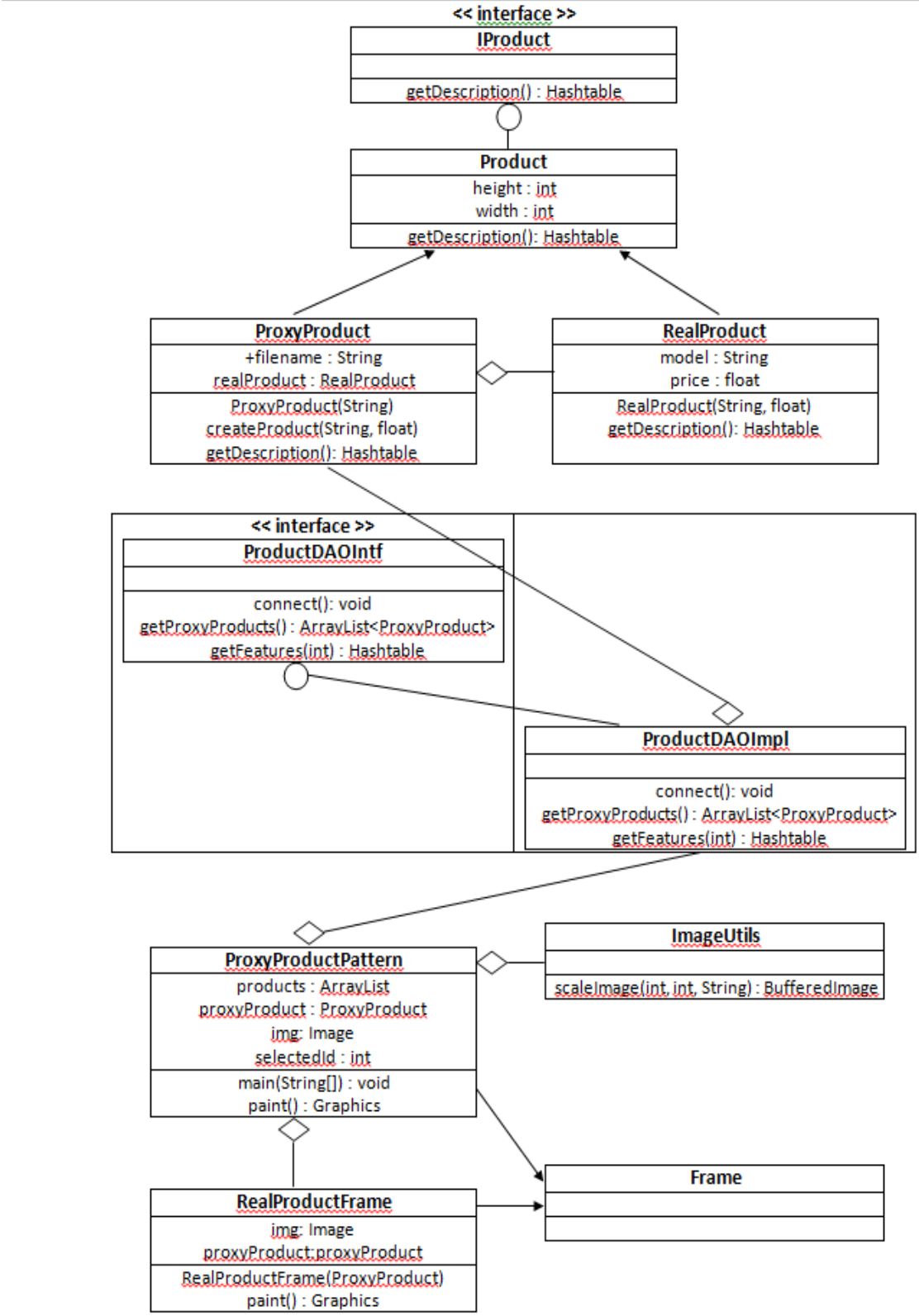
ix) GUI should be synchronized with the backend Database.

a. On inserting a record in the Product table, the proxy object should dynamically appear on the GUI.
b. On deleting a record in the Product table, the proxy object should dynamically disappear from the GUI.
c. On updating a record in the Product table, the proxy object should dynamically update the information displayed on the GUI.

Implementation Details:

1. The module employs hash table for storing the product features in the form of a name/value pair.
2. The module employs Graphics2D class for scaling the image appropriately for generating proxy and real objects.
3. The module employs java.awt package for creating Frame and employs anonymous inner classes for handling mouse-click event and window event.
4. The module employs Type-IV MySQL JDBC driver for connecting to the back end database for bi-directional communication between database and Java application.
5. Database interface is separated out from its implementation to provide a disconnected DAO interface to an end user.
6. The application employs 2-tier architecture separating out presentation tier from the data tier.

5.2 Application Architecture :

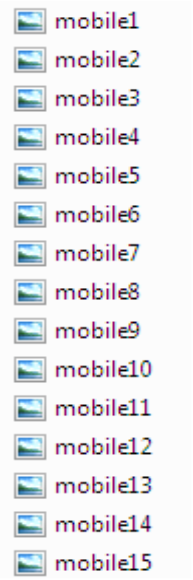


System Requirements:

JAR Files

mysql-connector-java-5.1.15-bin.jar

Resources – Images



5.3 Complete Source Code

IProduct.java

```
import java.util.Hashtable;
interface IProduct
{
  public Hashtable getDescription();
}
```

Product.java

```
import java.util.Hashtable;
class Product implements IProduct
{
/* Attributes of a Product */
  int height;
```

```
  int width;
  public Hashtable getDescription()
  {
    return null;
  }
}
```

ProxyProduct.java

```
import java.util.Hashtable;
import java.util.Map;

class ProxyProduct extends Product
{
/* Attributes of Proxy Product *
  String filename;
  RealProduct realProduct;

  public ProxyProduct(String filename)
  {
     this.filename=filename;
     height=100;
     width=100;
  }

  public Hashtable getDescription()
  {
/* Populate Hashtable with features of Proxy Product */
     Hashtable features=new Hashtable();
     if (realProduct==null)
     {
       features.put("filename",filename);
       return features;
     }
```

```
    else
      return realProduct.getDescription();
  }

  public void createProduct(String model,float price)
  {
   realProduct=new RealProduct(model,price);
  }
}
```

RealProduct.java

```
import java.util.Hashtable;
import java.util.Map;

class RealProduct extends Product
{
/* Attributes of  Real Product */
   String model;
   float price;

   public RealProduct(String model,float price)
   {
    height=700;
    width=700;
    this.model=model;
    this.price=price;
   }

  public Hashtable getDescription()
  {
/* Populate Hashtable with features of Real Product */
     Hashtable features = new Hashtable();
     features.put("Model",model);
```

```
    features.put("Price",price);
    return features;
  }
}
```

ProductDAOIntf.java

```
import java.util.ArrayList;
import java.util.Hashtable;

interface ProductDAOIntf
{
/* DAO Interface */
  public void connect();
  public ArrayList<ProxyProduct> getProxyProducts();
  public Hashtable getFeatures(int id);
}
```

ProductDAOImpl.java

```
import java.util.ArrayList;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Hashtable;

public class ProductDAOImpl implements ProductDAOIntf
{
   Connection con;
   public void connect()
   {
     try
```

```
    {
/* Load MySQL Driver in memory*/
       Class.forName("com.mysql.jdbc.Driver");
       con=DriverManager.getConnection("jdbc:mysql://localhost:3306/designpattern","root","");
    }
    catch(ClassNotFoundException e)
    {
       System.out.println(e);
    }
    catch(SQLException e)
    {
      System.out.println(e);
    }
  }

 public ArrayList<ProxyProduct> getProxyProducts()
 {
/* Retrieve Products from a table and store in an array list */
  ArrayList<ProxyProduct> products=new ArrayList<ProxyProduct>();
  try
  {
   Statement st=con.createStatement();
   ResultSet rs=st.executeQuery("select filename from Product");
   while(rs.next())
   {
    products.add(new ProxyProduct(rs.getString(1)));
   }
  }

  catch(SQLException e)
  {
```

```
    System.out.println(e);
  }
  return products;
 }

 public Hashtable getFeatures(int id)
 {
/* Retrieve features of a selected product and store in a Hashtable */
  Hashtable features = new Hashtable();
  try
  {
    Statement st=con.createStatement();
    ResultSet rs=st.executeQuery("select model,price from Product WHERE id = "+id);
    if (rs.next())
    {
        features.put("Model",rs.getString(1));
        features.put("Price",rs.getFloat(2));
        return features;
   }
  }

  catch(SQLException e)
  {
    System.out.println(e);
  }
  return null;
 }
}
```

ProxyProductPattern.java

```
import javax.swing.ImageIcon;
import java.awt.image.BufferedImage;
import java.awt.Image;
import java.awt.Color;
import java.awt.Graphics2D;
import java.io.File;
import javax.imageio.ImageIO;
import java.awt.RenderingHints;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Font;
import java.awt.TextArea;
import java.util.Hashtable;
import java.util.ArrayList;

/* Class for Scaling Images using 2D Graphics Library */
class ImgUtils
{
  public BufferedImage scaleImage(int WIDTH, int HEIGHT, String filename)
  {
    BufferedImage bi = null;
    try
    {
/* Scale image with custom width and height attributes */
        ImageIcon ii = new ImageIcon(filename);//path to image
        bi = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = (Graphics2D) bi.createGraphics();
        g2d.addRenderingHints(newRenderingHints( RenderingHints.KEY_RENDERING,
```

```
                                        RenderingHints.VALUE_RENDER_QUALITY));
      g2d.drawImage(ii.getImage(), 0, 0, WIDTH, HEIGHT, null);
    }
    catch (Exception e)
    {
      e.printStackTrace();
      return null;
    }
    return bi;
  }
 }

public class ProxyProductPattern extends Frame
{
  ArrayList products=null;
  ProxyProduct proxyProduct;
  Image img;
  int selectedId=0;
  public ProxyProductPattern()
  {
    setTitle("Proxy Pattern");
    setSize(1000,1000);
    setVisible(true);
    addMouseListener(new MouseAdapter()
    {
      public void mousePressed(MouseEvent e)
      {
        /* Use mapping function to map x and y coordinates to id of the proxy product  and instantiate
           appropriate real product  */
        int x=e.getX();
        int y=e.getY();
        x=(x)/150+1;
        y=(y)/150+1;
        selectedId=4*(y-1)+x;
```

```
        new RealProductFrame(proxyProduct,selectedId);
      }
    });

    addWindowListener(new WindowAdapter()
    {
      public void windowClosing(WindowEvent e)
      {
        System.exit(0);
      }
    });
  }

  public void paint(Graphics g)
  {
    int x=50,y=50;
    int count=1;
    ProductDAOIntf productDAO=new ProductDAOImpl();
    productDAO.connect();
    products=productDAO.getProxyProducts();

    for (Object obj:products)
    {
     proxyProduct=(ProxyProduct)obj;
/* Scale image with custom width and height attributes */
     BufferedImage img=new ImgUtils().scaleImage(proxyProduct.height,
                                  proxyProduct.width,proxyProduct.filename);
     g.drawImage(img, x, y, this);
     g.setFont(new Font("Arial",Font.BOLD,12));
     g.drawString(proxyProduct.filename,x+20,y+20+proxyProduct.height);
     x+=150;
     /* Display four proxy products in a single row */
     if ((count % 4)==0)
     {
```

```
      y+=150;
      x=50;
     }
     count++;
    }
  }

  public static void main(String[] args)
  {
     ProxyProductPattern pattern=new ProxyProductPattern();
  }
}
/* Frame class for diaplying real product */
class RealProductFrame extends Frame
{
   ProxyProduct proxyProduct;
   int selectedId;

   public RealProductFrame(ProxyProduct proxyProduct, int selectedId)
  {
    setTitle("Image Scaling");
    setSize(1000,1000);
    setVisible(true);
    this.proxyProduct=proxyProduct;
    this.selectedId=selectedId;
    ProductDAOIntf productDAO=new ProductDAOImpl();
    productDAO.connect();
    Hashtable features=productDAO.getFeatures(selectedId);
    proxyProduct.createProduct(features);
    addWindowListener(new WindowAdapter()
    {
      public void windowClosing(WindowEvent e)
      {
        dispose();
```

```
    }
  });
 }
 public void paint(Graphics g)
 {
   String filename="Mobile"+selectedId+".jpg";
   BufferedImage img=new ImgUtils().scaleImage(proxyProduct.realProduct.width,
                                               proxyProduct.realProduct.height,filename);
   g.drawImage(img, 50, 50, this);
   g.setFont(new Font("Arial",Font.BOLD,24));
   Hashtable features=proxyProduct.getDescription();
   int y=70;
 /* Display detailed description of real product by reading hashtable, features */
   for(Object key: features.keySet())
   {
     String skey=(String)key;
     g.drawString(key + " : " +features.get(key).toString(),300,y+proxyProduct.realProduct.height);
     y+=30;
   }
 }
}
```

Executable Batch File: run.bat

```
javac IProduct.java
pause
javac Product.java
pause
javac RealProduct.java
pause
javac ProxyProduct.java
pause
javac ProxyProductPattern.java
pause
```

5.4 Test Cases and Screen Shots

Test Case 1: Execute the application

Desired Output : Proxy objects of dimension 100 x 100 are created by loading the information from the Product table and their short description displayed below each on a frame window. Four proxy objects are displayed in a single row.

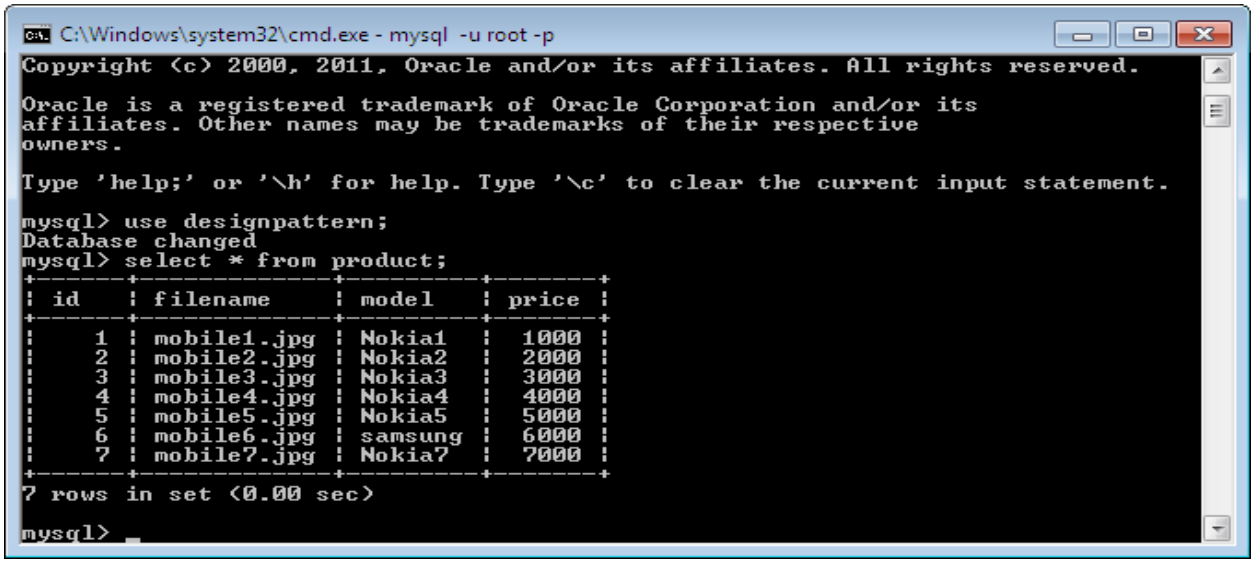
Actual Output :

Test Case 2: Click on any proxy product displayed on the Frame window.

Desired Output : Corresponding real object of dimension 400 x 300 is created by reading the relevant information from the Product table and its detailed description displayed below it on a frame window.

Actual Output :

The content of the product table in MySQL database is depicted below:



Test Case 3: Insert two more rows in the table and add the corresponding images to the folder and re-execute the application.

```
insert into product values(8,"mobile8.jpg","Nokia6",8000.0);
insert into product values(9,"mobile9.jpg","Nokia7",9000.0);
```

Desired Output : The newly added products should dynamically appear on the frame window.

Actual Output :

Double-click on the newly added proxy product. The real product with detailed information should be displayed.

Test Case 4: Delete any row from the Product table and re-execute the application.

Desired Output : The deleted product should dynamically disappear from the frame window.

Test Case 5: Update the name and price of any mobile and re-execute the program.

Update product set model='Nokia6',price=6500 where id=6;

Desired Output : The updated product information should dynamically appear on the frame window.

The SQL script for creating a database designpattern, and creation of Product table and populating it with some data is shown below:

Script.txt

```
create database designpattern;

use designpattern;

create table Product(id int, filename char(50),model char(50), price float);
```

```
insert into product values(1,"mobile1.jpg","Nokia1",1000.0);
insert into product values(2,"mobile2.jpg","Nokia2",2000.0);
insert into product values(3,"mobile3.jpg","Nokia3",3000.0);
insert into product values(4,"mobile4.jpg","Nokia4",4000.0);
insert into product values(5,"mobile5.jpg","Nokia5",5000.0);
insert into product values(6,"mobile6.jpg","Nokia6",6000.0);
insert into product values(7,"mobile7.jpg","Nokia7",7000.0);
insert into product values(8,"mobile8.jpg","Nokia6",8000.0);
insert into product values(9,"mobile9.jpg","Nokia7",9000.0);

Update product set model='Nokia6',price=6500 where id=6;
```


5.5 Module Limitations :

1. Module is database dependent. Only MySQL database is supported.
2. Dimensions of proxy object and real object are not configurable.
3. No. of proxy objects displayed in a row is now configurable. (Exactly four proxy objects are accomodated in a single row).

The second and third limitations are overcome in Model 5 and Model 6, respectively discussed in next chapters where the configuration information is stored in XML and JSON format.

Chapter 6
Proxy Design Pattern Implementation Model 5

Purpose : Model 5 is an enhancement over model 4 for generating GUI interface to an end user for generating multiple proxy objects of custom dimension displaying short description of each. The product information is persistently stored in a MySQL database management system. On clicking any particular proxy object, the corresponding real object of custom dimension is generated by loading the requisite information from the back end database management system and detailed description of the features specific to it is displayed to an end user. The image configuration information pertaining to proxy image dimension, real image dimension, no. of proxy images to be displayed in a single row, is stored in XML format which is parsed using JDOM parser available in Java.

6.1 Module Requirement Specification:

The proposed module should be capable of performing the following tasks:

i) Module should be capable of connecting the MySQL database employed Type-IV JDBC driver and retrieving the requisite information from a product table.

ii) Module should generate a Graphical User Interface (GUI) to an end user.

iii) Module should instantiate multiple proxy object of custom dimension stored in config.json file, by retrieving the current information from the Product table.

iv) The module should employ collection classes for storing product information and feature information specific to the selected product.

v) Each row should display no. of proxy objects as configured in config.json file.

vi) On creating a proxy object, module should display a short description of features specific to that proxy object to an end user

vii) Module should allow the user to click on proxy object to instantiate a corresponding real object of custom dimension stored in config.xml file. For this the module should keep track of the proxy object clicked by the user.

viii) Module should display a detailed description to an end user on creating a real object corresponding to the proxy object clicked by the user by retrieving the feature-related information corresponding to the selected product from the back end.

ix) GUI should be synchronized with the backend Database.

a. On inserting a record in the Product table, the proxy object should dynamically appear on the GUI.

b. On deleting a record in the Product table, the proxy object should dynamically disappear from the GUI.

c. On updating a record in the Product table, the proxy object should dynamically update the information displayed on the GUI.

Implementation Details:

1. The module employs hash table for storing the product features in the form of a name/value pair.
2. The module employs Graphics2D class for scaling the image appropriately for generating proxy and real objects.
3. The module employs java.awt package for creating Frame and employs anonymous inner classes for handling mouse-click event and window event.
4. The module employs the JSON Simple parser in Java for retrieving the configuration information stored in config.json file.

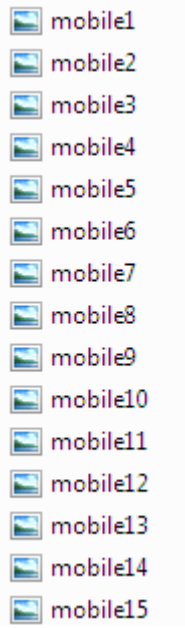
System Requirements

JAR Files


json-simple-1.1.1.jar


mysql-connector-java-5.1.15-bin.jar

Resources – Images

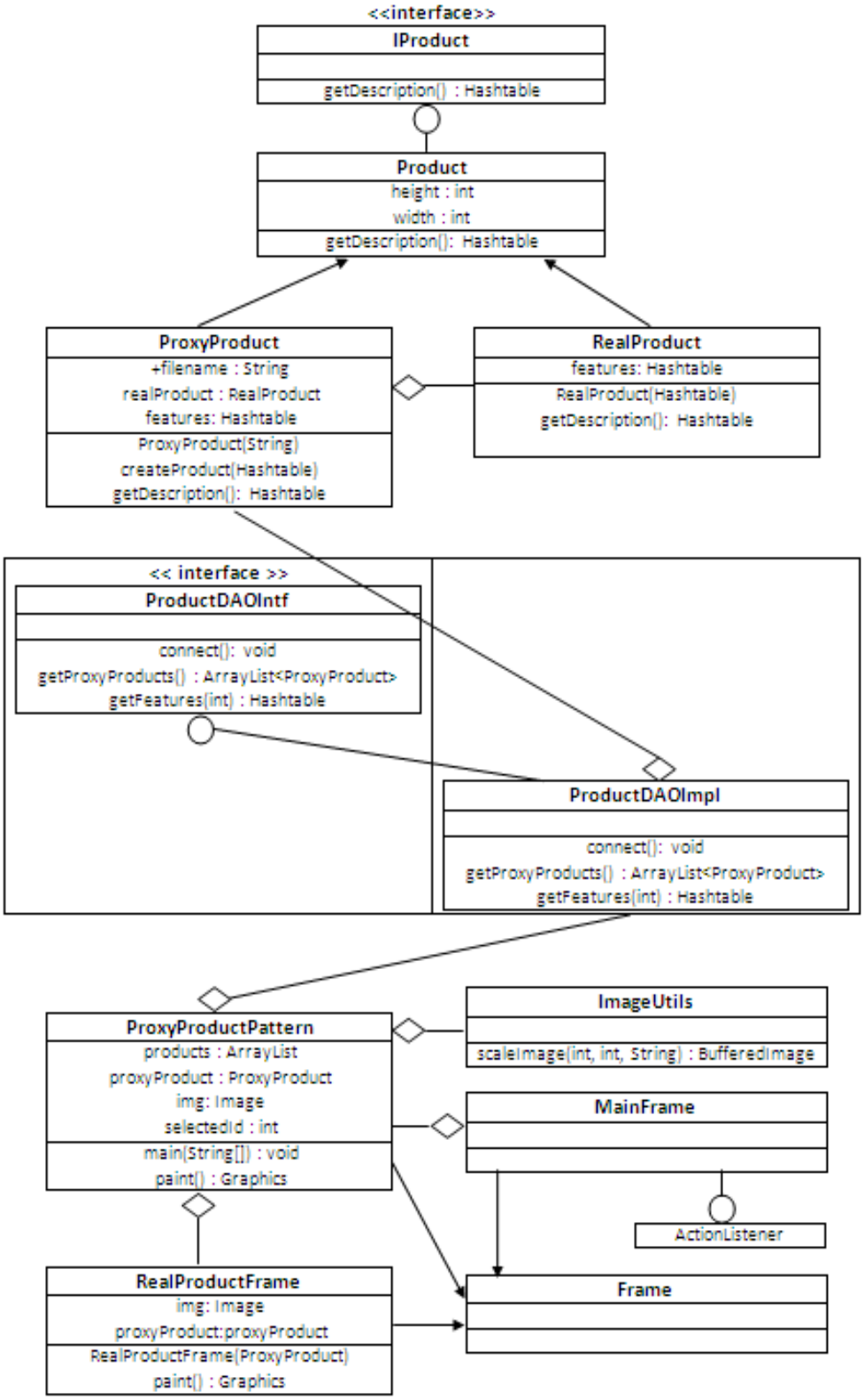


SQL Script

Script.txt

```
create database designpattern;
use designpattern;
create table Product(id int, filename char(50),model char(50), price float);

insert into product values(1,'mobile1.jpg','Nokia1',1000.0);
insert into product values(2,'mobile2.jpg','Nokia2',2000.0);
insert into product values(3,'mobile3.jpg','Nokia3',3000.0);
insert into product values(4,'mobile4.jpg','Nokia4',4000.0);
insert into product values(5,'mobile5.jpg','Nokia5',5000.0);
insert into product values(6,'mobile6.jpg','Nokia6',6000.0);
insert into product values(7,'mobile7.jpg','Nokia7',7000.0);
insert into product values(8,'mobile8.jpg','Nokia6',8000.0);
insert into product values(9,'mobile9.jpg','Nokia7',9000.0);

Update product set model='Nokia6',price=6500 where id=6;
```

6.2 Application Architecture :



6.3 Complete Source Code

IProduct.java

```
import java.util.Hashtable;
interface IProduct
{
  public Hashtable getDescription();
}
```

Product.java

```
import java.util.Hashtable;


class Product implements IProduct
{
/* Attributes of a Product */
  int height;
  int width;
  public Hashtable getDescription()
  {
    return null;
  }
}
```


ProxyProduct.java

```
import java.util.Hashtable;
import java.util.Map;
import java.io.File;
import java.io.IOException;
import java.util.List;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
```

```
import org.jdom2.input.SAXBuilder;

class ProxyProduct extends Product
{
/* Attributes of Proxy Product */
  String filename;
  RealProduct realProduct;
  Hashtable features;

  public ProxyProduct(String filename)
  {
            this.filename=filename;
           /* Parse config.xml file for retrieving image configuration information */
            SAXBuilder builder = new SAXBuilder();
            File xmlFile = new File("config.xml");
            try
            {
                  /* Obtain reference to the document root and traverse through the
                     child nodes of proxyImage node*/
                  Document document = (Document) builder.build(xmlFile);
                  Element rootNode = document.getRootElement();
                  List list = rootNode.getChildren("proxyImage");

                  for (int i = 0; i < list.size(); i++) {
                    Element node = (Element) list.get(i);
                    height=Integer.parseInt(node.getChildText("height"));
                    if (height>250)
                          height=250;
                    width=Integer.parseInt(node.getChildText("width"));
                    if (width>250)
                          width=250;
```

```
                }
            }
            catch (IOException io)
            {
                System.out.println(io.getMessage());
            }
            catch (JDOMException jdomex)
            {
                System.out.println(jdomex.getMessage());
            }
    }

    public Hashtable getDescription()
    {
      Hashtable features=new Hashtable();
      if (realProduct==null)
      {
        features.put("filename",filename);
        return features;
      }
      else
        return realProduct.getDescription();
    }

    public void createProduct(Hashtable features)
    {
     realProduct=new RealProduct(features);
    }
}
```

RealProduct.java

```
import java.util.Hashtable;
import java.util.Map;
import java.io.File;
import java.io.IOException;
import java.util.List;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;

/* Frame class for diaplying real product */
class RealProduct extends Product
{
  Hashtable features;

  public RealProduct(Hashtable features)
  {
        this.features=features;
        SAXBuilder builder = new SAXBuilder();
        File xmlFile = new File("config.xml");
        try {
              /* Obtain reference to the document root and traverse through the child nodes of
                 proxyImage node*/
              Document document = (Document) builder.build(xmlFile);
              Element rootNode = document.getRootElement();
              List list = rootNode.getChildren("realImage");

              for (int i = 0; i < list.size(); i++) {
                Element node = (Element) list.get(i);
                height=Integer.parseInt(node.getChildText("height"));
```

```
                    if (height>1200)
                            height=1200;
                     width=Integer.parseInt(node.getChildText("width"));
                     if (width>1000)
                            width=1000;
                    }
        }
         catch (IOException io)
         {
                    System.out.println(io.getMessage());
         }
         catch (JDOMException jdomex)
        {
                    System.out.println(jdomex.getMessage());
         }
   }

  public Hashtable getDescription()
  {
    return features;
  }

  public void setDescription(Hashtable features)
  {
    this.features=features;
  }
}
```

ProductDAOImpl.java

```
import java.util.ArrayList;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Hashtable;

interface ProductDAOIntf
{
  public void connect();
  public ArrayList<ProxyProduct> getProxyProducts();
  public Hashtable getFeatures(int id);
}

public class ProductDAOImpl implements ProductDAOIntf
{
   Connection con;
   public void connect()
   {
     try
     {
       /* Load MySQL Driver in memory*/
       Class.forName("com.mysql.jdbc.Driver");
       con=DriverManager.getConnection("jdbc:mysql://localhost:3306/designpattern","root","");
     }
     catch(ClassNotFoundException e)
     {
       System.out.println(e);
```

```
    }
    catch(SQLException e)
    {
      System.out.println(e);
    }
  }

 public ArrayList<ProxyProduct> getProxyProducts()
 {
/* Retrieve Products from a table and store in an array list */
  ArrayList<ProxyProduct> products=new ArrayList<ProxyProduct>();
  try
  {
    Statement st=con.createStatement();
    ResultSet rs=st.executeQuery("select filename from Product");
    while(rs.next())
    {
     products.add(new ProxyProduct(rs.getString(1)));
    }
  }
  catch(SQLException e)
  {
     System.out.println(e);
  }
  return products;
 }

 public Hashtable getFeatures(int id)
 {
   String cname="";
   String tname="";
```

```
  Hashtable features = new Hashtable();
/* Retrieve features of a selected product and store in a Hashtable */
 try
 {
   Statement st=con.createStatement();
   ResultSet rs=st.executeQuery("select * from Product WHERE id = "+id);
  /* Retrieve ResultSet Metadata */
   ResultSetMetaData rsmd=rs.getMetaData();
   int nCols=rsmd.getColumnCount();
   if (rs.next())
   {
     for(int i=1;i<=nCols;i++)
     {
      cname=rsmd.getColumnName(i);
      tname=rsmd.getColumnTypeName(i);
      if (tname.equals("INT"))
        features.put(cname,rs.getInt(i));
      if (tname.equals("CHAR"))
        features.put(cname,rs.getString(i));
      if (tname.equals("FLOAT"))
        features.put(cname,rs.getFloat(i));
  }
   return features;
  }
 }
 catch(SQLException e)
 {
   System.out.println(e);
 }
 return null;
}
```

```
  public static void main(String[] args)
  {
     ProductDAOIntf p=new ProductDAOImpl();
     p.connect();
     p.getFeatures(1);
  }
}
```

MainFrame.java

```
import javax.swing.ImageIcon;
import java.awt.image.BufferedImage;
import java.awt.Image;
import java.awt.Color;
import java.awt.Graphics2D;
import java.io.File;
import java.io.FileOutputStream;
import javax.imageio.ImageIO;
import java.awt.RenderingHints;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Font;
import java.awt.TextArea;
import java.awt.FlowLayout;
import java.awt.Color;
import java.util.Hashtable;
import java.util.ArrayList;
```

```
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JTextField;
import javax.swing.JOptionPane;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.io.File;
import java.io.IOException;
import java.util.List;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;

/* Class for Scaling Images using 2D Graphics Library */
class ImgUtils
{
 public BufferedImage scaleImage(int WIDTH, int HEIGHT, String filename)
 {
  BufferedImage bi = null;
  try
  {
     /* Scale image with custom width and height attributes */
     ImageIcon ii = new ImageIcon(filename);//path to image
     bi = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
     Graphics2D g2d = (Graphics2D) bi.createGraphics();
     g2d.addRenderingHints(new RenderingHints(RenderingHints.KEY_RENDERING,
```

```
                                        RenderingHints.VALUE_RENDER_QUALITY));
      g2d.drawImage(ii.getImage(), 0, 0, WIDTH, HEIGHT, null);
    }
    catch (Exception e)
    {
      e.printStackTrace();
      return null;
    }
    return bi;
  }
 }

class ProxyProductPattern extends Frame
{
  ArrayList products=null;
  ProxyProduct proxyProduct;
  Image img;
  int selectedId=0;
  int count=0;
  public ProxyProductPattern()
  {
    setTitle("Proxy Pattern");
    setSize(1200,1000);
    setVisible(true);
    SAXBuilder builder = new SAXBuilder()
    File xmlFile = new File("config.xml");
    try
    {
          /* Parse config.xml file for retrieving image configuration information */
         /* Obtain reference to the document root and traverse through the child nodes of
            proxyImage*/
```

```
                Document document = (Document) builder.build(xmlFile);
                Element rootNode = document.getRootElement();
                List list = rootNode.getChildren("proxyImage");

                for (int i = 0; i < list.size(); i++) {
                  Element node = (Element) list.get(i);
                  count=Integer.parseInt(node.getChildText("count"));
    }
 }
 catch (IOException io)
 {
       System.out.println(io.getMessage());
 }
  catch (JDOMException jdomex)
 {
      System.out.println(jdomex.getMessage());
  }
addMouseListener(new MouseAdapter()
{
  public void mousePressed(MouseEvent e)
  {
 /* Use mapping function to map x and y coordinates to id of the proxy product
    and instantiate appropriate real product  */
    int x=e.getX();
    int y=e.getY();
    x=(x)/(1200/count)+1;
    y=(y)/(proxyProduct.height+50)+1;
    selectedId=count*(y-1)+x;
    new RealProductFrame(proxyProduct,selectedId);
  }
});
```

```
  addWindowListener(new WindowAdapter()
  {
    public void windowClosing(WindowEvent e)
    {
      dispose();
    }
  });
}

public void paint(Graphics g)
{
  int x=50,y=50;
  int icount=1;
  ProductDAOIntf productDAO=new ProductDAOImpl();
  productDAO.connect();
  products=productDAO.getProxyProducts();

  for (Object obj:products)
  {
   proxyProduct=(ProxyProduct)obj;
   BufferedImage img=new ImgUtils().scaleImage(proxyProduct.height,
                                    proxyProduct.width,proxyProduct.filename);
   g.drawImage(img, x, y, this);
   g.setFont(new Font("Arial",Font.BOLD,12));
   g.drawString(proxyProduct.filename,x+proxyProduct.width/3,y+20+proxyProduct.height);
   x+=1200/count;
   /* Display configured no. of proxy products in a single row */
   if ((icount % count)==0)
   {
     y+=proxyProduct.height+50;
```

```
      x=50;
     }
     icount++;
    }
  }
}
/* Frame class for diaplying real product */
class RealProductFrame extends Frame
{
   ProxyProduct proxyProduct;
   int selectedId;

   public RealProductFrame(ProxyProduct proxyProduct, int selectedId)
   {
    setTitle("Image Scaling");
    setSize(1000,1000);
    setVisible(true);
    this.proxyProduct=proxyProduct;
    this.selectedId=selectedId;
    ProductDAOIntf productDAO=new ProductDAOImpl();
    productDAO.connect();
    Hashtable features=productDAO.getFeatures(selectedId);
    proxyProduct.createProduct(features);
    addWindowListener(new WindowAdapter()
    {
      public void windowClosing(WindowEvent e)
      {
        dispose();
      }
    });
  }
```

```
  public void paint(Graphics g)
  {
/* Scale image with custom width and height attributes */
     String filename="Mobile"+selectedId+".jpg";
     BufferedImage img=new ImgUtils().scaleImage(proxyProduct.realProduct.width,
                                                 proxyProduct.realProduct.height,filename);
     g.drawImage(img, 50, 50, this);
     g.setFont(new Font("Arial",Font.BOLD,24));
     Hashtable features=proxyProduct.getDescription();
     int y=70;
     int x=50;
     for(Object key: features.keySet())
     {
          String skey=(String)key;
       g.drawString(key + " : " +features.get(key).toString(),
                        proxyProduct.realProduct.width/3+50,y+proxyProduct.realProduct.height);
       y+=30;
     }
  }
}
/* Configuration class for storing configuration information in XML File */
class Configuration extends JFrame implements ActionListener
{
  JLabel lbl1,lbl2,lbl3,lbl4,lbl5;
  JTextField tf1,tf2,tf3,tf4,tf5;
  JButton b;
  int pheight,pwidth,pcount,rheight,rwidth;
  public Configuration()
 {
    lbl1=new JLabel("Proxy Image Height : ");
```

```
lbl1.setBounds(100,50,400,30);
lbl1.setFont(new Font("Arial",Font.BOLD,24));
lbl1.setForeground(Color.red);

tf1=new JTextField(50);
tf1.setBounds(350,50,100,30);

lbl2=new JLabel("Proxy Image Width :");
lbl2.setBounds(100,150,400,30);
lbl2.setFont(new Font("Arial",Font.BOLD,24));
lbl2.setForeground(Color.red);

tf2=new JTextField(50);
tf2.setBounds(350,150,100,30);

lbl3=new JLabel("Real Image Height :");
lbl3.setBounds(100,250,400,30);
lbl3.setFont(new Font("Arial",Font.BOLD,24));
lbl3.setForeground(Color.red);

tf3=new JTextField(50);
tf3.setBounds(350,250,100,30);

lbl4=new JLabel("Real Image Width :");
lbl4.setBounds(100,350,400,30);
lbl4.setFont(new Font("Arial",Font.BOLD,24));
lbl4.setForeground(Color.red);

tf4=new JTextField(50);
tf4.setBounds(350,350,100,30);
```

```
lbl5=new JLabel("No. of Proxy Objects/Row :");
lbl5.setBounds(100,450,400,30);
lbl5.setFont(new Font("Arial",Font.BOLD,20));
lbl5.setForeground(Color.red);

tf5=new JTextField(50);
tf5.setBounds(350,450,100,30);

b=new JButton("Store Config Info");
b.setBounds(150,550,200,30);
b.addActionListener(this);

SAXBuilder builder = new SAXBuilder();
File xmlFile = new File("config.xml");
try
{
            /* Parse config.xml file for retrieving image configuration information */
              Document document = (Document) builder.build(xmlFile);
              Element rootNode = document.getRootElement();
            /* Obtain reference to the document root and traverse through the child nodes*/
              List list = rootNode.getChildren("proxyImage");
              for (int i = 0; i < list.size(); i++)
              {
                 Element node = (Element) list.get(i);
                 pheight=Integer.parseInt(node.getChildText("height"));
                  pcount=Integer.parseInt(node.getChildText("count"));
                  if (pheight>250)
                              pheight=250;
                 pwidth=Integer.parseInt(node.getChildText("width"));
                 if (pwidth>250)
                              pwidth=250;
```

```
            }
            list = rootNode.getChildren("realImage");
            for (int i = 0; i < list.size(); i++) {
                Element node = (Element) list.get(i);
                rheight=Integer.parseInt(node.getChildText("height"));
                if (rheight>1200)
                        rheight=1200;
                rwidth=Integer.parseInt(node.getChildText("width"));
                if (rwidth>1000)
                        rwidth=1000;
              }
    }
    catch (IOException io)
    {
            System.out.println(io.getMessage());
    }
    catch (JDOMException jdomex)
    {
            System.out.println(jdomex.getMessage());
    }
tf1.setText(String.valueOf(pheight));
tf2.setText(String.valueOf(pwidth));
tf3.setText(String.valueOf(rheight));
tf4.setText(String.valueOf(rwidth));
tf5.setText(String.valueOf(pcount));
add(lbl1);
add(tf1);
add(lbl2);
add(tf2);
add(lbl3);
add(tf3);
```

```
   add(lbl4);
   add(tf4);
   add(lbl5);
   add(tf5);
   add(b);
   setSize(800,800);
   setLayout(null);
   setVisible(true);
}

public void actionPerformed(ActionEvent e)
{
  if (e.getSource() ==  b)
  {
     String line="";
     try
    {
     /* Serialize input data to config.xml file */
     FileOutputStream fos=new FileOutputStream("config.xml");
     line="<?xml version='1.0'?>"+"\r\n";
     fos.write(line.getBytes());
     line="<config>"+"\r\n";
     fos.write(line.getBytes());
     line="<proxyImage>"+"\r\n";
     fos.write(line.getBytes());
     line="<height>"+tf1.getText()+"</height>"+"\r\n";
     fos.write(line.getBytes());
     line="<width>"+tf2.getText()+"</width>"+"\r\n";
     fos.write(line.getBytes());
     line="<count>"+tf5.getText()+"</count>"+"\r\n";
     fos.write(line.getBytes());
```

```
      line="</proxyImage>"+"\r\n";
      fos.write(line.getBytes());
      line="<realImage>"+"\r\n";
      fos.write(line.getBytes());
      line="<height>"+tf4.getText()+"</height>"+"\r\n";
      fos.write(line.getBytes());
      line="<width>"+tf3.getText()+"</width>"+"\r\n";
      fos.write(line.getBytes());
      line="</realImage>"+"\r\n";
      fos.write(line.getBytes());
      line="</config>"+"\r\n";
      fos.write(line.getBytes());
      fos.close();
      JOptionPane.showMessageDialog(this,"Configuration File Updated Successfully...");
    }
    catch(IOException e1)
    {
       System.out.println("Error in Writing...");
    }
    }
}
}

public class MainFrame extends JFrame implements ActionListener
{
  JMenuBar mb;
  JMenu config;
  JMenu proxy;
  JMenuItem change,view,exit;
  JMenuItem viewproxy;
  JLabel label1;
```

```
JLabel label2;
public MainFrame()
{
 /* Create a Display Menu Bar */
 change=new JMenuItem("Change");
 change.addActionListener(this);

 view=new JMenuItem("View");
 view.addActionListener(this);

 exit=new JMenuItem("Exit");
 exit.addActionListener(this);

 config=new JMenu("Config Info");
 config.setFont(new Font("Arial",Font.BOLD,16));
 config.add(change);
 config.add(view);
 config.addSeparator();
 config.add(exit);

 proxy=new JMenu("Proxy Products");
 proxy.setFont(new Font("Arial",Font.BOLD,16));
 viewproxy=new JMenuItem("View");
 viewproxy.addActionListener(this);

 proxy.add(viewproxy);
 mb=new JMenuBar();
 mb.setBounds(0,0,800,20);
 mb.add(config);
 mb.add(proxy);
```

```
  add(mb);

  label1=new JLabel("      Proxy Design Pattern");
  label1.setFont(new Font("Arial",Font.BOLD,48));
  label1.setForeground(Color.red);
  label1.setBounds(50,50,1000,500);
  add(label1);

  label2=new JLabel("An Industry Perspective");
  label2.setFont(new Font("Arial",Font.BOLD,36));
  label2.setForeground(Color.green);
  label1.setBounds(50,80,1000,500);
  add(label2);
}

public void actionPerformed(ActionEvent e)
{
  if(e.getSource()==viewproxy)
  {
     ProxyProductPattern pattern=new ProxyProductPattern();
  }
  if(e.getSource()==exit)
  {
   System.exit(0);
  }
   if(e.getSource()==change)
  {
    new Configuration();
  }
}
```

```
  public static void main(String[] args)
  {
    MainFrame f=new MainFrame();
    f.setSize(800,800);
    f.setVisible(true);
    f.setDefaultCloseOperation(EXIT_ON_CLOSE);
  }
}
```

Executable Batch File

run.bat

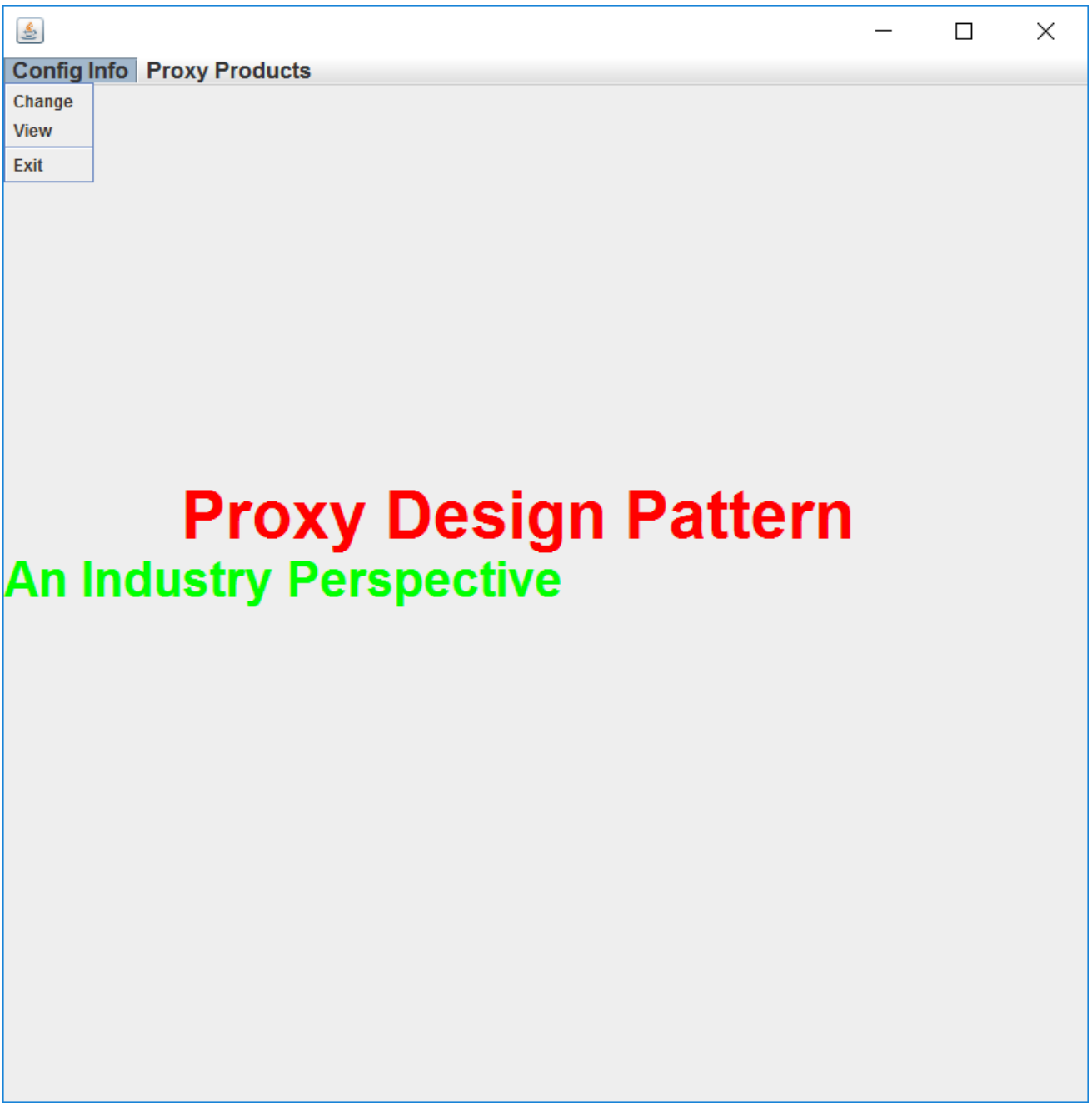
```
set classpath=f:\designpatterns\model5\mysql-connector-java-5.1.15-bin.jar;
f:\designpatterns\model5\jdom-2.0.5.jar;.

javac IProduct.java
javac Product.java
javac ProxyProduct.java
javac RealProduct.java
javac ProductDAOImpl.java
javac MainFrame.java
java MainFrame
```


6.4 Test Cases and Screen Shots

Test Case 1: Execute the application

Desired Output : A frame window with a menu for changing image configuration information and for viewing proxy objects should be displayed.

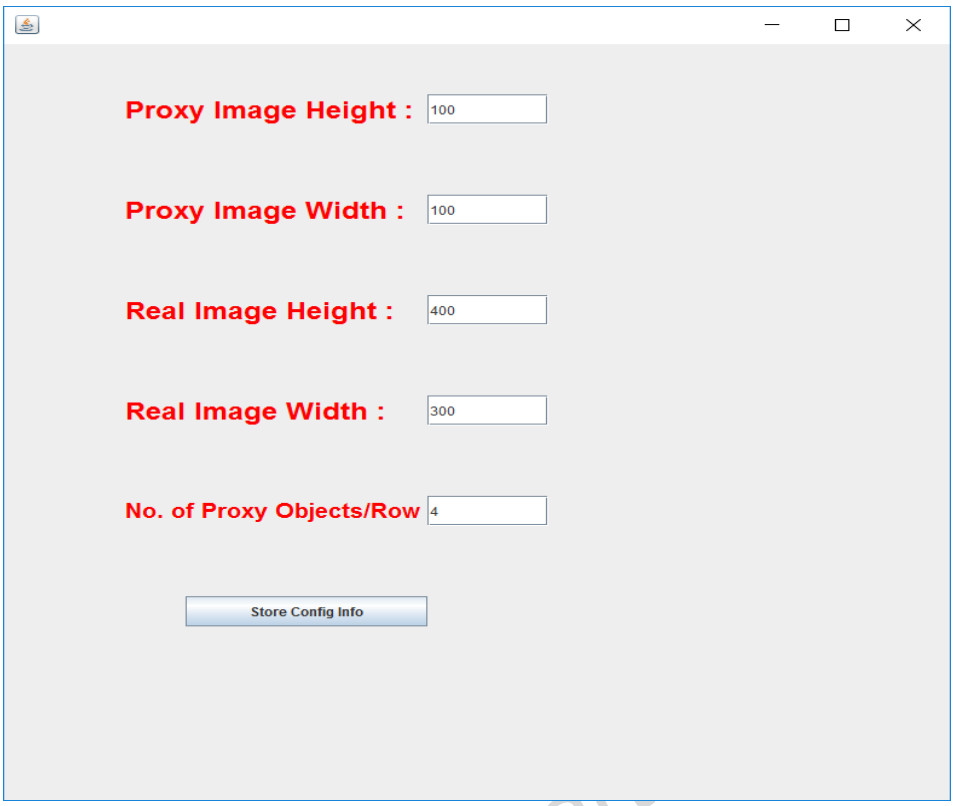
Actual Output :



Test Case 2: Click on “Change” menu option

Desired Output : A frame window for accepting image configuration information and should be displayed.

Actual Output :



Structure of XML File Generated by the Module

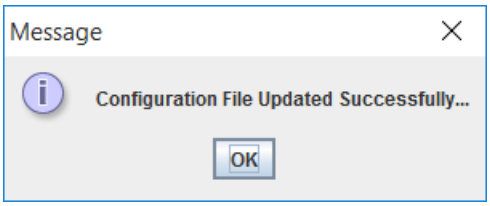
config.xml

```
<?xml version='1.0'?>
<config>
<proxyImage>
<height>200</height>
<width>200</width>
<count>5</count>
</proxyImage>
<realImage>
<height>400</height>
<width>300</width>
</realImage>
</config>
```

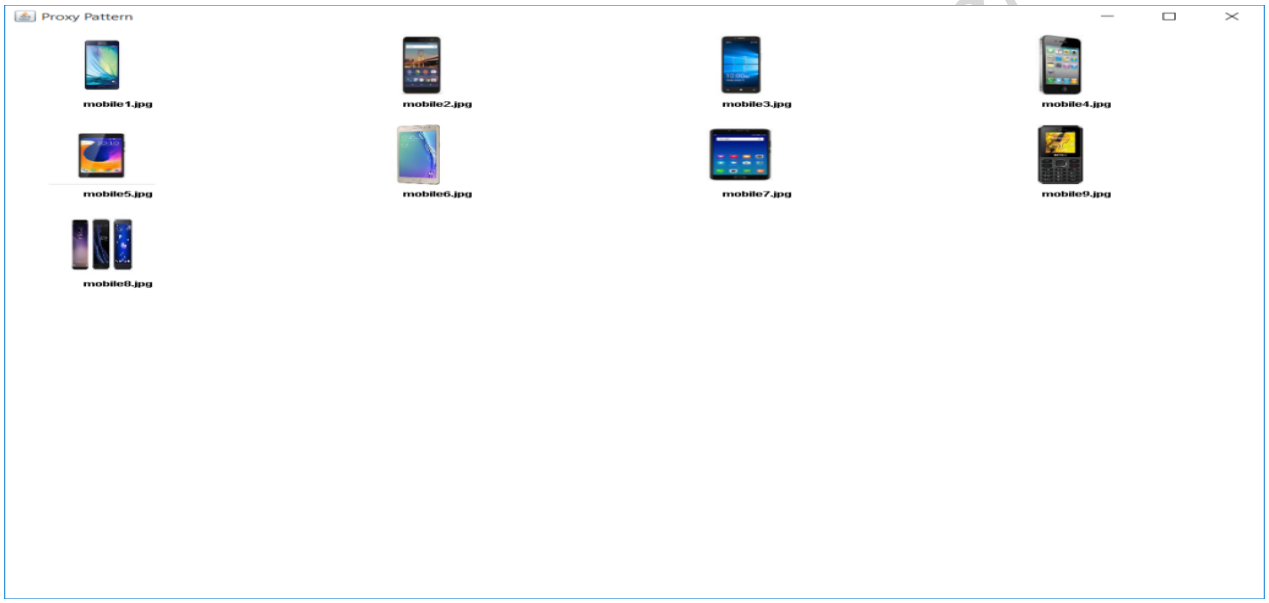
Test Case 3: Enter image configuration information and click on "Store Config Info" button.

Desired Output : The image configuration information should be saved in a config.xml file and a message "**Configuration File Updated Successfully**" should be displayed to an end user.

Actual Output :



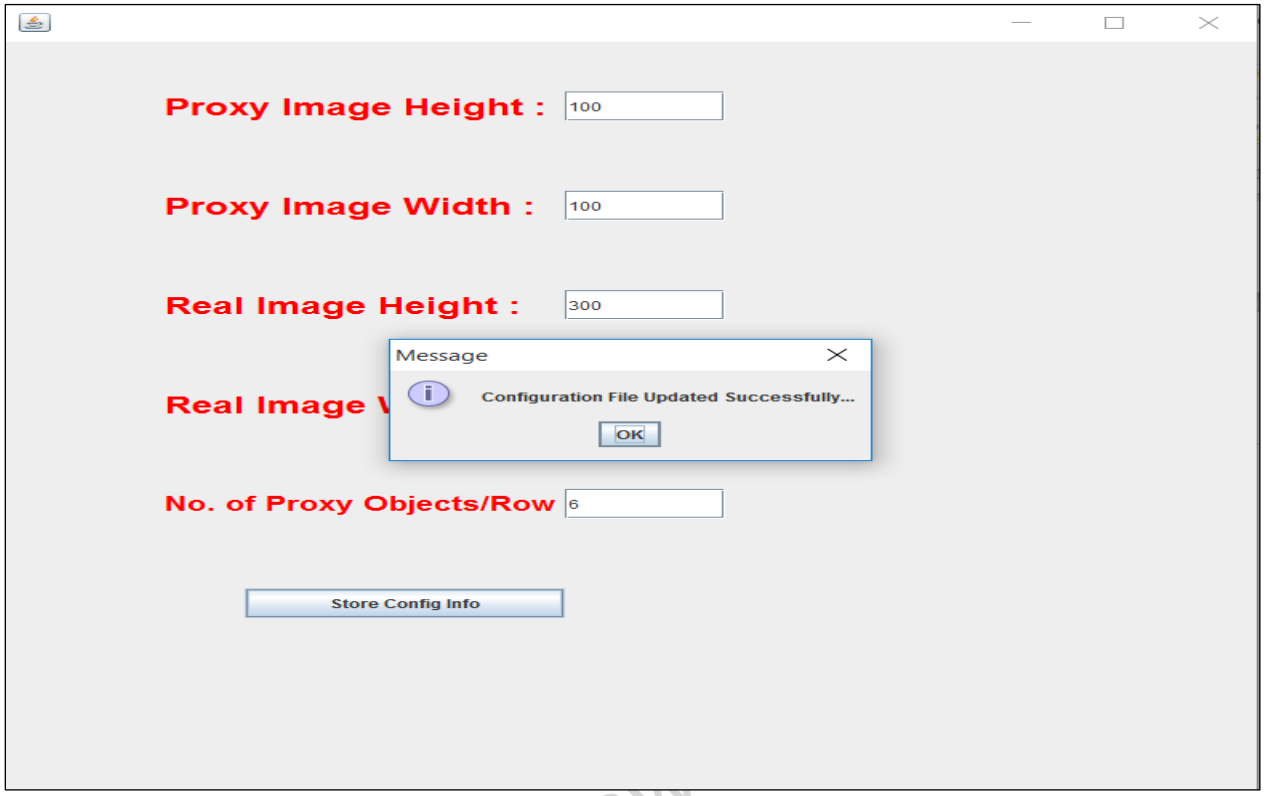
Click on menu option "Proxy Products". The images of proxy products, of dimension 100 x 100 are displayed with four proxy products in a single row as shown below.



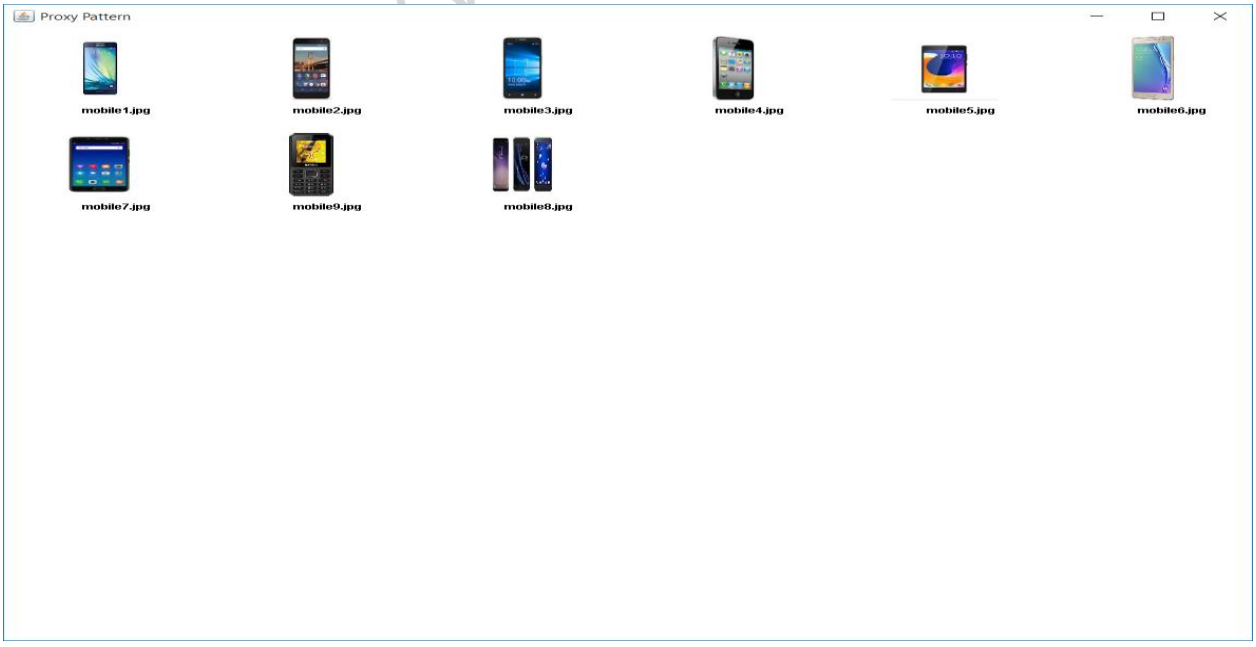
Test Case 4: Click on "Change" menu option and change no. of Proxy Products/row to 6 and 2, respectively and click on "Store Config Info" button.

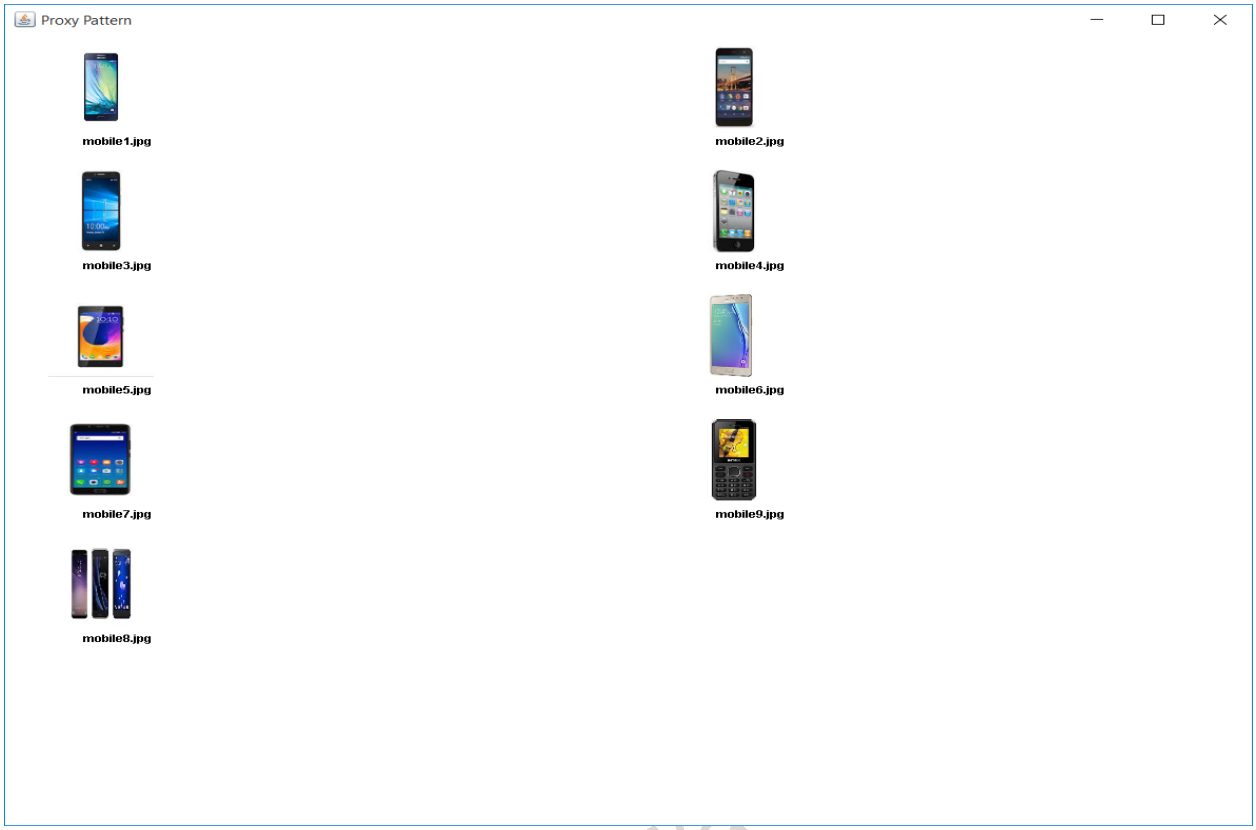
Desired Output : The image configuration information should be saved in a config.xml file and a message "configuration file Updated Successfully" should be displayed to an end user. On clicking the menu option "Proxy Products" 6 proxy products/2 proxy products should be accommodated in a single row as shown below:

Actual Output :



Click on menu option "Proxy Products". The images of proxy products, of dimension 100 x 100 are displayed with four proxy products in a single row as shown below.





6.5 Module Limitations :

1. Module is database dependent. Only MySQL database is supported.

The limitation is overcome in Model 7 discussed next where the generic model is designed and developed for storing configuration information in one of the formats

- XML
- JSON

and user is given the flexibility of selecting the backend from one of the following

- MS-Access
- MySQL
- Oracle

Chapter 7
Proxy Design Pattern Implementation Model 6

Purpose : Model 6 is an enhancement over model 4 and 5 for generating GUI interface to an end user for generating multiple proxy objects of custom dimension displaying short description of each. The product information is persistently stored in a MySQL database management system. On clicking any particular proxy object, the corresponding real object of custom dimension is generated by loading the requisite information from the back end database management system and detailed description of the features specific to it is displayed to an end user. The image configuration information pertaining to proxy image dimension, real image dimension, no. of proxy images to be displayed in a single row, is stored in JSON format which is parsed using JSON Simple Parser available in Java

6.1 Module Requirement Specification:

The proposed module should be capable of performing the following tasks:

i) Module should be capable of connecting the MySQL database employed Type-IV JDBC driver and retrieving the requisite information from a product table.

ii) Module should generate a Graphical User Interface (GUI) to an end user.

iii) Module should instantiate multiple proxy object of custom dimension stored in config.xml file, by retrieving the current information from the Product table.

iv) The module should employ collection classes for storing product information and feature information specific to the selected product.

v) Each row should display no. of proxy objects as configured in config.xml file.

vi) On creating a proxy object, module should display a short description of features specific to that proxy object to an end user

vii) Module should allow the user to click on proxy object to instantiate a corresponding real object of custom dimension stored in config.xml file. For this the module should keep track of the proxy object clicked by the user.

viii) Module should display a detailed description to an end user on creating a real object corresponding to the proxy object clicked by the user by retrieving the feature-related information corresponding to the selected product from the back end.

ix) GUI should be synchronized with the backend Database.

a. On inserting a record in the Product table, the proxy object should dynamically appear on the GUI.
b. On deleting a record in the Product table, the proxy object should dynamically disappear from the GUI.
c. On updating a record in the Product table, the proxy object should dynamically update the information displayed on the GUI.

Implementation Details:

1. The module employs hash table for storing the product features in the form of a name/value pair.
2. The module employs Graphics2D class for scaling the image appropriately for generating proxy and real objects.
3. The module employs java.awt package for creating Frame and employs anonymous inner classes for handling mouse-click event and window event.
4. The module employs the JDOM parser for retrieving the configuration information stored in config.xml file.

7.2 Application Architecture :

The module employs the same application architecture as in Chapter 6.

7.3 Complete Source Code

IProduct.java

```
import java.util.Hashtable;
interface IProduct
{
  public Hashtable getDescription();
}
```

Product.java

```
import java.util.Hashtable;
class Product implements IProduct
{
/* Attributes of a Product */
  int height;
  int width;
  public Hashtable getDescription()
  {
    return null;
  }
}
```

ProxyProduct.java

```
import java.util.Hashtable;
import java.util.Map;
import java.io.File;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.List;
import java.io.FileReader;
import java.util.Iterator;
import java.util.Map;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.*;

class ProxyProduct extends Product
{
/* Attributes of Proxy Product */
  String filename;
```

```
RealProduct realProduct;
Hashtable features;
String value;

public ProxyProduct()
{
}

public ProxyProduct(String filename)
{
  this.filename=filename;
   try
   {
  /* Code  to parse file "config.json" */
    Object obj = new JSONParser().parse(new FileReader("config.json"));

    /* typecasting obj to JSONObject */
    JSONObject jo = (JSONObject) obj;

    /* Getting proxyImage Object */
    Map proxy = ((Map)jo.get("proxyImage"));

    /* Iterating address Map */
    Iterator<Map.Entry> itr = proxy.entrySet().iterator();
    while (itr.hasNext()) {
       Map.Entry pair = itr.next();
       {
        value=(String)pair.getKey();
        if (value.equals("height"))
          height=Integer.parseInt(pair.getValue().toString());
        else if(value.equals("width"))
```

```
            width=Integer.parseInt(pair.getValue().toString());
          }
        }
    }
    catch(ParseException e)
    {
       System.out.println("Error in Parsing...");
    }
    catch(FileNotFoundException e)
    {
       System.out.println("Configuration File Does Not Exist...");
    }
    catch(IOException e)
    {
       System.out.println("Error in Reading File...");
    }
    catch(NumberFormatException e)
    {
       System.out.println("Cannot Convert into Number...");
    }
     if (height>250)
       height=250;
      if (width>250)
       width=250;
 }

public Hashtable getDescription()
 {
    Hashtable features=new Hashtable();
    /* Create and return features if realProduct is null, otherwise invoke
    getDescription() method of real product */
```

```
    if (realProduct==null)
    {
      features.put("filename",filename);
      return features;
    }
    else
      return realProduct.getDescription();
  }

  public void createProduct(Hashtable features)
  {
    realProduct=new RealProduct(features);
  }
}
```

RealProduct.java

```
import java.util.Hashtable;
import java.util.Map;
import java.util.Hashtable;
import java.util.Map;
import java.io.File;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.List;
import java.io.FileReader;
import java.util.Iterator;
import java.util.Map;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.*;
```

```
class RealProduct extends Product
{
/* Attributes of  Real Product */
  Hashtable features;
  String value;

  public RealProduct()
  {
  }

  public RealProduct(Hashtable features)
  {
   this.features=features;
   try
     {
    /* Code to Parsing file "config.json" */
      Object obj = new JSONParser().parse(new FileReader("config.json"));

      /* Typecasting obj to JSONObject */
      JSONObject jo = (JSONObject) obj;

      /* Getting realImage Object */
      Map real = ((Map)jo.get("realImage"));

      /* Iterating address Map */
      Iterator<Map.Entry> itr = real.entrySet().iterator();
      while (itr.hasNext()) {
         Map.Entry pair = itr.next();
        {
          value=(String)pair.getKey();
          if (value.equals("height"))
```

```
            height=Integer.parseInt(pair.getValue().toString());
          else if(value.equals("width"))
            width=Integer.parseInt(pair.getValue().toString());
         }
      }
   }
   catch(ParseException e)
   {
       System.out.println("Error in Parsing...");
   }
   catch(FileNotFoundException e)
   {
      System.out.println("Configuration File Does Not Exist...");
   }
   catch(IOException e)
   {
      System.out.println("Error in Reading File...");
   }
   catch(NumberFormatException e)
   {
     System.out.println("Cannot Convert into Number...");
   }
    if (height>1200)
      height=1200;
    if (width>1000)
      width=1000;
 }

public Hashtable getDescription()
{
   return features;
```

```
  }

  public void setDescription(Hashtable features)
  {
     this.features=features;
  }
}
```

ProductDAOImpl.java

```
import java.util.ArrayList;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Hashtable;

interface ProductDAOIntf
{
  public void connect();
  public ArrayList<ProxyProduct> getProxyProducts();
  public Hashtable getFeatures(int id);
}

public class ProductDAOImpl implements ProductDAOIntf
{
   Connection con;
   public void connect()
   {
     try
```

```
    {
      /* Load MySQL Driver in memory*/
      Class.forName("com.mysql.jdbc.Driver");
      con=DriverManager.getConnection("jdbc:mysql://localhost:3306/designpattern","root","");
    }
    catch(ClassNotFoundException e)
    {
      System.out.println(e);
    }
    catch(SQLException e)
    {
     System.out.println(e);
    }
  }

 public ArrayList<ProxyProduct> getProxyProducts()
 {
  ArrayList<ProxyProduct> products=new ArrayList<ProxyProduct>();
/* Retrieve Products from a table and store in an array list */
  try
  {
   Statement st=con.createStatement();
   ResultSet rs=st.executeQuery("select filename from Product");
   while(rs.next())
   {
    products.add(new ProxyProduct(rs.getString(1)));
   }
  }
  catch(SQLException e)
  {
    System.out.println(e);
```

```
 }
 return products;
}

public Hashtable getFeatures(int id)
{
 /* Populate Hashtable with features of Real Product */
 String cname="";
 String tname="";
 Hashtable features = new Hashtable();
 /* Retrieve features of a selected product and store in a Hashtable */
 try
 {
   Statement st=con.createStatement();
   ResultSet rs=st.executeQuery("select * from Product WHERE id = "+id);
   /* Retrieve ResultSet Metadata */
   ResultSetMetaData rsmd=rs.getMetaData();
   int nCols=rsmd.getColumnCount();
   if (rs.next())
   {
    for(int i=1;i<=nCols;i++)
    {
     cname=rsmd.getColumnName(i);
     tname=rsmd.getColumnTypeName(i);
     if (tname.equals("INT"))
       features.put(cname,rs.getInt(i));
     if (tname.equals("CHAR"))
       features.put(cname,rs.getString(i));
     if (tname.equals("FLOAT"))
       features.put(cname,rs.getFloat(i));
  }
```

```
    return features;
   }
  }
  catch(SQLException e)
  {
    System.out.println(e);
  }
  return null;
 }

  public static void main(String[] args)
  {
    ProductDAOIntf p=new ProductDAOImpl();
    p.connect();
    p.getFeatures(1);
  }
}
```

MainFrame.java

```
import javax.swing.ImageIcon;
import java.awt.image.BufferedImage;
import java.awt.Image;
import java.awt.Color;
import java.awt.Graphics2D;
import java.io.File;
import java.io.FileOutputStream;
import javax.imageio.ImageIO;
import java.awt.RenderingHints;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.WindowAdapter;
```

```
import java.awt.event.WindowEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Font;
import java.awt.TextArea;
import java.awt.FlowLayout;
import java.awt.Color;
import java.util.Hashtable;
import java.util.ArrayList;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JTextField;
import javax.swing.JOptionPane;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.io.FileReader;
import java.util.Iterator;
import java.util.Map;
import java.util.LinkedHashMap;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.util.List;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.*;
/* Class for Scaling Images using 2D Graphics Library */
```

```
class ImgUtils
{
  public BufferedImage scaleImage(int WIDTH, int HEIGHT, String filename)
  {
    BufferedImage bi = null;
    try
    {
/* Scale image with custom width and height attributes */
        ImageIcon ii = new ImageIcon(filename);//path to image
        bi = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = (Graphics2D) bi.createGraphics();
        g2d.addRenderingHints(new RenderingHints(RenderingHints.KEY_RENDERING,
                                                 RenderingHints.VALUE_RENDER_QUALITY));
        g2d.drawImage(ii.getImage(), 0, 0, WIDTH, HEIGHT, null);
    }
    catch (Exception e)
    {
        e.printStackTrace();
        return null;
    }
    return bi;
  }
 }


class ProxyProductPattern extends Frame
{
  ArrayList products=null;
  ProxyProduct proxyProduct;
  Image img;
  int selectedId=0;
  String value;
```

```
int count=0;
public ProxyProductPattern()
{
 setTitle("Proxy Pattern");
 setSize(1200,1000);
 setVisible(true);
 try
 {
    /* parsing file "config.json" */
    Object obj = new JSONParser().parse(new FileReader("config.json"));
    /* Typecasting obj to JSONObject */
    JSONObject jo = (JSONObject) obj;

    /* Getting proxyImage Object */
    Map proxy = ((Map)jo.get("proxyImage"));

    /* Iterating address Map */
    Iterator<Map.Entry> itr = proxy.entrySet().iterator();
    while (itr.hasNext()) {
       Map.Entry pair = itr.next();
       {
        value=(String)pair.getKey();
        if (value.equals("count"))
          count=Integer.parseInt(pair.getValue().toString());
       }
    }
 }
 catch(ParseException e)
 {
    System.out.println("Error in Parsing...");
 }
```

```
 catch(FileNotFoundException e)
 {
     System.out.println("Configuration File Does Not Exist...");
 }
 catch(IOException e)
 {
    System.out.println("Error in Reading File...");
 }
 catch(NumberFormatException e)
 {
     System.out.println("Cannot Convert into Number...");
 }
addMouseListener(new MouseAdapter()
{
  public void mousePressed(MouseEvent e)
  {
    /* Instantiate Real Product on clicking Proxy Product */
    int x=e.getX();
    int y=e.getY();
    x=(x)/(1200/count)+1;
    y=(y)/(proxyProduct.height+50)+1;
   /* Use mapping function to map x and y coordinates to id of the proxy product
       and instantiate appropriate real product  */
    selectedId=count*(y-1)+x;
    new RealProductFrame(proxyProduct,selectedId);
  }
});

addWindowListener(new WindowAdapter()
{
  public void windowClosing(WindowEvent e)
```

```
    {
      dispose();
    }
  });
}

public void paint(Graphics g)
{
  int x=50,y=50;
  int icount=1;
  ProductDAOIntf productDAO=new ProductDAOImpl();
  productDAO.connect();
  products=productDAO.getProxyProducts();

  for (Object obj:products)
  {
      proxyProduct=(ProxyProduct)obj;
      /* Scale image with custom width and height attributes */
      BufferedImage img=new ImgUtils().scaleImage(
      proxyProduct.height,proxyProduct.width,proxyProduct.filename);
      g.drawImage(img, x, y, this);
      g.setFont(new Font("Arial",Font.BOLD,12));
      g.drawString(proxyProduct.filename,x+proxyProduct.width/3,y+20+proxyProduct.height);
      x+=1200/count;
      /* Display count no. of  proxy products in a single row */
      if ((icount % count)==0)
      {
          y+=proxyProduct.height+50;
          x=50;
      }
      icount++;
```

```
    }
  }
}
/* Frame class for diaplying real product */
class RealProductFrame extends Frame
{
   ProxyProduct proxyProduct;
   int selectedId;

   public RealProductFrame(ProxyProduct proxyProduct, int selectedId)
   {
    setTitle("Image Scaling");
    setSize(1000,1000);
    setVisible(true);
    this.proxyProduct=proxyProduct;
    this.selectedId=selectedId;
    /* Get detailed features of proxy object of selected ID */
    ProductDAOIntf productDAO=new ProductDAOImpl();
    productDAO.connect();
    Hashtable features=productDAO.getFeatures(selectedId);
    proxyProduct.createProduct(features);
    addWindowListener(new WindowAdapter()
    {
      public void windowClosing(WindowEvent e)
      {
        dispose();
      }
    });
  }

  public void paint(Graphics g)
```

```
  {
    String filename="Mobile"+selectedId+".jpg";
    /* To draw scaled version of image */
    BufferedImage img=new ImgUtils().scaleImage(proxyProduct.realProduct.width,
                                                proxyProduct.realProduct.height,filename);
    g.drawImage(img, 50, 50, this);
    g.setFont(new Font("Arial",Font.BOLD,24));
    Hashtable features=proxyProduct.getDescription();
    int y=70;
    int x=50;
    /* Display detailed description of real product by reading hashtable, features */
    for(Object key: features.keySet())
    {
          String skey=(String)key;
          g.drawString(key + " : " +features.get(key).toString(),
          proxyProduct.realProduct.width/3+50,y+proxyProduct.realProduct.height);
          y+=30;
    }
  }
}

class Configuration extends JFrame implements ActionListener
{
JLabel lbl1,lbl2,lbl3,lbl4,lbl5;
JTextField tf1,tf2,tf3,tf4,tf5;
JButton b;
String value;
int pheight,pwidth,pcount,rheight,rwidth;
public Configuration()
 {
/* Instantiate and add the controls to layout */
```

```
lbl1=new JLabel("Proxy Image Height : ");
lbl1.setBounds(100,50,400,30);
lbl1.setFont(new Font("Arial",Font.BOLD,24));
lbl1.setForeground(Color.red);

tf1=new JTextField(50);
tf1.setBounds(350,50,100,30);

lbl2=new JLabel("Proxy Image Width :");
lbl2.setBounds(100,150,400,30);
lbl2.setFont(new Font("Arial",Font.BOLD,24));
lbl2.setForeground(Color.red);

tf2=new JTextField(50);
tf2.setBounds(350,150,100,30);

lbl3=new JLabel("Real Image Height :");
lbl3.setBounds(100,250,400,30);
lbl3.setFont(new Font("Arial",Font.BOLD,24));
lbl3.setForeground(Color.red);

tf3=new JTextField(50);
tf3.setBounds(350,250,100,30);

lbl4=new JLabel("Real Image Width :");
lbl4.setBounds(100,350,400,30);
lbl4.setFont(new Font("Arial",Font.BOLD,24));
lbl4.setForeground(Color.red);

tf4=new JTextField(50);
tf4.setBounds(350,350,100,30);
```

```
lbl5=new JLabel("No. of Proxy Objects/Row :");
lbl5.setBounds(100,450,400,30);
lbl5.setFont(new Font("Arial",Font.BOLD,20));
lbl5.setForeground(Color.red);

tf5=new JTextField(50);
tf5.setBounds(350,450,100,30);

b=new JButton("Store Config Info");
b.setBounds(150,550,200,30);
b.addActionListener(this);
try
  {
 /* Retrieve configuration information stored in config.json file and display */
 /*  Parsing file "config.json" */
   Object obj = new JSONParser().parse(new FileReader("config.json"));

   /* Typecasting obj to JSONObject */
   JSONObject jo = (JSONObject) obj;

   /* Getting proxyImage Object */
   Map proxy = ((Map)jo.get("proxyImage"));

   /* Iterating proxy Map */
   Iterator<Map.Entry> itr = proxy.entrySet().iterator();
   while (itr.hasNext()) {
      Map.Entry pair = itr.next();
      {
       value=(String)pair.getKey();
       if (value.equals("height"))
```

```
            pheight=Integer.parseInt(pair.getValue().toString());
          else if(value.equals("width"))
            pwidth=Integer.parseInt(pair.getValue().toString());
          else if(value.equals("count"))
            pcount=Integer.parseInt(pair.getValue().toString());
         }
      }

       /* Getting proxyImage Object */
      Map real = ((Map)jo.get("realImage"));
      /* Iterating proxy Map */
      itr = real.entrySet().iterator();
      while (itr.hasNext()) {
         Map.Entry pair = itr.next();
         {
          value=(String)pair.getKey();
          if (value.equals("height"))
            rheight=Integer.parseInt(pair.getValue().toString());
          else if(value.equals("width"))
            rwidth=Integer.parseInt(pair.getValue().toString());
         }
      }
    }
    catch(ParseException e)
    {
       System.out.println("Error in Parsing...");
    }
    catch(FileNotFoundException e)
    {
       System.out.println("Configuration File Does Not Exist...");
    }
```

```
    catch(IOException e)
    {
       System.out.println("Error in Reading File...");
    }
    catch(NumberFormatException e)
    {
       System.out.println("Cannot Convert into Number...");
    }
   tf1.setText(String.valueOf(pheight));
   tf2.setText(String.valueOf(pwidth));
   tf3.setText(String.valueOf(rheight));
   tf4.setText(String.valueOf(rwidth));
   tf5.setText(String.valueOf(pcount));
   add(lbl1);
   add(tf1);
   add(lbl2);
   add(tf2);
   add(lbl3);
   add(tf3);
   add(lbl4);
   add(tf4);
   add(lbl5);
   add(tf5);
   add(b);
   setSize(800,800);
   setLayout(null);
   setVisible(true);
}

public void actionPerformed(ActionEvent e)
{
```

```
if (e.getSource() ==  b)
{
  /* Accept image configuration  information from end user  and store in config.json file */
  String line="";
  try
  {
    /* Creating JSONObject */
    JSONObject jo = new JSONObject();
    /* For proxy data, first create LinkedHashMap */
    Map m = new LinkedHashMap(3);
    m.put("height", Integer.parseInt(tf1.getText()));
    m.put("width", Integer.parseInt(tf2.getText()));
    m.put("count", Integer.parseInt(tf5.getText()));

    /* Putting address to JSONObject */
    jo.put("proxyImage", m);
    m = new LinkedHashMap(2);
    m.put("height", Integer.parseInt(tf4.getText()));
    m.put("width", Integer.parseInt(tf3.getText()));

    /* Adding map to list */
     jo.put("realImage", m);

    /* Writing JSON to file:"config.json"  */
    /* Serialize input data to config.json file */
    PrintWriter pw = new PrintWriter("config.json");
    pw.write(jo.toJSONString());
    pw.flush();
    pw.close();
    JOptionPane.showMessageDialog(this,"Configuration File Updated Successfully...");
}
```

```
   catch(IOException e1)
   {
       System.out.println("Error in Writing...");
    }
   }
 }
}

public class MainFrame extends JFrame implements ActionListener
{
  JMenuBar mb;
  JMenu config;
  JMenu proxy;
  JMenuItem change,view,exit;
  JMenuItem viewproxy;
  JLabel label1;
  JLabel label2;
  public MainFrame()
  {
/* Create and Display Menu Bar */
    change=new JMenuItem("Change");
    change.addActionListener(this);

    view=new JMenuItem("View");
    view.addActionListener(this);

    exit=new JMenuItem("Exit");
    exit.addActionListener(this);

    config=new JMenu("Config Info");
    config.setFont(new Font("Arial",Font.BOLD,16));
```

```
  config.add(change);
  config.add(view);
  config.addSeparator();
  config.add(exit);

  proxy=new JMenu("Proxy Products");
  proxy.setFont(new Font("Arial",Font.BOLD,16));
  viewproxy=new JMenuItem("View");
  viewproxy.addActionListener(this);

  proxy.add(viewproxy);
  mb=new JMenuBar();
  mb.setBounds(0,0,800,20);
  mb.add(config);
  mb.add(proxy);
  add(mb);
  label1=new JLabel("      Proxy Design Pattern");
  label1.setFont(new Font("Arial",Font.BOLD,48));
  label1.setForeground(Color.red);
  label1.setBounds(50,50,1000,500);
  add(label1);

  label2=new JLabel("An Industry Perspective");
  label2.setFont(new Font("Arial",Font.BOLD,36));
  label2.setForeground(Color.green);
  label1.setBounds(50,80,1000,500);
  add(label2);
}

public void actionPerformed(ActionEvent e)
{
```

```
   if(e.getSource()==viewproxy)
   {
      ProxyProductPattern pattern=new ProxyProductPattern();
   }
   if(e.getSource()==exit)
   {
    System.exit(0);
   }
    if(e.getSource()==change)
   {
     new Configuration();
   }
 }

 public static void main(String[] args)
 {
   MainFrame f=new MainFrame();
   f.setSize(800,800);
   f.setVisible(true);
   f.setDefaultCloseOperation(EXIT_ON_CLOSE);
 }
}
```


7.4 Test Cases and Screen Shots

Same Test Cases and User Interfaces as in Chapter 6 are generated except the image configuration information is stored in JSON file format instead of XML file format. The structure of the JSON file created by the module is shown below:

Structure of JSON File Generated by the Module

config.json

```
{"proxyImage":{"height":100,"width":100,"count":6},"realImage":{"height":400,"width":300}}
```

7.5 Module Limitations :

1. Module is database dependent. Only MySQL database is supported.
2. Image configuration information is stored only in JSON format.
3. The model works for a single type of product.

The limitations are overcome in Model 7 discussed next where the generic model is designed and developed for storing configuration information in one of the formats

- XML
- JSON

and user is given the flexibility of selecting the backend from one of the following

- MS-Access
- MySQL
- Oracle

The model works for any product type whose information is stored in Product table.

Chapter 8
Proxy Design Pattern Implementation Model 7 (Generic Model)

Purpose : Module 7 is an enhancement over all previous modules for generating GUI interface to an end user for generating multiple proxy objects of custom dimension displaying short description of each. The module retrieves the product information stored in desperate database management systems corresponding to MS-Access, MySQL or Oracle. On clicking any particular proxy object, the corresponding real object of custom dimension is generated by loading the requisite information from the back end database management system and detailed description of the features specific to it is displayed to an end user. The generic model is designed and developed for storing configuration information in one of the formats

- XML
- JSON

and user is given the flexibility of selecting the backend from one of the following

- MS-Access
- MySQL
- Oracle

8.1 Module Requirement Specification:

The proposed module should be capable of performing the following tasks:

i) Module should be capable of connecting the desperate database management systems employing Type-I and Type-IV JDBC driver for retrieving the requisite information from a product table.

ii) Module should generate a Graphical User Interface (GUI) to an end user.

iii) Module should instantiate multiple proxy object of custom size by retrieving the current information from the Product table and configuration information from either XML or JSON file.

iv) The module should employ collection classes for storing product information and feature information specific to the selected product.

v) Each row should display no of proxy objects as configured in either XML or JSON file.

vi) On creating a proxy object, module should display a short description of features specific to that proxy object to an end user

vii) Module should allow the user to click on proxy object to instantiate a corresponding real object of custom size by retrieving the configuration information from either XML or JSON file. For this the module should keep track of the proxy object clicked by the user.

viii) Module should display a detailed description to an end user on creating a real object corresponding to the proxy object clicked by the user by retrieving the feature-related information corresponding to the selected product from the back end.

ix) GUI should be synchronized with the backend Database.

a. On inserting a record in the Product table, the proxy object should dynamically appear on the GUI.
b. On deleting a record in the Product table, the proxy object should dynamically disappear from the GUI.
c. On updating a record in the Product table, the proxy object should dynamically update the information displayed on the GUI.

Implementation Details:

1. The module employs JTabbedDialog class for displaying different tabs for customizing and displaying proxy products.
2. The module employs hash table for storing the product features in the form of a name/value pair.
3. The module employs Graphics2D class for scaling the image appropriately for generating proxy and real objects.
4. The module employs java.awt package for creating Frame and employs anonymous inner classes for handling mouse-click event and window event.
5. The module dynamically selects either config.xml file or config.json file for retrieving configuration information depending on the option selected by the user.

6. The module dynamically selects either dbconfig.xml file or dbconfig.json file for retrieving database configuration information depending on the option selected by the user and loading appropriate JDBC driver into the memory for communication with the backend.
7. The connectivity information specific to each back end is depicted in the following table:

Back End	Database Name	Datasource Name	User Name	Password	Host String
MS-Access	--	designpattern	--	--	--
MySQL	designpattern	--	root	--	--
Oracle	--	--	system	siber	orcl

Back End	Driver Class Name	JDBC Driver Type	JAR File
MS-Access	sun.jdbc.odbc.JdbcOdbcDriver	Type-I	--
MySQL	com.mysql.jdbc.Driver	Type-IV	mysql-connector-java-5.1.15-bin.jar
Oracle	oracle.jdbc.driver.OracleDriver	Type-IV	ojdbc14.jar

Back End	JDBC-Url
MS-Access	jdbc:odbc:designpattern
MySQL	jdbc:mysql://localhost:3306/designpattern
Oracle	jdbc:oracle:thin:@192.168.30.94:1521:orcl

8.2 Application Architecture :

The module employs the same application architecture as in Chapter 5.

System Requirements

JAR Files

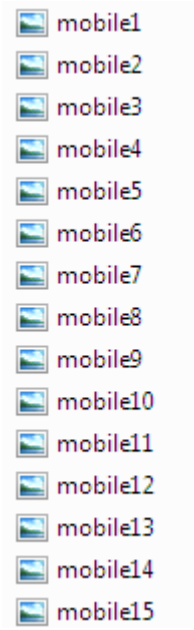

ojdbc14.jar


jdom-2.0.5.jar


json-simple-1.1.1.jar


mysql-connector-java-5.1.15-bin.jar

Resources – Images



MS-Access Database

Database Name : DesignPattern

Table Name : Product

product

id	filename	model	price	discount
1	mobile1.jpg	Nokia1	1000	10
2	mobile2.jpg	Nokia2	2000	0
3	mobile3.jpg	Nokia3	3000	0
4	mobile4.jpg	Nokia4	4000	0

SQL Script

Script.txt

```
create database designpattern;
use designpattern;
create table Product(id int, filename char(50),model char(50), price float);

insert into product values(1,'mobile1.jpg','Nokia1',1000.0);
insert into product values(2,'mobile2.jpg','Nokia2',2000.0);
```

```
insert into product values(3,'mobile3.jpg','Nokia3',3000.0);
insert into product values(4,'mobile4.jpg','Nokia4',4000.0);
insert into product values(5,'mobile5.jpg','Nokia5',5000.0);
insert into product values(6,'mobile6.jpg','Nokia6',6000.0);
insert into product values(7,'mobile7.jpg','Nokia7',7000.0);
insert into product values(8,'mobile8.jpg','Nokia6',8000.0);
insert into product values(9,'mobile9.jpg','Nokia7',9000.0);

Update product set model='Nokia6',price=6500 where id=6;
```

8.3 Complete Source Code

IProduct.java

```
import java.util.Hashtable;
interface IProduct
{
  public Hashtable getDescription();
}
```


Product.java

```
import java.util.Hashtable;
class Product implements IProduct
{
/* Attributes of a Product */
  int height;
  int width;
  public Hashtable getDescription()
  {
    return null;
  }
}
```

RealProduct.java

```
import java.util.Hashtable;
import java.util.Map;
import java.util.Hashtable;
import java.util.Map;
import java.io.File;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.List;
import java.io.FileReader;
import java.util.Iterator;
import java.util.Map;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.*;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
import java.util.List;

class RealProduct extends Product
{
/* Attributes of  Real Product */
  Hashtable features;
  String value;

  /* Default Constructor */
  public RealProduct()
  {
  }
```

```
public RealProduct(Hashtable features, String format)
{
 this.features=features;
 if (format.equals("JSON"))
 {
  /* Parse JSON File */
  try
  {
   /* Parsing file "config.json" */
   Object obj = new JSONParser().parse(new FileReader("config.json"));

   /* Typecasting obj to JSONObject */
   JSONObject jo = (JSONObject) obj;

   /* Getting realImage Object */
   Map real = ((Map)jo.get("realImage"));

   /* Iterating address Map */
   Iterator<Map.Entry> itr = real.entrySet().iterator();
   while (itr.hasNext()) {
      Map.Entry pair = itr.next();
      {
       value=(String)pair.getKey();
       if (value.equals("height"))
         height=Integer.parseInt(pair.getValue().toString());
       else if(value.equals("width"))
         width=Integer.parseInt(pair.getValue().toString());
      }
   }
 }
```

```
  catch(ParseException e)
  {
    System.out.println("Error in Parsing...");
  }
  catch(FileNotFoundException e)
  {
    System.out.println("Configuration File Does Not Exist(6)...");
  }
  catch(IOException e)
  {
    System.out.println("Error in Reading File...");
  }
  catch(NumberFormatException e)
  {
    System.out.println("Cannot Convert into Number...");
  }
}
else
{
  /* Parse XML File */
  SAXBuilder builder = new SAXBuilder();
  File xmlFile = new File("config.xml");
  try
  {
 /* Obtain reference to the document root and traverse through the child nodes
     of realImage node*/
        Document document = (Document) builder.build(xmlFile);
        Element rootNode = document.getRootElement();
        List list = rootNode.getChildren("realImage");

        for (int i = 0; i < list.size(); i++) {
```

```
            Element node = (Element) list.get(i);
            height=Integer.parseInt(node.getChildText("height"));
            width=Integer.parseInt(node.getChildText("width"));
          }
      }
      catch (IOException io)
      {
            System.out.println(io.getMessage());
      }
      catch (JDOMException jdomex)
      {
            System.out.println(jdomex.getMessage());
      }
    }
    /* Scale Down the Image Sizes Properly */
       if (height>1200)
          height=1200;
       if (width>1000)
          width=1000;
    }

    public Hashtable getDescription()
    {
      return features;
    }

    public void setDescription(Hashtable features)
    {
      this.features=features;
    }
}
```

ProxyProduct.java

```
import java.util.Hashtable;
import java.util.Map;

import java.io.File;
import java.io.IOException;

import java.io.FileNotFoundException;
import java.util.List;

import java.io.FileReader;
import java.util.Iterator;
import java.util.Map;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.*;

import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
import java.util.List;

class ProxyProduct extends Product
{
 /* Attributes of Proxy Product */
  String filename;
  RealProduct realProduct;
  Hashtable features;
```

```
String value;
String format;

/* Default Constructor */
public ProxyProduct()
{
}

public ProxyProduct(String filename,String format)
{
  this.filename=filename;
  this.format=format;
  if (format.equals("JSON"))
  {
   /* Parsing JSON File */
   try
   {
    /* Parsing file "config.json" */
    Object obj = new JSONParser().parse(new FileReader("config.json"));

    /* Typecasting obj to JSONObject */
    JSONObject jo = (JSONObject) obj;

    /* Getting proxyImage Object */
    Map proxy = ((Map)jo.get("proxyImage"));

    /* Iterating address Map */
    Iterator<Map.Entry> itr = proxy.entrySet().iterator();
    while (itr.hasNext())
    {
       Map.Entry pair = itr.next();
```

```
        {
         value=(String)pair.getKey();
         if (value.equals("height"))
           height=Integer.parseInt(pair.getValue().toString());
         else if(value.equals("width"))
           width=Integer.parseInt(pair.getValue().toString());
        }
      }
  }
  catch(ParseException e)
  {
    System.out.println("Error in Parsing...");
  }
  catch(FileNotFoundException e)
  {
    System.out.println("Configuration File Does Not Exist...");
  }
  catch(IOException e)
  {
    System.out.println("Error in Reading File...");
  }
  catch(NumberFormatException e)
  {
    System.out.println("Cannot Convert into Number...");
  }
}
else
{
 /* Parsing XML File */
 SAXBuilder builder = new SAXBuilder();
 File xmlFile = new File("config.xml");
```

```
try
{
   /* Obtain reference to the document root and traverse through the child nodes*/
      Document document = (Document) builder.build(xmlFile);
      Element rootNode = document.getRootElement();
      List list = rootNode.getChildren("proxyImage");

      for (int i = 0; i < list.size(); i++)
   {
         Element node = (Element) list.get(i);
         height=Integer.parseInt(node.getChildText("height"));
         width=Integer.parseInt(node.getChildText("width"));
      }

}
catch (IOException io)
{
      System.out.println(io.getMessage());
}
catch (JDOMException jdomex)
{
      System.out.println(jdomex.getMessage());
}

}

/* Scale down height and width Properly */
if (height>250)
    height=250;
if (width>250)
    width=250;
```

```
 }

 public Hashtable getDescription()
 {
   Hashtable features=new Hashtable();
   if (realProduct==null)
   {
     features.put("filename",filename);
     return features;
   }
   else
     return realProduct.getDescription();
 }

 public void createProduct(Hashtable features)
 {
  realProduct=new RealProduct(features,format);
 }
}
```


ProductDAOImpl.java

```
import java.util.ArrayList;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Hashtable;
import java.util.Map;
```

```
import java.util.Iterator;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
import java.util.List;
import java.io.*;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.*;
import javax.swing.*;

interface ProductDAOIntf
{
  public void connect();
  public ArrayList<ProxyProduct> getProxyProducts();
  public Hashtable getFeatures(int id);
}

public class ProductDAOImpl implements ProductDAOIntf
{
   Connection con;
   String format;
   String backEnd="",dsn="",dbName="",userName="",password="",hostString="";
   String value="";

   /* Default Constructor */
   public ProductDAOImpl()
   {
   }
```

```
public ProductDAOImpl(String format)
{
  this.format=format;
}

public void connect()
{
 if (format.equals("XML"))
 {
      /* Parse XML File */
      SAXBuilder builder = new SAXBuilder();
         File xmlFile = new File("dbconfig.xml");
         try {
                  Document document = (Document) builder.build(xmlFile);
                  Element rootNode = document.getRootElement();
                  List list = rootNode.getChildren("database");
                  for (int i = 0; i < list.size(); i++) {
                    Element node = (Element) list.get(i);
                    backEnd=node.getChildText("backEnd");
                    dsn=node.getChildText("dsn");
                    dbName=node.getChildText("dbName");
                    userName=node.getChildText("userName");
                    password=node.getChildText("password");

                  }
         }
      catch(FileNotFoundException e)
      {
        System.out.println("Database Configuration File Does Not Exist...");
      }
      catch (IOException io)
```

```
        {
            System.out.println(io.getMessage());
          }
        catch (JDOMException jdomex)
        {
            System.out.println(jdomex.getMessage());
          }
      }
else
{
  try
  {
    /* Parsing file "config.json" */
    Object obj = new JSONParser().parse(new FileReader("dbconfig.json"));

    /* Typecasting obj to JSONObject */
    JSONObject jo = (JSONObject) obj;

    /* Getting proxyImage Object  */
    Map proxy = ((Map)jo.get("database"));

    /* Iterating proxy Map  */
    Iterator<Map.Entry> itr = proxy.entrySet().iterator();
    while (itr.hasNext()) {
       Map.Entry pair = itr.next();
       {
        value=(String)pair.getKey();
        if (value.equals("backEnd"))
          backEnd=pair.getValue().toString();
        else if(value.equals("dsn"))
          dsn=pair.getValue().toString();
```

```
            else if(value.equals("dbName"))
              dbName=pair.getValue().toString();
            else if(value.equals("userName"))
              userName=pair.getValue().toString();
            else if(value.equals("password"))
              password=pair.getValue().toString();
            else if(value.equals("hostString"))
              hostString=pair.getValue().toString();
           }
        }
    }
    catch(ParseException e1)
    {
      System.out.println("Error in Parsing...");
    }
    catch(FileNotFoundException e1)
    {
      System.out.println("Configuration File Does Not Exist...");
    }
    catch(IOException e1)
    {
      System.out.println("Error in Reading File...");
    }
    catch(NumberFormatException e1)
    {
      System.out.println("Cannot Convert into Number...");
    }
  }
  try
  {
       if (backEnd.equals("MySQL"))
```

```
        {
          /* Load MySQL Type-IV JDBC Driver in memory*/
          Class.forName("com.mysql.jdbc.Driver");
         con=DriverManager.getConnection("jdbc:mysql://localhost:3306/"+dbName,"root","");
        }
        else if(backEnd.equals("MS-Access"))
        {
          /* Load JDBC-ODBC Bridge Type-IV JDBC Driver in memory*/
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          con=DriverManager.getConnection("jdbc:odbc:"+dsn,"","");
        }
        else if(backEnd.equals("Oracle"))
        {
          /* Load Oracle Thin Type-IV JDBC Driver in memory*/
          Class.forName("oracle.jdbc.driver.OracleDriver");
          con=DriverManager.getConnection("jdbc:oracle:thin:@192.168.30.94:1521:"+
                                                    hostString,userName,password);
        }
    }
    catch(ClassNotFoundException e)
    {
       System.out.println(e);
    }
    catch(SQLException e)
    {
      System.out.println(e);
    }
}

public ArrayList<ProxyProduct> getProxyProducts()
{
```

```
 /* Populate ArrayList with objects of ProxyProduct */
 ArrayList<ProxyProduct> products=new ArrayList<ProxyProduct>();
 try
 {
   Statement st=con.createStatement();
   ResultSet rs=st.executeQuery("select filename from Product");
   while(rs.next())
   {
    products.add(new ProxyProduct(rs.getString(1),format));
   }
 }
 catch(SQLException e)
 {
    System.out.println(e);
 }
 return products;
}

public Hashtable getFeatures(int id)
{
  String cname="";
  String tname="";
  Hashtable features = new Hashtable();
  /* Populate Hashtable with features of Real Product */
  try
  {
    Statement st=con.createStatement();
    ResultSet rs=st.executeQuery("select * from Product WHERE id = "+id);
    ResultSetMetaData rsmd=rs.getMetaData();
    int nCols=rsmd.getColumnCount();
    if (rs.next())
```

```
    {
      for(int i=1;i<=nCols;i++)
      {
       cname=rsmd.getColumnName(i);
       tname=rsmd.getColumnTypeName(i);
       System.out.println("tname="+tname);
       if (tname.equals("INT"))
         features.put(cname,rs.getInt(i));
       if (tname.equals("CHAR"))
         features.put(cname,rs.getString(i));
       if (tname.equals("FLOAT"))
         features.put(cname,rs.getFloat(i));
    }
     return features;
    }
  }
  catch(SQLException e)
  {
     System.out.println(e);
  }
  return null;
 }
}
```

MainFrame.java

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.image.BufferedImage;
import java.awt.*;
import java.awt.Color;
import java.awt.Graphics2D;
```

```
import java.io.File;
import java.io.FileOutputStream;
import javax.imageio.ImageIO;
import java.awt.RenderingHints;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Font;
import java.awt.TextArea;
import java.awt.FlowLayout;
import java.awt.Color;
import java.util.Hashtable;
import java.util.ArrayList;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JTextField;
import javax.swing.JOptionPane;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

import java.io.FileReader;
import java.util.Iterator;
import java.util.Map;
import java.util.LinkedHashMap;
```

```
import java.io.IOException;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.util.List;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.*;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
import java.util.List;

/* Class for Scaling Images using 2D Graphics Library */
class ImgUtils
{
  public BufferedImage scaleImage(int WIDTH, int HEIGHT, String filename)
  {
    BufferedImage bi = null;
    try
    {
       ImageIcon ii = new ImageIcon(filename);//path to image
       bi = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
       Graphics2D g2d = (Graphics2D) bi.createGraphics();
       g2d.addRenderingHints(new RenderingHints(RenderingHints.KEY_RENDERING,
                                          RenderingHints.VALUE_RENDER_QUALITY));
       g2d.drawImage(ii.getImage(), 0, 0, WIDTH, HEIGHT, null);
    }
    catch (Exception e)
    {
       e.printStackTrace();
```

```
        return null;
    }
    return bi;
  }
 }

class ProxyProductPattern extends Frame
{
  ArrayList products=null;
  ProxyProduct proxyProduct;
  Image img;
  int selectedId=0;
  String value;
  int count=0;
  static String format;

  public ProxyProductPattern(String format)
  {
    setTitle("Proxy Pattern");
    setSize(1200,1000);
    setVisible(true);
    this.format=format;
    if (format.equals("JSON"))
   {
     try
     {
      /* Parsing file "config.json" */
      Object obj = new JSONParser().parse(new FileReader("config.json"));

      /* Typecasting obj to JSONObject */
        JSONObject jo = (JSONObject) obj;
```

```
 /*  Getting proxyImage Object */
   Map proxy = ((Map)jo.get("proxyImage"));

 /* Iterating address Map */
 Iterator<Map.Entry> itr = proxy.entrySet().iterator();
 while (itr.hasNext()) {
   Map.Entry pair = itr.next();
   {
     value=(String)pair.getKey();
     if (value.equals("count"))
       count=Integer.parseInt(pair.getValue().toString());
   }
 }
}
catch(ParseException e)
{
  System.out.println("Error in Parsing...");
}
catch(FileNotFoundException e)
{
  System.out.println("Configuration File Does Not Exist(1)...");
}
catch(IOException e)
{
  System.out.println("Error in Reading File...");
}
catch(NumberFormatException e)
{
  System.out.println("Cannot Convert into Number...");
}
```

```
}
else
{
   SAXBuilder builder = new SAXBuilder();
   File xmlFile = new File("config.xml");
    try
   {
        /* Obtain reference to the document root and traverse through the child nodes*/
        Document document = (Document) builder.build(xmlFile);
        Element rootNode = document.getRootElement();
        List list = rootNode.getChildren("proxyImage");
        for (int i = 0; i < list.size(); i++) {
          Element node = (Element) list.get(i);
          count=Integer.parseInt(node.getChildText("count"));
        }
   }
   catch(FileNotFoundException e)
   {
     JOptionPane.showMessageDialog(this,"XML Image Configuration File Does Not
              Exist...");
   }
   catch (IOException io)
   {
        System.out.println(io.getMessage());
   }
   catch (JDOMException jdomex)
   {
        System.out.println(jdomex.getMessage());
   }
 }
```

```
  addMouseListener(new MouseAdapter()
  {
   public void mousePressed(MouseEvent e)
   {
     int x=e.getX();
     int y=e.getY();
     x=(x)/(1200/count)+1;
     y=(y)/(proxyProduct.height+50)+1;
     selectedId=count*(y-1)+x;
     new RealProductFrame(proxyProduct,selectedId,ProxyProductPattern.format);
   }
  });

  addWindowListener(new WindowAdapter()
  {
   public void windowClosing(WindowEvent e)
   {
    dispose();
   }
  });
}

public void paint(Graphics g)
{
  int x=50,y=50;
  int icount=1;
  ProductDAOIntf productDAO=new ProductDAOImpl(format);
  productDAO.connect();
  products=productDAO.getProxyProducts();
  for (Object obj:products)
  {
```

```
    proxyProduct=(ProxyProduct)obj;
    BufferedImage img=new ImgUtils().scaleImage(proxyProduct.height,
                                                proxyProduct.width,proxyProduct.filename);
    g.drawImage(img, x, y, this);
    g.setFont(new Font("Arial",Font.BOLD,12));
    g.drawString(proxyProduct.filename,x+proxyProduct.width/3,y+20+proxyProduct.height);
    x+=1200/count;
    if ((icount % count)==0)
    {
      y+=proxyProduct.height+50;
      x=50;
    }
    icount++;
   }
  }
}
/* Frame class for diaplying real product */
class RealProductFrame extends Frame
{
  ProxyProduct proxyProduct;
  int selectedId;

   public RealProductFrame(ProxyProduct proxyProduct, int selectedId,String format)
  {
   setTitle("Image Scaling");
   setSize(1000,1000);
   setVisible(true);
   this.proxyProduct=proxyProduct;
   this.selectedId=selectedId;
   ProductDAOIntf productDAO=new ProductDAOImpl(format);
   productDAO.connect();
```

```
  Hashtable features=productDAO.getFeatures(selectedId);
  proxyProduct.createProduct(features);

  addWindowListener(new WindowAdapter()
  {
    public void windowClosing(WindowEvent e)
    {
      dispose();
    }
  });
}

public void paint(Graphics g)
{
  String filename=proxyProduct.filename;
  String prod=filename.split("[0-9]")[0];
  String filename1=prod+selectedId+".jpg";
  BufferedImage img=new ImgUtils().scaleImage(proxyProduct.realProduct.width,
                                    proxyProduct.realProduct.height,filename);
  g.drawImage(img, 50, 50, this);
  g.setFont(new Font("Arial",Font.BOLD,24));
  Hashtable features=proxyProduct.getDescription();
  int y=70;
  int x=50;
  for(Object key: features.keySet())
  {
    String skey=(String)key;
    g.drawString(key + " : " +features.get(key).toString(),
              proxyProduct.realProduct.width/3+50,y+proxyProduct.realProduct.height);
    y+=30;
  }
```

```
  }
}

public class MainFrame extends       JFrame implements ActionListener
{
      private          JTabbedPane tabbedPane;
      private          JPanel      panel1;
      private          JPanel      panel2;
      private          JPanel      panel3;
     /* Option Button Controls for Page 1 */
      JRadioButton rb1,rb2;
      ButtonGroup bg;
      /* Option Button Controls for Page 2 */
      JRadioButton rb3,rb4;
      ButtonGroup bg1;
      /* Option Button Controls for Page 3 */
      JRadioButton rb5,rb6;
      ButtonGroup bg2;
      /*Controls for Page 1 */
      JLabel lbl1,lbl2,lbl3,lbl4,lbl5;
      JTextField tf1,tf2,tf3,tf4,tf5;
      JButton b;
      String value;
      /*Controls for Page 2 */
      JLabel lbl6,lbl7,lbl8,lbl9,lbl10,lbl11;
      JTextField tf6,tf7,tf8,tf9,tf10,tf11,tf12;

      JComboBox jc;
      JButton b1;
      String[] db={"Select BackEnd","MS-Access","MySQL","Oracle"};
      String backEnd="",dsn="",dbName="",userName="",password="",hostString="";
```

```
    int pheight,pwidth,pcount,rheight,rwidth;

        public MainFrame()
        {
                setTitle( "Tabbed Pane Application" );
                setSize( 800, 800 );
                setBackground( Color.white );
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                JPanel topPanel = new JPanel();
                topPanel.setLayout( new BorderLayout() );
                getContentPane().add( topPanel );

                /* Create the tab pages */
                createPage1();
                createPage2();
                createPage3();

                /* Create a tabbed pane */
                tabbedPane = new JTabbedPane();
                tabbedPane.addTab( "Configure Images", panel1 );
                tabbedPane.addTab( "Configure Back End", panel2 );
                tabbedPane.addTab( "View Proxy Products", panel3 );
                topPanel.add( tabbedPane, BorderLayout.CENTER );
        }

public void actionPerformed(ActionEvent e)
{
 if (e.getSource() == rb1)
 {
   /* Executed  when user clicks on 'XML Format' button on 'Configure Images' page */
```

```
SAXBuilder builder = new SAXBuilder();
File xmlFile = new File("config.xml");
try
{
      Document document = (Document) builder.build(xmlFile);
      Element rootNode = document.getRootElement();
      List list = rootNode.getChildren("proxyImage");

      for (int i = 0; i < list.size(); i++)
   {
        Element node = (Element) list.get(i);
        pheight=Integer.parseInt(node.getChildText("height"));
        pcount=Integer.parseInt(node.getChildText("count"));
        if (pheight>250)
            pheight=250;
        pwidth=Integer.parseInt(node.getChildText("width"));
        if (pwidth>250)
            pwidth=250;
      }

       list = rootNode.getChildren("realImage");
      for (int i = 0; i < list.size(); i++) {
      Element node = (Element) list.get(i);
         rheight=Integer.parseInt(node.getChildText("height"));
      if (rheight>1200)
        rheight=1200;
         rwidth=Integer.parseInt(node.getChildText("width"));
      if (rwidth>1000)
        rwidth=1000;
              }
        }
```

```
        catch (IOException io)
        {
           System.out.println(io.getMessage());
        }
        catch (JDOMException jdomex)
        {
           System.out.println(jdomex.getMessage());
        }
    tf1.setText(String.valueOf(pheight));
    tf2.setText(String.valueOf(pwidth));
    tf3.setText(String.valueOf(rheight));
    tf4.setText(String.valueOf(rwidth));
    tf5.setText(String.valueOf(pcount));
 }

if (e.getSource() == rb2)
 {
   /* Executed  when user clicks on 'JSON Format' button on 'Configure Images' page */

   try
    {
   /* Parsing file "config.json" */
     Object obj = new JSONParser().parse(new FileReader("config.json"));

     /* Typecasting obj to JSONObject */
     JSONObject jo = (JSONObject) obj;

     /* Getting proxyImage Object */
     Map proxy = ((Map)jo.get("proxyImage"));

     /* Iterating proxy Map */
```

```
    Iterator<Map.Entry> itr = proxy.entrySet().iterator();
    while (itr.hasNext()) {
       Map.Entry pair = itr.next();
       {
        value=(String)pair.getKey();
        if (value.equals("height"))
          pheight=Integer.parseInt(pair.getValue().toString());
        else if(value.equals("width"))
          pwidth=Integer.parseInt(pair.getValue().toString());
        else if(value.equals("count"))
          pcount=Integer.parseInt(pair.getValue().toString());
       }
    }

    /* Getting proxyImage Object */
    Map real = ((Map)jo.get("realImage"));

    /* Iterating proxy Map */
    itr = real.entrySet().iterator();
    while (itr.hasNext()) {
       Map.Entry pair = itr.next();
       {
        value=(String)pair.getKey();
        if (value.equals("height"))
          rheight=Integer.parseInt(pair.getValue().toString());
        else if(value.equals("width"))
          rwidth=Integer.parseInt(pair.getValue().toString());
       }
    }
  }
  catch(ParseException e1)
```

```
    {
      System.out.println("Error in Parsing...");
    }
    catch(FileNotFoundException e1)
    {
      System.out.println("Configuration File Does Not Exist(2)...");
    }
    catch(IOException e1)
    {
      System.out.println("Error in Reading File...");
    }
    catch(NumberFormatException e1)
    {
      System.out.println("Cannot Convert into Number...");
    }

    tf1.setText(String.valueOf(pheight));
    tf2.setText(String.valueOf(pwidth));
    tf3.setText(String.valueOf(rheight));
    tf4.setText(String.valueOf(rwidth));
    tf5.setText(String.valueOf(pcount));
  }

  if (e.getSource() == rb3)
  {
    /* Executed  when user clicks on 'XML Format' button on 'Configure Back End' page */
            SAXBuilder builder = new SAXBuilder();
            File xmlFile = new File("dbconfig.xml");
            try
            {
                    Document document = (Document) builder.build(xmlFile);
```

```
                Element rootNode = document.getRootElement();
                List list = rootNode.getChildren("database");
                for (int i = 0; i < list.size(); i++) {
                Element node = (Element) list.get(i);
                 backEnd=node.getChildText("backEnd");
                 dsn=node.getChildText("dsn");
                 dbName=node.getChildText("dbName");
                 userName=node.getChildText("userName");
                 password=node.getChildText("password");
                 hostString=node.getChildText("hostString");
                }
          }
       catch (IOException io)
       {
          System.out.println(io.getMessage());
       }
       catch (JDOMException jdomex)
       {
          System.out.println(jdomex.getMessage());
        }

  jc.setSelectedItem(backEnd);
  tf6.setText(dsn);
  tf7.setText(dbName);
  tf8.setText(userName);
  tf9.setText(password);
  tf10.setText(hostString);
 }

 if (e.getSource() ==  rb4)
{
```

```
/* Executed  when user clicks on 'JSON Format' button on 'Configure Back End' page */
  try
 {
/* Parsing file "config.json" */
   Object obj = new JSONParser().parse(new FileReader("dbconfig.json"));

   /* Typecasting obj to JSONObject
   JSONObject jo = (JSONObject) obj;

   /* Getting proxyImage Object */
   Map proxy = ((Map)jo.get("database"));

   /* Iterating proxy Map */
   Iterator<Map.Entry> itr = proxy.entrySet().iterator();
   while (itr.hasNext()) {
      Map.Entry pair = itr.next();
      {
       value=(String)pair.getKey();
       if (value.equals("backEnd"))
         backEnd=pair.getValue().toString();
       else if(value.equals("dsn"))
         dsn=pair.getValue().toString();
       else if(value.equals("dbName"))
         dbName=pair.getValue().toString();
        else if(value.equals("userName"))
         userName=pair.getValue().toString();
         else if(value.equals("password"))
         password=pair.getValue().toString();
          else if(value.equals("hostString"))
         hostString=pair.getValue().toString();
      }
```

```
      }
    }
    catch(ParseException e1)
    {
      System.out.println("Error in Parsing...");
    }
    catch(FileNotFoundException e1)
    {
      System.out.println("Configuration File Does Not Exist(3)...");
    }
    catch(IOException e1)
    {
      System.out.println("Error in Reading File...");
    }
    catch(NumberFormatException e1)
    {
      System.out.println("Cannot Convert into Number...");
    }

   jc.setSelectedItem(backEnd);
   tf6.setText(dsn);
   tf7.setText(dbName);
   tf8.setText(userName);
   tf9.setText(password);
   tf10.setText(hostString);
}

if (e.getSource() ==  rb5)
{
    /* Executed  when user clicks on 'XML Format' button on 'View Proxy Products' page */
  File f1=new File("config.xml");
```

```
  File f2=new File("dbconfig.xml");
  if (!f1.exists())
    JOptionPane.showMessageDialog(this,"XML Image Configuration File Does Not Exist...");
  else if (!f2.exists())
     JOptionPane.showMessageDialog(this,"XML Database Configuration File Does Not
                                                Exist...");
  else
    new ProxyProductPattern("XML");
}

if (e.getSource() ==  rb6)
{
  File f3=new File("config.json");
  File f4=new File("dbconfig.json");
  if (!f3.exists())
     JOptionPane.showMessageDialog(this,"JSON Image Configuration File Does Not Exist...");
  else if (!f4.exists())
     JOptionPane.showMessageDialog(this,"JSON Database Configuration File Does Not
                                                Exist...");
  else
    new ProxyProductPattern("JSON");
}

   if (e.getSource() ==  b1)
   {
   if(rb3.isSelected())
   {
     try
     {
       FileOutputStream fos=new FileOutputStream("dbconfig.xml");
      String line="";
```

```
String backend = (String)jc.getSelectedItem();
line="<?xml version='1.0' ?>"+"\r\n";
fos.write(line.getBytes());

line="<dbInfo>"+"\r\n";
fos.write(line.getBytes());

line="<database>"+"\r\n";
fos.write(line.getBytes());
line="  <backEnd>";
fos.write(line.getBytes());

fos.write(backend.getBytes());
line="</backEnd>"+"\r\n";
fos.write(line.getBytes());

line="  <dsn>"+tf6.getText()+"</dsn>"+"\r\n";
fos.write(line.getBytes());
line="  <dbName>"+tf7.getText()+"</dbName>"+"\r\n";
fos.write(line.getBytes());

line="  <userName>"+tf8.getText()+"</userName>"+"\r\n";
fos.write(line.getBytes());

line="  <password>"+tf9.getText()+"</password>"+"\r\n";
fos.write(line.getBytes());

line="  <hostString>"+tf10.getText()+"</hostString>"+"\r\n";
fos.write(line.getBytes());
```

```
    line="</database>"+"\r\n";
    fos.write(line.getBytes());

    line="</dbInfo>"+"\r\n";
    fos.write(line.getBytes());
    fos.close();
    JOptionPane.showMessageDialog(this,"Database Configuration XML File Updated
                                        Successfully...");
 }
  catch(IOException e1)
  {
   System.out.println("Error in Writing...");
  }
}
  else
  {
/* Executed  when user clicks on 'JSON Format' button on 'View Proxy Products' page */
   try
   {
     /* creating JSONObject */
     JSONObject jo = new JSONObject();
      String backend = (String)jc.getSelectedItem();
     /* for proxy data, first create LinkedHashMap */
     Map m = new LinkedHashMap(6);
     m.put("backEnd", backend);
     m.put("dsn", tf6.getText());
     m.put("dbName", tf7.getText());
     m.put("userName", tf8.getText());
     m.put("password", tf9.getText());
     m.put("hostString", tf10.getText());
     /* putting address to JSONObject */
```

```
      jo.put("database", m);
      /* writing JSON to file:"config.json"  */
      PrintWriter pw = new PrintWriter("dbconfig.json");
      pw.write(jo.toJSONString());
      pw.flush();
      pw.close();
    }
    catch(IOException e1)
    {
      System.out.println("Error in Writing...");
    }
    JOptionPane.showMessageDialog(this,"Database Configuration JSON File Updated
                                        Successfully...");
   }
}

 if (e.getSource() ==  b)
 {
 String line="";
 if(rb2.isSelected())
 {
    try
   {
     /* creating JSONObject */
     JSONObject jo = new JSONObject();

     /* For proxy data, first create LinkedHashMap */
     Map m = new LinkedHashMap(3);
     m.put("height", Integer.parseInt(tf1.getText()));
     m.put("width", Integer.parseInt(tf2.getText()));
     m.put("count", Integer.parseInt(tf5.getText()));
```

```
 /* Putting address to JSONObject */
 jo.put("proxyImage", m);

 m = new LinkedHashMap(2);
 m.put("height", Integer.parseInt(tf4.getText()));
 m.put("width", Integer.parseInt(tf3.getText()));

 /* Adding map to list */
  jo.put("realImage", m);

 /* Writing JSON to file:"config.json" */
 PrintWriter pw = new PrintWriter("config.json");
 pw.write(jo.toJSONString());
 pw.flush();
 pw.close();
 JOptionPane.showMessageDialog(this,"Configuration JSON File Updated Successfully...");
 }
 catch(IOException e1)
 {
   System.out.println("Error in Writing...");
 }
}
else
{
 line="";
 try
 {
FileOutputStream fos=new FileOutputStream("config.xml");
line="<?xml version='1.0'?>"+"\r\n";
fos.write(line.getBytes());
```

```
    line="<config>"+"\r\n";
    fos.write(line.getBytes());
    line="<proxyImage>"+"\r\n";
    fos.write(line.getBytes());
    line="<height>"+tf1.getText()+"</height>"+"\r\n";
    fos.write(line.getBytes());
    line="<width>"+tf2.getText()+"</width>"+"\r\n";
    fos.write(line.getBytes());
    line="<count>"+tf5.getText()+"</count>"+"\r\n";
    fos.write(line.getBytes());
    line="</proxyImage>"+"\r\n";
    fos.write(line.getBytes());
    line="<realImage>"+"\r\n";
    fos.write(line.getBytes());
    line="<height>"+tf4.getText()+"</height>"+"\r\n";
    fos.write(line.getBytes());
    line="<width>"+tf3.getText()+"</width>"+"\r\n";
    fos.write(line.getBytes());
    line="</realImage>"+"\r\n";
    fos.write(line.getBytes());
    line="</config>"+"\r\n";
    fos.write(line.getBytes());
    fos.close();
  }
  catch(IOException e1)
  {
    System.out.println("Error in Writing...");
  }
      JOptionPane.showMessageDialog(this,"Configuration XML File Updated Successfully...");
 }
}
```

```
}
 /* Create Layout for Tab 1 */
  public void createPage1()
  {
    panel1 = new JPanel();
    panel1.setLayout( null );

    rb1=new JRadioButton("XML Format");
    rb1.setBounds(100,10,100,30);
    rb1.addActionListener(this);

    rb2=new JRadioButton("JSON Format");
    rb2.setBounds(300,10,100,30);
    rb2.addActionListener(this);
    bg=new ButtonGroup();
    bg.add(rb1);bg.add(rb2);

    lbl1=new JLabel("Proxy Image Height : ");
    lbl1.setBounds(100,50,400,30);
    lbl1.setFont(new Font("Arial",Font.BOLD,24));
    lbl1.setForeground(Color.red);

    tf1=new JTextField(50);
    tf1.setBounds(350,50,100,30);

    lbl2=new JLabel("Proxy Image Width :");
    lbl2.setBounds(100,150,400,30);
    lbl2.setFont(new Font("Arial",Font.BOLD,24));
    lbl2.setForeground(Color.red);

    tf2=new JTextField(50);
```

```
tf2.setBounds(350,150,100,30);

lbl3=new JLabel("Real Image Height :");
lbl3.setBounds(100,250,400,30);
lbl3.setFont(new Font("Arial",Font.BOLD,24));
lbl3.setForeground(Color.red);

tf3=new JTextField(50);
tf3.setBounds(350,250,100,30);

lbl4=new JLabel("Real Image Width :");
lbl4.setBounds(100,350,400,30);
lbl4.setFont(new Font("Arial",Font.BOLD,24));
lbl4.setForeground(Color.red);

tf4=new JTextField(50);
tf4.setBounds(350,350,100,30);

lbl5=new JLabel("No. of Proxy Objects/Row :");
lbl5.setBounds(100,450,400,30);
lbl5.setFont(new Font("Arial",Font.BOLD,20));
lbl5.setForeground(Color.red);

tf5=new JTextField(50);
tf5.setBounds(350,450,100,30);

b=new JButton("Store Config Info");
b.setBounds(150,550,200,30);
b.addActionListener(this);

panel1.add(rb1);
```

```
    panel1.add(rb2);
    panel1.add(lbl1);
    panel1.add(tf1);
    panel1.add(lbl2);
    panel1.add(tf2);
    panel1.add(lbl3);
    panel1.add(tf3);
    panel1.add(lbl4);
    panel1.add(tf4);
    panel1.add(lbl5);
    panel1.add(tf5);
    panel1.add(b);
   }
/* Create Layout for Tab 2 */
  public void createPage2()
  {
    panel2 = new JPanel();
    panel2.setLayout( null );

    rb3=new JRadioButton("XML Format");
    rb3.setBounds(100,10,100,30);
    rb3.addActionListener(this);

    rb4=new JRadioButton("JSON Format");
    rb4.setBounds(300,10,100,30);
    rb4.addActionListener(this);
    bg1=new ButtonGroup();
    bg1.add(rb3);bg1.add(rb4);

    lbl6=new JLabel("Back End  : ");
    lbl6.setBounds(100,50,400,30);
```

```
lbl6.setFont(new Font("Arial",Font.BOLD,24));
lbl6.setForeground(Color.red);

jc=new JComboBox(db);

jc.setBounds(350,50,100,30);

lbl7=new JLabel("Data Source Name :");
lbl7.setBounds(100,150,400,30);
lbl7.setFont(new Font("Arial",Font.BOLD,24));
lbl7.setForeground(Color.red);

tf6=new JTextField(50);
tf6.setBounds(350,150,100,30);

lbl8=new JLabel("Database Name :");
lbl8.setBounds(100,250,400,30);
lbl8.setFont(new Font("Arial",Font.BOLD,24));
lbl8.setForeground(Color.red);

tf7=new JTextField(50);
tf7.setBounds(350,250,100,30);

lbl9=new JLabel("User Name :");
lbl9.setBounds(100,350,400,30);
lbl9.setFont(new Font("Arial",Font.BOLD,24));
lbl9.setForeground(Color.red);

tf8=new JTextField(50);
tf8.setBounds(350,350,100,30);
```

```
lbl10=new JLabel("Password :");
lbl10.setBounds(100,450,400,30);
lbl10.setFont(new Font("Arial",Font.BOLD,20));
lbl10.setForeground(Color.red);

tf9=new JTextField(50);
tf9.setBounds(350,450,100,30);

lbl11=new JLabel("Host String :");
lbl11.setBounds(100,550,400,30);
lbl11.setFont(new Font("Arial",Font.BOLD,20));
lbl11.setForeground(Color.red);

tf10=new JTextField(50);
tf10.setBounds(350,550,100,30);

b1=new JButton("Store Back End Info");
b1.setBounds(150,650,200,30);
b1.addActionListener(this);

panel2.add(rb3);
panel2.add(rb4);
panel2.add(lbl6);
panel2.add(jc);
panel2.add(lbl7);
panel2.add(tf6);
panel2.add(lbl8);
panel2.add(tf7);
panel2.add(lbl9);
panel2.add(tf8);
panel2.add(lbl10);
```

```
    panel2.add(tf9);
    panel2.add(lbl11);
    panel2.add(tf10);
    panel2.add(b1);
  }
/* Create Layout for Tab 3 */
 public void createPage3()
 {
            panel3 = new JPanel();
            panel3.setLayout( null );
             rb5=new JRadioButton("XML Format");
             rb5.setBounds(100,10,100,30);
             rb5.addActionListener(this);
             rb6=new JRadioButton("JSON Format");
             rb6.setBounds(300,10,100,30);
             rb6.addActionListener(this);
             bg2=new ButtonGroup();
             bg2.add(rb5);
             bg2.add(rb6);
             panel3.add(rb5);
             panel3.add(rb6);
 }

   /* Main method to get things started */
  public static void main( String args[] )
  {
                    /* Create an instance of the test application  */
                    MainFrame mainFrame          = new MainFrame();
                    mainFrame.setVisible( true );
  }
}
```

JSON Files Generated by the Module

config.json

{"proxyImage":{"height":100,"width":100,"count":6},"realImage":{"height":400,"width":300}}

dbconfig.json

{"database":{"backEnd":"MySQL","dsn":"","dbName":"designpattern","userName":"root","password":""
,"hostString":""}}

XML Files Generated by the Module

config.xml

```
<?xml version='1.0'?>
<config>
<proxyImage>
<height>200</height>
<width>200</width>
<count>5</count>
</proxyImage>
<realImage>
<height>400</height>
<width>300</width>
</realImage>
</config>
```


dbconfig.xml (for Oracle)

```
<?xml version='1.0' ?>
<dbInfo>
<database>
<backEnd>Oracle</backEnd>
<dsn></dsn>
<dbName></dbName>
```

```
<userName>system</userName>
<password>siber</password>
<hostString>orcl</hostString>
</database>
</dbInfo>
```

dbconfig.xml (for MySQL)

```
<?xml version='1.0' ?>
<dbInfo>
<database>
<backEnd>MySQL</backEnd>
<dsn></dsn>
<dbName>designpattern</dbName>
<userName>root</userName>
<password></password>
<hostString></hostString>
</database>
</dbInfo>
```


dbconfig.xml (for MS-Access)

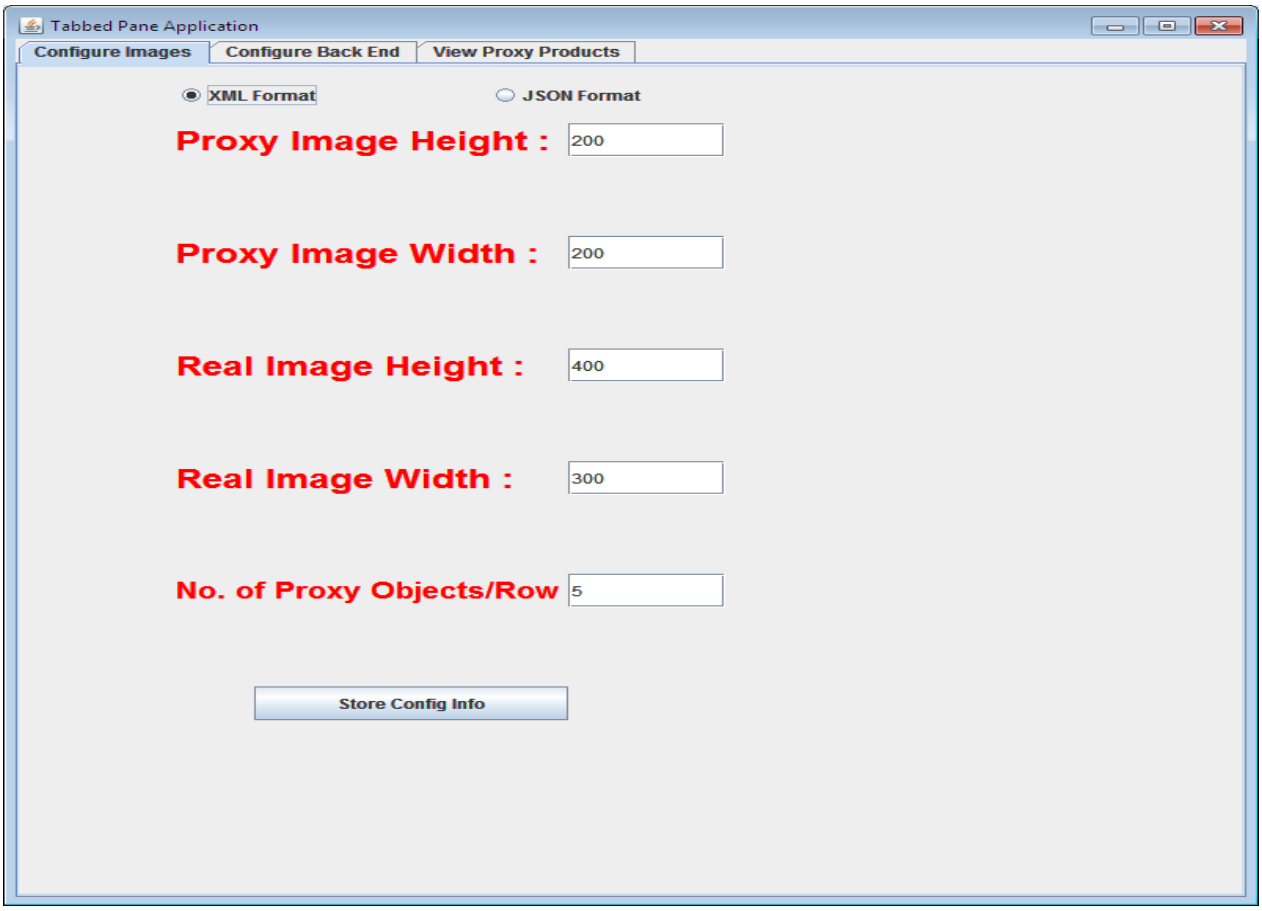
```
<?xml version='1.0' ?>
<dbInfo>
<database>
<backEnd>MS-Access</backEnd>
<dsn>designpattern</dsn>
<dbName></dbName>
<userName></userName>
<password></password>
</database>
</dbInfo>
```

8.5 Test Cases and Screen Shots

Test Case 1: Execute the application

Desired Output : A frame window with tabbed dialog for changing image configuration information, back end information and for viewing proxy objects should be displayed. The first page of the tabbed dialog should be visible.

Actual Output :

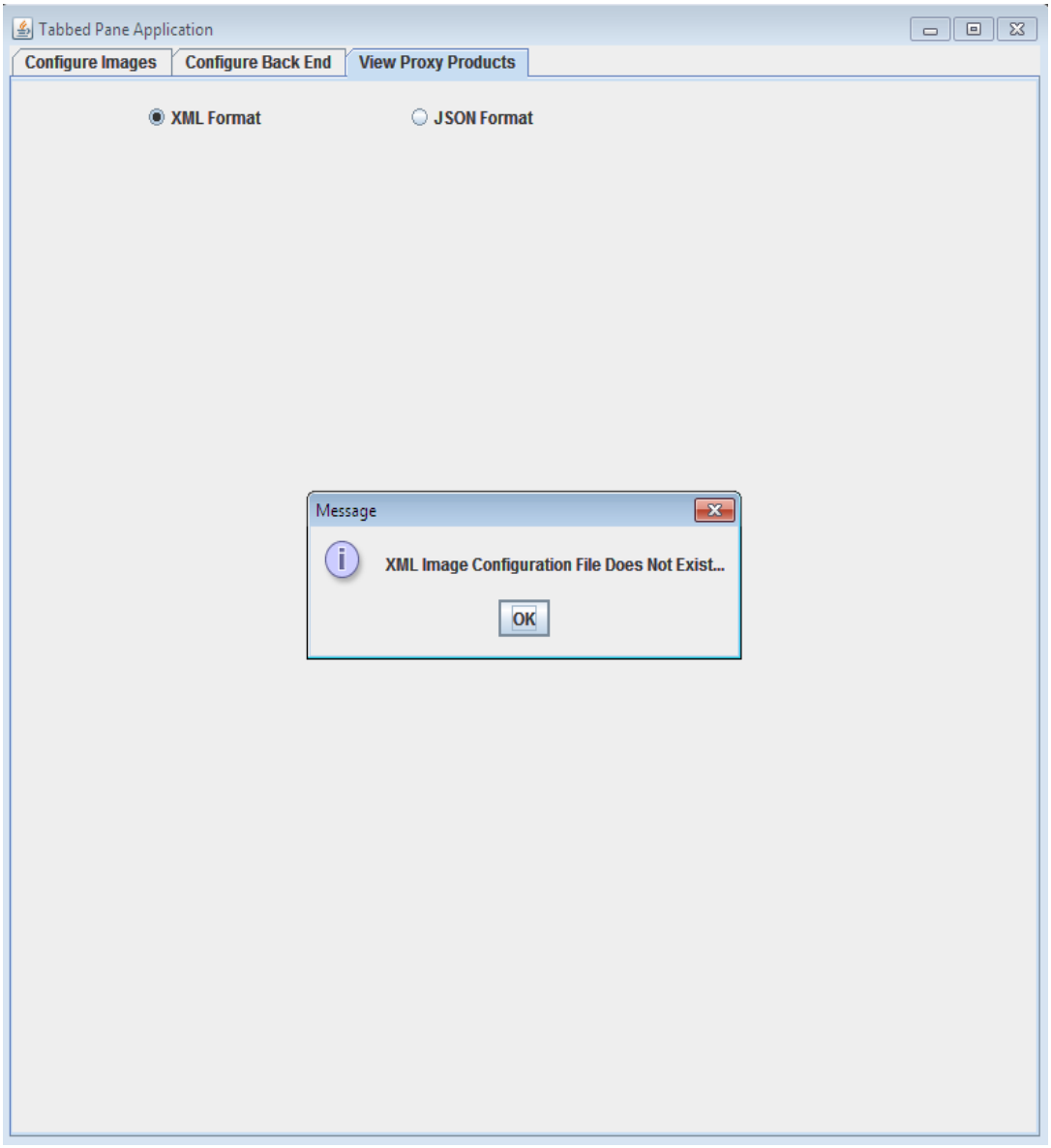


Test Case 2: Click , on "**View Proxy Products**" tab and select "**XML Format**" option button.

Pre-Condition : "**config.xml**" file does not exist.

Desired Output : Message "XML Image Configuration Information Does not Exist…" message should be displayed.

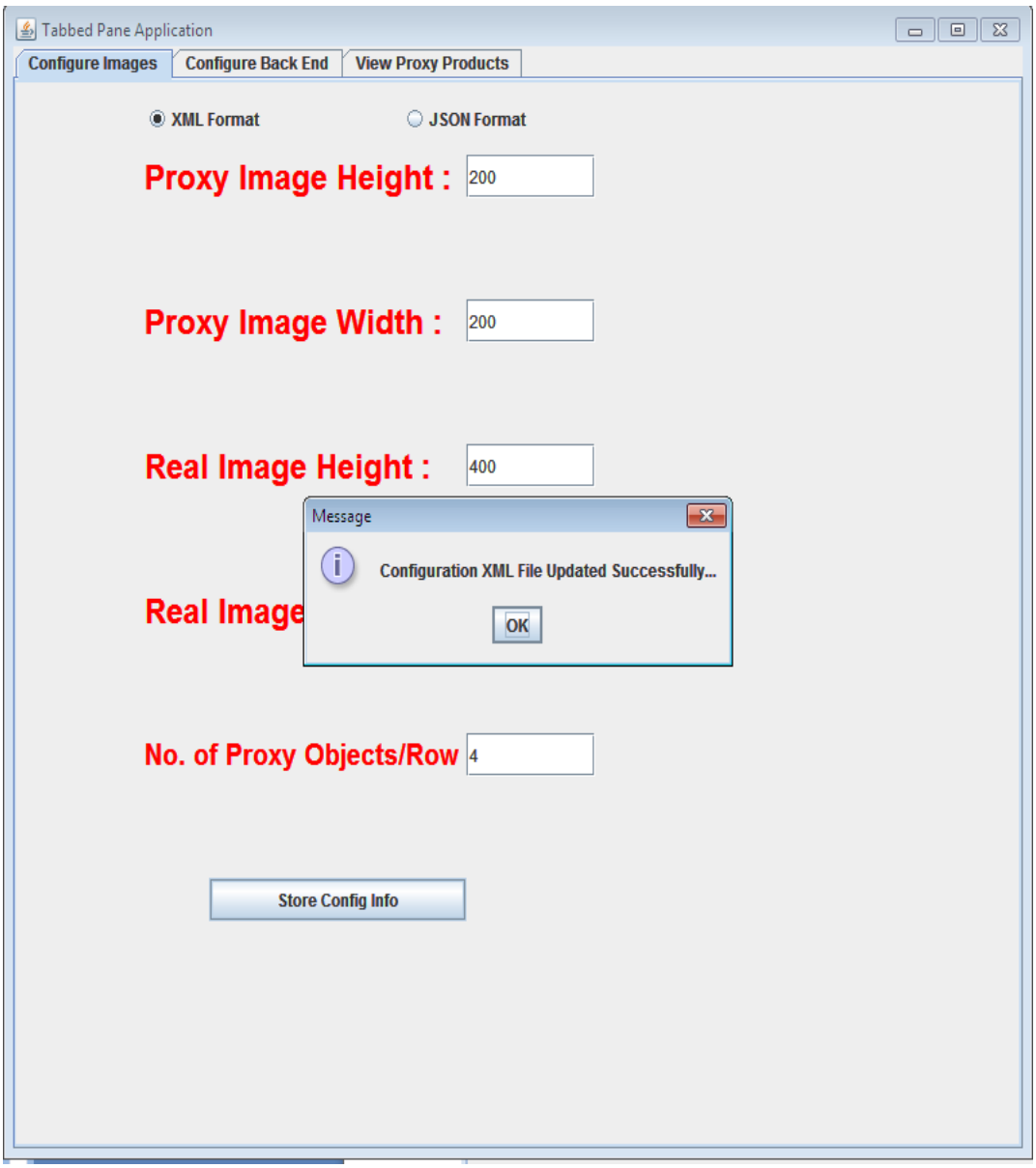
Actual Output:



Test Case 3: Click on "**Configure Images**" tab and select "XML Format" option button and enter the image configuration information and click on "**Store Config Info**" button.

Desired Output : The image configuration information should be saved in a config.xml file and a message "Configuration XML File Updated Successfully" should be displayed to an end user.

Actual Output:

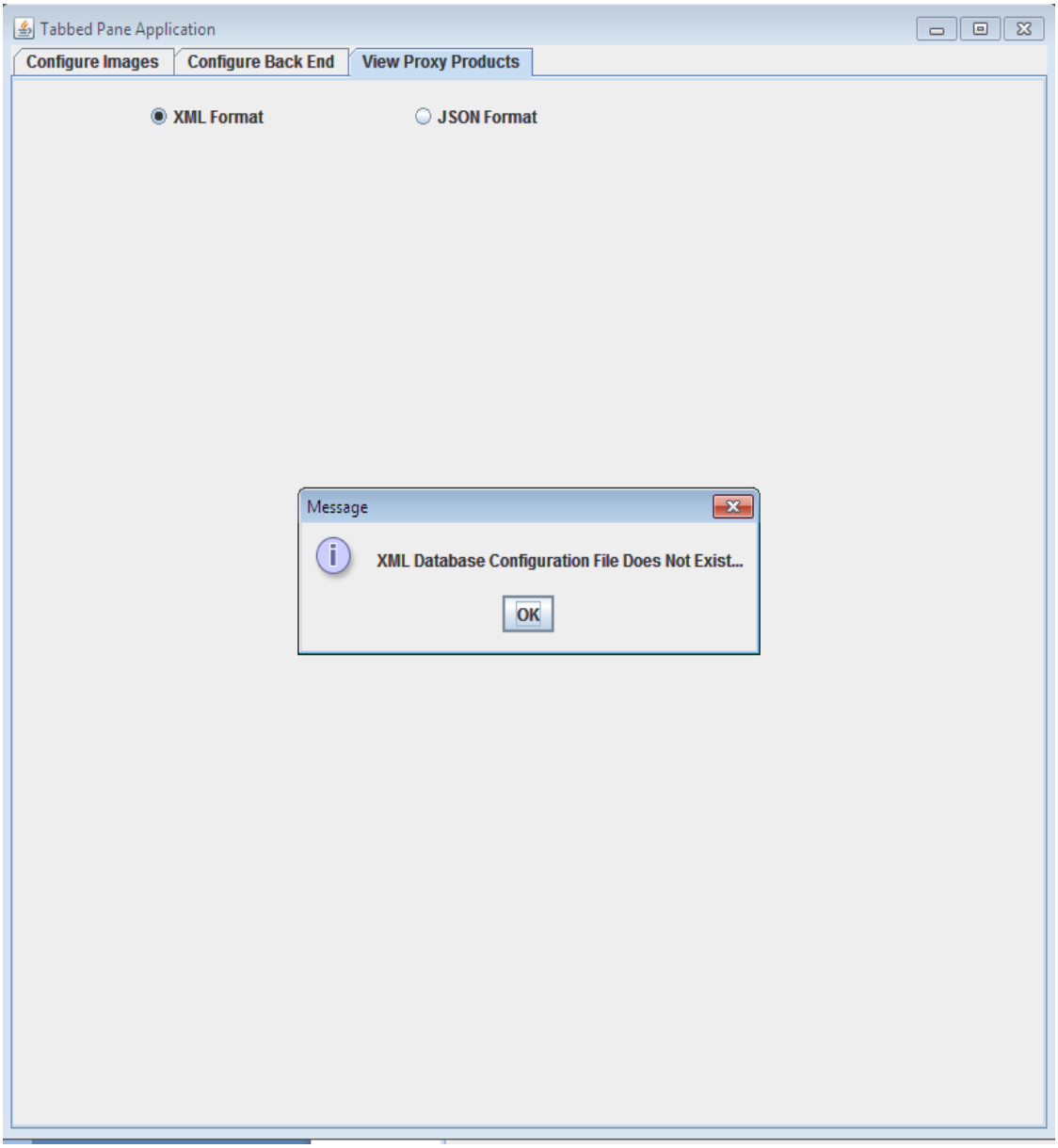


Test Case 4: Click , on "**View Proxy Products**" tab and select "**XML Format**" option button.

Pre-Condition : "**config.xml**" file exists, "**dbconfig.xml**" file does not exist.

Desired Output : Message "XML Image Configuration Information Does not Exist…" message should be displayed.

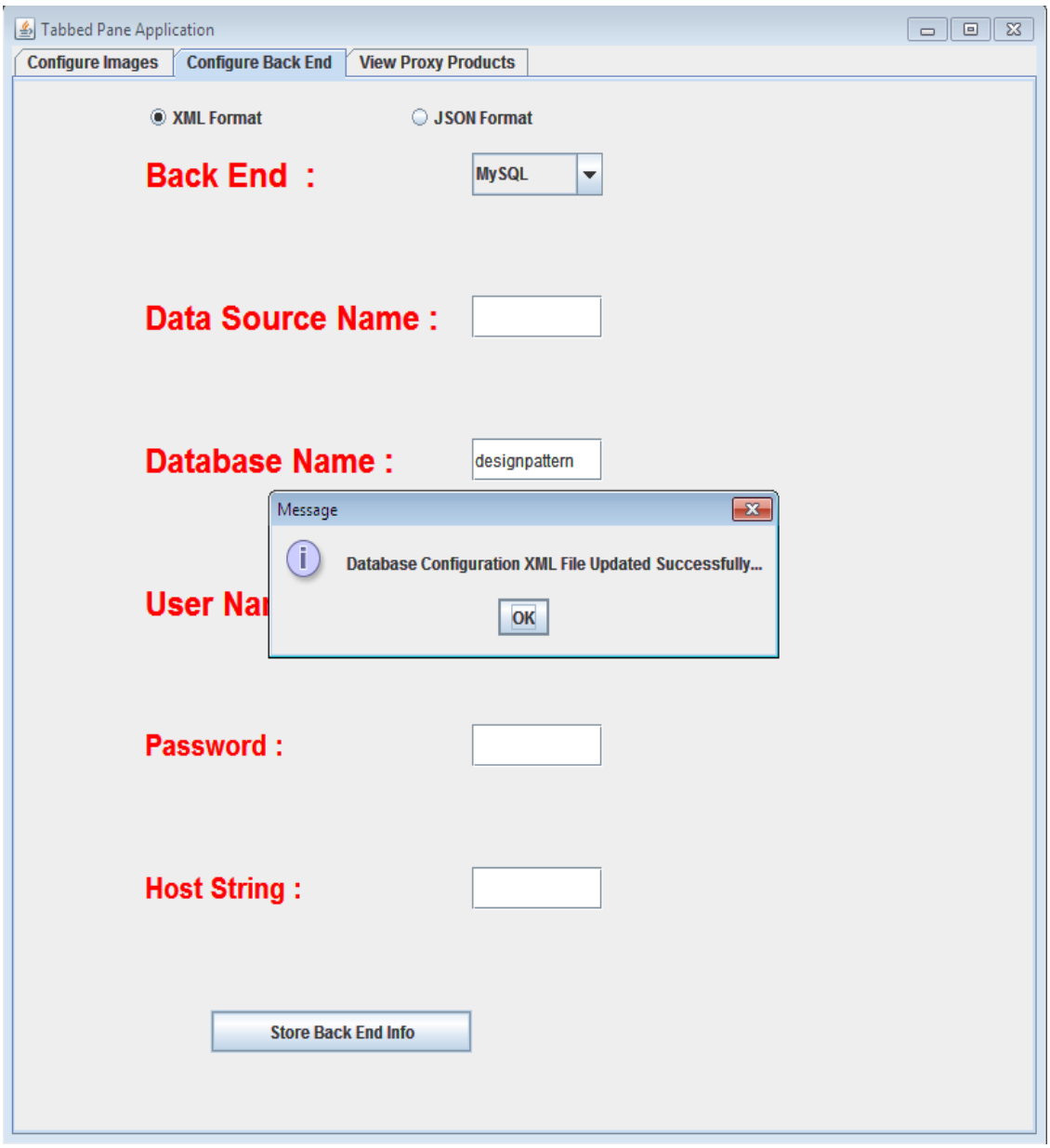
Actual Output:



Test Case 5: Click on "**Configure Back End**" tab and select "XML Format" option button and enter the back end configuration information and click on "**Store Back End Info**" button.

Desired Output : The image configuration information should be saved in a config.xml file and a message "Database Configuration XML File Updated Successfully" should be displayed to an end user.

Actual Output:

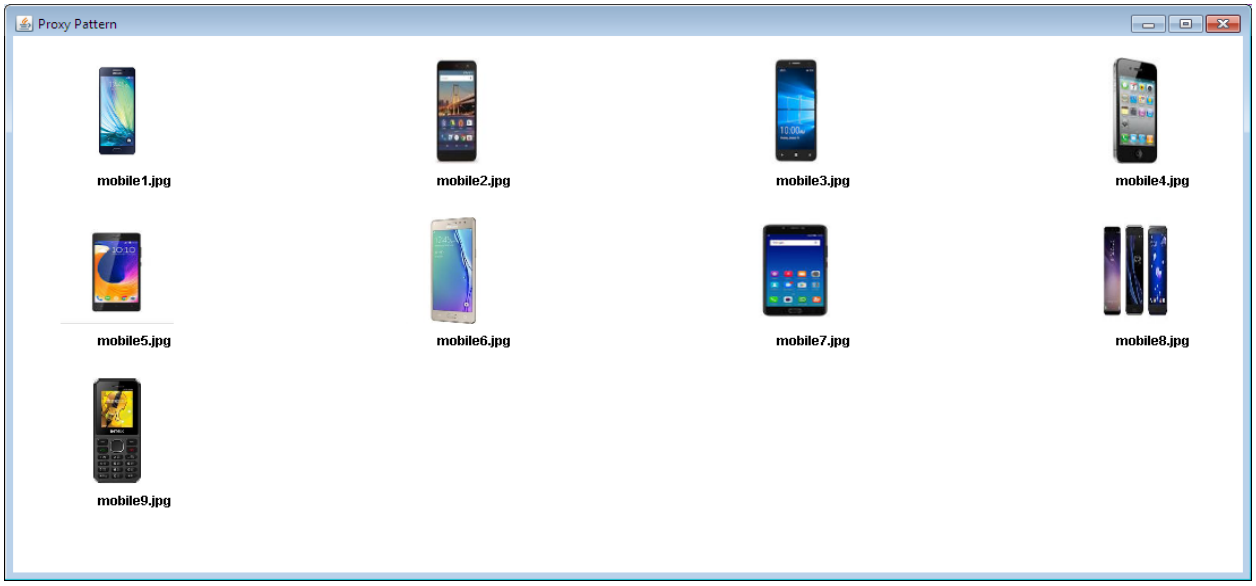


Test Case 6: Click , on "**View Proxy Products**" tab and select "**XML Format**" option button.

Pre-Condition : Both "**config.xml**" and "**dbconfig.xml**" exist.

Desired Output : Based on image configuration information stored in "config.xml" file, count no. of proxy products should be displayed in a single row by reading the relevant backend information stored in "dbconfig.xml" file.

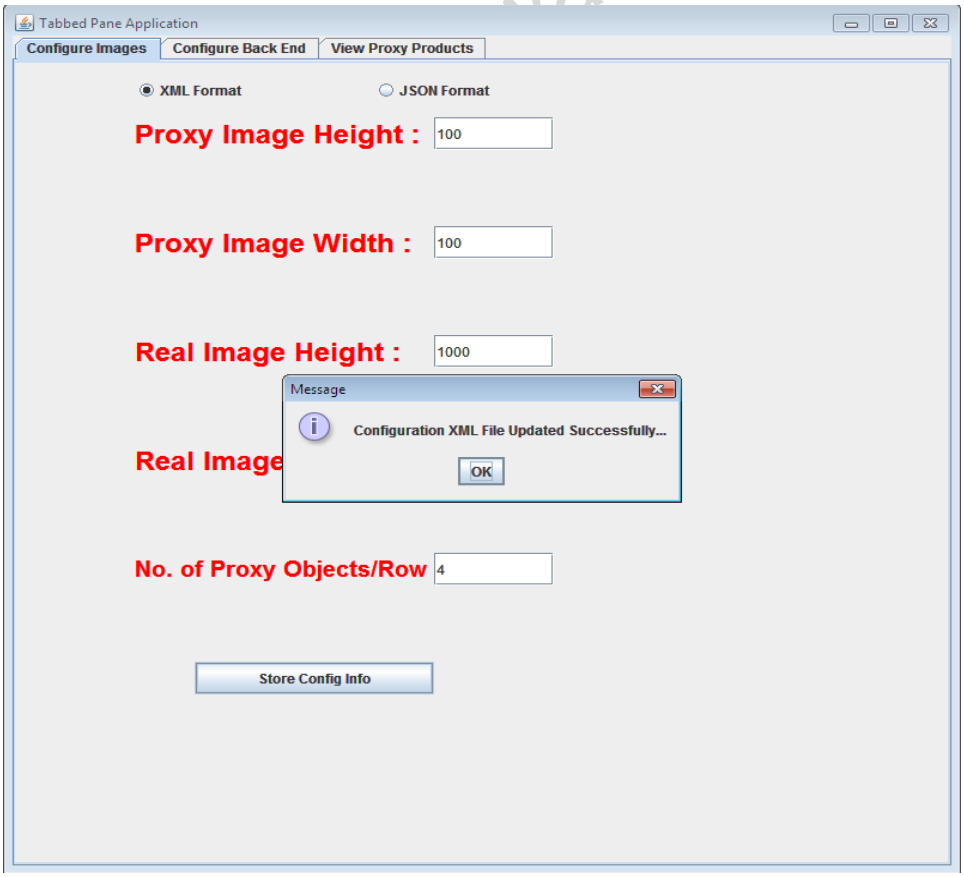
Actual Output:



Test Case 7: Click on "**Configure Images**" tab, and select "**XML Format**" option button.

Desired Output : The current image configuration information should be displayed in edit text boxes which can be updated by an end user.

Actual Output:



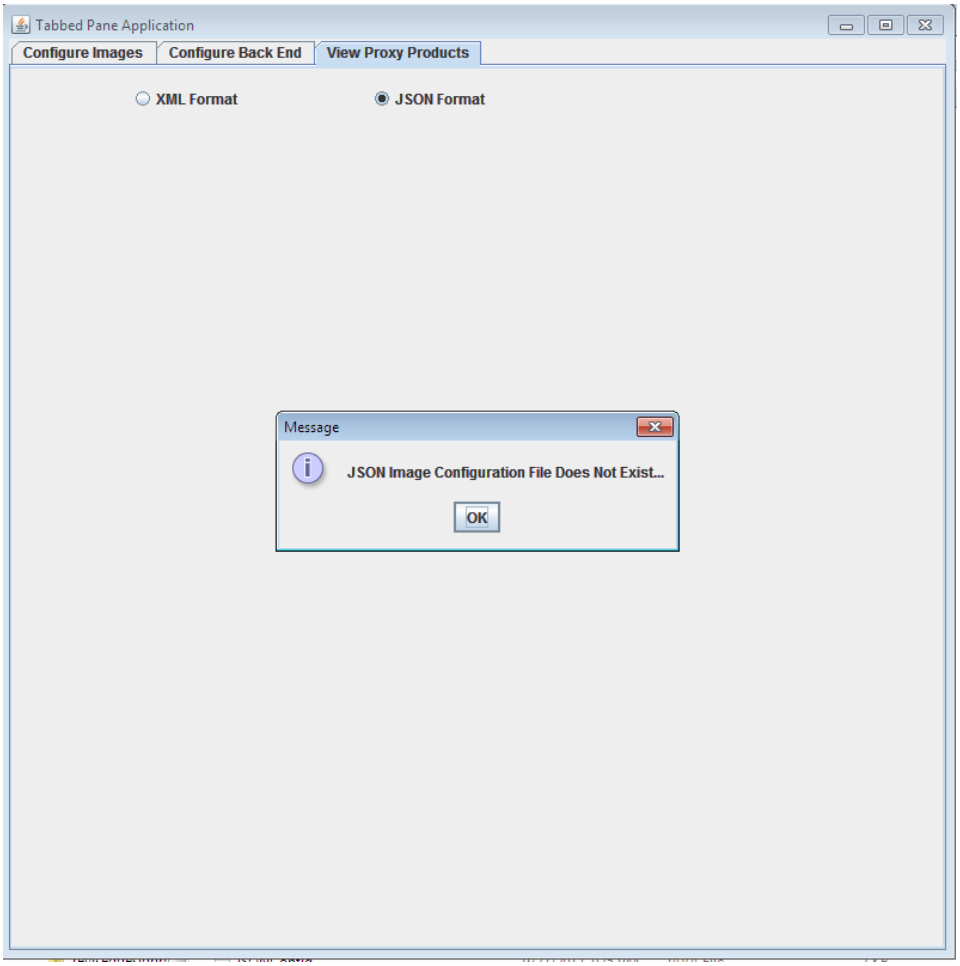
Same Test Cases are repeated for JSON File Format.

Test Case 8: Click on "**View Proxy Products**" tab and select "**JSON Format**" option button.

Pre-Condition : "**config.json**" file does not exist.

Desired Output : Message "JSON Image Configuration Information Does not Exist…" message should be displayed to an end user.

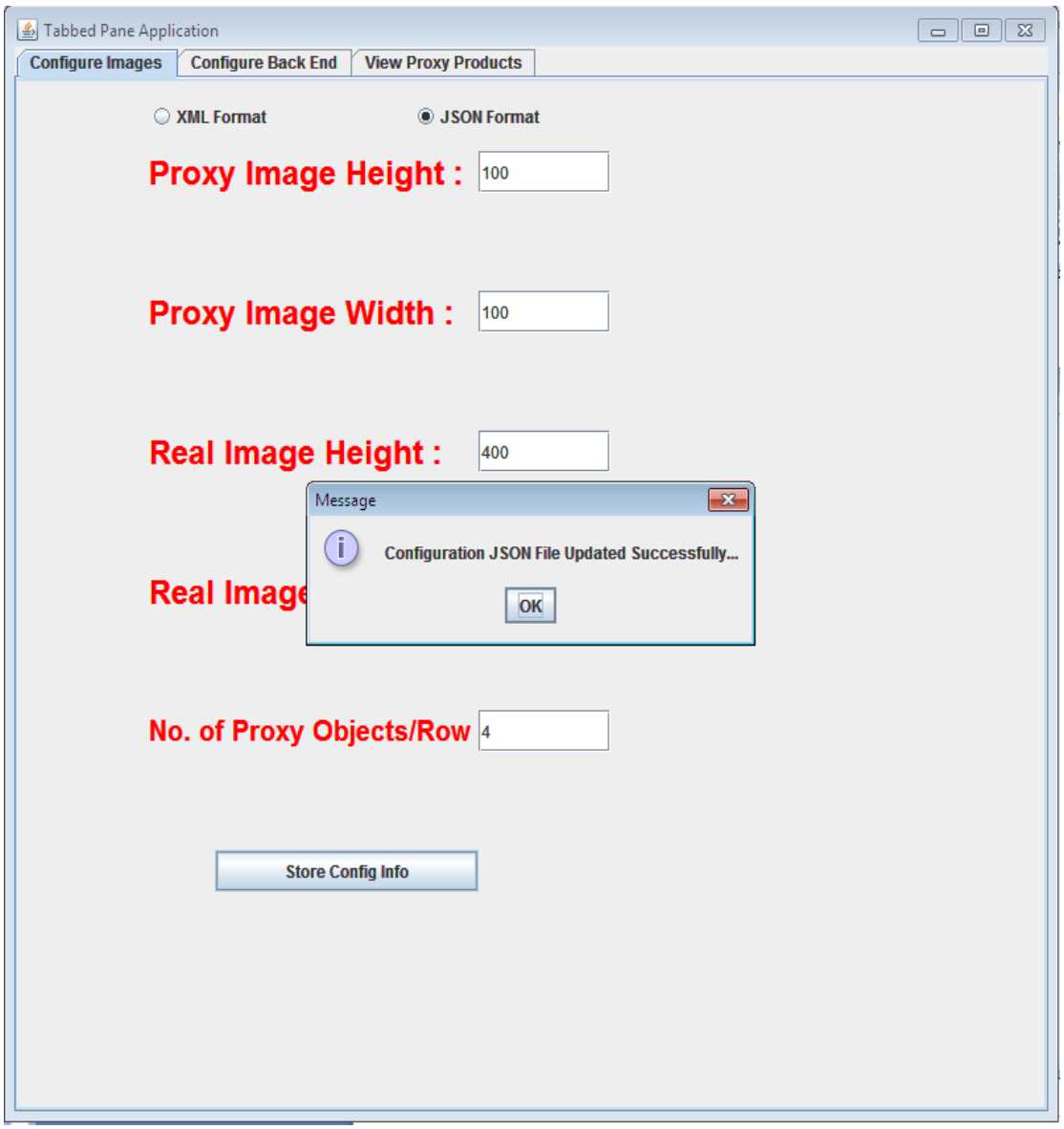
Actual Output:



Test Case 9: Click on "**Configure Images**" tab and select "**JSON Format**" option button and enter the image configuration information and click on "**Store Config Info**" button.

Desired Output : The image configuration information should be saved in a config.json file and a message "Configuration JSON File Updated Successfully" should be displayed to an end user.

Actual Output:

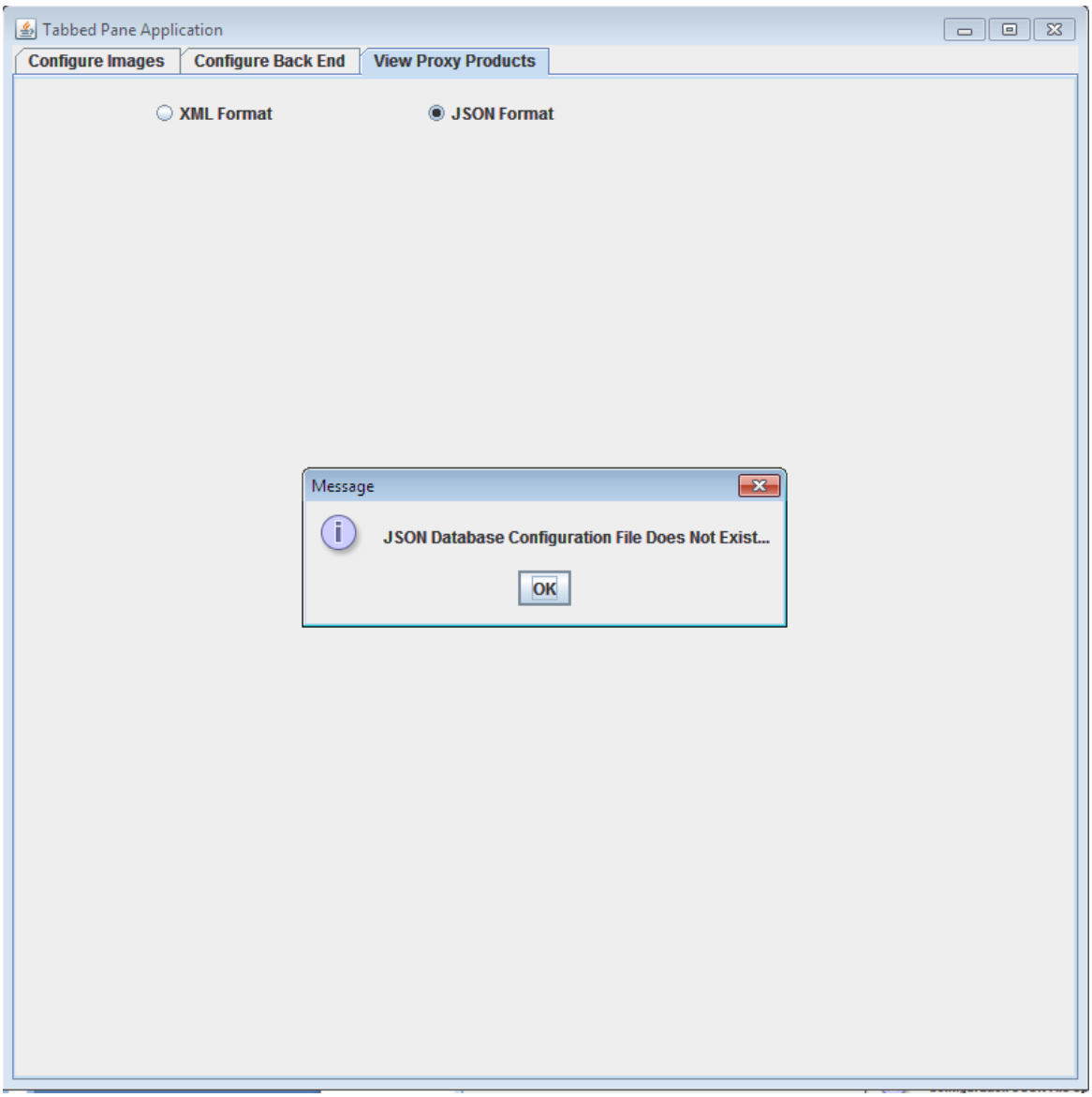


Test Case 10: Click , on "**View Proxy Products**" tab and select "**JSON Format**" option button.

Pre-Condition : "**config.json**" file exists, "**dbconfig.json**" file does not exist.

Desired Output : Message "JSON Image Configuration Information Does not Exist…" message should be displayed.

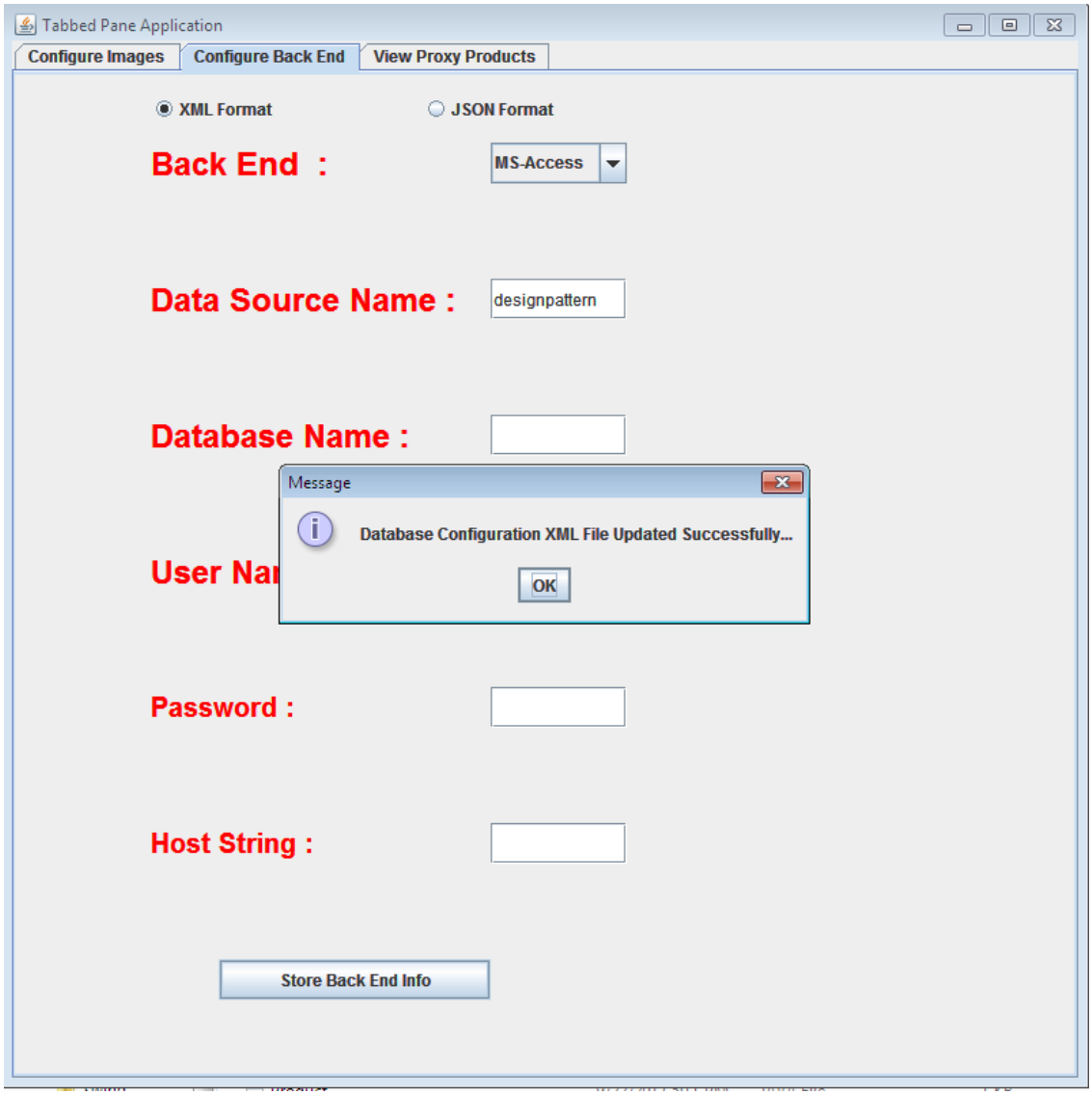
Actual Output:



Test Case 11: Click on "**Configure Back End**" tab and select "**JSON Format**" option button and enter the back end configuration information and click on "**Store Back End Info**" button.

Desired Output : The image configuration information should be saved in a config.json file and a message "Database Configuration JSON File Updated Successfully" should be displayed to an end user.

Actual Output:



Test Case 12: Click , on "**View Proxy Products**" tab and select "**JSON Format**" option button.

Pre-Condition : Both "**config.json**" and "**dbconfig.json**" exist.

Desired Output : Based on image configuration information stored in "config.json" file, count no. of proxy products should be displayed in a single row by reading the relevant backend information stored in "dbconfig.json" file.

Actual Output:

Test Case 13: Click on "mobile2.jpg" image.

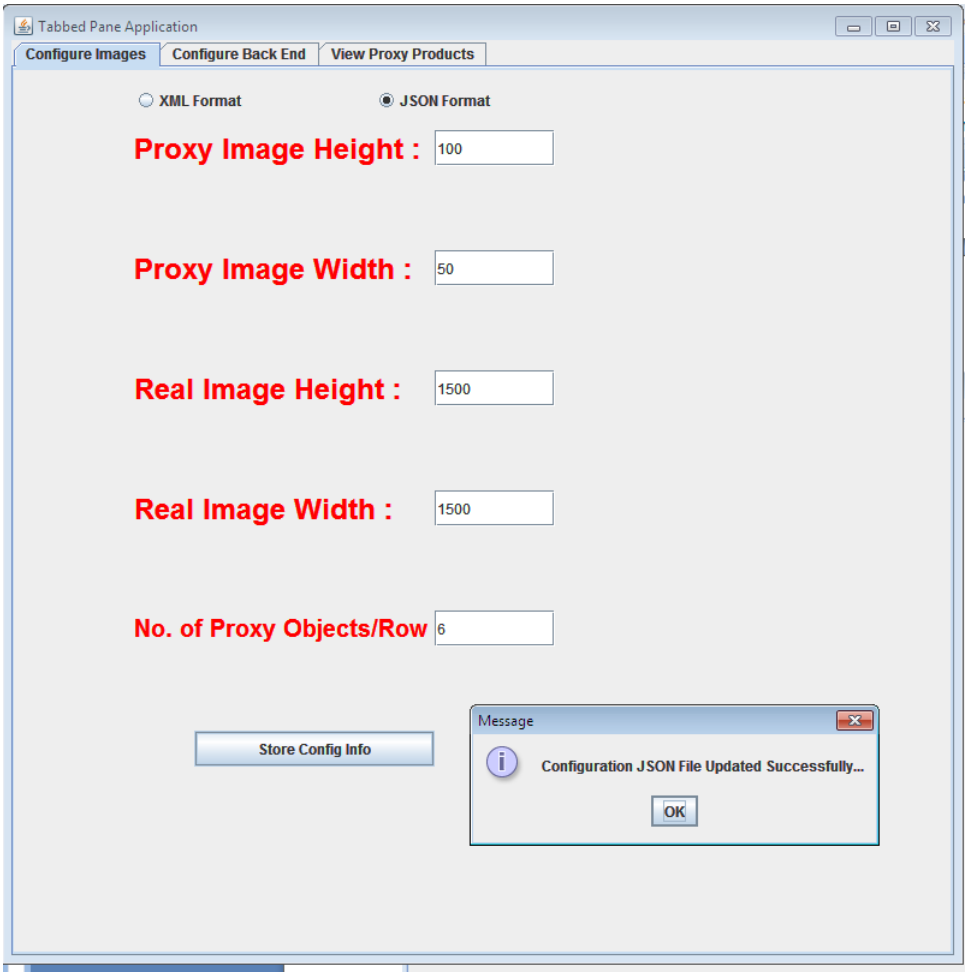
Desired Output : Real product by reading the image configuration information stored in config.json file should be displayed along with its detailed information.

Actual Output:

Test Case 14: Click on "**Configure Images**" tab, and select "**JSON Format**" option button.

Desired Output : The current image configuration information should be displayed in edit text boxes which can be updated by an end user.

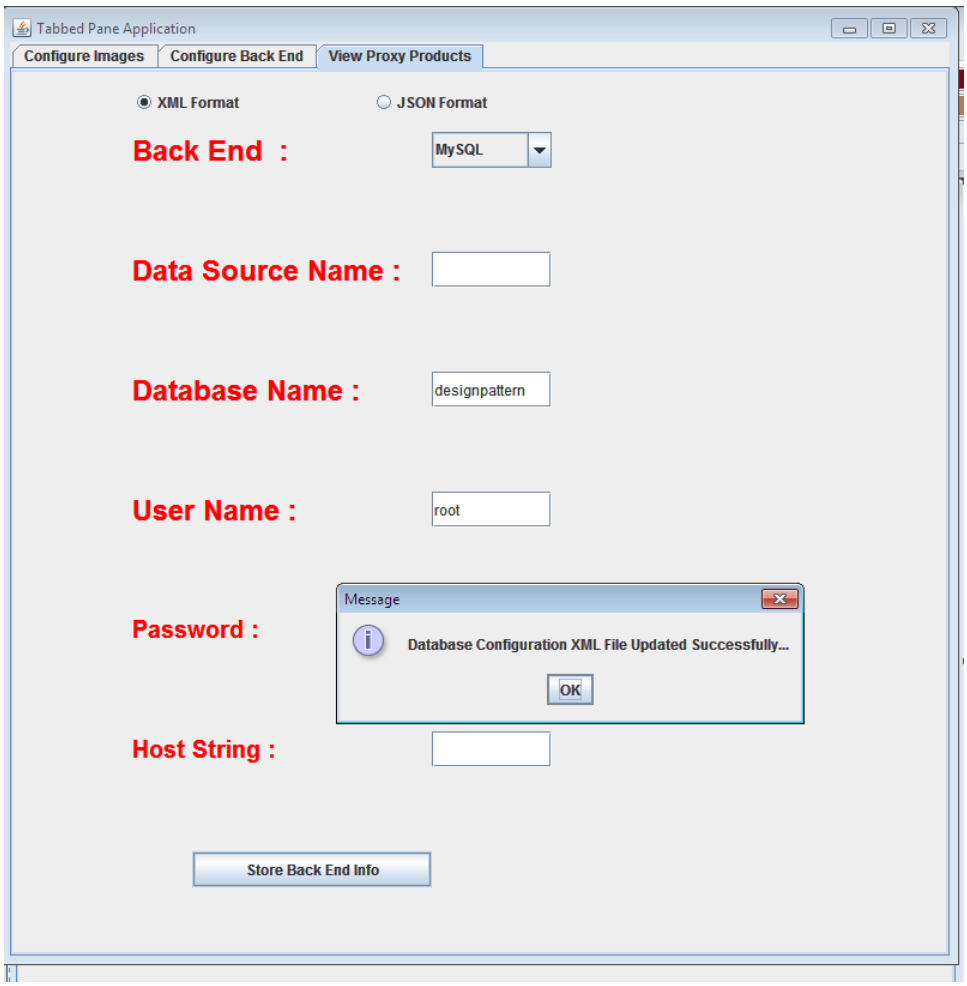
Actual Output:



Test Case 15: (for MySQL Database) Select **"Configure Back End"** tab and select "**JSON Format**" option button, enter the backend configuration information and click on "Store Back End Info" button.

Desired Output : The back end configuration information should be saved in a dbconfig.xml file and a message "Database Configuration XML File Updated Successfully" should be displayed to an end user.

Actual Output:



Test Case 16: Click , on "**View Proxy Products**" tab and select "**XML Format**" option button.

Pre-Condition : Both "**config.xml**" and "**dbconfig.xml**", storing MySQL connectivity information exist.

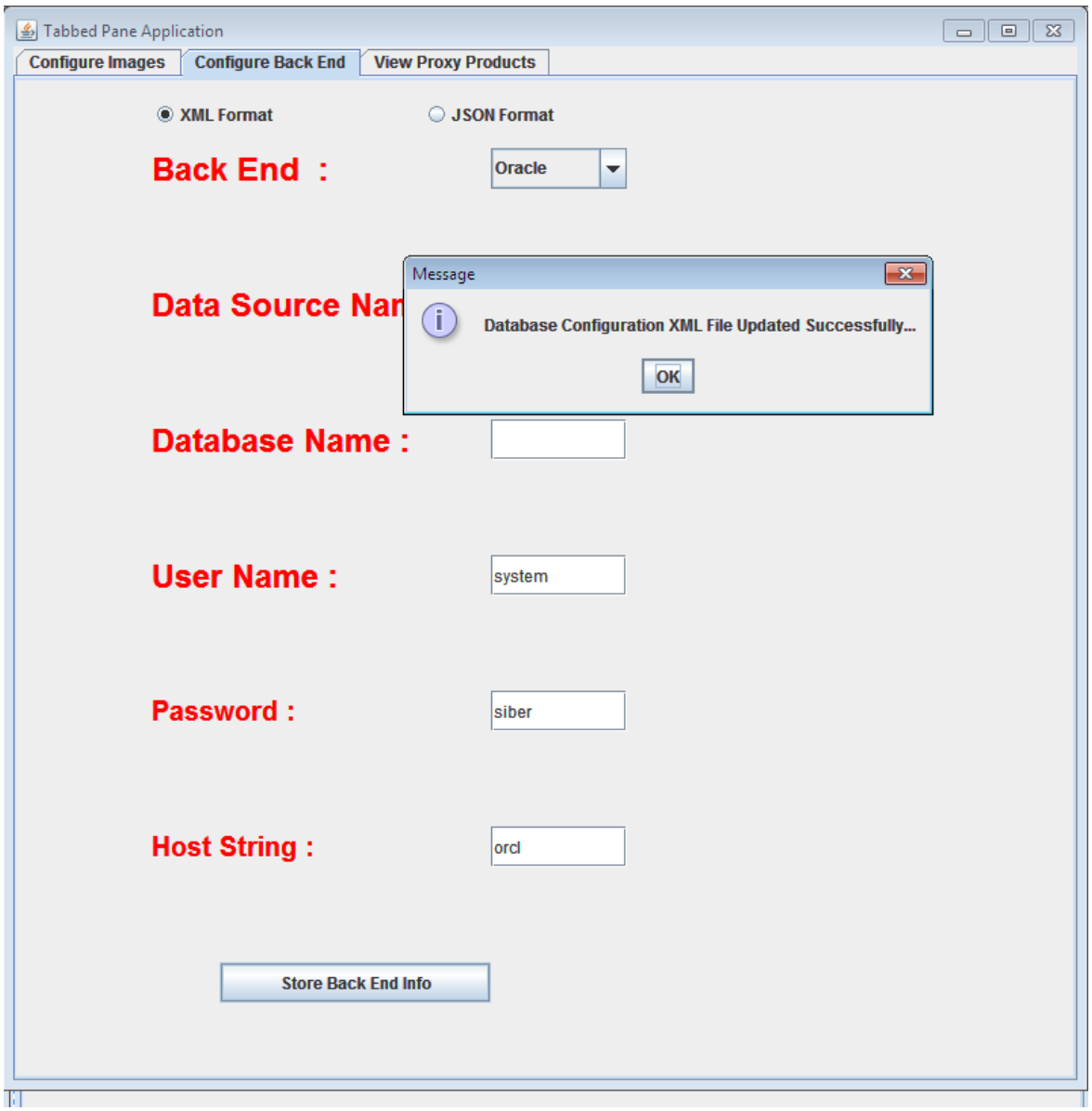
Desired Output : Based on image configuration information stored in "config.json" file, count no. of proxy products should be displayed in a single row by reading the relevant backend information stored in "dbconfig.json" file.

Actual Output:

Test Case 17: (for Oracle Database) Select **"Configure Back End"** tab and select "**JSON Format**" option button, enter the backend configuration information and click on "**Store Back End Info**" button.

Desired Output : The back end configuration information should be saved in a dbconfig.xml file and a message "Database Configuration JSON File Updated Successfully" should be displayed to an end user.

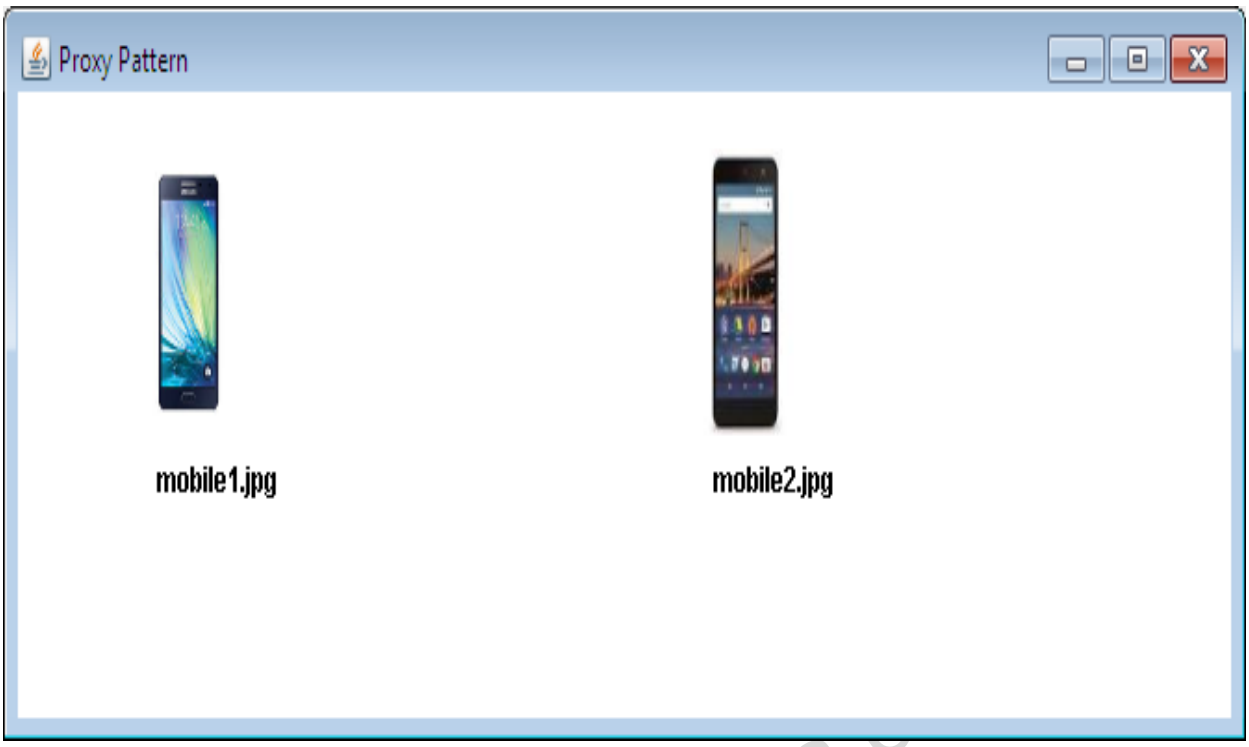
Actual Output:



Test Case 18: Click , on "**View Proxy Products**" tab and select "**XML Format**" option button.

Pre-Condition : Both "**config.xml**" and "**dbconfig.xml**", storing Oracle connectivity information exist.

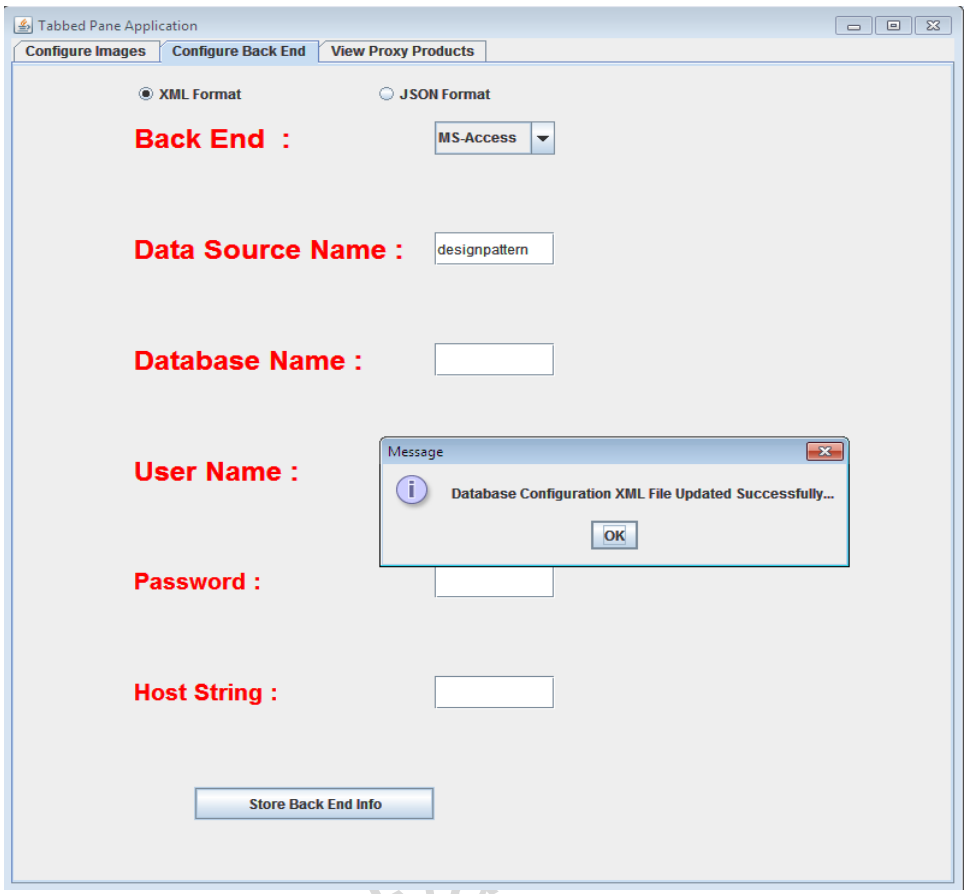
Desired Output : Based on image configuration information stored in "config.xml" file, count no. of proxy products should be displayed in a single row by reading the relevant backend information stored in "dbconfig.xml" file.



Test Case 19: (for MS-Access Database) Select **"**Configure Back End**"** tab and select "XML Format" option button, enter the backend configuration information and click on "Store Back End Info" button.

Desired Output : The back end configuration information should be saved in a dbconfig.xml file and a message "Database Configuration XML File Updated Successfully" should be displayed to an end user.

Actual Output:



Test Case 20: Click , on "**View Proxy Products**" tab and select "**XML Format**" option button.

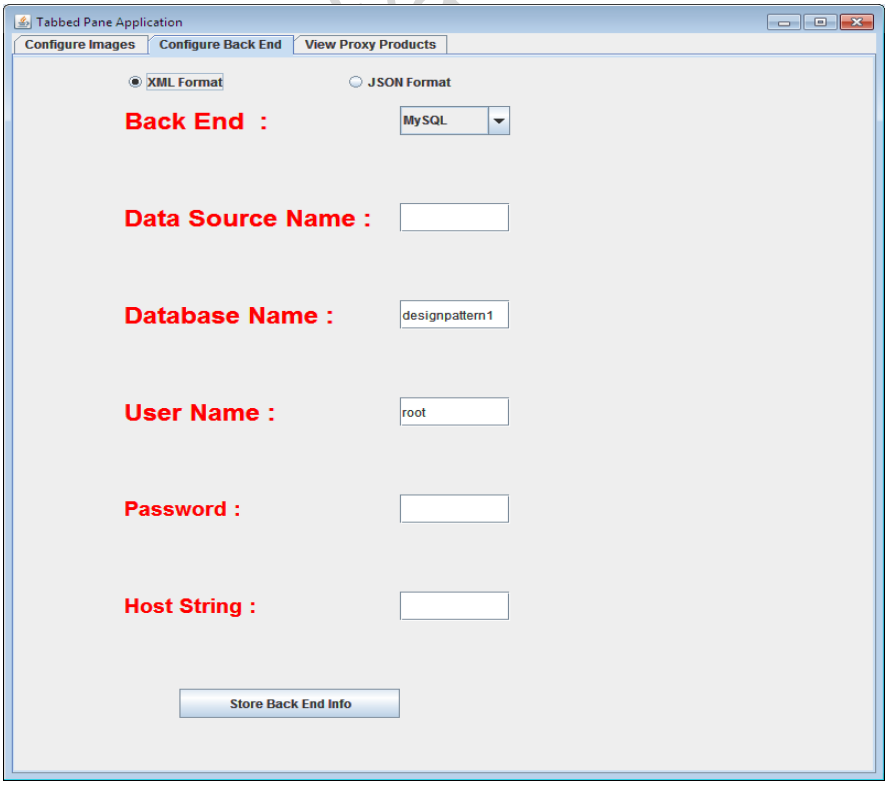
Pre-Condition : Both "**config.xml**" and "**dbconfig.xml**", storing MS-Access connectivity information exist.

Desired Output : Based on image configuration information stored in "config.json" file, count no. of proxy products should be displayed in a single row by reading the relevant backend information stored in "dbconfig.json" file.

Test Case 21: Execute the SQL script given below:

```
create database designpattern1;
use designpattern1;
create table Product(id int, filename char(50),model char(50), price float);

insert into product values(1,"laptop1.jpg","Dell1",10000.0);
insert into product values(2,"laptop2.jpg","Dell2",20000.0);
insert into product values(3,"laptop3.jpg","Dell3",30000.0);
insert into product values(4,"laptop4.jpg","Dell4",40000.0);
insert into product values(5,"laptop5.jpg","Dell5",50000.0);
insert into product values(6,"laptop6.jpg","Dell6",60000.0);
insert into product values(7,"laptop7.jpg","Dell7",70000.0);
insert into product values(8,"laptop8.jpg","Dell6",80000.0);
insert into product values(9,"laptop9.jpg","Dell7",90000.0);
```

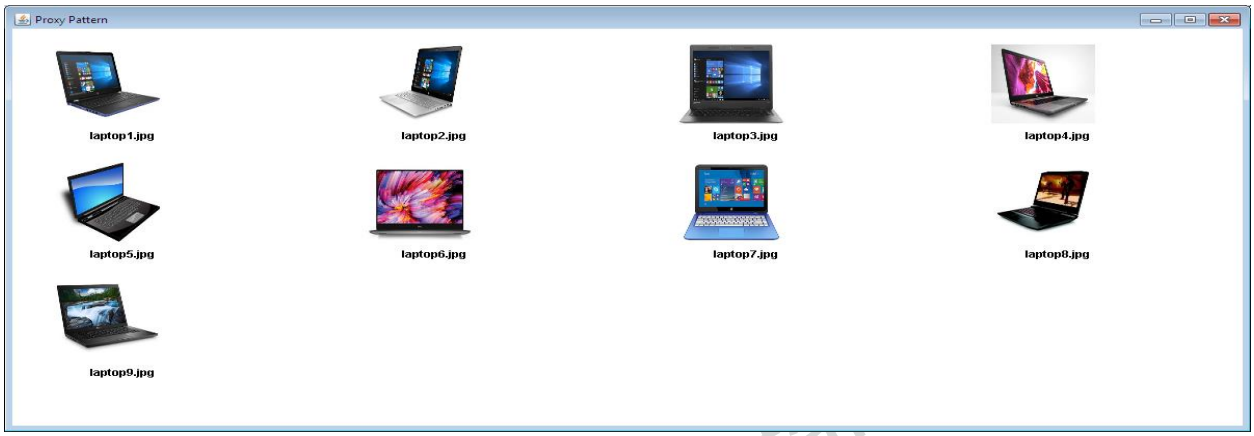

Click on "Configure Back End" tab and enter the back end information as shown below:



Click , on "**View Proxy Products**" tab and select "**XML Format**" option button.

Pre-Condition : Both "**config.xml**" and "**dbconfig.xml**", storing MySQL connectivity information exist.

Desired Output : Based on image configuration information stored in "config.xml" file, count no. of proxy products should be displayed in a single row by reading the relevant backend information stored in "dbconfig.xml" file.



Test Case 22: Click on any laptop.

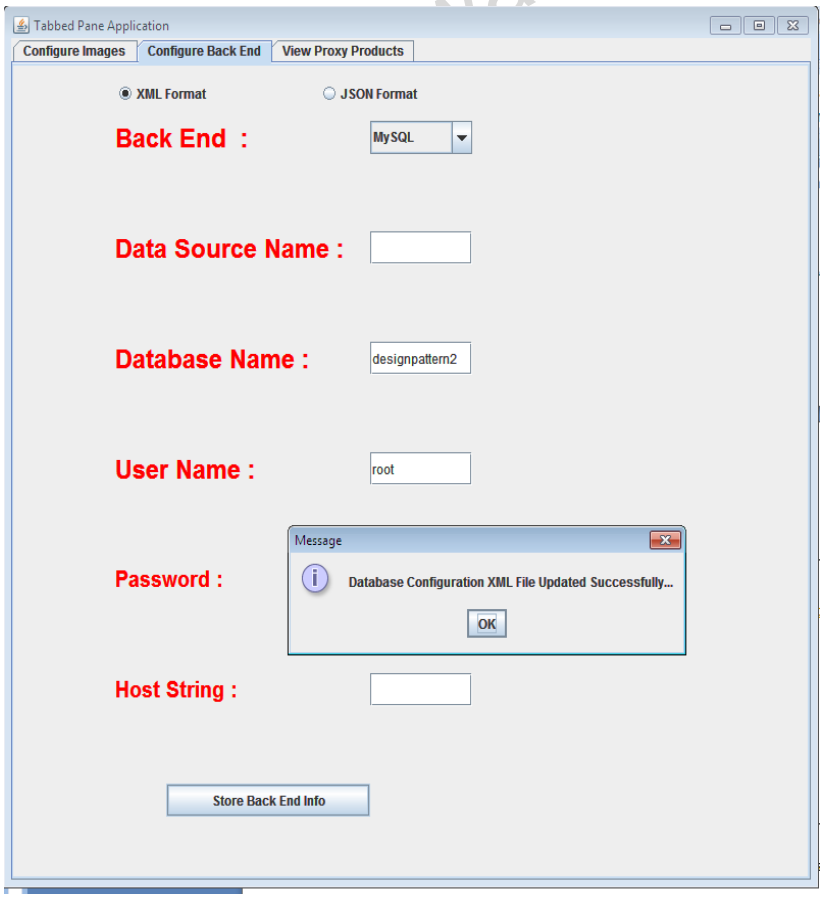
Desired Output : The real product based on the dimension stored in config.xml file should be displayed along with detailed information.

Actual Output:

Test Case 23: Execute the SQL script given below:

```
create database designpattern2;
use designpattern2;
create table Product(id int, filename char(50),model char(50), price float);

insert into product values(1,"pendrive1.jpg","SanDisk1",100.0);
insert into product values(2,"pendrive2.jpg","SanDisk2",200.0);
insert into product values(3,"pendrive3.jpg","SanDisk3",300.0);
insert into product values(4,"pendrive4.jpg","SanDisk4",400.0);
insert into product values(5,"pendrive5.jpg","SanDisk5",500.0);
insert into product values(6,"pendrive6.jpg","SanDisk6",600.0);
insert into product values(7,"pendrive7.jpg","SanDisk7",700.0);
insert into product values(8,"pendrive8.jpg","SanDisk6",800.0);
insert into product values(9,"pendrive9.jpg","SanDisk7",900.0);
```

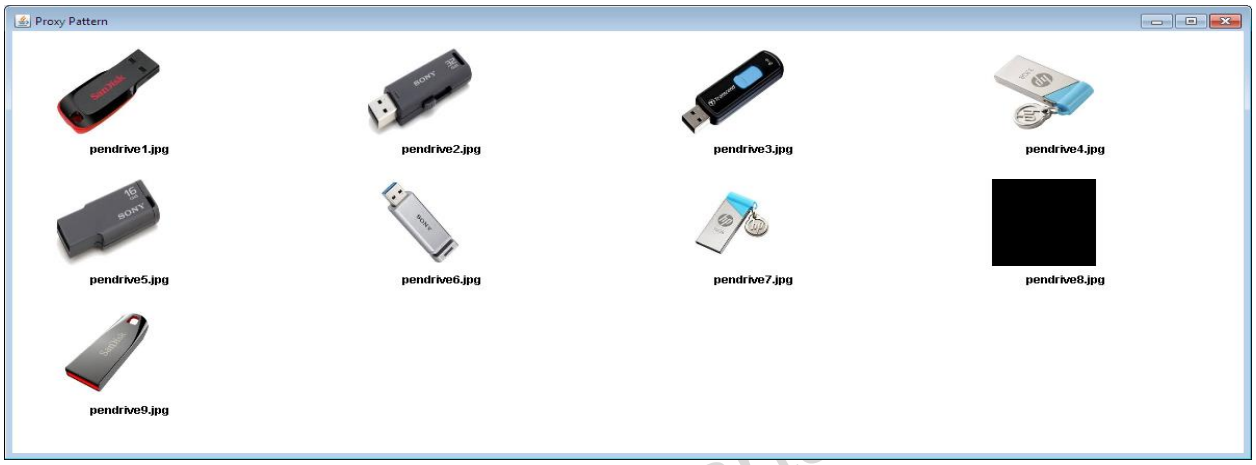
Click on “Configure Back End” tab and enter the back end information as shown below:



Click , on "**View Proxy Products**" tab and select "**XML Format**" option button.

Pre-Condition : Both "**config.xml**" and "**dbconfig.xml**", storing MySQL connectivity information exist.

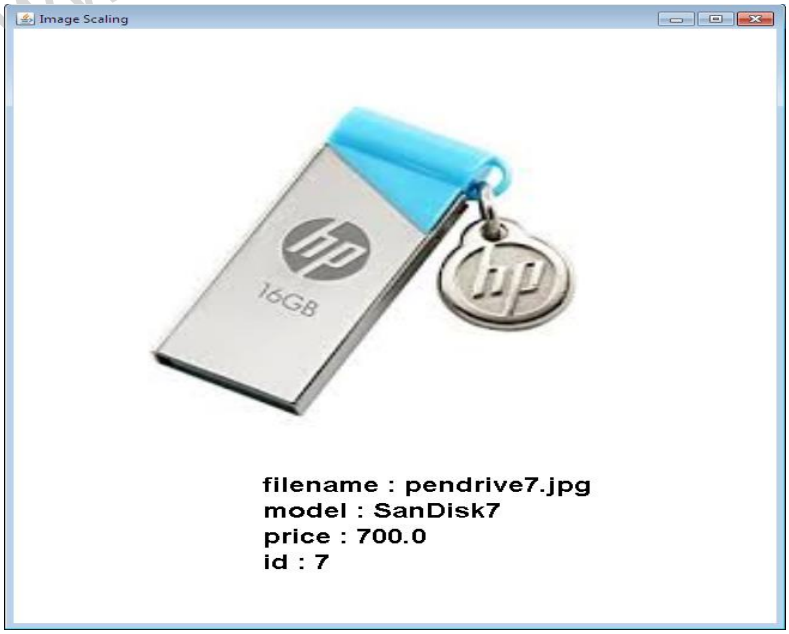
Desired Output : Based on image configuration information stored in "config.xml" file, count no. of proxy products should be displayed in a single row by reading the relevant backend information stored in "dbconfig.xml" file.



Test Case 24: Click on any pen drive.

Desired Output : The real product based on the dimension stored in config.xml file should be displayed along with detailed information.

Actual Output:



References :

1. http://www.geeksforgeeks.org/software-design-patterns/
2. https://dzone.com/refcardz/design-patterns
3. http://www.oodesign.com/
4. https://en.wikipedia.org/wiki/Software_design_pattern
5. https://en.wikipedia.org/wiki/Design_Patterns
6. https://www.tutorialspoint.com/design_pattern
7. https://www.journaldev.com/1827/java-design-patterns-example-tutorial
8. https://sourcemaking.com/design_patterns
9. https://www.javatpoint.com/design-patterns-in-java
10. http://www.vogella.com/tutorials/DesignPatterns/article.html
11. https://sourcemaking.com/design_patterns