Lab 6

Jiho Kim, Garth Slaney

ENSF 480 | Principles of Software Design

Submission Date: October 29, 2020

University of Calgary

ExA

```
import java.util.ArrayList;
/**
 * A subject with a array list of doubles
 * @author Garth-Work
 *
 */
public class DoubleArrayListSubject implements Subject {
	/**
	 * The data to be stored
	 */
	private ArrayList<Double> data;
	/**
	 * All observers tracking this data
	 */
	private ArrayList<Observer> observers;

	public DoubleArrayListSubject() {
		data = new ArrayList<Double>();
		observers = new ArrayList<Observer>();
	}
	/**
	 * Add an entry to the list and notify all obervers
	 * @param d Double to be added
	 */
	public void addData(double d) {
		data.add(d);
		notifyAllObervers();
	}
	/**
	 * Change a list entry and notify all observers
	 * @param d Double to be added
	 * @param postion Postion to change
	 */
	public void setData(double d, int postion) {
		data.set(postion, d);
		notifyAllObervers();
	}
	/**
	 * Copy double array to array list
	 * @param arr Array to be populated to array
	 */
	public void populate(double[] arr){
		for(double d : arr) {
			data.add(d);
```

```
                }
                notifyAllObervers();
        }

        @Override
        public void registerObserver(Observer o) {
                observers.add(o);
                o.update(data);
        }

        @Override
        public void remove(Observer o) {
                observers.remove(o);
        }

        @Override
        public void notifyAllObervers() {
                for(Observer o: observers)
                        o.update(data);
        }
        /**
         * Display the list through all observers
         */
        public void display() {
                if(data.size() == 0)
                {
                        System.out.println("Empty List ..");
                }
                else {
                        notifyAllObervers();
                }
        }

}

import java.util.ArrayList;
/**
 * Display data in five rows
 * @author Garth-Work
 *
 */
public class FiveRowsTable_Observer implements Observer{
        /**
         * Subject that is tracked
         */
        private Subject sub;
```

```
	/**
	 * The array of data
	 */
	private ArrayList<Double> arr;

	public FiveRowsTable_Observer(Subject mydata) {
		sub = mydata;
		sub.registerObserver(this);
	}

	@Override
	public void update(ArrayList<Double> arr) {
		this.arr = arr;
		display();
	}
	/**
	 * Display the data in five rows
	 */
	private void display() {
		System.out.println("\nNotification to FiveRowsTable Observer: Data
Changed:");

		ArrayList<ArrayList<Double>> temp = new
ArrayList<ArrayList<Double>>();

		for(int i = 0; i<5; i++) {
			temp.add(new ArrayList<Double>());
		}

		for(int i = 0; i < arr.size(); i++) {
			temp.get(i%5).add(arr.get(i));
		}

		for(int i = 0; i<5; i++) {
			for( double d : temp.get(i)) {
				System.out.print(d + "  ");
			}
			System.out.println();
		}

		System.out.println();
	}

}
```

```
import java.util.ArrayList;
/**
 * The interface for an observer
 * @author Garth-Work
 *
 */
public interface Observer {
	/**
	 * Change the data stored
	 * @param arr Data to be changed to
	 */
	void update(ArrayList<Double> arr);
}

/* ENSF 480 - Lab 2
 * M. Moussavi
 */

public class ObserverPatternController {
	public static void main(String []s) {
		double [] arr = {10, 20, 33, 44, 50, 30, 60, 70, 80, 10, 11, 23, 34, 55};
		System.out.println("Creating object mydata with an empty list -- no
data:");
		DoubleArrayListSubject mydata = new DoubleArrayListSubject();
		System.out.println("Expected to print: Empty List ...");
		mydata.display();
		mydata.populate(arr);
		System.out.println("mydata object is populated with: 10, 20, 33, 44, 50,
30, 60, 70, 80, 10, 11, 23, 34, 55 ");
		System.out.print("Now, creating three observer objects: ht, vt, and hl ");
		System.out.println("\nwhich are immediately notified of existing data with
different views.");
		ThreeColumnTable_Observer ht = new
ThreeColumnTable_Observer(mydata);
		FiveRowsTable_Observer vt = new FiveRowsTable_Observer(mydata);
		OneRow_Observer hl = new OneRow_Observer(mydata);
		System.out.println("\n\nChanging the third value from 33, to 66 -- (All
views must show this change):");
		mydata.setData(66.0, 2);
		System.out.println("\n\nAdding a new value to the end of the list -- (All
views must show this change)");
		mydata.addData(1000.0);
		System.out.println("\n\nNow removing two observers from the list:");

		mydata.remove(ht);
```

```
                mydata.remove(vt);
                System.out.println("Only the remained observer (One Row ), is
notified.");
                mydata.addData(2000.0);
                System.out.println("\n\nNow removing the last observer from the list:");
                mydata.remove(hl);
                System.out.println("\nAdding a new value the end of the list:");
                mydata.addData(3000.0);
                System.out.println("Since there is no observer -- nothing is displayed ...");
                System.out.println("\nNow, creating a new Three-Column observer that
will be notified of existing data:");
                ht = new ThreeColumnTable_Observer(mydata);
        }
}

import java.util.ArrayList;
/**
 * Display the data in one row
 * @author Garth-Work
 *
 */
public class OneRow_Observer implements Observer{
        /**
         * Subject that is tracked
         */
        private Subject sub;
        /**
         * The array of data
         */
        private ArrayList<Double> arr;

        public OneRow_Observer(Subject mydata) {
                sub = mydata;
                sub.registerObserver(this);
        }

        @Override
        public void update(ArrayList<Double> arr) {
                this.arr = arr;
                display();
        }
        /**
         * Display the data in one row
         */
        private void display() {
```

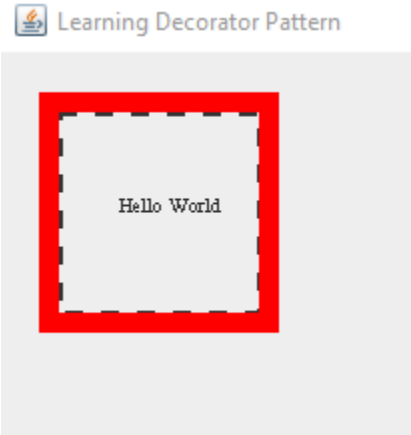
```
                System.out.println("\nNotification to OneRow_Table Observer: Data
Changed:");
                for(double d : arr)
                        System.out.print(d + "  ");
                System.out.println();
        }
}

/**
 * An interface for subject
 * @author Garth-Work
 *
 */
public interface Subject {
        /**
         * Add an observer
         * @param o Observer to be addded
         */
        void registerObserver(Observer o);
        /**
         * Remove an observer
         * @param o Observer to be removed
         */
        void remove(Observer o);
        /**
         * Notify all obervers
         */
        void notifyAllObervers();
}

import java.util.ArrayList;
/**
 * Display the data in three columns
 * @author Garth-Work
 *
 */
public class ThreeColumnTable_Observer implements Observer {
        /**
         * Subject that is tracked
         */
        private Subject sub;
        /**
         * The array of data
         */
        private ArrayList<Double> arr;
```

```
	public ThreeColumnTable_Observer(Subject mydata) {
		sub = mydata;
		sub.registerObserver(this);
	}

	@Override
	public void update(ArrayList<Double> arr) {
		this.arr = arr;
		display();
	}
	/**
	 * Display data in three columns
	 */
	private void display() {
		System.out.println("\nNotification to Three-Column Table Observer: Data
Changed:");
		int i;
		for(i = 0; i < arr.size()-2; i+=3) {
			System.out.println(arr.get(i) + "	" + arr.get(i +1) + "	" +
arr.get(i+2));
		}
		while(i < arr.size()) {
			System.out.print(arr.get(i++) + "	" );
		}
		System.out.println();
	}

}
```

```
$ java ObserverPatternController
Creating object mydata with an empty list -- no data:
Expected to print: Empty List ...
Empty List ..
mydata object is populated with: 10, 20, 33, 44, 50, 30, 60, 70, 80, 10, 11, 23,
 34, 55
Now, creating three observer objects: ht, vt, and hl
which are immediately notified of existing data with different views.

Notification to Three-Column Table Observer: Data Changed:
10.0     20.0     33.0
44.0     50.0     30.0
60.0     70.0     80.0
10.0     11.0     23.0
34.0     55.0

Notification to FiveRowsTable Observer: Data Changed:
10.0  30.0  11.0
20.0  60.0  23.0
33.0  70.0  34.0
44.0  80.0  55.0
50.0  10.0


Notification to OneRow_Table Observer: Data Changed:
10.0  20.0  33.0  44.0  50.0  30.0  60.0  70.0  80.0  10.0  11.0  23.0  34.0  55
.0


Changing the third value from 33, to 66 -- (All views must show this change):

Notification to Three-Column Table Observer: Data Changed:
10.0     20.0     66.0
44.0     50.0     30.0
60.0     70.0     80.0
10.0     11.0     23.0
34.0     55.0

Notification to FiveRowsTable Observer: Data Changed:
10.0  30.0  11.0
20.0  60.0  23.0
66.0  70.0  34.0
44.0  80.0  55.0
50.0  10.0


Notification to OneRow_Table Observer: Data Changed:
10.0  20.0  66.0  44.0  50.0  30.0  60.0  70.0  80.0  10.0  11.0  23.0  34.0  55
.0


Adding a new value to the end of the list -- (All views must show this change)

Notification to Three-Column Table Observer: Data Changed:
10.0     20.0     66.0
44.0     50.0     30.0
60.0     70.0     80.0
10.0     11.0     23.0
34.0     55.0     1000.0


Notification to FiveRowsTable Observer: Data Changed:
10.0  30.0  11.0
20.0  60.0  23.0
66.0  70.0  34.0
44.0  80.0  55.0
50.0  10.0  1000.0
```

```
Notification to FiveRowsTable Observer: Data Changed:
10.0  30.0  11.0
20.0  60.0  23.0
66.0  70.0  34.0
44.0  80.0  55.0
50.0  10.0  1000.0


Notification to OneRow_Table Observer: Data Changed:
10.0  20.0  66.0  44.0  50.0  30.0  60.0  70.0  80.0  10.0  11.0  23.0  34.0  55
.0  1000.0


Now removing two observers from the list:
Only the remained observer (One Row ), is notified.

Notification to OneRow_Table Observer: Data Changed:
10.0  20.0  66.0  44.0  50.0  30.0  60.0  70.0  80.0  10.0  11.0  23.0  34.0  55
.0  1000.0  2000.0


Now removing the last observer from the list:

Adding a new value the end of the list:
Since there is no observer -- nothing is displayed ...

Now, creating a new Three-Column observer that will be notified of existing data
:

Notification to Three-Column Table Observer: Data Changed:
10.0     20.0     66.0
44.0     50.0     30.0
60.0     70.0     80.0
10.0     11.0     23.0
34.0     55.0     1000.0
2000.0     3000.0
```

ExB



ExC

Code for B&C

```
import java.awt.Graphics;

// class used to show text.
public class Text implements Component {
  protected int x;
  protected int y;
  protected String text;

  Text(String message, int a, int b) {
    text = message;
    x = a;
    y = b;
  }

  public void draw(Graphics g) {

    g.drawString(text, x, y);
  }
```

```
}


import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Stroke;
import java.awt.BasicStroke;

// this class is used to display the dashed border
public class BorderDecorator extends Decorator{

  // normal constructor
  BorderDecorator(Component com, int a, int b, int c, int d) {
    super(com, a, b, c, d);
  }

  // this constructor is used when another Decorator is used as a parameter. Displays the
Decorator by setting dec.
  BorderDecorator(Decorator temp, int a, int b, int c, int d) {
    super(temp.cmp, a, b, c, d);
    dec = temp;
  }

  public void draw(Graphics g) {
    if(dec != null) {
      dec.draw(g);
    }

    Stroke dashed = new BasicStroke(3, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_BEVEL, 0, new float[]{9}, 0);
    Graphics2D g2d = (Graphics2D) g;
    g2d.setStroke(dashed);

    g.drawRect(x,y,width,height);
  }
}


import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Stroke;
import java.awt.BasicStroke;
import java.awt.Color;

// used to display thick border that's red.
public class ColouredFrameDecorator extends Decorator{
```

```
  private int thickness;

  // normal constructor
  ColouredFrameDecorator(Component com, int a, int b, int c, int d, int e) {
    super(com, a, b, c, d);
    thickness = e;
  }

  // constructor for when another Decorator has to be shown.
  ColouredFrameDecorator(Decorator temp, int a, int b, int c, int d, int e) {
    super(temp.cmp, a, b, c, d);
    thickness = e;
    dec = temp;
  }

  public void draw(Graphics g) {
    if(dec != null) {
      dec.draw(g);
    }

    Graphics2D g2d = (Graphics2D) g;

    // save previous stroke style
    Stroke oldStroke = g2d.getStroke();
    Color oldColor = g2d.getColor();

    // set new stroke and draw rectangle
    g2d.setStroke(new BasicStroke(thickness));
    g2d.setColor(Color.red);
    g2d.drawRect(x, y, width, height);

    // set stroke back to what it was previously
    g2d.setStroke(oldStroke);
    g2d.setColor(oldColor);
  }
}


import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Color;
import java.awt.AlphaComposite;

// class used to fill rectangle with a transparent background.
public class ColouredGlassDecorator implements Component{
```

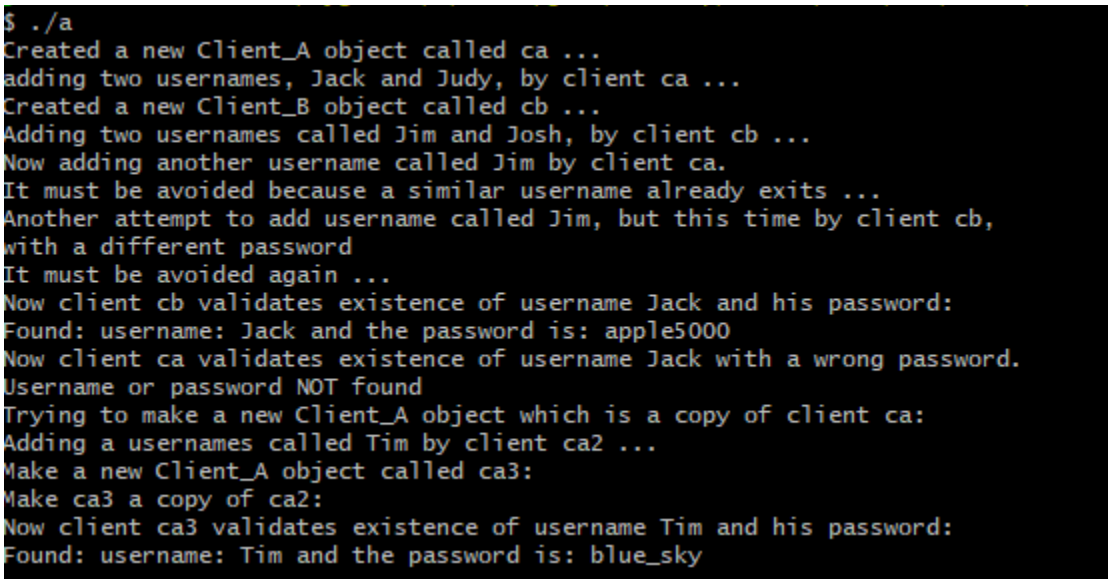
```
  protected ColouredFrameDecorator cfd;
  protected int x;
  protected int y;
  protected int width;
  protected int height;

  ColouredGlassDecorator(ColouredFrameDecorator a, int b, int c, int d, int e) {
    cfd = a;
    x = b;
    y = c;
    width = d;
    height = e;
  }

  public void draw(Graphics g) {
    cfd.draw(g);
    Graphics2D g2d = (Graphics2D) g;
    g2d.setColor(Color.green);
    g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
1*0.1f));
    g2d.fillRect(x, y, width, height);
  }
}


import java.awt.Graphics;

public interface Component {

  public void draw(Graphics g);

}


// abstract class used to implement component. This will be used as the parent class to
many different classes.
public abstract class Decorator implements Component {
  protected Decorator dec = null;
  protected Component cmp;
  protected int x;
  protected int y;
  protected int width;

  public int height;
```

```
  // constructor
 Decorator(Component com, int a, int b, int c, int d) {
   cmp = com;
   x = a;
   y = b;
   width = c;
   height = d;
  }
}


import java.awt.Font;
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class DemoDecoratorPattern extends JPanel {
        Component t;

   public DemoDecoratorPattern(){
         t = new Text ("Hello World", 60, 80);
  }

   public void paintComponent(Graphics g){
                        // set font and show text on jpanel
            int fontSize = 10;
            g.setFont(new Font("Times New Roman", Font.PLAIN, fontSize));
                        t = new Text ("Hello World", 60, 80);
                        t.draw(g);

/*
            // Now lets decorate t with BorderDecorator: x = 30, y = 30, width = 100, and
height 100
            t = new BorderDecorator(t, 30, 30, 100, 100);
                        t.draw(g);
            // Now lets add a ColouredFrameDecorator with x = 25, y = 25, width = 110,
height = 110,
                  // and thickness = 10.
            t = new ColouredFrameDecorator(t, 25, 25, 110, 110, 10);
*/

                  // used to draw the border and fill the rectangle with green colour
                  t = new ColouredGlassDecorator(new ColouredFrameDecorator(
                              new BorderDecorator(t, 30, 30, 100, 100), 25, 25, 110, 110,
10), 25, 25, 110, 110
                              );
```

```
            // Now lets draw the product on the screen
            t.draw(g);
    }

        public static void main(String[] args) {
      DemoDecoratorPattern panel = new DemoDecoratorPattern();
      JFrame frame = new JFrame("Learning Decorator Pattern");
      frame.getContentPane().add(panel);
      frame.setSize(400,400);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setLocationRelativeTo(null);
      frame.setVisible(true);
        }
}
```

ExD

This program does not allow creating objects of LoginServer as the login server constructor is private.



Code

```
#include <string>
#include <iostream>
#include  <vector>
#include "LoginServer.hpp"

LoginServer* LoginServer::instance= nullptr;;

LoginServer* LoginServer::getInstance(){
	if(instance == nullptr){
		instance = new LoginServer();
	}
	return instance;
}

User* LoginServer::validate(string username, string password){
	//Iterate over all users in vector
	for(vector<User>::iterator u = users.begin(); u != users.end(); u++){
		if(u -> username == username && u ->password == password){
			return &(*u); //defrence then get the adress to return only a pointer
to a user
		}
	}
	return nullptr;
```

```
}


//Client_B.hpp
#ifndef Client_B_H // include guard
#define Client_B_H
#include <iostream>
#include <string>
#include "LoginServer.hpp"

class Client_B{
	private:
	LoginServer* instance;

	public:
	Client_B() {
		instance = LoginServer::getInstance();
	}

	void add(string username, string  password){
		instance -> add(username, password);
	}

	User* validate(string username, string password){
		return instance -> validate(username, password);
	}
};

#endif


//Client_A.hpp
#ifndef Client_A_H // include guard
#define Client_A_H
#include <iostream>
#include <string>
#include "LoginServer.hpp"
class Client_A{
	private:
	LoginServer* instance;

	public:
	Client_A() {
		instance = LoginServer::getInstance();
	}
```

```
        void add(string username, string  password){
                instance -> add(username, password);
        }

        User* validate(string username, string password){
                return instance -> validate(username, password);
        }
};
#endif


//
//  main.cpp
//  SigletonPattern
//
#include "Client_A.hpp"
#include "Client_B.hpp"
#include "LoginServer.hpp"
#include <iostream>
using namespace std;

int main() {

   Client_A ca;
   cout << "Created a new Client_A object called ca ..." << endl;

   cout << "adding two usernames, Jack and Judy, by client ca ..." << endl;
   ca.add("Jack", "apple5000");
   ca.add("Judy", "orange$1234");

   Client_B cb;
   cout << "Created a new Client_B object called cb ... " << endl;
   cout << "Adding two usernames called Jim and Josh, by client cb ..." << endl;
   cb.add("Jim", "brooks$2017");
   cb.add("Josh", "mypass2000");

   cout << "Now adding another username called Jim by client ca.\n";
   cout << "It must be avoided because a similar username already exits ..." << endl;
   ca.add("Jim", "brooks$2017");
   cout << "Another attempt to add username called Jim, but this time by client cb,\n";
   cout << "with a different password\n";
   cout << "It must be avoided again ..." << endl;
   cb.add("Jim", "br$2017");

   cout << "Now client cb validates existence of username Jack and his password: " <<
endl;
```

```
   if( User *u = cb.validate("Jack", "apple5000"))
      cout << "Found: username: " << u->username << " and the password is: " << u-
>password <<  endl;
   else
      cout << "Username or password NOT found" << endl;
   cout << "Now client ca validates existence of username Jack with a wrong password. "
<< endl;
   if( User *u = ca.validate("Jack", "apple4000"))
      cout << "Found: username is: " << u->username << " and password is: " << u-
>password <<  endl;
   else
      cout << "Username or password NOT found" << endl;

   cout << "Trying to make a new Client_A object which is a copy of client ca:" << endl;
   Client_A ca2 = ca;
   cout << "Adding a usernames called Tim by client ca2 ..." << endl;
   cb.add("Tim", "blue_sky");
   cout << "Make a new Client_A object called ca3:" << endl;
   Client_A ca3;
   cout << "Make ca3 a copy of ca2:" << endl;
   ca3 = ca2;
   cout << "Now client ca3 validates existence of username Tim and his password: " <<
endl;
   if( User *u = ca3.validate("Tim", "blue_sky"))
      cout << "Found: username: " << u->username << " and the password is: " << u-
>password <<  endl;
   else
      cout << " Tim NOT found" << endl;
#if 1
   cout << "Lets now make a couple of objects of LoginServer by main funciton:" <<
endl;
   LoginServer x;
   LoginServer y = x;
   x = y;
   cout << "Now LoginServer x validates existence of username Tim and his password: "
<< endl;
   if( User *u = y.validate("Tim", "blue_sky"))
      cout << "Found: username: " << u->username << " and the password is: " << u-
>password <<  endl;
   else
      cout << "Tim NOT found" << endl;
#endif

   return 0;
}
```

```
#ifndef LoginServer_H // include guard
#define LoginServer_H
#include <string>
#include <iostream>
#include  <vector>

using namespace std;
struct User{
	string username;
	string password;
};

class LoginServer{
	private:
	vector<User> users;
	static LoginServer* instance;
	LoginServer(){}

	public:
	LoginServer(const LoginServer& src) = delete;

	void operator=(const LoginServer& rhs) = delete;

	static LoginServer* getInstance();

	void add(string username, string password){
		users.push_back({username, password});
	}

	User* validate(string username, string password);
};
#endif
```