Course: ENSF 614 – Fall 2021

Lab 7

Student Names:

Kody YingCheng Kou & Zach Frena

Submission Date: Nov 23rd, 2021

Exercise A and B:

DemoDecoratorPattern.java

****The source code for exercises A and B are identical except for portions in the DemoDecoratorPattern.java file (commented out for relevant exercise)****

```java
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){

       int fontSize = 10;
        //portion for Exercise A
       /*
       g.setFont(new Font("TimesRoman", Font.PLAIN, fontSize));

       // Now lets decorate t with BorderDecorator: x = 30, y = 30, width = 100, and height 100
       t = new BorderDecorator(t, 30, 30, 100, 100);

       // Now lets add a ColouredFrameDecorator with x = 25, y = 25, width = 110, height = 110,
      // and thickness = 10.
       t = new ColouredFrameDecorator(t, 25, 25, 111, 111, 10);

       // Now lets draw the product on the screen

        t = new ColouredGlassDecorator(t, 25, 25, 110, 110);
       t.draw(g);
        */

         //portion for exercise B:
        g.setFont(new Font("TimesRoman", Font.PLAIN, fontSize));// GlassFrameDecoratorinfo: x = 25, y = 25,
width = 110, and height = 110
        t=  new ColouredGlassDecorator(
                new ColouredFrameDecorator(
                        new BorderDecorator(
                                t, 30, 30, 100, 100), 25, 25, 110, 110, 10), 25, 25,110, 110);
        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);
        Graphics g = panel.getGraphics();
        panel.paintComponent(g);


        frame.setVisible(true);


   }
}
```

Text.java

```java
import java.awt.*;


public class Text implements Component{

    int x;
    int y;
    String text;

    public Text(String text, int x, int y) {
        this.text = text;
        this.x = x;
        this.y =y ;
    }

    public void draw(Graphics g){
        g.setColor(new Color(0,100,0));
        g.drawString(text, x, y);

    }
}
```

Decorator.java

```java
public abstract class Decorator implements Component{
    Component cmp;
    int x;
    int y;
    int width;
    public int height;

    public Decorator(Component cmp, int x, int y, int width, int height){
        this. cmp = cmp;
        this.x = x;
        this.y = y;
        this.width=width;
        this.height = height;
    }


}
```

Component.java

```java
import java.awt.*;

public interface Component {
    public void draw(Graphics g);
}
```

BorderDecorator.java

```java
import java.awt.*;

public class BorderDecorator extends Decorator{

    public BorderDecorator(Component cmp, int x, int y, int width, int height) {
        super(cmp, x, y, width, height);
    }

    public void draw(Graphics g){
        cmp.draw(g);

        Graphics2D g2 = (Graphics2D) g;
        Stroke oldStroke = g2.getStroke();
        Stroke dashes = new BasicStroke(2,
                BasicStroke.CAP_BUTT,
                BasicStroke.JOIN_BEVEL,
                0,
                new float[]{4},
                0);
        g2.setStroke(dashes);
        g2.drawLine(x, y, x+width, y);
        g2.drawLine(x, y, x, y+height);
        g2.drawLine(x+width, y, x+width, y+height);
        g2.drawLine(x, y+height, x+width, y+height);

        g2.setStroke(oldStroke);





    }
}
```

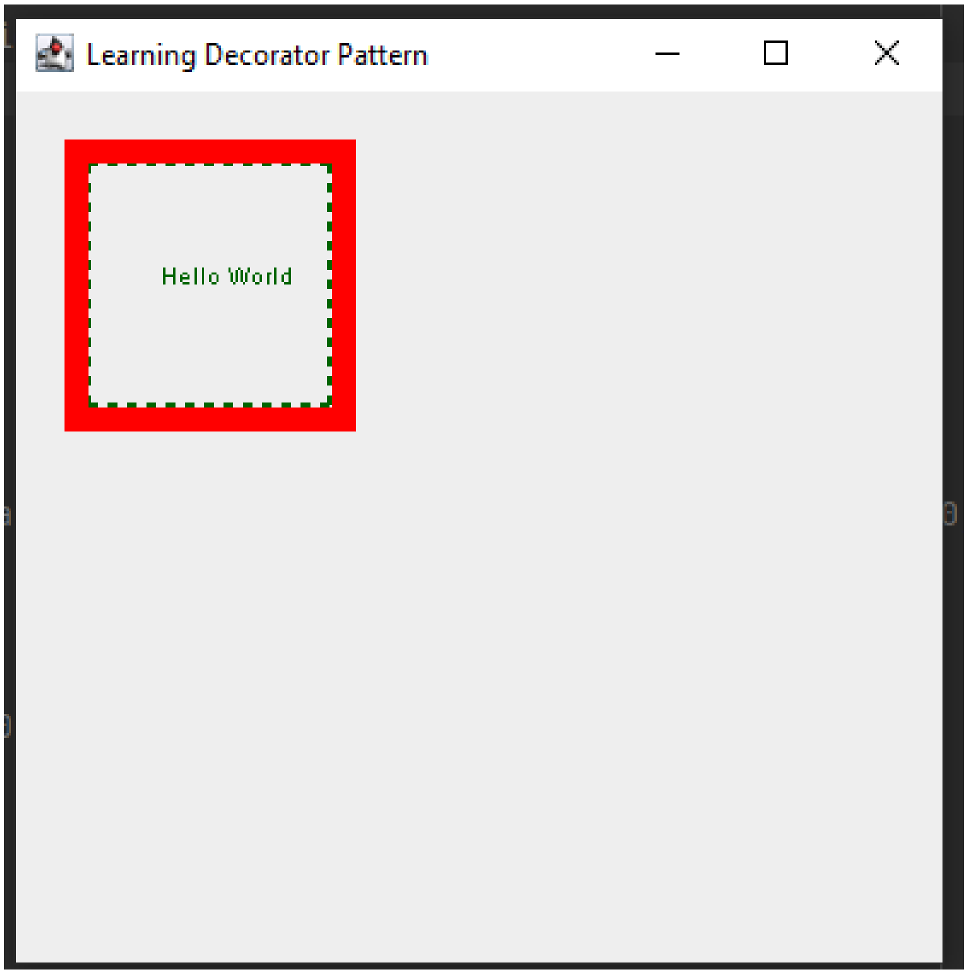
ColouredFrameDecorator.java

```java
import java.awt.*;

public class ColouredFrameDecorator extends Decorator{
    int thickness;

    public ColouredFrameDecorator(Component cmp, int x, int y, int width, int height, int thickness) {
        super(cmp, x, y, width, height);
        this.thickness = thickness;
    }

    public void draw(Graphics g){
        cmp.draw(g);
        Graphics2D g2 = (Graphics2D) g;
        Stroke oldStroke = g2.getStroke();
        Color oldColor = g2.getColor();
        g2.setStroke(new BasicStroke(thickness));
        g2.setColor(Color.red);
        g2.drawRect(x, y, width, height);
        g2.setStroke(oldStroke);
        g2.setColor(oldColor);

    }
}
```

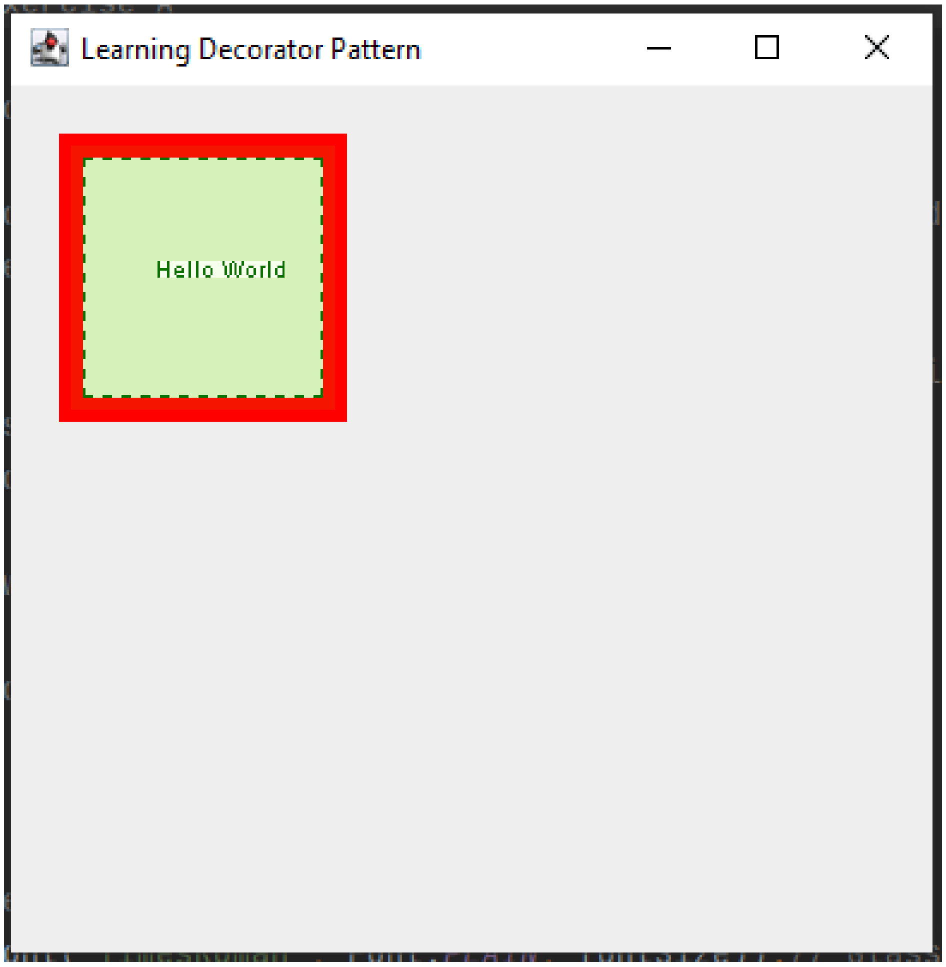
ColouredGlassDectorator.java

```
import java.awt.*;

public class ColouredGlassDecorator extends Decorator{


    public ColouredGlassDecorator(Component cmp, int x, int y, int width, int height){
        super(cmp, x, y, width, height);
    }


    public void draw(Graphics g){
        cmp.draw(g);
        Color c = new Color(124, 252, 0, 20);
        g.setColor(c);
        g.fillRect(x, y, width, height);


    }
}
```

Program Output Exercise A:



Program Output Exercise B:



Hello World

Exercise C: Developing Singleton Pattern in C++

Main.cpp

```
//
//  main.cpp
//  SigletonPattern
//
#include "Client_A.hpp"
#include "Client_B.hpp"
#include "LoginServer.hpp"
#include "User.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 0
    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;
}
```

User.cpp

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
```

User.hpp

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;


#ifndef User_HPP
#define User_HPP

typedef struct User{
public:
string username;
string password;
}User;
#endif
```

Client_A.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include "LoginServer.hpp"
#include "Client_A.hpp"
#include "User.hpp"
using namespace std;

void Client_A::add(string username, string password){
    instance = LoginServer::getInstance();
    (*instance).add(username, password);
}

User* Client_A::validate(string username, string password){
    instance = LoginServer::getInstance();
    return (*instance).validate(username,password);
}
```

Client_A.hpp

```
#include <iostream>
#include <string>
#include <vector>
#include "LoginServer.hpp"
using namespace std;

#ifndef Client_A_HPP
#define Client_A_HPP

class Client_A{
public:
void add(string username, string password);
User* validate(string username, string password);

private:
LoginServer* instance;

};


// LoginServer* LoginServer::instance = nullptr;
#endif
```

Client_B.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>
#include "LoginServer.hpp"
#include "Client_B.hpp"
using namespace std;

void Client_B::add(string username, string password){
    instance = LoginServer::getInstance();
    (*instance).add(username, password);
}

User* Client_B::validate(string username, string password){
    instance = LoginServer::getInstance();
    return (*instance).validate(username, password);
}
```

Client_B.hpp

```cpp
#include <iostream>
#include <string>
#include <vector>
#include "LoginServer.hpp"
using namespace std;


#ifndef Client_B_HPP
#define Client_B_HPP

class Client_B{
public:
void add(string username, string password);
User* validate(string username, string password);

private:
LoginServer* instance;
};

// LoginServer* LoginServer::instance = nullptr;

#endif
```

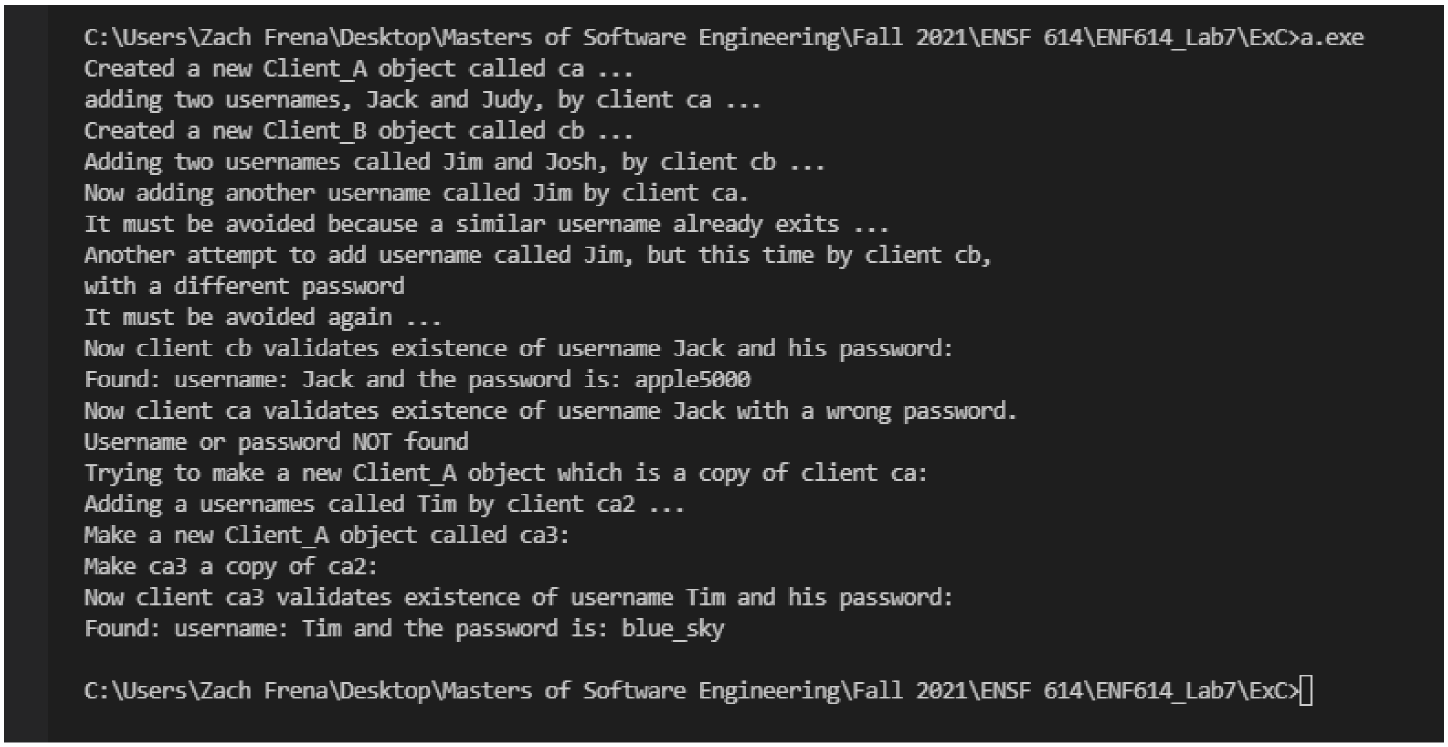
LoginServer.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>
#include "user.hpp"
#include "LoginServer.hpp"
using namespace std;

LoginServer::LoginServer(){

}

LoginServer::LoginServer(const LoginServer& src){ //copy constructor?
}

LoginServer& LoginServer::operator =(const LoginServer& rhs){

    return *this;
}


void LoginServer::add(string username, string password){
    for(int i = 0; i <users.size(); i++){
        if(username.compare(users.at(i).username) == 0){
            return;
        }
    }
    users.push_back({username, password});
}

User* LoginServer::validate (string username, string password){
    for(int i = 0; i<users.size(); i++){
        if(username.compare(users.at(i).username)==0 &&
        password.compare(users.at(i).password)==0){
            return &users.at(i);
        }
    }
    return nullptr;
}


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

LoginServer* LoginServer::instance = nullptr;
```

LoginServer.hpp

```cpp
#include <iostream>
#include <string>
#include <vector>
#include "user.hpp"
using namespace std;



#ifndef LoginServer_HPP
#define LoginServer_HPP

class LoginServer{
public:
    void add(string username, string password);
    User* validate (string username, string password);
    static LoginServer* getInstance();

private:
LoginServer();
LoginServer(const LoginServer& src);
LoginServer& operator =(const LoginServer& rhs);

protected:
vector<User> users = vector<User>();
static LoginServer* instance;

};
#endif
```

Program Output:



```
C:\Users\Zach Frena\Desktop\Masters of Software Engineering\Fall 2021\ENSF 614\ENF614_Lab7\ExC>a.exe
Created a new Client_A object called ca ...
adding two usernames, Jack and Judy, by client ca ...
Created a new Client_B object called cb ...
Adding two usernames called Jim and Josh, by client cb ...
Now adding another username called Jim by client ca.
It must be avoided because a similar username already exits ...
Another attempt to add username called Jim, but this time by client cb,
with a different password
It must be avoided again ...
Now client cb validates existence of username Jack and his password:
Found: username: Jack and the password is: apple5000
Now client ca validates existence of username Jack with a wrong password.
Username or password NOT found
Trying to make a new Client_A object which is a copy of client ca:
Adding a usernames called Tim by client ca2 ...
Make a new Client_A object called ca3:
Make ca3 a copy of ca2:
Now client ca3 validates existence of username Tim and his password:
Found: username: Tim and the password is: blue_sky

C:\Users\Zach Frena\Desktop\Masters of Software Engineering\Fall 2021\ENSF 614\ENF614_Lab7\ExC>
```

<u>Exercise D part 2:</u>

1) No, you can not create objects of LoginServer.

2) After changing the #if 0 to #if 1 in the second segment of main.cpp, the program does not allow creating objects of LoginServer and results in a compilation error. This is due to LoginServer adapting the Singleton design pattern and only allowing one instance of LoginServer, the constructor of LoginServer is private and can not be called unless there is no instances of LoginServer, then it would be called by getInstance.