Lab 7 Report

Course: ENSF 619 - Fall 2020
Lab #: Lab 7
Student Name: Davis Allan, 10016543
Submission Date: Nov 6 2020

Exercise A

Source code

Observer.java

```
package ExerciseA;
/*
 * File Name: Observer.java
 * Lab # and Assignment #: Lab 7 Exercise A
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
import java.util.ArrayList;

/**
 * Interface Name: Observer
 *
 * Provides the method definition necessary for an Observer
 */
public interface Observer {
    void update(ArrayList<Double> arr);
}
```

Subject.java

```
package ExerciseA;
/*
 * File Name: Subject.java
 * Lab # and Assignment #: Lab 7 Exercise A
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */

/**
 * Interface Name: Observer
 *
 * Provides the method definitions necessary for a Subject
 */
public interface Subject {
    void registerObserver(Observer o);
    void removeObserver(Observer o);
```

```
    void notifyAllObservers();
}
```

ObserverPatternController.java

```
package ExerciseA;
/*
 * File Name: ObserverPatternController.java
 * Lab # and Assignment #: Lab 7 Exercise A
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */

/**
 * Class Name: ObserverPatternController
 *
 * Provides the driver code to test the implementation of the Observer
design pattern
 */
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.removeObserver(ht);
        mydata.removeObserver(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.removeObserver(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.print("\nNow, creating a new Three-Column observer that
will be notified of existing data:");
        ht = new ThreeColumnTable_Observer(mydata);
    }
}
```

FiveRowsTable_Observer.java

```
package ExerciseA;
/*
 * File Name: FiveRowsTable_Observer.java
 * Lab # and Assignment #: Lab 7 Exercise A
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
import java.util.ArrayList;

/**
 * Class Name: FiveRowsTable_Observer
 *
 * Provides the data fields and methods to create an Observer that will
display the data
 * in 5 rows, with as many columns as needed anytime an update occurs.
 */
```

```java
public class FiveRowsTable_Observer implements Observer {
    /**
     * ArrayList containing the data
     */
    private ArrayList<Double> data;

    /**
     * The Subject that this Observer will observe
     */
    private Subject subject;

    /**
     * Constructs a FiveRowsTable_Observer and registers the Subject that
it will be observing
     * @param subject the Subject to be observed
     */
    public FiveRowsTable_Observer(Subject subject) {
        this.subject = subject;
        subject.registerObserver(this);
    }

    /**
     * Updates the observed data and then calls the display method
     * @param arr the updated data to be displayed
     */
    @Override
    public void update(ArrayList<Double> arr) {
        data = arr;
        display();
    }

    /**
     * Displays the values of the data in 5 rows, with as many columns as
needed
     */
    public void display() {
        System.out.println("\nNotification to Five-Row Table
ExerciseA.Observer: Data Changed:");
        int cols = (int) Math.round(data.size() / 5.0);

        for (int i = 0; i < 5; i++) {
            int index = i;
            for (int j = 0; j < cols; j++) {
```

```
                if (index >= data.size())
                    break;
                System.out.print(data.get(index) + " ");
                index += 5;
            }
            System.out.print("\n");
        }
    }
}
```

ThreeColumnTable_Observer.java

```
package ExerciseA;
/*
 * File Name: ThreeColumnTable_Observer.java
 * Lab # and Assignment #: Lab 7 Exercise A
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
import java.util.ArrayList;

/**
 * Class Name: ThreeColumnTable_Observer
 *
 * Provides the data fields and methods to create an Observer that will display the data
 * in 3 columns, with as many rows as needed anytime an update occurs.
 */
public class ThreeColumnTable_Observer implements Observer {
    /**
     * ArrayList containing the data
     */
    private ArrayList<Double> data;

    /**
     * The Subject that this ExerciseA.Observer will observe
     */
    private Subject subject;

    /**
     * Constructs a ThreeColumnTable_Observer and registers the Subject that it will be observing
```

```
     * @param subject the Subject to be observed
     */
    public ThreeColumnTable_Observer(Subject subject) {
        this.subject = subject;
        subject.registerObserver(this);
    }

    /**
     * Updates the observed data and then calls the display method
     * @param arr the updated data to be displayed
     */
    @Override
    public void update(ArrayList<Double> arr) {
        data = arr;
        display();
    }

    /**
     * Displays the values of the data in 5 rows, with as many columns as
needed
     */
    public void display() {
        System.out.println("\nNotification to Three-Column Table
ExerciseA.Observer: Data Changed:");
        int rows = (int) Math.round(data.size() / 3.0);
        int index = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < 3; j++) {
                if (index >= data.size())
                    break;
                System.out.print(data.get(index) + " ");
                index++;
            }
            System.out.print("\n");
        }
    }
}
```

OneRow_Observer.java

```
package ExerciseA;
/*
 * File Name: OneRow_Observer.java
```

```java
 * Lab # and Assignment #: Lab 7 Exercise A
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
import java.util.ArrayList;

public class OneRow_Observer implements Observer {
    /**
     * ArrayList containing the data
     */
    private ArrayList<Double> data;

    /**
     * The Subject that this Observer will observe
     */
    private Subject subject;

    /**
     * Constructs a OneRow_Observer and registers the Subject that it will be observing
     * @param subject the Subject to be observed
     */
    public OneRow_Observer(Subject subject) {
        this.subject = subject;
        subject.registerObserver(this);
    }

    /**
     * Updates the observed data and then calls the display method
     * @param arr the updated data to be displayed
     */
    @Override
    public void update(ArrayList<Double> arr) {
        data = arr;
        display();
    }

    /**
     * Displays the values of the data in 5 rows, with as many columns as needed
     */
    public void display() {
```

```
        System.out.println("\nNotification to One-Row ExerciseA.Observer:
Data Changed:");
        for (Double d : data) {
            System.out.print(d + " ");
        }
    }
}
```

DoubleArrayListSubject.java

```
package ExerciseA;
/*
 * File Name: ExerciseA.DoubleArrayListSubject.java
 * Lab # and Assignment #: Lab 7 Exercise A
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
import java.util.ArrayList;

/**
 * Class Name: DoubleArrayListSubject
 *
 * Represents a Subject in the Observer design pattern that holds an
ArrayList of data to be displayed
 * by the various Observers
 */
public class DoubleArrayListSubject implements Subject{

    /**
     * ArrayList of Observers watching this Subject
     */
    private ArrayList<Observer> observers;

    /**
     * ArrayList containing the data
     */
    public ArrayList<Double> data;

    /**
     * Constructs the DoubleArrayListSubject object and initializes its
data members to empty
     * lists
```

```
     */
    public DoubleArrayListSubject() {
        data = new ArrayList<Double>();
        observers = new ArrayList<Observer>();
    }

    /**
     * Adds the data to the list and notifies all Observers of the change
     * @param value number to be added
     */
    public void addData(Double value) {
        data.add(value);
        notifyAllObservers();
    }

    /**
     * Sets the data at a specified index to the new specified value and
notifies
     * all Observers of the change
     * @param value the new value for the data
     * @param index index at which the update will occur
     */
    public void setData(Double value, int index) {
        if (index >= data.size()) { //ensures that the index is valid
            System.out.println("Index out of bounds, unable to add data");
            return;
        }
        data.set(index, value);
        notifyAllObservers();
    }

    /**
     * Populates the list with the data supplied by its argument
     * @param arr Array containing double values
     */
    public void populate(double[] arr) {
        for (double d : arr) {
            data.add(d);
        }
    }

    /**
     * Displays the values of the data to the console, outputs a message
```

```
if the list is empty,
     * otherwise notifies all Observers to display the contents of the
data
     */
    public void display() {
        if (data.size() == 0) {
            System.out.println("Empty List...");
            return;
        }
        notifyAllObservers();
    }

    /**
     * Adds the Observer to the observer list and updates the Observer
with the current data
     * @param o the Observer to be registered
     */
    @Override
    public void registerObserver(Observer o) {
        observers.add(o);
        o.update(data);
    }

    /**
     * Removes the Observer from the observer list
     * @param o the Observer to be removed
     */
    @Override
    public void removeObserver(Observer o) {
        observers.remove(o);
    }

    /**
     * Updates all Observers in the observer list with the current data
     */
    @Override
    public void notifyAllObservers() {
        for (Observer o : observers) {
            o.update(data);
        }
    }
}
```

Output

```
ExerciseA.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 ExerciseA.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 Five-Row Table ExerciseA.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 One-Row ExerciseA.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 ExerciseA.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 Five-Row Table ExerciseA.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 One-Row ExerciseA.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 ExerciseA.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 Five-Row Table ExerciseA.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 One-Row ExerciseA.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 One-Row ExerciseA.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 ExerciseA.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

Process finished with exit code 0
```

Exercise B and C

Source Code

BorderDecorator.java

```
package ExerciseBandC;
/*
 * File Name: BorderDecorator.java
 * Lab # and Assignment #: Lab 7 Exercise B and C
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
import java.awt.*;

/**
 * Class Name: BorderDecorator
 *
 * Adds a dashed line border in the shape of a rectangle to the Component
 */
public class BorderDecorator extends Decorator {

    /**
     * Constructs a BorderDecorator object and initializes all data
members from the
     * specified arguments
     * @param c Component to be wrapped
     * @param x x coordinate
     * @param y y coordinate
     * @param width width of the rectangle
     * @param height height of the rectangle
     */
    public BorderDecorator(Component c, int x, int y, int width, int
height) {
        super(c);
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

    /**
```

```
     * Displays any previous components and draws the dashed line
rectangle on the GUI
     * @param g Graphics to be drawn
     */
    @Override
    public void draw(Graphics g) {
        cmp.draw(g);
        Graphics2D g2d = (Graphics2D) g;
        Stroke oldStroke = g2d.getStroke();
        Stroke dashed = new BasicStroke(3, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_BEVEL, 0, new
                float[]{9}, 0);

        g2d.setStroke(dashed);
        g2d.drawRect(x, y, width, height);
        g2d.setStroke(oldStroke);
    }
}
```

ColouredFrameDecorator.java

```
package ExerciseBandC;
/*
 * File Name: ColouredFrameDecorator.java
 * Lab # and Assignment #: Lab 7 Exercise B and C
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
import java.awt.*;

/**
 * Class Name: ColouredFrameDecorator
 *
 * Adds a colored rectangle to the Component
 */
public class ColouredFrameDecorator extends Decorator {
    /**
     * thickness of the border
     */
    private int thickness;

    /**
```

```
     * Constructs the ColouredFrameDecorator object and initializes its
data members from
     * the specified arguments
     * @param c Component to be wrapped
     * @param x x coordinate
     * @param y y coordinate
     * @param width width of the rectangle
     * @param height height of the rectangle
     * @param thickness thickness of the rectangle line
     */
    public ColouredFrameDecorator(Component c, int x, int y, int width,
int height, int thickness) {
        super(c);
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.thickness = thickness;
    }

    /**
     * Displays any previous components and draws the coloured rectangle
on the GUI
     * @param g Graphics to be drawn
     */
    @Override
    public void draw(Graphics g) {
        cmp.draw(g);
        Graphics2D g2d = (Graphics2D) g;
        Stroke oldStroke = g2d.getStroke();
        Color oldColor = g2d.getColor();
        g2d.setStroke(new BasicStroke(thickness));
        g2d.setColor(Color.red);
        g2d.drawRect(x, y, width, height);
        g2d.setStroke(oldStroke);
        g2d.setColor(oldColor);
    }
}
```

ColouredGlassDecorator.java

```
package ExerciseBandC;
/*
```

```java
 * File Name: ColouredGlassDecorator.java
 * Lab # and Assignment #: Lab 7 Exercise B and C
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
import java.awt.*;

/**
 * Class Name: ColouredGlassDecorator
 *
 * Provides the methods to draw a colored semi-transparent glass frame at the specified location
 */
public class ColouredGlassDecorator extends Decorator {

    /**
     * Constructs a ColouredGlassDecorator object and initializes all its data members
     * from the provided arguments
     * @param c Component that is being wrapped
     * @param x the x coordinate
     * @param y the y coordinate
     * @param width the width of the rectangle
     * @param height the height of the rectangle
     */
    public ColouredGlassDecorator(Component c, int x, int y, int width, int height) {
        super(c);
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

    /**
     * Draws the previous component and the new coloured glass at the specified location
     * @param g Graphics object to be drawn
     */
    @Override
    public void draw(Graphics g) {
        cmp.draw(g);
```

```
        Graphics2D g2d = (Graphics2D) g;
        Stroke oldStroke = g2d.getStroke();
        Color oldColor = g2d.getColor();
        g2d.setColor(Color.green);

g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 1 *
0.1f));
        g2d.fillRect(25, 25, 110, 110);
        g2d.setStroke(oldStroke);
        g2d.setColor(oldColor);
    }
}
```

Component.java

```
package ExerciseBandC;
/*
 * File Name: Component.java
 * Lab # and Assignment #: Lab 7 Exercise B and C
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
import java.awt.*;

/**
 * Interface Name: Component
 *
 * Defines the draw method definition for all Components to implement
 */
public interface Component {
    void draw(Graphics g);
}
```

Decorator.java

```
package ExerciseBandC;

/*
 * File Name: Decorator.java
 * Lab # and Assignment #: Lab 7 Exercise B and C
 * Lab section: B01
```

```
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */

/**
 * Class Name: Decorator
 *
 * Abstract class that all Decorators will extend from. Provides the
protected data members
 * for all Decorators to inherit.
 */
public abstract class Decorator implements Component {
    protected Component cmp;
    protected int x, y, width, height;

    public Decorator(Component c) {
        cmp = c;
    }
}
```

DemoDecoratorPattern.java

```
package ExerciseBandC;
/*
 * File Name: DemoDecoratorPattern.java
 * Lab # and Assignment #: Lab 7 Exercise B and C
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
import java.awt.*;
import javax.swing.JFrame;
import javax.swing.JPanel;

/**
 * Class Name: DemoDecoratorPattern
 *
 * Provides the driver code to test the implementation of the Decorator
design pattern
 */
public class DemoDecoratorPattern extends JPanel {
    Component t;
```

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

    public void paintComponent(Graphics g){
            //EXERCISE A CODE:
        int fontSize = 10;
            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, 110, 110, 10);

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

            //EXERCISE C CODE: uncomment lines to test it
//        g.setFont(new Font("TimesRoman", Font.PLAIN, fontSize));
//        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);
    }
}
```

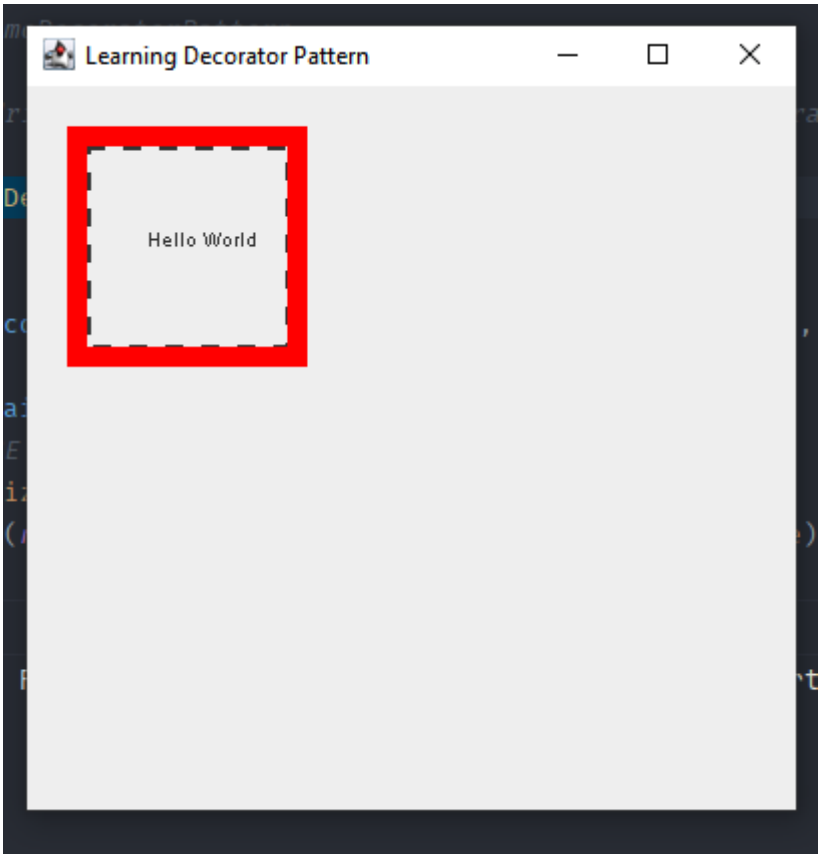
Text.java

```java
package ExerciseBandC;
/*
 * File Name: Text.java
 * Lab # and Assignment #: Lab 7 Exercise B and C
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
import java.awt.*;

/**
 * Class Name: Text
 *
 * Provides the methods to draw a String to the GUI at the specified
location
 */
public class Text implements Component {
    /**
     * the x and y coordinates of where the text will be drawn
     */
    private int x, y;

    /**
     * the String to be drawn
     */
    private String text;

    /**
     * Constructs a Text object and initializes its data members from the
     * specified arguments
     * @param text the String to be drawn
     * @param x the x coordinate
     * @param y the y coordinate
     */
    public Text(String text, int x, int y) {
        this.text = text;
        this.x = x;
        this.y = y;
    }

    /**
     * Draws the String contained in the text member variable at the
specified location
```

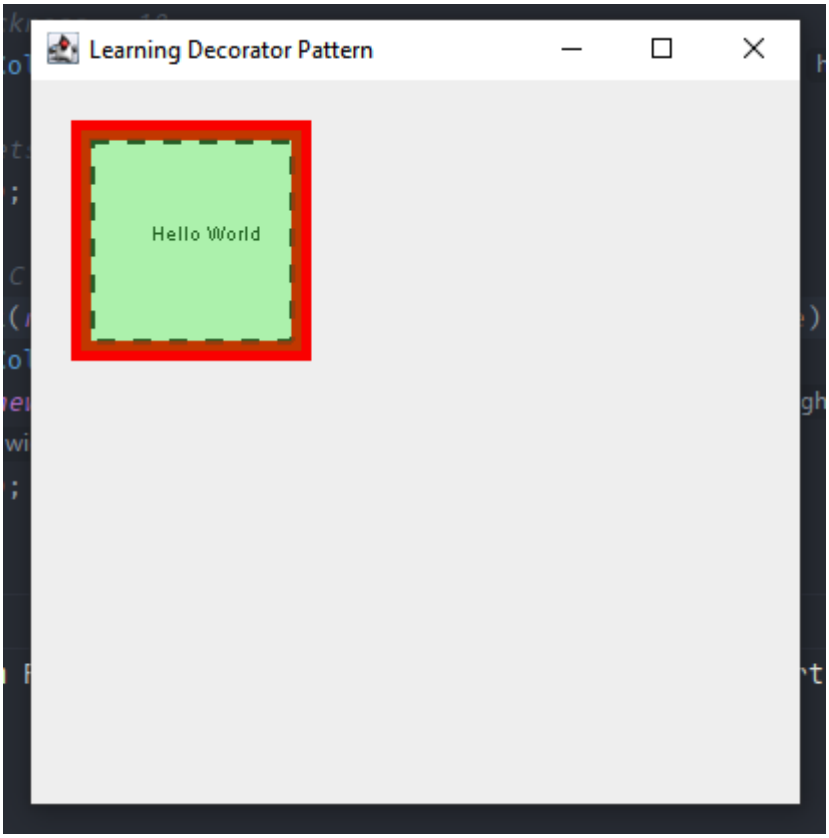
```
     * @param g Graphics object to be drawn
     */
    @Override
    public void draw(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.drawString(text, x, y);
    }
}
```

Program Output

Exercise B



Exercise C


Learning Decorator Pattern
Hello World

Exercise D

Source Code

main.cpp

```
/*
 * File Name: main.cpp
 * Lab # and Assignment #: Lab 7 Exercise D
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
#include "Client_A.hpp"
#include "Client_B.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;
}
```

LoginServer.cpp

```
/*
 * File Name: LoginServer.cpp
 * Lab # and Assignment #: Lab 7 Exercise D
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
#include "LoginServer.hpp"
#include "User.hpp"
#include <iostream>
#include <string>
using namespace std;

LoginServer* LoginServer::instance = 0;

LoginServer::LoginServer() {

}

LoginServer::LoginServer(const LoginServer& src) {
    instance = LoginServer::getInstance();
    users = vector<User> (users);
}

LoginServer& LoginServer::operator=(const LoginServer& rhs) {
    if (this != &rhs) {
        instance = LoginServer::getInstance();
        users = vector<User> (users);
    }
    return *this;
}

LoginServer* LoginServer::getInstance(){
```

```
    if (instance == NULL) {
        instance = new LoginServer;
    }
    return instance;
}

void LoginServer::add(string username, string password) {
    struct User user;
    user.password = password;
    user.username = username;

    for (int i = 0; i < (int) users.size(); i++) {
        if (users.at(i).username.compare(username) == 0) {
            cout << "Username already exists, unable to add user!" <<
endl;
            return;
        }
    }
    users.push_back(user);
    cout << "User successfully added!" << endl;
}

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

Client_A.cpp

```
/*
 * File Name: Client_A.cpp
 * Lab # and Assignment #: Lab 7 Exercise D
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
#include "User.hpp"
#include "Client_A.hpp"
```

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

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

Client_A::Client_A(const Client_A& source) {
    instance = LoginServer::getInstance();
}

Client_A& Client_A::operator = (const Client_A& rhs) {
    if (this != &rhs) {
        instance = LoginServer::getInstance();
    }
    return *this;
}

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

User* Client_A::validate(string username, string password) {
    User* foundUser = instance->validate(username, password);
    return foundUser;
}
```

Client_B.cpp

```
/*
 * File Name: Client_B.cpp
 * Lab # and Assignment #: Lab 7 Exercise D
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Nov 6 2020
 */
#include "User.hpp"
#include "Client_B.hpp"

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

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

Client_B::Client_B(const Client_B& source) {
    instance = LoginServer::getInstance();
}

Client_B& Client_B::operator = (const Client_B& rhs) {
    if (this != &rhs) {
        instance = LoginServer::getInstance();
    }
    return *this;
}


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

User* Client_B::validate(string username, string password) {
    User* foundUser = instance->validate(username, password);
    return foundUser;
}
```

Program Output (with `if 0`)

```
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\davis\Desktop\ENSF 619\Labs\Lab7> g++ -Wall *.cpp -o Singleton.exe
PS C:\Users\davis\Desktop\ENSF 619\Labs\Lab7> .\Singleton.exe
Created a new Client_A object called ca ...
adding two usernames, Jack and Judy, by client ca ...
User successfully added!
User successfully added!
Created a new Client_B object called cb ...
Adding two usernames called Jim and Josh, by client cb ...
User successfully added!
User successfully added!
Now adding another username called Jim by client ca.
It must be avoided because a similar username already exits ...
Username already exists, unable to add user!
Another attempt to add username called Jim, but this time by client cb,
with a different password
It must be avoided again ...
Username already exists, unable to add user!
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 ...
User successfully added!
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
PS C:\Users\davis\Desktop\ENSF 619\Labs\Lab7>
```

Questions

1. My program does not allow the creation of LoginServer objects
2. Because it is a requirement of the Singleton pattern that the objects constructor is made private and only ever called if there is no previous instance of the Singleton object already in existance.