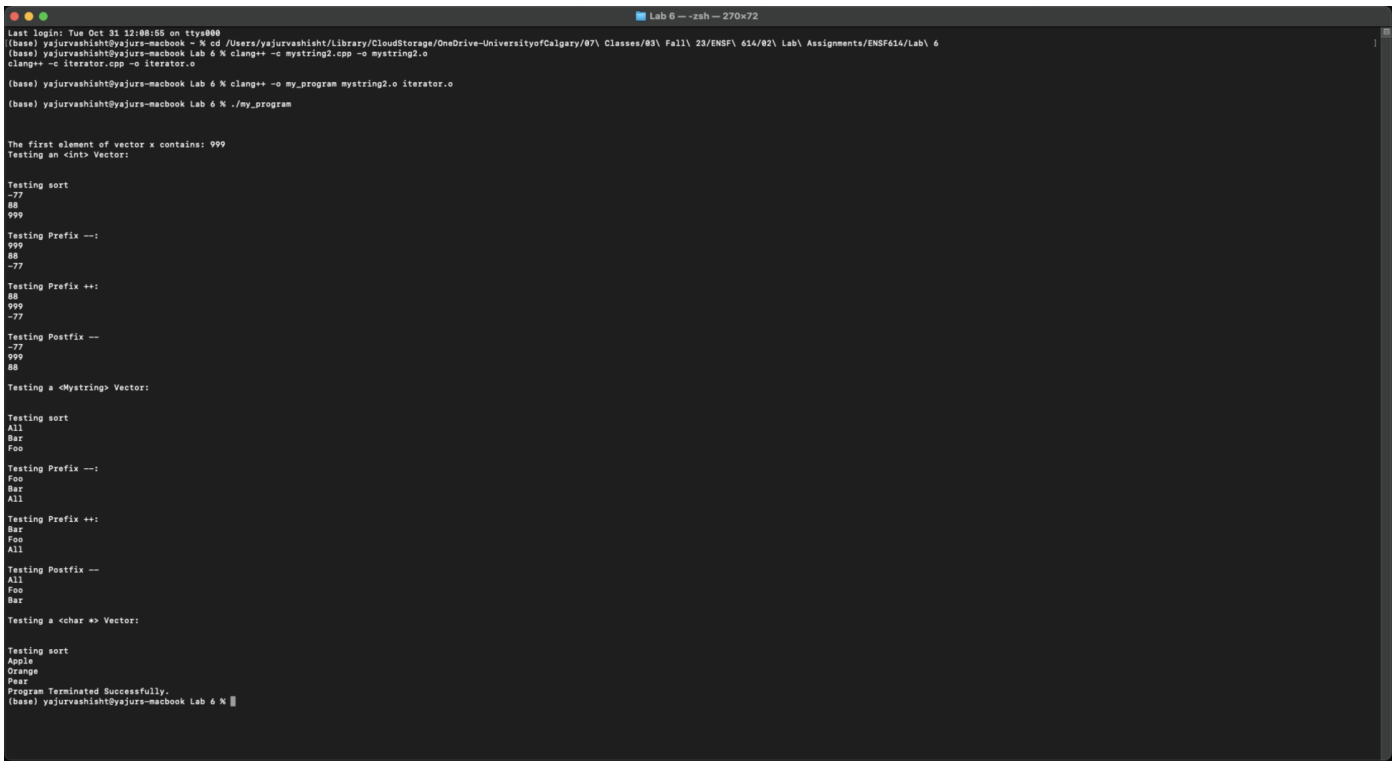
Course: ENSF 614 – Fall 2023
Lab #: Lab 6
Instructor: Moussavi
Student Names: Yajur Vashisht, Balkarn Gill
Submission Date: November 10th, 2023

Exercise A:




```cpp
// iterator.cpp
// ENSF 614 - Fall 2023 - Lab 6, Ex A
// Yajur Vashisht, Balkarn Gill
// UCID - 30200252, 30202219

#include <iostream>
#include <assert.h>
#include "mystring2.h"

using namespace std;

template<class T>
class Vector {
public:

   class VectIter{
       friend class Vector;
   private:
       Vector<T> *v; // points to a vector object of type T
```

```
    int index;    // represents the subscript number of the vector's
    // array.
public:
    VectIter(Vector<T>& x);

    T operator++();
    //PROMISES: increments the iterator's indices and return the
    //          value of the element at the index position. If
    //          index exceeds the size of the array it will
    //          be set to zero. Which means it will be circulated
    //          back to the first element of the vector.

    T operator++(int);
    // PROMISES: returns the value of the element at the index
    //           position, then increments the index. If
    //           index exceeds the size of the array it will
    //           be set to zero. Which means it will be circulated
    //           back to the first element of the vector.

    T operator--();
    // PROMISES: decrements the iterator index, and return
    //           the value of the element at the index. If
    //           index is less than zero it will be set to the
    //           last element in the array. Which means it will be
    //           circulated to the last element of the vector.

    T operator--(int);
    // PROMISES: returns the value of the element at the index
    //           position, then decrements the index. If
    //           index is less than zero it will be set to the
    //           last element in the aray. Which means it will be
    //           circulated to the last element of the vector.

    T operator *();
    // PROMISES: returns the value of the element at the current
    //           index position.
};

Vector(int sz);
~Vector();

T & operator[](int i);
```

```
   // PROMISES: returns existing value in the ith element of
   //           array or sets a new value to  the ith element in
   //           array.

   void ascending_sort();
   // PROMISES: sorts the vector values in ascending order.

private:
   T *array;               // points to the first element of an array of T
   int size;               // size of array
   void swap(T&, T&); // swaps the values of two elements in array
public:
};

template <typename T>
void Vector<T>::ascending_sort()
{
   for(int i=0; i< size-1; i++)
       for(int j=i+1; j < size; j++)
           if(array[i] > array[j])
               swap(array[i], array[j]);
}

// Specialization of ascending_sort function for MyString type
template <>
void Vector<Mystring>::ascending_sort()
{
   for (int i = 0; i < size - 1; i++) {
       for (int j = i + 1; j < size; j++) {
           if (strcmp(array[i].c_str(), array[j].c_str()) > 0) {
               swap(array[i], array[j]);
           }
       }
   }
}

// Specialization of ascending_sort function for const char* type
template <>
void Vector<const char*>::ascending_sort()
{
   for (int i = 0; i < size - 1; i++) {
       for (int j = i + 1; j < size; j++) {
```

```
            if (strcmp(array[i], array[j]) > 0) {
                swap(array[i], array[j]);
            }
        }
    }
}

template <typename T>
void Vector<T>::swap(T& a, T& b)
{
    T tmp = a;
    a = b;
    b = tmp;
}

template <typename T>
T Vector<T>::VectIter::operator *()
{
    return v -> array[index];
}

template <typename T>
Vector<T>::VectIter::VectIter(Vector& x)
{
    v = &x;
    index = 0;
}

template <typename T>
Vector<T>::Vector(int sz)
{
    size=sz;
    array = new T [sz];
    assert (array != NULL);
}

template <typename T>
Vector<T>::~Vector()
{
    delete [] array;
    array = NULL;
}
```

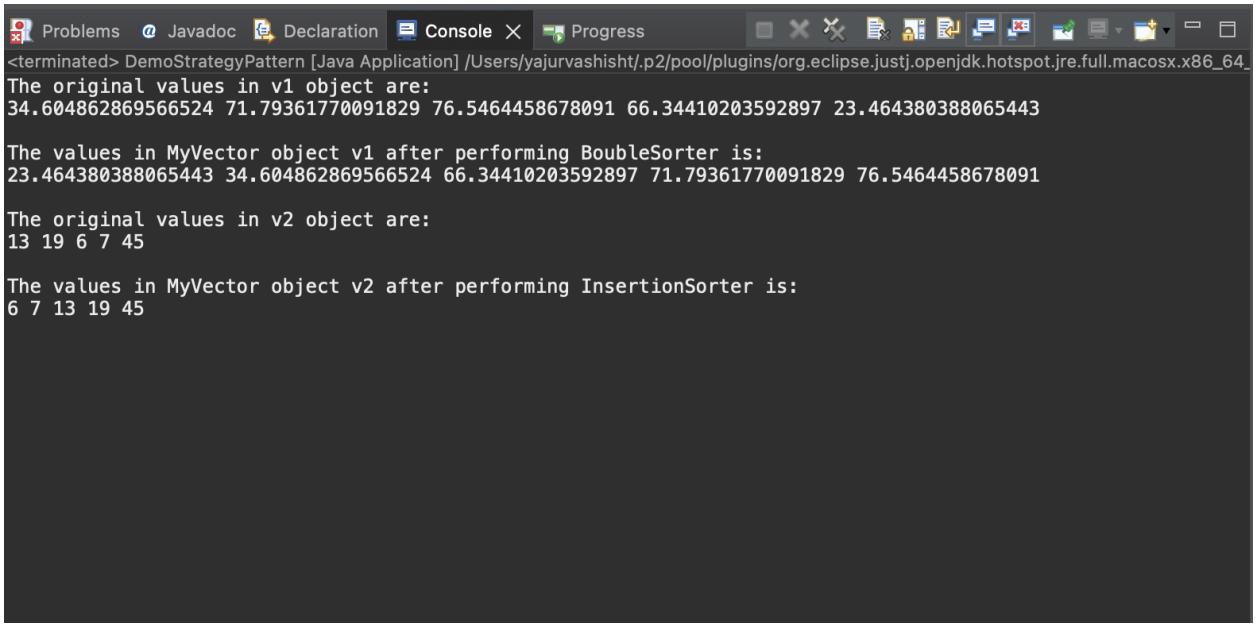
```
template <typename T>
T & Vector<T> ::operator [] (int i)
{
   return array[i];
}

template <typename T>
T Vector<T>::VectIter::operator++() {
   index++;
   if (index >= v->size) {
       index = 0;
   }
   return v->array[index];
}

template <typename T>
T Vector<T>::VectIter::operator++(int) {
   T currentVal = v->array[index];
   index++;
   if (index >= v->size) {
       index = 0;
   }
   return currentVal;
}

template <typename T>
T Vector<T>::VectIter::operator--() {
   index--;
   if (index < 0) {
       index = v->size-1;
   }
   return v->array[index];
}

template <typename T>
T Vector<T>::VectIter::operator--(int) {
   T currentVal = v->array[index];
   index--;
   if (index < 0) {
       index = v->size-1;
   }
```

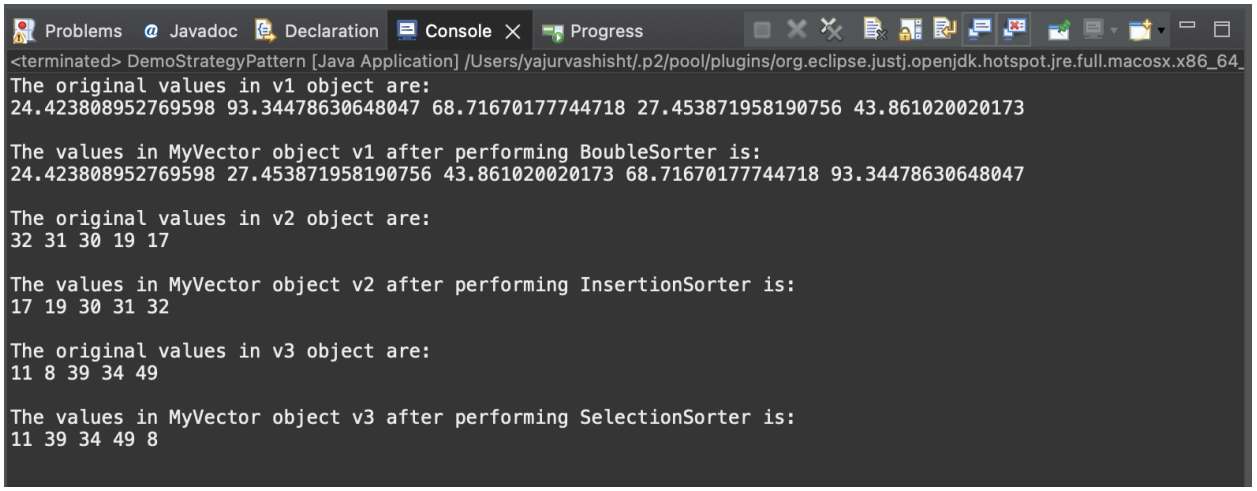
```
   return currentVal;
}


int main()
{

   Vector<int> x(3);
   x[0] = 999;
   x[1] = -77;
   x[2] = 88;

   Vector<int>::VectIter iter(x);

   cout << "\n\nThe first element of vector x contains: " << *iter;

   // the code between the  #if 0 and #endif is ignored by
   // compiler. If you change it to #if 1, it will be compiled

#if 1
   cout << "\nTesting an <int> Vector: " << endl;

   cout << "\n\nTesting sort";
   x.ascending_sort();

   for (int i=0; i<3 ; i++)
       cout << endl << iter++;

   cout << "\n\nTesting Prefix --:";
   for (int i=0; i<3 ; i++)
       cout << endl << --iter;

   cout << "\n\nTesting Prefix ++:";
   for (int i=0; i<3 ; i++)
       cout << endl << ++iter;

   cout << "\n\nTesting Postfix --";
   for (int i=0; i<3 ; i++)
       cout << endl << iter--;

   cout << endl << endl;
```

```
   cout << "Testing a <Mystring> Vector: " << endl;
   Vector<Mystring> y(3);
   y[0] = "Bar";
   y[1] = "Foo";
   y[2] = "All";;

   Vector<Mystring>::VectIter iters(y);

   cout << "\n\nTesting sort";
   y.ascending_sort();

   for (int i=0; i<3 ; i++)
       cout << endl << iters++.c_str();

   cout << "\n\nTesting Prefix --:";
   for (int i=0; i<3 ; i++)
       cout << endl << (--iters).c_str();

   cout << "\n\nTesting Prefix ++:";
   for (int i=0; i<3 ; i++)
       cout << endl << (++iters).c_str();

   cout << "\n\nTesting Postfix --";
   for (int i=0; i<3 ; i++)
       cout << endl << iters--.c_str();

   cout << endl << endl;
   cout << "Testing a <char *> Vector: " << endl;
   Vector<const char*> z(3);
   z[0] = "Orange";
   z[1] = "Pear";
   z[2] = "Apple";;

   Vector<const char*>::VectIter iterchar(z);

   cout << "\n\nTesting sort";
   z.ascending_sort();

   for (int i=0; i<3 ; i++)
       cout << endl << iterchar++;

#endif
```

```
    cout << "\nProgram Terminated Successfully." << endl;


    return 0;
}
```

Exercise B/C:






```java
/* ENSF 614 - Lab 6 - Exercise B and C
* Yajur Vashisht, November 2023
*/

import java.util.ArrayList;

public class BubbleSorter<E extends Number & Comparable<E>> implements Sorter<E> {

   @Override
   public void sort(ArrayList<Item<E>> storage) {
```

```
        for (int i = 0; i < storage.size(); i++) {
            for (int j = 0; j < storage.size() - 1; j++) {
                if (storage.get(j).getItem().compareTo(storage.get(j + 1).getItem()) >
0) {

                    // Switching i and j
                    Item<E> temp = (Item<E>) storage.get(j);
                    storage.set(j, storage.get(j + 1));
                    storage.set(j + 1, temp);
                }
            }
        }
    }
}
```

```
/* ENSF 614 - Lab 6 - Exercise C and D
* M. Moussavi, October 2021
*/

import java.util.Random;

public class DemoStrategyPattern {
    public static void main(String[] args) {
        // Create an object of MyVector<Double> with capacity of 50 elements
        MyVector<Double> v1 = new MyVector<Double>(50);
        // Create a Random object to generate values between 0
        Random rand = new Random();
        // adding 5 randomly generated numbers into MyVector object v1
        for (int i = 4; i >= 0; i--) {
            Item<Double> item;
            item = new Item<Double>(Double.valueOf(rand.nextDouble() * 100));
            v1.add(item);
        }
        // displaying original data in MyVector v1
        System.out.println("The original values in v1 object are:");
        v1.display();
        // choose algorithm bubble sort as a strategy to sort object v1
        v1.setSortStrategy(new BubbleSorter<Double>());
        // perform algorithm bubble sort to v1
        v1.performSort();
```

```
        System.out.println("\nThe values in MyVector object v1 after performing
BoubleSorter is:");
        v1.display();
        // create a MyVector<Integer> object V2
        MyVector<Integer> v2 = new MyVector<Integer>(50);
        // populate v2 with 5 randomly generated numbers
        for (int i = 4; i >= 0; i--) {
            Item<Integer> item;
            item = new Item<Integer>(Integer.valueOf(rand.nextInt(50)));
            v2.add(item);
        }
        System.out.println("\nThe original values in v2 object are:");
        v2.display();
        v2.setSortStrategy(new InsertionSorter<Integer>());
        ;
        v2.performSort();
        System.out.println("\nThe values in MyVector object v2 after performing
InsertionSorter is:");
        v2.display();

        // create a MyVector<Integer> object V3
        MyVector<Integer> v3 = new MyVector<Integer>(50);
        // populate v2 with 5 randomly generated numbers
        for (int i = 4; i >= 0; i--) {
            Item<Integer> item;
            item = new Item<Integer>(Integer.valueOf(rand.nextInt(50)));
            v3.add(item);
        }
        System.out.println("\nThe original values in v3 object are:");
        v3.display();
        v3.setSortStrategy(new SelectionSorter<Integer>());
        ;
        v3.performSort();
        System.out.println("\nThe values in MyVector object v3 after performing
SelectionSorter is:");
        v3.display();
    }
}
```

```
/* ENSF 614 - Lab 6 - Exercise B and C
* Yajur Vashisht, November 2023
```

```java
*/

import java.util.ArrayList;

public class InsertionSorter<E extends Number & Comparable<E>> implements Sorter<E> {

   public void sort(ArrayList<Item<E>> storage) {
       int n = storage.size();
       for (int i = 1; i < n; i++) {
           int j = i;
           if (storage.get(j).getItem().compareTo(storage.get(j - 1).getItem()) < 0) {
               while ((j >= 1) && (storage.get(i).getItem().compareTo(storage.get(j -
1).getItem()) < 0)) {
                   j--;
               }
               Item<E> temp = (Item<E>) storage.get(i);
               storage.remove(i);
               storage.add(j, temp);
           }
       }
   }

}
```

```java
/* ENSF 614 - Lab 6 Exercise C and D
* M. Moussavi, October 2021
*
*/

class Item <E extends Number & Comparable<E> >{
   private E item;
   public Item(E value) {
       item = value;
   }

   public void setItem(E value){
       item = value;
   }

   public E getItem(){
       return item;
```

```
    }
}
```

```
/* ENSF 614 - Lab 6 - Exercise B and C
* Yajur Vashisht, November 2023
*/

import java.util.*;

public class MyVector<E extends Number & Comparable<E>> {

    private ArrayList<Item<E>> storageM;
    private Sorter<E> sorter;

    public MyVector(int n) {
        storageM = new ArrayList<>(n);
    }

    @SuppressWarnings("unchecked")
    public MyVector(ArrayList<E> arr) {
        storageM = (ArrayList<Item<E>>) arr.clone();
    }

    public void add(Item<E> value) {
        storageM.add((Item<E>) value);
    }

    public void setSortStrategy(Sorter<E> s) {
        this.sorter = s;
    }

    public void performSort() {
        sorter.sort(storageM);
    }

    public void display() {
        for (int i = 0; i < storageM.size(); i++) {
            System.out.print(storageM.get(i).getItem() + " ");
        }
        System.out.println();
    }
```

```
}
```

```
/* ENSF 614 - Lab 6 - Exercise B and C
* Yajur Vashisht, November 2023
*/


import java.util.ArrayList;

public class SelectionSorter <E extends Number & Comparable<E>> implements Sorter<E> {

   @Override
   public void sort(ArrayList<Item<E>> storage) {
       for (int i = 0; i < storage.size(); i++) {
           int small = i;
           for (int j = 0; j < storage.size(); j++) {
               if (storage.get(j).getItem().compareTo(storage.get(small).getItem()) <
0) {
                   small = j;
               }
           }
           storage.add(i, storage.remove(small));
       }
   }
}
```

```
import java.util.*;

public interface Sorter<E extends Number & Comparable<E>> {
   public void sort(ArrayList<Item<E>> storage);
}
```

Exercise D:

```
<terminated> ObserverPatternController [Java Application] /Users/gill/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_17.0.7.v20230425-1502/jre/bin/java  (Nov. 12, 2023, 5:20:18 p.m. – 5:20
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 Five-Rows Table 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 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 Five-Rows Table 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 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
```

```java
package exerciseD;
import java.util.ArrayList;
public class DoubleArrayListSubject implements Subject {

	public ArrayList<Double> data;
	private ArrayList<Observer> observers;

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

	public void display() {
		if (data.isEmpty()) {
			System.out.println("Empty List ...");
		} else {
			System.out.println(data.toString());
		}
	}
	public void addData(Double value) {
		data.add(value);
		notifyAllObservers();
```

```
	}

	public void setData(Double value, int index) {
		data.set(index, value);
		notifyAllObservers();
	}

	public void populate(double[] values) {
		for (Double x: values) {
			data.add(x);
		}
		notifyAllObservers();
	}

	public void registerObserver(Observer o) {
		if (!observers.contains(o)){
			observers.add(o);
		}
		o.update(data);
	}

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

	public void notifyAllObservers() {
		for (Observer o: observers) {
			o.update(data);
		}
	}
	@Override
	public void removeObserver(Observer o) {
		// TODO Auto-generated method stub

	}
}
```

```
package exerciseD;
import java.util.ArrayList;
public class FiveRowsTable_Observer implements Observer{

	public FiveRowsTable_Observer(Subject s) {
		s.registerObserver(this);
	}

	public void update(ArrayList<Double> data) {
```

```
          System.out.println("\nNotification to Five-Rows Table
Observer: Data Changed:");
          display(data);
     }

     private void display(ArrayList<Double> data) {
          for (int i=0; i<5; i++) {
               int j = i;
               while (j < data.size()) {
                    System.out.print(data.get(j)+" ");
                    j += 5;
               }
               System.out.println();
          }
     }
}
```

```
package exerciseD;
import java.util.ArrayList;
interface Observer {
   void update(ArrayList<Double> data);
}
```

```
package exerciseD;
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.print("\nNow, creating a new Three-Column
observer that will be notified of existing data:");
          ht = new ThreeColumnTable_Observer(mydata);
     }
}
```

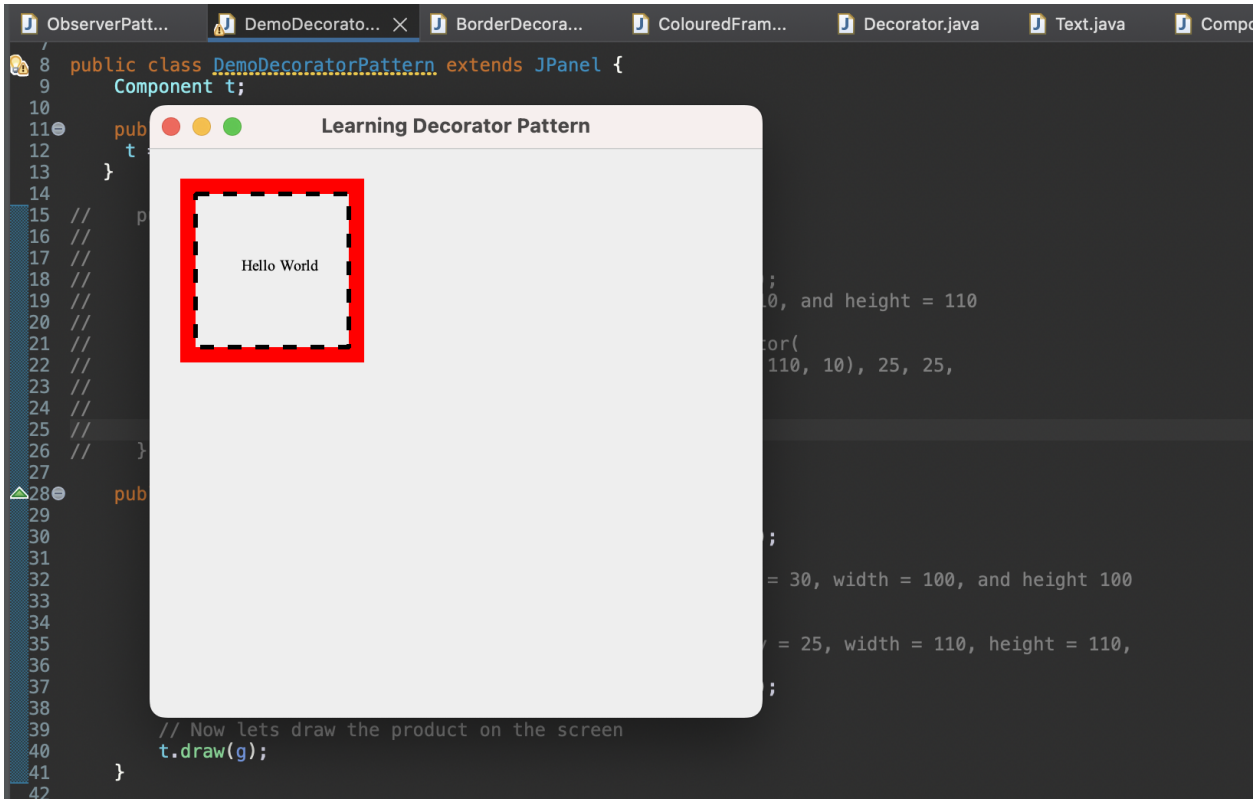
```
package exerciseD;
import java.util.ArrayList;
public class OneRow_Observer implements Observer {

     public OneRow_Observer(Subject s) {
          s.registerObserver(this);
     }

     public void update(ArrayList<Double> data) {
          System.out.println("\nNotification to One-Row
Observer: Data Changed:");
          display(data);
     }
```

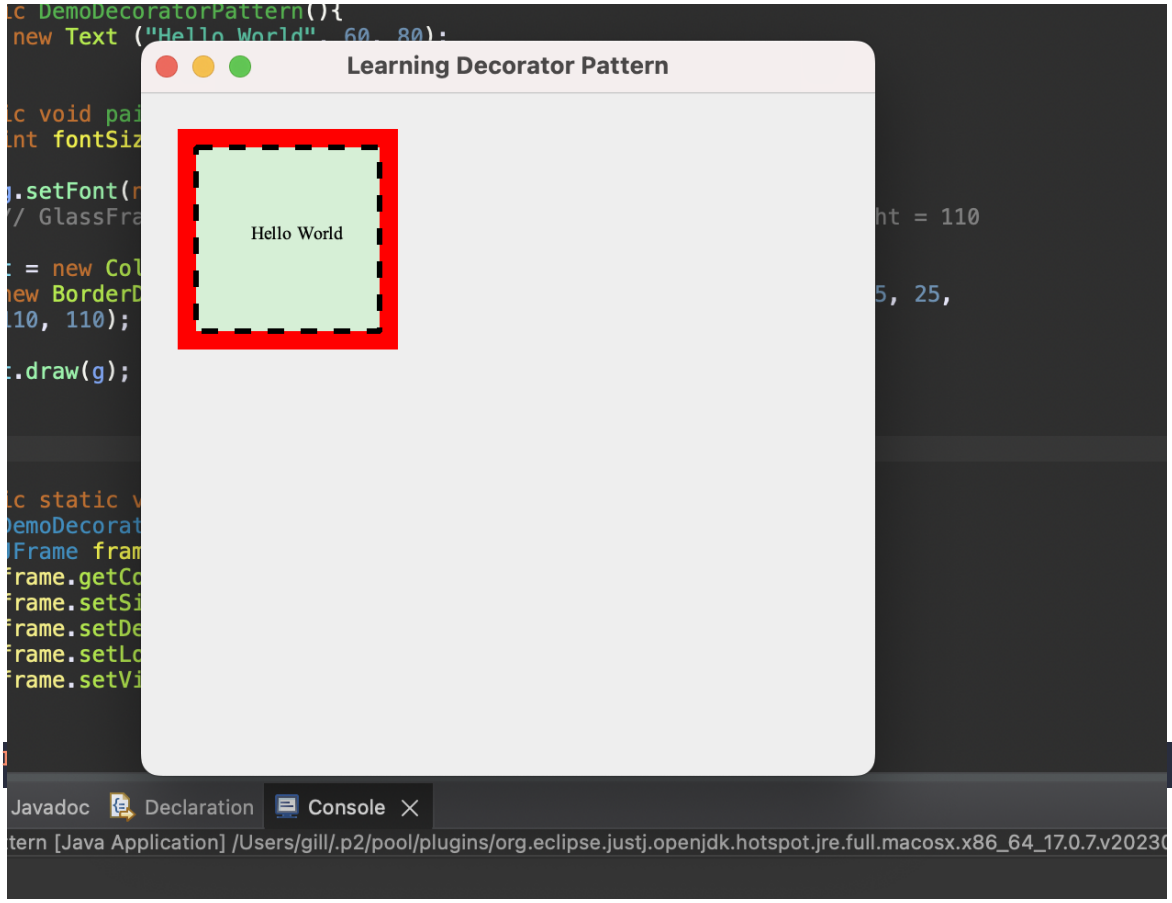
```
    private void display(ArrayList<Double> data) {
        for (Double d: data) {
            System.out.print(d + " ");
        }
    }
}
```

```
package exerciseD;
import java.util.ArrayList;
interface Subject {
   void registerObserver(Observer o);
   void removeObserver(Observer o);
   void notifyAllObservers();
}
```

```
package exerciseD;
import java.util.ArrayList;
public class ThreeColumnTable_Observer implements Observer{

    public ThreeColumnTable_Observer(Subject s) {
        s.registerObserver(this);
    }

    public void update(ArrayList<Double> data) {
        System.out.println("\nNotification to Three-Column
Table Observer: Data Changed:");
        display(data);
    }

    private void display(ArrayList<Double> data) {
        for (int i=0; i<data.size(); i++) {
            System.out.print(data.get(i));
            if (((i+1) % 3) == 0) {
                System.out.println();
            } else {
                System.out.print(" ");
            }
        }
        System.out.println();
    }
}
```

Exercise E/F:


ObserverPatt...
DemoDecorato...
BorderDecora...
ColouredFram...
Decorator.java
Text.java
public class DemoDecoratorPattern extends JPanel {
Component t;
Learning Decorator Pattern
Hello World
// Now lets draw the product on the screen
t.draw(g);


Learning Decorator Pattern
Hello World
Javadoc
Declaration
Console

```
import java.awt.*;

public class BorderDecorator extends Decorator{

   public BorderDecorator(Component c, int x, int y, int w, int h) {
       this.cmp = c;
       this.x = x;
       this.y = y;
       this.width = w;
       this.height = h;
   }

   public void draw(Graphics g) {
       //TODO
       Stroke dashed = new BasicStroke(3, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_BEVEL, 0, new float[]{9},
               0);
       Graphics2D g2d = (Graphics2D) g;
       g2d.setStroke(dashed);
       g2d.drawRect(30, 30, 100, 100);
       cmp.draw(g);
   }
}
```

```
package exerciseE;

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

public class ColouredFrameDecorator extends Decorator{

   private int thickness;

   public ColouredFrameDecorator(Component c, int x, int y, int w, int h, int t) {
       this.cmp = c;
       this.x = x;
```

```
        this.y = y;
        this.width = w;
        this.height = h;
        this.thickness = t;
    }

    public void draw(Graphics 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);

        cmp.draw(g);
    }
}
```

```
package exerciseE;

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

public class ColouredGlassDecorator extends Decorator{

    public ColouredGlassDecorator(Component c, int x, int y, int w, int h) {
        this.cmp = c;
        this.x = x;
        this.y = y;
        this.width = w;
        this.height = h;
    }
```

```
   public void draw(Graphics g) {

       Graphics2D g2d = (Graphics2D) g;
       Color oldColor = g2d.getColor();
       Composite oldComposite = g2d.getComposite();

       g2d.setColor(Color.green);
       g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 1 *
0.1f));
       g2d.fillRect(25, 25, 110, 110);

       g2d.setComposite(oldComposite);
       g2d.setColor(oldColor);

       cmp.draw(g);
   }
}
```

```
package exerciseE;

import java.awt.Graphics;

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

```
package exerciseE;


public abstract class Decorator implements Component {

   protected Component cmp;
   protected int x;
   protected int y;
   protected int width;
   protected int height;

}
```

```java
package exerciseE;

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;

       g.setFont(new Font("TimesRoman", Font.PLAIN, fontSize));
       // GlassFrameDecorator info: 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);
   }
}
```

```java
package exerciseE;

import java.awt.Graphics;

public class Text implements Component{
   private int x;
   private int y;
   private String text;

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


   public void draw(Graphics g) {
       g.drawString(text, x, y);
   }
}
```