Course: ENSF 614 - Fall 2023

Lab #: Lab 6

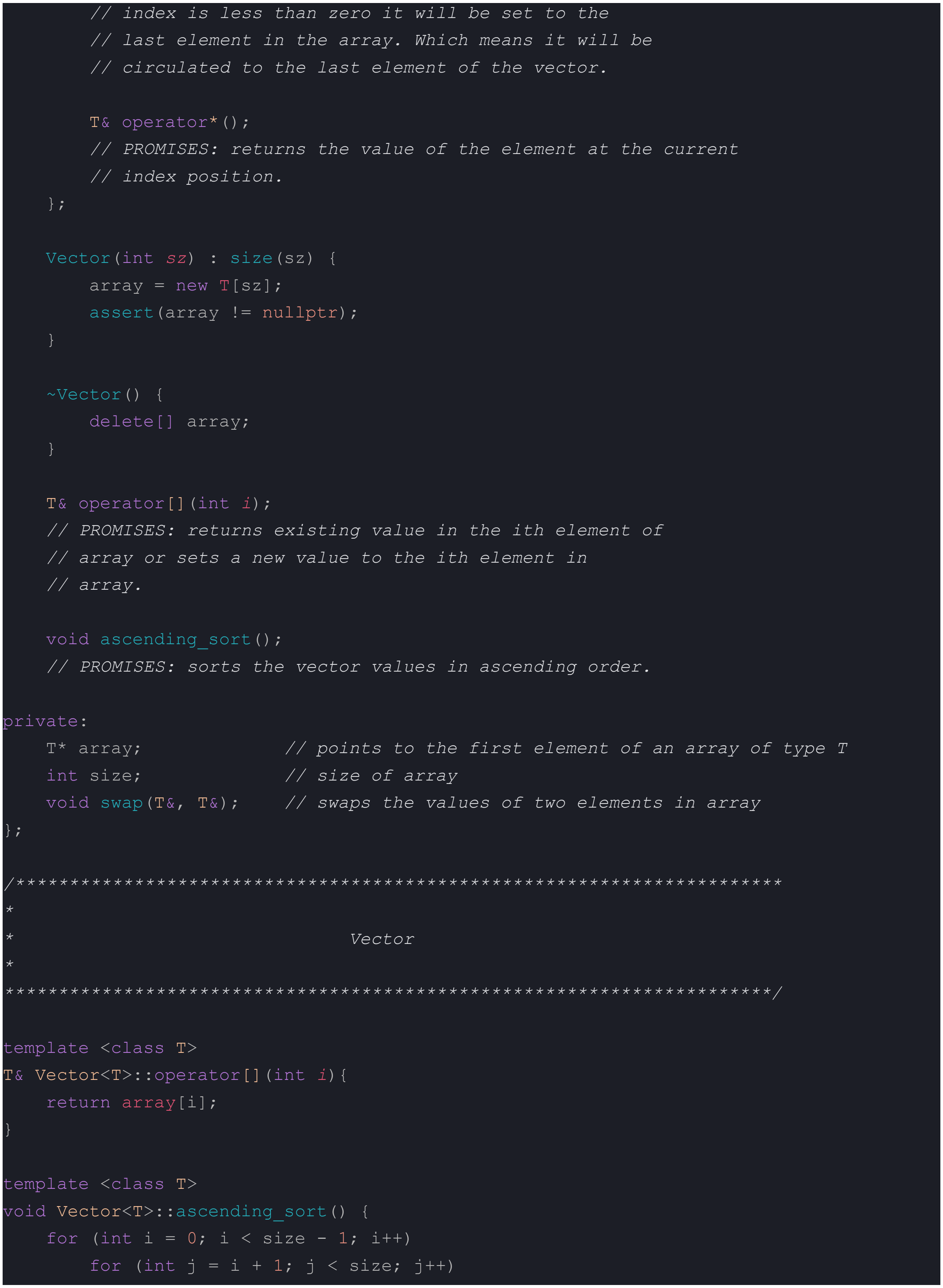
Instructor: Mahmood Moussavi

Student Names: Braden Tink and Christian Valdez

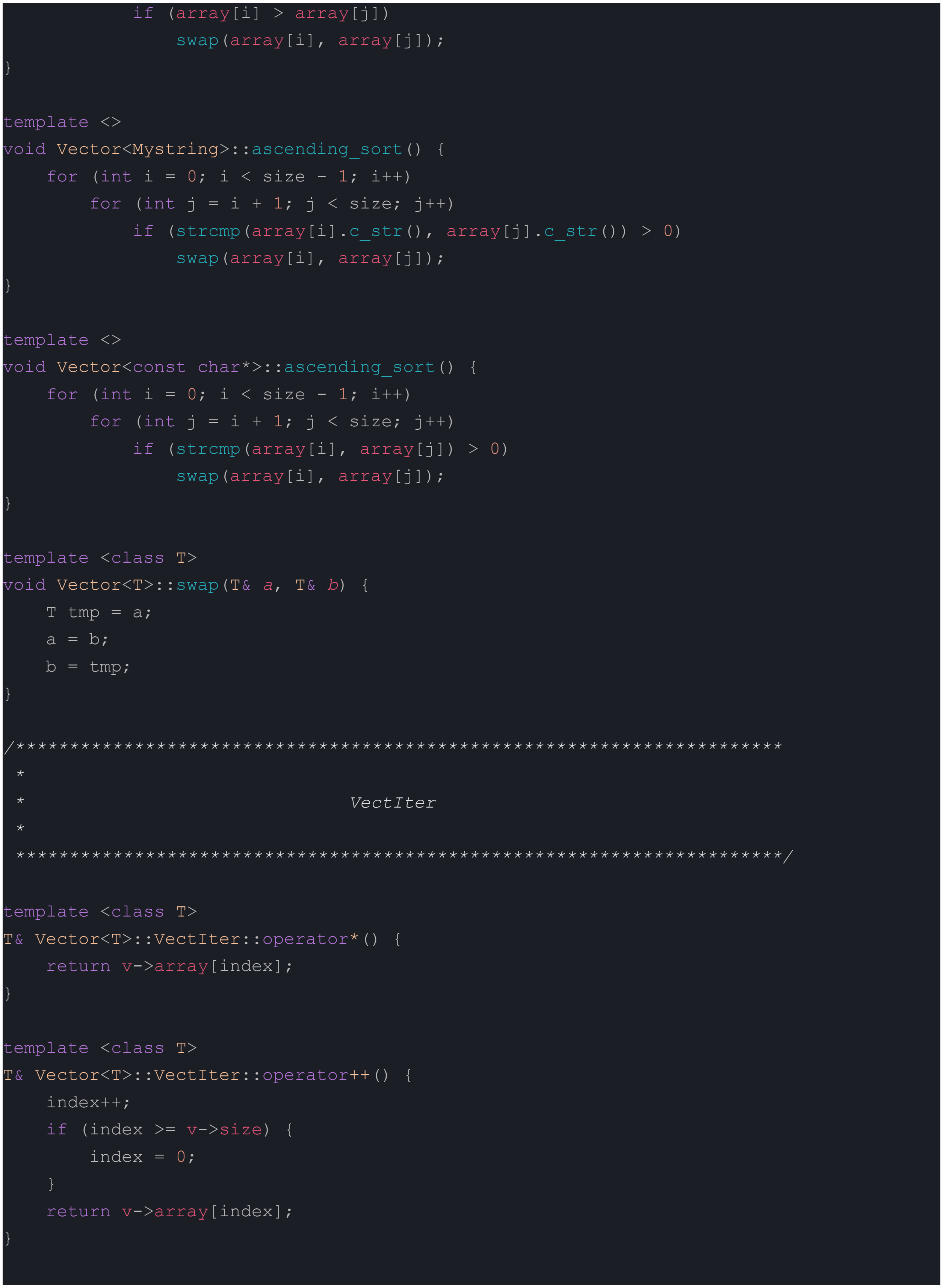
Submission Date: November 11, 2023

Exercise A

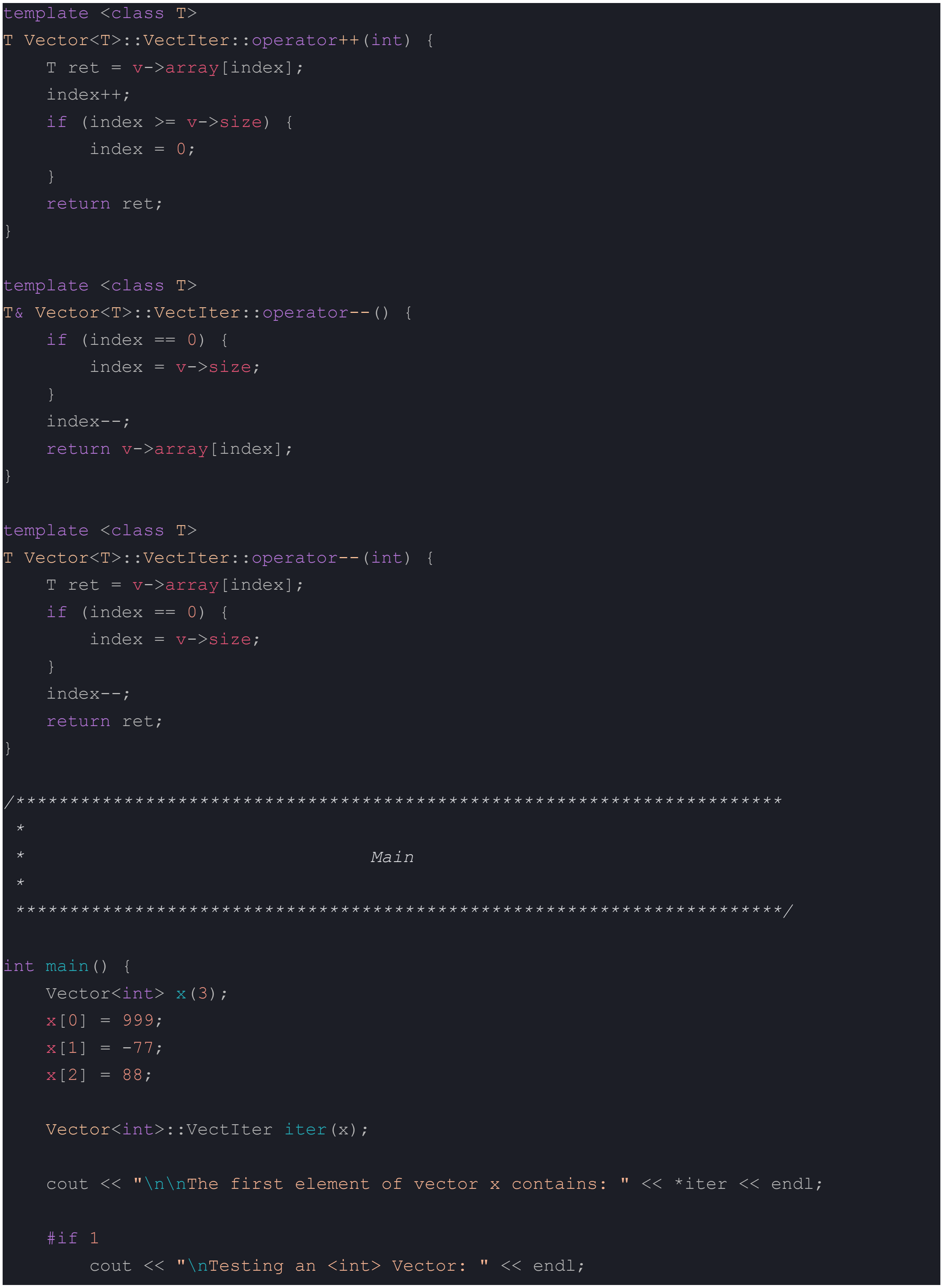
```
// iterator.cpp
// ENSF 614 Fall 2023 LAB 6 - EXERCISE A
// Created By: Braden Tink and Christian Valdez
// Submitted On: Nov 11, 2023

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

using namespace std;

template <class T>
class Vector {
public:
    class VectIter {
        friend class Vector<T>;
    private:
        Vector* v; // points to a vector object of type T
        int index; // represents the subscript number of the vector's array.
    public:
        VectIter(Vector& x) : v(&x), index(0) {}

        T& operator++();
        // PROMISES: increments the iterator's index 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
```



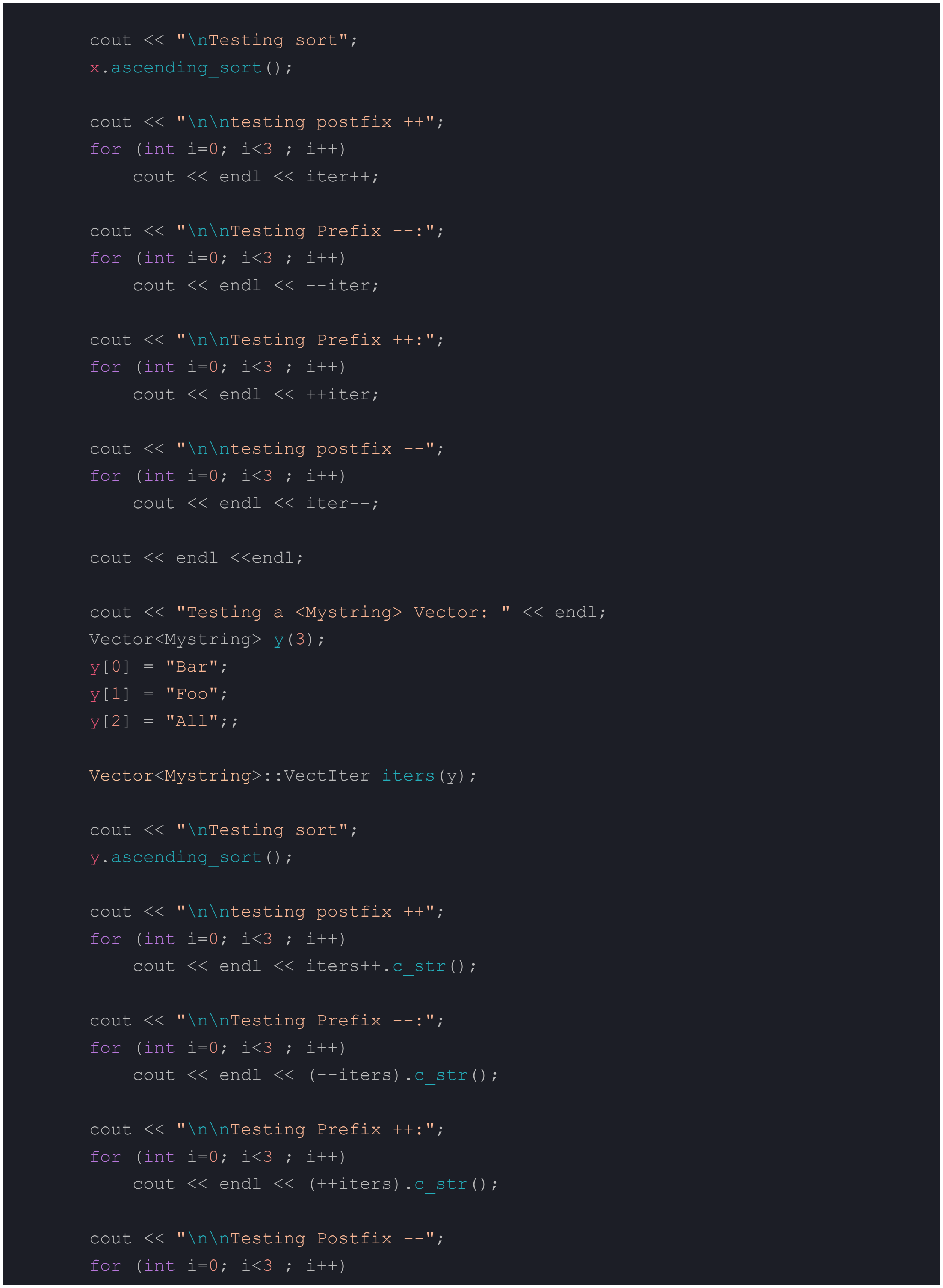
```
        // index is less than zero it will be set to the
        // last element in the array. Which means it will be
        // circulated to the last element of the vector.

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

    Vector(int sz) : size(sz) {
        array = new T[sz];
        assert(array != nullptr);
    }

    ~Vector() {
        delete[] array;
    }

    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 type T
    int size;             // size of array
    void swap(T&, T&);    // swaps the values of two elements in array
};

/***********************************************************************
*
*                               Vector
*
***********************************************************************/

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

template <class T>
void Vector<T>::ascending_sort() {
    for (int i = 0; i < size - 1; i++)
        for (int j = i + 1; j < size; j++)
```



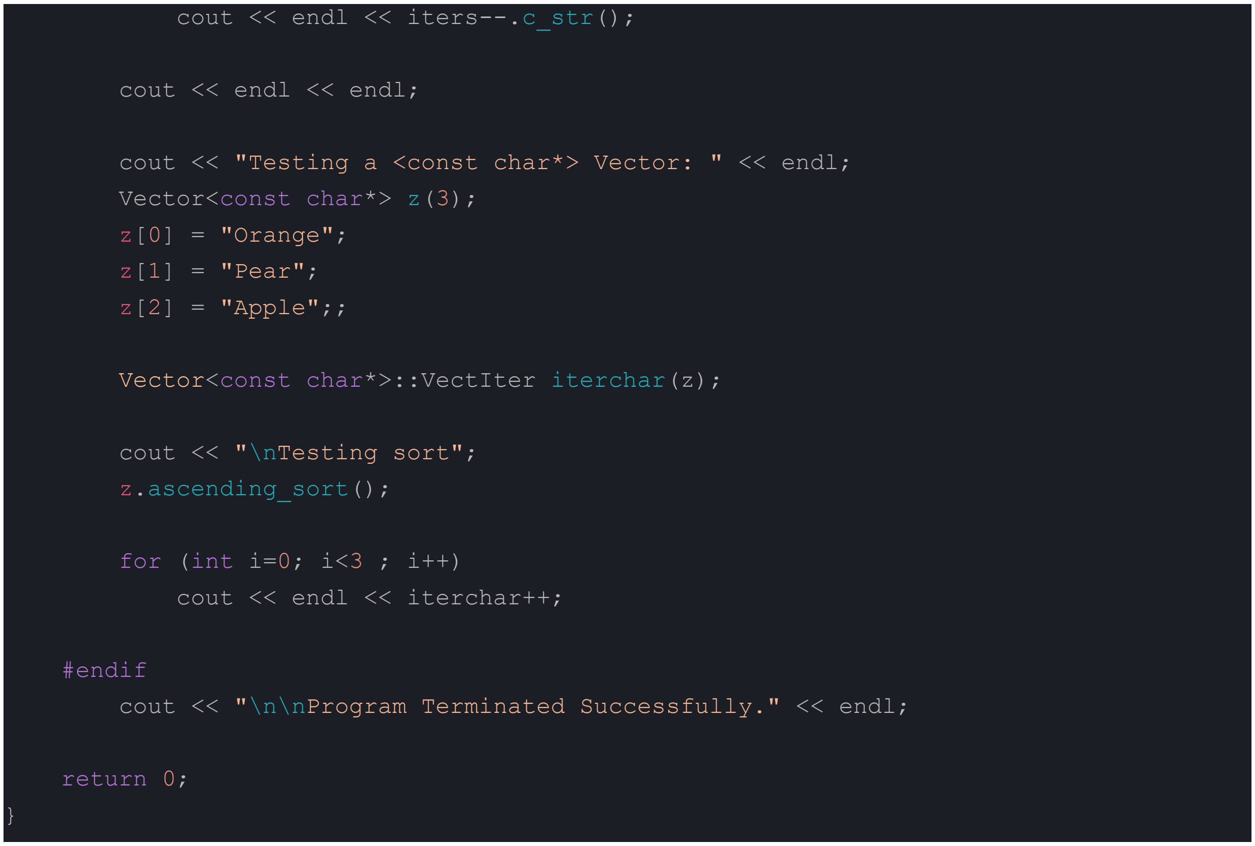
```
            if (array[i] > array[j])
                swap(array[i], array[j]);
}

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]);
}

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

/************************************************************************
 *
 *                              VectIter
 *
 ************************************************************************/

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

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



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

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

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

/***********************************************************************
 *
 *                                Main
 *
 ***********************************************************************/

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 << endl;

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

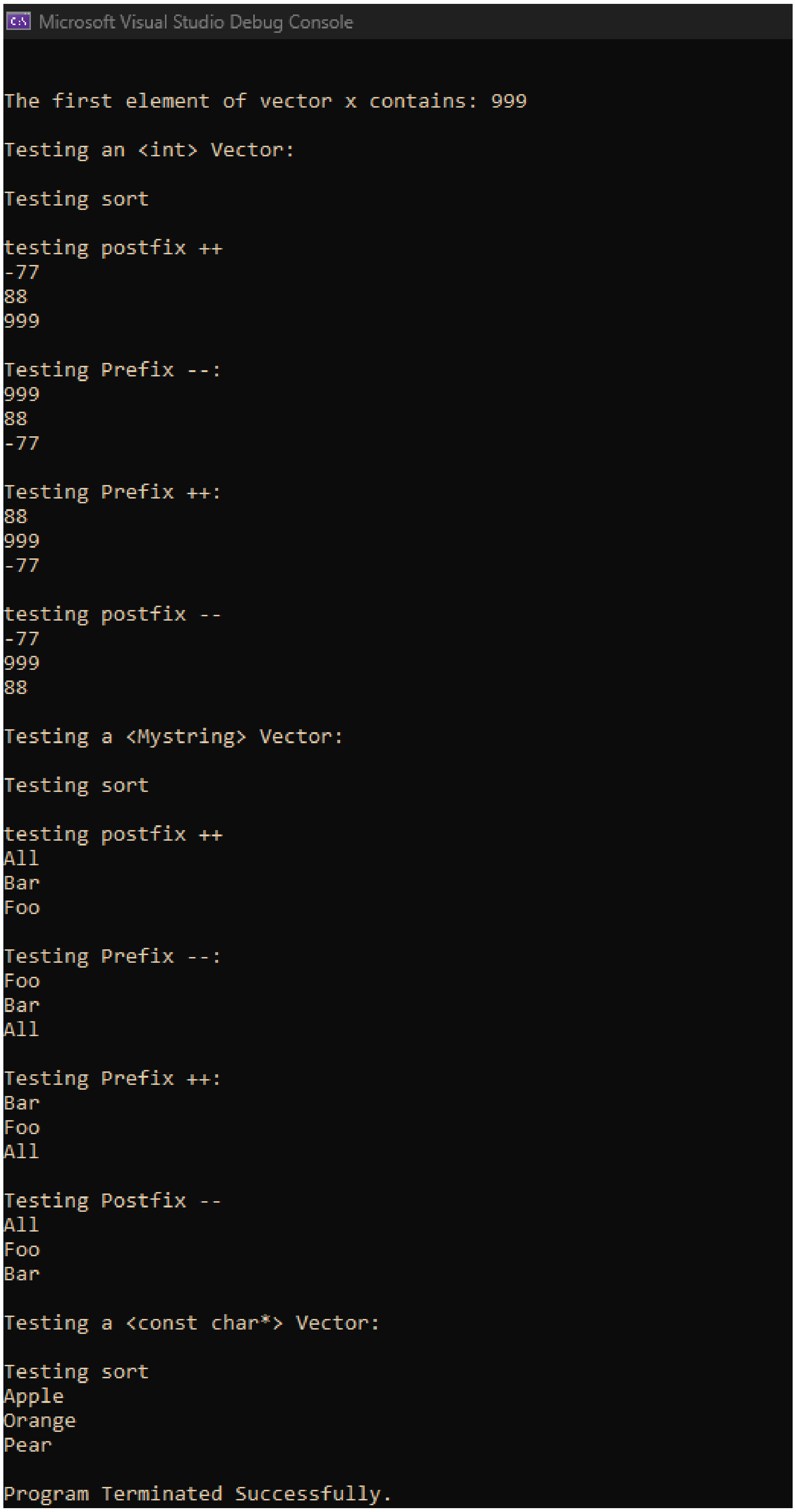


```
    cout << "\nTesting sort";
    x.ascending_sort();

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

    cout << "\n\ntesting postfix ++";
    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 <const char*> Vector: " << endl;
        Vector<const char*> z(3);
        z[0] = "Orange";
        z[1] = "Pear";
        z[2] = "Apple";;

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

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

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

    #endif
        cout << "\n\nProgram Terminated Successfully." << endl;

    return 0;
}
```

Sample Output

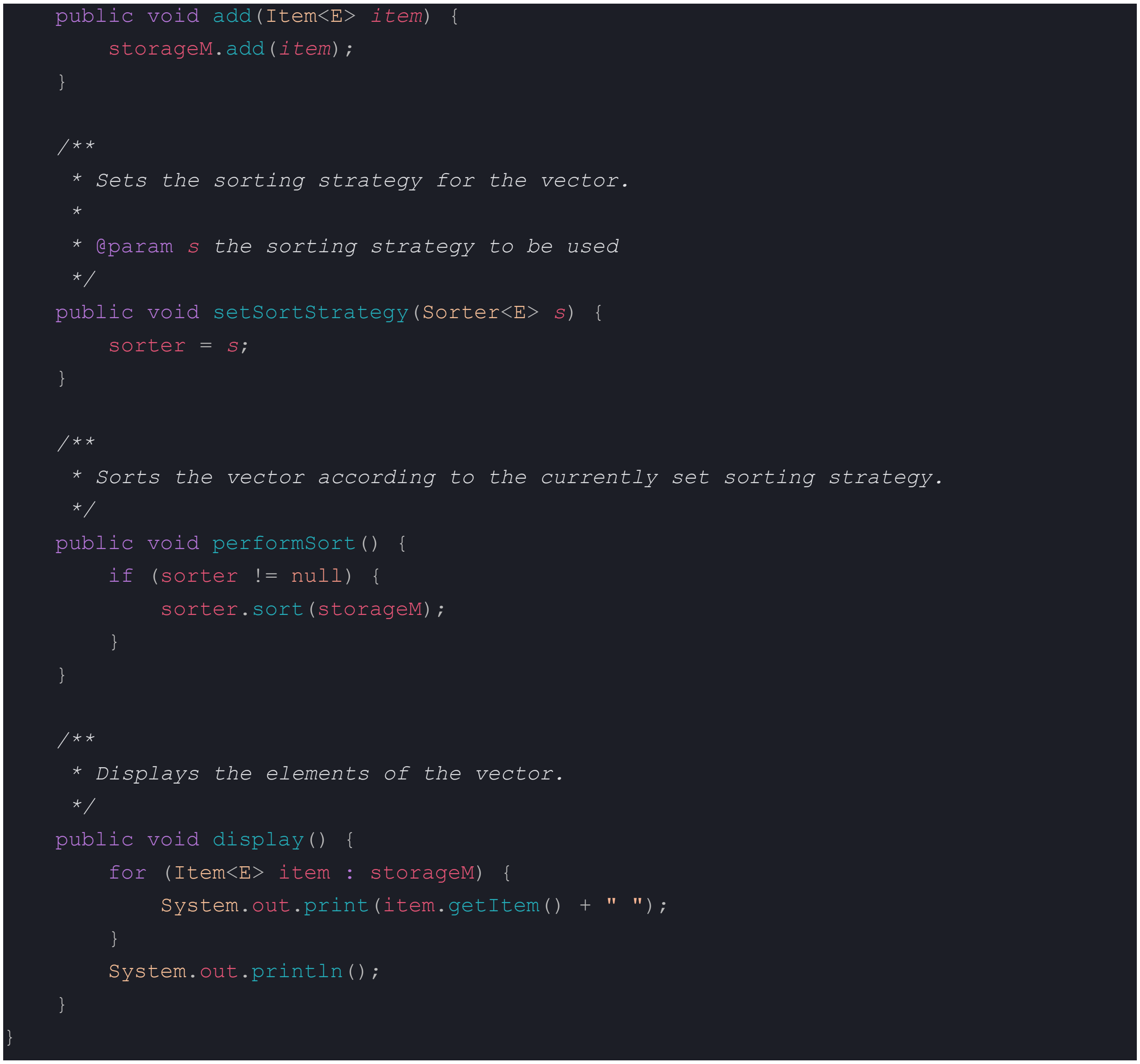


```

The first element of vector x contains: 999

Testing an <int> Vector:

Testing sort

testing postfix ++
-77
88
999

Testing Prefix --:
999
88
-77

Testing Prefix ++:
88
999
-77

testing postfix --
-77
999
88

Testing a <Mystring> Vector:

Testing sort

testing postfix ++
All
Bar
Foo

Testing Prefix --:
Foo
Bar
All

Testing Prefix ++:
Bar
Foo
All

Testing Postfix --
All
Foo
Bar

Testing a <const char*> Vector:

Testing sort
Apple
Orange
Pear

Program Terminated Successfully.
```

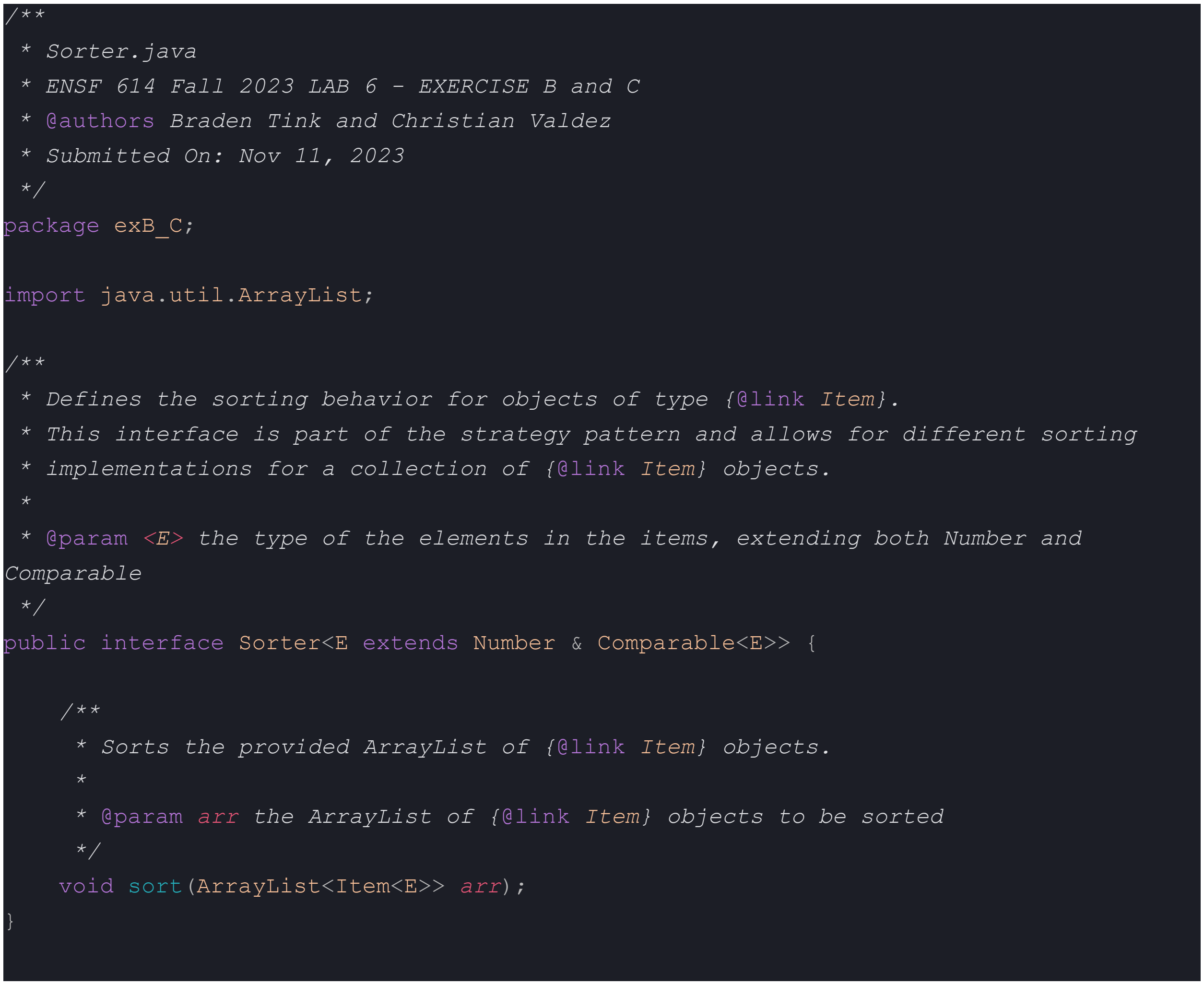
Exercise B and C

```java
/**
 * Item.java
 * ENSF 614 Fall 2023 LAB 6 - EXERCISE B and C
 * @authors Braden Tink and Christian Valdez
 * Submitted On: Nov 11, 2023
 */

package exB_C;

/**
 * Represents a generic item that holds a value of a type that extends both Number and
Comparable.
 * This class provides methods to set and get the value of the item.
 *
 * @param <E> the type of the value, which extends both Number and Comparable
 */
public class Item<E extends Number & Comparable<E>> {
    private E value; // Renamed 'item' to 'value' for better readability

    /**
     * Constructs an Item with the specified value.
     *
     * @param value the value to be stored in this item
     */
    public Item(E value) {
        this.value = value;
    }

    /**
     * Sets the value of this item.
     *
     * @param value the new value to be stored in this item
     */
    public void setItem(E value) {
        this.value = value;
    }

    /**
     * Returns the current value stored in this item.
     *
     * @return the current value
     */
    public E getItem() {
        return value;
    }
}
```

```java
/**
 * MyVector.java
 * ENSF 614 Fall 2023 LAB 6 - EXERCISE B and C
 * @authors Braden Tink and Christian Valdez
 * Submitted On: Nov 11, 2023
 */

package exB_C;
import java.util.ArrayList;

/**
 * Represents a generic vector that stores elements of type {@link Item}. It allows for sorting
 * the elements using different sorting strategies defined by the {@link Sorter} interface.
 *
 * @param <E> the type of the elements, extending both Number and Comparable
 */
public class MyVector<E extends Number & Comparable<E>>{
    private ArrayList<Item<E>> storageM;
    private Sorter<E> sorter;

    /**
     * Constructs a MyVector with an initial capacity.
     *
     * @param size the initial capacity of the vector
     */
    public MyVector(int size) {
        storageM = new ArrayList<>(size);
    }

    /**
     * Constructs a MyVector using elements from an existing ArrayList.
     *
     * @param arrList the ArrayList containing items to be added to the vector
     */
    public MyVector(ArrayList<Item<E>> arrList) {
        storageM = new ArrayList<>(arrList.size());
        for(Item<E> item : arrList) {
            storageM.add(item);
        }
    }
    /**
     * Adds an item to the vector.
     *
     * @param item the item to be added
     */
```

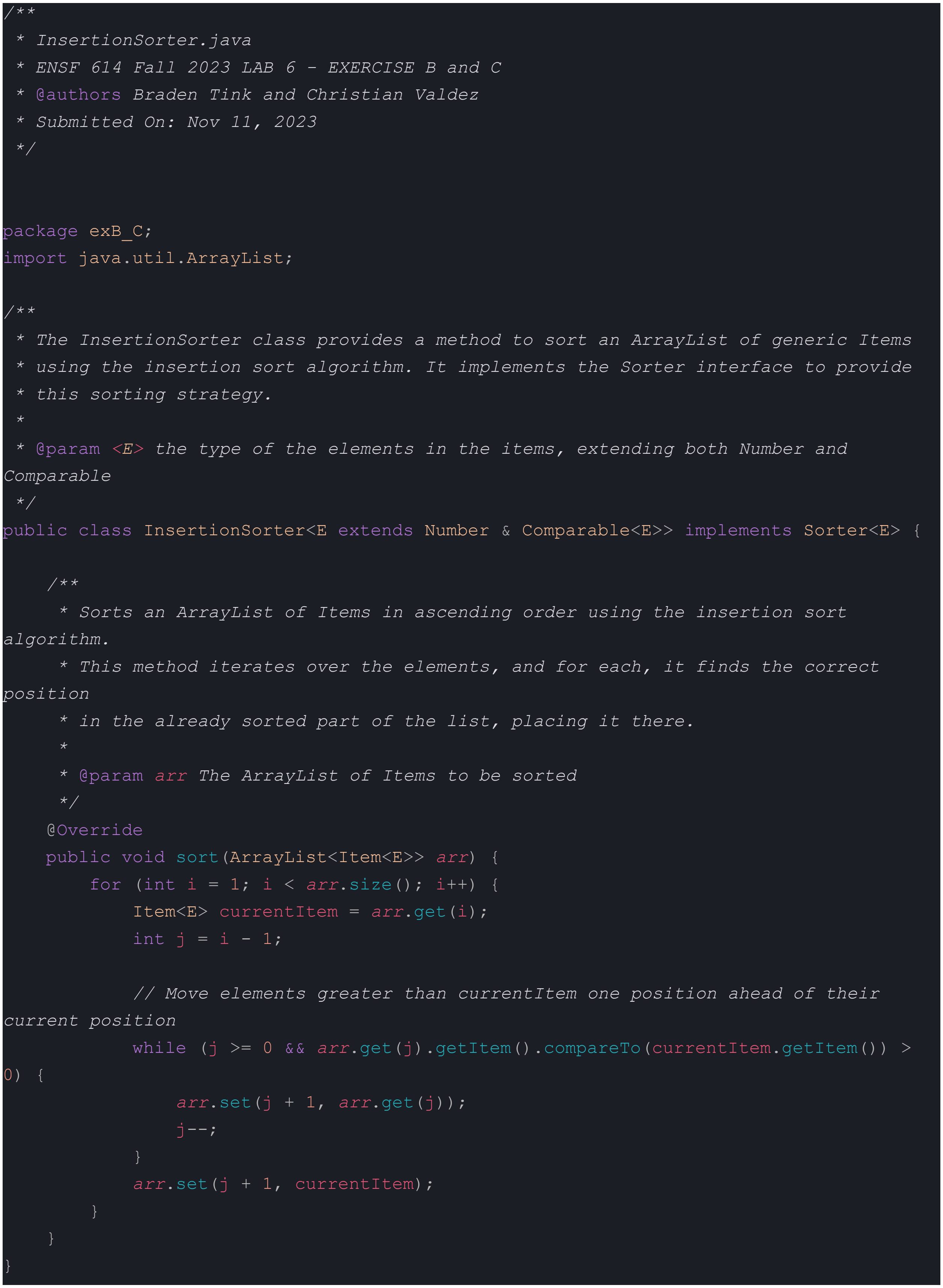


```java
    public void add(Item<E> item) {
        storageM.add(item);
    }

    /**
     * Sets the sorting strategy for the vector.
     *
     * @param s the sorting strategy to be used
     */
    public void setSortStrategy(Sorter<E> s) {
        sorter = s;
    }

    /**
     * Sorts the vector according to the currently set sorting strategy.
     */
    public void performSort() {
        if (sorter != null) {
            sorter.sort(storageM);
        }
    }

    /**
     * Displays the elements of the vector.
     */
    public void display() {
        for (Item<E> item : storageM) {
            System.out.print(item.getItem() + " ");
        }
        System.out.println();
    }
}
```



```java
/**
 * Sorter.java
 * ENSF 614 Fall 2023 LAB 6 - EXERCISE B and C
 * @authors Braden Tink and Christian Valdez
 * Submitted On: Nov 11, 2023
 */
package exB_C;

import java.util.ArrayList;

/**
 * Defines the sorting behavior for objects of type {@link Item}.
 * This interface is part of the strategy pattern and allows for different sorting
 * implementations for a collection of {@link Item} objects.
 *
 * @param <E> the type of the elements in the items, extending both Number and
Comparable
 */
public interface Sorter<E extends Number & Comparable<E>> {

    /**
     * Sorts the provided ArrayList of {@link Item} objects.
     *
     * @param arr the ArrayList of {@link Item} objects to be sorted
     */
    void sort(ArrayList<Item<E>> arr);
}
```

```java
/**
 * BubbleSorter.java
 * ENSF 614 Fall 2023 LAB 6 - EXERCISE B and C
 * @authors Braden Tink and Christian Valdez
 * Submitted On: Nov 11, 2023
 */


package exB_C;
import java.util.ArrayList;

/**
 * The BubbleSorter class provides a method to sort an array list of generic Items
 * using the bubble sort algorithm. It implements the Sorter interface to provide
 * this sorting strategy.
 *
 * @param <E> the type of the elements in the items, extending both Number and
Comparable
 */
public class BubbleSorter<E extends Number & Comparable<E>> implements Sorter<E> {

    /**
     * Sorts an array list of Items using the bubble sort algorithm. In each
iteration,
     * adjacent items are compared and swapped if they are in the wrong order, thus
     * "bubbling" the highest or lowest value to the top of the list, depending on the
     * order of sorting.
     *
     * @param arr the array list of Items to be sorted
     */
    @Override
    public void sort(ArrayList<Item<E>> arr) {
        int n = arr.size();
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                Item<E> currentItem = arr.get(j);
                Item<E> nextItem = arr.get(j + 1);

                // Swap items if they are in the wrong order
                if (currentItem.getItem().compareTo(nextItem.getItem()) > 0) {
                    arr.set(j, nextItem);
                    arr.set(j + 1, currentItem);
                }
            }
        }
    }
}
```

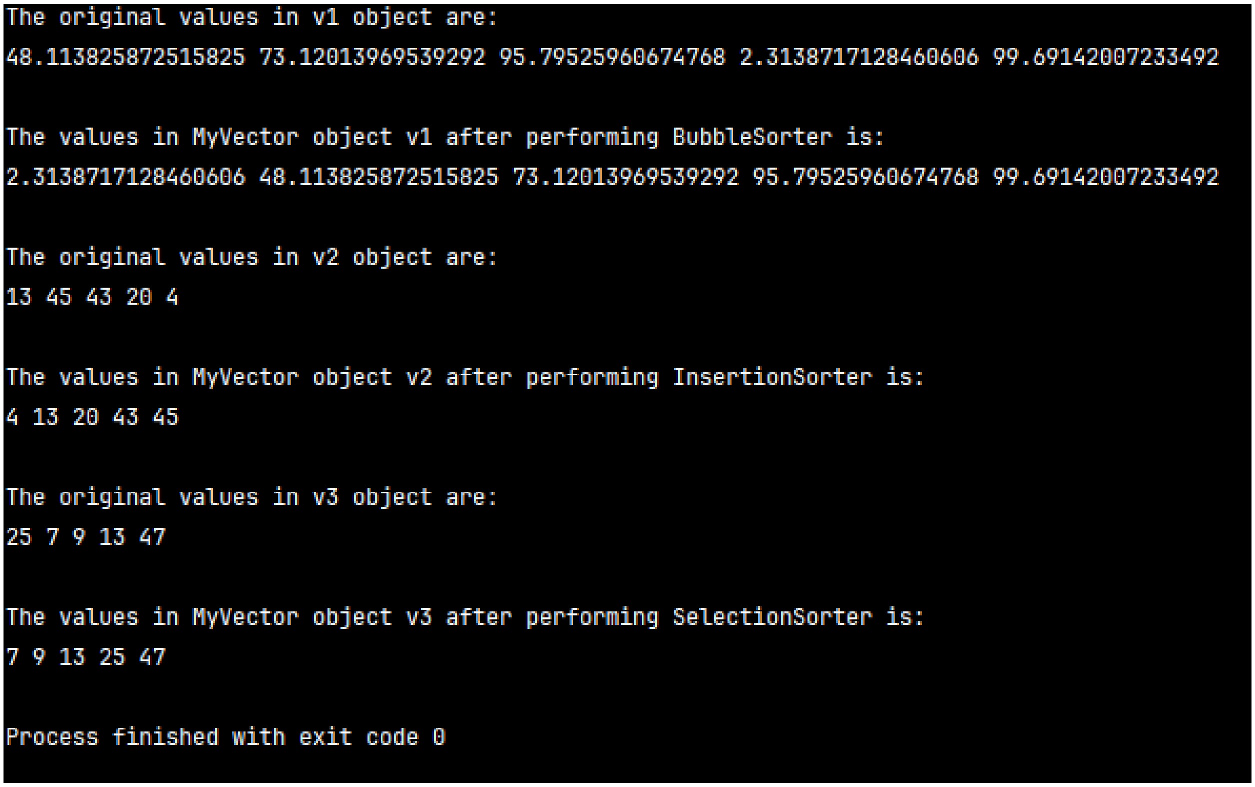


```java
/**
 * InsertionSorter.java
 * ENSF 614 Fall 2023 LAB 6 - EXERCISE B and C
 * @authors Braden Tink and Christian Valdez
 * Submitted On: Nov 11, 2023
 */


package exB_C;
import java.util.ArrayList;

/**
 * The InsertionSorter class provides a method to sort an ArrayList of generic Items
 * using the insertion sort algorithm. It implements the Sorter interface to provide
 * this sorting strategy.
 *
 * @param <E> the type of the elements in the items, extending both Number and
Comparable
 */
public class InsertionSorter<E extends Number & Comparable<E>> implements Sorter<E> {

    /**
     * Sorts an ArrayList of Items in ascending order using the insertion sort
algorithm.
     * This method iterates over the elements, and for each, it finds the correct
position
     * in the already sorted part of the list, placing it there.
     *
     * @param arr The ArrayList of Items to be sorted
     */
    @Override
    public void sort(ArrayList<Item<E>> arr) {
        for (int i = 1; i < arr.size(); i++) {
            Item<E> currentItem = arr.get(i);
            int j = i - 1;

            // Move elements greater than currentItem one position ahead of their
current position
            while (j >= 0 && arr.get(j).getItem().compareTo(currentItem.getItem()) >
0) {
                arr.set(j + 1, arr.get(j));
                j--;
            }
            arr.set(j + 1, currentItem);
        }
    }
}
```

```
/**
 * SelectionSorter.java
 * ENSF 614 Fall 2023 LAB 6 - EXERCISE B and C
 * @authors Braden Tink and Christian Valdez
 * Submitted On: Nov 11, 2023
 */

package exB_C;
import java.util.ArrayList;

/**
 * The SelectionSorter class provides a method to sort an array list of generic Items
 * using the selection sort algorithm. It implements the Sorter interface to provide
 * this sorting strategy.
 *
 * @param <E> the type of the elements in the items, extending both Number and
Comparable
 */
public class SelectionSorter<E extends Number & Comparable<E>> implements Sorter<E> {

    /**
     * Sorts an array list of Items using the selection sort algorithm.
     * This method iterates over the array list, finds the minimum element from the
unsorted
     * part, and places it at the beginning. The process is repeated until the whole
array is sorted.
     *
     * @param arr the array list of Items to be sorted
     */
    public void sort(ArrayList<Item<E>> arr) {
        int n = arr.size();

        for (int i = 0; i < n - 1; i++) {
            // Find the minimum element in the unsorted part of the array
            int minIdx = i;
            for (int j = i + 1; j < n; j++) {
                E current = arr.get(j).getItem();
                E min = arr.get(minIdx).getItem();

                // Compare current element with the minimum found so far
                if (current.compareTo(min) < 0) {
                    minIdx = j;
                }
            }

            // Swap the found minimum element with the first element
            Item<E> temp = arr.get(minIdx);
            arr.set(minIdx, arr.get(i));
            arr.set(i, temp);
        }
    }
}
```

```
/**
 * DemoStrategy.java
 * ENSF 614 Fall 2023 LAB 6 - EXERCISE B and C
 * @authors Braden Tink and Christian Valdez
 * Submitted On: Nov 11, 2023
 */


package exB_C;
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
BubbleSorter 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 an object of MyVector<Double> with capacity of 50 elements
        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 BubbleSorter<Integer>());
        v3.performSort();
        System.out.println("\nThe values in MyVector object v3 after performing 
SelectionSorter is:");
        v3.display();
    }
}
```

Sample Output



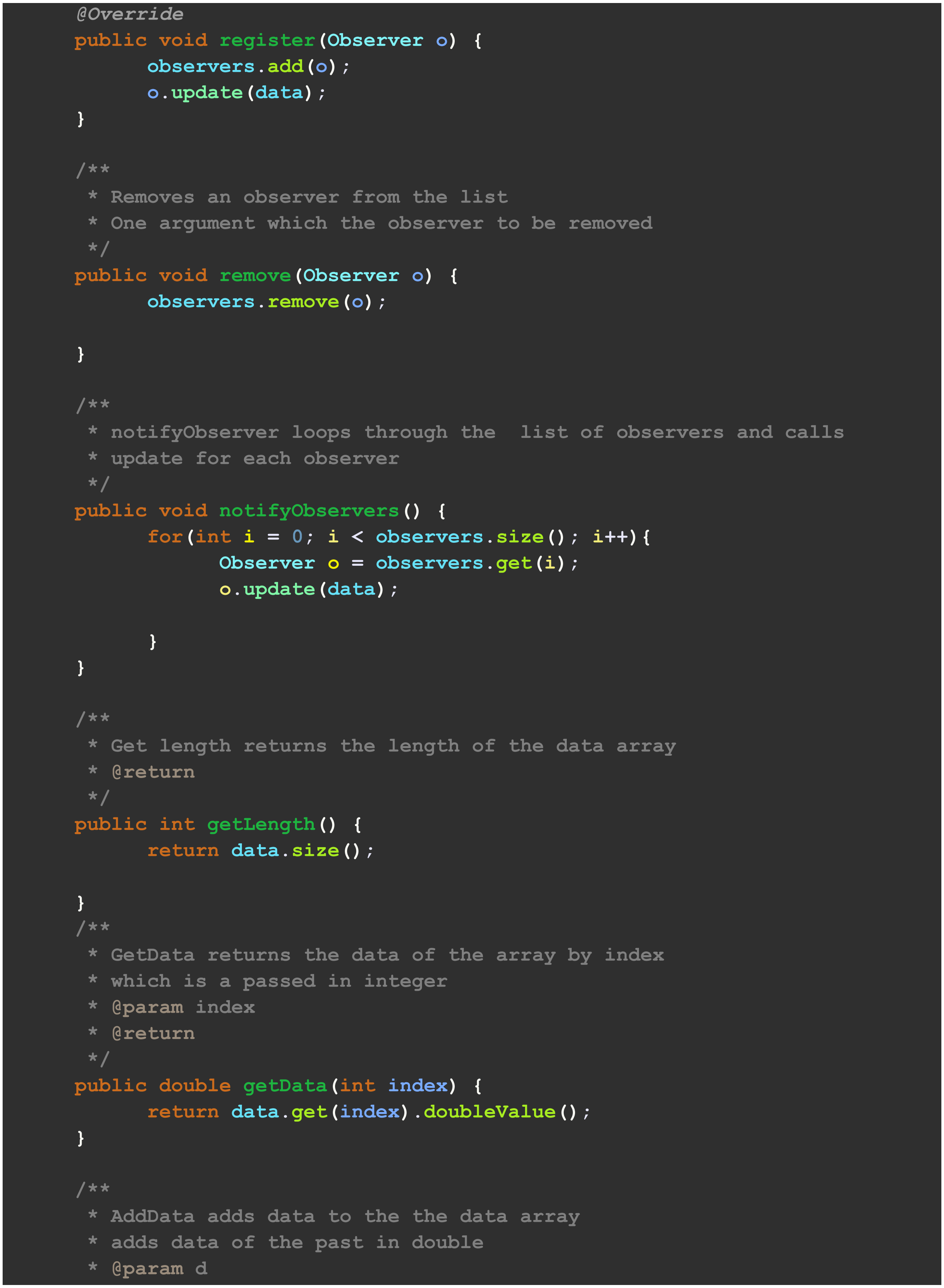
```
The original values in v1 object are:
48.113825872515825 73.12013969539292 95.79525960674768 2.3138717128460606 99.69142007233492

The values in MyVector object v1 after performing BubbleSorter is:
2.3138717128460606 48.113825872515825 73.12013969539292 95.79525960674768 99.69142007233492

The original values in v2 object are:
13 45 43 20 4

The values in MyVector object v2 after performing InsertionSorter is:
4 13 20 43 45

The original values in v3 object are:
25 7 9 13 47

The values in MyVector object v3 after performing SelectionSorter is:
7 9 13 25 47

Process finished with exit code 0
```

Exercise D

Sample Code

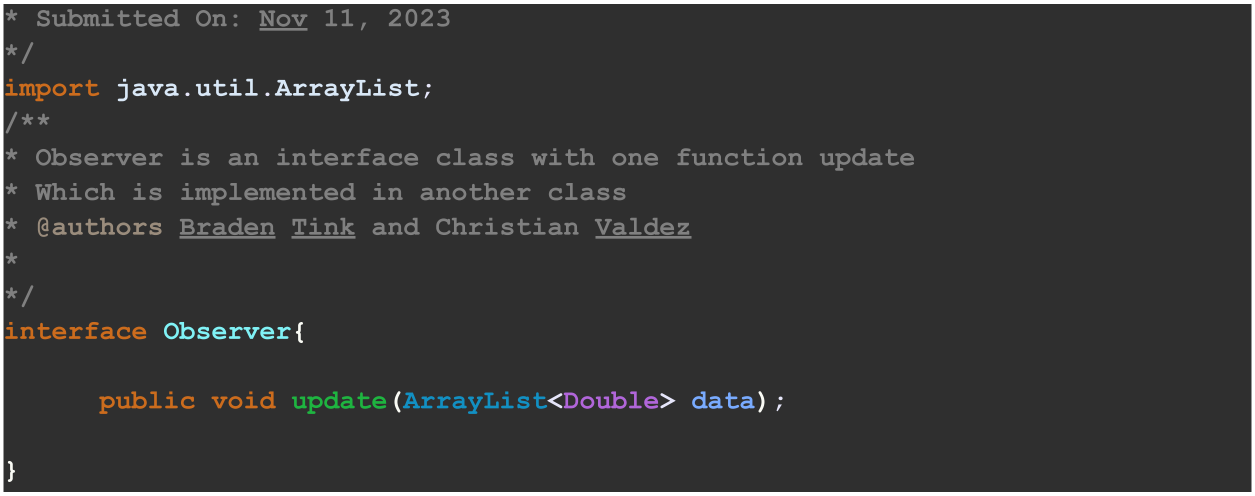
```
/**
* ObserverPattern.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE D
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
/**
* ObserverPatternController has one function being the main function
* @authors Braden Tink and Christian Valdez
*
*/
public class ObserverPatternController {
      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);


      }
}
```

```
/**
* DoubleArrayListSubject.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE D
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.util.ArrayList;
/**
* DoubleArrayListSubject class implements Subject
* @authors Braden Tink and Christian Valdez
* Class has three class variables
*
*/
public class DoubleArrayListSubject implements Subject{
      private double temp;
      public ArrayList<Observer> observers;
      ArrayList<Double> data;

      /**
       * constructor creates a new ArrayList for observers and
       * data as another
       */
      public DoubleArrayListSubject() {
            observers = new ArrayList<Observer>();
            data = new ArrayList<Double>();
      }

      /**
       * Register function registers a observer to the list
       * Take one argument being an observer
       */
```

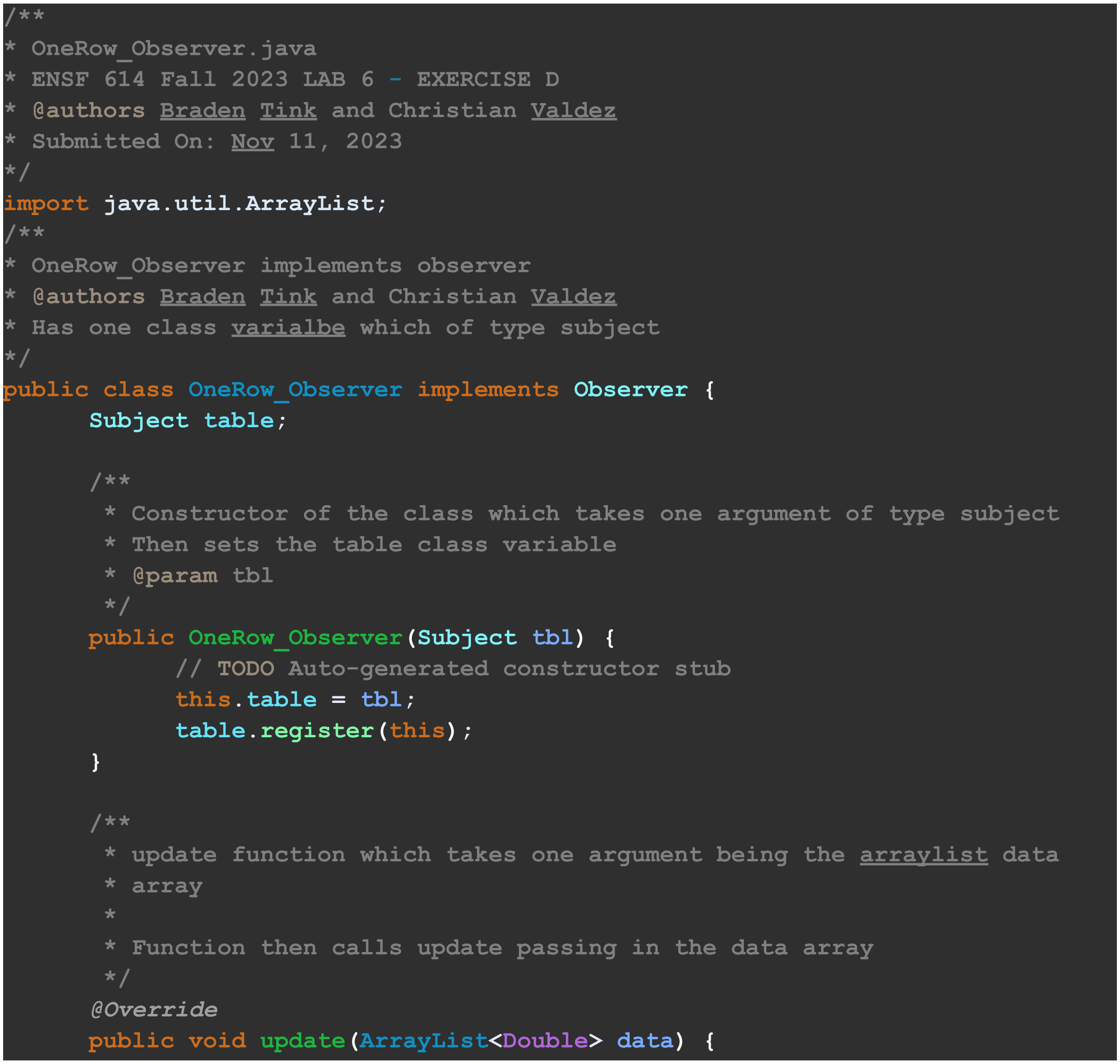


```
    @Override
    public void register(Observer o) {
          observers.add(o);
          o.update(data);
    }

    /**
     * Removes an observer from the list
     * One argument which the observer to be removed
     */
    public void remove(Observer o) {
          observers.remove(o);

    }

    /**
     * notifyObserver loops through the  list of observers and calls
     * update for each observer
     */
    public void notifyObservers() {
          for(int i = 0; i < observers.size(); i++){
                Observer o = observers.get(i);
                o.update(data);

          }
    }

    /**
     * Get length returns the length of the data array
     * @return
     */
    public int getLength() {
          return data.size();

    }
    /**
     * GetData returns the data of the array by index
     * which is a passed in integer
     * @param index
     * @return
     */
    public double getData(int index) {
          return data.get(index).doubleValue();
    }

    /**
     * AddData adds data to the the data array
     * adds data of the past in double
     * @param d
     */
    public void addData(Double d) {
        data.add(d);
        notifyObservers();

    }

    /**
     * setData sets the value of the data array
     * the index and data value are passed in as
     * arguments
     * @param d
     * @param pos
     */
    public void setData(Double d, int pos) {
        data.set(pos, d);
        notifyObservers();
    }

    /**
     * populate function adds an array of doubles to the data
     * array by calling the addData function with the index
     * @param d
     */
    public void populate(double[] d) {
        int n = d.length;

        for(int i = 0; i < n; i++) {

            addData(d[i]);

        }

    }

    /**
     * display is function is a non implemented as it is implemented by the observers
     */
    public void display() {


    }
}
```

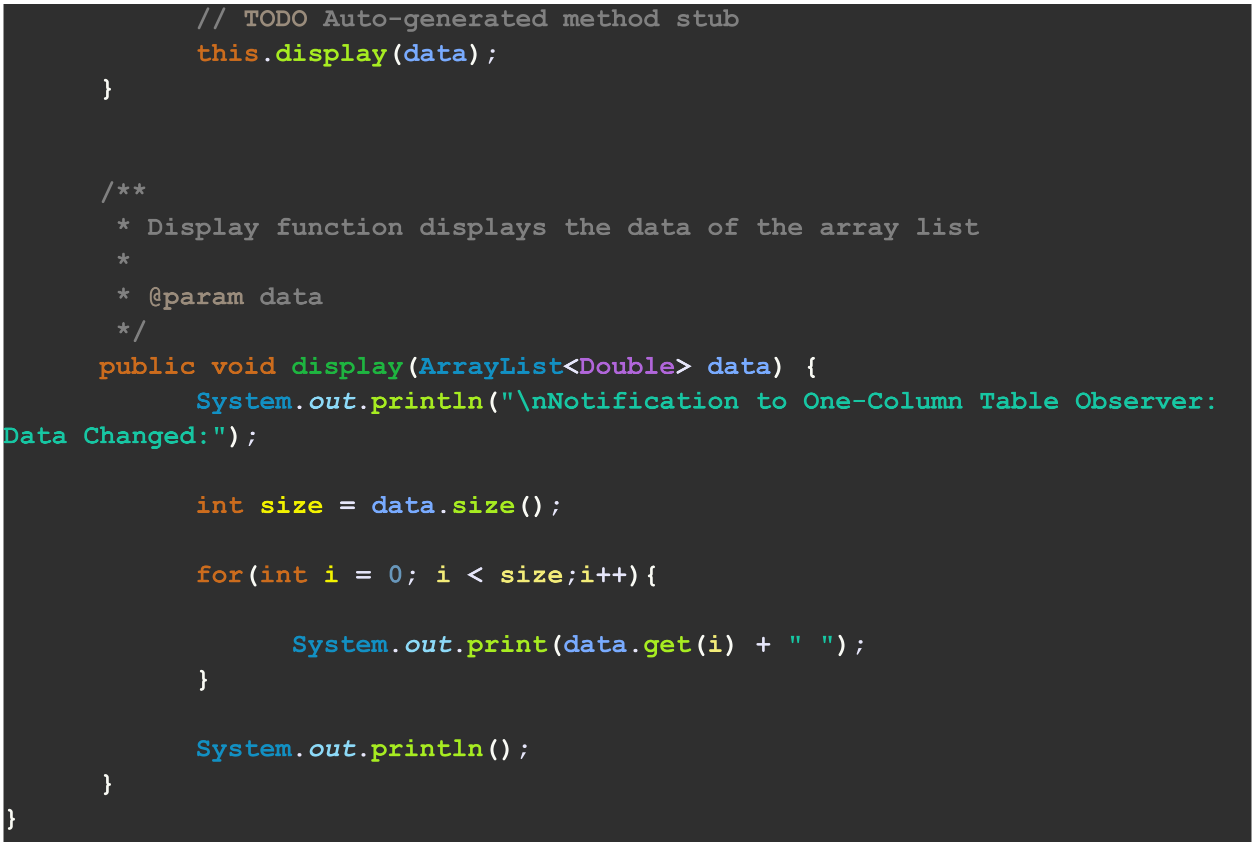
```
/**
* Observer.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE D
* @authors Braden Tink and Christian Valdez
```



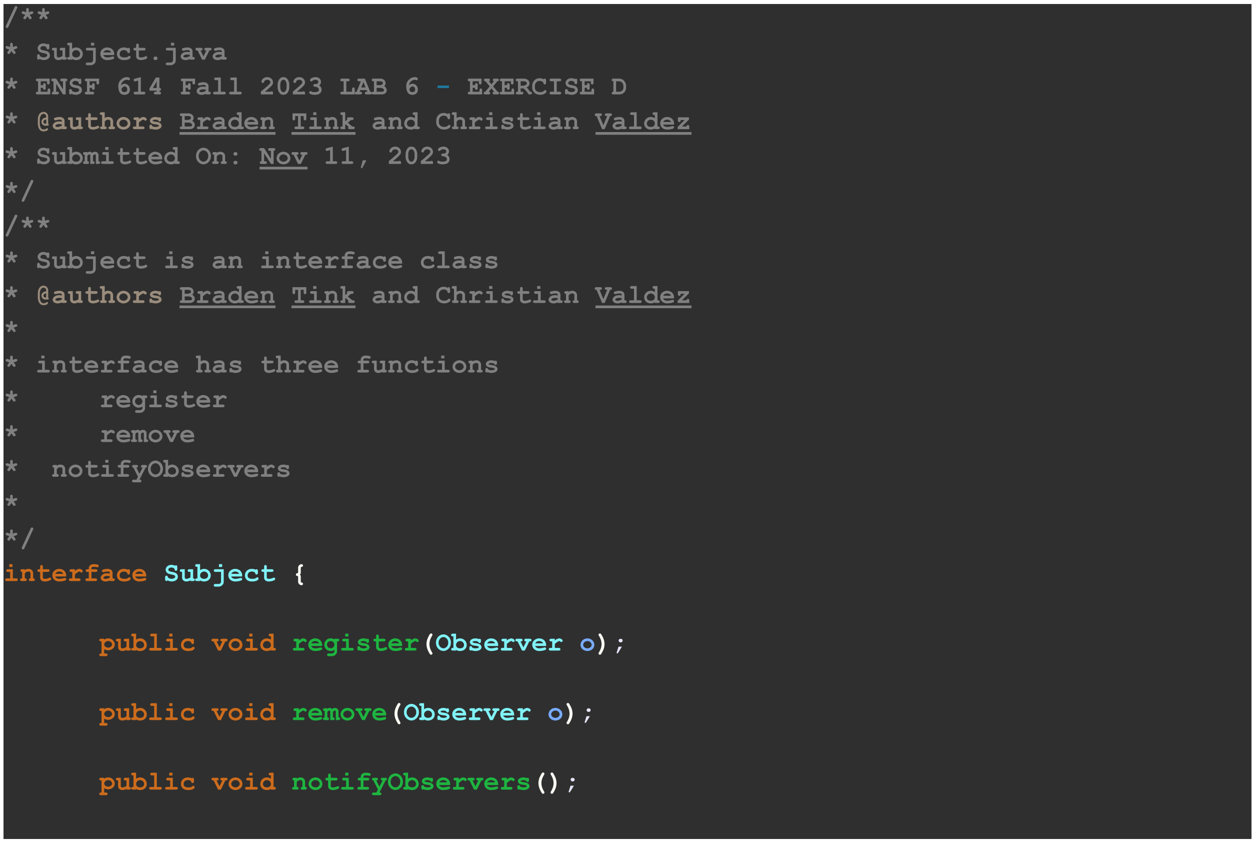
```java
* Submitted On: Nov 11, 2023
*/
import java.util.ArrayList;
/**
* Observer is an interface class with one function update
* Which is implemented in another class
* @authors Braden Tink and Christian Valdez
*
*/
interface Observer{

    public void update(ArrayList<Double> data);

}
```



```java
/**
* OneRow_Observer.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE D
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.util.ArrayList;
/**
* OneRow_Observer implements observer
* @authors Braden Tink and Christian Valdez
* Has one class varialbe which of type subject
*/
public class OneRow_Observer implements Observer {
    Subject table;

    /**
     * Constructor of the class which takes one argument of type subject
     * Then sets the table class variable
     * @param tbl
     */
    public OneRow_Observer(Subject tbl) {
        // TODO Auto-generated constructor stub
        this.table = tbl;
        table.register(this);
    }

    /**
     * update function which takes one argument being the arraylist data
     * array
     *
     * Function then calls update passing in the data array
     */
    @Override
    public void update(ArrayList<Double> data) {
```

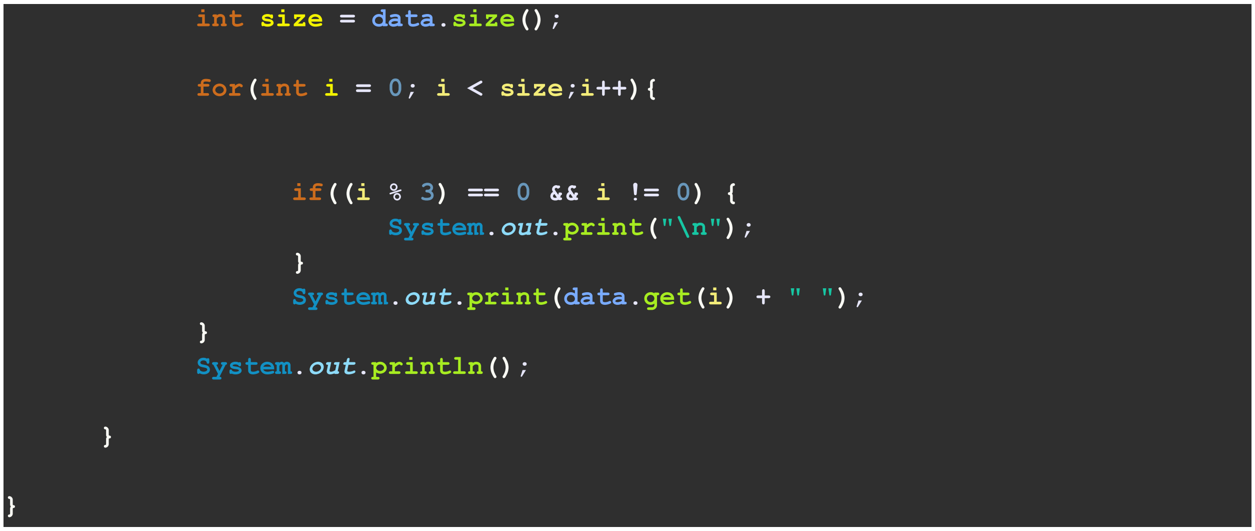


```java
            // TODO Auto-generated method stub
            this.display(data);
      }


      /**
       * Display function displays the data of the array list
       *
       * @param data
       */
      public void display(ArrayList<Double> data) {
            System.out.println("\nNotification to One-Column Table Observer: Data Changed:");

            int size = data.size();

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

                  System.out.print(data.get(i) + " ");
            }

            System.out.println();
      }
}
```

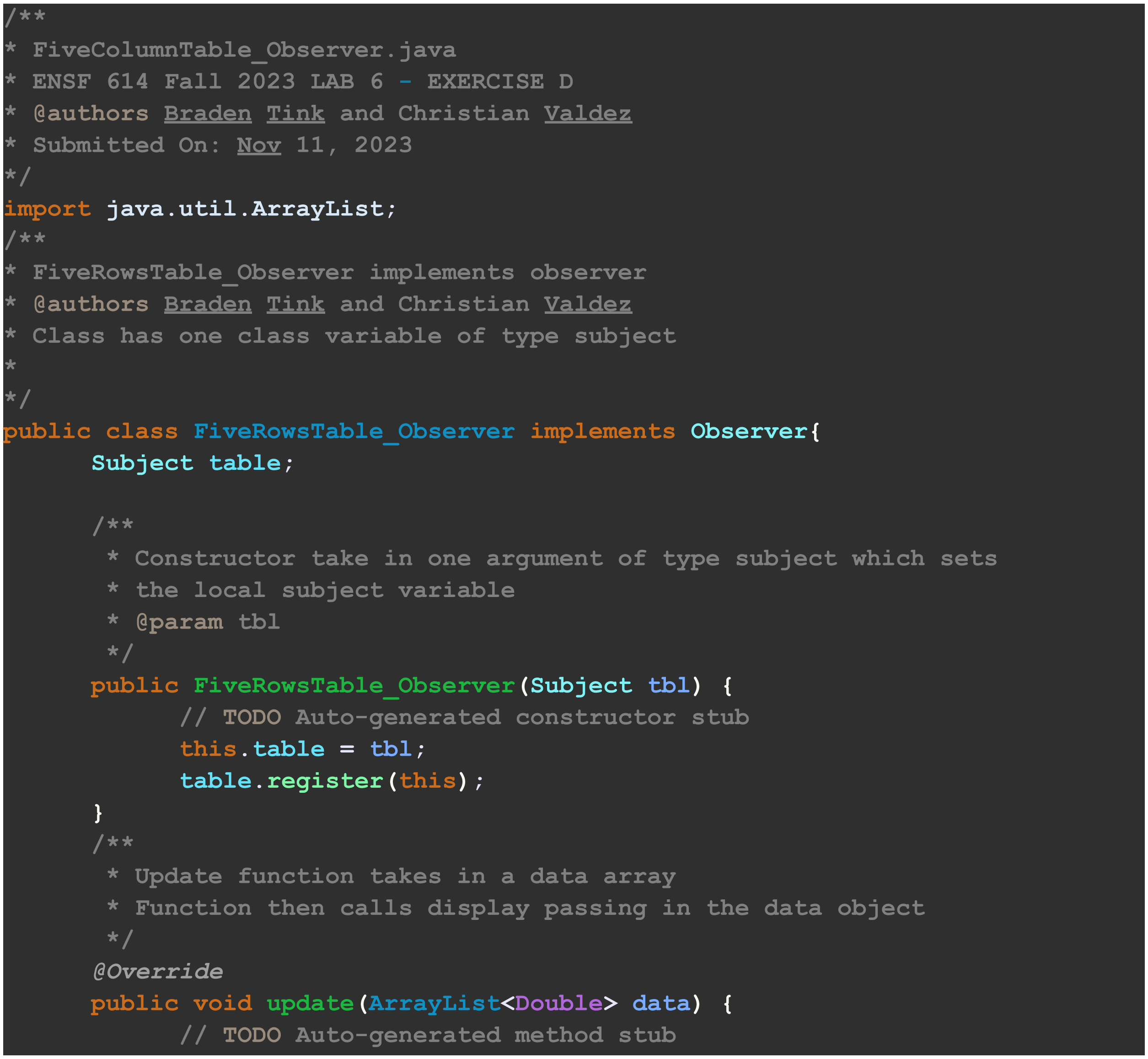


```java
/**
* Subject.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE D
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
/**
* Subject is an interface class
* @authors Braden Tink and Christian Valdez
*
* interface has three functions
*     register
*     remove
*  notifyObservers
*
*/
interface Subject {

      public void register(Observer o);

      public void remove(Observer o);

      public void notifyObservers();

}
```

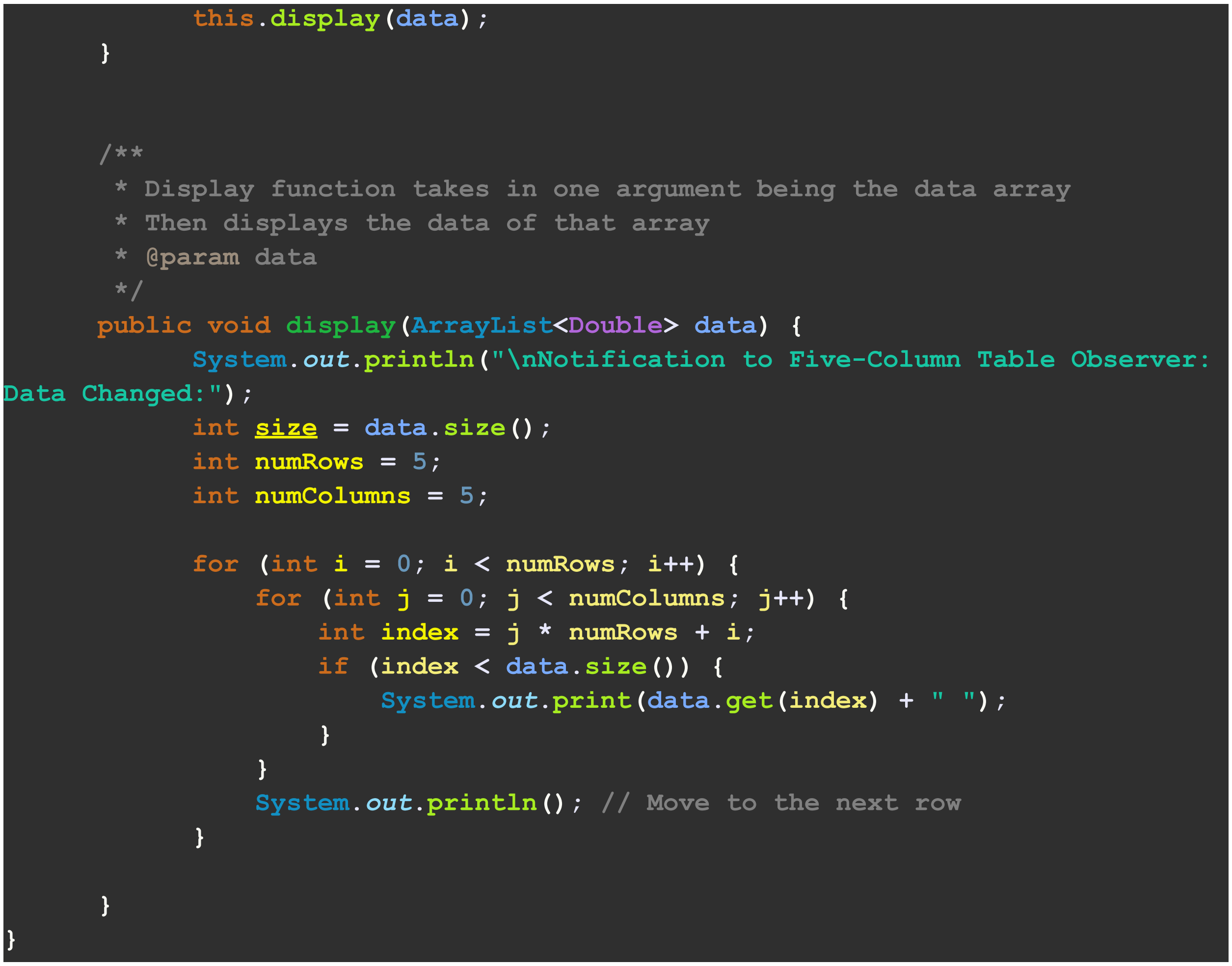
```
/**
* ThreeColumnTable_Observer.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE D
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.util.ArrayList;
/**
* ThreeColumnTable_Observer implements observer
* @authors Braden Tink and Christian Valdez
* Class has one class variable of type subject
*
*/
public class ThreeColumnTable_Observer implements Observer {
	Subject table;

	/**
	 * Constructor take in one argument of type subject which sets
	 * the local subject variable
	 * @param tbl
	 */
	public ThreeColumnTable_Observer(Subject tbl) {
		// TODO Auto-generated constructor stub
		this.table = tbl;
		table.register(this);
	}

	/**
	 * Update function takes in a data array
	 * Function then calls display passing in the data object
	 */
	@Override
	public void update(ArrayList<Double> data) {
		// TODO Auto-generated method stub
		this.display(data);
	}


	/**
	 * Display function takes in one argument being the data array
	 * Then displays the data of that array
	 * @param data
	 */
	public void display(ArrayList<Double> data) {
		System.out.println("\nNotification to Three-Column Table Observer: Data Changed:");
```



```
        int size = data.size();

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


            if((i % 3) == 0 && i != 0) {
                System.out.print("\n");
            }
            System.out.print(data.get(i) + " ");
        }
        System.out.println();

    }

}
```

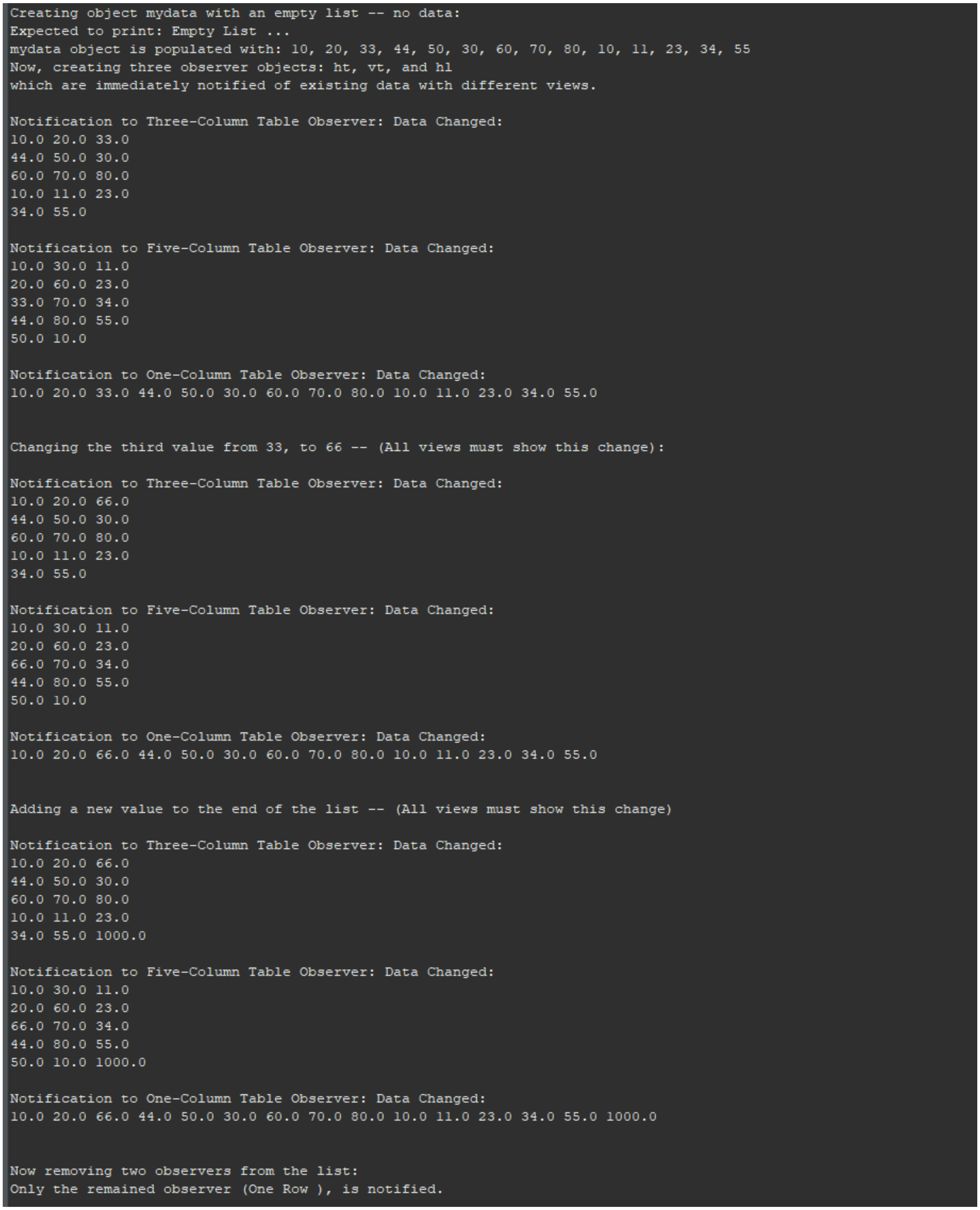


```
/**
* FiveColumnTable_Observer.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE D
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.util.ArrayList;
/**
* FiveRowsTable_Observer implements observer
* @authors Braden Tink and Christian Valdez
* Class has one class variable of type subject
*
*/
public class FiveRowsTable_Observer implements Observer{
    Subject table;

    /**
     * Constructor take in one argument of type subject which sets
     * the local subject variable
     * @param tbl
     */
    public FiveRowsTable_Observer(Subject tbl) {
        // TODO Auto-generated constructor stub
        this.table = tbl;
        table.register(this);
    }
    /**
     * Update function takes in a data array
     * Function then calls display passing in the data object
     */
    @Override
    public void update(ArrayList<Double> data) {
        // TODO Auto-generated method stub
```

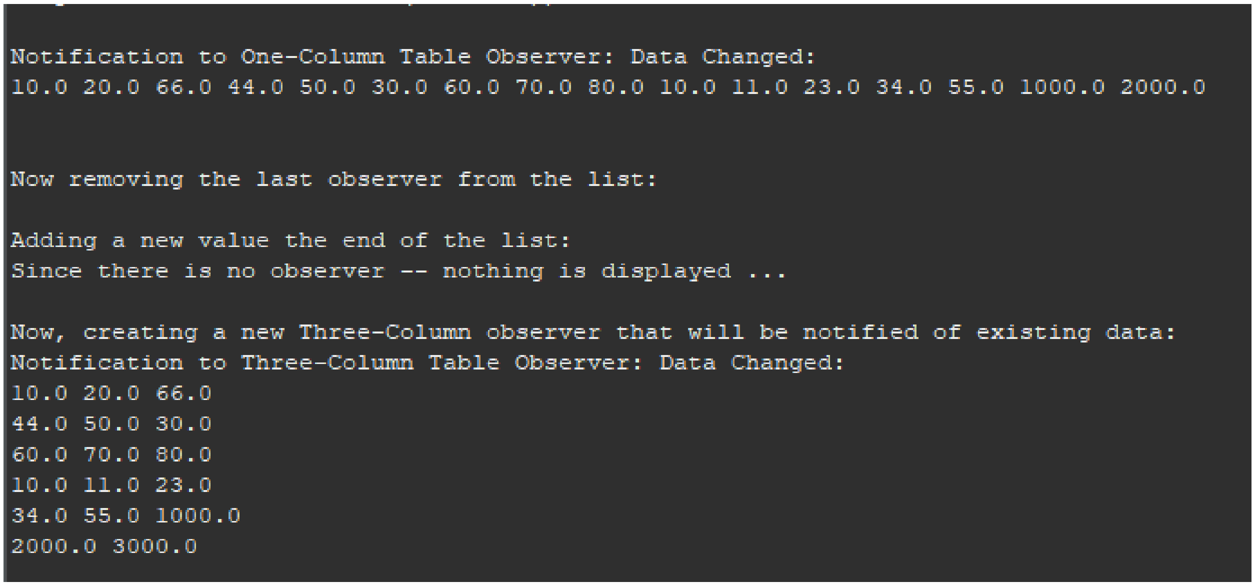


```
            this.display(data);
      }


      /**
       * Display function takes in one argument being the data array
       * Then displays the data of that array
       * @param data
       */
      public void display(ArrayList<Double> data) {
            System.out.println("\nNotification to Five-Column Table Observer:
Data Changed:");
            int size = data.size();
            int numRows = 5;
            int numColumns = 5;

            for (int i = 0; i < numRows; i++) {
                for (int j = 0; j < numColumns; j++) {
                    int index = j * numRows + i;
                    if (index < data.size()) {
                        System.out.print(data.get(index) + " ");
                    }
                }
                System.out.println(); // Move to the next row
            }

      }
}
```

Sample Output



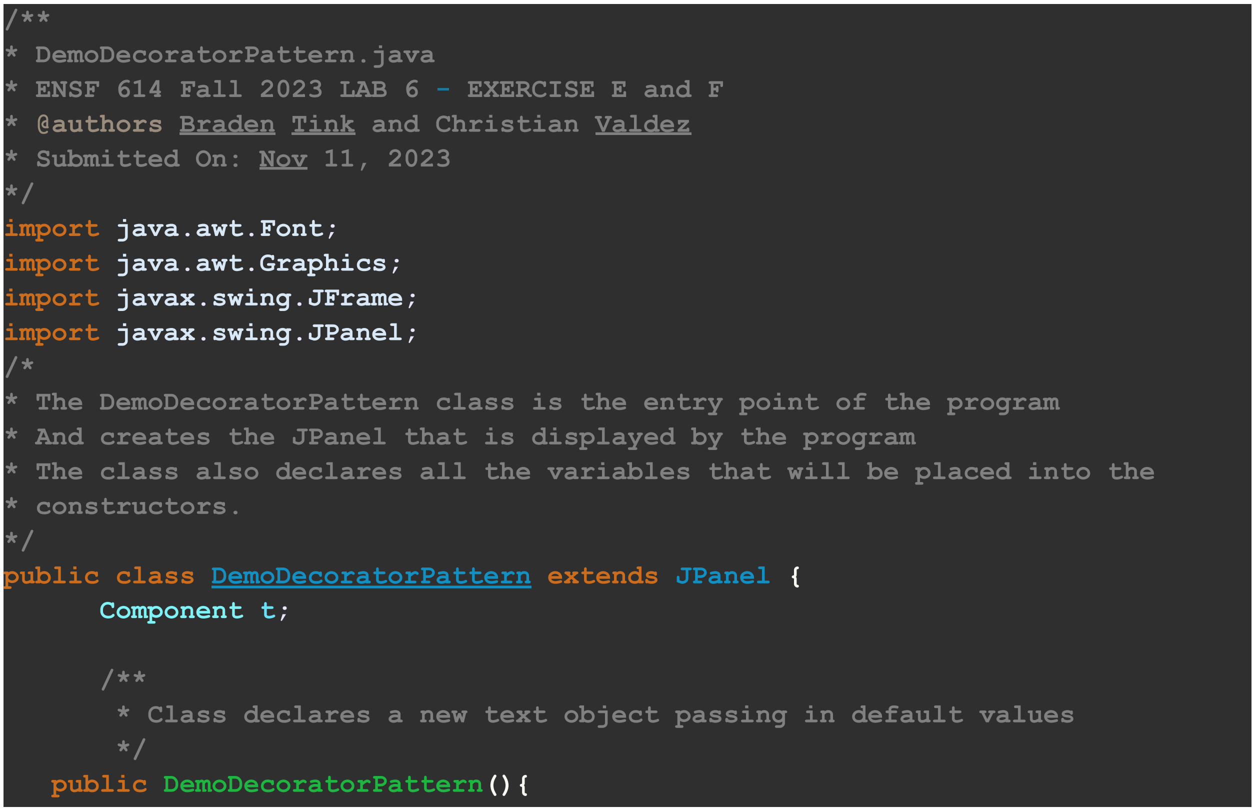
```
Creating object mydata with an empty list -- no data:
Expected to print: Empty List ...
mydata object is populated with: 10, 20, 33, 44, 50, 30, 60, 70, 80, 10, 11, 23, 34, 55
Now, creating three observer objects: ht, vt, and hl
which are immediately notified of existing data with different views.

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-Column 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-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


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-Column 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-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


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 Five-Column 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 1000.0

Notification to One-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


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



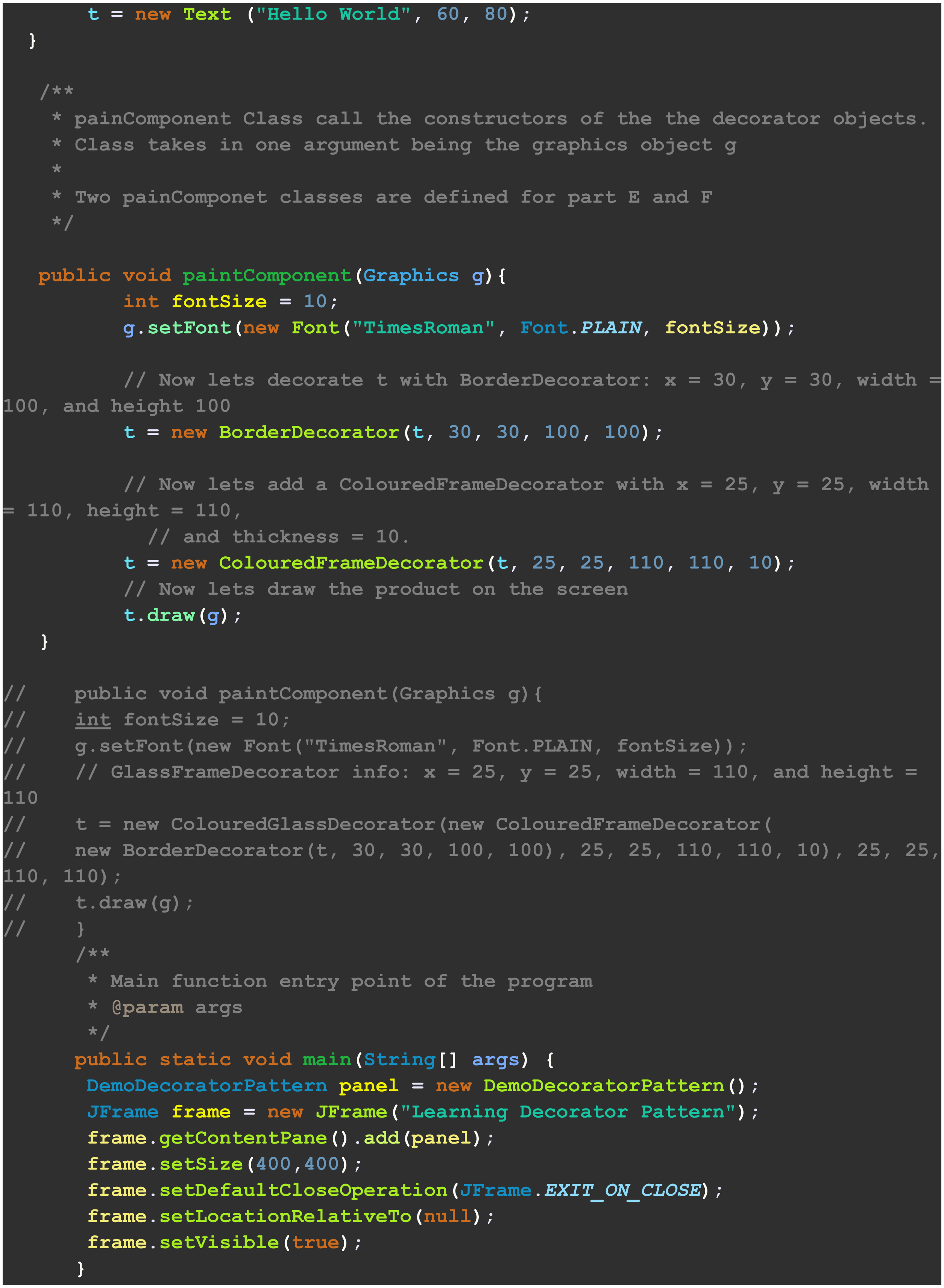
```
Notification to One-Column Table Observer: Data Changed:
10.0 20.0 66.0 44.0 50.0 30.0 60.0 70.0 80.0 10.0 11.0 23.0 34.0 55.0 1000.0 2000.0


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
```

Exercise E

Sample Code



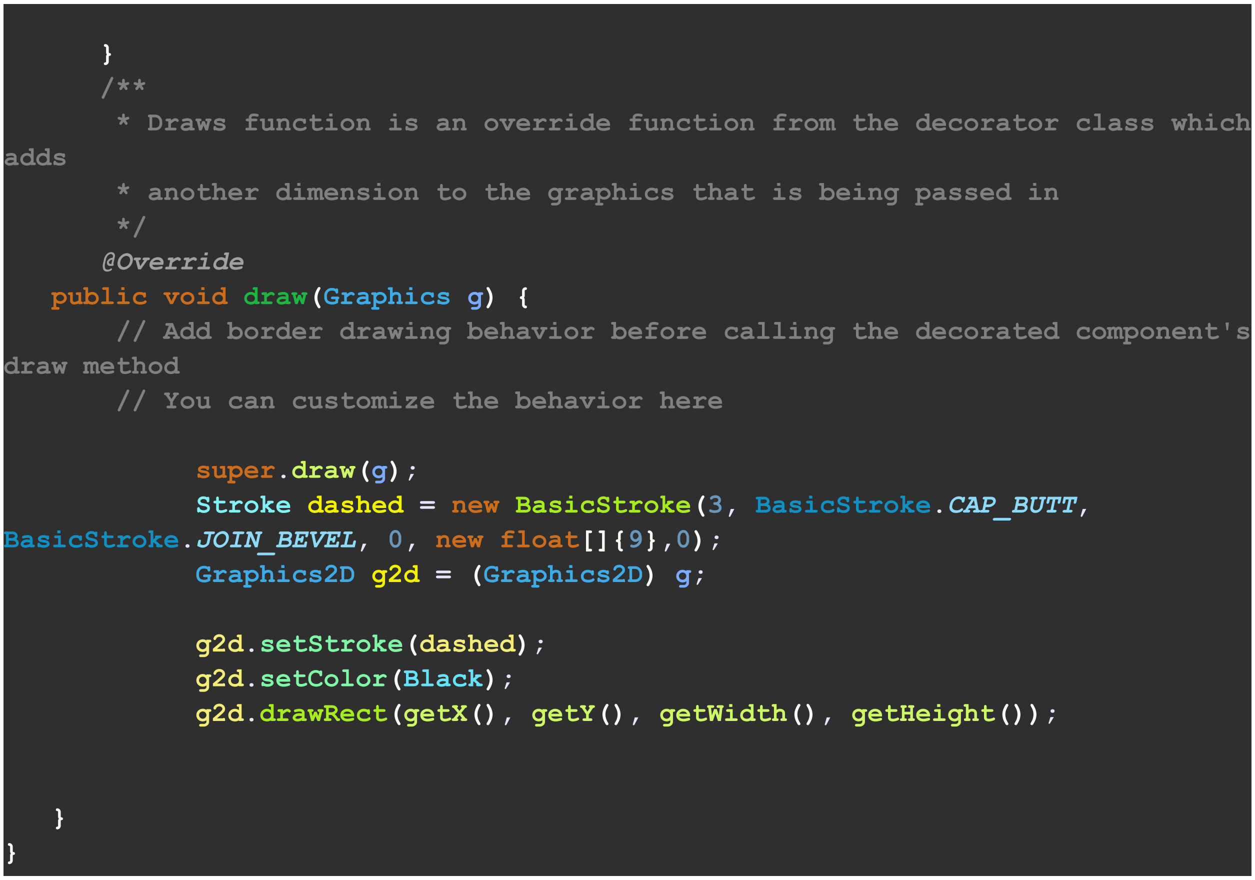
```
/**
* DemoDecoratorPattern.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE E and F
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.awt.Font;
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JPanel;
/*
* The DemoDecoratorPattern class is the entry point of the program
* And creates the JPanel that is displayed by the program
* The class also declares all the variables that will be placed into the
* constructors.
*/
public class DemoDecoratorPattern extends JPanel {
      Component t;

      /**
       * Class declares a new text object passing in default values
       */
   public DemoDecoratorPattern(){
```



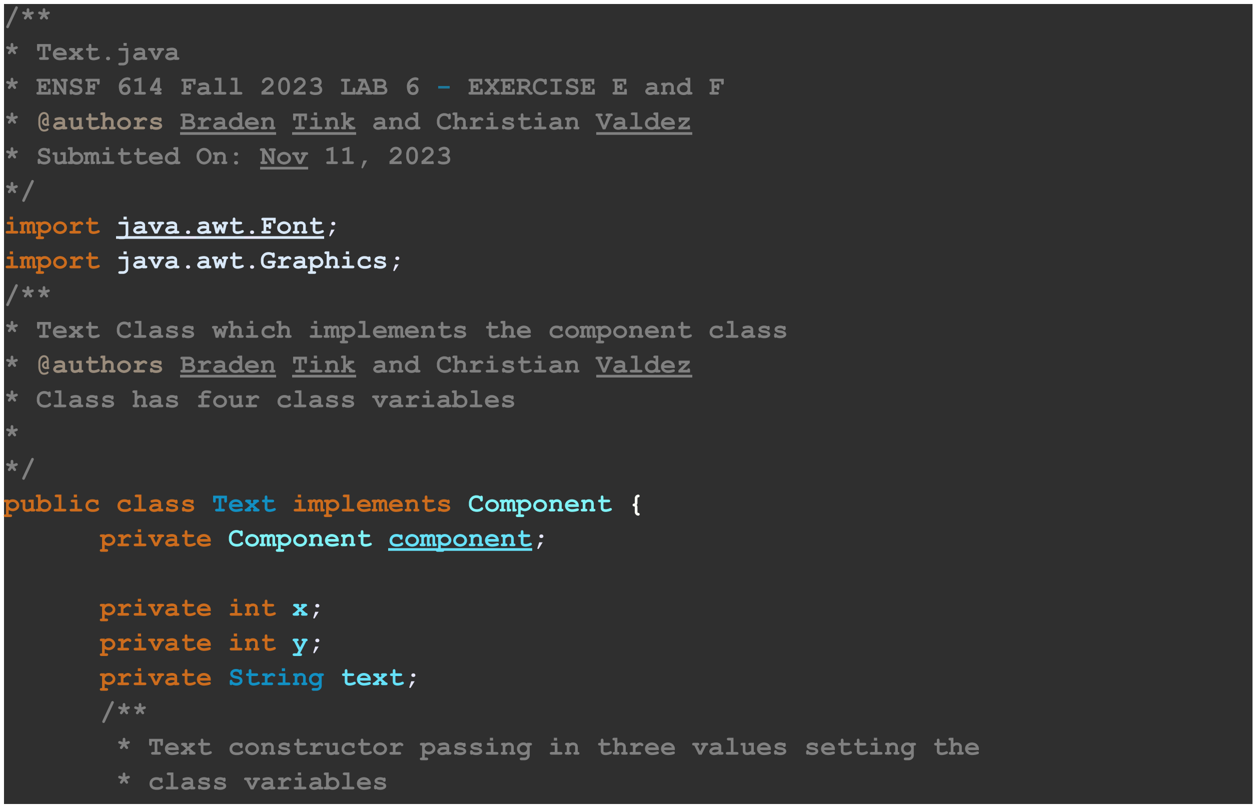
```
       t = new Text ("Hello World", 60, 80);
  }

   /**
    * painComponent Class call the constructors of the the decorator objects.
    * Class takes in one argument being the graphics object g
    *
    * Two painComponet classes are defined for part E and F
    */

   public void paintComponent(Graphics g){
          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);
   }

//    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);
//    }
      /**
       * Main function entry point of the program
       * @param args
       */
      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);
      }
```

```
}
```

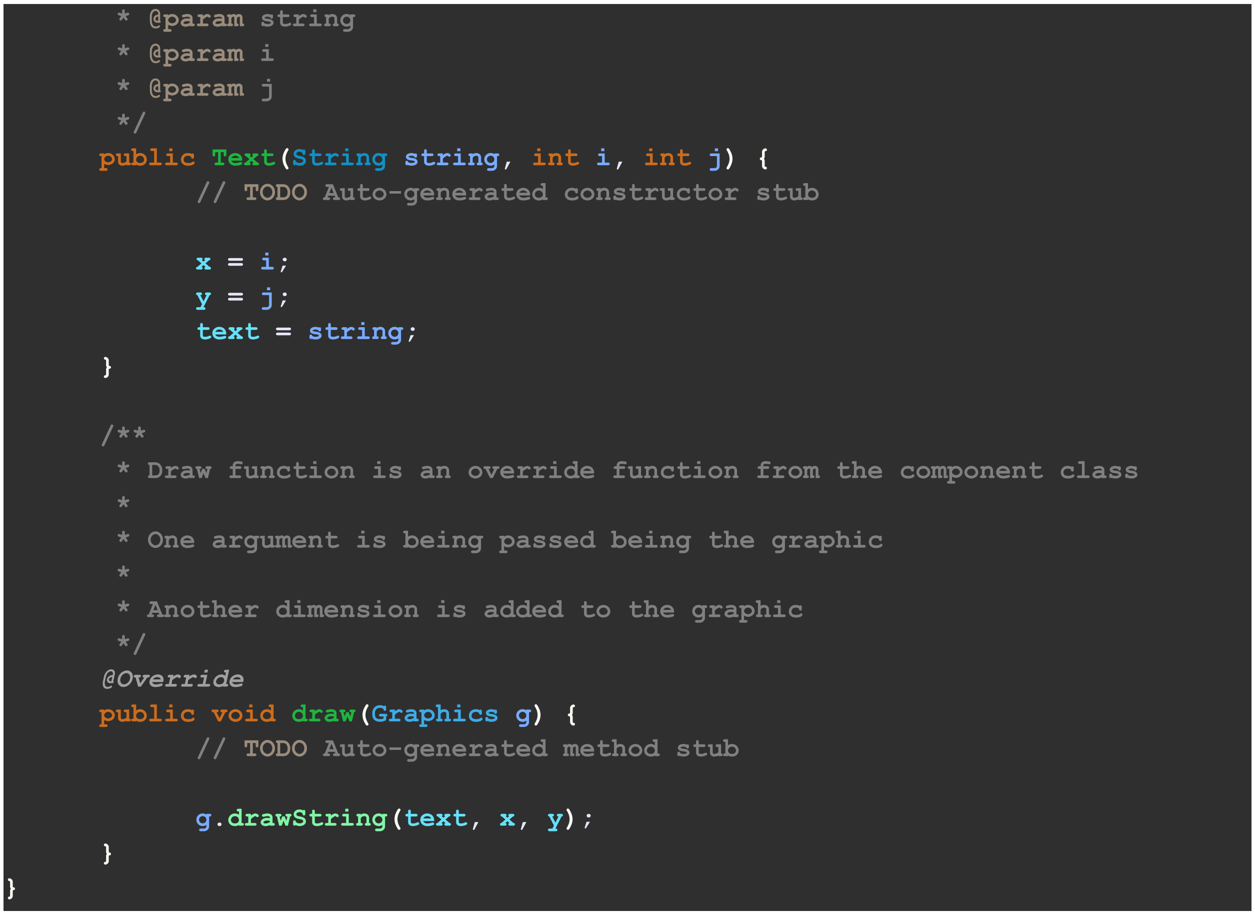
```
/**
* ColouredGlassDecorator.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE E and F
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.awt.Graphics;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Stroke;
/**
* BorderDecorator class extend the decorator class
* Class adds another dimension to the graphic
* @authors Braden Tink and Christian Valdez
* Class has one class variable which is the color black being used
* for the border color
*
*/
public class BorderDecorator extends Decorator{

   Color Black = Color.BLACK;

   /**
    * Constructor for the class passing in one argument being the
    * component which is passed to the super constructor
    * @param cmp
    */
   public BorderDecorator(Component cmp) {
      super(cmp);
   }

   /**
    * Constructor for the class passing in five value all being passed back to
the
    * super constructor
    * @param component
    * @param i
    * @param j
    * @param k
    * @param l
    */
      public BorderDecorator(Component component, int i, int j, int k, int l)
{
            // TODO Auto-generated constructor stub
            super(component, i, j, k, l);
```



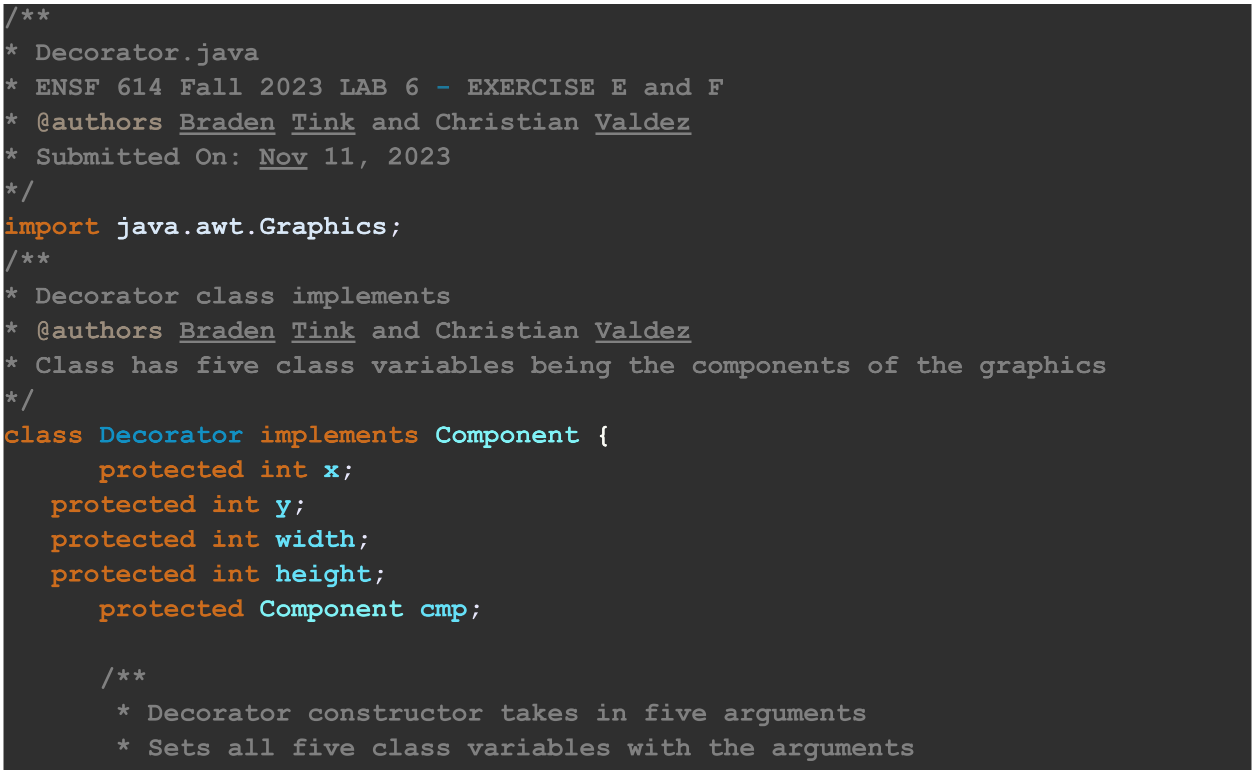
```
      }
      /**
       * Draws function is an override function from the decorator class which
adds
       * another dimension to the graphics that is being passed in
       */
      @Override
   public void draw(Graphics g) {
       // Add border drawing behavior before calling the decorated component's
draw method
       // You can customize the behavior here

            super.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);
            g2d.setColor(Black);
            g2d.drawRect(getX(), getY(), getWidth(), getHeight());


   }
}
```



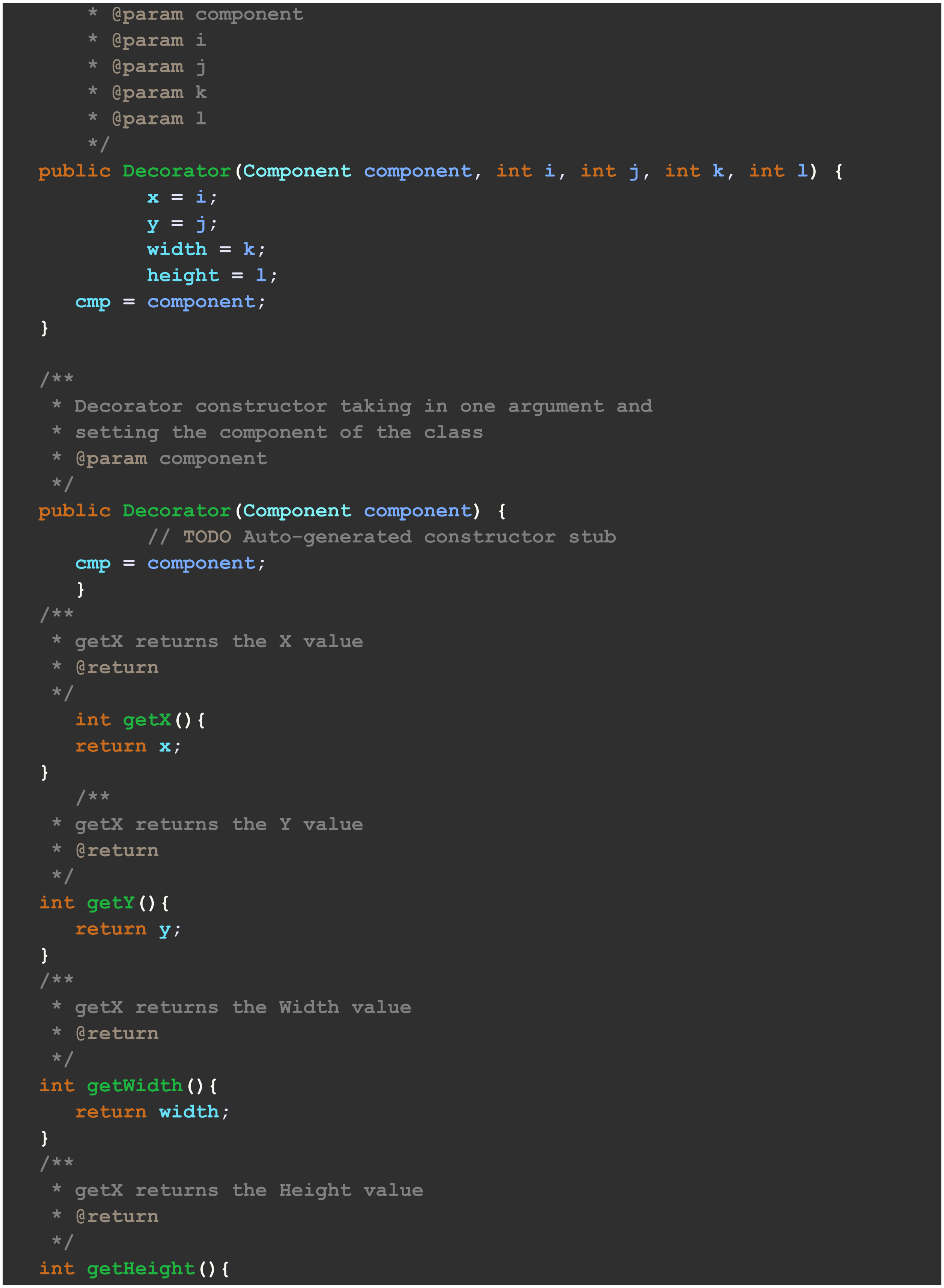
```
/**
* Text.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE E and F
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.awt.Font;
import java.awt.Graphics;
/**
* Text Class which implements the component class
* @authors Braden Tink and Christian Valdez
* Class has four class variables
*
*/
public class Text implements Component {
      private Component component;

      private int x;
      private int y;
      private String text;
      /**
       * Text constructor passing in three values setting the
       * class variables
```



```
       * @param string
       * @param i
       * @param j
       */
      public Text(String string, int i, int j) {
            // TODO Auto-generated constructor stub

            x = i;
            y = j;
            text = string;
      }

      /**
       * Draw function is an override function from the component class
       *
       * One argument is being passed being the graphic
       *
       * Another dimension is added to the graphic
       */
      @Override
      public void draw(Graphics g) {
            // TODO Auto-generated method stub

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



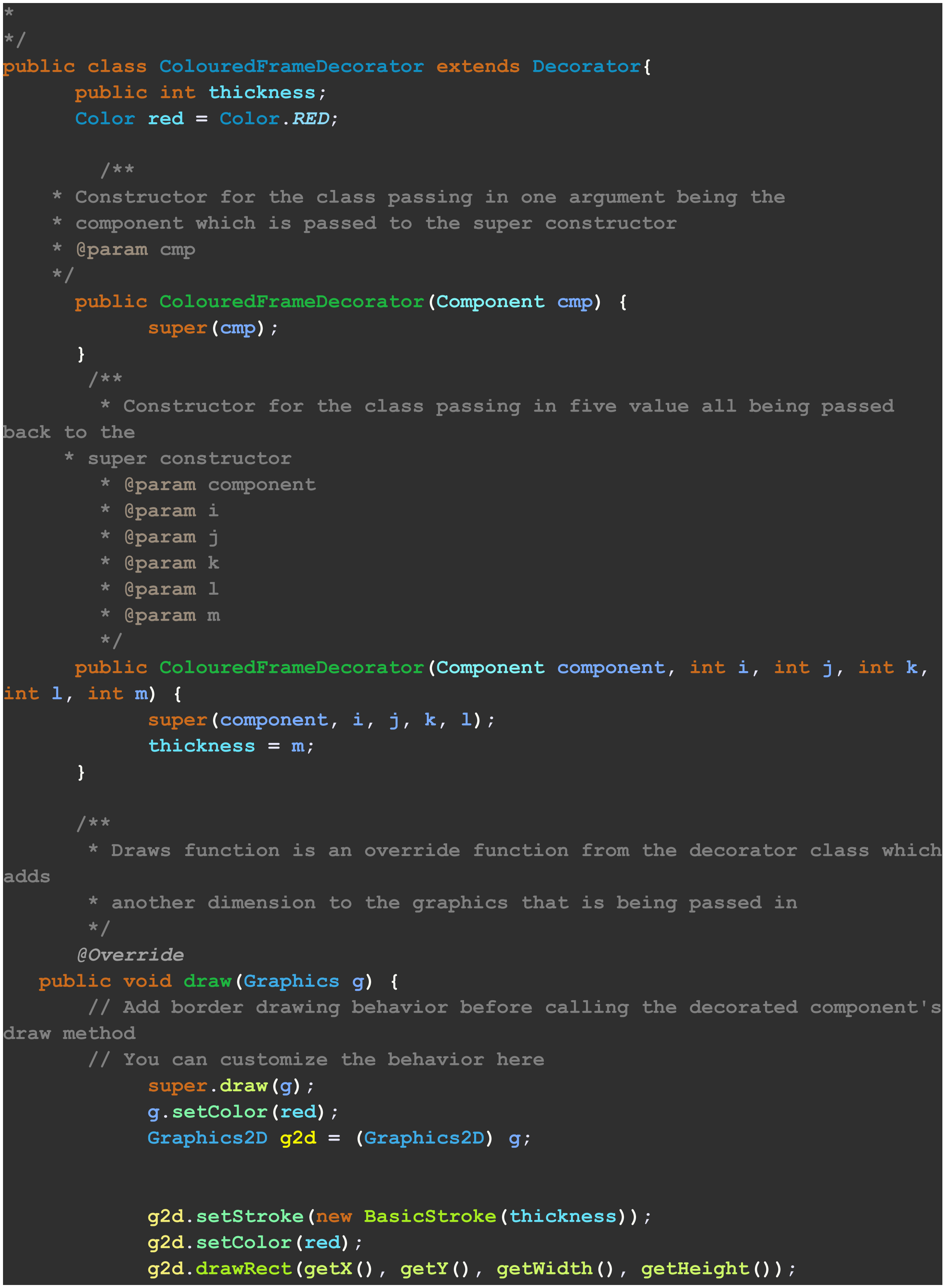
```
/**
* Decorator.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE E and F
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.awt.Graphics;
/**
* Decorator class implements
* @authors Braden Tink and Christian Valdez
* Class has five class variables being the components of the graphics
*/
class Decorator implements Component {
      protected int x;
   protected int y;
   protected int width;
   protected int height;
      protected Component cmp;

      /**
       * Decorator constructor takes in five arguments
       * Sets all five class variables with the arguments
```



```
    * @param component
    * @param i
    * @param j
    * @param k
    * @param l
    */
public Decorator(Component component, int i, int j, int k, int l) {
         x = i;
         y = j;
         width = k;
         height = l;
   cmp = component;
}

/**
 * Decorator constructor taking in one argument and
 * setting the component of the class
 * @param component
 */
public Decorator(Component component) {
         // TODO Auto-generated constructor stub
   cmp = component;
   }
/**
 * getX returns the X value
 * @return
 */
   int getX(){
   return x;
}
   /**
 * getX returns the Y value
 * @return
 */
int getY(){
   return y;
}
/**
 * getX returns the Width value
 * @return
 */
int getWidth(){
   return width;
}
/**
 * getX returns the Height value
 * @return
 */
int getHeight(){
      return height;
   }
   /**
    * Draw function takes in one argument the graphics then
    * calls draw on passing in that graphic
    */
   @Override
      public void draw(Graphics g) {
            // TODO Auto-generated method stub
              cmp.draw(g);

      }
 }
```

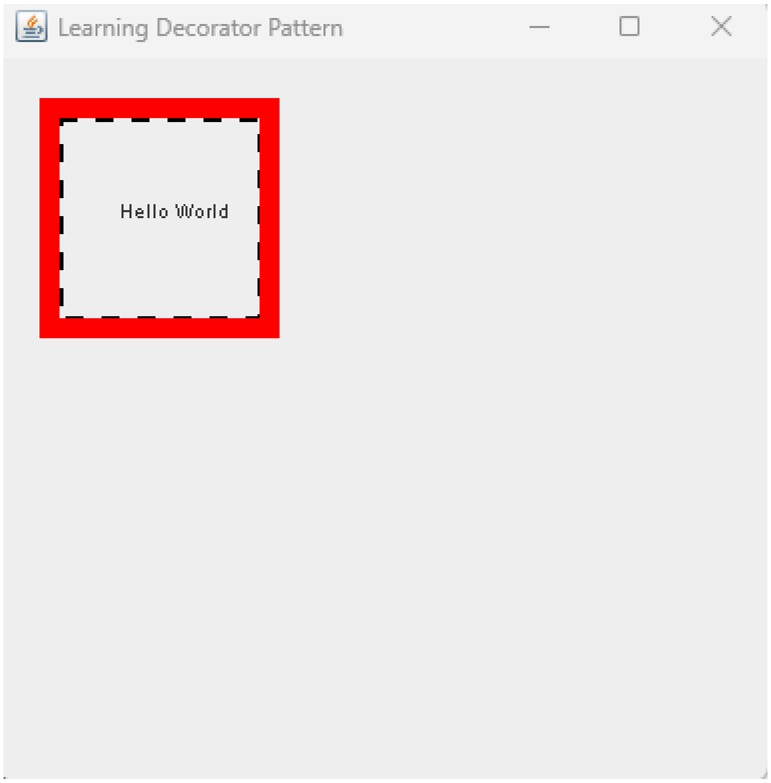
```
/**
* Component.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE E and F
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.awt.Graphics;
/***
* Component class that contains one function
* being the draw function that has one argument being a
* graphic
* @authors Braden Tink and Christian Valdez
*
*/
interface Component {
      void draw(Graphics g);

}
```

```
/**
* ColouredFrameDecorator.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE E and F
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.awt.Graphics;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;
/**
* ColouredFrameDecorator class extend the decorator class
* Class adds another dimension to the graphic
* @authors Braden Tink and Christian Valdez
* Class has one class variable which is the color red being used
* for the border color
```



```
*
*/
public class ColouredFrameDecorator extends Decorator{
      public int thickness;
      Color red = Color.RED;

        /**
    * Constructor for the class passing in one argument being the
    * component which is passed to the super constructor
    * @param cmp
    */
      public ColouredFrameDecorator(Component cmp) {
            super(cmp);
      }
       /**
        * Constructor for the class passing in five value all being passed
back to the
     * super constructor
        * @param component
        * @param i
        * @param j
        * @param k
        * @param l
        * @param m
        */
      public ColouredFrameDecorator(Component component, int i, int j, int k,
int l, int m) {
            super(component, i, j, k, l);
            thickness = m;
      }

      /**
       * Draws function is an override function from the decorator class which
adds
       * another dimension to the graphics that is being passed in
       */
      @Override
   public void draw(Graphics g) {
       // Add border drawing behavior before calling the decorated component's
draw method
       // You can customize the behavior here
            super.draw(g);
            g.setColor(red);
            Graphics2D g2d = (Graphics2D) g;


            g2d.setStroke(new BasicStroke(thickness));
            g2d.setColor(red);
            g2d.drawRect(getX(), getY(), getWidth(), getHeight());
    }
}
```

Sample Output



Hello World

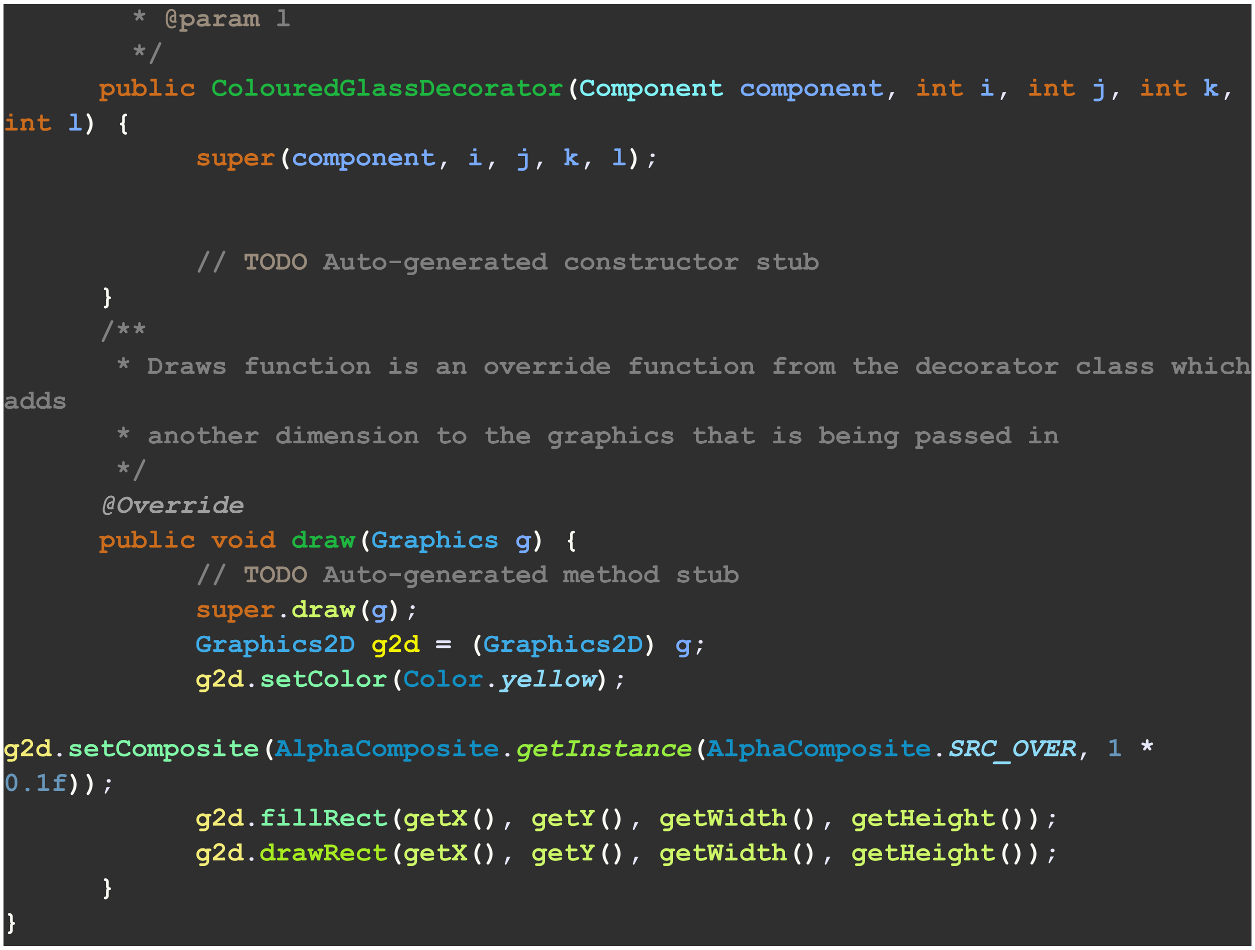
Exercise F

Classes (All class will be the same expect an added class for F and the change to the PaintComponet Class)

```
/**
* DemoDecoratorPattern.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE E and F
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.awt.Font;
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JPanel;
/*
* The DemoDecoratorPattern class is the entry point of the program
* And creates the JPanel that is displayed by the program
* The class also declares all the variables that will be placed into the
* constructors.
```

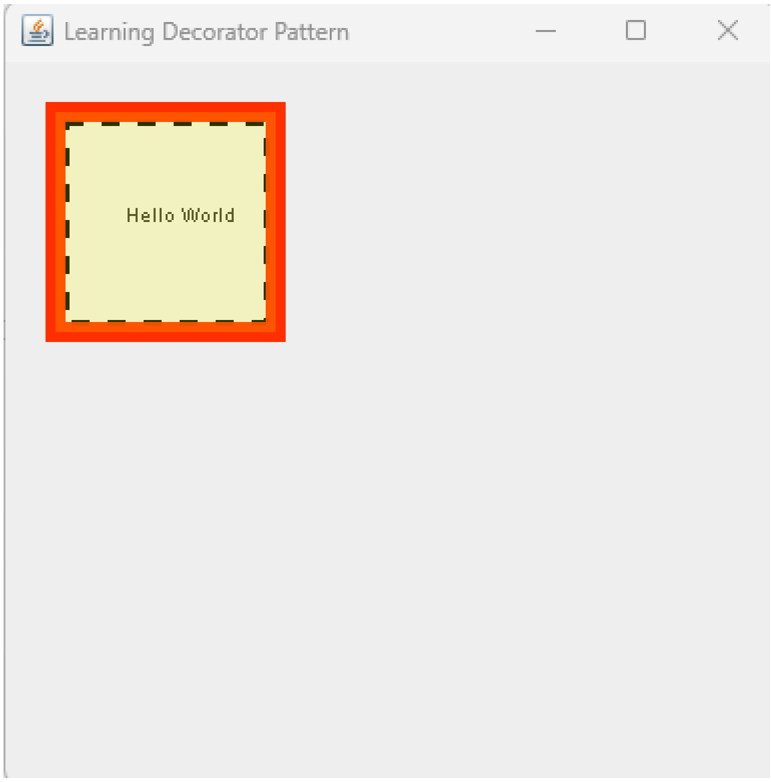
```
*/
public class DemoDecoratorPattern extends JPanel {
      Component t;

      /**
       * Class declares a new text object passing in default values
       */
   public DemoDecoratorPattern(){
       t = new Text ("Hello World", 60, 80);
  }

   /**
    * painComponent Class call the constructors of the the decorator objects.
    * Class takes in one argument being the graphics object g
    *
    * Two painComponet classes are defined for part E and F
    */

//    public void paintComponent(Graphics g){
//        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);
//    }

   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);
   }
      /**
       * Main function entry point of the program
       * @param args
       */
      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);
      }
}
```

```
/**
* BubbleSorter.java
* ENSF 614 Fall 2023 LAB 6 - EXERCISE E and F
* @authors Braden Tink and Christian Valdez
* Submitted On: Nov 11, 2023
*/
import java.awt.Graphics;
import java.awt.AlphaComposite;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;
/**
* ColouredGlassDecorator class extend the decorator class
* Class adds another dimension to the graphic
* @authors Braden Tink and Christian Valdez
* class has one class variable which is the color yellow being used
* for the border color
*/
public class ColouredGlassDecorator extends Decorator {

      Color yellow = Color.YELLOW;
       /**
     * Constructor for the class passing in one argument being the
     * component which is passed to the super constructor
     * @param cmp
     */
   public ColouredGlassDecorator(Component cmp) {
      super(cmp);
   }

   /**
       *Constructor for the class passing in five value all being passed back
to the
     * super constructor
         * @param component
         * @param i
         * @param j
         * @param k
```



```
        * @param l
        */
      public ColouredGlassDecorator(Component component, int i, int j, int k,
int l) {
            super(component, i, j, k, l);


            // TODO Auto-generated constructor stub
      }
      /**
       * Draws function is an override function from the decorator class which
adds
       * another dimension to the graphics that is being passed in
       */
      @Override
      public void draw(Graphics g) {
            // TODO Auto-generated method stub
            super.draw(g);
            Graphics2D g2d = (Graphics2D) g;
            g2d.setColor(Color.yellow);

g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 1 *
0.1f));
            g2d.fillRect(getX(), getY(), getWidth(), getHeight());
            g2d.drawRect(getX(), getY(), getWidth(), getHeight());
      }
}
```

Sample Output



Learning Decorator Pattern

Hello World