ENSF480 Lab 5 F2021

**Names:** Beau McCartney, Quentin Jennings

**Course Name:** Principles of Software Design

**Lab Section:** B02 (Dr. Moshirpour)

**Course Code:** ENSF 480 Fall 2021

**Assignment Number:** Lab 5

**Submission Date:** 04/11/2021

**Exercise A Code:**

/\* ENSF 480 - Lab 5 - Exercise A and B

    File: MyVector.java

    For: Lab 5 Exercise A

    Written By: Quentin J, Beau M

 \*/

import java.util.ArrayList;

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

    private ArrayList<Item<E>> storageM;

    private Sorter<E> sorter;

    public MyVector(final int n) {

        storageM = new ArrayList<Item<E>>(n);

    }

    public MyVector(final ArrayList<Item<E>> arr) {

        storageM = new ArrayList<>(arr);

    }

    public void add(final Item<E> value) {

        storageM.add(value);

    }

    public void setSortStrategy(final Sorter <E> s) {

        sorter = s;

    }

    public void performSort() {

        sorter.sort(storageM);

    }

    public void display() {

        for (Item<E> item : storageM) {

            System.out.print(item.getItem() + "\t");

        }

    }

}

/\* ENSF 480 - Lab 5 - Exercise A and B

    File: Sorter.java

    For: Lab 5 Exercise A

    Written By: Quentin J, Beau M

 \*/

import java.util.ArrayList;

public interface Sorter <E extends Number & Comparable<E>> {

    public void sort(ArrayList<Item<E>> arr); // diff return type?

}

/\* ENSF 480 - Lab 5 - Exercise A and B

    File: BubbleSorter.java

    For: Lab 5 Exercise A

    Written By: Quentin J, Beau M

 \*/

import java.util.ArrayList;

//bubble sort - iterates, comparing with the next element and swapping until sorted

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

    @Override

    public void sort(ArrayList<Item<E>> arr) {

        //iterates each element except the last

        for(int i = 0; i < arr.size() - 1; i++){

            //iterates to the end for each i - nested loop

            for(int j = 0; j < arr.size() - i - 1; j++) {

                //if the left element is bigger than the right, swap them

                if(arr.get(j).getItem().compareTo(arr.get(j + 1).getItem()) > 0) {

                    Item<E> temp = arr.get(j);

                    arr.set(j, arr.get(j + 1));

                    arr.set(j + 1, temp);

                }

            }

        }

    }

}

/\* ENSF 480 - Lab 5 - Exercise A and B

    File: InsertionSorter.java

    For: Lab 5 Exercise A

    Written By: Quentin J, Beau M

 \*/

import java.util.ArrayList;

//insersion sort - inserts unsorted values from right into its correct position on the left

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

    @Override

    public void sort(ArrayList<Item<E>> arr) {

        //iterates all but first value

        for(int i = 1; i < arr.size(); i++) {

            Item<E> key = arr.get(i); //key stores the current item

            int j = i-1; //the element before i

            //pushes key back until it finds its place

            while(j >= 0 && arr.get(j).getItem().compareTo(key.getItem()) > 0) {

                arr.set(j + 1, arr.get(j));

                j--;

            }

            arr.set(j + 1, key);

        }

    }

}

**Exercise A**Text

Description automatically generated **Output:**

**Exercise B Source Code:**

/\* ENSF 480 - Lab 5 - Exercise A and B

    File: SelectionSorter.java

    For: Lab 5 Exercise B

    Written By: Quentin J, Beau M

 \*/

import java.util.ArrayList;

//selection sort - selects the smallest element for the current slot and swaps, iterates left to right

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

    @Override

    public void sort(ArrayList<Item<E>> arr) {

        //iterates all but the last value

        for(int i = 0; i < arr.size() - 1; i++) {

            int min = i; //index of smallest item

            //finds the smallest item past i

            for(int j = i + 1; j < arr.size(); j++) {

                if(arr.get(j).getItem().compareTo(arr.get(min).getItem()) < 0)

                    min = j;

            }

            //swaps smallest index and current index items

            Item<E> temp = arr.get(i);

            arr.set(i, arr.get(min));

            arr.set(min, temp);

        }

    }

}

**Exercise C Source Code:**

/\* ENSF 480 - Lab 5 - Exercise C

    File: Subject.java

    Written By: Quentin J, Beau M

 \*/

public interface Subject {

    public void registerObserver(Observer o);

    public void remove(Observer o);

    public void notifyAllObservers();

}

/\* ENSF 480 - Lab 5 - Exercise C

    File: Observer.java

    Written By: Quentin J, Beau M

 \*/

import java.util.ArrayList;

public interface Observer {

    public void update(ArrayList<Double> arr);

    public void display();

}

/\* ENSF 480 - Lab 5 - Exercise C

    File: DoubleArrayListSubject.java

    Written By: Quentin J, Beau M

 \*/

import java.util.ArrayList;

public class DoubleArrayListSubject implements Subject {

    private ArrayList<Double> data; // our data

    private ArrayList<Observer> observers; // observers of this subject

    // initializes data members to new ArrayLists<>

    public DoubleArrayListSubject() {

        data = new ArrayList<Double>();

        observers = new ArrayList<Observer>();

    }

    // adds a data point, notifies observers

    public void addData(double d) {

        this.data.add(d);

        notifyAllObservers();

    }

    // adds a data point, notifies observers

    public void setData(double d, int index) {

        data.set(index, d);

        notifyAllObservers();

    }

    // populates data with an array, notifies all observers

    public void populate(double [] arr) {

        data.clear();

        for (double d : arr) {

            data.add(d);

        }

        notifyAllObservers();

    }

    // adds an observer to the array

    public void registerObserver(Observer o) {

        observers.add(o);

        o.update(data);

    }

    // removes an observer from the array

    public void remove(Observer o) {

        observers.remove(o);

    }

    // notifies all observers

    public void notifyAllObservers() {

        for (Observer o : observers) {

            o.update(data);

        }

    }

    // observes all display methods

    public void display() {

        if (observers.isEmpty()) {

            System.err.println("Empty List ...");

        } else {

            for (Observer o : observers) {

                o.display();

            }

        }

    }

}

/\* ENSF 480 - Lab 5 - Exercise C

    File: OneRow\_Observer.java

    Written By: Quentin J, Beau M

 \*/

import java.util.ArrayList;

public class OneRow\_Observer implements Observer {

    private ArrayList<Double> data; // our data

    private DoubleArrayListSubject subject;

    public OneRow\_Observer(DoubleArrayListSubject subject) {

        this.subject = subject;

        subject.registerObserver(this);

    }

    @Override

    public void update(ArrayList<Double> arr) {

        data = new ArrayList<>(arr);

        System.out.println("\nNotification to One-Row Table Observer: Data Changed:");

        display();

    }

    public void display() {

        if(data.isEmpty()) {

            System.out.println("Empty List ...");

        }

        else {

            for(Double d : data) {

                System.out.print(d + "\t");

            }

            System.out.println();

        }

    }

}

/\* ENSF 480 - Lab 5 - Exercise C

    File: ThreeColumnTable\_Observer.java

    Written By: Quentin J, Beau M

 \*/

import java.util.ArrayList;

public class ThreeColumnTable\_Observer implements Observer {

    private ArrayList<Double> data; // our data

    private DoubleArrayListSubject subject;

    public ThreeColumnTable\_Observer(DoubleArrayListSubject subject) {

        this.subject = subject;

        subject.registerObserver(this);

    }

    @Override

    public void update(ArrayList<Double> arr) {

        data = new ArrayList<>(arr);

        System.out.println("\nNotification to Three-Column Table Observer: Data Changed:");

        display();

    }

    public void display() {

        if(data.isEmpty()) {

            System.out.println("Empty List ...");

        }

        else {

            int counter = 0;

            for(Double d : data) {

                System.out.print(d + "\t");

                counter++;

                if(counter == 3) {

                    System.out.println();

                    counter = 0;

                }

            }

            System.out.println();

        }

    }

}

/\* ENSF 480 - Lab 5 - Exercise C

    File: FiveRowsTable\_Observer.java

    Written By: Quentin J, Beau M

 \*/

import java.util.ArrayList;

public class FiveRowsTable\_Observer implements Observer {

    private ArrayList<Double> data; // our data

    private DoubleArrayListSubject subject;

    public FiveRowsTable\_Observer(DoubleArrayListSubject subject) {

        this.subject = subject;

        subject.registerObserver(this);

    }

    @Override

    public void update(ArrayList<Double> arr) {

        data = new ArrayList<>(arr);

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

        display();

    }

    public void display() {

        if(data.isEmpty()) {

            System.out.println("Empty List ...");

        }

        else {

            int colNum = data.size() / 5; //first we find the number of columns needed

            if(data.size() % 5 != 0)

                colNum++; //adds an extra column for overflow if needed

            for(int i = 0; i < 5; i++) { //row

                for(int j = 0; j < colNum; j++) { //col

                    int index = j \* 5 + i;

                    if(index < data.size())

                        System.out.print(data.get(index) + "\t");

                }

                System.out.println();

            }

            System.out.println();

        }

    }

Text

Description automatically generated**Exercise C Output:**