

Name: Jack Shenfield & William Watson

Course Name: Advanced System Analysis and Software Design

Course Code: ENSF 614

Assignment Number: Lab 05

Submission Date: 29/10/2025

Exercise B

Source code:

```
/* ENSF 614 - Lab 5 Exercise A and B
 * File Name: Item.java
 * M. Moussavi, October 2024
 * Lab Section: B01
 * Completed by: Jack Shenfield & William Watson
 * Submission Date: Oct 29, 2025
 */

package src.Exercises_AB;

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 5 Exercise A and B
```

```
* File Name: MyVector.java
```

```
* M. Moussavi, October 2024
```

```
* Lab Section: B01
```

```
* Completed by: Jack Shenfield & William Watson
```

```
* Submission Date: Oct 29, 2025
```

```
*/
```

```
package src.Exercises_AB;
```

```
import java.util.ArrayList;
```

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

```
    private ArrayList<Item<E>> storageM;
```

```
    private Sorter<E> sorter;
```

```
    // list length constructor
```

```
    public MyVector(int x) {
```

```
        storageM = new ArrayList<>(x);
```

```
    }
```

```
    // copy constructor
```

```
    public MyVector(ArrayList<E> arr) {
```

```
        storageM = new ArrayList<>(arr.size());
```

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

```
        E element = arr.get(i); // get the value
        storageM.add(new Item<>(element)); // wrap as an Item
    }
}

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

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

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

public void display() {
    System.out.print("[ ");

    for (int i = 0; i < storageM.size(); i++) {
        System.out.print(storageM.get(i).getItem() + " "); // assuming
        Item<E> has getValue()
    }
}
```

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

```
/* ENSF 614 - Lab 5 Exercise A and B  
 * File Name: Sorter.java  
 * M. Moussavi, October 2024  
 * Lab Section: B01  
 * Completed by: Jack Shenfield & William Watson  
 * Submission Date: Oct 29, 2025  
 */
```

```
// Interface class for my sorter classes
```

```
package src.Exercises_AB;
```

```
import java.util.ArrayList;
```

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

```
    void sort(ArrayList<Item<E>> array); // function definition to be overridden
```

```
}
```

```
/* ENSF 614 - Lab 5 Exercise A and B  
 * File Name: SelectionSort.java  
 * M. Moussavi, October 2024
```

```
* Lab Section: B01
* Completed by: Jack Shenfield & William Watson
* Submission Date: Oct 29, 2025
*/
```

```
package src.Exercises_AB;
```

```
import java.util.ArrayList;
```

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

```
    @Override
```

```
    public void sort(ArrayList<Item<E>> array) {
```

```
        int n = array.size();
```

```
        for (int i = 0; i < n - 1; i++) {
```

```
            int minIndex = i;
```

```
            // find smallest array element index
```

```
            for (int j = i + 1; j < n; j++) {
```

```
                E current = array.get(j).getItem();
```

```
                E smallest = array.get(minIndex).getItem();
```

```
                if (current.compareTo(smallest) < 0) {
```

```
                    minIndex = j;
```

```
                }
```

```

    }

    // swap if a lesser element is found
    if (minIndex != i) {
        Item<E> temp = array.get(i);
        array.set(i, array.get(minIndex));
        array.set(minIndex, temp);
    }
}
}
}
}
}

```

```

/* ENSF 614 - Lab 5 Exercise A and B

```

```

 * File Name: InsertionSort.java

```

```

 * M. Moussavi, October 2024

```

```

 * Lab Section: B01

```

```

 * Completed by: Jack Shenfield & William Watson

```

```

 * Submission Date: Oct 29, 2025

```

```

 */

```

```

package src.Exercises_AB;

```

```

import java.util.ArrayList;

```

```

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

```

```

    @Override

```

```

public void sort(ArrayList<Item<E>> array) {
    int n = array.size();

    for (int i = 1; i < n; i++) {
        Item<E> keyItem = array.get(i); // current value
        E keyValue = keyItem.getItem();

        int j = i - 1;

        // search for place to insert keyValue
        while (j >= 0 && array.get(j).getItem().compareTo(keyValue) > 0) {
            array.set(j + 1, array.get(j));
            j--;
        }

        array.set(j + 1, keyItem);
    }
}
}

```

```

/* ENSF 614 - Lab 5 Exercise A and B
 * File Name: BubbleSort.java
 * M. Moussavi, October 2024
 * Lab Section: B01
 * Completed by: Jack Shenfield & William Watson
 * Submission Date: Oct 29, 2025
 */

```



```
package src.Exercises_AB;

import java.util.ArrayList;

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

    @Override
    public void sort(ArrayList<Item<E>> array) {
        int n = array.size();

        boolean swapped; // check if swap occurred

        for (int i = 0; i < n - 1; i++) { // for each value
            swapped = false;
            for (int j = 0; j < n - i - 1; j++) { // search up until the value
                E a = array.get(j).getItem();
                E b = array.get(j + 1).getItem();

                if (a.compareTo(b) > 0) { // swap if needed
                    Item<E> temp = array.get(j);
                    array.set(j, array.get(j + 1));
                    array.set(j + 1, temp);
                    swapped = true;
                }
            }
        }

        // if no swaps needed, break
    }
}
```

```
        if (!swapped)

            break;
    }
}
}
```

```
/* ENSF 614 - Lab 5 Exercise A and B
 * File Name: DemoStrategyPattern.java
 * M. Moussavi, October 2024
 * Lab Section: B01
 * Completed by: Jack Shenfield & William Watson
 * Submission Date: Oct 29, 2025
 */
```

```
package src.Exercises_AB;
```

```
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 BubbleSort<Double>());

// perform algorithm bubble sort to v1
v1.performSort();

System.out.println("\nThe values in MyVector object v1 after performing
BubbleSort 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 InsertionSort<Integer>());
    ;
    v2.performSort();
    System.out.println("\nThe values in MyVector object v2 after performing
InsertionSort is:");
    v2.display();

    // create a MyVector<Integer> object V2
    MyVector<Integer> v3 = new MyVector<Integer>(50);

    // populate v3 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: (copied test
pattern from v2):");
    v3.display();
    v3.setSortStrategy(new SelectionSort<Integer>());

```

```

        ;

        v3.performSort();

        System.out.println("\nThe values in MyVector object v2 after performing
SelectionSort is:");

        v3.display();
    }
}

```

Program output:

```

<terminated> DemoStrategyPattern [Java Application] /Users/bs.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.aarch64_21.0.7.v20250502-0916/jre/bin/java (Oct 27, 2025, 7:16:11 p
The original values in v1 object are:
[ 52.30594680200651 18.955683713271377 75.09095125409922 79.8176106610829 49.082386786569366 ]

The values in MyVector object v1 after performing BubbleSort is:
[ 18.955683713271377 49.082386786569366 52.30594680200651 75.09095125409922 79.8176106610829 ]

The original values in v2 object are:
[ 27 31 5 30 45 ]

The values in MyVector object v2 after performing InsertionSort is:
[ 5 27 30 31 45 ]

The original values in v3 object are: (copied test pattern from v2):
[ 48 44 26 29 12 ]

The values in MyVector object v2 after performing SelectionSort is:
[ 12 26 29 44 48 ]

```

Exercise C

Source code:

```
/* ENSF 614 - Lab 5 Exercise C
 * File Name: Subject.java
 * M. Moussavi, October 2024
 * Lab Section: B01
 * Completed by: Jack Shenfield & William Watson
 * Submission Date: Oct 29, 2025
 */
public interface Subject {
    void registerObserver(Observer o);
    void removeObserver(Observer o);
    void notifyAllObservers();
}
```

```
/* ENSF 614 - Lab 5 Exercise C
 * File Name: Observer.java
 * M. Moussavi, October 2024
 * Lab Section: B01
 * Completed by: Jack Shenfield & William Watson
 * Submission Date: Oct 29, 2025
 */
```

```
import java.util.ArrayList;

public interface Observer {
    void update(ArrayList<Double> data);
}
```

```
/* ENSF 614 - Lab 5 Exercise C
 * File Name: DoubleArrayListSubject.java
 * M. Moussavi, October 2024
 * Lab Section: B01
 * Completed by: Jack Shenfield & William Watson
 * Submission Date: Oct 29, 2025
 */
```

```
import java.util.ArrayList;

public class DoubleArrayListSubject implements Subject {

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

    DoubleArrayListSubject() {
        data = new ArrayList<>();
    }
}
```

```
observers = new ArrayList<>();  
}
```

```
@Override
```

```
public void registerObserver(Observer o) {  
    if (o == null || observers.contains(o)) {  
        return;  
    }  
    observers.add(o);  
    o.update(data);  
}
```

```
@Override
```

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

```
@Override
```

```
public void notifyAllObservers() {  
    for( Observer o : observers) {  
        o.update(new ArrayList<>(data));  
    }  
}
```

```
public void addData(double val) {  
    data.add(val);  
}
```



```

        notifyAllObservers();
    }

    public void setData(double val, int index) {
        data.set(index, val);
        notifyAllObservers();
    }

    public void populate(double[] vals) {
        data.clear();
        for(double v : vals) {
            data.add(v);
        }
        notifyAllObservers();
    }

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

    public void display() {
        System.out.println("Current data: " + data);
    }
}

```

```
* File Name: FiveRowsTable_Observer.java
* M. Moussavi, October 2024
* Lab Section: B01
* Completed by: Jack Shenfield & William Watson
* Submission Date: Oct 29, 2025
*/
```

```
import java.util.ArrayList;
```

```
public class FiveRowsTable_Observer implements Observer {
```

```
    private ArrayList<Double> data;
```

```
    private Subject subject;
```

```
    public FiveRowsTable_Observer(Subject subject) {
```

```
        this.subject = subject;
```

```
        this.data = new ArrayList<>();
```

```
        subject.registerObserver(this);
```

```
    }
```

```
    @Override
```

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

```
        // TODO Auto-generated method stub
```

```
        this.data = new ArrayList<>(data);
```

```
        display();
```

```
    }
```

```

private void display() {

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

    int length = data.size();

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

        for(int j = i; j < length; j += 5) {
            System.out.print(data.get(j) + ", ");
        }
        System.out.println();
    }
    System.out.println();
}
}

```

```

/* ENSF 614 - Lab 5 Exercise C
 * File Name: ThreeColumnTable_Observer.java
 * M. Moussavi, October 2024
 * Lab Section: B01
 * Completed by: Jack Shenfield & William Watson
 * Submission Date: Oct 29, 2025
 */

```

```

import java.util.ArrayList;

```

```
public class ThreeColumnTable_Observer implements Observer {

    private ArrayList<Double> data;
    private Subject subject;

    public ThreeColumnTable_Observer(Subject subject) {
        this.subject = subject;
        this.data = new ArrayList<>();
        subject.registerObserver(this);
    }

    @Override
    public void update(ArrayList<Double> data) {
        // TODO Auto-generated method stub
        this.data = new ArrayList<>(data);
        display();
    }

    private void display() {

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

        int length = data.size();

        for(int i = 0; i < length; i += 3) {
```

```

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

        if(i+1 < length) {
            System.out.print(data.get(i+1) + ", ");
        }

        if (i+2 < length) {
            System.out.print(data.get(i+2));
        }

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

```

```

/* ENSF 614 - Lab 5 Exercise C
 * File Name: OneRow_Observer.java
 * M. Moussavi, October 2024
 * Lab Section: B01
 * Completed by: Jack Shenfield & William Watson
 * Submission Date: Oct 29, 2025
 */

```

```

import java.util.ArrayList;

```

```

public class OneRow_Observer implements Observer {

```

```
private ArrayList<Double> data;
private Subject subject;

public OneRow_Observer(Subject subject) {
    this.subject = subject;
    this.data = new ArrayList<>();
    subject.registerObserver(this);
}

@Override
public void update(ArrayList<Double> data) {
    // TODO Auto-generated method stub
    this.data = new ArrayList<>(data);
    display();
}

private void display() {

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

    int length = data.size();

    for(int i = 0; i < length; i++) {
        System.out.print(data.get(i) + ", ");
    }
}
```

```

        System.out.println();

        System.out.println();

    }

}

```

Program output:

```

Creating object mydata with an empty list -- no data:
Expected to print: Empty List ...
Current data: []
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 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-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 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-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, 1000.0,

Notification to One-Row 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-Row 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,
```