**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:

A screenshot of a computer

AI-generated content may be incorrect.

Exercise B

Exercise C