Practical Task 3.1

(Pass Task)

Submission deadline: 11:59 pm Sunday, August 9th
Discussion deadline: 11:59 pm Sunday, August 30th

Task Objective

In this task you will implement a number of the traditional sorting routines (*Bubble Sort*, *Insertion Sort*, and *Selection Sort*) by extending the Vector class you worked on in Task 1.1P.

Background

In task 1.1P you implemented sorting algorithms that will work with the Vector class which will provide the ability to sort the items in the vector. To accomplish this you will be using interfaces with Generics.

A vector is not a sorter so it does not make sense to create new class that inherits from the Vector class to incorporate the ability to sort. Additionally, we want the client program to be able to change the method of sorting used by the vector. Therefore, we are not going to add the code for sorting directly into the Vector class. Instead we want to use an association relationship between our Vector and our Sorting method. This will give us the ability to flexibly switch between possible sorting algorithms. A simple solution to implement this linkage is an interface that every class providing sorting functionality must implement. As part of this task you have been provided with a fully implemented Sorter interface, named *ISorter*, and can be found in the ISorter.cs file.

This interface comes with a single generic method.

- **Sort<K>(K[] sequence, IComparer<K> comparer)**

  Sorts the elements in the entire one-dimensional array of generic type K using the specified comparer IComparer<K>. When the comparer is not specified, i.e. is *null*, the default comparer Comparer<K>.Default is applied.

Note that it imposes a constraint on the type K such that K must implement the IComparable<K> interface. This is done to guarantee that the actual data type substituting K in practice implements a default comparer, therefore the Sort<K> can sort elements of that type according to the default ordering rule determined by the IComparable<K>.

Essentially, the attached template of the Vector<T> class has a public property, named *Sorter*, implementing the aforementioned *ISorter* interface. The purpose of the property is to refer to a particular instance of the sorting algorithm that is currently in use by the *Vector<T>* class. Note that it realizes so-called *Class Aggregation* as a particular form of class relationship in terms of object-oriented programming design. This makes the chosen sorting algorithm encapsulated as a class, e.g. *BubbleSort* or *InsertionSort*, so that it becomes "a part of" the *Vector<T>* class. (To review possible class relationships, explore Chapter 4 of the SIT232 Workbook, page 99). One may read and write to this property; that is, to get/set a reference to the object serving as a sorting algorithm. Obviously, by changing the value of this property, one may also switch the sorting approach in use.

By default, this *Sorter* property of the *Vector<T>* class refers to the built-in internal class *DefaultSorter*, which implements the *Sort<K>* method prescribed by the imposed *ISorter* interface. The *DefaultSorter* class delegates sorting to the *Array.Sort* method.

Task Details

Using the Vector class you implemented as part of Task 1.1P work through the following steps to complete the task:

1. Download the source code files attached to this task. These files include the Tester class, *Vector* class and the *ISorter* interface. Create a new Microsoft Visual Studio project and import the Vector.cs file, (or alternatively, extend the project inherited from Task 1.1 by copying the missing code from the enclosed template for the *Vector<T>* class to your version of *Vector<T>* from Task 1.1P – note copy all the bits that are changed including the class headers use of the *IComparable* interface). Import the Tester.cs and ISorter.cs files to the project to access the prepared *Main* method important for the purpose of debugging and testing the required algorithmic solutions.
2. Compile and test the code. Then inspect the code provided and ensure you *understand* how it works.
3. Implement three sorting algorithms: Bubble Sort, Insertion Sort, and Selection Sort. For this purpose, create three new classes, i.e. *BubbleSort*, InsertionSort, and SelectionSort, respectively. Ensure that the new classes implement the ISorter interface along with its prescribed method *Sort<K>*. Each class must have a default constructor. You may add any extra private methods and attributes if necessary. You should rely on the code of *DefaultSorter* (in the vector class) as an example of how your classes are to be implemented. Therefore, explore the code of the method and how it deals with the default comparer, i.e. the *Comparer<K>.*
4. As you progress with the implementation of the algorithms, you should start using the Tester class in order to test them for potential logical issues and runtime errors. This (testing) part of the task is as important as coding. You may wish to extend it with extra test cases to be sure that your solutions are checked against other potential mistakes. To enable the tests, remember to uncomment the corresponding code lines. Remember that you may change the sorting approach for an instance of the Vector<T> class by referring its Sorter property to a new object. For example,

   ```
   vector.Sorter = new BubbleSort();
   ```

   should enable the Bubble Sort algorithm encoded via the *BubbleSort* class. Check whether you test the right algorithm.

Expected Printout

Appendix A provides an example printout for your program. If you are getting a different printout then your implementation is not ready for submission. Please ensure your program prints out Success for all tests before submission.

Further Notes

- Explore Chapter 7 of SIT221 Workbook available in CloudDeakin in Resources → Additional Course Resources → Resources on Algorithms and Data Structures → SIT221 Workbook. It starts with general explanation of algorithm complexity and describes Bubble Sort, Insertion Sort, and Selection Sort algorithms. Study the provided examples and follow the pseudocodes as you progress with coding of the algorithms.
- The implementation of the Insertion Sort algorithm is also detailed in Chapter 3.1.2 of the course book "Data Structures and Algorithms in Java" by Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser (2014). You may access this book on-line for free from the reading list application in CloudDeakin available in Resources → Additional Course Resources → Resources on Algorithms and Data Structures → Course Book: Data structures and algorithms in Java.

- If you still struggle with such OOP concepts as Generics and their application, you may wish to read Chapter 11 of SIT232 Workbook available in Resources → Additional Course Resources → Resources on Object-Oriented Programming. You may also have to read Chapter 6 of SIT232 Workbook about Polymorphism and Interfaces as you need excellent understanding of these topics to progress well through the practical tasks of the unit. Make sure that you are proficient with them as they form a basis to design and develop programming modules in this and all the subsequent tasks. You may find other important topics required to complete the task, like exceptions handling, in other chapters of the workbook.
- We will test your code in Microsoft Visual Studio 2017. Find the instructions to install the community version of Microsoft Visual Studio 2017 available on the SIT221 unit web-page in CloudDeakin at Resources → Additional Course Resources → Software → Visual Studio Community 2017. You are free to use another IDE if you prefer that, e.g. Visual Studio Code. But we recommend you to take a chance to learn this environment.

Marking Process and Discussion

To get your task completed, you must finish the following steps strictly on time.

1. Work on your task either during your allocated lab time or during your own study time.
2. Once the task is complete you should make sure that your program implements all the required functionality, is compliable, and has no runtime errors. Programs causing compilation or runtime errors will not be accepted as a solution. You need to test your program thoroughly before submission. Think about potential errors where your program might fail. Note we can sometime use test cases that are different to those provided so verify you have checked it more thoroughly than just using the test program provided.
3. Submit your solution as an answer to the task via the OnTrack submission system. This first submission must be prior to the submission "S" deadline indicated in the unit guide and in OnTrack.
4. If your task has been assessed as requiring a "Redo" or "Resubmit" then you should prepare a new submission. You will have 1 (7 day) calendar week from the day you receive the assessment from the tutor. This usually will mean you should revise the lecture, the readings indicated, and read the unit discussion list for suggestions. After your submission has been corrected and providing it is still before the due deadline you can resubmit.
5. If your task has been assessed as correct, either after step 3 or 4, you can "discuss" with your tutor. This first discussion must occur prior to the discussion "D".
6. Meet with your tutor or answer question via the intelligent discussion facility to demonstrate/discuss your submission. Be on time with respect to the specified discussion deadline.
7. The tutor will ask you both theoretical and practical questions. Questions are likely to cover lecture notes, so attending (or watching) lectures should help you with this compulsory interview part. The tutor will tick off the task as complete, only if you provide a satisfactory answer to these questions.
8. If you cannot answer the questions satisfactorily your task will remain on discussion and you will need to study the topic during the week and have a second discussion the following week.
9. Please note, due to the number of students and time constraints tutors will only be expected to mark and/or discuss your task twice. After this it will be marked as a "Exceeded Feedback".
10. If your task has been marked as "Exceeded Feedback" you are eligible to do the redemption quiz for this task. Go to this unit's site on Deakin Sync and find the redemption quiz associated with this task. You get three tries at this quiz. Ensure you record your attempt.
    I. Login to Zoom and join a meeting by yourself.
    II. Ensure you have both a camera and microphone working
    III. Start a recording.
    IV. Select Share screen then select "Screen". This will share your whole desktop. Ensure Zoom is including your camera view of you in the corner.
    V. Bring your browser up and do the quiz.

VI. Once finished select stop recording.
VII. After five to ten minutes you should get an email from Zoom providing you with a link to your video. Using the share link, copy this and paste in your chat for this task in OnTrack for your tutor to verify the recording.

11. Note that we will not check your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail one of the deadlines, you fail the task and this reduces the chance to pass the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work through the unit.
12. Final note, A “Fail” or “Exceeded Feedback” grade on a task does not mean you have failed the unit. It simply means that you have not demonstrated your understanding of that task through OnTrack. Similarly failing the redemption quiz also does not mean you have failed the unit. You can replace a task with a task from a higher grade.

Appendix A

This section displays the printout produced by the attached Tester class, specifically by its *Main* method. It is based on our solution. The printout is provided here to help with testing your code for potential logical errors. It demonstrates the correct logic rather than an expected printout in terms of text and alignment.

```
Test A: Sort integer numbers applying Default Sort with AscendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [100,120,122,175,213,236,263,299,312,333,511,596,722,724,752,772,780,958,966,995]
 :: SUCCESS

Test B: Sort integer numbers applying Default Sort with DescendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [995,966,958,780,772,752,724,722,596,511,333,312,299,263,236,213,175,122,120,100]
 :: SUCCESS

Test C: Sort integer numbers applying Default Sort with EvenNumberFirstComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [724,596,958,752,120,122,966,772,722,100,780,312,236,213,995,263,175,299,511,333]
 :: SUCCESS

Test D: Sort integer numbers applying BubbleSort with AscendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [100,120,122,175,213,236,263,299,312,333,511,596,722,724,752,772,780,958,966,995]
 :: SUCCESS

Test E: Sort integer numbers applying BubbleSort with DescendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [995,966,958,780,772,752,724,722,596,511,333,312,299,263,236,213,175,122,120,100]
 :: SUCCESS

Test F: Sort integer numbers applying BubbleSort with EvenNumberFirstComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [724,596,958,752,120,122,966,772,722,100,780,312,236,213,995,263,175,299,511,333]
 :: SUCCESS

Test G: Sort integer numbers applying SelectionSort with AscendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [100,120,122,175,213,236,263,299,312,333,511,596,722,724,752,772,780,958,966,995]
 :: SUCCESS

Test H: Sort integer numbers applying SelectionSort with DescendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [995,966,958,780,772,752,724,722,596,511,333,312,299,263,236,213,175,122,120,100]
 :: SUCCESS
```

```
Test I: Sort integer numbers applying SelectionSort with EvenNumberFirstComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [236,312,780,100,722,966,724,122,120,752,958,596,772,175,511,333,213,299,995,263]
 :: SUCCESS

Test J: Sort integer numbers applying InsertionSort with AscendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [100,120,122,175,213,236,263,299,312,333,511,596,722,724,752,772,780,958,966,995]
 :: SUCCESS

Test K: Sort integer numbers applying InsertionSort with DescendingIntComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [995,966,958,780,772,752,724,722,596,511,333,312,299,263,236,213,175,122,120,100]
 :: SUCCESS

Test L: Sort integer numbers applying InsertionSort with EvenNumberFirstComparer:
Intital data: [333,236,312,780,100,722,511,966,213,724,122,120,263,175,752,958,596,299,995,772]
Resulting order: [236,312,780,100,722,966,724,122,120,752,958,596,772,333,511,213,263,175,299,995]
 :: SUCCESS


 ---------- SUMMARY ----------
Tests passed: ABCDEFGHIJKL
```