/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Interfaces: Part 4 (Program.cs)

\*

\* Core Topics:

\* 1. Explicit implementation of interfaces.

\* 2. Resolving ambiguity among different interface members.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

usingSystem**;**

usingSystem.Collections.Generic**;**

usingSystem.Text**;**

namespaceInterfaceDemo

**{**

interfaceISortable

**{**

voidSort**();**

**}**

interfaceISearchable

**{**

intIndexOf**(**objecto**);**

**bool Found(object o);**

**}**

**interface IDiscoverable**

**{**

**bool Found(object o);**

**}**

// By implementing IDiscoverable, there is a naming ambiguity

// between it and ISearchable. No warning or error occurs, but

// we should indicate to the compiler and client code what our

// intentions are.

internalclassCollection **:** ISearchable**,** ISortable**, IDiscoverable**

**{**

privateint **[]** \_internalList**;**

publicCollection**(**intsize**)**

**{**

\_internalList=newint**[**size**];**

GenerateNumbers**();**

**}**

publicintIndexOf**(**objecto**)**

**{**

intindex=-1**;**

inttargetData=0**;**

try

**{**

targetData= **(**int**)**o**;**

for **(**inti=0**;** i<\_internalList.Length**;** i++**)**

**{**

if **(**\_internalList**[**i**]** ==targetData**)**

**{**

index=i**;**

break**;**

**}**

**}**

**}**

catch **(**Exceptione**)**

**{**

index=-1**;**

Console.WriteLine**(**"\nEXCEPTION: {0}"**,** e.Message**);**

Console.WriteLine**(**" {1}"**,** e.StackTrace**);**

**}**

returnindex**;**

**}**

// We are explicitly saying that this method applies to the one

// identified in ISearchable. Notice we had to remove the access

// modifier. The compiler doesn't like it. Also notice that if

// you compile without adding a Found() method for

// IDiscoverable, the compiler generates an error.

**bool ISearchable.Found(object o)**

**{**

boolfound=false**;**

intindex=0**;**

inttargetData=0**;**

try

**{**

targetData= **(**int**)**o**;**

while **(**!found&&index<\_internalList.Length**)**

**{**

if **(**\_internalList**[**index**]** ==targetData**)**

**{**

found=true**;**

**}**

else

**{**

index++**;**

**}**

**}**

**}**

catch **(**Exceptione**)**

**{**

index=-1**;**

Console.WriteLine**(**"\nEXCEPTION: {0}"**,** e.Message**);**

Console.WriteLine**(**" {1}"**,** e.StackTrace**);**

**}**

returnfound**;**

**}**

**bool IDiscoverable.Found(object o)**

**{**

boolresult=false**;**

ISearchablesearchableObject=thisasISearchable**;**

if **(**null!=searchableObject**)**

**{**

result=searchableObject.Found**(**o**);**

**}**

returnresult**;**

**}**

publicvoidSort **()**

**{**

// Because \_internalList is an array, the Array type has a

// sort method, and the Array type knows how to sort ints,

// we can simply rely on the Array.Sort() method to do the

// sorting for us. If we were using our own custom type

// rather than int, we would have to implement

// IComparable to implement how to compare two objects.

Array.Sort**(**\_internalList**);**

**}**

publicoverridestringToString**()**

**{**

StringBuildersb=newStringBuilder**();**

for **(**inti=0**;** i<\_internalList.Length**;** i++**)**

**{**

sb.AppendFormat**(**"{0} "**,** \_internalList**[**i**]);**

**}**

returnsb.ToString**();**

**}**

privatevoidGenerateNumbers**()**

**{**

Randomr=newRandom**(**DateTime.Now.Millisecond**);**

for **(**inti=0**;** i<\_internalList.Length**;** i++**)**

**{**

\_internalList**[**i**]** =r.Next**(**1**,** 10000**);**

System.Threading.Thread.Sleep**(**71**);**

**}**

**}**

**}**

classProgram

**{**

privatestaticvoidSortTheList**(**ISortableunsortedList**)**

**{**

unsortedList.Sort**();**

**}**

privatestaticintGetTargetNumber**()**

**{**

inttargetNumber=-1**;**

Console.Write**(**"\nWhat number are you looking for: "**);**

stringuserInput=Console.ReadLine**();**

if **(**!int.TryParse**(**userInput**,** outtargetNumber**))**

**{**

targetNumber=-1**;**

**}**

returntargetNumber**;**

**}**

staticvoidMain**(**string**[]** args**)**

**{**

boolkeepGoing=true**;**

intlistSize=0**;**

CollectionlistOfNumbers=null**;**

NewCollectionlistOfStrings=null**;**

Console.Clear**();**

Console.Write**(**"How many elements do you want to generate: "**);**

stringuserInput=Console.ReadLine**();**

if **(**!int.TryParse**(**userInput**,** outlistSize**))**

**{**

listSize=0**;**

**}**

if **(**listSize<=0**)**

**{**

keepGoing=false**;**

**}**

else

**{**

listOfNumbers=newCollection**(**listSize**);**

**}**

while **(**keepGoing**)**

**{**

Console.WriteLine**(**"\nChoose from the following options:\n"**);**

Console.WriteLine

**(**"\t1. Get the index of an item in the list."**);**

Console.WriteLine**(**"\t2. See if an item is in the list."**);**

Console.WriteLine**(**"\t3. Display the entire list."**);**

Console.WriteLine**(**"\t4. Sort the entire list."**);**

Console.WriteLine**(**"\t5. Generate a list of strings."**);**

Console.WriteLine**(**"\t6. Display the list of strings."**);**

Console.WriteLine**(**"\t7. Sort the list of strings."**);**

Console.WriteLine**(**"\tE. Exit."**);**

Console.Write**(**"\nYour choice: "**);**

stringuserOption=Console.ReadLine**();**

userOption=userOption.ToUpper**();**

inttargetNumber=-1**;**

switch **(**userOption**)**

**{**

case"1"**:**

targetNumber=GetTargetNumber**();**

if **(**targetNumber>0**)**

**{**

intindex=listOfNumbers.IndexOf**(**targetNumber**);**

if **(**index>=0**)**

**{**

Console.WriteLine

**(**"\n{0} is found at index {1}.\n"**,**

targetNumber**,** index**);**

**}**

else

**{**

Console.WriteLine

**(**"\n{0} is not in the list.\n"**,**

targetNumber**);**

**}**

**}**

break**;**

case"2"**:**

targetNumber=GetTargetNumber**();**

if **(**targetNumber>0**)**

**{**

**IDiscoverable discoverableList =**

**listOfNumbers as IDiscoverable;**

if **(**null!=discoverableList**)**

**{**

if **(discoverableList.Found(targetNumber))**

**{**

Console.WriteLine

**(**"\n{0} is in the list.\n"**,**

targetNumber**);**

**}**

else

**{**

Console.WriteLine

**(**"\n{0} is NOT in the list.\n"**,**

targetNumber**);**

**}**

**}**

else

**{**

Console.WriteLine **(**"\nThis list does not "+

"support IDiscoverable.\n"**);**

**}**

**}**

break**;**

case"3"**:**

Console.Write**(**"\nThe list: {0}\n"**,**

listOfNumbers.ToString**());**

break**;**

case"4"**:**

//// By doing the following, the only methods that are

//// accessible via unsortedNumberList are those that

//// are found in ISortable and System.Object.

//ISortable unsortedNumberList =

// listOfNumbers as ISortable;

//if (null != unsortedNumberList)

//{

// SortTheList(unsortedNumberList);

SortTheList**(**listOfNumbers**);**

Console.WriteLine

**(**"\nThe list of numbers has been sorted.\n"**);**

//}

//else

//{

// Console.WriteLine("The current collection " +

// "does not support ISortable!");

//}

break**;**

case"5"**:**

listOfStrings=newNewCollection**(**listSize**);**

break**;**

case"6"**:**

Console.Write**(**"\nThe list: \n{0}"**,**

listOfStrings.ToString**());**

break**;**

case"7"**:**

//// By doing the following, the only methods that are

//// accessible via unsortedNumberList are those that

//// are found in ISortable and System.Object.

//ISortable unsortedStringList =

// listOfStrings as ISortable;

//if (null != unsortedStringList)

//{

// SortTheList(unsortedStringList);

SortTheList**(**listOfStrings**);**

Console.WriteLine

**(**"\nThe list of strings has been sorted.\n"**);**

//}

//else

//{

// Console.WriteLine("The current collection " +

// "does not support ISortable!");

//}

break**;**

case"E"**:**

keepGoing=false**;**

break**;**

default**:**

Console.WriteLine

**(**"\nYour option {0} is not valid. Try again!\n"**,**

userOption**);**

break**;**

**}**

**}**

Console.Write**(**"\n\nPress <ENTER> to end: "**);**

Console.ReadLine**();**

**}**

**}**

**}**