Exercises about classes (abstract and interface)

Exercise 14.01

This exercise is building up from existing code.

The existing code is a namespace, containing an abstract class and that class contains a Main() method.

Nothing in this code can be changed, except moving Main() to another place, if that seems more logic to you.

The abstract class and the Main() are correct, but the Main() code can't be runned, because some implementation is missing.

You just add a class (or two) to make it work.

This new class is an implementation of the abstract class "Tutorial".

Below the code you find the tasks step by step to make the Main() code workable.

Code

```
using System;

namespace TryoutSomeStuff
{

  public abstract class Tutorial
  {
    abstract public void SetTutorial(int idTutorial, string descriptionTutorial);
    abstract public string GetTutorial();

    static void Main()
    {
      CopyPasteTutorial cpTutorialCSharp = new CopyPasteTutorial();
      CopyPasteTutorial cpTutorialDatabase = new CopyPasteTutorial();

      cpTutorialCSharp.SetTutorial(1, "C# .Net");
      cpTutorialDatabase.SetTutorial(2, "SQL Server");
```

Notes

```
            Console.WriteLine(cpTutorialCSharp.GetTutorial());
            Console.WriteLine(cpTutorialDatabase.GetTutorial());

            Console.ReadLine();
        }

    }

}
```

End code

- In the code above you find 2 things:
  - An abstract class “Tutorial”.
  - A Main method that is testing the class CopyPasteTutorial.
- The class CopyPasteTutorial is missing.
  - You need to develop it.
- These are the steps:
  - Create a class “CopyPasteTutorial”.
  - This class inherits from Tutorial.
  - This class has 2 variables.
    - A key for the tutorial.
    - A description for the tutorial.
  - You can work with properties, but it is not necessary.
  - You need to implement the members that are defined in the abstract class.
    - SetTutorial(2 arguments).
    - GetTutorial().
- When finished, the Main() must work.
- Conceptual is it better to have a class “TestProgram” where the Main is located.
  - Do you understand why this remark is given?

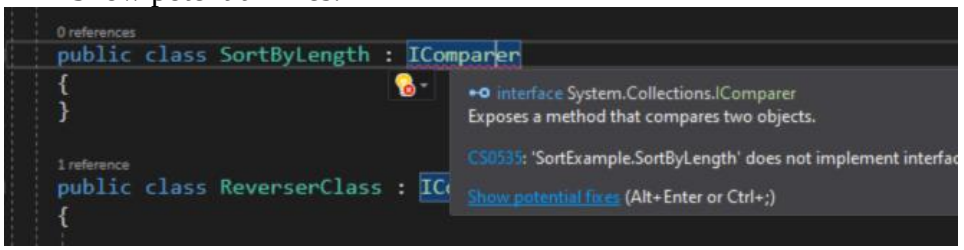
Notes

Exercise 14.02

- This example code below contains two interfaces.
  - IComparer.
  - IEnumerable.
- Both interfaces are part of the .NET framework.
  - IComparer is how things (objects) are compared to each other.
  - IEnumerable is how you can loop thru things (objects).
- Your first task.
  - Write comments in the working code, so that you can explain to somebody else how the code works.
- Your second task.
  - Write the code in the class “SortByLength” that sorts the list of words in another way.
    - Longest word first.
  - This class is now placed in comment.
  - Prove that your new way of sorting works in a test method.

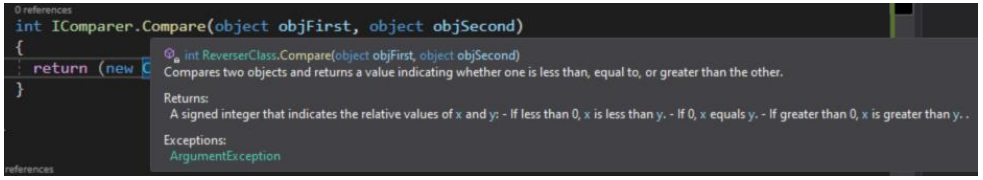
Tip

- Show potential fixes.



- What does IComparer.Compare() do?
  - IComparer is an interface (part of .NET).
  - This is what you must implement to sort it correctly.

Notes



Code

```
using System;
using System.Collections;

namespace TryoutSomeStuff
{

  public class SortExample
  {

    //public class SortByLength : IComparer
    //{
    //}

    public class ReverserClass : IComparer
    {

      int IComparer.Compare(object objFirst, object objSecond)
      {
        return (new CaseInsensitiveComparer().Compare(objSecond, objFirst));
      }

    }

    public static void PrintIndexAndValues(IEnumerable aList)
    {
      int intCounter = 0;

      foreach (var item in aList)
      {
        Console.WriteLine($" [{intCounter++}]: {item}");
      }

      Console.WriteLine();
    }

  }

  public class TestProgram
  {

    public static void Main()
    {
```

Notes

.....

.....

.....

.....

.....

.....

.....

COPY PASTE)

```
        string[] arrWords = {"The", "quick", "brown", "fox", "jumps", "over", "the",
"lazy", "dog"};

        Console.WriteLine("The array as starting point:");
        SortExample.PrintIndexAndValues(arrWords);

        Array.Sort(arrWords);
        Console.WriteLine("After sorting by default:");
        SortExample.PrintIndexAndValues(arrWords);

        Array.Sort(arrWords, new SortExample.ReverserClass());
        Console.WriteLine("After sorting with the ReverserClass:");
        SortExample.PrintIndexAndValues(arrWords);
        Console.ReadLine();
      }

    }

}
```

End Code

Notes

.....

.....

.....

.....

.....

.....

.....

Exercise 14.03

- The starting point of this exercise is the code attached to slide 18 of Part 01 – C# Class Object.pptx.
  - The example of the ArkOfNoah.
- This code has the class of a Panda.
- This code has a Main() that proves that the class Panda works.
- 3 Pandas are created.
- Your task:
  - Create a class "AllThePandas".
  - This contains a list of pandas.
  - Pandas has a method Add.
    - This adds a panda to the list of pandas.
    - The only parameter is the panda to add.
  - Pandas has also an indexer (read only).
    - This gives back the x-th item of the list, which will be a panda.
    - Indexer is zero-based.
      - Meaning, the first panda has index 0.
  - Pandas has a property Count (read only).
    - This gives back the number of pandas in the list.
    - This is an alternative of Panda.Population.
- The code below is the Main() test routine for your new pandas class.
  - This code can't be changed.
  - The only subject to change are the classes Panda and AllThePandas.

You have to add the namespace "System.Collections.Generic".

Notes

.....

.....

.....

.....

.....

.....

.....

COPY PASTE)

Code

```
class TryoutPanda
{

  static void Main()
  {
    AllThePandas allThePandasOfTheWorld = new AllThePandas();

    Panda aFirstPanda = new Panda("Pan Dah");
    Panda aSecondPanda = new Panda("Pan Deh");
    Panda aThirdPanda = new Panda("Pan Daa");

    allThePandasOfTheWorld.Add(aFirstPanda);
    allThePandasOfTheWorld.Add(aSecondPanda);
    allThePandasOfTheWorld.Add(aThirdPanda);

    Console.WriteLine("We created {0} pandas with the names:",
allThePandasOfTheWorld.Count);

    for (int counter = 0; counter < allThePandasOfTheWorld.Count; counter++)
    {
      Console.WriteLine("Is Panda {0} selected? {1}",
allThePandasOfTheWorld[counter].Name, allThePandasOfTheWorld[counter].IsSelected);
    }

    Console.WriteLine();
    Console.WriteLine("Population of pandas at the moment: {0}", Panda.population);

    Console.ReadLine();
  }

}
```

End Code

Variant

- This exercise is similar but with a dictionary.
- Change 14.01 to add a class “CopyPasteTutorials” that contains a dictionary of CopyPasteTutorial.
- You have the key as unique identifier for the dictionary.

Notes

Exercise 14.04

- The starting point of this exercise is the example 00109-h Operator.zip.
  - Matrix calculation.
  - This file can be found on GitHub and on Moodle.
- There is an indexer defined in the class “cpMatrix”.
  - Do you understand how it works?
- The operator “+” is already implemented here.
- The operator “true” and “false” are also implemented.
- Try also to understand how the ToString() works.
- Your tasks:
  - Add the operator “-” (subtract matrixes from each other).
    - Every element is subtracted from its corresponding element.
  - Add the operator “*” (multiplication).
    - Every element is multiplied with a corresponding element.
  - Add the operator “/” (division).
    - Every element is divided by the corresponding element.
    - Division by 0 is not allowed, when this occurs, the operation fails and a nice message must be shown on the console.
      - Tip: use somewhere a try … catch ….

Notes