Exercises about classes (abstract and interface)

Exercise 14.01: Get and Set in methods (not properties)

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");

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

Notes

```
            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

Variant 1

- 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.

Variant 2

This is tricky.

You have a complete different design.

Do not change the existing code. Rewrite it completely from scratch (in another project).

- Change the exercise completely.
  - All is allowed.
- In stead of having the methods get and set, work it out with properties that have a get and a set.

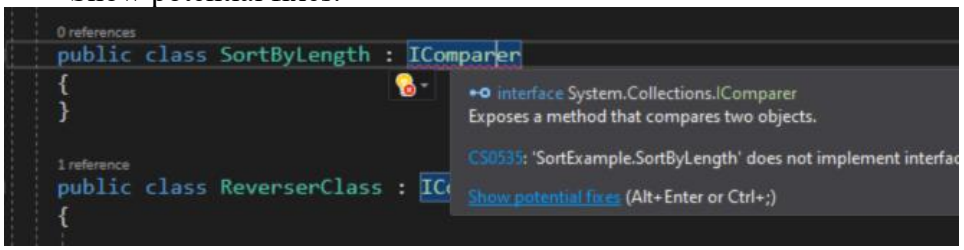
Notes

Exercise 14.02: I want to compare, I want to enumerate

- 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

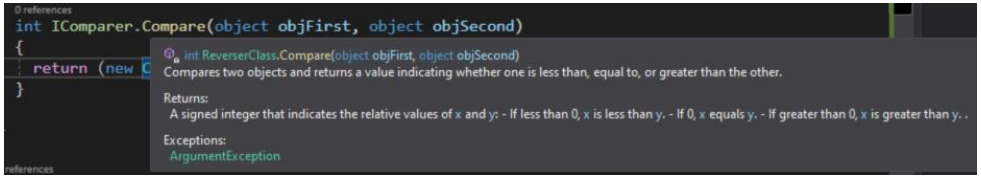
- Show potential fixes.



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

Notes

COPY PASTE)



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()
    {

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

      Console.WriteLine("The array as starting point:");
```

Notes

.....

.....

.....

.....

.....

.....

.....

COPY PASTE)

```
            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

There are several solutions.

You can rewrite the compare by using another compare. This is a chicken and the egg story, but it is correct.

You can also rewrite the compare by exactly programming what is mentioned in the screenshot that shows the return value of a compare. That is what I ask.

Notes

Exercise 14.03: Panda here, panda there, panda everywhere

- 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

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

Notes

Exercise 14.04: An Integer Array

- Create a class IntegerArray, that contains an array of integers.
- We will execute some action on that array.
  - You decide if it is done with a method or done with a dynamic property.
- Make sure you have tested all the functionalities of your class.

Members of the class IntegerArray

- A constructor with a given length and a default value.
  - The result is that in the class you have an array of a specific length, completely filled with that value.
  - When the given length is negative, a NegativeValueException is thrown, and the exception is catched in the testroutine.
  - When the given length is 0, you take by default the length 1.
- A constructor with a given length.
  - The result is that in the class you have an array of a specific length, completely filled with random integer values.
  - When the given length is negative, a NegativeValueException is thrown, and the exception is catched in the testroutine.
  - When the given length is 0, you take by default the length 1.
- A constructor with a given length and a minimum and maximum value.
  - The result is that in the class you have an array of a specific length, completely filled with random integer values between the minimum and maximum value (borders included).
  - When the given length is negative, a NegativeValueException is thrown, and the exception is catched in the testroutine.
  - When the given length is 0, you take by default the length 1.
  - When the minimum is larger than the maximum, the minimum value is used to fill the array.
- You must be able to return the array of the class.

Notes

- You must be able to return the length of the array.
- You must be able to return the sum of all the values of the array. Use for this the data type BigInteger.
- You must be able to change the value at a specific position. If the index is outside the boundaries of the array, a nice error message is shown.
- You must be able to return the value at a specific position. If the index is outside the boundaries of the array, a nice error message is shown.
- You must be able to return the reverse of the array of the class.
  - When the array is 1 – 2 – 3, the array 3 – 2 – 1 must be returned.
- You must be able to reverse the array.
  - Pay attention. This is not the same action as the one above.
- You must be able to return the position of the first found specific value in the array.
  - When the searched value is not there, return -1.
- You must be able to return the number of found values in the array.
  - You look for 3, and if 3 is 10 times in the array, 10 is returned.
- You must be able to return the smallest value of the array.
- You must be able to return the largest value of the array.
- You must be able to return the average of the array.
- You must be able to return all the positions where the smallest value is found.
- You must be able to return all the positions where the largest value is found.
- You must be able to shift down.
  - Call the method ShiftDown.
  - Every value is moved one place to the back.
  - The value at the last position is placed at the first position.
- You must be able to shift up.
  - Call the method ShiftUp.
  - Every value is moved one place to the front.

Notes

- The value at the first position is placed at the last position.

- You must be able to shift down a number of positions.
  - Call the method ShiftDown.
  - Every value is moved a number of positions to the back.
  - Every value that falls of the array must be placed at the front.
  - When the number of positions is negative, you do a ShiftUp.
- You must be able to shift up a number of positions.
  - Call the method ShiftUp.
  - Every value is moved a number of positions to the front.
  - Every value that falls of the array must be placed at the back.
  - Whe the number of positions is negative, you do a ShiftDown.
- You must be able to compare two IntegerArray.
  - When every value of both IntegerArray are the same, You return a true for being equal. Otherwise you return a false.

Notes

.....

.....

.....

.....

.....

.....

.....

Exercise 14.05: The Matrix

- 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.
      - This is wrong in mathematics, but I don’t care. You multiply matrixes like I just say here.
  - Add the operator “/” (division).
    - Every element is divided by the corresponding element.
      - This is wrong in mathematics, but I don’t care. You multiply matrixes like I just say here.
    - 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

Exercise 14.06: A Person with Class

- Create a class Person. (oh, no, not again 😊)
- This class has a normal property “Gender”.
  - Boolean: True (male) or False (female).
- So there is a private variable that contains the gender of the person.
- In the constructor you decide if a person is male or female.
  - Later you can change the gender, using the property.
- It also has a method “GetGender”.
  - This returns the string “Male” if the person is a male.
  - This returns the string “Female” if the person is a female.

You could also write a property to get the gender.

You could also override the ToString() method to get the gender.

Just, for the exercise, create a method.

- Write a test program that tests the class “Person”.

This above should be routine, and cause no difficulties.

Now a fun variant

- Create an interface “IPerson”.
  - The interface has a method “GetGender” that returns a string.
- Create a class “Man”.
  - The method “GetGender” should return “Male”.
- Create a class “Woman”.
  - The method “GetGender” should return “Female”.
- Rewrite your test program using the interface and new classes.

Notes

COPY PASTE)

Notes

Can all your tests be rewritten?

What is according to you the best technical solution? Working with a single class, or working with the interface and classes?

You can start a discussion with Vincent if you want.

The exercise above is a typical design pattern

What you have done here is the Strategy design pattern.

In short, it means that implementation of functionality should be done thru interfaces, not just by creating methods in a class.

A full demo of this principle will be given with a Karel the Robot implementation during the course.

There is a very big exercise on this, but it is with events and this you will see in the second part of the course.

Copy Paste Exercises Fuzzy Duck and Ducky Fuzz.

You do not have seen delegates and events yet, but the solutions and methods are given in the GitHub directory or even on Moodle.

At this moment we have only used events. Not created them by ourself.

The group work you are doing is also this principle. Implement against interfaces.

Exercise 14.07: Number crunching

This is an exercise about lists. I describe here what I want, I do not give the method on how to solve it. Solving the problem is your task.

Think in classes. Organize your code in classes. Your task is to put everything in the correct location. Use interfaces, use abstract classes.

Every member of the class should be tested, in a separate program.

There is a bit of mathematics in the exercise, but I will explain that.

There are 3 mathematical terms in the exercise. The Einstein tips will give you the needed explanation.

We will create some functionality that already exists, but it is an exercise.

We will create the Least Common Multiple and the Greatest Common Divisor of 2 numbers.

The class BigInteger has a method "GreatestCommonDivisor". I'm well aware.

I want you to create it by yourself.

Part 1. Ask 2 numbers

Your software should ask for 2 numbers.

The result will be showing the greatest common divisor and the least common multiple of the 2 numbers.

So asking a number, again, should here be in a class, that can be reused in all your other programs. Normally you have that already done in previous exercises. You can even put the class in another project (Class library).

Notes

Notes

.....

.....

.....

.....

.....

.....

.....

COPY PASTE)

Part 2. Factorize a number into a list of prime numbers

The number you have entered must result in a list. We will factorize the number into a list of prime numbers.

See Wikipedia of what is meant, but I will explain it with an example. https://simple.wikipedia.org/wiki/Fundamental_theorem_of_arithmetic

Every number can be written as a multiplication of prime numbers. I give another example than the Wikipedia page.

- Suppose I have 100. One hundred can be divided by 2. Two is the first prime number.
- So 100 = 2 * 50. (Remember the 2, by putting it in a list).
- We continue with 50.
- So 50 = 2 * 25 (Remember the 2 again, by putting it in the list).
- We continue with 25.
- The next prime number is 3, but 25 is not divisible by 3, so the next prime will be used and that is 5.
- So 25 = 5 * 5 (Remember the 5 in the list)
- We continue with 5.
- 5 = 5 * 1 (Remember the 5 in the list.
- We stop, because we have reached 1.

So 100 can be written as 2 * 2 * 5 * 5.

If you follow the same logic for 210 you will notice that 210 can be written as 2 * 3 * 5 * 7.

Write a class with the functionality that a number can be converted into a list of primes.

Part 3. Calculate the greatest common divisor of 2 numbers

- Get the list of primes for the 2 numbers.
  - 100 → 2 * 2 * 5 * 5
  - 210 → 2 * 3 * 5 * 7

- Generate with the 2 lists another list.
  - Every prime number that is common, should be put in the new list.
  - Remove or mark that prime from the 2 original lists as been taken.
    - If primes are twice common, they should be put twice in the new list.
  - In this example we have a 2 and a 5 that are common.
- Multiply all the numbers in that list, and your result will be 10.
  - 10 is the greatest common divisor of 100 and 210.
  - 10 = 2 * 5.

Part 3. Calculate the least common multiple of 2 numbers

- Get the list of primes for the 2 numbers.
  - 100 → 2 * 2 * 5 * 5.
  - 210 → 2 * 3 * 5 * 7.
- Generate with the 2 lists another list.
  - Every prime number that is common, should be put in the new list.
  - Remove or mark that prime from the 2 original lists as been taken.
    - If primes are twice common, they should be put twice in the new list.
  - Add all not taken primes of the first list and put them in the new list.
  - Add all not taken primes of the second list and put them in the new list.
  - In this example we have a 2 and a 5 that are common, added with another 2, 3, 5 and 7.
- Multiply all the numbers in that list, and your result will be .
  - 2100 is the greatest common divisor of 100 and 210.
  - 2100 = 2 * 2 * 3 * 5 * 5 * 7.

Notes

Class Exercises in framework Karel the Robot

Solve them in Visual Studio.

Example 09999-e Karel the Robot.zip contains all demos given during the courses.

You can adapt, extend the existing code by solving those exercises. Create a new GitHub repository to save your progress. Make me contributor if it, when you want me to evaluate the exercises.

Exercise 14.08:

-
  -

Notes