Міністерство освіти і науки України
Національний технічний університет України
Київський Політехнічний Інститут імені Ігоря Сікорського
Кафедра ОТ

Лабораторна робота № 7

з дисципліни "Основи Web-програмування"

Виконав:
студент 2-го курсу
Групи: ІП-64 ФІОТ
Гордієнко Нікіта
Заліковка: №6404

Київ-2018

Постановка задачі до комп’ютерного практикуму № 7

При виконанні комп’ютерого практикуму слід реалізувати наступні задачі:

a) Перезавантажити віртуальний метод bool Equals (object obj), таким чином, щоб об‘єкти були рівними, якщо рівні всі дані об‘єктів. Для кожного з класів самостійно визначити, які атрибути використовуються для порівняння;
b) Визначити операції == та != . При цьому врахувати, що визначення операцій повинно бути погоджено з перезавантаженим методом Equals, тобто критерії, за якими перевіряється рівність об'єктів в методі Equals, повинні використовуватися і при перевірці рівності об'єктів в операціях == та !=;
c) Перевизначити віртуальний метод int GetHashCode(). Класи базової бібліотеки, що викликають метод GetHashCode() з призначеного користувальницького типу, припускають, що рівним об'єктів відповідають рівні значення хеш-кодів. Тому в разі, коли під рівністю об'єктів розуміється збіг даних (а не посилань), реалізація методу GetHashCode() повинна для об'єктів з однаковими даними повертати рівні значення хеш-кодів.
d) Визначити метод object DeepCopy() для створення повної копії об'єкта. Визначені в деяких класах базової бібліотеки методи Clone() та Copy() створюють обмежену (shallow) копію об'єкта - при копіюванні об'єкта копії створюються тільки для полів структурних типів, для полів, на які посилаються типи, копіюються тільки посилання. В результаті в обмеженій копії об'єкта поля-посилання вказують на ті ж об'єкти, що і в вихідному об'єкті. Метод DeepCopy() повинен створити повні копії всіх об'єктів, посилання на які містять поля типу. Після створення повна копія не залежить від вихідного об'єкта - зміна будь-якого поля або властивості вихідного об'єкта не повинно призводити до зміни копії. При реалізації методу DeepCopy() в класі, який має поле типу System.Collections.ArrayList, слід мати на увазі, що визначені в класі ArrayList конструктор ArrayList (ICollection) і метод Clone() при створенні копії колекції, що складається з елементів, на які посилаються типів, копіюють тільки посилання. Метод DeepCop () повинен створити як копії елементів колекції ArrayList, так і повні копії об'єктів, на які посилаються елементи колекції. Для типів, що містять колекції, реалізація методу DeepCopy()

спрощується, якщо в типах елементів колекцій також визначити метод DeepCopy ().

e) Перезавантажити віртуальний метод string ToString() для формування строки з інформацією про всі елементи списку
f) Підготувати демонстраційний приклад, в котрому будуть використані всі розроблені методи
g) Підготувати звіт з результатами виконаної роботи.

При виконанні комп’ютерого практикуму слід реалізувати наступні задачі:

a) Визначити клас, котрий містить типізовану колекцію та котрий за допомогою подій інформує ппо зміни в колекції.

   Колекція складається з об'єктів силочних типів. Колекція змінюється при видаленні/додаванні елементів або при зміні одного з вхідних в колекцію посилань, наприклад, коли одному з посилань присвоюється нове значення. В цьому випадку у відповідних методах або властивості класу кидаються події.

   При зміні даних об'єктів, посилання на які входять в колекцію, значення самих посилань не змінюються. Цей тип змін не породжує подій.

   Для подій, що сповіщають про зміни в колекції, визначається свій делегат. Події реєструються в спеціальних класах-слухачах.
b) Реалізувати обробку помилок, при цьому необхідно перевизначити за допомогою наслідування наступні події:
   1) StackOverflowException
   2) ArrayTypeMismatchException
   3) DivideByZeroException
   4) IndexOutOfRangeException
   5) InvalidCastException
   6) OutOfMemoryException
   7) OverflowException
c) Підготувати демонстраційний приклад, в котрому будуть використані всі розроблені методи
d) **Підготувати звіт з результатами виконаної роботи.**

Власний варіант завдання:

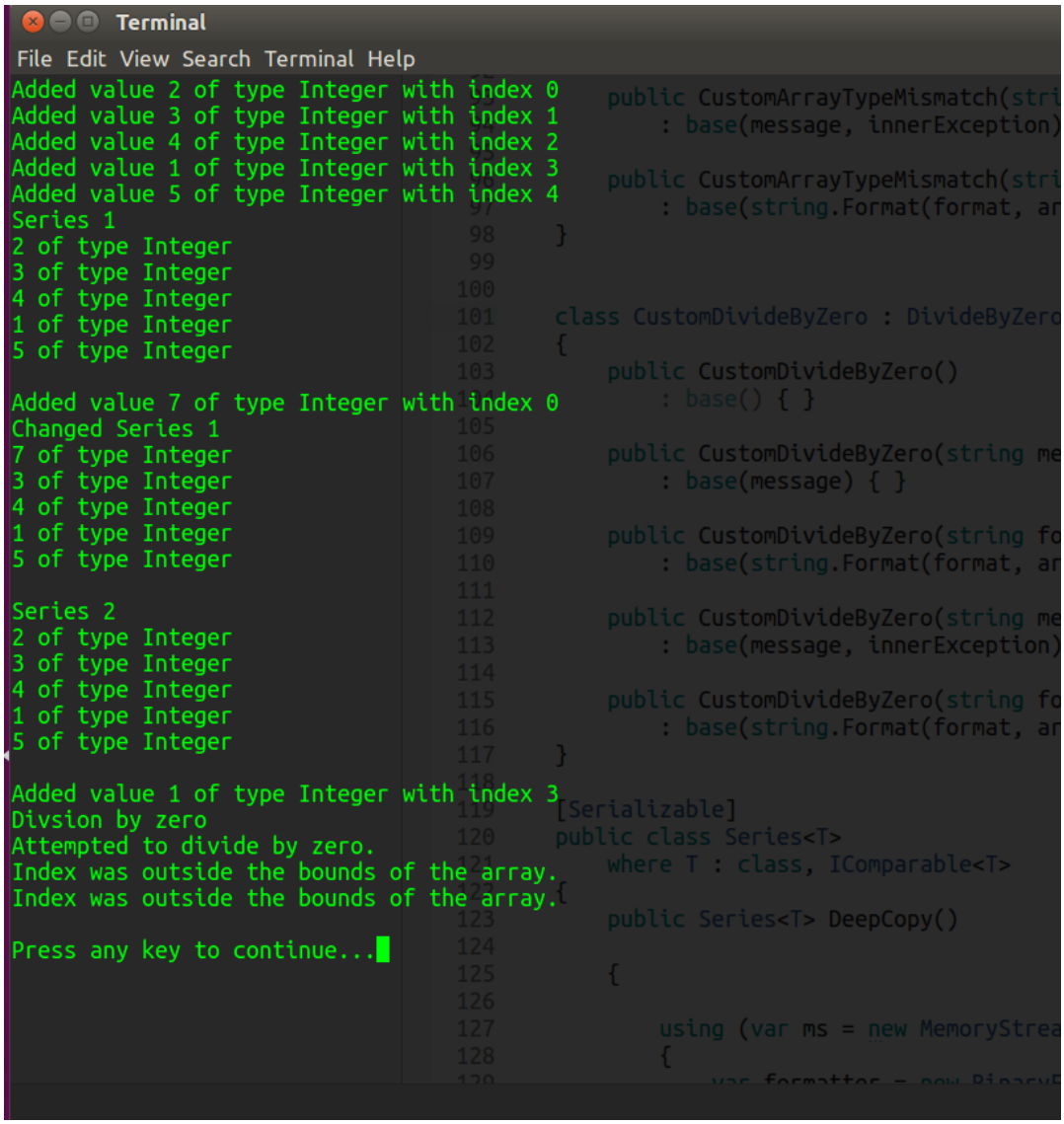
Вариант 5

Создать абстрактный класс Number с виртуальными методами, реализующими арифметические операции. На его основе реализовать классы Integer и Real.

Создать класс Series (набор), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти.

Предусмотреть возможность вывода характеристик объектов списка.

Скріншоти:


Terminal
File Edit View Search Terminal Help
Added value 2 of type Integer with index 0
Added value 3 of type Integer with index 1
Added value 4 of type Integer with index 2
Added value 1 of type Integer with index 3
Added value 5 of type Integer with index 4
Series 1
2 of type Integer
3 of type Integer
4 of type Integer
1 of type Integer
5 of type Integer
Added value 7 of type Integer with index 0
Changed Series 1
7 of type Integer
3 of type Integer
4 of type Integer
1 of type Integer
5 of type Integer
Series 2
2 of type Integer
3 of type Integer
4 of type Integer
1 of type Integer
5 of type Integer
Added value 1 of type Integer with index 3
Divsion by zero
Attempted to divide by zero.
Index was outside the bounds of the array.
Index was outside the bounds of the array.
Press any key to continue...

Код Програми:

```
using System;
using System.Linq;
using System.Reflection;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

namespace laba2
{
   internal class Program
   {
      private static void Main()
      {
         var series = new Series<Integer>(5);

         series[0] = new Integer(2);
         series[1] = new Integer(3);
         series[2] = new Integer(4);
         series[3] = new Integer(1);
         series[4] = new Integer(5);


         Console.WriteLine("Series 1");
         Console.WriteLine(series);
         var series2 = series.DeepCopy();
         series[0] = series[1] + series[2];
         Console.WriteLine("Changed Series 1");
         Console.WriteLine(series);
         Console.WriteLine("Series 2");
         Console.WriteLine(series2);

         series[3] = series[2] / series[1];
         var series3 = new Series<Real>(5);
         try
         {
            series3[3] = series[2].ToReal() / new Real(0);
         }
         catch (CustomDivideByZero e)
         {
            Console.WriteLine(e.Message);
         }


         try
         {
            series3[50] = series[2].ToReal() / new Real(2);
         }
         catch (IndexOutOfRangeException e)
         {
            Console.WriteLine(e.Message);
```

```
        }
        var series4 = new Series<Integer>(0);
        try
        {
           series4.OrderBy();
        }
        catch (IndexOutOfRangeException e)
        {
           Console.WriteLine(e.Message);
        }
     }
  }


  class CustomInvalidCast : InvalidCastException
  {
     public CustomInvalidCast()
        : base() { }

     public CustomInvalidCast(string message)
        : base(message) { }

     public CustomInvalidCast(string format, params object[] args)
        : base(string.Format(format, args)) { }

     public CustomInvalidCast(string message, Exception innerException)
        : base(message, innerException) { }

     public CustomInvalidCast(string format, Exception innerException, params
object[] args)
        : base(string.Format(format, args), innerException) { }
  }

  class CustomArrayTypeMismatch : ArrayTypeMismatchException
  {
     public CustomArrayTypeMismatch()
        : base() { }

     public CustomArrayTypeMismatch(string message)
        : base(message) { }

     public CustomArrayTypeMismatch(string format, params object[] args)
        : base(string.Format(format, args)) { }

     public CustomArrayTypeMismatch(string message, Exception innerException)
        : base(message, innerException) { }

     public CustomArrayTypeMismatch(string format, Exception innerException,
params object[] args)
        : base(string.Format(format, args), innerException) { }
  }
```

```
  class CustomDivideByZero : DivideByZeroException
  {
    public CustomDivideByZero()
      : base() { }

    public CustomDivideByZero(string message)
      : base(message) { }

    public CustomDivideByZero(string format, params object[] args)
      : base(string.Format(format, args)) { }

    public CustomDivideByZero(string message, Exception innerException)
      : base(message, innerException) { }

    public CustomDivideByZero(string format, Exception innerException, params
object[] args)
      : base(string.Format(format, args), innerException) { }
  }

  [Serializable]
  public class Series<T>
    where T : class, IComparable<T>
  {
    public Series<T> DeepCopy()

    {

      using (var ms = new MemoryStream())
      {
        var formatter = new BinaryFormatter();
        formatter.Serialize(ms, this);
        ms.Position = 0;

        return (Series<T>)formatter.Deserialize(ms);
      }

    }
    private static void Show_Message(string message)
    {
      Console.WriteLine(message);
    }
    public delegate void AddedNewValue(string message);
    public event AddedNewValue Added;


    private T[] array;

    public Series(int size)
    {
      this.Added += Show_Message;
```

```
    array = new T[size];
}

public int Length
{
    get { return array.Length; }
}

public void OrderBy(bool desc = false)
{
    if(desc == true)
        Sort((x, y) => x.CompareTo(y) < 0);
    else
        Sort((x, y) => x.CompareTo(y) > 0);
}

protected void Sort(Func<T, T, bool> func)
{
    if (Length == 0)
        throw new IndexOutOfRangeException();
    for (var i = 0; i < Length - 1; i++)
        for (var j = i + 1; j < Length; j++)
            if (func(array[i], array[j]))
            {
                var temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
}

public T this[int index]
{
    get {
        if (index >= Length)
            throw new IndexOutOfRangeException();
        return array[index];
    }

    set { if (Added != null)
        {
            if (index >= Length)
                throw new IndexOutOfRangeException();
            Added($"Added value {value} with index {index}");
        }
        array[index] = value; }
}
public override string ToString()
{
    string res = "";
    for (var i = 0; i < Length; i++)
        res = string.Concat(res, string.Concat(array[i] + "\n"));
    return res;
```

```
    }

}
[Serializable]
public class Real : Number<double>
{
    public Real(double value = 0)
    {
        Value = value;
    }

    public static Real operator +(Real one, Real other)
    {
        return new Real(one.Value + other.Value);
    }
    public static Real operator -(Real one, Real other)
    {
        return new Real(one.Value - other.Value);
    }
    public static Real operator *(Real one, Real other)
    {
        return new Real(one.Value * other.Value);
    }
    public static Real operator /(Real one, Real other)
    {
        if (DoubleEquals(other.Value, 0))
        {
            Console.WriteLine("Divsion by zero");
            throw new CustomDivideByZero();
        }
        return new Real(one.Value / other.Value);
    }
    public override bool Equals(object obj)
    {
        if (!(obj is Real))
            return false;
        else
            return DoubleEquals(((Real)obj).Value, Convert.ToDouble(Value));


    }
    public Real DeepCopy()
    {
        Real obj = new Real(this.Value);
        return obj;
    }

    private static bool DoubleEquals(double a, double b, double epsilon = 0.0001)
    {
        return Math.Abs(a - b) < epsilon;
    }
    public override int GetHashCode()
```

```
    {
        return Convert.ToInt32(this.Value);
    }

    public static bool operator ==(Real one, Real two)
    {
        if ((Object)one == null || (Object)two == null)//проверить на null
            return false;

        return one.Equals(two);
    }
    public static bool operator !=(Real one, Real two)
    {
        if ((Object)one == null || (Object)two == null)//проверить на null
            return true;

        return !one.Equals(two);
    }
}
[Serializable]
public class Integer : Number<int>
{

    public Integer(int value = 0)
    {
        Value = value;
    }

    public static Integer operator +(Integer one, Integer other)
    {
        return new Integer(one.Value + other.Value);
    }

    public static Integer operator -(Integer one, Integer other)
    {
        return new Integer(one.Value - other.Value);
    }
    public static Integer operator *(Integer one, Integer other)
    {
        return new Integer(one.Value * other.Value);
    }
    public static Integer operator /(Integer one, Integer other)
    {

        if (other.Value == 0)
        {
            Console.WriteLine("Divsion by zero");
            throw new CustomDivideByZero();
        }
        return new Integer(one.Value / other.Value);
    }
```

```
    public Real ToReal()
    {
        return new Real(Convert.ToDouble(this.Value));
    }

    public override bool Equals(object obj)
    {
        if (!(obj is Number<int>))
            return false;
        else
            return ((Number<int>)obj).Value == this.Value;

    }
    public Integer DeepCopy ()
    {
        Integer obj = new Integer(this.Value);
        return obj;
    }

    public override int GetHashCode()
    {
        try
        {
            return Convert.ToInt32(this.Value * 10000);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            return -1;
        }
    }

    public static bool operator ==(Integer one, Integer two)
    {
        if ((Object)one == null || (Object)two == null)//проверить на null
            return false;

        return one.Equals(two);
    }
    public static bool operator !=(Integer one, Integer two)
    {
        if ((Object)one == null || (Object)two == null)//проверить на null
            return true;

        return !one.Equals(two);
    }
}

[Serializable]
public abstract class Number<T> : IComparable<Number<T>>
    where T : struct, IComparable<T>
{
```

```
        public T Value { get; protected set; }
        public int CompareTo(Number<T> other)
        {

            if (other == null)
               throw new CustomInvalidCast();
            return Value.CompareTo(other.Value);

        }

        public override string ToString()
        {
           return string.Format("{0} of type {1}", Value, GetType().Name);
        }
        //public abstract Number<T> DeepCopy();


    }
}
```