Day 03 .NET and C# Exercises

1) Struct with properties
Make a ConsoleApplication.
Implement a struct called Time, representing a time of day 00:00-23:59.
Internally the time is stores as minutes since midnight in a private field.
F.ex. 623 minutes is 10 hour and 23 minutes.

Declare the struct Time with the private field.
Declare two properties for getting and setting Hour and Min. Be careful.

Add a constructor with a string parameter e.g. ”01:23”.
Add another constructor with 2 int parameters for hour and minutes.

Throw an ArgumentException if a value set is illegal (less than 0 or greater than 23 for Hour)

Add a ToString() method returning the time of day as f.ex. ”10:23”, the signature is

```
public override string ToString()
```

Test your struct in Main:

```
Time t1, t2;
t1=new Time("08:30");
t2=t1;
t2.Hour=t2.Hour+2; // hvis t2 passerer 23:59 fratrækkes 24 timer
Console.WriteLine(t1.ToString());
Console.WriteLine(t2.ToString());

Console.ReadLine();
```

2) Operator overloading
Tilføj operator overloading for + og - til Time struct'en løsning fra opgave 1, således at kan skrive

Ex. 10:20 + 11:10 = 21:30 16:30 – 06:20 = 10:10
22:30 + 04:10 = 02:40 11:20 – 13:20 = 22:00

Husk ved + at hvis 23:59 overskrides fratrækkes 1440 minutter; og ved – at hvis antal minutter bliver negativ adderes 1440 minutter.

Har du tid så tilføj også operator overloading for operatorerne `<`, `>`, `==` og `!=`

*** 3)** Parameteroverførsel – og equals på struct

Forklar output fra følgende Console Application (Output er vist efter koden så de behøver ikke at kører programmet i VS).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{

  public struct Person {
    public string Name { get; set; }
    public int Nr { get; set; }

    public override string ToString()
    {
      return Name+"-"+Nr;
    }
  }

  class Program
  {
    public static void ValParam(Person p) {
      p = new Person { Name = "Berit" };
    }

    public static void RefParam(ref Person p)
    {
      p = new Person { Name = "Berit" };
    }

    public static void ValCopy(Person s1, Person s2)
    {
      s2 = s1;
    }

    public static void RefCopy(Person s1, ref Person s2) {
      s2 = s1;
    }


    static void Main(string[] args)
    {
      Person p=new Person { Name="Axel" };
      ValParam(p);
      Console.WriteLine(p.Name);

      RefParam(ref p);
      Console.WriteLine(p.Name);

      Console.WriteLine("----------------------------");

      Person s1 = new Person { Name = "Dorit", Nr=1 };
      Person s2 = new Person { Name = "Erik", Nr=2 };

      ValCopy(s1, s2);
      Console.WriteLine(s1+" "+s2+"   "+s1.Equals(s2));

      RefCopy(s1, ref s2);
      Console.WriteLine(s1 + " " + s2 + "   " + s1.Equals(s2));

      Console.WriteLine("----------------------------");

      s1.Nr = 101; s2.Nr = 102;

      Console.WriteLine(s1 + " " + s2 + "   " + s1.Equals(s2));

      Console.ReadLine();
    }
  }
}
```

Output:

```
Axel
Berit
----------------------------
Dorit-1 Erik-2   False
Dorit-1 Dorit-1   True
----------------------------
Dorit-101 Dorit-102   False
```

4) Indexer

Create a class (In a Console Application)

```
    public class MonthlySales
    {
       int[] sales = new int[12];

    }
```

We don't want to access January sales index 0 but index 1, or indeks "Jan". Therefore we want add an indekser to MonthlySales, well actyally two indekser, such that we can write like:

```
     MonthlySales sales = new MonthlySales();

     sales[12]=24300;        // set sales in December to 24300
     sales["Aug"]=12800;     // set sales in August to 12800
     int feb=sales["Feb"];   // get sales in February
```

Test the class in the Main method.

5) Indexer

Write a class GrowableArray (in a Console Application), which has a private field of type int[]-array.
GrowableArray should have an indexer, such that an object of GrowableArray can be used as shown below

```
GrowableArray g=new GrowableArray();
g[0]=7;
g[3]=9;
g[8]=13;
Console.WriteLine(g[3]);
```

A facility for the indexer is that if a new element is assigned to an index outside the current array, then a new Array of the necessary size is allocated, all values from the old is copyed to the new array, the new value is assigned, and finally the new array replaces the old array in the private field of GrowableArray.

Test the class.

6) Generics

Take a copy of your complete solution of exercise 5 and make the class generics such that the private array can be of any type.

Test your new modified class by using GrowableArray<double>, GrowableArray<string> or GrowableArray<DateTime>.