# Homework: Other Types in OOP (Enumerations, Structures, Generic Classes, Attributes)

This document defines the homework assignments from the ["OOP" Course @ Software University](https://softuni.bg/courses/oop/). Please submit as homework a single zip / rar / 7z archive holding the solutions (source code) of all below described problems. The solutions should be written in C#.

## Galactic GPS

Create a struct **Location** that holds fields of type double **latitude** and **longitude** of a given location. Create an enumeration **Planet** that holds the following constants: Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune.

* Add an enum field of type **Planet*.*** Encapsulate all fields. Validate data (search in Internet what are the valid ranges for latitude and longitude).
* Add a constructor that holds 3 parameters: **latitude**, **longitude** and **planet**.
* Override **ToString()** to print the current location in the format **<latitude>, <longitude> - <location>**

|  |  |
| --- | --- |
| **Sample Source Code** | **Output** |
| Location home = new Location(18.037986, 28.870097, Planet.Earth);  Console.WriteLine(home); | 18.037986, 28.870097 - Earth |

## Fraction Calculator

Create a **struct** **Fraction** that holds the **numerator** and **denominator** of a fraction as fields. A fraction is the division of two rational numbers (e.g. **22 / 7**, where 22 is the numerator and 7 is the denominator).

* The struct constructor takes the numerator and denominator as arguments. They are integer numbers in the range [-9223372036854775808 … 9223372036854775807].
* Validate the input through properties. The denominator cannot be 0. Throw proper exceptions with descriptive messages.
* Overload the **+** and **-** operators to perform **addition** and **subtraction** on objects of type Fraction. The result should be a **new Fraction**.
* Override **ToString()** to print the fraction in floating-point format.

|  |  |
| --- | --- |
| **Sample Source Code** | **Output** |
| Fraction fraction1 = new Fraction(22, 7);  Fraction fraction2 = new Fraction(40, 4);  Fraction result = fraction1 + fraction2;  Console.WriteLine(result.Numerator);  Console.WriteLine(result.Denominator);  Console.WriteLine(result); | 368  28  13.142857142857142857142857143 |

## Generic List

Write a generic class **GenericList<T>** that keeps a list of elements of some parametric type **T**.

* Keep the elements of the list in an **array with a certain capacity**, which is given as an optional parameter in the class constructor. Declare the default capacity of 16 as a constant.
* Implement methods for:
  + **adding** an element
  + **accessing** element by index
  + **removing** element by index
  + **inserting** element at given position
  + **clearing** the list
  + **finding** element index by given value
  + checking if the list **contains** a value
  + **printing** the entire list (override **ToString()**).
* Check all input parameters to avoid accessing elements at invalid positions.
* Implement **auto-grow functionality**: when the internal array is full, create a new array of double size and move all elements to it.
* Create generic methods **Min<T>()** and **Max<T>()** for finding the minimal and maximal element in the **GenericList<T>**. You may need to add generic constraints for the type **T** to implement **IComparable<T>**.

## Generic List Version

Create a **[Version]** attribute that can be applied to structures, classes, interfaces, enumerations and methods and holds a version in the format **major.minor** (e.g. 2.11). Apply the version attribute to **GenericList<T>** class and display its version at runtime.