# Homework: Common Type System

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#.

## Value and Reference Types

Describe the differences between value and reference types. Answer the following questions:

* Which .NET data types are **value types**? Which types are **reference types**?
* How do **value** and **reference** types differ when **assigned to a variable**?
* How do **value** and **reference** types behave when **passed as argument to a method**?
* Where **in memory** are **value types allocated**? Where are **reference types allocated**?
* What is data **mutation**? Can **value** types be **mutated**? Can **reference** types be **mutated**?
* What is the difference between **mutating** and **setting** (assigning) a variable? Give examples.
* What are the **out** and **ref** keywords used for? In what do they differ?

Submit your answers as a **.txt** file as part of your homework.

## Customer

Define a class Customer, which contains data about a customer – **first name**, **middle** **name** and **last name**, **ID** (EGN), **permanent address**, **mobile phone**, **e-mail**, list of **payments** and **customer type**.

* Define a class **Payment** which holds a **product name** and **price**.
* Define an enumeration for the customer type, holding the following types of customers: **One-time** , **Regular**, **Golden**, **Diamond**.
* Override the standard methods, inherited by **System.Object**: **Equals()**, **ToString()**, **GetHashCode()** and operators **==** and **!=**.
* Implement the **ICloneable** interface. The **Clone()** method should make a deep copy of all object fields into a **new** object of type **Customer**.
* Implement the **IComparable<Customer>** interface to compare customers by **full name** (as first criteria, in lexicographic order) and by **ID** (as second criteria, in ascending order).

## String Disperser

Define a class **StringDisperser**.

* The constructor should take several strings as arguments.
* Override the standard methods, inherited by **System.Object**: **Equals()**, **ToString()**, **GetHashCode()** and operators **==** and **!=**.
* Implement the **ICloneable** interface. The **Clone()** method should make a deep copy of all object fields into a **new** object of type **StringDisperser**.
* Implement the **IComparable<StringDisperser>** interface to compare string dispersers by their **total string value** lexicographically.
* Implement the **IEnumerable** interface to allow **foreach** on objects of type **StringDisperser**. The items returned should be the **characters** of each string.

|  |  |
| --- | --- |
| **Input** | **Output** |
| StringDisperser stringDisperser = new StringDisperser("gosho", "pesho", "tanio");  foreach (var ch in stringDisperser)  {  Console.Write(ch + " ");  } | g o s h o p e s h o t a n i o |

## \*\* Custom Tree

Define the data structure **binary search tree**. It should support the following operations:

* **Adding** a new element
* **Searching** elements
* **Deleting** elements

It is not necessary to keep the tree balanced. Implement the standard methods from **System.Object** – **ToString()**, **Equals()**, **GetHashCode()** and the operators for comparison **==** and **!=**. Implement **IEnumerable<T>** to traverse the tree.

**Add()** and implement the **ICloneable** interface for deep copy of the tree.

**Tip**: Define two separate types – structure **BinarySearchTree** (for the tree) and class **TreeNode** (for the tree elements).