Homework: Delegates and Events

This document defines the homework assignments from the "OOP" Course @ Software University. 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#.

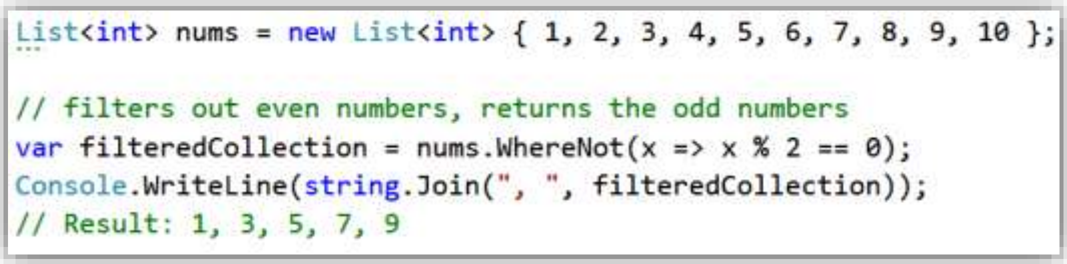
Problem 1. Custom LINQ Extension Methods

Create your own LINQ extension methods:

- **public static IEnumerable<T> WhereNot<T>(this IEnumerable<T> collection, Func<T, bool> predicate)**

Works just like **Where(predicate)** but filters the non-matching items from the collection.

Example:



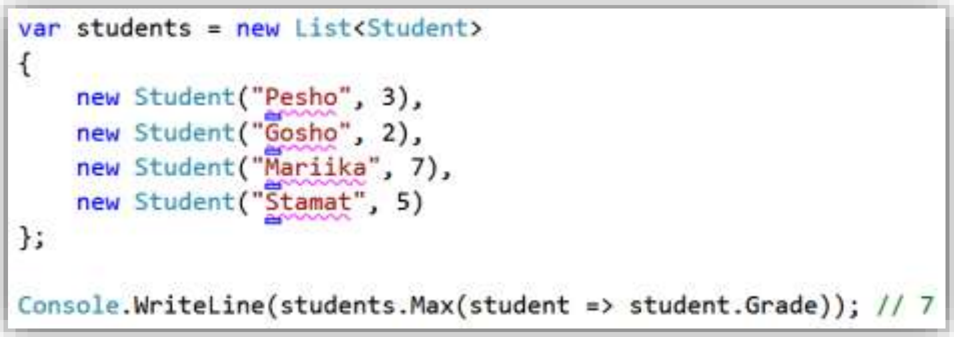
```
List<int> nums = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// filters out even numbers, returns the odd numbers
var filteredCollection = nums.WhereNot(x => x % 2 == 0);
Console.WriteLine(string.Join(", ", filteredCollection));
// Result: 1, 3, 5, 7, 9
```

- **public static TSelector Max<TSource, TSelector>(this IEnumerable<TSource>, Func<TSource, TSelector>)**

The Max method works on a collection of type TSource (analogous to T, can be any type: int, string, Student, etc.). It accepts a Func<TSource, TSelector>, where TSelector is another type. For example, if we have a class Student and every student has a name and a grade of type int, the Max method may be applied on a collection of students and could return the maximal grade in the list of students. The Func would take a student and get the Grade property, then the method would compare all students' grades and return the largest.

Example:



```
var students = new List<Student>
{
    new Student("Pesho", 3),
    new Student("Gosho", 2),
    new Student("Mariika", 7),
    new Student("Stamat", 5)
};

Console.WriteLine(students.Max(student => student.Grade)); // 7
```

Problem 2. Interest Calculator

Create a delegate **CalculateInterest** (or use **Func<…>**) with parameters **sum of money**, **interest** and **years** that calculates the interest according to the method it points to. The result should be rounded to 4 places after the decimal point.

- Create a method **GetSimpleInterest**(sum, interest, years). The interest should be calculated by the formula ***A = sum * (1 + interest * years)***.

- Create a method **GetCompoundInterest**(sum, interest, years). The interest should be calculated by the formula ***A = sum * (1 + interest / n)***$^{year * n}$, where **n** is the number of times per year the interest is compounded. Assume **n** is always 12.

Create a class **InterestCalculator** that takes in its constructor **money**, **interest**, **years** and **interest calculation delegate**. Using this class calculate the interest for the following input parameters:

Money	**Interest**	**Years**	**Type**	**Result**
500	5.6%	10	compound	874.1968
2500	7.2%	15	simple	5200.0000

Problem 3. Asynchronous Timer

Create a class **AsyncTimer** that executes a given method each **t** seconds.

- The constructor should accept 3 arguments: **Action** (a method to be called on every **t** milliseconds), **ticks** (the number of times the method should be called) and **t** (time interval between each tick in milliseconds).
- The main program's execution **should NOT be paused at any time** (use some kind of background execution).

Problem 4. Student Class

Write a class **Student** that holds name and age. Add an event **PropertyChanged** that should fire whenever a property of **Student** is changed, displaying a message in the format **Property changed: <property> (from <old value> to <new value>)**. Create a custom **PropertyChangedEventArgs** class to keep the property name, old value and new value. See the example below:

Sample Source Code

```
Student student = new Student("Peter", 22);
student.PropertyChanged += (sender, eventArgs) =>
{
    Console.WriteLine("Property changed: {0} (from {1} to {2})",
        eventArgs.PropertyName, eventArgs.OldValue, eventArgs.NewValue);
};
student.Name = "Maria";
student.Age = 19;
```

Sample Output

```
Property changed: Name (from Peter to Maria)
Property changed: Age (from 22 to 19)
```