Programming II
Theoretical test information

Autumn 2016

General information

The theoretical test will contain multiple choice, multiple response, matching and open response questions.

Date and time: November 7, 2016: 9:00 - 10:20

Rooms and subgroup distribution: TBA

You can prepare and bring to the test a **personal** helper document: 2 A4 pages max, formatted Courier 12pt. You can put any information you like inside this file. The document should be viewed in offline mode during the test.

The test will be taken through the Canvas system. Make sure you have access to your account.

During the test you can ask course instructors / teaching assistants for clarification / translation of the test questions.

The following activities are strictly prohibited and will result in a "0" grade for the test (cannot be retaken)

- Using electronic devices, taking pictures of test questions
- Using any course materials apart from the personal helper file
- Using Visual Studio or any other programming toolset
- Navigating to shared network folders
- Communicating with classmates through social networks or any other channels, posting question/question answers, etc.

Please note that a supervisor program will monitor and record all your actions at the computer during the test.

You are expected to know the fundamental syntax of the C# language related to object-oriented programming without the help from Visual Studio: declaring classes with fields, methods, properties (full and auto), events, initializing objects + C# syntax related to the test topics - see next section.

You are not expected to remember the exact names of methods inside standard .Net classes. A small number of typos will be allowed in open-response questions.

List of topics

1. OOP fundamentals: classes and objects, fields, methods, properties, constructors. Principles of OOP.
2. C# collections: Array, List, LinkedList, Dictionary: common features and differences. Main operations and their efficiency in different collections.
3. Delegates: named and anonymous delegates, delegate variables, events, lambda expressions
4. Abstract programming with interfaces: interface declaration and implementation in classes, standard .NET interfaces: IEnumerable, ICollection, IList, IDisposable
5. Inheritance: base and child classes, calling base class constructors, private, public and protected, virtual and override, abstract classes and methods. The "object" class and its members.
6. Templates and patterns: singleton, factory, dependency injection

Examples of questions

- A dictionary (associative container):
  - Contains key-value pairs
  - Can have duplicate keys
  - Can have duplicate values
  - Is used for fast search (lookup) by value
- What is the difference between a delegate instance and an event?
  - Events cannot be passed as method parameters
  - A delegate variable can be connected to a single handler only while events can be linked to multiple handler methods
  - Handlers for an event can only be assigned and deleted one by one from outer classes
  - An event declaration does not require a delegate type
- A C# program contains the following classes:

```
public class EmployeeManager
{
   public event Action<Employee> NewEmployeeAdded;
   ...
}

public class UI
```

```
{
   EmployeeManager _manager = new EmployeeManager();

   private void AssignHandlers() {
      ???(2)
   }

   ???(1)
}
```

Complete the code: replace ???(1) with an empty handler method for the NewEmployeeAdded event. In ???(2) assign the new method as a handler for the event in _manager.

- What can be put instead of ??? in the following code fragment?

```
using (var c = new ???())
{

}
```

  - Any reference type
  - Any type implementing IDisposable
  - Any value type
  - Any type implementing IEnumerable

- Which of the following 6 parts of the interface declaration are INCORRECT?

```
public interface IEmployeeRepository
{
    List<Employee> employees;                 // # 1

    IEnumerable<Employee> Employees { get; }  // # 2
    public void AddEmployee(Employee e);      // # 3
    void DeleteEmployee(Employee e) { }       // # 4
    void SaveAll();                           // # 5

    event OnEmployeeRemoved(Employee e);      // # 6
}
```