C# and Microsoft .NET

Classes and Objects

Trainer: Georgi Panayotov
E-mail: smg@smg-bg.net
Phone / Viber: +359877347912

PRAGMATIC
www.pragmatic.bg

Last time…

- Chess POC
  - Clearer after the second session?
  - …at least valuable exercise ☺
  - Maybe… OK, it **was** a bad strategy from the lector
- Feedback
  - Simpler examples about the theory
- Tic Tac Toe example **till the end of the week**
  - Much simpler than the Chess POC
  - Same structure
- Homework solutions **till the end of the week**
- Feedback form **till the end of the week**
- http://learn.pragmatic.bg
- Facebook group ☺
  - C# and .NET (October 2018)
  - https://www.facebook.com/groups/185158865702470/

Object-Oriented Programming (OOP)

Object-oriented programming (**OOP**) is a **programming** language model (paradigm) organized around **objects** rather than "actions" and data rather than logic. Historically, a **program** has been viewed as a logical procedure that takes input data, processes it, and produces output data.

Someone Important (a.k.a. Google)

OOP vs. Procedural Programming

- Code structure
- Maintainability
- Code reuse within / across projects
- Dealing with complexity
- Dealing with code duplication (aka Copy Paste Development ☺)
- etc.

OOP Principles

- Abstraction

  Hide all but relevant details from end users / programmers
- Encapsulation

  Hide the internal details from end users / programmers
- *Inheritance**
- *Polymorphism**

Encapsulation vs. Abstraaction

“**Encapsulation** is wrapping, just hiding properties **and** methods (implementation). **Encapsulation** is used for hiding the code **and** data in a single unit to protect the data from the outside the world. **Class** is the best example of **encapsulation**. **Abstraction** refers to showing only the necessary details to the intended user”

Someone Important (a.k.a. Google)

What is a class

- What is a class
  - A *class* is a construct that enables you to create your own custom **types** by grouping together variables of other types, methods and *events*
  - It defines the data / state and behaviour
  - Discrete model of a real-life entity
- What is an object
  - Instance of a class
  - Multiple instances of class
  - Instantiate an object (e.g. new keyword)

Class

```
access_modifier class class_name
{
   // fields & constants
   // properties
   // constructors
   // methods
   …
}
```

Access Modifiers

- Type (e.g. struct, class, *interface**)
- Type Scoped
  - Public
  - Internal
- Member Scoped
  - Public
  - Private
  - Internal
  - *Protected**
  - *Private protected**
  - *Internal Protected**
- Default Access Modifier

Class Anatomy

- Nested Types (structures, classes, enumerations, *interfaces**, ...)
- Constants and read-only fields (dynamic constants)
- Fields
- Properties
- *Indexers**
- *Events**
- Constructors
- *Finalizers**
- Methods
- *Operators**
- etc.

Fields and Constants

```
access_modifier class class_name
{
   access_modifier [const|readonly] data_type field_name;
   // properties
   // constructors
   // methods
}
```

Properties

```
access_modifier class class_name
{
   // fields and constants
   access_modifier data_type property_name
   {
            access_modifier get { return this. class_member; }
            access_modifier set { this.class_member = value; }
   }
   access_modifyer data_type property_name
   {
          access_modifier get;
          access_modifier set;
   }
   access_modifyer data_type property_name { get; }
  // constructors
  // methods
}
```

Constructors

```
access_modifier class class_name
{
   // fields and constants
   // properties

   void class_name() { }
   void class_name(param_type1 param1, param_type2
    param2,…) { }

   // methods
}
```

Methods

```
access_modifier class class_name
{
   // fields and constatns
   // properties
   // constructors
   access_modifier return_type method_name (parameters_list);
   access_modifier return_type method_name (parameters_list2);
}
```

Static

- What does the **static** keyword stand for?
- Static Fields, Constants and Properties
- Static Methods
- Static Constructors
- Static Classes
- Standard static classes in .NET
  - System.Console
  - System.Environment
  - System.Math

Base Class

- Inheritance*
- Object
  - Object() **/* constructor */**
  - bool Equals(Object objA, Object objB) **/* static */**
  - bool ReferenceEquals(Object objA, Object objB) **/* static */**
  - bool Equals(Object obj)
  - int GetHashCode()
  - Type GetType()
  - string ToString()
  - MemberwiseClone(); **/* protected */**

Namespaces

- What is a namespace
  - The **namespace** keyword is used to declare a scope that contains a set of related types
  - You can use a **namespace** to organize code elements and to create globally unique types
  - Used for logical organization of the source code
- Syntax
  - namespace namespace_name {}
  - namespace namespace_name.nested_namespace {}
- Default namespace for project
- Using **namespaces** (e.g. using keyword)

Class Libraries

- What is a Class Library
  - Defines namespaces with types (classes, structs, interfaces*, etc.) that can be reused across applications
  - Used for physical organization of the source code
- Project Templates
  - .NET Framework
  - .NET Core
  - .NET Standard
- Using Class Libraries / Project References


Questions?