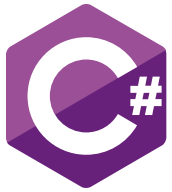
Classes, Structures and Enumerations

Agenda

- Classes
- Fields, constants and properties
- Structures
- Enumerations
- Assemblies
- Add a reference to a project
- Access modifiers
- Comments

Classes



Classes

It is a custom reference type

Almost anything can be represented by a class

A class = State (constants, fields, properties) + Behavior (methods)

Class declaration

```
namespace MyNamespace
{
    public class MyClass
    {
        // Class members
    }
}
```

Members

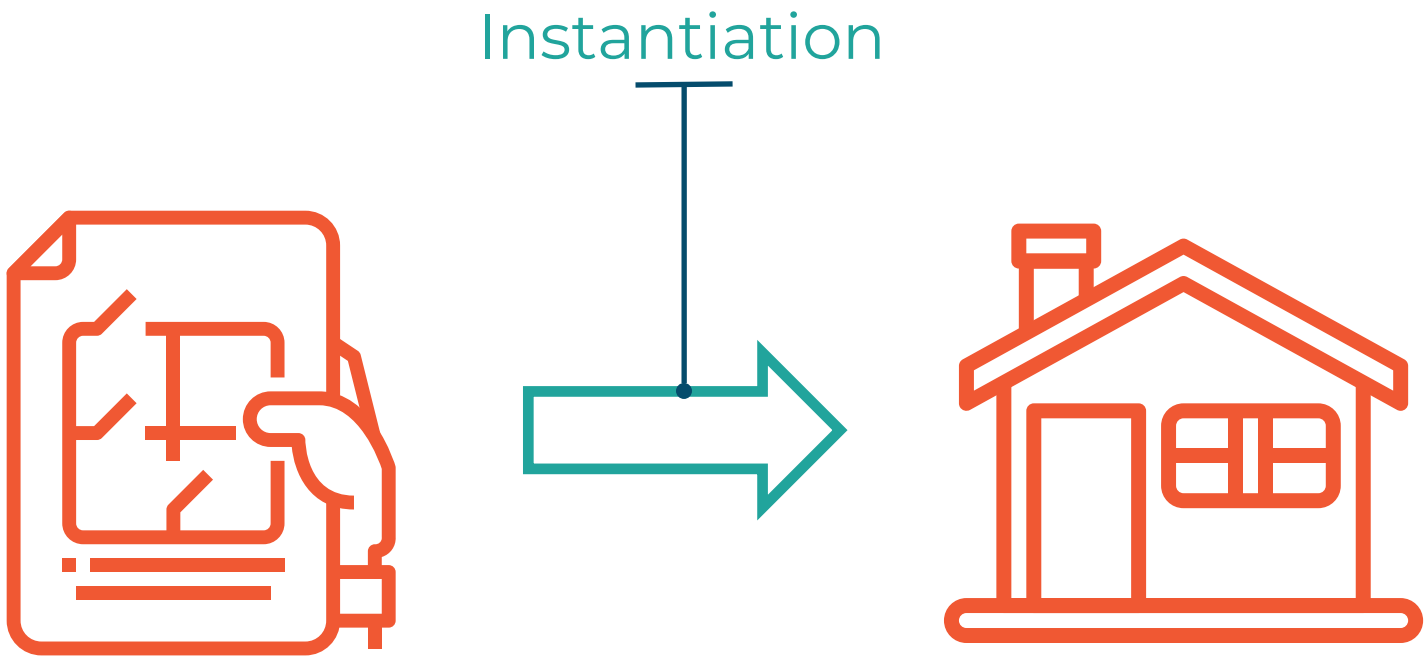
- Constants
- Fields
- Properties
- Constructors
- Methods

Object

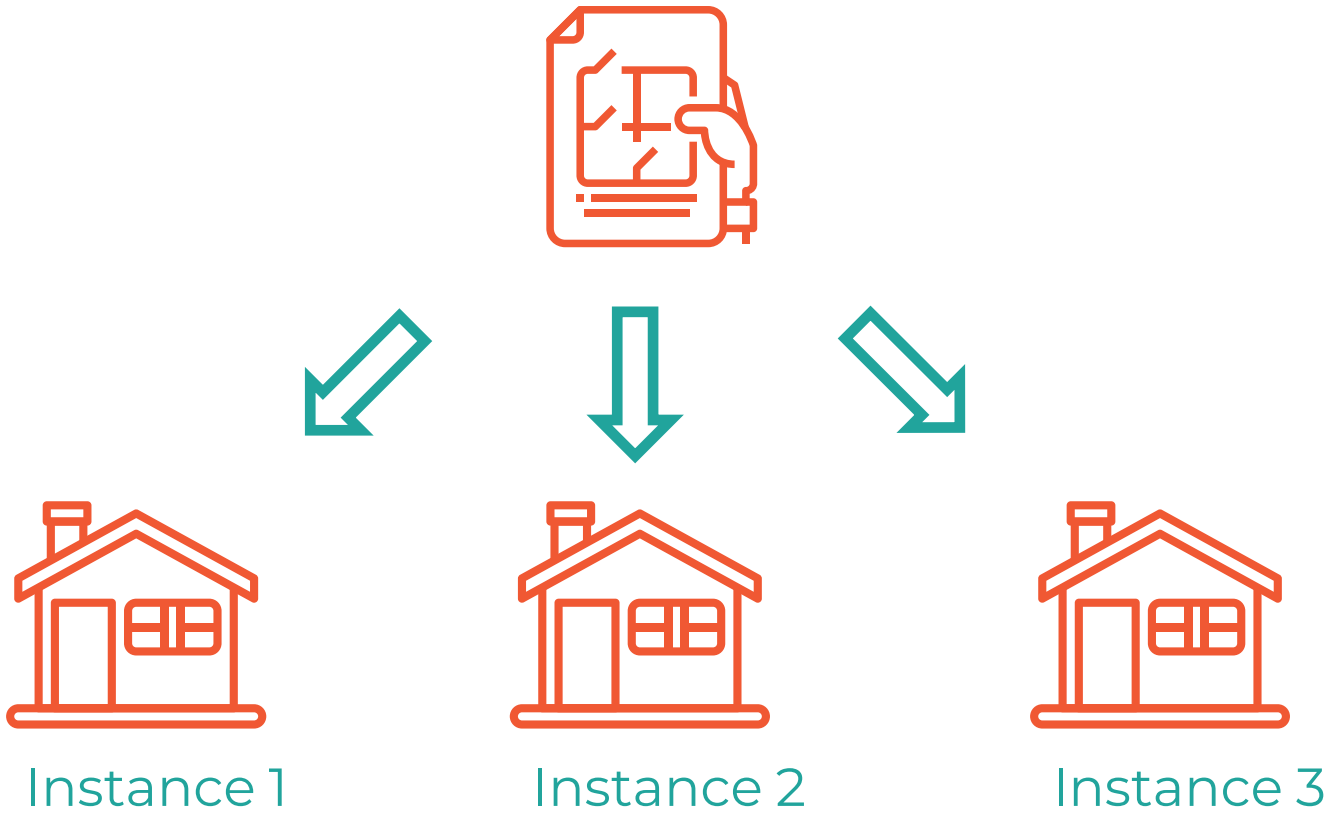
It is an instance of the class

Created with the new keyword

Class vs object



Instances



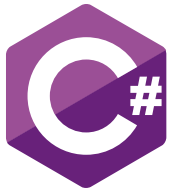
Object instantiation

```
// Explicit declaration
MyClass myObjectInstance = new MyClass();
// Implicit declaration
var myObjectInstance = new MyClass();
// Target-typed new expressions
MyClass myObjectInstance = new();
```

Demo

Create and instantiate classes

Constants, fields and properties



Field

Variable defined on a class or structure

Global variable of the type it belongs to

Should always have a private visibility

Backing field : private variable that holds the data that is exposed by a property

Field declaration

```
private string _title;
```

Constant

- Field that contains data that is unlikely to change
- User defined types, events and properties can’t be constants
- Must be initialized when declared
- Possible to change a constant visibility

Constant declaration

```
private const float ACCELERATION = 9.8f;
```

Properties

- Allows access to private fields
- Can perform a computation
- Get/Set property accessors
- You can simplify the syntax with auto-implemented properties

Property declaration

```
private string _myField;
public string  MyProperty
{
    get
    {
        return _myField;
    }

    set
    {
        _myField  = value;
    }
}
```

Property declaration with expression body definitions

```
private string _myField;
public string MyProperty
{
    get => _myField;
    set => _myField = value;
}
```

Auto-implemented property

```
public int Id { get; set; }
```

Object initializers

Allows to assign a value to public fields and properties

Object initializer

```
Movie movie = new Movie()
{
    Id = 1,
    Title = "Title 1"
};
```

Object initializer

```
var movie = new Movie()
{
    Id = 1,
    Title = "Title 1"
};
```

Object initializer

```
Movie movie = new()
{
    Id = 1,
    Title = "Title 1"
};
```

Use a field or property

```
int movieId = movie.Id;
```

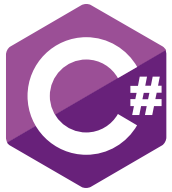
Set a field or property

```
movie.Id = 1;
```

Demo

Create constants, fields and properties

Structures



Structure

Like a class = State (constants, fields, properties) + Behavior (methods)

Structure

It is used in general to create small data centric types with little behavior

It’s a value type

Structure declaration

```
namespace MyNamespace
{
    public struct MyStructure
    {
        // Struct members
    }
}
```

Structure

It can’t inherit from a class or struct and be inherited

Interface support only

No parameter less constructor

You can't initialize an instance field or property at its declaration

Consider defining a struct instead of a class if instances of the type are small and commonly short-lived or are commonly embedded in other objects

.NET Documentation

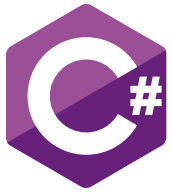
Choose between class and struct

https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/choosing-between-class-and-struct

Demo

Create and instantiate a structure

Enumerations



Enumerations

Represent a choice from a set of mutually exclusive values

Enumerations

Created with the `enum` keyword

Set of named constants

Each value is associated with an integer value

Enums

```
public enum Planet
{
    Mercury,
    Venus,
    Earth,
    Mars,
    Jupiter,
    Saturn,
    Neptune,
    Uranus
}
```

Enum class

Base class for all enumerations

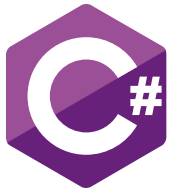
Provides useful methods :
`IsDefined, GetValues, Parse, TryParse, ToObject`

Demo

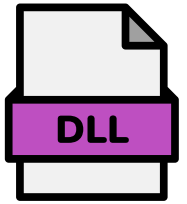
Create enumerations

Use the Enum static methods

Assemblies



Assemblies

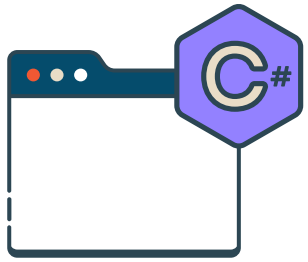


An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality

.NET Documentation

Project/Assembly

Project folder

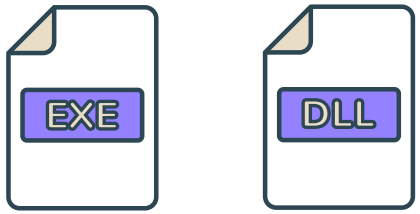


publish

.NET project

- Console app
- ASP.NET Core app
- Class library ...

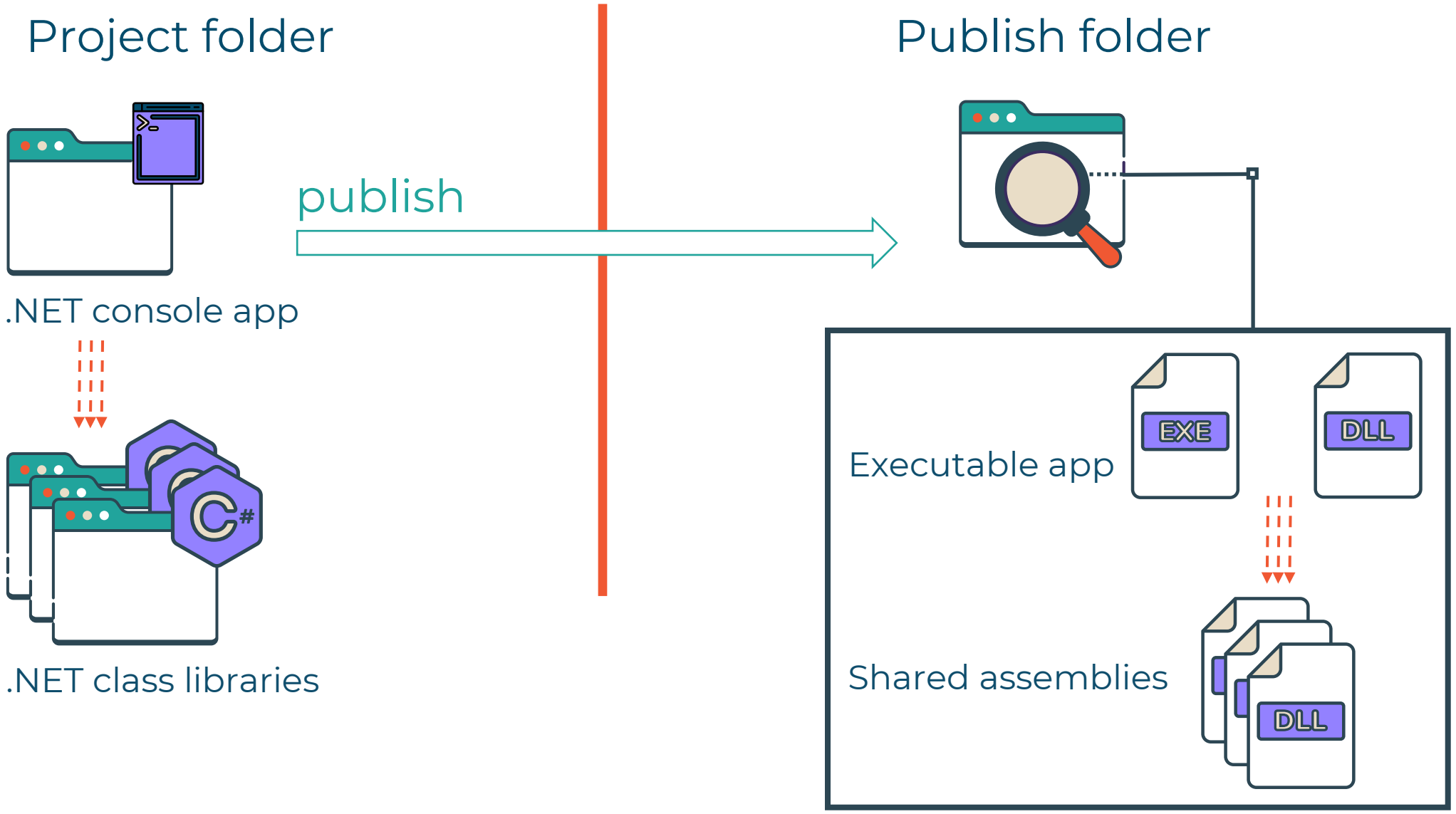
Publish folder



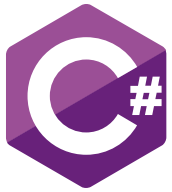
Deployment unit

- EXE
- DLL (Class library)

Project vs Assembly



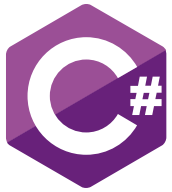
Add a reference to a class library



Demo

Add project references to a class library and publish the app

Access modifiers



Access modifiers

The accessibility level defines how a type or type member can be accessed from its containing assembly or from an assembly that references it

Type access modifier

```
namespace MyNamespace
{
    public class MyClass
    {
        // code
    }
}
```

Member access modifier

```
namespace MyNamespace
{
    public class MyClass
    {
        private string _privateField;
    }
}
```

Public

The type or member can be accessed by any other code in the same assembly or another assembly that references it

Private

The type or member can be accessed only by code inside the type they are declared in

Example

```
namespace MyNamespace
{
    public class MyClass
    {
        private string _privateField;
        public void MyMethod()
        {
            string privateField = _privateField;
            // use privateField
        }
    }
}
```

Protected

The type or member can be accessed only by code in the same class, or in a class that is derived from that class

Internal

The type or member can be accessed by any code in the same assembly, but not from another assembly

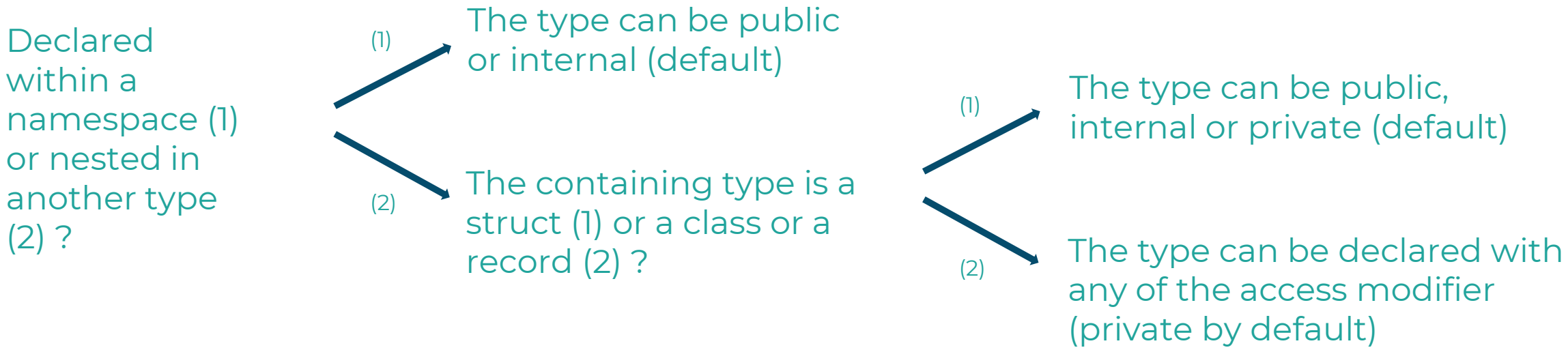
Protected internal

The type or member can be accessed by any code in the assembly in which it's declared, or from within a derived class in another assembly

Private protected

The type or member can be accessed only within its declaring assembly, by code in the same class or in a type that is derived from that class

Type visibility



Member visibility

The containing type is a struct (1) or a class or a record (2) ?

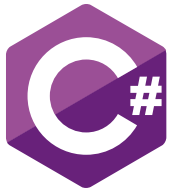
The member can be public, internal or private (default)

The member can be declared with any of the access modifier (private by default)

Demo

Understand access modifiers

Comments



Comments

A comment is text that won’t be considered by the compiler

A comment gives explanations about the code

Comments

A comment can be

- A single line comment
- A multiple line comment

Single-line comment

```
// This is a single-line comment
```

Multiple-line comment

```
/*
    This is a multiple-line comment :
      ...
      ...
      ...
*/
```

Comments

With comments, you can create a documentation for your code

XML Documentation extension before December 2020

VS Code now officially supports the documentation

Demo

Create single line comments

Create multiple line comments

XML Documentation

Summary

You can create custom types by creating a class or a structure

A class is a reference type, a structure is a value type

A class or a structure is used to represent (model) an object or a system

A class or a structure can contain members like fields, constants or properties

A structure does not support inheritance

An enumeration is a custom type that represents a choice from a set of values

Summary

- You can create an object, an instance of a type by instantiating it with a new expression
- You can define a type or member visibility using access modifiers
- Comments a pieces of text that are not executed
- Comments give explanation about the types, the members and the code
- You can create an XML documentation with the comment tags that you create