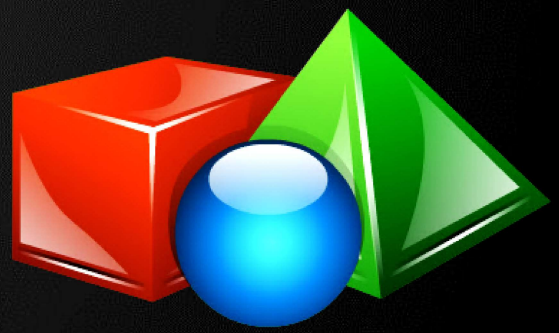
Telerik Academy

Using Classes and Objects

Using the Standard .NET Framework Classes



C# Advanced

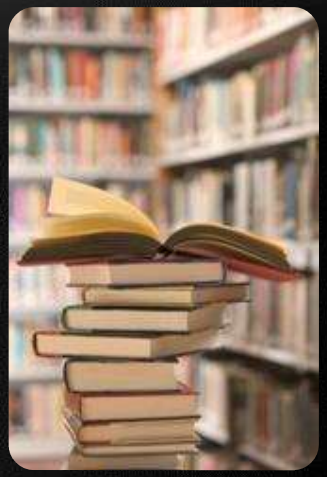
Telerik Software Academy

https://telerikacademy.com

Follow us

Telerik Academy

Table of Contents



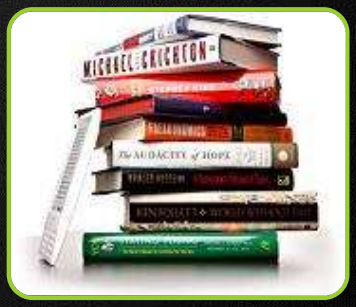
- Classes and Objects
  - What are Objects?
  - What are Classes?
- Classes in C#
  - Declaring an object
  - Fields and Properties: Instance and Static
  - Instance and Static Methods
  - Constructors
- Enumerations
- Value and Reference Types

Follow us

Telerik Academy

Table of Contents

- Structures
- Namespaces
- Random class
- Introduction to .NET Common Type System



Follow us

Telerik Academy

What are Objects?

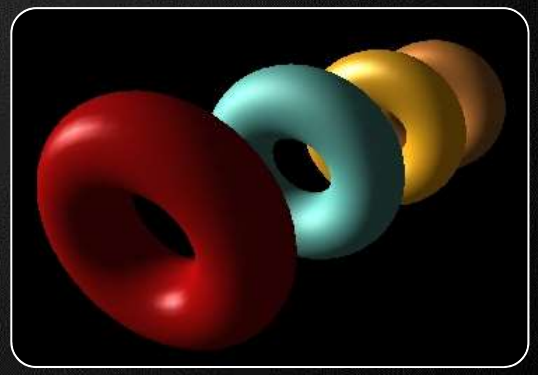
- Software objects model real-world objects or abstract concepts
  - *Examples*:
    - bank, account, customer, dog, bicycle, queue
- Real-world objects have states and behaviors
  - Account' states:
    - holder, balance, type
  - Account' behaviors:
    - withdraw, deposit, suspend

Follow us

Telerik Academy

What are Objects?

- How do software objects implement real-world objects?
  - Use variables/data to implement states
  - Use methods/functions to implement behaviors
- An object is a software bundle of variables and related methods



Follow us

Telerik Academy

Objects Represent

- Things from the real world
  - ✓ checks
  - ✓ people
  - ✓ shopping list
- Things from the computer world
  - ✓ numbers
  - ✓ characters
  - ✓ queues
  - ✓ arrays

Follow us

What is a Class?

- The formal definition of `class`:
  - *Definition by Google*
    - `Classes` act as templates from which an instance of an object is created at run time. Classes define the properties of the object and the methods used to control the object's behavior.

Follow us

Classes

- Classes provide the structure for objects
  - Define their prototype, act as template
- Classes define:
  - Set of `attributes`
    - Represented by variables and properties
    - Hold their state
  - Set of actions (behavior)
    - Represented by methods
- A class defines the methods and types of data associated with an object

Follow us

Telerik Academy

Classes – *Example*

```
Account
```

Class Name

```
+Owner: Person
+Ammount: double
```

Attributes

(Properties and Fields)

```
+Suspend()
+Deposit(sum:double)
+Withdraw(sum:double)
```

Operations

(Methods)

Follow us

Telerik Academy

Objects

- An `object` is a concrete `instance` of a particular class
- Creating an object from a class is called instantiation
- Objects have state
  - Set of values associated to their attributes
- *Example*:
  - Class: `Account`
  - Objects: Ivan's `account`, Peter's `account`

Follow us

Telerik Academy

Objects – *Example*

- Class

```
Account

+Owner: Person
+Ammount: double

+Suspend()
+Deposit(sum:double)
+Withdraw(sum:double)
```

Follow us

Telerik Academy

Objects – *Example*

- Objects

```
ivanAccount

+Owner="Ivan Kolev"
+Ammount=5000.0
```

```
peterAccount

+Owner="Peter Kirov"
+Ammount=1825.33
```

Follow us

Telerik Academy

Value and Reference Types

Follow us

Telerik Academy

Value and Reference Types

- In programming there are two categories of types
  - Value types
  - Reference types
- Placed in different areas of memory
  - Value types live in the execution stack*
    - Freed when become out of scope
  - Reference types live in the managed heap(dynamic memory)
    - Freed by the garbage collector
- *Note*: this does not mean that value types, which are part of reference types live on the stack. E.g., integers in a `List<int>` do not live on the stack

Follow us

Telerik Academy

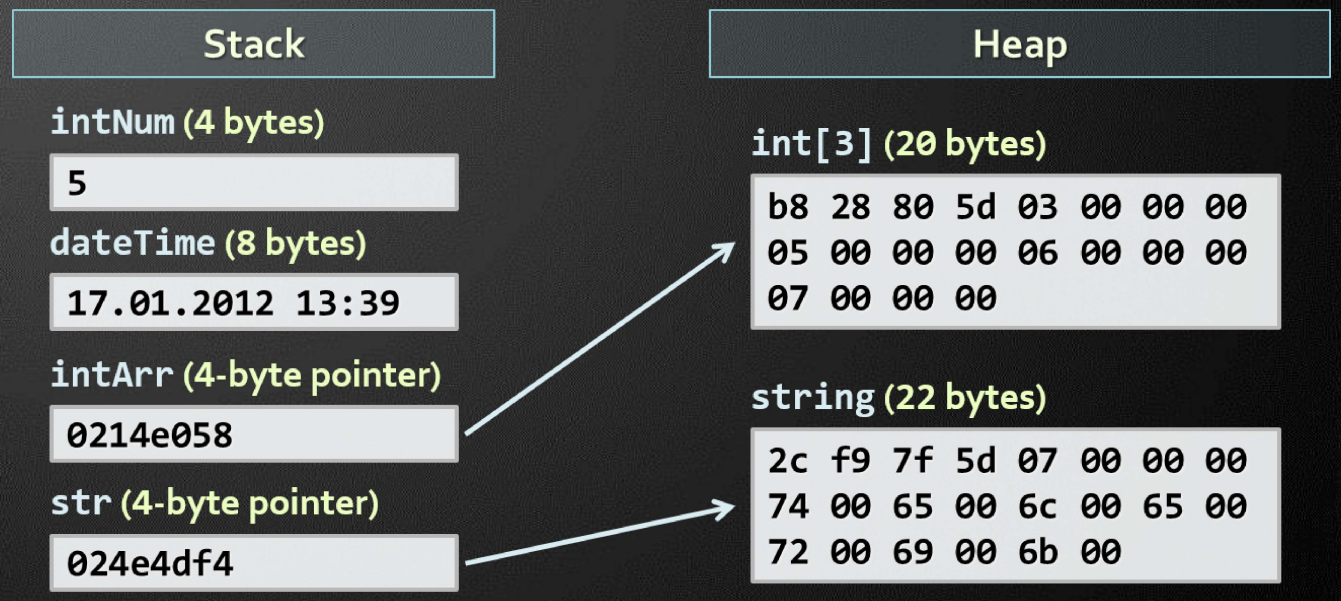
Value and Reference Types *Examples*

- Value types
  - Most of the primitive types
  - Structures
    - *Examples*: `int`, `float`, `bool`, `DateTime`
- Reference types
  - Classes and Interfaces
  - Strings
  - Arrays
    - *Examples*: `string`,`Random`,`object`,`int[]`

Follow us

Value and Reference Types

```
int intNum = 5;
DateTime date = DateTime.Now;
int[] intArr = new int[] { 5, 6, 7 };
string str = "telerik";
```



Follow us

Telerik Academy

Classes in C#

Follow us

Telerik Academy

Classes in C#

- `Classes` – basic units that compose programs
- Implementation is encapsulated (hidden)
- `Classes` in C# can contain:
  - Fields (member variables)
  - Properties
  - Methods
  - Constructors
  - Inner types
  - Etc. (events, indexers, operators, …)

Follow us

Telerik Academy

Classes in C# – *Examples*

- *Example* of `classes` (structures):
  - `System.Console`
  - `System.String` (`string` in C#)
  - `System.Int32` (`int` in C#)
  - `System.Array`
  - `System.Math`
  - `System.Random`
  - `System.Collections.Generics.List<T>`

Follow us

Telerik Academy

Declaring Objects

- An instance of a class or structure can be defined like any other variable:

```
using System;
...
// Define two variables of type DateTime
DateTime today;
DateTime halloween;
```

- Instances cannot be used if they are not initialized

```
// Declare and initialize a structure instance
DateTime today = DateTime.Now;
```

Follow us

Telerik Academy

Fields and Properties

Accessing Fields and Properties

Follow us

Fields

- `Fields` are data members of a class
  - Can be variables and constants (read-only)
- Accessing a field doesn't invoke any actions of the object
  - Just accesses its value
- *Example*:
  - `string.Empty` (the `""` string)

Follow us

Telerik Academy

Accessing Fields

- `Constant fields` can be only read
- `Variable fields` can be read and modified
- Usually properties are used instead of directly accessing variable fields
- *Examples*:

```
// Accessing read-only field
String empty = String.Empty;

// Accessing constant field
int maxInt = Int32.MaxValue;
```

Follow us

Telerik Academy

Properties

- `Properties` look like fields
  - Have name and type
  - Can contain code, executed when accessed
- Usually used as wrappers
  - To control the access to the data fields
  - Can contain more complex logic
- Can have two components called `accessors`
  - `get` for reading their value
  - `set` for changing their value

Follow us

Properties

- According to the implemented accessors properties can be:
  - Read-only (`get` accessor only)
  - Read and write (both `get` and `set` accessors)
  - Write-only (`set` accessor only)
- *Example* of read-only property:
  - `String.Length`
- *Example* of read-write property:
  - `Console.BackgroundColor`

Follow us

Accessing Properties and Fields

Example

```
DateTime christmas = new DateTime(2009, 12, 25);

int day = christmas.Day;
int month = christmas.Month;
int year = christmas.Year;

Console.WriteLine(
  "Christmas day: {0}, month: {1}, year: {2}",
  day, month, year);

Console.WriteLine(
  "Day of year: {0}", christmas.DayOfYear);

Console.WriteLine("Is {0} leap year: {1}",
  year, DateTime.IsLeapYear(year));
```

Follow us

Telerik Academy

Accessing Properties and Fields

Demo

Follow us

Telerik Academy

Instance and Static Members

Accessing Object and Class Members

Follow us

Instance and Static Members

- Fields, properties and methods can be:
  - Instance (or object members)
  - Static (or class members)
- Instance members are specific for each object
  - *Example*: different dogs have different name
- Static members are common for all instances of a class
  - *Example*: `DateTime.MinValue` is shared between all instances of `DateTime`

Follow us

Telerik Academy

Accessing Members – Syntax

- Accessing instance members
  - The name of the instance, followed by the name of the member (field or property), separated by dot (".")

```
<instance_name>.<member_name>
```

- Accessing static members
  - The name of the class, followed by the name of the member

```
<class_name>.<member_name>
```

Follow us

Instance and Static Members *Examples*

- *Example* of instance member
  - `String.Length`
    - Each string object has a different length
    - E.g. `"I like C#".Length` → 9
- *Example* of static member
  - `Console.ReadLine()`
    - The console is only one (global for the program)
    - Reading from the console does not require to create an instance of it

Follow us

Telerik Academy

Methods

Calling Instance and Static Methods

Follow us

Telerik Academy

Methods

- Methods manipulate the data of the object to which they belong or perform other tasks
- *Examples*:
  - `Console.WriteLine(someObject)`
  - `Console.ReadLine()`
  - `String.Substring(index, length)`
  - `Array.GetLength(index)`
  - `List<T>.Add(item)`
  - `DateTime.AddDays(count)`

Follow us

Telerik Academy

Instance Methods

- Instance methods - manipulate the data of a specified object or perform any other tasks
  - If a value is returned, it depends on the particular class instance
- Syntax:
  - The name of the instance, followed by the name of the method, separated by dot

```
<object_name>.<method_name>(<parameters>)
```

Follow us

Telerik Academy

Calling Instance Methods
Examples

- Calling instance methods of `String`:

```
String sampleLower = new String('a', 5);
String sampleUpper = sampleLower.ToUpper();

Console.WriteLine(sampleLower); // aaaaa
Console.WriteLine(sampleUpper); // AAAAA
```

- Calling instance methods of `DateTime`:

```
DateTime now = DateTime.Now;
DateTime later = now.AddHours(8);

Console.WriteLine("Now: {0}", now);
Console.WriteLine("8 hours later: {0}", later);
```

Follow us

Telerik Academy

Calling Instance Methods

Demo

Follow us

Telerik Academy

Static Methods

- Static methods are common for all instances of a class (shared between all instances)
  - Returned value depends only on the passed parameters
  - No particular class instance is available
- Syntax:
  - The name of the class, followed by the name of the method, separated by dot

```
<class_name>.<method_name>(<parameters>)
```

Follow us

Telerik Academy

Calling Static Methods

Examples

```
using System;

double radius = 2.9;
double area = Math.PI * Math.Pow(radius, 2);
Console.WriteLine("Area: {0}", area);
// Area: 26,4207942166902

double precise = 8.7654321;
double round3 = Math.Round(precise, 3);
double round1 = Math.Round(precise, 1);
Console.WriteLine(
    "{0}; {1}; {2}", precise, round3, round1);
// 8,7654321; 8,765; 8,8
```

Follow us

Telerik Academy

Calling Static Methods

Demo

Follow us

Telerik Academy

Constructors

Follow us

Telerik Academy

Constructors

- `Constructors` are special methods used to assign initial values of the fields in an object
  - Executed when an object of a given type is being created
  - Have the same name as the class that holds them
  - Do not return a value
- A class may have several constructors with different set of parameters

Follow us

Telerik Academy

Constructors

- Constructor is invoked by the new operator

```
<instance_name> = new <class_name>(<parameters>)
```

- *Examples*:

```
String s = new String(new char[]{'a','b','c'});
```

```
String s = new String('*', 5); // s = "*****"
```

```
DateTime dt = new DateTime(2009, 12, 30);
```

```
DateTime dt = new DateTime(2009, 12, 30, 12, 33, 59);
```

```
Int32 value = new Int32();
```

Follow us

Parameterless Constructors

- The constructor without parameters is called default(parameterless) constructor
- *Example*:
  - Creating an object for generating random numbers with a default seed

```
// parameterless constructor Random() invoked
Random randomGenerator = new Random();
```

Follow us

Constructor with Parameters

- *Example*
  - Creating objects for generating random values with specified initial seeds

```
using System;
...
Random randomGenerator1 = new Random(123);
Console.WriteLine(randomGenerator1.Next());
// 2114319875

Random randomGenerator2 = new Random(456);
Console.WriteLine(randomGenerator2.Next(50));
// 47
```

Follow us

Telerik Academy

Generating Random Numbers

Demo

Follow us

Telerik Academy

More Constructor *Examples*

- Creating a `DateTime` object for a specified date and time

```
using System;

DateTime halloween = new DateTime(2009, 10, 31);
Console.WriteLine(halloween);

DateTime julyMorning = new DateTime(2009, 7, 1, 5, 52, 0);
Console.WriteLine(julyMorning);
```

- Different constructors are called depending on the different sets of parameters

Follow us

Telerik Academy

Creating DateTime Objects

Demo

Follow us

Telerik Academy

Enumerations

Types Limited to a Predefined Set of Values

Follow us

Telerik Academy

Enumerations

- `Enumerations` in C# are types whose values are limited to a predefined set of values
  - E.g. the days of week
  - Declared by the keyword `enum` in C#
  - Hold values from a predefined set

```
public enum Color { Red, Green, Blue, Black }
…
Color color = Color.Red;
Console.WriteLine(color); // Red
color = 5; // Compilation error!
```

Follow us

Telerik Academy

Enumerations

Demo

Follow us

Telerik Academy

Structures

What are Structures? When to Use Them?

Follow us

Structures

- `Structures` in C# are similar to classes
  - Structures are value types (directly hold a value)
  - Classes are reference types (pointers)
- Structures are usually used for storing data structures, without any other functionality
- Structures can have fields, properties, etc.
  - Using methods is not recommended
- *Example* of structure
  - `System.DateTime` – represents a date and time

Follow us

Telerik Academy

Structures

Demo

Follow us

Telerik Academy

.NET Common Type System

Brief Introduction

Follow us

Common Type System (CTS)

- CTS defines all data types supported in .NET Framework
  - Primitive types (e.g. `int`, `float`, `object`)
  - Classes (e.g. `String`, `Console`, `Array`)
  - Structures (e.g. `DateTime`)
  - Arrays (e.g. `int[]`, `string[,]`)
  - Etc.
- Object-oriented by design

Follow us

Telerik Academy

CTS and Different Languages

- CTS is common for all .NET languages
  - C#, F#, VB.NET, J#, JScript.NET, ...
- CTS type mappings:

CTS Type	C# Type	VB.NET Type
System.Int32	int	Integer
System.Single	float	Single
System.Boolean	bool	Boolean
System.String	string	String
System.Object	object	Object

Follow us

System.Object: CTS Base Type

- `System.Object` (`object` in C#) is a base type for all other types in CTS
  - Can hold values of any other type:

```
string s = "test";
object obj = s;
```

- All .NET types derive common methods from `System.Object`, e.g. `ToString()`

```
DateTime now = DateTime.Now;
string nowInWords = now.ToString();
Console.WriteLine(nowInWords);
```

Follow us

Telerik Academy

Namespaces

Organizing Classes Logically into Namespaces

Follow us

Telerik Academy

What is a Namespace?

- `Namespaces` are used to organize the source code into more logical and manageable way
- Namespaces can contain
  - Definitions of classes, structures, interfaces and other types and other namespaces
- Namespaces can contain other namespaces
- For example:
  - `System` namespace contains `Data` namespace
  - The name of the nested namespace is `System.Data`

Follow us

Telerik Academy

Full Class Names

- A full name of a class is the name of the class preceded by the name of its namespace

```
<namespace_name>.<class_name>
```

- *Example*:
  - `Array` class, defined in the `System` namespace
  - The full name of the class is `System.Array`

Follow us

Telerik Academy

Including Namespaces

- The `using` directive in C#:

```
using <namespace_name>
```

- Allows using types in a namespace, without specifying their full name
- *Example*:

```
using System;
DateTime date;
```

- instead of

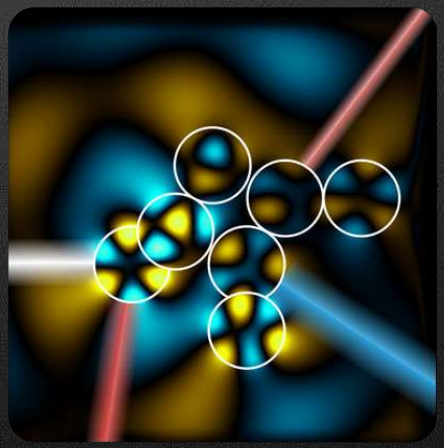
```
System.DateTime date;
```

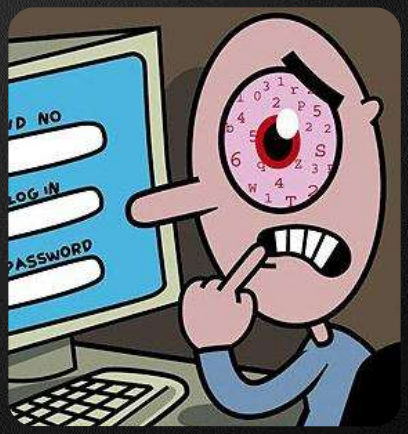
Follow us

Telerik Academy

The Random Class

Password Generator Demo





Follow us

Telerik Academy

The Random Class

- The Random class
  - Generates random integer numbers

```
Random rand = new Random();
for (int number = 1; number <= 6; number++)
{
   int randomNumber = rand.Next(49) + 1;
   Console.Write("{0} ", randomNumber);
}
```

- This generates 6 random int in range [1..49]
- Always use a single Random instance!
  - This will avoid abnormalities

Follow us

Telerik Academy

Password Generator *Example*

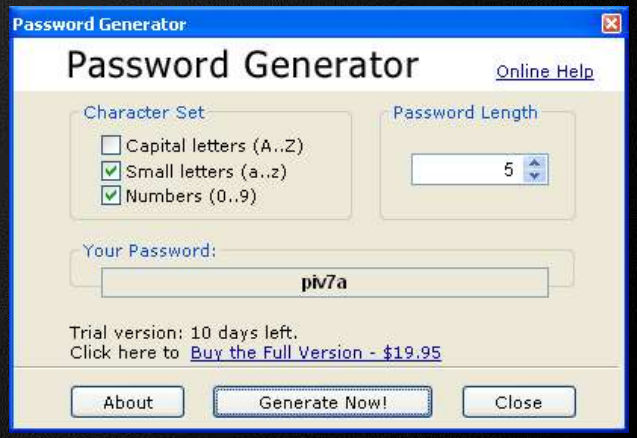
- Write a program to generate a random password between 8 and 15 characters
  - The password contains of at least two capital letters, two small letters, one digit and three special characters
- Constructing the password generator class:
  - Start from an empty password
  - Place 2 random capital letters at random positions
  - Place 2 random small letters at random positions
  - Place 1 random digit at random positions
  - Place 3 special characters at random positions

Follow us

Telerik Academy

Password Generator

- Now we have exactly 8 characters
  - To make the password length between 8 and 15 we add between 0 and 7 random characters
    - Capital / small letters / digits / special character
    - Inserts each of them at random position



Follow us

Password Generator Class

```
class RandomPasswordGenerator
{
  private const string CapitalLetters=
                                "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  private const string SmallLetters =
                                "abcdefghijklmnopqrstuvwxyz";
  private const string Digits = "0123456789";
  private const string SpecialChars =
                         "~!@#$%^&*()_+=**{}[]\\|':;.,/?<>";
  private const string AllChars = CapitalLetters +
                       SmallLetters + Digits + SpecialChars;

  private static Random rnd = new Random();

 // the example continues…
```

Follow us

Telerik Academy

Password Generator Class

```
static void Main()
{
   StringBuilder password = new StringBuilder();
   for (int i = 1; i <= 2; i++)
   {
      char capitalLetter = GenerateChar(CapitalLetters);
      InsertAtRandomPos(password, capitalLetter);
   }
   for (int i = 1; i <= 2; i++)
   {
      char smallLetter = GenerateChar(SmallLetters);
      InsertAtRandomPos(password, smallLetter);
   }
   char digit = GenerateChar(Digits);
   InsertAtRandomPos(password, digit);
   for (int i = 1; i <= 3; i++)
   {
      char specialChar = GenerateChar(SpecialChars);
      InsertAtRandomPos(password, specialChar);
   }
// the example continues…
```

Follow us

Telerik Academy

Password Generator Class

```
    int count = rnd.Next(8);
    for (int i = 1; i <= count; i++)
    {
        char specialChar = GenerateChar(AllChars);
        InsertAtRandomPosition(password, specialChar);
    }
    Console.WriteLine(password);
}

static void InsertAtRandomPos(StringBuilder password, char character)
{
    int randomPosition = rnd.Next(password.Length + 1);
    password.Insert(randomPosition, character);
}

static char GenerateChar(string availableChars)
{
    int randomIndex = rnd.Next(availableChars.Length);
    char randomChar = availableChars[randomIndex];
    return randomChar;
}
```

Follow us

Telerik Academy

Summary

- Two types of objects
  - Value and Reference Types
- `Classes` provide the template for objects
- `Objects` are instances of classes
- `Classes` have different members
  - Methods, fields, properties, etc.
  - Instance and static members
  - Members can be accessed
  - Methods can be called
- `Structures` are used for storing data

Follow us

Telerik Academy

Summary

- Namespaces group related classes
- Namespaces help organizing the classes
- Common Type System (CTS) defines the types for all .NET languages
  - Value types
  - Reference types

Follow us