VIA University College

DNP-A18

Course Introduction

An Introduction to the course, C# and .NET

Course Overview

Practical course information and a course plan for the next half year

Introduction to .NET

An overview of the .NET platform

Introduction to C#

Going through the basics of C#

Working with .NET Core

How to get started working with .NET

Exercises

Get familiar with C# and basic console applications

DNP Teachers This Semester

Jakob Knop Rasmussen

MSc. in Computer Science, AU
Assistant Professor at VIA
E-mail: jknr@via.dk
Office: A.301a

Christian Flinker Sandbeck

Assistant Professor at VIA
E-mail: chfs@via.dk
Office: A.304

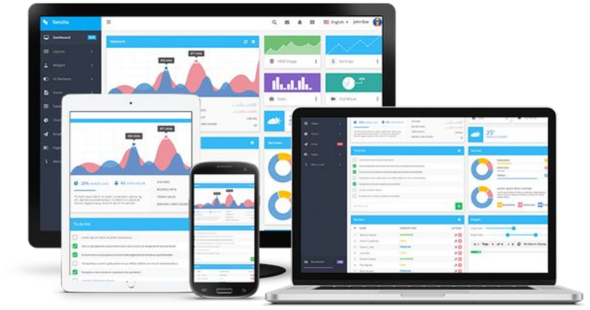
Expectations

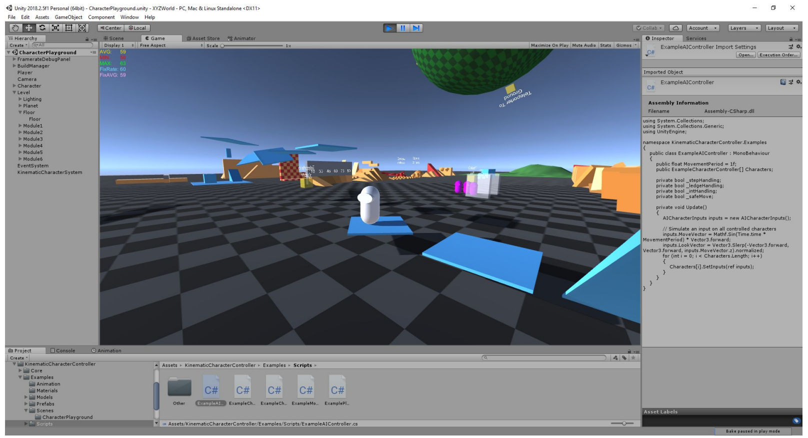
"Internet Technologies, C# and .NET"

What do you think is going to happen?

What do you expect to learn?

.NET Use Cases




Unity 2018.2.5f1 Personal (64bit) - CharacterPlayground.unity - XYZWorld - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
Scene
Game
Asset Store
Animator
Inspector
Services
ExampleAIController Import Settings
Imported Object
ExampleAIController
Assembly Information
Filename
Assembly-CSharp.dll
Project
Console
Animation
Assets > KinematicCharacterController > Examples > Scripts

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace KinematicCharacterController.Examples
{
    public class ExampleAIController : MonoBehaviour
    {
        public float MovementPeriod = 1f;
        public ExampleCharacterController[] Characters;

        private bool _stepHandling;
        private bool _ledgeHandling;
        private bool _intHandling;
        private bool _safeMove;

        private void Update()
        {
            AICharacterInputs inputs = new AICharacterInputs();

            // Simulate an input on all controlled characters
            inputs.MoveVector = Mathf.Sin(Time.time *
MovementPeriod) * Vector3.forward;
            inputs.LookVector = Vector3.Slerp(-Vector3.forward,
Vector3.forward, inputs.MoveVector.z).normalized;
            for (int i = 0; i < Characters.Length; i++)
            {
                Characters[i].SetInputs(ref inputs);
            }
        }
    }
}
```

Course Description

Learn to

Program in C# and build (web) applications and services using .NET Core

Four lessons once a week for 14 weeks

5 ECTS = ~140h of the students time = ~7h/week outside of the class!

Course Format

In class

1) Solve exercises
2) Cover relevant theory (plenary)

Class presentations will not cover all parts of the curriculum

Self-study (very important)

1) Read the literature
2) Finish the exercises
3) Apply the theory in your semester project

“Learning by doing”

Teachers are facilitators, not lecturers

We are here to help you learn - use us!

Course Literature

All course literature will be uploaded to itslearning

Curriculum consists of literature + slides

The Exam

Oral examination

Joint exam with SEP3 and SDJ3

Group presentation followed by individual examination

Group presentation of the project - 5 minutes per person

Individual examination - 35 minutes
(including examination in DNP1, SEP3 and SDJ3)

IF YOU DON'T WORK ON YOUR SEMESTER PROJECT
INSTRUCTO
YOU'RE GONNA HAVE A BAD TIME
imgflip.com

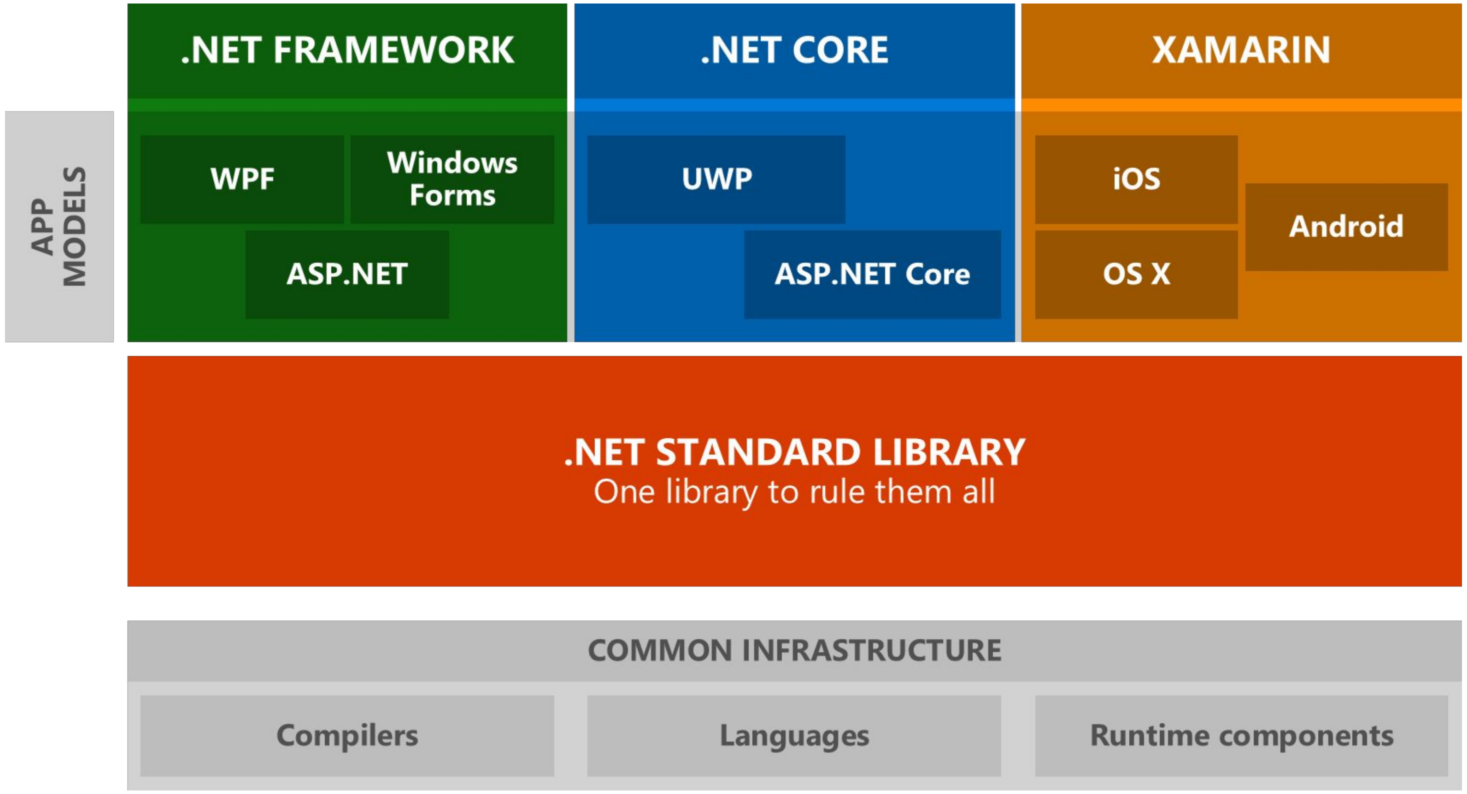
Course Plan

Week	Topic
35	Course Introduction
36	C# Programming
37	Advanced C# Programming
38	.NET
39	More .NET
40	Unit Testing and TDD
41	Class Libraries
42	Autumn Break
43	Consuming Web Services
44	Data Access
45	Exposing Web Services
46	Web Applications
47	User Management and Security
48	Web Applications and Deployment
49	Course wrap-up and exam info

Console applications (.NET Core)

Web apps/services (ASP.NET Core)

.NET Platform Today



.NET Core

"General purpose development platform maintained by Microsoft and the .NET community"

Cross-platform
Windows, Linux and macOS. Can be used in device, cloud, and embedded/IoT scenarios

Fast
Build with performance and scalability in mind

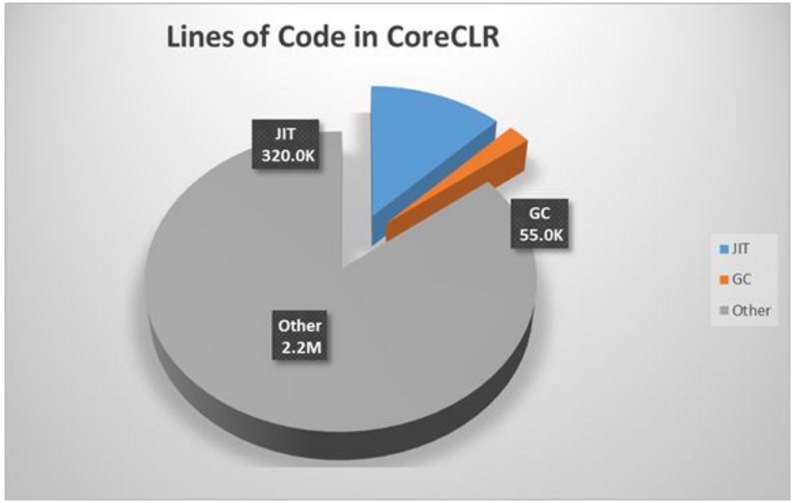
Lightweight
No impact deployment and a modular development model perfect for containers

Open Source
Runtime, libraries, compiler, languages and tools developed in the open on GitHub

.NET Core Composition

.NET Core is composed of

- .NET runtime (CoreCLR)
- A set of framework libraries (CoreFX)
- A set of SDK tools (CLI) and language compilers (Roslyn)
- The “dotnet” app host, which is used to launch .NET Core apps



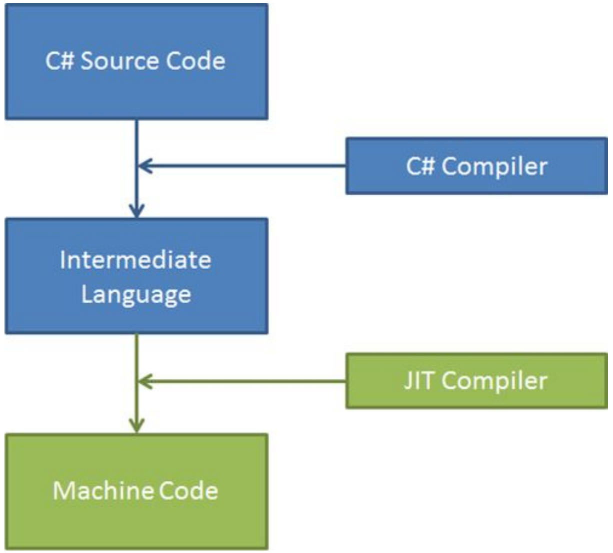
Common Language Runtime (CLR)

"in-memory" application that translates IL Code to Native Code

Intermediate Language
Higher-level .NET languages, such as C#, compile down to a hardware-agnostic instruction set, which is called Intermediate Language (IL)

Just-in-time (JIT) compilation
Compiler that translates IL to machine Code. Occurs during execution of application.

Managed Code
Code whose execution is managed by a runtime like the CLR



C#

```
if (a > b) max = a; else max = b;
```

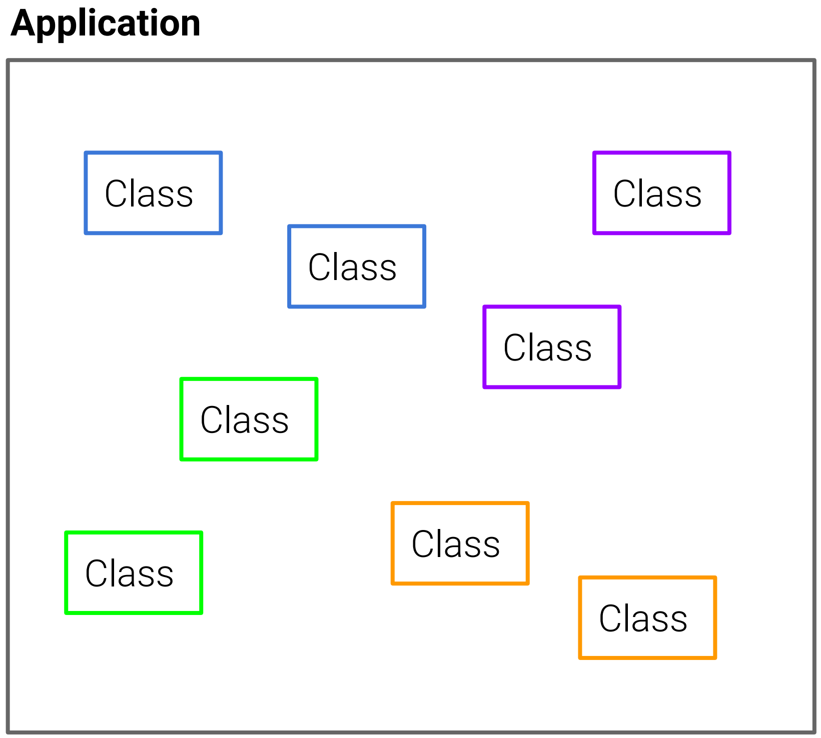
CIL

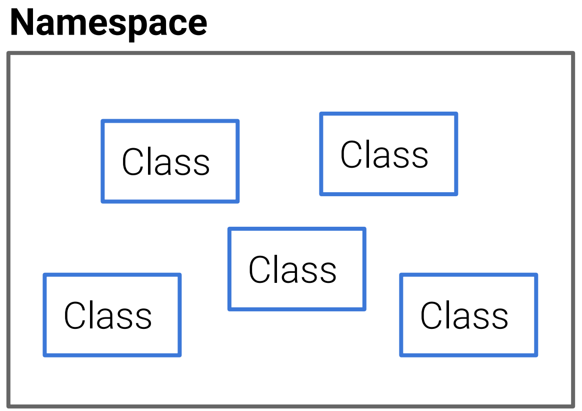
```
L_0004: ldloc.0
L_0005: ldloc.1
L_0006: ble.s
L_000c
L_0008: ldloc.0
L_0009: stloc.2
L_000a: br.s
L_000e
L_000c: ldloc.1
L_000d: stloc.2
```

Intel Code

```
mov ebx,[-4]
mov edx,[-8]
cmp ebx,edx
jle 17
mov ebx,[-4]
mov [-12],ebx
...
```

Application Structure

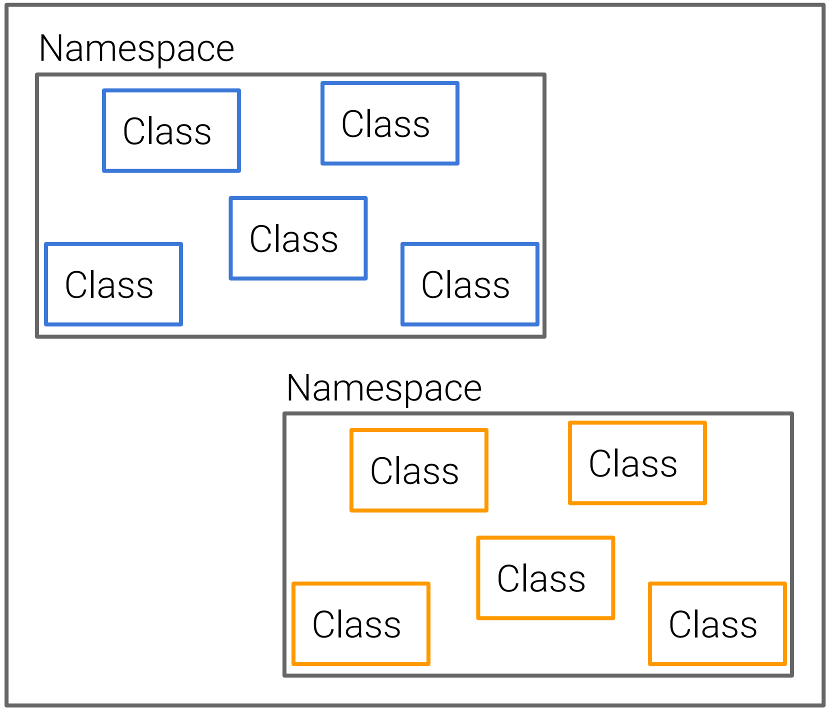




Container of related classes

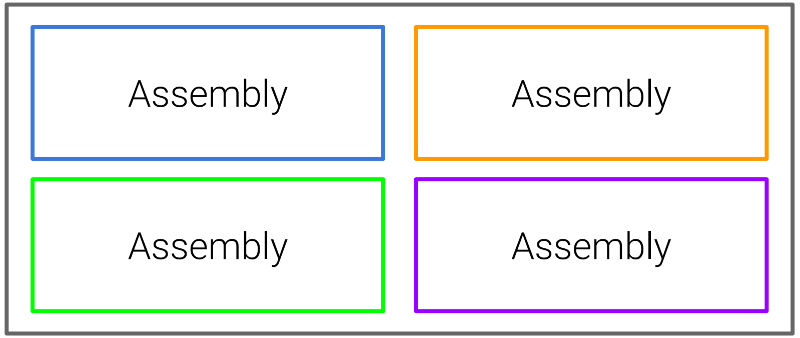
Assemblies

Assembly (DLL or EXE)



Containers of related namespaces

Application



Assemblies are the smallest unit for deployment

What is C#?

Very similar to Java
70% Java, 10% C++, 5% Visual Basic, 15% new

As in Java
- Object-orientation (single-inheritance)
- Interfaces
- Generics
- Exceptions
- Threads
- Namespaces (similar to Java packages)
- Strong and static typing
- Garbage collection
- Reflection
- Dynamic loading of code
- ...

As in C++
- Struct types
- Operator overloading
- Pointer arithmetic in unsafe code
- Some syntactic details

Types

Common primitive types (predefined)

	C# Type	.NET Type	Bytes	Range
Integral Numbers	**byte**	Byte	1	0 to 255
	short	Int16	2	-32,768 to 32,767
	int	Int32	4	-2.1B to 2.1B
	long	Int64	8	...
Real Numbers	**float**	Single	4	-3.4×10^{38} to 3.4×10^{38}
	double	Double	8	...
	decimal	Decimal	16	-7.9×10^{28} to 7.9×10^{28}
Character	**char**	Char	2	Unicode Characters
Boolean	**bool**	Boolean	1	True / False

Common non-primitive types (user defined)

String, Array, Enum, Class, Struct

All types are derived from the **System.Object** class

Reference Types and Value Types

Value Types

Structures

- Allocated on stack
- Memory allocation done automatically
- Immediately removed when out of scope

Primitive types, Custom structures

Reference Types

Classes

- You need to allocate memory
- Memory allocated on heap
- Garbage collected by CLR

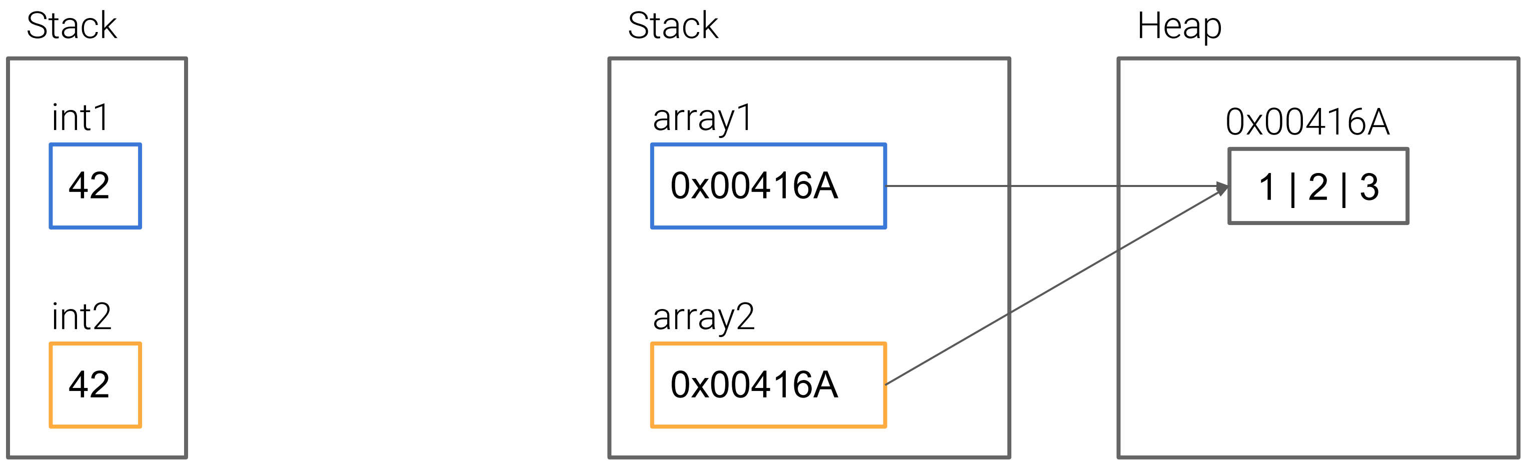
Arrays, Strings, Custom classes

Copying Types

```
var anotherObject = someObject;
```

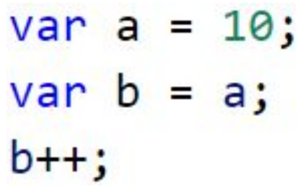
Value Types

Reference Types



memory address is copied, not the actual value

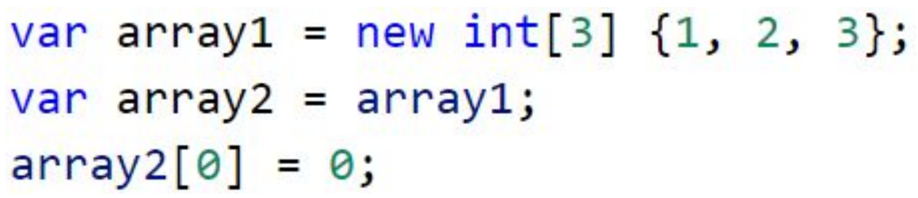
Quiz Time!



```
var a = 10;
var b = a;
b++;
```

Is a 10 or 11?

Quiz Time!

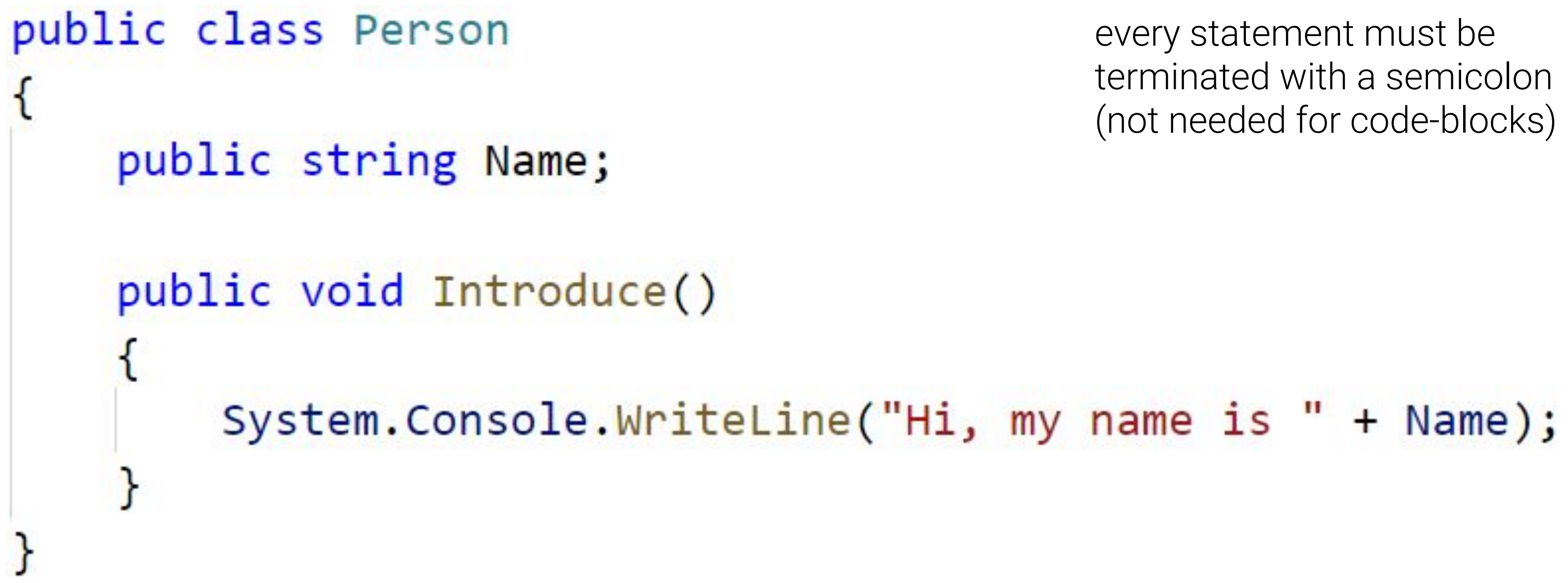


```
var array1 = new int[3] {1, 2, 3};
var array2 = array1;
array2[0] = 0;
```

is array1[0] 1 or 0?

Declaring Classes

access modifier   keyword   identifier



```
public class Person
{
    public string Name;

    public void Introduce()
    {
        System.Console.WriteLine("Hi, my name is " + Name);
    }
}
```

every statement must be terminated with a semicolon (not needed for code-blocks)

Creating Objects

```
int number = 42;
Person person = new Person();
```

Person type can be replaced with “var”
The compiler will automatically infer the type!

```
var person = new Person();
```

```
person.Name = "Jakob";
person.Introduce();
```

Object initializer syntax

```
var person = new Person {
    FirstName = "Jakob",
    LastName = "Knop"
};
```

Static Modifier

```
public class Calculator
{
    public static int Add(int a, int b)
    {
        return a + b;
    }
}
```

Static fields are associated with the class rather than any object

```
class Program
{
    static void Main(string[] args)
    {
        int result = Calculator.Add(1,2);
    }
}
```

Method now accessible directly on the class itself!

Example
Console.WriteLine - we do not have to create a new Console each time we want to call the WriteLine method

Arrays

Array
A data structure to store a collection of variables of the same type

don't do this...

```
int number1;
int number2;
int number3;
```

do this instead!

```
int[] numbers = new int[3];
```

Accessing array elements (zero indexed)

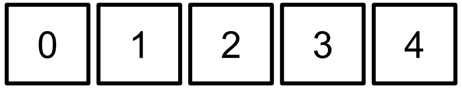
```
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;
```

in one line!

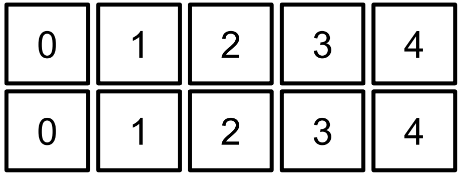
```
int[] numbers = new int[3] { 1, 2, 3};
```

Types of Arrays

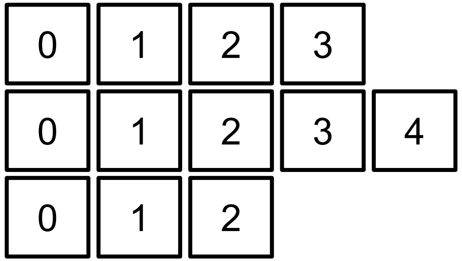
Single-dimensional



Multi-dimensional



Jagged



"An array of arrays"

Array methods

Clear(), Copy(), IndexOf(), Reverse(), Sort(), Length (property)

Array Examples

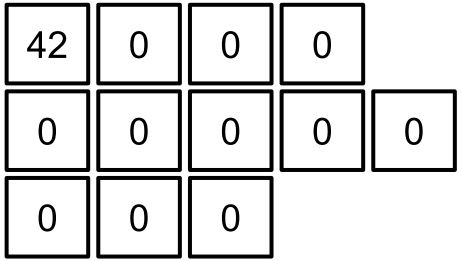
```
// single-dimensional array of five integers
int[] array1D = new int[5];
// two-dimensional array of four rows and two columns
int[,] array2D = new int[4, 2];
// three-dimensional array
int[, ,] array3D = new int[4, 2, 3];
```

Array initialization

```
int[] array1D = new int[] { 1, 3, 5, 7, 9 };
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
// The same array with dimensions specified.
int[,] array2Db = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

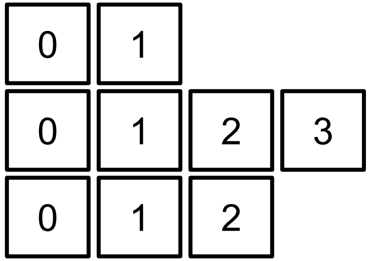
Array Examples

Jagged array



```
var array = new int[3][];
array[0] = new int[4];
array[1] = new int[5];
array[2] = new int[3];

array[0][0] = 42;
```



```
var array = new int[3][];
array[0] = new int[] { 0, 1 };
array[1] = new int[] { 0, 1, 2, 3 };
array[2] = new int[] { 0, 1, 2 };
```

or

```
int[][] array = new int[][]
{
    new int[] { 0, 1 },
    new int[] { 0, 1, 2, 3 },
    new int[] { 0, 1, 2 }
};
```

Array Examples

Rectangular, multidimensional array

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

3x5

```
var matrix = new int[3,5] {
    { 1, 2, 3, 4, 5 },
    { 6, 7, 8, 9, 10 },
    { 11, 12, 13, 14, 15}
};
var element = matrix[2, 4];
```

What is the type and value of element?

Lists

Array: **fixed** size
List: **dynamic** size

List is a Generic type

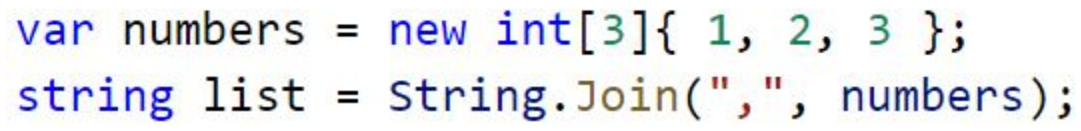
```
var numbers = new List<int>();
```

```
var numbers = new List<int>() { 1, 2, 3, 4 };
```

Useful methods
Add(), AddRange(), Remove(), RemoveAt(), IndexOf(), Contains(), Count (property)

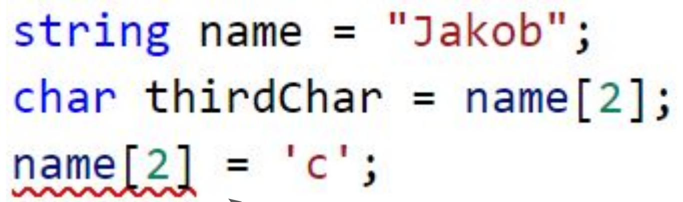
99% of the time Lists are used over arrays

Strings



```
var numbers = new int[3]{ 1, 2, 3 };
string list = String.Join(",", numbers);
```

String is a sequence of zero or more unicode characters



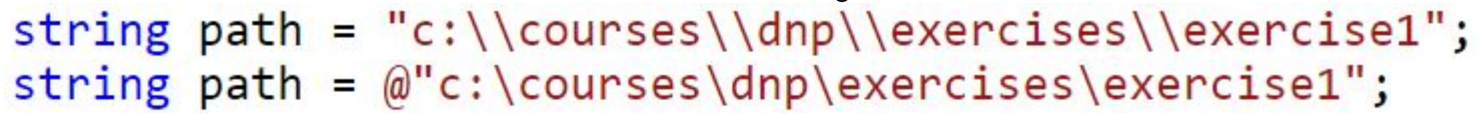
```
string name = "Jakob";
char thirdChar = name[2];
name[2] = 'c';
```

Strings are **immutable** - the methods that “modify” a string simply return a new string!

Escape Characters

Char	Description
\n	New Line
\t	Tab
\\	Backslash
\'	Single Quotation Mark
\"	Double Quotation Mark

Verbatim strings



```
string path = "c:\\courses\\dnp\\exercises\\exercise1";
string path = @"c:\courses\dnp\exercises\exercise1";
```

String Examples

```csharp
// string literal
string firstName = "Jakob";
// string concatenation
string name = firstName + lastName;
```

```csharp
// format string
string name = string.Format("{0} {1}", firstName, lastName);
```

```csharp
// string interpolation
string name = $"{firstName} {lastName}";
```

new, nicer syntax for C# 6

String Methods

Formatting

- ToLower() // “hello class”
- ToUpper() // “HELLO CLASS”
- Trim() // gets rid of white-space (good for user input!)

Searching

- IndexOf(‘x’)
- LastIndexOf(‘class’)

Substrings

- Substring(startIndex)
- Substring(startIndex, length)

Replacing

- Replace(‘.”, “!”)
- Replace(“teacher”,”jakob”)

Null Checking

- String.IsNullOrEmpty(str)
- String.IsNullOrWhiteSpace(str)

Splitting

- Split(‘ ‘)

Converting to numbers

```
string s = “100”;
int i = int.Parse(s);
int j = Convert.ToInt32(s);
```

Converting numbers to strings

```
int i = 100
string s = i.ToString(); // “100”
string t = i.ToString(“c”); // “$100.00”
```

StringBuilder

```
var builder = new StringBuilder();
```

System.Text

A **mutable** string

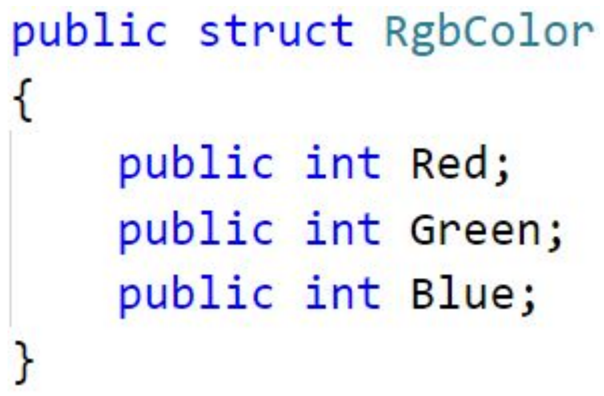
Easy and fast to create and manipulate strings

Methods focused on **manipulating** strings:

- Append()
- Insert()
- Remove()
- Replace()
- Clear()

no searching!

Structs



```
public struct RgbColor
{
    public int Red;
    public int Green;
    public int Blue;
}
```

Similar to classes
Structs combines related fields and methods together

When to use structs?
When you want to define a small, lightweight object (e.g. a point with fields x and y).

Enums

A set of name/value pairs (constants)

Use when you have a number of related constants

internally an integer

```
public enum Directions {
      Up = 0,
      Right = 1,
      Down = 3,
      Left = 4
}
```

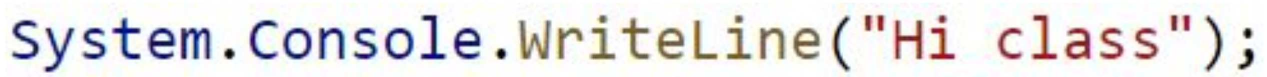
```
var direction = Directions.Up;
```

```
// enum to int
var intDirection = (int)Directions.Up;
// int to enum
var direction = (Directions)intDirection;
// enum to string
var stringDirection = direction.ToString();
// string to enum (parsing)
direction = (Directions) Enum.Parse(typeof(Directions), stringDirection);
```

The Console Class

Provides access to the **standard input** (keyboard), **standard output** (screen) and **standard error streams** (screen)

Only meaningful for console applications



```
System.Console.WriteLine("Hi class");
```

The Write and WriteLine methods

- **Console.Write** and **Console.WriteLine** display information on the console screen
- Both methods are overloaded

The Read and ReadLine methods

- **Read**: reads the next character
- **ReadLine**: reads the entire input line

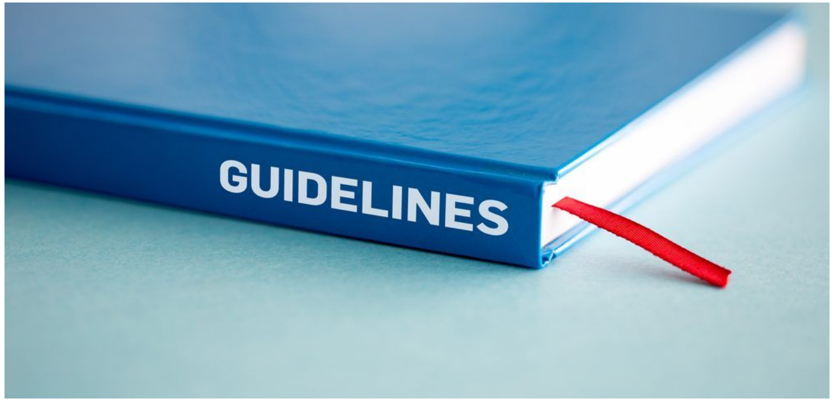
Naming Guidelines

In C#, most stuff use Pascal case

Pascal Case
Local variables, methods, ...

Camel Case
Parameters, private fields

Hungarian Notation
Don't use it, nobody likes it!



Favor **readability** over brevity

Read the guidelines, and **follow them**!

Comments

Single-line comment

```
// Here is a single-line comment
string teacher = "Jakob";
```

Multi-line comment

```
/*
    Here is a multi-line comment
*/
string teacher = "Jakob";
```

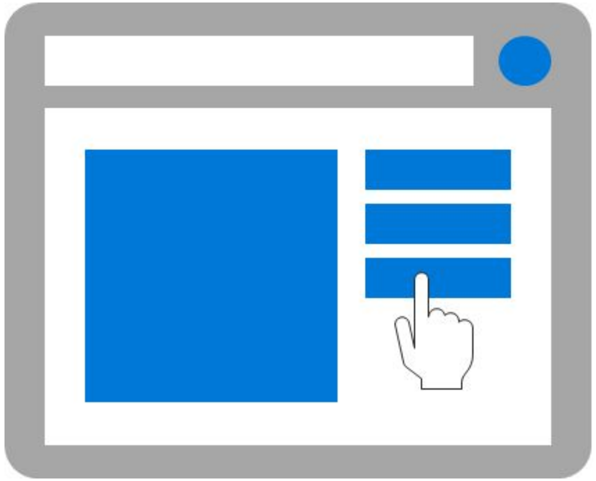
```
// Here is another multi-line
// comment
string teacher = "Jakob";
```

← standard way of doing multi-line comments

Good developers write good code; great ones also write good comments.

Use comments to explain **whys**, **hows**, **constraints**, etc. - NOT the whats

Workloads



Web

Console

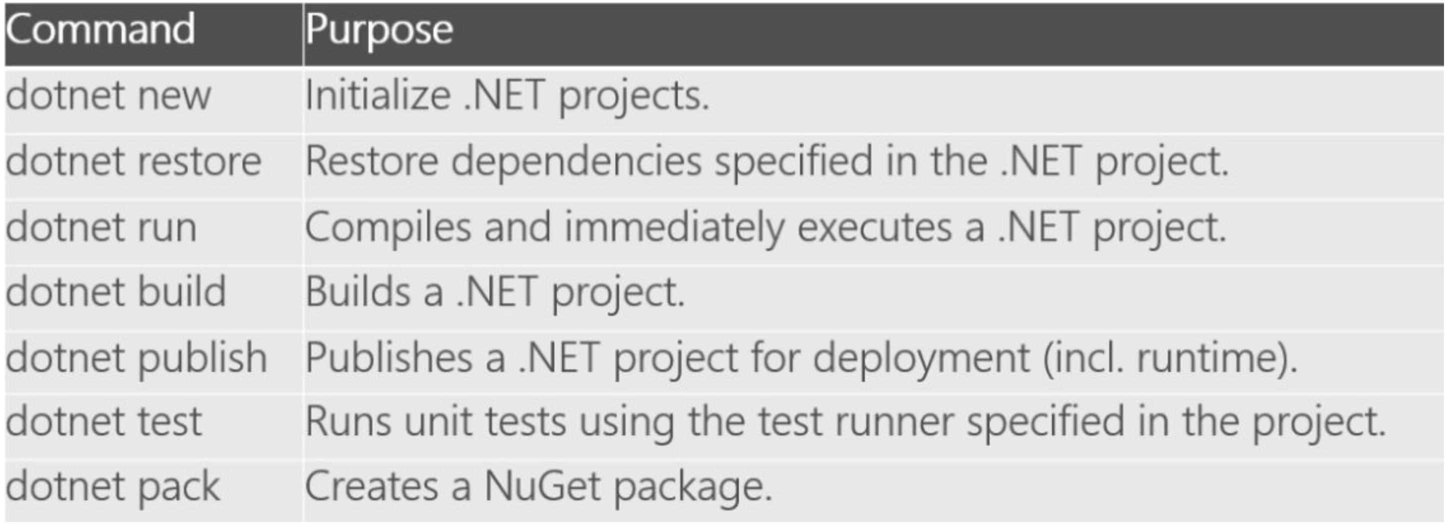
.NET Core CLI

$$\underbrace{\texttt{Dotnet}}_{\text{the driver}}\ \underbrace{\texttt{build}}_{\text{verb (command)}}\ \underbrace{\texttt{--output [path]}}_{\text{verb arguments}}$$

$\uparrow$

NuGet package

.NET Core CLI Commands



Command	Purpose
dotnet new	Initialize .NET projects.
dotnet restore	Restore dependencies specified in the .NET project.
dotnet run	Compiles and immediately executes a .NET project.
dotnet build	Builds a .NET project.
dotnet publish	Publishes a .NET project for deployment (incl. runtime).
dotnet test	Runs unit tests using the test runner specified in the project.
dotnet pack	Creates a NuGet package.

.NET Core CLI Tools

MSBuild - The .csproj File

It’s the magic that happens between **dotnet run** and your program actually producing output

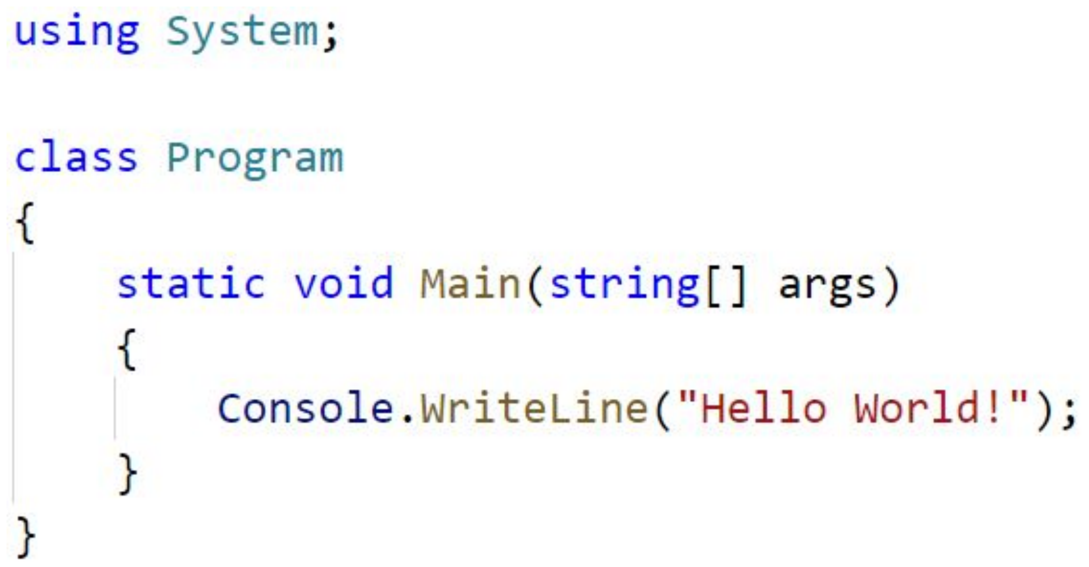
Default console template:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.1</TargetFramework>
  </PropertyGroup>

</Project>
```

Your First App



```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

Visual Studio Code

Quick tour

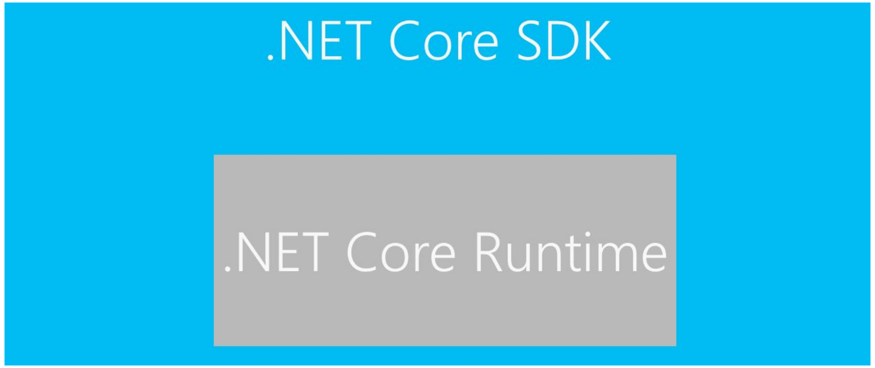
VSCode Tips and Tricks

Before You Start

1) Install the .NET Core SDK. Includes:
- .NET Core Tools
- .NET Core Runtime(s)

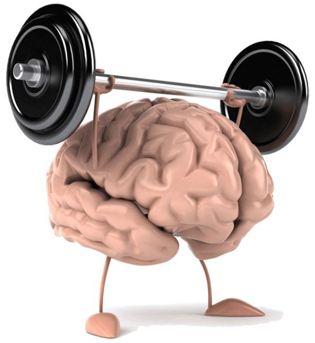
Typically installed globally via installer
Type “which dotnet” or “where dotnet” to locate dotnet in your path

2) Install Visual Studio Code
3) Install the "C# for Visual Studio Code" extension inside Visual Studio Code



Exercises

Exercises and setup can be found on the course website!



Where do I start?

- https://dot.net (download .NET Core)
- https://docs.microsoft.com/en-us/dotnet/ (.NET documentation)
- https://github.com/dotnet (source code)

.NET Glossary