
# Towards the .NET Junior Developer

The extremely solid course

# Lesson 2

.NET Basics – Part II

# Agenda

- **.NET development tools**
- **.NET source code basics**
  - .NET solution structure
  - Namespaces
  - Access modifiers and scopes
  - Code structure
  - Operators
  - Cycles
- **Other .NET key concepts**
  - Enumerators
  - Strings
  - Exceptions handling
- **Debugging basics**
- **Books of the day**
- **Links of the day**
- **Hometask**

# .NET development tools

Brief overview

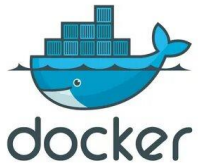
# .NET developer common toolset

## Integrated Development Environment (IDE)


## Text editors

## Database management


## Other

# Visual Studio live demo

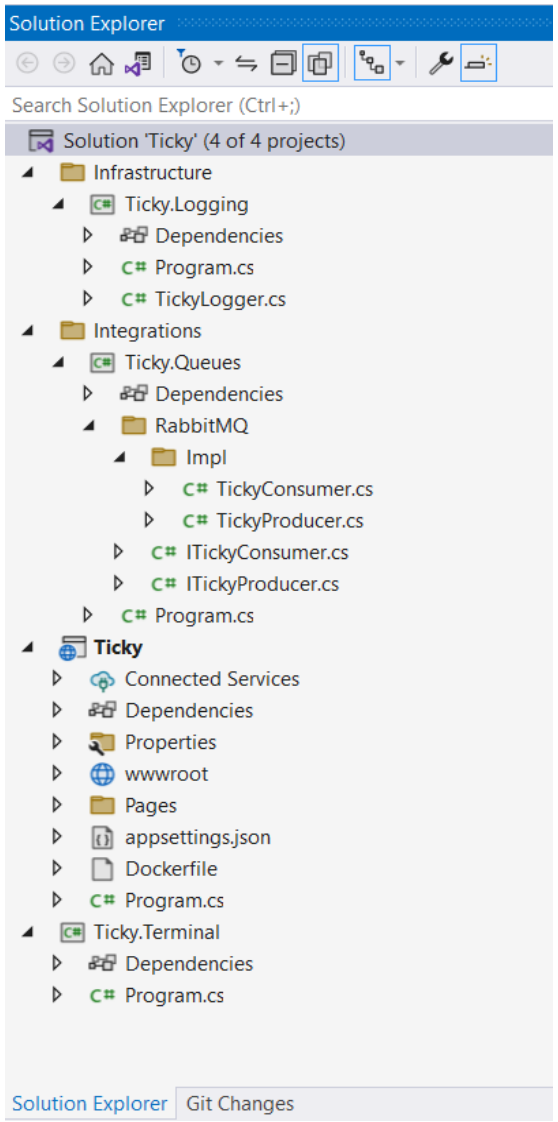
# JetBrains Rider live demo

# Notepad++ live demo

# .NET source code basics

Core aspects

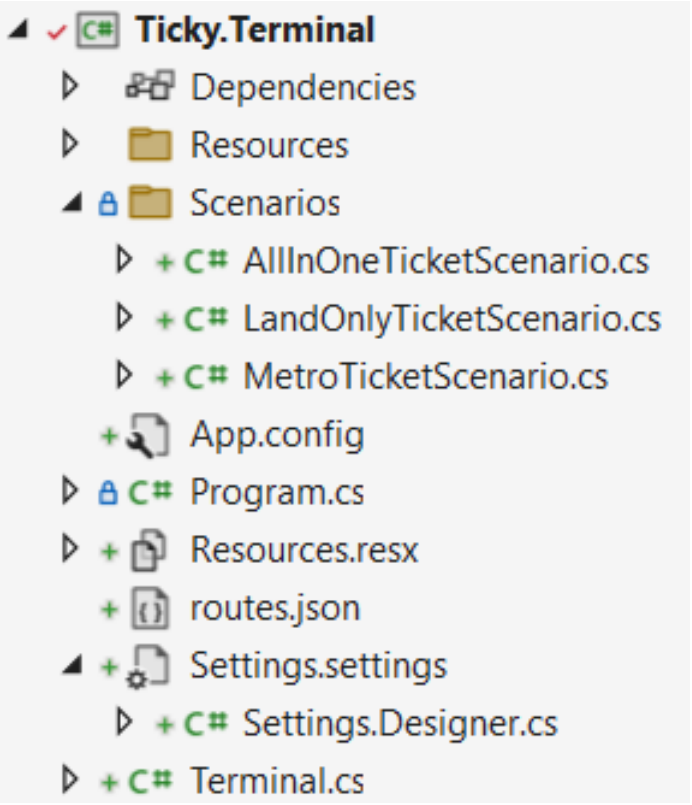
# .NET Solution Structure

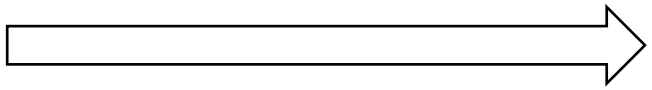



## Solution (*.sln)

- solution folders
- solution level files
- projects (*.csproj)
  - project dependencies
  - project level folders
  - source files
  - project level files (resources, configs etc.)

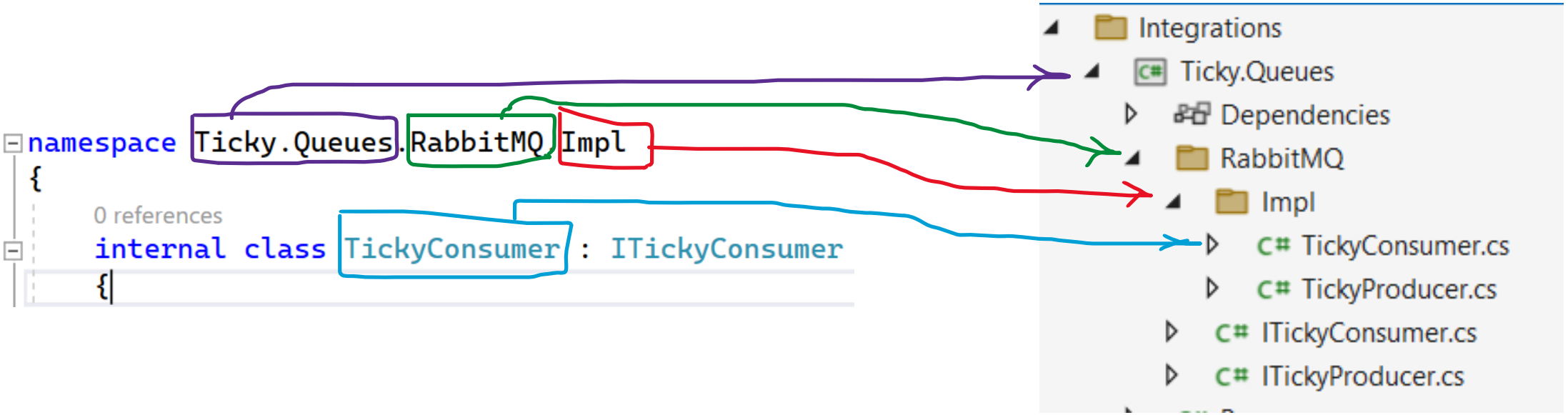
# Assemblies


Ticky.Terminal
Dependencies
Resources
Scenarios
AllInOneTicketScenario.cs
LandOnlyTicketScenario.cs
MetroTicketScenario.cs
App.config
Program.cs
Resources.resx
routes.json
Settings.settings
Settings.Designer.cs
Terminal.cs



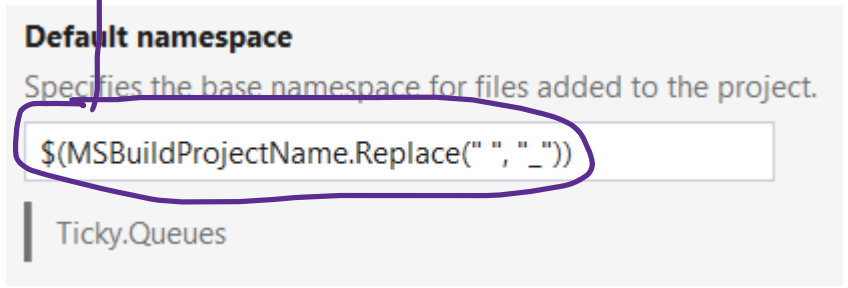

MyAssembly.dll
Assembly manifest
Type metadata
MSIL code
Resources

# Namespaces


namespace Ticky.Queues.RabbitMQ.Impl
{
0 references
internal class TickyConsumer : ITickyConsumer
{
Integrations
Ticky.Queues
Dependencies
RabbitMQ
Impl
TickyConsumer.cs
TickyProducer.cs
ITickyConsumer.cs
ITickyProducer.cs

# Namespaces

```
namespace Ticky.Queues.RabbitMQ.Impl
{
    0 references
    internal class TickyConsumer : ITickyConsumer
    {
```



# Access modifiers and scopes

| Caller's location | `public` | `protected internal` | `protected` | `internal` | `private protected` | `private` |
|---|---|---|---|---|---|---|
| Within the class | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Derived class (same assembly) | ✔ | ✔ | ✔ | ✔ | ✔ | ✘ |
| Non-derived class (same assembly) | ✔ | ✔ | ✘ | ✔ | ✘ | ✘ |
| Derived class (different assembly) | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ |
| Non-derived class (different assembly) | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ |

# Access modifiers and scopes

```
public sealed class CardService : ICardService
{
    private readonly ICardInfoStorage _storage;
    1 reference
    public CardService(ICardInfoStorage storage)
    {
        _storage = storage;
    }

    2 references
    public async Task IssueCard(DateTime issueDate, DateTime untilDate, TransportCardType cardType)
    {
        var cardInfo = cardType switch
        {
            TransportCardType.Bus => PrepareBusCardInfo(issueDate, untilDate),
            _ => throw new InvalidOperationException($"Card type {cardType} is not supported yet"),
        };

        await _storage.SaveCard(cardInfo);
    }
```

# Access modifiers and scopes

```
IEnumerable<FileInfo> FilterIssues(IEnumerable<FileInfo> files)
{
    var issueWord = "issue";
    var filteredFiles = files.Where(file =>
    {
        var name = file.Name;
        if (name.Contains(issueWord, StringComparison.InvariantCultureIgnoreCase))
        {
            Console.WriteLine($"Found issue file {name}");
            return true;
        }
        return false;
    });

    return filteredFiles;
}
```

# Live access modifiers demo

# Code structure

```
using System;

namespace Lesson2
{
    1 reference
    internal class ComplexClass
    {
        private const string ConstantField = "This field couldn't be changed";
        private readonly string _readonlyField;
        private string? _localData;

        0 references
        public string? FullAccessProperty { get; set; }

        0 references
        public ComplexClass()
        {
            _readonlyField = "This field can be changed only in constructor";
        }

        0 references
        public void MethodWithNoReturn()
        {
            _localData = "I was in the abyss...";
            Console.WriteLine("Void type is just the same thing as 'nothing'");
        }

        0 references
        public string MethodWithReturn(string phrase)
        {
            return $"Let me say something: {phrase}";
        }

        0 references
        public static void StaticMethod()
        {
            Console.WriteLine("This method does the same thing" +
                " for EVERY instance of ComplexClass");
        }
    }
}
```

- usings
- namespace
  - type (class, record etc.)
    - fields
    - properties
    - constructor (s)
    - methods

# Operators

- Operators and expressions
  - **Overview**
  - Arithmetic operators
  - Boolean logical operators
  - Bitwise and shift operators
  - Equality operators
  - Comparison operators
  - Member access operators and expressions
  - Type-testing operators and cast expression
  - User-defined conversion operators
  - Pointer-related operators
  - Assignment operators
  - Lambda expressions
  - Patterns
  - + and += operators
  - - and -= operators
  - ?: operator
  - ! (null-forgiving) operator
  - ?? and ??= operators
  - => operator
  - :: operator
  - await operator
  - default value expressions
  - delegate operator
  - is operator
  - nameof expression
  - new operator
  - sizeof operator
  - stackalloc expression
  - switch expression
  - true and false operators
  - with expression

# Operators

```
int first = 42;
int second = 6;

// Arithmetic operators
var sum = first + second;
var diff = first - second;
var mul = first * second;
var div = first / second;

var third = first += second;
var fourth = first -= second;
var fifth = first *= second;
var sixth = first /= second;

var postIncremented = first++;
var preIncremented = ++first;
var postDecremented = first--;
var preDecremented = --first;

var moreThan = first > second;
var lessThan = first < second;
var moreOrEqualsThan = first >= second;
var lessOrEqualsThan = first <= second;
var equals = first == second;
var notEquals = first != second;
```

```
// Bit operators
var bitAnd = first & second;
var bitOr = first | second;
var bitXor = first ^ second;
var bitRightShift = first >> second;
var bitLeftShift = first << second;

// Logical operators
var firstBool = true;
var secondBool = false;

var logicalAnd = firstBool && secondBool;
var logicalOr = firstBool || secondBool;

// Conditional operators
if (first == 42)
{
    Console.WriteLine("Wow!");
}
else if (first == 6)
{
    Console.WriteLine("Not bad");
}
else
{
    Console.WriteLine("So boring");
}
```

# Operators

```
string? nullableString = null;

var resultString = nullableString == null
    ? "default string"
    : nullableString;

var otherResultString = nullableString
    ?? "default string";

// Type checking
var isString = nullableString is string;

// Type conversion
object str = "some string";
if (str as string != null)
{
    // Do something useful
}

object otherStr = "some other string";
if (otherStr is string valuableString)
{
    // Do something useful with valuableString variable
}

// Other
var intSize = sizeof(int);
var methodName = nameof(ExploreOperators);
```

# Implicit typed variables

# Cycles

## *for*

```
public void DealWithForCycle(int[] numbers)
{
    for (var i = 0; i < numbers.Length; i++)
    {
        Console.WriteLine(numbers[i]);
    }
}
```

## *while*

```
public void DealWithWhileCycle(int[] numbers)
{
    var i = 0;
    while (i < numbers.Length)
    {
        Console.WriteLine(numbers[i++]);
    }
}
```

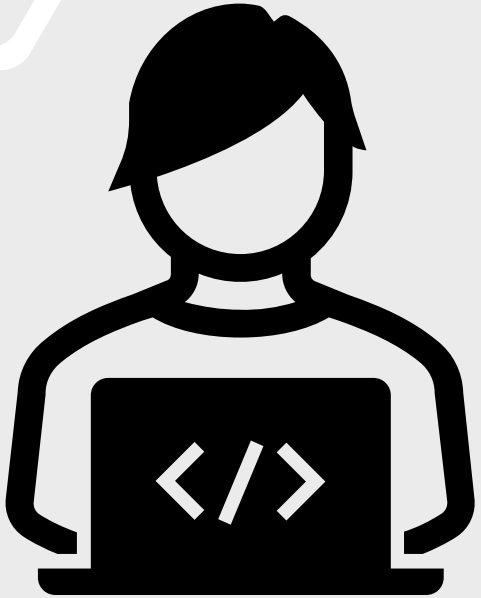
## *do while*

```
public void DealWithDoWhileCycle(int[] numbers)
{
    var i = 0;
    do
    {
        Console.WriteLine(numbers[i++]);
    } while (i < numbers.Length);
}
```

## *foreach*

```
public void DealWithForeachCycle(int[] numbers)
{
    foreach(var num in numbers)
    {
        Console.WriteLine(num);
    }
}
```

# Let’s practice!



# It’s coffee time!

# Other .NET key concepts

Core aspects

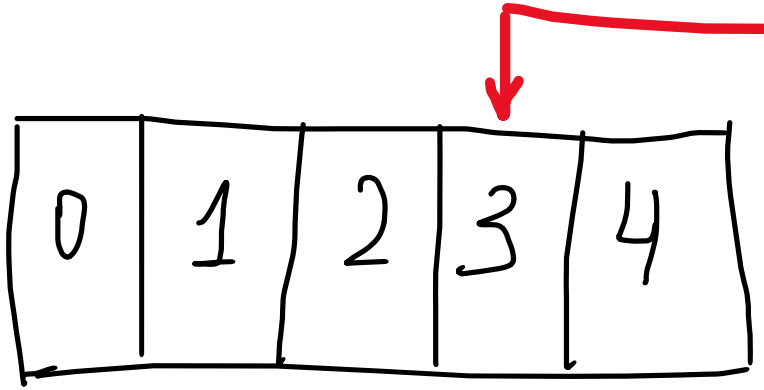
# Enumerators

# Enumerators

*How to enumerate all children?*

- “heads count”
- call by name
- call by surname
- visual analysis
- AI (computer vision)
- ask parents to find their child after the classes

# Enumerators




class CustomEnum

Collection

Can be enumerated

IEnumerable<T>

Enumerator

Can enumerate

IEnumerator<T>

# Enumerators

```
namespace System.Collections.Generic
{
    public interface IEnumerable<out T> : IEnumerable
    {
        IEnumerator<T> GetEnumerator();
    }
}
```

```
namespace System.Collections.Generic
{
    public interface IEnumerator<out T> : IEnumerator, IDisposable
    {
        T Current { get; }
    }
}
```

```
namespace System.Collections
{
    public interface IEnumerator
    {
        object Current { get; }

        bool MoveNext();
        void Reset();
    }
}
```

# Enumerators

*Can we do something like this?*

```
static void Main(string[] args)
{
    foreach (var item in new CustomEnumerable("ho-ho-ho!"))
    {
        Console.WriteLine(item);
    }
}
```

# Enumerators

*Yes, VR!*

```
internal class CustomEnumerable
{
    private readonly string _str;


    1 reference
    public CustomEnumerable(string str)
    {
        _str = str;
    }

    1 reference
    public CustomEnumerator GetEnumerator()
    {
        return new CustomEnumerator(_str);
    }
}
```

```
internal class CustomEnumerator
{
    private readonly string _str;

    private int _counter = -1;

    1 reference
    public CustomEnumerator(string str)
    {
        _str = str;
    }

    1 reference
    public char Current => _str[_counter];

    1 reference
    public bool MoveNext()
    {
        var hasNext = _counter < _str.Length - 1;
        if (hasNext)
        {
            _counter++;
        }

        return hasNext;
    }
}
```

# Enumerators live demo

# Strings

*Strings…*

… are objects

… store char arrays inside

… can be interned (deduplicated by CLR)

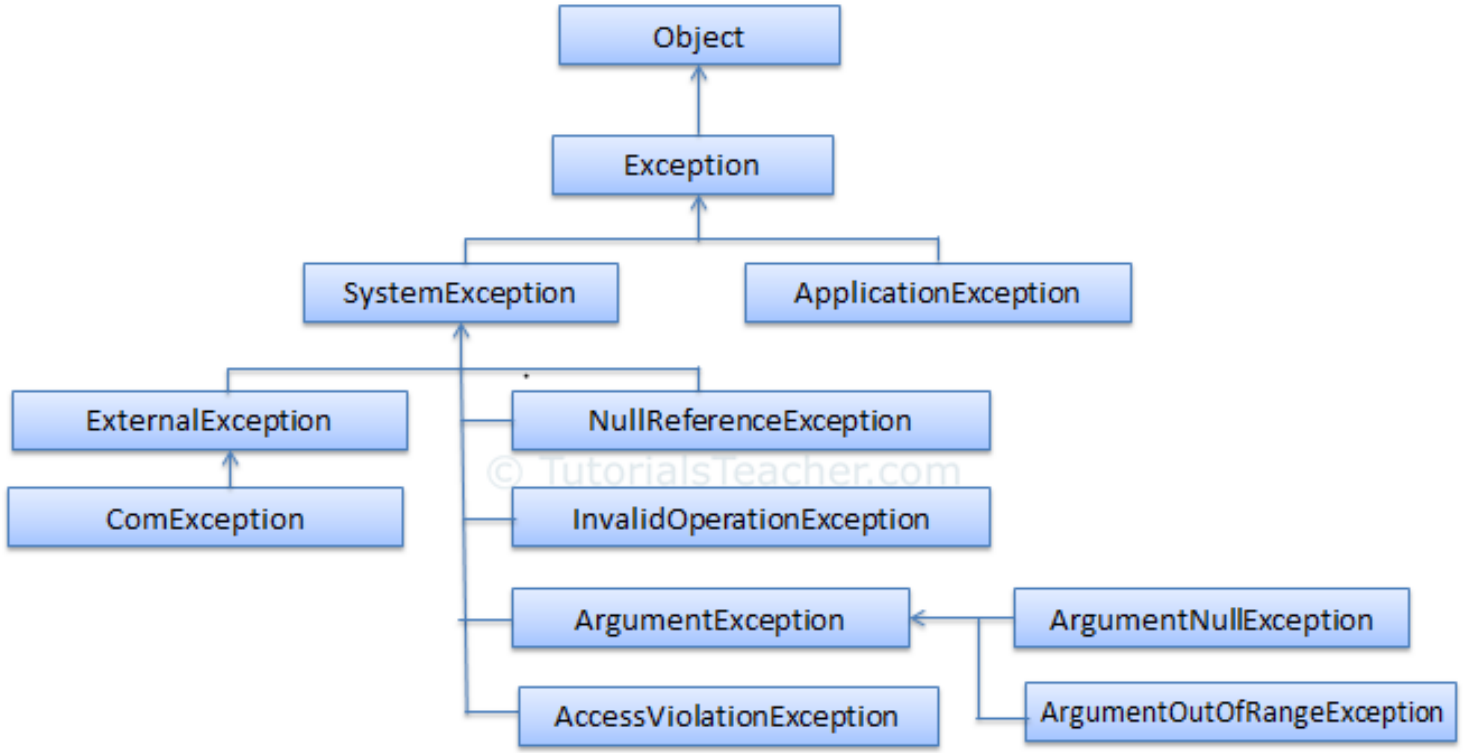
… can be formatted

… can be interpolated

… are very complex thing (encoding, equality, regional aspects etc.)

# Strings live demo

# Exceptions


Object
Exception
SystemException
ApplicationException
ExternalException
NullReferenceException
© TutorialsTeacher.com
ComException
InvalidOperationException
ArgumentException
ArgumentNullException
AccessViolationException
ArgumentOutOfRangeException

# Exceptions live demo

# It’s coffee time!

# Debugging basics

Core aspects

# Visual Studio debugging live demo

# Books of the day

[Jeffry Richter – CLR via C#. 4th Edition]

# Links of the day

C# Operators – MSDN Docs

Implicitly typed variables - MSDN Docs

IEnumerator<T> - MSDN Docs

Strings and other objects formatting – MSDN Docs

Exceptions - MSDN Docs

Exception handling best practices - Habr

# Hometask

- Search for the information about code life cycle. How can you describe compilation process? What is the main reason to use Intermediate Language? What is JIT compilation?

- Write your first program. You should create a console application which will meet the next requirement:
  - ✓ application asks user for the simple questions: name, surname, age, hobby. After all questions will be asked, application prints the summary like this:

    *Name: Andrei*

    *Surname: Ivanov*

    *Age: 25*

    *Hobby: guitar*
  - ✓ application should check the data before saving it. For example, if user entered “asd” on the “Age” step, application should print the notification “Please, enter the valid age. It should be a number” and wait for the correct information

# That’s all for this time!