# Ultimate C# Masterclass - List of topics

*This document is a part of “Ultimate C# Masterclass” Udemy course:* [*https://www.udemy.com/course/ultimate-csharp-masterclass/?referralCode=43763BBDA97D7D0B571A*](https://www.udemy.com/course/ultimate-csharp-masterclass/?referralCode=43763BBDA97D7D0B571A)

# C# Fundamentals

### Our first C# program

* How to create C# projects
* What IDE is
* What comments are
* How to print messages to the console
* How to configure the options of Visual Studio

### From a text file to an executable program

* What compilation is
* What the program performing it is called (compiler)
* What the compilation result is
* What \*.dll files are

### Programmer's most important skill

* What the most important skill of any programmer is
* What to do when our programs don't work as expected
* What StackOverflow is

### Variables

* What variables are
* What int and string types are
* What the variable’s declaration and initialization are, and what the difference between them is
* The shortcut that duplicates a line of code

### Naming variables & introduction to clean code

* How to name variables
* What reserved keywords are, and how to use them as variable names
* Why clean code is important
* Shortcut for renaming elements in code

### Operators

* The most basic C# operators ( +, -, \*, /, + +, - - )
* What operators precedence is

### Implicitly typed variables

* The purpose of the var keyword
* What implicitly and explicitly typed variables are

### User input

* How to read user input from the console
* What code snippets are
* What the role of warnings is

### Debugging with breakpoints

* How to use breakpoints to debug a program
* What QuickWatch window is and how to use it

### Comments

* How to add single-line and multiline comments to our code
* Shortcuts for commenting and uncommenting the code
* How to write in multiple lines at once

### Boolean type. Logical negation, equality, comparison, and modulo operators

* New type: bool
* ==, !=, !, >, >=, <, <=, % operators

### AND (&&) and OR (||) logical operators

* AND (&&) and OR (||) logical operators
* What short-circuiting optimization is
* In what order we should put logical expressions
* How to name boolean variables

### if/else conditional statement

* if/else conditional statements

### Scope

* What scopes are
* Where particular variables can be used and where they can not
* What code blocks are
* What nested if statements are

### Methods - part 1 - void methods

* What methods are
* How to define void methods
* The difference between defining the method and calling it
* The difference between method’s parameter and argument

### Methods - part 2 - non-void methods

* How to define non-void methods
* What the purpose of the return keyword is
* How to debug methods
* Under what circumstances we can remove the else following the if
* How to use Quick Actions and Refactorings menu
* What code refactoring is

### Methods - part 3 - parameters types and the return type. Static typing in C#

* The difference between statically and dynamically typed programming languages
* How to use the correct types of method parameters and the return type
* The difference between runtime errors and compilation errors

### Parsing a string to an int (int.Parse method)

* What parsing is
* How to convert a string to an int using the int.Parse method

### The first exception

* What exceptions are

### Adding a new project to a solution

* How to add more than one project to a solution
* How to switch between projects

### Assignment - Simple Calculator - Solution

* How to perform code refactoring in practice
* How to transform a string to its uppercase version

### String interpolation

* What string interpolation is, and how to use it

### Switch statement

* What switch statements are
* One of the purposes of the default keyword
* What expressions are

### Char

* What chars are

### A need for loops

* What loops are and what they are useful for
* What kinds of loops we can define in C#

### While loop - part 1

* How to define a while loop

### += and ++ operators. Infinite loops

* +=, \*=, ++ operators
* What infinite loops are

### While loop - part 1

* How to define a while loop - practice

### Do-while loop

* How to define a do-while loop
* The difference between the while and the do-while loops

### For loop

* How to define a for loop

### Break

* The purpose of the break keyword

### Continue

* The purpose of the continue keyword

### Nested loops

* What nested loops are

### Loops performance

* What the performance of a program is
* How loops can affect the program’s performance
* What we can do to avoid performance degradation when using loops

### Arrays

* What arrays are
* The purpose of the index from end operator (^)
* How to easily fill a new array with items using the array initializer
* What the greatest disadvantage of arrays is

### Multi-dimensional arrays

* What multi-dimensional arrays are

### Foreach loop

* How to use the foreach loop

### Lists

* What Lists are
* The difference between Lists and arrays
* The most crucial methods we can use with Lists

### “out” keyword

* The purpose of the out keyword

### TryParse method

* How to parse a string to an int without the risk of a runtime error
* The keyboard shortcut for formatting the code

### Assignment - TODO List - Implementation - User options & adding a TODO

* Practice the mechanisms learned in this section

### Assignment - TODO List - Implementation - Listing and Removing TODOs

* Practice the mechanisms learned in this section
* How to easily change a foreach loop into a for a loop

### Assignment - TODO List - Refactoring

* How to perform code refactoring in practice
* How to easily jump to a method definition
* What the recommended order of methods in a file is

# Basics of Object-Oriented Programming

### The issues in our code. A need for object-oriented programming

* Why we need object-oriented programming
* What procedural programming is and what problems it can cause
* What an antipattern is
* What spaghetti code antipattern is
* What high-quality code must always be ready for

### Introduction to object-oriented programming

* What object-oriented programming is
* What classes, objects and instances are
* What the benefits of object-oriented programming are

### Understanding OOP with the DateTime type

* How OOP can be used in practice
* New type: DateTime
* What a constructor is

### Abstraction

* What abstraction is
* The benefits of hiding the implementation details from the users of a class

### Our first class

* How to define a class
* What the fields of a class are
* What the default values of fields are
* What a default constructor is

### Data hiding

* What data hiding is and why using it is a good idea
* What members of a class are
* What access modifiers are
* Public and private access modifiers
* What the default access modifier for fields is

### Custom constructor

* How to define custom constructors in a class
* What the recommended naming for fields is

### C# restriction on code outside classes. Top-level statements

* Whether C# code can be defined outside classes or if it must always be contained within one
* What top-level statements are

### Methods in classes

* How to define methods in classes
* How methods should be named
* The rule of thumb we can use to find out what the default access modifiers in C# are

### Encapsulation

* What encapsulation is
* How is encapsulation different from data hiding
* The benefits of using encapsulation

### Methods overloading

* What methods overloading is
* What rules we must follow when defining many methods with the same name in a class
* How to quickly create a constructor using Visual Studio

### Constructors overloading. Calling one constructor from another

* How to overload constructors
* How to call one constructor from another using the “this” keyword

### Expression-bodied methods

* How to make methods shorter by converting them to expression-bodied methods
* The difference between a statement and an expression

### "this" keyword (current instance reference)

* How to use the “this” keyword to refer to the current instance of a class

### Optional parameters

* How to define optional parameters
* How to set the default value of a parameter

### Validation of constructor parameters

* How to validate the constructor parameters
* The purpose of the nameof expression
* Why having public fields is risky

### Readonly and const

* How to prevent a field from being modified
* What immutable objects are
* The difference between readonly and const

### Limitations of fields. A need for properties

* The limitations of fields
* How can those limitations be addressed using methods
* Why we need properties

### Properties

* What properties are
* What a backing field of a property is
* What accessors are
* What the differences between fields and properties are
* When should we use fields, and when properties

### Object initializer

* What object initializers are
* The purpose of the init accessor

### Computed properties

* How to create computed properties
* When to use parameterless methods, and when computed properties

### Static methods and classes

* What static methods are
* What static classes are
* What are the limitations of static methods
* Why all const fields are implicitly static

### Static fields, properties and constructors

* What static fields and properties are
* What a static constructor is
* Whether using static fields and properties is a good or bad practice

### Single Responsibility Principle - introduction

* What the Single Responsibility Principle is (S in SOLID)
* What SOLID principles are
* How to read from and write to a text file

### Single Responsibility Principle - refactoring (part 1)

* How to perform step-by-step refactoring of a class, so it meets the Single Responsibility Principle
* What a repository is
* What the new line symbol for Unix and non-Unix systems is

### Single Responsibility Principle - refactoring (part 2)

* What is the recommended order of methods in a class
* What is the risk of having some properties public, even only for getting

### Single Responsibility Principle - refactoring (part 3)

* What the DRY principle is
* When code duplications are not a bad thing

### Files, namespaces, usings

* How to add new files to a project
* How to move classes to their own files
* What namespaces are
* What using directives are
* What file-scoped namespace declarations are

### Global using directives

* What global using directives are
* How to measure the time of code execution

### Assignment - Dice Roll Game - Random

* How to generate pseudorandom numbers
* What the seed of the pseudorandom numbers generator is
* What a dependency of a class is

### Assignment - Dice Roll Game - Magic Number antipattern

* What the magic number antipattern is and how to avoid it

### Assignment - Dice Roll Game - Designing classes

* How to create a neat classes design in practice

### Assignment - Dice Roll Game - Enums

* What enum types are and how to declare them
* What the underlying type of enums is
* What casting is

### Assignment - Dice Roll Game - Ternary conditional operator

* How to use the ternary conditional operator
* How to logically place types in namespaces

# Object-Oriented Programming: Polymorphism, Inheritance, Interfaces

### A need for polymorphism

* Why do we sometimes need to manipulate different types in an uniform way
* What polymorphism is

### Inheritance

* What inheritance is
* What kind of relationship it creates between types
* What base classes and derived classes are

### Inheriting members from the base class. Protected access modifier

* How to make base class members accessible in the derived class
* The purpose of the protected access modifier

### Overriding members from the base class. Virtual methods and properties

* How to override the implementation of the method or a property from a base class in the derived classes
* What virtual methods and properties are
* What method hiding is

### Virtual methods - practice

* How to use virtual methods in practice

### A deeper inheritance hierarchy

* In this video, we will learn How to define an inheritance hierarchy of more than two classes.

### Multiple inheritance

* Why in C# a class cannot derive directly from more than one base class
* What the diamond problem is

### System.Object and the ToString method

* What the System.Object class is and what method it contains
* What the ToString method is and what its basic implementation is

### Inheriting constructors and the "base" keyword

* If the base class constructor is called when a derived class object is created
* How to create constructors that set the state defined in both the base type and the derived type
* The purpose of the “base” keyword

### Implicit conversion

* What implicit conversion is
* New types: decimal and double

### Explicit conversion

* What explicit conversion is
* What problems may it cause

### Upcasting and downcasting

* The difference between upcasting and downcasting
* Which of them is risky

### “is” operator

* The purpose of the “is” operator

### Null

* What null is

### “as” operator

* The purpose of the “as” operator
* “as” operator limitations
* The difference between conversion with the cast expression, and with the “as” operator

### Abstract classes

* How to prevent a class from being instantiated
* What abstract classes are

### Abstract methods

* What abstract methods and properties are

### A need for abstract methods

* Why we need abstract methods
* The difference between abstract and non-abstract virtual methods

### Sealed classes and methods

* What sealed classes and methods are
* What the reasons for sealing classes and methods may be

### Static classes are always sealed

* Why static classes are implicitly sealed
* Why overriding of static methods is not possible

### Extension methods

* How to define extension methods
* How to create multiline string literals

### A need for interfaces

* Under what circumstances using an abstract class as a base type is not a good idea
* Why we need interfaces

### Interfaces

* What interfaces are
* What kind of relationship they create between types

### Interfaces vs abstract classes

* What the differences between interfaces and abstract classes are

### JSON

* What JSON and XML are
* How to serialize a C# object to JSON format, and how to deserialize JSON string to a c# object
* How to escape the quote character in a string

### Assignment - Cookies Cookbook - High-level design

* How to define the high-level logic of the application
* How to easily create well-designed interfaces

### Assignment - Cookies Cookbook - Dependency Inversion and Dependency Injection

* What the Dependency Inversion Principle is (D in SOLID)
* What Dependency Injection is, and how it is different from Dependency Inversion
* What coupling is
* What target-typed new expressions are

### Assignment - Cookies Cookbook - Designing data types

* What generic types and methods are
* What the IEnumerable interface is
* How to reduce code repetition between related types in practice

### Assignment - Cookies Cookbook - Printing data object. LINQ.

* What LINQ is
* How to access the index of the current iteration in a foreach loop

### Assignment - Cookies Cookbook - Printing the ingredients

* How to create a basic storage class for objects of a given type

### Assignment - Cookies Cookbook - Composing the recipe by the user

* How to find an item in a collection by one of its properties

### Assignment - Cookies Cookbook - Reading and writing from and to a \*.txt file

* How to implement reading from and writing to a \*.txt file

### Assignment - Cookies Cookbook - Reading and writing from and to a \*.json file

* How to implement reading from and writing to a \*.json file

### Assignment - Cookies Cookbook - Template Method Design Pattern

* What the Template Method design pattern is, and how to use it in practice

### Assignment - Cookies Cookbook - Cleanup and project organizing

* How to organize a project into namespaces in practice
* How to apply some action in the Visual Studio for the entire project

# Exceptions and error handling

### Exception object

* What exceptions are
* What the System.Exception class is and what data it contains
* How to see detailed information about an exception that occurred in our code

### Stack trace

* What stack trace is and how it is useful
* Why unhandled exceptions are bad news

### Handling exceptions. Try-catch-finally

* How to handle exceptions with the try-catch-finally blocks

### Multiple catch blocks

* How to catch exceptions of a specific type
* How to define multiple catch blocks for a single try block
* Why the order of catch blocks matter

### Throwing exceptions explicitly

* How to throw exceptions from our code explicitly
* How throwing an exception allows us not to return a value from a method
* When throwing an exception is a good design choice
* How to define valuable exception messages

### Built-in exception types

* What are some of the built-in exception types

### StackOverflowException. Recursive methods

* What StackOverflowException is
* What recursive methods are
* What risks may recursive methods cause and how they can be mitigated

### Precise exceptions

* Why it is important to be precise when using exceptions
* Why in most cases, we should avoid throwing and catching the base SystemException type.

### Rethrowing exceptions. Throw vs throw ex

* What rethrowing exceptions is
* What the difference between “throw” and “throw ex” is, and which one we should use
* Why the InnerException property is so important

### Rethrowing a System.Exception object

* How to manage exceptions if we don't know exactly what exceptions can be thrown from some code
* What might be a good reason to catch the exception of the System.Exception type

### Global try-catch block

* What a global try-catch block is

### Code inside the catch block

* What code we should and what we should not put into the catch block
* What happens if an exception is thrown from a catch block
* What nested try-catch blocks are

### Exception filters

* What exception filters are
* How exception filters can let us better control what exceptions will be processed by a catch block

### Custom exceptions

* How to define custom exceptions

### When to use custom exceptions

* When we should define custom exceptions
* What the Principle of least surprise is

### Exceptions as a hidden part of a method signature

* That exceptions that a method may throw are, in a way, a hidden part of this method's signature

### Two extreme schools of using exceptions

* What the drawbacks of using exceptions are
* What the goto statement is and why using it is considered a bad idea
* What the alternatives for using exceptions are
* What issues we can have if we decide not to use exceptions at all

### Smart usage of exceptions - throw

* How to be smart about using exceptions
* when throwing exceptions explicitly from our code is a good idea

### Smart usage of exceptions - catch

* How to use the catch block smartly
* When we should put code in a try-catch block
* What a catch block should do

### Assignment - Game Data Parser - Sunny day scenario

* How to quickly build C# classes based on types defined in JSON

### Assignment - Game Data Parser - Flow controlled by exceptions

* How exceptions can control the flow of the application
* How to use the "default" keyword to assign a default value to a variable of any type

### Assignment - Game Data Parser - Adding details to JsonException

* How to wrap existing exceptions into new ones
* How to change the font color in console applications

### Assignment - Game Data Parser - Custom logger

* How to create a simple exceptions logger
* What are some of the logging libraries for .NET

### Assignment - Game Data Parser - Fewer exceptions

* How can the number of try-catch block in an application be reduced

### Assignment - Game Data Parser - Refactoring - Extracting methods

* How to extract smaller, more focused methods from a single, big method

### Assignment - Game Data Parser - Refactoring - SRP, DI, and classes decoupling

* How to refactor the code to meet Single Responsibility Principle and the Dependency Inversion Principle

# Generic types & advanced use of methods

### Introduction to generic types

* What generic types are and what their purpose is

### Understanding how List works under the hood

* How the List works under the hood
* What operations can negatively impact the List's performance
* What data structures are

### Simplified List

* How to implement a simplified List

### Simplified List (deleting an item at a given index)

* How to implement a simplified List
* What indexers are

### A need for generic types. Implementing a generic type

* Why generic types are a crucial part of the language
* How to create generic types
* The purpose of the default keyword

### A need for tuples

* How to handle a situation when we want to return more than one result from a method
* What an algorithm is
* How to implement an algorithm for finding a minimal and maximal number in a collection

### Tuples

* How to define generic types in practice
* How to implement a custom tuple
* How to use build-in tuples

### C# without generics. ArrayList

* How programmers used to manage their code when generics were not a part of C#
* What an ArrayList is and what issues it can cause

### Generic methods

* How to define generic methods
* How the compiler infers the type parameter from the context in which some method is used

### Generic methods with multiple type parameters

* How to define generic methods with more than one type parameters
* How to convert the collection of one type into a collection of another type

### Convert.ChangeType method. Typeof keyword and the Type object

* How to convert objects of any type into objects of any other type
* What the Type class is
* The purpose of the "typeof" keyword and the GetType method

### A need for type constraints

* What type constraints are and what their purpose is
* The purpose of the “where” keyword
* How the parameterless constructor constraint works

### Improving the performance of the List. Measuring the time of the method’s execution

* How we can improve the performance of a list to which many items are added one by one
* How to measure the time of the code execution using the Stopwatch class

### Type constraints - the constraint on the base type

* More about type constraints
* How we can limit the generic type arguments only to types derived from a certain base class

### IComparable interface. Ordering objects

* How to sort Lists of various types
* What the IComparable<T> interface is and how to implement it

### Type constraints - the constraint on the implemented interface

* How we can limit the generic type arguments only to types implementing a certain interface

### Type constraints - numeric types. Generic math

* What type constraint allows us to limit the type argument to only numeric types
* What the generic math feature is

### Type constraints - summary. Multiple type constraints

* How to define multiple constraints for a single type parameter
* How to manage constraints for multiple generic type parameters.

### Funcs and Actions

* How we can assign methods to variables
* What Funcs and Actions are
* How Funcs and Actions can help us reduce code repetitions

### Lambda expressions

* What lambda expressions are

### Delegates

* What delegates are
* The difference between delegates and Funcs or Actions
* What a multicast delegate is

### Dictionary - introduction

* What Dictionaries are

### Dictionary - practice

* How to use Dictionaries in practice

### A need for the Strategy design pattern

* What code could benefit from using the Strategy design pattern

### Refactoring the code using Funcs and lambda expressions

* How to refactor the code using Funcs and lambda expressions

### Open-Closed Principle. Strategy design pattern

* What the Strategy design pattern is
* What the Open-Closed Principle is (O in SOLID)
* How to use Funcs, Dictionaries and generic methods in practice

### Generic filtering of collections

* How to access the collections of keys of a Dictionary
* How to define generic methods in practice

### Caching

* What caching is

### Assignment - Custom Cache - Implementation

* How to implement a simple caching mechanism

### Assignment - Custom Cache - Decorator design pattern

* What the Decorator design pattern is
* How it can help us follow the Open-Closed Principle

### Assignment - Custom Cache - Composing many Decorators together

* How we can compose many decorators together to add more than one feature to a decorated object

# 

# LINQ

### What is LINQ

* What LINQ is
* What the benefits of using it are
* What LINQ queries look like

### LINQ and extension methods

* How it is possible that we can call the same LINQ methods on different types of collections

### LINQ, IEnumerable and method chaining

* What is the relation between LINQ and IEnumerable interface

### Deferred execution

* What deferred execution is
* How it can improve the performance of our applications

### Any

* How to use the Any method from LINQ

### All

* How to use the All method from LINQ

### Count

* How to use the Count and LongCount methods from LINQ

### Contains

* How to use the Contains method from LINQ

### OrderBy

* How to use the OrderBy, OrderByDescending, ThenBy, ThenByDescending methods from LINQ

### First and Last

* How to use the First and Last methods from LINQ (along with FirstOrDefault and LastOrDefault)

### Where

* How to use the Where method from LINQ

### Select

* How to use the Select method from LINQ

### Average. Anonymous types

* What anonymous types are
* How they can be used with LINQ
* How to calculate the average value in a collection of numbers

### Assignment - Refactoring to LINQ - Nested loop and code readability

* How to refactor code to use LINQ
* Whether making code shorter is always a good idea

### Assignment - Refactoring to LINQ - Find and Replace Windows

* How to use the Find and Replace tool to perform an advanced search of a given phrase
* How to rearrange Visual Studio windows
* What regular expressions are

### Assignment - Refactoring to LINQ - Fewer loops & multiline strings formatting

* How to refactor loops using LINQ queries
* Why we must be careful when formatting the code using multiline string literals

### Assignment - Refactoring to LINQ - Checking if collection has duplicates

* How to find an item matching a given predicate with LINQ (practice)
* How to reduce the number of times LINQ iterates an input collection
* How to check if a collection contains any duplicated elements

# 

# .NET under the hood

### .NET and C#

* What .NET is and how it is different from C#
* What are the examples of .NET-compatible programming languages
* What are the examples of .NET-related technologies
* What the differences between .NET Framework, .NET Core and .NET are

### Common Intermediate Language (CIL)

* What the Common Intermediate Language is
* How and when it is compiled into binary code by the Just-in-Time compiler
* How can C# code can communicate with other .NET-compatible languages

### Common Language Runtime (CLR)

* What the Common Language Runtime (CLR) is

### Memory of a program. The stack and the heap

* How memory is organized in .NET applications
* What the stack and the heap are and what the difference between them is

### Value semantics vs reference semantics

* What value semantics and reference semantics are.

### Value types vs reference types

* What the difference between value types and reference types is

### Value types vs reference types - practical tips

* How to use value and reference types in practice
* What an impact of changing a class to a struct may be
* What some of the benefits of using immutable types may be

### “ref” keyword

* What the purpose of the “ref” modifier is
* What the difference between “ref” and “out” modifiers is

### Using “ref” with reference types

* How the “ref” modifier can be used with parameters of reference types

### Unified type system. A need for boxing and unboxing

* What a unified type system is in C# and why it is so important

### Boxing and unboxing

* What boxing and unboxing are

### Boxing and unboxing - performance cost

* What the performance impact of boxing and unboxing is
* What the size of a reference is

### Garbage Collector - introduction

* What Garbage Collector is
* Under what circumstances it may be triggered to start its work
* How it can affect the performance of our applications

### Garbage Collector - Memory fragmentation and defragmentation

* What memory fragmentation is
* How it can be fixed in a process called defragmentation

### Garbage Collector - The Mark-And-Sweep algorithm

* How the Garbage Collector decides what objects can be removed from memory using the Mark-and-Sweep algorithm
* What Reference counting is and what disadvantages it has
* What a circular reference is

### Garbage Collector - Generations of objects

* What generations of objects are
* How they improve the performance of the Garbage Collector
* What the Large Objects Heap is
* What it means that an object is pinned

### Memory Leaks

* What memory leaks are
* How having static fields in classes may cause the risk of memory leaks

### Finalizers

* What destructors (also known as finalizers) are
* When we should define them

### Dispose method - introduction

* What he purpose of the Dispose method coming from the IDisposable interface is
* What managed and unmanaged resources are.

### Dispose method - writing to a file with the StreamWriter

* How to create a class that writes to a file using the StreamWriter class

### Dispose method - reading from a file with the StreamReader

* How to use the StreamWriter class to read from a file
* Why disposing of unmanaged resources is so critical

### Dispose method - implementation

* How to use the Dispose method to free unmanaged resources
* What using statement is
* What syntactic sugar is

### CSV Files

* What CSV files are
* Why there is a significant chance you will work with them one day

### Reading CSV files

* How to create a simple class that can load data from a CSV file
* How to use StreamReader in practice
* Why the backslash ( \ ) is a special character and how it can allow us to escape other special characters in strings

### Assignment - CSV Processing Improvements - Code analysis & tips

* What the existing code in the assignment solution does

### Assignment - CSV Processing Improvements - Reducing the size of Dictionaries

* How we can reduce a size of a Dictionary representing a row in a CSV

### Assignment - CSV Processing Improvements - Reducing the number of boxings

* How to reduce the number of boxing operations in an application
* Why some performance improvements only make sense for specific input data

### Assignment - CSV Processing Improvements - Analysis

* That when making performance improvements, we must be aware of both the nature of the data we operate on and the users' expectations

# 

# Advanced C# types

### Reflection

* What reflection is and what it is useful for
* How to use the Type object in practice
* What the upsides and downsides of reflection are

### Attributes

* What attributes are and how to use them
* How to define a custom attribute
* How to use reflection in practice
* What metadata is

### Limitations of attributes parameters types

* The limitations of attributes parameters types

### Structs

* What structs are and how they differ from classes
* Why structs should rather be small
* How to define type constraints to only allow using value types or reference types as a generic type parameter

### Structs vs classes - crucial differences

* The most crucial differences between structs and classes

### Structs vs classes - low-level differences

* What the low-level differences between classes and structs are

### Choosing between structs and classes

* When we should use classes and when we should use structs

### Why should we make structs immutable?

* Why making structs immutable is a good idea

### Non-destructive mutation

* What non-destructive mutation is

### “With” expression

* How to use the "with" expression to perform non-destructive mutation

### Readonly structs

* How to enforce the immutability of a struct

### A close look at the System.Object type. The ReferenceEquals method.

* What methods we can call on any object in C#
* What the ReferenceEquals method does

### Equals method

* What the Equals method from the System.Object type is
* What its default behavior for value and reference types is

### Overriding the Equals method in classes

* How to override the Equals method in classes

### Overriding the Equals method in structs

* What may be the reasons for overriding the Equals method in structs
* How to generate the Equals method override with the help of Visual Studio

### IEquatable<T> interface

* What the IEquatable<T> interface is and why we should bother implementing it
* How it is different from the IComparable<T> interface
* What happens if there are two methods with the same name in a type, which could both be used with a given argument

### == operator

* What the == operator does
* How its behavior differs for value and reference type

### Operators overloading

* How operator overloading works in C#
* How to overload the +, ==, != operators
* Which operators can be overloaded, and which cannot

### Overloading of implicit and explicit conversion operators

* What implicit and explicit conversion operators are and how to overload them

### Hash functions

* What hash functions are and how they relate to the GetHashCode method
* What the characteristics of a good hash function are
* What hash conflict is and why it is inevitable

### Default implementation of the GetHashCode method

* What the default implementation of the GetHashCode method is for value and reference types

### When to override the GetHashCode method

* When we should override the default implementation of the GetHashCode method

### Overriding the GetHashCode method

* How to override the GetHashCode method in our types
* How to use the HashCode.Combine method

### ValueTuples

* What ValueTuples are and how they are different from tuples

### Benefits of immutable types

* The benefits of using immutable types
* What pure functions are
* The downsides of using immutable types

### Records

* What records and positional records are
* What the benefit of using them is

### Record structs

* What record structs are

### Nullable value types

* What nullable value types are

### Nullable reference types

* What nullable reference types are
* What code review is

### Null-forgiving operator

* The purpose of the null-forgiving operator
* How the warnings shown in the Visual Studio can help us improve our code

### Using nullable reference types. Generic type constraints related to nullability

* When to use nullable reference types
* How to disable or enable this feature in specific parts of the code using preprocessor directives
* The generic type constraints for nullable and non-nullable types

### APIs

* What APIs are

### Querying an API using C#

* How to read data from a public API in our C# programs
* How to use asynchronous methods and how to await their execution

### A class for querying APIs

* How to create a class that reads JSON strings from any open API
* The limitations of using the await keyword and how to make a method asynchronous
* Deserializing JSON data to C# objects in practice

### Assignment - Star Wars Planets Stats - JsonPropertyAttribute and DTOs

* The role of the JsonPropertyName attribute
* What DTOs are
* How to use Quick Actions shortcut

### Assignment - Star Wars Planets Stats - Exceptions handling

* Handling exceptions in our apps in practice
* How to quickly find out what exceptions may be thrown by a method
* What documentation comments are

### Assignment - Star Wars Planets Stats - Type design

* How to design the types our programs use
* How to use the QuickWatch window

### Assignment - Star Wars Planets Stats - Converting DTO to a custom type

* Overloading the explicit conversion operator in practice
* Why we shouldn't mingle the types coming from the API with our custom types (and how it can be avoided)

### Assignment - Star Wars Planets Stats - Finishing the app and the MaxBy method

* How to use the MaxBy and MinBy methods from LINQ

### Assignment - Star Wars Planets Stats - Refactoring

* How to use the messages related to the code analysis performed by Visual Studio
* The purpose of the null-coalescing assignment operator.

### Assignment - Star Wars Planets Stats - Splitting the class

* How to split a class into smaller classes, to make it compliant with the Single Responsibility Principle

### Assignment - Star Wars Planets Stats - Universal table printer

* Using reflection in practice
* How to format strings

# 

# Collections

### The role of the IEnumerable interface

* What the main interfaces related to collections are
* What the role of the IEnumerable interface is

### A close look at the IEnumerable interface

* What methods are defined in the IEnumerable interface
* How they are used when a foreach loop is executed

### Implementing IEnumerable

* How to implement the IEnumerable interface in custom collections

### Implicit and explicit interface implementation

* What methods are required by the IEnumerable<T> interface
* What Implicit and explicit interface implementations are

### Implementing IEnumerable<T>

* How to implement the IEnumerable<T> interface
* What backward compatibility is
* What named arguments are

### Indexers

* What indexers are and how to define custom indexers

### Collection initializers

* How to implement collection initializers in our custom collection

### ICollection and IList interfaces

* About two interfaces related to collections - ICollection and IList interfaces

### Breaking of Interface Segregation Principle

* What happens if a type is forced to implement an interface that it cannot implement in any reasonable way

### Interface Segregation Principle

* What the Interface Segregation Principle is (I in SOLID)

### The benefits of readonly collections

* What readonly collections are and what the benefits of using them are

### Readonly collections. ReadOnlyCollection and ReadOnlyDictionary

* How a collection can be made readonly

### Big O Notation

* What Big O notation is
* How it can help us understand the complexity of an algorithm

### Binary search algorithm

* How the binary search algorithm works
* What Divide-and-conquer strategy for solving problems is

### Binary search algorithm - implementation

* How to implement the binary search algorithm in C#

### Binary search algorithm - complexity

* What the time complexity of the binary search algorithm is
* What the logarithmic complexity is
* Why using binary search may be a good idea
* How to use the built-in version of this algorithm

### Improving performance when using Lists

* Tips for improving code performance when using Lists
* How to generate a collection of numbers using Enumerable.Range method
* How to write long numbers in a readable way

### Linked list

* What linked lists are

### Linked list vs List

* How basic operations differ for Linked lists and lists
* What the performance differences between those data structures are
* When to use Lists, and when Linked lists

### Dictionaries under the hood

* How Dictionaries word under the hood
* What hash tables are
* Why overriding the GetHashCode and Equals methods together is so important

### Performance of Dictionaries

* What the performance of basic operations performed on Dictionaries is

### HashSet

* What HashSets are and what the use cases for them may be
* How to remove the duplicates from a collection efficiently

### Queue

* What queues are
* What FIFO stands for
* What priority queues are

### Stack

* What stacks are
* What LIFO stands for

### “Params” keyword

* The purpose of the “params” keyword

### A need for yield statement

* What may be the use cases for yield statements

### yield statement - behavior analysis

* How the code using yield statements behaves

### yield statement and iterators

* How the yield statement works
* What the role of iterators is

### yield statement - practice. yield break statement

* How to use iterators in practice
* The purpose of the yield break statement

### Implementing IEnumerable interface using iterators

* How to implement the IEnumerable interface using iterators

### Assignment - Custom Linked List - Data Structures

* How to implement the Node data structure required by the linked list

### Assignment - Custom Linked List - The AddToFront method

* How to implement adding new items to the front of the linked list

### Assignment - Custom Linked List - Implementing IEnumerable

* How to implement the GetEnumerator method using iterators in practice

### Assignment - Custom Linked List - Adding new items at the end of the list

* How to implement the Add and AddToEnd methods for a linked list

### Assignment - Custom Linked List - The Clear method

* How to implement the Clear method for a linked list
* Why it is a bad idea to modify a collection that is being iterated

### Assignment - Custom Linked List - Removing items and the Contains method

* How to implement the Remove and Contains methods for a linked list

### Assignment - Custom Linked List - The CopyTo method

* How to implement the CopyTo method for a linked list

### Assignment - Custom Linked List - Summary and performance review. Private classes

* How the implemented SinglyLinkedList differs from the built-in LinkedList
* What nested classes and private classes are

# 

# Projects, assemblies, solutions

### Projects and solutions

* About C# projects and solutions
* How to add more than one project to the solution

### Project properties

* What the project’s properties are and how we can change them

### Debug build vs Release build

* The difference between Debug and Release builds

### Assemblies

* What assemblies are
* What the difference between a project and an assembly is

### Referencing types from another assembly

* How to reference an existing assembly in our code

### Referencing types from another project

* How to reference one project from another

### Internal access modifier. Principles of using access modifiers

* The purpose of the internal access modifier
* What happens when the type has a stricter access modifier than the members it contains
* Why the access modifiers we use within types and methods must be consistent

### Protected internal access modifier

* The purpose of the protected internal access modifier

### Private protected access modifier

* The purpose of the private protected access modifier
* The purpose of the file access modifier

### Access modifiers - summary

* The differences between access modifiers
* Which access modifiers should be used in what context

### How to structure the code in a solution

* How to split a solution into projects smartly
* What circular dependencies are

### NuGet

* What NuGet is

### \*.csproj files

* What \*.csproj files are

### \*.sln files

* What \*.sln files are

### Updating the .NET version

* How to update the version of .NET in the code we work on

# 

# Strings

### Char

* More about chars
* Basic methods for character manipulation

### Char representation in memory. Character encoding

* How characters are stored in memory
* What character encoding is
* How UTF-16 encoding works

### Managing various encodings

* How to deal with various encodings of characters

### Immutability of strings

* About the immutability of strings
* What the underlying data structure for strings is

### Strings - value or reference types?

* Whether strings are value or reference types

### Strings as members in structs

* Why structs should not contain fields or properties of reference types
* Why strings are an exception

### A need for StringBuilder

* What problems can be caused by an incremental building of large strings

### StringBuilder

* How to optimize the process of the incremental building of strings by using the StringBuilder class

### String interning

* What string interning is

### Flyweight design pattern

* The flyweight design pattern

### Advanced string formatting

* Advanced formatting of strings
* More about string.Format method

### Culture-specific string formatting

* About culture-specific string formatting
* What CultureInfo object is

### Specific culture vs Invariant culture

* What the difference between specific and invariant cultures is

### Assignment - Tickets Data Aggregator - Reading text from PDF

* How to extract textual data from a PDF document
* What OCR is

### Assignment - Tickets Data Aggregator - List all PDFs from a folder

* How to list all files with specific extensions from a given folder

### Assignment - Tickets Data Aggregator - Splitting a string by multiple separators

* How to split a string using a group of separators.

### Assignment - Tickets Data Aggregator - Parsing culture-specific strings

* Using specific cultures when managing strings
* DateOnly and TimeOnly types

### Assignment - Tickets Data Aggregator - Saving all results in a text file

* How to use invariant culture in practice
* How to use StringBuilder in practice

### Assignment - Tickets Data Aggregator - Refactoring

* How to refactor the code in practice, and how to split large method into smaller ones

### Assignment - Tickets Data Aggregator - Compliance with the SRP

* How to identify the responsibilities of a class in practice
* How to make a class compliant with the Single Responsibility Principle

### 

# 

# Numeric types

### Decimal number system

* How the decimal number system works

### Binary number system

* How the binary number system works

### Maximal numbers on a given number of digits

* The relation between the number of digits in a number and the maximal value of this number

### Numbers in memory. Integer

* How numbers are represented in a computer's memory
* What the size and range of an integer is

### Adding binary numbers

* How to add binary numbers

### Numeric overflow & silent failures

* What numeric overflow is
* Why it may be dangerous
* What silent failures are

### “checked” keyword

* The purpose of the “checked” keyword
* How to deal with numeric overflows

### Checked context - when to use it?

* When the checked keyword should be used
* What the alternatives for using the checked context may be

### Scope of the checked context. Unchecked keyword

* How to enable checking for numeric overflow globally for a project
* What the purpose of the "unchecked" keyword is
* What the scope of a checked context is

### Integral numeric types overview.

* About various integral numeric types

### Floating-point numbers

* What floating-point numbers are

### Double and float

* How float and double work

### Smart usage of floating point numbers

* How to deal with the fact that floats and doubles are not perfectly precise
* What are reasonable use cases for them
* What NaN is

### Decimal

* How to represent fractions in C# precisely
* More about decimal
* How is decimal different from double

# Events

### A need for communication between objects

* How sending notifications between software components may be implemented

### A need for the Observer design pattern

* What the characteristics of a good notification mechanism are

### Observer design pattern

* What the Observer design pattern is

### Defining an item and subscribing to it

* How to define events
* How to subscribe to an events

### Raising events

* How to raise events
* The purpose of the Invoke method
* What the purpose of the null-propagating operator is ( ?. operator )

### EventHandler delegate & EventArgs type

* How to use the built-in EventHandler delegate
* What EventArgs type is

### Event vs delegate members

* The difference between an event and a member of a delegate type

### Windows Forms - introduction

* What Windows Forms framework is

### The first Windows Forms app

* How to create a simple Windows Forms app

### Understanding Windows Forms files

* How a form we create using a designer is represented in C# file
* What regions are and how they can be defined
* What partial classes are

### Events in Windows Forms

* How to handle events that happen in the user interface of Windows Forms apps

### Windows Forms - basic UI elements

* basic UI elements used in Windows Forms apps

### Assignment - Numeric Types Suggester - User Interface & basic events

* Creating interfaces and configuring events in practice

### Assignment - Numeric Types Suggester - Handling KeyPress event

* How to handle the KeyPress events raised by the textboxes to ensure only valid input is accepted
* How to use the sender parameter of an event handler method

### Assignment - Numeric Types Suggester - Numbers validation and BigInteger type

* How to implement the validation of values in a form
* How to parse strings to BigIntegers

### Assignment - Numeric Types Suggester - Choosing Numeric Types

* How to implement the algorithm for finding the proper numeric type based on the parameters of the numbers

# Unit testing

### Manual tests vs Automated tests

* What automated tests are
* How they are better than manual tests
* Why is it worth

### Setting up the testing environment

* How to create a test project
* What NuGet packages are required to write and run unit tests

### The first unit test

* How to write a unit test
* What an assertion is

### Running unit tests

* How to run and debug unit tests
* What the Test Explorer is

### Naming unit tests

* How to name unit tests

### Test messages

* How to specify the test messages

### AAA pattern

* What the AAA pattern is

### Test cases

* What test cases are and how to define them

### Naming parameterized tests

* How to name tests consisting of many test cases

### TestCaseSource attribute

* What the limitations of the TestCase attribute are
* How to bypass them by using the TestCaseSource attribute

### Assertions related to exceptions

* How to write an assertion checking if an exception was thrown
* Whether it is a good idea to have more than one assert in a test

### Value of unit tests

* What the the value of unit tests is in practice
* What code coverage is

### Basic assertions

* The most basic types of assertions we can make in NUnit tests

### Testing private methods

* How to approach testing private methods

### Testing internal methods

* How to approach testing internal methods

### Benefits of unit tests - no fear of refactoring

* How unit tests enable refactoring and improve code quality
* How it relates to the costs of producing software

### Benefits of unit tests - better design

* How unit tests help create better code design
* What TDD is

### Benefits of unit tests - early bug detection

* How unit tests help detect bugs early

### Downsides of unit tests

* The downsides of unit tests
* The difference between white-box testing and black-box testing
* When it makes sense to skip unit tests

### Assignment - Fibonacci Generator Tests - Solution

* Creating unit tests in practice

### Testing classes depending on other classes

* The need to test classes that depend on other classes

### A need for mocks

* What issues may be caused by using actual dependencies of a class in unit tests

### Mocks

* What mocks are and how to use them
* How to install the Moq library
* How to install NuGet packages without opening the NuGet package manager

### Controlling the mock behavior

* How to control the behavior of a mock

### The benefits of using mocks

* What the benefits of using mocks are

### Advanced mock setup

* Advanced techniques for controlling the behavior of mocks

### Assertions checking if a method was called

* How to assert that a certain method was called on a mock

### Advanced assertions on method calls

* Advanced techniques for verifying the behavior of mocks

### Clean code in unit tests

* About the importance of clean code in unit tests

### Common setup for tests

* How to define a common setup for multiple tests
* What the CUT object is
* How using the nullable reference types feature may affect the code of our tests

### Tests, Dependency Inversion and Dependency Injection

* The importance of Dependency Inversion and Dependency Injection in the context of unit tests

### Untestable code - no Dependency Inversion

* How to fix the testability issues in code that breaks the Dependency Inversion Principle

### Untestable code - static classes and methods

* How using static classes may make our code untestable

### Other kinds of software tests

* Different types of tests than unit tests (integration, end-to-end, performance, smoke and regression tests)

### Assignment - Unit tests for GuessingGame - Basic scenarios

* How to define unit tests in practice

### Assignment - Unit tests for GuessingGame - Checking the messages

* Verifying if specific methods were executed on mocks in practice

### Assignment - Unit tests for GuessingGame - Approaches for messages validation

* Various approaches to testing the exact messages produced by the code

### Assignment - Unit tests for GuessingGame - Resource files

* What resource files are
* How to define them
* How to read their content

# Clean Code

### The importance of clean code

* Why keeping code clean is so important

### Bad decisions related to code quality

* What mistakes may lead to the fall of a software project
* What the Definition of Done (DoD) is

### Tech debt

* What tech debt is

### Being professional

* How we, the developers, should handle communication with the project management regarding the quality of the code

### What is clean code?

* What clean code is

### The importance of meaningful names

* What the importance of meaningful names is

### Renaming. The Boy Scout rule

* How to rename things, and change bad names into better ones
* What the Boy Scout Rule is

### Dealing with problematic naming

* What to do when we try to name something but a clear and expressive name simply cannot be found

### Expressive names

* How to name things so that our intentions are expressed clearly and precisely

### Long and short names

* How long the names should be

### Principle of the least surprise

* What the principle of the least surprise is

### Bad names - meaningless words

* What meaningless words we should avoid when naming things

### Bad names - overspecific names

* Why using overspecific names is not a good idea.

### Bad names - Hungarian notation

* What Hungarian notation is and why it is better to avoid it

### Bad names - confusing names

* Examples of names that are confusing to the reader
* How we can avoid this

### Bad names - abbreviations

* Why using abbreviations is usually a mistake

### Reasonable abbreviations. Conventional names

* When using abbreviations may be acceptable
* Conventions in naming

### Context

* The importance of the context in which a name functions

### Refactoring case study - naming

* Refactoring with a focus on improving naming

### Good signatures of methods

* How to design methods signatures well

### Number of parameters

* How the number of method’s parameters can affect the cleanliness of the code

### Fewer parameters - splitting the method

* How to reduce the number of method parameters by splitting this method

### Fewer parameters - bundling related parameters

* How to reduce the number of method parameters by bundling multiple parameters together

### Fewer parameters - avoiding boolean parameters

* Why it is often better to avoid boolean parameters

### Small methods

* How big the methods should be

### One method, one job

* How many tasks one method should perform

### One method, one job - refactoring example

* An example of a method that fails to do one thing only
* How such methods could be refactored

### Levels of abstraction

* What levels of abstraction are

### Composing different levels of abstraction

* How to correctly compose operations that are at different levels of abstraction

### Levels of abstraction within methods

* How to achieve methods in which the levels of abstraction are correctly utilized

### Refactoring case study - methods - introduction

* Refactoring methods in practice
* Identifying issues in methods design

### Refactoring case study - methods - signature

* Refactoring of a method’s signature

### Refactoring case study - methods - body

* Refactoring of a method’s body by extracting smaller methods

### Comments

* Why adding comments is usually a bad idea

### The worst comments

* Common examples of using comments in an extremely poor manner
* What the alternative solutions could be

### The reasonable comments

* Scenarios when using comments may be a good idea

### When to make methods static? Private methods

* When making methods static is a good idea (private methods)

### The risk of making public methods static

* Why making public methods static may be a bad idea

### When to make methods static? Public methods

* When making a public method static may be a good idea

### Composition over inheritance

* The differences and similarities between composition and inheritance
* Composition over Inheritance principle

### The issues of inheritance

* What issues inheritance causes

### Replacing inheritance with composition

* How we can refactor the code to use composition rather than inheritance

### Benefits of composition

* Benefits of composition over inheritance

### Assignment - Password Generator Refactoring - Existing code

* The purpose of the Enumerable.Repeat method

### Assignment - Password Generator Refactoring - Fixing naming

* How to improve naming in code

### Assignment - Password Generator Refactoring - Improving design

* How to improve design in code

# Multithreading & asynchrony

### The computer’s processor

* How the computer’s processor works

### Threads and processes

* What a thread is
* What multithreading means
* What a process is
* The difference between a thread and a process

### Concurrency vs parallelism

* The difference between concurrency and parallelism

### Asynchrony

* What asynchrony is
* The difference between asynchrony and multithreading

### A single-threaded program

* How a basic, single-threaded application is processed by the CPU
* How to check the number of cores in CPU
* How to list all threads in the application while debugging

### Starting a new thread. The Thread class

* How to start a new thread using the Thread class
* The difference between foreground and background threads

### Multithreaded app’s code flow

* How the flow of the program changes when we use multiple threads

### The benefits of multithreading and asynchronous programming

* Why we need multithreading and asynchronous programming

### The cost of threads. ThreadPool

* The cost of creating new threads
* How it can be reduced by using the ThreadPool

### Task Parallel Library (TPL)

* What TPL, the Task Parallel Library is

### Task class

* What the Task class is
* How to create and start new Tasks

### Returning a value from a Task

* How to define Tasks that return a value
* The purpose of the Thread.Sleep method

### Waiting for the Task result

* How to wait for the Task to return its result
* What a blocking operation is

### Wait and WaitAll methods

* The purpose of the Wait and WaitAll methods

### Continuations. The ContinueWith method

* How to wait for task completion in a non-blocking way
* What task continuations are
* How we can use them to perform some action after a task is completed

### Chaining continuations. Continuations of multiple tasks

* How to define a chain of continuations, that will all be executed one after the other
* How to schedule a continuation of multiple tasks

### Canceling a Task

* How cancellations of tasks work
* What a cancellation token is and how to use it

### Task lifecycle

* What the lifecycle of a task is
* How we can check the task's current status
* What child tasks are
* The purpose of the Task.FromResult method

### OperationCanceledException

* The typical way of canceling tasks
* The role of the OperationCanceledException

### Exceptions thrown by other threads

* How exceptions are managed when they are thrown on other threads

### Exceptions in tasks

* What happens if an exception is thrown within a task
* What an AggregateException is

### Asynchronous exception handling

* How to handle exceptions thrown from tasks asynchronously
* The purpose of the TaskContinuationOptions enum

### Handling AggregateException

* How to handle exceptions carried within AggregateException

### Multiple continuations for one task

* How to set up multiple continuations for a single task

### Handling task cancellation

* How to handle task cancellation

### The need for synchronization

* Why we need synchronization mechanisms when working with multithreading
* An example of an operation performed by multiple threads that may result in an unexpected program output
* What thread safety is

### Atomic operations

* What asynchrony is
* The difference between asynchrony and multithreading

### Race condition

* What asynchrony is
* The difference between asynchrony and multithreading

### Locks

* What a lock is
* How to use it to prevent two threads from accessing some shared resources at the same time
* What a critical section is

### The need for async/await

* Why we need async/await

### “Await” keyword

* The purpose of the “await” keyword

### Async methods

* How to define async methods
* What types can be returned from async methods
* What types we can use the "await" keyword with

### Asynchrony vs multithreading

* The difference between asynchrony and multithreading (revisited)
* The purpose of the Task.Delay method

### The flow of an asynchronous program

* The flow of an asynchronous program

### Async/await and threads

* The relation between the async/await pattern and the creation of new threads

### Async/await summary

* The summary of the async/await topic

### Async/await practice

* Using async/await in practice

### Exceptions in async methods

* How to deal with exceptions thrown in async methods

### Downsides of multithreading & asynchrony

* The downsides of multithreading and asynchrony

### Using async methods in practice. HttpClient

* How to communicate with APIs using asynchronous methods from the HttpClient class

### Assignment - Quote Finder - Fetching data

* Using async methods in practice

### Assignment - Quote Finder - Smart asynchrony

* How to write optimal code fetching data with multiple requests
* The purpose of the Task.WhenAll method

### Assignment - Quote Finder - Single-threaded processing

* The purpose of the StringSplitOptions and StringComparison enums

### Assignment - Quote Finder - Multi-threaded processing

* Using multithreading in practice

### Assignment - Quote Finder - Refactoring

* Practice how code can be refactored and divided into classes
* The naming convention for async methods