Target: 5 concepts per day minimum

* Multidimensional concepts count as 2

Extreme: 2 concepts per day minimum

Commit message: ‘’

Types: €

1. value types (simple, enum, struct, nullable, tuple)
2. reference types (class, interface, array, delegate?), record types?
3. 6 C# types are user definable: class, enum, struct, interface, delegate, and tuple.
4. Array – don’t need to be declared
5. Nullable type T?

Boxing: €

1. To make value type to reference type, why not use pointers then?

Generics: €

1. Type parameter in a class

.NET Framework €

1. CLR (Common Language Runtime)
   1. CLR acts as an interface between the os and the .NET system
   2. CLR provides services related to automatic garbage collection, exception handling, and resource management.
   3. Code that's executed by the CLR is sometimes referred to as "managed code." "Unmanaged code," is compiled into native machine language that targets a specific platform.
   4. Language interoperability is a key feature of .NET. IL (aka MSIL or CIL) code produced by the csc conforms to the Common Type Specification (CTS), it can interact with code that was generated from the .NET versions of F#, VB, C++, and more.
   5. The CLR performs JIT compilation, converting IL code to native machine code.
   6. What is managed code?
      1. Managed code is just code whose execution is managed by a runtime, in this case the runtime is CLR.
      2. CLR is in charge of taking the managed code, compiling it into machine code and then executing it.
      3. Contrast to this, for unmanaged code (C++ and C), the everything from memory management to security considerations are a burden of the programmer.
      4. C# allows you to use unmanaged constructs such as pointers directly in code by utilizing what is known as “unsafe context” which designates a piece of code for which the execution is not managed by the CLR.
   7. CLS (Common Language specification)
   8. CTS (Common type system)
2. BCL/FCL (Base class library or framework class library)

Managed vs Native: €

1. Managed code is code that is executed by .NET runtime, which provides addition features like memory management, type safety, and security.
   1. Easier to write and debug
   2. More portable, assuming the other system has .NET runtime installed
2. Native code is directly executed by the computer’s processor and does not have features like type safety or memory management.
   1. Faster and more efficient
   2. Less portable since its specific to the processor and OS on which it was compiled.
   3. Native code is a type of machine code but not all machine code is native code

What is a runtime? €

1. A runtime is a program that is responsible for executing code at the execution phase of a program. Runtimes provide the environment in which code is executed, and include additional features to support execution of the code.
2. Examples: OS runtimes (windows os), Language runtimes (.NET), browser runtimes (JS)

How C# code is compiled: €

1. Write your code, compile it using the csc.exe (included in the .NET SDK), this creates an assembly (with MSIL and metadata) in the form of a dll or exe file, depending on the type of project you create.
   1. MSIL (binary represented) is a low-level, platform-agnostic (aka cross platform) code generated by C# compiler. MSIL is not executed directly by the computer’s hardware, but rather by the .NET runtime, which is a VM that runs on top of your OS.
   2. When you compile C# code, the csc translate it into MSIL, stored in the assembly in the form of dll or exe.
   3. When you run your program, the .NET runtime loads the assembly and executes the MSIL code, which is then converted to native machine code by the JIT compiler.
   4. Any machine that has the .NET runtime installed can now run the code.
   5. MSIL and native code are both binary but different types of binary. MSIL are run using the .NET runtime regardless of any OS but native code is specific to particular os and cannot be executed without being recompiled.
2. If we were to include other files in the created assembly, like images or data files, just add them to your project and they will be automatically included in the assembly when its built.
3. Use ‘AssemblyInfo.cs’ file to specify metadata about your assembly, like version number etc.
4. When you run your program, the .NET runtime loads the assembly and executes the MSIL code, which is converted to native machine code by the Just-In-Time (JIT) compiler.
5. Summary: csc, .NET runtime to run MSIL, JIT, computer processor.

Build vs compile: €

1. Bulid refers to the process of creating a software program or component from source code. This involves compiling source code, linking it with other libraries or modules, and creating an exe or library file.
2. Compile refers to the process of translating source code into machine code or other intermediate representation that can be executed by a computer processor.

Library vs package: €

1. A library is a collection of code that can be used by other programs. Libraries are typically compiled binaries that can be linked with other programs at compile time or runtime.
2. A package is a collection of code that are bundled together and distributed as a single unit. Packages are typically used to distribute libraries and modules, as well as their dependencies in a standardized format.
3. Package is a collection of modules and libraries are collection of packages.
   1. Library -> package -> modules

What is a code contract in C#: €

1. A way to express design-by-contract style programming, where you can specify preconditions, postconditions, and invariants for a method or class.
   1. Helpful for debugging
   2. Invariants are conditions that must always be true for a class. For instance the sum of the values of all elements in a particular data structure should always be positive.
2. Data contracts: specifies the format and structure of data that is transmitted between two parties, such as between client and server.
3. Service contracts: Specifies the operations that a service can perform and the data that is can exchange with clients.
4. Message contracts: Specifies the structure of messages between a client and a service.

Primitive and non-primitive types: €

1. Examples of primitive: int, long, short, uint, ulong, ushort, decimal, etc
2. Examples of non-primitive: arrays, classes, enums, interfaces, strings, structs
3. Non-primitive types can be constructed from primitive types

What are attributes in C#: €

1. An attribute in C# is a piece of metadata which is added to a program element (ie class, method, or property) to provide additional information about that element.
2. declared using `[AttributeName]` syntax, applied to a program element by placing them immediately before the element they are intended to modify
3. Can also specify arguments for an attribute
4. To define custom attribute:
   1. we have to create a class and inherit from System.Attribute
   2. [AttributeUsage(AttributeTargets.Method)] this is for Method, we can have this for class, property and more
   3. Can also define optional arguments for your custom attribute by using the System.ComponentModel.DefaultValueAttribute class

Interpreter vs compiler: €

1. Interpreted languages:
   1. Source code -> interpreter (one line at a time) -> executable code (without IL) -> back to source code (repeat til end)
   2. Portable, we are not sending machine code but rather source code
   3. Slower and needs to be interpreted every time you run, public code (give source code to others)
   4. Errors are displayed for each single instructions
   5. Examples: Python
   6. Interpreter can be thought of as a VM that executes code directly without having to compile it to machine code then execute it via the computer processor.
2. Compiled languages:
   1. Convert source code to machine code and only run the exe file
   2. Not cross-platform, compilation time (from source code to machine code)
   3. Private code (since we already have the executable file), faster, optimized specific cpu
   4. Errors are displayed after entire program is compiled
   5. Examples: C, C++
3. Hybrid language:
   1. Source code -> compiler -> bytecode (aka IL) -> interpreter
   2. Examples: C#

What are DLL files?

1. exe files run in their own memory or own address space
2. dll files run inside some other memory space, it cannot run on its own
   1. It needs a host to run it
3. dll are for reusability