C# and Object-Oriented Programming

Eng. Wael Hosny Radwan

Course Overview

- Day 1: C# Language Fundamentals
- Day 2: OOP Fundamentals
- Day 3: Inheritance and Polymorphism
- Day 4: Collections and Generics
- Day 5: Exception Handling and File I/O
- Day 6: Delegates and Events
- Day 7: Advanced Topics

Day 1

C# Language Fundamentals

Agenda for Day 1

- Introduction to C# and .NET
- Data Types: Value vs. Reference
- Variables and Constants
- Operators
- Control Flow Statements (If/Else, Switch, Loops)
- Methods: Definition and Calling
- Arrays

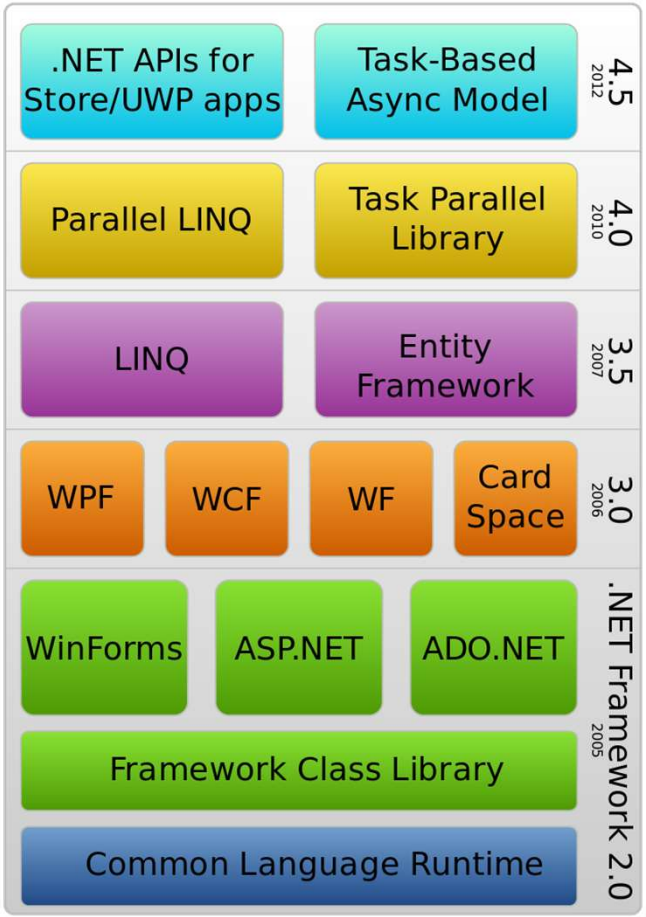
What is C#?

- Modern, Object-Oriented, Type-Safe Language
- Developed by Microsoft
- Part of the .NET Ecosystem
- Syntax: C++, Java-like
- **Key Features**: Strong Typing, Automatic Garbage Collection, LINQ, Async/Await

Understanding .NET

- **Platform**: Free, Cross-Platform, Open-Source
- **CLR (Common Language Runtime)**: Execution Engine (GC, Exception Handling)
- **CTS (Common Type System)**: Type Declaration & Management
- **CLS (Common Language Specification)**: Interoperability Rules
- **FCL (Framework Class Library)**: Reusable Code Library

.NET Framework



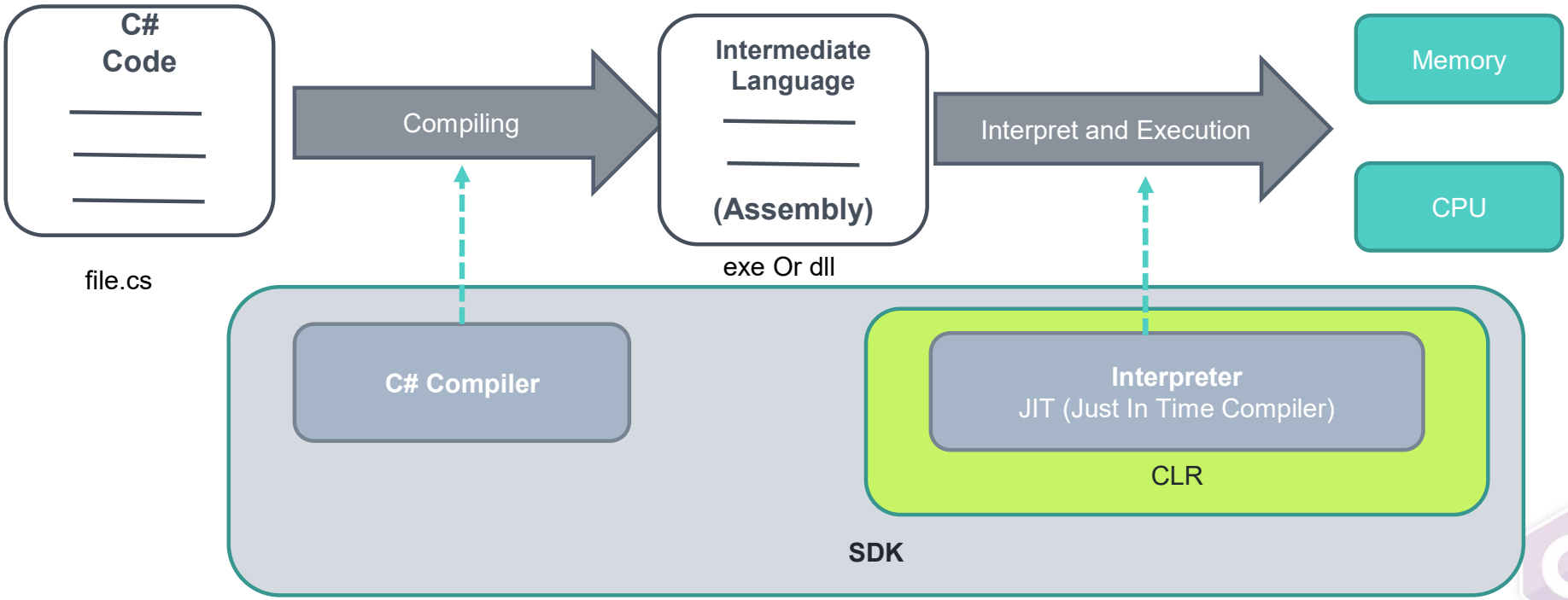
.NET Core (.NET)

- Rewriting .NET framework To make it Platform Independent and open source produce **.NET** (previously named .NET Core)
  - CLR ➔CoreCLR
  - BCL ➔CoreFX
- .NET Core 1 (2016) , 2(2017),3(2019)
- .NET Core 4 skipped (confusing with .NET Framework 4.x)
- .NET 5 (2020)
- .NET 6 (2021) **(LTS)**
- .NET 7.0 (2022)
- .NET 8.0 (2024) **(LTS)**
- Latest Version .NET 9.0 (2025)

C# Compilation Process

- **Source Code (.cs)** -> C# Compiler (Roslyn) -> **MSIL/CIL**
- **MSIL + Metadata** -> **Assembly (.exe or .dll)**
- **Assembly** -> CLR (JIT Compiler) -> **Native Machine Code**

C# Compilation Process



"Hello, World!" in C#

- Code Example:

```
using System;
namespace ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

- Explanation: **`using`**, **`class`**, **`Main`** method, **`Console.WriteLine`**

"Hello, World!" in C# (top level statement)

- Namespace , **Program** class, **Main** method
  - Generated during compilation
  - Using statements in the beginning of the file
  - Only one file like this

```
Console.WriteLine("Hello, World!");
```

Global using (.NET 6)

- ***global using*** statement
- Show all files➔obj➔net6.0 (or 9.0) ➔ ConsoleApp1.GlobalUsings.g.cs

```
// <auto-generated/>
global using global::System;
global using global::System.Collections.Generic;
global using global::System.IO;
global using global::System.Linq;
global using global::System.Net.Http;
global using global::System.Threading;
global using global::System.Threading.Tasks;
```

Data Types: Overview

- **Purpose**: Classify data, memory allocation, define operations
- **Two Main Categories**:
  - Value Types
  - Reference Types

Memory Management

- ***CLR*** is responsible for memory management it divides memory into two regions (division based on how to treat Memory both are RAM)
  - Stack Memory
  - Heap Memory
- By dividing memory into these two regions, the .NET Framework is able to efficiently manage memory usage and avoid common memory-related issues like stack overflows and heap fragmentation

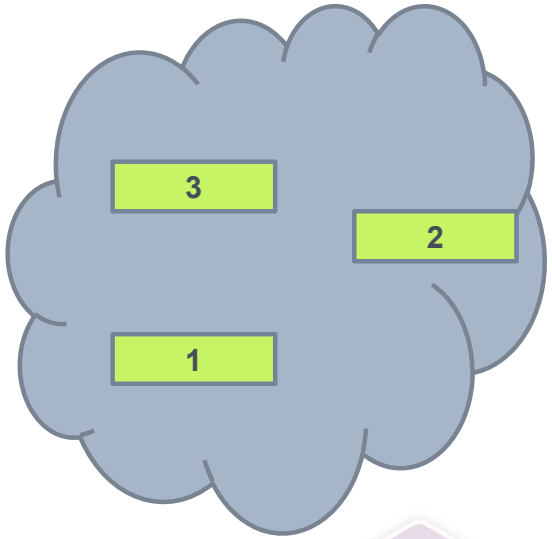
Stack Memory

- Stack memory is a special region of memory Used to store ***Small*** variables and ***temporary*** variables created by Methods (*local variables*)
- Data Stored ***Sequential*** (On top of each other)
- ***Limited*** and predetermined at compile-time in size
- Fast Access
- Variables in It can't be resized

Stack

Heap Memory

- Heap memory is a region of memory used for ***dynamic*** memory allocation and ***big*** variables and ***global*** variables and variables with long life
- Data Stored ***Scattered*** (collection of memory blocks)
- ***No Size Limits(but system constraint)***
- Not Fast access like Stack
- Variables can be resized
- memory is allocated during the execution of instructions written by programmers (*runtime*)
- Managed By ***Garbage Collector***



Heap

Value Types

- **Storage**: Store data directly in memory
- **Assignment**: Copies the value
- **Examples**: `int`, `float`, `double`, `char`, `bool`, `struct`, `enum`
- **Demo**:

```
int a = 5;
int b = a;
b = 10; // a is still 5
```

Reference Types

- **Storage**: Store a reference (memory address) to data on the heap
- **Assignment**: Copies the reference
- **Examples**: `string`, `class`, `interface`, `delegate`, `object`
- **Demo**:

```
string s1 = "hello";
string s2 = s1;
s2 = "world"; // s1 is still "hello"
```

Common Data Types

- **Integers**: `byte`, `short`, `int`, `long` (signed/unsigned variants)
- **Floating-Point**: `float`, `double`, `decimal` (precision differences)
- **Boolean**: `bool` (`true`/`false`)
- **Character**: `char` (single Unicode character)
- **String**: `string` (sequence of characters, immutable)
- `var` **keyword**: Implicitly typed local variables

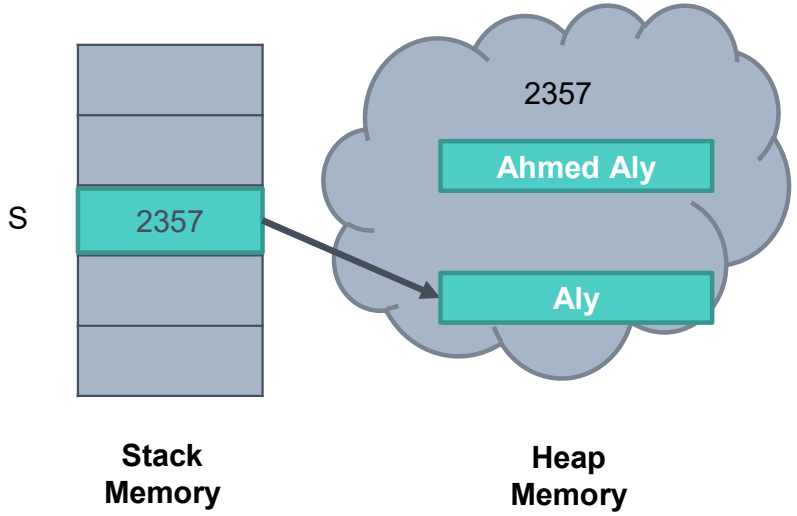
Variables and Constants

- **Variables**:
  - Named storage for data
  - Declaration: `dataType` *`variableName`*;
  - Initialization: `dataType` *`variableName`* = `value`;
  - Naming: `camelCase`
- **Constants**:
  - Values fixed at compile time
  - `const` keyword
  - Must be initialized at declaration
  - Naming: `PascalCase` or `ALL_CAPS`

String (reference type)

- String is a reference type
- String is immutable variable

```
string S = "Ahmed Aly";
Console.WriteLine(s);
S = "Aly";
```



String

- Declaring and initialization of string

```
// Declare without initializing.
string message1;

// Declare and Initialize to null.
string message2 = null;

// Initialize as an empty string.
// Use the Empty constant instead of the literal "".
string message3 = System.String.Empty;

// Initialize with a regular string literal.
string oldPath = "c:\\Program Files\\Microsoft Visual Studio 8.0";
```

String

- Methods
  - Static (called through string keyword)
    - `Format`
    - `Concat` (Full Name Example)
    - `Compare` two versions

String

- Instance method (called through variable name)
  - `StartWith`
  - `EndWith`
  - `ToLower`
  - `ToUpper`
  - `Trim`
  - `Replace`
  - `TocharArray()`
  - `PadLeft() PadRight()`

Input and Output Methods (console application)

- Output Methods
  - WriteLine() , Write()
    - Printing Literal string

```
Console.WriteLine("Hello World!");
```

    - Printing value of variable in Literal string

```
int x,y;
x = 100;
y = 200;
Console.WriteLine("value of x={0} \t value of y={1}", x,y);
```

```
string s = "Ahmed Aly";
Console.WriteLine(s);
```

Input and Output Methods (console application)

- Output Methods
  - Special Characters

Symbol	Meaning (prints)
\t	Tab spacing
\n	New line
\\	backslash
\’	Single quotes
\”	Double quotation
\r	Carriage return from beginning of the line

Input and Output Methods (console application)

- Input Methods
  - **ReadLine()**
    - Reads string from user input
    - Needs user to press Enter Button to finish the process

```
Console.WriteLine("Enter Your Name");
string name=Console.ReadLine();
```

  - **Read()**
    - Reads one character from user Input and return its Unicode number
    - If multiple character reads the first one

```
Console.WriteLine("Enter character");
int code=Console.Read();
```

Input and Output Methods (console application)

- Input Methods
  - **`ReadKey()`**
    - Reads the keyboard button pressed by the user

Console Helper Methods and Properties

- Methods
  - `Console.ResetColor`
  - `Console.Clear()`
  - `SetCursorPosition`
- Properties
  - `Console.BackgroundColor`
  - `Console.ForegroundColor`
  - `Console.WindowHeight`
  - `Console.WindowWidth`

Conversion between data types

- Implicit Casting
  - For compatible data types (numeric datatypes)
  - Automatic conversion from *smaller* to *bigger*


```
int x = 100;
float f;
f = x;
```

Conversion between data types

- Explicit casting
  - For compatible data types (numeric datatypes)
  - It may cause data lost
  - From bigger to smaller


```
float f=3.15f;
int x ;
x = (int) f;
```

Convert to type

Conversion between data types

- Conversion without casting
  - For incompatible data types
  - using methods (from string to numbers)
    - `Parse ( )`

```
Console.WriteLine("Enter Number");
string s = Console.ReadLine();
int x;
x=int.Parse(s);
```

    - `TryParse ()`

```
Console.WriteLine("Enter Number");
string s = Console.ReadLine();
int x;
int.TryParse(s, out x);
```

Conversion between data types

- Conversion without casting
  - using methods
    - `ToString ( )`

```
int x = 10;
string s = x.ToString();
```

  - ***Convert*** class Methods

```
int x = 10;
string s = Convert.ToString(x);
x= Convert.ToInt32(s);
```

Operators

- **Arithmetic**: `+`, `-`, `*`, `/`, `%`
- **Assignment**: `=`, `+=`, `-=`, etc.
- **Comparison**: `==`, `!=`, `<`, `>`, `<=`, `>=`
- **Logical**: `&&` (AND), `||` (OR), `!` (NOT)
- **Increment/Decrement**: `++`, `--` (prefix/postfix)
- **String Concatenation**: `+`
- **Operator Precedence**:

Control Flow: Overview

- **Purpose**: Dictate execution order
- **Categories**:
  - Conditional Statements
  - Looping Statements

Conditional Statements: `if/else`

- `if` **statement**: Execute code if condition is true
- `else if`: Check additional conditions
- `else`: Execute if no `if`/`else if` conditions are met
- Code Example:

```
if (score >= 90) { /* A */ }
else if (score >= 80) { /* B */ }
else { /* C */ }
```

Conditional Statements: Ternary operator ? :

```
condition ? consequent : alternative
```

```
int x = 10;
string v;
if (x==10)
{
    v = "10";
}
else
{
    v = "Other Number";
}
```

```
v = (x == 10) ? "10" : "Other Number";
```

Conditional Statements: Switch

- `switch` **statement**: Multi-way branching based on a single value
- `case` **labels**: Match specific values
- `break` **keyword**: Exit `switch` block
- `default`: Executed if no `case` matches
- Code Example:

```
switch (day)
{
  case "Mon":
        ...
        break;
  default:
        ...
        break;
}
```

Looping Statements: For Loop

- `for` **loop**:** Fixed number of iterations
- **Syntax**: `for` (*`init`*; *`condition`*; *`iterator`*)
- **Code Example**:

```
for (int i = 0; i < 5; i++)
{
  Console.WriteLine(i);
}
```

Looping Statements: While & Do-While

- `while` **loop**: Executes as long as condition is true (pre-test)
- **Code Example**:

```
while (count > 0) { count--; }
```

- `do-while` **loop**: Executes at least once, then checks condition (post-test)
- **Code Example**:

```
do { Console.WriteLine("Once"); } while (false);
```

Control Flow: break, continue

- **break** statement :
  - The break statement will terminate looping and continue executing the code that follows after the loop (if any).
- **continue** statement:
  - The continue statement will terminate the current loop and continue with the next loop.

```
for (int i = 2; i ≥ 0; i--)
{
  …
  if (x == 10)
   break;
  Console.WriteLine("{0}", i);
 }
```

```
for (int i = 2; i ≥ 0; i--)
 {
      …
   if (x == 10)
     continue;
   Console.WriteLine("{0}", i);

 }
```

Assignment

- Install Visual Studio (Community Edition)
  - https://learn.microsoft.com/en-us/visualstudio/install/create-an-offline-installation-of-visual-studio?view=vs-2022
    - Download vs bootstrapper (vs_community.exe)
    - For .NET web and .NET desktop development for only one language

```
vs_community.exe --layout G:\VSCommunity\localVSlayout --add Microsoft.VisualStudio.Workload.ManagedDesktop --add Microsoft.VisualStudio.Workload.NetWeb --add Microsoft.VisualStudio.Workload.NativeDesktop --includeRecommended --includeOptional --lang en-US
```

- Get sum , average for 2 numbers entered by the user

Assignment

- VSCode
- .NET SDK (https://dotnet.microsoft.com/en-us/download)
- Install VSCode
  - Extension
    - C# Extension
    - C# Dev Kit extension (optional)
    - vscode-solution-explorer
  - Ctrl+Shift+p → .NET Generate Assets for build and debug
  - “launch.json”→ line 17 → `"console": "externalTerminal"`

Assignment

- Create simple menu and get user selection from it
  - Option 1
    - To calculate sum
  - Option 2
    - get max
  - Option 3
    - get min
- Simple calculator
  - Get 2 number from user and get operator (* or / or + or -)
  - Calculate the result
- Magic Box

Assignment

- ***Magic Box***
  - Row - - Column - -
  - Row++

6	1	8
7	5	3
2	9	4