C#.NET

Introduction to .NET

What is C#?

C# (pronounced “C-sharp”) is a modern, object-oriented programming language developed by Microsoft in the early 2000s as part of its .NET initiative.

It is designed to be simple, versatile, and robust, allowing developers to build a wide range of ap

C# drew inspiration from C++ and JAVA, combining their best features while addressing some of their limitations.

C# Origins and Evolution

- Anders Hejlsberg is known as the founder of C# language.
- The current version of C# is C# 13.0 released in 2024 in .NET 9.0

C# Version History

Version	Features	Year of release
C# 1.0	Basic Features	2002
C# 2.0	Generics, Partial types, Anonymous methods, Nullable types, Static classes	2005
C# 3.0	Var, LINQ, Lambda expression, Auto-implemented properties, Anonymous types, Extension methods	2007
C# 4.0	Dynamic binding, Named and Optional arguments	2010
C# 5.0	Asynchronous methods, Caller info attributes	2012
C# 6.0	Auto-property initializers, Null-propagating operator, Exception filters, Using static members, ...	2015

Key Features of C#

1. **Object-Oriented**
2. **Type-Safe**
3. **Platform Independence with .NET**
4. **Rich Standard Library**
5. **Garbage Collection**
6. **Interoperability**
7. **Versatility**
    - Desktop Applications
    - Web Applications
    - Game Development
    - Cloud Services Mobile Apps
8. **Modern Features**

What is Visual Studio?

IDE developed by Microsoft

It is a powerful tool used by developers to create, debug, and deploy applications for various platforms, including Windows, web, cloud, mobile, and more.

Visual Studio supports multiple programming languages.

Key Features of Visual Studio

- Code Editor
- Debugging Tools
- Project templates
- Version Control Integration
- Testing Tools
- Multi-Language support
- Cloud and Azure Integration
- Extension Support
- Cross-Platform Development
- Built-In Terminal and Git Tools

Editions of Visual Studio

COMMUNITY EDITION

PROFESSIONAL EDITION

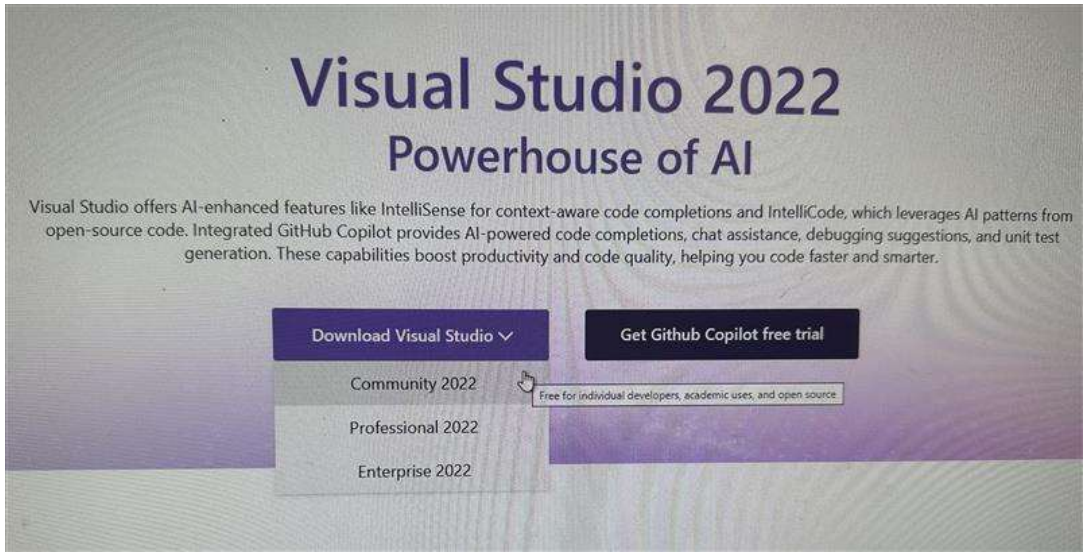
ENTERPRISE EDITION

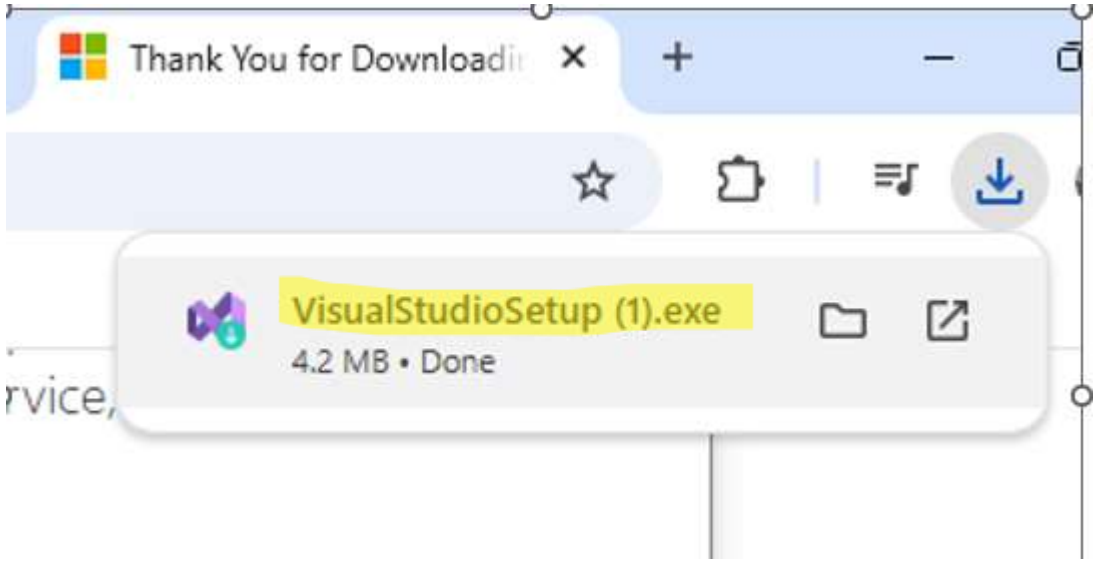
- Latest version 2022

Let's Install Visual Studio

Visual studio downloading and installing

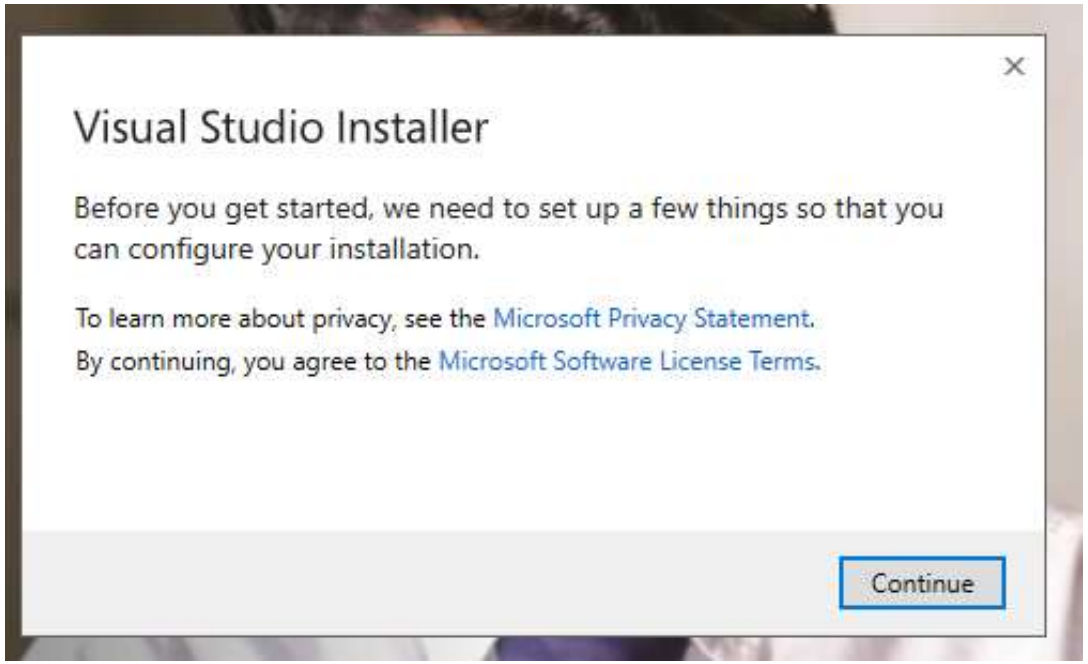
Select the edition you want to download and run the .exe file to install.

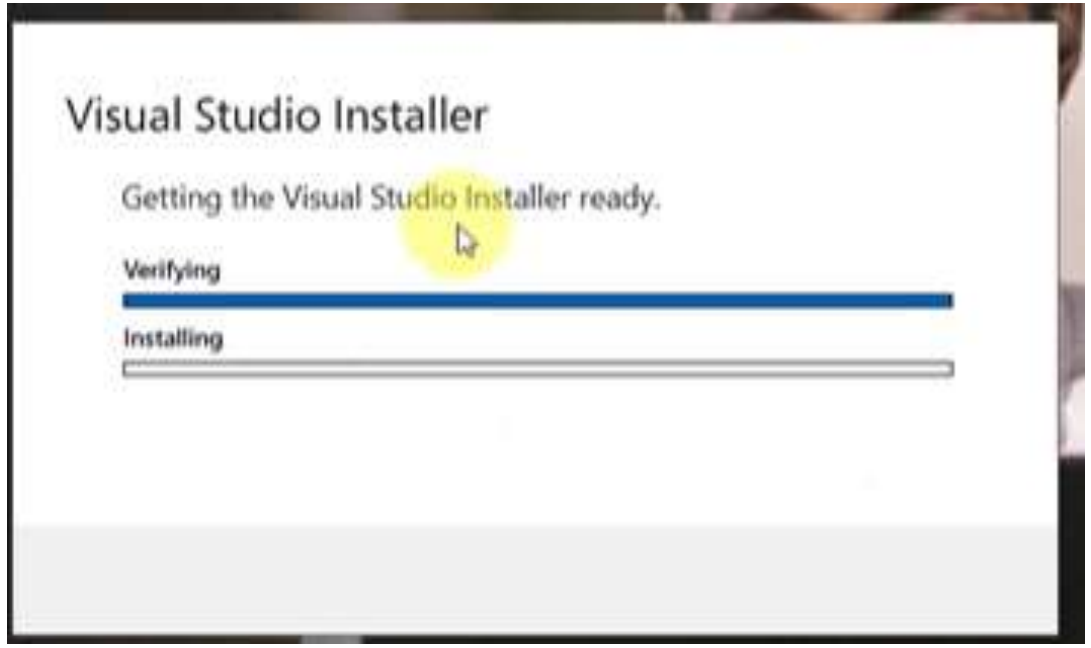




Visual studio downloading and installing

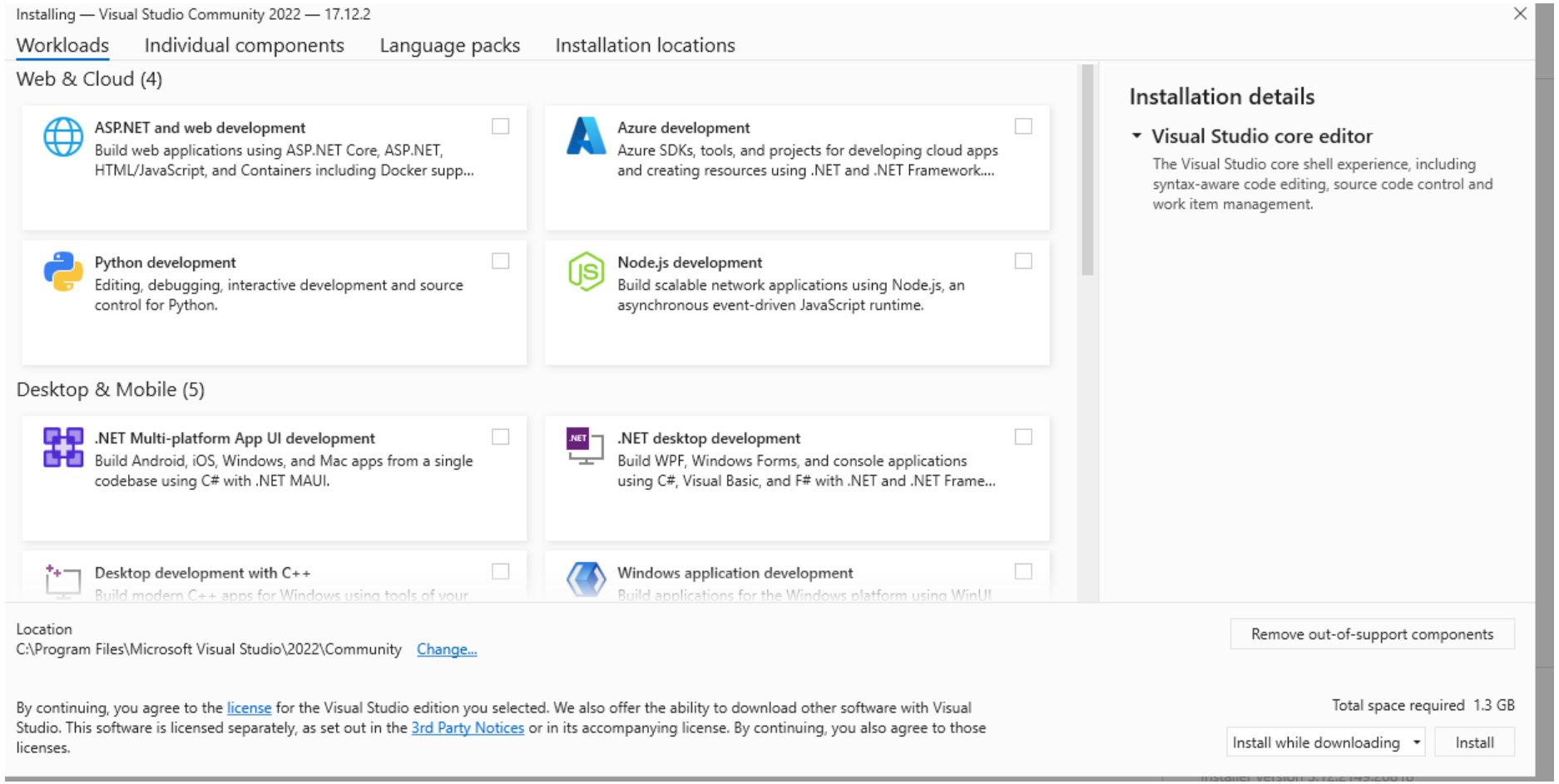
Click continue and install





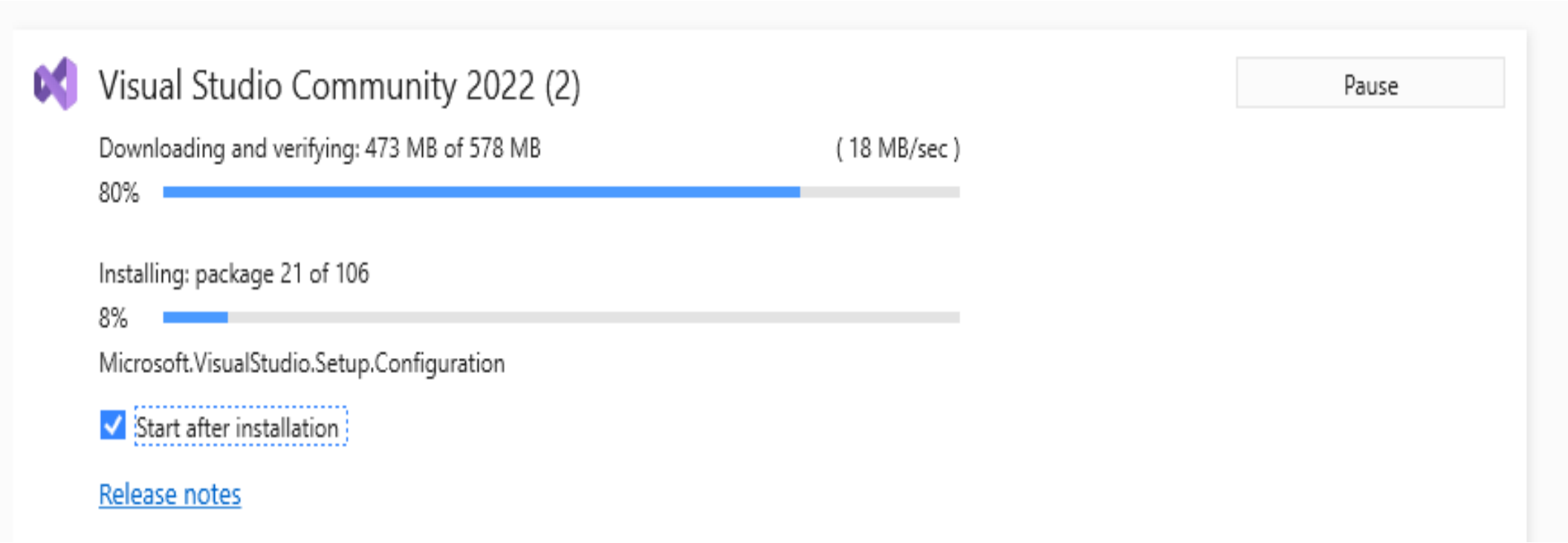
Visual studio downloading and installing

Select components you want to install



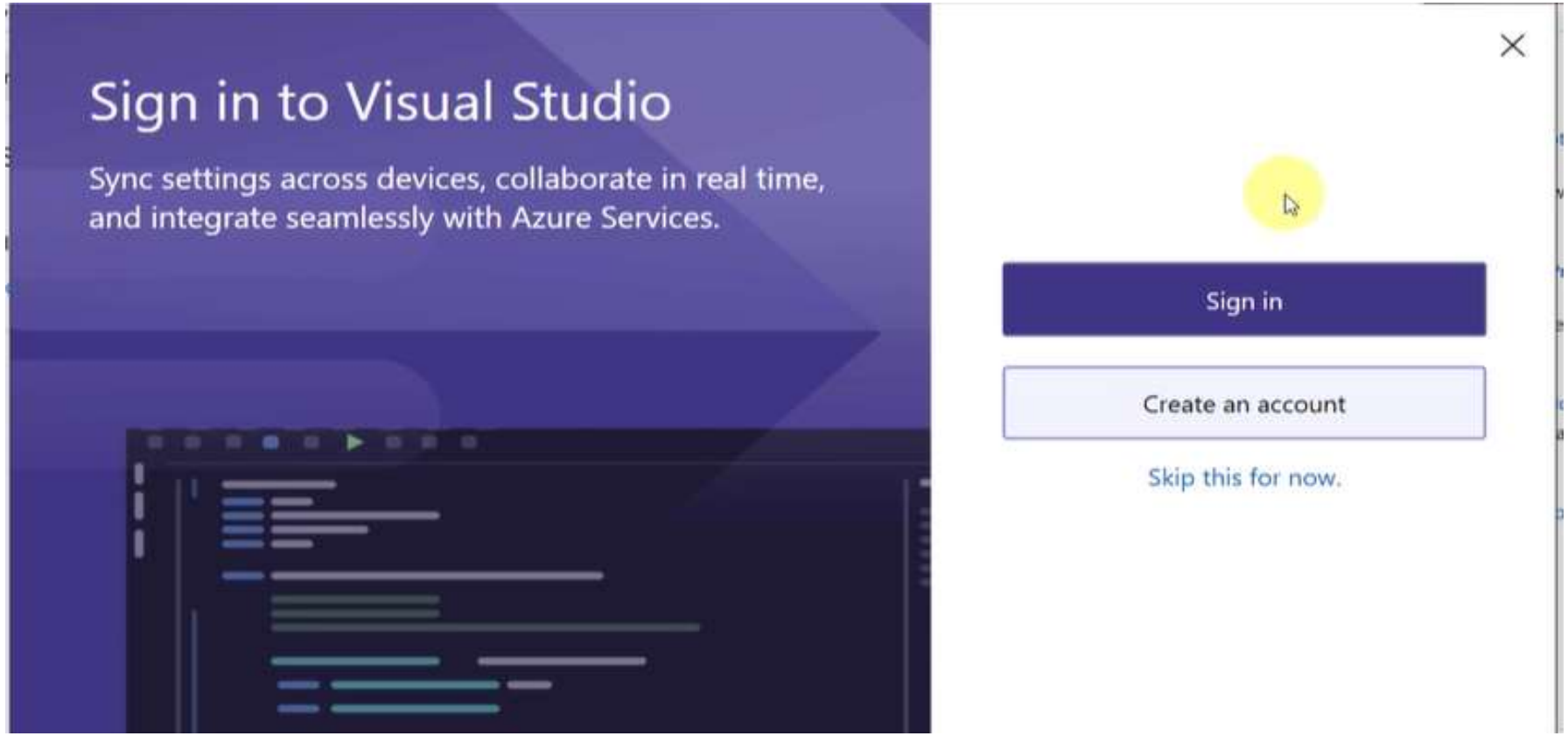
Visual studio downloading and installing

Installation in progress



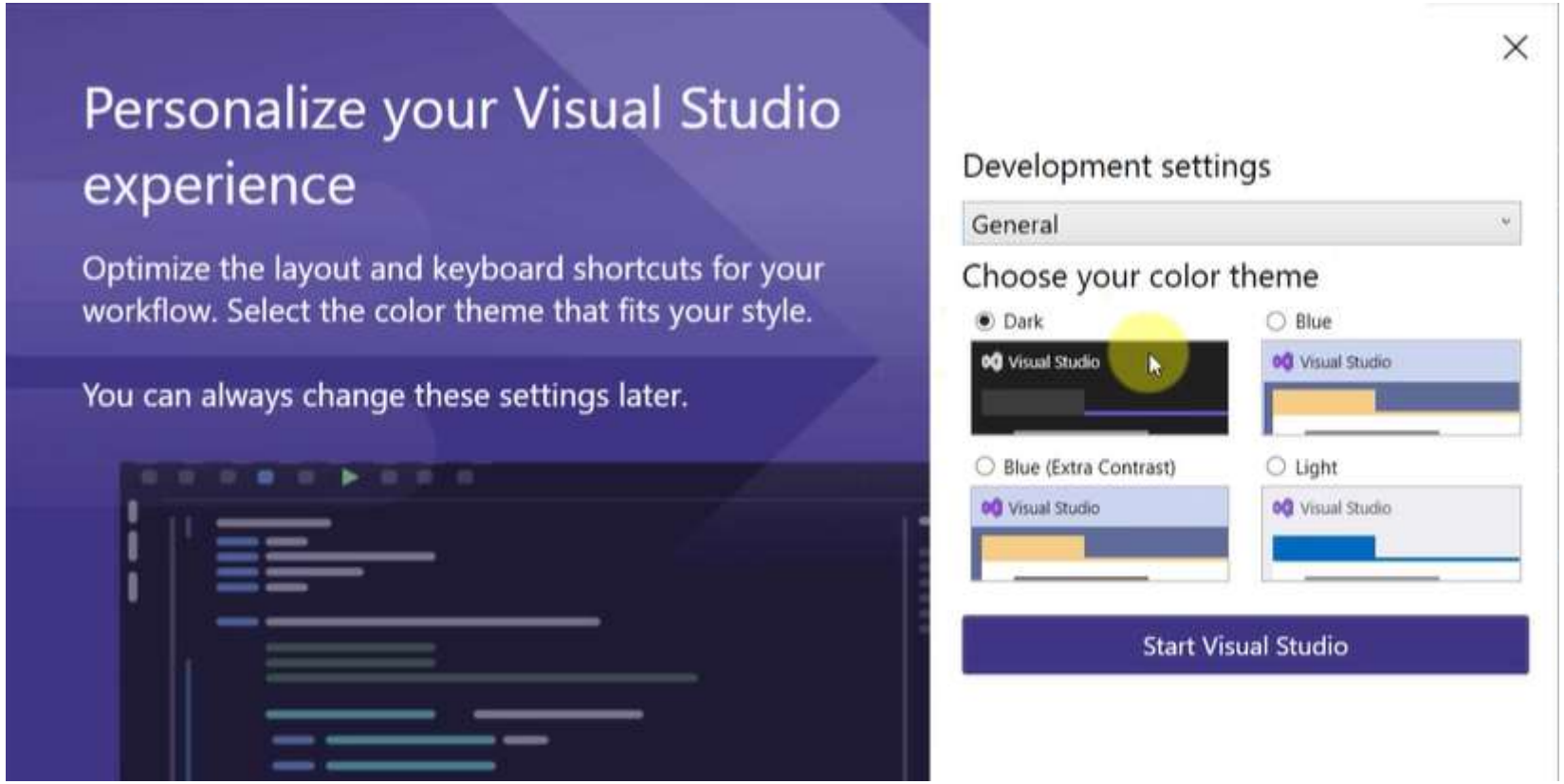
Visual studio downloading and installing

Add account or Click ‘Skip for now’



Visual studio downloading and installing

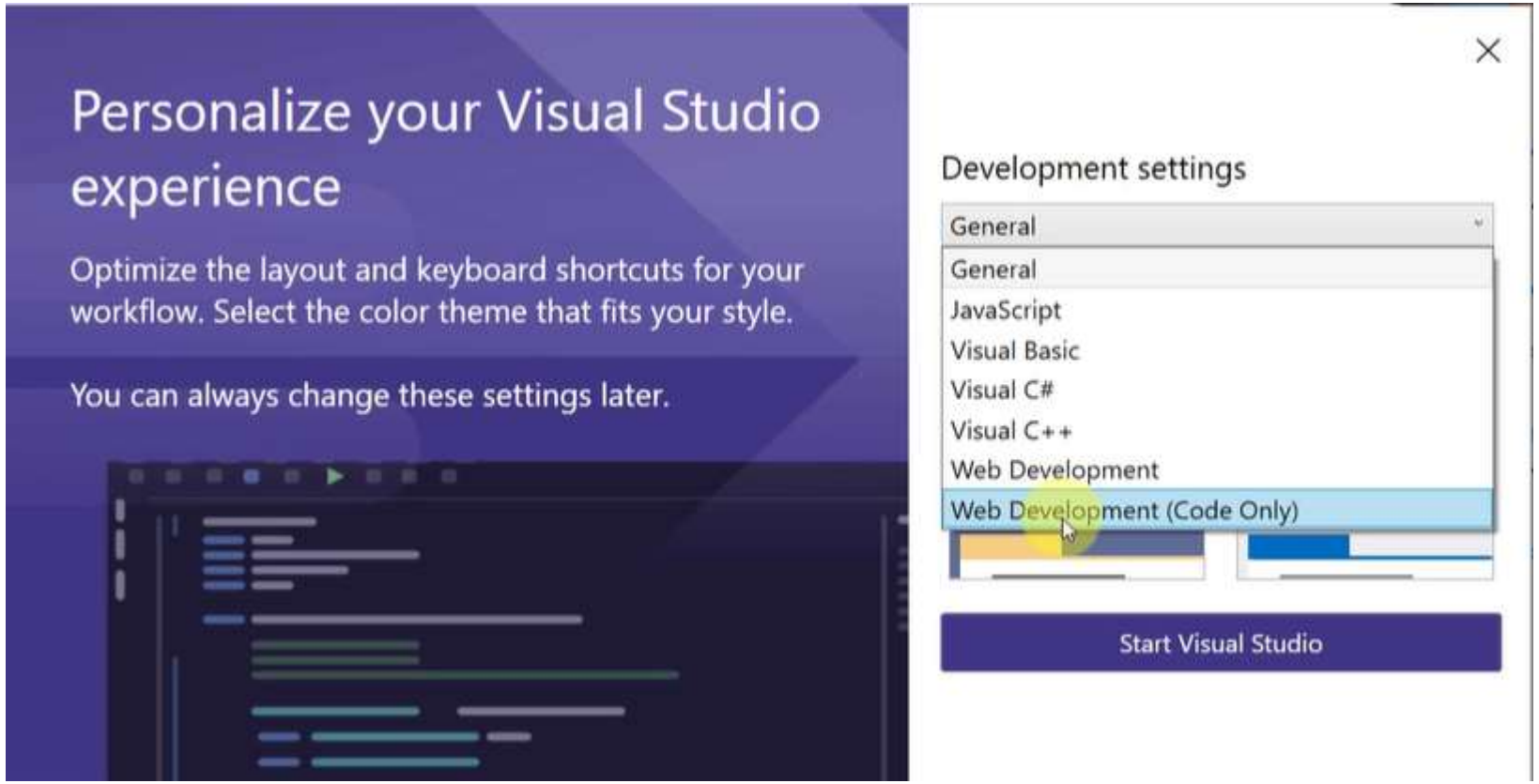
Select development settings and theme



Visual studio downloading and installing

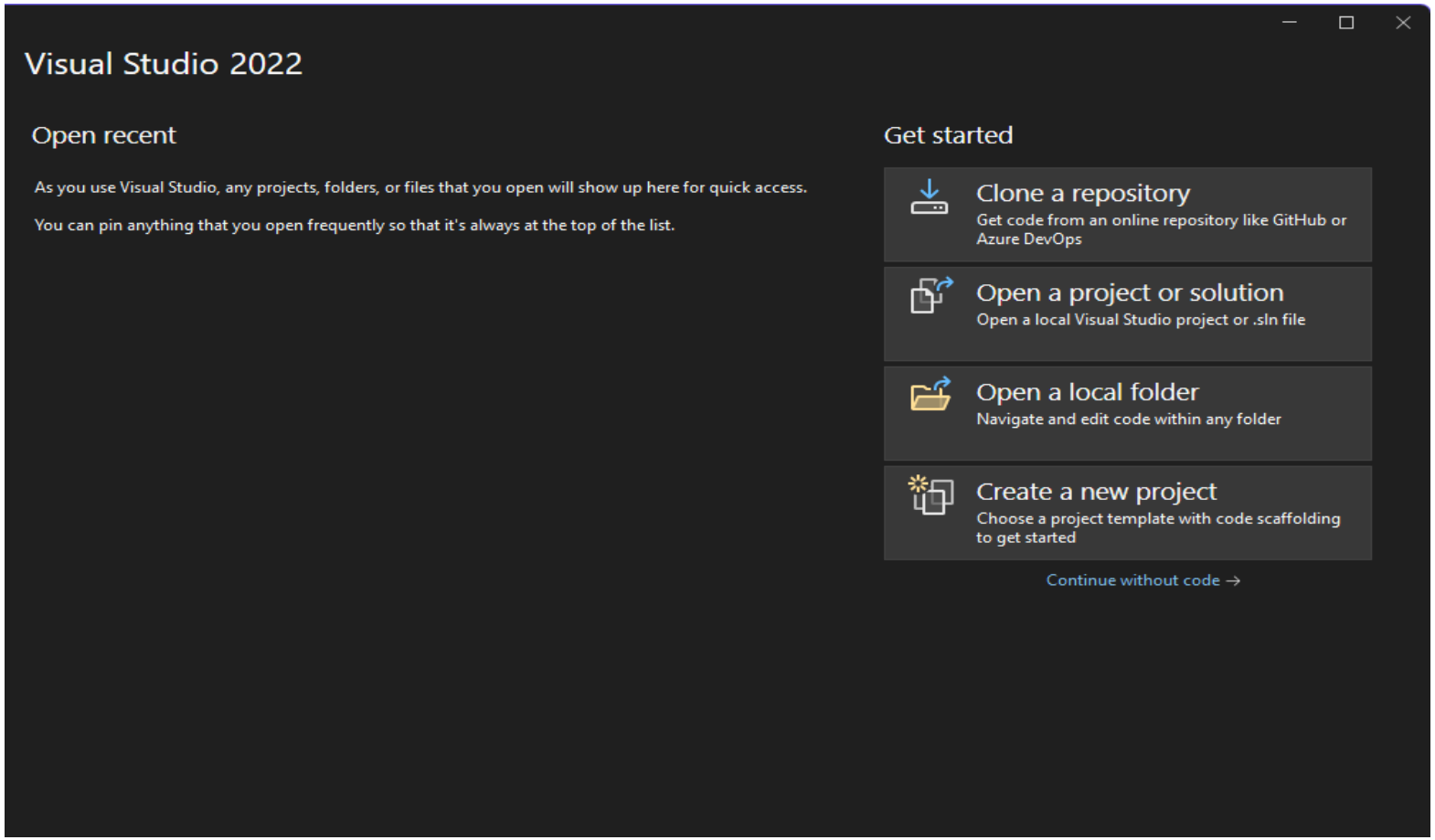
Select development settings and theme

Click on ‘Start Visual Studio’



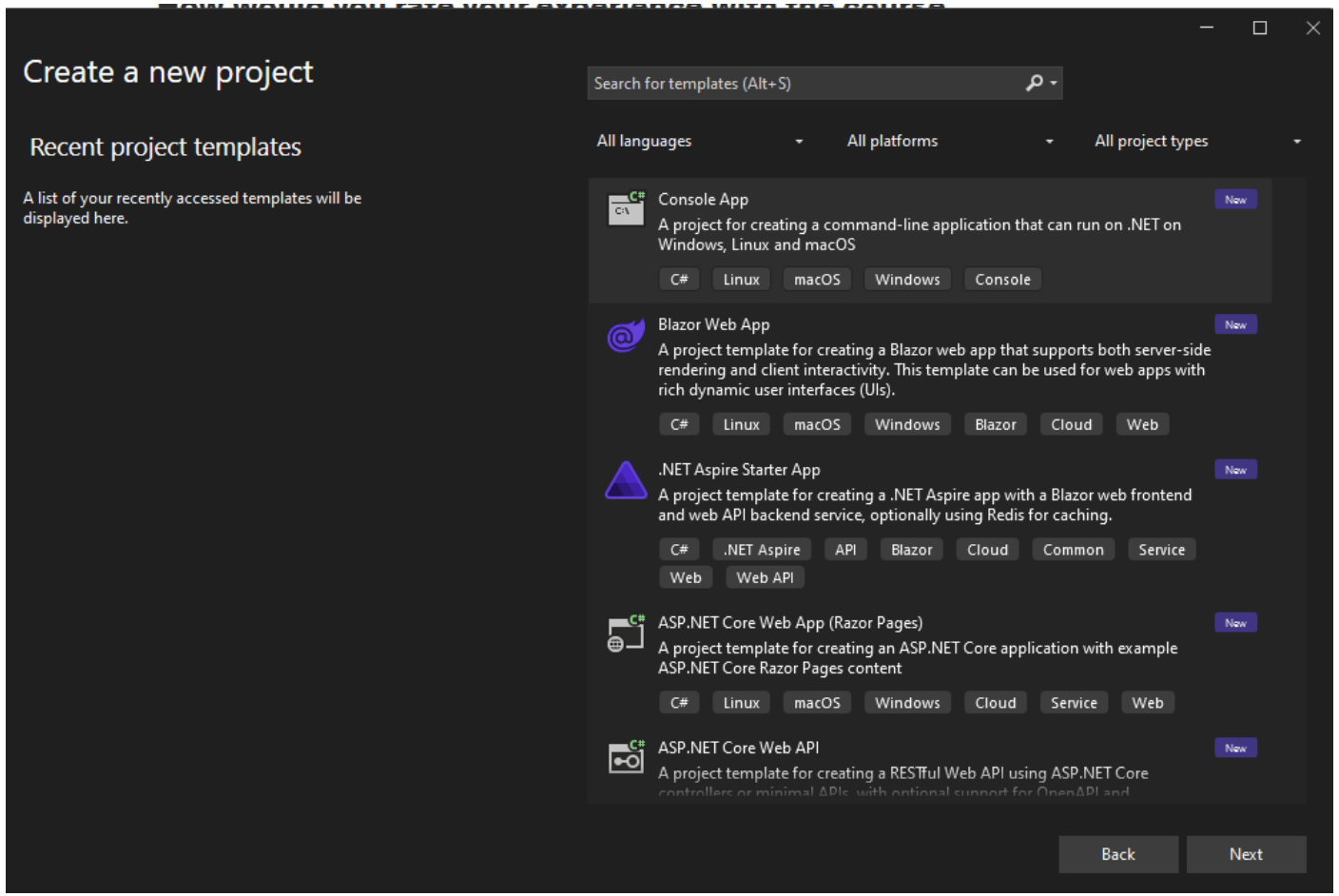
Create new project

Click on ‘Create a new project’



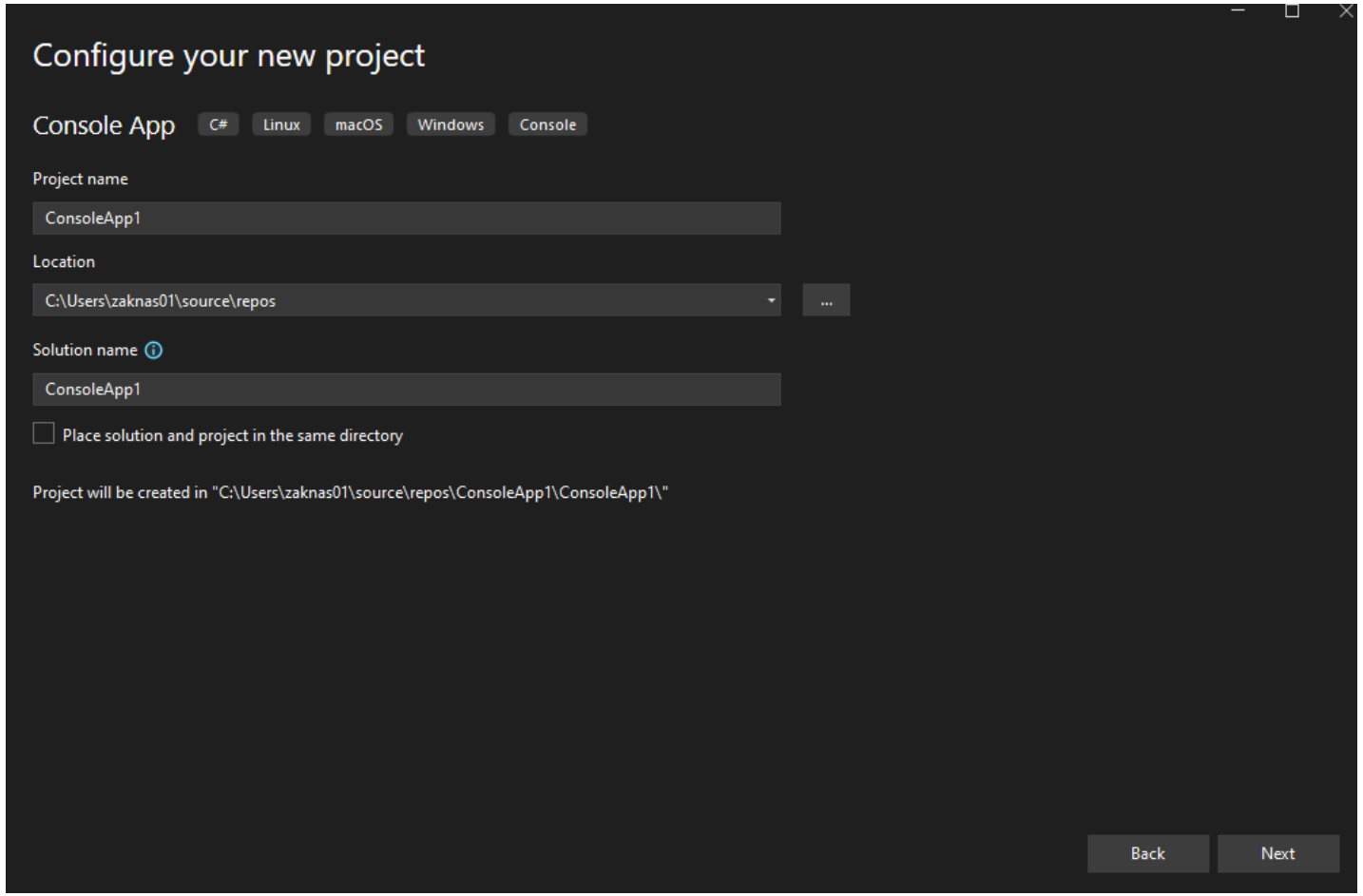
Create new project

Select “Console App” template and click next



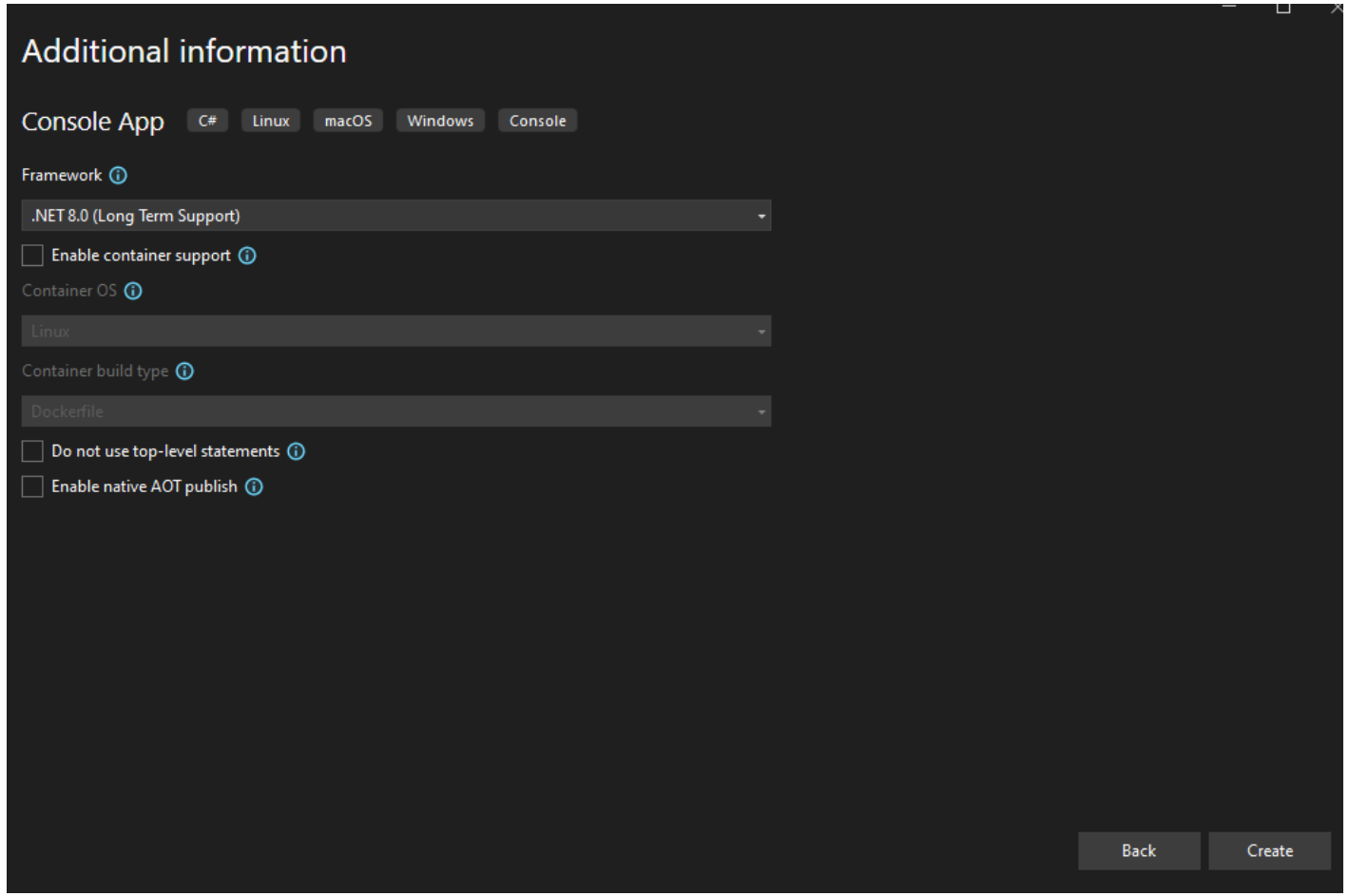
Create new project

Change Project Name and if required Location and Solution Name . Click Next



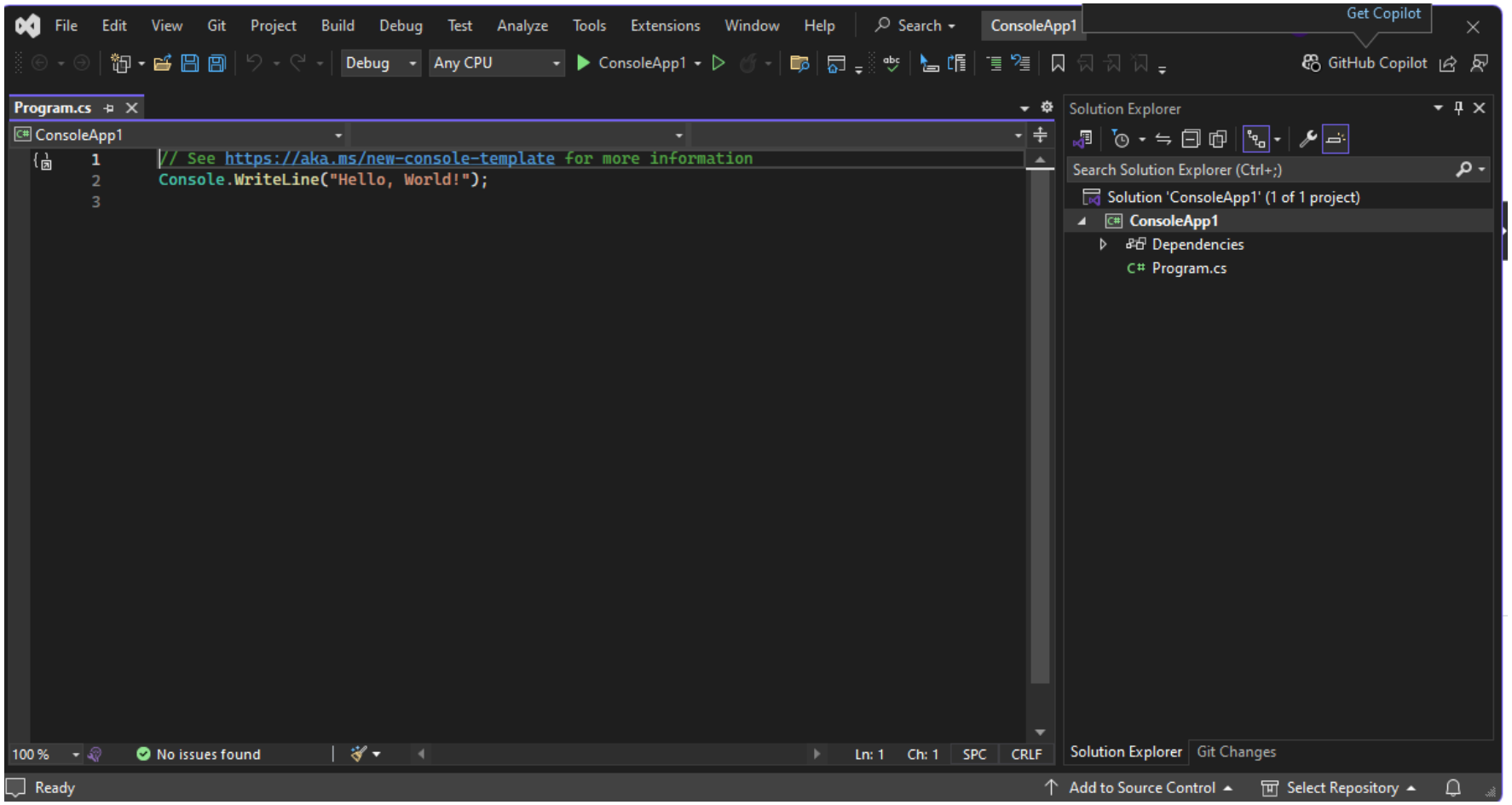
Create new project

Select Framework and click “Create” button



Create new project

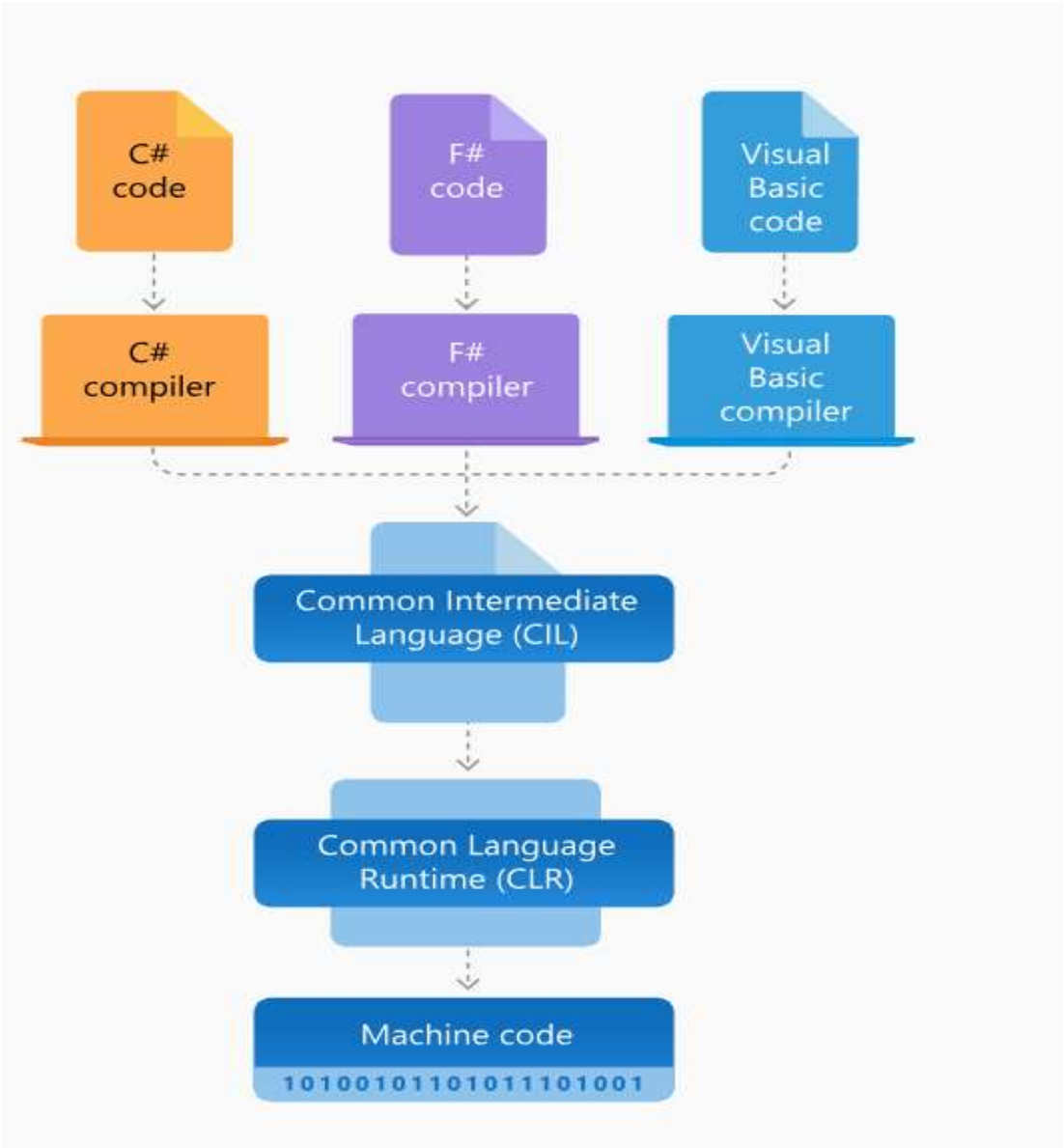
Your first project is ready!!



Frameworks and Compiling

- **.NET Framework** is used to build and run websites, services, desktop apps, and console apps.
- .NET Framework is used to build and run websites, services, desktop apps, and console apps.

  It's designed to:
    - Provide a consistent programming environment
    - Minimize software deployment and versioning conflicts
    - Promote safe execution of code
    - Eliminate performance problems
- NET Framework includes:
  - A common language runtime that manages memory and other system services
  - A class library that provides reliable code for app development
- **Compiling** is the transformation from Source Code (human-readable) into machine code (computer-executable)


C# code
F# code
Visual Basic code
C# compiler
F# compiler
Visual Basic compiler
Common Intermediate Language (CIL)
Common Language Runtime (CLR)
Machine code
10100101101011101001

C# Comments

```
// This is a comment
Console.WriteLine("Hello World!");
```

- **Comments** can be used to explain C# code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

- **Single-line Comments**
  - Single-line comments start with two forward slashes (//).
  - Any text between // and the end of the line is ignored by C# (will not be executed).

C# Comments

```
/* The code below will print the words Hello World
to the screen, and it is amazing */
Console.WriteLine("Hello World!");
```

- **Multi- Line Comments**
  - Multi-line comments start with /* and ends with */.
  - Any text between /* and */ will be ignored by C#.

Single or multi-line comments?

It is up to you which you want to use. Normally, we use // for short comments, and /* */ for longer.

Top Level Statements

C#

```
using System;

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

C#

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

- Top-level statements enable you to avoid the extra ceremony required by placing your program's entry point in a static method in a class.
- The typical starting point for a new console application looks like the following code (image1)
- The preceding code is the result of running the dotnet new console command and creating a new console application.(image 2)

Using System Directives

- The using directive in C# imports types from a namespace, allowing you to use them without specifying the namespace's full name.
- For example, using System.Text imports all types from the System.Text namespace

Declaring Variables

- To create a variable, you must specify the type and assign it a value
- Syntax :
  - *type variableName = value;*

  Type is C# type, eg : int or string

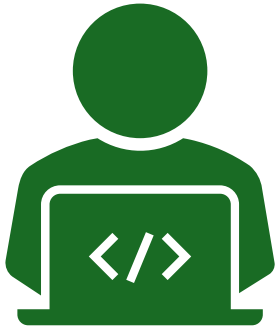
  variableName is the name of the variable, eg: x or name

  The ***equal sign*** is used to assign values to the variable.

```
int myNum = 5;
double myDoubleNum = 5.99D;
char myLetter = 'D';
bool myBool = true;
string myText = "Hello";
```

User Input and the Data Type String

Print the variables using
Console.WriteLine() method



Read input variables using
Console.ReadLine() method

Overview of Visual Studio

- UI overview
- Folder Structure
- Complier errors
- Runtime error

Quiz 1

- What is Visual Studio?
  - A text editor
  - An IDE used for C# development
  - A web browser

Quiz 1

- **What is an IDE?**
  - An integrated development environment – a software, that helps us develop C#
  - A shortcut for idea
  - International developer enterprise

Quiz 1

- **What does Console.WriteLine("Hello World") do?**
  - It shows Hello World in the console
  - It wants me to write Hello World into the console
  - It deletes your System32 folder from your computer

Quiz 1

- **What is the purpose of the Hello World program?**
  - To cause errors in software
  - To perform complex calculations
  - To demonstrate a simple output in a new programming environment

Coding Standards

Below are some of the best practices which all the .Net Developers should follow:

1. Class and Method names should always be in Pascal Case
2. Method argument and Local variables should always be in Camel Case
3. Avoid the use of underscore while naming identifiers
4. Avoid the use of System data types and prefer using the Predefined data types.
5. Always prefix an interface with letter I.
6. For better code indentation and readability always align the curly braces vertically.
7. Always use the using keyword when working with disposable types. It automatically disposes the object when program flow leaves the scope.
8. Always declare the variables as close as possible to their use.
9. Always declare the properties as private so as to achieve Encapsulation and ensure data hiding.
10. Always separate the methods, different sections of program by one space.
11. Constants should always be declared in UPPER_CASE.

C# Variables

Variables are containers for storing data values.

In C#, there are different **types** of variables (defined with different keywords), for example:

Int	Double	Char	String	bool

Data Types

- Value Types
- Reference Types
- Pointer Types
- User-Defined Types
- Structured Types
- Nullable Types

Value Data Types

- **Stored in:** Stack memory.
- **What they hold:** The actual data.
- **When copied:** A new copy of the value is created, so changes to one copy don’t affect the other.
- **Examples:** int, float, bool, char, struct, enum.

Reference Data Types

- **Stored in:** Heap memory (the reference/pointer is stored in the stack).
- **What they hold:** A reference (address) to the memory location where the data is stored.
- **When copied:** The reference (not the actual data) is copied, so changes to the data affect all references.
- **Examples:** string, class, array, object, delegate.

Feature	String	Array
Mutability	Immutable (cannot change directly)	Mutable (can change elements)
Storage Behavior	Stored in a string pool (for reuse)	Stored in memory (unique copy)
Purpose	Text data that rarely changes at a character level	Collections of items meant to be updated
Performance	New string created for every change	Same array updated in place

Key Differences

Aspect	Value Types	Reference Types
Storage Location	Stack	Heap (reference on Stack)
Holds	Actual data	Address/reference to the data
Copy Behavior	Creates a new copy	Copies the reference
Impact of Change	Changes to one don't affect the other	Changes affect all references
Examples	int, float, bool	string, class, array

Lab 1

Create a simple C# program that declares an integer variable, assigns it the value 10, and prints the value to the console.

Alert!

The result of execution for the default string should be:

"10"

Parsing

- Parsing allows us to convert data from one data type into another data type.
- **Int32.Parse(String) Method** is used to convert the string representation of a number to its 32-bit signed integer equivalent.
- **Return Value:** It is a 32-bit signed integer equivalent to the number contained in *str*.
- **Exceptions:**
  - **ArgumentNullException**: If *str* is null.
  - **FormatException**: If *str* is not in the correct format.
  - **OverflowException**: If *str* represents a number less than *MinValue* or greater than *MaxValue*.

Quiz 2

What does the int.Parse() method do?

- Converts a string to an integer
- Changes an integer to a string
- Calculates the sum of two integers

Quiz 2

Which of the following will throw an error?

- Int.Parse(“1234”)
- Int.Parse(“Hello”)
- Int.Parse(“2024”)

Quiz 2

Why do we use type conversion?

- To use more memory
- To ensure variables match the expected data types
- To make the program run faster

Lab 2

Addition Calculator

- Let's use **int** DataType
- Parsing

String concatenation

The + operator can be used between strings to combine them. This is called **concatenation**

```
string firstName = "John ";
string lastName = "Doe";
string name = string.Concat(firstName, lastName);
Console.WriteLine(name);
```

You can also use the string.Concat() method to concatenate two strings.

```
string firstName = "John ";
string lastName = "Doe";
string name = firstName + lastName;
Console.WriteLine(name);
```

```
string firstName = "John";
string lastName = "Doe";
string name = $"My full name is: {firstName} {lastName}";
Console.WriteLine(name);
```

Another option of string concatenation, is **string interpolation**, which substitutes values of variables into placeholders in a string.

String interpolation

```
var name = "Mark";
var date = DateTime.Now;
// Composite formatting:
Console.WriteLine("Hello, {0}! Today is {1}, it's {2:HH:mm} now.", name, date.DayOfWeek, date);
```

A composite format string consists of fixed text intermixed with indexed placeholders, called format items.

String formatting

Using double Datatype

double.Parse(userInput)

- Cannot implicitly convert from double to int
- Can convert int to double

Implicit Conversion

- **Implicit Type Conversion** is also known as '**automatic type conversion**'.
- No Data loss
- converting a smaller type to a larger type size

`char -> int -> long -> float -> double`

```
int a = 123;
long b = a;      // implicit conversion from int to long
```

Explicit Conversion

- Explicit conversions can occur in cast expressions
- Converting a larger type to a smaller size type
- Also called casting

`double` -> `float` -> `long` -> `int` -> `char`

```
int a = 123;
long b = a;
int c = (int)b; // explicit conversion from long to int
```

Conversion Helpers

Parse

Convert

It is also possible to convert data types explicitly by using built-in methods, such as Convert.ToBoolean, Convert.ToDouble, Convert.ToString, Convert.ToInt32 (int) and Convert.ToInt64 (long)

Implicitly typed local variables

- Var keyword
- Compiler determines and assigns the most appropriate type.

```
// i is compiled as an int
var i = 5;

// s is compiled as a string
var s = "Hello";

// a is compiled as int[]
var a = new[] { 0, 1, 2 };
```

```
235
236
237
238
```

```
string value = "C-Sharp Corner";

value = value.Substring(0, 7);
```

Debugging and Breakpoints

- Debugging is the process of identifying the root cause of an error and correcting
- We can set a breakpoint on any line of executable code to pause the debugger to see some values

Quiz 3

What is a breakpoint in debugging?

- A point that stops the execution of the program
- An error in the code
- A type of variable

Quiz 3

Which of the following is a compile-time error?

- ○ Missing semicolon at the end of a statement
- ○ Dividing by zero
- ○ Accessing out of range elements in an array

Quiz 3

What does a runtime error mean?

- ○ An error that is caught when writing code
- ○ An error that occurs when the program is running
- ○ An error that can be ignored

Lab : Fixing bugs

- Create a C# program that correctly calculates and displays the average of three numbers. The provided code has several bugs that prevent it from compiling and running correctly.

```
namespace Coding.Exercise
{
   public class Exercise
   {
      public void CalculateAverage()
      {
         int num1 = 10;
         int num2 = 20;
         int num3 = 30;
         int average = (num1 + num2 + num3) / 3
         Console.WriteLn("The average is: " + average);
      }
   }
}
```

Characters that are Reserved in C#

- Braces ({ })
- Parentheses (( ))
- Square Brackets ([ ])
- Semicolon (;)
- Colon (:)
- Comma (,)
- Period (.)
- Question Mark (?)
- Plus (+)

Operators

- **Operators** are used to perform operations on variables and values.
  - Arithmetic
  - Assignment
  - Comparison
  - Logical

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	x++
--	Decrement	Decreases the value of a variable by 1	x--

Arithmetic Operators

- Arithmetic operators are used to perform common mathematical operations

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Assignment Operators

Assignment operators are used to assign values to variables.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Comparison/Relational Operators

Comparison operators are used to compare two values (or variables).

Operator	Name	Description	Example
&&	Logical and	Returns True if both statements are true	x < 5 && x < 10
||	Logical or	Returns True if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns False if the result is true	!(x < 5 && x < 10)

Logical Operators

As with comparison operators, you can also test for True or False values with logical operators.

Logical Operators

- **Bool or Boolean** – Important datatype for decision making
- A boolean type is declared with the **bool** keyword and can only take the values **true** or **false.**

```
bool isCSharpFun = true;
```

- **if condition** : Statement to specify a block of C# code to be executed if a condition is True
- Logical operators are
  - AND &&
  - OR ||
  - NOT !

Logical Operator – OR ||

- Returns True if one of the statements is true.
- Variants of OR statements

  true || true --> true
  true || false --> true
  false || true --> true
  false || false --> false

Logical Operator – AND &&

- If both the operands are non zero then condition becomes true.
- Variants of AND statements

  true || true --> true
  true || false --> false
  false || true --> false
  false || false --> false

Relational operators lab

Use < <= > >=

Equality Operator and not equal

==

Checks if the values of two operands are equal or not, if yes then condition becomes true.

!=

Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

Quiz 4

Which logical operator is used to determine if either of two conditions is true?

- AND
- OR
- NOT

Quiz 4

What does the NOT operator do?

- Reverses the condition’s outcome
- Combines two conditions
- None of the above

Quiz 4

Which operator is used to check if both conditions are true?

- AND
- OR
- NOT

The conditional statements of C#

- if
- if-else
- if-else-if
- Nested if
- Switch
- Nested switch

```
if(condition)
     {
          //code to be executed
     }
```

if condition

The if statement checks the given condition. If the condition evaluates to be true then the block of code/statements will execute otherwise not.

```
if (condition)
{
  // block of code to be executed if the condition is True
}
else
{
  // block of code to be executed if the condition is False
}
```

else condition

- Use **else** to specify a block of code to be executed, if the same condition is false.

DataType Bool and Conditional Statements - LAB

- Create a simple C# program that uses a boolean variable to store whether a user is logged in and prints different messages based on the value. The final output must be "Welcome back!"

Alert!

The result of execution for the default string should be:

"Welcome back!"

```
if (condition1)
{
  // block of code to be executed if condition1 is True
}
else if (condition2)
{
  // block of code to be executed if the condition1 is false and condition2 is True
}
else
{
  // block of code to be executed if the condition1 is false and condition2 is False
}
```

- Use the **else if** statement to specify a new condition if the first condition is False.

else if condition

Quiz 5

What does an if statement do?

- ○ Executes a block of code if the condition is true
- ○ Executes a block of code if the condition is false
- ○ None of the above

Quiz 5

Which keyword is used to execute a code block when the if condition is not met?

- if
- Else
- Elseif

Quiz 5

What is the purpose of the ElseIf clause?

- ○ To specify a new condition if the first condition is true
- ○ To execute code without conditions
- ○ To specify a new condition if the first condition is false

Nested if statements

if statement inside an if statement is known as nested if

```
if (condition1)
{
     // code to be executed
     // if condition2 is true
     if (condition2)
     {
         // code to be executed
         // if condition2 is true
     }
}
```

Scope of variables

- Class Level Scope
- Method Level Scope
- Block Level Scope

Party program - Lab

- Learn to use nested if statements
- Learn scope of variables

One Line if statements

- curly brackets { } not required
- If the curly brackets { } are not used with if statements then the statement just next to it is only considered associated with the if statement.

Switch Statements

- Use the switch statement to select one of many code blocks to be executed.
- When C# reaches a break keyword, it breaks out of the switch block.

```
switch(expression)
{
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
    break;
}
```

Use Switch or if statement?

Use If-else When

- **Complex Conditions**: If your decision-making involves complex conditions, logical operators, or multiple variables, if-else provides the flexibility needed.
- **Ranges**: When your conditions involve ranges of values rather than specific values.

Use Switch When

- **Distinct Values**: When you have a variable with distinct, discrete values, and you want to execute different blocks of code for each value.
- **Readability**: Switch statements can enhance the readability of your code, especially when dealing with a large number of conditions based on the same variable.

Using the Switch Statement - LAB

- Create a simple C# program using a switch statement to replace an if-else structure for printing days of the week based on an integer value.

Alert!

The result of execution for the default string should be:

"Wednesday"

```
int day = 3;
// TODO: Rewrite this if-else structure using a switch
    statement.
if (day == 1)
  Console.WriteLine("Monday");
else if (day == 2)
  Console.WriteLine("Tuesday");
else if (day == 3)
  Console.WriteLine("Wednesday");
else
  Console.WriteLine("Another day");
```

Let's a create a little Quiz App! - LAB

1

Trim() method – Remove white/empty spaces before and after string

2

Lowercase() method – Converts string to lower case

3

Increment operator – Easy way for incrementing the count by 1 using ++ operator.

4

Decrement operator - Easy way for decrementing the count by 1 using -- operator.

Explore few operators yourself

Declare an integer variable “number” and try printing the variable after using each of the operators below :

- ŋụ̣ṇčês
- ŋụ̣ṇčês
- ŋụ̣ṇčês
- ŋụ̣ṇčês
- ŋụ̣ṇčês ‘ ॰
- ŋụ̣ṇčês ’ ॰
- ŋụ̣ṇčês , ॰
- ŋụ̣ṇčês ’ ॰

Explore few operators yourself

Use Modulo Operator ‘%’

Loops in C#

Types of loops in C#

FOR LOOP

WHILE LOOP

DO WHILE LOOP

FOREACH LOOP

The For Loop

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

```
for (statement 1; statement 2; statement 3)
{
  // code block to be executed
}
```

When you know exactly how many times you want to loop through a block of code, use the for loop.

For Loop – Count down challenge

Escape Characters r and n

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Code	Result
\n	New Line
\t	Tab
\b	Backspace

Thread.Sleep(1000)

Suspending the current thread for the specified amount of time

Time in milliseconds

Rocket Landing Project - Lab

Rocket Landing Project - Lab

Here are your objectives:

1. String Representation: Start with a string that represents the rocket. This string will include line breaks and slashes to create a simple rocket shape.

2. Console Display: Your goal is to display this rocket string in the console and make it appear as though the rocket is descending step by step.

3. Console Clear: Use the `Console.Clear()` method to clear the console at each step of the descent. This will make the movement of the rocket smooth and continuous.

4. Descent Mechanism: Implement a mechanism to add an empty line above the rocket for each iteration. This will simulate the rocket descending.

5. Final Message: Once the rocket has landed, display a message such as "The rocket has landed. Woohoo! Another successful landing!"

Remember, the key part of this challenge is to ensure the rocket descends smoothly, giving the appearance of a controlled landing. You might need to experiment with adding empty lines and clearing the console to achieve the desired effect.

```
while (condition)
{
  // code block to be executed
}
```

The while loop loops through a block of code as long as a specified condition is True.

While loop

Guess the Number Challenge - Lab

Difference between for and while loop

for Loop	While loop
Use when you **know** how many times to repeat.	Use when you **don't know** how many times to repeat.
Everything (start, stop, step) is set up in one line.	The condition is checked first, and steps happen inside the loop.

Quiz 7

Which statement about the while loop is true?

- It checks the loop condition after executing the loop body
- It executes the loop body as long as the condition remains true
- It executes regardless of the initial condition

Quiz 7

When would you use a while loop instead of a for loop?

- When you know the exact number of iterations beforehand.
- When the number of iterations depends on a condition evaluated during runtime.
- When iterating through collection or arrays

Quiz 7

What happens if the condition in a while loop never becomes false?

- The loop will pause execution to prevent freezing.
- The loop terminates after a default number of iterations.
- The loop becomes infinite, which can cause the program to hang.

Adventure Game - Lab

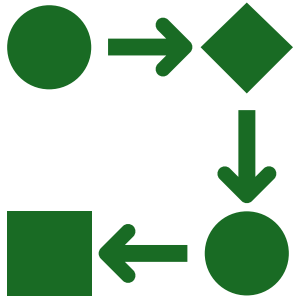
```
do {
    // Code block to be executed at least once
    // This block will execute at least once
} while (condition);
```

Do While Loops

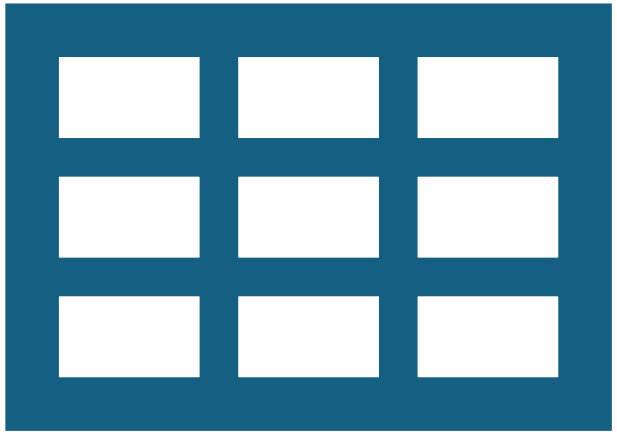
It is similar to the while loop, but with one key difference: the condition is evaluated after the execution of the loop's body, ensuring that the loop's body is executed at least once.

break and continue

The break statement can also be used to jump out of a loop.



The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.



Arrays

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- To declare an array, define the variable type with **square brackets**
- Array **indexes** start with 0: [0] is the first element. [1] is the second element, etc.

```
foreach (type variableName in arrayName)
{
  // code block to be executed
}
```

foreach loop

Used exclusively to loop through elements in an **array**.

Quiz 8

What is the primary function of a for loop in C#?

- To execute a block of code a limited number of times
- To execute code indefinitely
- To check a condition at the end of each iteration

Quiz 8

Which loop is best used when the number of iterations is not known beforehand?

- For loop
- While loop
- Foreach loop

Quiz 8

What does the break statement do in a loop?

- ○ Pauses the loop execution temporarily
- ○ Terminates the loop immediately
- ○ Skips the current iteration and proceeds to the next one

Two-Dimensional Arrays

To create a 2D array, add each array within its own set of curly braces, and insert a comma (,) inside the square brackets

```
int[,] numbers = { {1, 4, 2}, {3, 6, 8} };
```

numbers is now an array with two arrays as its elements. The first array element contains three elements: 1, 4 and 2, while the second array element contains 3, 6 and 8. To visualize it, think of the array as a table with rows and columns

	COLUMN 0	COLUMN 1	COLUMN 2
ROW 0	1	4	2
ROW 1	3	6	8

3D Arrays

A three-dimensional array would have two commas: int[,,]

Using Arrays -Lab

Create a simple C# program that declares an array of integers, initializes it with values, and uses a foreach loop to print each value to the console.

Alert!

The result of execution for each default value should be printed on new lines as:

,

,

,

–

—

ADVANCED LAB:
Working with Two-Dimensional Arrays and Loops

Ảỷì ẻâ Ăẻẵźśư ỵ ụ ẤẲ³ẵỷỷv ɔỷẻừ Ấữ ẻôẤźśưźïẻụẤM źẵẻẤ ỂỈ ẤữÈ v 4 ŝư ì ửźśv ửẻụẤẻỷỷẻỂẤE śữ Ấźửâ ɔì ỷźẤẽẻụẽ º ụẻâ źẤữ ì Ấẽº ừ Ấv ḳẤẻ ẽ ữ Ấỷv È 6Ấẻửî ẤỷỷśửôźẤữ ì Ấẽº ừ źẤâv Ấữ ì Ấẽv ửźv ụ 9

Ňữ ì Ấỷỷv ɔỷẻừ Ấữ v º ụ 8

#9 Ấẽ ẽ ụẻỷì Ấẻửî ẤźửźïẻụẤM Ấẻẵźấỉ Ỉ ẤữÈ v 4 ŝư ì ửźśv ửẻụẤẻỷỷẻỂẤE śữ Ấữ ì Ấỵv ụỷ È śửɔẤẻụ º ì ź8

#Ấ Ấ

\Ấ Ấ

–Ấ Ấ–

¯9 Ảẻụẽ º ụẻâ Ấữ ì Ấẽº ừ Ấv ḳẤẻ ẽ ữ Ấỷv È ẤźửẤữ ì ẤỵỷỷẻỂ9Ấ

/ 9 ĴỷśửôẤữ ì Ấẽº ừ Ấv ḳẤẻ ẽ ữ Ấỷv È Ấâv Ấữ ì Ấẽv ửźv ụ 9Ấ

Ấụ ỷôɔẤỂ ì Ấỷì źº ụôẤv ḳẤỈ ẽº ɑv ửẤỵv ỷẤữ ì Ấì ḳẻº ụôẤźôỷśửɔẤữ v º ụ Ấẽ ì 8

-

#-

¯\

HINT : Use nested loops to iterate through the array and calculate the sums.

Methods in C#

A method is a code block that contains a series of statements. A program causes the statements to be executed by calling the method and specifying any required method arguments. In C#, every executed instruction is performed in the context of a method.

```
<Access Specifier> <Return Type> <Method Name>(Parameter List) {
  Method Body
}
```

Defining Methods in C#

- **Access Specifier** – This determines the visibility of a variable or a method from another class.
- **Return type** – A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is **void**.
- **Method name** – Method name is a unique identifier, and it is case sensitive. It cannot be same as any other identifier declared in the class.
- **Parameter list** – Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
- **Method body** – This contains the set of instructions needed to complete the required activity.

https://www.tutorialspoint.com/csharp/csharp_methods.htm

```
public int Add(int num1, int num2)
{
    int result = num1 + num2;
    return result;
}
```

Void method with Parameter - lab

When method with parameters is called, you need to pass the parameters to the method

Scope of variables

Method level variables have scope inside the method only, cannot be used outside.

Moving to the classical Template - Top Level Statements

A C# program consists of the following parts –

- Namespace declaration
- A class
- Class methods
- Class attributes
- A Main method
- Statements and Expressions
- Comments

```
using System;

namespace HelloWorldApplication {
   class HelloWorld {
      static void Main(string[] args) {
         /* my first program in C# */
         Console.WriteLine("Hello World");
         Console.ReadKey();
      }
   }
}
```

WRITING FIRST PROGRAM

tutorials.EU

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

Reference of .Net Framework Namespaces

Namespace name

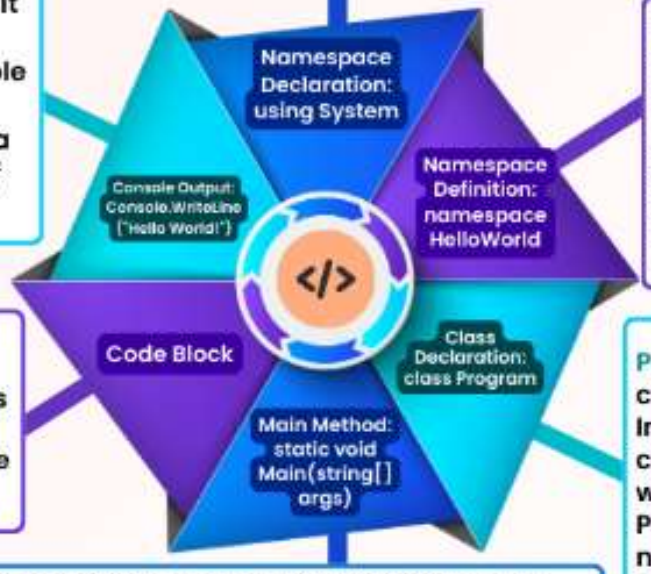
class name

Method

Method to Display value on Console

Purpose: This line tells the compiler that the program is using the System namespace, which includes fundamental classes necessary for basic operations, like input/output. The Console class is part of this namespace

Purpose: This line of code outputs the string "Hello World!" to the console window. It calls the WriteLine method of the Console class, which automatically adds a newline at the end of the output.



Purpose: Namespaces are used to organize code and prevent name conflicts. In this example, HelloWorld is the namespace that encapsulates the program's logic.

Purpose: The code block {} following the Main method contains the statements that are executed when the program runs.

Purpose: Defines a class named Program. In C#, all executable code is contained within classes. Program is a common name used for the main class where the entry point of the program is located.

Purpose: This is the main entry point of any C# console application. The Main method must be static, and it typically returns void (though it can also return int), indicating it does not return a value.
Parameters: string[] args represents an array of command-line arguments that can be passed to the program when it is executed.

Moving our Methods outside of the Main Method - lab

Fields

When a variable is declared directly in a class, it is often referred to as a **field** (or attribute).

Quiz 9

What is a void method in C#?

- ○ A method that returns a specific data type.
- ○ A method that returns no value.
- ○ A method that cannot accept parameters

Quiz 9

How does a method with parameters differ from one without?

- It cannot be called within the Main method
- It requires specific information to operate
- It runs faster than methods without parameters

Quiz 9

What is the purpose of moving methods outside of the Main method?

- ○ To make the program run faster
- ○ To reduce the readability of the code
- ○ To improve code organization and reusability

Quiz 9

What defines the scope of a variable in C#?

- The size of the data type.
- The location where the variable is declared.
- The number of times it is used.

Quiz 9

How do instance variables differ from local variables?

- ○ Instance variables are declared within methods.
- ○ Instance variables are accessible throughout the class.
- ○ Instance variables have shorter lifetimes.

Quiz 9

What is the main advantage of using fields in a class?

- ○ They allow for methods to be shorter.
- ○ They can be modified by any method in the class.
- ○ They increase the processing speed of operations.

WeatherSimulator - Lab

OOP

Object Oriented Programming

Creating our first Class

Create Clas
Create Object
Create fields
Create Custom Constructor

Class

- A class is a blueprint of an Object
- It has actions/abilities – so called member functions or methods
- It has properties – so called member variables
- Inheritance possible
- Can be used like a Datatype (e.g. String is a Class)

Object

- **Objects** are instances of classes.
- Objects are used to store data and perform actions in a program.
- **Key Concepts of Objects in C#:**

1. **Properties:**
   Objects have attributes or traits, called properties, that define their characteristics.
2. **Methods:**
   Objects can perform actions using methods, which are functions defined in the class.
3. **Fields:**
   These are variables that belong to the object, used to store data.
4. **Creation:**
   You create an object from a class using the new keyword. This is called **instantiation**.

Member Variables

- Field inside the class but outside the method
- Represents data and behavior of classes and structs
- Member variables are kept private
- These variables can only be accessed using the public member functions.
- Member variables are named starting with underscrore symbol “_”

Constructors

- A constructor is a **special method** that is used to initialize objects.
- Gets automatically invoked whenever an instance of the class is created
- Like methods, a constructor also contains the collection of instructions that are executed at the time of Object creation
- It is used to assign initial values to the **data members** of the same class.

```
public class Person
{
   private string last;
   private string first;

   public Person(string lastName, string firstName)
   {
      last = lastName;
      first = firstName;
   }

   // Remaining implementation of Person class.
}
```

Key points About Constructors

- Constructor of a class must have the same name as the class name in which it resides.
- A constructor can not be abstract, final, and Synchronized.
- Within a class, you can create only one static constructor.
- A constructor doesn’t have any return type, not even void.
- A static constructor cannot be a parameterized constructor.
- A class can have any number of constructors.
- Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

Types of Contrustors

- Default Constructor
- Parameterized Constructor
- Copy Constructor
- Private Constructor
- Static Constructor

Encapsulation

Involves **hiding the internal state and behavior** of an object and allowing controlled access to it through **publicly exposed methods or properties**.

<u>Encapsulation is implemented by:</u>

1. Declaring fields as private
2. Using properties with getters and setters
3. Using access modifiers

Properties

- Properties are the special type of class members that provides a flexible mechanism to read, write, or compute the value of a private field.
- **Encapsulation** and hiding of information can also be achieved using properties.
- It uses pre-defined methods which are “get” and “set” methods which help to access and modify the properties.
- **Accessors:** The block of “set” and “get” is known as “Accessors”.

Properties Syntax

```
<access_modifier> <return_type> <property_name>
{
        get { // body }
        set { // body }
}
```

<access_modifier> can be public, private, protected or internal.

<return_type> can be any valid C# type.

<property_name> can be user-defined.

Get Accessor: It specifies that the value of a field can access publicly. It returns a single value, and it specifies the *read-only property*.

Set Accessor: It will specify the assignment of a value to a private field in a property. It returns a single value, and it specifies the *write-only property*.

Setting property conditions - Demo

Setting property conditions - Demo

Simpler form of property when no get set methods have no condition - Demo

Types of Properties

- **Read and Write Properties:** When property contains both get and set methods.
- **Read-Only Properties:** When property contains only get method.
- **Write Only Properties:** When property contains only set method.
- **Auto Implemented Properties:** When there is no additional logic in the property accessors, and it introduce in C# 3.0.

Multiple Constructors

- Class can have multiple constructors.
- C# can distinguish the constructors with different signatures. i.e. the constructor must have the same name but with different parameters list.
- We can overload constructors in different ways as follows:
    - By using different **type of arguments**
    - By using different **number of arguments**
    - By using different **order of arguments**

Default Constructor

- A constructor with no parameters is called a default constructor.
- A default constructor has every instance of the class to be initialized to the same values.
- The default constructor initializes all numeric fields to zero and all string and object fields to null inside a class.

Parameterized Constructor

- A constructor having at least one parameter is called as parameterized constructor.
- It can initialize each instance of the class to different values.
- Also called as custom constructors

Copy Constructor

- This constructor creates an object by copying variables from another object.
- Its main use is to initialize a new instance to the values of an existing instance.

Private Constructor

- If a constructor is created with private specifier is known as Private Constructor.
- It is not possible for other classes to derive from this class.
- It’s not possible to create an instance of this class.

Static Constructor

- Static Constructor has to be invoked only once in the class
- It has been invoked during the creation of the first reference to a static member in the class.
- A static constructor is initialized static fields or data of the class and to be executed only once.

Quiz

What is the purpose of a constructor in a class?

- To initialize an object of the class
- To destroy an object of the class
- To define methods for the class

Quiz

Can a class have more than one constructor?

- ○ Yes, a class can have multiple constructors with different parameters
- ○ No, a class can only have a constructor
- ○ Yes, but only if they have the same parameters

Quiz

What is a default constructor?

- ○ A constructor that does not take any parameters
- ○ A constructor that initializes all variables to zero
- ○ A constructor that is automatically generated by the compiler

More about Methods..

Optional parameters

- Also called as default parameter.
- Use a default parameter value, by using the equals sign (=).
- Can be used with method, constructor, indexer or delegate.
- Any call must provide arguments for all required parameters but can omit arguments for optional parameters.
- Optional parameters are defined at the end of the parameter list, after any required parameters

```
static void MyMethod(string child1, string child2, string child3)
{
  Console.WriteLine("The youngest child is: " + child3);
}

static void Main(string[] args)
{
  MyMethod(child3: "John", child1: "Liam", child2: "Liam");
}
```

It is also possible to send arguments with the key: value syntax.

The order of the arguments does not matter.

Named Parameters

Static Keywords

- **static** is a modifier in C# which is applicable for the following:
    - Classes
    - Variables
    - Methods
    - Constructor
- To create a static member, precede its declaration with the keyword *static*.
- When a member is declared static, it can be accessed with the name of its class directly.

Static Class

- A static class is declared with the help of *static* keyword.
- A static class can only contain static data members, static methods, and a static constructor.
- It is not allowed to create objects of the static class.
- Static classes are ***sealed***, means one cannot inherit a static class from another class.

Static Variable

- A static variable is declared with the help of static keyword.
- When a variable is declared as static, then a single copy of the variable is created and shared among all objects at the class level.
- Static variables are accessed with the name of the class, they do not require any object for access.

Static Method

- A static method is declared with the help of the static keyword.
- Static methods are accessed with the name of the class.
- A static method can access only static fields and methods directly, while non-static fields and methods require an instance of the class to be accessed.

Static Constructor

- A static constructor is declared with the help of static keyword.
- Static Constructor has to be invoked only once in the class, and it has been invoked during the creation of the first reference to a static member in the class.
- A static constructor is initialized static fields or data of the class and to be executed only once.

Using Static Methods : lab

- Create a simple C# program that declares a static method which takes an integer as a parameter, doubles it, and returns the result. Call this static method from the Main method and print the result to the console.
- The result of execution for the default integer should be:"20"

Limitations of Static Keyword

- **static** keyword cannot be used by indexers, finalizers, or types other than classes.
- A static member is not referenced through an instance.
- In C#, it is not allowed to use this to reference static methods or property accessors.
- In C#, if static keyword is used with the class, then the static class always contain static members.

Usage Considerations

- **Memory Usage**: Static members are allocated once and live for the duration of the application, which can be more memory-efficient for shared data.
- **Global State**: Static members maintain a global state which can lead to issues with data consistency, especially in multithreaded scenarios. Care should be taken to manage access to static members, possibly using synchronization mechanisms if needed.
- **Testing Challenges**: Classes that use static members can be harder to test due to their persistent state across tests. This can be mitigated by using patterns such as dependency injection to abstract away static dependencies.

Modifier	Description
public	The code is accessible for all classes
private	The code is only accessible within the same class
protected	The code is accessible within the same class, or in a class that is inherited from that class.
internal	The code is only accessible within its own assembly, but not from another assembly.

C# has the following access modifiers

Access modifiers

	public	protected	internal	protected internal	private	private protected
Entire program	Yes	No	No	No	No	No
Containing class	Yes	Yes	Yes	Yes	Yes	Yes
Current assembly	Yes	No	Yes	Yes	No	No
Derived types	Yes	Yes	No	Yes	No	No
Derived types within current assembly	Yes	Yes	Yes	Yes	No	Yes

Accessibility table of modifiers

Private Modifier

- Access is only granted to the containing class.
- Any other class inside the current or another assembly is not granted access to these members.

```
private TypeName
```

Public Constructor

- Access is granted to the entire program.
- This means that another method or another assembly which contains the class reference can access these members or types.
- This access modifier has the most permissive access level in comparison to all other access modifiers.

```
public TypeName
```

Readonly fields

- Declare a field using readonly modifier
- It indicates that the assignment to the fields is only the part of the declaration or in a constructor to the same class.
- Such types of fields can only be assigned or reassigned multiple times only at the declaration or in a constructor.
- They are not assigned after the constructor exit.
- Syntax : **public readonly int MyReadOnlyField;**

Readonly properties

- This has only get accessor

ReadOnly Keyword	Const Keyword
In C#, readonly fields can be created using readonly keyword	In C#, constant fields are created using const keyword.
ReadOnly is a runtime constant.	Const is a compile time constant.
The value of readonly field can be changed.	The value of the const field can not be changed.
It cannot be declared inside the method.	It can be declared inside the method.
In readonly fields, we can assign values in declaration and in the constructor part.	In const fields, we can only assign values in declaration part.
It can be used with static modifiers.	It cannot be used with static modifiers.

Constants and read only

Quiz

What is a read-only property in C#?

- ○ A property that can only be set within the constructor and not modified afterwards
- ○ A property that can be set and modified at any time
- ○ A property that cannot be set at all

Quiz

How do you declare a read-only property?

- By using the readonly keyword before the property type
- By using only the get accessor without the set accessor
- By using the const keyword

Quiz

Where can you assign a value to a read-only property?

- In the constructor of the class
- Anywhere within the class
- In any method of the class

this keyword

this keyword is used to refer to the current instance of the class.

Different ways to use ‘this’ keyword in C#

- Using ‘this’ keyword to refer current class instance members
- Using this() to invoke the constructor in same class
- Using ‘this’ keyword to invoke current class method
- Using ‘this’ keyword as method parameter
- Using this keyword to declare an indexer

C# Collections

Collection classes are specialized classes for data storage and retrieval.

They provide a way to store, manage, and manipulate groups of related data in a program.

Collections are part of the **System.Collections** namespace (and its generic version, **System.Collections.Generic**) and are widely used for managing dynamic data.

Types of Collections in C#

Non-Generic Collections	Generic Collections	Specialized Collections
ArrayList	List	ObservableCollection
Hashtable	Dictionary<TKey, Tvalue>	ConcurrentBag
Queue	Queue and Stack	
Stack	HashSet	

Why Use Collections?

DYNAMIC SIZE

EASE OF USE

TYPE-SAFETY (GENERIC)

BUILT-IN SORTING & SEARCHING

Lists in C#

A **List<T>** is a generic collection that stores elements in a linear sequence. Unlike arrays, lists are dynamic which means their size can increase or decrease as needed.

What Is the ‘T’ in List<T>?

In the list, the letter **T** stands for the type of object.

This type is passed as a parameter during runtime.

As we said, T means the Object Type, so, if you want to create a list that has nothing but strings then you must use object type- string in place of T.

Ľîșʧ șʧsîŋĝ ŋǎṇêș ŋêx Ľîșʧ șʧsîŋĝ

ŋǎṇêș Ađđ Aľîçê

ŋǎṇêș Ađđ Bộč

ŋǎṇêș Ađđ Kộḥŋ

Ľîșʧ îŋʧ ŋụṇčêsș ŋêx Ľîșʧ îŋʧ

ŋụṇčêsș Ađđ ,

ŋụṇčêsș Ađđ ,

ŋụṇčêsș Ađđ ,

Properties of Lists

Capacity

- Number of elements List<T> can contain.
- The default capacity of a List<T> is 0.

Count

- Number of elements in List <T>.

Core Methods of Lists

Add

- **Add(T item)**
- Adds an item to the end of the list.

Remove

- **Remove(T item)**
- Removes the first occurrence of a specific object from the list.

Index

- **Index(int index, T item)**
- Inserts an element at the specified index.

Find

- **Find(Predicate<T> match)**
- Searches for an element that matches the conditions defined by a specified predicate.

Core Methods of Lists

Sort	Clear	ToArray	IndexOf
Sort() Sorts the elements in the entire List<T>	**Clear()** Removes all elements from the list.	**ToArray()** Copies the elements of the List<T> to new array.	**IndexOf(T item)** Returns the index of the first occurrence of a specific object in the list.

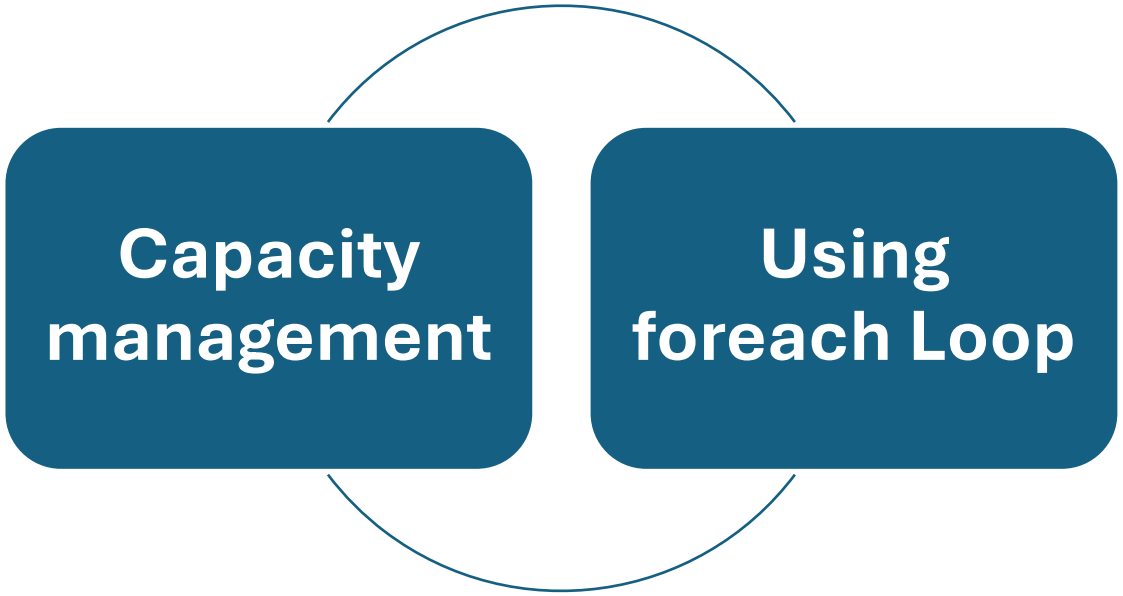
Typical use cases of Lists

DATABASE RECORDS

DATA MANIPULATION

RETURN TYPES OF COLLECTIONS

Best practices for Lists



Difference between Array and List

Aspect	Array	List
Size	Fixed size	Size can adjust dynamically
Performance	Faster	Slight overhead due to dynamic feature
Type	Stores items of the same type	Lists can store items of different types
Memory	Allocate memory when they are created	Only allocates memory as needed

Let's get familiar with Lists -Demo

Declare

Initialize

Add items

Remove items

Initializing a List with Items -Lab

Create a simple C# program that declares a list of integers, initializes it with the values 1, 2, and 3, and prints each value to the console. Loop through the collection and print each number out followed by a space.

The result of execution for the default string should be:

"1 2 3 "

Sort a list

List<T>.Sort() Method is used to sort the elements or a portion of the elements in the List<T>

Quiz

How do you declare a list of integers in C#?

- List<int> myList = new List<int>();
- int[] myList = new int[];
- var myList = new Dictionary<int>();

Quiz

Which method is used to sort a list in C#?

- ○ myList.Order();
- ○ myList.Sort();
- ○ myList.Arrange();

Quiz

How can you add an item to a list in C#?

- ○ myList.Add(10);
- ○ myList.Insert(10);
- ○ myList.Put(10)

Lists – FindAll

List<T>.FindAll(Predicate<T>) Method is used to get all the elements that match the conditions defined by the specified predicate

This method returns a List<T> containing all the elements that match the conditions defined by the specified predicate otherwise it returns an empty List<T>

Lambda expressions

- Lambda expressions in C# are used like anonymous functions, with the difference that in Lambda expressions you don't need to specify the type of the value that you input thus making it more flexible to use.
- '=>' is the lambda operator which is used in all lambda expressions.
- '=>' is read as "goes to" or "becomes".
- The Lambda expression is divided into two parts :
  - the left side is the input/parameter
  - right is the expression or statement block.
- Example : **x => x * x**

  This reads as : Take an input x and turn it into x multiplied by x

Lambda expressions

The Lambda Expressions can be of two types:

Expression Lambda:
Consists of the input and the expression.
Syntax:

```
input => expression;
```

Statement Lambda:
Consists of the input and a set of statements to be executed.
Syntax:

```
input => { statements };
```

C# | Predicate Delegate

A Predicate delegate is an in-built generic type **delegate**.

Lambda expressions using Any() method of List : Demo

- Any() : Gives bool results if expression satisfies a condition

Using the Any Method on a List

Create a list of integers with values { 1, 6, 3 }. Use the Any method to check if any element in the list is greater than 5, and print "True" or "False" based on the result.

The result of execution for the default list should be:

"True"

List with complex objects - Demo

- Create list of a class
- Add values to the list
- Use Where() method
- System.Linq

Quiz

What does the Where method in LINQ do?

- It sorts the elements of a list
- It filters the elements of a list based on a condition
- It adds a new element to the list

Quiz

How do you use LINQ to find elements in a list greater than 5?

- myList.Where(x=>x>5);
- myList.Select(x=>x>5);
- myList.Filter(x=> x>5);

Quiz

Which namespace must be included to use LINQ methods like <u>Where</u>?

- System.Linq
- System.Collections.Generic
- System.Text

Legacy ArrayList

- It represents an ordered collection of an object that can be indexed individually.
- It is basically an alternative to an array.
- However, unlike array you can add and remove items from a list at a specified position using an **index** and the array resizes itself automatically.
- It also allows dynamic memory allocation, adding, searching and sorting items in the list.

Nullable types

Allows to assign a null value to a variable that is known as the Nullable type.

Syntax :

Nullable<data_type> variable_name = null;

or

datatype? variable_name = null;

How to access the value of Nullable type variables?

HasValue()

- Checks If the variable has value or it is null

GetValueOrDefault()

- Get an original assigned value if it is not null. You will get the default value if it is null.
- The default value for null will be 0.

Usage in Real scenarios

Nullable types are particularly useful in scenarios where a variable may not be applicable Or where values are missing, for example :

Database interactions :

- Database fields may return null if they are not set
- Nullable types are perfect handing such case

Form Data :

- When dealing with forms where users may not provide all fields like optional survey responses
- Nullable types can indicate whether a response was given

Benefits of using Nullable Types

Safety :

- Prevents runtime errors that can occur when dealing with types that may not have values.

Clarity :

- Makes clear in your code that a variable might not always have a value.
- readability and maintainability.

Dictionaries

- A C# dictionary is a generic collection of key-value pairs.
- The keys must be unique, but the values can be duplicated.
- Dictionaries are implemented as hash tables (Legacy concept) so their keys can quickly access them.
- Namespace : **using System.Collections.Generic**

```
Dictionary<int,string> employees = new Dictionary<int,string>();
```

Dictionary methods - Demo

Add	Access	Update	Remove	Iterate through
Add value to the dictionary	Access the data of dictionary	Update the data of dictionary	Remove the data of dictionary	Iterate through dictionary using foreach loop

Dictionary methods - Demo

ContainsKey()	TryAdd()	Get()	TryGetValue()
Check if key exits	Check for key before Adding value	Retrieve a value from a dictionary	Retrieves the value for specified without erroring if key is not present

Dictionary demo

- Use var keyword
- Declare string,string array
- Another way to initialize
- Iterate

Dictionary with complex types - Demo

Complex Object as the Value of a Dictionary -Lab

Create a simple C# program that declares a dictionary with string keys and values of type List<int>, initializes it with one key-value pair, and prints the values of the list for the given key to the console.

The result of execution for the default string should be:

"1 2 3 " (Plus a space after the "3")

Why use this Dictionary instead of an array

- Key-based Access
- Handling Non-Sequential and Sparse Data
- Flexibility in Modifying the Collection
- Uniqueness of Keys
- Descriptive Code and Maintenance
- Performance

Error handling in C#

Exception handling

Keyword	Definition
try	Used to define a try block. This block holds the code that may throw an exception.
catch	Used to define a catch block. This block catches the exception thrown by the try block.
finally	Used to define the finally block. This block holds the default code.
throw	Used to throw an exception manually.

```
try
{
    // Code to try goes here.
}
catch (SomeSpecificException ex)
{
    // Code to handle the exception goes here.
    // Only catch exceptions that you know how to handle.
    // Never catch base class System.Exception without
    // rethrowing it at the end of the catch block.
}
```

```
try
{
    // Code to try goes here.
}
finally
{
    // Code to execute after the try block goes here.
}
```

```
try
{
    // Code to try goes here.
}
catch (SomeSpecificException ex)
{
    // Code to handle the exception goes here.
}
finally
{
    // Code to execute after the try (and possibly catch) blocks
    // goes here.
}
```

try block

- The **try** statement allows you to define a block of code to be tested for errors while it is being executed
- Associated **catch** blocks are used to handle any resulting exceptions.
- A try block requires one or more associated catch blocks, or a finally block, or both.
- We can have, try-catch statement, a try-finally statement, and a try-catch-finally statement.
- A try block without a catch or finally block causes a compiler error.

catch block

- The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.
- catch block can specify the type of exception to catch.
- **Multiple catch blocks are allowed**
- Cannot be without try block

The finally block

The **finally** statement lets you execute code, after try…catch, regardless of the result.

If present, the finally block executes last, after the try block and any matched catch block

A finally block always runs, whether an exception is thrown or a catch block matching the exception type is found.

The Finally Keyword - Lab

Create a simple C# program that demonstrates the use of the finally block. The program should:

1. Declare a method PrintWithFinally which contains a try block that writes "Trying..." to the console.
2. Add a finally block that writes "Finally executed." to the console.
3. Ensure the finally block executes even if the try block does not throw an exception.

The result of execution for the default string should be:

"Trying..." "Finally executed."

Debug Log

- When the program is run, you can use methods of the Debug class to produce messages that help you to monitor the program execution sequence, to detect malfunctions, or to provide performance measurement information.
- By default, the messages that the Debug class produces appear in the Output window of the Visual Studio Integrated Development Environment (IDE).

Throw keyword

- The **throw** statement allows you to create a custom error.
- The throw statement is used together with an exception class.
- There are many exception classes available in C#:
  - ArithmeticException
  - FileNotFoundException
  - IndexOutOfRangeException
  - TimeOutException

Exception work with call stack - Demo

Quiz

What is the primary purpose of a try block in C#?

- To define code that will always execute
- To enclose code that might throw an exception
- To handle exceptions and provide alternate code execution

Quiz

What happens if an exception is thrown and there is no matching catch block?

- ○ The program continues execution normally
- ○ The program terminates and an error message is displayed
- ○ The exception is ignored

Quiz

Which block is sued to handle exceptions that occur in a try block?

- Finally
- Catch
- else

Quiz

Which keyword is used to re-throw an exception in a catch block?

- throw
- rethrow
- catch

Quiz

What is the purpose of the finally block?

- To execute code only if no exceptions were thrown
- To execute code that should run regardless of whether an exception was thrown
- To catch any exceptions not handled by previous catch blocks

Quiz

Which of the following best describes a situation where multiple catch blocks are used?

- To handle different types of exceptions separately.
- To ensure that at least one catch block executes.
- To improve performance by handling exceptions faster.

How to Handle Exceptions Properly

1

Use Specific Exceptions

2

Use Finally for Cleanup

3

Log Exceptions

4

Avoid Exception Handling for Control Flow

Inheritance

Base Class and Derived Class

- In C#, it is possible to inherit fields and methods from one class to another.
- Derived Class (child or sub class) - the class that inherits from another class
- Base Class (parent or super) - the class being inherited from
- To inherit from a class, use the **:** symbol.

```
class derived-class : base-class
{
    // methods and fields
    .
    .
}
```

Type of inheritance

Single Inheritance

Multilevel Inheritance

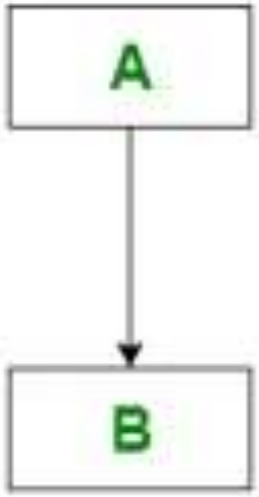
Hierarchical Inheritance

Multiple Inheritance

Single Inheritance

Class inherits from one base class

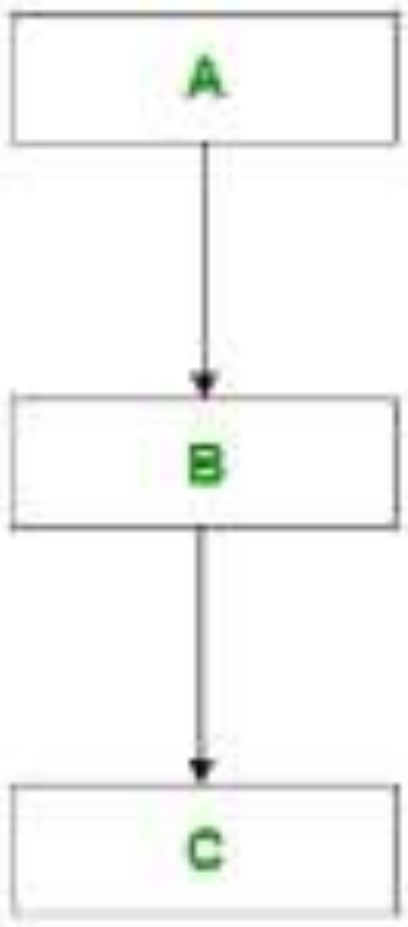
Class DerivedClass: BaseClass {}



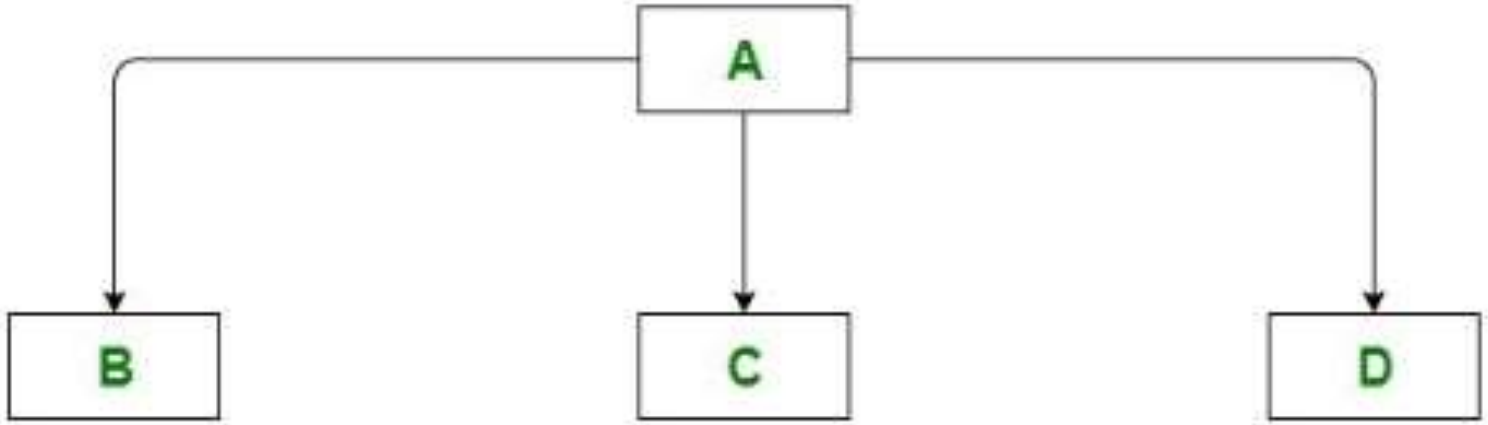
Single Inheritance

Multilevel Inheritance

Class derived from another derived class.



Multilevel Inheritance

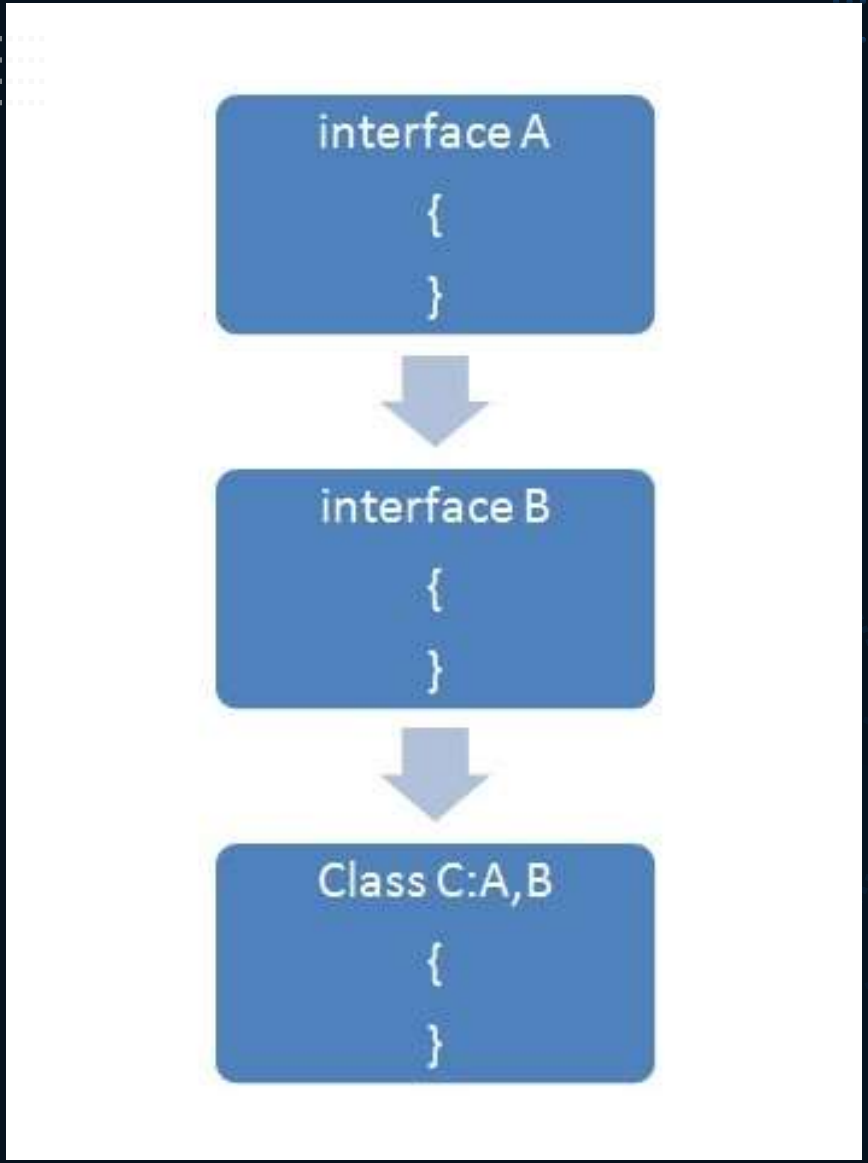


Multiple classes inherit from one base class

Hierarchical Inheritance

Multiple Inheritance

- Not directly supported in C#
- But can be achieved using Interfaces



Access modifiers and Inheritance

- **Public** available to everyone.
- **Private** only available to this class.
- **Protected** available to every deriving class.

Method Overriding

- Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.

- To perform method overriding in C#, you need to use **virtual** keyword with base class method and **override** keyword with derived class method.

3 keywords for Method Overriding

- **virtual keyword:** This modifier or keyword use within base class method. It is used to modify a method in *base class* for *overridden* that particular method in the derived class.

- **override:** This modifier or keyword use with derived class method. It is used to modify a *virtual* or *abstract* method into *derived class* which presents in base class.

- **base Keyword:** This is used to access members of the base class from derived class. It basically used to access constructors and methods or functions of the base class.

Constructor inheritance

- Derived class to inherit the constructor of its base class.
- This means that the derived class can use the constructor of the base class to initialize its own fields and properties.

Constructor Inheritance

Constructors of base class are called before the constructors of the derived class.

It is important because :

- Proper Initialization
- Consistent state
- Reuse of initialization code

Constructor Inheritance with Multiple Derived Classes

All base class constructors will be executed first-Demo

The sealed Keyword

If you don't want other classes to inherit from a class, use the sealed keyword

Quiz

What is method overriding in C#?

- ○ A technique to redefine a base class method in a derived class
- ○ A technique to define multiple methods with the same name but different parameters
- ○ A technique to hide a base class method

Quiz

Which keyword is used to define a method for overriding in the base class?

- virtual
- override
- abstract

Quiz

Which keyword is used to override a method in the derived class?

- override
- virtual
- new

Quiz

What is constructor inheritance in C#?

- The process of calling a base class constructor from a derived class constructor
- The process of inheriting properties and methods from a base class
- The process of creating multiple constructors in a single class

Quiz

How do you call a base class constructor from a derived class constructor?

- ○ Using the base keyword followed by parentheses and arguments if any
- ○ Using this keyword followed by parentheses and arguments if any
- ○ Using the super keyword followed by parentheses and arguments if any

Quiz

Why is constructor inheritance useful?

- To initialize base class members before initializing the derived class members
- To hide the base class constructor
- To avoid writing any constructors in the derived class

Abstraction

- Data **abstraction** is the process of hiding certain details and showing only essential information to the user.
- Abstraction can be achieved with either **abstract classes** or **interfaces**
- The abstract keyword is used for classes and methods.
- **Abstract class**: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method**: can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).
- An abstract class can have both abstract and regular methods.
- Object of the Abstract class cannot be created.

Benefits of Abstraction

1. Reduces Complexity
2. Enhances Code Reusability
3. Improves Flexibility

Difference between Encapsulation and Abstraction

Aspect	Encapsulation	Abstraction
Definition	Hiding the internal state of an object by restricting access to its fields	Hiding the implementation details and showing only essential features.
Purpose	To ensure **data security** and prevent unauthorized access.	To reduce **complexity** and focus on what an object does.
Implementation	Achieved using **access modifiers** (private, public, etc.) and properties	Achieved using **abstract classes** and **interfaces**.
Focus	Focuses on **how** the data is protected	Focuses on **what** the object does.
Visibility	Makes the internal state visible only through controlled access points	Hides implementation logic from the user.

Abstarct Classes

- An abstract class is declared with the help of **abstract** keyword.
- Object of the Abstract class cannot be created.
- Class that contains the abstract keyword with some of its methods(not all abstract method) is known as an **Abstract Class.**
- Class that contains the abstract keyword with all of its methods is known as pure **Abstract Base Class.**
- **Abstract method**: can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).
- You are not allowed to declare the abstract methods outside the abstract class.
- You are not allowed to declare an abstract class as Sealed Class or Static Class

Interface in C#

- An interface defines a contract.
- Interfaces will contain only the declaration of the members. The implementation of the interface's members will be given by class who implements the interface implicitly or explicitly.
- Interfaces can contain properties, methods, events, and indexers but not fields.
- Interfaces specify what a class must do and not how.

Interface in C#

- Interfaces can’t have private members.
- By default, all the members of Interface are public and abstract.
- The interface will always define with the help of keyword ‘**interface**‘.
- Interface cannot contain fields because they represent a particular implementation of data.
- Multiple inheritance is possible with the help of Interfaces but not with classes.

Interface in C#

Syntax for Interface Declaration:

```
interface  <interface_name >
{
    // declare Events
    // declare indexers
    // declare methods
    // declare properties
}
```

Syntax for Implementing Interface:

```
class class_name : interface_name
```

Create a Cat Class that implements IAnimal -Lab

- Create **another implementation of the IAnimal Interface**, a "Cat" class.
- It needs to be able to eat and make sounds. Similar to what we have already done before!

Why use Interfaces?

Abstraction

Polymorphism

Decoupling

Reusability

Testability

Abstract Classes vs Interfaces

Feature	Abstract Class	Interfaces
Inheritance	Single inheritance	Multiple inheritance
Implementation	Can have both abstract and concrete methods	Only method declarations (no implementation)
Access Modifiers	Supports access modifiers	Does not support access modifiers

Polymorphism

Polymorphism derived from Greek meaning “many forms” refers to the ability in programing where a single interface or method can operate in multiple ways based on the object it interacts with.

One Interface, Many implementations :

A single function or method can handle objects of various types. The implementation depends on the specific object being referenced.

Why Polymorphism

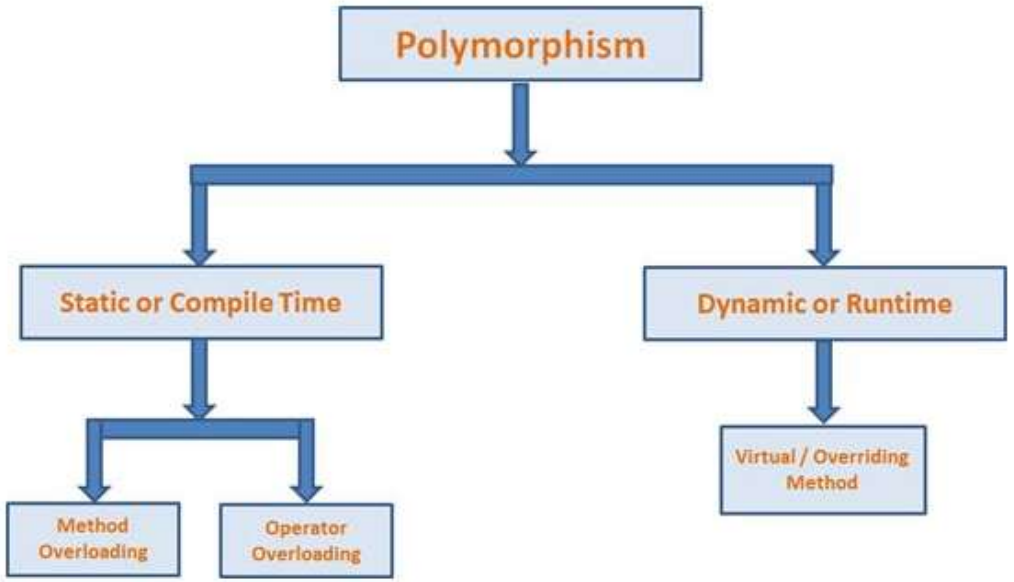
FLEXIBILITY

CODE MAINTENANCE

SIMPLIFIES CODE

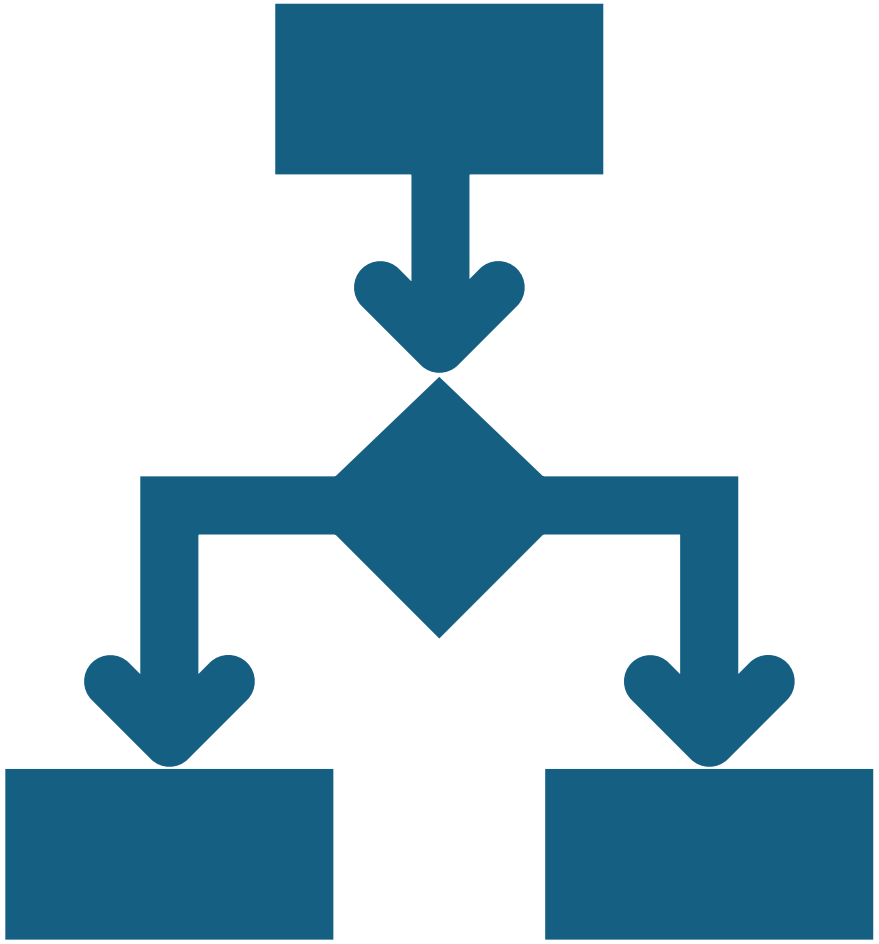
Types of Polymorphism

- Dynamic / Runtime Polymorphism
- Static / Compile Time Polymorphism



Dynamic / Runtime Polymorphism

- Late binding.
- Here, the method name and the method signature (the number of parameters and parameter type must be the same and may have a different implementation).
- Method overriding is an example of dynamic polymorphism.



Static or compile-time polymorphism

- Method overloading is an example of Static polymorphism
- The method/function has the same name but different signatures in overloading.
- It is also known as Early binding.
- It is also known as compile-time polymorphism because the decision of which method is to be called is made at compile time.

Compile-time Polymorphism- Demo

Quiz

What is polymorphism in C#?

- The ability of different classes to be treated as instances if the same class through inheritance.
- The ability to define multiple methods with same name but different parameters
- The ability to hide a base class method

Quiz

Which keyword is used to override a method in a derived class?

- ○ override
- ○ virtual
- ○ new

Quiz

How can polymorphism be achieved in C#?

- By using inheritance and interfaces
- By using only inheritance
- By using only interfaces

Decoupling code and Enhancing Testability - Demo

Dependency Injection

Dependency Injection (DI)

- Dependency Injection (DI) is a **software design pattern** that helps developers build better software.
- It allows us to develop **loosely-coupled** code that is easy to maintain.
- Dependency Injection reduces the hard-coded dependencies among your classes by injecting those dependencies at run time instead of design time technically.
- Makes a **class independent of its dependencies**.
- Helps follow **SOLID's** dependency inversion and single responsibility principles.

Types of DI

- Constructor Injection
- Property/Setter Injection
- Method Injection

Constructor Injection

Dependencies are **provided through a class constructor**, ensuring that the class receives all its dependencies at the time of instantiation.

Setter Injection

Dependencies are assigned **to public setter methods**, allowing for the **injection of dependencies after the object is created.**

Injection using Interfaces - Demo

Method Injection

Dependencies are only **passed in the method.**

Quiz

What is Constructor Injection in C#?

- A method of providing dependencies through a class constructor at instantiation
- A method of assigning dependencies via public setter methods
- A method of providing dependencies through an interface methods

Quiz

What is a major advantage of Constructor Injection?

- ○ Ensures all dependencies are provided at object creation time
- ○ Allows changing dependencies after object creation
- ○ Requires additional interfaces to be defined for setting dependencies

Quiz

Which of the following is true about Setter Injection?

- ○ Ensures all dependencies are provided at object creation time
- ○ Dependencies might not be set at object creation time, leading to potential null references
- ○ Provides a clear contract for injecting dependencies through interfaces

Advantages of Dependency Injection in C#

- Improved Testability
- Enhanced Flexibility
- Decoupling of Components
- Simplified Object generation
- Support for Multiple Configurations

Multiple inheritance using Interface

A class can inherit two Interfaces.

Structs in C#

What are structs?

Structure is a value type data type.

It helps you to make a single variable hold related data of various data types.

The **struct** keyword is used for creating a structure

Features of Structures

- Value Type
- Doesn’t support Inheritance
- Cannot have a Parameterless Constructor
- No Garbage Collection for Value Types
- Supports Interfaces

Classes vs Structs

Feature	Classes	Structures
Memory	Reference types	Value types
Support inheritance	Yes	No
Default constructor	Yes	Yes

Defining a Structure

To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member for your program.

```
Access_Modifier struct structure_name
{

   // Fields
   // Parameterized constructor
   // Constants
   // Properties
   // Indexers
   // Events
   // Methods etc.

}
```

Quiz

What is a struct in C#?

- ○ A value type that can contain data and methods
- ○ A reference type that can contain data and methods
- ○ A class that encapsulates data

Quiz

How is a struct different from a class in C#?

- Structs can inherit from other structs
- Structs are value types and classes are reference types
- Structs can have destructors

Quiz

When should you use a struct instead of a class in C#?

- When you need inheritance and polymorphism
- When you need to handle large amounts of data
- When you need a small, lightweight object

When to Use Structures?

Use structs when:

- You need a small, simple type with fixed data.
- Performance is critical, and you want to avoid the overhead of heap allocation.
- You don’t need inheritance or polymorphism

When NOT to Use Structures?

Avoid structs if:

- Your object has a lot of fields (large size can cause performance issues).
- You need advanced object-oriented features like inheritance.
- You frequently need to modify the object.

```
enum Enum_variable
{
    string_1...;
    string_2...;
    .
    .
}
```

Enums

Enumeration (or enum) is a value data type in C#.

An **enum** is a special "class" that represents a group of constants (unchangeable/read-only variables).

Enumeration is declared using ***enum*** keyword directly inside a namespace, class, or structure.

DateTime

- DateTime is a structure of value Type like int, double etc
- It is available in System namespace
- It implements interfaces like IComparable, IFormattable, IConvertible, ISerializable, IComparable, IEquatable.

Generics

- Generics are a way to make your code more flexible and reusable by allowing it to work with any data type.
- Think of generics as templates that you can fill in with different types when you use them.

Generic Class

- The Generic class can be defined by putting the <T> sign after the class name.
- The System.Collection.Generic namespace also defines a number of classes that implement many of these key interfaces.
- Examples : Collection<T>, Dictionary<TKey, TValue>, List<T>

Why Generics ?

Reusability

Type Safety

Performance

Delegates

- A delegate is a type that represents references to methods with a specific parameter list and return type.
- A **delegate** is a reference type variable.
- The reference can be changed at runtime.
- Delegates are especially used for implementing events and the call-back methods.
- All delegates are implicitly derived from the **System.Delegate** class.

Why Delegates?

Delegates provide a way to **pass methods as parameters**, enabling **flexible and extensible code designs**.

When and Where to use Delegates?

When?

Use delegates when you need a way to **encapsulate a method and pass it as an argument**.

Where?

Delegates are used in **callback mechanisms, event handling and designing flexible APIs**

Declaration of Delegates

- Delegate type can be declared using the **delegate** keyword.
- Once a delegate is declared, delegate instance will refer and call those methods whose return type and parameter-list matches with the delegate declaration.

```
[modifier] delegate [return_type] [delegate_name] ([parameter_list]);
```

Instantiation & Invocation of Delegates

A delegate object is created with the help of **new** keyword.

```
[delegate_name]  [instance_name] = new [delegate_name](calling_method_name);
```

Multicast delegates

- Multicast delegates are delegates that hold references to more than one method.
- When they are invoked, they call all the methods they reference in the order they were added.
- All methods are called in First in First Out(FIFO) order.
- ‘+’ or ‘+=’ Operator is used to add the methods to delegates.
- ‘–’ or ‘-=’ Operator is used to remove the methods from the delegates list.

When to use Multicast Delegates?

You would use a multicast delegate when you need to **notify multiple event handlers** or **execute multiple related methods** in response to a **single event or operation.**

Quiz

What is a multicast delegate in C#?

- ○ A delegate that holds reference to multiple methods
- ○ A delegate that can only call one method at a time
- ○ A delegate that can only be used with events

Quiz

What happens if a multicast delegate is invoked?

- Only the first method in the delegate's invocation list is called
- The methods are called randomly
- All methods in the delegate's invocation list are called in sequence

Quiz

How can you remove a method from a multicast delegate?

- Using the += operator
- Using the -= operator
- Using the = operator

Events

An Events **lets one class tell others when something important happens**. It uses a special method called a **delegate**

This means one part of the program can alert others without needing direct connections.

Why use an Event?

Event allows a class to **send updates without knowing who gets them**. This makes the system more flexible and organized.

It helps different parts of the program work together without being tightly coupled

Publisher-Subscriber

- A **publisher** is an object that contains the definition of the event and the delegate. The event-delegate association is also defined in this object. A publisher class object invokes the event, and it is notified to other objects.
- A **subscriber** is an object that accepts the event and provides an event handler. The delegate in the publisher class invokes the method (event handler) of the subscriber class.

Where to use an Event?

Events are common in **logging, monitoring, data changes and button clicks**. They are used **wherever notifications are needed**.

Any situation where one action triggers other responses can benefit from events.

Events

Event Args :

‘EventArgs’ is a class that provides data related to an event.

Safe Event Invocation :

Ensures safety invocation of events , when there is no event to be invoked.

Unsubscribing from Events

Unsubscribe from events when they are no longer necessary

Quiz

What is an event in C#?

- A method that is called periodically
- A class that encapsulates data
- A way for a class to notify other classes or objects when something happens

Quiz

How do you subscribe to an event in C#?

- By using the +=operator followed by the event handler method
- By using the =operator followed by the event handler method
- By calling the event like a method

Quiz

What is the purpose of the EventArgs class?

- To provide a base class for classes containing event data
- To handle exceptions
- To format output strings

Regular Expressions

- **Regex**, short for regular expression, is a sequence of characters that **defines a search pattern.**
- It is a versatile tool used for **pattern matching and manipulation of text.**
- With a regular expression, you can specify rules or patterns to search for specific sequences of characters within a larger text.
- These patterns can include literal characters, metacharacters, character classes, quantifiers, and more.

Why Use regular Expressions?

- **Pattern matching:** Finding and extracting specific sequences of characters in text.
- **String validation**: Checking if a string conforms to a specific pattern or format.
- **Data extraction**: Parsing and extracting data from structured text, such as log files or web pages.
- **Text manipulation:** Replacing or modifying specific patterns in a string.

Quiz

What are regular expressions (regex)?

- ○ A sequence of characters that define a search pattern
- ○ A programming language
- ○ A type of database

Quiz

How are regex patterns used in data validation?

- By creating databases
- By ensuring that use-entered data adheres to the required format, such as checking if an email address or URL is valid
- By generating random data

Quiz

Which of the following is a use case for regular expressions?

- ○ Formatting documents
- ○ Ensuring that an email address or URL is valid
- ○ Compiling code

Regex Demo

Null-Coalescing Operator in C#

- **?? operator** is known as Null-coalescing operator
- It will return the value of its left-hand operand if it is not null. If it is null, then it will evaluate the right-hand operand and returns its result.
- Or if the left-hand operand evaluates to non-null, then it does not evaluate its right-hand operand.
- **Syntax : p ??q**

Extension Methods

Allow us to add methods to an existing class without

- Changing its source code or
- Creating a new class that inherits from it
- C# extension method is a static method of a static class, where the "this" modifier is applied to the first parameter. The type of the first parameter will be the type that is extended.

Extension methods are only in scope when you explicitly import the namespace into your source code with a using directive.

Dynamic keyword

- The type of the variable is determined at **runtime**.
- The dynamic type variable is created using **dynamic keyword.**
- Operations on dynamic variables are resolved at runtime, allowing for more flexibility but potentially sacrificing performance.
- Useful for interoperability with dynamic languages like Python or JavaScript.
- IntelliSense is limited for dynamic variables, as the compiler cannot determine the exact type at compile time.

Threads

- A **thread** is defined as the execution path of a program
- Each thread defines a unique flow of control
- **System.Threading** namespace offers classes that allow you to manipulate threads.
- Threads make it possible to execute several program pieces concurrently, enhancing the program's efficiency.

Multithreading

Use multithreading when your application has tasks that can run **concurrently, independently, and without needing to be sequentially organized.**

How to Use Threads?

Threads are created by extending the Thread class.

The extended Thread class then calls the **Start()** method to begin the child thread execution.

Why Use Threads?

IMPROVED PERFORMANCE

ENHANCED USER EXPERIENCE

ASYNCHRONOUS PROGRAMMING

Threading Best Practices

Use Asynchronous programing : async and await

Thread Safety: When multiple threads access shared resources, you need to ensure thread safety to avoid race conditions and data corruption. Use synchronization mechanisms like locks to coordinate access to shared resources.

Thread Pool: The ThreadPool class provides a pool of reusable threads to efficiently manage thread creation and destruction.

Task Parallel Library (TPL): TPL simplifies parallel programming by offering high-level constructs like Parallel.For and Parallel.ForEach.

Asynchronous Programing

- Asynchronous programming indicates that a method can execute without waiting for another method to complete.
- Using **async and await**, we can run the programs parallelly.

Async and Await

- The **async** modifier is used to specify that a method/lambda expression is asynchronous.
- The **await** operator suspends evaluation of the sync method until the asynchronous operation completes.

Threads vs Asynchronous programming

Threads	Asynchronous programming
Multiple execution paths running concurrently.	A single thread of execution that switches between tasks.
Often used for CPU-bound tasks (tasks that require a lot of processing power).	Primarily used for I/O-bound tasks (tasks that involve waiting for input or output, like network requests or file operations).
Can be more complex to implement and manage, especially when dealing with shared resources.	Generally easier to implement and manage than threads.

Configuration Files in C#

Configuration files are essential for storing application settings, connection strings, and other critical information outside of the code itself. C# supports various methods to manage configuration data:

- App.config
- Web.config
- JSON Configuration

Add App.Config File

1

In **Solution Explorer**, right-click the project node, and then select **Add** > **New Item**. (The **Add New Item** dialog box appears.)

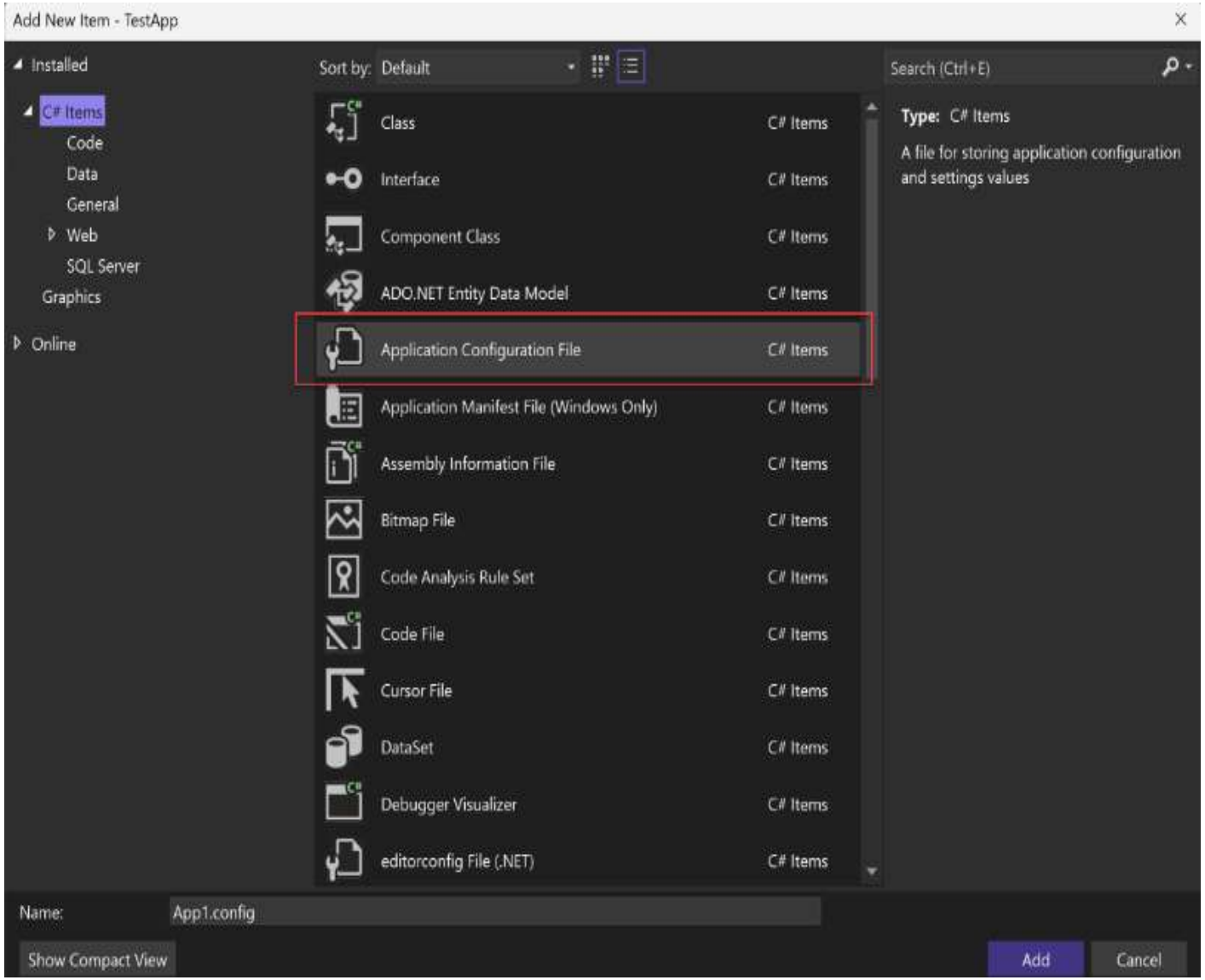
2

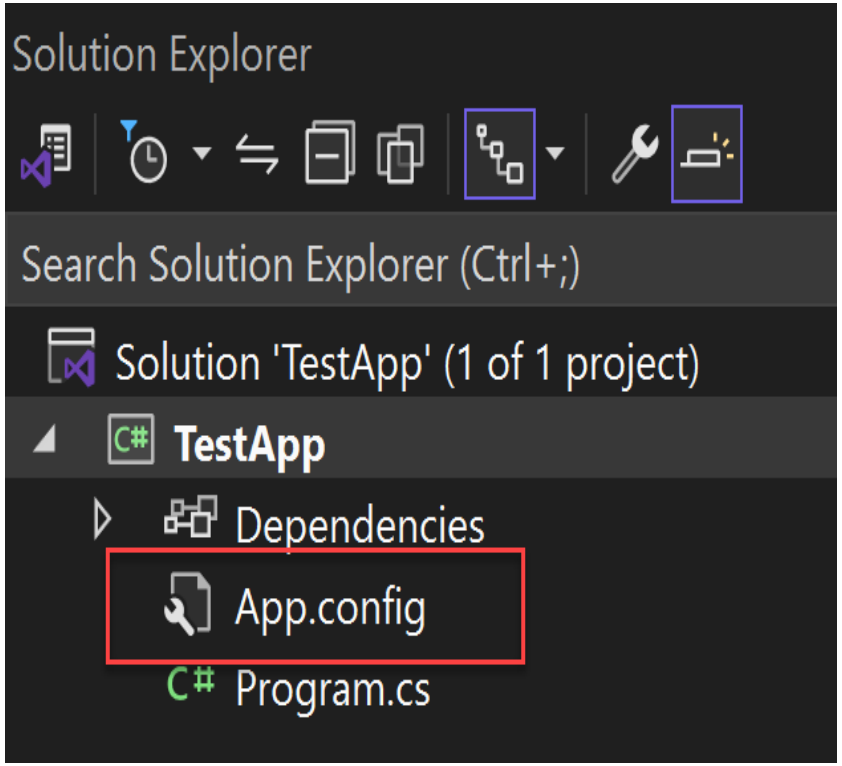
In the middle pane, select the **Application Configuration File** template.

3

Select the **Add** button.

A file named *App.config* is added to your project.


Add New Item - TestApp
Installed
C# Items
Code
Data
General
Web
SQL Server
Graphics
Online
Sort by: Default
Search (Ctrl+E)
Class
Interface
Component Class
ADO.NET Entity Data Model
Application Configuration File
Application Manifest File (Windows Only)
Assembly Information File
Bitmap File
Code Analysis Rule Set
Code File
Cursor File
DataSet
Debugger Visualizer
editorconfig File (.NET)
C# Items
Type: C# Items
A file for storing application configuration and settings values
Name:
App1.config
Show Compact View
Add
Cancel


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'TestApp' (1 of 1 project)
TestApp
Dependencies
App.config
Program.cs

Inside Config file

- **AppSettings** : AppSettings provide an easy way to access string values, based on a certain key.
- **ConnectionStrings** : The connectionStrings section is the place to put your connection strings to any databases you want to access
- **Custom Configuration Sections** <configSections></configSections>
- **applicationSettings and userSettings Section**

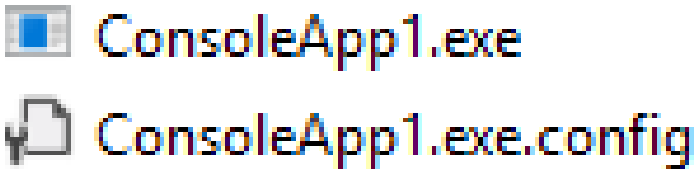
  <userSettings></userSettings>

  <applicationSettings></applicationSettings>

Compile Config file

What Happens When We Compile?

- When you compile your application, the compiler actually copies the app.config file to the output folder but gives it another name.



Using Databases with C#

ADO.NET : This low-level approach gives you granular control over database operations. You'll need to manually handle connection strings, SQL queries, and data retrieval.

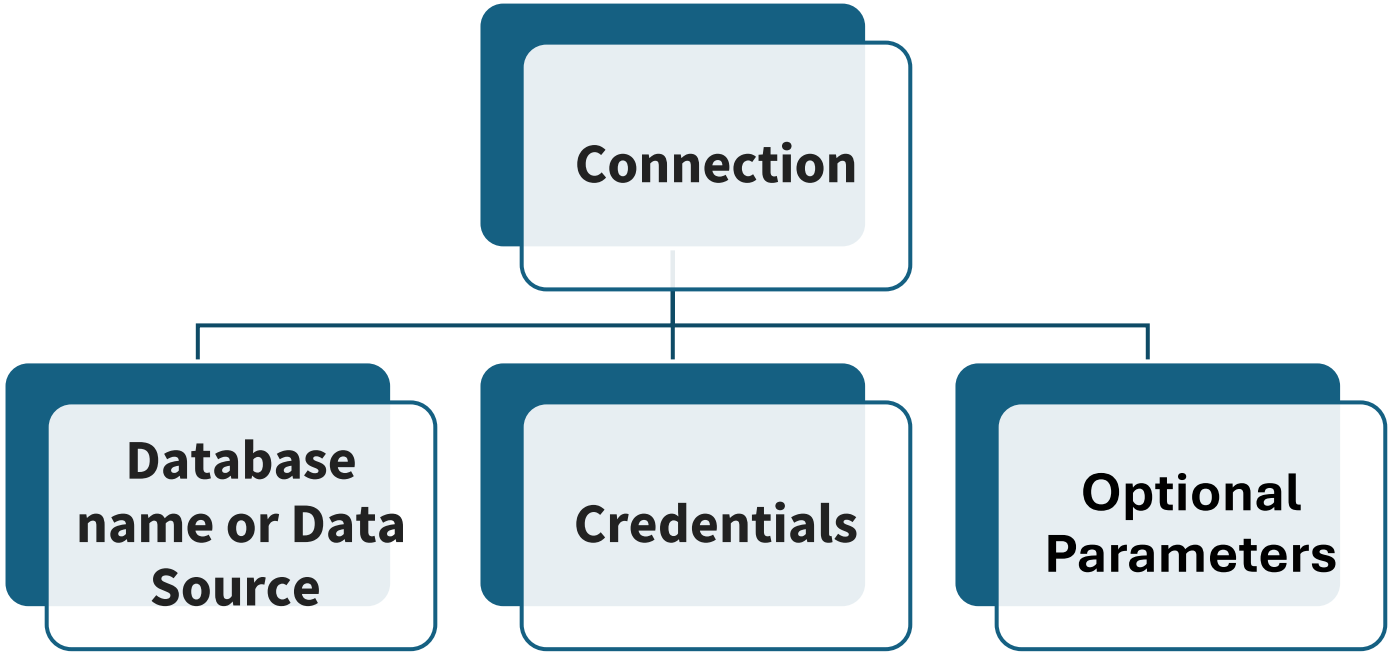
Entity Framework (EF) : Object-Relational Mapping (ORM): EF maps C# classes to database tables, simplifying database interactions. It handles SQL generation, query execution, and data mapping automatically.

Dapper : Micro-ORM: A lightweight ORM that provides a simple and efficient way to execute SQL queries and map results to C# objects.

Other ORMs and Data Access Libraries :

- Nhibernate
- Dapper Extensions
- Massive

Create SQL Connection



Create SQL Connection

- The **SqlConnection** class makes a connection with the database.
- **System.Data.SqlClient** namespace
- Two important methods of SqlConnection.

  Open() : The open() method is used to open the Database connection.

  Close() : The close() method is used to close the Database connection.

SqlCommand

The SqlCommand class is used to **execute the SQL statements.**

SqlCommand uses SqlConnection work with that database.

Selecting data from the database

- C# can execute ‘SQL’ select command against the database.
- The ‘SQL-SELECT’ statement can be used to fetch data from a specific table in the database.
- SqlDataReader : Used to read a row in table one by one.
- To create a SqlDataReader, you must call the ExecuteReader method of .the SqlCommand object

Inserting data into the database

C# can also be used to insert records into the database.

Values can be specified in C# for each row that needs to be inserted into the database.

Updating data into the database

- C# can also be used to update existing records into the database.
- New values can be specified in C# for each row that needs to be updated into the database.

Deleting data from a database

- C# can also be used to delete records into the database.
- Delete commands to specify which rows need to be deleted can be specified in C#.

LINQ

- Language Integrated Query (LINQ) is a powerful set of technologies in .NET that allows you to query and manipulate data from various sources using a consistent syntax.
- It bridges the gap between object-oriented programming and declarative data querying.
- *System.Linq* namespace

Three Parts of a Query Operation

1

Obtain the data source.

2

Create the query.

3

Execute the query.

```
//The data source
string[] names = new string[] { "Vishal", "Beth", "Priyanka" };

//Query creation
var query :IOrderedEnumerable<string> = from name :string in names
    orderby name ascending
        select name;

//Query execution
foreach (var i :string? in query)
{
    Console.WriteLine(i); //Output will be Vishal,Beth,Priyanka
}
```

Print out the entries of a string array sorted by name.

LINQ Example

Basic LINQ Syntax

from clause: Specifies the data source.

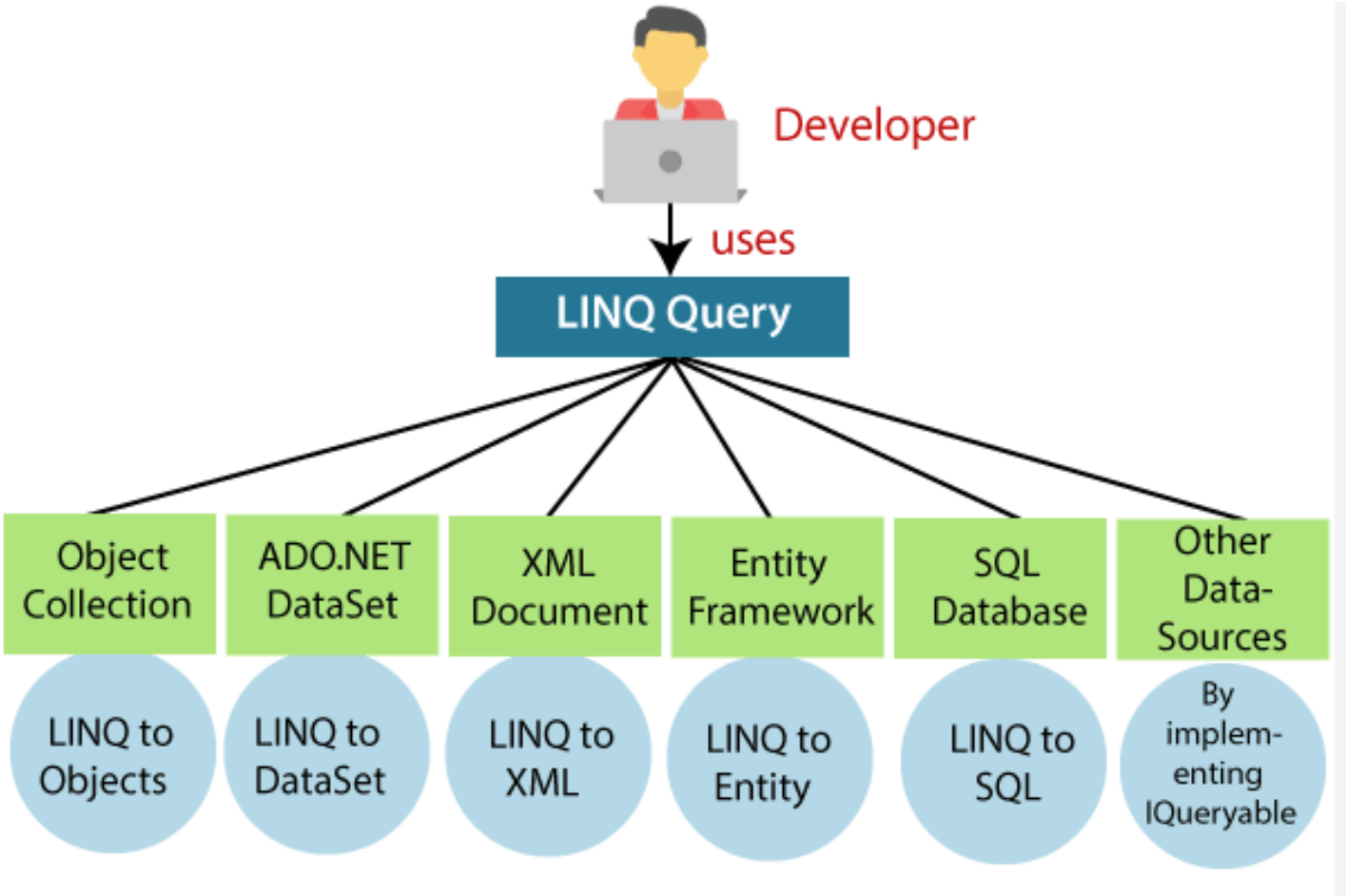
where clause: Filters the data based on a condition.

select clause: Projects the desired results.

orderby clause: Sorts the results.

join clause: Joins data from multiple sources.

group clause: Groups data based on a key.



LINQ DataSources

LINQ DEMO – Print Odd numbers

LINQ To Object - Demo

- Linq to objects
- Sorting
- Filtering
- Where
- Join
- Select
- From
- OrderBy
- Reverse
- Descending

Program.cs

LINQ DEMO

Create Collections based on other collection

LINQ to XML - DEMO

- LINQ to XML is used to query XML data sources.
- Use System.Xml.Linq

LINQ to ADO.NET

- LINQ to ADO.NET deals with data from external sources, basically anything ADO.NET can connect to.
- Any class that implements IEnumerable<T> or IQueryable<T> (in the System.Query namespace) can be queried.

- **LINQ to SQL** (DLinq) {Queries performed against the relation database only Microsoft SQL Server Supported}
- **LINQ to DataSet** {Supports queries by using ADO.NET data sets and data tables}
- **LINQ to Entities**

Benefits of using LINQ

Improved Readability: LINQ queries are written in a concise and expressive syntax, making them easier to read and understand.

Code Reusability: LINQ queries can be easily reused and adapted to different data sources.

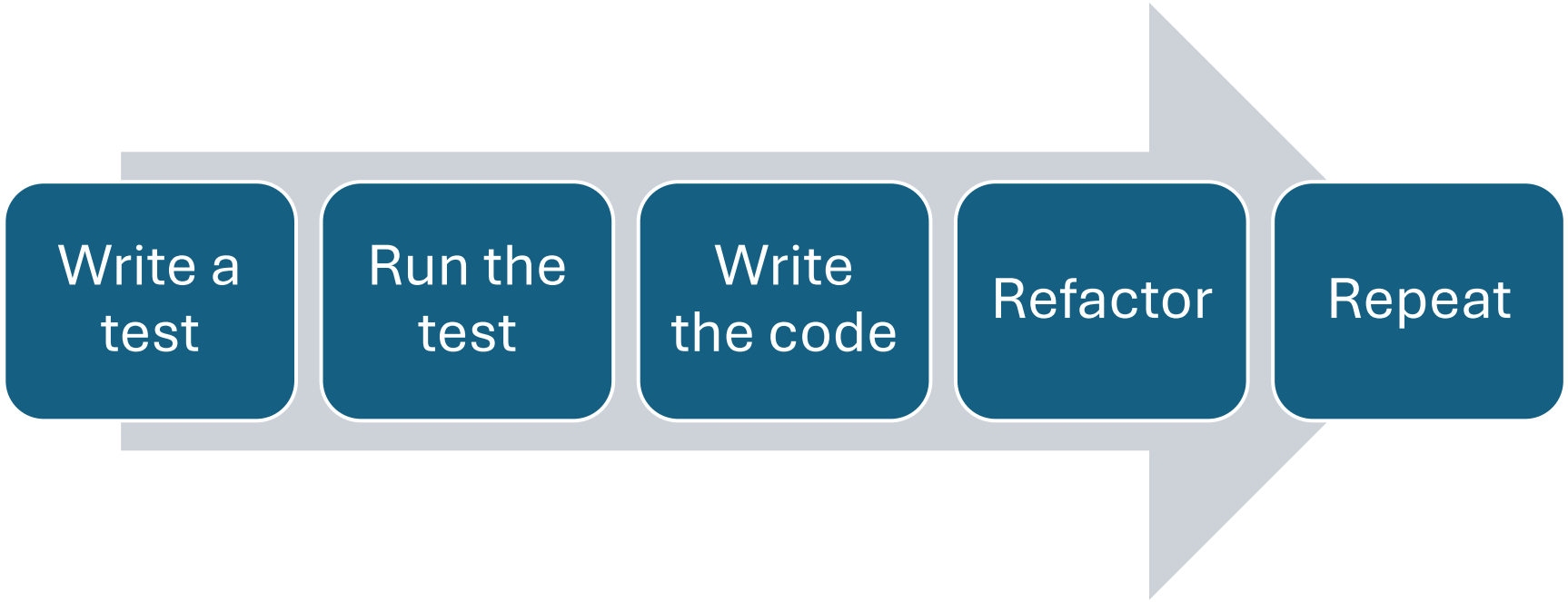
Strong Type Safety: LINQ leverages the power of the .NET type system, providing strong type checking and preventing many common errors.

Enhanced Productivity: LINQ simplifies common data manipulation tasks, leading to increased developer productivity.

What is Unit Testing?

- Unit testing involves **testing individual units of code** (e.g., classes, methods, functions) in isolation.
- The goal is to verify that each unit of code behaves as expected under different conditions.

What is Test-Driven Development (TDD)?



TDD is a development methodology where you write unit tests before writing the actual code.

Benefits of Unit Testing & TDD

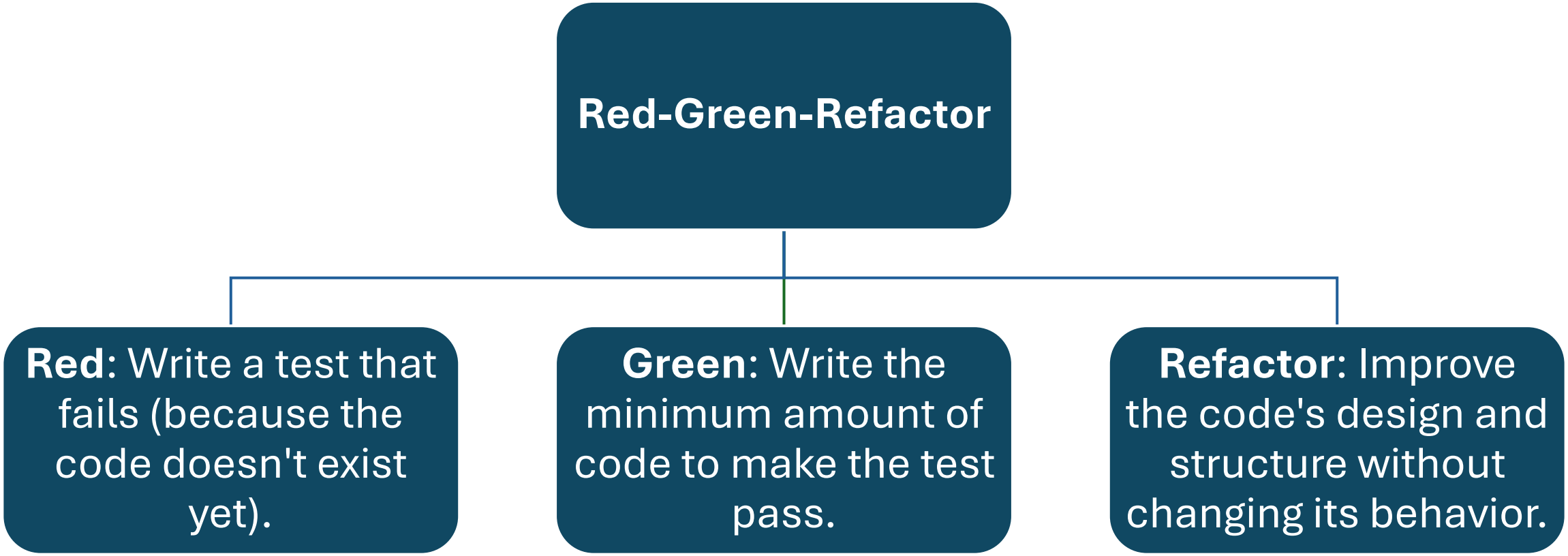
EARLY BUG DETECTION

IMPROVED CODE QUALITY

INCREASED CONFIDENCE

BETTER DOCUMENTATION

TDD in Practice



Setting up a Unit Testing Environment

1. Choose a Testing Framework :
    - xUnit.net
    - MSTest
    - Nunit
2. Install the Testing Framework
3. Tools :

- Visual Studio Test Explorer: Provides a user interface for running and debugging tests.
- Code Coverage Tools: Measure the percentage of code that is covered by unit tests.

Writing Unit Tests

1

Arrange: Set up the necessary objects and data for the test.

2

Act: Execute the method or code under test.

3

Assert: Verify the expected outcome using assertions provided by the testing framework (e.g., Assert.Equal(), Assert.True(), Assert.Throws()).

SOLID PRINCIPLES

Single Responsibility Principle

Open-Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

Single Responsibility Principle (SRP)

A class should have only one reason to change.

Open-Closed Principle (OCP)

Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

Liskov Substitution Principle (LSP)

Subtypes must be substitutable for their base types.

Interface Segregation Principle (ISP)

Clients should not be forced to depend on methods they do not use.

Dependency Inversion Principle (DIP)

High-level modules should not depend on low-level modules. Both should depend on abstractions.

Clean code

- Better code quality
- Improved maintainability
- Enhanced reusability
- Reduced complexity
- Fewer bugs

Clean code

- Meaningful Names
- Keep Methods Short and Focused
- Proper Comments
- Consistent Formatting
- Avoid Magic Numbers/Hardcoding
- Use Meaningful Data Structures
- Write Unit Tests
- Use Design Patterns and Design principles
- Code Reviews
- Refactor Regularly