

# INTRODUCTION TO .NET FRAMEWORK



# Objectives

In this module You will learn

- .NET Framework
- .NET Framework Architecture
- Common Language Runtime
- CLR Services
- Framework Class Library
- Common Type System
- Languages supported by .NET



# .NET Platform

A development platform for building applications for Windows, Windows Phone and Windows Server



# .NET Framework

The .NET Framework is a technology that supports building and running the next generation of applications and XML Web Services



# .NET Framework Architecture

## What's new in .NET Framework 4.5?

**Windows Presentation Foundation**

- Built-in Ribbon controls ★
- Databinding improvements
- Ability to add breakpoints to databindings
- Data source change aware views (Live Shaping) ★
- Validation improvements
- Improved legacy UI integration
- Dispatcher improvements
- Speed-up of large data sets

**Windows 8 support ★**

- Support for Windows Runtime (WinRT)
- .NET Profile for Metro-style apps
- Improved support for sharing DLLs between .NET profiles

**ASP.NET**

- ASP.NET Web API - a new framework for REST endpoints
- Support for implementing WebSocket receivers
- Built-in JavaScript + CSS combining and minification
- Asynchronous pipeline support (Response, Request, HttpHandlers)
- ASP.NET MVC 4
- Async controllers
- Built-in mobile templates → jQuery Mobile support
- Alternate views (e.g. print version, mobile site)
- Support for Recipes: Intelligent codegen
- Did you know? ASP.NET Web Pages is yet another way to write the web pages. Web Pages are MVC-like, but they have their own Model View and are typically developed with WebMatrix.
- ASP.NET Web Pages 2
- New site templates
- Versatile validation support
- Support for OAuth and OpenID ★
- HTML5 support ★
- Performance improvements: Multicore JIT, 35 % faster startup, memory optimizations, assembly sharing between sites, pre-fetch support
- Request validation improvements
- Web Forms
- Strongly typed data binding
- MVC-like support for Models ★
- HTML encoded binding expressions
- HTML5 support ★
- Control support for new semantic elements
- Validator and UpdatePanel now support new HTML5 elements
- Tooling
- Markup editors improved (a lot)
- IIS Express used by default
- Page Inspector
- New templates and snippets

**Windows Communication Foundation**

- New channels (UDP multicast, WebSockets, ...) ★
- Simplified configuration, VS config validation
- Support for contract-first development
- Asynchronous operations
- Streaming improvements

**Windows Workflow Foundation**

- C# Expressions
- State machine workflows are back! ★
- Workflow versioning
- Code-first activity design
- Faster execution
- Designer usability improvements

**Managed Extensibility Framework 2.0**

- Support for generic parts ★
- Debugging improvements
- Support for explicit and convention-based bindings between objects ★
- Support for binding POCOs; no more attribute requirements

**ADO.NET**

- Sparse columns support improved (SQL Server)
- Passwords are now stored encrypted
- Asynchronous operations

**SQL Express LocalDB**

- New light version of SQL Express for developer use. Supported in .NET 4.5, separate patch for 4.0 is coming

**SQL Server 2012 ("Denali") Support**

- High Availability support on connection-string level
- Fast failover across multiple subnets
- Support for new spatial data types (polygons, arcs etc.) ★

**Entity Framework 4.5**

- Enumeration support
- Migrations for schema changes ★
- Designer improvements
- Spatial data type support ★
- Table-valued function support ★
- Multi-result sproc support

**Base Class Library**

- Networking improvements (IPv6 enhanced, IDN, EAI etc.)
- Key interfaces (e.g. file IO) now support async
- New ArraySegment and ReadOnlyDictionary classes
- Support for CLR objects over 2 GB in size
- Resource file management performance improved
- Unicode improvements (v6, console support)
- Background JIT on multicore platforms

**Task Parallel Library**

- Task thread controls improved: Task.WaitAll/WaitAny, various timeout primitives available
- TPL Dataflow: Tools for parallel data flow processing

**C# 5.0**

- Support for async programming: async and await keywords
- Methods can access call site info as parameters (CallerInfo)

**Visual Basic 11**

- Iterator implementations (Yield)
- Async and Await equal to C#
- "Global" keyword for namespace referencing
- Call Hierarchy view available

**F# 3.0**

- Type providers
- Query expressions (LINQ)
- Auto-implemented properties
- CallerInfo attributes (as in C#)

**Visual C++ 11**

- C++11 standard support improved
- Auto-vectorization and parallelization of loops
- GPU-driven processing (C++ AMP)
- Intellisense for C++/CLI

**Legend:**

- = Asynchrony support
- = Performance improvement
- ★ = Significant new feature

**Microsoft tech·days Helsinki | 2012**

**SANKO**  
Suomen Aktiivisten .NET-kehittäjien Kerho  
[www.facebook.com/sanko.net](http://www.facebook.com/sanko.net)

**offbeat solutions**  
[www.offbeat.fi](http://www.offbeat.fi)

# .NET Framework Architecture

The .NET Framework consists of the Common Language Runtime and the .NET Framework class Library

The Common Language runtime is the foundation of the .NET Framework

The class library is a comprehensive, object-oriented collection of reusable types

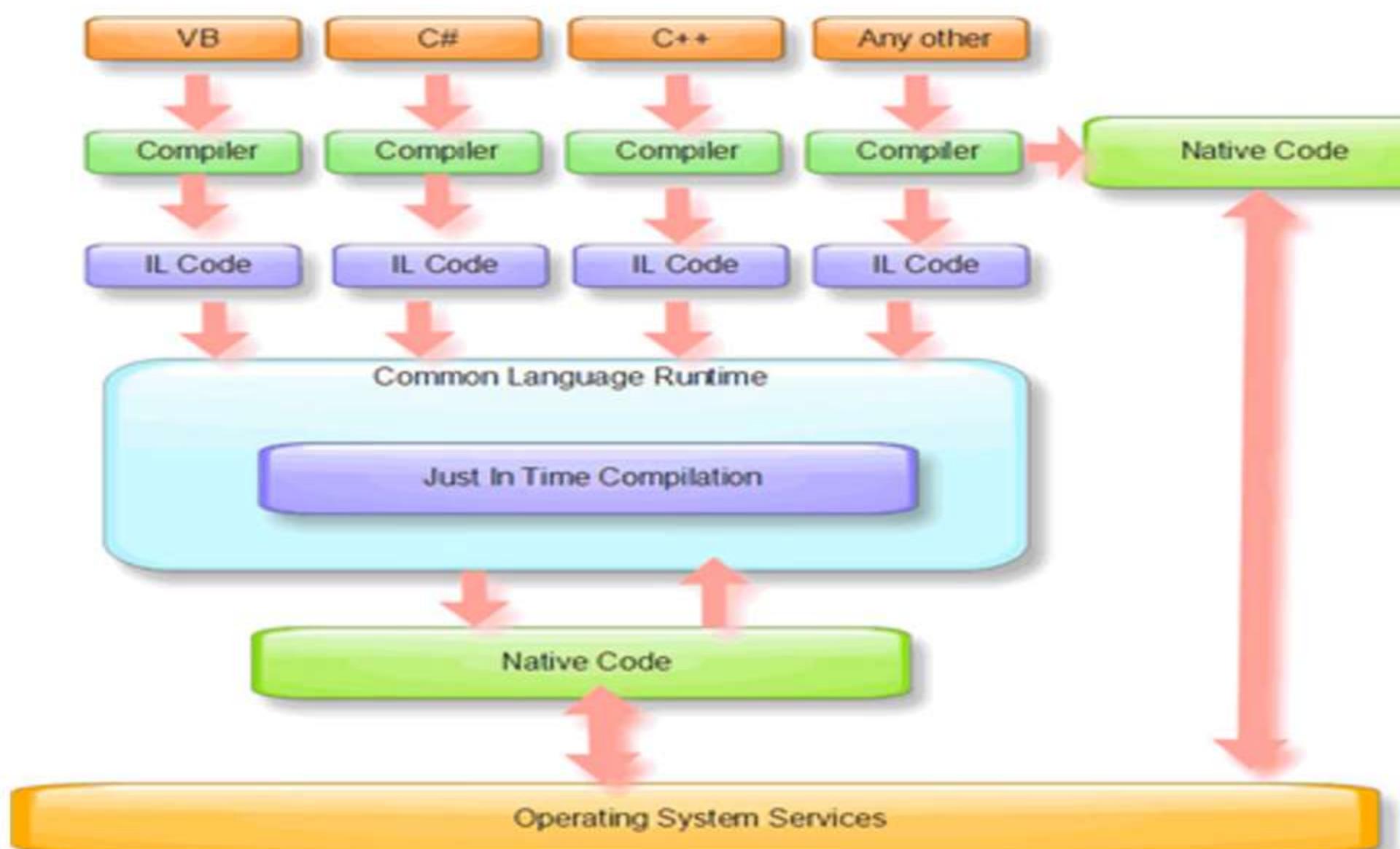
Using the class library you can develop applications ranging from

- Traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.

# Common Language Runtime (CLR)

The .NET Framework provides a run-time environment called the common language runtime

CLR runs the code and provides various services that make the development process easier

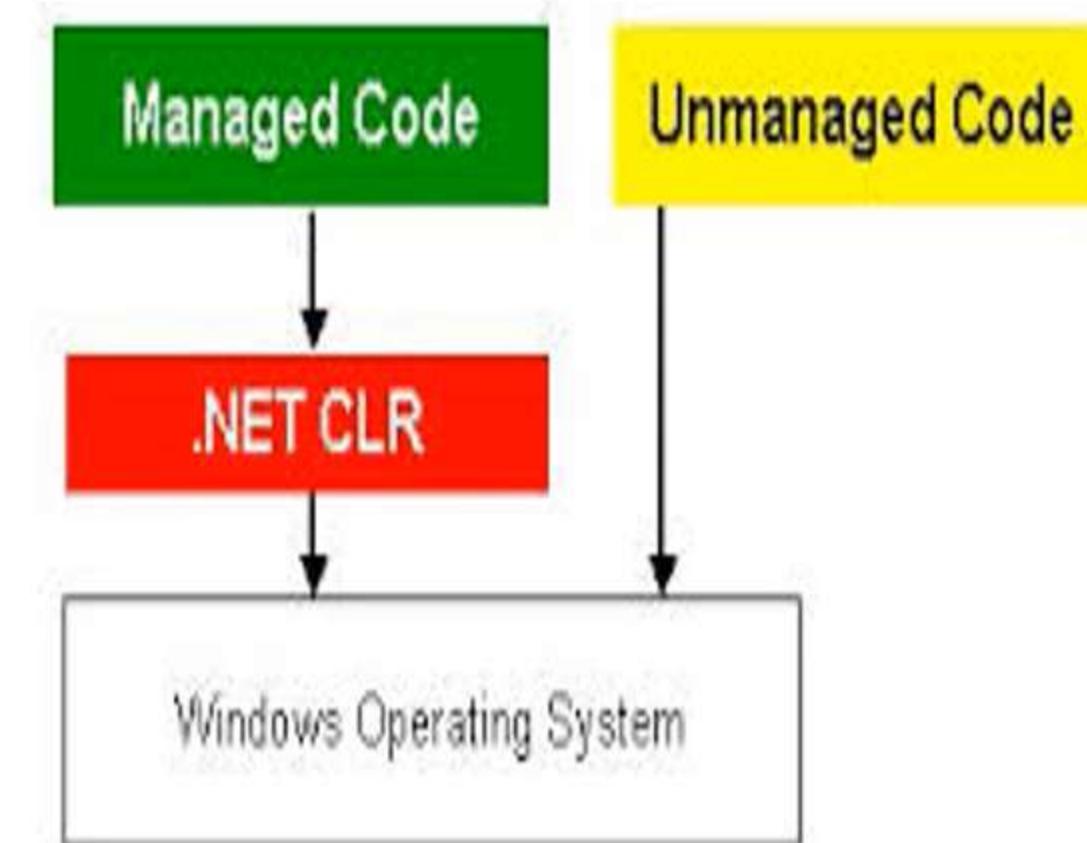


# CLR and Code Management

Code that is developed with a language compiler that targets the common language runtime is called managed code

The managed code is benefited from the services provided by the CLR

Unmanaged Codes are directly handled by the Operating System



# Intermediate Language

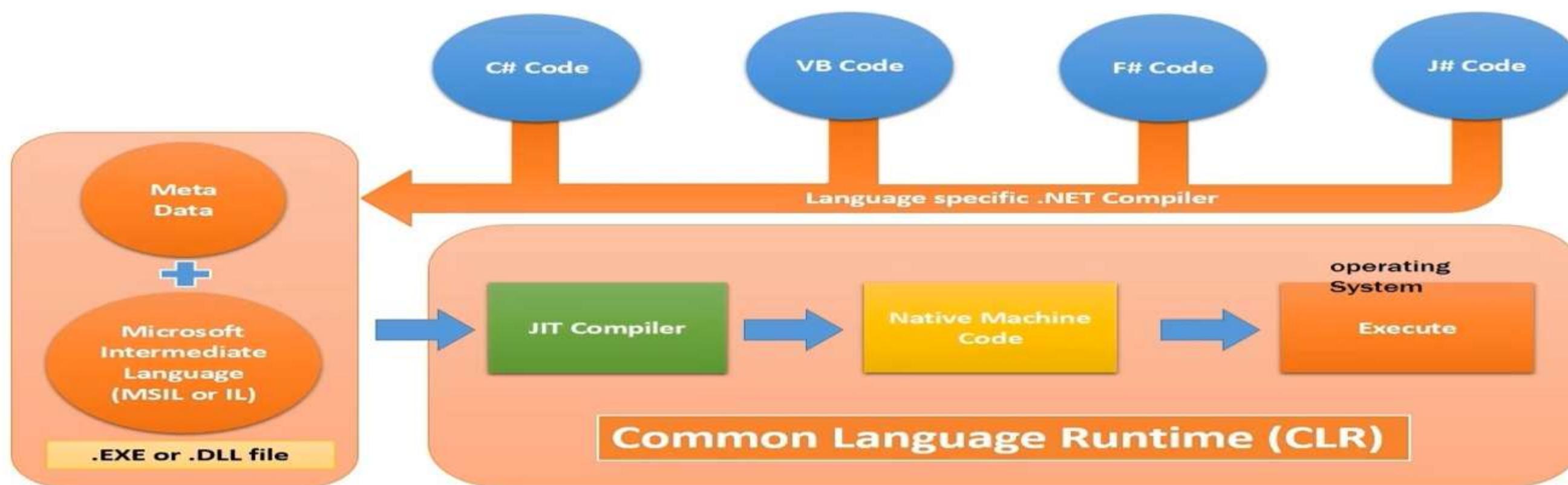


What is IL?

IL or Microsoft Intermediate Language(MSIL) is a CPU-independent set of instructions that can be efficiently converted to native code. MSIL includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations.



# How CLR Works ?



Application written in .NET language is compiled to Intermediate Language (IL)

Metadata Engine creates Metadata information for the application

At the end of compilation an *exe* or *dll* (IL) file(**assembly**) is created with Metadata and written onto the disk.

When the application is run, the classes and required library classes are loaded by the class loader and verified for type safety.

JIT compiler processes IL by translating it into native code which is taken by .NET runtime manager

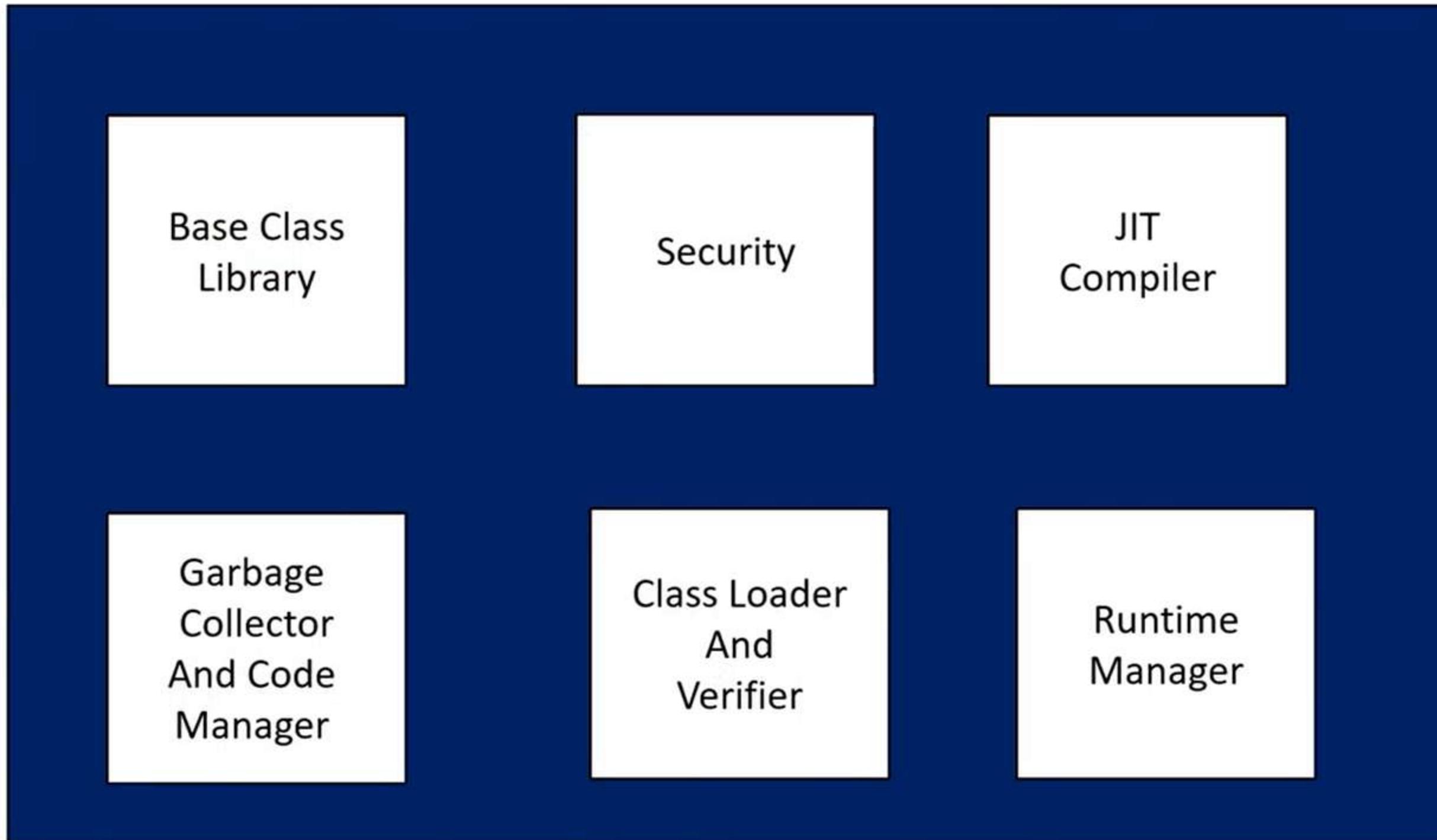
# Assembly

Assemblies form the fundamental unit of deployment, version control, reuse, activation scoping, and security permissions for a .NET-based application.

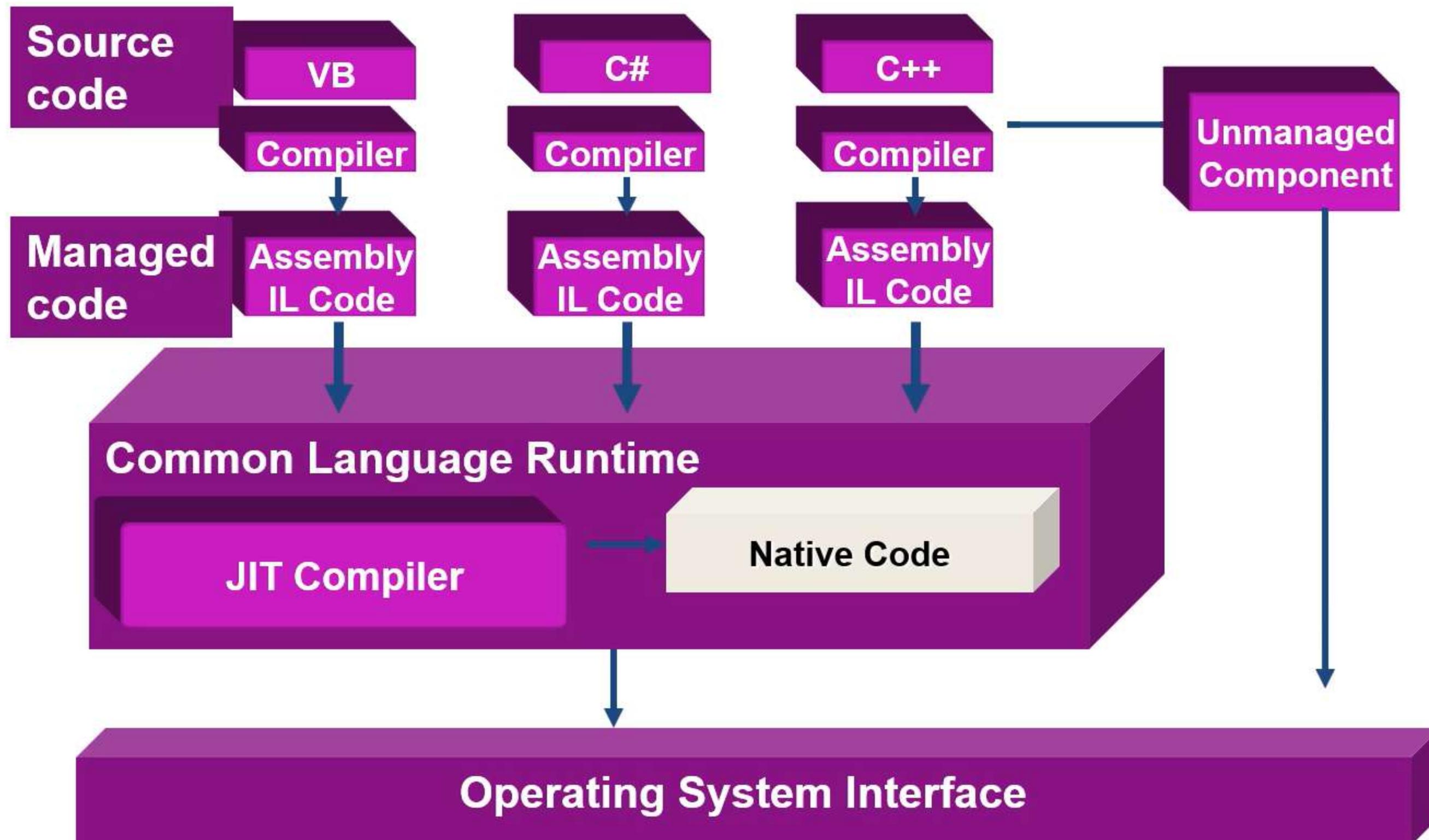
Assemblies can contain one or more modules.

Assemblies are implemented as .exe or .dll files.

# CLR - Components

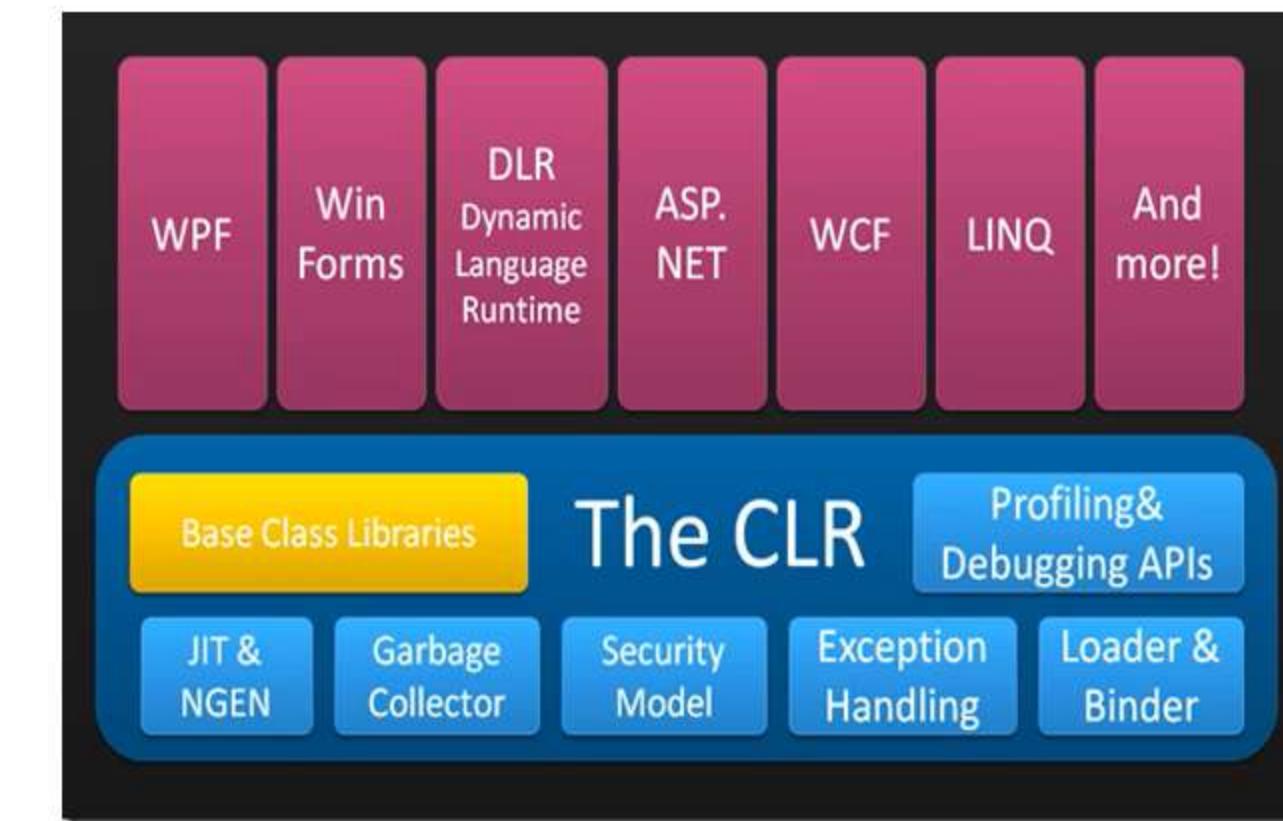
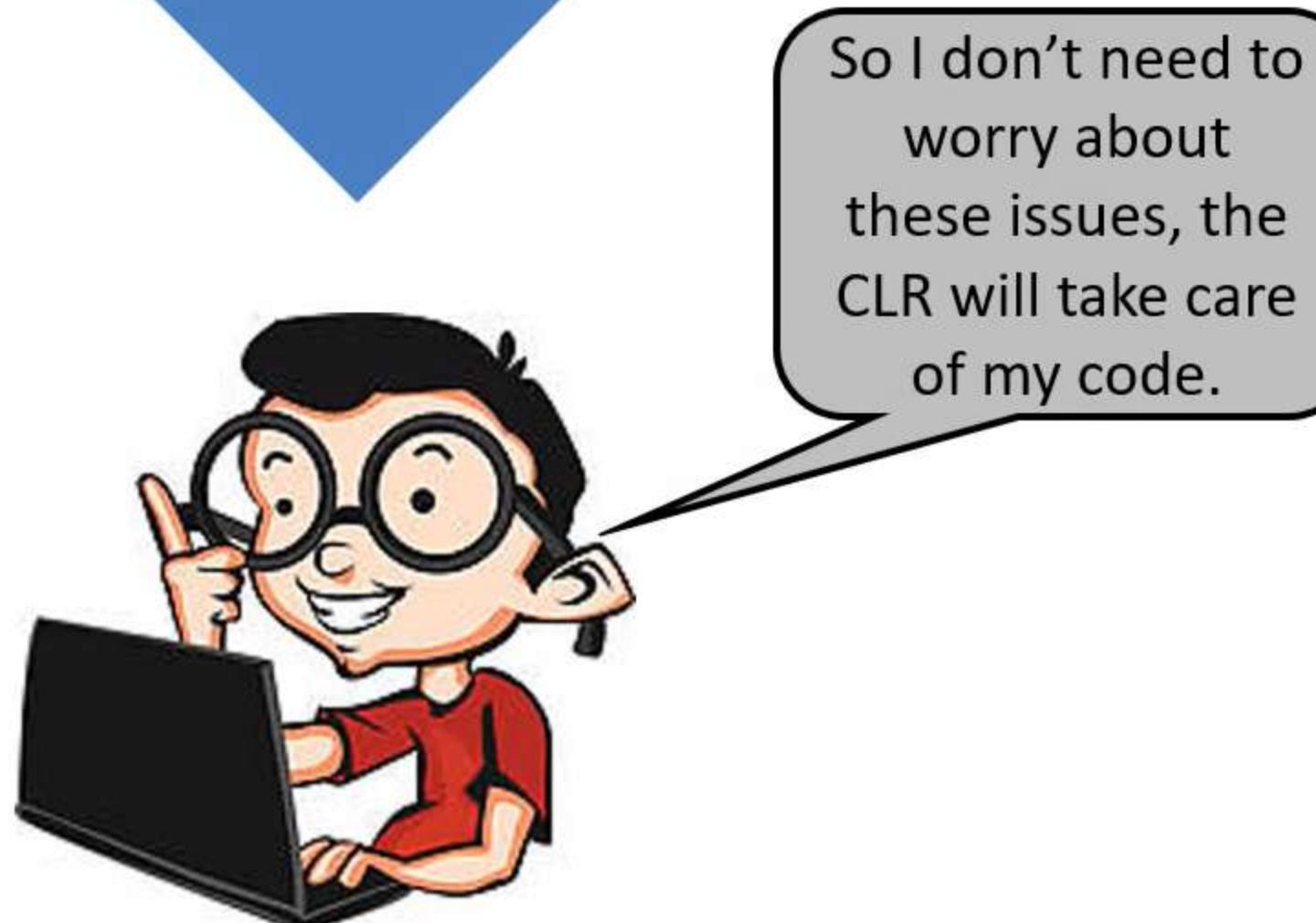


# CLR -Execution Model



# Common Language Runtime Services

- The CLR provides the runtime environment along with various services such as just in time compilation, automatic memory management, multithreading, type safety, security ,exception handling and much more.



# Exception Handling

The Common Language Runtime Supports structured exception handling

Yes! You can  
run I will  
handle the  
problems.



Whenever an error condition occurs in the code , execution is changed to the exception handler which is specifically geared to handle the exception condition.

# Memory management

The garbage collector (GC) within the CLR manages the allocation and release of memory for your application

The GC checks for objects in the managed heap that are no longer being used and reclaims the memory



# Code Access Security

The .NET Framework provides a security mechanism called code access security

Code access security allows code to be trusted to varying degrees depending on where the code originates and on other aspects of the code's identity



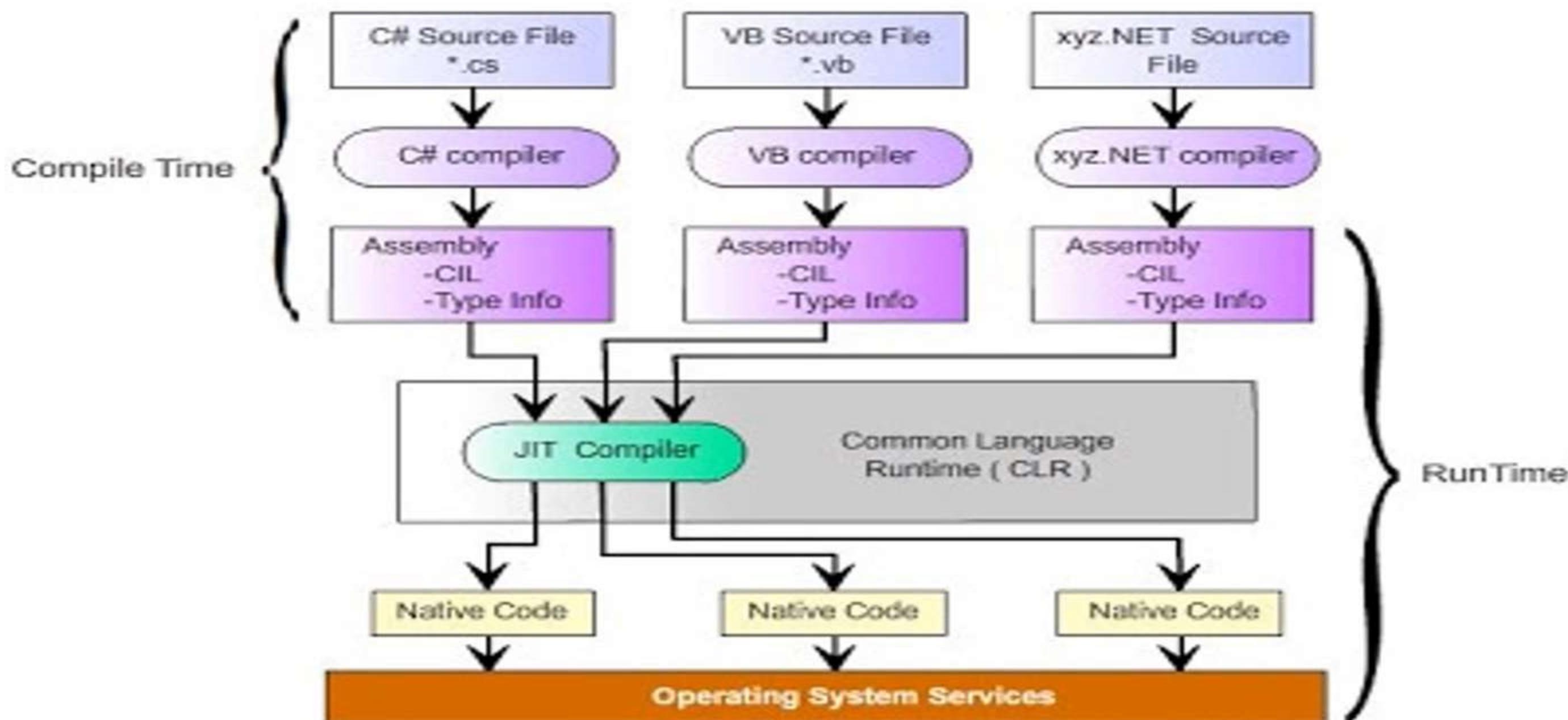
Wow! Even if I download code from an unknown origin it will run with protection.

Your code is monitored....



# Just In Time Compiler(JIT)

The Just In Time Compiler converts the IL code into native code .



# Just In Time Compiler(JIT)



Like a super man who arrives just in time of need

The JIT compiler converts the MSIL code as needed during execution and stores the resulting native code in memory so that it is accessible for subsequent calls in the context of that process.



# Type Safety

As a part of JIT compilation the CLR verifies the meta data and the IL that they are type safe to be compiled into native code .

I am a part of CLR. I will check the code for safety. Only type safe code can pass through me.

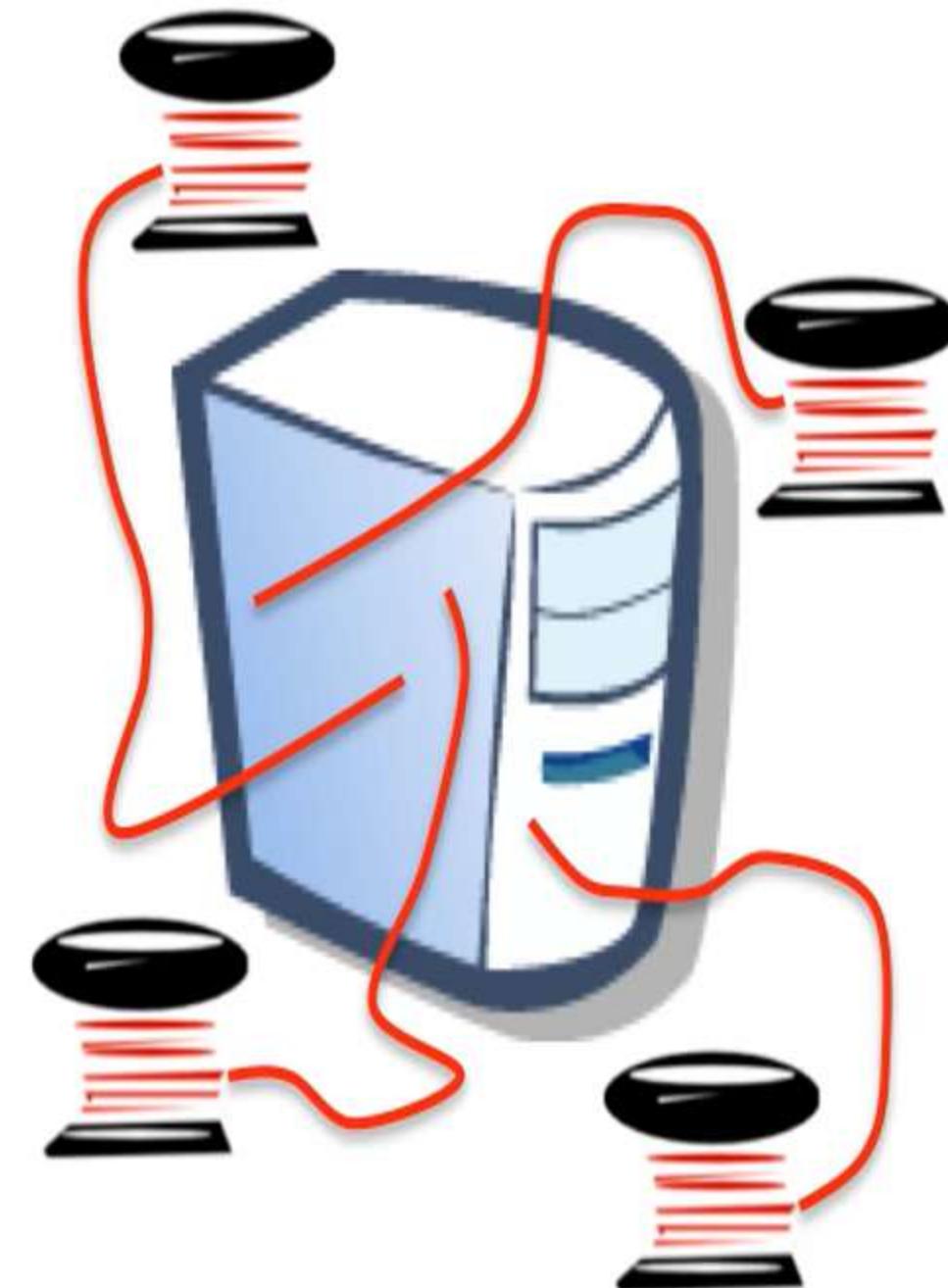


```
.method private hidebysig static void Main(string[] args) cil managed
{
    .entrypoint
    // Code size     107 (0x6b)
    .maxstack 4
    .locals init ([0] class OptionalandNamedArguements.SalaryCalculator salaryCalculator,
                [1] float64 employeeBonus,
                [2] class OptionalandNamedArguements.ISalaryCalculator iSalaryCalculator)
    IL_0000: nop
    IL_0001: newobj   instance void OptionalandNamedArguements.SalaryCalculator::ctor()
    IL_0006: stloc.0
    IL_0007: ldloc.0
    IL_0008: ldc.r8   100000.
    IL_0011: ldc.r8   2.
    IL_001a: ldc.i4.1
    IL_001b: callvirt  instance float64 OptionalandNamedArguements.SalaryCalculator::GetBonus(float64,
                                                float64,
                                                int32)

    IL_0020: stloc.1
    IL_0021: ldstr    "Bonus of employee while called using class instanc"
    + "e directly {0}"
    IL_0026: ldloc.1
    IL_0027: box      [mscorlib]System.Double
    IL_002c: call     void [mscorlib]System.Console::WriteLine(string,
                                                               object)
    IL_0031: nop
    IL_0032: newobj   instance void OptionalandNamedArguements.SalaryCalculator::ctor()
    IL_0037: stloc.2
    IL_0038: ldloc.2
```

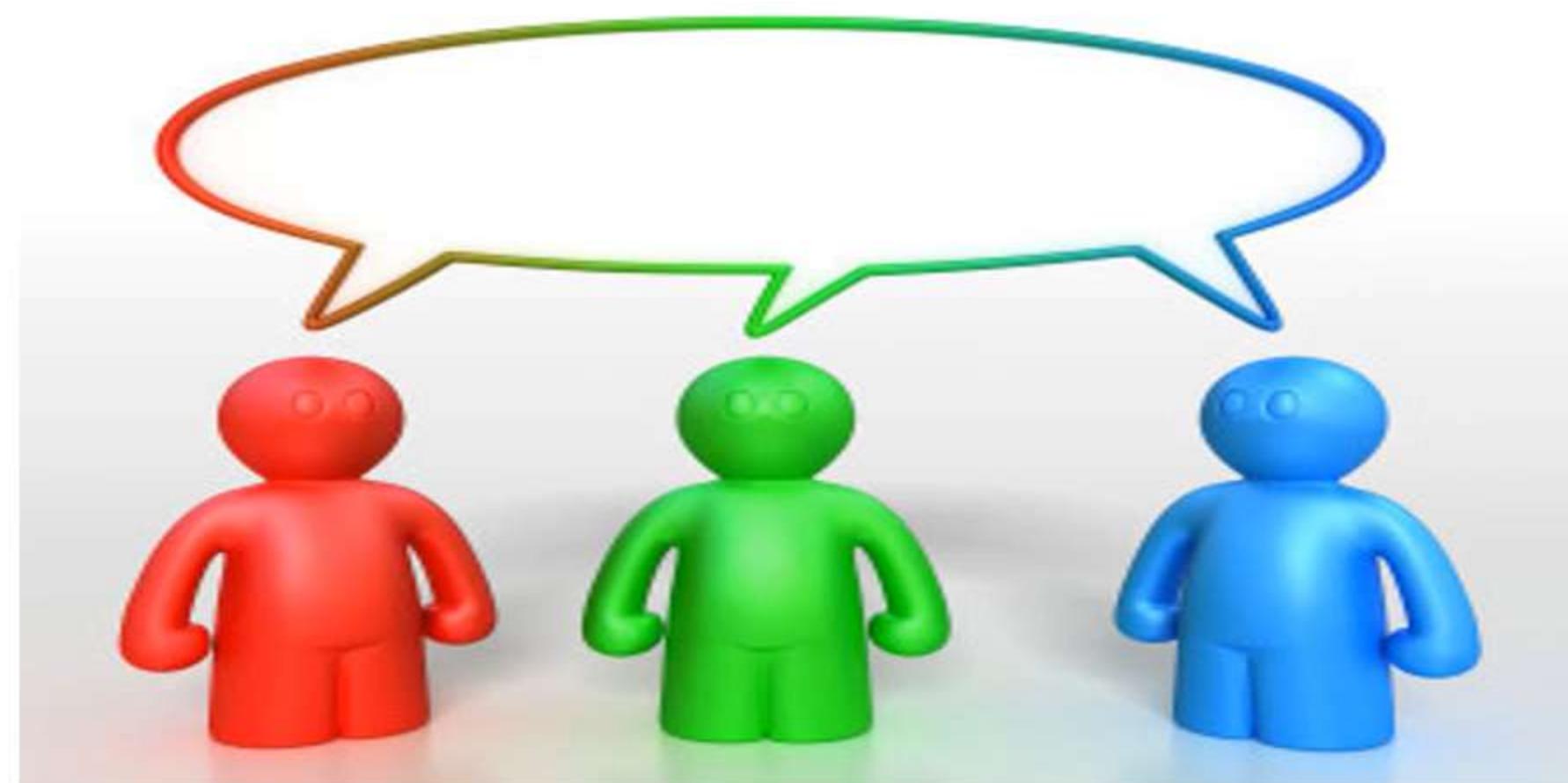
# Multi Threading

- Threading enables a program to perform concurrent processing so that more than one operation takes place at a time.
- For example, you can use threading to monitor input from the user, perform background tasks, and handle simultaneous streams of input.



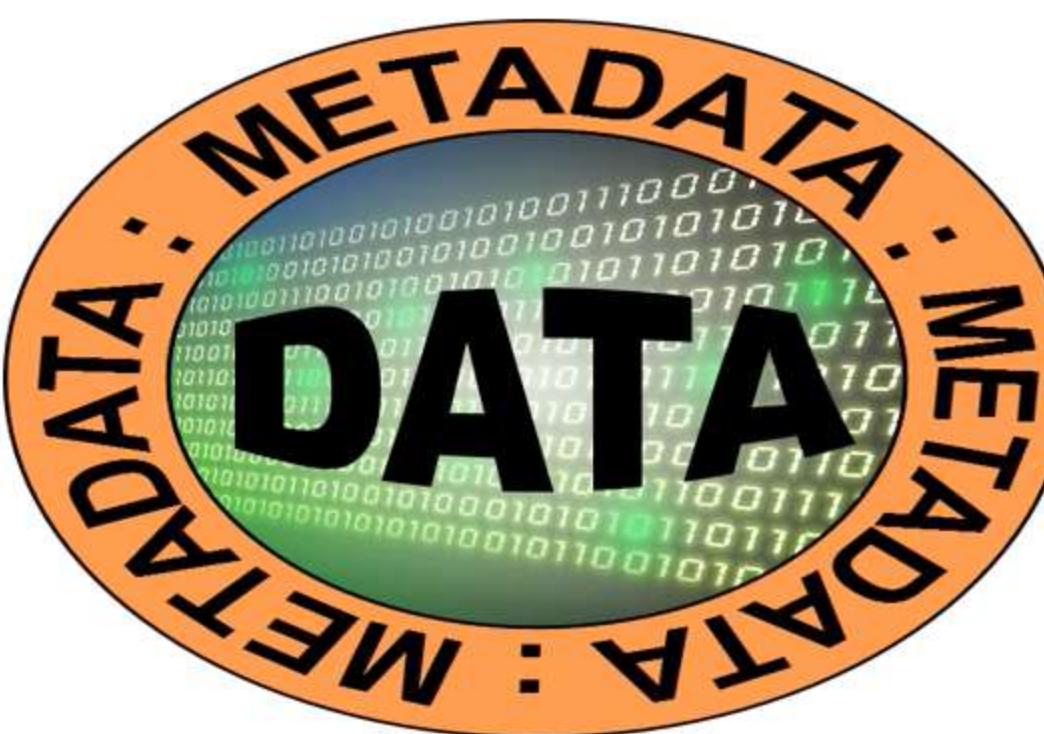
# Cross language Interoperability

The common language runtime provides the necessary foundation for language interoperability by specifying and enforcing a common type system and by providing metadata.



# Metadata

What is  
meta data?



Metadata is binary information describing your program that is stored either in a common language runtime portable executable (PE) file or in memory.

Every type and member that is defined and referenced in a module or assembly is described within metadata.

When code is executed, the runtime loads metadata into memory and references it to discover information about your code's classes, members, inheritance, and so on.

## Type Information

Automatically  
bound into  
assembly

Inseparable

Stored in binary  
format

Describes every  
class type

Used by  
VisualStudio.NET  
IntelliSense

# Design Goals of CLR

Loading and Execution of the applications



Memory Management and Garbage Collection



Run time type checking



Querying the Metadata



Code security



Exception Handling



Interoperability & Cross language integration



Solving of the problems like versioning



# Common Type System

- The common type system defines how types are declared, used, and managed in the common language runtime.
- It is an important part of the runtime's support for cross-language integration.

## Common Type System

Type

Value Type

Reference Type

# Common Type System

## CTS performs the following functions:

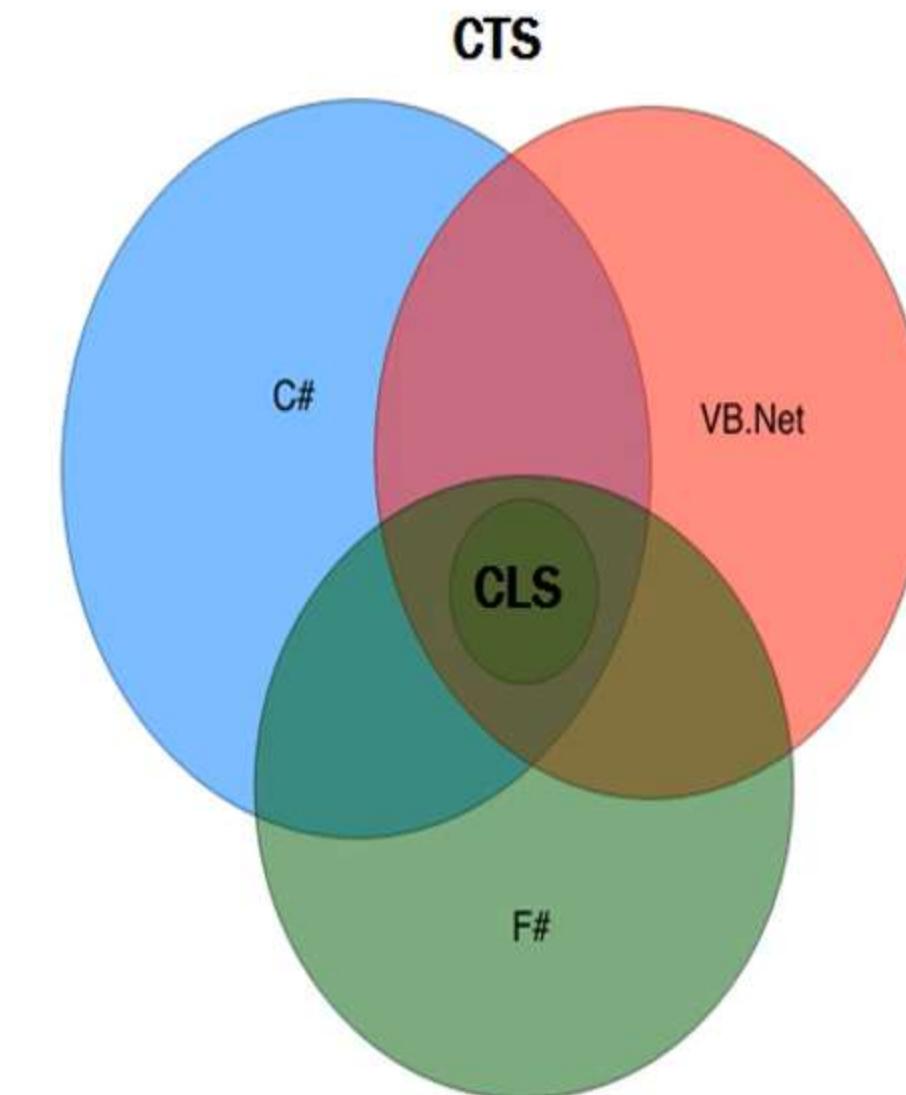
Establishes a framework that enables cross language integration, type safety and high performance code execution

Provides an object-oriented model that supports the complete implementation of many programming languages

Defines rules that languages must follow, which helps to ensure that objects written in different languages can interact with each other

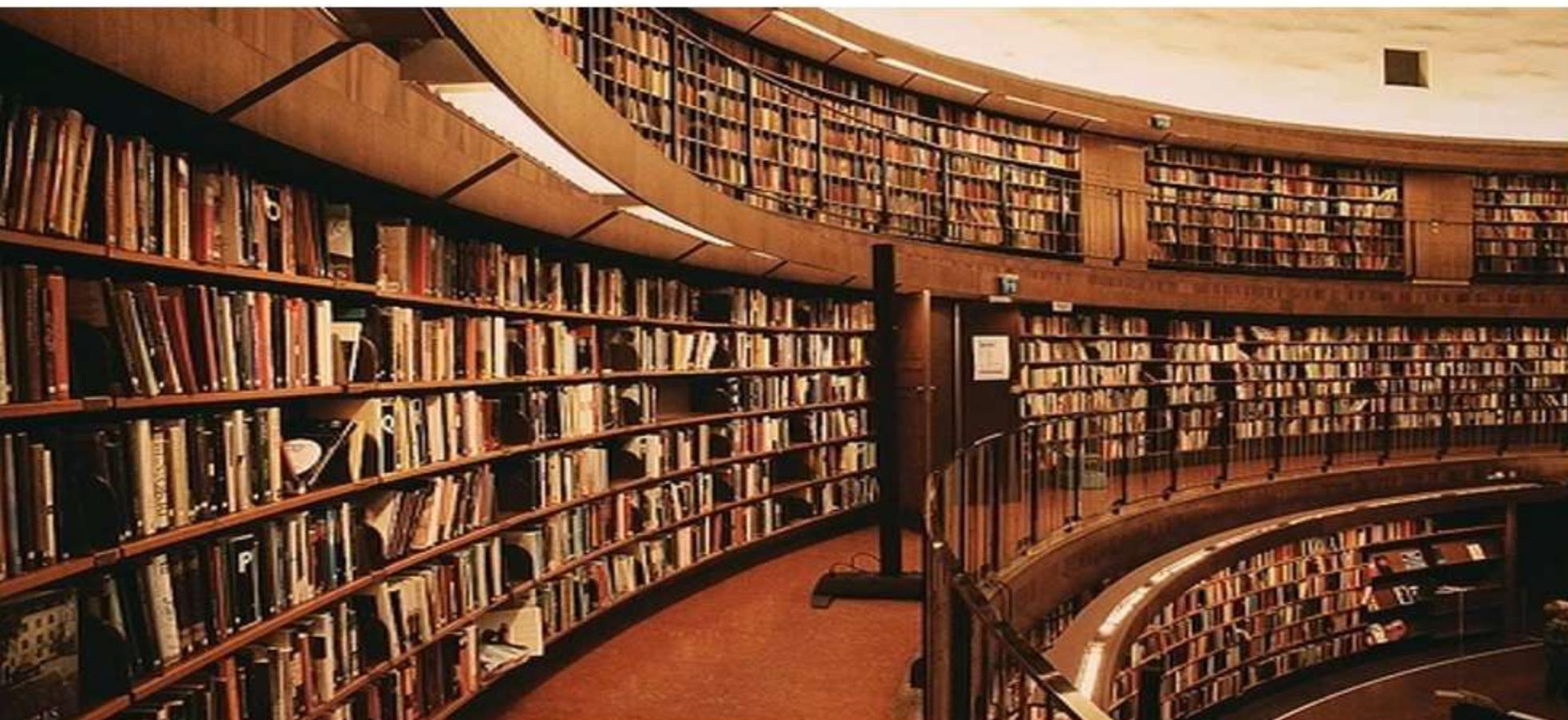
# Common Language Specification

- The .NET Framework provides the Common Language Specification (CLS), which describes a fundamental set of language features and defines rules for how those features are used.

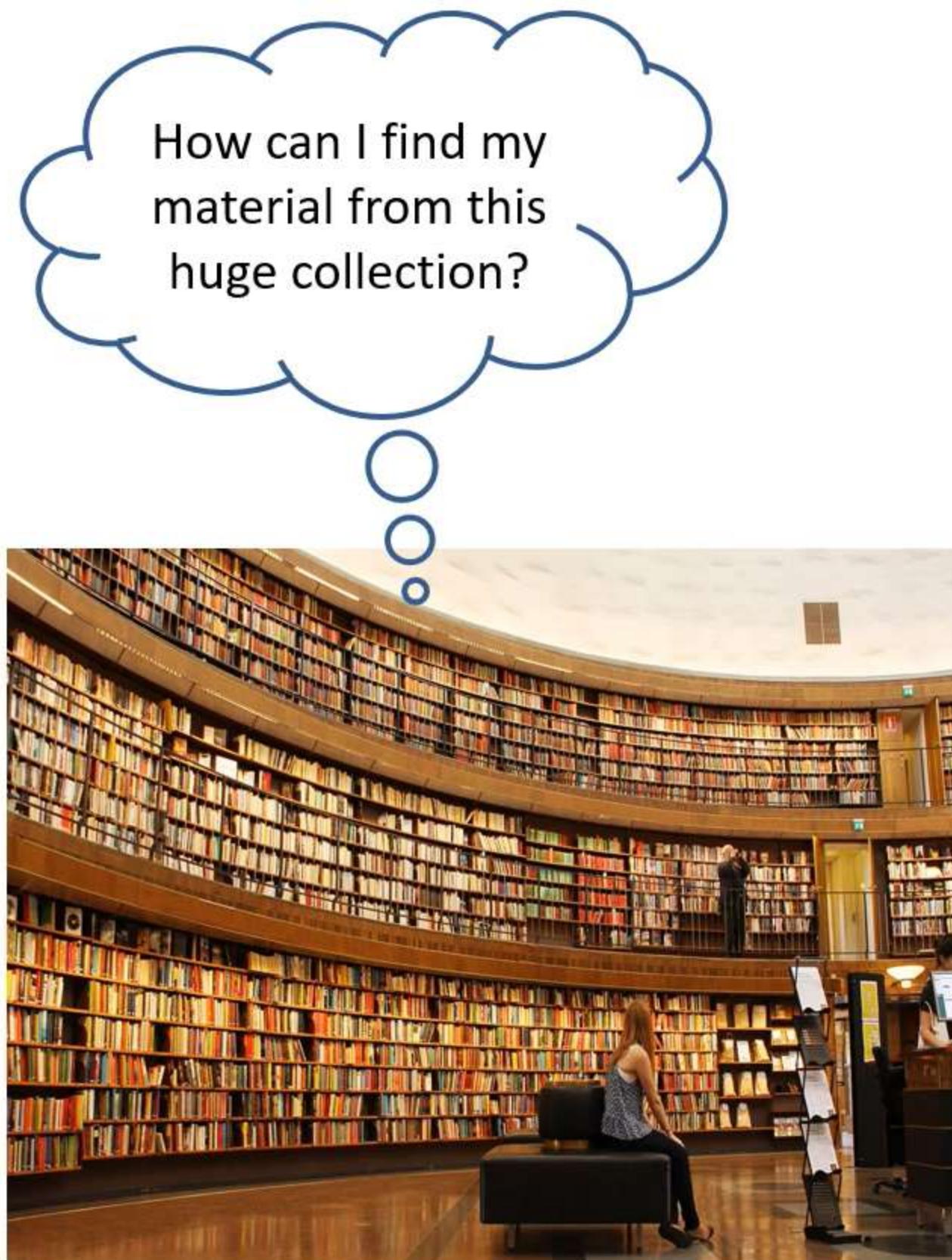


# Framework Class Library

- The .NET Framework class library is a library of classes, interfaces and value types that provide access to system functionality.
- It is the foundation on which .NET Framework applications, components and controls are built.



# Namespace

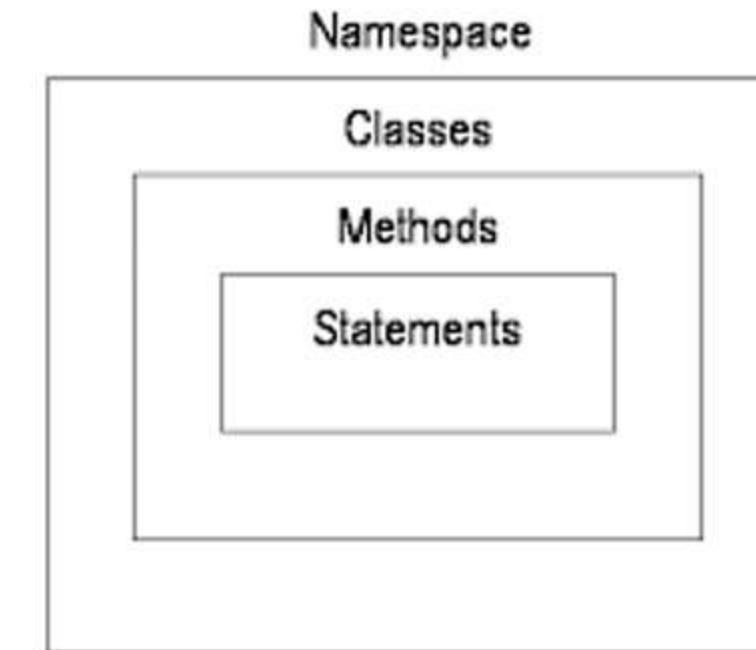


# The Framework Class library Namespaces

- The Framework Class library is arranged into hierarchical namespaces that contain structures, enumerations, delegates, interfaces and classes within which methods and properties are defined.



# Namespace

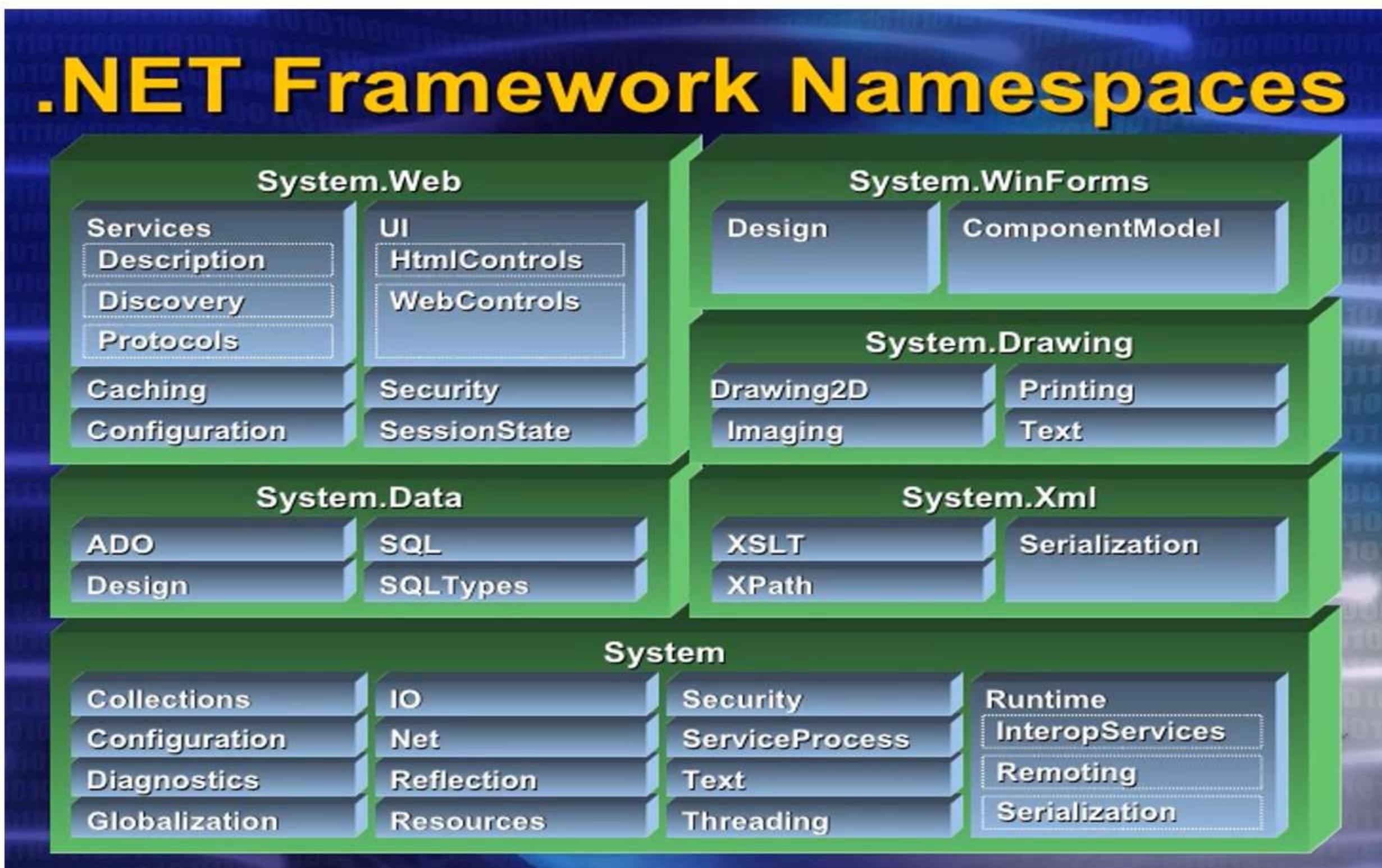


Related functionalities are stored under a single class or interface.  
Related classes and interfaces are stored under namespace.



For example, the System.Console namespace contains ReadLine and WriteLine Functions.

# The Framework Namespaces



# Languages supported by .NET

The .NET Platform programming languages — including Visual Basic .NET, Visual C#, Managed Extensions for C++, and many other programming languages from various vendors — use .NET Framework services and features through a common set of unified classes.

The following languages are incorporated with the Visual Studio IDE

Visual C++

Visual C#

Visual Basic

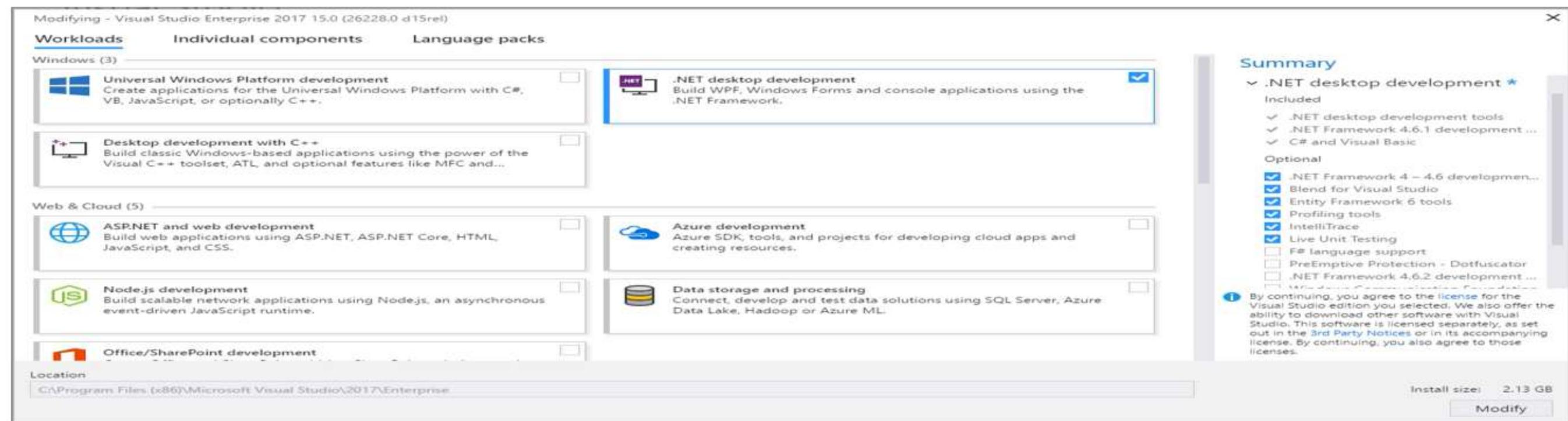
Java Script

Visual F#

Various third-party language compilers are available that generate managed code that conforms to the CTS, and hence can run under CLR.

# Exploring Visual Studio 2017

- Microsoft visual studio 2017 enables the developer to write an accurate code more efficiently.
- It is considered to be one of the best Integrated development environment(IDE).
- While installing VS 2017, we can pick and choose the different components that we need to install.
- The visual studio 2017installer uses the concepts of workloads.



# New Enhanced Features

- Redefined fundamentals --Visual Studio makes it easier and faster to install just the features you need, when you need them. And provides options to fine-tune your installation.
- Performance and productivity- The new Visual Studio Performance Center can help you optimize your IDE start-up time.
- A new identity service allows the user to share user accounts across Team Explorer, Azure Tools, Microsoft Store publishing, and more.
- Live unit testing gives you live unit test results and code coverage in the editor while you are coding.



# Start Page

Start Page - Microsoft Visual Studio

File Edit View Project Debug Team Tools Test Analyze Window Help

Quick Launch (Ctrl+Q) sajitha nazer SN

Server Explorer Output Ready

Start Page

Recent

This week

- lambdaDemo.sln  
C:\Users\Saji\source\repos\lambdaDemo\lambdaDemo.sln
- LINQDemo.sln  
C:\Users\Saji\source\repos\LINQDemo\LINQDemo.sln

Last week

- ConsoleApp4.sln  
C:\Users\Saji\source\repos\ConsoleApp4\ConsoleApp4.sln
- OSIM.sln  
D:\DevProjects\OSIM\OSIM.sln

This month

- NUnitTest.sln  
D:\saji\C#\NUnit\NUnitTest\_Clean\NUnitTest\NUnitTest.sln
- CabRequisition.sln  
D:\CaseStudyJuly2018\CabRequisition.sln

Open

Get code from a remote version control system or open something on your local drive.

Checkout from:

- Visual Studio Team Services
- GitHub

Open Project / Solution

Open Folder

Open Website

New project

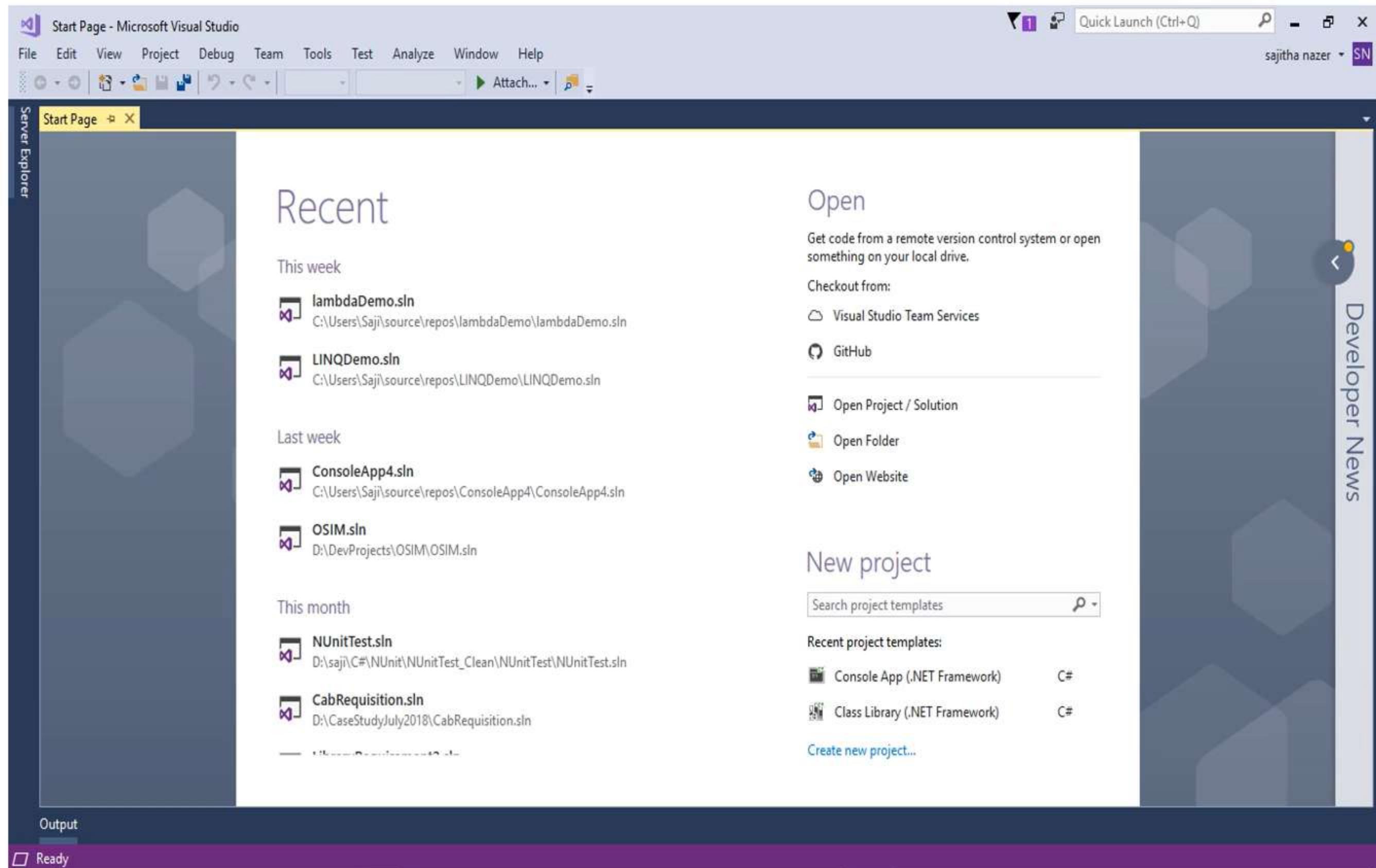
Search project templates

Recent project templates:

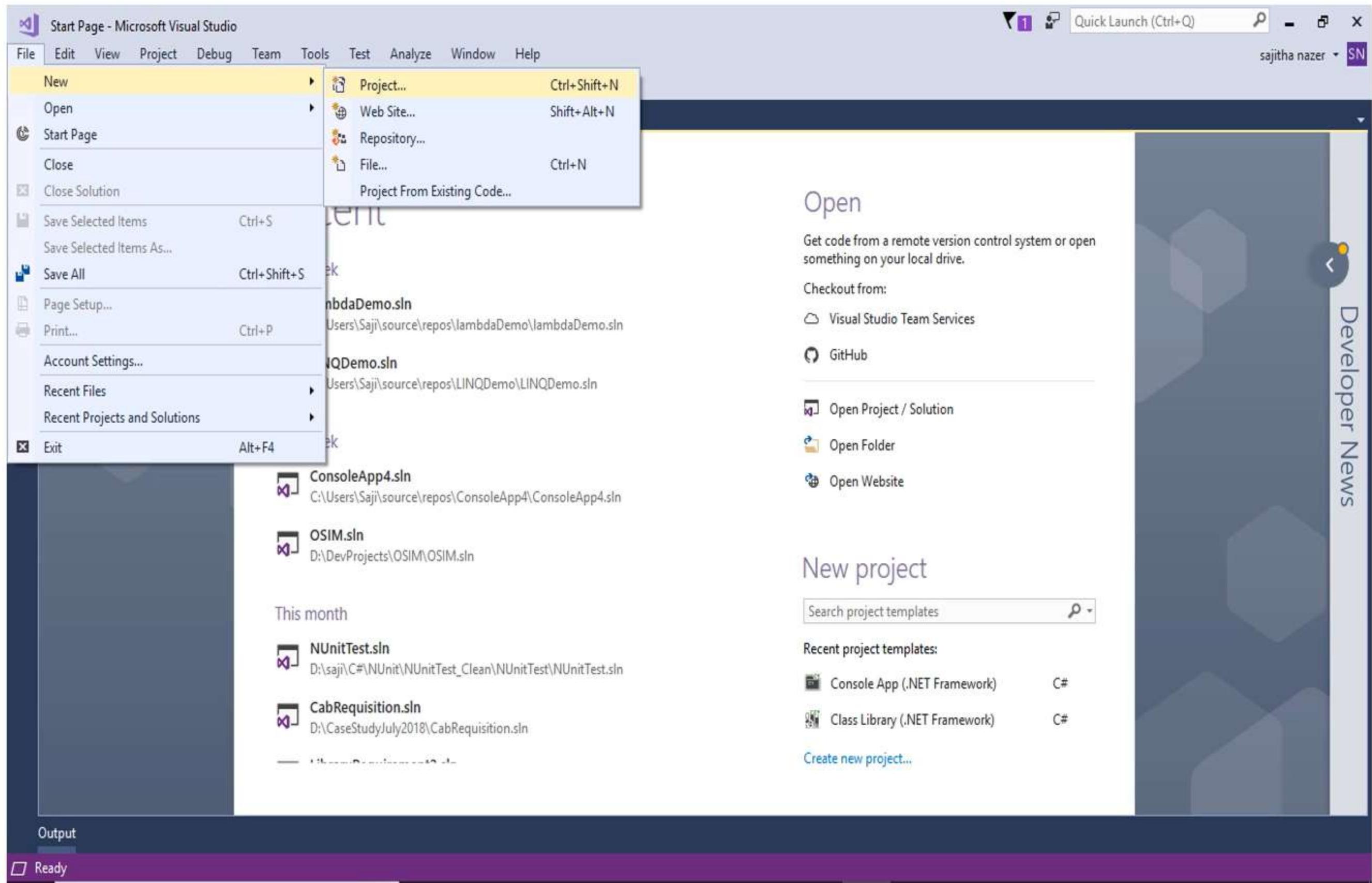
- Console App (.NET Framework) C#
- Class Library (.NET Framework) C#

Create new project...

Developer News



# Menu



The screenshot shows the Microsoft Visual Studio Start Page. The **File** menu is open, with the **New** option highlighted. A submenu is displayed, containing the following items:

- Project... (Ctrl+Shift+N)
- Web Site... (Shift+Alt+N)
- Repository...
- File... (Ctrl+N)
- Project From Existing Code...

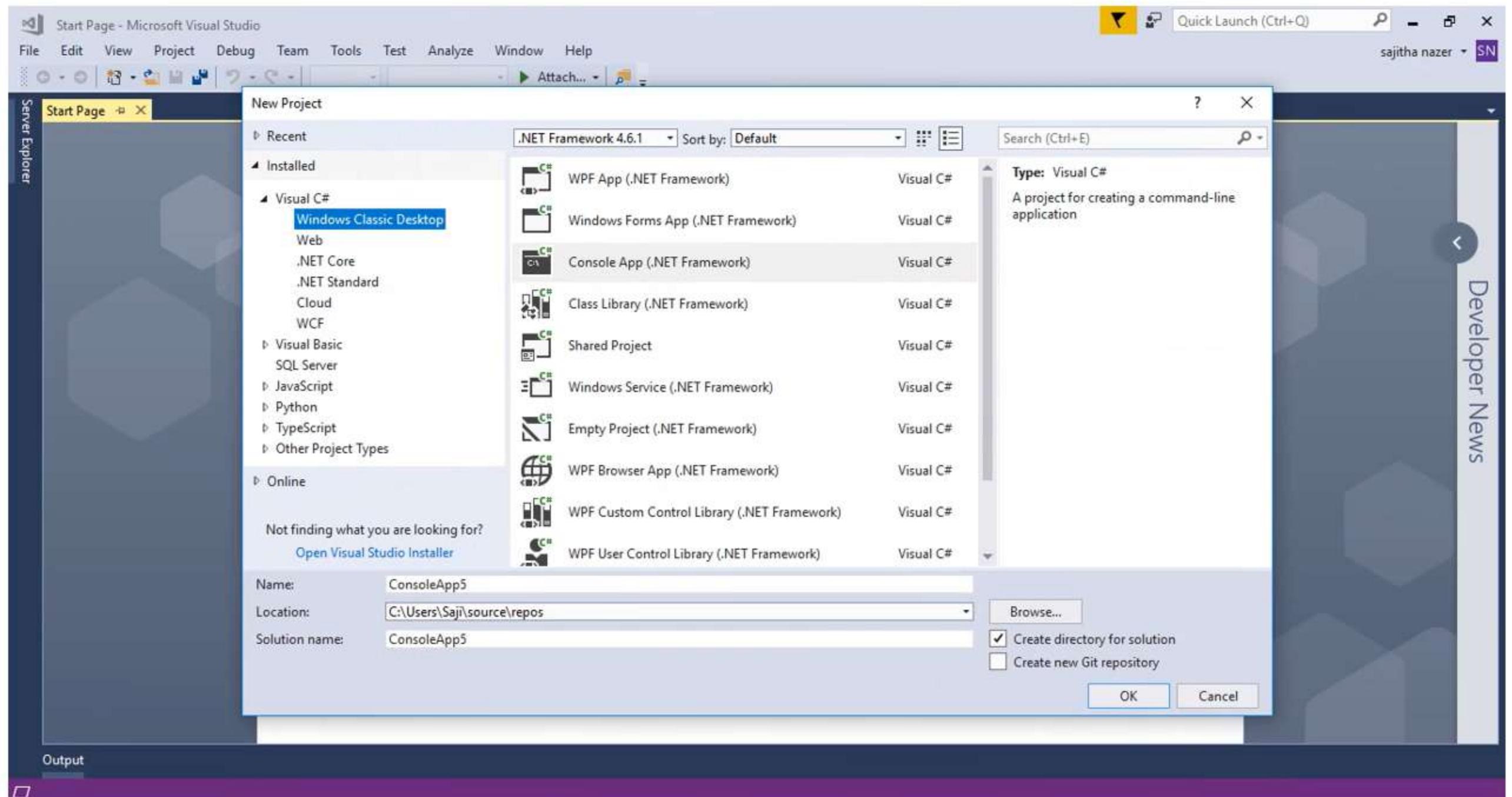
The main workspace displays recent files and projects. Under **Recent Files**, there are entries for **lambdaDemo.sln** and **LINQDemo.sln**. Under **This month**, there are entries for **NUnitTest.sln** and **CabRequisition.sln**.

The **Output** tab at the bottom left shows the message "Ready".

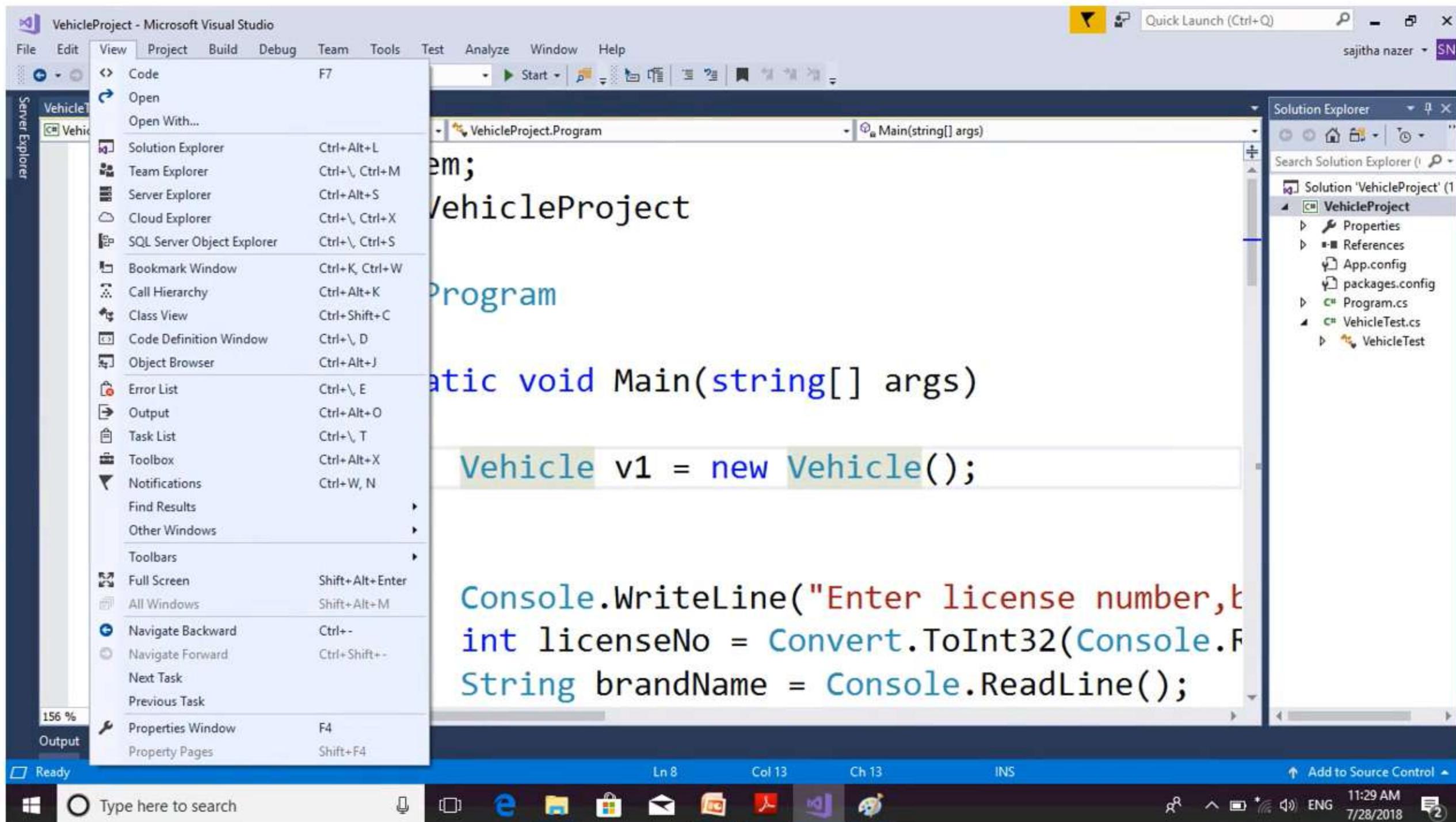
The **Developer News** sidebar on the right features a large image of a person's head and shoulders, with the text "Developer News" written vertically next to it.

The top right corner of the window shows the user's name, **sajitha nazer**, and a small profile icon.

# Project dialog box



# Project dialog box



# Summary

- .NET Framework
- .NET Framework Architecture
- Common Language Runtime
- CLR Services
- Framework Class Library
- Common Type System
- Common language Specification
- Languages supported by .NET



# INTRODUCTION TO C#



# Objectives

In this module you will learn

- Introduction to C# Language
- Features of C#
- C# Compilation and Execution
- General Structure of a C# Program
- Creating and Using a DLL



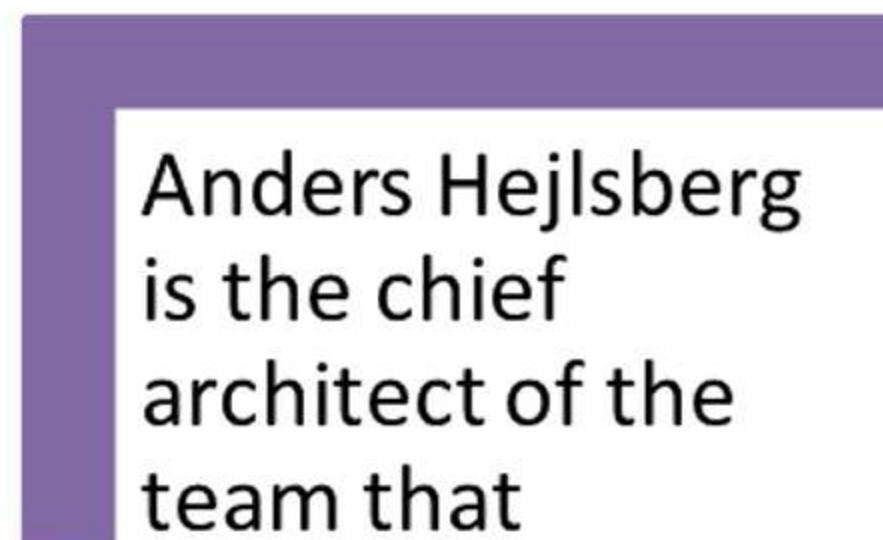
# Why C#

- C# is an elegant and type-safe object-oriented language.
- It enables developers to build a variety of secure and robust applications that run on the .NET Framework.
- C# is used to create Windows client applications, XML Web services, distributed components, client-server applications, database applications, and much, much more.....

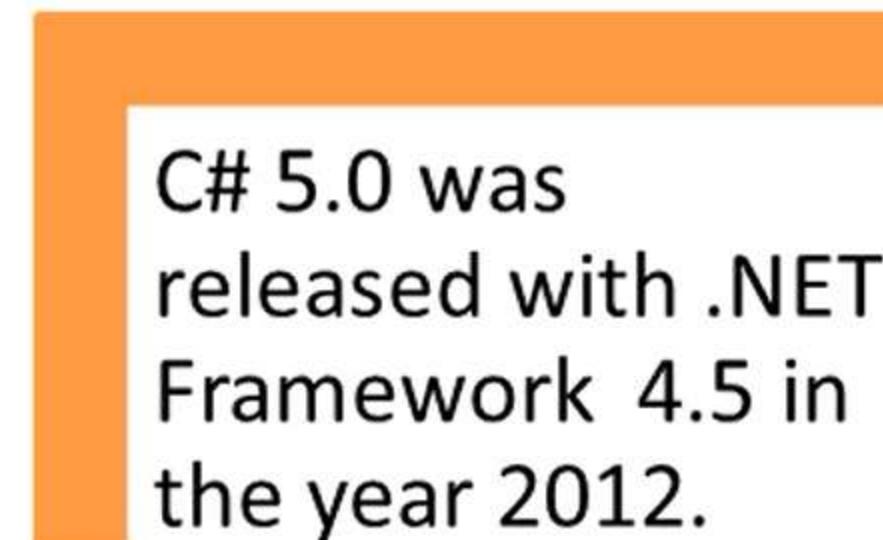
# Introduction to C#



In 2000, Microsoft announced the C# programming Language



Anders Hejlsberg is the chief architect of the team that developed C#



C# 5.0 was released with .NET Framework 4.5 in the year 2012.

# C# Basics

C# (pronounced "C sharp") is an Object Oriented programming language that is designed for building a variety of applications that run on the Microsoft .NET Platform (Framework).

The C# language has been standardized and is described by the ECMA-334 C# Language Specification.

Several vendors apart from Microsoft produce C# compilers. Visual C# is an implementation of the C# language by Microsoft.

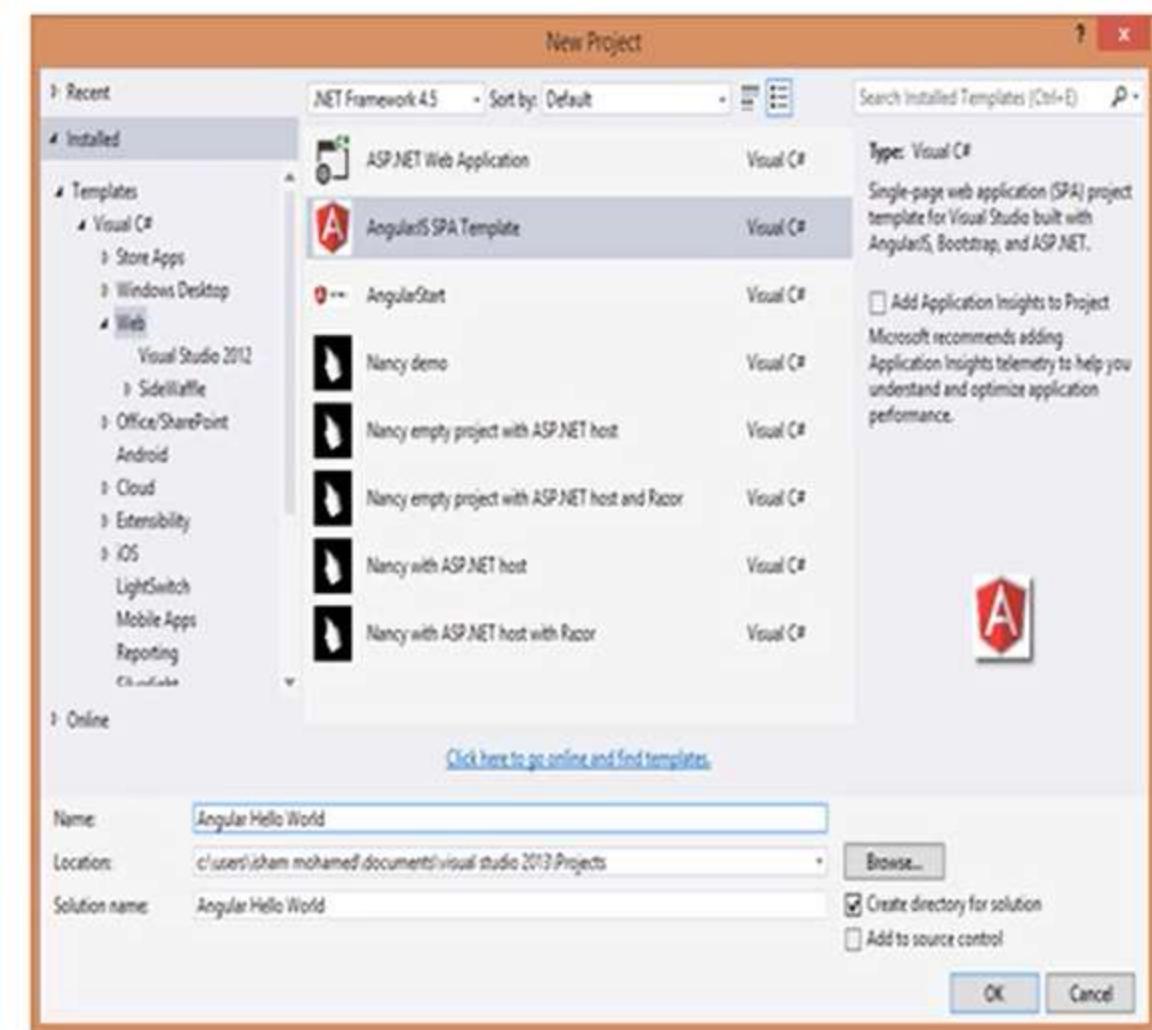
Microsoft Visual Studio 2012 IDE (Integrated Development Environment) supports Visual C# with an editor, compiler, project templates, debugger, and other tools.

# Visual Studio

Visual studio is a complete set of development tools used to create applications with .NET Framework

Visual C# makes use of the IDE which allows to share tools and facilitates in the creation of mixed-language solutions

Visual Studio provides many different application templates to help you create programs, and several programming languages in which to write them



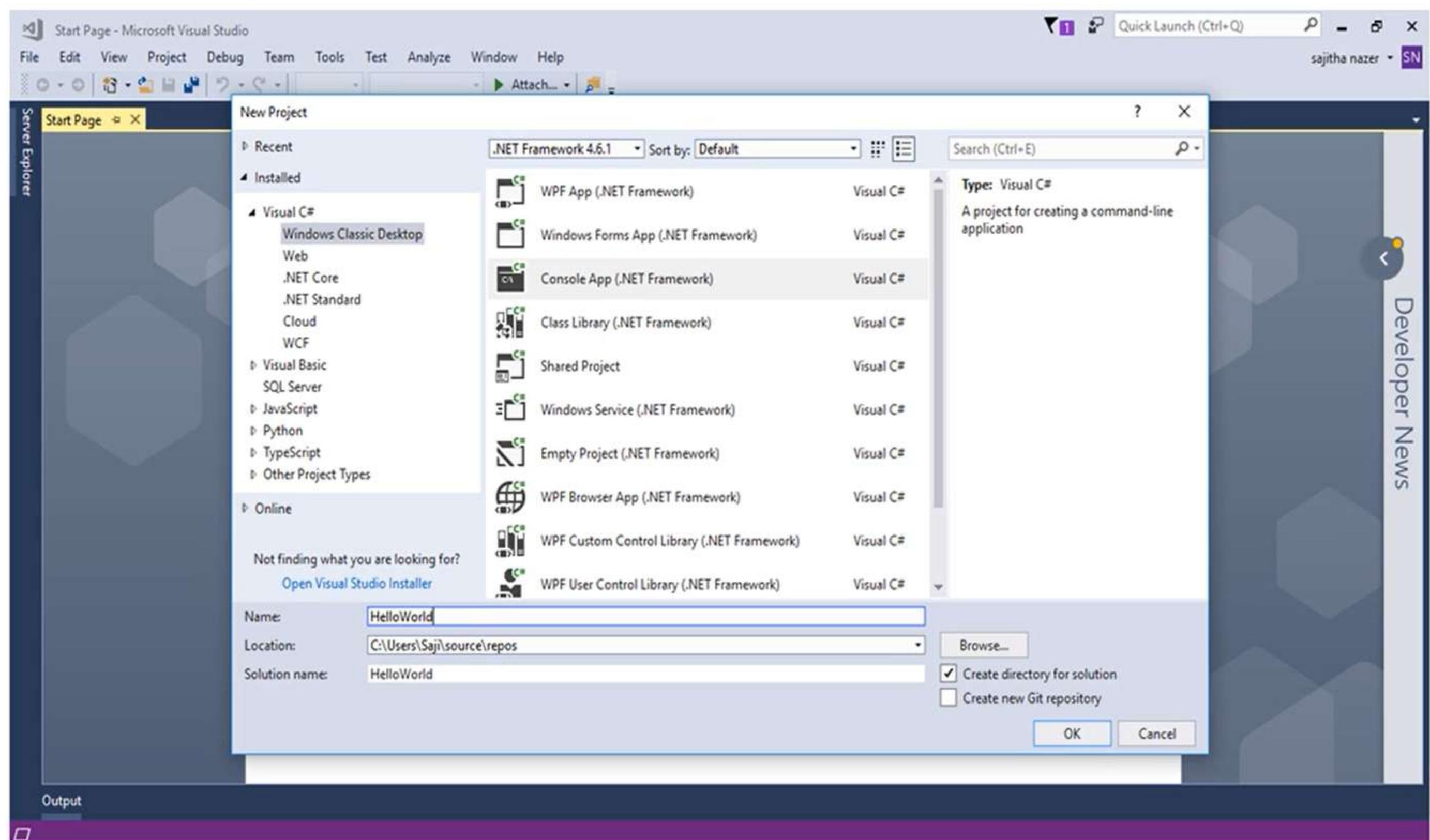
# Starting with a simple C# Program

Creating a simple C# "Hello World!" program in Visual Studio 2012.

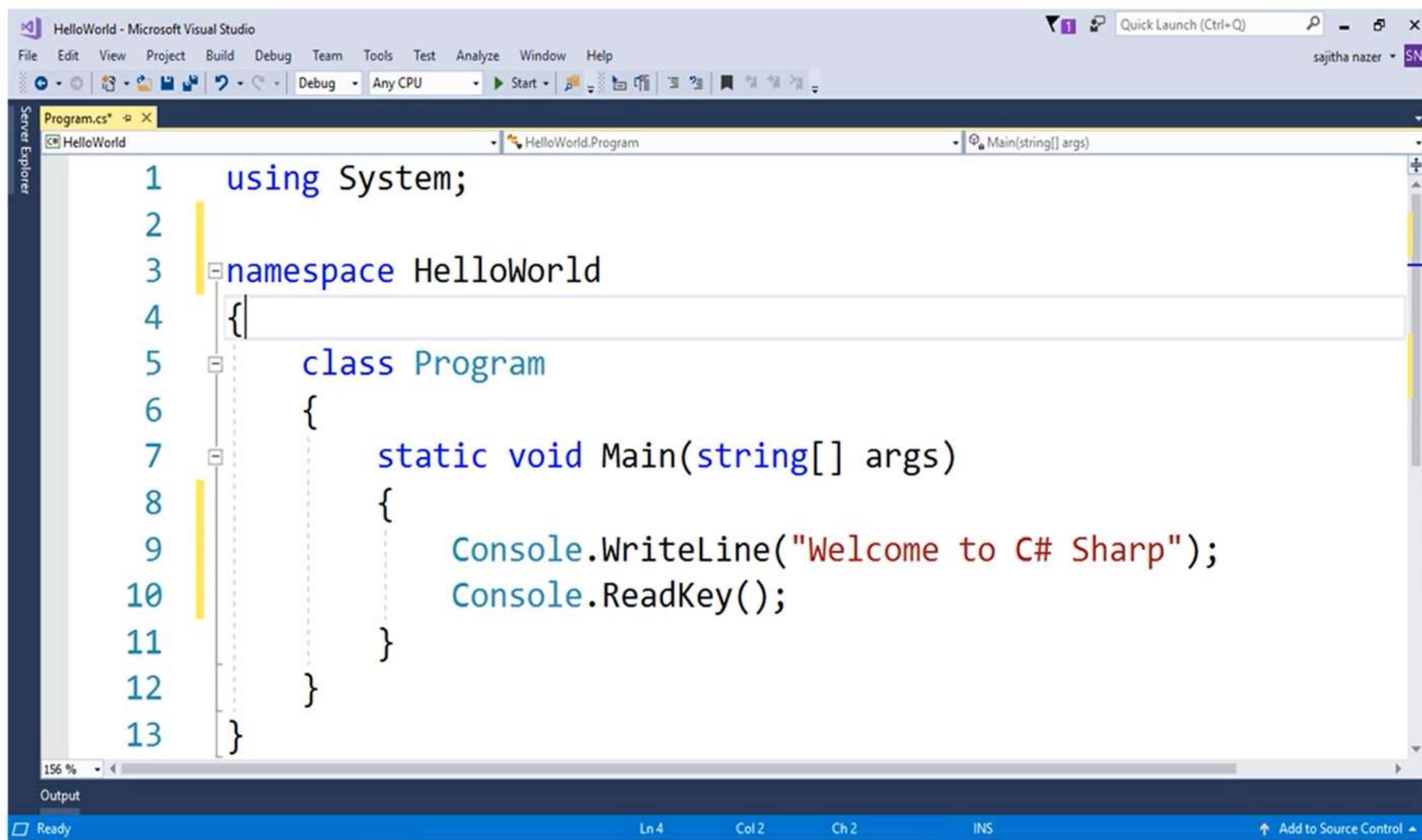
To create and run a Console Application

- Start Visual Studio.
- Choose **File, New, Project**.
- The **New Project** dialog box opens.
- Select Visual C# Project from Project Type and select Console Application from Template.
- Specify a name “HelloWorld” for the project, and click OK

# Using Visual Studio 2017



# Using Visual Studio 2017 contd.



A screenshot of the Microsoft Visual Studio 2017 interface. The title bar reads "HelloWorld - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Test, Analyze, Window, and Help. The toolbar has icons for Save, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The status bar at the bottom shows "Ready", "Ln 4 Col 2 Ch 2", "INS", and "Add to Source Control".

The code editor window displays the file "Program.cs" with the following content:

```
1  using System;
2
3  namespace HelloWorld
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Welcome to C# Sharp");
10             Console.ReadKey();
11         }
12     }
13 }
```

# Using Visual Studio 2017 contd..

Open the Program.cs in Solution Explorer.  
Replace the contents of Program.cs with the code

```
using System;
class HelloWorld
{
    public static void Main()
    {
        Console.WriteLine("Hello World");
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
```

Choose the F5 key to run the project. A Command Prompt window appears that contains the line Hello World!

# Compiling & Running the Application using Command Prompt

Type the HelloWorld.cs Program in a notepad.

Save the file as HelloWorld.cs (optional)

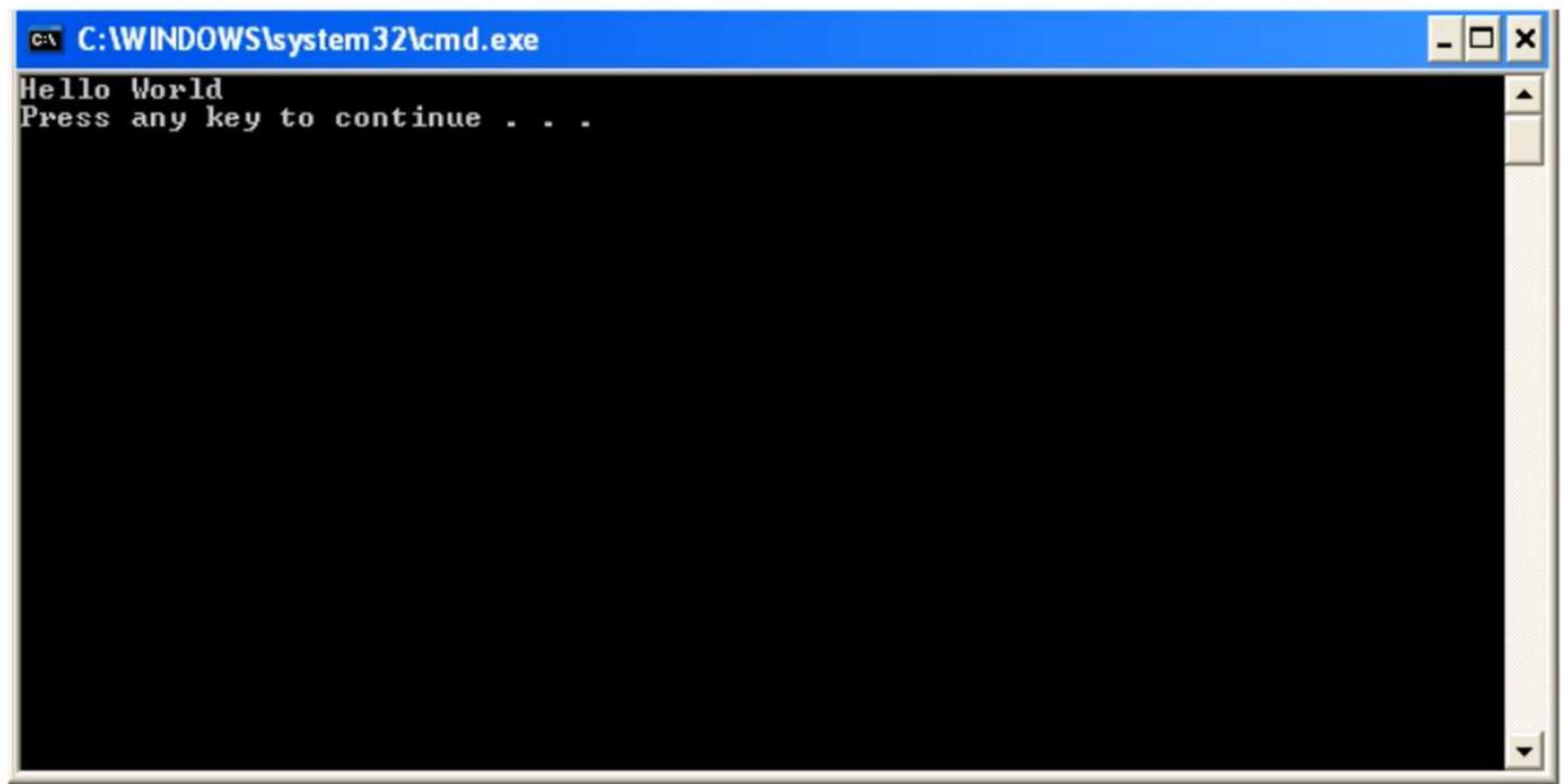
Type the following command to invoke C# compiler.

- `>csc HelloWorld.cs`

Run the application with the following command at the command prompt.

- `>HelloWorld`

# Command Prompt Contd..



# C# Compiler (csc) Options

Switch	Description
/target(/t)	Specifies the type of file output
/t:exe	An .exe file (Console application)
/t:library	A DLL file with manifest
/t:module	A DLL file without manifest
/t:winexe	A windows application
/unsafe	Allows code containing unsafe keyword to compile

# Program Structure

## COMMENTS

//

This is a first line comment

/\* \*/

Multi line comment is for a block of  
text enclosed in between

///

XML documentation comment (placed  
after /// for creating XML document for the  
code)

# Program Structure

## namespaces

is used for compartmentalizing the classes

It avoids name collisions

.NET class library is organized into hierachal namespaces

## Using directive

allows to access namespace

Syntax of **using** directive:  
**using namespace**

**using System;** makes all the classes and name spaces in System namespace available to your program

# Program Structure contd..

- Defining a Class
  - C# is a purely object oriented programming language.
  - Every method must appear in a class
- C# class definition

```
class classname
{
    // body of the class
}
```

# Program Structure – Main()

- Main() Method – Any C# application must contain a *Main()* method. It is an entry point to an application.
- public is an access modifier that tells that it is available for outside call.
- static keyword tells the compiler that Main() method is a global method and that the class does not need to be instantiated for the method to be called.
- Console class - Console is a class in the System namespace
- WriteLine() is a static method of Console class that outputs zero or more values with a newline character

```
public static void Main()
{
    Console.WriteLine("Hello World");
    Console.ReadKey();
}
```

# Input and Output Operation using a console Application



System.Console includes the following methods

Clear()

Read()

.ReadKey()

ReadLine()

Write()

WriteLine()

# Console Output

```
using System;
namespace ExamplePgms
{
    public class HelloWorld
    {
        public static void Main(string[] args)
        {
            Console.WriteLine(" Welcome to {0} programming ! \n {1}", "C#", "BYE");
        }
    }
}
```

Output:

```
Welcome to C# Programming !
BYE
```

WriteLine method can display formatted data.

{0} and {1} are format items for which the index starts with 0.

Each format item is a placeholder for a value.

In the above program {0} is a place holder for the object “C#” and {1} is for the object “BYE”.

# Console Input

```
public static string ReadLine()
```

The console's `ReadLine` method reads the next line of characters from the standard input stream

The `Console.ReadLine` method when executed waits for the user to type a string of characters and press the Enter key

To input integer values, the `Convert` class' `ToInt32` method is used with the `ReadLine` method

```
int myNumber = Convert.ToInt32(Console.ReadLine());
```

# What is DLL

A DLL is a library that contains code and data that can be used by more than one program at the same time.

For example, in Windows operating systems, the Comdlg32 DLL performs common dialog box related functions.

A program can be modularized into separate components using DLL.

# Creating and using DLL

A DLL class will not contain the main method.

A DLL file is created just like any other C# class with methods.

while compiling a library it should be specified as target:library so that the output is stored with the .dll extension

To use a dll file with any executable file it has to be included using the namespace.

A dynamic linking library (DLL) is linked to your program at run time.

# Creating DLL

```
namespace UtilityMethod

{
    public class MathsUtility
    {
        public static long Multiply(long x, long y)
        {
            return (x * y);
        }
        public static long Add(long x, long y)
        {
            return (x + y);
        }
    }
}
```

To compile the program issue the following command

```
csc /target:library /out:MathLibrary.DLL MathsUtility.cs
```

# Using DLL

```
using UtilityMethod;

namespace LibTest
{
    class Program
    {
        static void Main(string[] args)
        {
            int num1 = 345;
            int num2 = 456;
            long product = MathsUtility.Multiply(num1,num2);
            Console.WriteLine(product);
            long sum = MathsUtility.Add(num1, num2);
            Console.WriteLine(sum);
            Console.ReadKey();
        }
    }
}
```

# Summary

- Features of C#
- C# Compilation and Execution
- General Structure of a C# Program
- Creating and Using a DLL



# C# FUNDAMENTALS



# Objectives

In this module you will learn

- Data Types in C#
- Value Types and Reference Types
- Boxing and UnBoxing
- Enum
- Operators
- Nullables



# Variables

A variable is a name given to a storage area that programs can manipulate

Variables store the assigned values in temporary memory locations

Variables have the following features:

- Name
- Address
- Data Type
- Value
- Scope
- Lifetime

# Data Types in C#

C# is a strongly typed language

Every variable and constant has a type

Data types includes:

int

char

long

bool

float

DateTime

double

string

decimal

# Common Type System (CTS)



Common Type System(CTS) is a rich type system supported by .NET framework

- It defines all the types and operations found in most programming languages

Each type in the CTS is defined as either

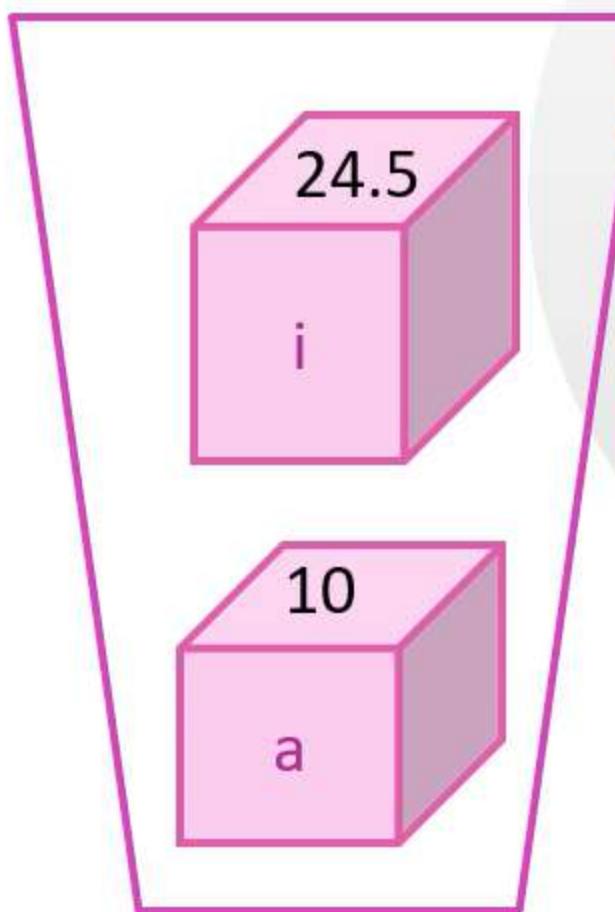
- a value type or
- a reference type

This includes all custom types in the .NET Framework class library and also user-defined types

# Value Types

## Value Types

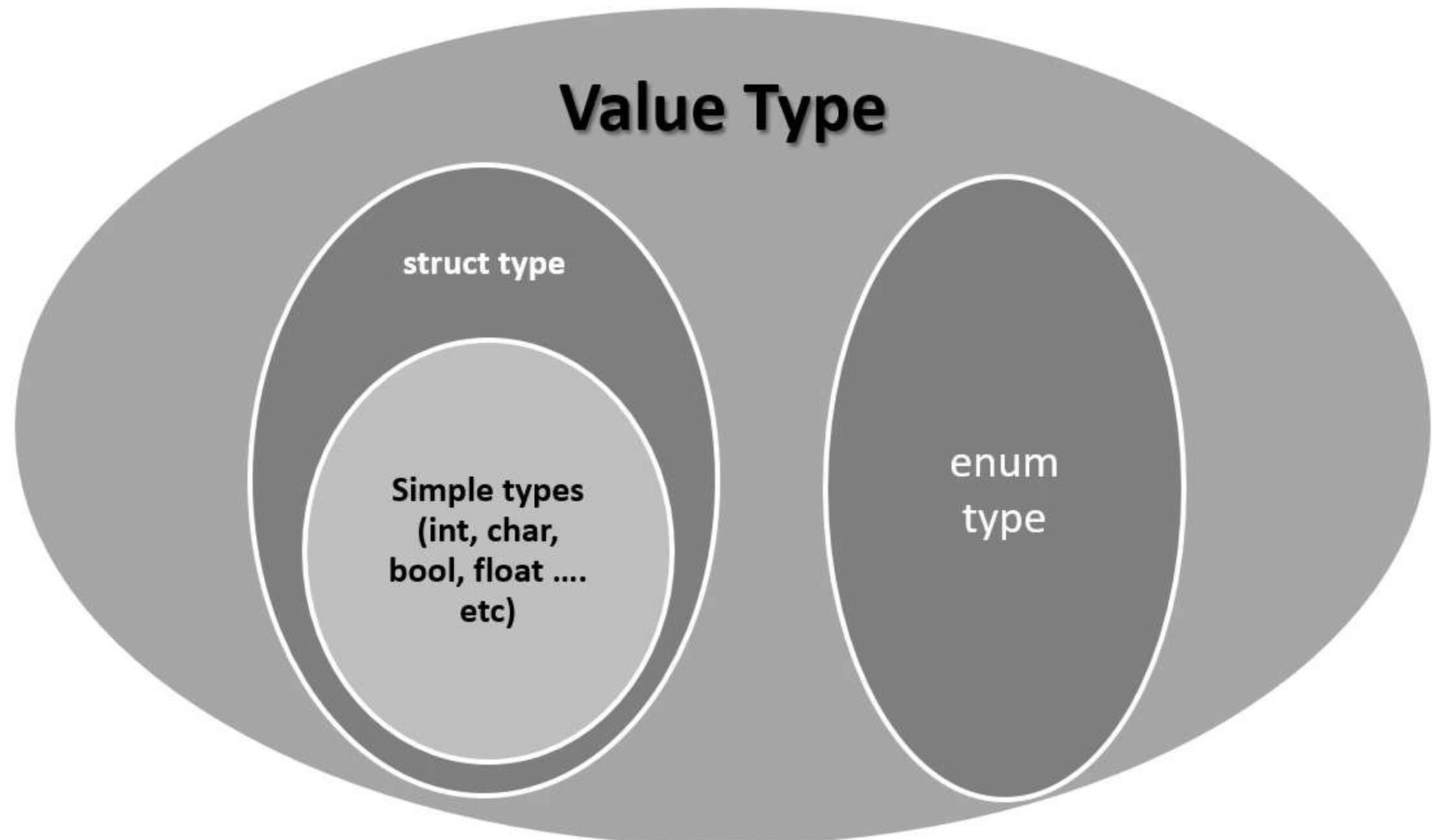
directly contain their values



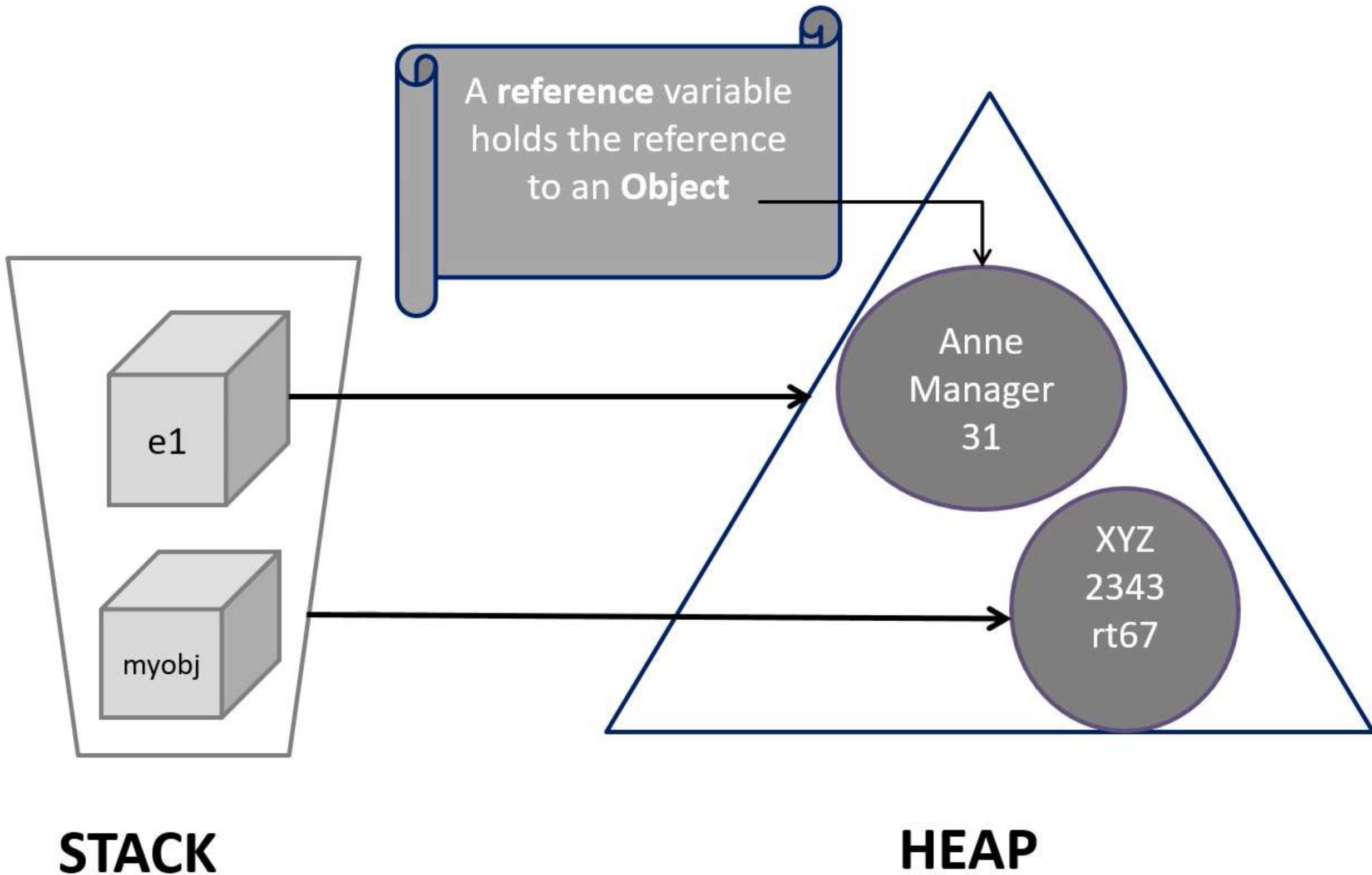
The memory is allocated inline  
when the variable is declared

Value is stored in the stack

# Value Type



# Reference Types



# Reference Types

A reference type can be a class, array, or interface .

At run time, a reference type variable contains null value until an instance of an object is created by the new operator, or assigned an object that has been created using new.

When the object is created, the memory is allocated on the managed heap.

Eg., MyClass obj = new MyClass();

MyClass obj1 = obj

# Value Types - Integral Types

Name	CTS Type	Size (bytes)	Description
<b>sbyte</b>	System.SByte	1	Ranges from -128 to 127
<b>short</b>	System.Int16	2	Ranges from -32,768 to 32,767
<b>int(default)</b>	System.Int32	4	Ranges from -2,147,483,648 to 2,147,483,647
<b>long</b>	System.Int64	8	Ranges from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
<b>byte</b>	System.Byte	1	Ranges from 0 to 255
<b>ushort</b>	System.UInt16	2	Ranges from 0 to 65535
<b>uint</b>	System.UInt32	4	Ranges from 0 to 4,294,967,295
<b>ulong</b>	System.UInt64	8	Ranges from 0 to 18,446,744,073,709,551,615
<b>char</b>	System.Char	2	Represents a single 16-bit (Unicode) character

# Floating Point Types and Decimal Types

Name	CTS Type	Size (bytes)	Description
<b>float</b>	System.Single	4	Ranges from $\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ with 7 digits precision. Requires the 'f' or 'F'
<b>double(default)</b>	System.Double	8	Ranges from $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ with 15-16 digits precision.

Name	CTS Type	Size (bytes)	Description
<b>decimal</b>	System.Decimal	16	Ranges from $\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$ with 28-29 digits precision. Requires the 'm' or 'M'

# Boolean Type and Reference Type

Name	CTS Type	Size (bytes)	Description
bool	System.Boolean	1	Contains either true or false

Name	CTS Type	Description
object	System.Object	The root type, from which all other types in the CTS derive (including value types)
string	System.String	Unicode character string

# Declaring and Assigning Variables



// Declaration

- `DataType variableName;`
- `DataType name1, name2;`
- `int myNum;`
- `float mark1,mark2,mark3;`



//Assignment

- `byte num = 10;`
- `int i = 5;`
- `char c = 'Z';`
- `float mark1=89;`

# Type Conversions

## Implicit conversions

- No type casting required
- Ex: int i=35;
- long l=i;

## Explicit conversions

- Type casting must be done explicitly
- Ex: long lg=24587L;  
byte b=(byte) lg;
- Loss of information may be there

# System.Convert class

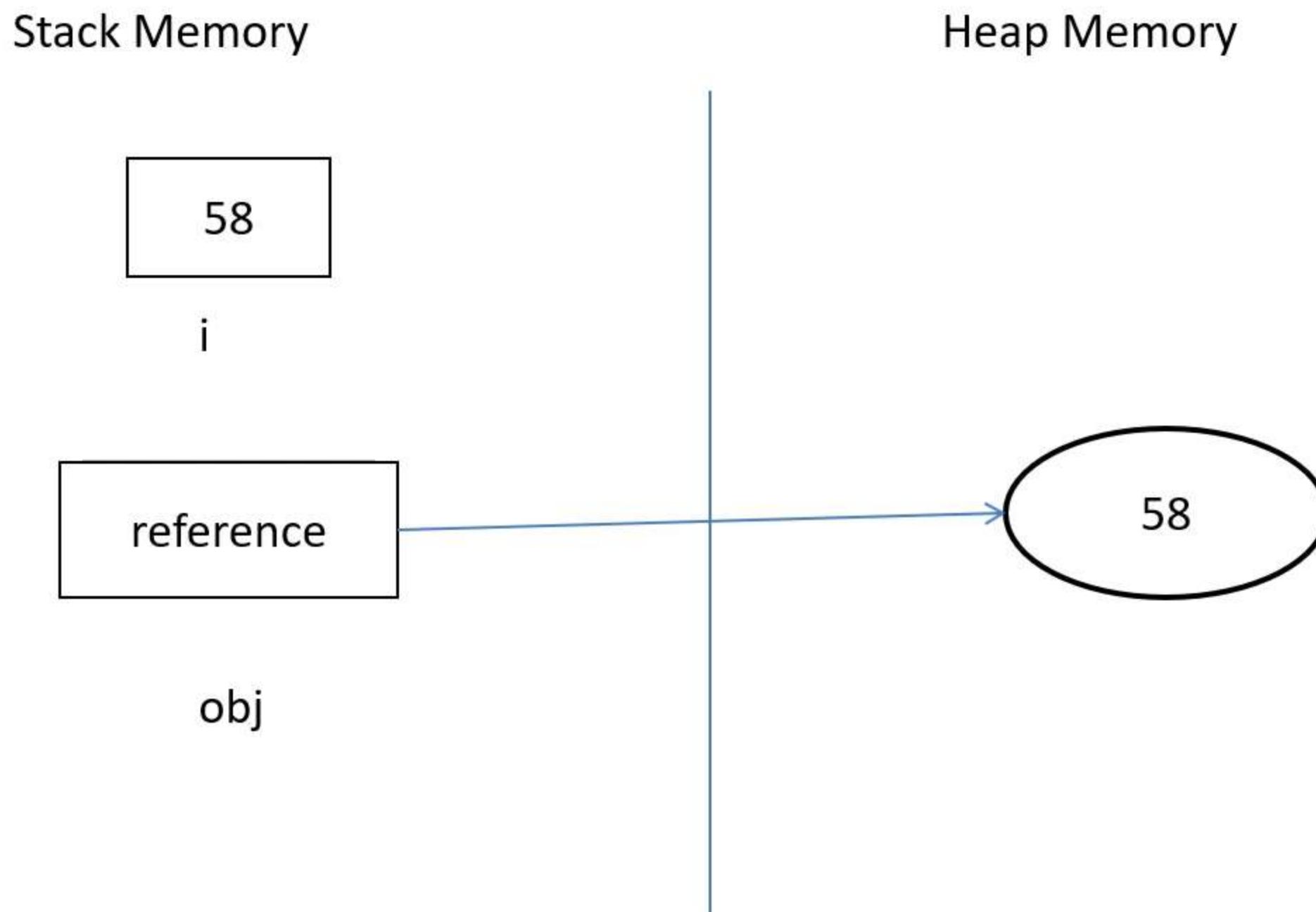
Converts a base data type to another base data type.

The Convert class contains a number of methods like ToInt32(), ToByte(), ToDecimal(), ToDateTime() ..etc. to convert a data type to the specified data type

```
string s="100";
int num= Convert.ToInt32(s);
//converts string s into an int data type
```

```
double d1 = 56.5;
int i = Convert.ToInt32(d1);
//Converts double d1 in to an int and hence only the value 56 is stored in i
where as the decimal part is truncated.
```

# Boxing



# Unboxing

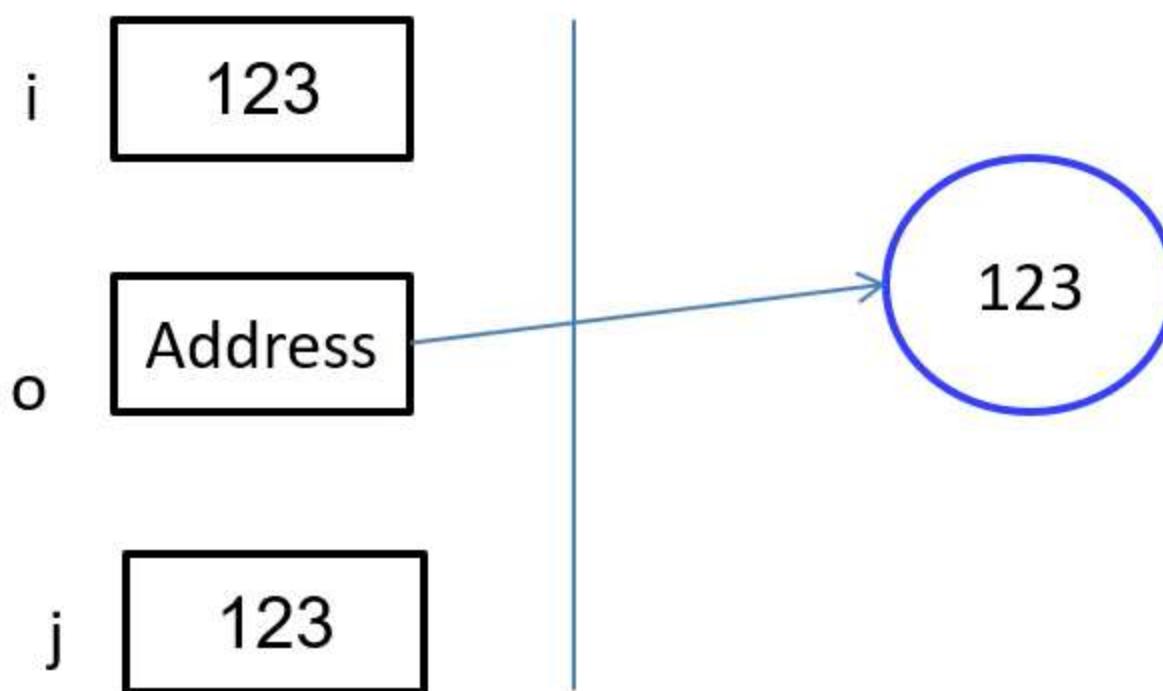
Unboxing is an explicit conversion from the type object to a value type or from an interface type to a value type that implements the interface

An unboxing operation consists of:

Checking the object instance to make sure that it is a boxed value of the given value type

Copying the value from the instance into the value-type variable

```
int i = 123; // a value type  
  
Object o = i; // boxing  
  
int j = (int)o; // unboxing
```



# Characters

// Assign an actual character with the single quotes

- char charData = 'a'

// Print the actual value of the char variable

- Console.WriteLine(charData);

// Print the unicode integer value corresponding to 'a'

- Console.WriteLine((int)charData);

# Enum

---

An enumeration is a set of named integer constants.

---

An enumerated type is declared using  
the **enum** keyword.

---

C# enumerations are value data type.

---

In other words, enumeration contains its own values  
and cannot inherit or cannot pass inheritance.

# Enum syntax

## Syntax:

```
enum <enum_name> {  
    enumeration list  
};
```

## Example:

```
enum Days  
{ Sun, Mon, tue, Wed, thu, Fri, Sat };
```

# Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

C# has rich set of built-in operators and provides the following type of operators –

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Miscellaneous Operators

# Miscellaneous Operators

Operator	Description	Example
sizeof()	Returns the size of a data type.	sizeof(int), returns 4.
typeof()	Returns the type of a class.	typeof(StreamReader);
&	Returns the address of an variable.	&a; returns actual address of the variable.
*	Pointer to a variable.	*a; creates pointer named 'a' to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y
is	Determines whether an object is of a certain type.	If( Ford is Car) // checks if Ford is an object of the Car class.
as	Cast without raising an exception if the cast fails.	Object obj = new StringReader("Hello");StringReader r = obj as StringReader;

# Nullable Types

- Nullable types are instances of the System.Nullable<T> struct.
- A nullable type can represent the correct range of values for its underlying value type, plus an additional null value.
- For example, a Nullable<SByte>, pronounced "Nullable of Int32," can be assigned any value from -128 to 127, or it can be assigned the null value.
- A Nullable<bool> can be assigned the values true, false, or null.

# Use of Nullable type

- Value types cannot be assigned null value.
- Numeric types and boolean type are value types and hence cannot have null values.
- While dealing with databases we may come across undefined columns.
- A nullable data type will be highly useful in dealing with databases and other data types which need to be assigned null in certain scenarios.

# Defining Nullable type

- A nullable variable type is defined by suffixing a question mark symbol(?) to the data type.
- A question mark symbol can only applied to a value type.

```
int? number =10;  
bool? isValid=null;  
char? grade= null;
```

# 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**

# Example

```
class Program {  
  
    static void Main(string[] args) {  
        double? num1 = null;  
        double? num2 = 6.32123;  
        double num3;  
        num3 = num1 ?? 9.77;  
        Console.WriteLine(" Value of num3: {0}", num3);  
        num3 = num2 ?? 9.77;  
        Console.WriteLine(" Value of num3: {0}", num3);  
        Console.ReadLine(); } }
```

## Output:

Value of num3: 9.77

Value of num3: 6.32123

# Summary

- Data Types in C#
- Value Types and Reference Types
- Boxing and UnBoxing
- Enum
- Operators
- Nullables





TEKNOTURF

# C# CONTROL STATEMENTS



# Objectives

In this module you will learn

- Decision Making
- Looping
- Sample Code Demo



# Decision Making

- Decision making structures requires the programmer to specify one or more conditions to be evaluated or tested by the program.
- Along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Sr.No.	Statement & Description
1 if statement	An <b>if statement</b> consists of a boolean expression followed by one or more statements.
2 if..else statement	An <b>if statement</b> can be followed by an optional <b>else statement</b> , which executes when the boolean expression is false.
3 nested if	You can use one <b>if</b> or <b>else if</b> statement inside another <b>if</b> or <b>else if</b> statement(s).
4 switch	A <b>switch</b> statement allows a variable to be tested for equality against a list of values.
5 nested switch	You can use one <b>switch</b> statement inside another <b>switch</b> statement(s).

# Looping

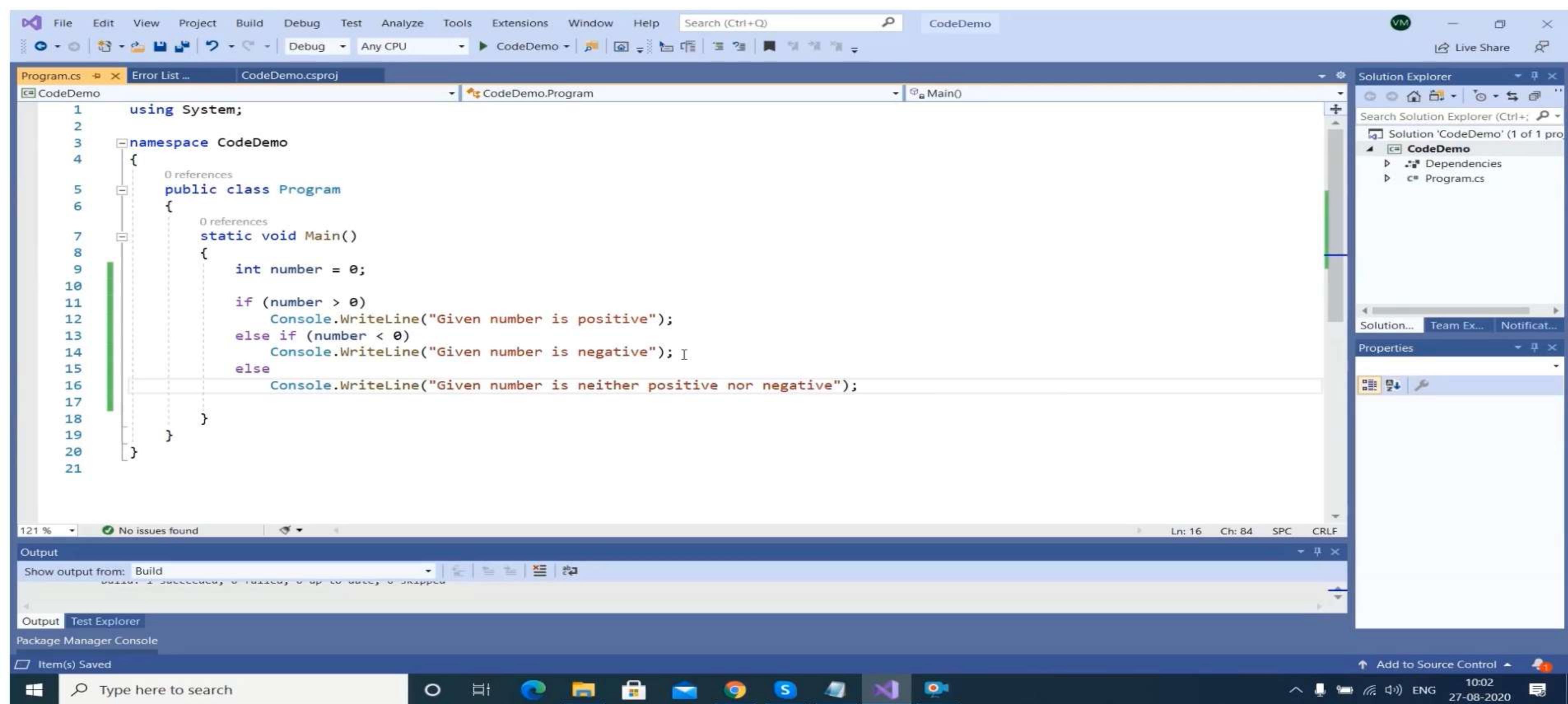
- A loop statement allows us to execute a statement or a group of statements multiple times and following is the general form of a loop statement in most of the programming languages.
- C# provides following types of loop to handle looping requirements.

Sr.No.	Loop Type & Description
1 while loop	It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.
2 for loop	It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3 do ... while	It is similar to a while statement, except that it tests the condition at the end of the loop body
4 nested loops	You can use one or more loop inside any another while, for or do..while loop.



# Code Demo

# If-else Demo



A screenshot of the Microsoft Visual Studio IDE interface. The main window displays a C# code editor with the following content:

```
1  using System;
2
3  namespace CodeDemo
4  {
5      public class Program
6      {
7          public static void Main()
8          {
9              int number = 0;
10
11             if (number > 0)
12                 Console.WriteLine("Given number is positive");
13             else if (number < 0)
14                 Console.WriteLine("Given number is negative");
15             else
16                 Console.WriteLine("Given number is neither positive nor negative");
17
18         }
19     }
20 }
21 }
```

The code implements a simple if-else statement to determine if a given number is positive, negative, or zero. The solution explorer shows a single project named "CodeDemo" containing a "Program.cs" file. The output window indicates no issues found.

# If-else Demo



A screenshot of Microsoft Visual Studio showing a C# code example for an if-else statement. The code checks if a number is positive, negative, or zero. The output window shows the program's execution and its result.

```
1  using System;
2
3  namespace CodeDemo
4  {
5      public class Program
6      {
7          static void Main()
8          {
9              int number = 0;
10
11             if (number > 0)
12                 Console.WriteLine("Given number is positive");
13             else if (number < 0)
14                 Console.WriteLine("Given number is negative");
15             else
16                 Console.WriteLine("Given number is neither positive nor negative");
17         }
18     }
19 }
20
21 }
```

Microsoft Visual Studio Debug Console

```
Given number is neither positive nor negative
C:\Users\Velu\source\repos\CodeDemo\CodeDemo\bin\Debug\netcoreapp3.1\CodeDemo.exe (process 2376) exited with code 0.
Press any key to close this window . . .
```

Output

Show output from: Build

Output Test Explorer

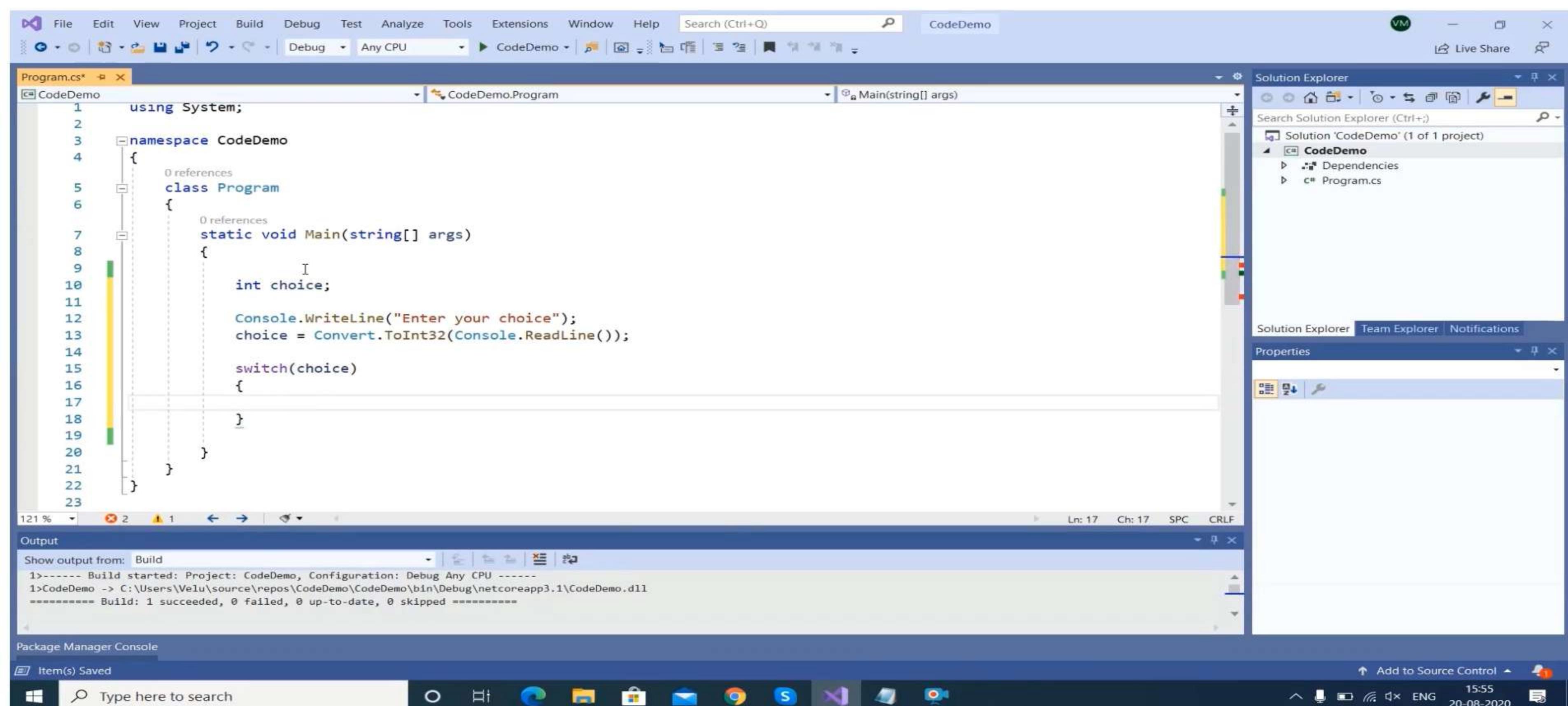
Package Manager Console

Build succeeded

Type here to search

10:02 27-08-2020

# Switch Demo



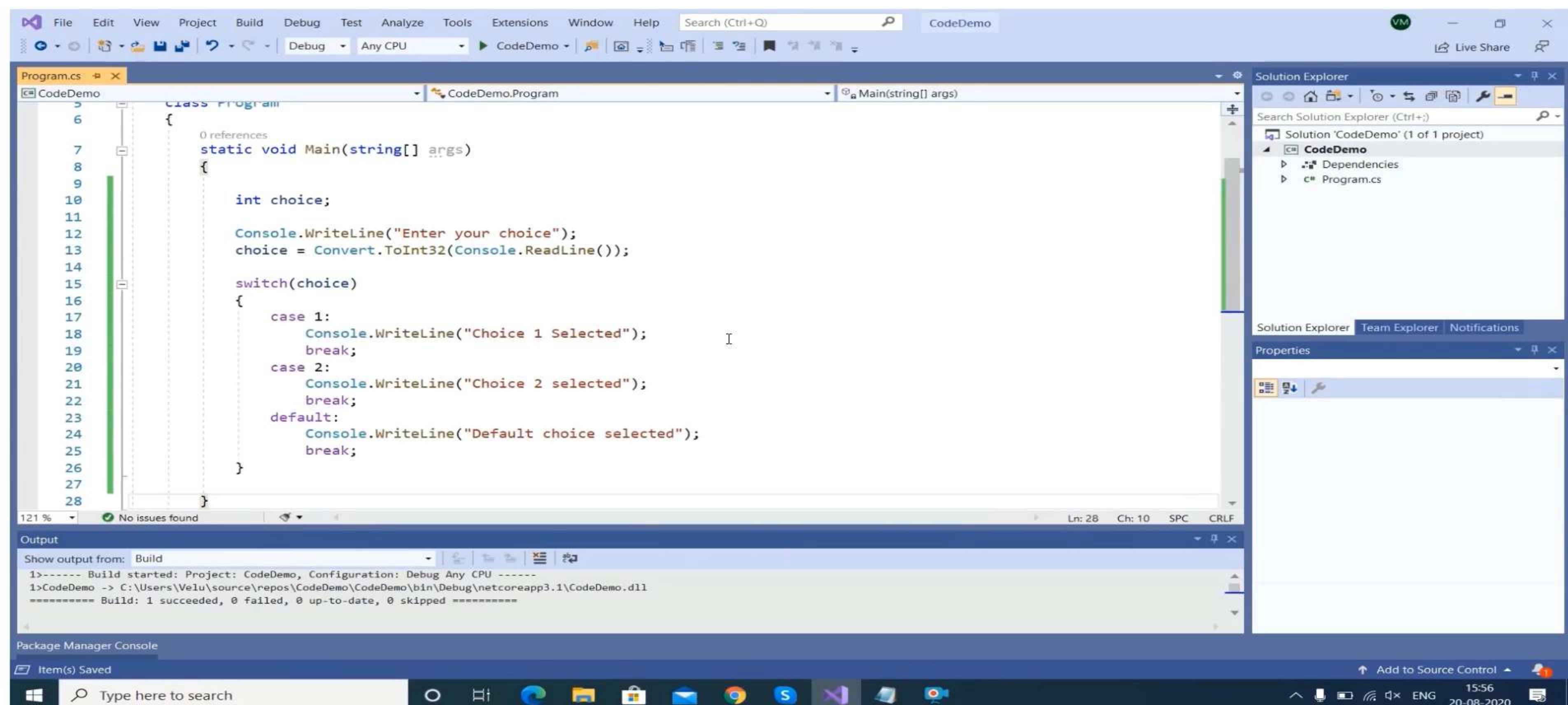
The screenshot shows a Microsoft Visual Studio interface with the following details:

- File Menu:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search (Ctrl+Q).
- Solution Explorer:** Shows a single project named "CodeDemo" with one file "Program.cs".
- Properties Window:** Available on the right side of the interface.
- Output Window:** Shows build logs:

```
1>----- Build started: Project: CodeDemo, Configuration: Debug Any CPU -----
1>CodeDemo -> C:\Users\Velu\source\repos\CodeDemo\CodeDemo\bin\Debug\netcoreapp3.1\CodeDemo.dll
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ======
```
- Code Editor:** Displays the "Program.cs" code:

```
1  using System;
2
3  namespace CodeDemo
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              int choice;
10
11             Console.WriteLine("Enter your choice");
12             choice = Convert.ToInt32(Console.ReadLine());
13
14             switch(choice)
15             {
16
17
18             }
19         }
20     }
21 }
22 }
```
- Task List:** Shows 2 errors and 1 warning.
- StatusBar:** Shows the current window is "CodeDemo" and the status is "VM".
- System Tray:** Shows the date and time as 20-08-2020 15:55.

# Switch Demo



A screenshot of the Microsoft Visual Studio IDE interface. The main window shows the code editor with a C# program named 'Program.cs'. The code demonstrates a switch statement:

```
class Program
{
    static void Main(string[] args)
    {
        int choice;

        Console.WriteLine("Enter your choice");
        choice = Convert.ToInt32(Console.ReadLine());

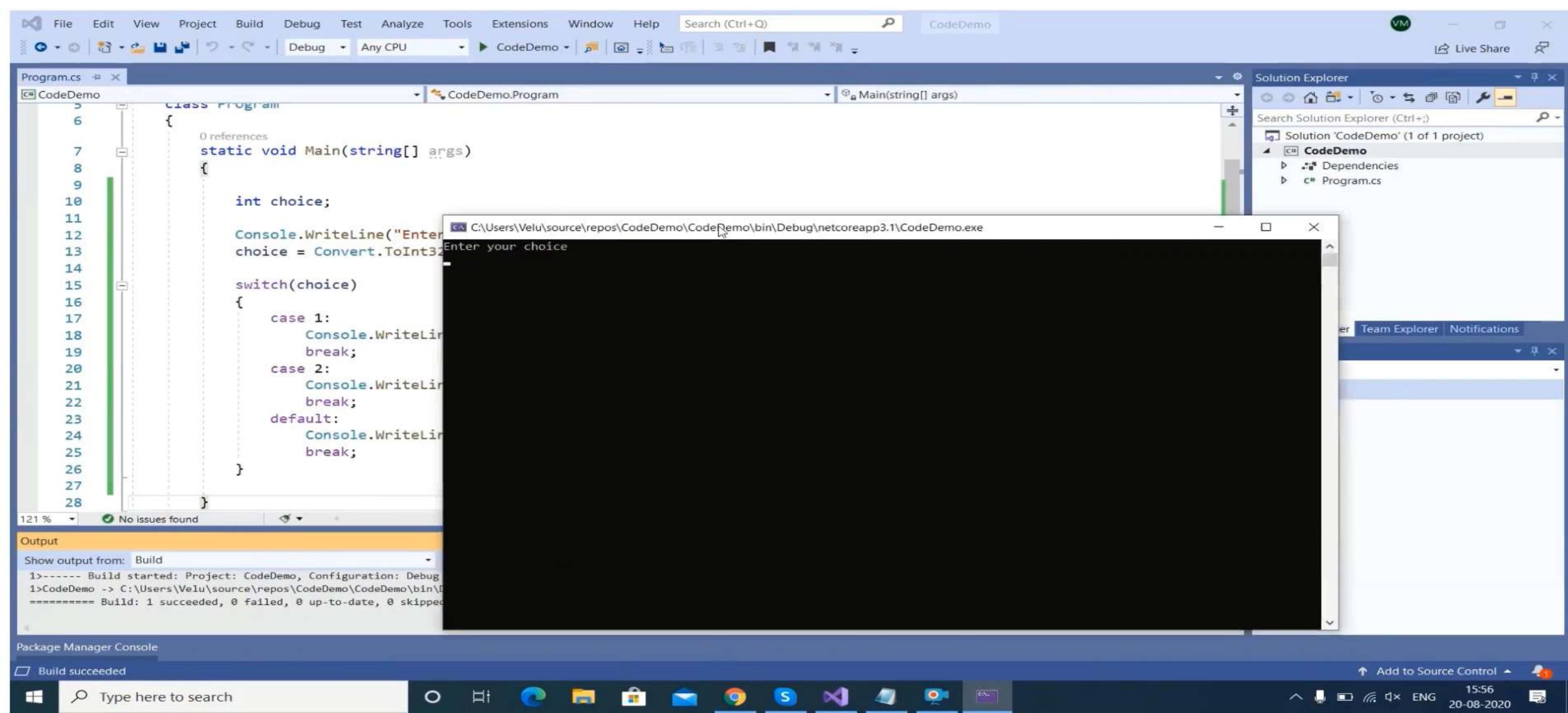
        switch(choice)
        {
            case 1:
                Console.WriteLine("Choice 1 Selected");
                break;
            case 2:
                Console.WriteLine("Choice 2 selected");
                break;
            default:
                Console.WriteLine("Default choice selected");
                break;
        }
    }
}
```

The Solution Explorer panel on the right shows a project named 'CodeDemo' with a single file 'Program.cs'. The Output window at the bottom displays the build log:

```
1>----- Build started: Project: CodeDemo, Configuration: Debug Any CPU -----
1>CodeDemo -> C:\Users\Velu\source\repos\CodeDemo\CodeDemo\bin\Debug\netcoreapp3.1\CodeDemo.dll
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

The status bar at the bottom indicates the system is running on 'VM' and shows the date and time as '20-08-2020 15:56'.

# Switch Demo



The screenshot shows a Microsoft Visual Studio interface with the following details:

- Project:\*\*** CodeDemo
- File:\*\*** Program.cs
- Code Editor Content:\*\***

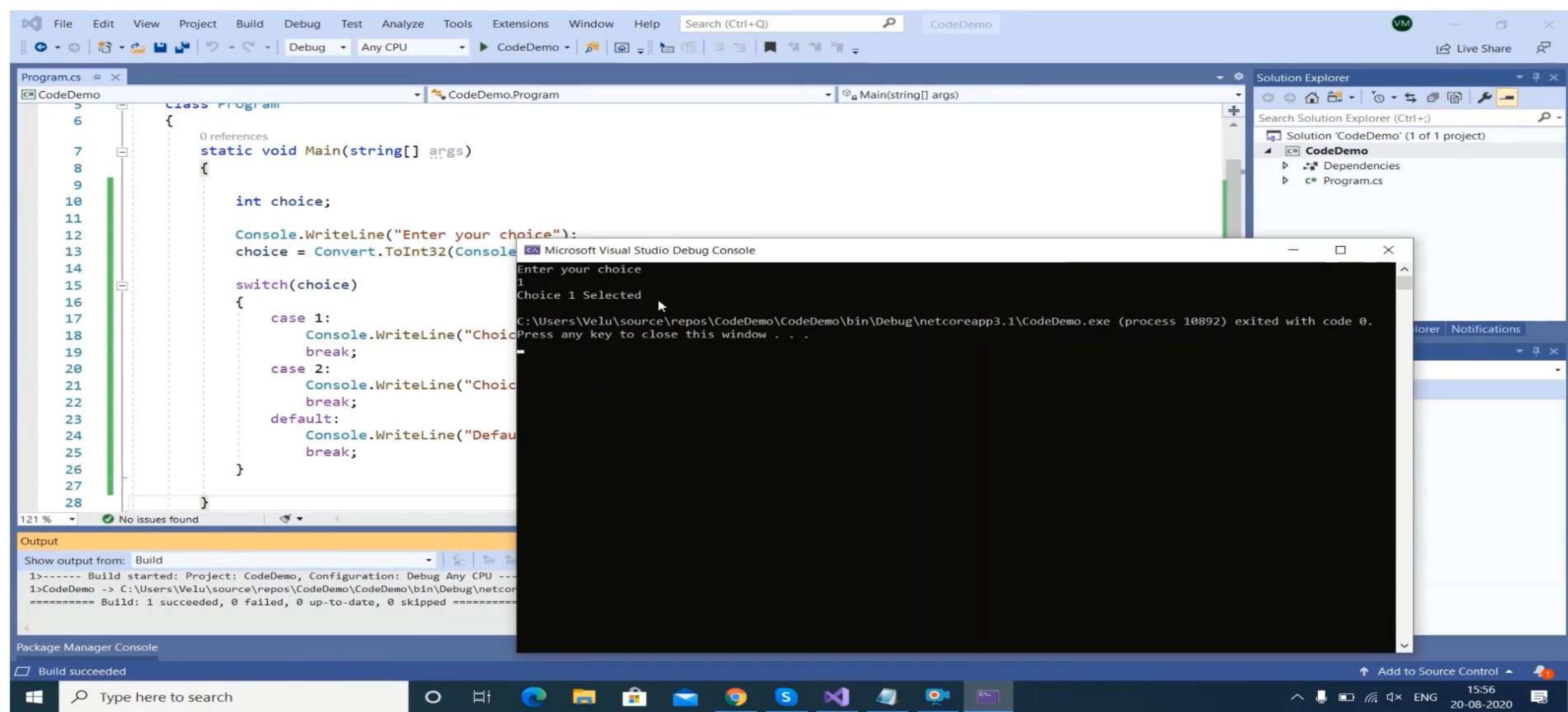
```
class Program
{
    static void Main(string[] args)
    {
        int choice;

        Console.WriteLine("Enter your choice");
        choice = Convert.ToInt32(Console.ReadLine());

        switch(choice)
        {
            case 1:
                Console.WriteLine("Choice 1 selected");
                break;
            case 2:
                Console.WriteLine("Choice 2 selected");
                break;
            default:
                Console.WriteLine("Default selected");
                break;
        }
    }
}
```
- Output Window:\*\***

```
1>----- Build started: Project: CodeDemo, Configuration: Debug -----
1>CodeDemo -> C:\Users\Velu\source\repos\CodeDemo\CodeDemo\bin\Debug\netcoreapp3.1\CodeDemo.exe
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
```
- Taskbar:\*\*** Shows various application icons including File Explorer, Task View, Start, Edge, Mail, Google Chrome, and others.

# Switch Demo



The screenshot shows a Microsoft Visual Studio interface with the following details:

- Project:** CodeDemo
- File:** Program.cs
- Code:** A C# program demonstrating a switch statement.

```
class Program
{
    static void Main(string[] args)
    {
        int choice;

        Console.WriteLine("Enter your choice");
        choice = Convert.ToInt32(Console.ReadLine());

        switch(choice)
        {
            case 1:
                Console.WriteLine("Choice 1 Selected");
                break;
            case 2:
                Console.WriteLine("Choice 2 Selected");
                break;
            default:
                Console.WriteLine("Default Selection");
                break;
        }
    }
}
```
- Output Window:** Shows the console output from running the program. The user entered "1", which triggered the "Choice 1 Selected" message.
- Solution Explorer:** Shows the project structure with files: CodeDemo, Dependencies, and Program.cs.
- Task List:** Shows a single item: "No issues found".
- Status Bar:** Shows "Build succeeded" and the system tray with various icons.

# Do-while Demo



A screenshot of the Microsoft Visual Studio IDE interface. The main window displays a C# code editor with the file `Program.cs` open. The code implements a `do-while` loop to calculate the factorial of a number. The Solution Explorer on the right shows a single project named `CodeDemo` containing a file `Program.cs`. The Output window at the bottom shows a successful build. The taskbar at the bottom of the screen includes icons for File Explorer, Task View, Edge browser, File Explorer, Mail, Settings, File Explorer, and Camera.

```
1  using System;
2
3  namespace CodeDemo
4  {
5      public class Program
6      {
7          static void Main()
8          {
9              int count = 1, num = 5, product;
10
11             do
12             {
13                 product = num * count;
14                 Console.WriteLine("{0} * {1} = {2}", num, count, product);
15                 count++;
16             } while (count <= 10);
17         }
18     }
19 }
20
21
```

Output

Show output from: Build

Build succeeded

121 % No issues found

Ln: 16 Ch: 35 OVR SPC CRLF

Output Test Explorer

Package Manager Console

Add to Source Control

15:52 ENG 27-08-2020

# Do-while Demo



A screenshot of the Microsoft Visual Studio IDE interface. The main window shows the code editor with a C# program named 'Program.cs'. The code demonstrates a do-while loop that prints the multiplication table of 5 from 1 to 10. The output window displays the results of the execution. The Solution Explorer shows the project structure with files like 'CodeDemo.csproj' and 'Program.cs'.

```
1  using System;
2
3  namespace CodeDemo
4  {
5      public class Program
6      {
7          static void Main()
8          {
9              int count = 1, num = 5, product;
10
11             do
12             {
13                 product = num * count;
14                 Console.WriteLine("{0}", product);
15                 count++;
16             } while (count <= 10);
17
18         }
19     }
20 }
```

Microsoft Visual Studio Debug Console

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50

C:\Users\Velu\source\repos\CodeDemo\CodeDemo\bin\Debug\netcoreapp3.1\CodeDemo.exe (process 14508) exited with code 0.
Press any key to close this window . . .
```

Output

Show output from: Build

Output Test Explorer

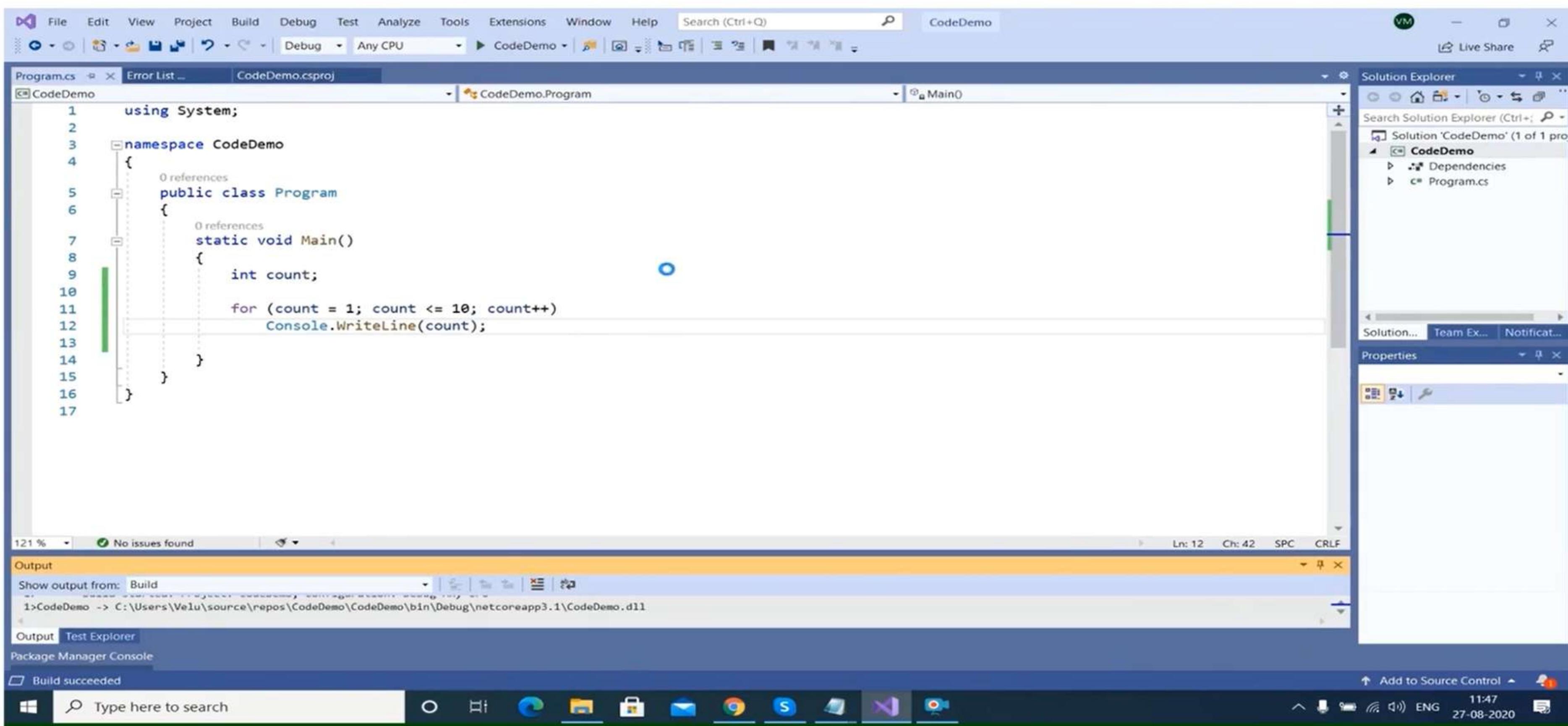
Package Manager Console

Build succeeded

Type here to search

15:52 27-08-2020

# For Demo



CodeDemo

Program.cs

```
1  using System;
2
3  namespace CodeDemo
4  {
5      public class Program
6      {
7          static void Main()
8          {
9              int count;
10
11             for (count = 1; count <= 10; count++)
12                 Console.WriteLine(count);
13
14         }
15     }
16 }
17
```

No issues found

Output

Show output from: Build

1>CodeDemo -> C:\Users\Velu\source\repos\CodeDemo\CodeDemo\bin\Debug\netcoreapp3.1\CodeDemo.dll

Build succeeded

Type here to search

11:47 ENG 27-08-2020

# For Demo



A screenshot of the Microsoft Visual Studio IDE interface. The main window shows a C# code editor with the file `Program.cs` open. The code contains a simple program that prints numbers from 1 to 10 to the console. The output window below shows the execution of the program, displaying the numbers 1 through 10, followed by the message "C:\Users\Velu\source\repos\CodeDemo\CodeDemo\bin\Debug\netcoreapp3.1\CodeDemo.exe (process 6852) exited with code 0. Press any key to close this window . . .". The Solution Explorer on the right shows the project structure with files `CodeDemo.csproj`, `Dependencies`, and `Program.cs`. The status bar at the bottom indicates a build succeeded.

```
1  using System;
2
3  namespace CodeDemo
4  {
5      public class Program
6      {
7          static void Main()
8          {
9              int count;
10
11             for (count = 1; count <= 10; count++)
12                 Console.WriteLine(count);
13
14         }
15     }
16 }
```

Microsoft Visual Studio Debug Console

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

C:\Users\Velu\source\repos\CodeDemo\CodeDemo\bin\Debug\netcoreapp3.1\CodeDemo.exe (process 6852) exited with code 0.  
Press any key to close this window . . .

121 % No issues found

Output

Show output from: Build

Output Test Explorer

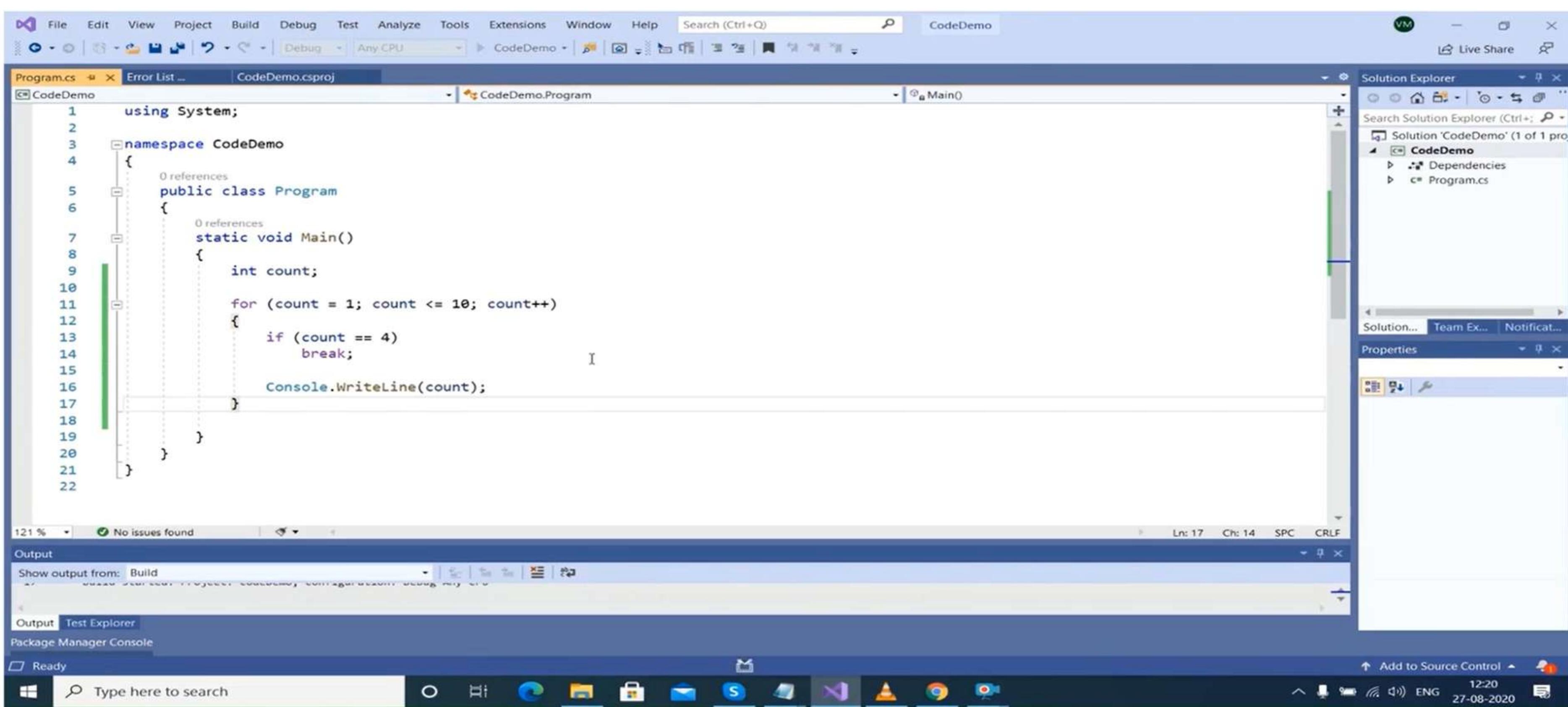
Package Manager Console

Build succeeded

Type here to search

11:47 ENG 27-08-2020

# Break and Continue Demo



The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays a C# code editor with the following code:

```
1  using System;
2
3  namespace CodeDemo
4  {
5      public class Program
6      {
7          static void Main()
8          {
9              int count;
10
11             for (count = 1; count <= 10; count++)
12             {
13                 if (count == 4)
14                     break;
15
16                 Console.WriteLine(count);
17             }
18         }
19     }
20 }
21
22 }
```

A green vertical bar in the margin indicates a break point is set on line 14. The status bar at the bottom shows "Ln: 17 Ch: 14 SPC CRLF".

The Solution Explorer sidebar shows a solution named "CodeDemo" containing a project named "CodeDemo" with files "Dependencies" and "Program.cs".

The Output window at the bottom left shows "No issues found".

The taskbar at the bottom includes icons for Start, Search, Task View, File Explorer, Mail, Settings, and a system tray with battery, signal, and date/time indicators.

# Break and Continue Demo



A screenshot of the Microsoft Visual Studio IDE interface. The main window shows the code editor with a C# program named 'Program.cs'. A green vertical bar on the left margin indicates a break point at line 10. The code contains a for loop that prints numbers from 1 to 10 to the console, but exits early at count == 4 due to a break statement. The 'Solution Explorer' pane on the right shows the project structure with files 'CodeDemo.csproj', 'Dependencies', and 'Program.cs'. The 'Debug Console' window at the bottom displays the output of the program, which only prints the numbers 1, 2, and 3 before exiting.

```
1  using System;
2
3  namespace CodeDemo
4  {
5      public class Program
6      {
7          static void Main()
8          {
9              int count;
10             for (count = 1; count <= 10; count++)
11             {
12                 if (count == 4)
13                     break;
14
15                 Console.WriteLine(count);
16             }
17         }
18     }
19 }
20
21 }
22 }
```

Microsoft Visual Studio Debug Console

```
1
2
3

C:\Users\Velu\source\repos\CodeDemo\CodeDemo\bin\Debug\netcoreapp3.1\CodeDemo.exe (process 17392) exited with code 0.
Press any key to close this window . . .
```

Output

Show output from: Build

Output Test Explorer

Package Manager Console

Build succeeded

Type here to search

121 % No issues found

VM Live Share

Solution Explorer

Search Solution Explorer (Ctrl+F)

Solution 'CodeDemo' (1 of 1 projects)

CodeDemo

Dependencies

Program.cs

12:20 27-08-2020 ENG

# Break and Continue Demo



A screenshot of the Microsoft Visual Studio IDE interface. The main window shows a C# code editor with the following code:

```
1  using System;
2
3  namespace CodeDemo
4  {
5      public class Program
6      {
7          static void Main()
8          {
9              int count;
10
11             for (count = 1; count <= 10; count++)
12             {
13                 if (count == 4)
14                     continue;
15
16                 Console.WriteLine(count);
17             }
18         }
19     }
20 }
21
22 }
```

The code includes a `continue` statement at line 14. A green vertical bar and a red dashed rectangle highlight the line `if (count == 4)`, indicating it is the current line of execution or a breakpoint. The Visual Studio interface also shows the Solution Explorer, Properties, and Output windows, and the taskbar at the bottom.

# Break and Continue Demo



A screenshot of the Microsoft Visual Studio IDE interface. The main window shows a code editor with C# code for a 'CodeDemo' project. A break point is set at line 10, which contains the line 'Console.WriteLine("Hello World");'. The output window displays the application's run results, showing it has exited with code 0. The Solution Explorer, Properties, and Task List panes are also visible on the right side of the interface.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

C:\Users\Velu\source\repos\CodeDemo\CodeDemo\bin\Debug\netcoreapp3.1\CodeDemo.exe (process 8676) exited with code 0.  
Press any key to close this window . . .

Output

Show output from: Build

Output Test Explorer

Package Manager Console

Build succeeded

Type here to search

121 % No issues found

Ln: 12 Ch: 14 SPC CRLF

12:21 27-08-2020

# Summary

- Decision Making
- Looping
- Sample Code Demo





TEKNOTURF

# C# CLASS AND OBJECTS



# Objectives

In this module you will learn

- OOP with C#
- Classes and Objects
- Constructors
- Structures in C#



# Classes and Objects

## What is an Object?

- Any real world entity with well defined properties is called an object
- Object will have a state, behaviour and a unique identity
- Objects can be tangible (physical entity) or intangible (cannot be touched or felt)



A wooden chair



A chemical process



A transaction

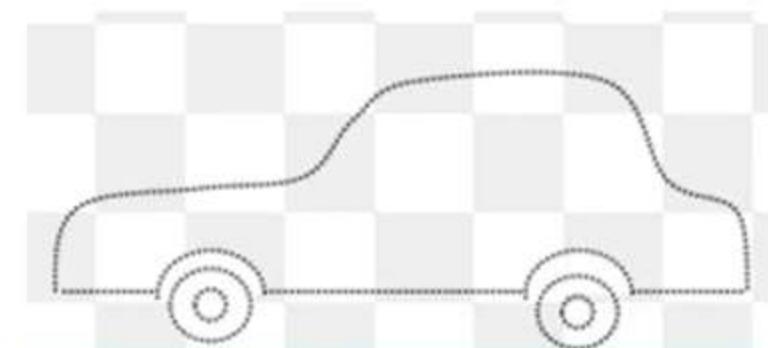
# Classes and Objects



What is a class?

- A *class* is a template or *blue print* that describes the behaviors/states that object of its type support
- A class defines the properties of objects and methods used to control an object's behavior

# Classes and Objects



Attributes :

model = "Mustang"

Color = "Yellow"

numOfPassengers = 0

amtOfGas = 16.5

Behaviour

Add / Remove Passenger

Get Tank Filled

Report when out of Gas

```
class Car
{
    //Attributes
    String model;
    String color;
    int numOfPassengers;
    float amtOfGas;

    //Methods
    void AddPassenger()
    {
        // do something
    }

    void RemovePassenger()
    {
        // do something
    }

    void FillTank()
    {
        // do something
    }

    void Report_OutOfGas()
    {
        // do something
    }
}
```

# Structure of a Class



Employee ID : 101  
Name : Rohith  
Salary : 25000.00

Employee: Object



Class: Employee

```
class Employee
{
    //Attributes
    int empID;
    String name;
    double salary;

    //Methods
    void PunchCard()
    {
        //do something
    }
    void DoProject()
    {
        //do something
    }
}
```

# Class Declaration

## Declaring a class

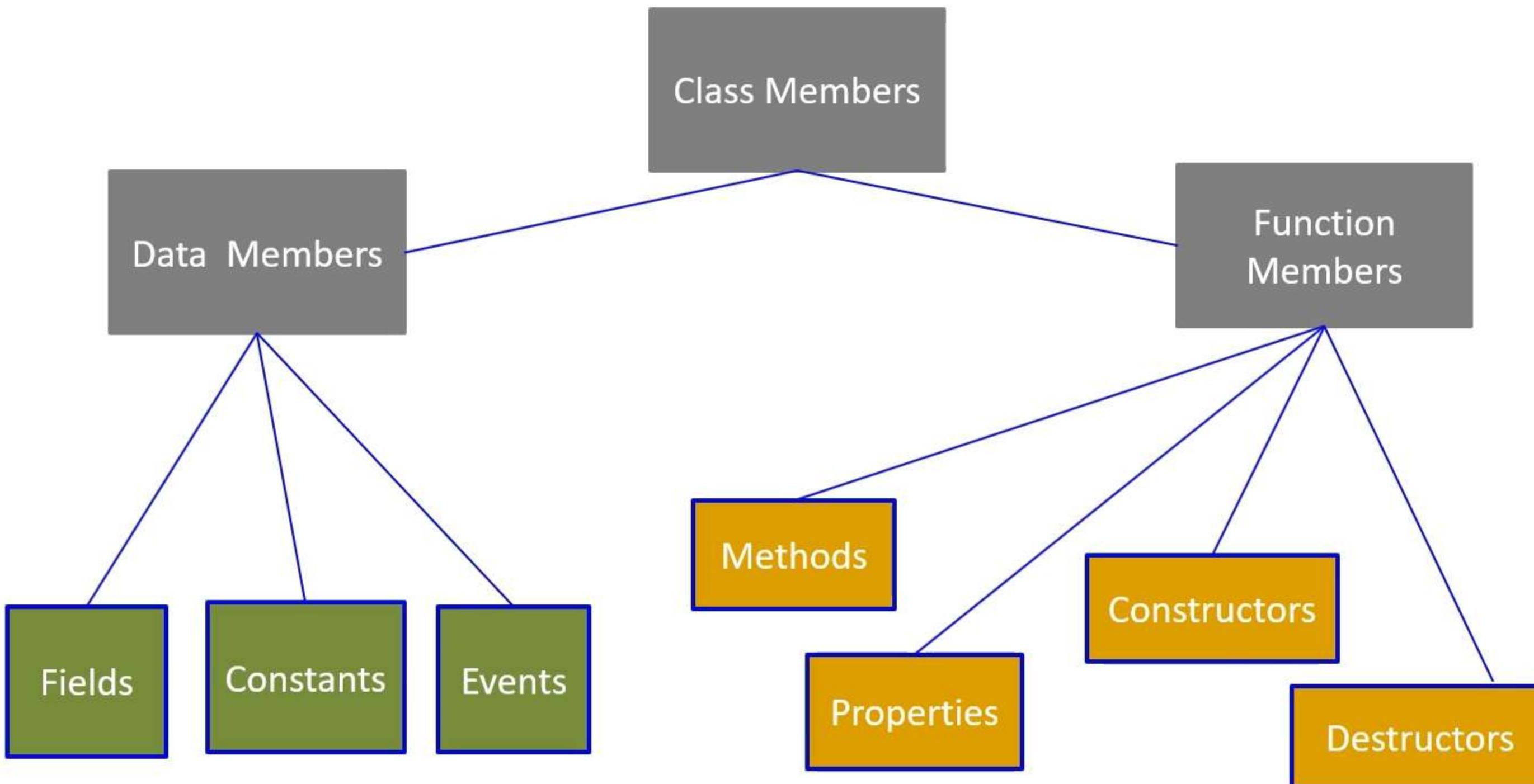
```
<modifier>* class <class_name> {  
    <attribute_declaration>*  
    <constructor_declaration>*  
    <method_declaration>*  
}
```

## Example:

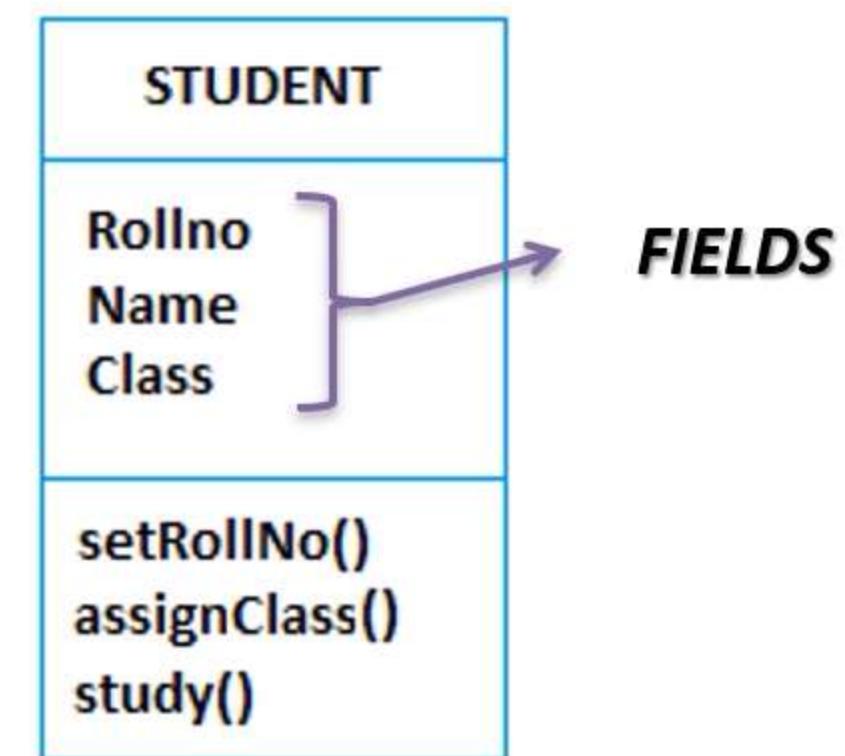
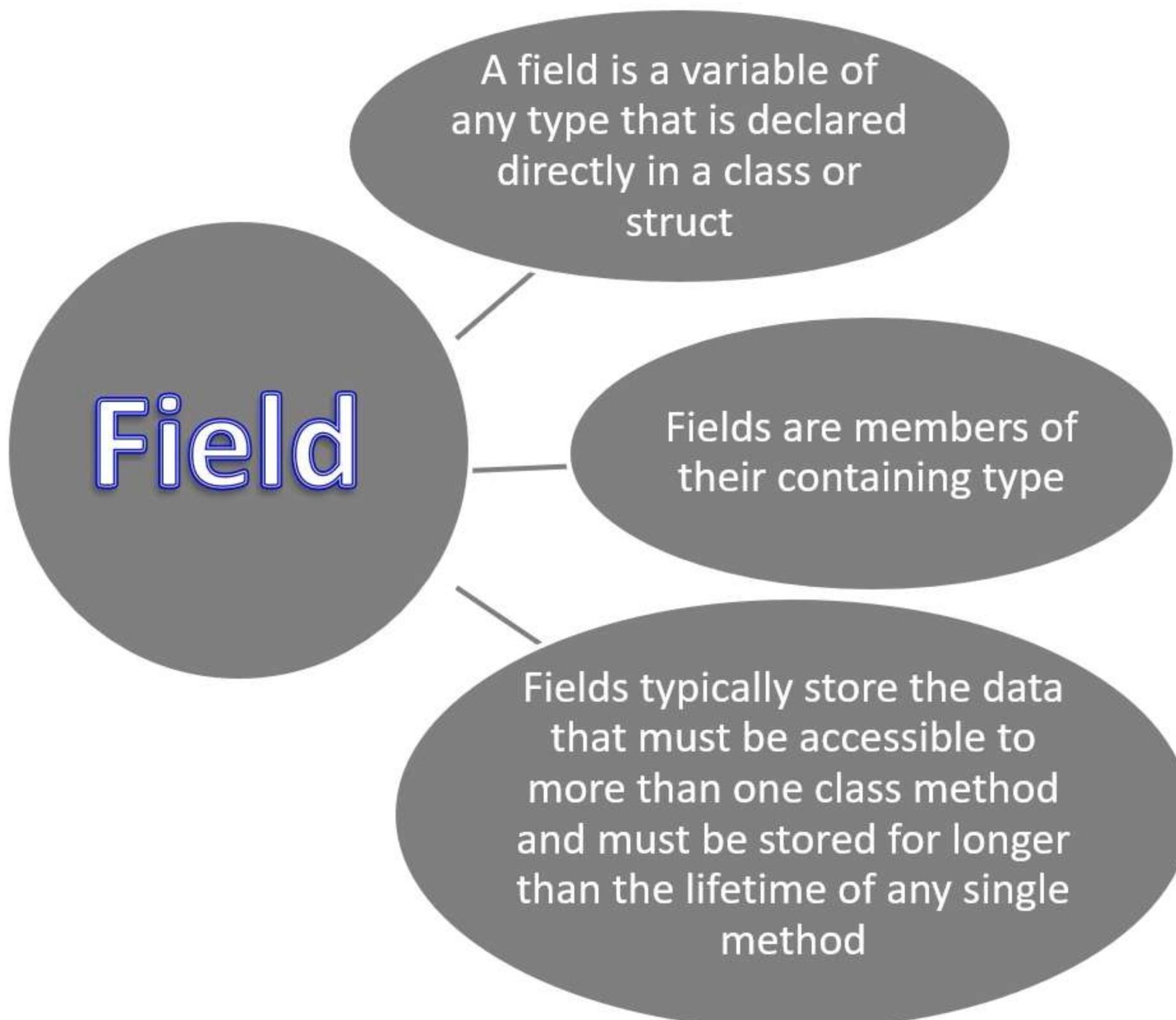
```
public class Employee {  
    private int emplId;  
    private String name;  
    private double salary;  
    public void SetEmplId(int id) {  
        emplId = id;  
    }  
}
```

# Members of class

The data and behaviors are the members of the class , and they include its fields, methods, properties, and events, and so on.



# Field



# Field

A class may have instance fields or static fields or both. Instance fields are specific to an instance of a type.

Example

```
private int i;  
static int j = 10;  
public byte b;
```

Fields in C# are set to default values automatically.

Numeric fields are set to zero.

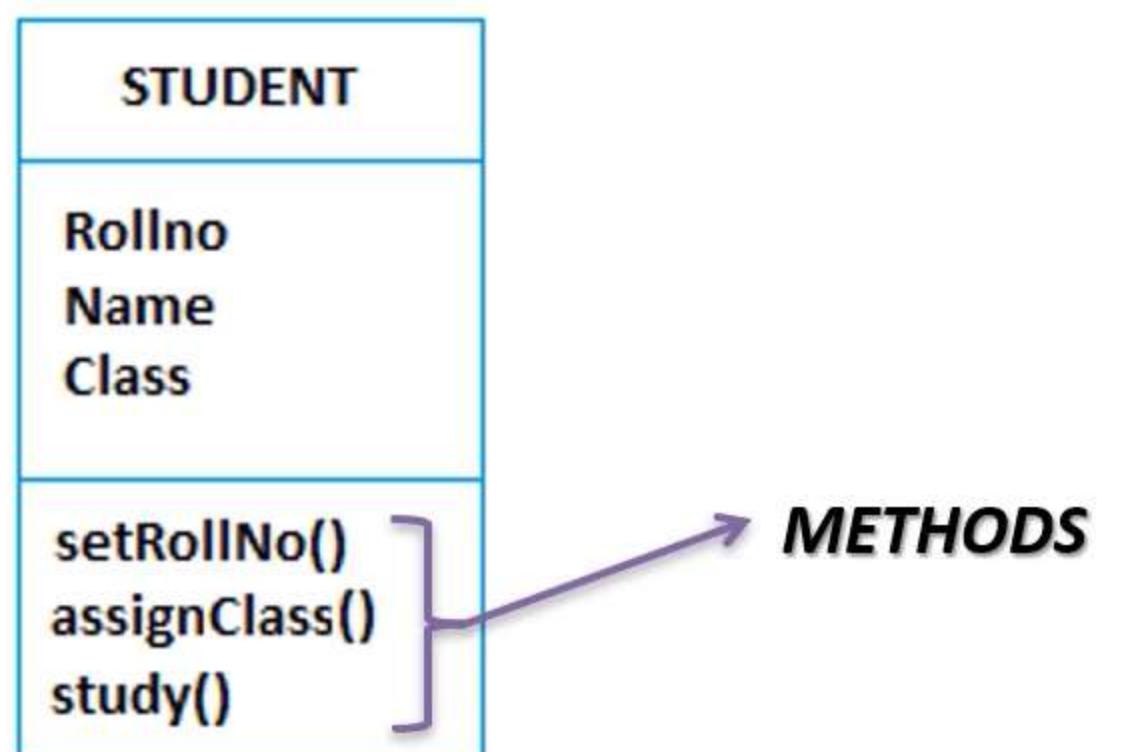
Boolean fields are set to false.

Reference fields are set to null

It is recommended that Fields should generally be **private**.

# Methods of Class

Methods are members of a class that provide a service for an object or perform some business logic



# Method Declaration

Methods describe the responsibility of the class. Methods are defined to perform some action on the fields that are within the class. Methods without the static keyword are called instance methods.

**Methods can accept parameters. Below is a basic syntax of a method:**

```
<visibility>* <modifier>* <return_type> <name> ( <argument>* ) {  
    <statement>*  
}
```

**Example:**

```
public void SetName(double nam) {  
    name = nam;  
}  
  
public int GetName() {  
    return name;  
}
```

# Objects

An object is basically a block of memory that has been allocated

A program may create many objects of the same class

Objects are also called instances, and they can be stored in either a named variable or in an array or collection

Because classes are reference types, a variable of a class object holds a reference to the address of the object on the managed heap

If a second object of the same type is assigned to the first object, then both variables refer to the object at that address

# Creating Object

## Creation of Object

- In C#, all Objects are created at run time in the heap area.

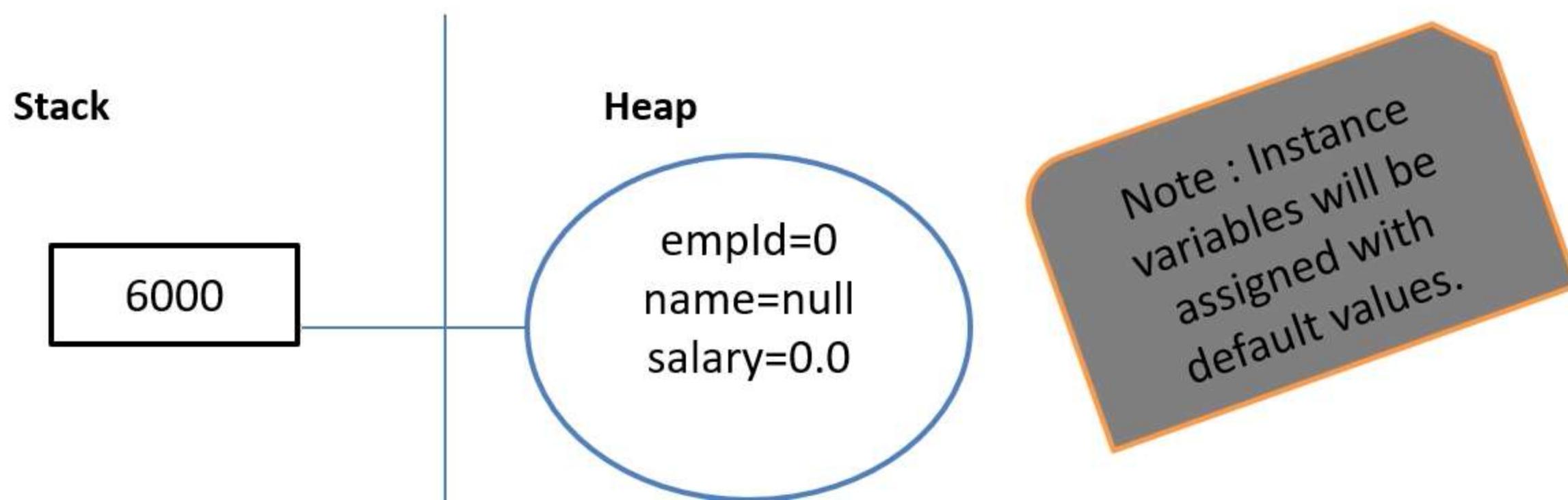
## Syntax

- `class_name reference_variable = new class_name(<parameter list>)`

## Example

- `Employee e=new Employee();`

Here variable 'e' is the name of the object. 'e' is the reference variable that holds the address of the Object created in Heap.



# Instantiating with new operator

Objects are initially unassigned



To use a class, we must first create an instance of a class



A new instance of a class is created using the **new** operator



The new operator allocates memory for the object



The new operator invokes the constructor to initialize the object



`Cookie c1=new Cookie();`

`Cookie c2=new Cookie();`

`Car car1=new Car();`

`Car car2=new Car();`

`Employee e1=new Employee();`

`Employee e2=new Employee();`

**Constructor:**  
Method having the  
same name as the  
class name is called  
constructor

# Create Object and Access Members

The attributes and methods of a class can be accessed using the dot operator as `<object>.<member>`

- //object creation
  - `Employee empObj=new Employee();`
- //assign meaningful state
  - `empObj.empId=101;`
  - `empObj.name="Rohit";`
  - `empObj.salary=25000;`
- //retrieve values
  - `Console.WriteLine("Emp ID "+empObj.empId);`
  - `Console.WriteLine("Name "+empObj.name);`
  - `Console.WriteLine("Salary "+empObj.salary);`

# Example for class

```
public class Employee {  
  
    //Attributes  
    public int empld;  
    public String name;  
    public double salary;  
  
    //Methods  
  
    public double  
    CalculateGrossPay(int bonus)  
    {  
        return salary + bonus  
    }  
}
```

```
public class EmployeeMain  
{  
    static void Main()  
    {  
        //object creation  
        Employee empObj=new Employee();  
        //assign meaningful state  
        empObj.empld=101;  
        empObj.name="Rohit";  
        empObj.salary=25000;  
        //retrieve values  
        Console.WriteLine("Emp ID  
"+empObj.empld);  
        Console.WriteLine("Name "+empObj.name);  
        Console.WriteLine("Grosspay:  
        empObj.GrossPay(5000));  
    }  
}
```

# Controlling the visibility of class members

Encapsulation ensures that the private data and the complexities of the inner working of a class are hidden so that it cannot be accessed outside the class

Encapsulation is an essential object oriented concept which hides the internal data and also provides a well defined public operation

Information Hiding is implemented using Access modifiers known as visibility modifiers

Access modifiers are keywords used to specify the declared accessibility of a member or a type

They specify if a field or method in a class can be invoked from another class or sub-class

There are **four** access modifiers which are used to restrict access:

- public
- protected
- internal
- private

# Accessibility Level

The four access modifiers public, protected, private and internal are used to specify one of the following declared accessibility levels for members

**public**

Access is not restricted

**protected**

Access is limited to the containing class or types derived from the containing class.

**internal**

Access is limited to the current assembly.

**protected  
internal**

Access is limited to the current assembly or types derived from the containing class.

**private**

Access is limited to the containing type.

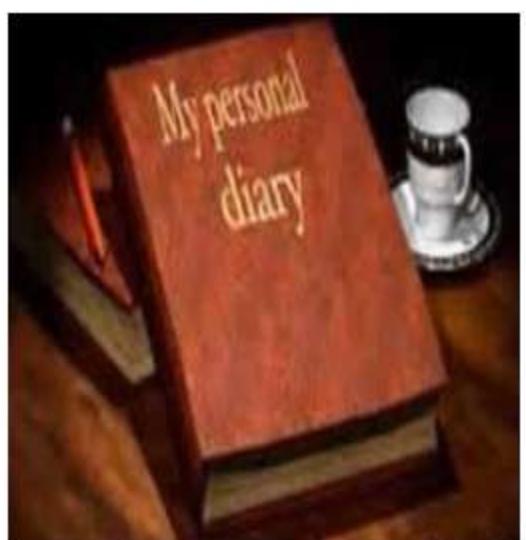
# Access Specifier

**public**

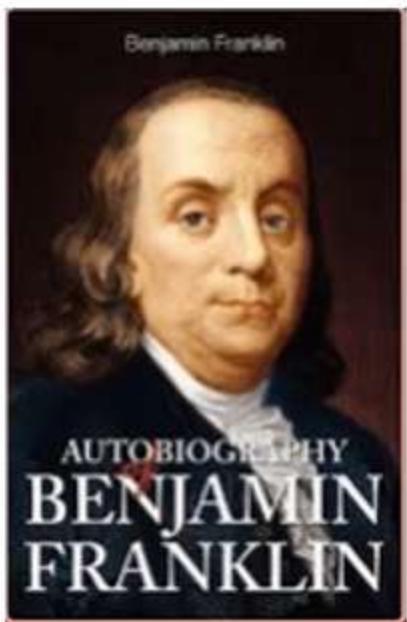
- Classes, methods, and fields declared as public can be accessed from any class in an application

**private**

- private methods and fields can only be accessed within the same class to which the methods and fields belong
- The fields of a class should be generally declared as private, so that they are not exposed to members outside the class



**private**

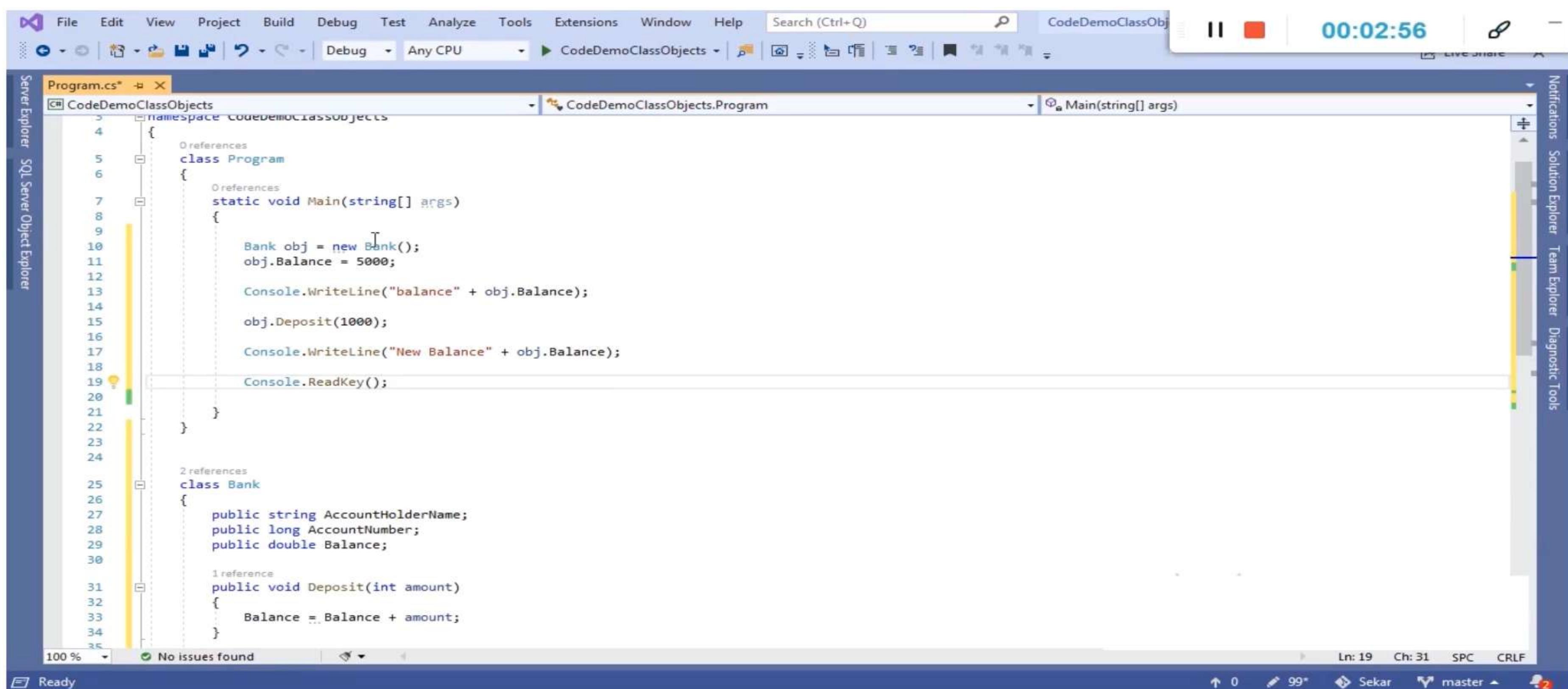


**public**



# Code Demo

# Class and Object Demo



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar displays "CodeDemoClassObj" and the status bar shows "00:02:56". The main code editor window contains the following C# code:

```
File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) CodeDemoClassObj
Debug Any CPU CodeDemoClassObjects Program Main(string[] args)
Program.cs* X
C# CodeDemoClassObjects
namespace CodeDemoClassObjects
{
    class Program
    {
        static void Main(string[] args)
        {
            Bank obj = new Bank();
            obj.Balance = 5000;

            Console.WriteLine("balance" + obj.Balance);

            obj.Deposit(1000);

            Console.WriteLine("New Balance" + obj.Balance);

            Console.ReadKey();
        }
    }

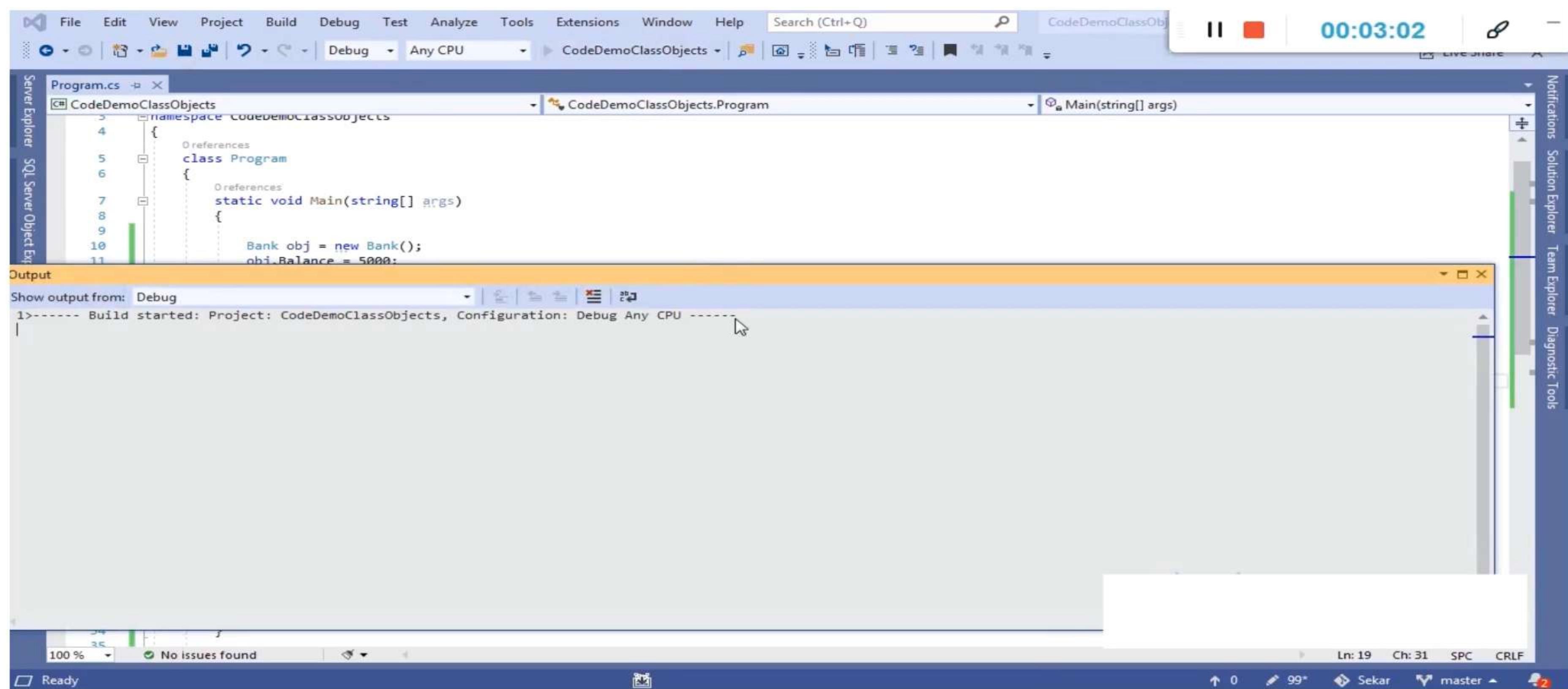
    class Bank
    {
        public string AccountHolderName;
        public long AccountNumber;
        public double Balance;

        public void Deposit(int amount)
        {
            Balance = Balance + amount;
        }
    }
}

2 references
1 reference
```

The code defines a `Program` class with a `Main` method. Inside `Main`, a `Bank` object is created and its `Balance` is set to 5000. It then calls the `Deposit` method with an argument of 1000, and prints the new balance. Finally, it reads a key from the console. A nested `Bank` class is defined with properties for `AccountHolderName`, `AccountNumber`, and `Balance`, and a `Deposit` method that adds the specified amount to the balance.

# Class and Object Demo



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar displays "CodeDemoClassObjects". The menu bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search field. The toolbar contains various icons for file operations. The status bar at the bottom shows "Ln: 19 Ch: 31 SPC CRLF".

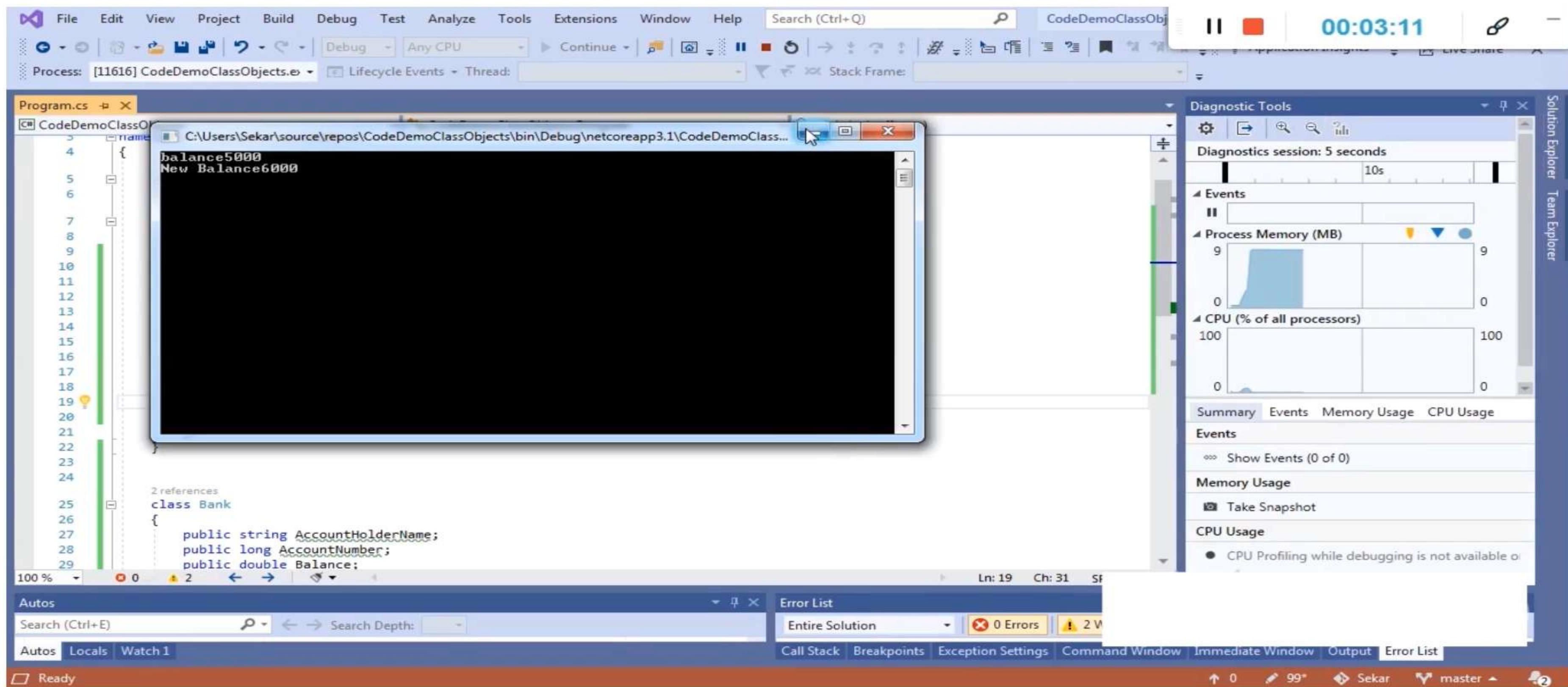
The main workspace shows the "Program.cs" file under the "CodeDemoClassObjects" project. The code defines a class "Program" with a static void Main method that creates a new "Bank" object and sets its balance to 5000.

```
1  namespace CodeDemoClassObjects
2  {
3      class Program
4      {
5          static void Main(string[] args)
6          {
7              Bank obj = new Bank();
8              obj.Balance = 5000;
9          }
10     }
11 }
```

The "Output" window at the bottom left shows the build log: "Build started: Project: CodeDemoClassObjects, Configuration: Debug Any CPU".

The status bar at the bottom right shows "Ln: 19 Ch: 31 SPC CRLF".

# Class and Object Demo



The screenshot shows a Visual Studio IDE interface during a debugging session. The title bar indicates the project is "CodeDemoClassObj" and the current file is "Program.cs". The status bar shows the time as "00:03:11".

The code editor displays a C# class definition:

```
4     {
5         balance5000
6         New Balance6000
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25     2 references
26     class Bank
27     {
28         public string AccountHolderName;
29         public long AccountNumber;
30         public double Balance;
```

The Immediate Window (top right) shows the output of the code execution:

```
C:\Users\Sekar\source\repos\CodeDemoClassObjects\bin\Debug\netcoreapp3.1\CodeDemoClass...
balance5000
New Balance6000
```

The Diagnostic Tools window (right side) provides performance monitoring data:

- Diagnostics session: 5 seconds
- Events: No events recorded.
- Process Memory (MB): Shows memory usage starting at ~1 MB and rising to ~9 MB.
- CPU (% of all processors): Shows CPU usage starting at 0% and rising to 100%.

The Solution Explorer and Team Explorer tabs are visible on the far right.

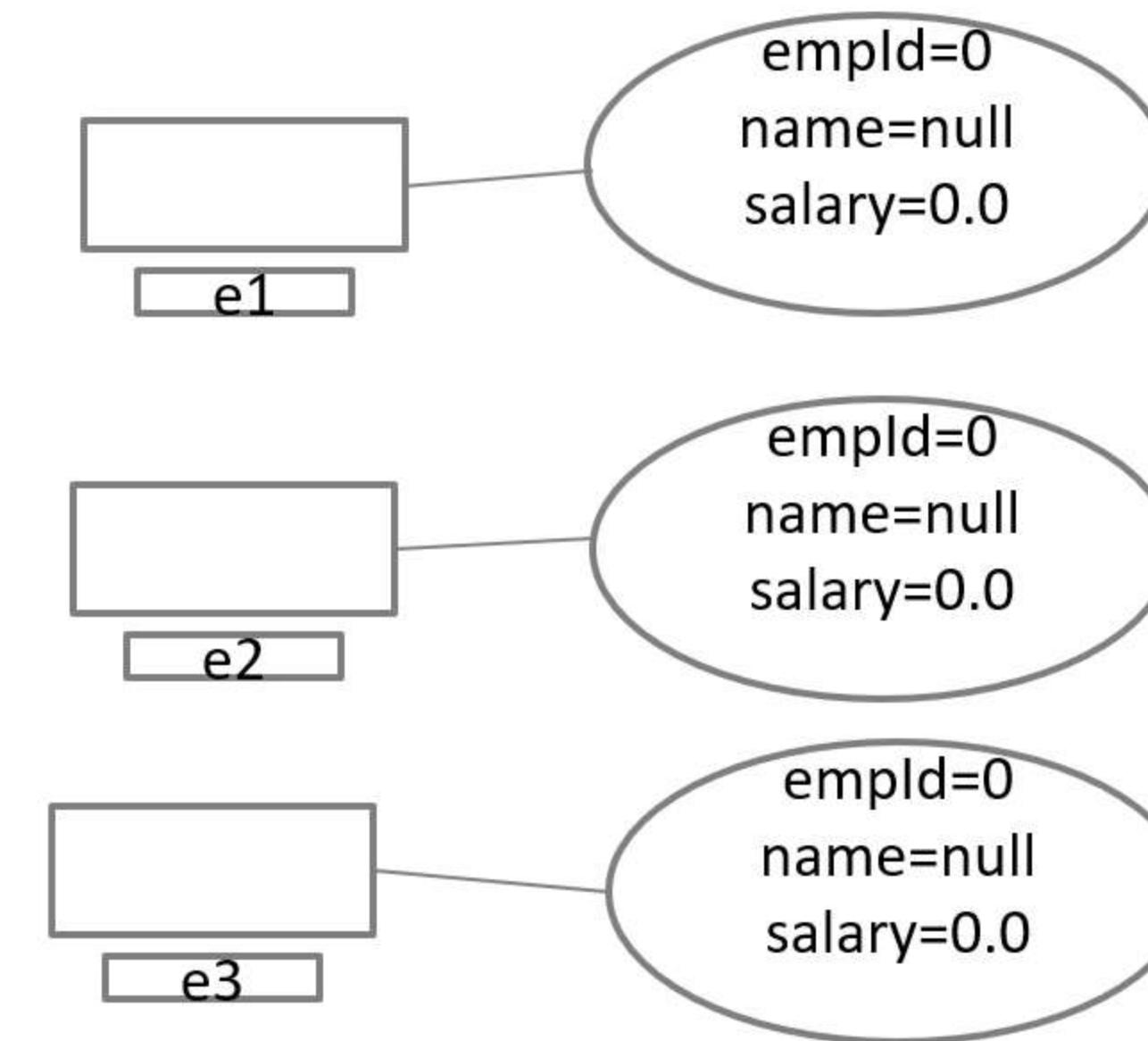
# Constructors

Whenever a class or struct is created, its constructor is called

Constructors are a special kind of member function that get invoked when an object is created

Constructors have the same name as the class or struct, and they usually initialize the data members of the new object

```
Employee e1=new Employee();  
Employee e2=new Employee();  
Employee e3=new Employee();
```



# Constructor

Constructor is a special method invoked implicitly when an object is created.

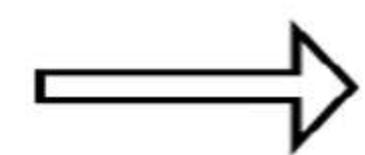
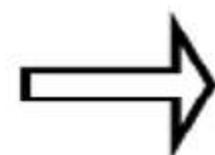
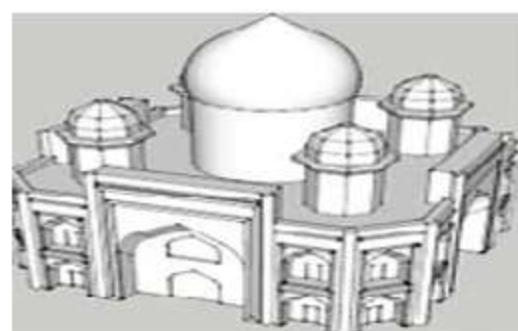
It initializes the instance variables with default value.

## Rules for writing a constructor

- Constructor name must be the same as the name of the class
- Constructors must not have a return type (not even void)

## Types of Constructors

- Default Constructor
- Parameterized Constructor



# Constructors

A class or struct may have multiple constructors that take different arguments

Default constructors are invoked whenever an object is instantiated by using the new operator and no arguments are provided

A constructor that takes zero parameters is called a default constructor

A constructor that takes one or more parameters is called a parameterized constructor

# Default Constructor

## Default Constructor (no argument constructor)

A constructor with no arguments is known as default constructor

### Example:

```
class Employee
{
    private int empld;
    private String name;

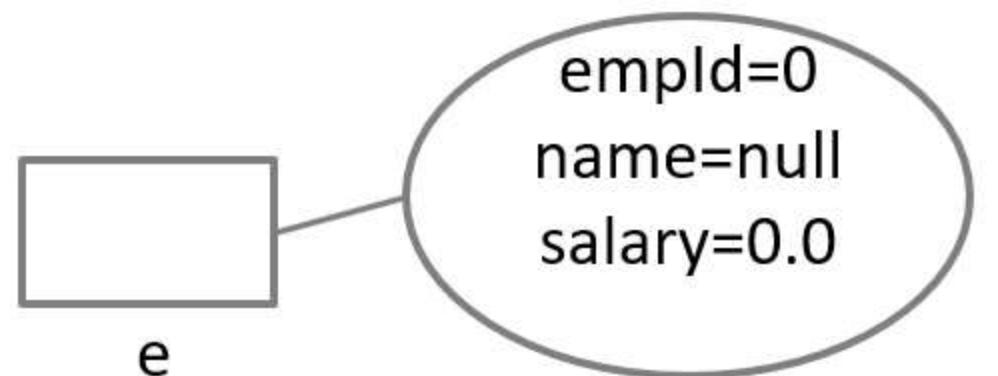
    public Employee()
    {
        Console.WriteLine("Default Constructor");
    }

    public static void Main(String a[])
    {
        Employee e=new Employee();
    }
}

Output : Default Constructor
```



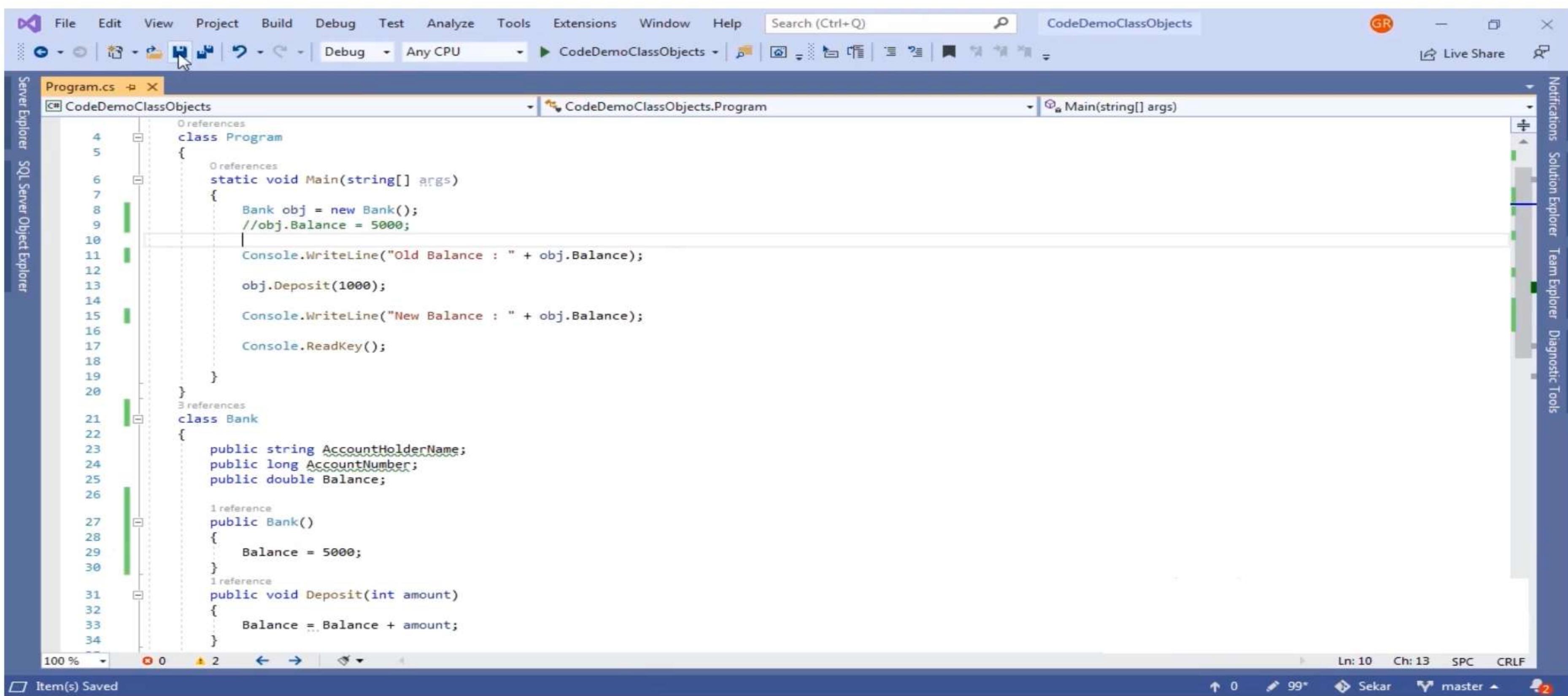
Default Constructor





# Code Demo

# Default Constructor Demo



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar displays "CodeDemoClassObjects". The menu bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar "Search (Ctrl+Q)". The toolbar contains various icons for file operations like Open, Save, and Build.

The main code editor window shows the following C# code:

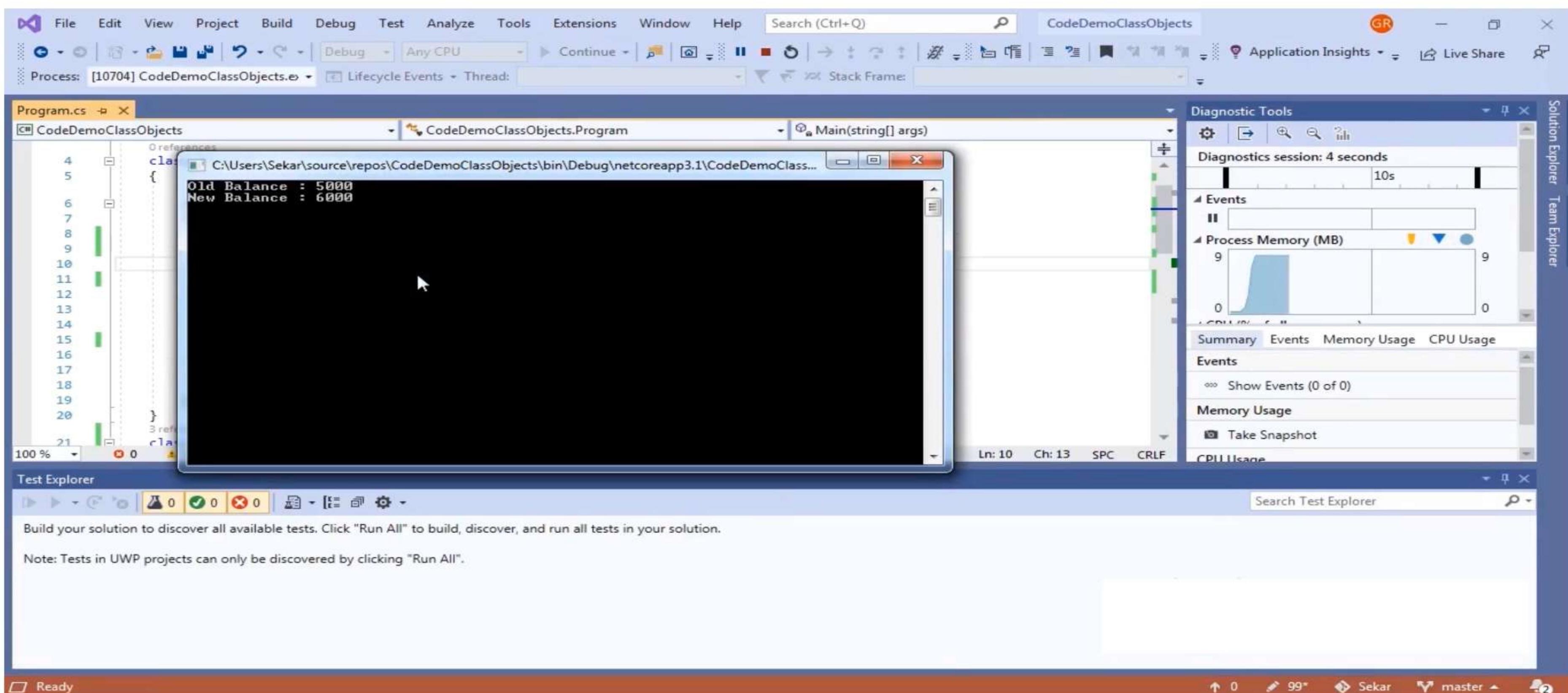
```
Program.cs
CodeDemoClassObjects
CodeDemoClassObjects.Program
Main(string[] args)

4  class Program
5  {
6      static void Main(string[] args)
7      {
8          Bank obj = new Bank();
9          //obj.Balance = 5000;
10
11         Console.WriteLine("Old Balance : " + obj.Balance);
12
13         obj.Deposit(1000);
14
15         Console.WriteLine("New Balance : " + obj.Balance);
16
17         Console.ReadKey();
18     }
19 }
20
21 class Bank
22 {
23     public string AccountHolderName;
24     public long AccountNumber;
25     public double Balance;
26
27     public Bank()
28     {
29         Balance = 5000;
30     }
31
32     public void Deposit(int amount)
33     {
34         Balance = Balance + amount;
35     }
}

100 % 0 2  Item(s) Saved  Ln: 10 Ch: 13 SPC CRLF
```

The code defines a `Program` class with a `Main` method. Inside `Main`, a `Bank` object is created with an initial balance of 5000. The `Deposit` method is called to add 1000 to the balance, and the final balance is printed to the console. The `Bank` class has properties for `AccountHolderName`, `AccountNumber`, and `Balance`, and a constructor that initializes `Balance` to 5000. The `Deposit` method adds a specified amount to the balance.

# Default Constructor Demo



The screenshot shows a Microsoft Visual Studio interface during a debugging session. The title bar indicates the project is "CodeDemoClassObjects" and the file is "Program.cs". The code editor displays the following C# code:

```
class Program
{
    static void Main(string[] args)
    {
        Old Balance : 5000
        New Balance : 6000
    }
}
```

A black rectangular overlay covers most of the code editor area, containing the output of the program: "Old Balance : 5000" and "New Balance : 6000". The status bar at the bottom shows "Ln: 10 Ch: 13 SPC CRLF".

The Diagnostic Tools window on the right shows a "Diagnostics session: 4 seconds" timeline from 0s to 10s. It includes sections for "Events" (empty), "Process Memory (MB)" (a graph showing memory usage starting at 0 and jumping to 9 MB), and "CPU Usage" (empty). Buttons for "Summary", "Events", "Memory Usage", and "CPU Usage" are present, along with "Events", "Show Events (0 of 0)", "Memory Usage", "Take Snapshot", and "CPU Usage".

The Test Explorer window at the bottom left shows 0 tests discovered, 0 passed, and 0 failed. It includes a "Search Test Explorer" field and notes about building the solution and running all tests.

# Parameterized Constructor

## Parametrized Constructor

Constructor with argument list is known as parameterized constructor.

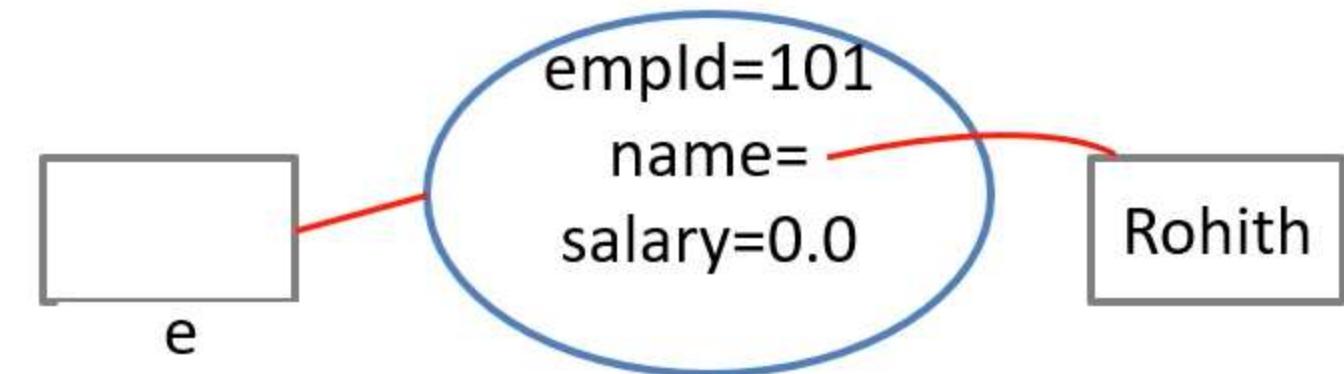
### Example:

```
class Employee
{
    private int empld;
    private String name;
    double salary;

    //Constructor
    public Employee(int id, String ename)
    {
        Console.WriteLine("In Parametrized Constructor");
        empld=id;
        name=ename;
    }
}
```

```
static void Main(String a[])
{
    Employee e=new Employee(101,"Rohith");
}
}
```

### Output : In Parametrized Constructor



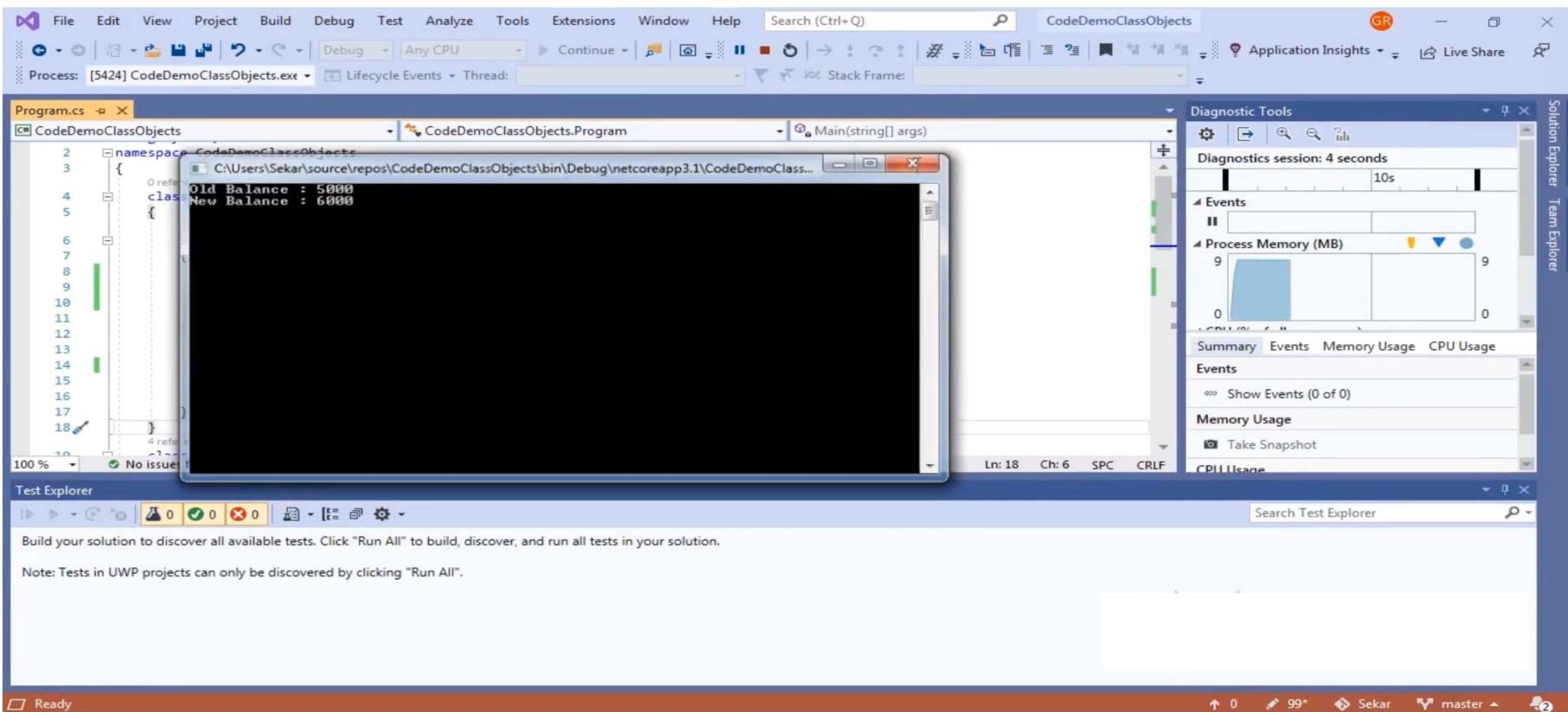


# Code Demo

# Parameterized Constructor Demo



# Parameterized Constructor Demo



The screenshot shows the Microsoft Visual Studio IDE interface during a debugging session. The top menu bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search bar. The title bar displays "CodeDemoClassObjects". The toolbar contains various icons for file operations like Open, Save, and Print.

The main code editor window shows the "Program.cs" file with the following C# code:

```
1  // Main entry point for the application
2  namespace CodeDemoClassObjects
3  {
4      class Program
5      {
6          static void Main(string[] args)
7          {
8              // ...
9          }
10     }
11 }
```

A tooltip or callout box highlights the line "Old Balance : 5000". The status bar at the bottom indicates "Ln: 18 Ch: 6 SPC CRLF".

To the right of the code editor is the "Diagnostic Tools" window, which shows a timeline from 0s to 10s. It displays two events: "Process Memory (MB)" which starts at 9 MB and drops to 0 MB, and "CPU Usage" which shows a single blue bar peaking around 9 MB. The "Events" tab is selected, showing 0 events.

Below the code editor is the "Test Explorer" window, which displays "0" for each category: Tests, Passes, and Failures. It also includes a "Search Test Explorer" bar and a note: "Build your solution to discover all available tests. Click 'Run All' to build, discover, and run all tests in your solution." and "Note: Tests in UWP projects can only be discovered by clicking 'Run All'".

# Constructors

If a constructor is not written explicitly for a class, what happens when an object is created?

\* C# compiler builds a default constructor for that class and instantiates the fields with the default value.



When a constructor is explicitly written by the developer, compiler will not provide the default constructor

If a constructor with parameters is declared and we want to create an object using no argument constructor, we must explicitly write a no argument constructor

# “this” keyword

```
class Employee{  
    int empld;  
    String name;  
  
    public Employee(int empld, String name)  
    {  
        empld = empld;  
        name = name;  
    }  
    void Display()  
    {  
        Console.WriteLine("ID : "+id+" Name : "+name);  
    }  
  
    static void Main(String args[])  
    {  
        Employee e1= new Employee(101,"Tom");  
        e1.Display();  
    }  
}
```

**Output of this code :**

ID : 0 Name : null

The reason for this output is parameter and instance variables have the same name.

**The solution for this problem is to use the keyword “this”**

# “this” keyword

“this” is a reference variable that refers to the current object

**Example :** Usage of “this” in constructor for initializing the attributes

```
class Employee{  
    int empld;  
    String name;  
  
    public Employee(int empld, String name)  
    {  
        this.empld = empld;  
        this.name = name;  
    }  
    void Display()  
    {  
        Console.WriteLine("ID :" + id + " Name :" + name);  
    }  
}
```

```
static void Main(String args[])  
{  
    Employee e1 = new Employee(101, "Tom");  
    e1.Display();  
}  
}
```

Output of this code :

ID : 101 Name : Tom

# “this” keyword

```
public class Rectangle {  
    private float length;  
    private float breadth;  
  
    //Constructor  
    public Rectangle(float length, float breadth)  
    {  
        this.length = length;  
        this.breadth = breadth;  
    }  
  
    //Function to calculate Area  
    public float CalcArea()  
    {  
        return length*breadth;  
    }  
}
```

```
//difference between the area of 2 rectangle  
public float DiffInArea(Rectangle r)  
{  
    float area_difference =  
        Math.Abs(this.calcArea() -  
                  r.CalcArea());  
    return area_difference;  
}  
static void Main(String a[]){  
    Rectangle r1=new Rectangle(6,9);  
    Rectangle r2=new Rectangle(12,15);  
    float diffInArea=r1.DiffInArea(r2);  
    Console.WriteLine("Difference in Area : "  
                     +diffInArea);  
}
```

# Constructor Overloading

A class can have any number of constructors but they should differ in the parameter list.

This technique is called Constructor Overloading.

The parameter list should vary in any of the following

- Number of parameters
- Type of parameter
- Order of parameter

The compiler identifies which constructor is to be invoked based on the number of parameters in the list and their type

# Constructor Overloading

```

public class Rectangle {
    private float length;
    private float breadth;

    //Constructor Overloading

    public Rectangle()
    {
        length=0;
        breadth=0;
    }

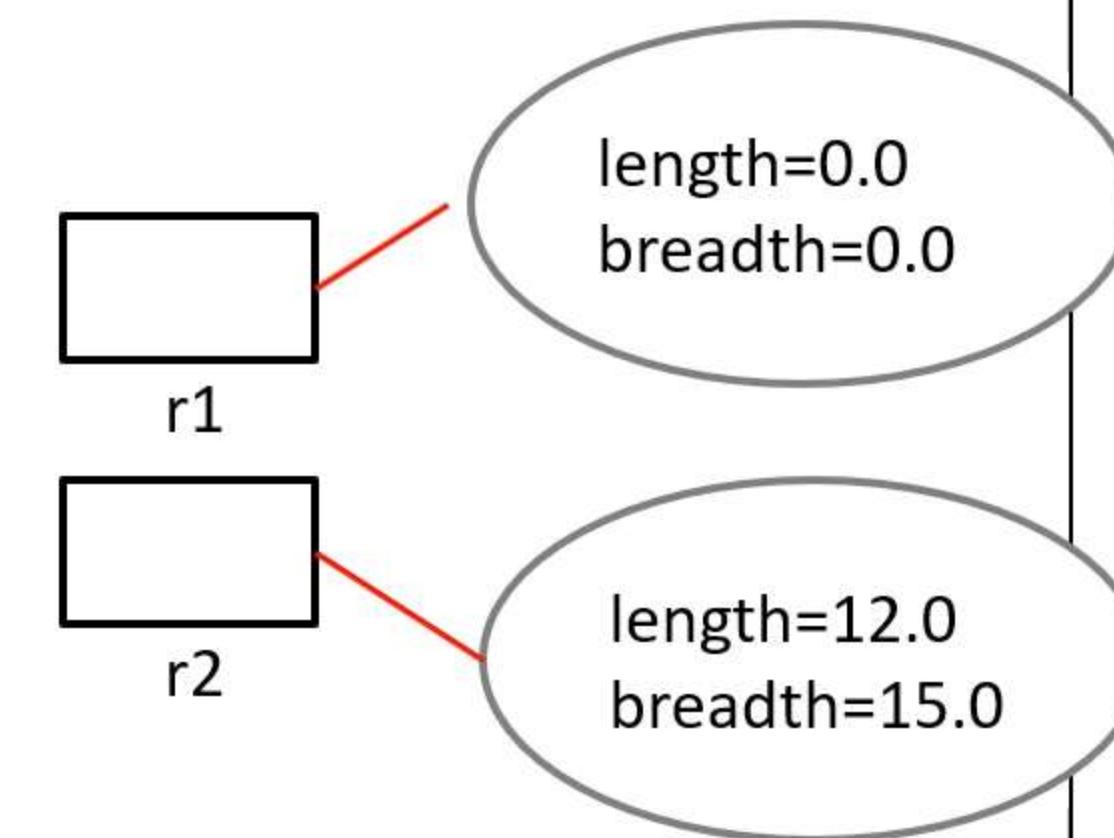
    public Rectangle(float length, float breadth)
    {
        this.length = length;
        this.breadth = breadth;
    }
}

```

```

static void Main(String a[])
{
    Rectangle r1=new Rectangle();
    Rectangle r2=new Rectangle(12,15);
}

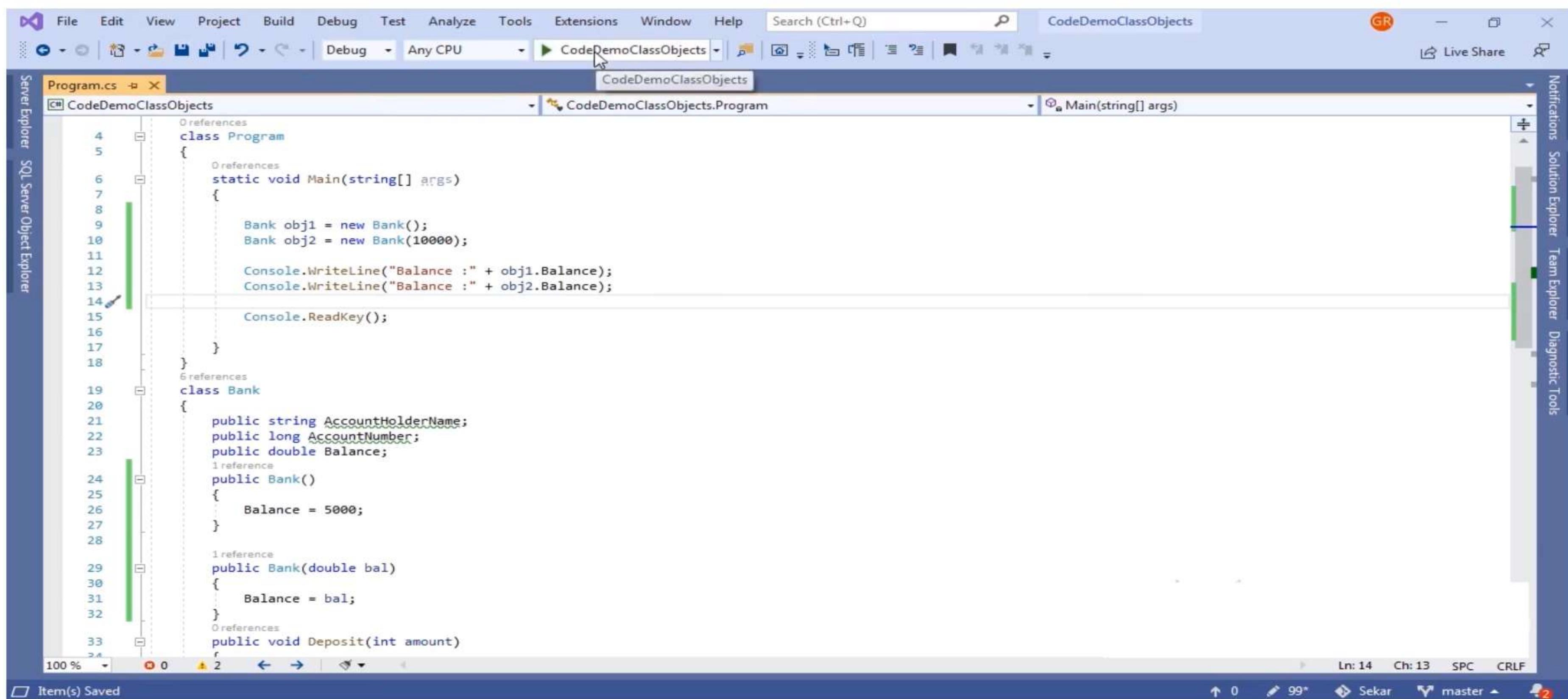
```





# Code Demo

# Constructor Overloading Demo

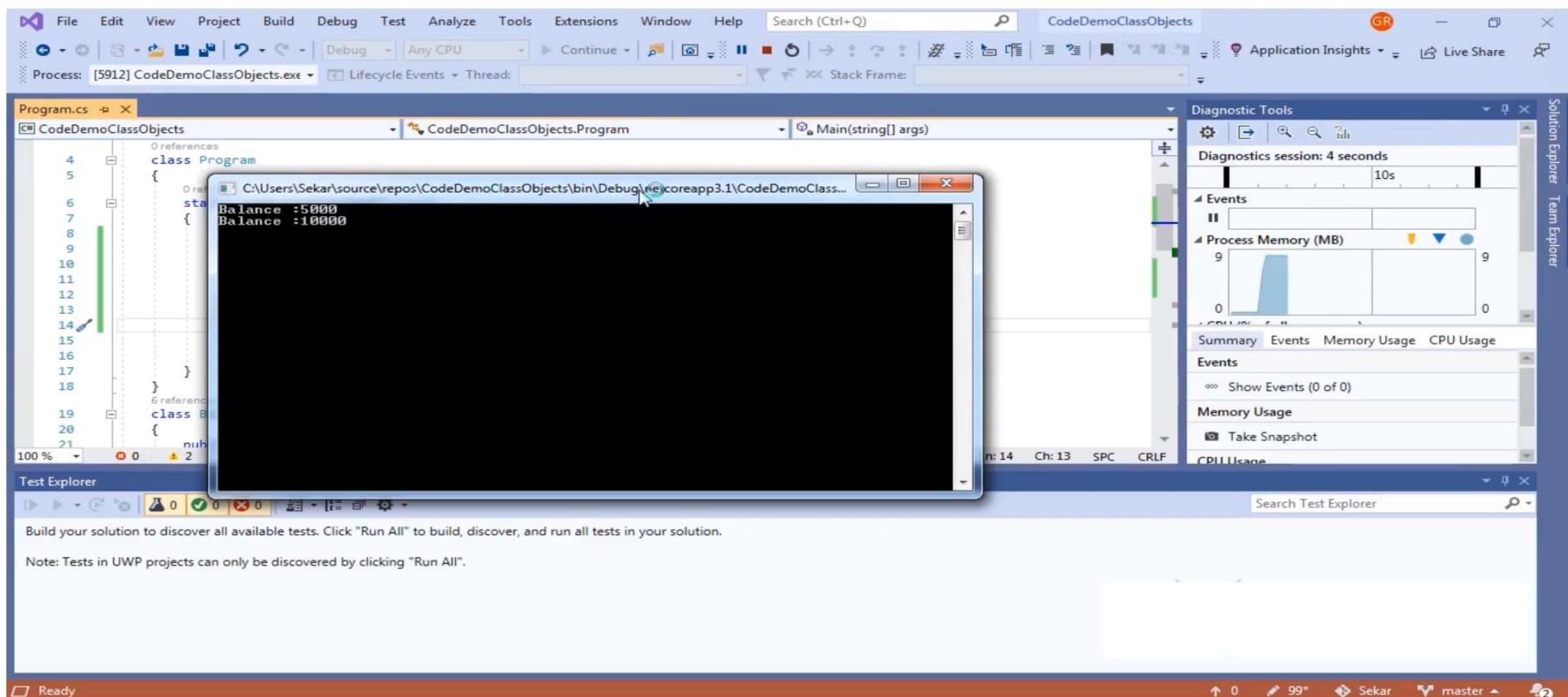


The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- Solution Explorer:** Shows the project "CodeDemoClassObjects" with files "Program.cs" and "Bank.cs".
- Toolbars:** Standard Visual Studio toolbars for File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Status Bar:** Displays "Ln: 14 Ch: 13 SPC CRLF" at the bottom right.
- Code Editor:** The "Program.cs" file contains the following C# code:

```
4 0 references
5 class Program
{
6     0 references
7     static void Main(string[] args)
8     {
9         Bank obj1 = new Bank();
10        Bank obj2 = new Bank(10000);
11
12        Console.WriteLine("Balance :" + obj1.Balance);
13        Console.WriteLine("Balance :" + obj2.Balance);
14
15        Console.ReadKey();
16    }
17}
18 6 references
19 class Bank
20 {
21     public string AccountHolderName;
22     public long AccountNumber;
23     public double Balance;
24     1 reference
25     public Bank()
26     {
27         Balance = 5000;
28     }
29     1 reference
30     public Bank(double bal)
31     {
32         Balance = bal;
33     }
34     0 references
35     public void Deposit(int amount)
```

# Constructor Overloading Demo



The screenshot shows a Microsoft Visual Studio interface with the following details:

- Title Bar:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search (Ctrl+Q), CodeDemoClassObjects, GR.
- Toolbar:** Standard icons for file operations, search, and application status.
- Status Bar:** Process: [5912] CodeDemoClassObjects.exe, Lifecycle Events, Thread: Stack Frame:.
- Diagnostic Tools Window:** Shows a diagnostics session for 4 seconds. It includes a timeline from 0s to 10s, an Events section, and a Process Memory (MB) chart. The memory usage starts at 0 MB, peaks at approximately 9 MB around 4 seconds, and returns to 0 MB by 10 seconds. Other tabs include Summary, Events, Memory Usage, and CPU Usage.
- Code Editor:** Displays the `Program.cs` file with the following code:

```
4  class Program
5  {
6      static void Main(string[] args)
7      {
8          Balance = 5000;
9          Balance = 10000;
10     }
11 }
12
13 class Balance
14 {
15     public int value { get; set; }
16 }
```
- Output Window:** Shows the output of the program:

```
C:\Users\Sekar\source\repos\CodeDemoClassObjects\bin\Debug\netcoreapp3.1\CodeDemoClass...
Balance :5000
Balance :10000
```
- Test Explorer:** A message says "Build your solution to discover all available tests. Click "Run All" to build, discover, and run all tests in your solution." A note below states: "Note: Tests in UWP projects can only be discovered by clicking "Run All"."
- Solution Explorer and Team Explorer:** Standard navigation panes for the solution.

# Constructors Access Specifiers

**Constructors can be marked as public, private, protected, internal or protected internal**

**These access modifiers define how users of the class can construct the class**

**A constructor can be declared static by using the static keyword**

**Static constructors are called automatically, immediately before any static fields are accessed, and are generally used to initialize static class members**

# Static Constructors

A static constructor is used to initialize any static data, or to perform a particular action once only.

It is called automatically before the first instance is created or any static members are referenced.

```
class SimpleClass {  
    // Static variable that must be initialized at run time.  
    static readonly long baseline;  
    // Static constructor is called only once, before any instance constructor is  
    //invoked or member is accessed.  
    public static SimpleClass()  
    {  
        baseline = DateTime.Now.Ticks;  
    }  
}
```

# Private Constructors

A private constructor is a special instance constructor, used in classes that contain static members only

If a class has one or more private constructors and no public constructors, other classes (except nested classes) cannot create instances

If no access modifiers are specified for the constructor it is private by default; the private modifier specifies that the class cannot be instantiated

# Structures

Structures in C#  
are declared with a  
keyword struct.

Structures contain  
related data and  
methods.

Structs are value  
type .

A struct can not be  
inherited from  
another struct  
variable

# Struct

```
public struct CoOrds
{
    public int x, y;

    public CoOrds(int p1, int p2)
    {
        x = p1;
        y = p2;
    }
}
```

```
class Program
{
    static void Main()
    {
        // Initialize:
        CoOrds pt1 = new CoOrds();
        CoOrds pt2 = new CoOrds(10, 10);

        // Display results:
        Console.Write("CoOrds 1: ");
        Console.WriteLine("x = {0}, y = {1}", pt1.x, pt1.y);

        Console.Write("CoOrds 2: ");
        Console.WriteLine("x = {0}, y = {1}", pt2.x, pt2.y);
        Console.ReadKey();
    }
}
```

# Summary

- OOP with C#
- Classes and Objects
- Constructors
- Structures in C#



# C# STRINGS



# Objectives

In this module you will learn

- String
- String Builder
- String Methods

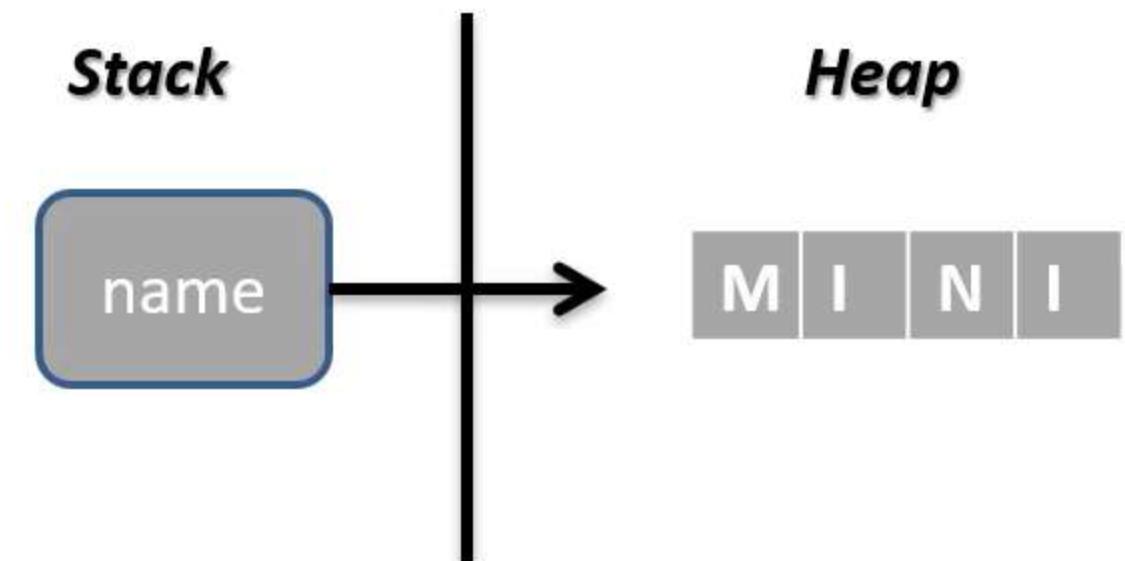
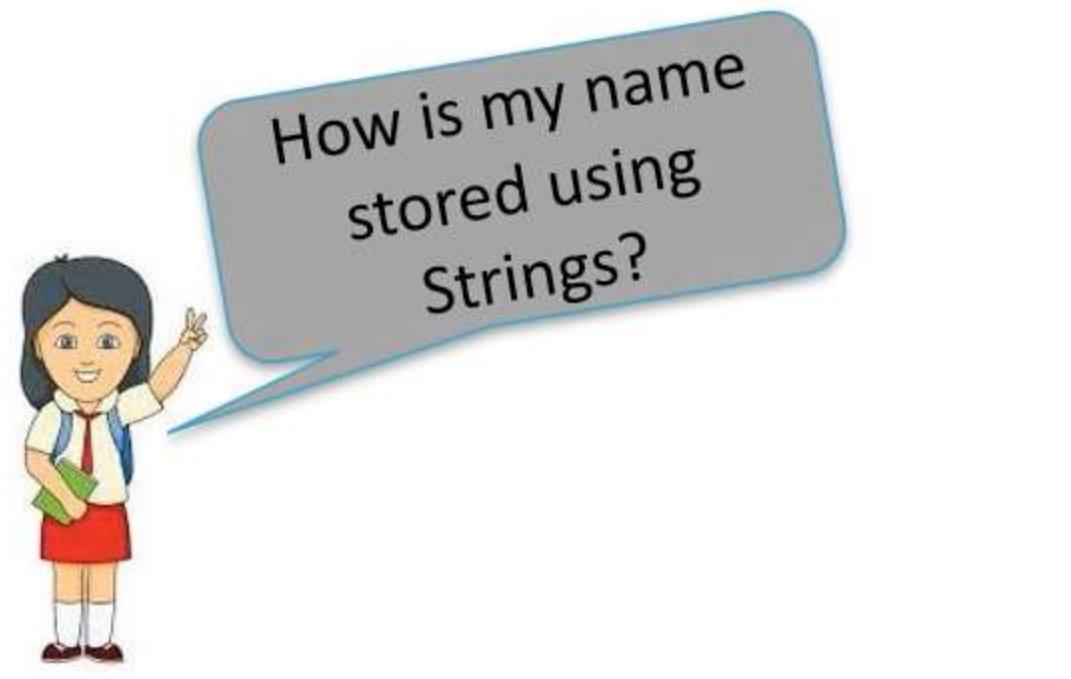


# String

String is a Reference data type and is used to store sequence of characters

**System.String** is a class specifically designed to store a string and allow a large number of operations on the string

The String class provides many methods for safely creating, manipulating, and comparing strings



# Declare and initialize String

```
string message1 = "Hello"; // returns "Hello"  
message1 += ", There"; // returns "Hello, There"  
string message2 = message1 + "!"; // returns "Hello, There!"
```

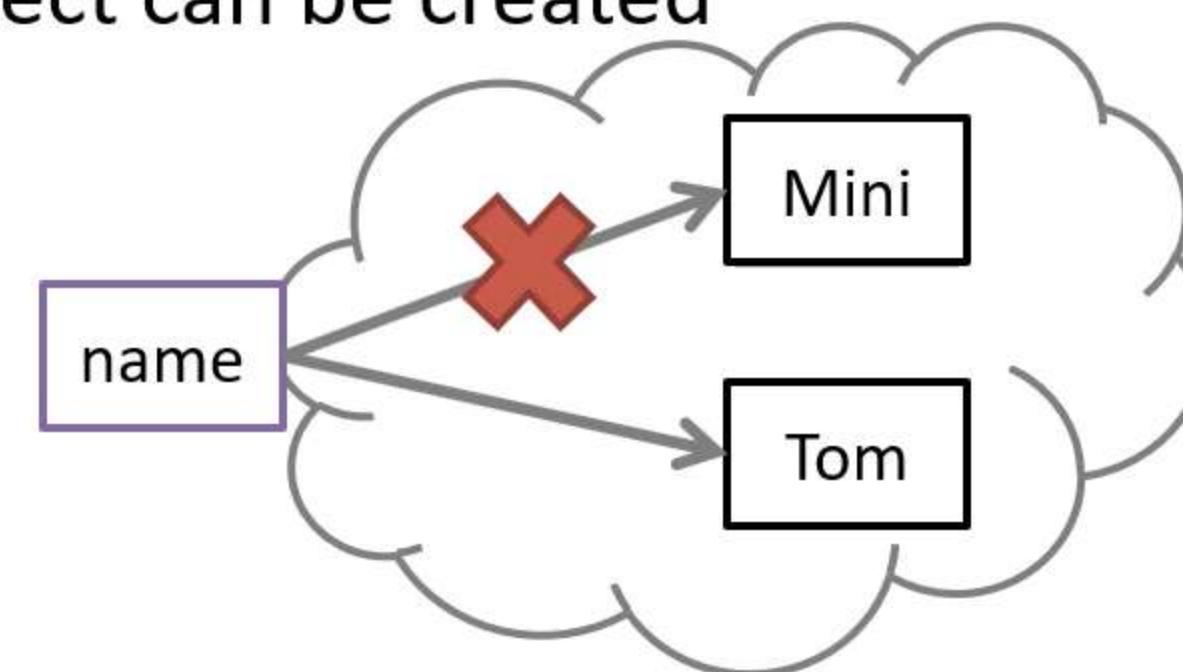
```
string message1; // Declare without initializing..  
string message2 = null; // Initialize to null.  
string message3 = System.String.Empty; // Initialize as an empty string  
//or the Empty constant instead of the literal "".
```

# Strings are Immutable

- String Objects are immutable
- Immutable simply means that which cannot be modified or changed
- Once string object is created its data or state cannot be changed but a new string object can be created

```
String name="Mini";  
Name="Tom"
```

The Object "Mini" is unmodified. Instead a new Object "Tom" is created and mapped with name reference.



# StringBuilder

StringBuilder class normally allocates more memory than is actually needed

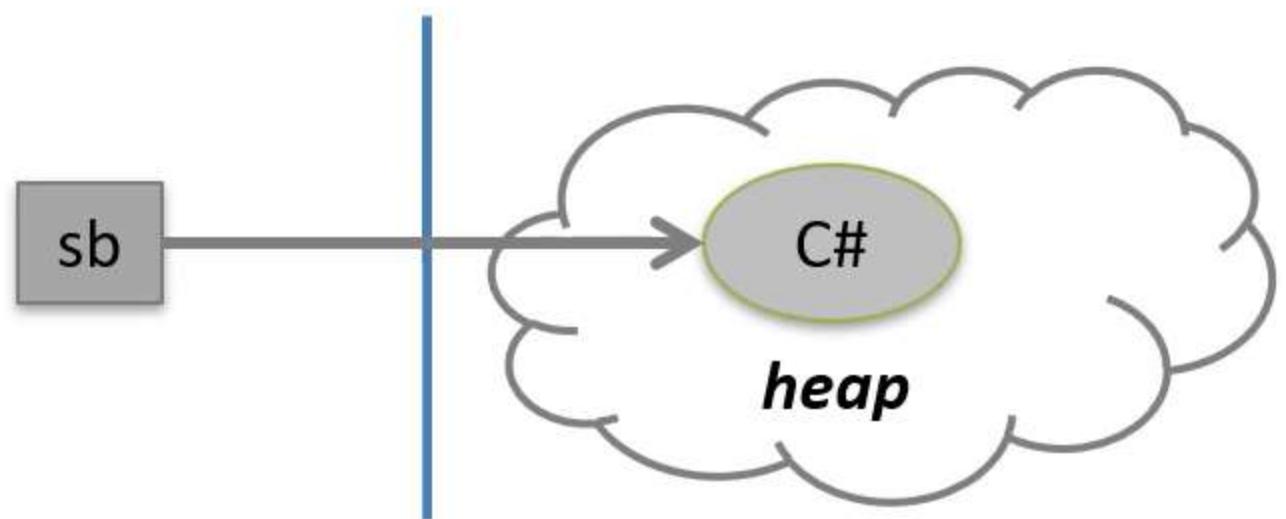
StringBuilder is mainly used for substitutions and appending or removing text from strings

The System.Text.StringBuilder class can be used when you want to modify a string without creating a new object

# StringBuilder

- StringBuilder is used to create a mutable String
- Creating a StringBuilder Object

```
StringBuilder sb = new StringBuilder("C#");
```



# String Builder

The StringBuilder class has two main properties:

- Length — Indicates the length of the string that it actually contains
- Capacity — Indicates the maximum length of the string in the memory allocation

Eg., `StringBuilder sb = new StringBuilder("Hello World",50);`

# Properties of the String Class

The String class has the following two properties –

Sr.No.	Property & Description
1	<b>Chars</b> Gets the <i>Char</i> object at a specified position in the current <i>String</i> object.
2	<b>Length</b> Gets the number of characters in the current String object.

# Methods of the String Class

The String class has numerous methods that help you in working with the string objects.

Method Name	Description
<u>Compare(String, String)</u>	It is used to compares two specified String objects. It returns an integer that indicates their relative position in the sort order.
<u>CompareTo(String)</u>	It is used to compare this instance with a specified String object. It indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified string.
<u>Concat(String, String)</u>	It is used to concatenate two specified instances of String.
<u>Contains(String)</u>	It is used to return a value indicating whether a specified substring occurs within this string.

# Methods

Method Name	Description
<u>EndsWith(String)</u>	It is used to check that the end of this string instance matches the specified string.
<u>Equals(String, String)</u>	It is used to determine that two specified String objects have the same value.
<u>GetEnumerator()</u>	It is used to retrieve an object that can iterate through the individual characters in this string.
<u>GetHashCode()</u>	It returns the hash code for this string.
<u>IndexOf(String)</u>	It is used to report the zero-based index of the first occurrence of the specified string in this instance.
<u>Join(String, String[])</u>	It is used to concatenate all the elements of a string array, using the specified separator between each element.

# Methods

Method Name	Description
<u>Replace(String, String)</u>	It is used to return a new string in which all occurrences of a specified string in the current instance are replaced with another specified string.
<u>Split(Char[])</u>	It is used to split a string into substrings that are based on the characters in an array.
<u>StartsWith(String)</u>	It is used to check whether the beginning of this string instance matches the specified string.
<u>Substring(Int32)</u>	It is used to retrieve a substring from this instance. The substring starts at a specified character position and continues to the end of the string.
<u>ToCharArray()</u>	It is used to copy the characters in this instance to a Unicode character array.

# Summary

- String
- String Builder
- String Methods



# C# ARRAYS



# Objectives

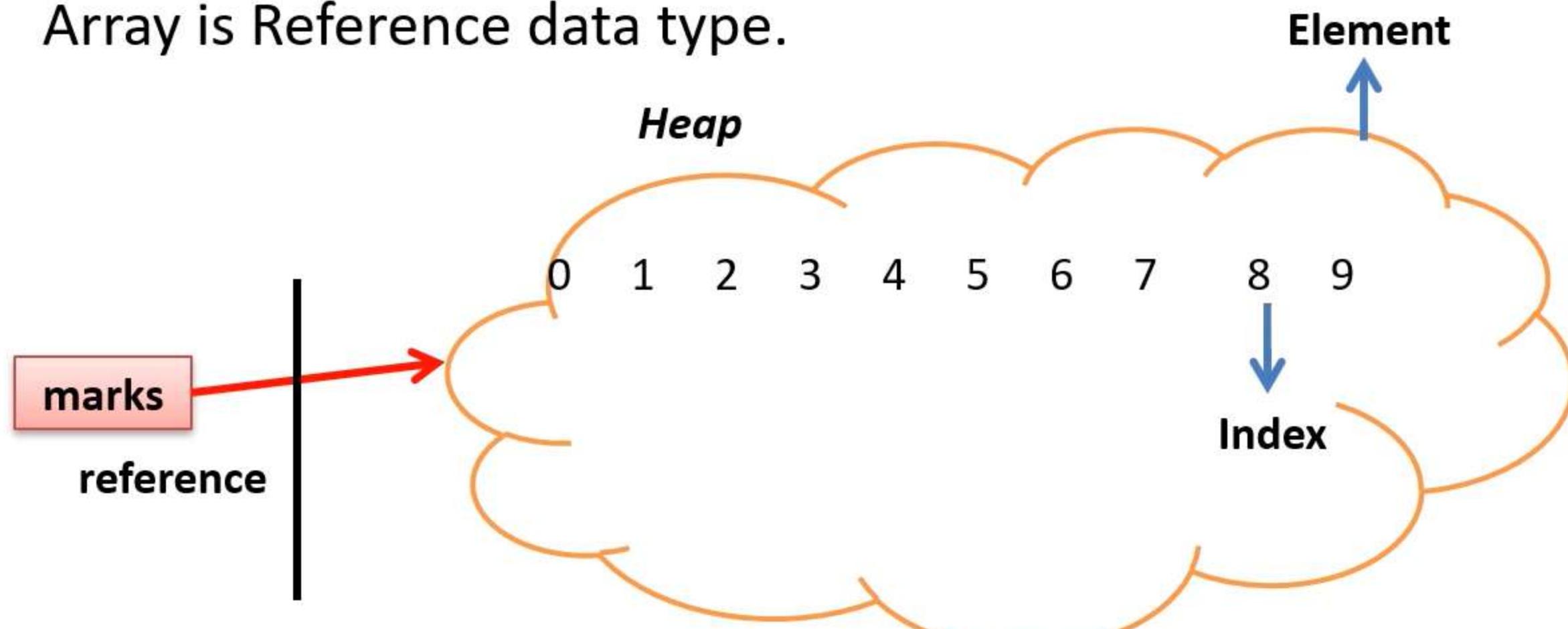
In this module you will learn

- Single Dimensional Arrays
- Multi-Dimensional Arrays
- Jagged arrays



# Arrays

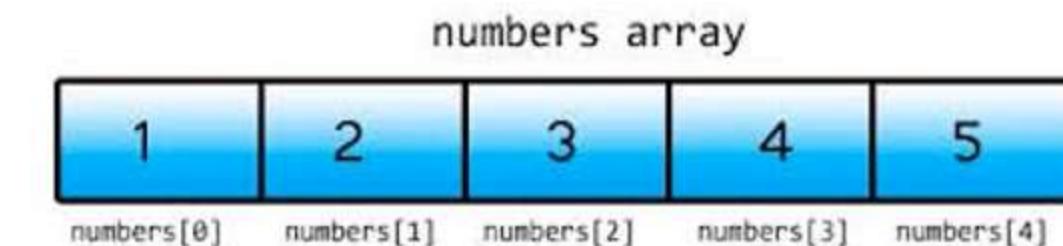
- Array is an ordered collection that stores elements of the same type
- Array elements are stored contiguously
- Array in C# is index based; the first element of the array is stored at 0 index
- Array is Reference data type.



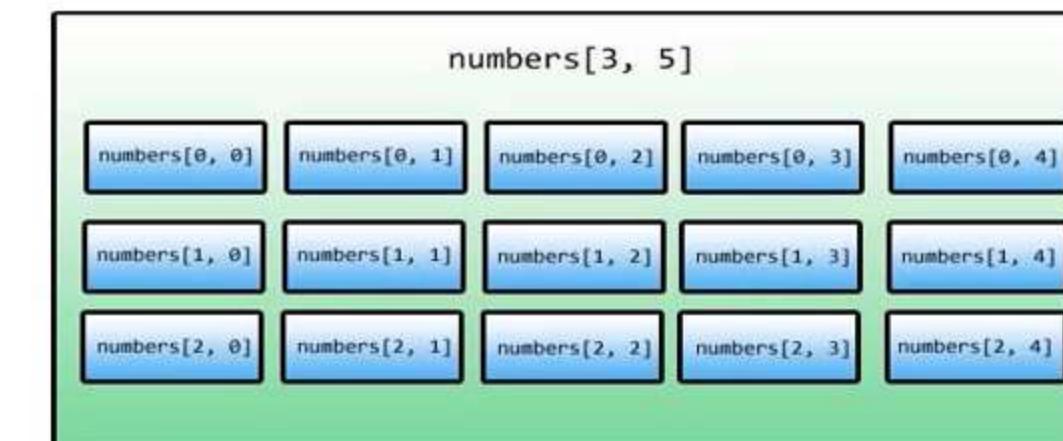
# Arrays Classification

## Arrays

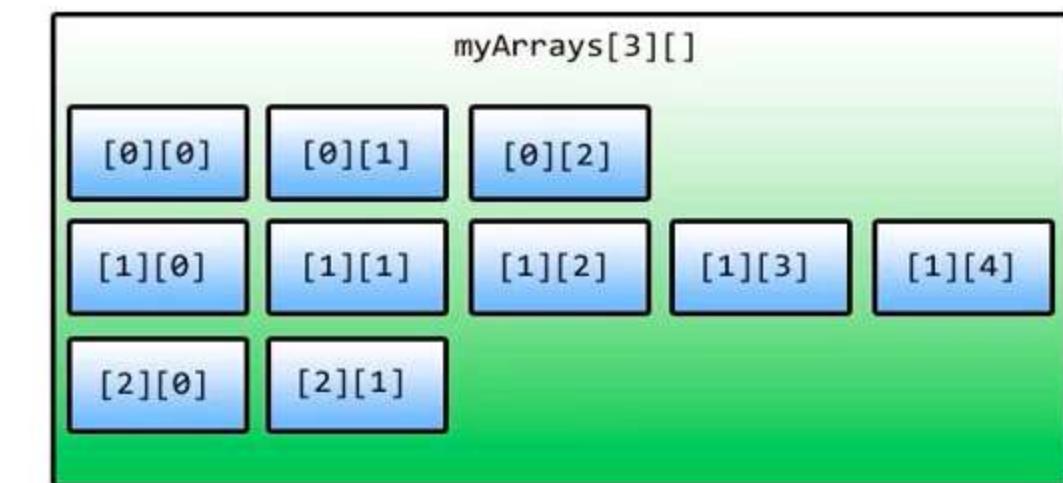
Single  
Dimensional Array



Multi Dimensional  
Array



Jagged Array



# Single Dimensional Array

## Declaring Single Dimensional Array

Data type[] ref\_name;

Int[] marks;

Stack

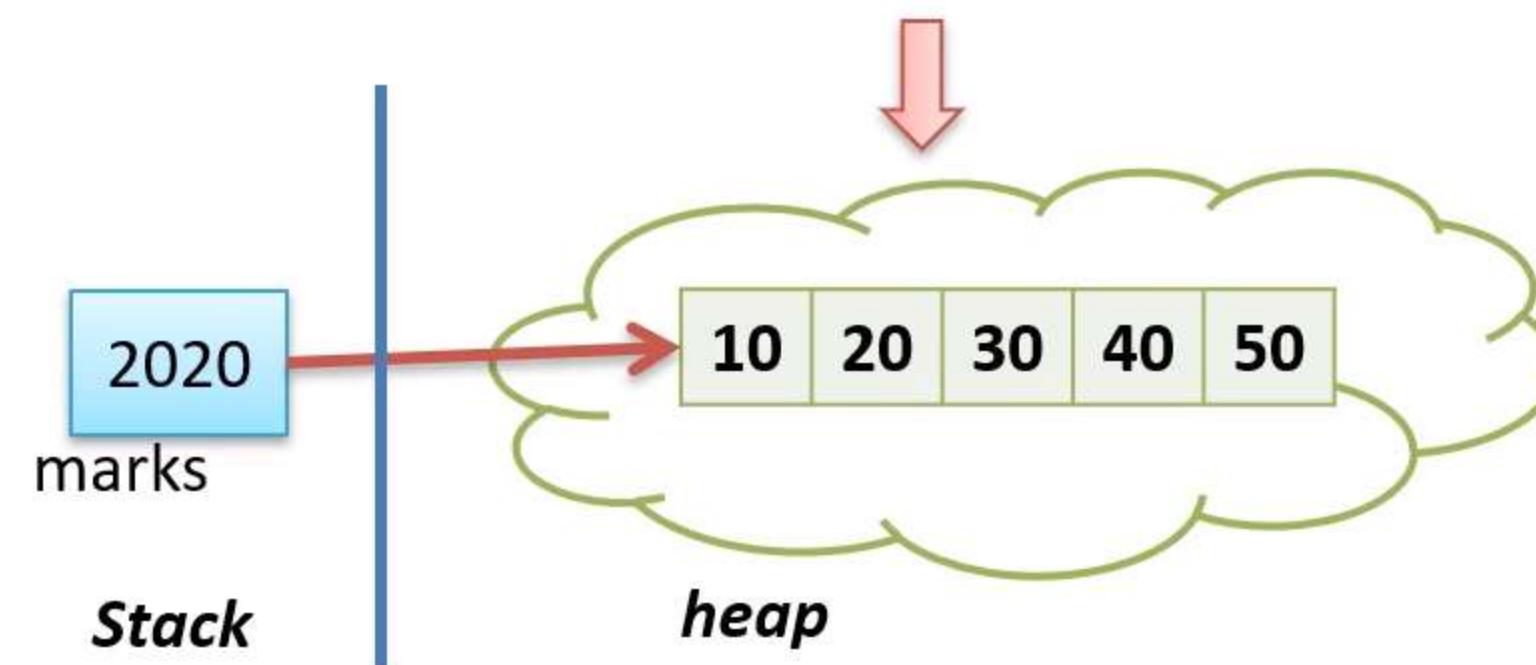
null

marks

## Initializing Array

Data type[] ref\_name ={val1,val2,.. Valn};

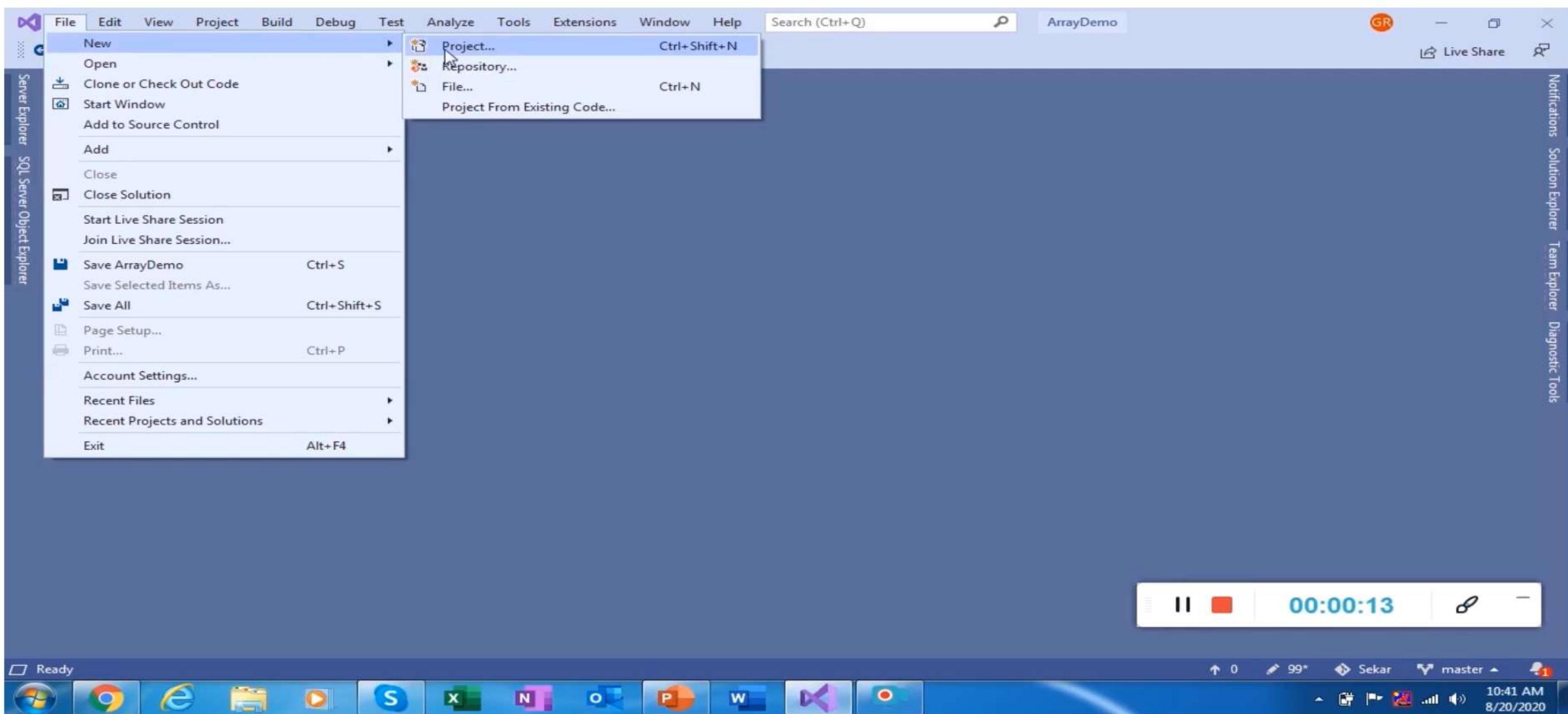
Int[] marks ={10,20,30,40,50};





# Code Demo

# Single Dimensional Array Demo



The screenshot shows the Microsoft Visual Studio interface. The 'File' menu is open, displaying options like 'New', 'Open', and 'Project...'. The 'Project...' option is highlighted with a blue selection bar. The status bar at the bottom right shows a timer at '00:00:13'. The taskbar at the bottom includes icons for various Windows applications.

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help

New Project... Ctrl+Shift+N  
Repository...  
File... Ctrl+N  
Project From Existing Code...

GR - X Live Share Notifications Solution Explorer Team Explorer Diagnostic Tools

Server Explorer SQL Server Object Explorer

Save ArrayDemo Ctrl+S Save Selected Items As...  
Save All Ctrl+Shift+S Page Setup... Print... Ctrl+P Account Settings...  
Recent Files Recent Projects and Solutions

Exit Alt+F4

Ready 0 99% Sekar master 1 10:41 AM 8/20/2020

# Single Dimensional Array Demo

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) ArrayDemo GR X Live Share Notifications Solution Explorer Team Explorer Diagnostic Tools

## Create a new project

Recent project templates

- Console App (.NET Core) C#
- ASP.NET Web Application (.NET Framework) C#
- SQL Server Database Project Query Language
- ASP.NET Core Web Application C#
- Unit Test Project (.NET Framework) C#
- Console App (.NET Framework) C#

All languages All platforms All project types

- Console App (.NET Core)**  
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.  
C# Linux macOS Windows Console
- Console App (.NET Core)**  
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.  
Visual Basic Windows Linux macOS Console
- ASP.NET Core Web Application**  
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.  
C# Linux macOS Windows Cloud Service Web
- Blazor App**  
Project templates for creating Blazor apps that run on the server in an ASP.NET Core app or in the browser on WebAssembly. These templates can be used to build web apps with rich dynamic user interfaces (UIs).  
C# Linux macOS Windows Cloud Web
- ASP.NET Web Application (.NET Framework)

Next

Ready 00:00:19 8/20/2020

# Single Dimensional Array Demo

Configure your new project

Console App (.NET Core) C# Linux macOS Windows Console

Project name:

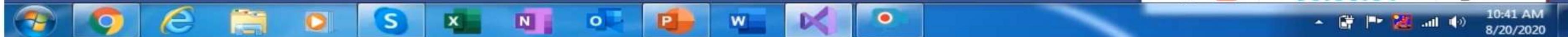
Location:

Solution:

Solution name:  

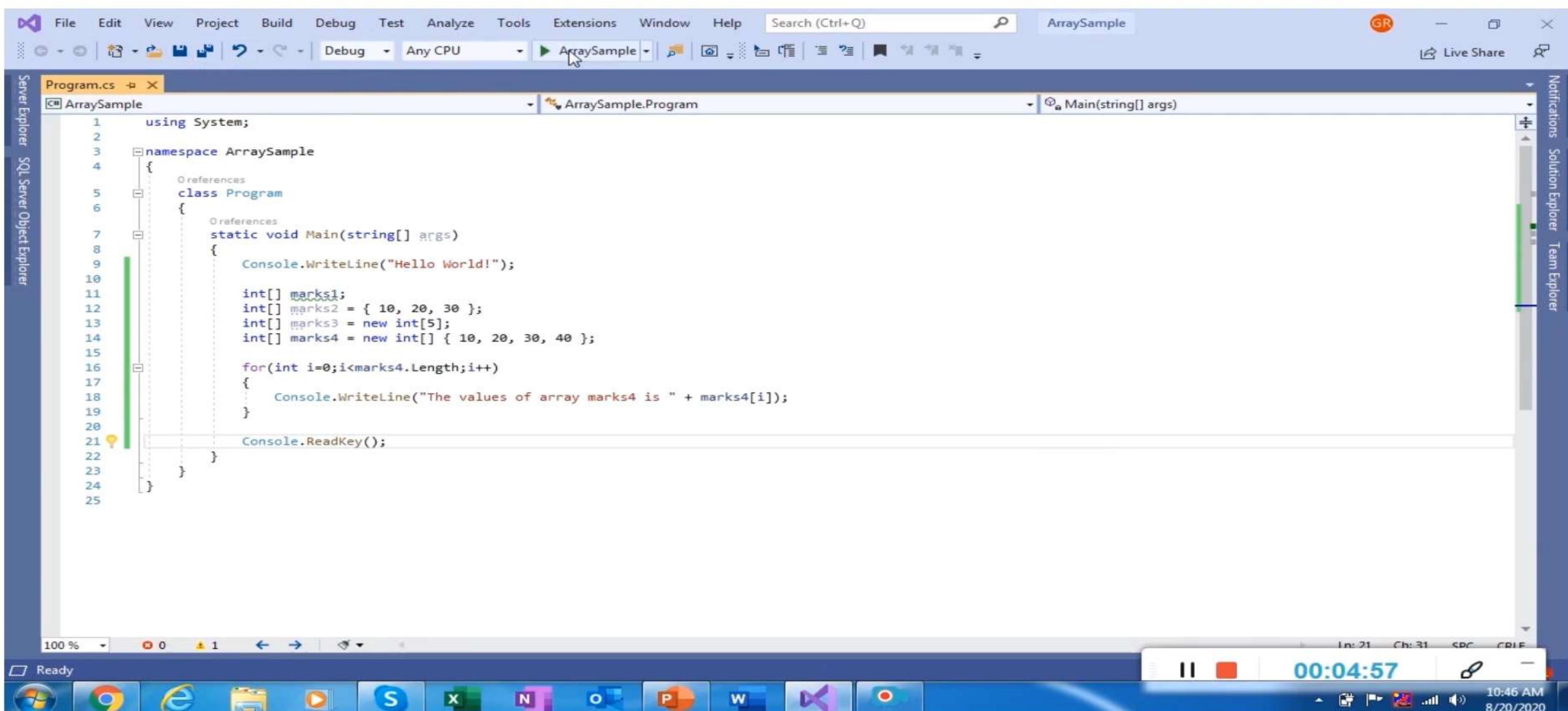
Place solution and project in the same directory

Creating project 'ArraySample'... 00:00:31



10:41 AM 8/20/2020

# Single Dimensional Array Demo

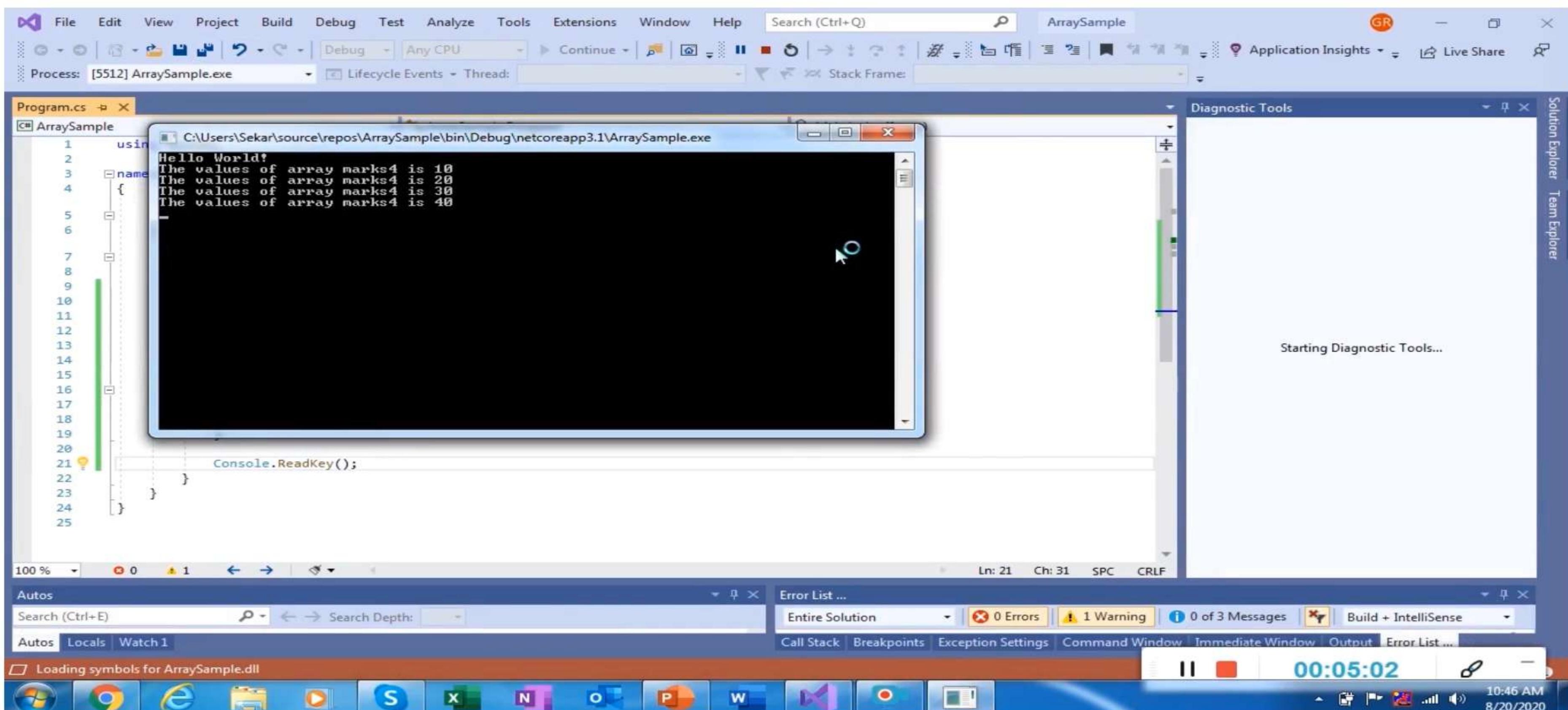


The screenshot shows a Microsoft Visual Studio interface with the following details:

- Title Bar:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search (Ctrl+Q), ArraySample.
- Solution Explorer:** Shows a project named "ArraySample" containing a file "Program.cs".
- Code Editor:** Displays the following C# code:

```
1  using System;
2
3  namespace ArraySample
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Hello World!");
10
11             int[] marks1;
12             int[] marks2 = { 10, 20, 30 };
13             int[] marks3 = new int[5];
14             int[] marks4 = new int[] { 10, 20, 30, 40 };
15
16             for(int i=0;i<marks4.Length;i++)
17             {
18                 Console.WriteLine("The values of array marks4 is " + marks4[i]);
19             }
20
21             Console.ReadKey();
22         }
23     }
24 }
25
```
- Status Bar:** Shows 100%, 0 errors, 1 warning, and other build-related information.
- Taskbar:** Shows icons for various Windows applications like File Explorer, Edge, and File Explorer.
- System Tray:** Shows the date and time (10:46 AM, 8/20/2020) and battery status.

# Single Dimensional Array Demo

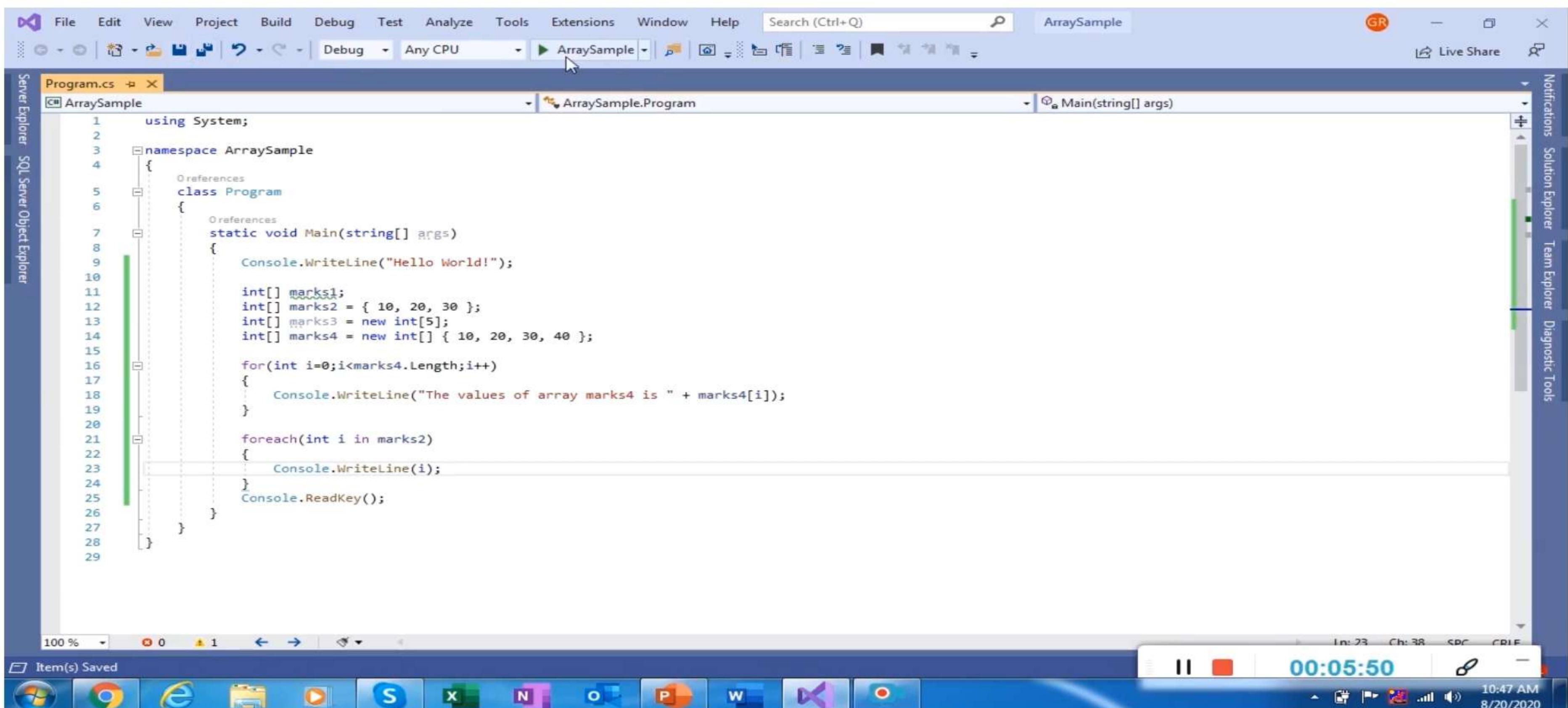


The screenshot shows a Microsoft Visual Studio IDE interface. The main window displays a C# code editor for a file named `Program.cs`. The code contains a `name` variable and a `marks4` array. A break point is set at the `Console.ReadKey();` line. A black terminal window is overlaid on the code editor, showing the output of the application's execution. The output reads:

```
C:\Users\Sekar\source\repos\ArraySample\bin\Debug\netcoreapp3.1\ArraySample.exe
Hello World!
The values of array marks4 is 10
The values of array marks4 is 20
The values of array marks4 is 30
The values of array marks4 is 40
```

The status bar at the bottom shows the time as 10:46 AM and the date as 8/20/2020.

# Single Dimensional Array Demo

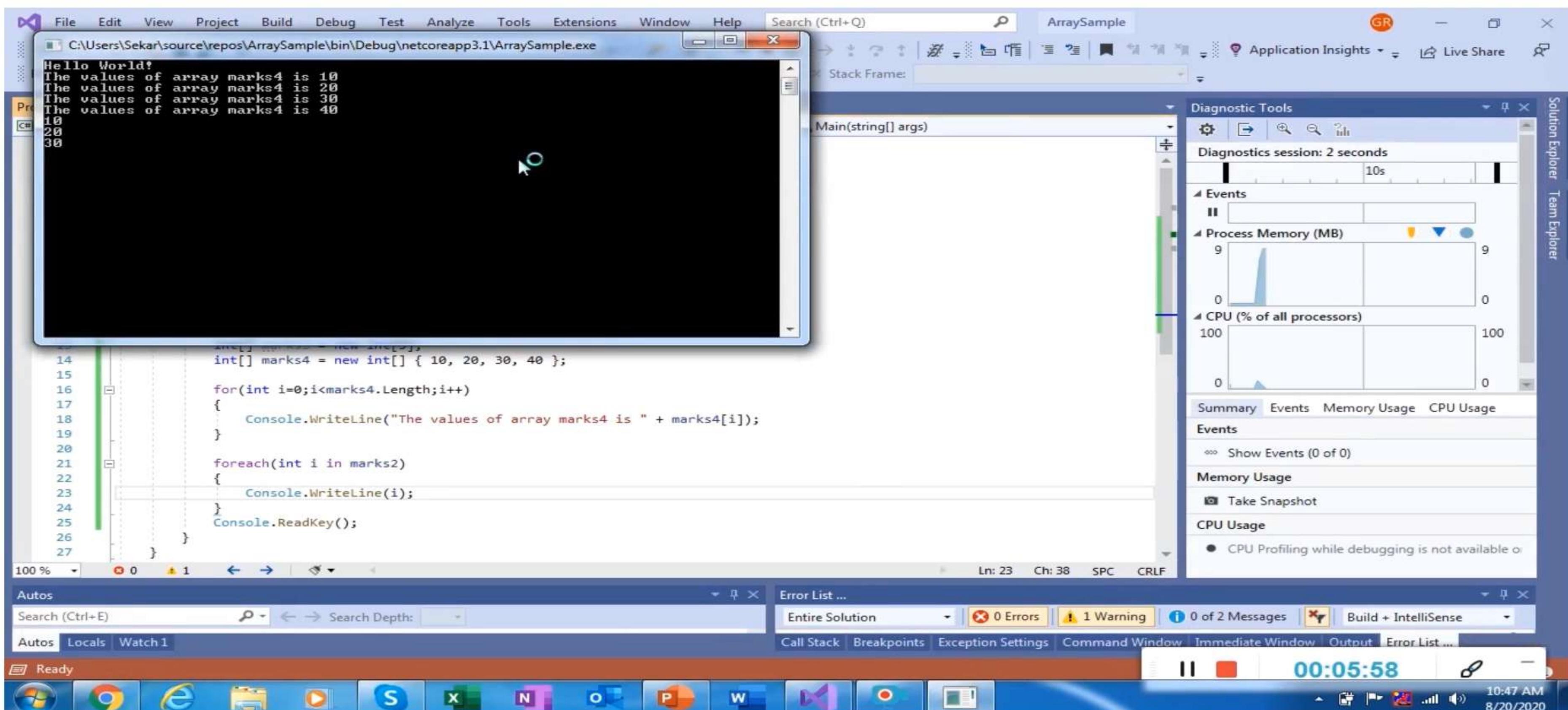


The screenshot shows a Microsoft Visual Studio interface with the following details:

- Title Bar:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search (Ctrl+Q), ArraySample.
- Solution Explorer:** Shows a solution named "ArraySample" containing a project "ArraySample" with a file "Program.cs".
- Code Editor:** Displays the "Program.cs" code:

```
1  using System;
2
3  namespace ArraySample
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Hello World!");
10
11             int[] marks1;
12             int[] marks2 = { 10, 20, 30 };
13             int[] marks3 = new int[5];
14             int[] marks4 = new int[] { 10, 20, 30, 40 };
15
16             for(int i=0;i<marks4.Length;i++)
17             {
18                 Console.WriteLine("The values of array marks4 is " + marks4[i]);
19             }
20
21             foreach(int i in marks2)
22             {
23                 Console.WriteLine(i);
24             }
25             Console.ReadKey();
26         }
27     }
28 }
29 }
```
- Status Bar:** Item(s) Saved, 00:05:50, 10:47 AM, 8/20/2020.

# Single Dimensional Array Demo

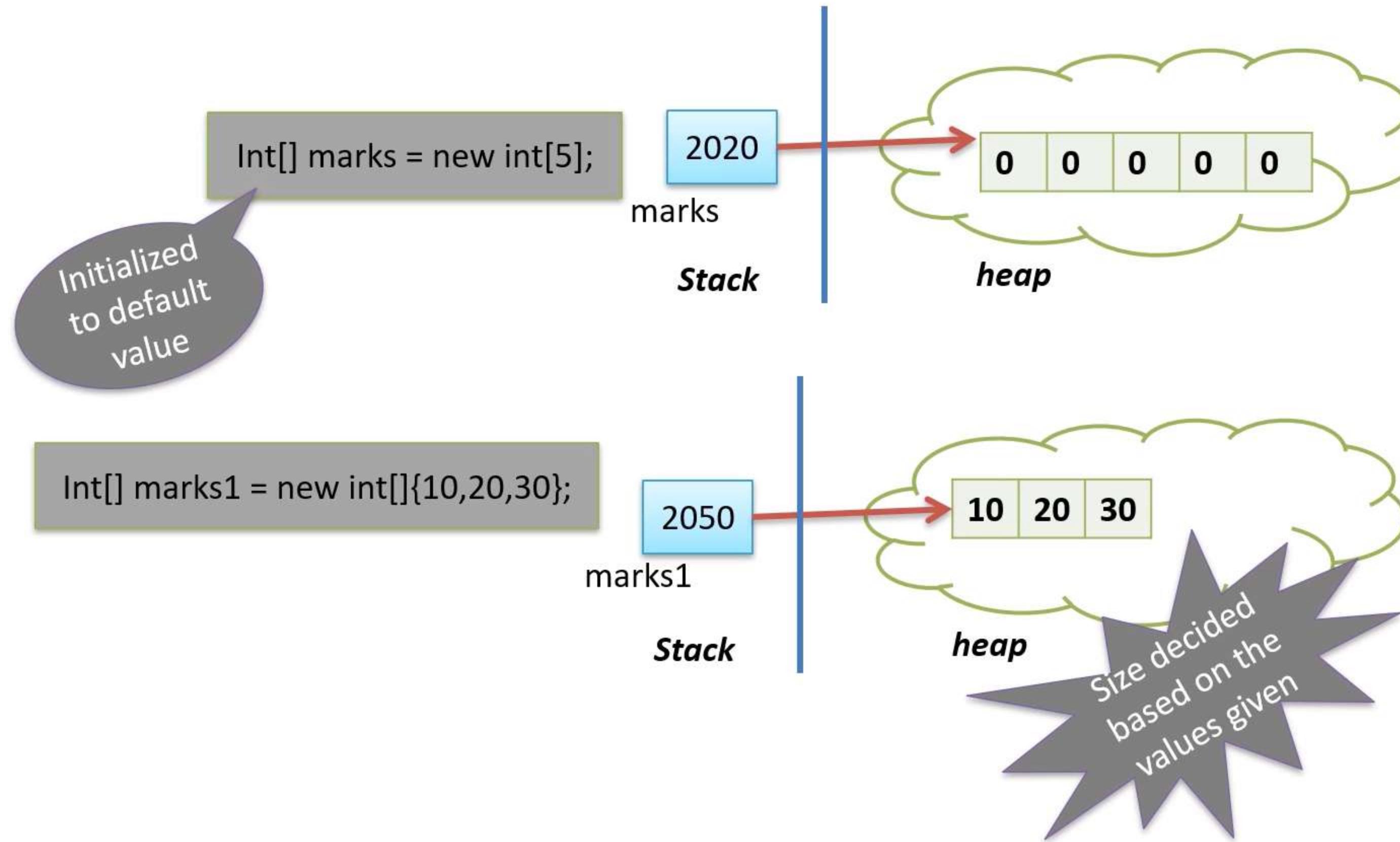


The screenshot shows a Microsoft Visual Studio interface with the following details:

- Title Bar:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search (Ctrl+Q), ArraySample.
- Toolbars:** Standard, Debug, Application Insights, Live Share.
- Diagnostic Tools:** Shows a timeline from 0s to 10s with sections for Events, Process Memory (MB), and CPU (% of all processors). A summary indicates a 2-second diagnostics session.
- Code Editor:** Displays the following C# code:

```
10
20
30
Hello World!
The values of array marks4 is 10
The values of array marks4 is 20
The values of array marks4 is 30
The values of array marks4 is 40
14
15
16
17
18
19
20
21
22
23
24
25
26
27
    int[] marks4 = new int[] { 10, 20, 30, 40 };
    for(int i=0;i<marks4.Length;i++)
    {
        Console.WriteLine("The values of array marks4 is " + marks4[i]);
    }
    foreach(int i in marks2)
    {
        Console.WriteLine(i);
    }
    Console.ReadKey();
}
```
- Output Window:** Shows the console output: "Hello World!", followed by four lines of text: "The values of array marks4 is 10", "The values of array marks4 is 20", "The values of array marks4 is 30", and "The values of array marks4 is 40".
- Taskbar:** Shows various application icons including Windows, Google Chrome, Internet Explorer, File Explorer, and Microsoft Office applications.
- System Tray:** Shows the date and time as 10:47 AM, 8/20/2020.

# Initializing an Array



# Assigning Elements in Array

```
int[] marks=new int[3];  
  
marks[0] = 10;  
  
marks[1] = 20;  
  
marks[2] = 30;
```

```
for(int i=0;i<3;i++)  
{  
    marks[i] = (i+1)*10;  
}
```

Assigning values  
through for loops

# Accessing Elements from Array

```
for(int i=0;i<marks.Length;i++)  
{  
    Console.WriteLine(marks[i]);  
}
```

*foreach loop*

**foreach** (data type iterator in  
array variable)

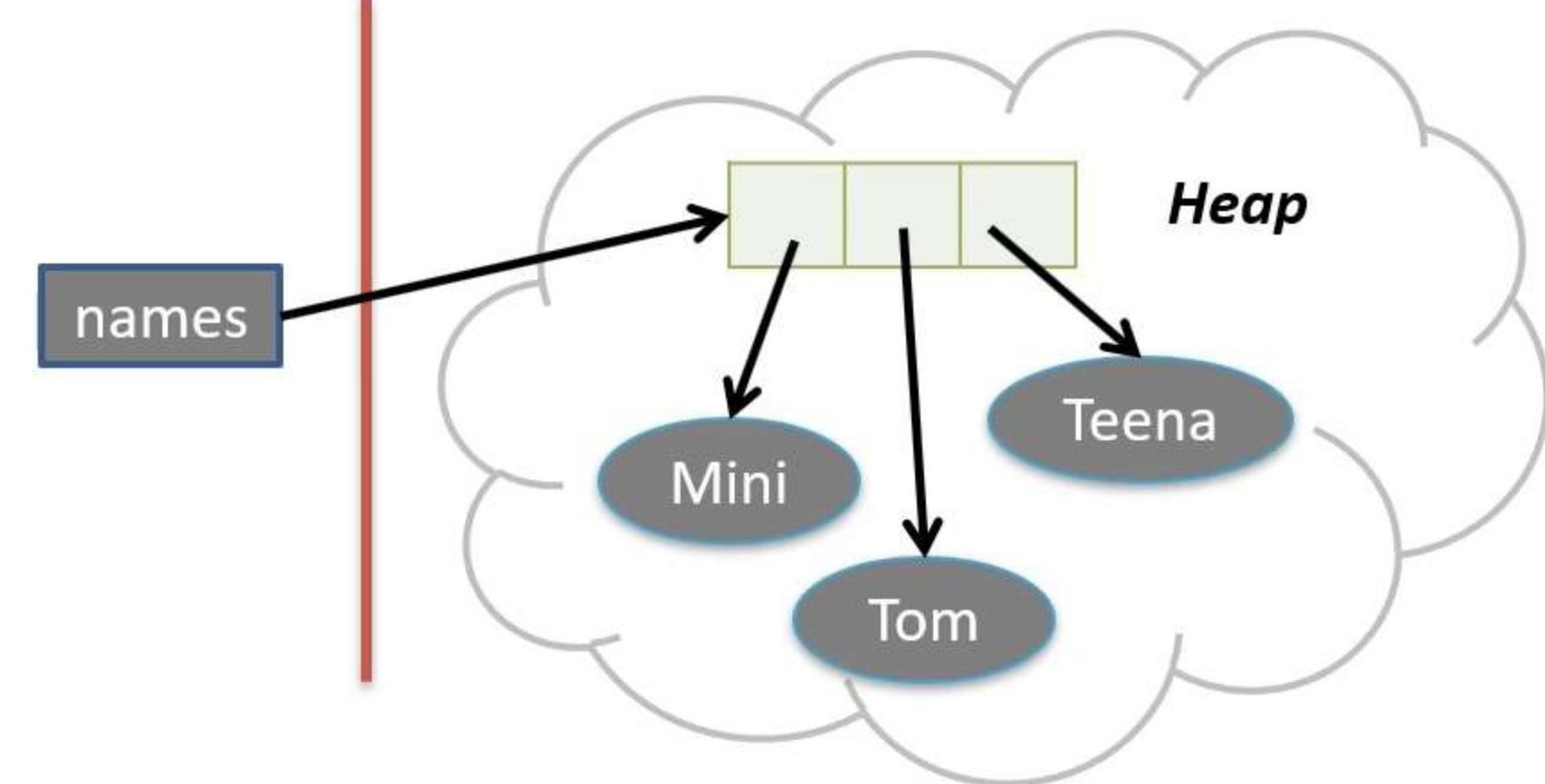


```
int[] numbers = {4, 5, 6, 1, 2, , -2, -1, 0};  
foreach (int i in numbers)  
{  
    System.Console.WriteLine(i);  
}
```

# Creating a String Array

```
String[] names = new String[3];  
names[0] = new String("Mini");  
names[1] = "Tom";  
names[2] = new String("Teena");
```

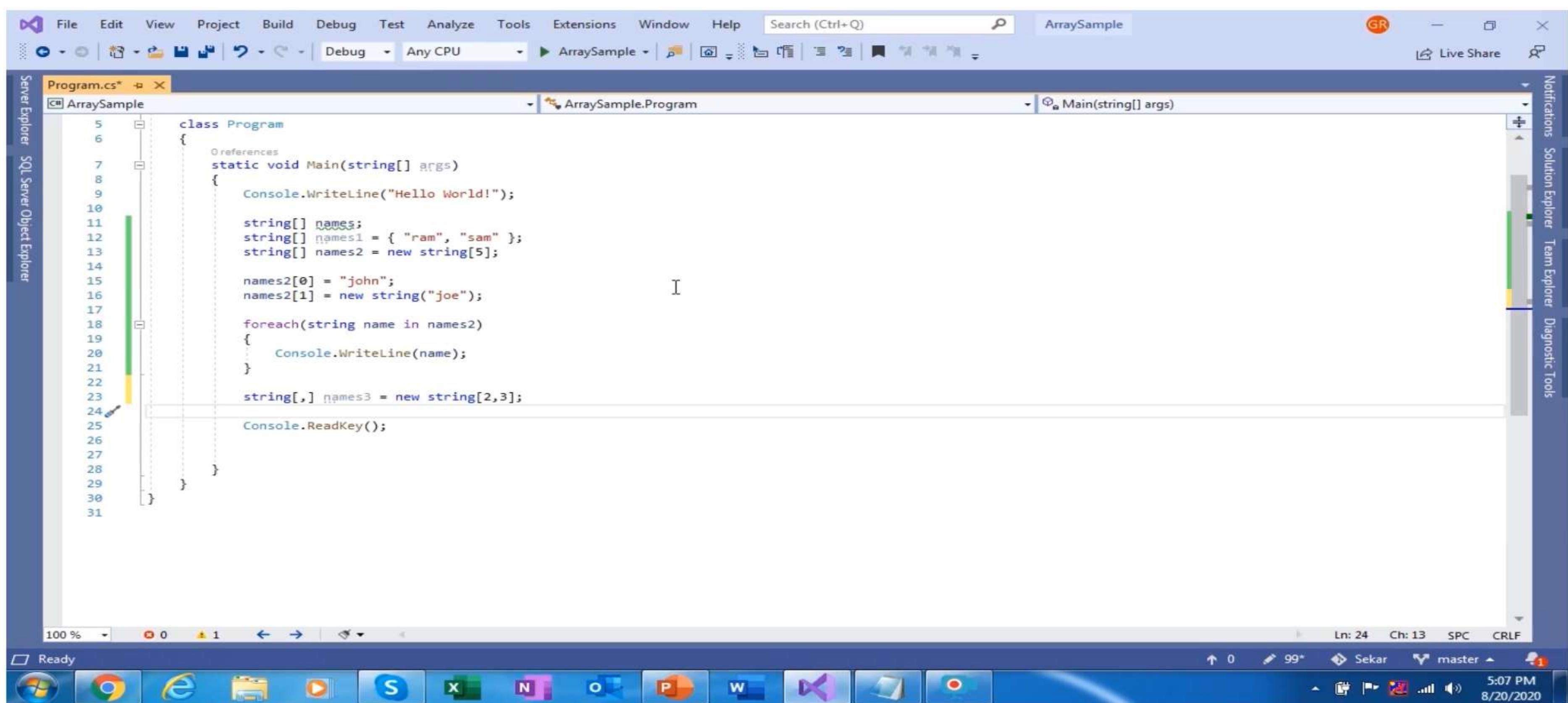
Arrays for reference  
data types holds  
the address of the  
array elements





# Code Demo

# String Array Demo



The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "ArraySample". The main code editor window displays the following C# code:

```
5 class Program
6 {
7     static void Main(string[] args)
8     {
9         Console.WriteLine("Hello World!");
10
11        string[] names;
12        string[] names1 = { "ram", "sam" };
13        string[] names2 = new string[5];
14
15        names2[0] = "john";
16        names2[1] = new string("joe");
17
18        foreach(string name in names2)
19        {
20            Console.WriteLine(name);
21        }
22
23        string[,] names3 = new string[2,3];
24
25        Console.ReadKey();
26
27    }
28
29 }
30
31 }
```

The code demonstrates various ways to declare and initialize string arrays in C#. It includes a standard array declaration, an array initialized with values, and a multidimensional array declaration. A foreach loop is used to iterate through the elements of the array. The program also includes a call to `Console.ReadKey()` at the end.

# System.Array

**System.Array** is the base class for all arrays

The **System.Array** class provides many other useful methods/properties, such as methods for sorting, searching, and copying arrays.

*System.Array* has a useful read-only property named **Length** that can be used to find the length, or size, of an array programmatically.

```
int[] marks = { 80, 70, 90, 85, 92 };
for (int i = 0; i < marks.Length; i++)
{
    Console.WriteLine(marks[i]);
}
```

# Length Property

- Length property
  - Gets the length of the array ie. The number of elements in an array.

<array\_name>.Length

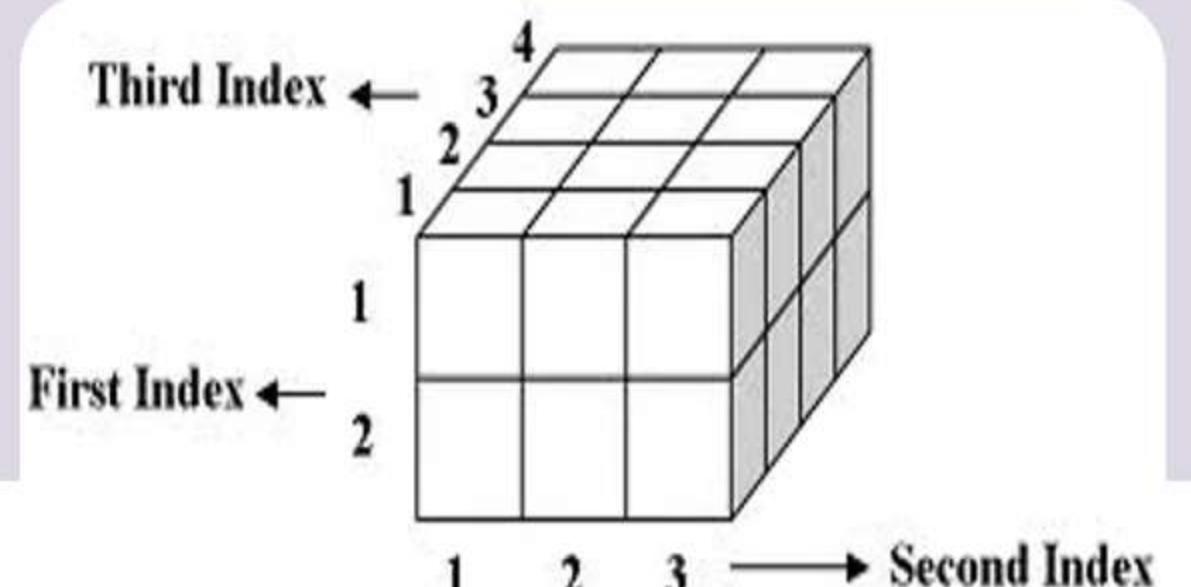
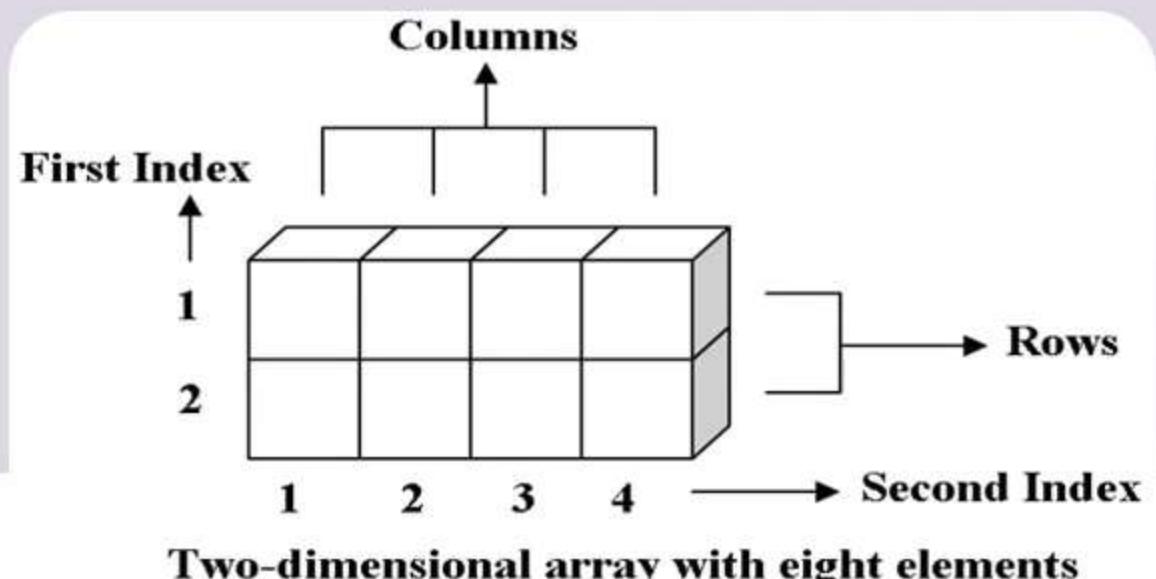
```
Int[] marks=new int[3];  
for(int i=0;i<marks.Length;i++)  
{  
    marks[i] = (i+1)*10;  
}
```

Iterates the  
loop for 3  
times

# Multidimensional Array

Arrays can have more than one dimension.

A multidimensional array is indexed by two or more integers.



## 2 D Array

- Arrays that have two dimensions
- `int array2D[ , ] = new int[2,3];`

## 3D Array

Arrays that have three dimensions  
`Int[ , , ] array3D=new int[2 , 3, 4];`

An n-dimensional array is where, n is the depth of the chain of arrays.

# 2-D Array

- Data is stored in the form of rows and columns
- Also known as matrix form

```
int matrix[ , ] = new int[3,3] → Columns
```

rows

All the elements of the array are auto-initialized to their default values (with zeroes).

0	[0,0]	[0,1]	[0,2]
1	[1,0]	[1,1]	[1,2]
2	[2,0]	[2,1]	[2,2]

matrix

0	0	0
0	0	0
0	0	0

*Heap*

# 2-D Array

## Initializing 2-D Array

```
int matrix[ , ] = new int[3,3] → Columns
```

rows

```
matrix[0, 0] = 20;  
matrix[0, 1] = 30;  
matrix[0, 2] = 40  
matrix[1, 0] = 20;  
matrix[1, 1] = 30;  
matrix[1, 2] = 40  
matrix[2, 0] = 20;  
matrix[2, 1] = 30;  
matrix[2, 2] = 40
```

0	[0,0]	[0,1]	[0,2]
1	[1,0]	[1,1]	[1,2]
2	[2,0]	[2,1]	[2,2]

matrix

10	20	30
40	50	60
70	80	90

Heap

# Initialize Arrays at Declaration

- int[,] numbers = new int[3, 2] { {1, 2}, {3, 4}, {5, 6} };
- string[,] siblings = new string[2, 2] { {"Mike", "Amy"}, {"Mary", "Albert"} };

Initialize arrays at declaration time  
by enclosing the initial values in  
curly braces {}

# Iterating through Multidimensional Array

**GetLength(i) method returns the length of the i th dimension  
Eg., matrix.GetLength(1) returns 20**

[0,0]	[0,1]	[0,2]
[1,0]	[1,1]	[1,2]
[2,0]	[2,1]	[2,2]

10	20	30
40	50	60
70	80	90

**Nested *for loops* to iterate over the array:**

```
for (int row = 0; row < matrix.GetLength(0);  
row++)  
{  
    for (int col = 0; col < matrix.GetLength(1);  
col++)  
    {  
        Console.WriteLine("Element at {0},{1} is  
        {2}",row,col,matrix[row,col]);  
    }  
}
```

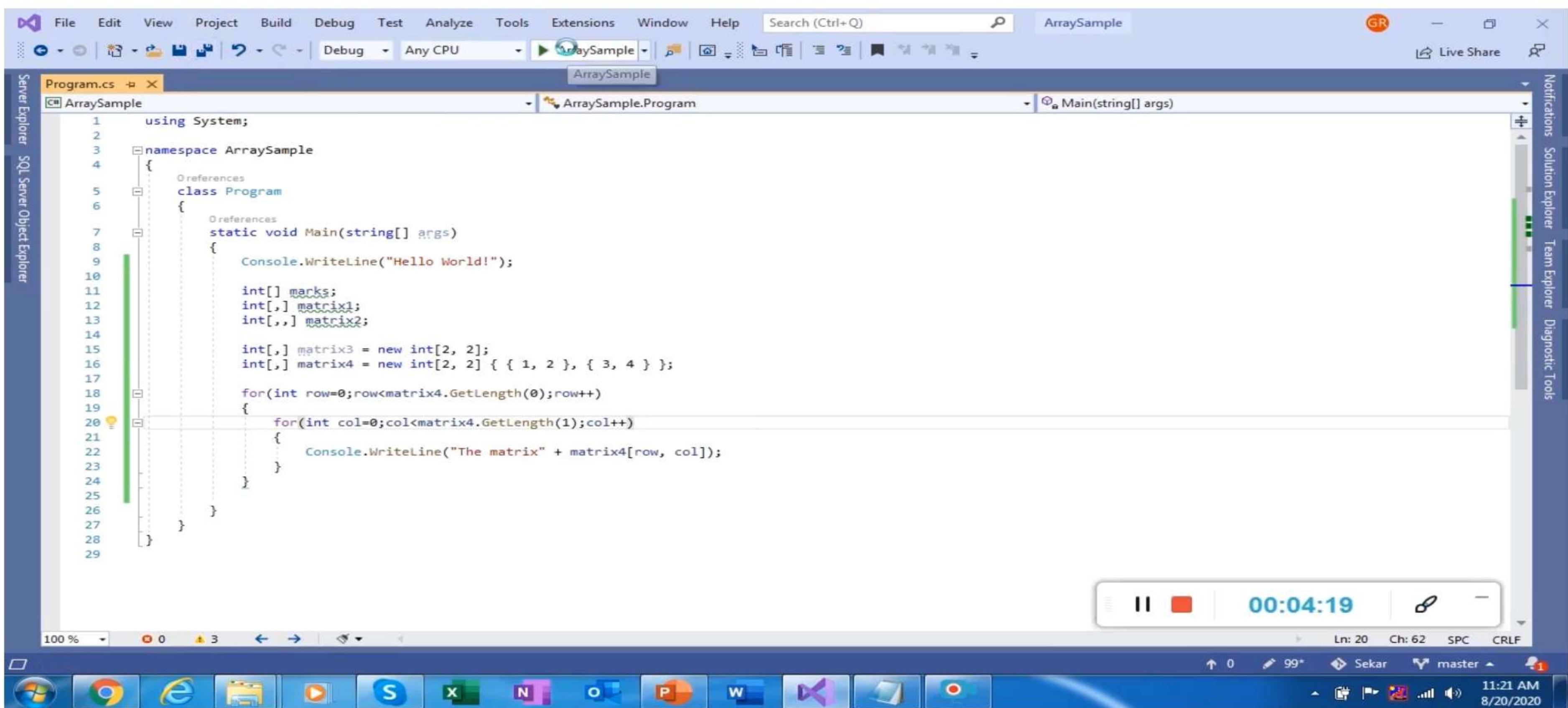
**Iterating the elements in a 2D  
Array using *foreach loop***

```
foreach(int i in numbers)  
{  
    Console.WriteLine("{0} ", i);  
}
```



# Code Demo

# Two Dimensional Array Demo



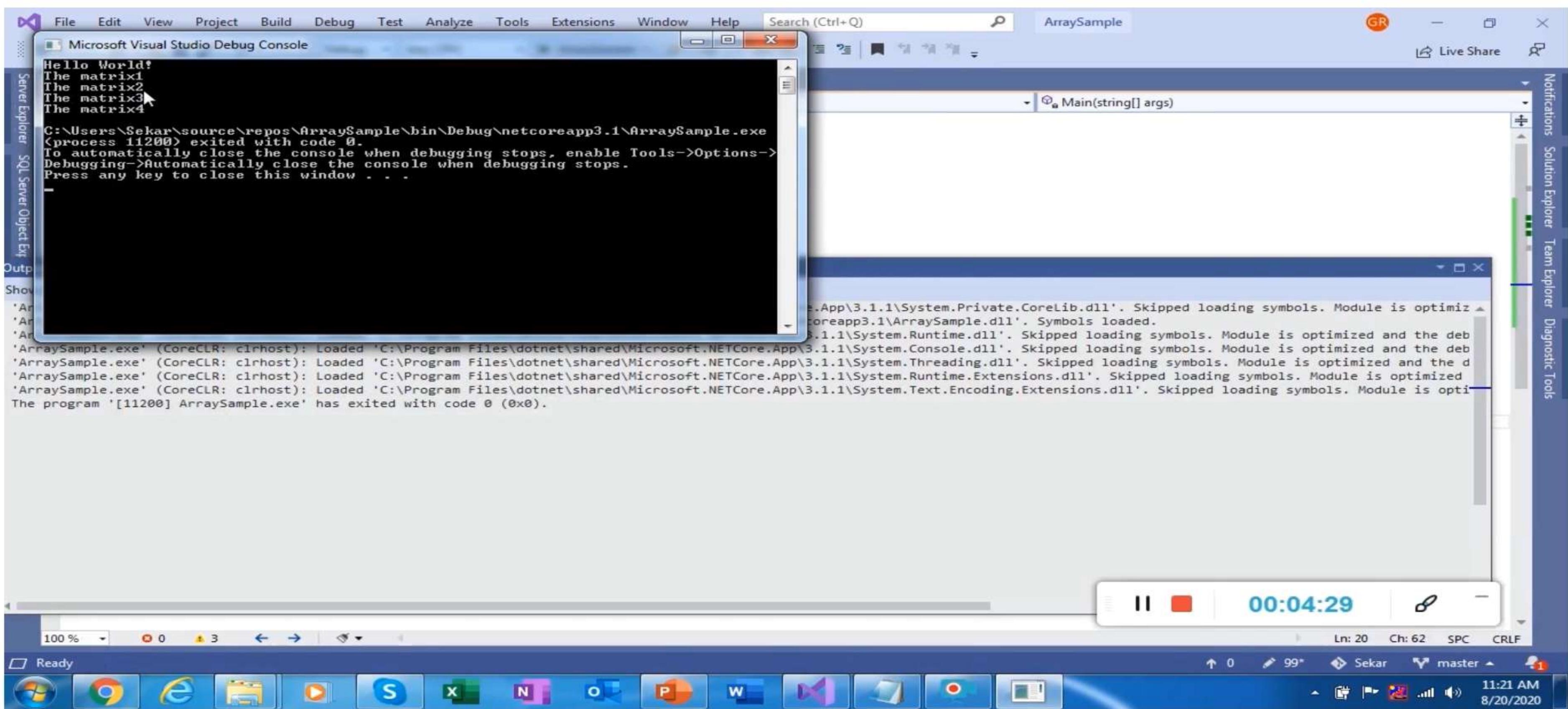
The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "ArraySample". The menu bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar "Search (Ctrl+Q)". The toolbar has icons for file operations like Open, Save, and Print, along with a "Live Share" button.

The main code editor window displays "Program.cs" with the following C# code:

```
1  using System;
2
3  namespace ArraySample
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Hello World!");
10
11             int[] marks;
12             int[,] matrix1;
13             int[, ,] matrix2;
14
15             int[,] matrix3 = new int[2, 2];
16             int[,] matrix4 = new int[2, 2] { { 1, 2 }, { 3, 4 } };
17
18             for(int row=0;row<matrix4.GetLength(0);row++)
19             {
20                 for(int col=0;col<matrix4.GetLength(1);col++)
21                 {
22                     Console.WriteLine("The matrix" + matrix4[row, col]);
23                 }
24             }
25         }
26     }
27 }
28
29 }
```

The status bar at the bottom shows "00:04:19" and other system information like "Ln: 20 Ch: 62 SPC CRLF". The taskbar at the bottom includes icons for various Windows applications like File Explorer, Edge, and Microsoft Office.

# Two Dimensional Array Demo



A screenshot of Microsoft Visual Studio showing a console application named "ArraySample". The application displays the output of a two-dimensional array manipulation program. The output window shows:

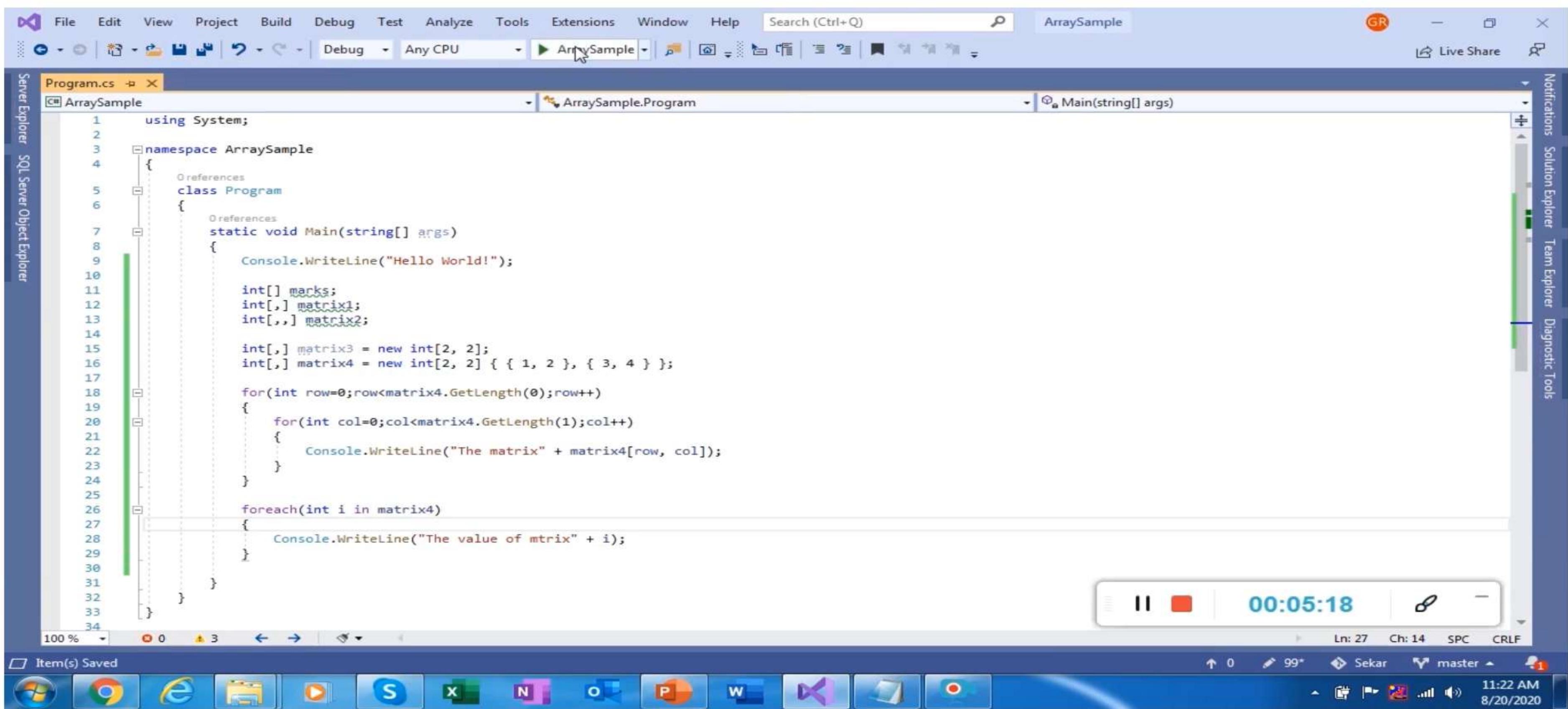
```
Hello World!
The matrix1
The matrix2
The matrix3
The matrix4

C:\Users\Sekar\source\repos\ArraySample\bin\Debug\netcoreapp3.1\ArraySample.exe
(process 11200) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->
Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

[Output window content]
System.Private.CoreLib.dll'. Skipped loading symbols. Module is optimized and the deb
coreapp3.1\ArraySample.dll'. Symbols loaded.
System.Runtime.dll'. Skipped loading symbols. Module is optimized and the deb
System.Console.dll'. Skipped loading symbols. Module is optimized and the deb
System.Threading.dll'. Skipped loading symbols. Module is optimized and the deb
System.Runtime.Extensions.dll'. Skipped loading symbols. Module is optimized and the deb
System.Text.Encoding.Extensions.dll'. Skipped loading symbols. Module is optimized and the deb
The program '[11200] ArraySample.exe' has exited with code 0 (0x0).
```

The status bar at the bottom shows the time as 11:21 AM and the date as 8/20/2020.

# Two Dimensional Array Demo

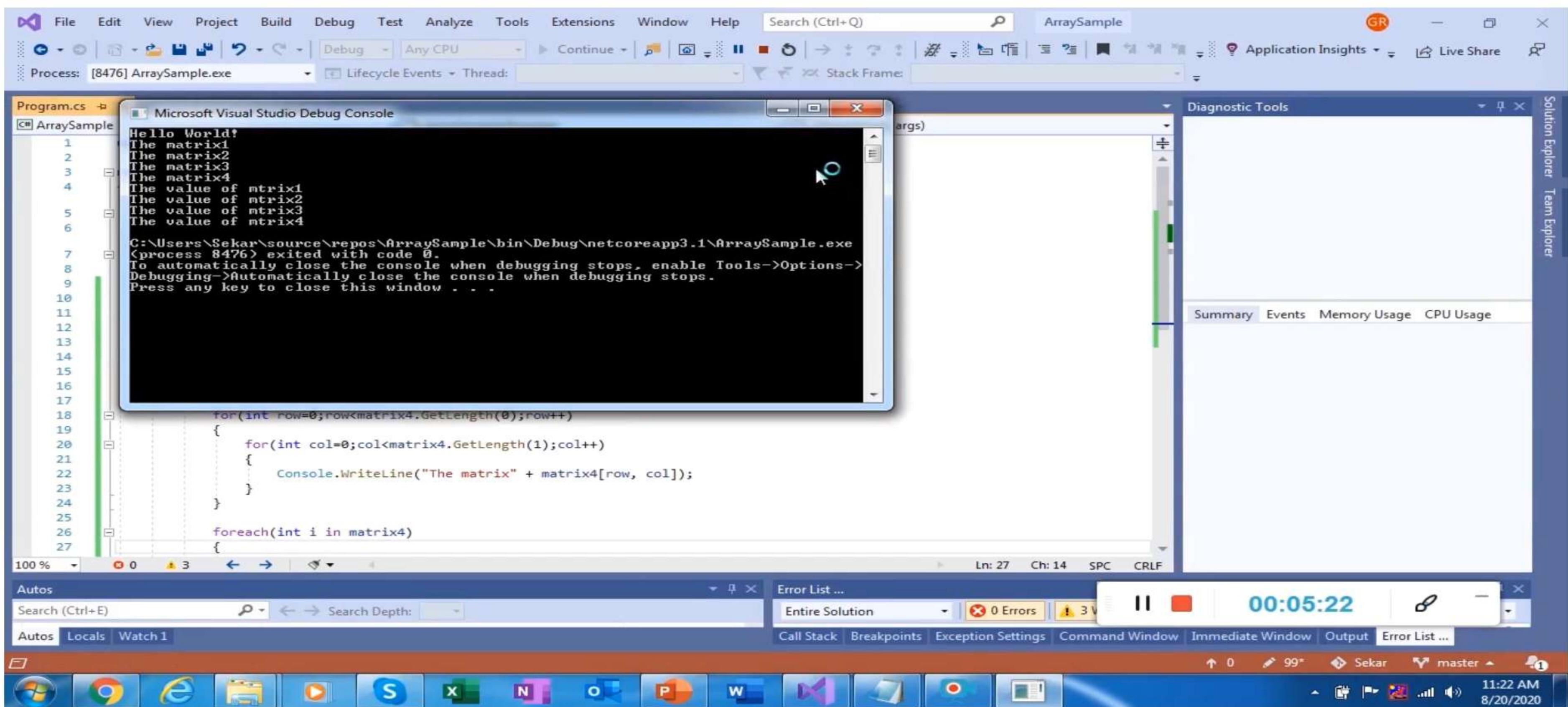


The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "ArraySample". The main code editor window displays the following C# code:

```
1  using System;
2
3  namespace ArraySample
4  {
5      class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Hello World!");
10
11             int[] marks;
12             int[,] matrix1;
13             int[, ,] matrix2;
14
15             int[,] matrix3 = new int[2, 2];
16             int[,] matrix4 = new int[2, 2] { { 1, 2 }, { 3, 4 } };
17
18             for(int row=0;row<matrix4.GetLength(0);row++)
19             {
20                 for(int col=0;col<matrix4.GetLength(1);col++)
21                 {
22                     Console.WriteLine("The matrix" + matrix4[row, col]);
23                 }
24             }
25
26             foreach(int i in matrix4)
27             {
28                 Console.WriteLine("The value of mtrix" + i);
29             }
30         }
31     }
32 }
33 }
```

The status bar at the bottom shows "00:05:18" and other standard status indicators. The right side of the interface includes the Solution Explorer, Team Explorer, and Diagnostic Tools panes.

# Two Dimensional Array Demo



The screenshot shows a Microsoft Visual Studio interface during a debugging session. The title bar reads "ArraySample". The toolbar includes standard options like File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Application Insights. The status bar at the bottom shows "0 Errors", "0 Warnings", and a timestamp of "00:05:22".

The main window displays the "Program.cs" code file:

```
1  Hello World!
2  The matrix1
3  The matrix2
4  The matrix3
5  The matrix4
6  The value of mtrix1
7  The value of mtrix2
8  The value of mtrix3
9  The value of mtrix4
10
11
12
13
14
15
16
17
18    for(int row=0;row<matrix4.GetLength(0);row++)
19    {
20      for(int col=0;col<matrix4.GetLength(1);col++)
21      {
22        Console.WriteLine("The matrix" + matrix4[row, col]);
23      }
24    }
25
26    foreach(int i in matrix4)
27    {
```

A "Microsoft Visual Studio Debug Console" window is open, showing the output of the program:

```
Hello World!
The matrix1
The matrix2
The matrix3
The matrix4
The value of mtrix1
The value of mtrix2
The value of mtrix3
The value of mtrix4

C:\Users\Sekar\source\repos\ArraySample\bin\Debug\netcoreapp3.1\ArraySample.exe
(process 8476) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->
Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

The "Diagnostic Tools" window is also visible on the right side of the interface.

# Jagged Array

A jagged Array is an array whose elements are arrays.

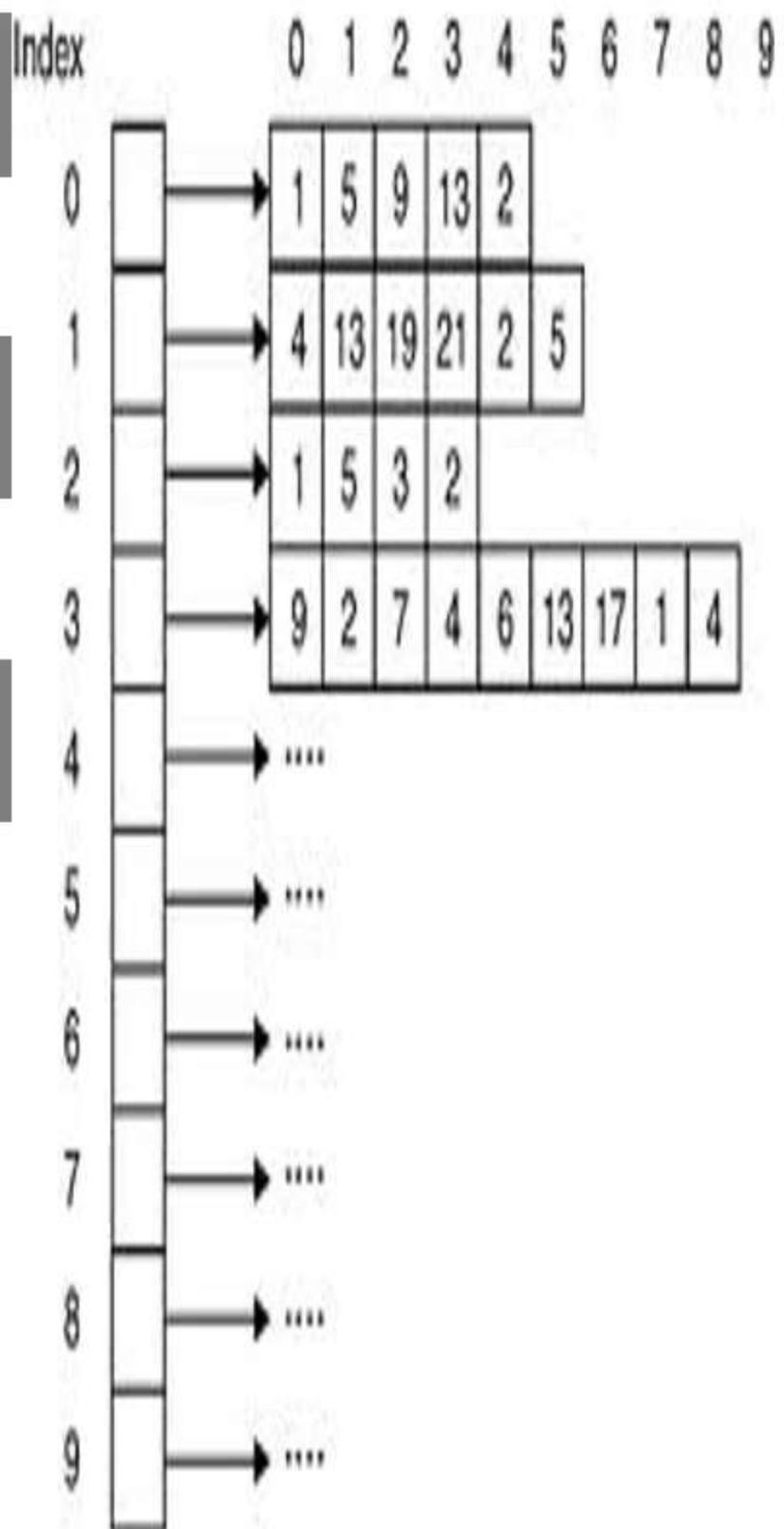
- A Jagged Array is sometimes called an “Array of Arrays.”

The elements of a Jagged Array can be of different dimensions and sizes.

- Each row of a Jagged Array can be of different length.

```
int numbers=new int[10][];
```

- numbers[0]=new int[5];
  - numbers[1]=new int[6];
  - numbers[2]=new int[4];
  - .....
  - Numbers[9]=new int[3];



# Jagged Array

```
string[][] names=new string[3][];  
for(int i=0;i<names.Length;i++){  
    int col=Convert.ToInt32( Console.ReadLine());  
    names[i]=new string[col];  
}  
  
for(int i=0;i<names.Length;i++){  
    for(int j=0;j<names[i].Length;j++){  
        Console.WriteLine(names[i][j]);  
    }  
    System.out.println("\n");  
}
```

Creates a Jagged Array named *names*

Allocates memory to columns

Gives the number of rows

Gives the number of columns in each row



# Iterating Elements of a Jagged Array

***foreach loop*** to access the elements of the array:

```
foreach (int[] row in rowLength)
{
    foreach (int col in row)
    {
        Console.WriteLine(col);
    }
    Console.WriteLine();
}
```

# Summary

- Single Dimensional Arrays
- Multi-Dimensional Arrays
- Jagged arrays

