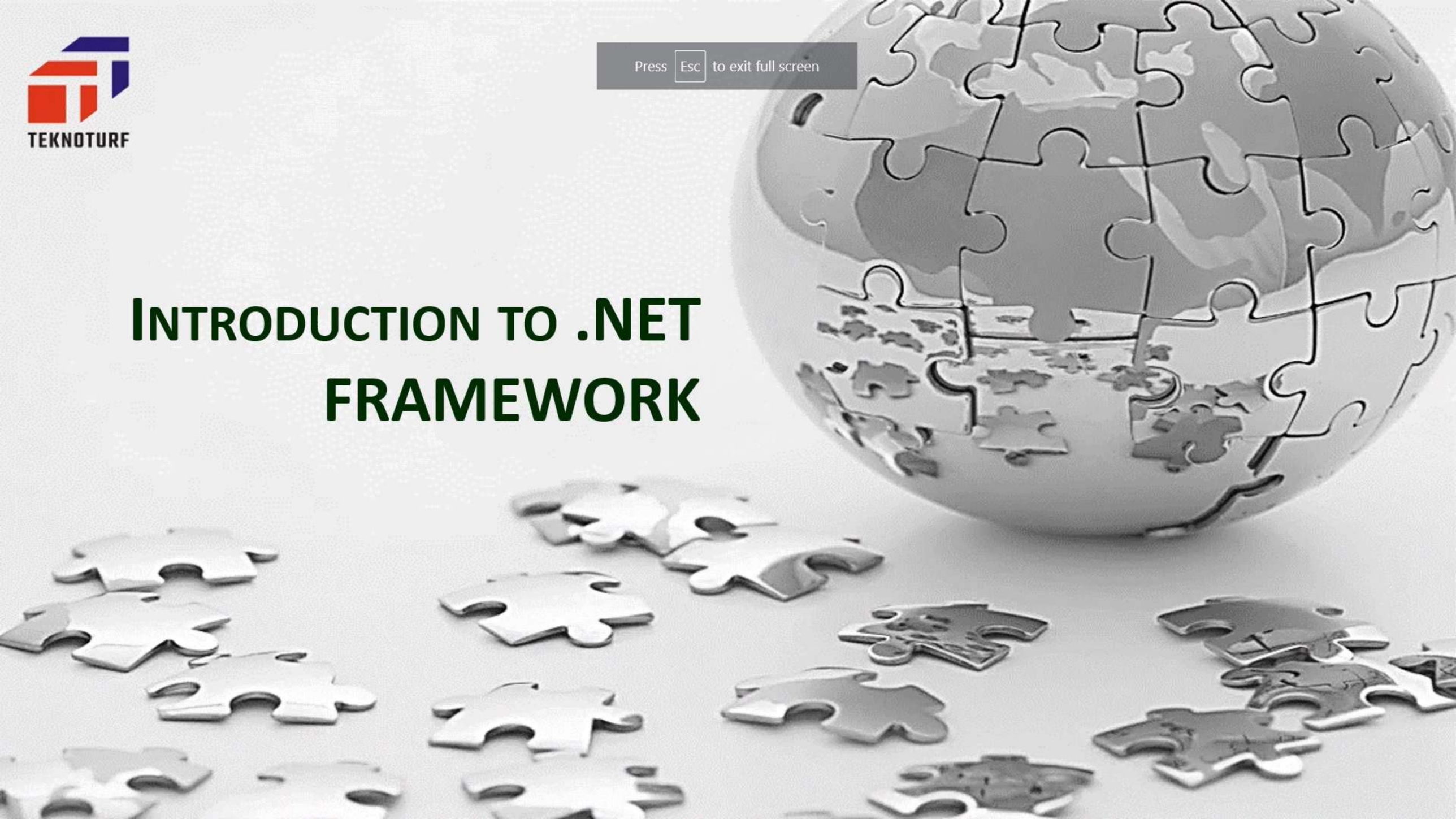
TEKNOTURF

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

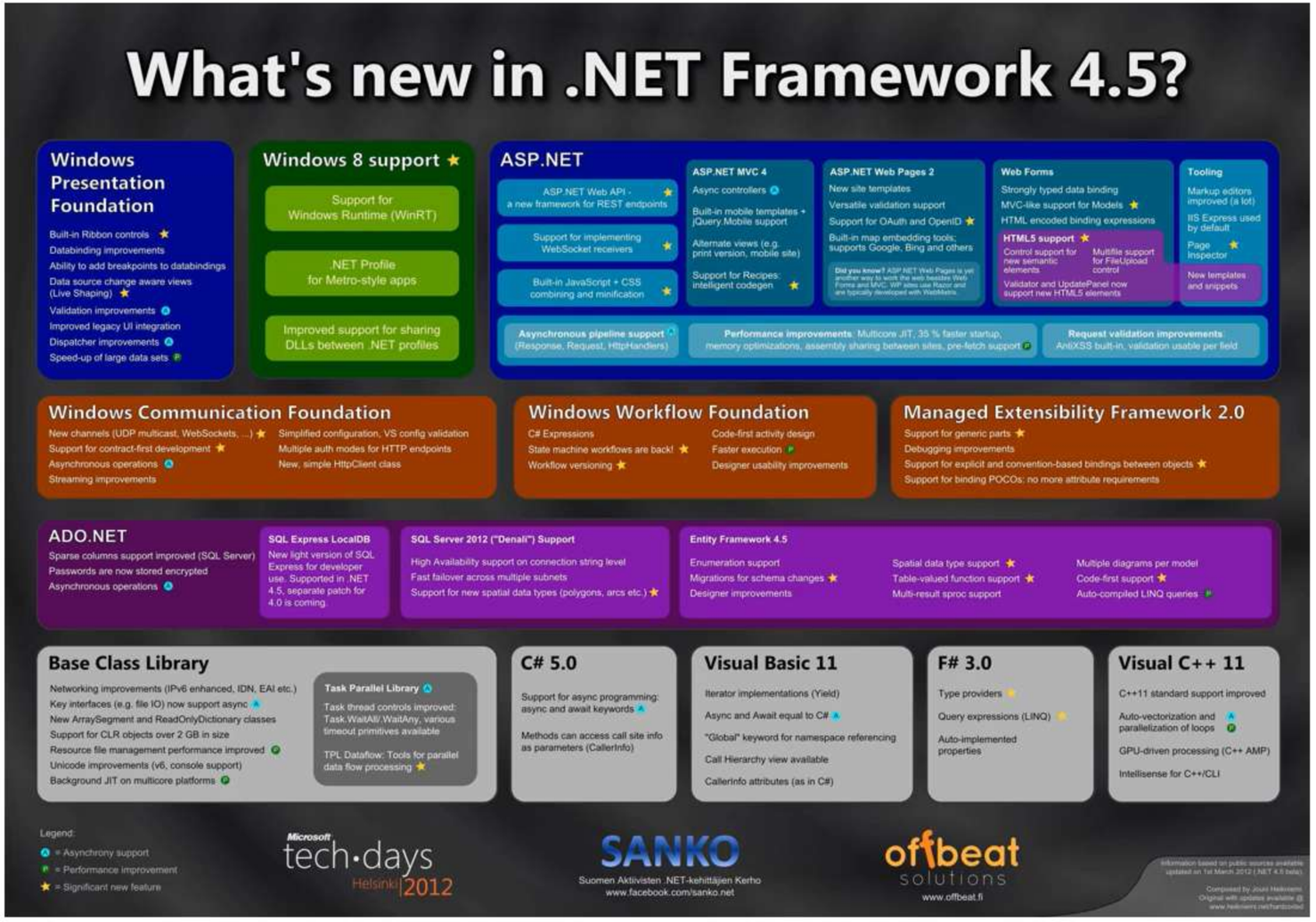
A development platform for building applications for Windows, Windows Phone and Windows Server

.NET Framework

TEKNOTURF

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

.NET Framework Architecture



.NET Framework Architecture

TEKNOTURF

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

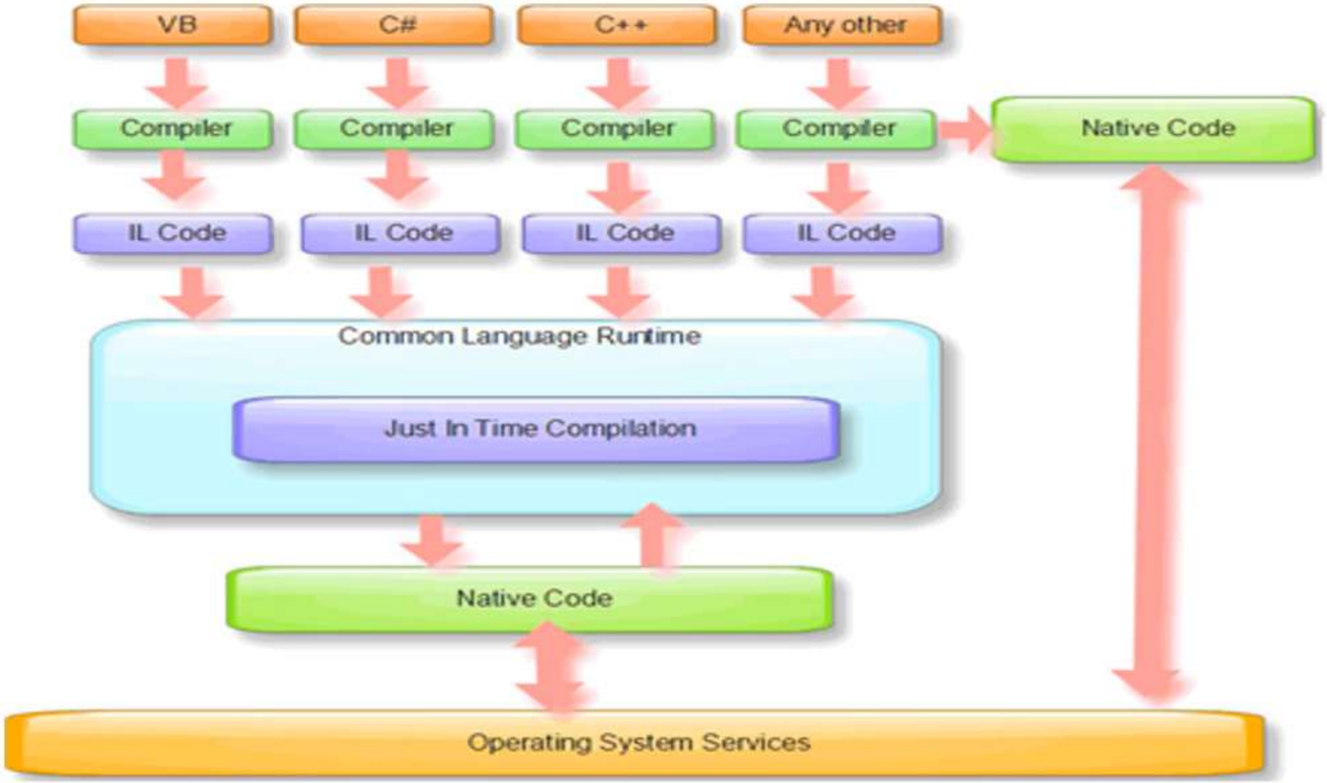
- Traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.

Common Language Runtime (CLR)

TEKNOTURF

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

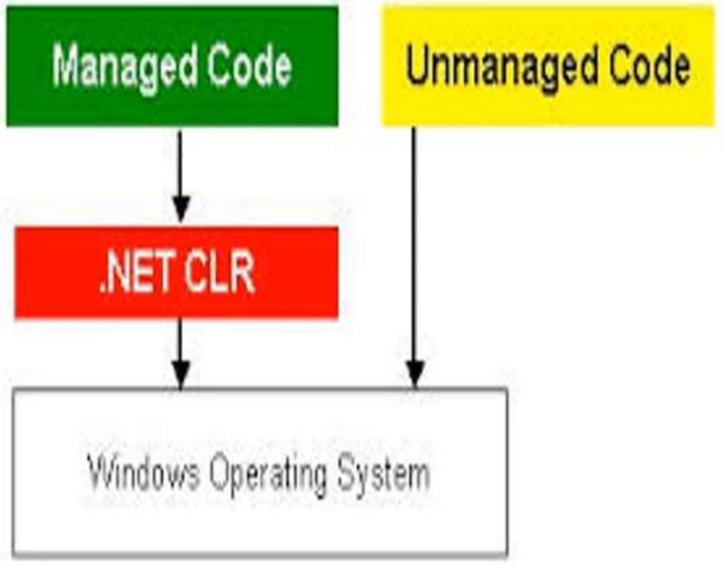


CLR and Code Management

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

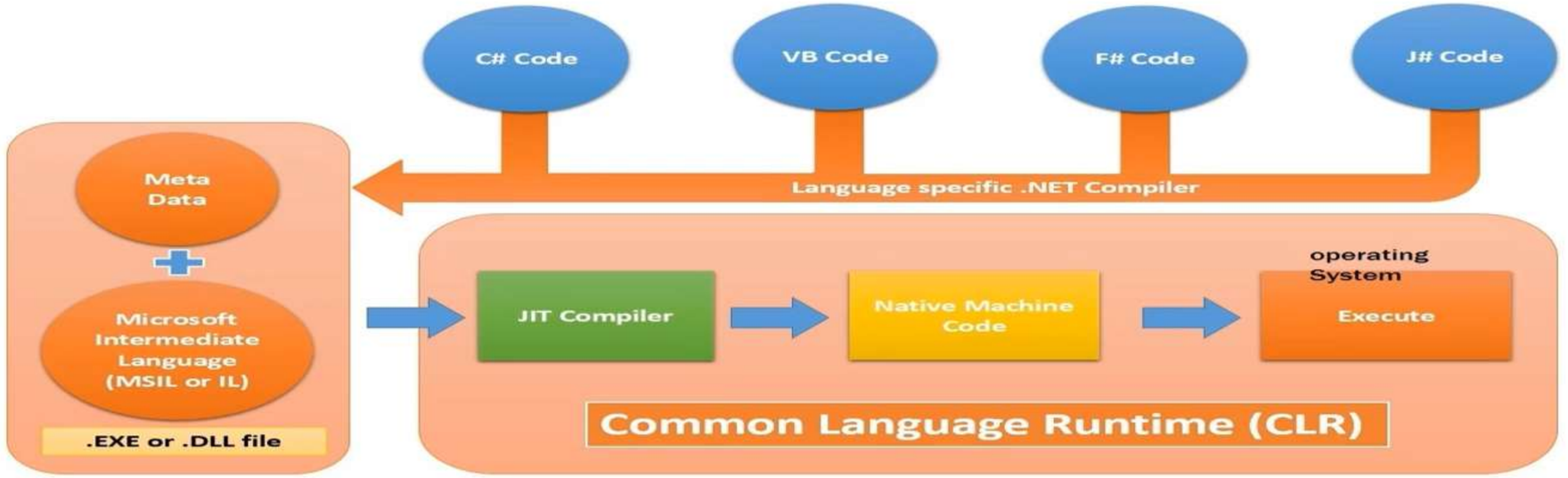


Intermediate Language

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

TEKNOTURF

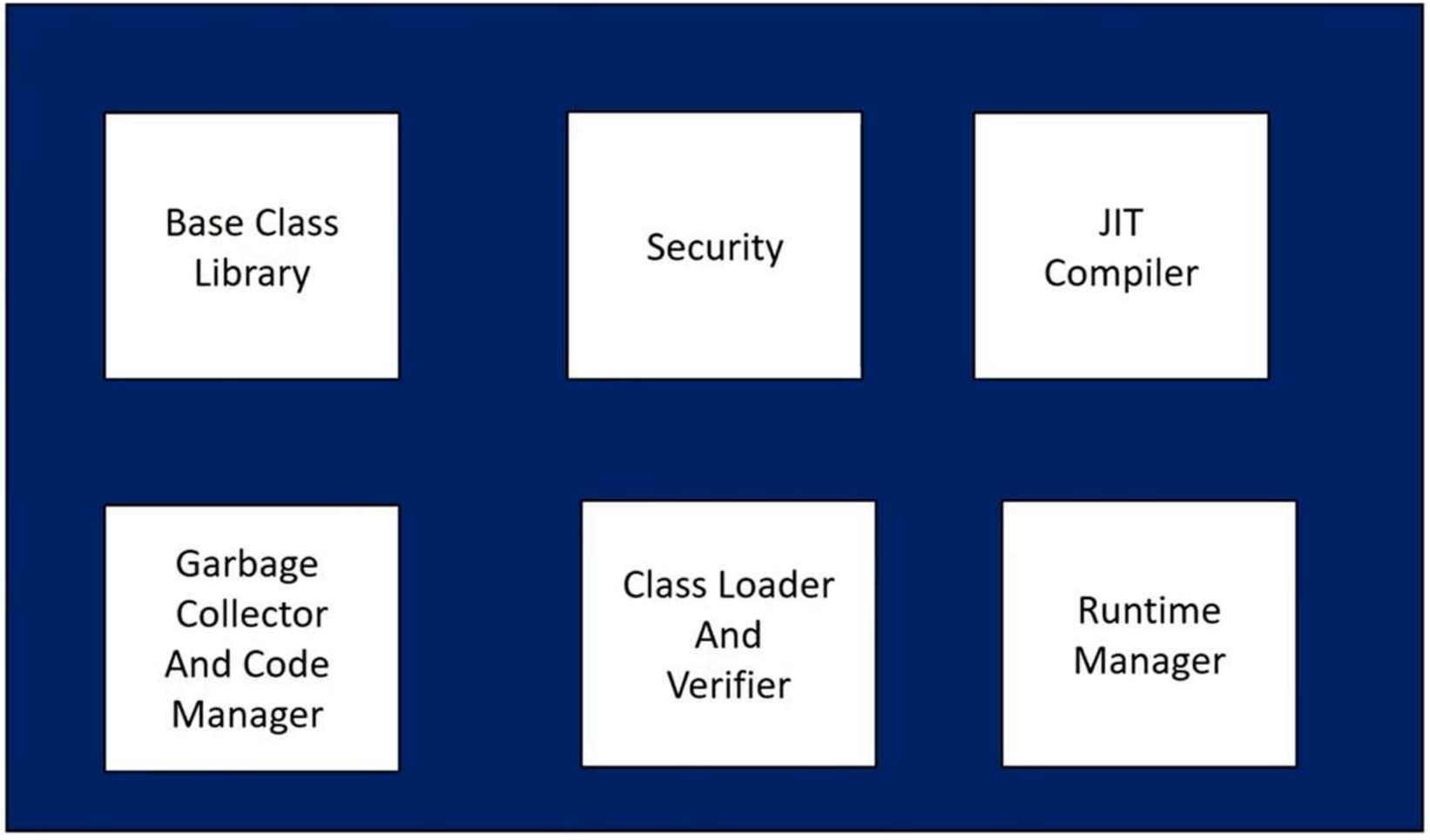
Assemblies form the fundamental unit of deployment, version control, reuse, activation scoping, and security permissions for a .NET-based application.

Assemblies can contain one or more modules.

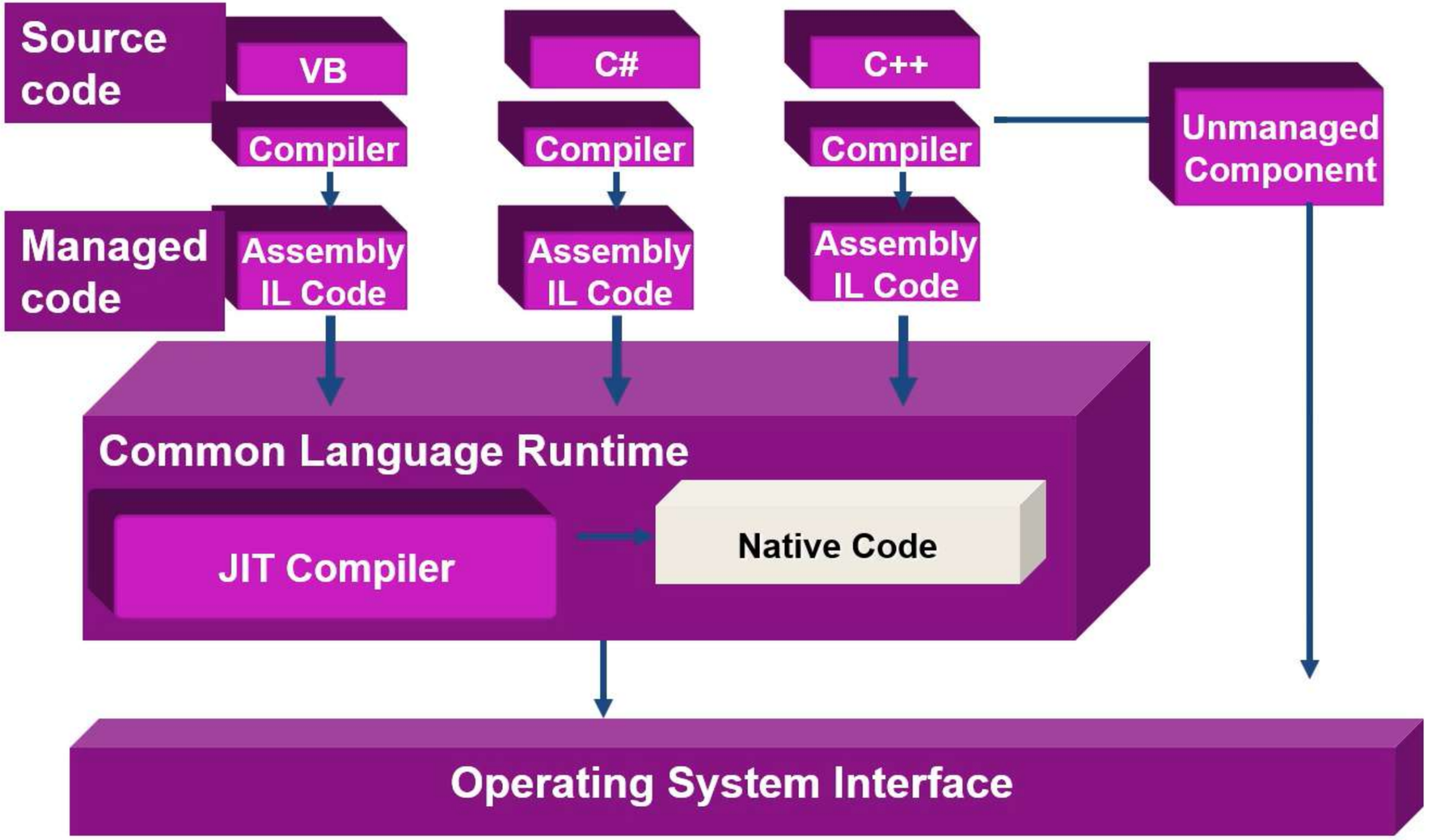
Assemblies are implemented as .exe or .dll files.

CLR - Components

TEKNOTURF



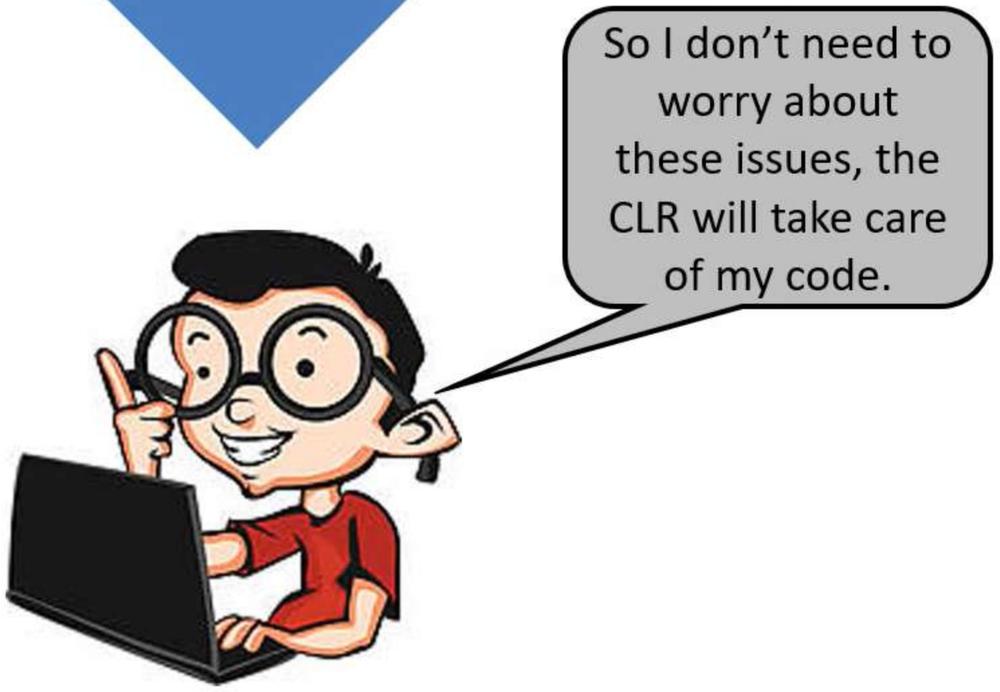
CLR -Execution Model

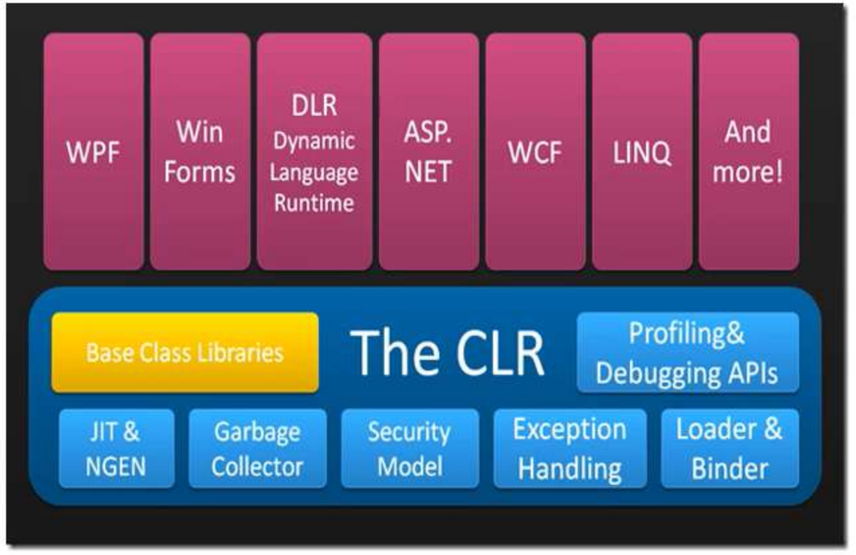


Common Language Runtime Services

TEKNOTURF

- 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

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

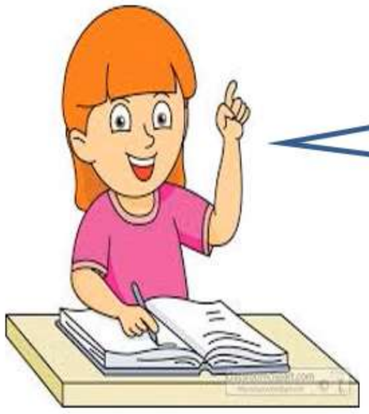
The garbage collector (GC) within the CLR manages the allocation and release of memory for your application

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

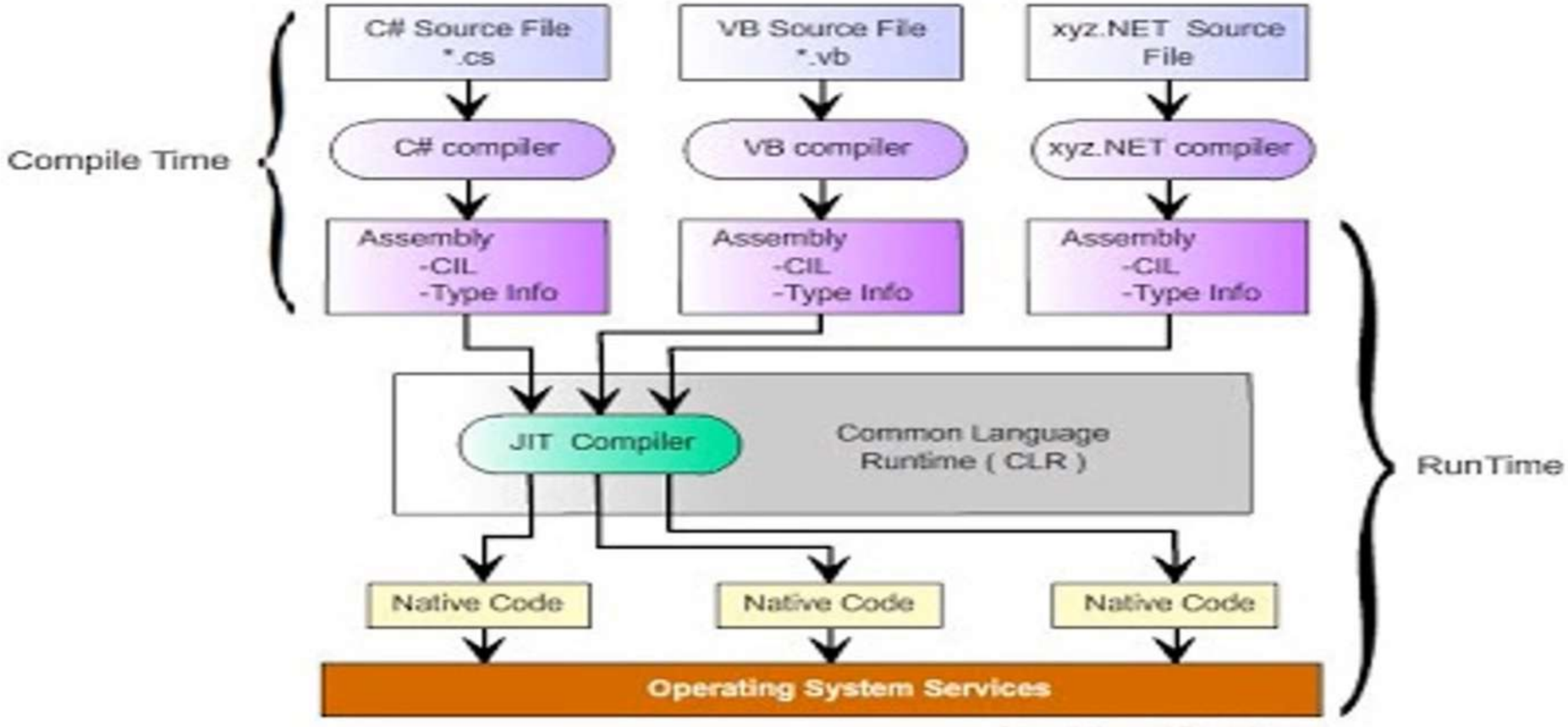


Wow! Even if I download code from an unknown origin it will run with protection.

Your code is monitored….

Just In Time Compiler(JIT)

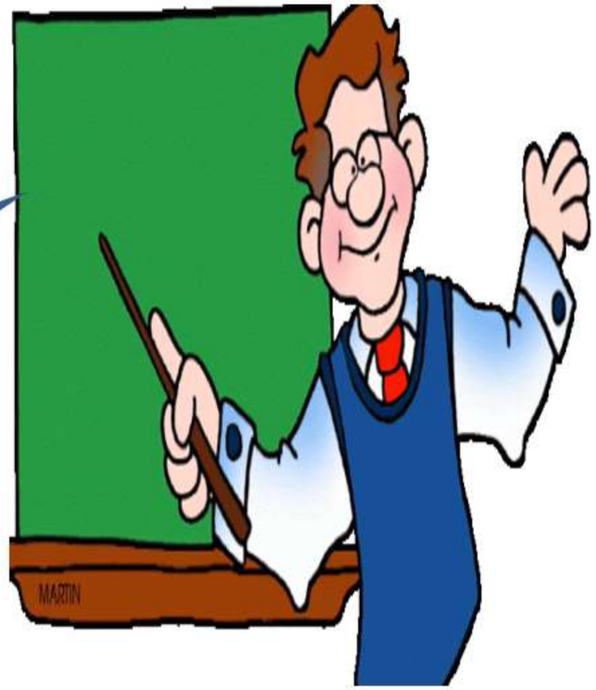
The Just In Time Compiler converts the IL code into native code .



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

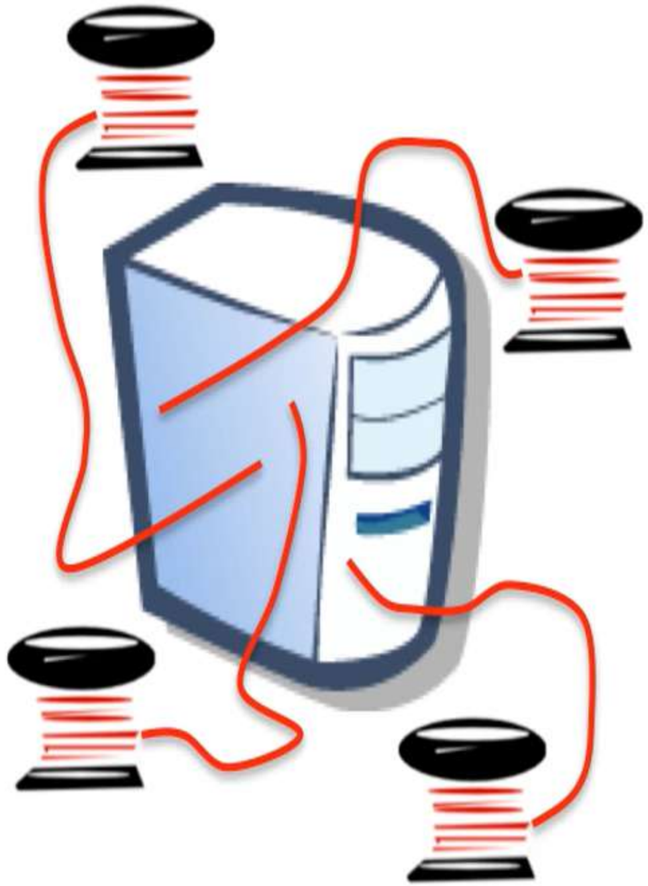
AS a part of JIT compilation the CLR verifies the meta data and the IL that they are type safe to be compiled into native code .

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

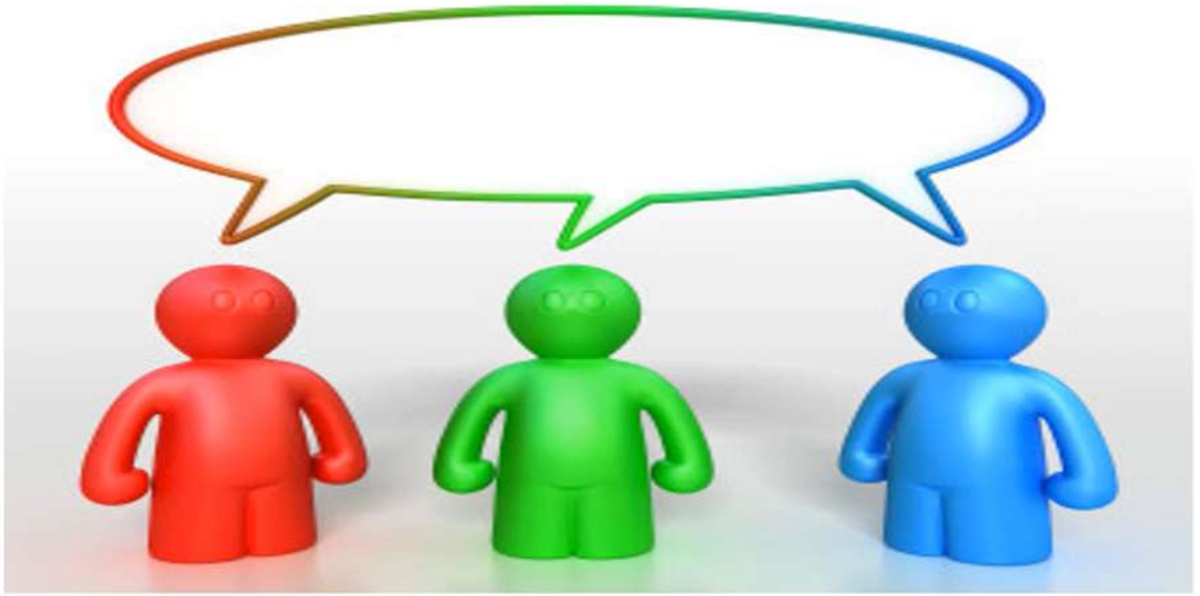
- Threading enables a program to perform concurrent processing so that more than one operation takes place at a time.
- For example, you can use threading to monitor input from the user, perform background tasks, and handle simultaneous streams of input.



Cross language Interoperability

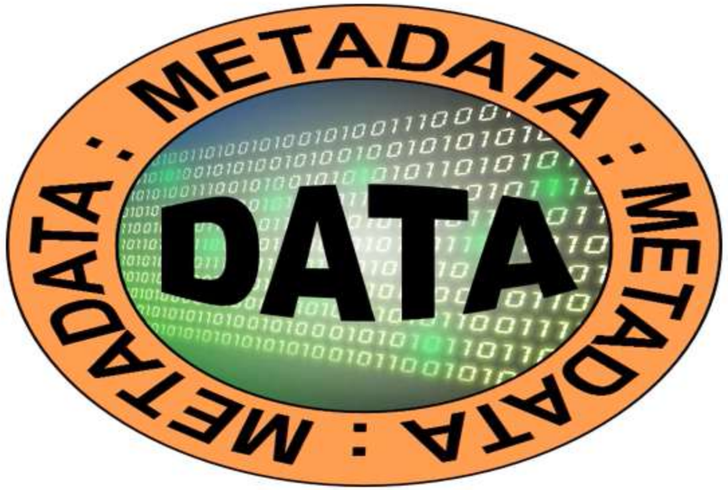
TEKNOTURF

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



Metadata

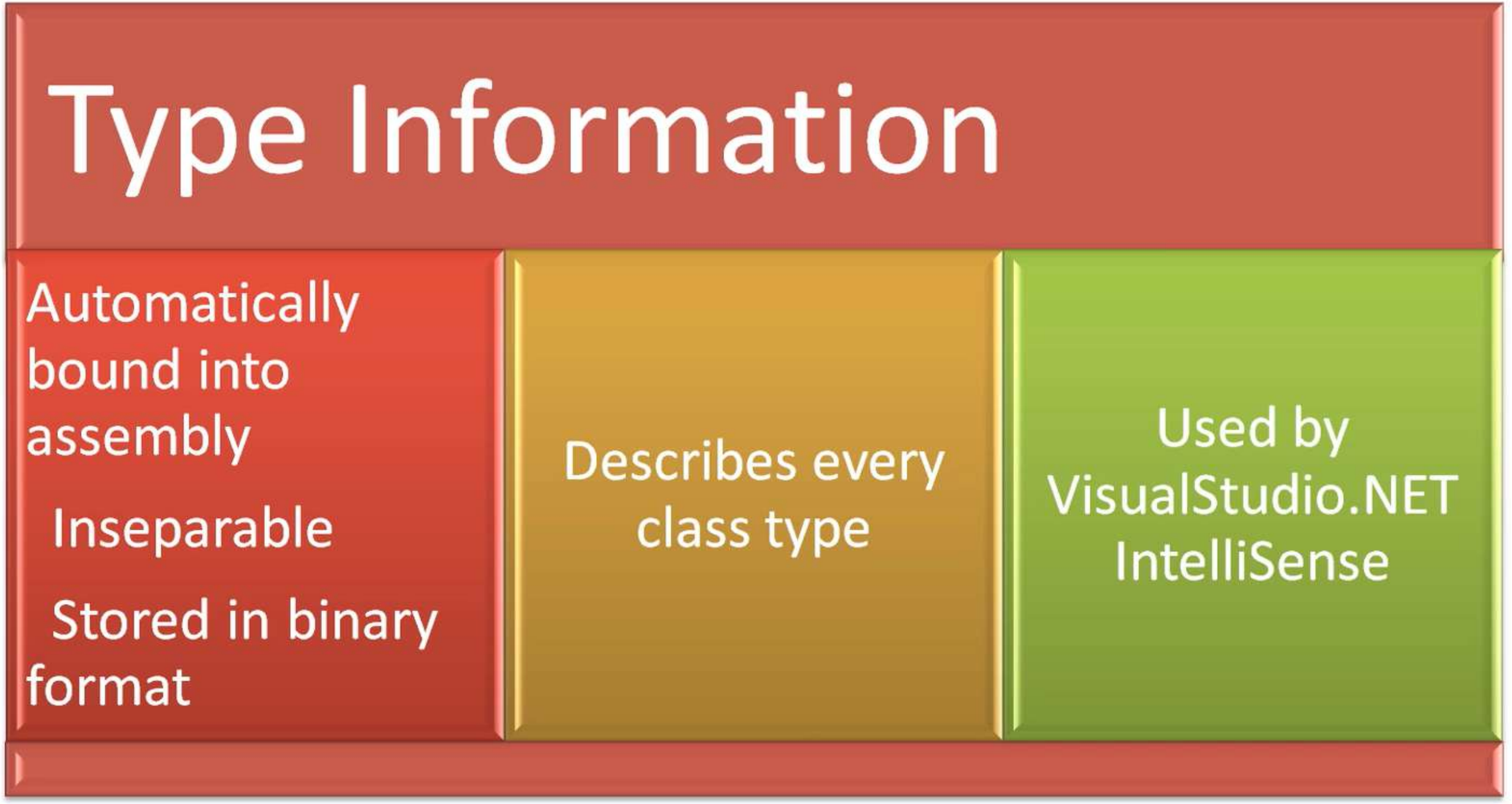
Metadata is binary information describing your program that is stored either in a common language runtime portable executable (PE) file or in memory.



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.

CLR - Metadata



Design Goals of CLR

TEKNOTURF

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.

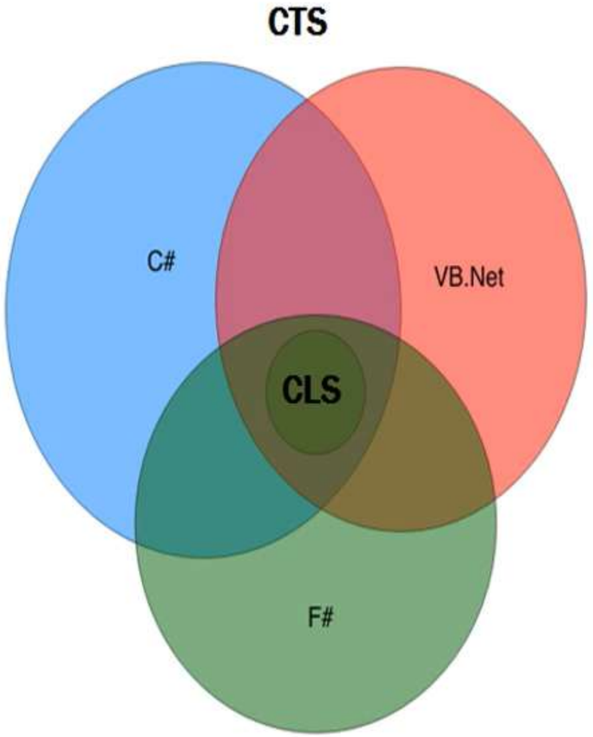
Common Type System

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

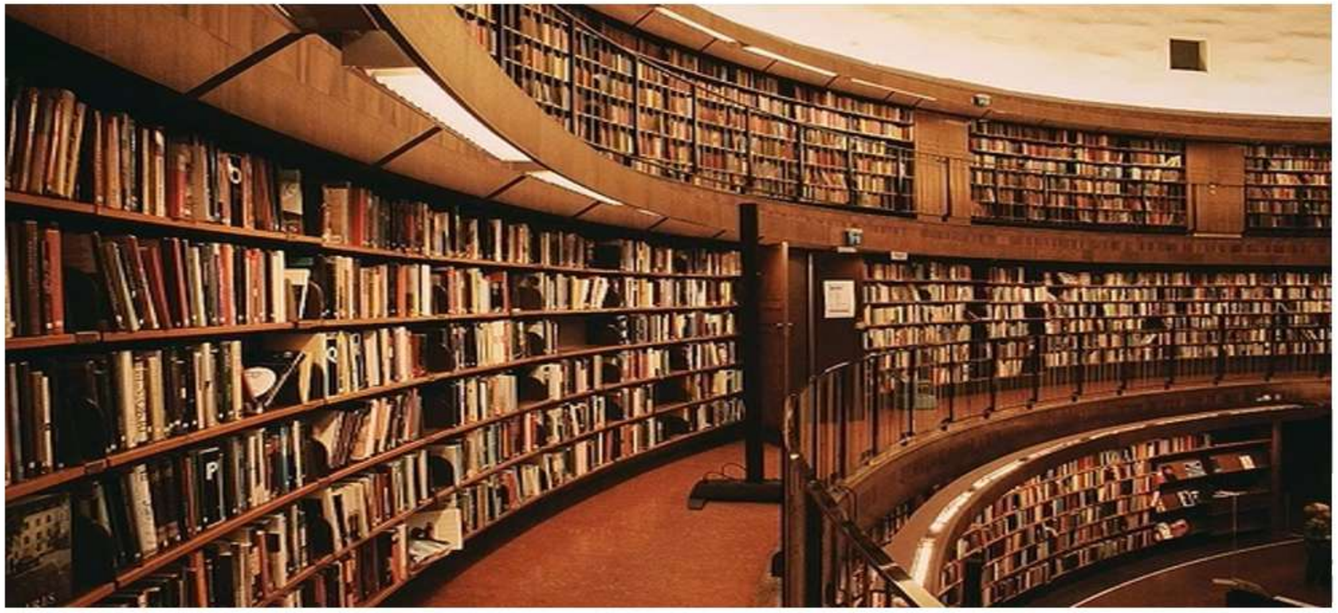
Common Language Specification

- 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

How can I find my material from this huge collection?

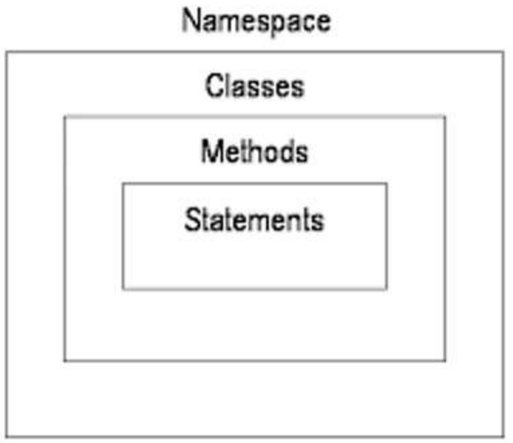
Namespace

The Framework Class library Namespaces

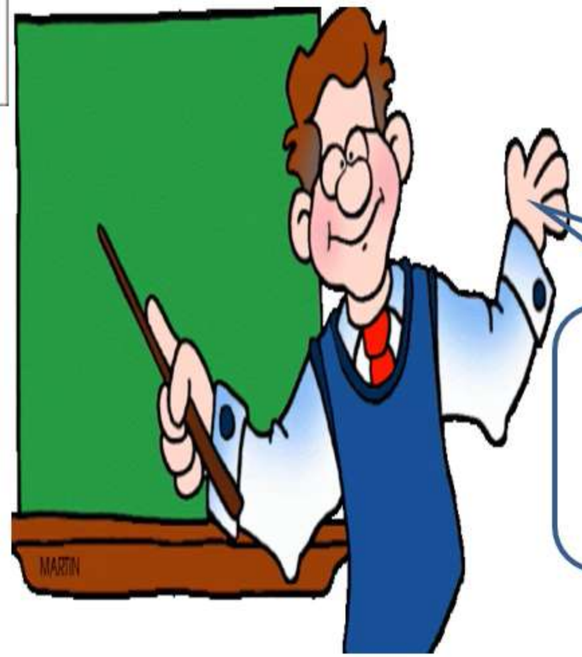
TEKNOTURF

- 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

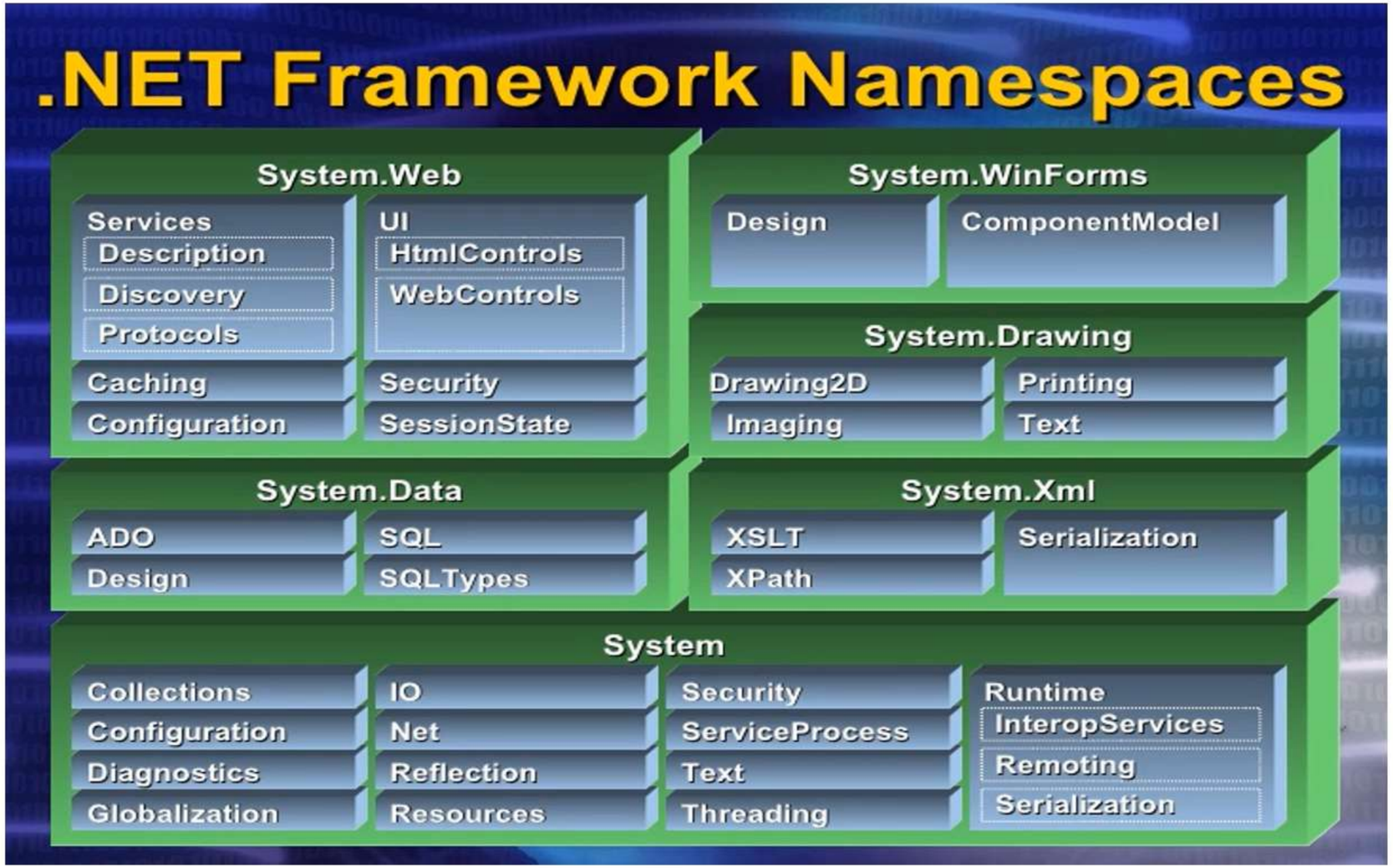


Related functionalities are stored under a single class or interface.
Related classes and interfaces are stored under namespace.



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.

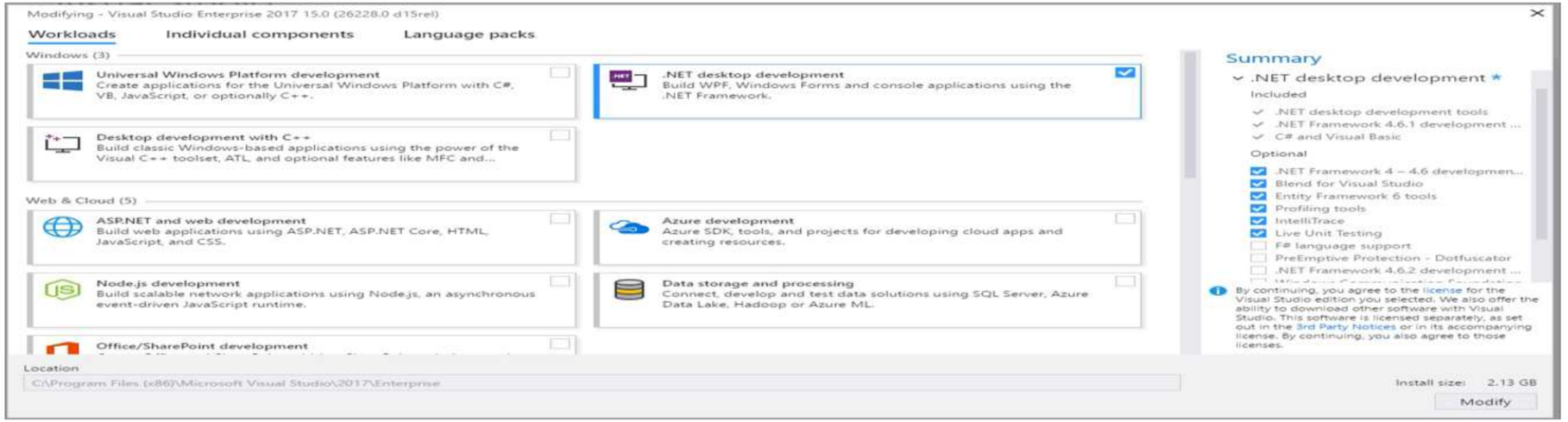
The following languages are incorporated with the Visual Studio IDE

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.

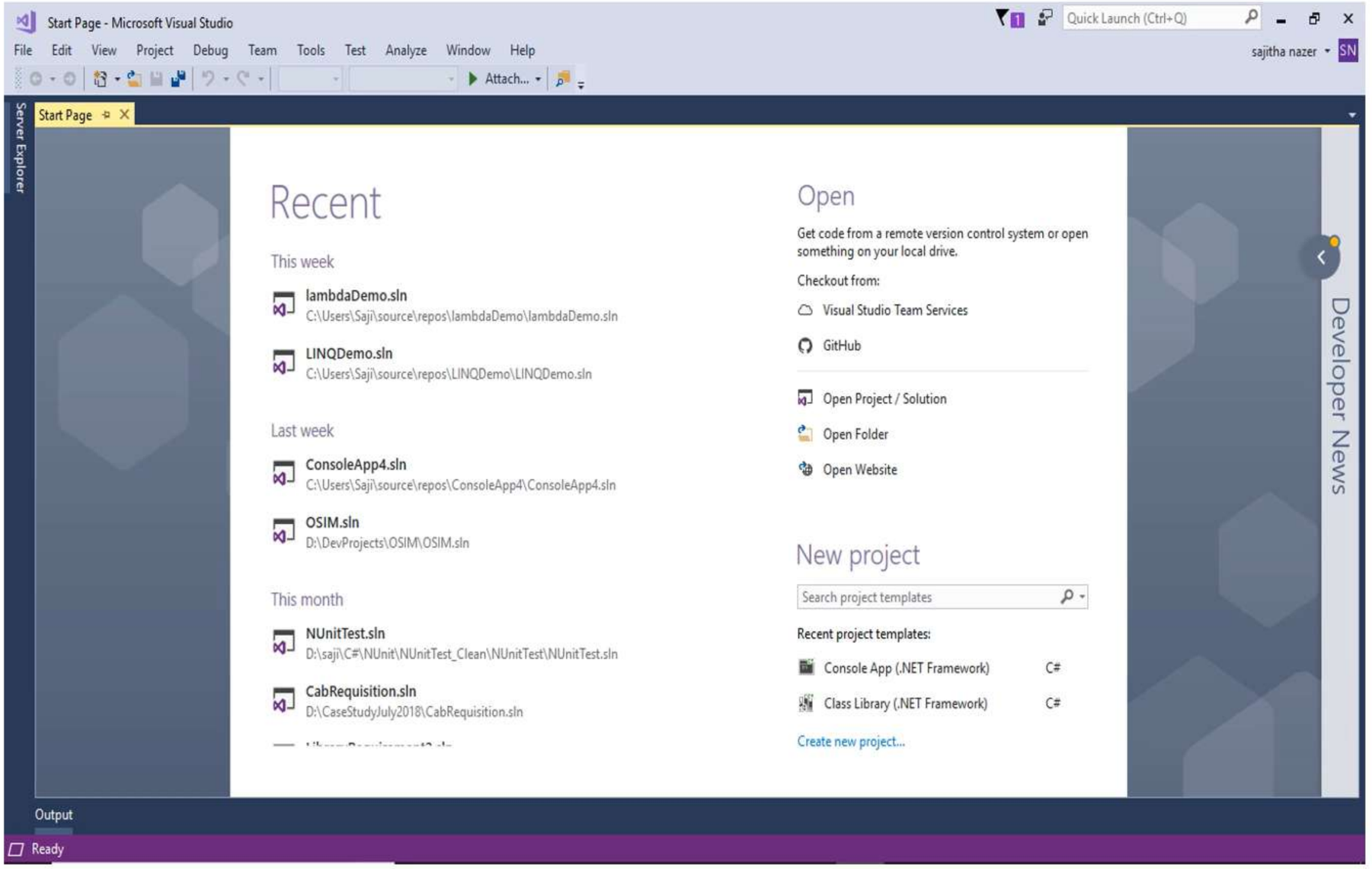


New Enhanced Features

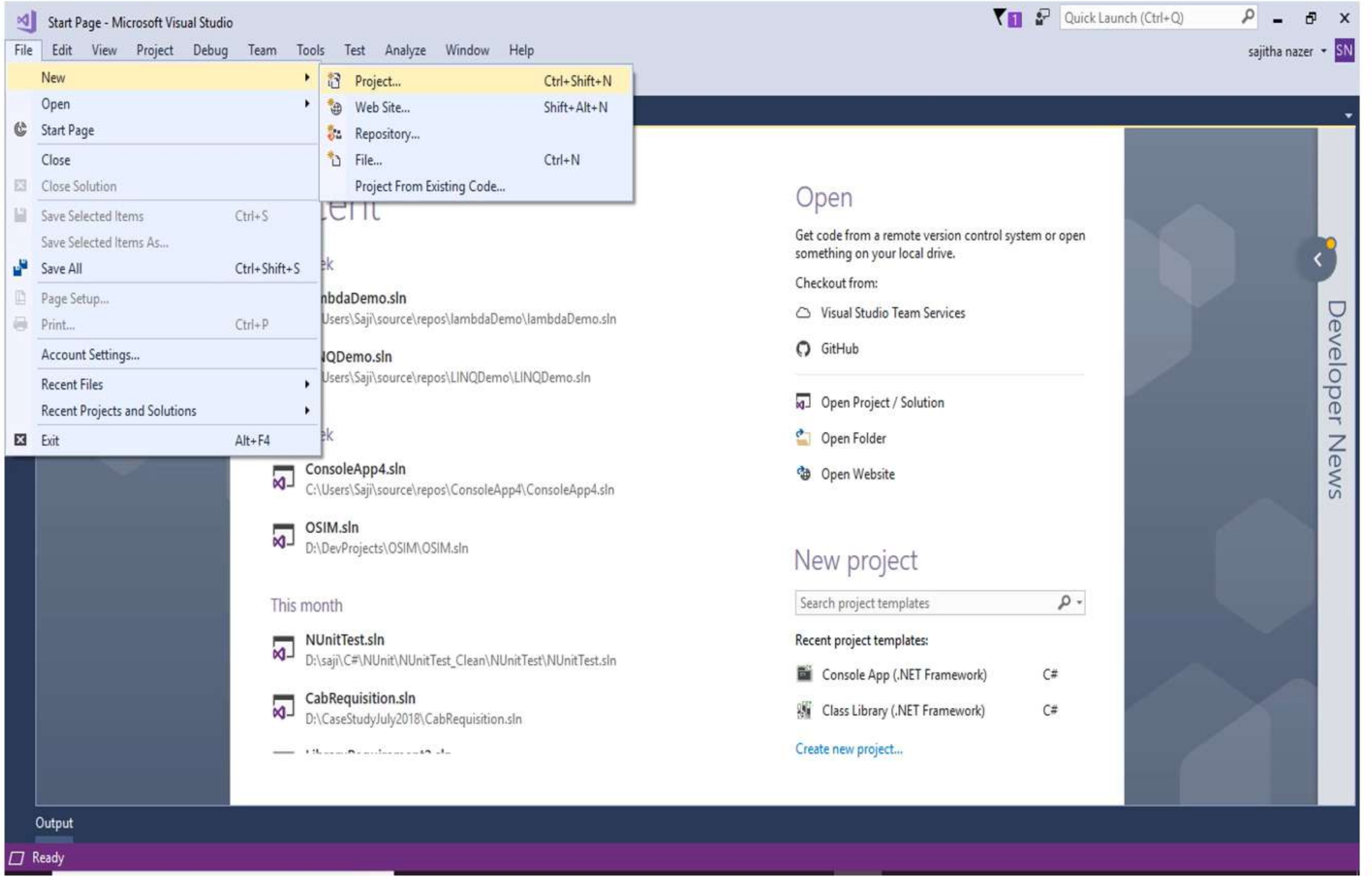
TEKNOTURF

- 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 is 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.

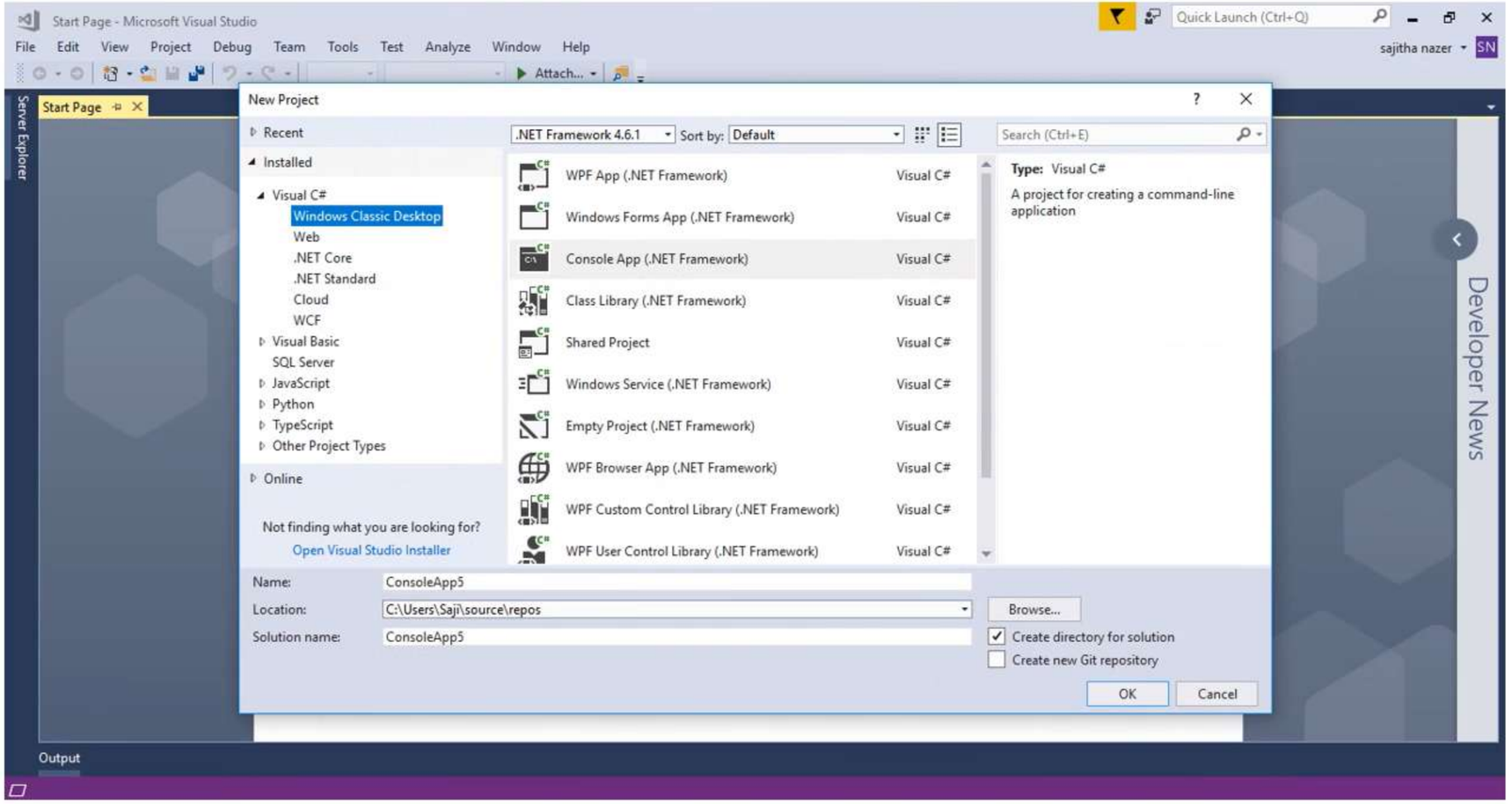
Start Page


Start Page - Microsoft Visual Studio
File Edit View Project Debug Team Tools Test Analyze Window Help
Quick Launch (Ctrl+Q)
sajitha nazer SN
Attach...
Server Explorer
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
Output
Ready

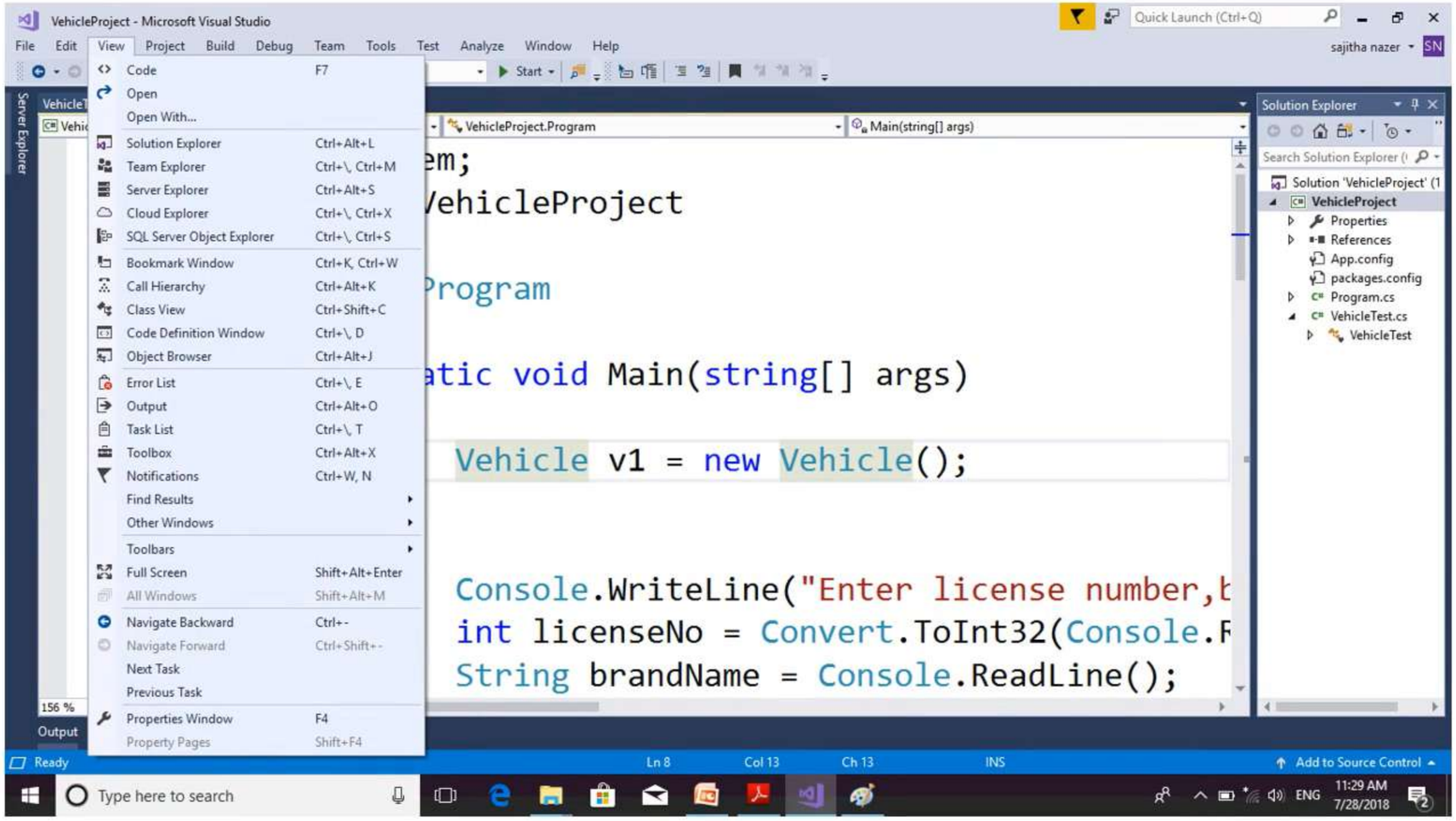
Menu


Start Page - Microsoft Visual Studio
File Edit View Project Debug Team Tools Test Analyze Window Help
Quick Launch (Ctrl+Q)
sajitha nazer SN
New
Open
Start Page
Close
Close Solution
Save Selected Items 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
Project... Ctrl+Shift+N
Web Site... Shift+Alt+N
Repository...
File... Ctrl+N
Project From Existing Code...
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
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
Output
Ready

Project dialog box



Project dialog box

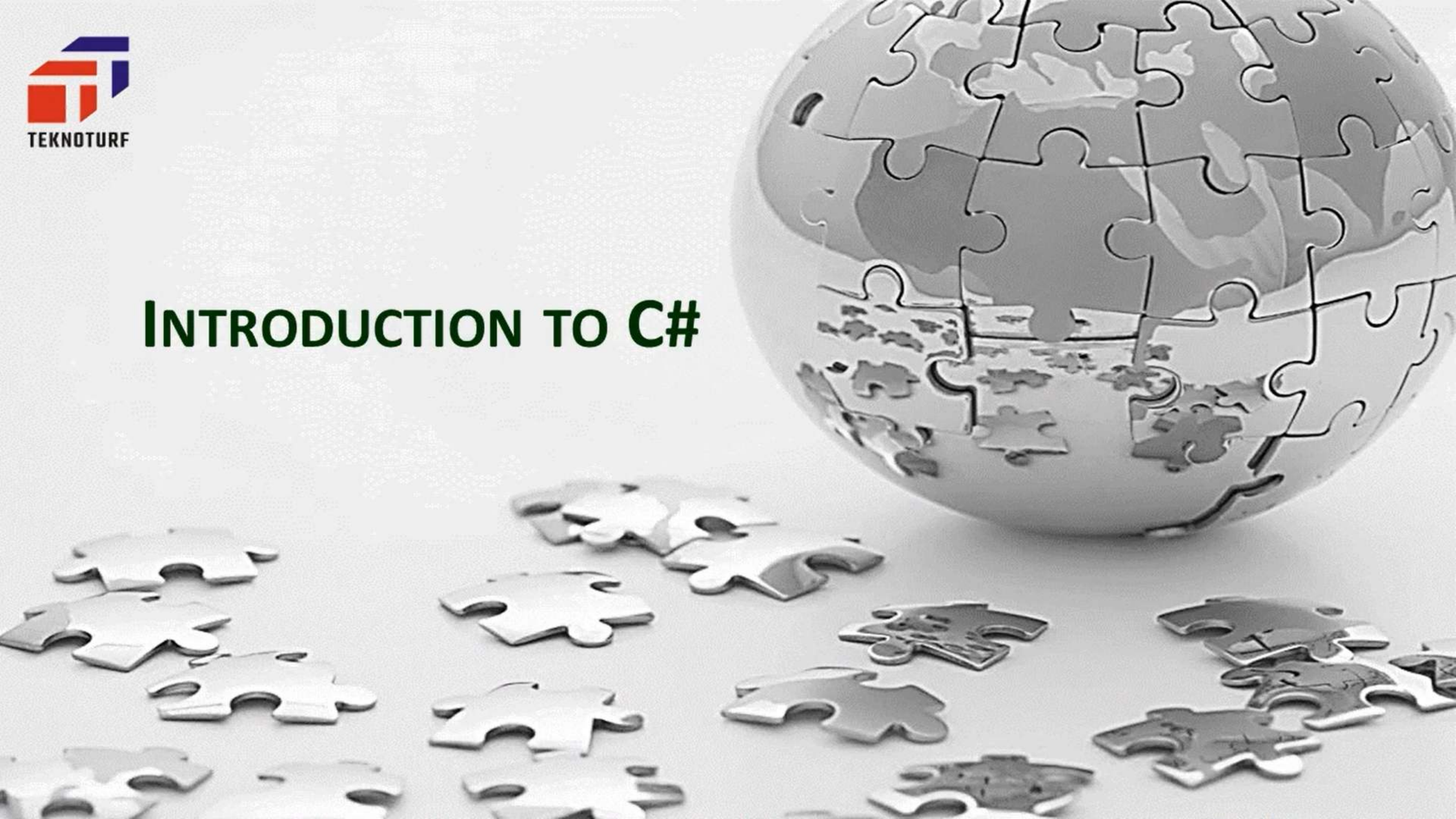


Summary

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

TEKNOTURF

INTRODUCTION TO C#



Objectives

TEKNOTURF

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#

TEKNOTURF

- 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#

TEKNOTURF

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

TEKNOTURF

- 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.

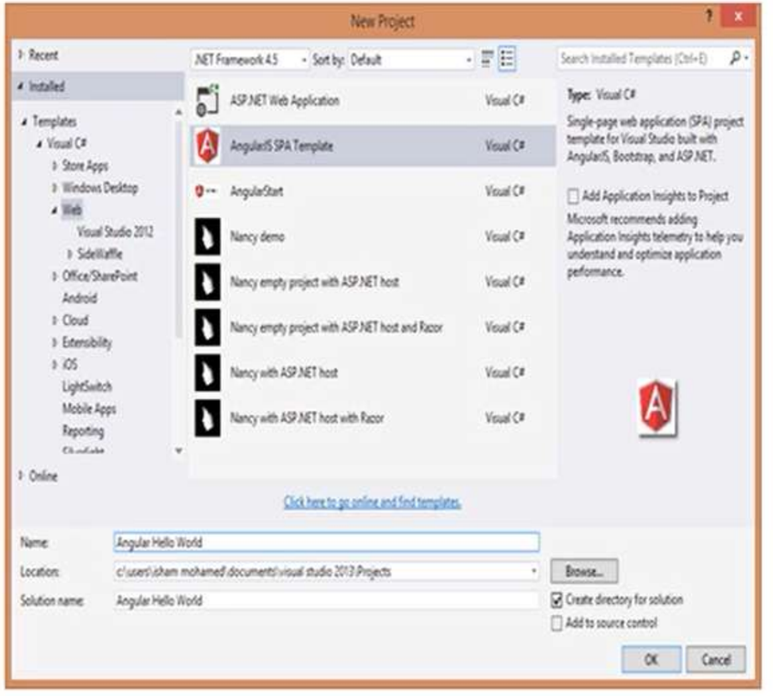
Visual Studio

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

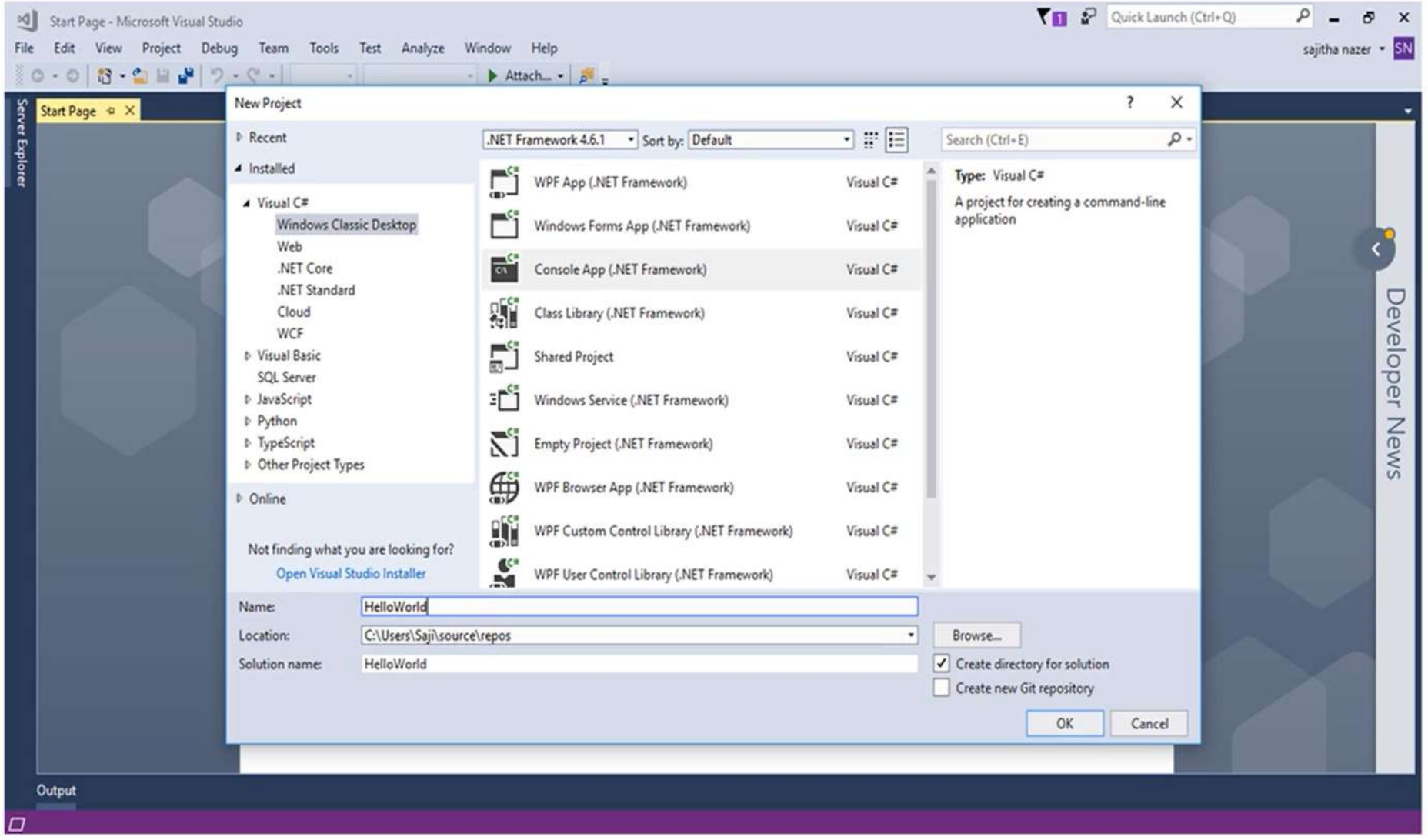
TEKNOTURF

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

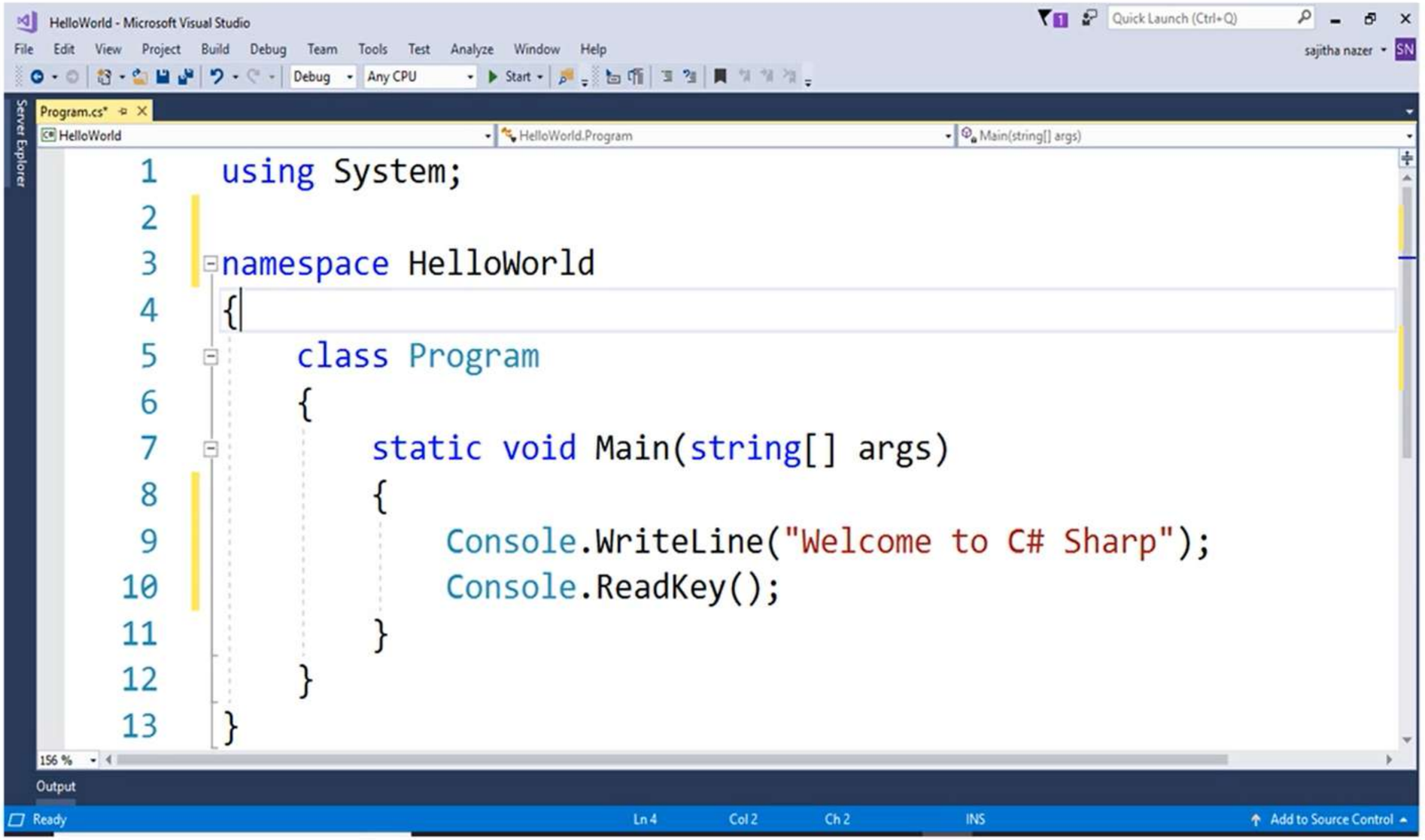
To create and run a Console Application

- 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.



Using Visual Studio 2017 contd..

TEKNOTURF

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

TEKNOTURF

Type the HelloWorld.cs Program in a notepad.

Save the file as HelloWorld.cs (optional)

Type the following command to invoke C# compiler.

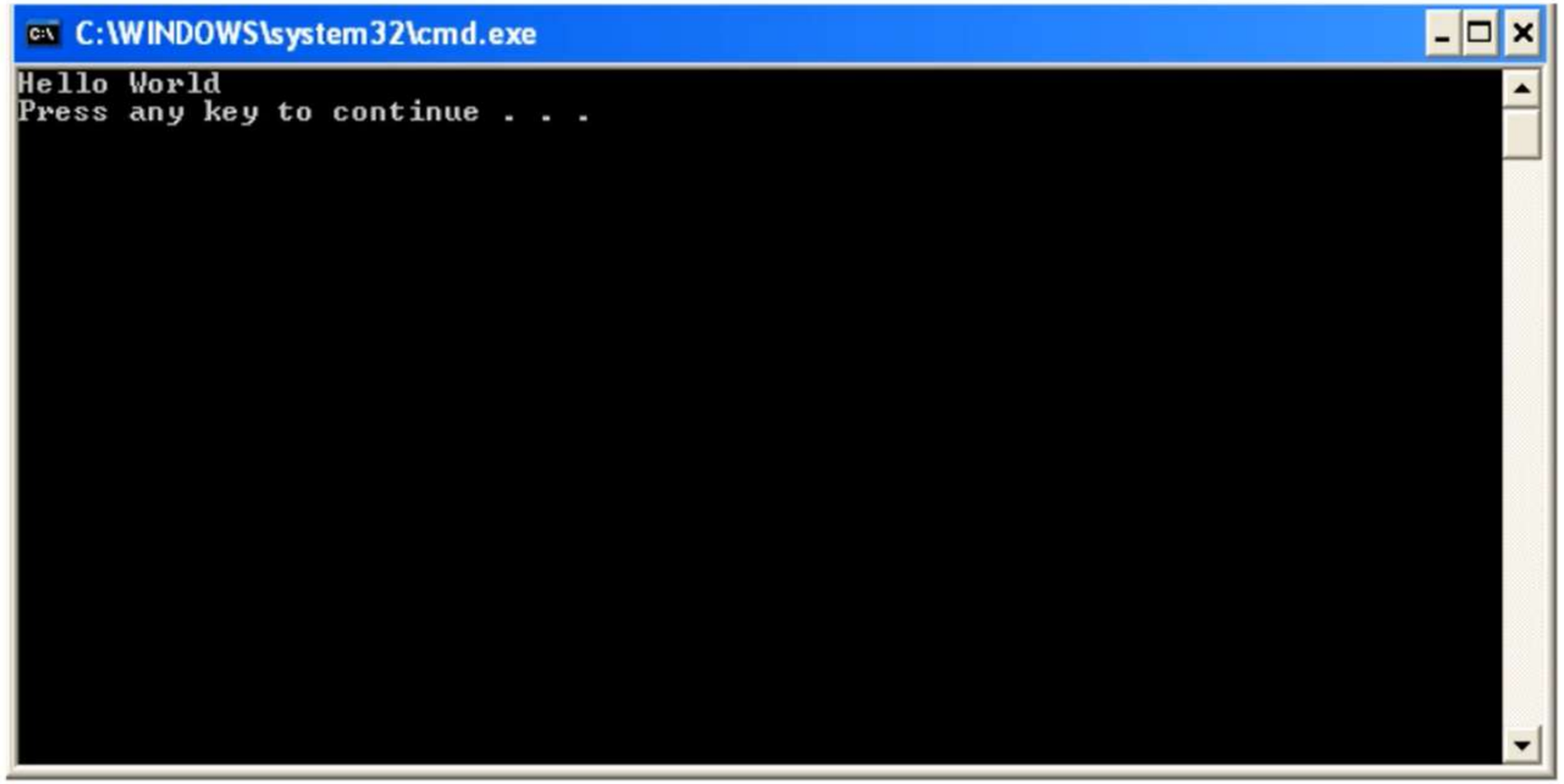
- >csc HelloWorld.cs

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

- >HelloWorld

Command Prompt Contd..

TEKNOTURF

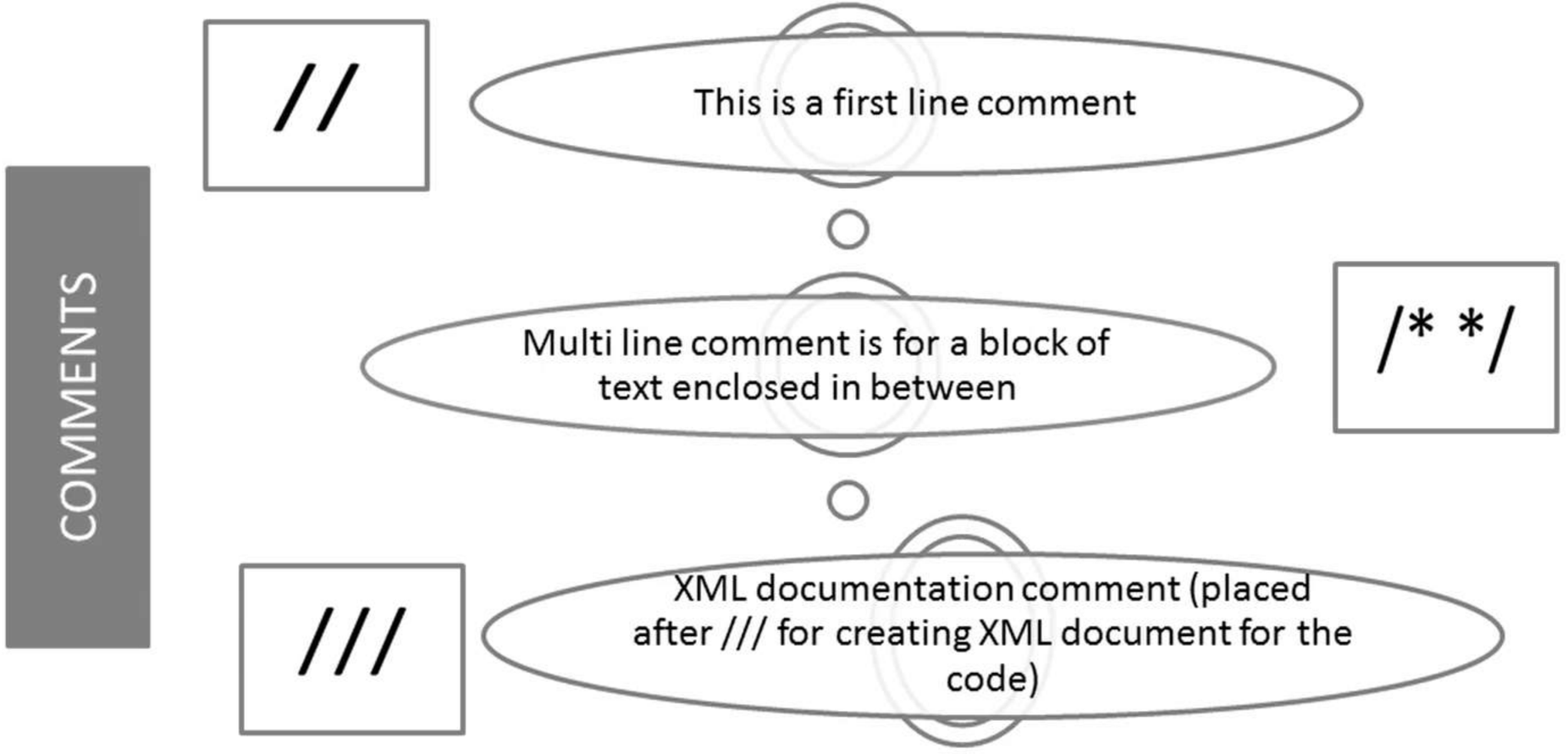


C# Compiler (csc) Options

TEKNOTURF

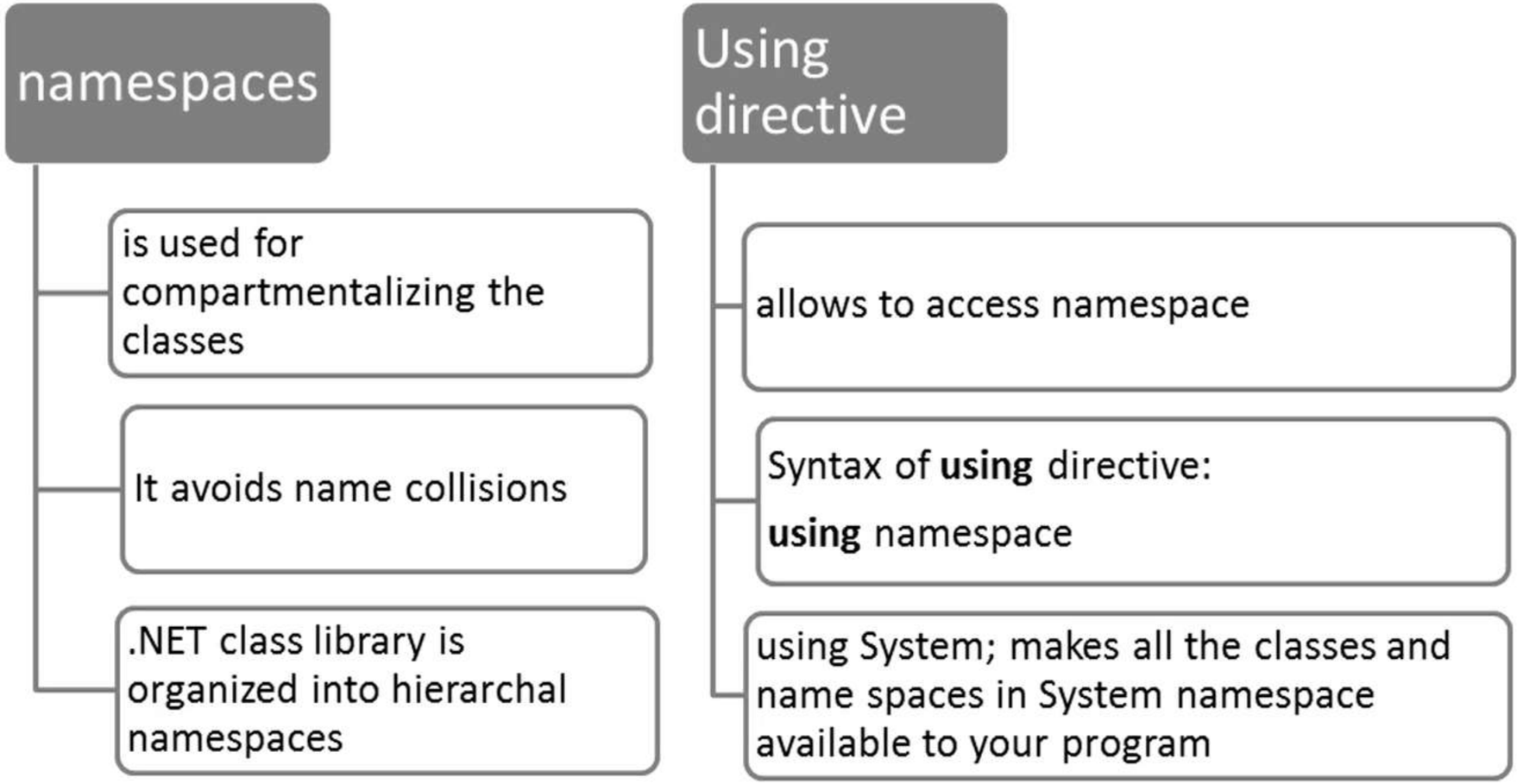
Switch	Description
/target(/t) /t:exe /t:library /t:module /t:winexe	Specifies the type of file output An .exe file (Console application) A DLL file with manifest A DLL file without manifest A windows application
/unsafe	Allows code containing unsafe keyword to compile

Program Structure



Program Structure

TEKNOTURF



Program Structure contd..

TEKNOTURF

- 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()

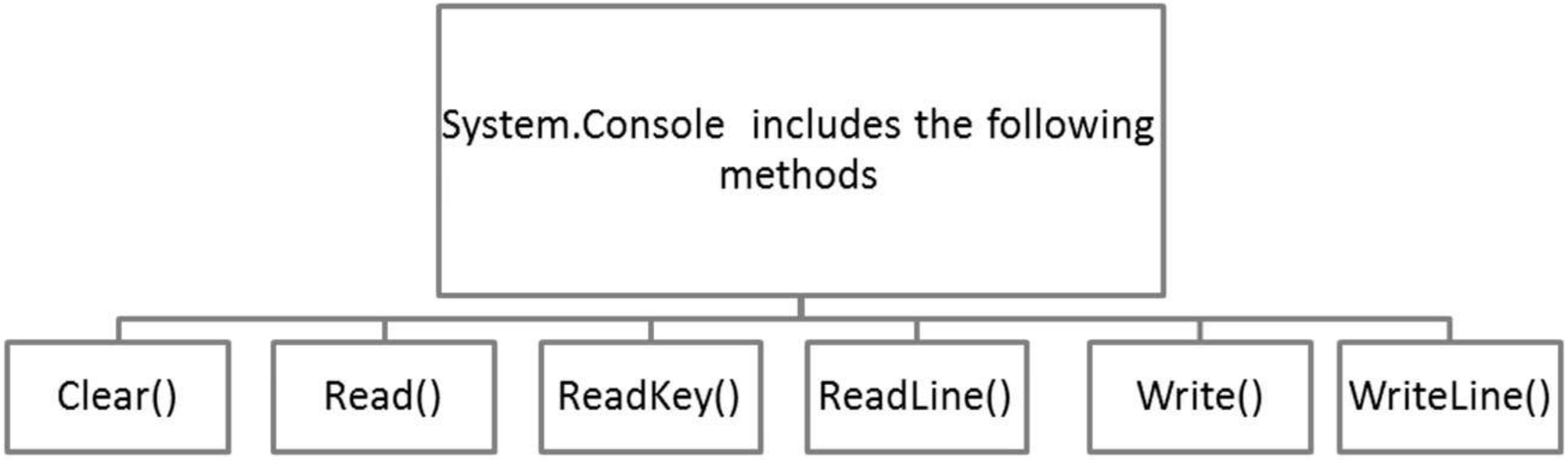
TEKNOTURF

- 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

TEKNOTURF



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

TEKNOTURF

```
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

TEKNOTURF

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

TEKNOTURF

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

TEKNOTURF

```
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

TEKNOTURF

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

TEKNOTURF

C# FUNDAMENTALS

Objectives

TEKNOTURF

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#

TEKNOTURF

C# is a strongly typed language

Every variable and constant has a type

Data types includes:

- int
- long
- float
- double
- decimal
- char
- bool
- DateTime
- string

Common Type System (CTS)

TEKNOTURF

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

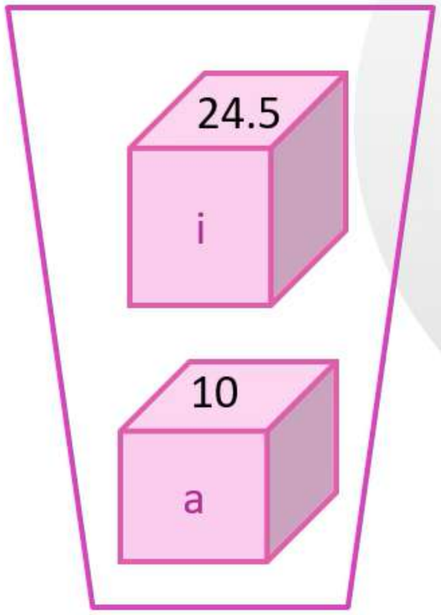
- It defines all the types and operations found in most programming languages

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



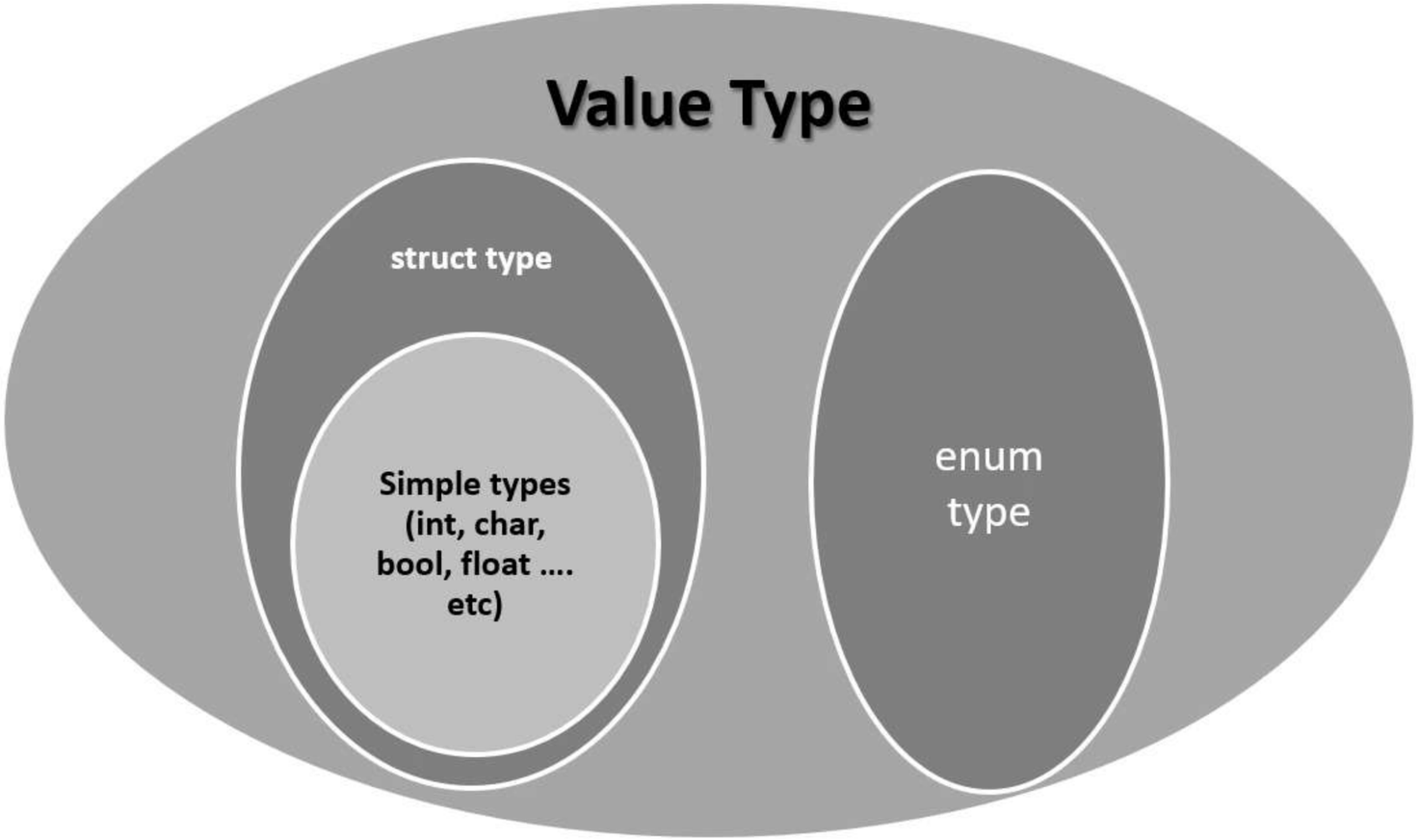
Value Types

directly contain their values

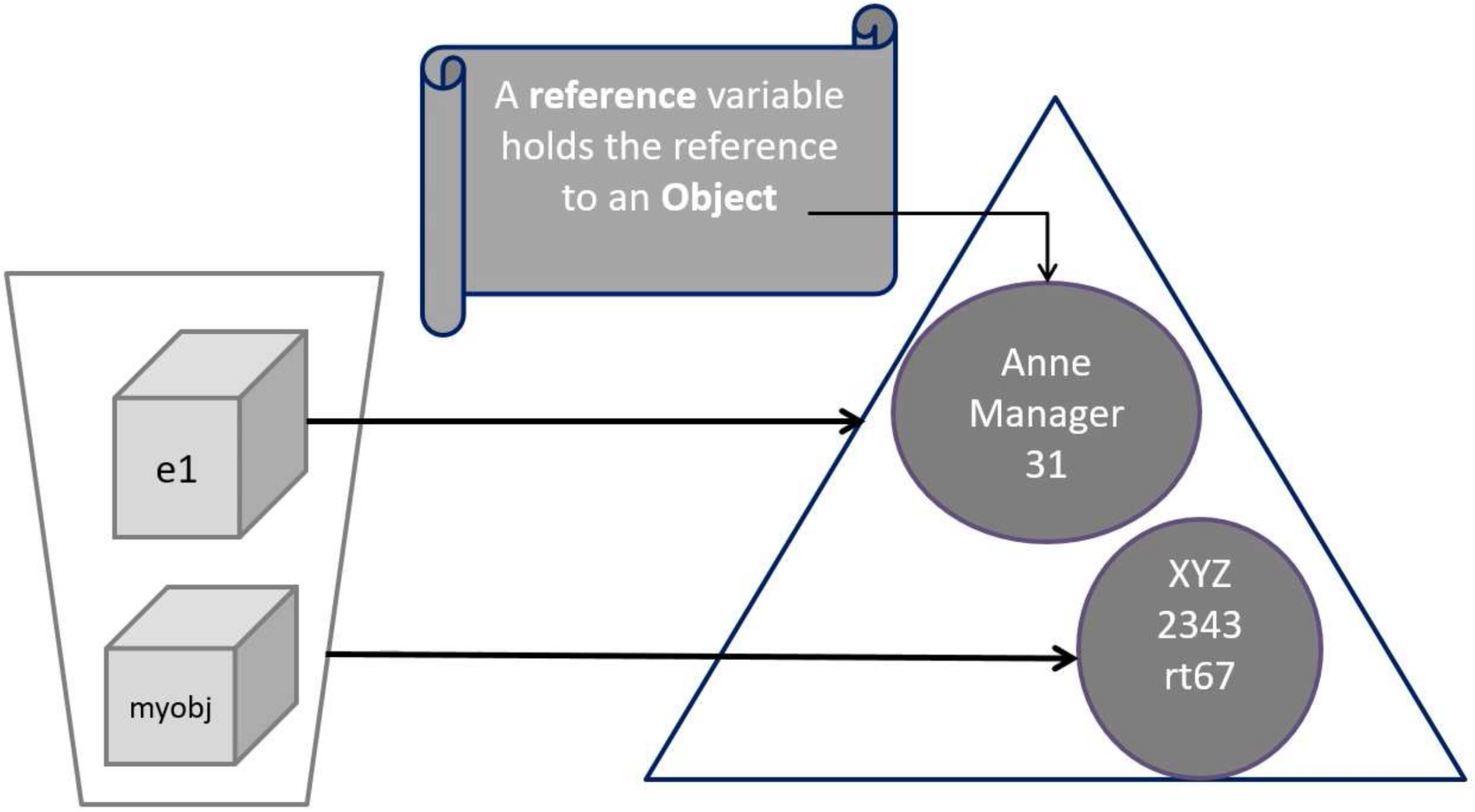
The memory is allocated inline when the variable is declared

Value is stored in the stack

Value Type


Value Type
struct type
Simple types
(int, char,
bool, float ....
etc)
enum
type

Reference Types



Reference Types

TEKNOTURF

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

TEKNOTURF

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

Floating Point Types and Decimal Types

TEKNOTURF

Name	CTS Type	Size (bytes)	Description
float	System.Single	4	Ranges from ±1.5 × 10-45 to ±3.4 × 1038 with 7 digits precision. Requires the 'f' or 'F'
double(default)	System.Double	8	Ranges from ±5.0 × 10-324 to ±1.7× 10308 with 15-16 digits precision.

Name	CTS Type	Size (bytes)	Description
decimal	System.Decimal	16	Ranges from ±1.0 × 10-28 to ±7.9× 1028 with 28-29 digits precision. Requires the 'm' or 'M'

Boolean Type and Reference Type

TEKNOTURF

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

TEKNOTURF

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

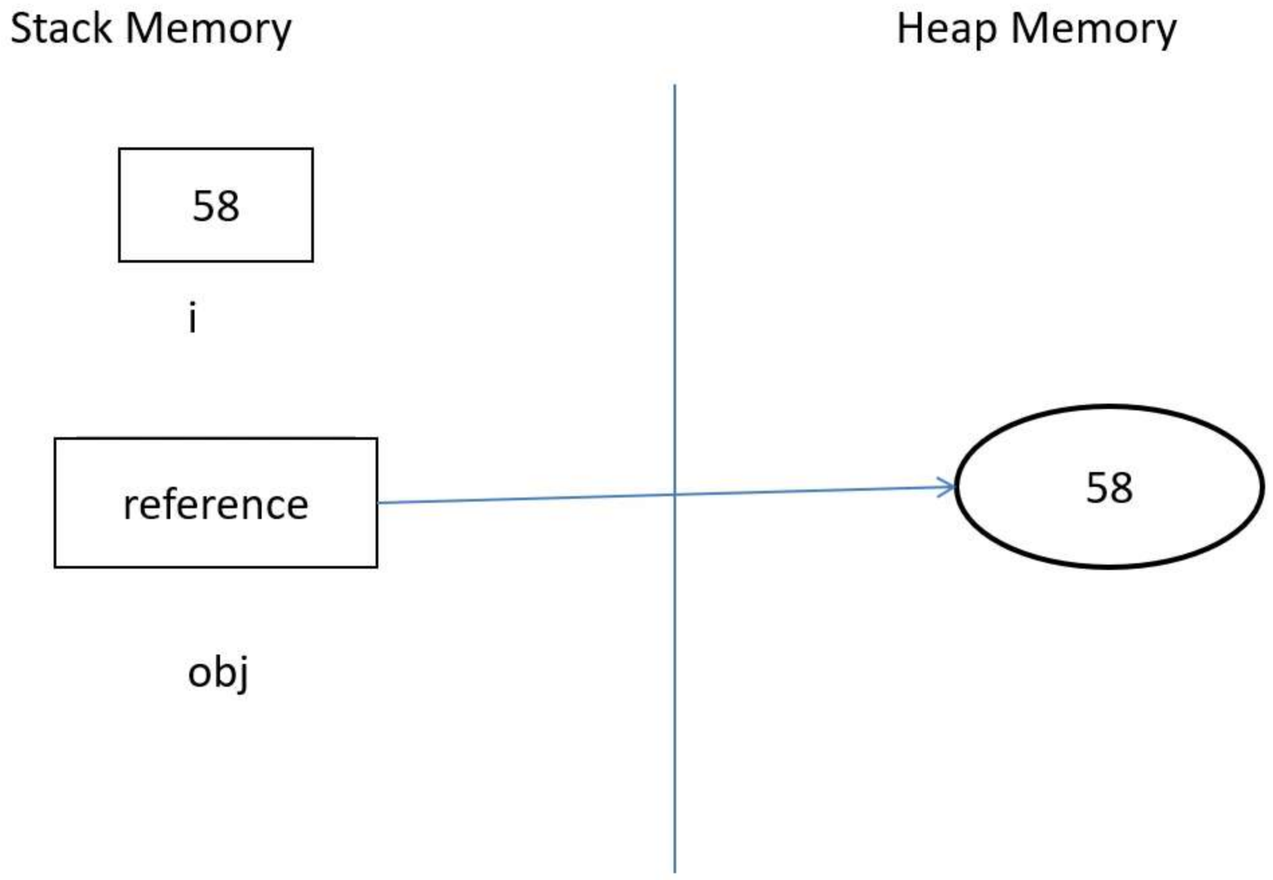
Converts a base data type to another base data type.

The Convert class contains a number of methods like ToInt32(), ToByte(), ToDecimal(), ToDataTime() ..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

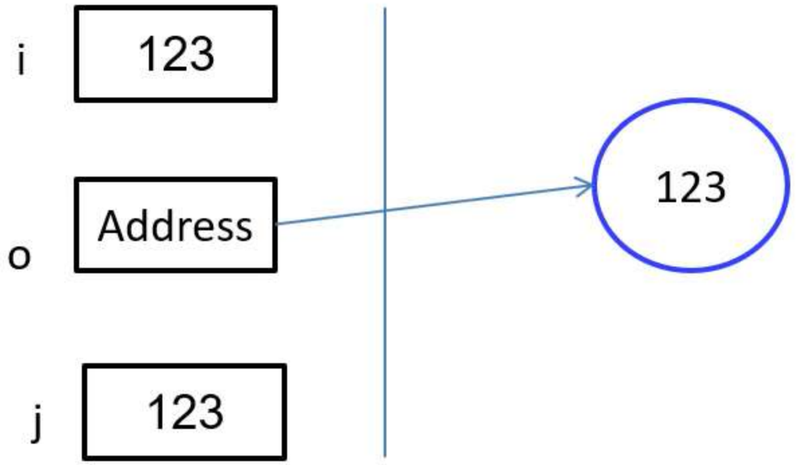
Unboxing is an explicit conversion from the type object to a value type or from an interface type to a value type that implements the interface

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

TEKNOTURF

// 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

TEKNOTURF

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

TEKNOTURF

Syntax:

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

Example:

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

Operators

TEKNOTURF

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

TEKNOTURF

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

TEKNOTURF

- 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

TEKNOTURF

- 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

TEKNOTURF

- 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

TEKNOTURF

- 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

TEKNOTURF

- 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

TEKNOTURF

- 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 **if statement** consists of a boolean expression followed by one or more statements.
2 if..else statement	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.
3 nested if	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).
4 switch	A **switch** statement allows a variable to be tested for equality against a list of values.
5 nested switch	You can use one **switch** statement inside another **switch** statement(s).

Looping

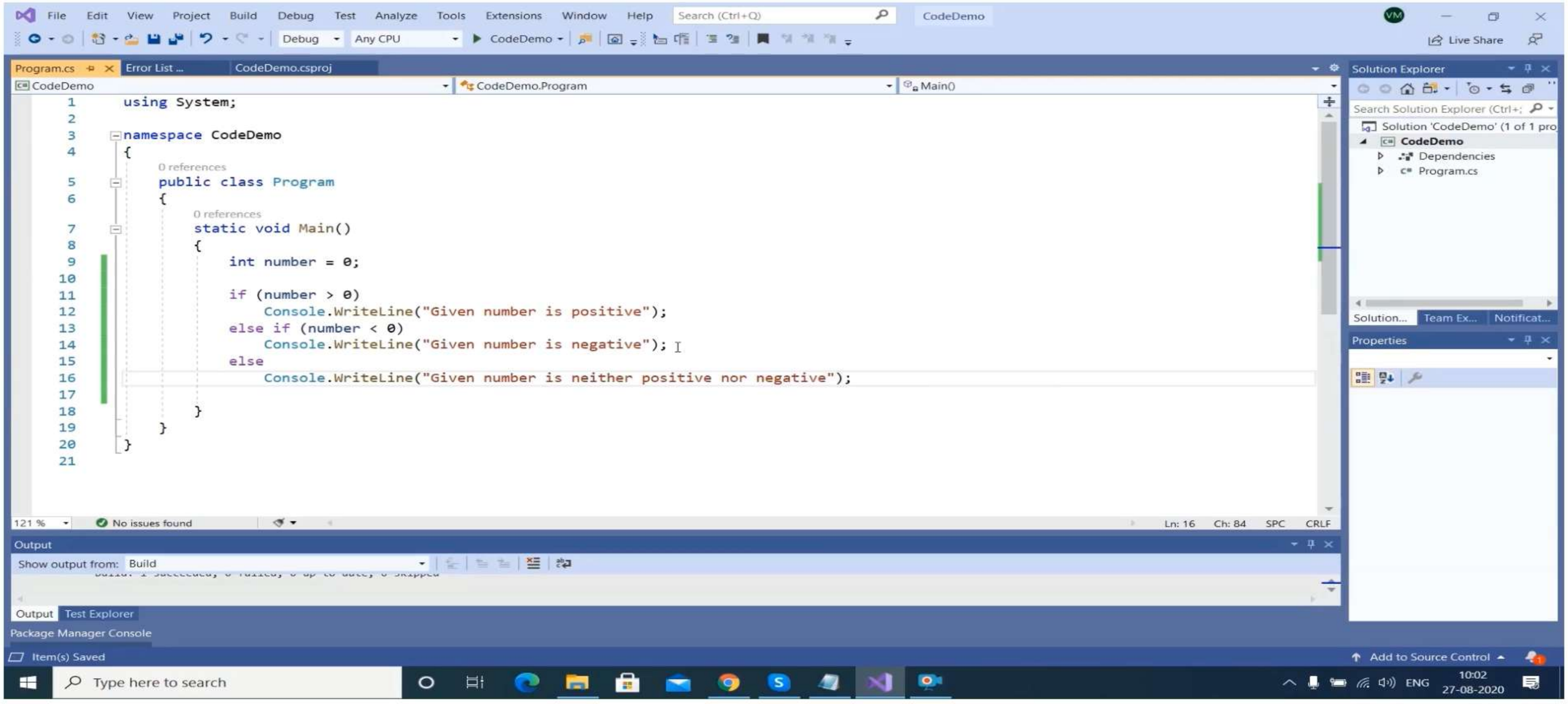
- A loop statement allows us to execute a statement or a group of statements multiple times and following is the general from of a loop statement in most of the programming languages.
- C# provides following types of loop to handle looping requirements.

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

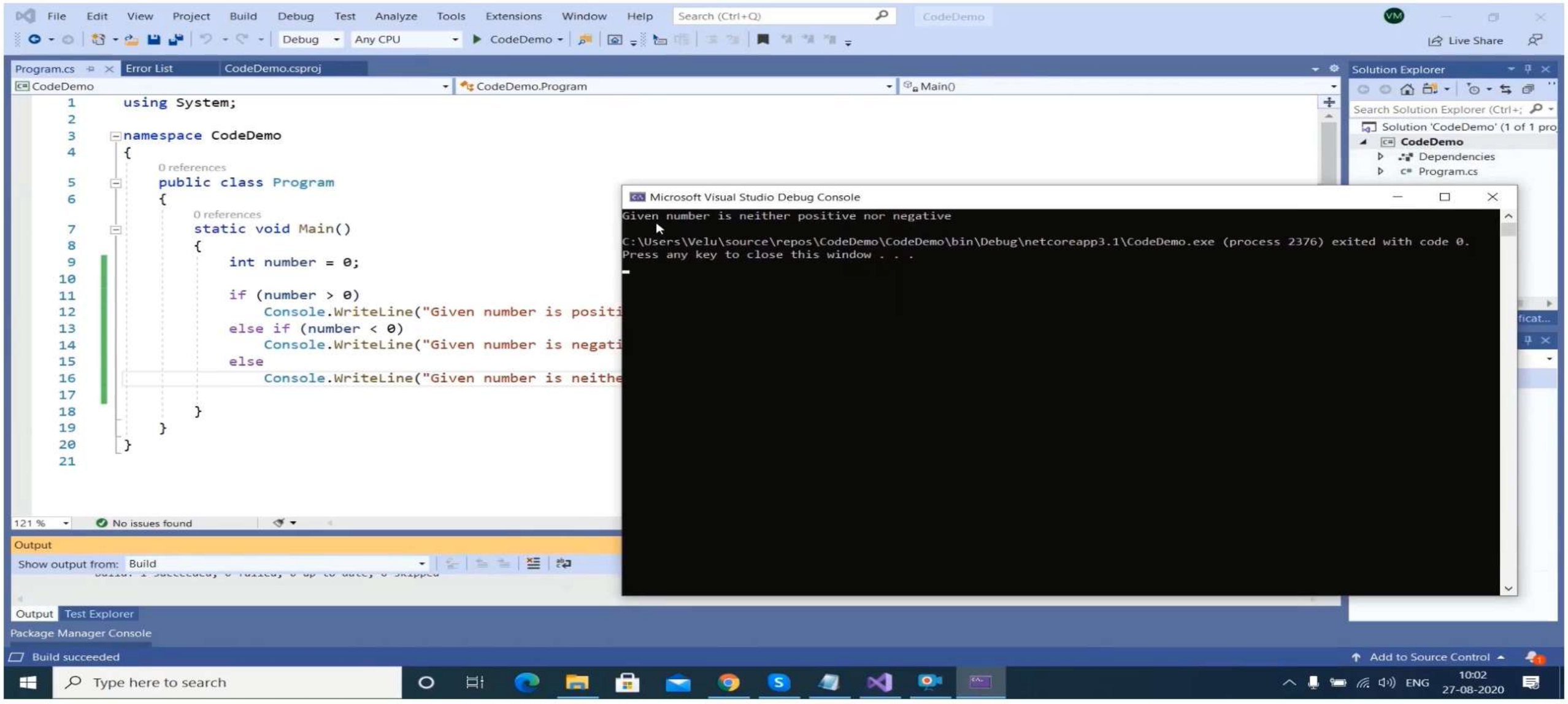
If-else Demo

TEKNOTURF



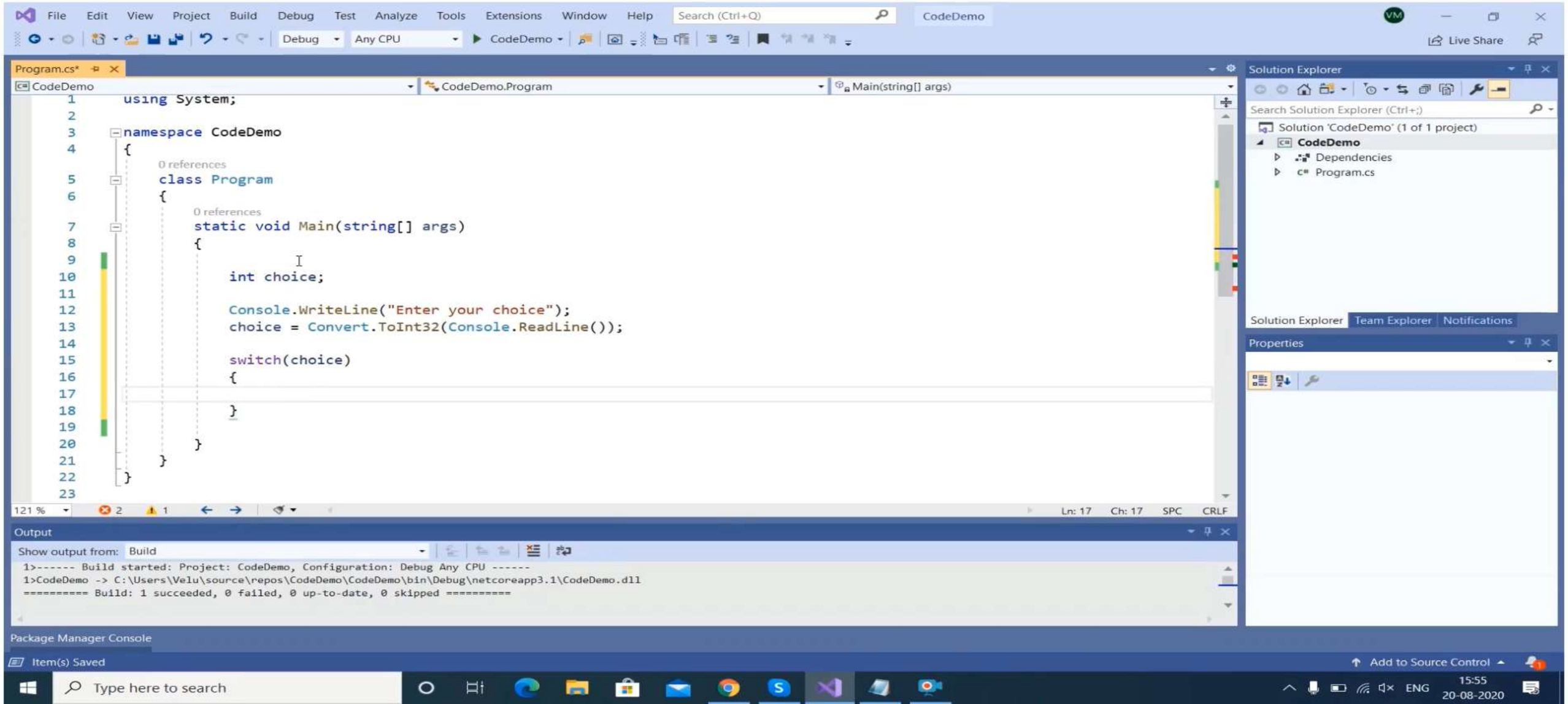
If-else Demo

TEKNOTURF



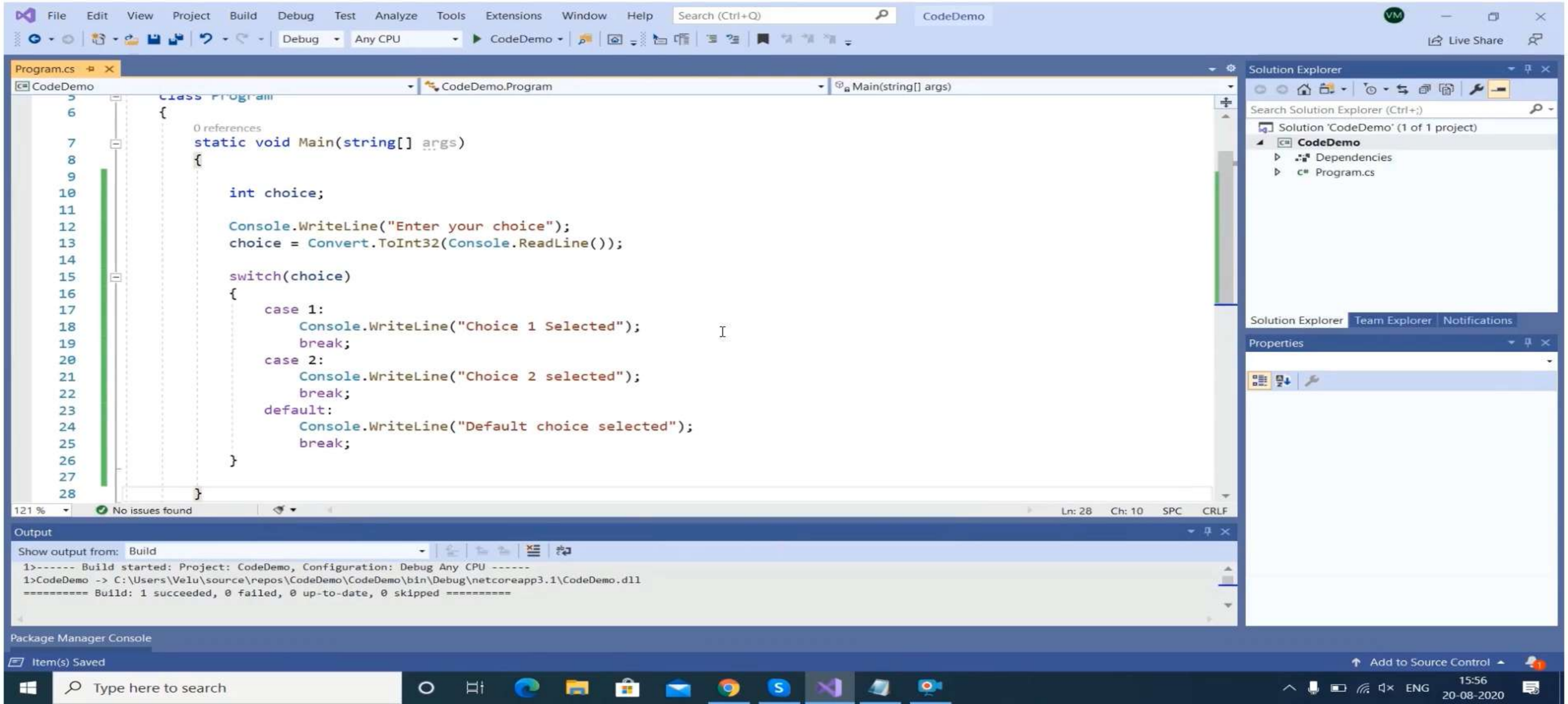
Switch Demo

TEKNOTURF



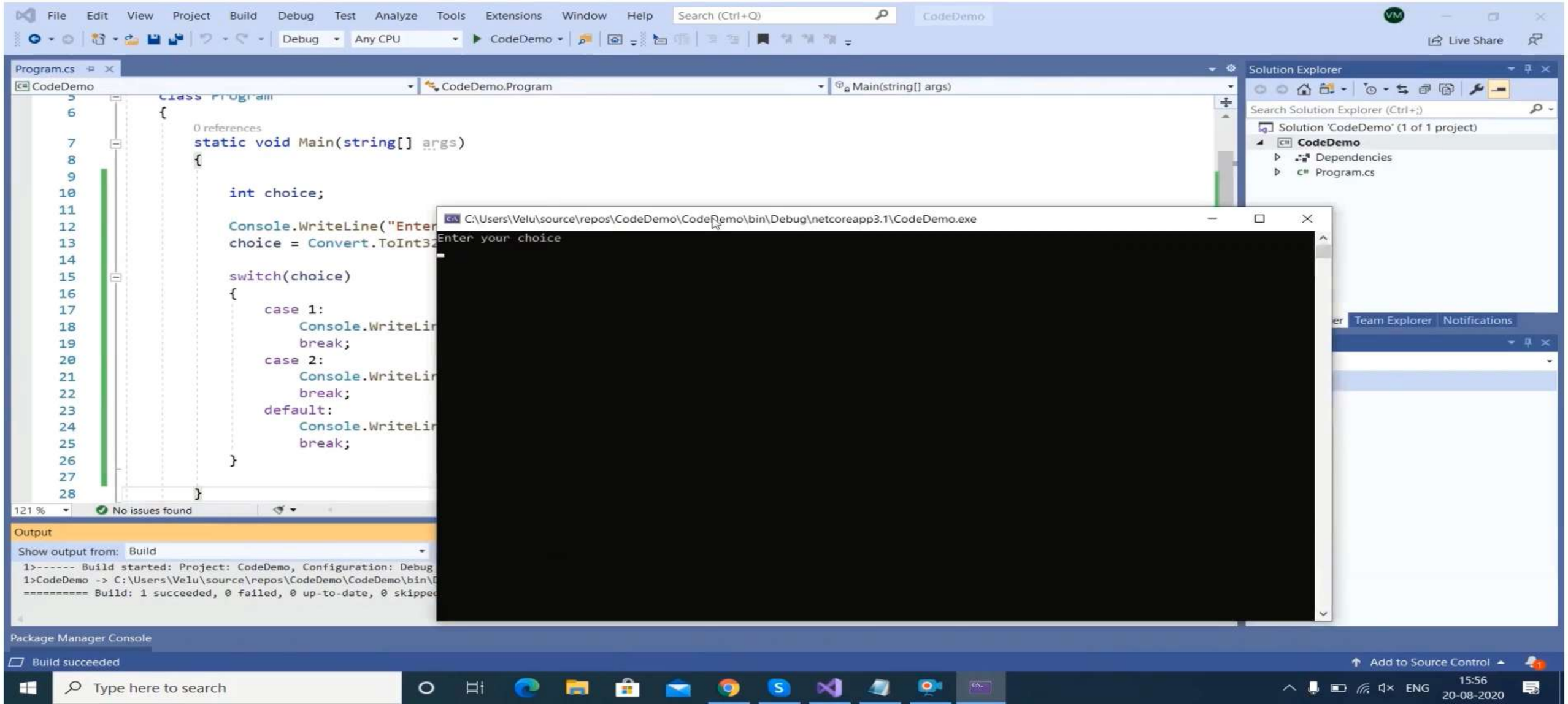
Switch Demo

TEKNOTURF



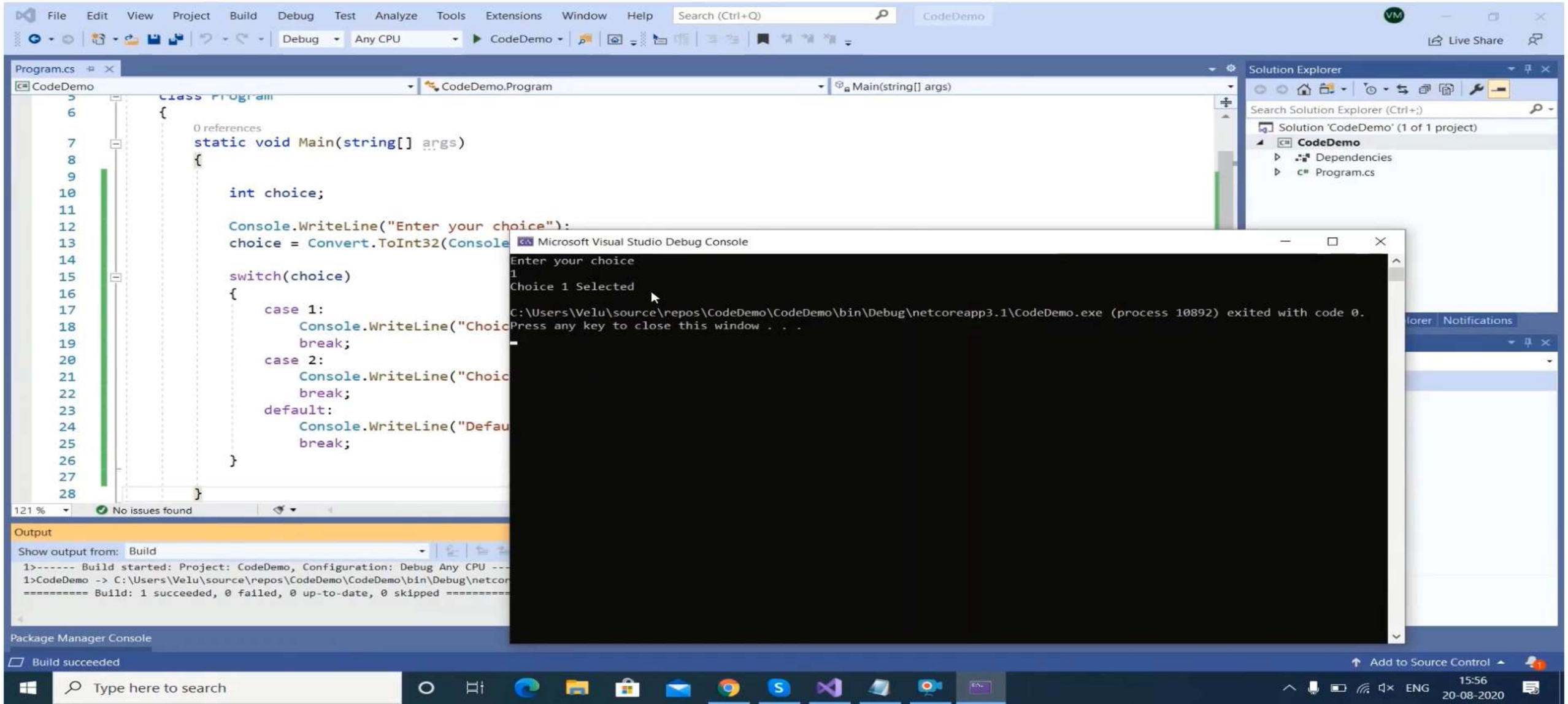
Switch Demo

TEKNOTURF



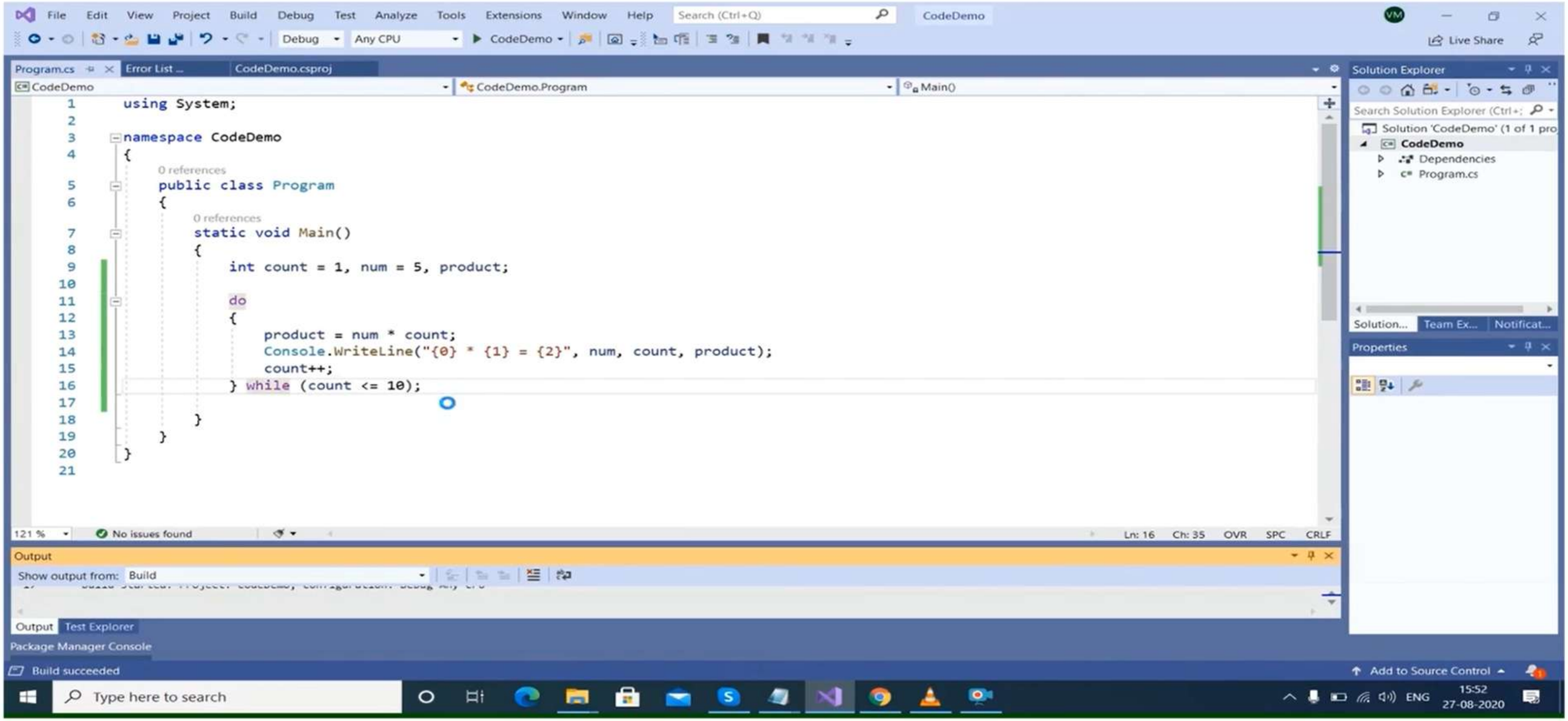
Switch Demo

TEKNOTURF



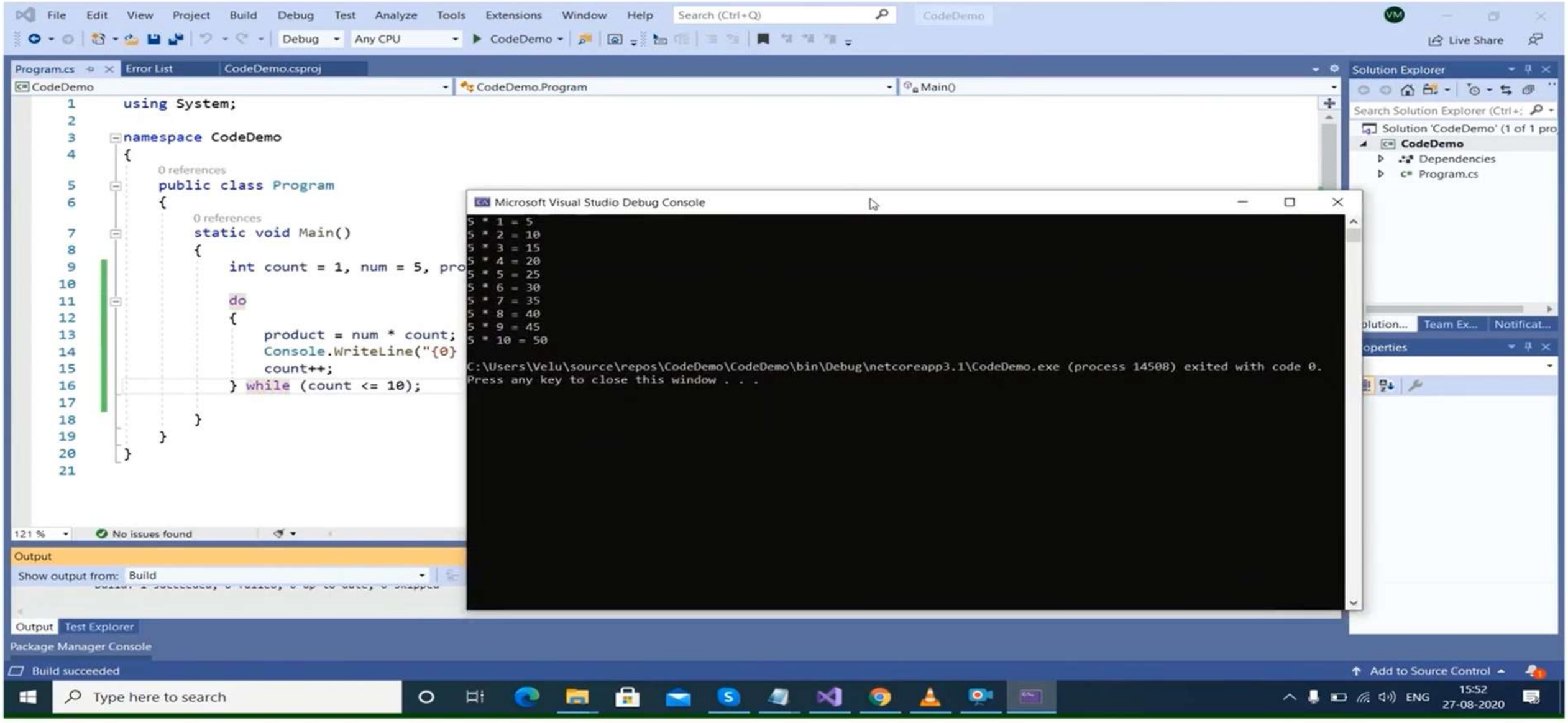
Do-while Demo

TEKNOTURF



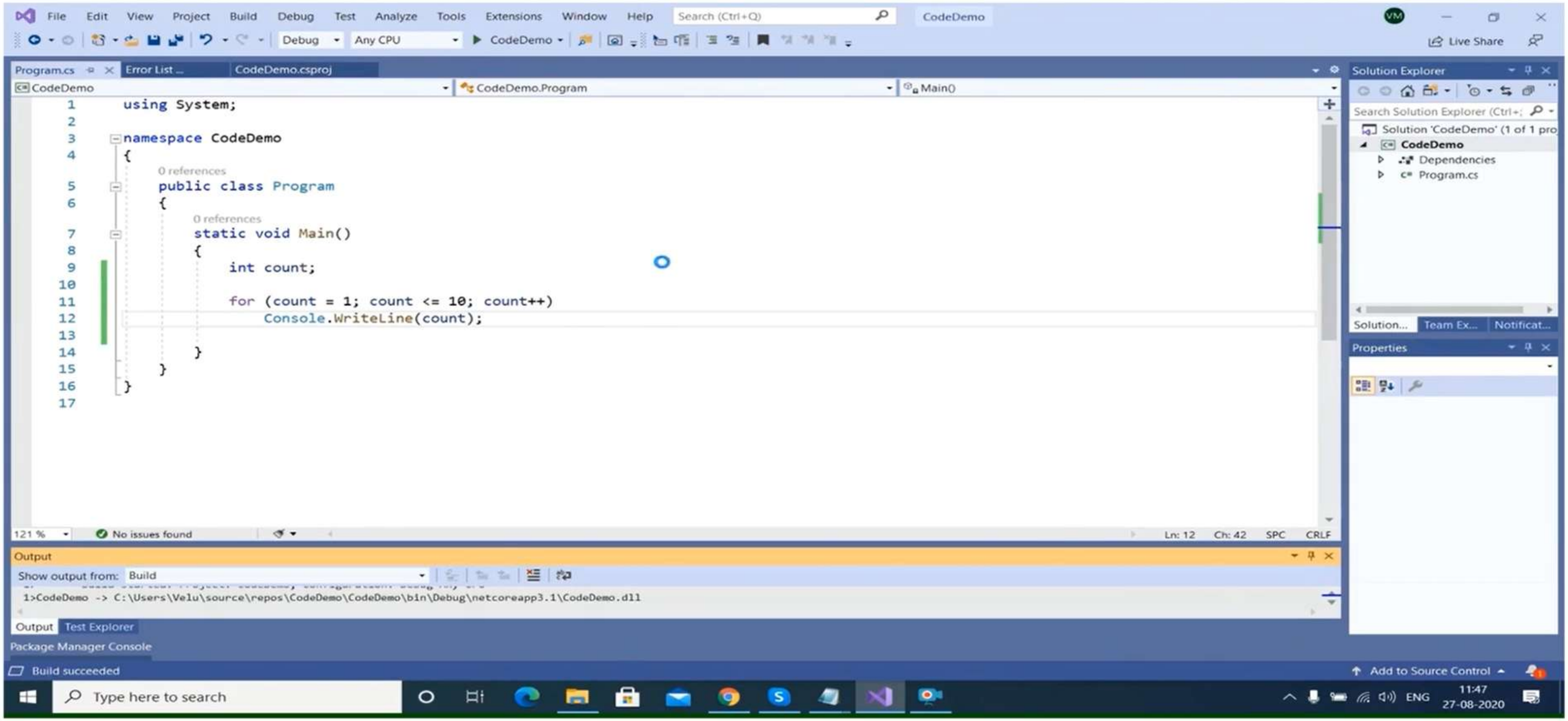
Do-while Demo

TEKNOTURF



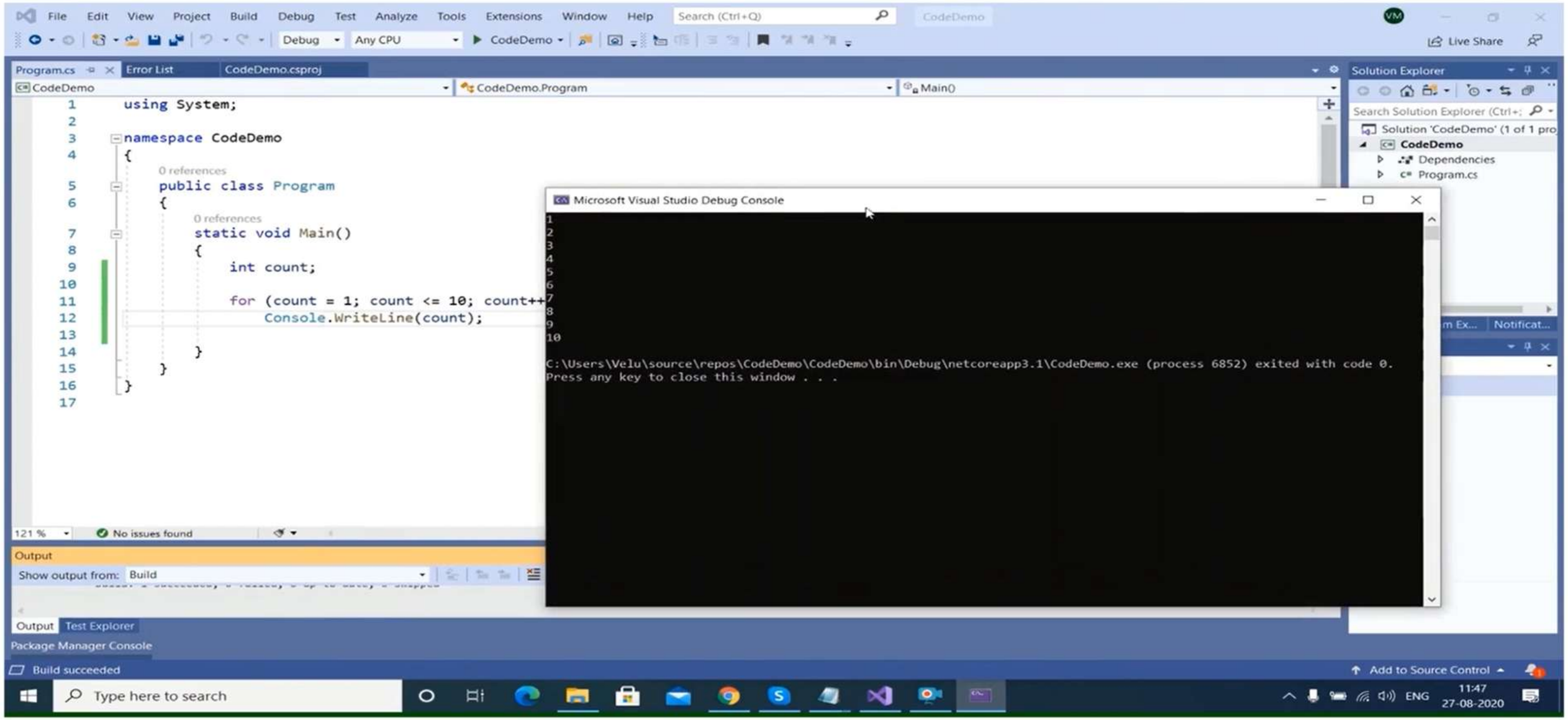
For Demo

TEKNOTURF



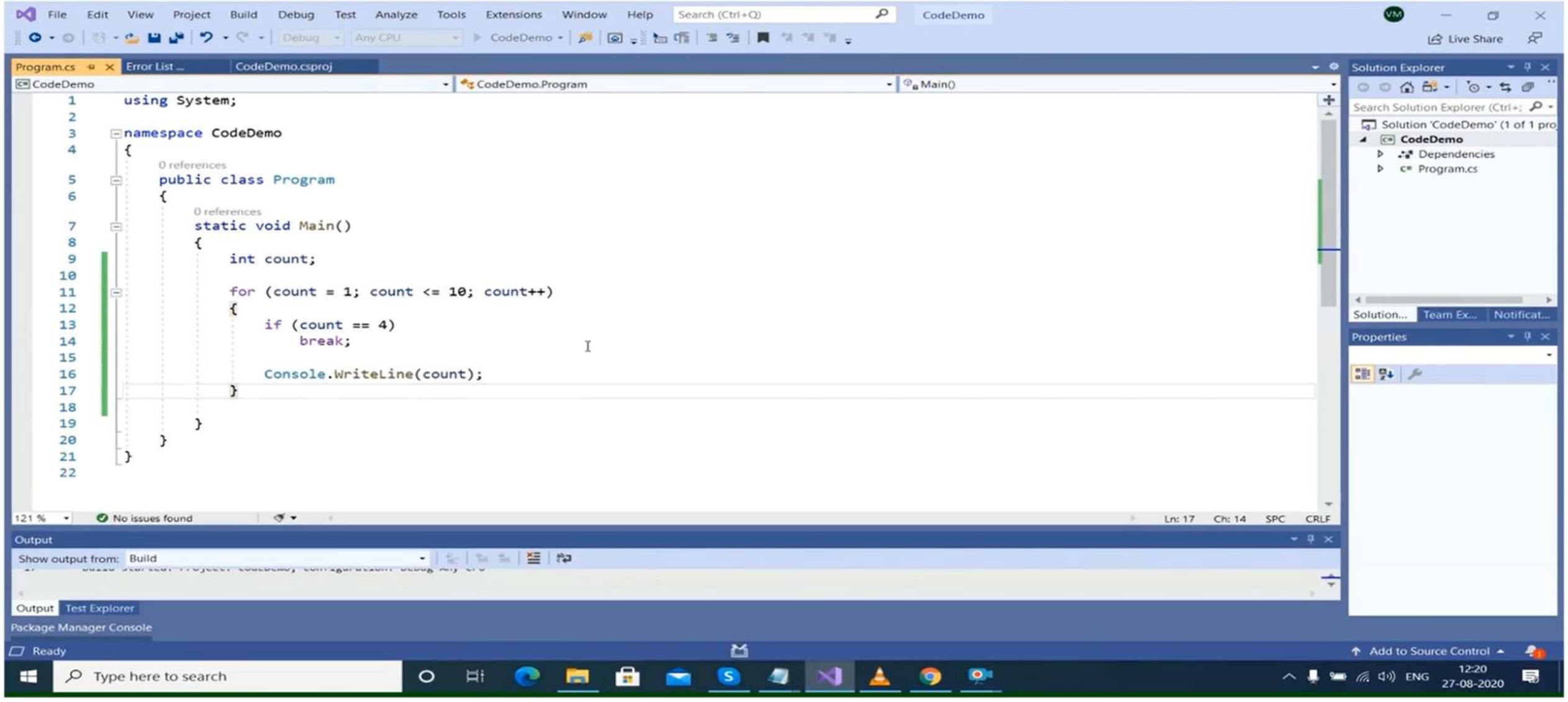
For Demo

TEKNOTURF



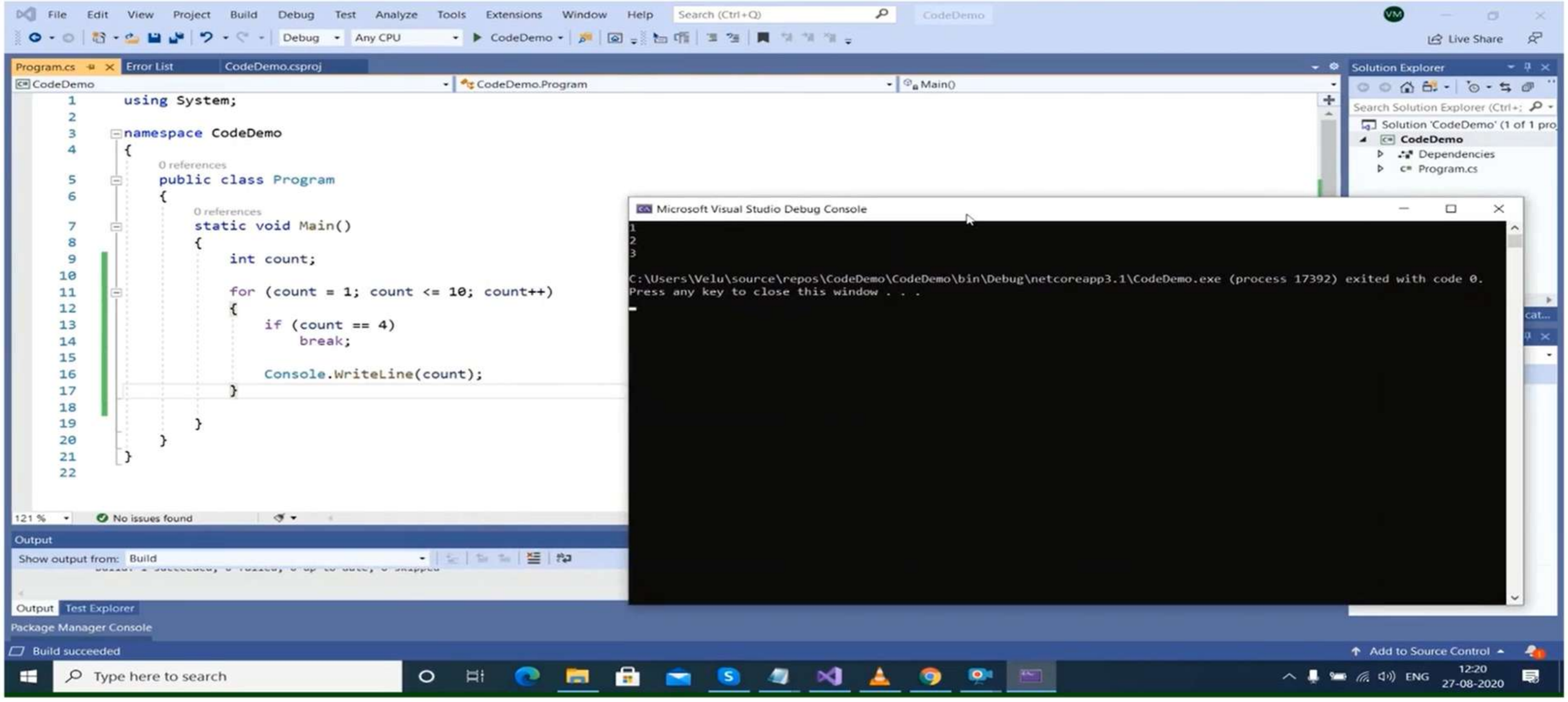
Break and Continue Demo

TEKNOTURF



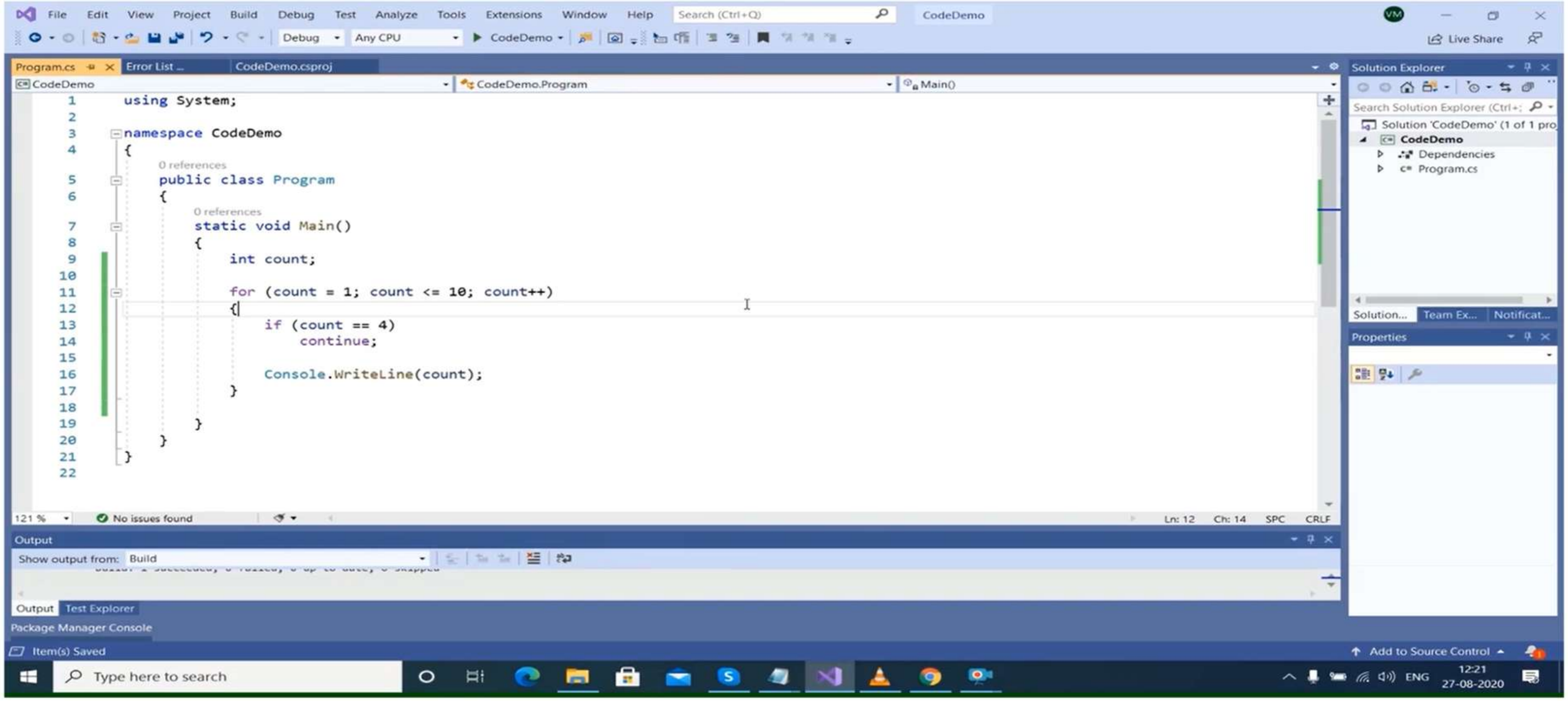
Break and Continue Demo

TEKNOTURF



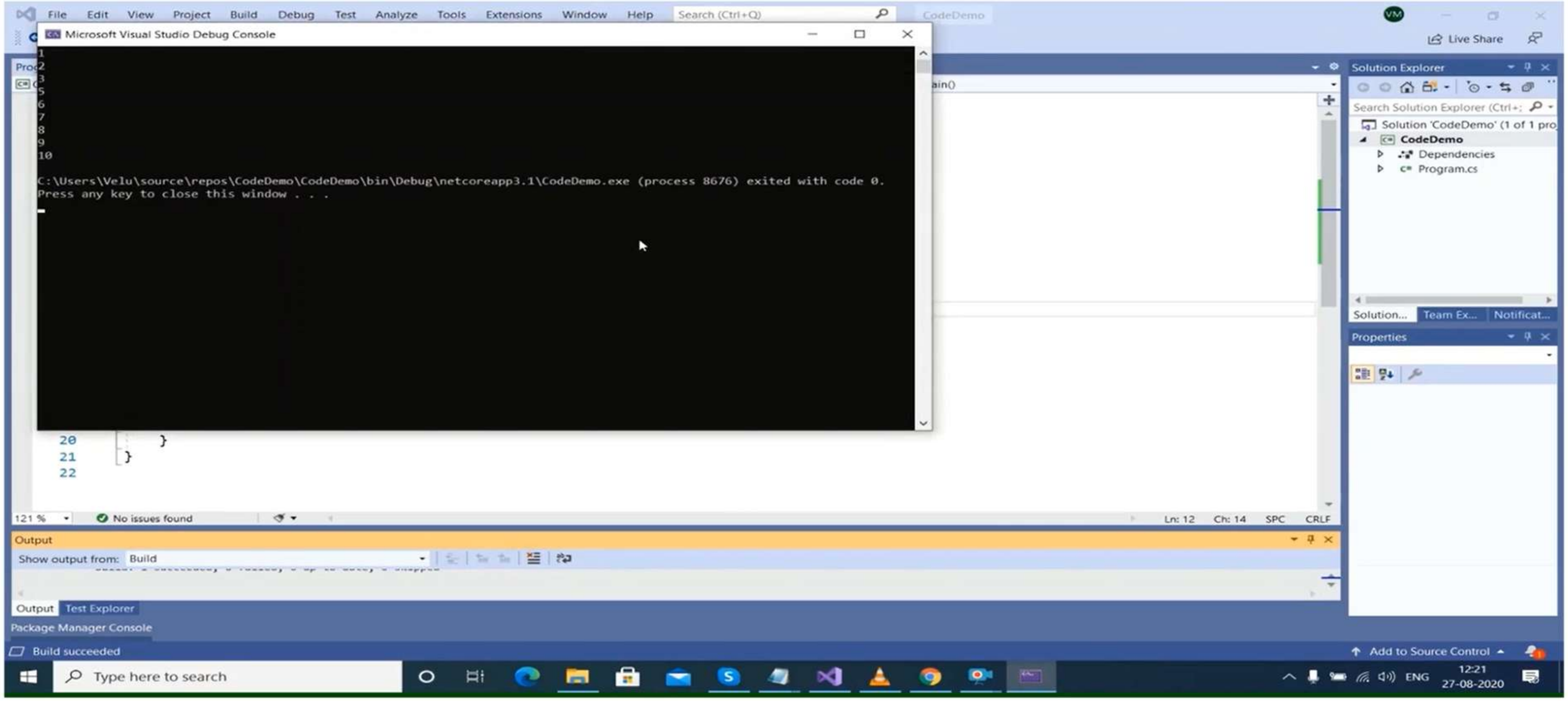
Break and Continue Demo

TEKNOTURF



Break and Continue Demo

TEKNOTURF



Summary

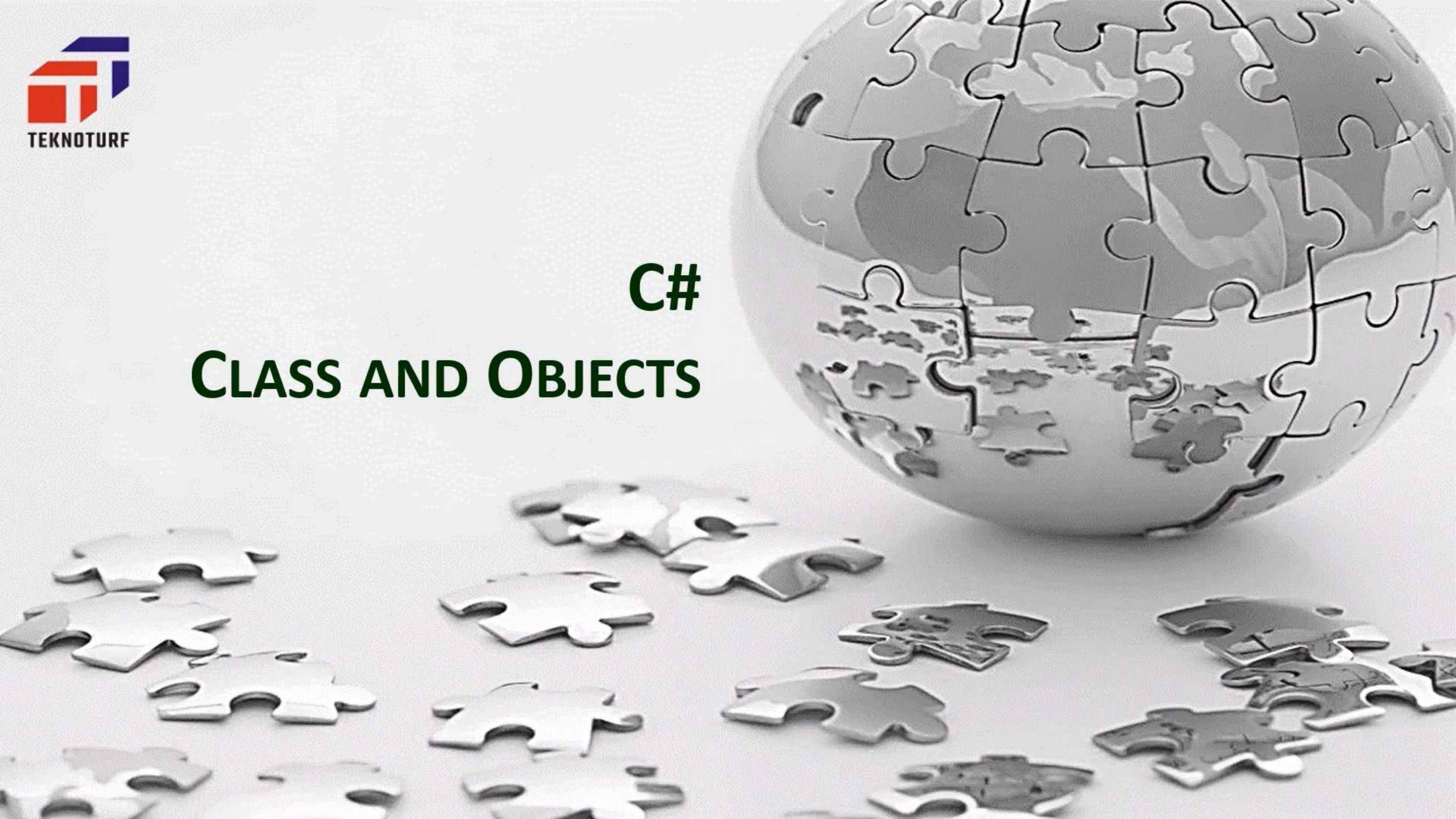
TEKNOTURF

- Decision Making
- Looping
- Sample Code Demo

TEKNOTURF

C#

Class and Objects



Objectives

TEKNOTURF

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

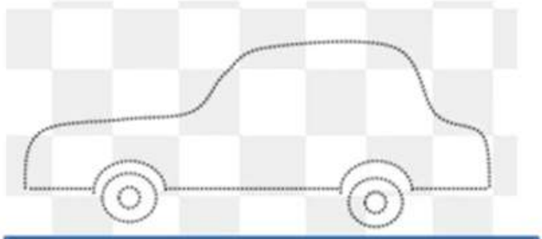
Classes and Objects

TEKNOTURF

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: Object

Class: Employee

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

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

}
```

Class Declaration

TEKNOTURF

Declaring a class

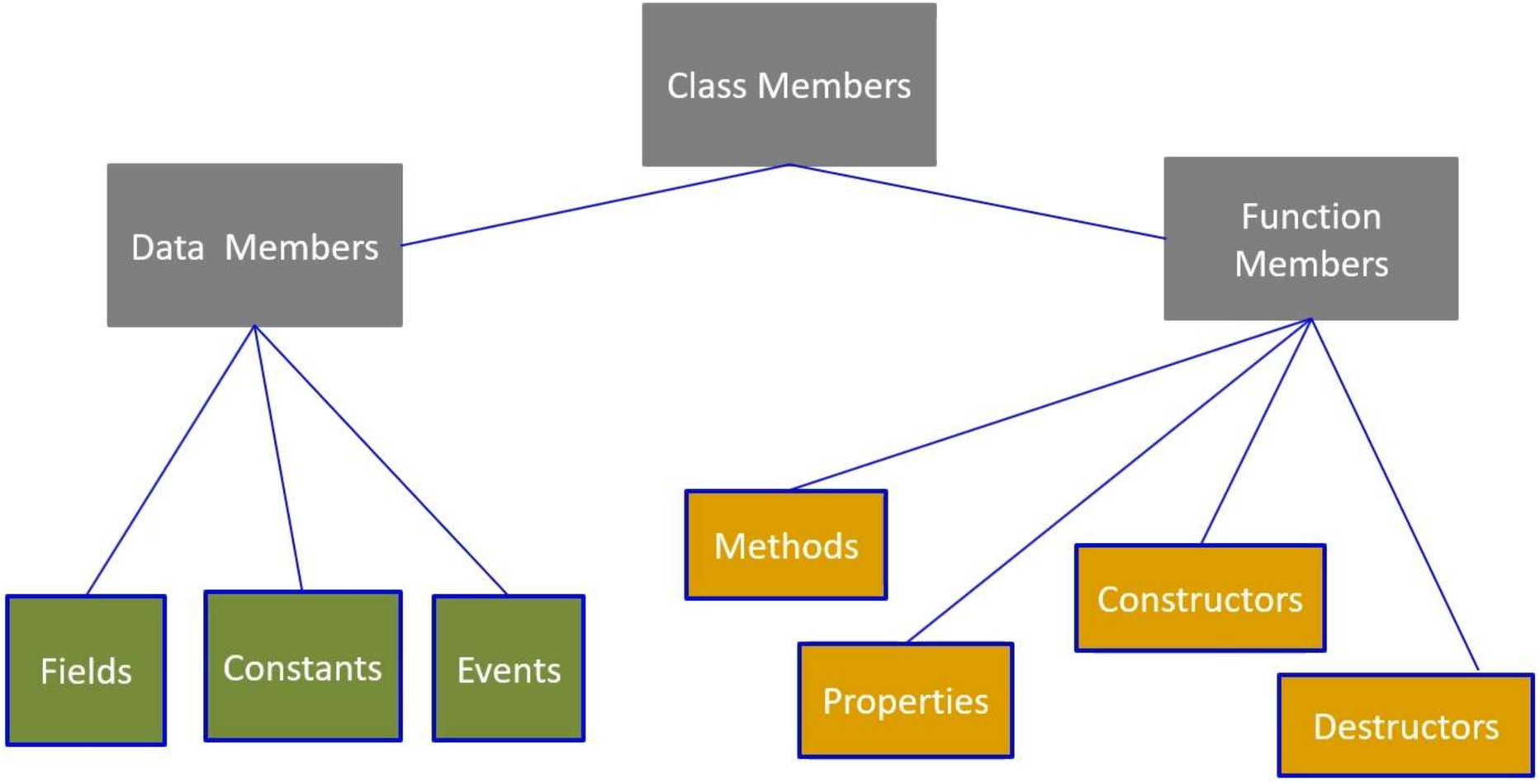
```
<modifier>* class <class_name> {
        <attribute_declaration>*
        <constructor_declaration>*
        <method_declaration>*
}
```

Example:

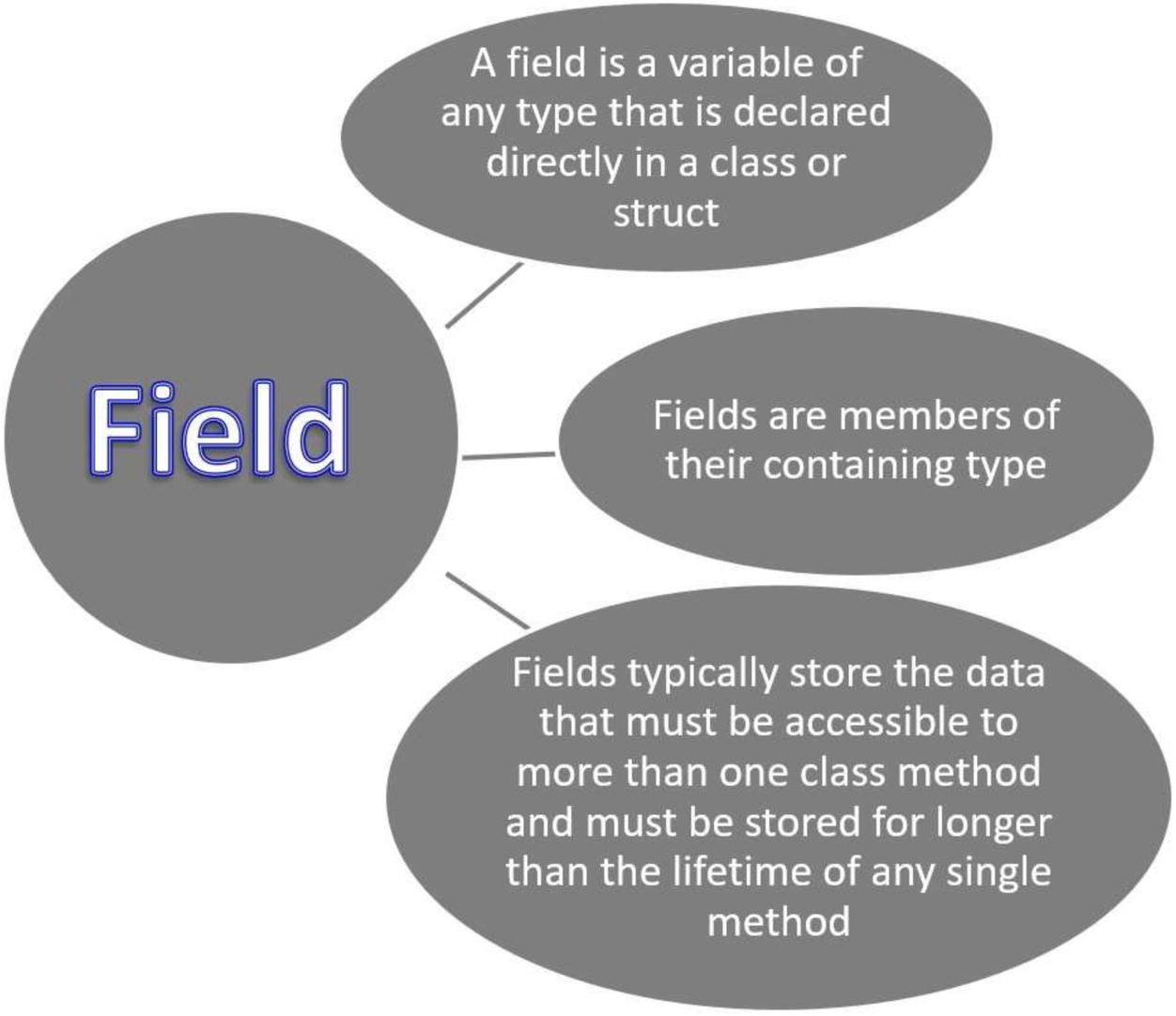
```
public class Employee {
    private int empId;
    private String name;
    private  double salary;
    public void SetEmpId(int  id) {
       empId = id;
    }
}
```

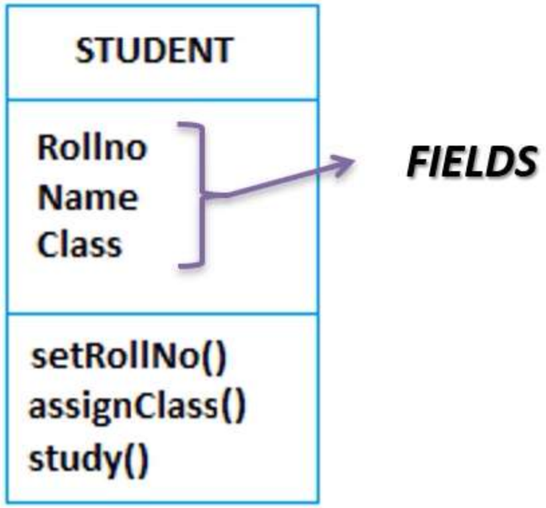
Members of class

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



Field


A field is a variable of any type that is declared directly in a class or struct
Field
Fields are members of their containing type
Fields typically store the data that must be accessible to more than one class method and must be stored for longer than the lifetime of any single method


STUDENT
Rollno
Name
Class
FIELDS
setRollNo()
assignClass()
study()

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.

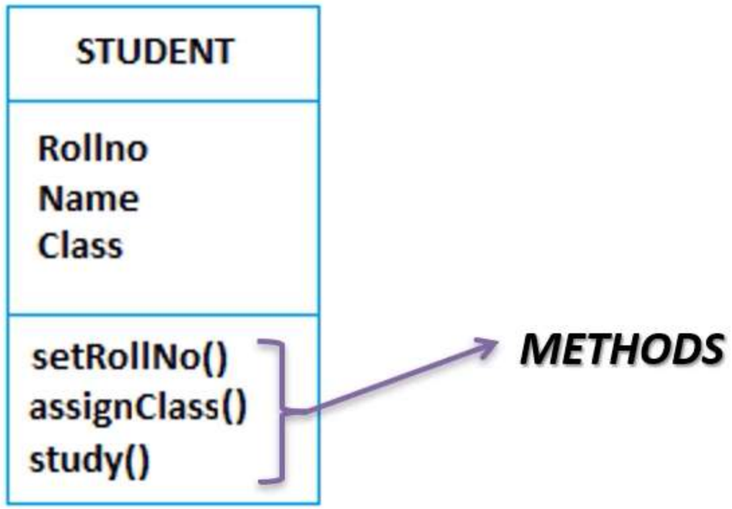
Reference fields are set to null

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

Methods of Class

TEKNOTURF

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

TEKNOTURF

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

TEKNOTURF

Creation of Object

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

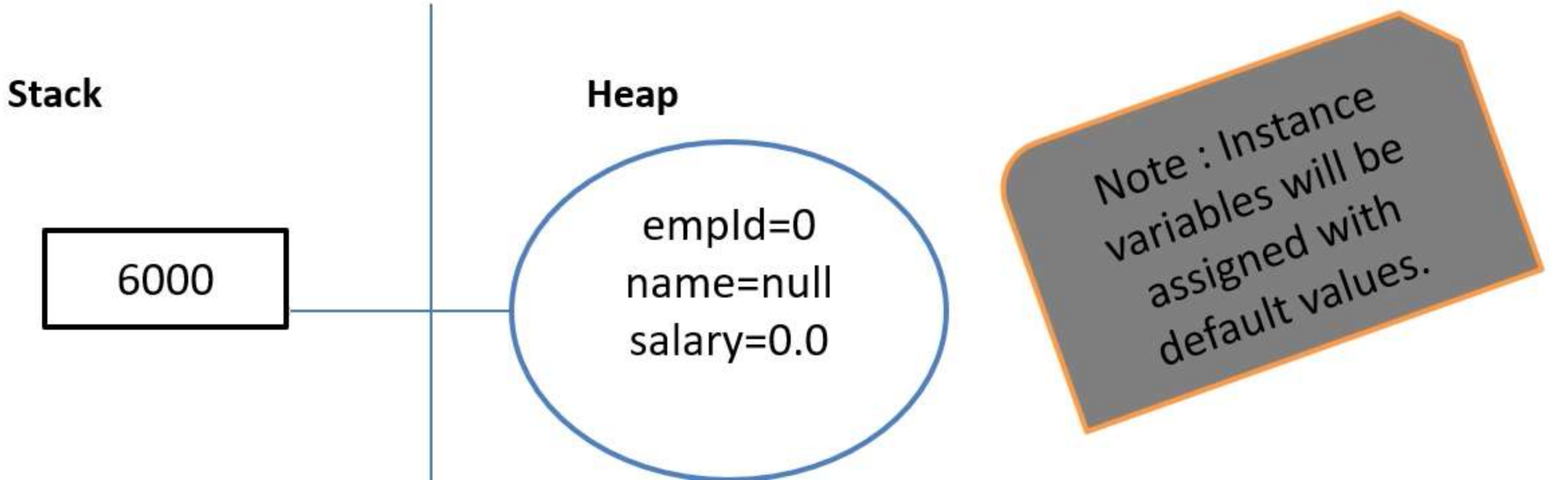
Syntax

- 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

TEKNOTURF

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 empId;
     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.empId=101;
          empObj.name="Rohit";
          empObj.salary=25000;
      //retrieve values
           Console.WriteLine("Emp ID
      "+empObj.empId);
       Console.WriteLine("Name  "+empObj.name);
 Console.WriteLine("Grosspay: "
      empObj.GrossPay(5000));     }
}
```

Controlling the visibility of class members

TEKNOTURF

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.

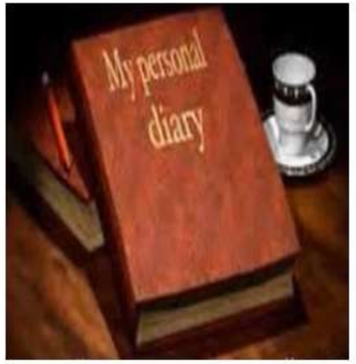
Access Specifier

public

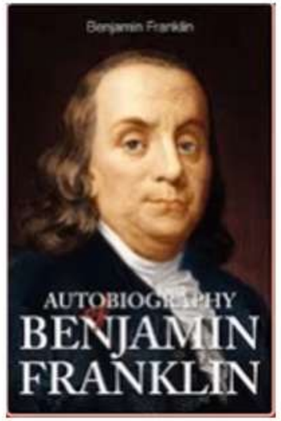
- Classes, methods, and fields declared as public can be accessed from any class in an application

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

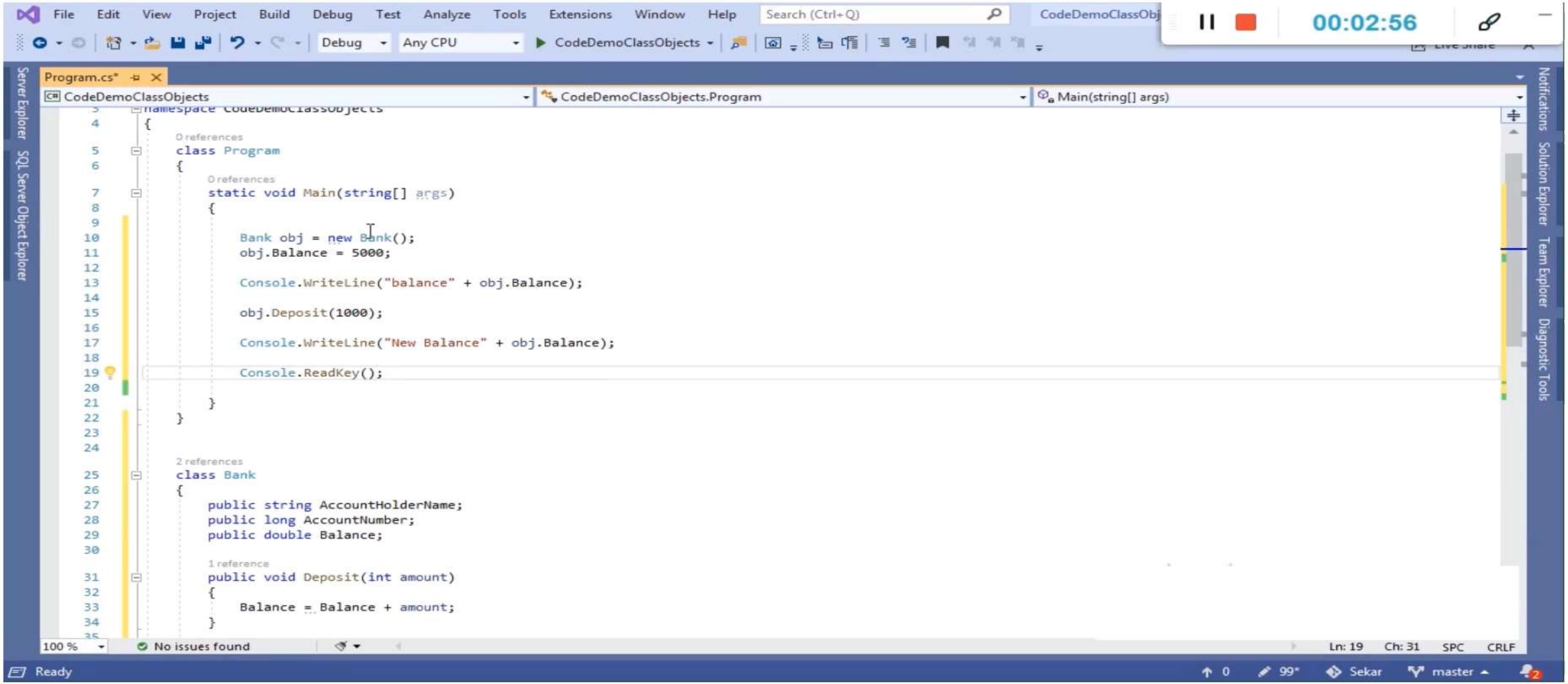


public

Code Demo

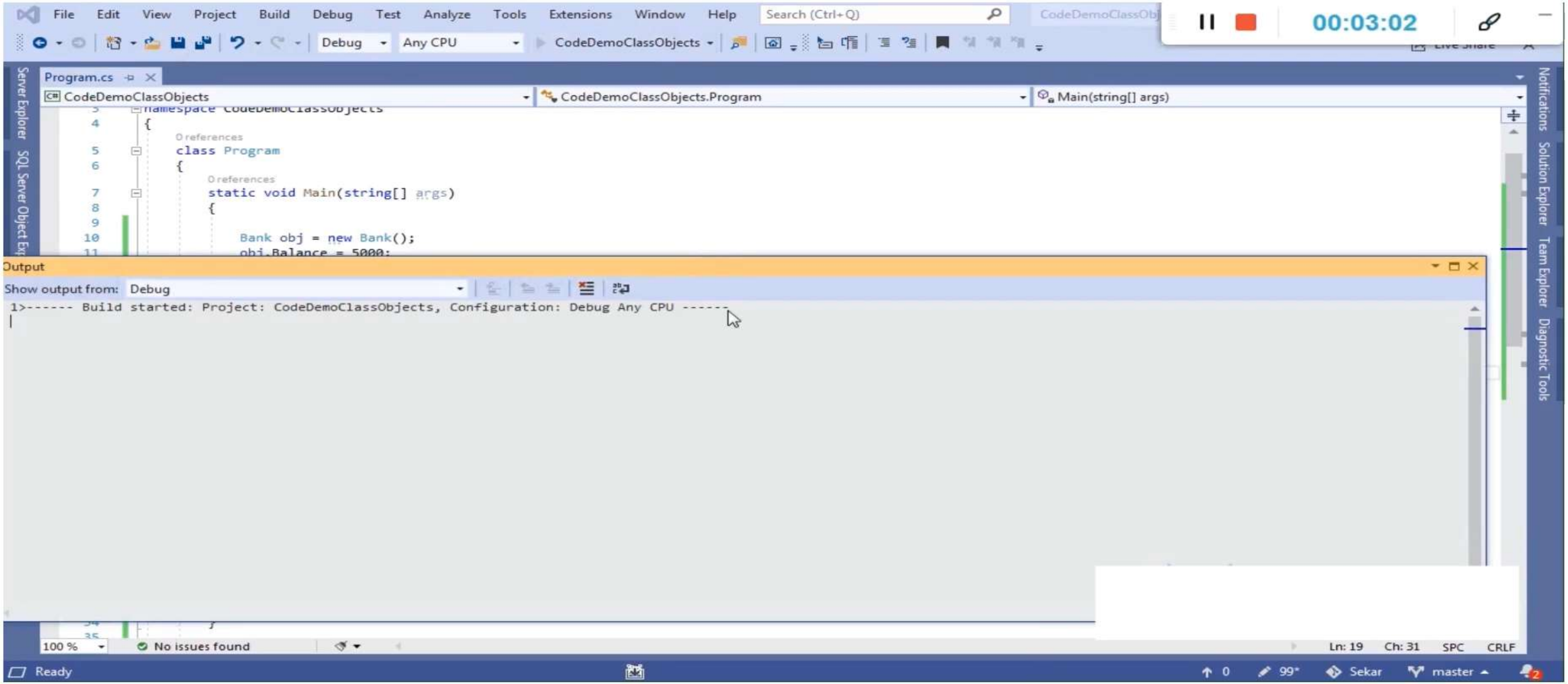
Class and Object Demo

TEKNOTURF



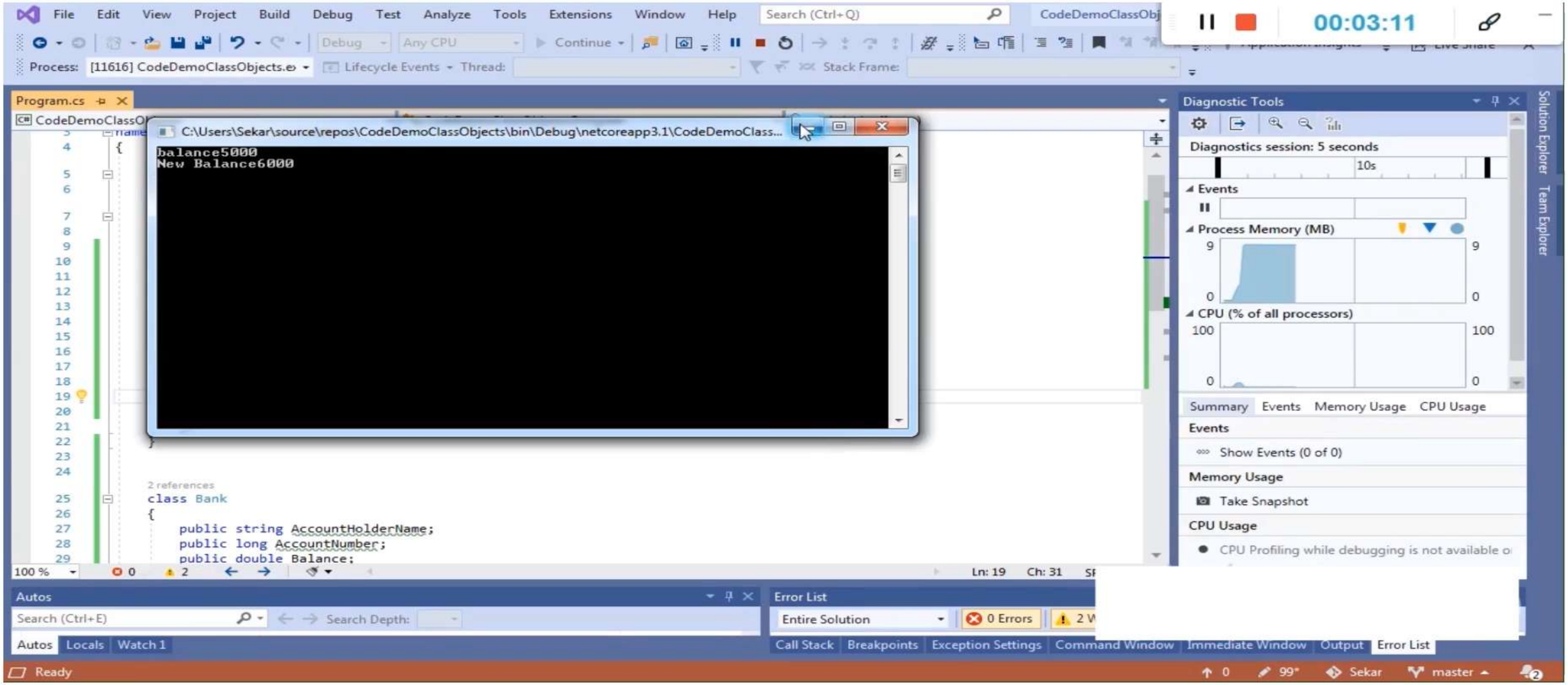
Class and Object Demo

TEKNOTURF



Class and Object Demo

TEKNOTURF



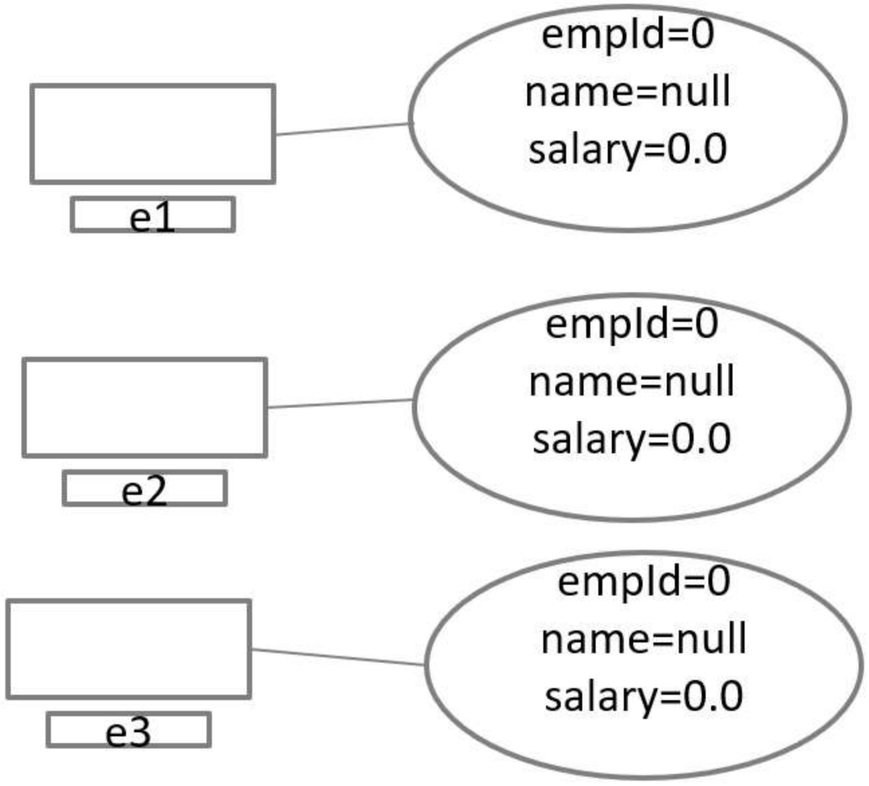
Constructors

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

TEKNOTURF

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

TEKNOTURF

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

A constructor that takes zero parameters is called a default constructor

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

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

Default Constructor

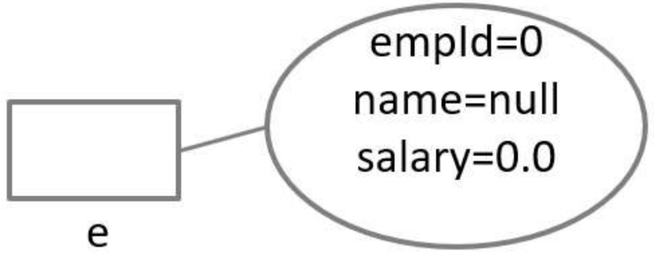
Default Constructor (no argument constructor)

A constructor with no arguments is known as default constructor

Example:

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

  public Employee()
 {
      Console.WriteLine("Default Constructor");
 }
 public static void Main(String a[])
{
       Employee e=new Employee();}
 }
}
```

Default Constructor

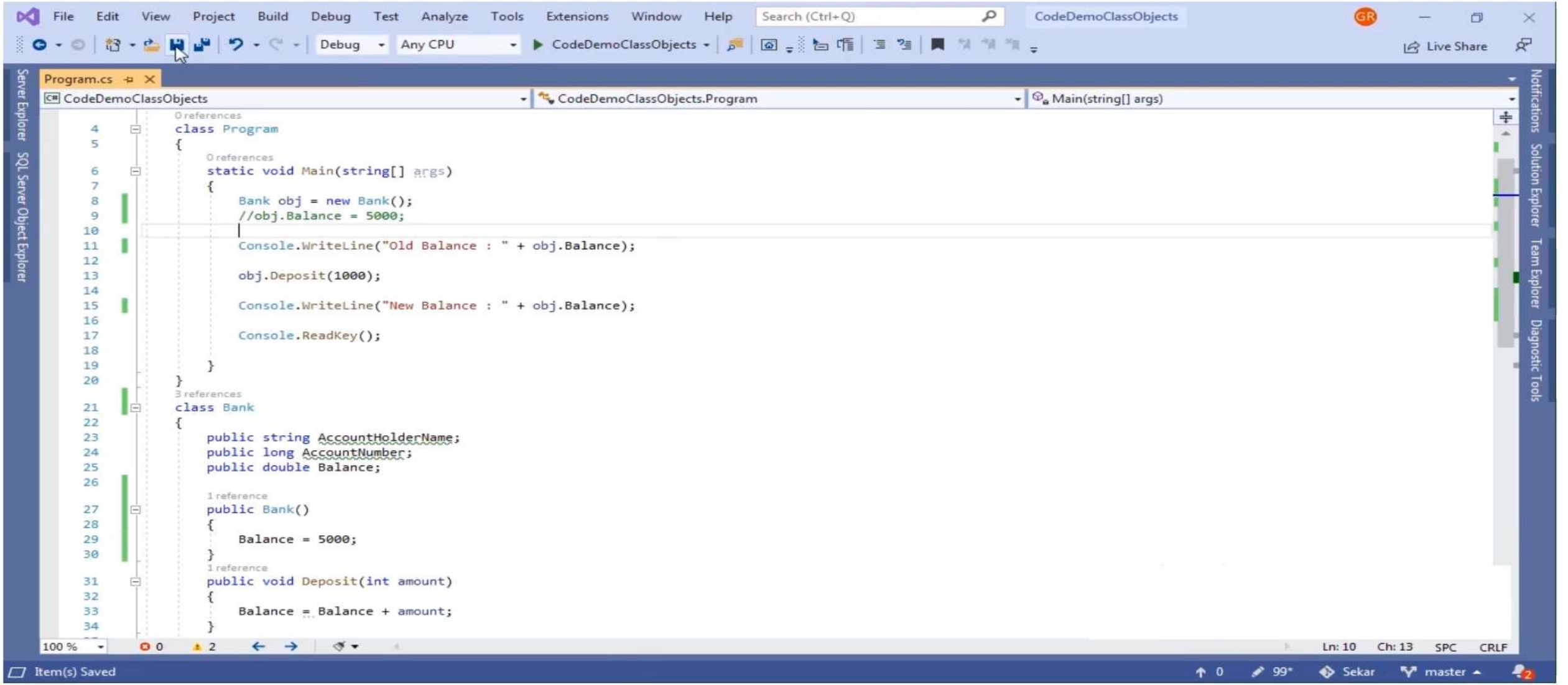


Output : Default Constructor

Code Demo

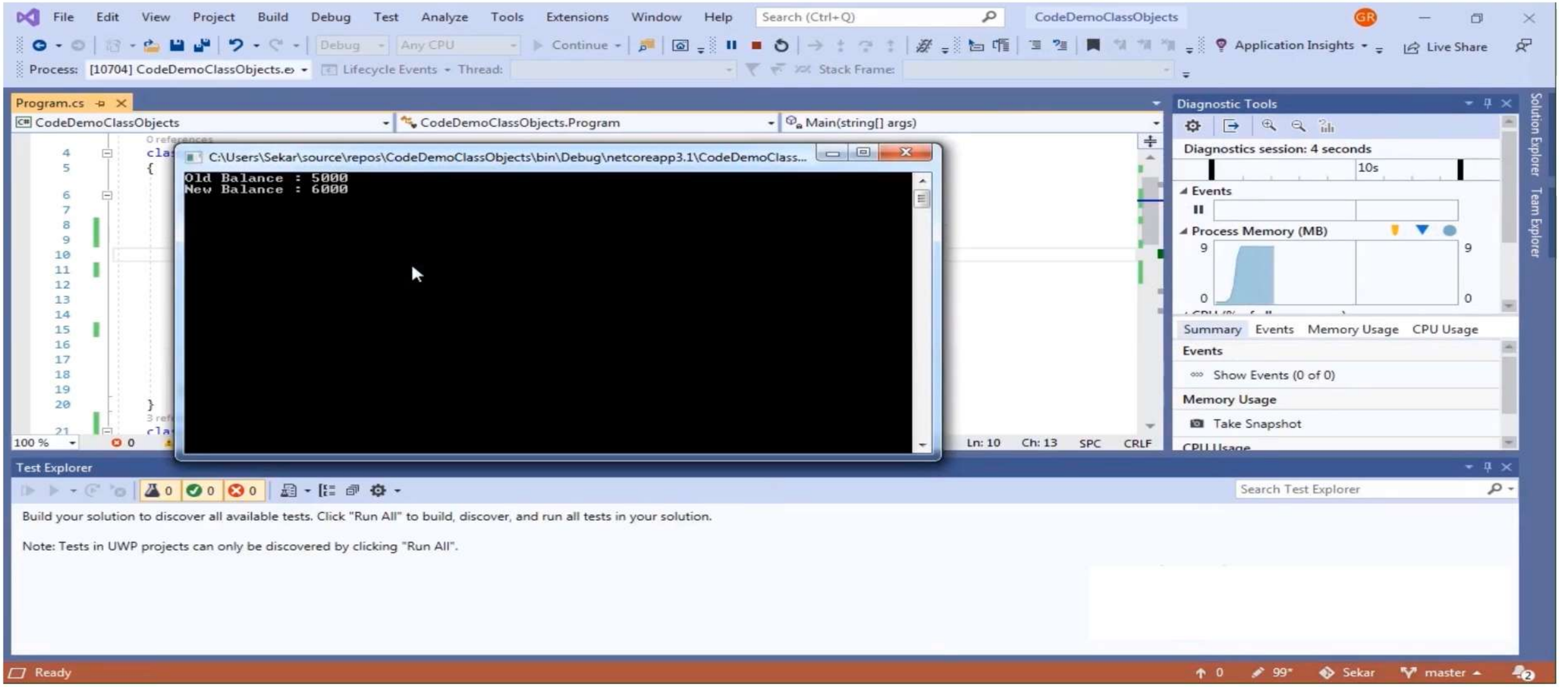
Default Constructor Demo

TEKNOTURF



Default Constructor Demo

TEKNOTURF



Parameterized Constructor

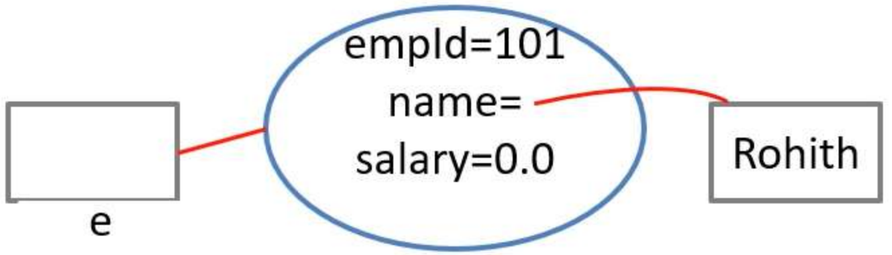
Parametrized Constructor

Constructor with argument list is known as parameterized constructor.

Example:

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

 //Constructor
 public Employee(int id,String ename)
 {
      Console.WriteLine("In Parametrized Constructor");
     empId=id;
      name=ename;
 }
  static void Main(String a[])
  {
        Employee e=new Employee(101,"Rohith");
   }
}
```

Output : In Parametrized Constructor

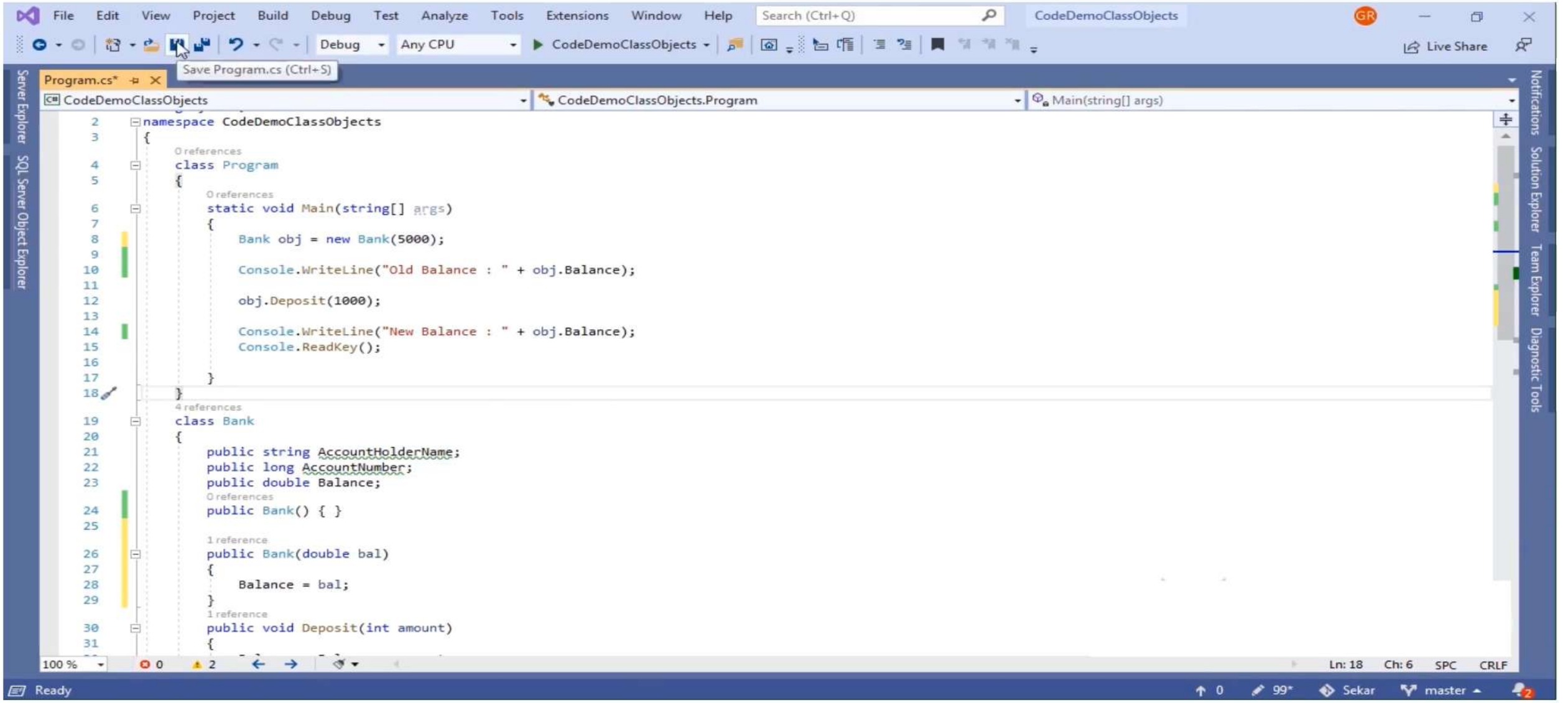


TEKNOTURF

Code Demo

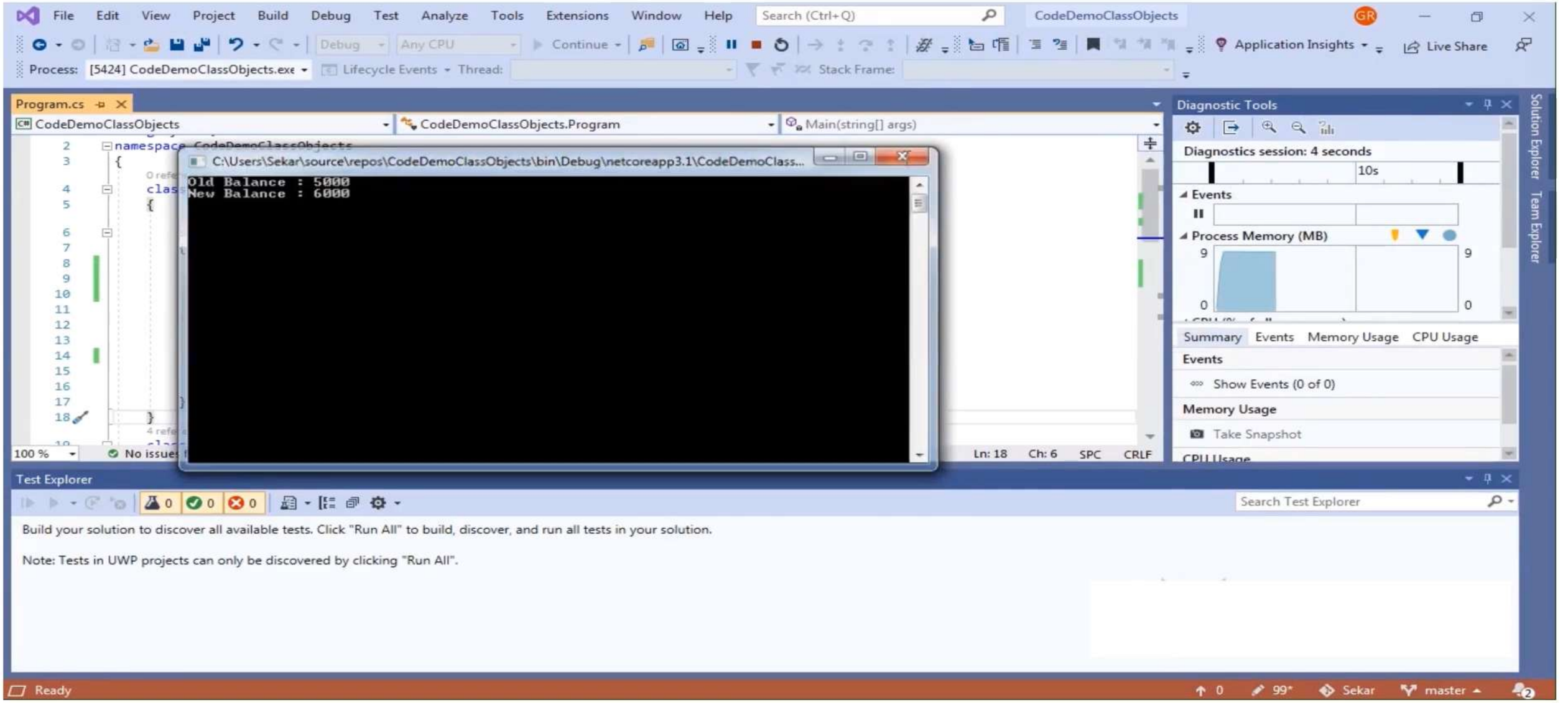
Parameterized Constructor Demo

TEKNOTURF



Parameterized Constructor Demo

TEKNOTURF



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 empId;
   String name;

   public Employee(int empId,String name)
   {
      empId = empId;
      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 empId;
   String name;

   public Employee(int empId,String name)
   {
      this.empId = empId;
      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

TEKNOTURF

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

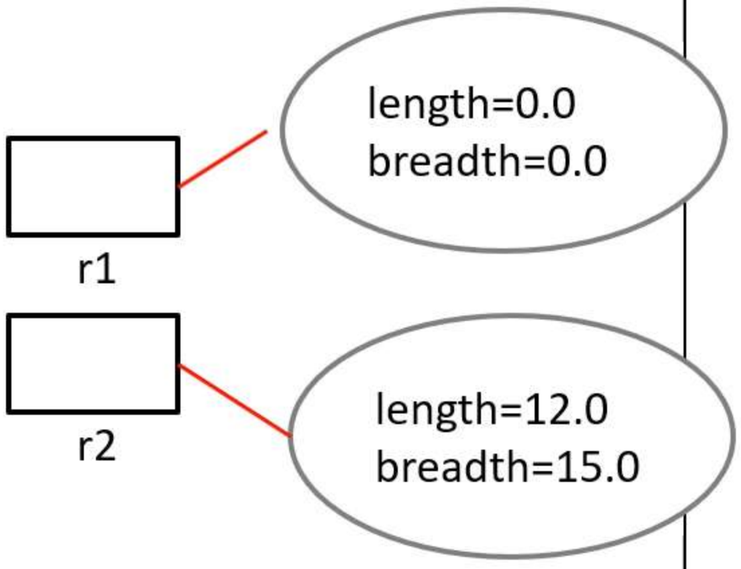
- Number of parameters
- Type of parameter
- Order of parameter

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);
}
```

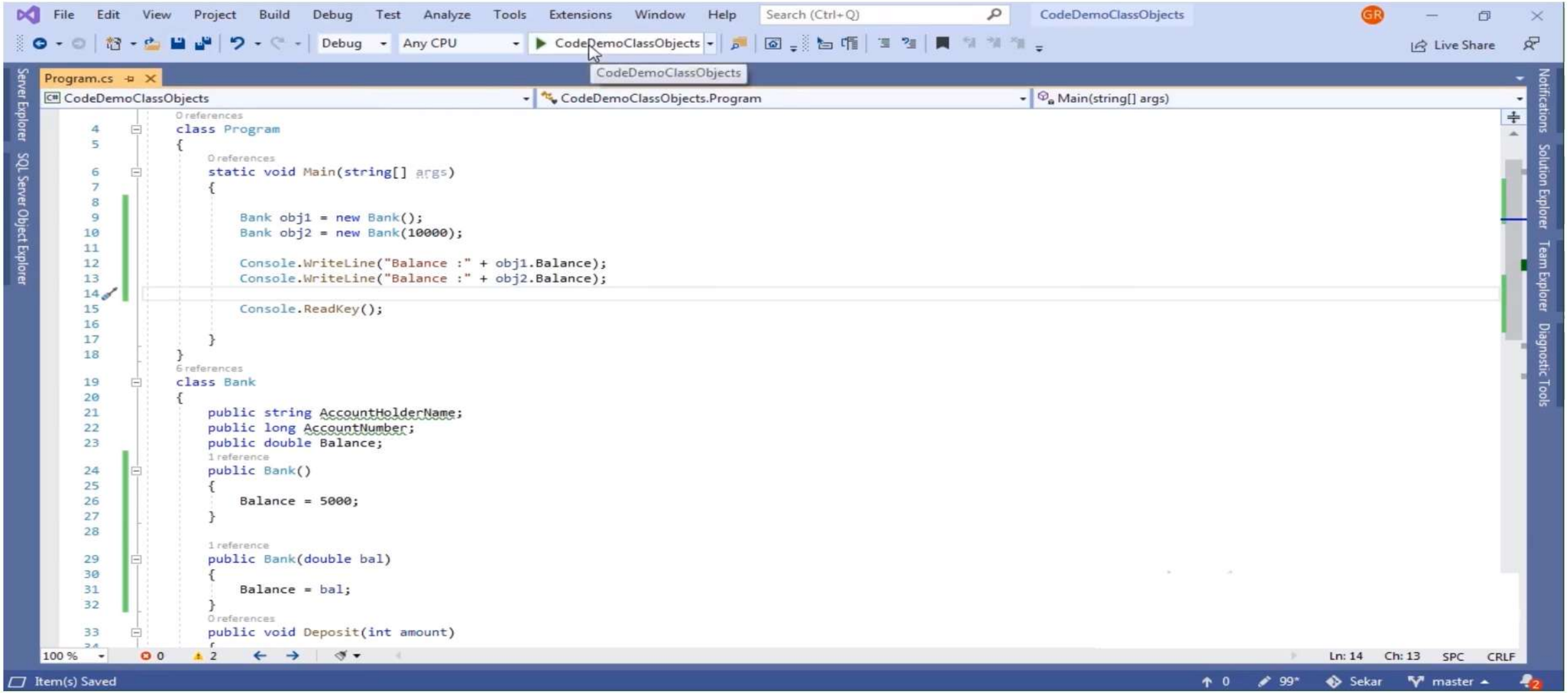


TEKNOTURF

Code Demo

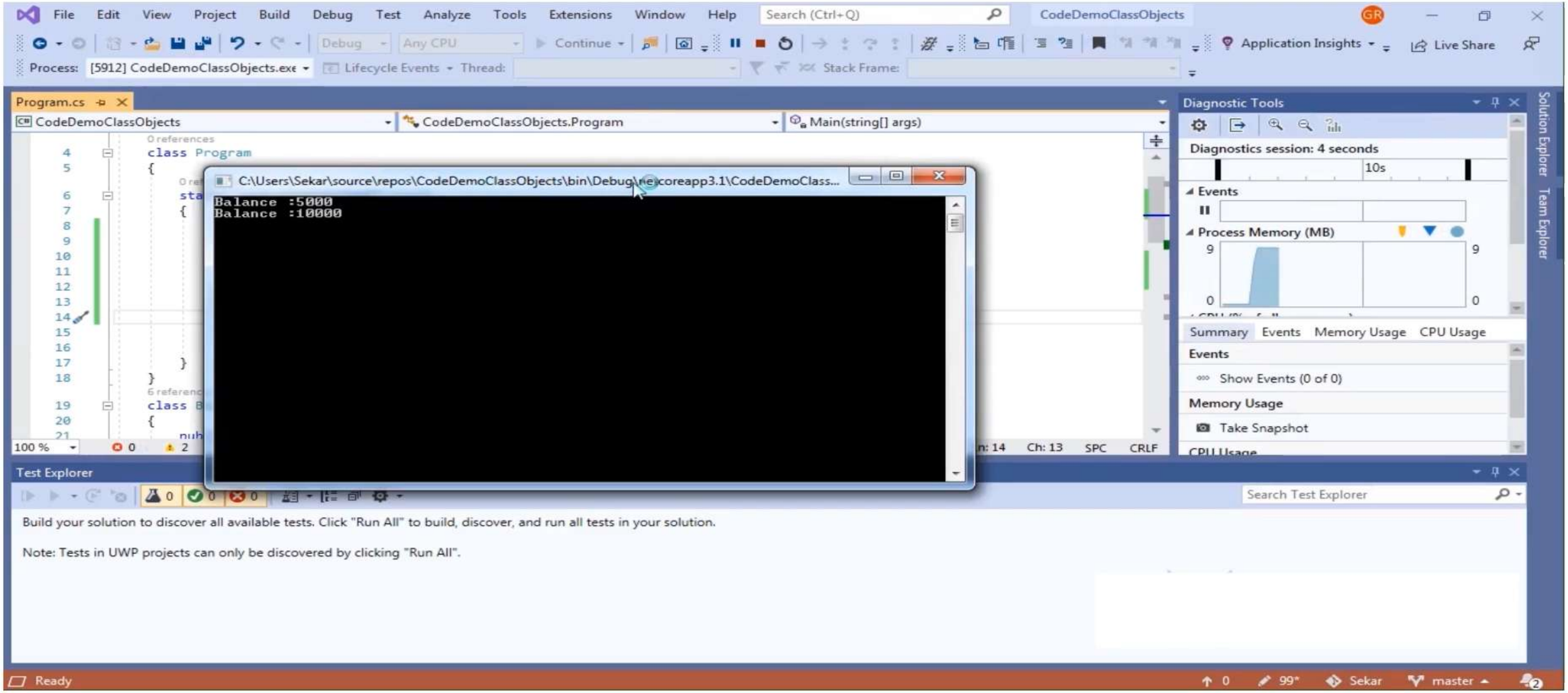
Constructor Overloading Demo

TEKNOTURF



Constructor Overloading Demo

TEKNOTURF



Constructors Access Specifiers

TEKNOTURF

- 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

TEKNOTURF

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

TEKNOTURF

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

TEKNOTURF

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

TEKNOTURF

C# STRINGS

Objectives

TEKNOTURF

In this module you will learn

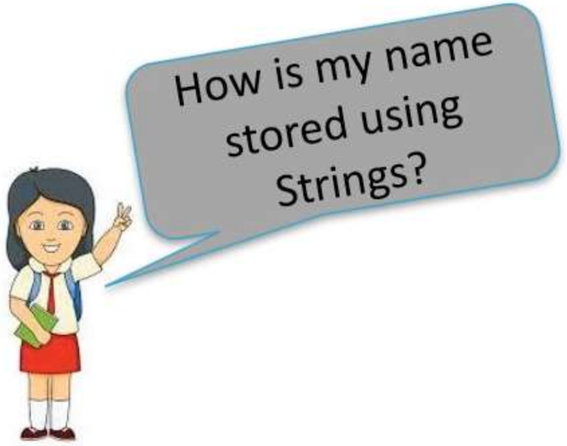
- String
- String Builder
- String Methods

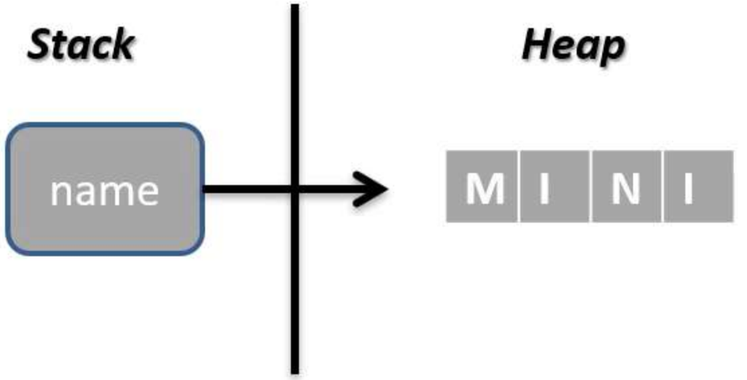
String

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

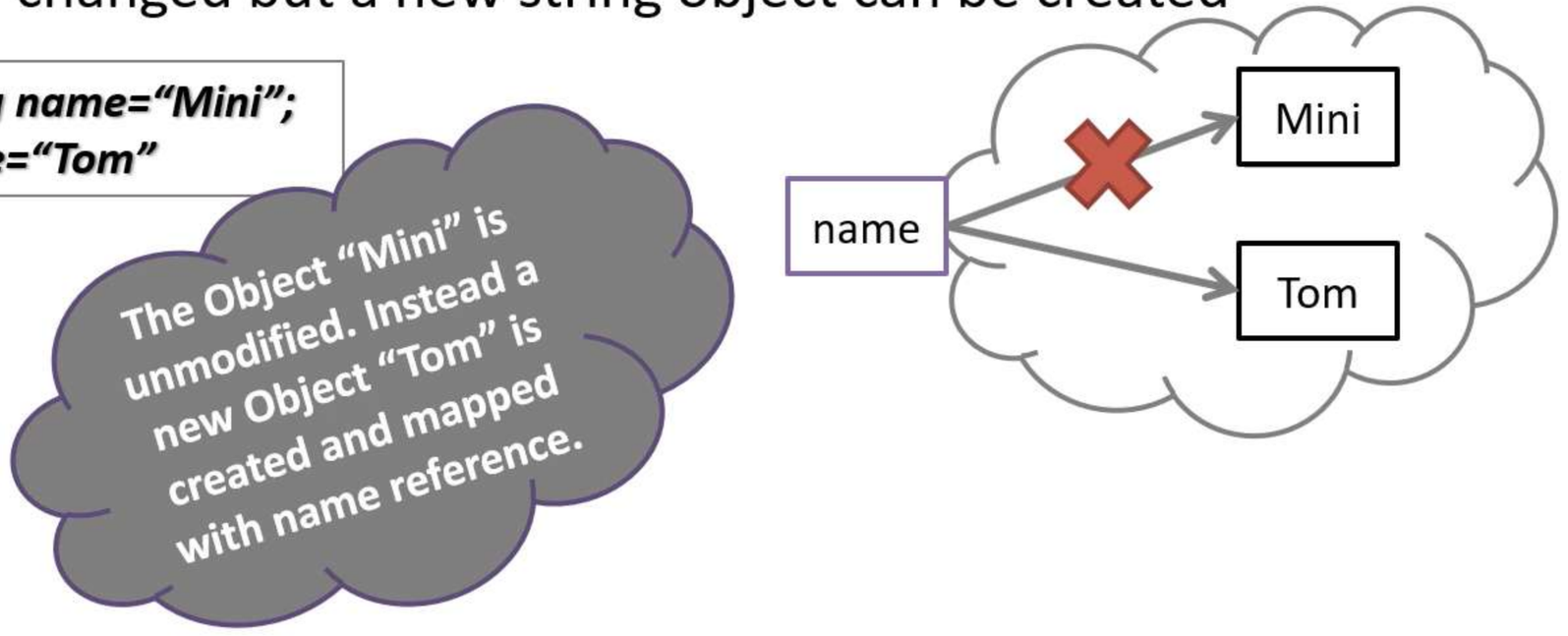
```
string message1 = "Hello"; // returns "Hello"
message1 += ", There"; // returns "Hello, There"
string message2 = message1 + "!"; // returns "Hello, There!"
```

```
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"



StringBuilder

TEKNOTURF

StringBuilder class normally allocates more memory than is actually needed

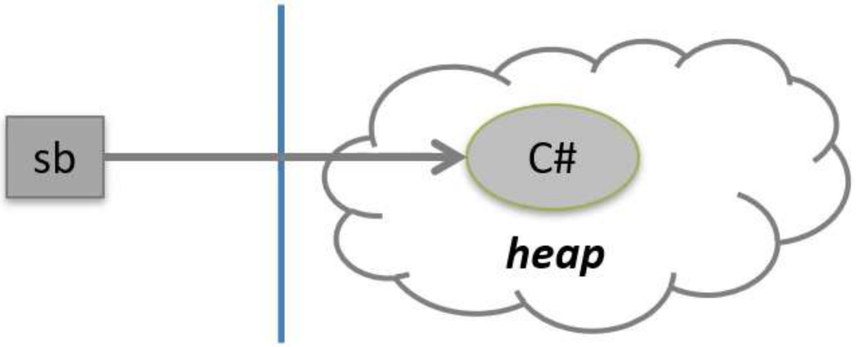
StringBuilder is mainly used for substitutions and appending or removing text from strings

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

TEKNOTURF

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

TEKNOTURF

The String class has the following two properties −

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

Methods of the String Class

TEKNOTURF

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

Method Name	Description
Compare(String, String)	It is used to compares two specified String objects. It returns an integer that indicates their relative position in the sort order.
CompareTo(String)	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.
Concat(String, String)	It is used to concatenate two specified instances of String.
Contains(String)	It is used to return a value indicating whether a specified substring occurs within this string.

Methods

TEKNOTURF

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

Methods

TEKNOTURF

Method Name	Description
Replace(String, String)	**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.**
Split(Char[])	It is used to split a string into substrings that are based on the characters in an array.
StartsWith(String)	It is used to check whether the beginning of this string instance matches the specified string.
Substring(Int32)	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.
ToCharArray()	It is used to copy the characters in this instance to a Unicode character array.

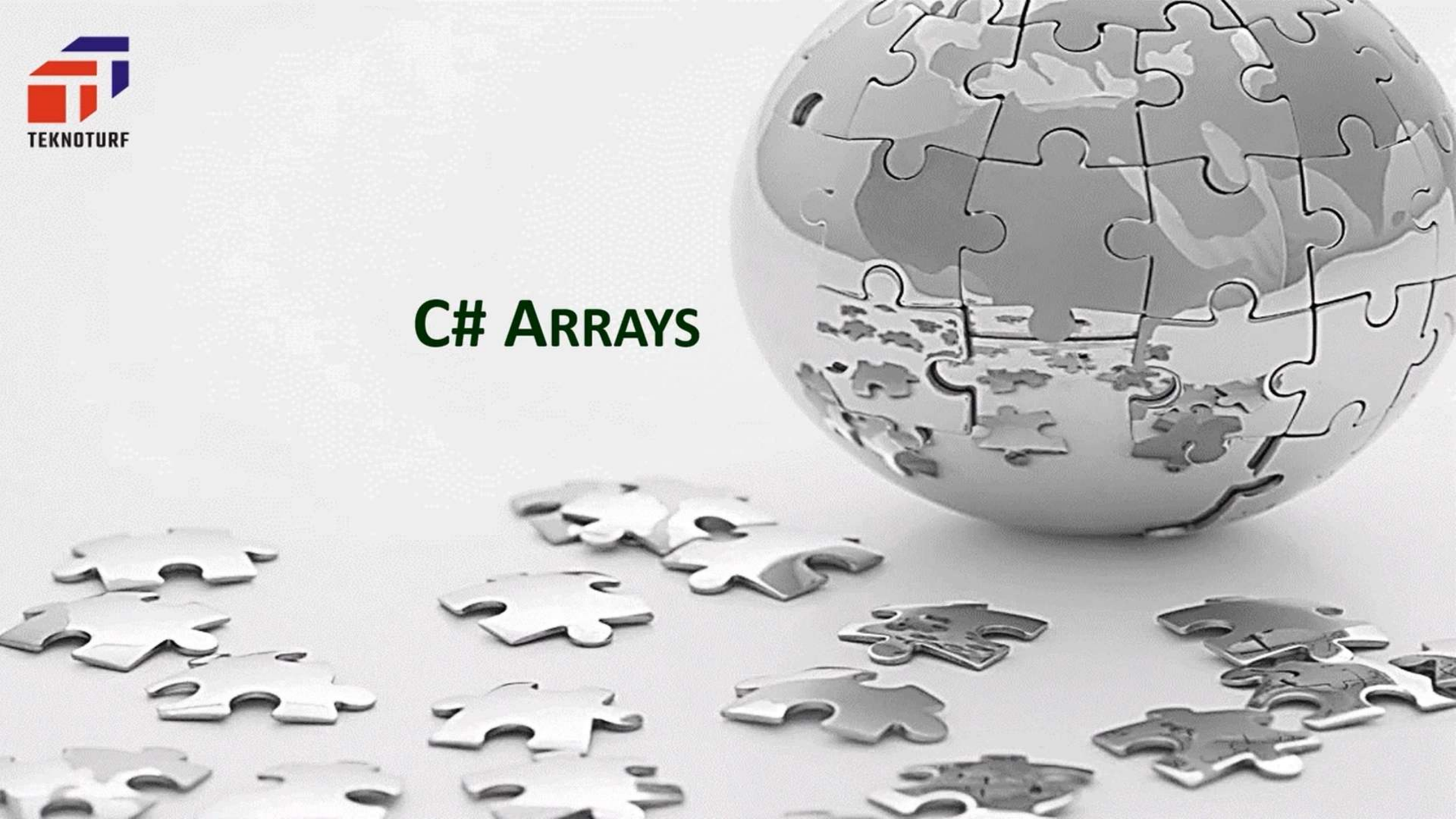
Summary

TEKNOTURF

- String
- String Builder
- String Methods

TEKNOTURF

C# ARRAYS



Objectives

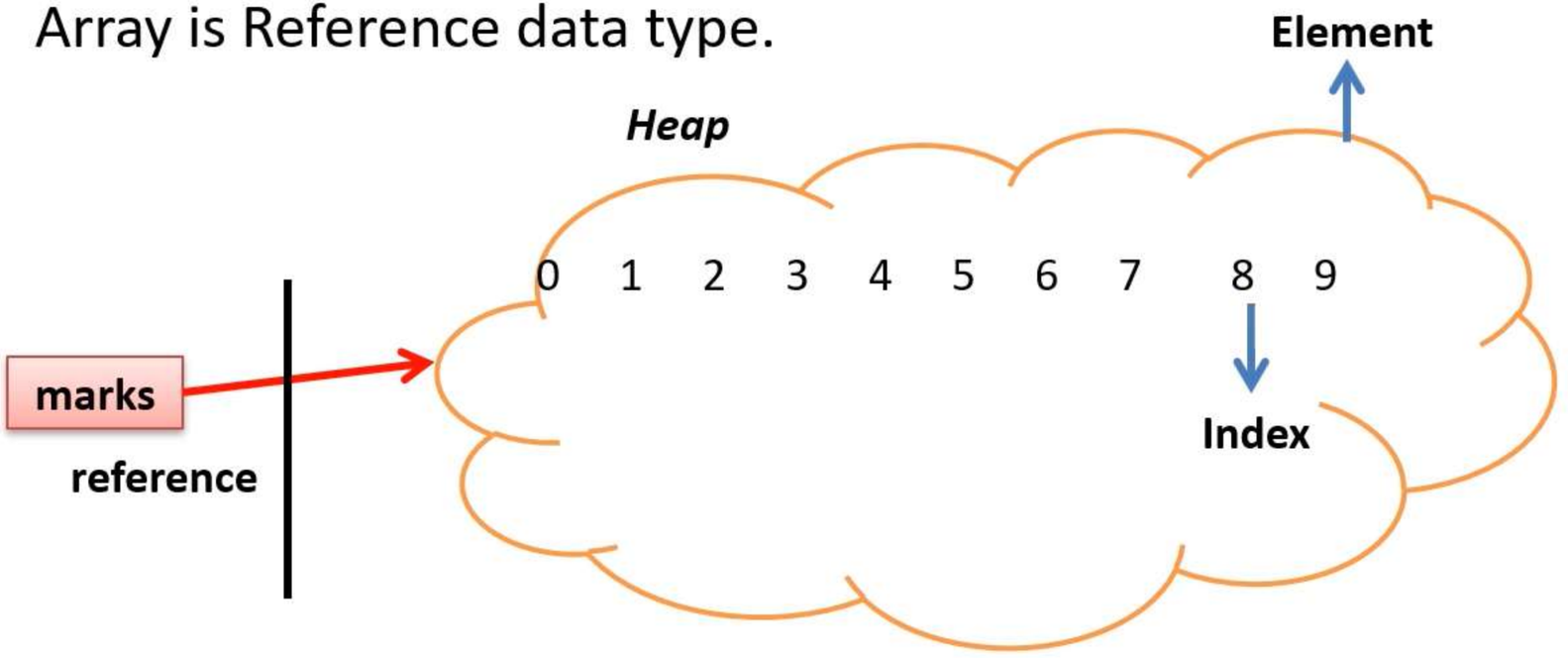
TEKNOTURF

In this module you will learn

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

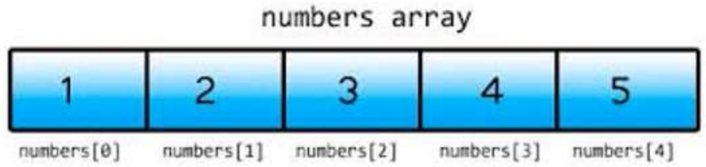
Arrays

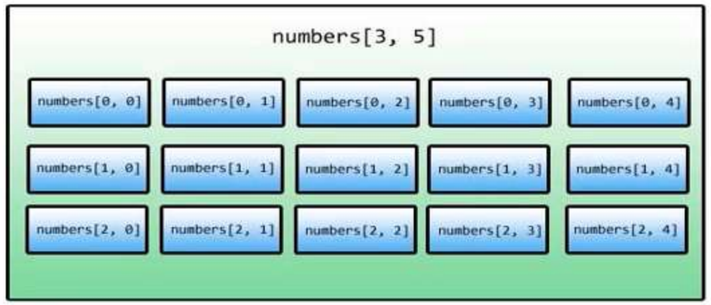
- Array is an ordered collection that stores elements of the same type
- Array elements are stored contiguously
- Array in C# is index based; the first element of the array is stored at 0 index
- Array is Reference data type.

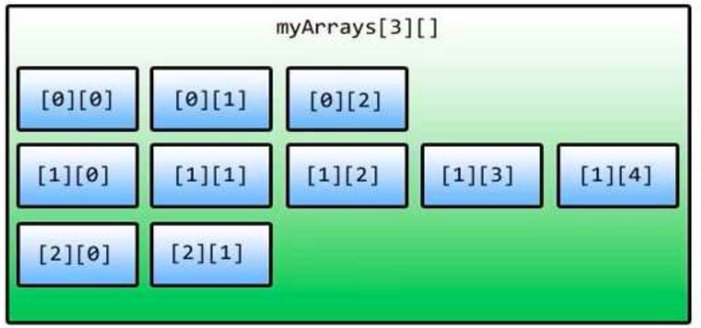


Arrays Classification

- Arrays
  - Single Dimensional Array
  - Multi Dimensional Array
  - Jagged Array



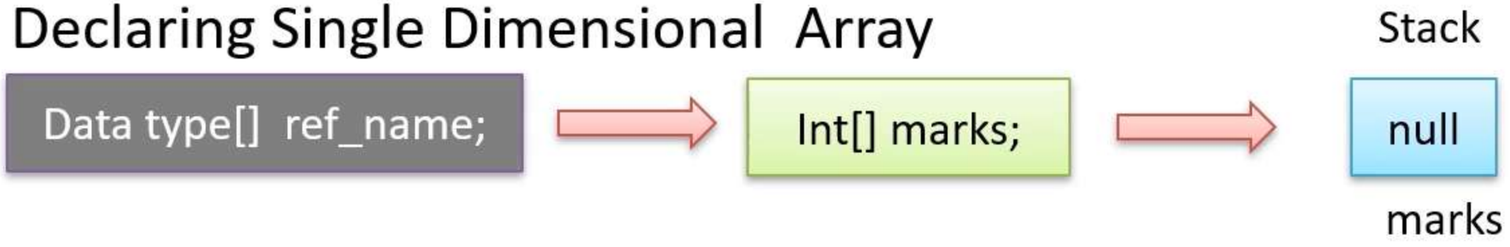




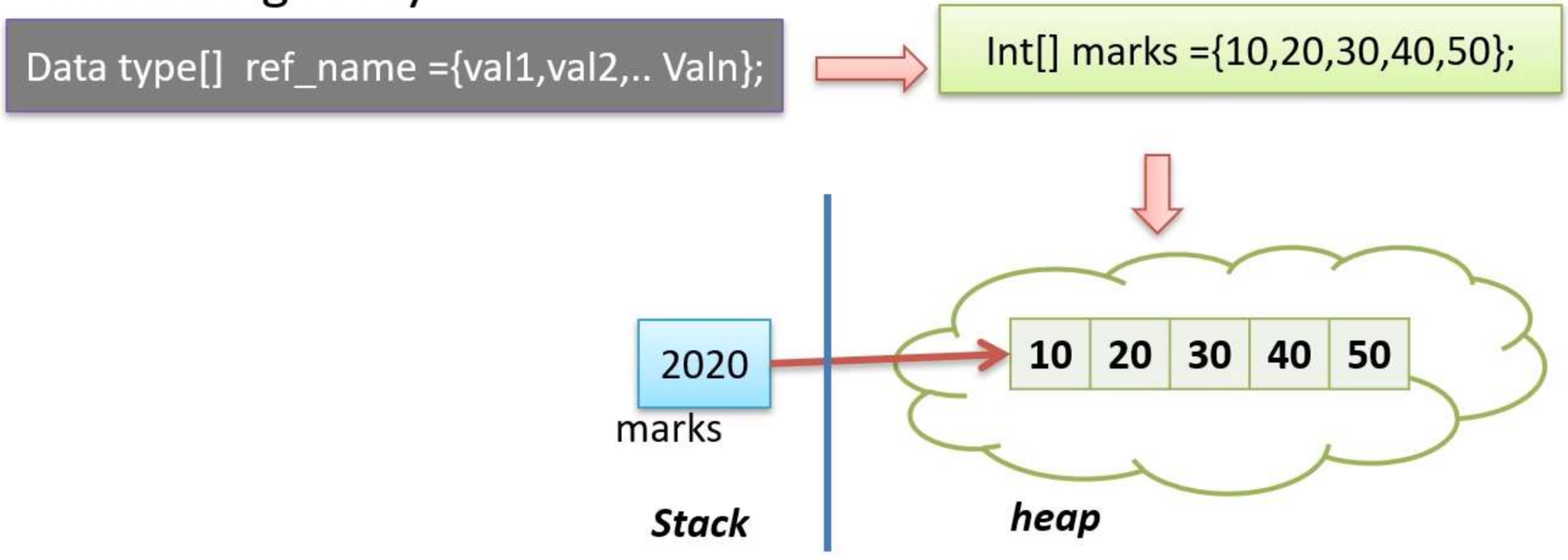
Single Dimensional Array

TEKNOTURF

Declaring Single Dimensional Array



Initializing Array

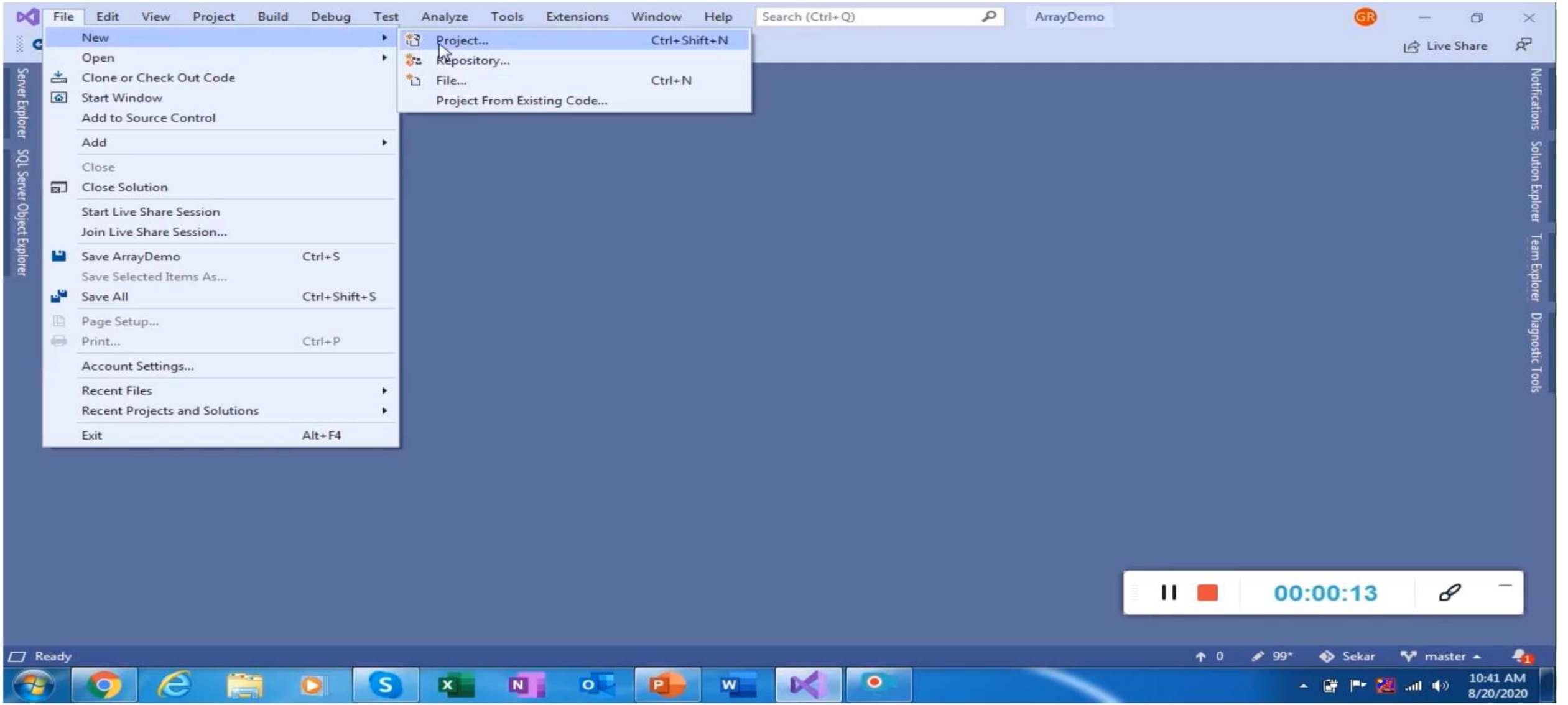


TEKNOTURF

Code Demo

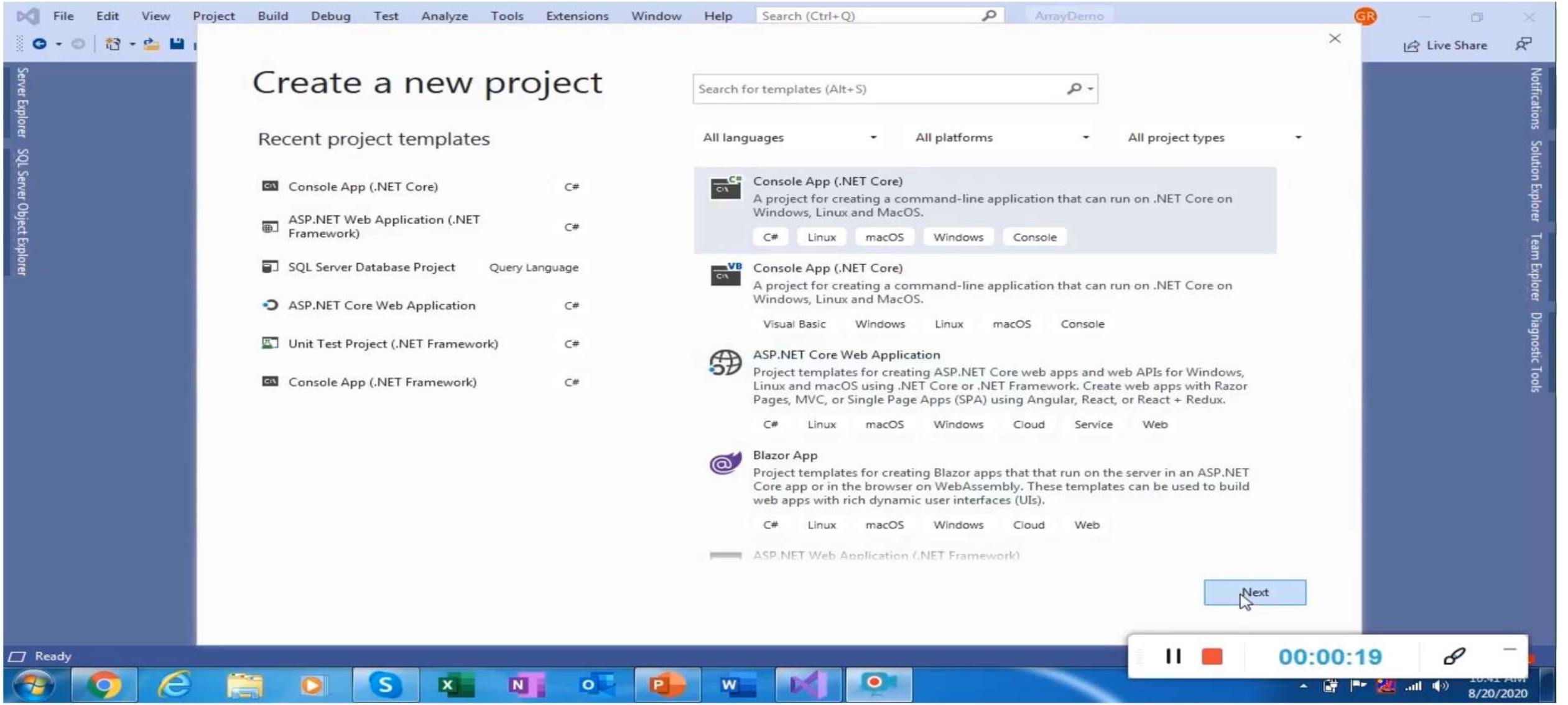
Single Dimensional Array Demo

TEKNOTURF

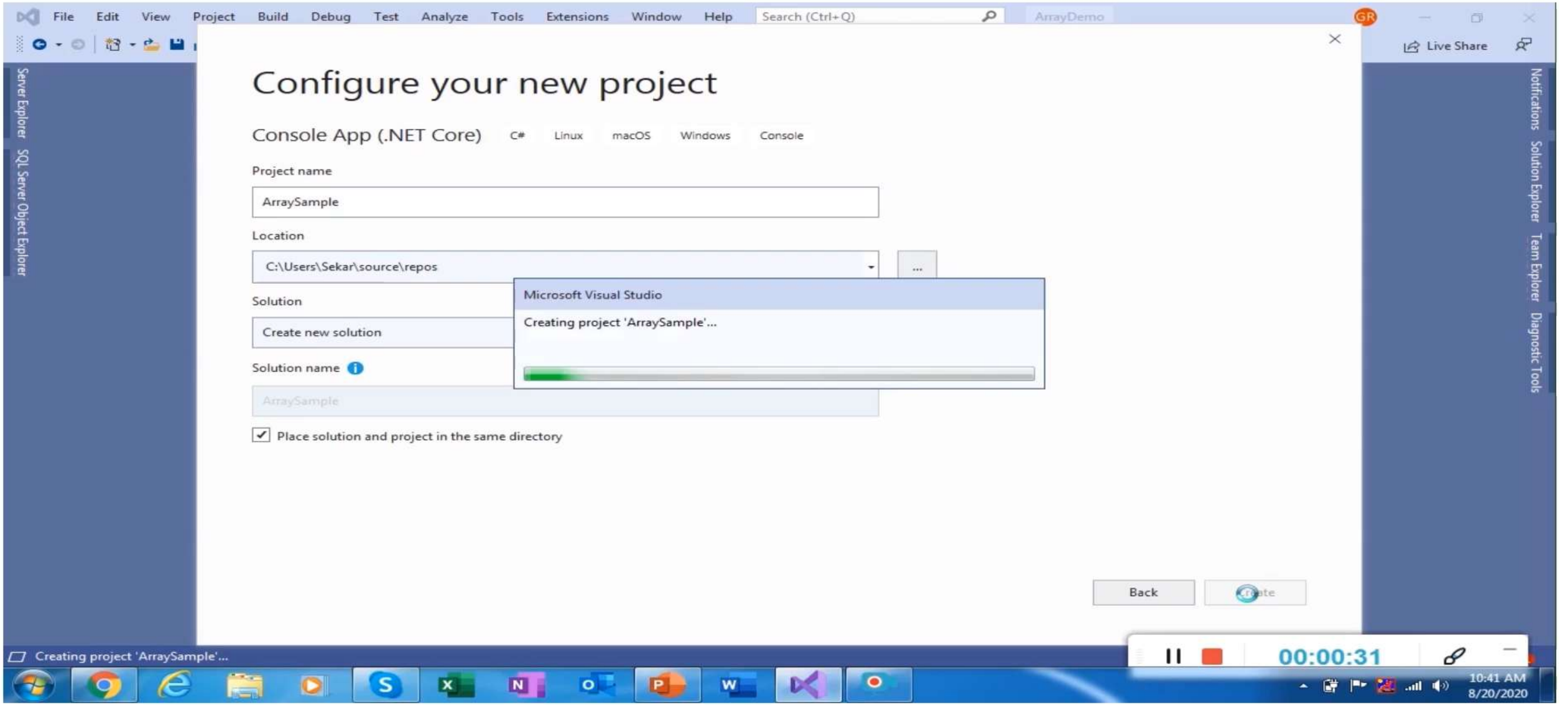


Single Dimensional Array Demo

TEKNOTURF

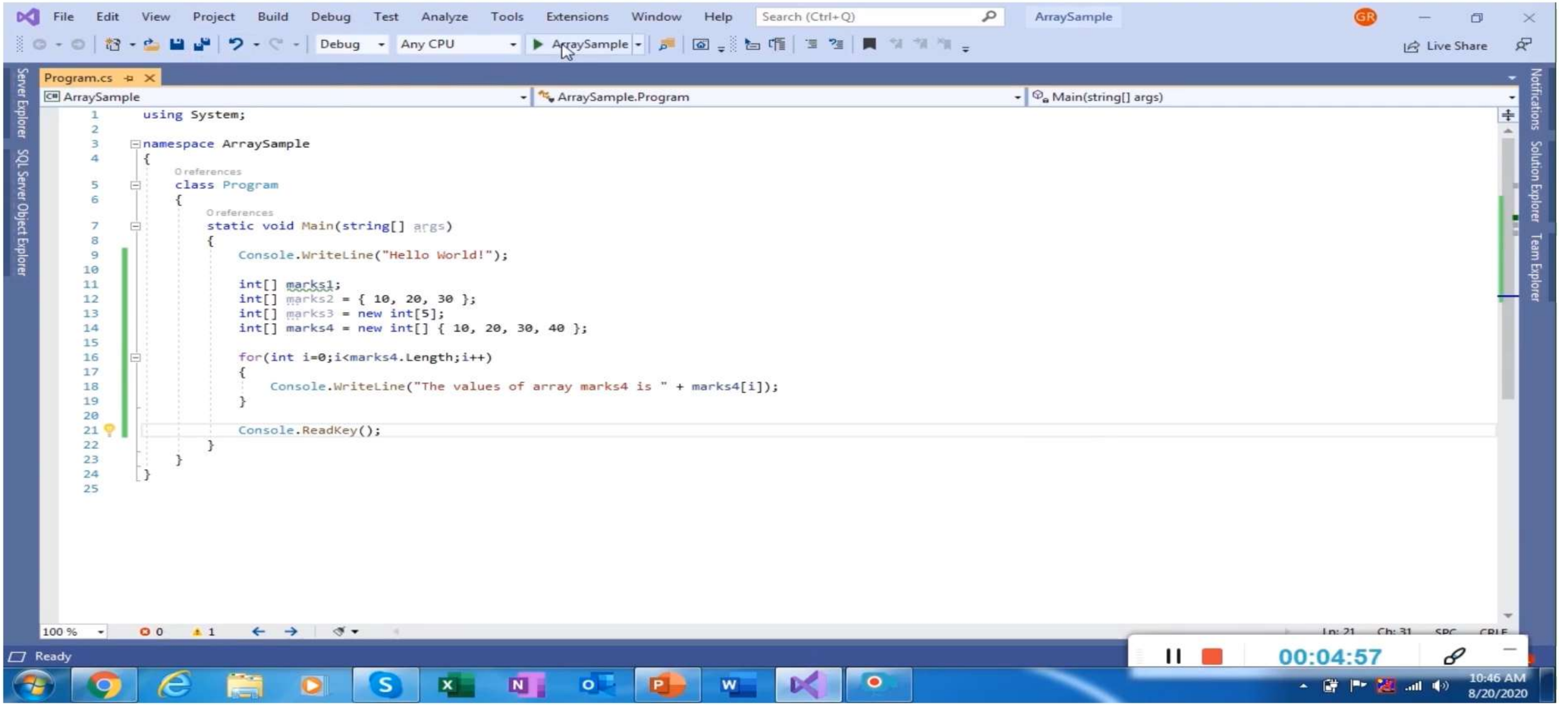


Single Dimensional Array Demo



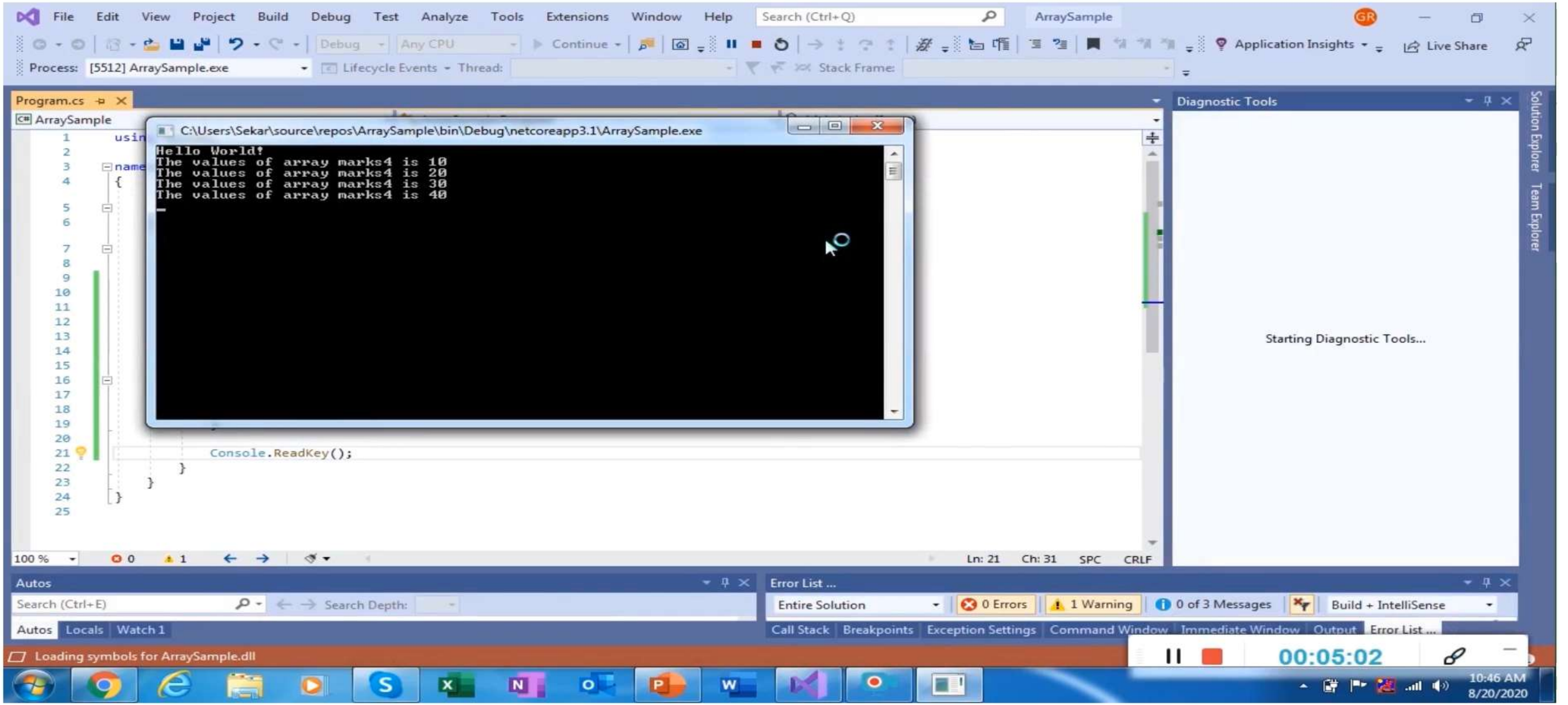
Single Dimensional Array Demo

TEKNOTURF



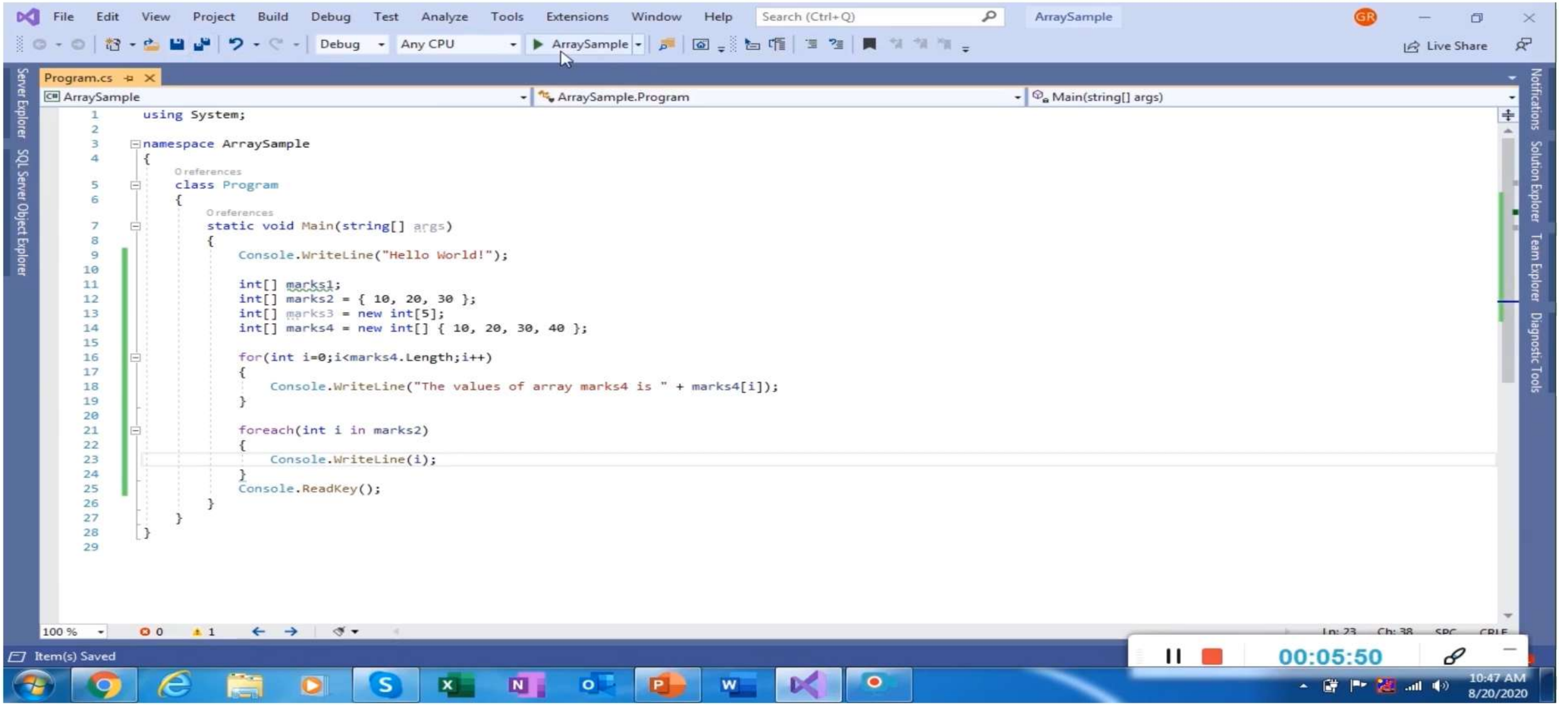
Single Dimensional Array Demo

TEKNOTURF

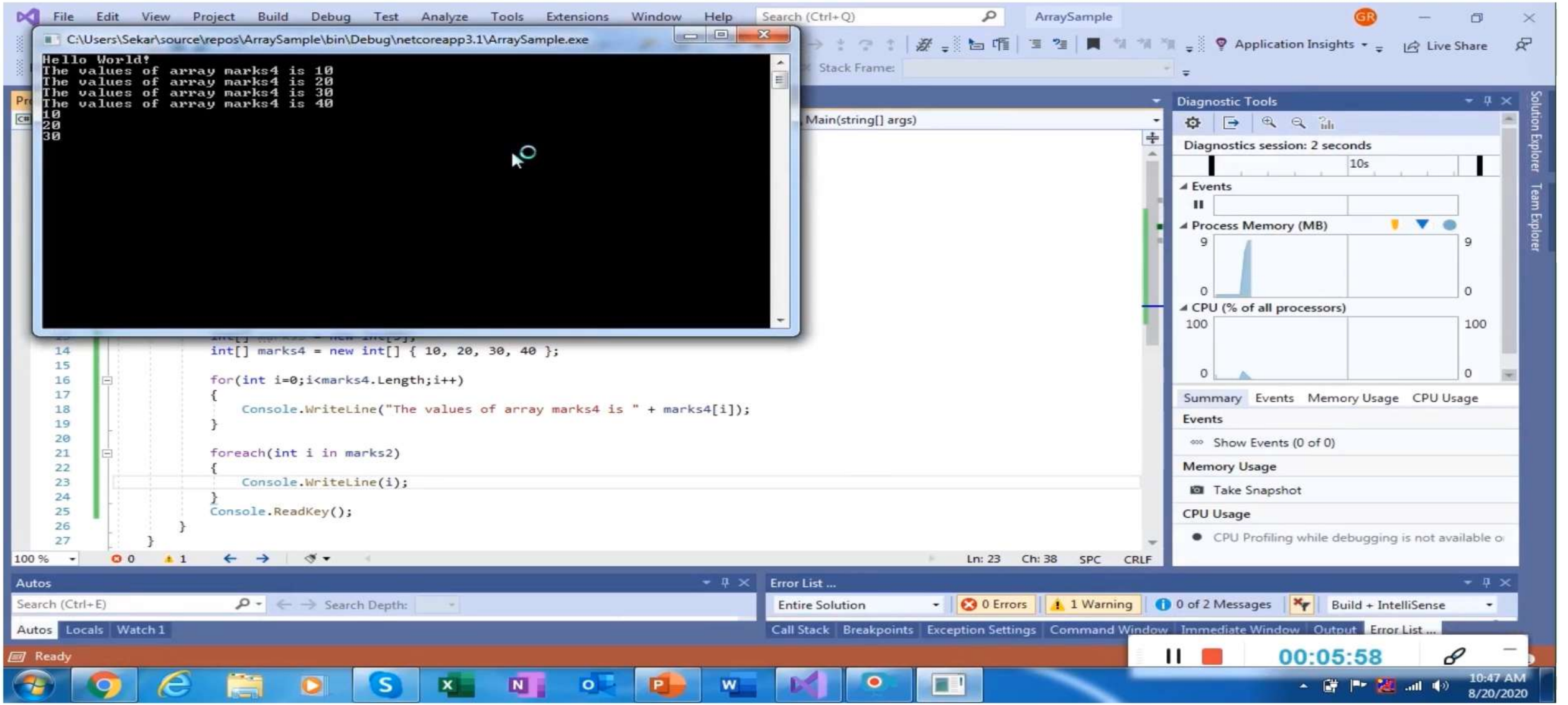


Single Dimensional Array Demo

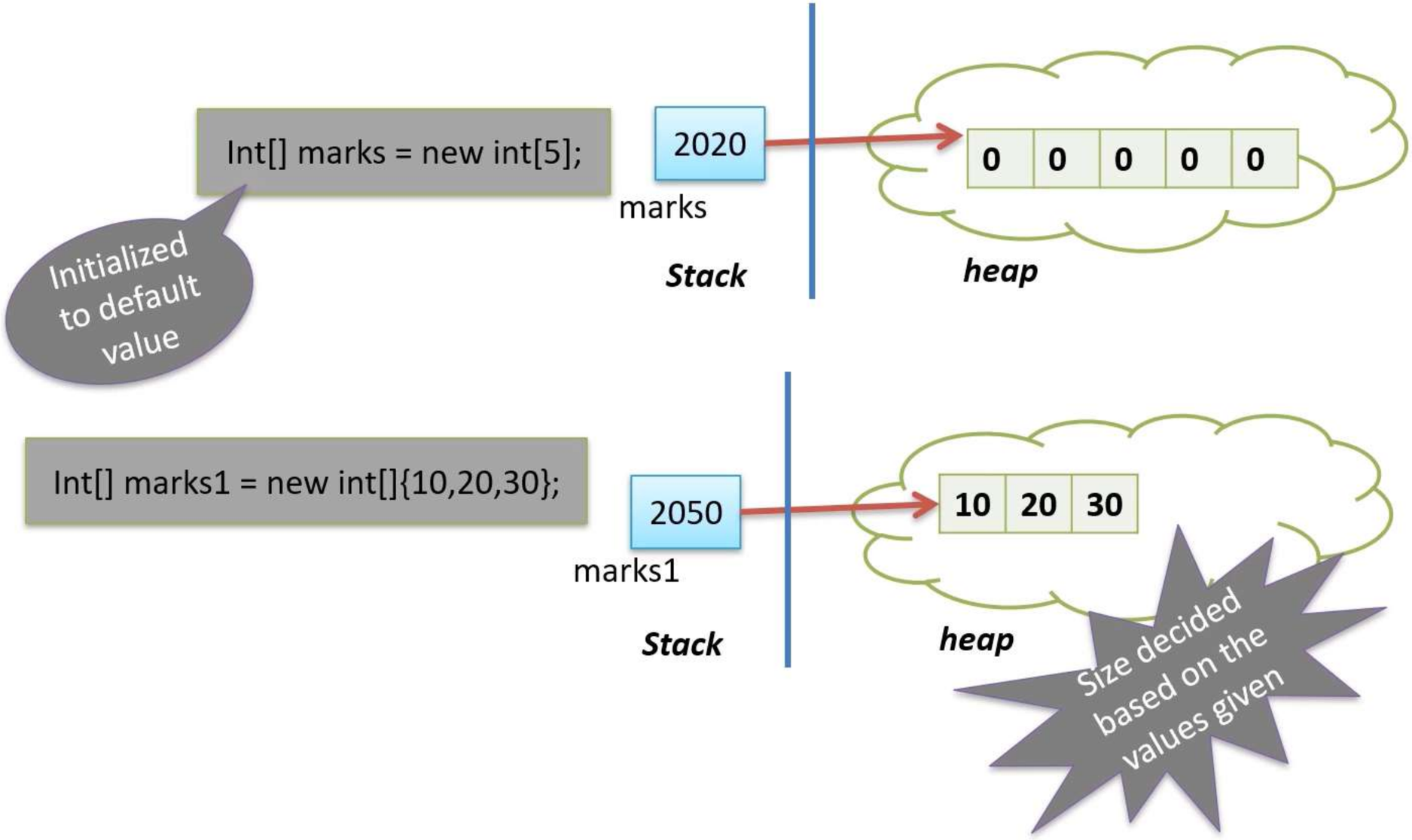
TEKNOTURF



Single Dimensional Array Demo



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

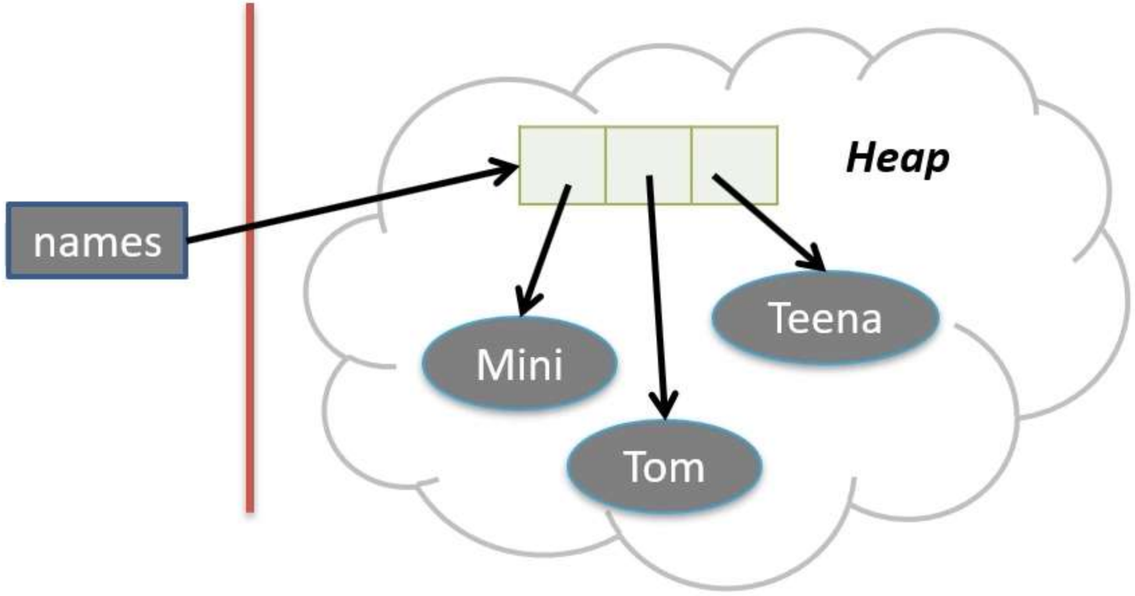
foreach (data type iterator in array variable)

```
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

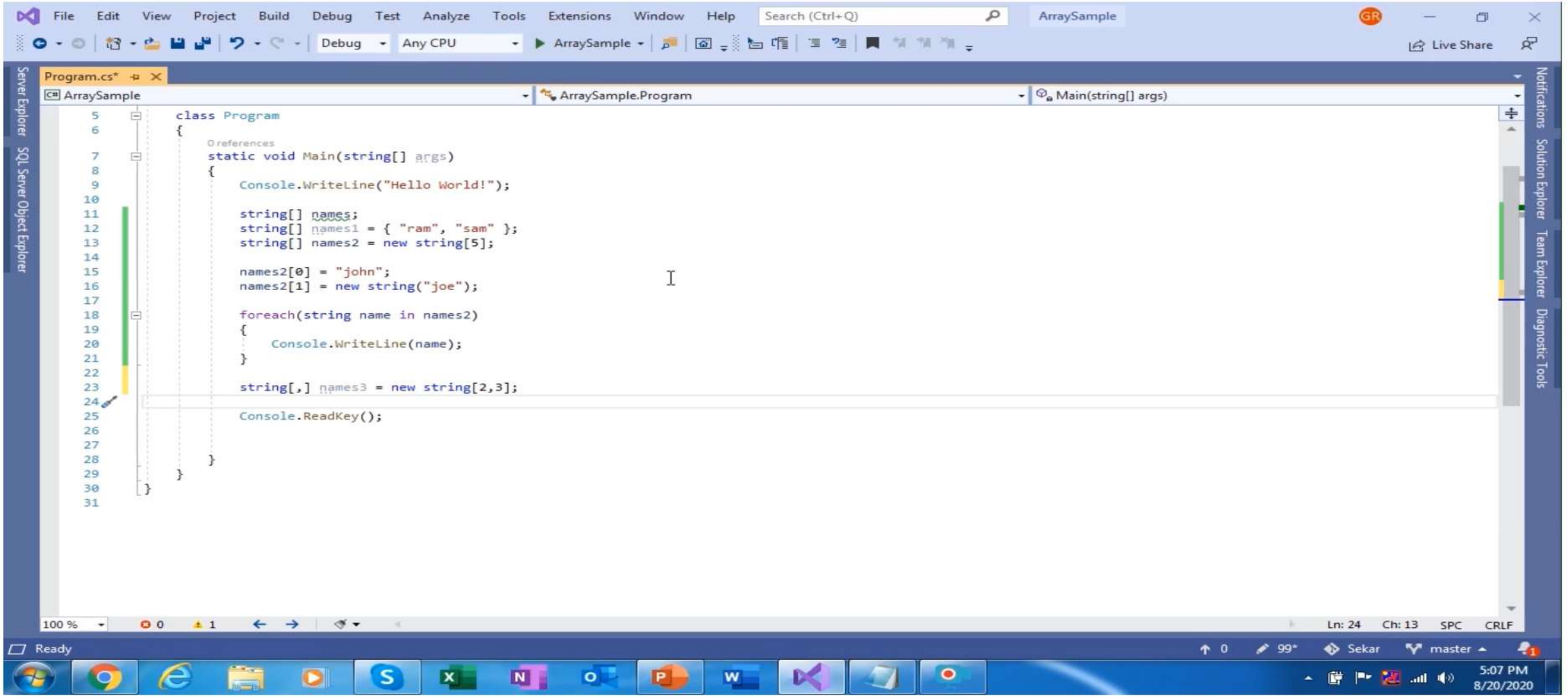


TEKNOTURF

Code Demo

String Array Demo

TEKNOTURF



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

TEKNOTURF

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

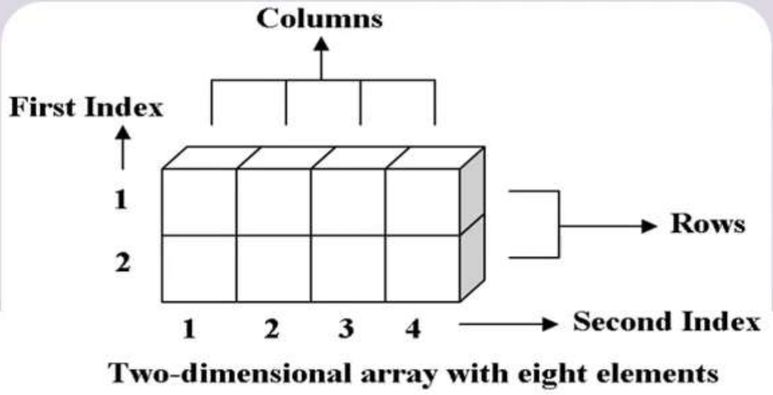
```
<array_name>.Length
```

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

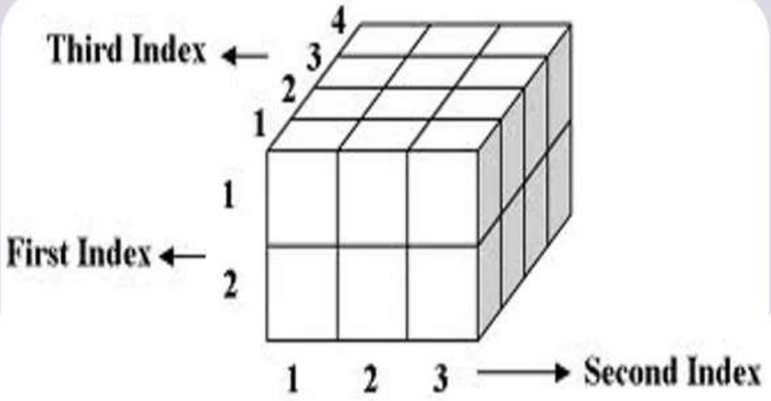
Multidimensional Array

Arrays can have more than one dimension.

A multidimensional array is indexed by two or more integers.



Two-dimensional array with eight elements



Three-dimensional array with twenty four elements

2 D Array

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

3D Array

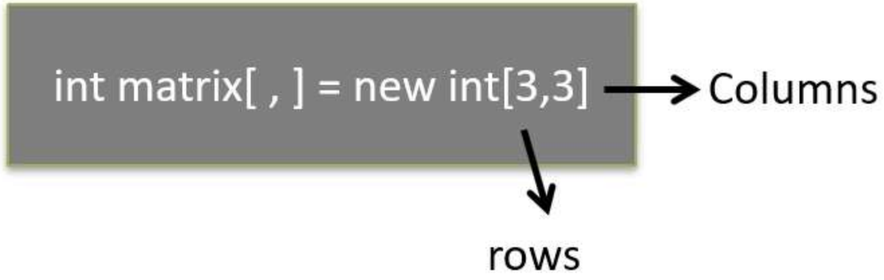
Arrays that have three dimensions

Int[, ,] array3D=new int[2 , 3, 4];

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

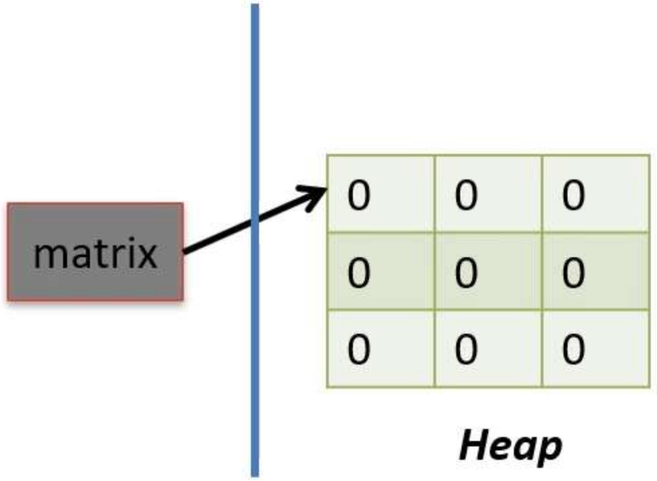
2-D Array

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



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

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



2-D Array

Initializing 2-D Array

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

Columns

rows

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

```
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
```

matrix

10	20	30
40	50	60
70	80	90

Heap

Initialize Arrays at Declaration

TEKNOTURF

- 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]);
  }
}
```

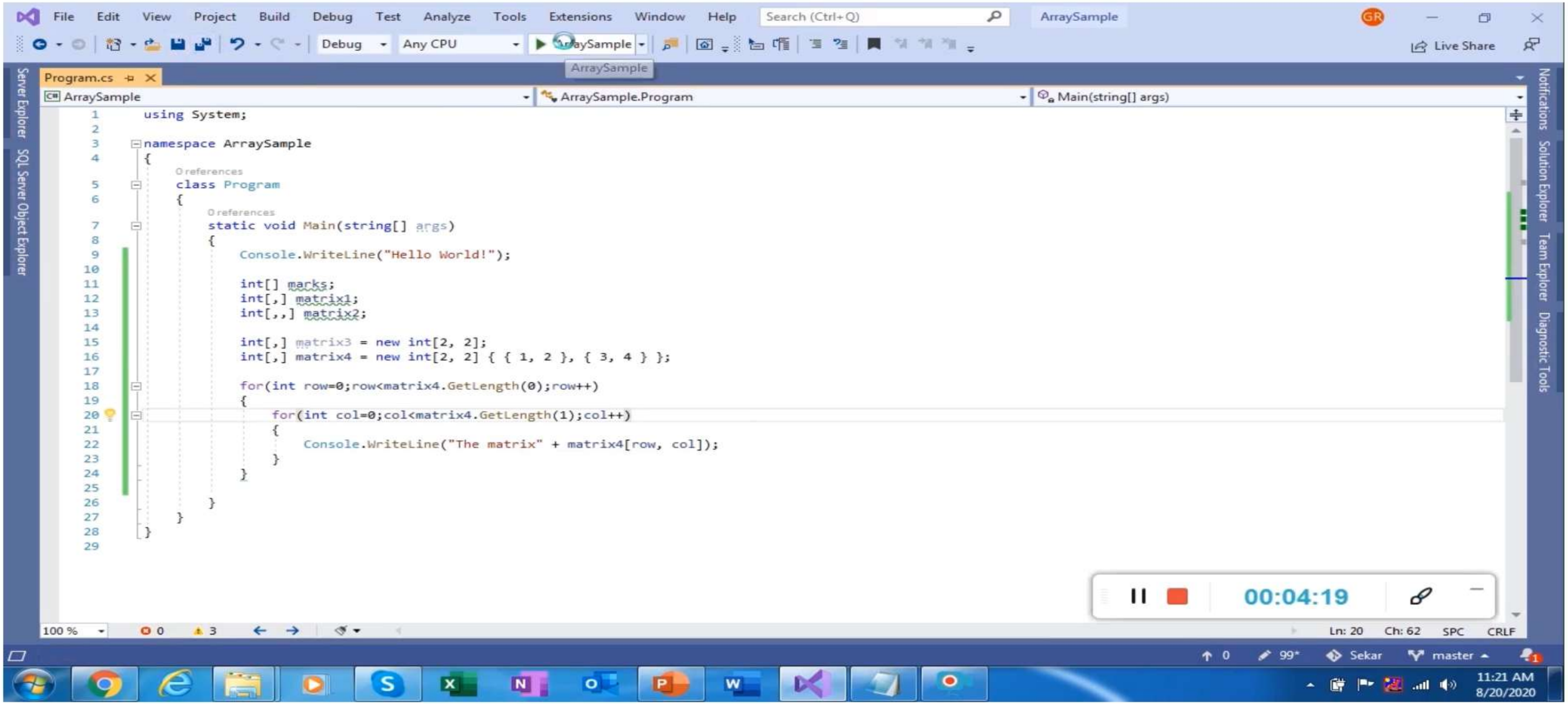
Iterating the elements in a 2D Array using *foreach loop*

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

Code Demo

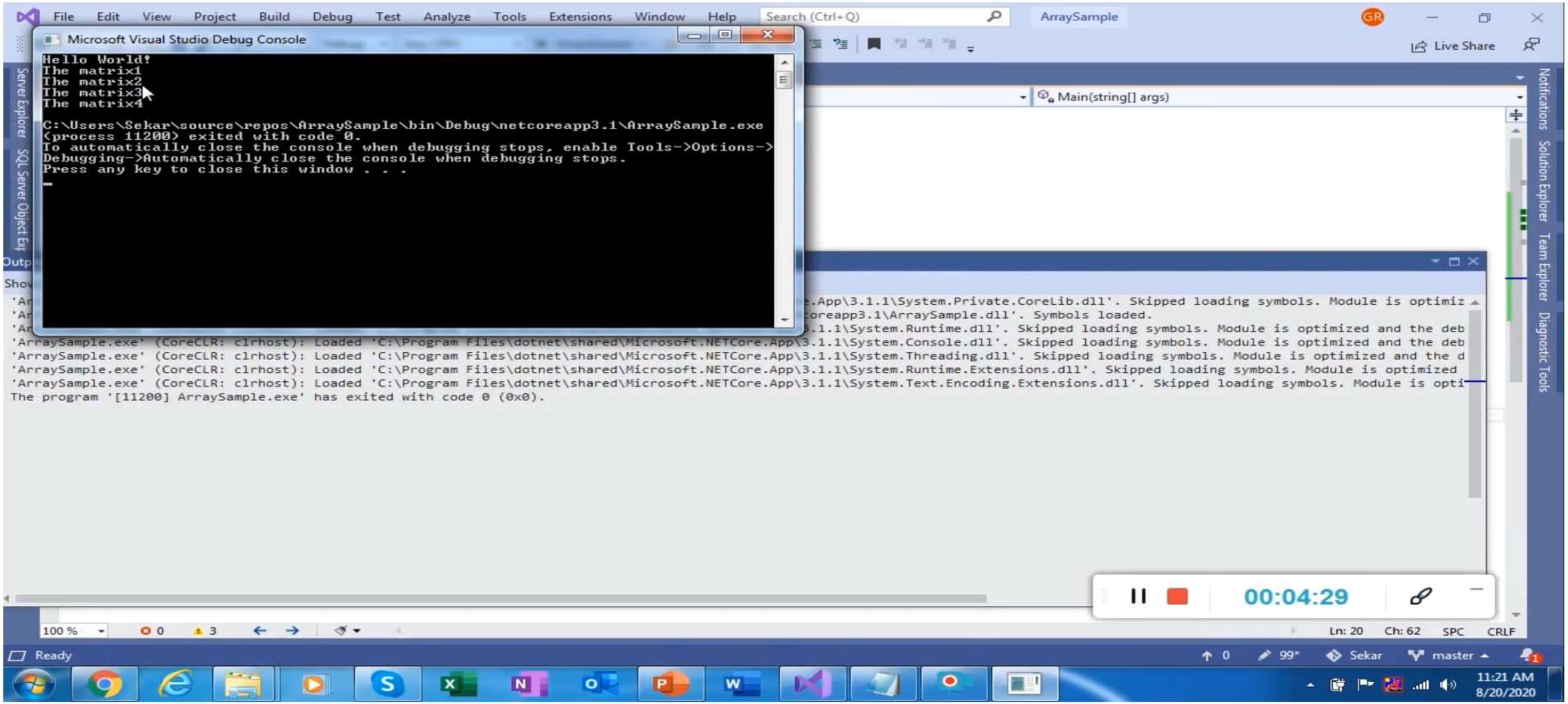
Two Dimensional Array Demo

TEKNOTURF



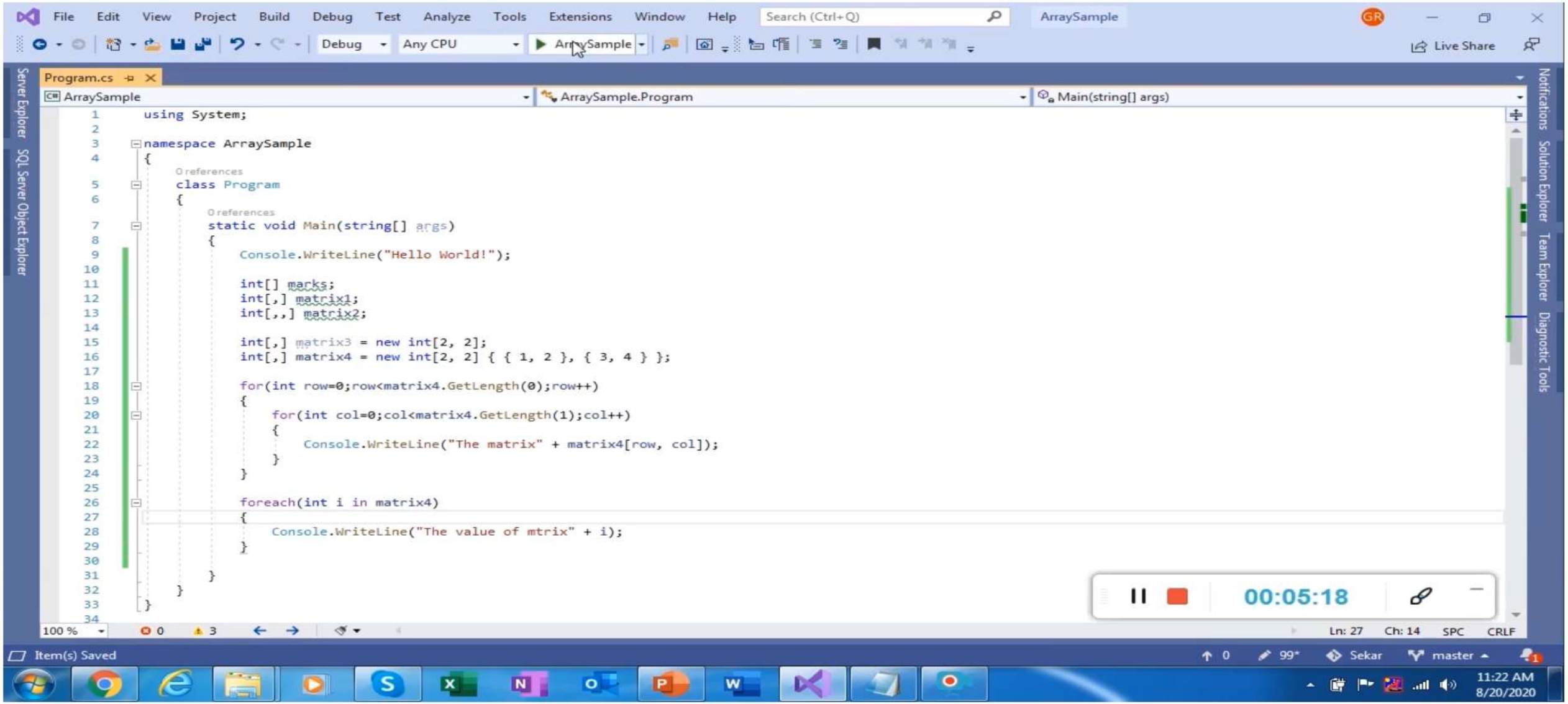
Two Dimensional Array Demo

TEKNOTURF



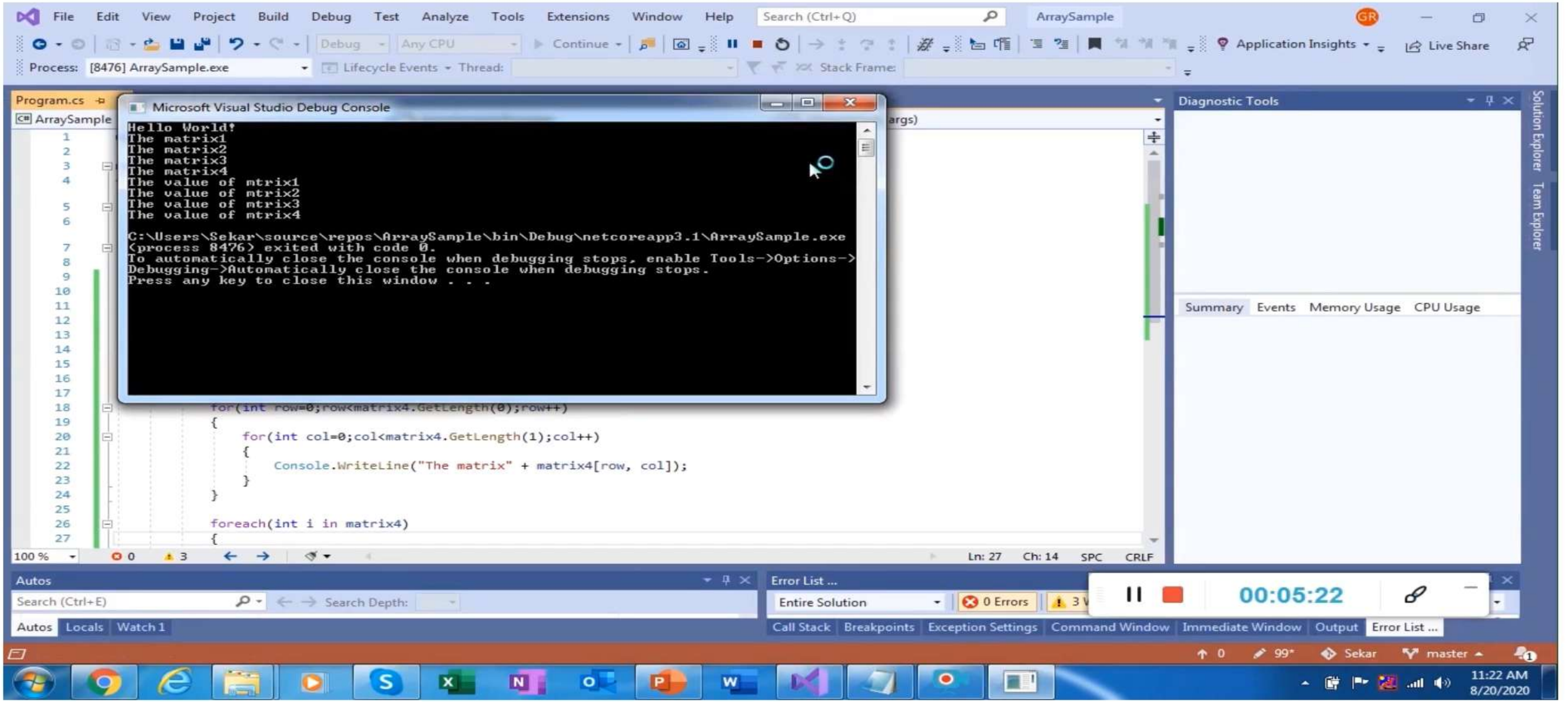
Two Dimensional Array Demo

TEKNOTURF



Two Dimensional Array Demo

TEKNOTURF



Jagged Array

A jagged Array is an array whose elements are arrays.

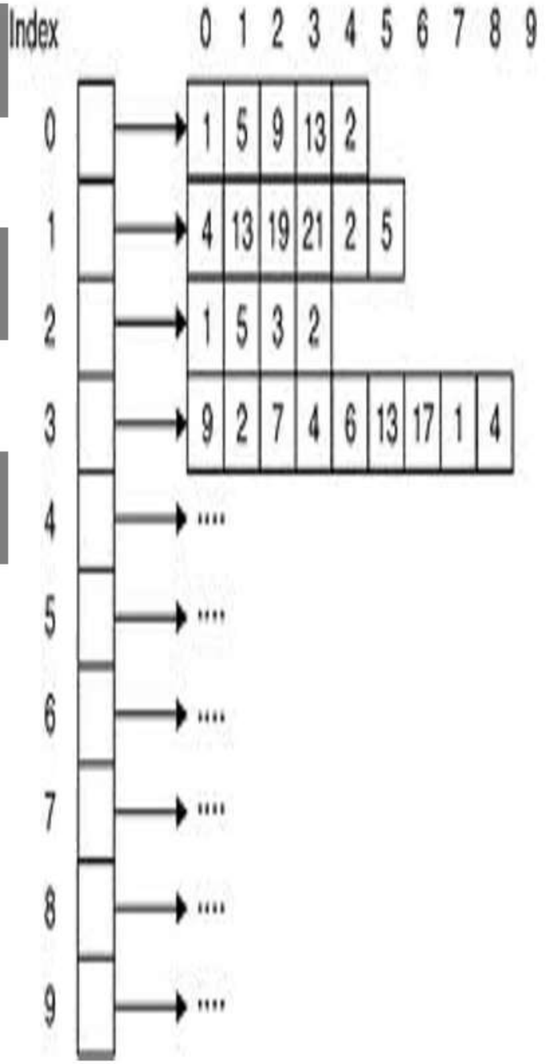
- A Jagged Array is sometimes called an “Array of Arrays."

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

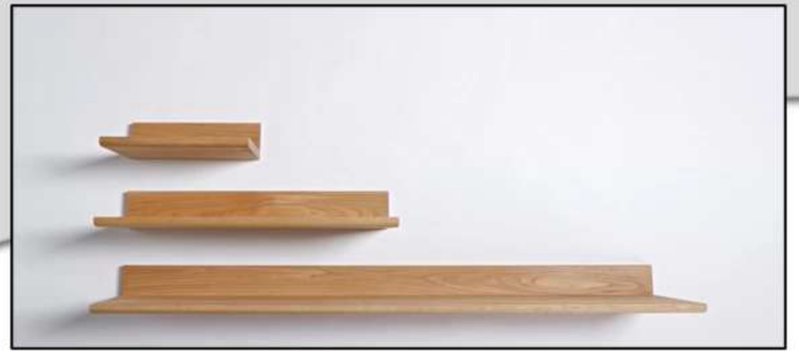
```
string[][] names=new string[3][];

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

TEKNOTURF

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

TEKNOTURF

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