MASTER OF COMPUTER APPLICATIONS (MCA)

PG Department of Computer Science and Technology

Sardar Patel University

Vallabh Vidyanagar, Gujarat, INDIA

PS02CMCA54 [The .NET Framework]

MR. BHARAT B. PATEL (ASSOCIATE PROFESSOR)

(✉) bb_patel@spuvvn.edu

C#.NET - Introduction

- C# (pronounced "See Sharp") is a modern, object-oriented, and type-safe programming language.
- C# enables developers to build many types of secure and robust applications that run in .NET.
- C# has its roots in the C family of languages and will be immediately familiar to C, C++, Java, and JavaScript programmers.
- C# is a general purpose object oriented programming language with multiple paradigms.
- It was designed for Common Language Infrastructure (CLI) in 2000 by Microsoft.

C#.NET - Introduction

- **C# is an object-oriented, component-oriented programming language.**
- **C# provides language constructs to directly support these concepts, making C# a natural language in which to create and use software components.**

Features of C#.NET

Object Oriented Language

- C# is an object oriented language. This means that development is easier if there are long codes in large projects as compared to procedure oriented programming languages.

Structured programming language

- The programs in C# can be structured by breaking them into smaller parts using functions. This makes the program easier to understand.

Simple to use

- It is quite simple to use C# as it has various features and functionalities. Moreover, it provides a structured approach that makes the program easier to understand.

Scalable language

- C# is an automatically scalable as well as updatable language. The old files are regularly updated and replaced with new ones.

Types of Applications supported by C#.NET

- Console Applications
- Window applications
- Web applications
- Web services and Web API
- Database applications
- Distributed applications
- Cloud native apps and services
- Windows client applications
- Windows libraries and components
- Windows services
- Native iOS and Android mobile apps.
- Backend services

General structure of C# Program

- **C# programs can consist of one or more files. Each file can contain zero or more namespaces.**
- **A namespace can contain types such as classes, structs, interfaces, enumerations, and delegates, in addition to other namespaces.**
- **The following is the skeleton of a C# program that contains all of these elements.**

General structure of C# Program

```
// A skeleton of a C# program
using System;
namespace YourNamespace
{
    class YourClass
    {
    }

    struct YourStruct
    {
    }

    interface IYourInterface
    {
    }

    delegate int YourDelegate();

    enum YourEnum
    {
    }

    namespace YourNestedNamespace
    {
        struct YourStruct
        {
        }
    }

    class YourMainClass
    {
        static void Main(string[] args)
        {
            //Your program starts here...
        }
    }
}
```

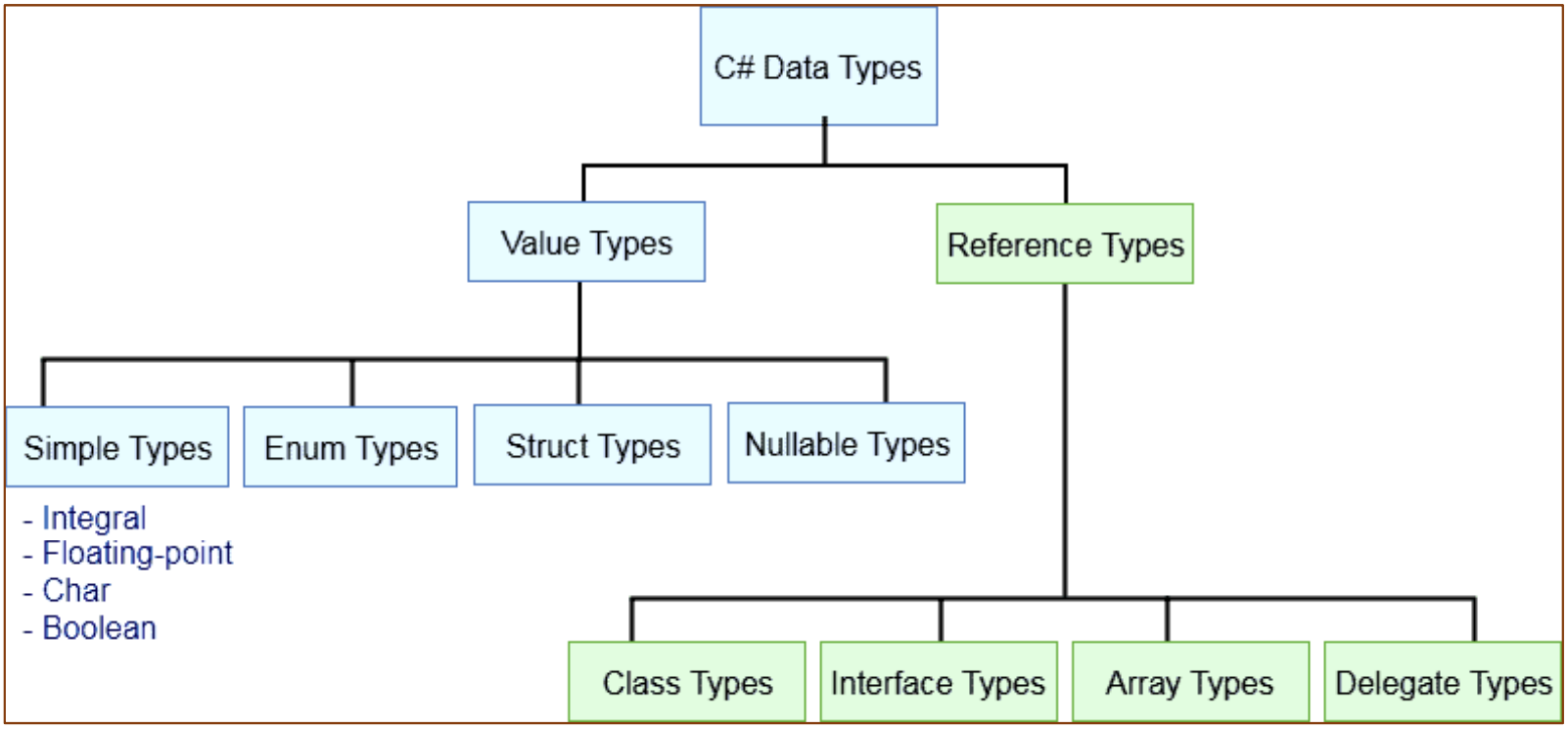
C#.NET Basic Data Types

- C# is a strongly-typed language. It means we must declare the type of a variable that indicates the kind of values it is going to store, such as integer, float, decimal, text, etc
- C# mainly categorized data types in two types

  [1] Value types and

  [2] Reference types
- Value types include simple types (such as int, float, bool, and char), enum types, struct types, and Nullable value types.
- Reference types include class types, interface types, delegate types, and array types.

C#.NET Basic Data Types



Default values

- **Every data type has a default value. Following table shows default value for different data type.**

Type	Default value
Any reference type	null
Any built-in integral numeric type	0 (zero)
Any built-in floating-point numeric type	0 (zero)
bool	false
char	'\0'

Variables in C#.NET

- **Variables are essentially locations in computer memory that are reserved for storing the data used by an application.**
- **Each variable is given a name by the programmer and assigned a value.**
- **The name assigned to the variable may then be used in the C# code to access the value assigned to the variable.**
- **This access can involve either reading the value of the variable, or changing the value.**
- **A variable must be declared as a particular type such as an integer, a character or a string.**

Variables in C#.NET

- Constants are immutable values which are known at compile time and do not change for the life of the program.
- Constants are declared with the const modifier.
- C# is what is known as a strongly typed language in that once a variable has been declared as a particular type it cannot subsequently be changed to a different type.
- Variable declarations require a type, a name and, optionally a value assignment.
- The following example declares an integer variable called opr but does not initialize it:

```
char opr;
```

Variables in C#.NET

- **The following example declares and initializes a variable using the assignment operator (=) :**

```
char opr= '+'
```

Constants in C#.NET

- **Constants are immutable values which are known at compile time and do not change for the life of the program.**
- **Constants are declared with the const modifier.**
- **A constant is similar to a variable in that it provides a named location in memory to store a data value.**
- **Constants differ in one significant way in that once a value has been assigned to a constant it cannot subsequently be changed. Constants are particularly useful if there is a value which is used repeatedly throughout the application code.**

Constants in C#.NET

- As with variables, constants have a type, a name and a value.
- Unlike variables, constants must be initialized at the same time that they are declared and must be prefixed with the const keyword:

```
const float pi = 3.14;
```

Type conversion

- Type conversion is converting one type of data to another type.
- It is also known as Type Casting.
- In C#, type casting has two forms :
  - **Implicit type conversion** − These conversions are performed by C# in a type-safe manner. For example, are conversions from smaller to larger integral types and conversions from derived classes to base classes.
  - **Explicit type conversion** − These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.

Boxing in C#.NET

- **Boxing is the process of converting a value type to the type object or to any interface type implemented by this value type.**
- **When the common language runtime (CLR) boxes a value type, it wraps the value inside a System.Object instance and stores it on the managed heap.**
- **Boxing is implicit.**
- **For example,**

```
int i = 123;
// The following line boxes i.
object o = i;
```

Unboxing in C#.NET

- **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.**
- **Unboxing is explicit.**
- **For example,**

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

Dialog-box controls in C#.NET

Name of the the Dialog-Box Control	Purpose
ColorDialog	Enables the user to select a color from a palette in a pre-configured dialog box and to add custom colors to that palette.
FolderBrowserDialog	Enables users to browse and select folders.
FontDialog	Exposes the fonts that are currently installed on the system.
OpenFileDialog	Allows users to open files via a pre-configured dialog box.
PageSetupDialog	Sets page details for printing via a pre-configured dialog box.
PrintDialog	Selects a printer, chooses the pages to print, and determines other print-related settings.
PrintPreviewDialog	Displays a document as it will appear when it is printed.
SaveFileDialog	Selects files to save and where to save them.