C# PROGRAMMING

Afzal Pasha
.NET Evangelist
Bangalore
7760212484

C# PROGRAMMING SYNTAX

- C# syntax looks quite similar to the syntax of Java because both inherit much of their syntax from C and C++.

C# PROGRAMMING SYNTAX

```
using System;
using System.Collections.Generic;
using System.Text;

namespace FirstProgram
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Microsoft  .NET Training");
      }
   }
}
```

C# PROGRAM COMMENTS

```
using System;
using System.Collections.Generic;
using System.Text;

namespace FirstProgram
{
   class Program
   {
      static void Main(string[] args)
      {
         //single Line Comments
          /* Multi Line Comments */
          /// XML Comments
       }
    }
}
```

THE CONSOLE CLASS

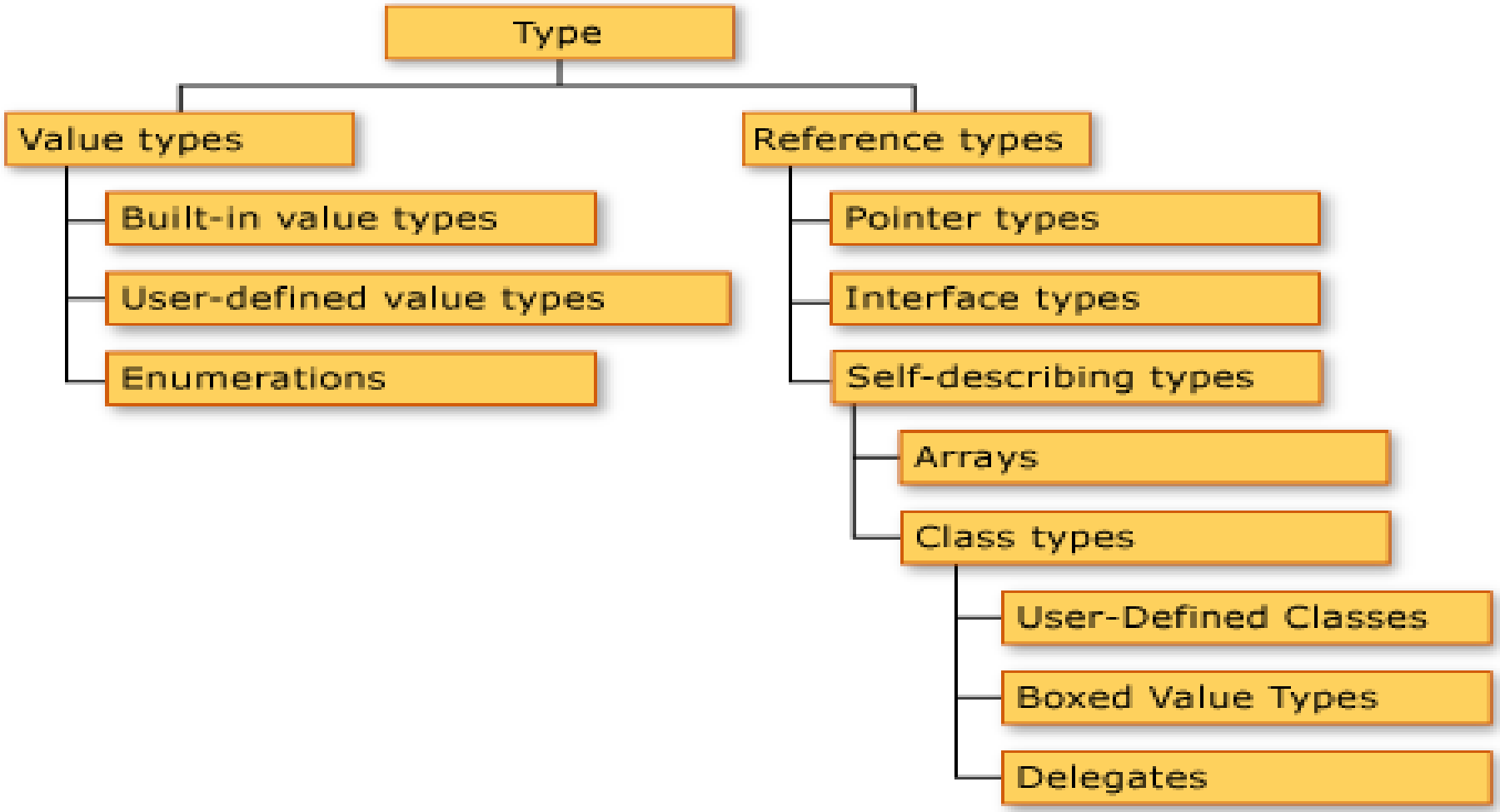
- Provides access to the standard input, standard output, and standard error streams
- Only meaningful for console applications
  - Standard input – keyboard
  - Standard output – screen
  - Standard error – screen

THE CONSOLE CLASS

Console Class Methods

- Console.Write and Console.WriteLine display information on the console screen
- Console.Read and Console.ReadLine read user input

DATA TYPES



DATA TYPES

Value Types

bool , byte, char
decimal, double, float,
int, long
sbyte, short
uint, ulong, ushort
enum, struct

DATA TYPES

Refernce Types

class, delegate
Interface, object, string

DATA TYPES CONVERTION

- Type conversion is a process of converting one type into another.

- Two types of conversions
  - Boxing
  - UnBoxing

DATA TYPES CONVERTION

Boxing

- The conversion of value type to reference type is known as boxing
- Boxing is an implicit conversion of a value type to the type Object.

DATA TYPES CONVERTION

UnBoxing

- converting reference type back to the value type is known as Unboxing.
- Unboxing is an explicit conversion from the type object to a value type
- The casting operator () is necessary for un-boxing.

DATA TYPES CONVERTION

```
class Program
  {
     static void Main(string[] args)
     {
        int x = 100;  //ValueType
        int y = 200; //ValueType

        object obj = new object(); //ReferenceType

        obj = x; //Boxing
        y = (int)obj; //UnBoxing

        Console.WriteLine("{0}, {1}", obj, y);
      }
}
```

COMMON OPERATORS

Common Operators	Example
• Equality operators	== !=
• Relational operators	< > <= >=
• Conditional operators	&& || ?:
• Increment operator	++
• Decrement operator	--
• Arithmetic operators	+ - * / %
• Assignment operators	= *= /= %= += -= <<= >>= &= ^= |=

STATEMENTS

Selection Statements

The if and switch statements

Iteration Statements

The while, do, for, and foreach statements

Jump Statements

The goto, break, and continue statements

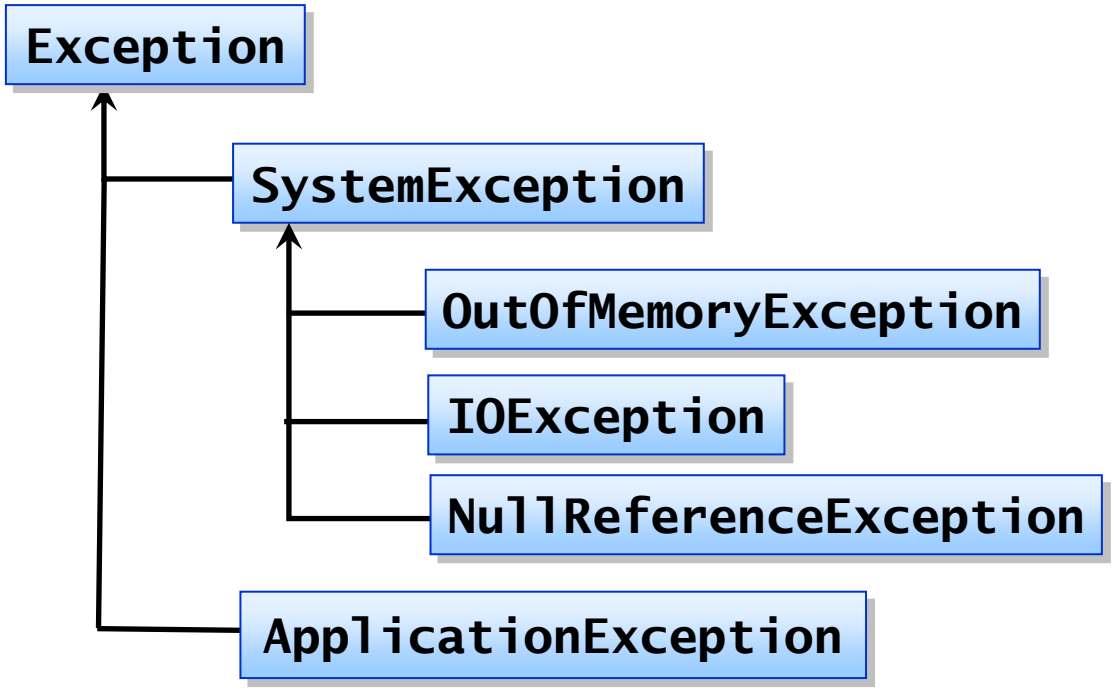
FOR EACH

Execute embedded statements for each element of the collection class

```
List<string> Names = new List<string>();
        Names.Add("Afzal");
        Names.Add("Ramya");
        Names.Add("Ravi");

    foreach (string str in Names)
    {
       Console.WriteLine(str);
    }
```

EXCEPTION OBJECTS



USING TRY AND CATCH BLOCKS

- Object-oriented solution to error handling
  - Put the normal code in a **try** block
  - Handle the exceptions in a separate **catch** block

```
try {
        Console.WriteLine("Enter a number");
        int i = int.Parse(Console.ReadLine());
}
catch (OverflowException Err)
{
        Console.WriteLine(Err);
}
```

MULTIPLE CATCH BLOCKS

- Each catch block catches one class of exception
- A try block can have one general catch block
- A try block is not allowed to catch a class that is derived from a class Err in an earlier catch block

```
try
{
        Console.WriteLine("Enter first number");
        int i = int.Parse(Console.ReadLine());
        Console.WriteLine("Enter second number");
        int j = int.Parse(Console.ReadLine());
        int k = i / j;
}
catch (OverflowException Err) {…}
catch (DivideByZeroException Err) {…}
```

THE FINALLY CLAUSE

- Throw an appropriate exception
- Give the exception a meaningful message

```
Monitor.Enter(x);
try {
    ...
}
finally {
    Monitor.Exit(x);
}
```

THE THROW STATEMENT

```
Console.WriteLine("Please enter Number");
        int x = Convert.ToInt32(Console.ReadLine());
       try
        {
           if (x < 10)
           {
              throw new Exception("Number is Lessthan 10");
           }
           else
           {              Console.WriteLine(x);          }
        }
        catch(Exception Ex)
        {
           Console.WriteLine(Ex);
        }
```

NULLABLE TYPES

NULLABLE TYPES

```
int? a = null;
int? b=null;
int? c = a / b;
Console.WriteLine(c.ToString());
```

THANK YOU