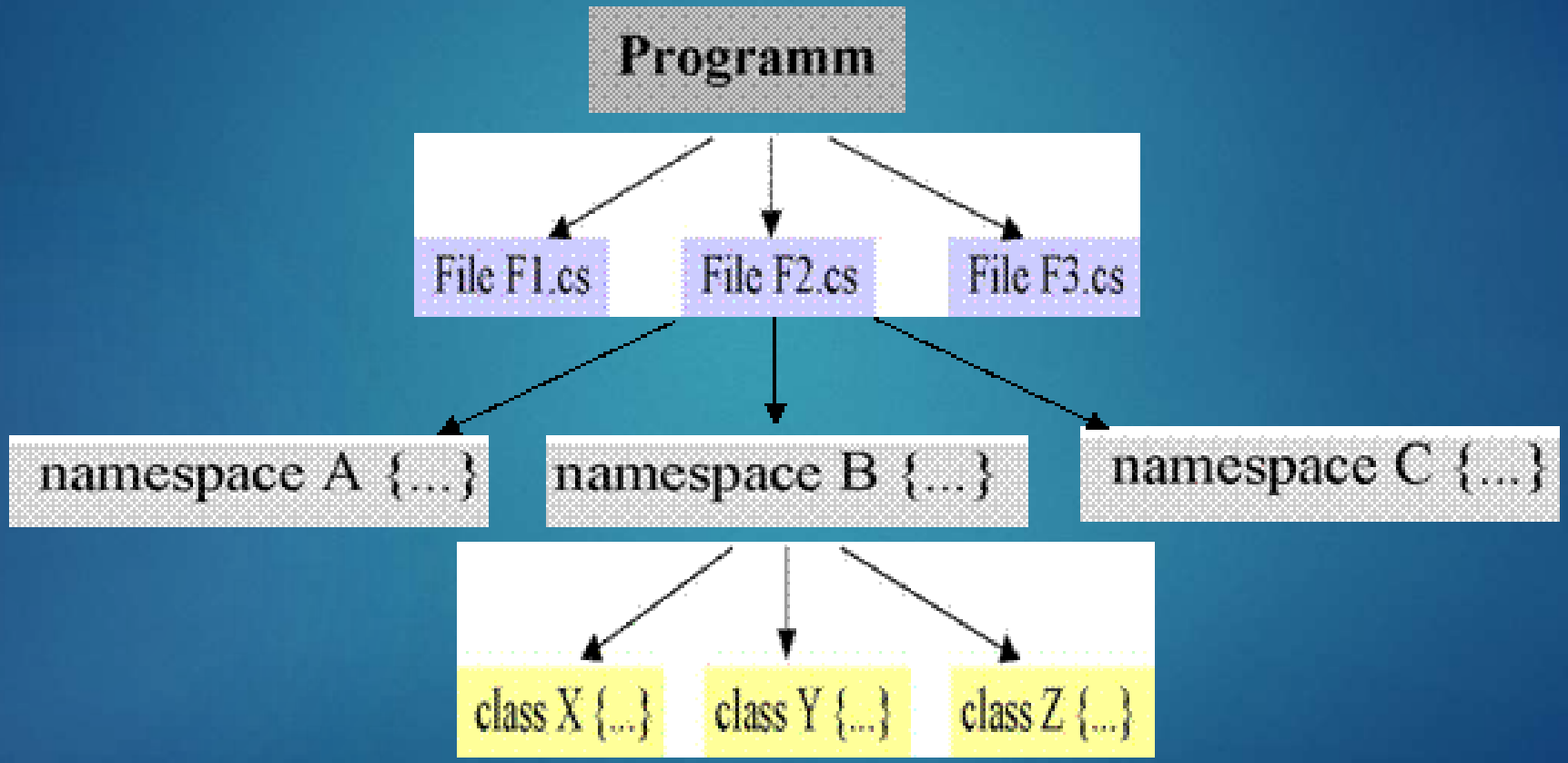
Lecture 1

FUNDAMENTALS OF C# LANGUAGE

Structure of C# Programs



First C# program

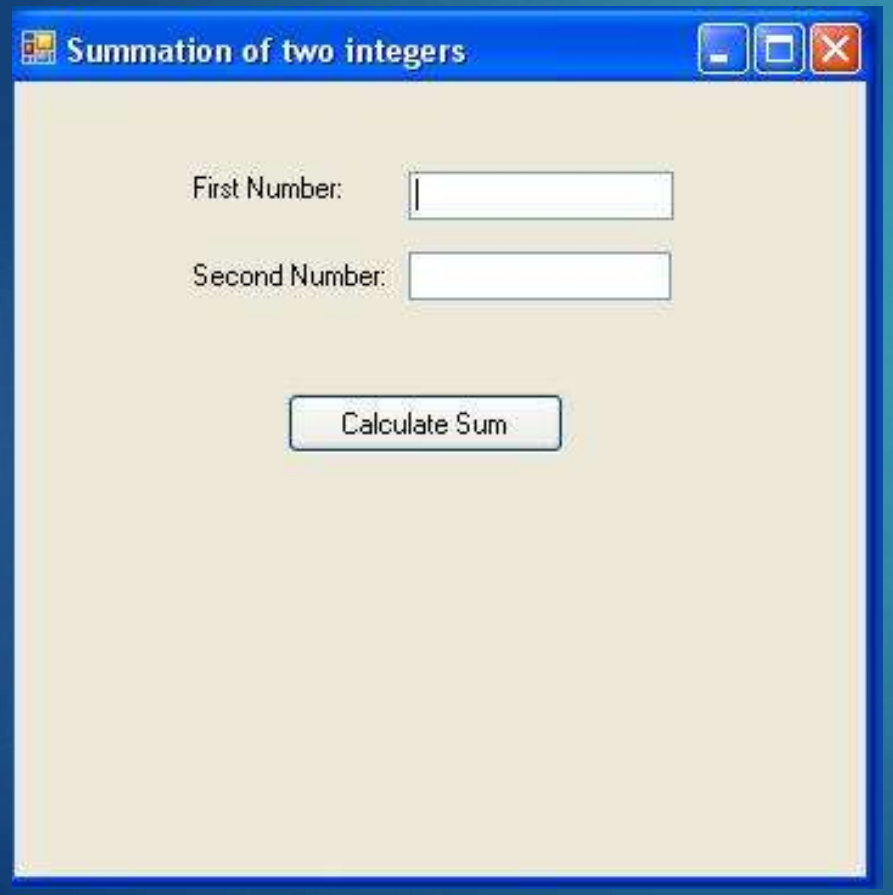
Summation of two integers as Console Application

- uses the namespace System
- entry point must be called Main
- output goes to the console
- file name and class name need not to be identical.

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

namespace TwoIntegersSum
{
    class Program
    {
        static void Main(string[] args)
        {
            int x, y;
            Console.Write("Enter the first integer: ");
            x = Int32.Parse( Console.ReadLine());
            Console.Write("\nEnter the second integer: ");
            y = Int32.Parse(Console.ReadLine());
            Console.WriteLine("\n\nSum of {0} and {1} is: {2}\n", x, y, x + y);
        }
    }
}
```

C# windows application

Summation of two integers as a windows application

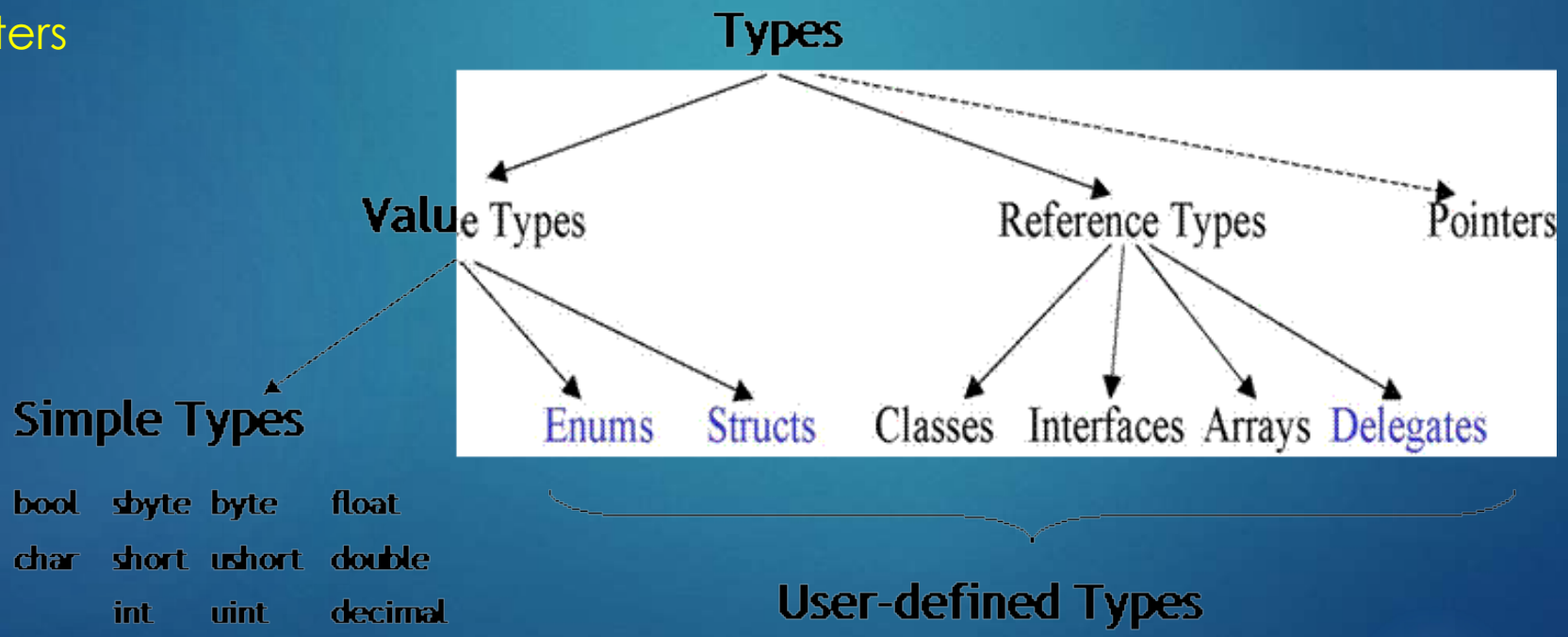



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace TwoIntegersSum
{
    public partial class twoIntegersSum : Form
    {
        public twoIntegersSum()
        {
            InitializeComponent();
        }

        private void sumButton_Click(object sender, EventArgs e)
        {
            int x, y, sum;
            x = Int32.Parse(firstTextBox.Text );
            y = Int32.Parse(secondTextBox.Text);
            sum = x + y;
            resultLabel.Text = "Summation is: " + sum;
        }
    }
}
```

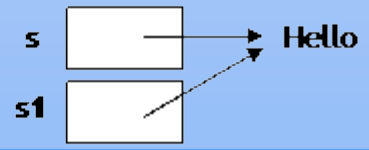
Types

Unified Type System

- Value types: simple types (bool, char, byte, short, int, long, uint, ulong, float, double, decimal), Enums, Structs;
- Reference types: Classes, Interfaces, Arrays, Delegates
- Pointers



Value Types versus Reference Types

	Value Types	Reference Types
variable contains	value	reference
stored	On stack	heap
initialization	0, false, '\0'	null
assignment	copies the value	copies the reference
example	int i = 17;	string s = "Hello";
	int j= i;	string s1= s;
		

Simple Types

	Long Form	Range
Sbyte	System.SByte	-128 .. 127
byte	System.Byte	0 .. 255
short	System.Int16	-32768 .. 32767
ushort	System.UInt16	0 .. 65535
int	System.Int32	-2147483648 .. 2147483647
uint	System.UInt32	0 .. 4294967295
long	System.Int64	-2^{63}.. $2^{63}-1$
ulong	System.UInt64	0 .. $2^{64}-1$
float	System.Single	±1.5E-45 .. ±3.4E38 (32 Bit)
double	System.Double	±5E-324 .. ±1.7E308 (64 Bit)
decimal	System.Decimal	±1E-28 .. ±7.9E28 (128 Bit)
bool	System.Boolean	true, false
char	System.Char	Unicode character

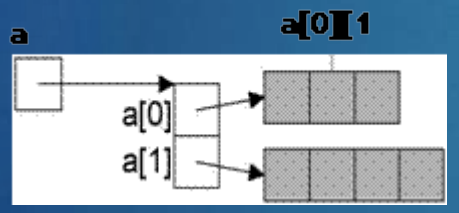
Enumerations

- Declaration

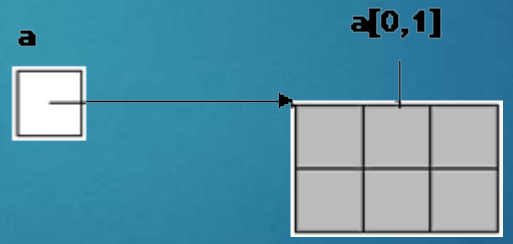
```
enum Color {red, blue, green} // values: 0, 1, 2
enum Access {personal=1, group=2, all=4}
enum Access1 : byte {personal=1,  group=2,  all=4}


Color c = Color.blue;     // enumeration constants must be qualified

Access a = Access.personal | Access.group;
if ((Access.personal & a) != 0) Console.WriteLine("access granted");
```

Arrays

```
int[] a = new int[3];
int[] b = new int[] {3, 4, 5};
int[] c = {3, 4, 5};

SomeClass[] d = new SomeClass[10];      // Array of references
SomeStruct[] e = new SomeStruct[10];    // Array of values (directly in the array)

int len = a.Length;    // number of elements in a
```




```
int[][] a = new int[2][];

a[0] = new int[3];

a[1] = new int[4];

int x = a[0][1];

int len = a.Length;    // 2

len = a[0].Length;     // 3
```




```
int[,] a = new int[2, 3];

int x = a[0, 1];

int len = a.Length; // 6

len = a.GetLength(0); // 2

len = a.GetLength(1); // 3
```

Strings

Can be used as standard type string

```
string s = "Alfonso";
```

Note

- Strings are immutable (use StringBuilder if you want to modify strings)
- Can be concatenated with +: `"Don " + s`
- Can be indexed: `s[i]`
- String length: `s.Length`
- Strings are reference types => reference semantics in assignments
- but their values can be compared with == and != : `if (s == "Alfonso") ...`
- Class String defines many useful operations:

  CompareTo, IndexOf, StartsWith, Substring, ...

Structs

- Declaration

```
Struct Point {
        public int  x, y;  // fields
        public Point (int x, int y)  // constructor
        {
            this.x = x;
            this.y = y;
        }
        public void MoveTo (int a, int b)      // methods
        {
            x = a;
            y = b;
        }
    }
```

- Use

```
Point p = new Point(3, 4);    // constructor initializes object on the stack

p.MoveTo(10, 20);         // method call
```

If statement

```
if ('0' <= ch && ch <= '9')
    val = ch -'0';
else if ('A' <= ch && ch <= 'Z')
    val = 10 + ch -'A';
else {
    val = 0;
    Console.WriteLine ("invalid character {0}", ch);
```

Switch statement

```
switch (country) {
      case "Germany": case "Austria": case "Switzerland":
            language = "German";
            break;
      case "England": case "USA":
            language = "English";
            break;
      case null:
            Console.WriteLine("no country specified");
            break;
      default :
            Console.WriteLine("don't know language of {0}",
country);
            break;
   }
```

Loops

while

```
while (i < n) {
        sum += i;
        i++;
}
```

do while

```
do {
        sum += a[i];
        i--;
} while (i > 0);
```

for

```
for (int i = 0; i < n; i++)
        sum += i;
```

Short form for

```
int i = 0;
while (i < n) {
    sum += i;
    i++;
}
```

Foreach statement

For iterating over collections and arrays

```
int[] a = {3, 17, 4, 8, 2, 29};
foreach (int x in a) sum += x;
```

```
string s = "Hello";
foreach (char ch in s) Console.WriteLine(ch);
```

```
Queue q = new Queue();
q.Enqueue("John"); q.Enqueue("Alice"); ...
foreach (string s in q) Console.WriteLine(s);
```

Return statement

Returning from a void method

```
void f(int x) {
    if (x == 0) return;
    ...
}
```

Returning a value from a function method

```
int max(int a, int b) {
    if (a > b) return a; else return b;
}

class C {
    static int Main() {
        ...
        return errorCode;        // The Main method can be declared as a function;
    }                            // the returned error code can be checked with the
                                 // DOS variable errorlevel
}
```