SWE344

Internet Protocols and Client-Server Programming

Module 1a: Introduction

Dr. El-Sayed El-Alfy alfy@kfupm.edu.sa

Mr. Bashir M. Ghandi bmghandi@ccse.kfupm.edu.sa

Computer Science Department
King Fahd University of Petroleum and Minerals

Objectives

- Present an overview of the course and the class policy
- Discuss the student expectations and the shades of the course title
- Introduce .NET Framework and its relation to C#
- Explore the development tools
- Discuss the C# programming basics
- Develop a small program in C#

Agenda

- What the Course is About, Its Learning Objectives
- Basics of .NET
- Introduction to C#

What is this course all about?

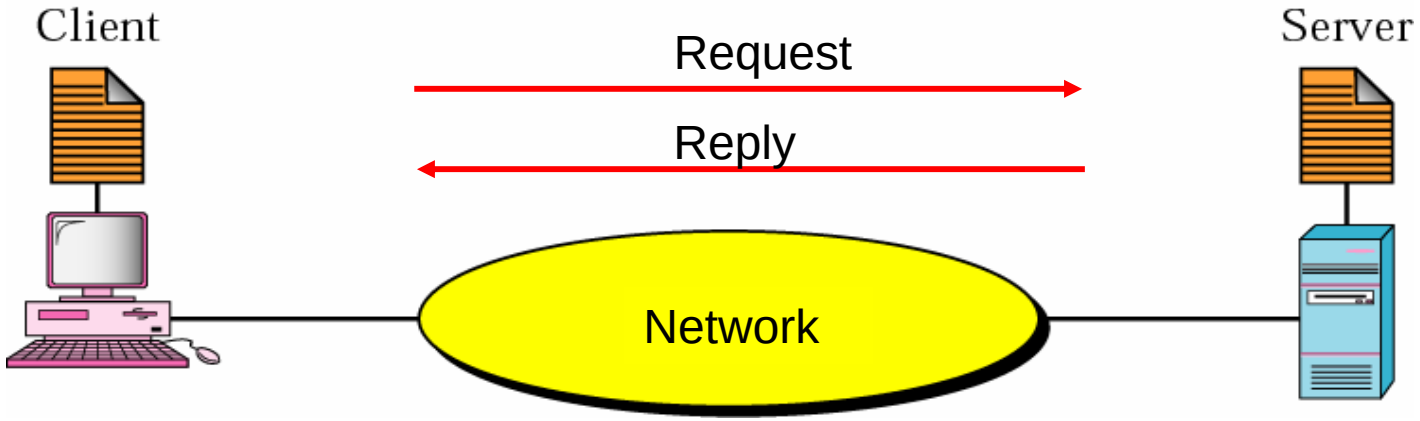
- This course explores the development of TCP/IP applications and their associated protocols. It utilizes hands-on programming and makes use of network monitoring tools. Several Client/Server applications are developed using the Socket interface.
- Tentative topics include
  - Overview of C# and .NET Framework
  - C# Programming Basics
  - OOP, GUI, Delegates, Events and Threads
  - TCP/IP Protocols and Client/Server Model
  - IP Addressing and Domain Name System (DNS)
  - Socket Programming using C# Sockets Helper Classes
  - Raw Socket Programming
  - Asynchronous and Multithreading C/S Programming
  - Application-Layer Programming: HTTP and Web Applications, SMTP, POP/MIME, FTP
  - UDP Broadcast and Multicast
  - Remoting as an example of object-oriented distributed application framework

Learning Objectives

- After taking this course, you should
  - Demonstrate understanding of developing C# applications in the .NET environment.
  - Recognize the basics of TCP/IP architecture and C/S model.
  - Describe and apply various socket programming concepts and mechanisms.
  - Develop client/server applications using socket interface.
  - Practice software engineering principles and methods in building network-aware applications.
  - Use software development tools effectively
  - Gain required skills to work in teams and present technical work.

What is a C/S Application?

- The Client-Server paradigm is the most prevalent model for distributed computing systems
- The Internet applications are based on the C/S model
- A typical network application has two processes
  - Client process: a program running on the local machine and requesting a service from another program (server) usually running at a remote computer
  - Server process: a program running on the remote computer to provide a service to the clients



Basics of .NET

- Microsoft introduced .NET technology in June 2000 as a programming platform that simplifies application development in the highly distributed environment of the Internet.
- Applications can be developed for MS Windows workstations and servers in a variety of programming languages
- A new programming language is developed specifically for the .NET platform is called C#
- C# is becoming a widely used programming language to create both network-aware and stand-alone applications for Windows systems

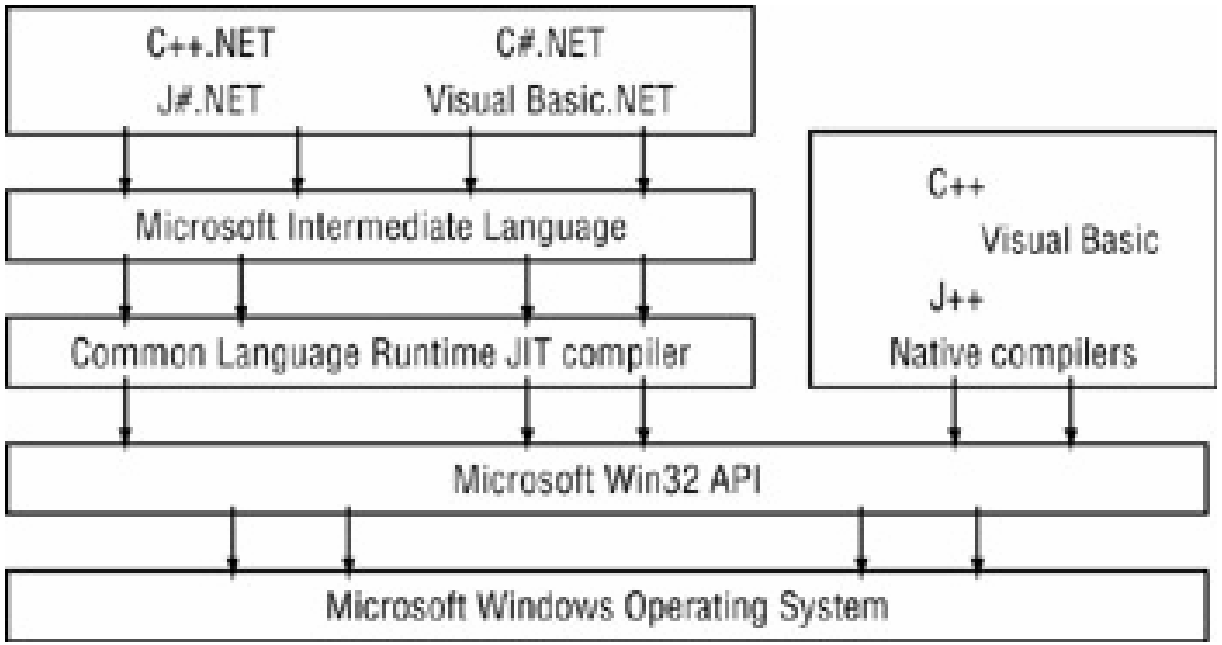
Basics of .NET …

- The .NET Framework Design Objectives:
  - To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
  - To provide a code-execution environment that minimizes software deployment and versioning conflicts.
  - To provide a code-execution environment that guarantees safe execution of code, including code created by an unknown or semi-trusted third party.
  - To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
  - To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
  - To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

(.NET Framework SDK Documentation)

Basics of .NET…

- What makes the .NET programming languages differ from previous versions of Windows programming languages?



They differ in the way programs are created and run on the Windows systems.

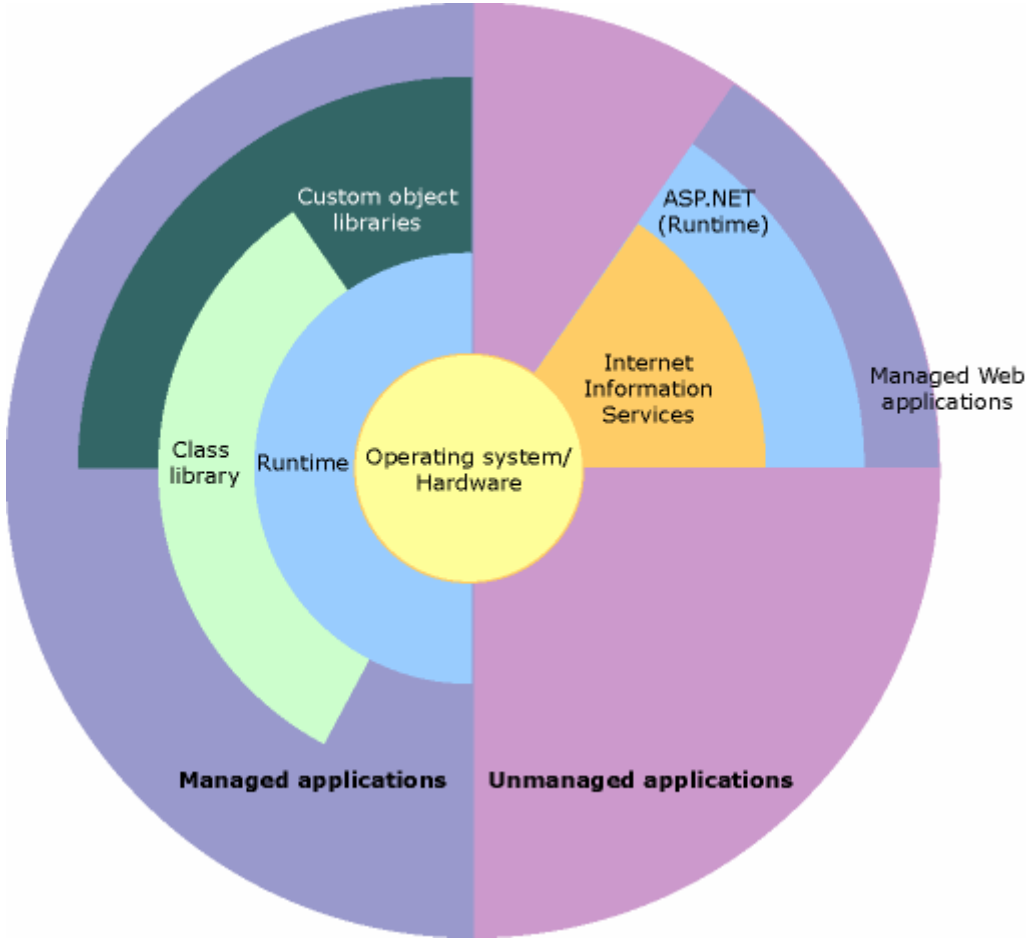
(C# Network Programming)

Basics of .NET…

- The .NET Framework has two main components:
  - Common Language Runtime (CLR)
    - Central part of the framework that executes .NET programs
    - Compilation process
      1. Programs compiled to Microsoft Intermediate Language (MSIL)
         - » Defines instructions for CLR
      2. MSIL code translated into machine code using Just In Time (JIT) compiler
         - » Produces machine code specifically tailored for a particular platform
         - » JIT compilation is only performed the first time you run the program (unless you turn off or reboot the computer) and the resulting machine code is automatically stored and reused
  - .NET Framework Class Library (FCL)
    - Pre-packaged components ready for reuse (classes, interfaces, structs, enumerators, etc)

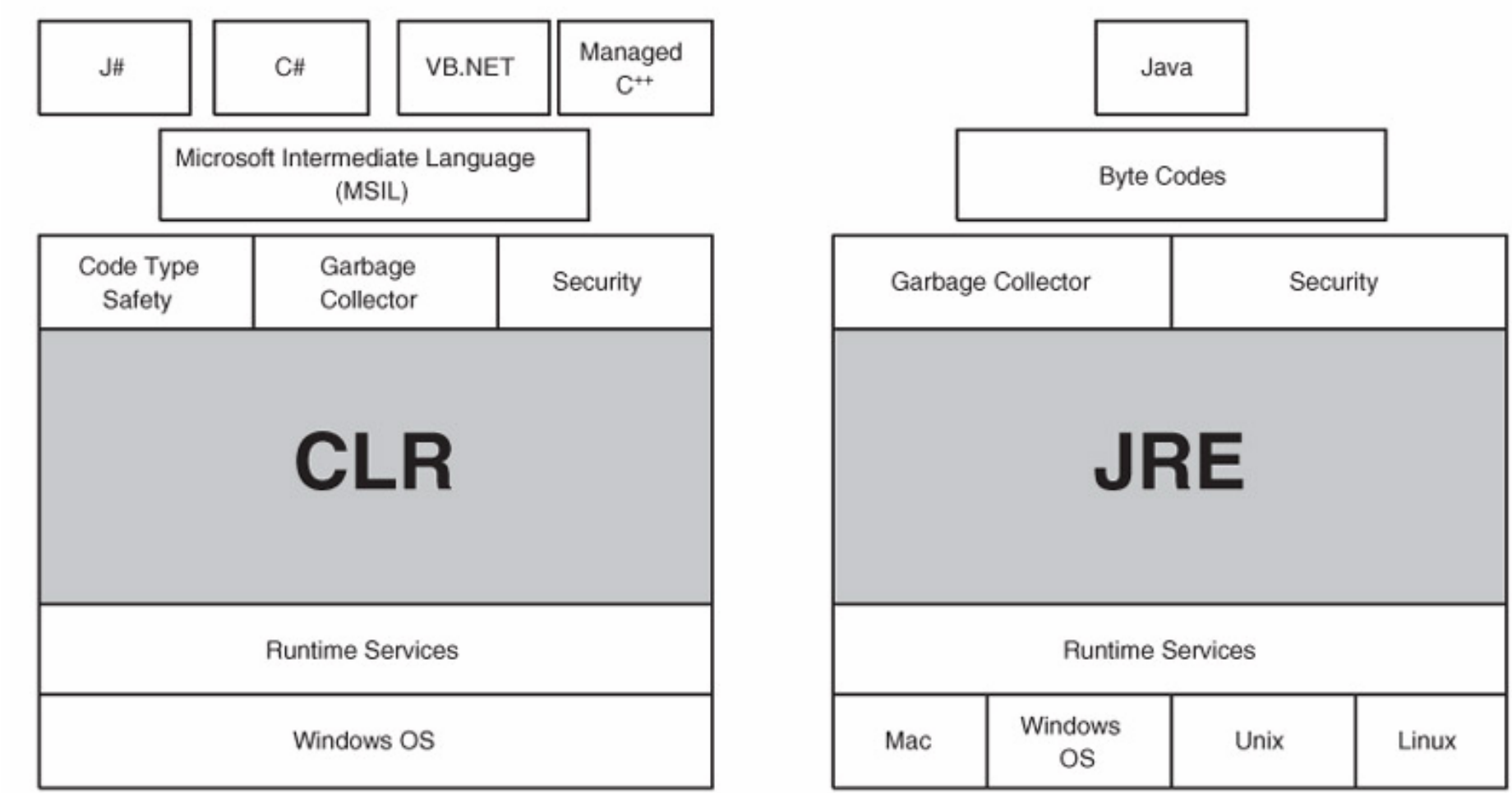
Basics of .NET…

- The relationship of CLR and FCL to your applications and to the overall system



Basics of .NET…

- How does it relate to Java?



- The .NET Framework is a collection of Software development tools (similar to JDK) that can be used to write, debug, compile and execute programs.

Key Features of .NET

- Language independence and integration
  - .NET programs not tied to particular language
  - Applications developed in any .NET compatible language
    - Visual Basic .NET, Visual C++ .NET, C# and more
  - Programs may consist of several .NET-compliant languages
  - Old and new components can be integrated
  - Programmers can contribute to applications using the language in which they are most competent
- Includes a rich Framework Class Library (FCL)
  - Pre-packaged components ready for reuse (classes, interfaces, structs, enumerators, etc)
  - Used by any .NET language
  - Make application development quicker and easier
  - Developers no longer need to be concerned with details of components

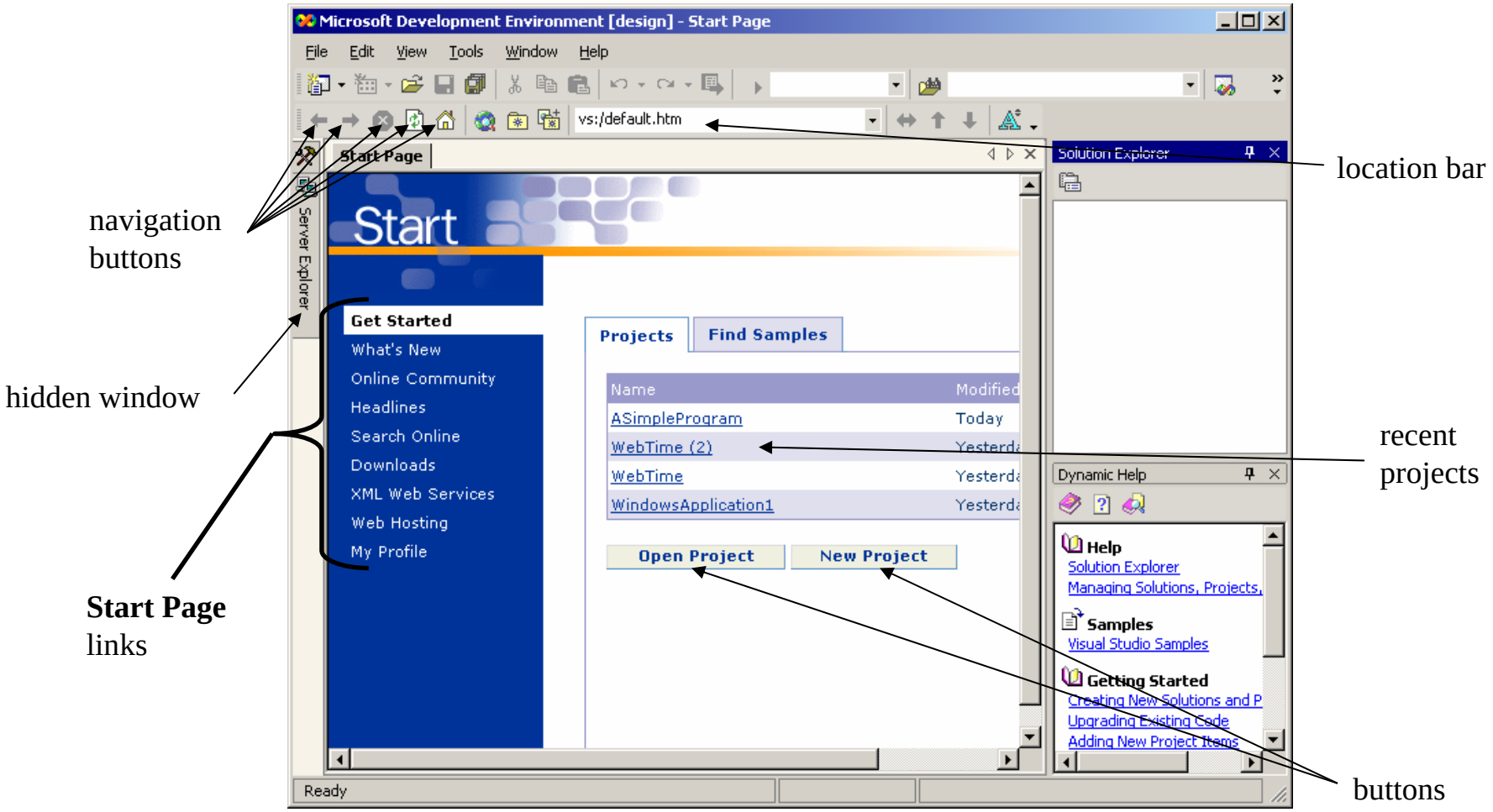
Key Features of .NET …

- Development of a variety of applications and services
  - Console applications, Windows Forms, ASP.NET applications, XML Web Services, etc
- New program development process and execution-management features
  - Manages memory, security and other features
    - Relieves programmer of many responsibilities
    - More concentration on program logic
    - Provides increased productivity
- Software reusability
  - Web services provide solutions for wide variety of companies
    - Cheaper than developing one-time solutions that can't be reused
    - Single applications perform all operations for a company via various Web services
      - Manage taxes, bills, investments and more
- Additional information available at Microsoft Web site www.microsoft.com/net

Development Environments

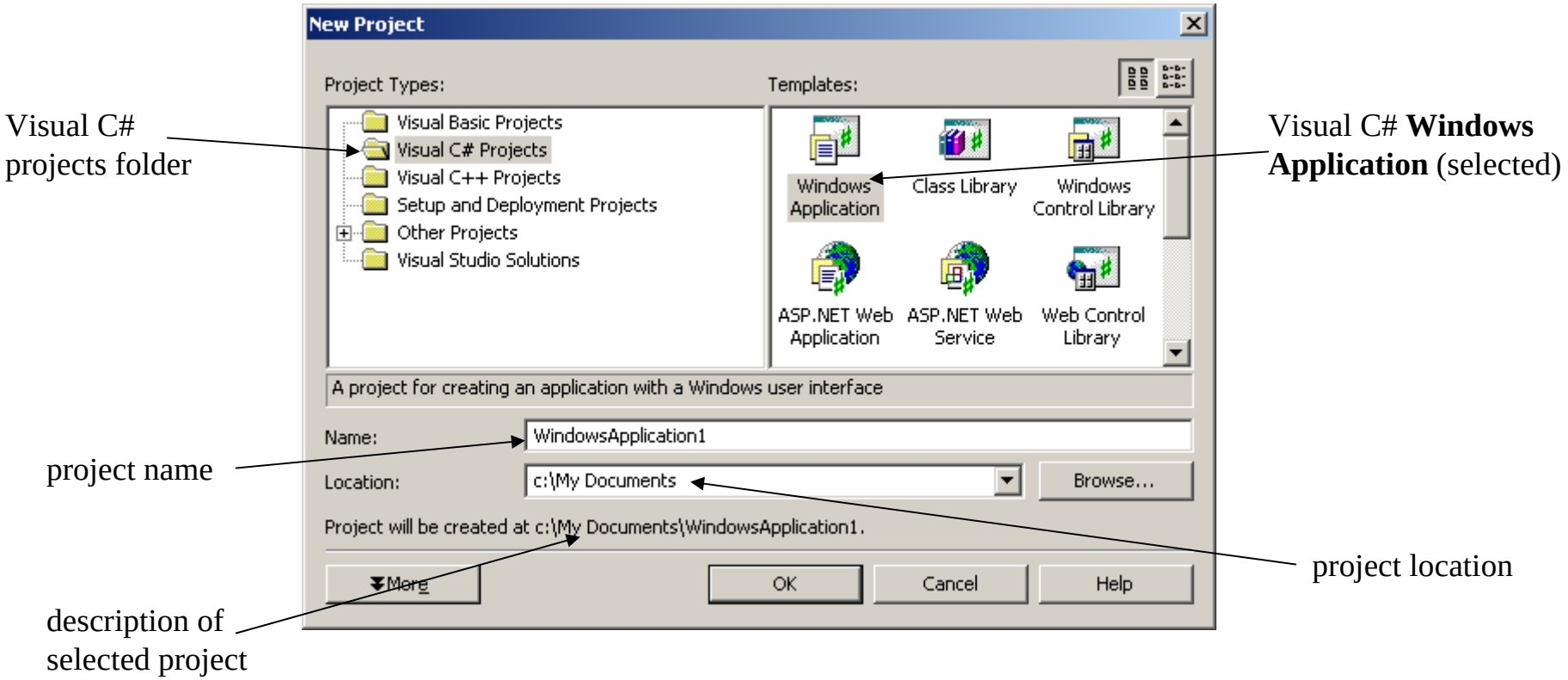
- .NET Framework SDK
  - Download at http://msdn.microsoft.com/downloads
- Visual Studio .NET (VS.NET)
  - Microsoft's Integrated Development Environment (IDE) used for Rapid Application Development (RAD)
    - Edit, compile, debug, run
  - More productive and easy to use development tool
  - Program in a variety of .NET languages
  - Create different types of applications
    - Console applications, windows applications, ASP.NET applications, XML Web Services
  - Tools to edit and manipulate several file types

Visual Studio .NET IDE Overview



(C# How To Program)

Visual Studio .NET IDE Overview …



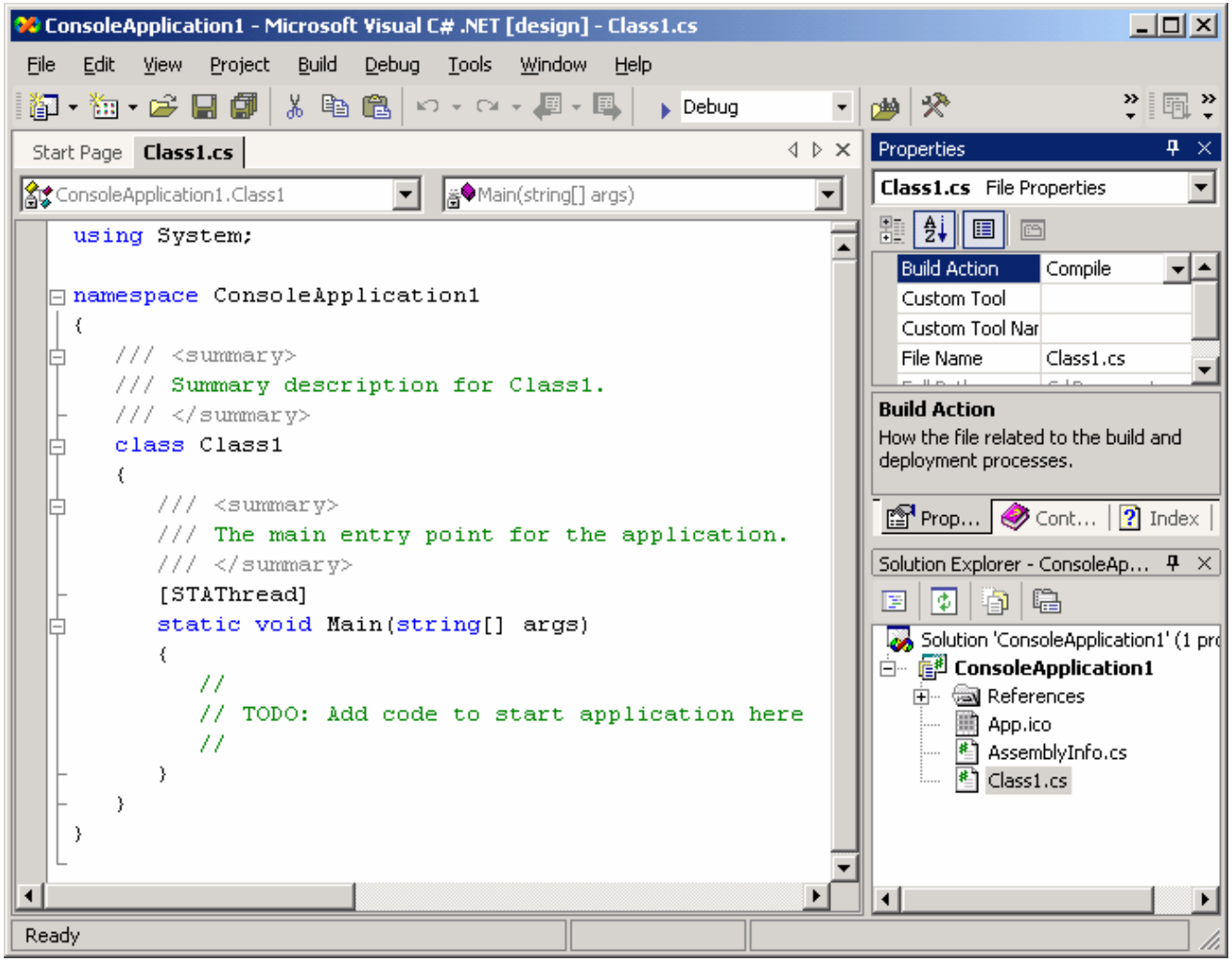
New Project dialog.

(C# How To Program)

Applications

- Console applications
  - Applications that display only text as output in a command window (also called console window)
  - No visual components
  - Only text output
  - Two types of the command windows
    - MS-DOS prompt
      - Used in Windows 95/98/ME
    - Command prompt
      - Used in windows 2000/NT/XP
- Windows applications
  - Applications that provide graphical user interface (GUI) with multiple types of visual controls, e.g. windows, dialogs, buttons, menus, etc

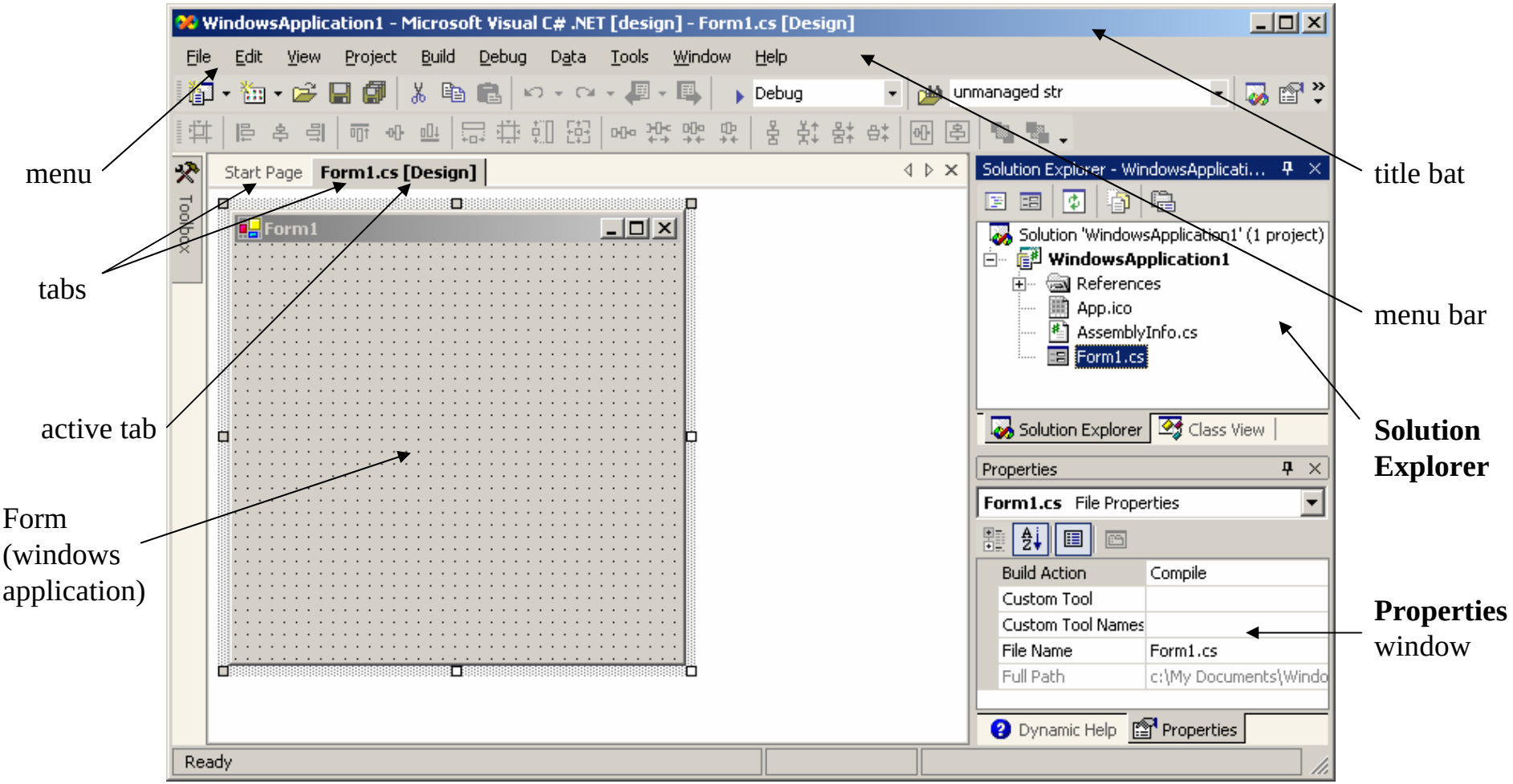
Applications …



Visual Studio .NET-generated console application.

(C# How To Program)

Applications …



Visual Studio .NET environment after a new project has been created.

(C# How To Program)

Applications …

- ASP.NET Applications
  - Programs that run over the Internet
  - Accessed using a web browser e.g. IE
  - Examples: online banking, stock trading, online auction systems, etc
- ASP.NET Web Services (XML Web Services)
  - Are also programs that run over the Internet
  - Used to offer a service that can be used in a distributed system of interconnected services
  - Example: MS Passport web service offers identification and authentication of web users

C# Language

- C# is a new language designed specifically for .NET platform to provide an optimum blend of simplicity, expressiveness, and performance.
- Many features of C# were designed in response to the strengths and weaknesses of other languages, particularly Java and C++.
- The C# language specification was written by Anders Hejlsberg and Scott Wiltamuth at Microsoft

(Essential of C#, 2002)

Hello World Program (v0)

```
1.  // Hello0.cs
2.  public class Hello0
3.  {
4.    public static void Main()
5.    {
6.      System.Console.WriteLine("Hello, World!");
7.    }
8.  }
```

Program execution starts at the Main() method

The **System.Console** class contains a **WriteLine** method that can be used to display a string to the console

```
Hello, World!
```

Hello World Program (v1)

To avoid fully qualifying classes throughout a program, you can use the using directive

```
1.  // Hello1.cs
2.  using System;
3.  public class Hello1
4.  {
5.    public static void Main()
6.    {
7.       Console.WriteLine("Hello, World!");
8.    }
9.  }
```

Console.WriteLine() is the same as System.Console.WriteLine()

A namespace such as System contains several classes and are used to avoid name conflict

Hello World Program (v2)

If you need access to the command line parameters passed in to your application, simply change the signature of the Main method to include them

```
1.  // Hello2.cs
2.  using System;
3.  public class Hello2
4.  {
5.    public static void Main(string[] argv)
6.    {
7.       Console.WriteLine(argv[0]);
8.    }
9.  }
```

argv[0] will contain the first parameter you enter after the application name

Hello World Program (v3)

```
1.  // Hello3.cs
2.  using System;
3.  public class Hello3
4.  {
5.    public static int Main(string[] argv)
6.    {
7.       Console.WriteLine("Hello, World!");
8.       return 0;
9.    }
10. }
```

The application can also return a code value to the operating system

Hello World Program (v3)

You can define your namespace and define all related classes inside it

```
1.  // Hello4.cs
2.  using System;
3.  namespace Greetings
4.  {
5.    public class Hello4
6.    {
7.      public static void Main(string[] argv)
8.      {
9.       Console.WriteLine("Hello, World!");
10.      return 0;
11.     }
12.   }
13. }
```

Anatomy of the Program

- Comments
  - Single line comment is preceded by //
  - Multiple line comments are enclosed between /* and */
  - Comments are ignored by the compiler
  - Used only for human readers to improve code readability
- White Space
  - Includes spaces, newline characters and tabs
- Guidelines for writing clear/readable code
  - Use meaningful identifiers (class name, object references, variable names, method parameters, etc)
  - Use white space and statement layout to promote clarity
  - Use comments intelligently
  - Use symbolic constants
  - Avoid large methods (use manageable components)

Anatomy of the Program …

- Namespaces
  - The .NET framework class library (FCL) is composed of namespaces (packages in Java)
  - A namespace is a group of classes and their methods
  - Allows the easy reuse of code and avoids name conflict
  - Namespaces are stored in .dll files called assemblies
  - When using members of a namespace
    - use fully-qualified name, e.g.,

```
System.Console.WriteLine("Salam Shabab");
```

    - include the namespace in the program with the using keyword
  - To declare a namespace, use the keyword, namespace and a pair of braces are used to enclose all members of a namespace
  - If a class is not enclosed in a namespace, then it is assumed to be part of a global namespace, which has no name

Anatomy of the Program …

- Some of the most important namespaces in the CLR

Namespace	Description
System	Contains classes that implement basic functionalities like Console I/O, mathematical operations, data conversions etc.
System.IO	Contains classes used for file I/O operations.
System.Net	Contains classes that provide access to Windows network functions.
System.Net.Sockets	Contains classes the provides access to windows socket interface
System.Collections	Contains classes that implement collections of objects such as linked list, queue, hash table etc.
System.Drawing	Contains classes that provide basic graphics functionalities.
System.Windows.Forms	Contains classes for Windows GUI applications
System.Threading	Contains classes that are used for multithreading programming.
System.Web	Classes that implement HTTP protocol to access web pages.

Anatomy of the Program …

- Class declaration
  - As in Java, a class is declared using the class keyword, and all members of a class must be enclosed inline within a pair of braces
- The Main() method
  - is the entry point of a C# application where program execution begins
  - Like in Java, it must be static, however, in C# it has three different signatures as follows:

```
public static void Main()
public static void Main( string[] args )
public static int Main( string[] args )
```

- Console I/O
  - The Console class of the System namespace has a number of static methods that enable a console application
    - to display strings and other types of data to the command window, e.g.
      - WriteLine() and Write() methods to print a single line of text
      - These two methods are overloaded to take different and variable parameters
    - For reading
      - ReadLine() and Read() methods

```
public static int Read();// reads a character
public static string ReadLine(); //reads a line as a string
```

Identifiers

- Identifiers are names programmers choose for their types, methods, variables, and so on.
- Naming conventions
  - An identifier must be a whole word
  - Can contain letters, digits, and underscores (_)
  - Can not start with digits
  - C# identifiers are case-sensitive
  - Must not conflict with a keyword (to ensure this start with @ symbol but not considered as part of the name)
  - Use mixed case when an identifier involves more than one word, e.g. WriteLine.
  - Names of variables start with small letters
  - Names of Namepaces, Classes, Interfaces, Structs, Enums, Properties and Methods start with capital letters
  - The Main method must start with a capital letter

Variables

- A variable represents a typed storage location
- A variable can be a local variable, a parameter, an array element, an instance field, or a static field.
- Every variable has an associated type, which essentially defines
  - the possible values the variable can have and
  - the operations that can be performed on that variable
- C# is a strongly typed language
  - Any variable must be declared to be of certain type, e.g.

```
double score;
```

- C# is type-safe
- Variables *must* be assigned a value before they are used.
  - either explicitly assigned a value or
  - automatically assigned a default value (occurs for static fields, class instance fields, and array elements not explicitly assigned a value)

Value Types and Reference Types

- As in Java, variables in C# are of two types, namely, value types (primitive types) and reference types.
- A third type called pointers can be used in unmanaged code
- Value types
  - C# has more value types than Java
  - Contains an actual value of the specified type
  - Programmer created
    - **`struct`**s
    - **`enum`**erations
- Reference types
  - Contain an address of an object
  - Programmer create
    - Classes
    - Interfaces
    - Delegates

Value Types

Type	.Net Framework (System) type	Signed?	Bytes Occupied	Possible Values
sbyte	System.Sbyte	Yes	1	-128 to 127
short	System.Int16	Yes	2	-32768 to 32767
int	System.Int32	Yes	4	-2147483648 to 2147483647
long	System.Int64	Yes	8	-9223372036854775808 to 9223372036854775807
byte	System.Byte	No	1	0 to 255
ushort	System.Uint16	No	2	0 to 65535
uint	System.UInt32	No	4	0 to 4294967295
ulong	System.Uint64	No	8	0 to 18446744073709551615
float	System.Single	Yes	4	Approximately $\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ with 7 significant figures
double	System.Double	Yes	8	Approximately $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ with 15 or 16 significant figures
decimal	System.Decimal	Yes	12	Approximately $\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$ with 28 or 29 significant figures
char	System.Char	N/A	2	Any Unicode character (16 bit)
bool	System.Boolean	N/A	1 / 2	true or false

Symbolic Constants

- A constant declaration is like a variable declaration, except that the value of the variable can't be changed after it has been declared, e.g.

```
const double PI = 3.14;
const double speedOfLight = 2.99792458E08;
```

Example

```
1.    class Addition  {
2.        static void Main( string[] args ) {
3.            string firstNumber,   // first string entered by user
4.                   secondNumber;  // second string entered by user
5.
6.            int number1,          // first number to add
7.                number2,          // second number to add
8.                sum;              // sum of number1 and number2
9.
10.           // prompt for and read first number from user as string
11.           Console.Write( "Please enter the first integer: " );
12.           firstNumber = Console.ReadLine();
13.
14.           // read second number from user as string
15.           Console.Write( "enter the second integer: " );
16.           secondNumber = Console.ReadLine();
17.
18.           // convert numbers from type string to type int
19.           number1 = Int32.Parse( firstNumber );
20.           number2 = Int32.Parse( secondNumber );
21.
22.           // add numbers
23.           sum = number1 + number2;
24.
25.          // display results
26.           Console.WriteLine( "sum is {0}.", sum );
27.
28.       } // end method Main
29.   } // end class Addition
```

```
Please enter the first integer: 45
Please enter the second integer: 72
The sum is 117.
```

Strings

- A string is an object that contains a sequence of Unicode characters
- String literals are written between double quotations, e.g., `“Welcome”`
- A string reference is declared using String or string, `string greetings=“Welcome!”;`
- You can concatenate two strings using +

```
string greetings = “Welcome ” + “from C#!”;
```

- You can display a string using

```
System.Console.WriteLine(greetings);
System.Console.WriteLine(“x= ” + 2);
```

- To get the length of a string use the Length property, e.g. `greetings.Length`
- Immutable strings (objects of string class type) can't be modified after creation
- Mutable strings (also called dynamic strings) are objects of type StringBuilder and can be modified -- (similar to StringBuffer in Java)

Manipulating Strings

- C# offers a wide range of string-handling features
- Testing equality of two strings (duplicate strings are removed; string interning)

```
string a = "hello"; string b = "hello";
Console.WriteLine(a == b); // True for String only
Console.WriteLine(a.Equals(b)); // True for all
  objects
Console.WriteLine(Object.ReferenceEquals(a, b)); //
  True!!
```

- Indexing strings – the characters in a string are accessed with a zero-based index

```
string s = "Going down?";
for (int i=0; i<s.Length; i++)
  Console.WriteLine(s[i]); // Prints s vertically
```

- Copying strings

```
string s2 = s1;
string s2 = string.Copy(s1);
```

Manipulating Strings …

static int Compare(s1, s2)	Overloaded. Compares two specified String objects.
int CompareTo(string)	Overloaded. Compares this instance with a specified object.
static string Copy(string)	Creates a new instance of String with the same value as a specified String.
bool EndsWith(string)	Determines if the end of this instance matches the specified String.
bool Equals(object)	Overloaded. Overridden. Determines whether two String objects have the same value.
CharEnumerator GetEnumerator()	Retrieves an object that can iterate through the individual characters in this instance.
int IndexOf(char) int IndexOf(string)	Overloaded. Reports the index of the first occurrence of a String, or one or more characters, within this instance.
String Insert(int, string)	Inserts a specified string at a specified index of this string. Returns the updated string.
int LastIndexOf(char) int LastIndexOf(string)	Overloaded. Reports the index position of the last occurrence of a specified Unicode character or String within this instance.
String PadLeft(int) String PadLeft(int, char)	Overloaded. Right-aligns the characters in this instance, padding on the left with spaces or a specified Unicode character for a specified total length.
String PadRight(int) String PadRight(int, char)	Overloaded. Left-aligns the characters in this string, padding on the right with spaces or a specified Unicode character, for a specified total length.
String Remove(int index, int count)	Deletes a specified number of characters from this instance beginning at a specified position.

Manipulating Strings …

String Replace(char, char) String Replace(string, string)	Overloaded. Replaces all occurrences of a specified Unicode character or String in this instance, with another specified Unicode character or String.
String[] Split(char[])	Overloaded. Identifies the substrings in this instance that are delimited by one or more characters specified in an array, then places the substrings into a String array.
bool StartsWith(string)	Determines whether the beginning of this instance matches the specified String.
String Substring(int start) String Substring(int start, int count)	Overloaded. Retrieves a substring from this instance.
char[] ToCharArray()	Overloaded. Copies the characters in this instance to a Unicode character array.
String ToLower()	Overloaded. Returns a copy of this String in lowercase.
String ToString()	Overloaded. Overridden. Converts the value of this instance to a String.
String ToUpper()	Overloaded. Returns a copy of this String in uppercase.
String Trim() String Trim(char[])	Overloaded. Removes all occurrences of a set of specified characters from the beginning and end of this instance.
String TrimEnd(char[])	Removes all occurrences of a set of characters specified in a Unicode character array from the end of this instance.
String TrimStart(char[])	Removes all occurrences of a set of characters specified in a Unicode character array from the beginning of this instance.

Example

```
1.  using System;
2.  public class FilenameProcessor {
3.    public static void Main(String[] args) {
4.      String fullName = "d:/workarea/lab02/MoveRec.java";
5.      char separator = '/';
6.      int dotPosition = fullName.IndexOf('.');
7.      int lastSlashPos = fullName.LastIndexOf(separator);
8.      Console.WriteLine("The full name is: "+fullName);
9.      String path = fullName.Substring(0, lastSlashPos);
10.     Console.WriteLine("The path is       : "+path);
11.     String fileName = fullName.Substring(lastSlashPos+1,
12.                       dotPosition-lastSlashPos-1);
13.     Console.WriteLine("The file name is : "+fileName);
14.     String fileExtension =
15.                   fullName.Substring(dotPosition+1);
16.     Console.WriteLine("The extension: "+fileExtension);
17.   }
18. }
```