PURBANCHAL UNIVERSITY

DOT NET PROGRAMMING

(Compiled Notes)

YEAR-IV *(BCA453CO)* *SEMESTER-II*

Jeevan Poudel
BCA VIII, 2077

Purbanchal University, Nepal

Dot Net Programming (BCA453CO)

(Compiled Notes)

BCA-VIII

Jeevan Poudel

श्री गोमेन्द्र बहुमुखी महाविद्यालय
वितामोड, झापा

चैत ६, २०७७
(2021)

यो पाठ्य सामग्री तयार पार्न साथ, सहयोग र हौसला प्रदान
गर्नुहुने आदरणीय गुरु श्री मदन उप्रेती र मेरा सबै साथीहरुप्रति
हार्दिक आभार प्रकट गर्दछु।

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER

1

OVERVIEW OF VB . NET AND C# . NET LANGUAGE

1.1 Introduction to . Net Framework

In computer (and software) programming, a framework is what is known as an abstraction where code can be altered to build application-specific software. The framework is a collection of Application Programming Interfaces, or APIs, that come with an enormous library of code that developers can use to write software, so they aren't forced to write all the code from scratch.

An abstraction is basically the act of removing elements to reduce it to its essential form. In terms of software, it's the process of creating a clean slate for developers to work with.

. NET Framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large class library named as Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages. Programs written for . Net Framework execute in a software environment (in contrast to a hardware environment) named the *Common Language Runtime (CLR)*. The CLR is an application virtual machine that provides services such as security, memory management, and ex-

ception handling. As such, computer code written using . NET Framework is called "managed code". FCL and CLR together constitute the . NET Framework.

FCL provides the user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications.

1.1.1 . Net Framework Architecture

.Net Framework Architecture is a programming model for the .Net platform that provides an execution environment and integration with various programming languages for simple development and deployment of various Windows and desktop applications. It consists of class libraries and reusable components.

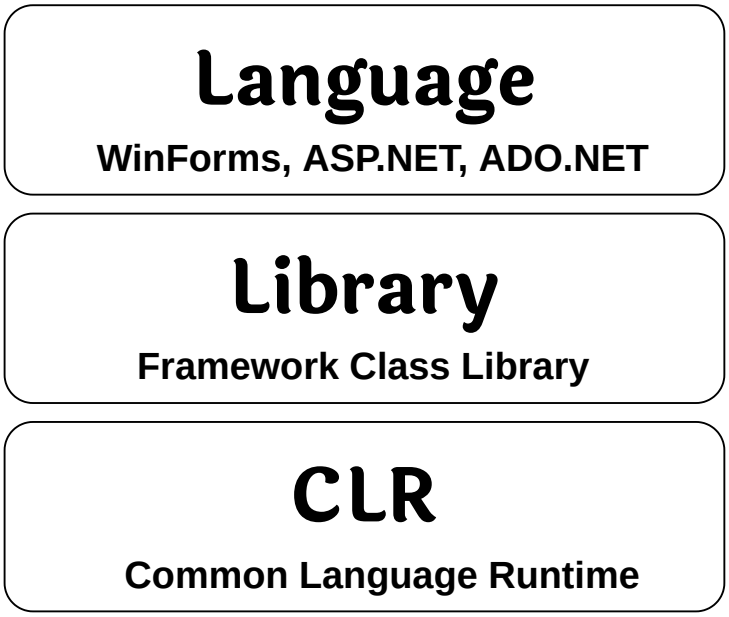

Figure 1.1: . NET framework architecture

The . Net Framework Architecture is the programming model for the . Net platform. It provides a managed execution environment, simplified development and deployment and integration with a wide variety of programming languages. The . Net Framework class library (FCL) is a comprehensive, object-oriented collection of reusable types that you can use to develop applications. The common language runtime (CLR) is the core runtime engine for executing applications in the . Net Framework. The CLR is fully protected

from the outside environment and highly optimized within, taking advantage of the services that the CLR provides such as security, performance, deployment facilities, and memory management , including garbage collection.

1.1.1.1 Common Language Runtime

The "Common Language Infrastructure" or CLI is a platform in . Net architecture on which the . Net programs are executed.

The CLI has the following key features:

- Exception Handling
- Garbage Collection
- Working with Various programming languages

1.1.1.2 Class Library

The .NET Framework includes a set of standard class libraries. A class library is a collection of methods and functions that can be used for the core purpose.

For example, there is a class library with methods to handle all file-level operations. So, there is a method which can be used to read the text from a file. Similarly, there is a method to write text to a file.

Most of the methods are split into either the `System.*` or `Microsoft.*` namespaces[1]. (The asterisk * just means a reference to all the methods that fall under the System or Microsoft namespace)

1.1.1.3 Languages

The types of applications that can be built in the .Net framework is classified broadly into the following categories

- ADO.Net – This technology is used to develop applications to interact with Databases such as Oracle or Microsoft SQL Server: This is used for developing Forms-based applications, which would run on an end user machine. Notepad is an example of a client-based application.
- `ASP.Net`: This is used for developing web-based applications, which are made to run on any browser.
- `ADO.Net`: This technology is used to develop applications to interact with Databases such as Oracle or Microsoft SQL Server.

[1] A namespace is a logical separation of methods.

Common Language Specification (CLS)

- CLS is a set of basic language features that . Net Languages needed to develop Applications and Services, which are compatible with the . Net Framework.
- When there is a situation to communicate Objects written in different . Net Complaint languages, those objects must expose the features that are common to all the languages. Common CLS ensures complete interoperability among applications, regardless of the language used to create the application.

Common Type System (CTS)

CTS describes a set of types that can be used in different . Net languages in common. That is, the CTS ensure that objects written in different . Net languages can interact with each other. For Communicating between programs written in any . Net complaint language, the types have to be compatible on the basic level.

These types can be Value Types or Reference Types. The Value Types are passed by values and stored in the stack. The Reference Types are passed by references and stored in the heap. CTS provides base set of Data Types which is responsible for cross language integration. The CLR can load and execute the source code written in any . Net language, only if the type is described in the CTS. Most of the members defined by types in the . Net FCL are CLS compliant Types.

Microsoft Intermediate Language (MSIL)

- MSIL stands for Microsoft Intermediate Language.
- We can call it as Intermediate Language (IL) or Common Intermediate Language (CIL).
- During the compile time, the compiler convert the source code into MSIL.
- MSIL is a CPU-independent set of instructions that can be efficiently converted to the native code.
- During the runtime the Common Language Runtime (CLR)'s Just In Time (JIT) compiler converts the MSIL code into native code to the Operating System.

When a compiler produces MSIL, it also produces Metadata. MSIL and Metadata are contained in a portable executable (PE) file. 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.

Just In Time Compiler (JIT)

The . Net languages, which conforms to the Common Language Specification (CLS), uses its corresponding runtime to run the application on different Operating Systems. During the code execution time, the Managed Code compiled only when it is needed, that is it converts the appropriate instructions to the native code for execution just before when each function is called. This process is called Just In Time (JIT) compilation, also known as Dynamic Translation. With the help of Just In Time Compiler (JIT) the Common Language Runtime (CLR) doing these tasks.

The Common Language Runtime CLR provides various JIT compilers and each works on a different architecture depending on OS. That is why the same MSIL can be executed on different OS without rewriting the source code. JIT compilation preserves memory and save time during application initialization.

Managed Code

Managed Code in Microsoft . Net Framework, is the code that has executed by the Common Language Runtime (CLR) environment. On the other hand Unmanaged Code is directly executed by the computer's CPU. Data types, error-handling mechanisms, creation and destruction rules, and design guidelines vary between managed and unmanaged object models.

The benefits of Managed Code include programmers convenience and enhanced security. Managed code is designed to be more reliable and robust than unmanaged code, examples are Garbage Collection, Type Safety etc. The Managed Code running in a CLR cannot be accessed outside the runtime environment as well as cannot call directly from outside the runtime environment. This makes the programs more isolated and at the same time computers are more secure. Unmanaged Code can bypass the . Net Framework and make direct calls to the OS. Calling unmanaged code presents a major security risk.

Difference between managed and unmanaged code

What is Managed Code? Managed code is the code that is written to target the services of the managed runtime execution environment such as Common Language Runtime in . Net Technology.

Figure 1.2: Managed Code

The Managed Code running under a CLR cannot be accessed outside the runtime environment as well as cannot call directly from outside the runtime environment. It refers to a contract of cooperation between natively executing code and the runtime. It offers services like garbage collection, run-time type checking, reference checking etc. By using managed code we can avoid many typical programming mistakes that lead to security holes and unstable applications, also, many unproductive programming tasks are automatically taken care of, such as type safety checking, memory management, destruction of unused objects etc.

What is Unmanaged Code? Unmanaged code compiles straight to machine code and directly executed by the Operating System. The generated code runs natively on the host processor and the processor directly executes the code generated by the compiler. It is always compiled to target a specific architecture and will only run on the intended platform. If we want to run the same code on different architecture then we will have to recompile the code using that particular architecture.

Unmanaged executable files are basically a binary image, `x86` code, directly loaded into memory. This approach typically results in the fastest code execution, but diagnosing and recovery from errors might difficult and time-consuming in most cases. The memory allocation, type safety, security, etc needs to be taken care of by the programmer and this will lead unmanaged code prone to memory leaks like buffer overruns, pointer overrides etc.

All code compiled by traditional `C/C++` compilers are Unmanaged Code. COM components, ActiveX interfaces, and Win32 API functions are examples of unmanaged code. Managed code is code written in many high-level programming languages that are available for use with the Microsoft . Net

Framework, including VB. Net, C#, J#, etc. Since Visual C++ can be compiled to either managed or unmanaged code it is possible to mix the two in the same application.

. Net Framework Design Principle

The following design principles of the . Net framework is what makes it very relevant to create . Net based applications.

The principal design features of Microsoft . Net Framework are:

- *Interoperability*
- *Portability*
- *Security*
- *Language independence*
- *Type safety*
- *Memory management*

Interoperability

Language interoperability is the ability of code to interact with code that is written using a different programming language. It can help maximize code reuse and, therefore, improve the efficiency of the development process. The . NET components can communicate with the existing COM components without migrating to those components into . NET. That means, this feature is a great help to reduce the migration cost and time.

Portability

Applications built on the . Net framework can be made to work on any Windows platform. And now in recent times, Microsoft is also envisioning to make Microsoft products work on other platforms, such as iOS and Linux.

Security

The . NET Framework has a good security mechanism. The inbuilt security mechanism helps in both validation and verification of applications. Every application can explicitly define their security mechanism. Each security mechanism is used to grant the user access to the code or to the running program.

Language Independence

The . Net Framework is language independent. It is possible to use . Net from many programming languages because they have all agreed on some

standards. This means that, as a programmer, we can develop in one of the many languages that target the . Net Framework, such as C#, C++/CLI, Eiffel, F#, IronPython, IronRuby, PowerBuilder, Visual Basic, Visual COBOL, and Windows PowerShell. The . Net Framework presents a Common Type System (CTS) that characterizes every conceivable data sorts and programming builds bolstered by Common Language Runtime and how they might communicate with each other fitting in with CLI determination. Because of this feature, . Net Framework supports the exchange of types and object instances between libraries and applications written using any conforming . Net language.

Type safety

Type-Safety is enforced by the CLR and the Language Compiler — in accordance to the CTS directives of the . Net Framework. Type-safe code cannot perform an operation on an object that is invalid for that object. This prevents ill-defined casts, wrong method invocations, and memory size issues when accessing an object. For example, if you have declared a variable as an integer, it cannot be assigned any value which is not an integer (by implicit conversion, or explicit conversion).

Memory Management

The Common Language runtime does all the work or memory management. The . Net framework has all the capability to see those resources, which are not used by a running program. It would then release those resources accordingly. This is done via a program called the "Garbage Collector", which runs as part of the . Net framework.

The garbage collector runs at regular intervals and keeps on checking which system resources are not utilized, and frees them accordingly.

1.2 Introduction to C# and VB

1.2.1 C#

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).

C# was developed by *Anders Hejlsberg* and his team during the development of . Net Framework. It is based on C++ and Java, but it has many

additional extensions used to perform component oriented programming approach.

C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

C# Features

- *Simple*

  C# is a simple language in the sense that it provides structured approach (to break the problem into parts), rich set of library functions, data types etc.

- *Modern Programming Language*

  C# programming is based upon the current trend and it is very powerful and simple for building scalable, interoperable and robust applications.

- *Object-Oriented*

  C# is object-oriented programming language. OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grow.

- *Type Safe*

  C# type safe code can only access the memory location that it has permission to execute. Therefore it improves a security of the program.

- *Interoperability*

  Interoperability process enables the C# programs to do almost anything that a native C++ application can do.

- Scalable and Updateable

  C# is automatic scalable and updateable programming language. For updating our application we delete the old files and update them with new ones.

- *Structured Programming Language*

  C# is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.

- *Rich Library*

  C# provides a lot of inbuilt functions that makes the development fast.

1.2.2 VB (Visual Basic) . NET

Visual Basic is a third-generation event-driven programming language first released by Microsoft in 1991. It evolved from the earlier DOS version called BASIC. BASIC means *Beginners' All-purpose Symbolic Instruction Code.*

VB . NET is a fully object-oriented programming language implemented in the . NET Framework. It was created to cater for the development of the web as well as mobile applications.

VB . Net is an update to Visual Basic that targets Microsoft . Net Framework. VB . Net has a lot of similarities to Visual Basic but also some differences. VB . Net is an object-oriented language, which supports the abstraction, encapsulation, inheritance, and polymorphism features. It is the most productive tool for rapidly creating a wide range of Windows, Web, Mobile, and Office applications built on the . Net Framework.

The Visual Basic language is designed to be human-readable and accessible to everyone from novice programmers to advanced system architects. All of this is built on top of the . Net Framework, which guarantees that programs written in Visual Basic run with unsurpassed scalability and reliability. The . Net Framework provides VB . Net programmers with the ability to create fully object oriented programs (OOPs), just like the ones created using Java, C# or C++. Also programs written in VB. Net will interoperate seamlessly with programs written in any other . Net languages such as Visual C#, Visual J#, or Visual C++.

Featues of VB . Net

VB . NET comes loaded with numerous features that have made it a popular programming language amongst programmers worldwide. These features include the following

- VB.NET is not case-sensitive
- It is an object-oriented programming language
- Garbage collection
- Events management
- Windows Forms
- Support for web application development

1.3 Feature of Object Oriented Programming

Object

Object is a collection of number of entities. Objects take up space in the memory. Objects are instances of classes. When a program is executed, the objects interact by sending messages to one another. Each object contain data and code to manipulate the data. Objects can interact without having know details of each others data or code.

Class

Class is a collection of objects of similar type. Objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class. Classes are user define data types.

Data Abstraction and Encapsulation

Combining data and functions into a single unit called class and the process is known as Encapsulation. Data encapsulation is important feature of a class. Class contains both data and functions. Data is not accessible from the outside world and only those function which are present in the class can access the data. The insulation of the data from direct access by the program is called data hiding or information hiding. Hiding the complexity of program is called Abstraction and only essential features are represented. In short, we can say that internal working is hidden.

Dynamic Binding

Refers to linking of function call with function definition is called binding and when it is take place at run time called dynamic binding.

Message Passing

The process by which one object can interact with other object is called message passing.

Inheritance

It is the process by which object of a class acquire the properties or features of objects of another class. The concept of inheritance provide the idea of re-usability means we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes.

Polymorphism

An operation may exhibit different behaviors in different instances. The behavior depends upon the types of data used in the operation. Its types:

- Static / Compile Time Polymorphism.
  - Method Overloading and
  - Operator Overloading
- Dynamic / Runtime Polymorphism.

1.4 Scope of . Net Technology

Following are the scope of . Net Technology:

- Cross platform mobile application development
- Desktop application
- Web application
- Web services
- Websites
- Games
- Virtual Reality (VR) application
- Database application, etc.

CHAPTER

2

LANGUAGE BASICS

2.1 Variables and Data Types

2.1.1 Variables

A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times. It is a way to represent memory location through symbol so that it can be easily identified. The basic variable type available in C# can be categorized as:

Variable Type	**Example**
Decimal types	decimal
Boolean types	True or false value, as assigned
Integral types	int, char, byte, short, long
Floating point types	float and double
Nullable types	Nullable data types

C# also allows defining other value types of variable such as `enum` and reference types of variables such as `class`.

Variable Declaration in C#

Syntax for variable definition in C# is:

```
<data_type> <variable_list>;
```

Here, `data_type` must be a valid C# data type including char, int, float, double, or any user-defined data type, and `variable_list` may consist of one or more identifier names separated by commas.

Some valid variable definitions are shown here:

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

Variable Declaration in VB . NET

The `Dim` statement is used for variable declaration and storage allocation for one or more variables. The Dim statement is used at *module*, *class*, *structure*, *procedure* or *block level*. Syntax for variable declaration in VB . Net is:

```
DIM <variable_name> As <data_type>
```

Some valid variable declarations along with their definition are shown here:

```
' VB
Dim StudentID As Integer
Dim StudentName As String
Dim Salary As Double
Dim count1, count2 As Integer
Dim status As Boolean
Dim exitButton As New System.Windows.Forms.Button
Dim lastTime, nextTime As Date
```

Variable Initialization

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is:

```
variable_name = value;
```

Variables can be initialized in their declaration. The initializer consists of an equal sign followed by a constant expression as:

```
' C#
<data_type> <variable_name> = value;

' VB
DIM <variable_name> As <data_type> = value
```

Variable can be initialized at the time of declaration as follows:

```
// C#
string studentName = "Jeevan";
```

```
' VB
Dim semesterID As Integer = 8
Dim teacherName As String = "Madan Uprety"
```

Rules for defining variables

- A variable can have alphabets, digits and underscore.
- A variable name can start with alphabet and underscore only. It can't start with a digit.
- No white space is allowed within variable name.
- A variable name must not be any reserved word or keyword e.g. char, float etc.

Some examples are:

```
// C#
int semesterId = 8, rollNo = 69;
byte x = 22;
double pi = 3.14159;
char name = `x';
```

2.1.2 Data Types

Data types specify the type of data that a valid C# variable can hold. C# is a strongly typed programming language because in C#, each type of data (such as integer, character, float, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described.

Value Data Types

In C#, the Value Data Types will directly store the variable value in memory and it will also accept both signed and unsigned literals. The derived class for these data types are `System.ValueType`

Reference Data Type

The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables.

In other words, they refer to a memory location. Using multiple variables, the reference types can refer to a memory location. If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value. Example of built-in reference types are: object, dynamic, and string.

Object Type In C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from Object. So basically it is the base class for all the data types. Before assigning values, it needs type conversion. When a variable of a value type is converted to object, it's called boxing. When a variable of type object is converted to a value type, it's called unboxing. Its type name is `System.Object`.

String Type It represents a sequence of Unicode characters and its type name is `System.String`. So, `string` and `String` are equivalent. Example:

```
// creating through string keyword
  string s1 = "Corona";

// creating through String class
  String s2 = "COVID-19";
```

Dynamic Type You can store any type of value in the dynamic data type variable. Type checking for these types of variables takes place at run-time.

```
// Syntax
  dynamic <variable_name> = value;

// Example
  dynamic d = 69;
```

Dynamic types are similar to object types except that type checking for object type variables takes place at compile time, whereas that for the dynamic type variables takes place at run time.

Pointer Type

Pointer type variables store the memory address of another type. To get the pointer details we have a two symbols: ampersand (&) and asterisk (*).

- ampersand (&): It is Known as Address Operator. It is used to determine the address of a variable.
- asterisk (*): It also known as Indirection Operator. It is used to access the value of an address.

```
// Syntax
  type* identifier;

// Example
  int* p1, p;   // Valid syntax
  int *p1, *p;   // Invalid
```

2.2 String and String Builder

2.2.1 String

String is a sequence of Unicode characters or array of characters. The array of characters is also termed as the text. So the string is the representation of the text.

A string is represented by class `System.String`. The `string` keyword is an alias for `System.String` class and instead of writing `System.String` one can use `String` which is a shorthand for `System.String` class. So, we can say `string` and `String` both can be used as an alias of `System.String` class. So `string` is an object of `System.String` class.

```
// Example:

// creating the string using string keyword
string courseName = "BCA";

// creating the string using String class
String semester = "Semester-VIII";

// creating the string using String class
System.String calenderYearNP ="2077 B.S.";
```

The String class is defined in the `.NET` base class library. In other words a String object is a sequential collection of `System.Char` objects which represents a string. The maximum size of String object in memory is 2GB or about

1 billion characters. `System.String` class is **immutable**, i.e once created its state cannot be altered.

String Class Properties

The String class has two properties as follows:

- Chars: It is used to get the Char object at a specified position in the current String object.
- Length: It is used to get the number of characters in the current String object.

2.2.2 String Builder

A string instance is immutable. Immutable means once we create a string object we cannot modify the value of the string Object in the memory. Any operation that appears to modify the string, it will discard the old value and it will create new instance in memory to hold the new value.

The `System.Text.StringBuilder` is mutable, that means once we create `StringBuilder` object we can perform any operation that appears to change the value without creating new instance for every time. It can be modified in any way and it doesn't require creation of new instance.

```
// String Example

string colors;
colors += "red";
colors += "blue";
colors += "green";
```

In the above code, string color will alter 3 times, each time the code perform a string operation `(+=)`. That mean 3 new string created in the memory. When you perform repeated operation to a string, the overhead associated with creating a new String object can be costly.

```
// StringBuilder Example

StringBuilder sb = new StringBuilder("");
sb.Append("red");
sb.Append("blue");
sb.Append("green ");
string colors = sb.ToString();
```

In the above code the `StringBuilder` object will alter 3 times, each time the code attempt a `StringBuilder` operation without creating a new object.

That means, using the `StringBuilder` class can boost performance when concatenating many strings together in a loop.

But immutable objects have some advantages also, such as they can be used across threads without fearing synchronization problems. On the other hand, when initializing a `StringBuilder`, you are going down in performance. Also many actions that you do with string can't be done with `StringBuilder` object.

Table 2.1: Differences between String and StringBuilder class

String	**StringBuilder**
`System.String` is immutable	`System.StringBuilder` is mutable
Concatenation is used to combine two strings	Append method is used
The first string is combined to the other string by creating a new copy in the memory as a string object, and then the old string is deleted	Insertion is done on the existing string.
`String` is efficient for small string manipulation	`StringBuilder` is more efficient in case large amounts of string manipulations have to be performed

2.3 Boxing and Unboxing

C# Type System contains three Types, they are:

- Value Types,
- Reference Types and
- Pointer Types.

C# allows us to convert a Value Type to a Reference Type, and back again to Value Types. The operation of Converting a Value Type to a Reference Type is called **Boxing** and the reverse operation is called **Unboxing**.

2.3.1 Boxing

```
int Val = 1;
Object Obj = Val; //Boxing
```

The first line we created a Value Type `Val` and assigned a value to Val. The second line, we created an instance of `Object Obj` and assign the value of Val to Obj. From the above operation `Object Obj = i` we saw converting a value of a Value Type into a value of a corresponding Reference Type. These types of operation is called Boxing.

2.3.2 UnBoxing

```
int Val = 1;
Object Obj = Val; //Boxing
int i = (int)Obj; //Unboxing
```

The first two line shows how to Box a Value Type. The next line int `i = (int) Obj` shows extracts the Value Type from the Object. That is converting a value of a Reference Type into a value of a Value Type. This operation is called UnBoxing.

Boxing and UnBoxing are computationally expensive processes. When a value type is boxed, an entirely new object must be allocated and constructed, also the cast required for UnBoxing is also expensive computationally.

Table 2.2: Difference between boxing and unboxing.

Boxing	**Unboxing**
It convert value type into an object type.	It convert an object type into value type.
Boxing is an implicit conversion process.	Unboxing is the explicit conversion process.
Here, the value stored on the stack copied to the object stored on the heap memory.	Here, the object stored on the heap memory copied to the value stored on the stack .

2.4 Operators

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.

2.4.1 Arithmetic Operators

The arithmetic operators perform arithmetic operations on all the numeric type operands such as sbyte, byte, short, ushort, int, uint, long, ulong, float, double, and decimal.

Operator	Description	Example
`+`	Adds two operands	`int x = 30 + 6;`
`-`	Subtracts second operand from the first	`int x = 70 - 1;`
`*`	Multiplies both operands	`int x = 4 * 9;`
`/`	Divides numerator by denominator	`int x = 138 / 2;`
`%`	Modulus Operator and remainder of after an integer division	`int x = 69 % 36;`
`++`	Increment operator increases integer value by one	`x++;`
`x--`	Decrement operator decreases integer value by one	`x--;`

2.4.2 Relational Operators

Relational operators are used to check the relationship between two operands. If the relationship is true the result will be `true`, otherwise it will result in `false`. Relational operators are used in decision-making and loops. Table 2.3 shows list of comparison operators.

Table 2.3: Comparison Operators.

Operator	Name	Example
`==`	Equal to	`6 == 4` evaluates to false
`>`	Greater than	`3 > -1` evaluates to true
`<`	Less than	`5 < 3` evaluates to false
`>=`	Greater than or equal to	`4 >= 4` evaluates to true
`<=`	Less than or equal to	`5 <= 3` evaluates to false
`!=`	Not equal to	`10 != 2` evaluates to true

2.4.3 Logical Operators

Logical operators are used to perform logical operation such as and, or. Logical operators operates on boolean expressions (true and false) and returns boolean values. Logical operators are used in decision-making and loops. Table 2.4 shows list of logical operators.

Table 2.4: Logical Operators.

Operator	Description	Example
`&&`	Computes the logical AND of its bool operands. Returns true both operands are true, otherwise returns false.	`x && y;`
`||`	Computes the logical OR of its bool operands. Returns true when any one operand is true.	`x || y;`

2.4.4 Compound Assignment Operators

The assignment operator `=` assigns its right-hand value to its left-hand variable, property, or indexer. It can also be used with other arithmetic, Boolean

logical, and bitwise operators. Table 2.5 shows list of compound assignment operators.

Table 2.5: Compound Assignment Operators.

Operator	Example	Same As
=	x = 6	x = 6
+=	x += 4	x = x + 4
-=	x -= 6	x = x - 6
*=	x *= 9	x = x * 9
/=	x /= 7	x = x / 7
%=	x % = 2	x = x % 2

2.4.5 Ternary Operator

The ternary operator `? :` operates on three operands. It is a shorthand for if-then-else statement.

```
// Syntax: ternary operator
variable = Condition? Expression1 : Expression2;
```

The ternary operator works as follows:

- If the expression stated by Condition is `true`, the result of `Expression1` is assigned to variable
- If it is `false`, the result of `Expression2` is assigned to variable

```
// Example
int number = 18;
string result;

result = (number % 2 == 0)? "Even Number" : "Odd Number";
```

2.5 Control statements

Program needs a way to make decisions as it makes progress through the execution of code. This is where control flow comes into play. The program

control flow is provided to us by the use of control statements. These are the special keywords in the C# language that allow the programmer to set up things like branching, looping, and even entire jumps to new points in the program during execution. Control statements:

- **Selection Statements**: This consists of `if`, `else`, `switch`, and `case` branching
- **Iteration Statements**: This consists of `do`, `for`, `foreach`, and `while` looping
- **Jump Statements**: This consists of `break`, `continue`, `return`, and `goto` statements

2.5.1 Conditional Statements

A conditional statement decides whether to execute code based on conditions. The if statement and the switch statement are the two types of conditional statements in C#.

2.5.2 C#

if statement

The if statement selects a branch for execution based on the value of a Boolean expression.

```
// C#
// Syntax: if statement
if (condition) {
// Statement(s)
} else {
// Statement(s)
}
```

The if statement evaluates its condition expression to determine whether to execute the if-body. Optionally, an else clause can immediately follow the if body, providing code to execute when the condition is false. Making the else-body another if statement creates the common cascade of `if, else if, else if, else if, else` statements.

switch statement

Switch statement can be used to branch the execution of a program to a set of statements that are inside of a case label, which is created with the

keyword `case`. The C# switch statement does this by matching the value inside the switch against the values that you specify in each case label.

- Restricted to integers, characters, strings, and enums
- Case labels are constants
- Default label is optional

```
// Syntax: switch statement
switch (expression) {
case value1:
// statements
  break;

case value2:
// statements
  break;
.
.
case valueN:
// statements
  break;
default:
// default statement
}
```

2.5.3 VB

If ... Then Statement

```
' VB Syntax: if ... then statement
If (condition) Then
  ' Statement(s)
End IF
```

2.5.3.1 If ... Then ... Else Statement

An If statement can be followed by an optional Else statement, it executes when the Boolean expression is false.

```
If (boolean_expression) Then
  'execute if Boolean expression is true
Else
  'execute if Boolean expression is false
End If
```

2.5.3.2 If ... ElseIf ... Else Statement

An If statement can be followed by an optional Else if...Else statement, which is very useful to test various conditions using single If...Else If statement.

```
' Syntax for if... else if ... else
If(boolean_expression 1)Then
  ' Executes when the boolean expression 1 is true
ElseIf( boolean_expression 2)Then
  ' Executes when the boolean expression 2 is true
ElseIf( boolean_expression 3)Then
  ' Executes when the boolean expression 3 is true
Else
  ' Executes when the none of the above condition is true
End If
```

2.5.3.3 Nested If Statement

It is always legal in VB.Net to nest If-Then-Else statements, which means you can use one If or ElseIf statement inside another If ElseIf statement(s).

```
If(boolean_expression 1)Then
  'Executes when the boolean expression 1 is true
    If(boolean_expression 2)Then
      'Executes when the boolean expression 2 is true
    End If
End If
```

2.5.3.4 Select Case Statement

A Select Case statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each select case.

```
' VB: Syntax: Select Case
Select (expression)
  Case case1
    'statements
  Case caseN
    'statements
  Case Else
    'statements
End Select
```

Loops in VB

Do loop

It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement.

```
' Syntax: sDo loop
Do (While | Until) (condition)
  ' statements
  ' Continue Do
  ' statements
  ' Exit Do
  ' statements
  Loop
  ' -or-
Do
   'statements
  ' Continue Do
  ' statements
  ' Exit Do
  ' statements
  Loop (While | Until ) (condition)
```

2.5.4 Iteration statements

An iteration statement creates a loop of code to execute a variable number of times. Iterating is also known as looping. Looping is when you need or want to execute a piece of code many times. The `for` loop, the `do` loop, the `while` loop, and the `foreach` loop are the iteration statements in C#.

2.5.4.1 do ... while loop

The `do...while` loop always runs its body once. After its first run, it evaluates its condition to determine whether to run its body again. If the `condition` is true, the body executes. If the condition evaluates to true again after the body has run, the body executes again. When the condition evaluates to false, the `do...while` loop ends.

```
// Syntax: do ... while loop
do {
  statement;
} while (condition);
```

2.5.4.2 for loop

For loop repeats the execution of a statement until a specified `condition` is false. It proves most useful when the developer knows the exact number of necessary iterations. Its statements initialize the iteration variable, state the related condition, and state the iteration operation. Note that all statements within a for statement are optional, and omissions can create infinite loops.

```
// Syntax: for loop
for (initialization; condition; iterator) {
  statement;
}
```

2.5.4.3 foreach loop

The `foreach` statement is similar to the for statement in that both allow code to iterate over the items of collections, but the foreach statement lacks an iteration index, so it works even with collections that lack indices altogether. When the collection is fully traversed, the loop will terminate.

```
// Syntax: foreach loop
foreach(int element in collection) {
  statement;
}
```

2.5.4.4 while

The while loop executes a statement(s) while a condition evaluates to true.

```
// Syntax: while loop
while (statement) {
statement;
}
```

2.5.5 Jump statements

Jump statements are used to transfer control from one point to another point in the program. There are five keywords in the Jump Statements: `break, continue, goto, return and throw`

break statement

The break statement is used to terminate the loop or statement in which it present. After that, the control will pass to the statements that present after

the break statement, if available. If the break statement present in the nested loop, then it terminates only those loops which contains break statement.

continue statement

This statement is used to skip over the execution part of the loop on a certain condition. After that, it transfers the control to the beginning of the loop. Basically, it skips its following statements and continues with the next iteration of the loop.

goto statement

This statement is used to transfer control to the labeled statement in the program. The label is the valid identifier and placed just before the statement from where the control is transferred.

return statement

This statement terminates the execution of the method and returns the control to the calling method. It returns an optional value. If the type of method is void, then the return statement can be excluded.

throw statement

This is used to create an object of any valid exception class with the help of `new` keyword manually. The valid exception must be derived from the Exception class.

2.6 Arrays and Strings

Arrays represent a set of items all belonging to the same type. The declaration itself may use a variable or a constant to define the length of the array. However, an array has a set length and it cannot be changed after declaration.

2.6.1 Array Declaration

The syntax for declaring an array follows:

```
// Syntax
datatype[] arrayName;
```

Where,

- `datatype`: defines the element type of the array
- `[]`: defines the size of the array
- `arrayName`: is the Name of array

```
// Example
int[] totalCount; // can store int values
string[] affectedCountries; // can store string values
double[] economicLoss; // can store double values
CoronaVirus[] covid19; // can store instances of CoronaVirus
    class which is a custom class
```

Only Declaration of an array doesn't allocate memory to the array. For that array must be initialized.

2.6.2 Array Initialization

An array is a reference type so the `new` keyword used to create an instance of the array. We can assign initialize individual array elements, with the help of the index.

```
// Syntax:
datatype[] arrayName= new datatype[size];
```

Where,

- `datatype`: specifies the type of data being allocated
- `size`: specifies the number of elements in the array
- `arrayName`: is the name of array variable
- `new`: allocates memory to an array according to its size

```
 // different ways of array declaration and initialization

// define array without assigning value
int[] year = new int[4];

// define array and assign value at the same time
int[] year = new int[4] {2074, 2075, 2076, 2077};

// defining array with elements
// that indicates the size of an array
int[] year = {2074, 2075, 2076, 2077};
```

Arrays are categorized as one-dimensional, multi-dimensional, or jagged. In the creation of an array, its number of dimensions and their length are set.

One Dimensional Array

This array contains only one row for storing the values. All values of this array are stored contiguously starting from 0 to the array size.

```
//Syntax: 1-D array declaration
double[] xx = new double[69];
```

Multidimensional Arrays

The multi-dimensional array contains more than one row to store the values. It is also known as a **Rectangular Array** in C# because it's each row length is same. It can be a 2D-array or 3D-array or more. Nested loop is required inorder to access multidimensional array.

```
// creates a two-dimensional array of
// four rows and three columns.
int[, ] arrayName = new int[4, 3];

//creates an array of three dimensions, 5, 3, and 2
int[, , ] arrayName = new int[5, 3, 2];

 // one comma indicates 2-dimensional
 // and two commas indicate 3-dimensional.
```

Jagged Arrays

A jagged array is an array whose elements are themselves arrays, not all necessarily of the same length. This is declared in the format

```
//jagged array declaration
datatype[][] arrayName
```

where, we have `[][]` instead of `[,]`.

Example,

```
// Example
int[][] Jag = new int[3][];
```

Before you can use Jag, its elements must be initialized. You can initialize the elements like this:

```
Jag[0] = new int[5]; // array of 5 integers
Jag[1] = new int[4]; // array of 4 integers
Jag[2] = new int[2]; // array of 2 integers
```

Each of the elements is a single-dimensional array of integers.

We can initialize the array when we create it:

```
Jag[0] = new int[] {1, 3, 5, 7, 9};
Jag[1] = new int[] {0, 2, 4, 6};
Jag[2] = new int[] {11, 22};
```

Array can also be initialized upon declaration like this:

```
int[][] Jag = new int[][] {
    new int[] {1, 3, 5, 7, 9},
    new int[] {0, 2, 4, 6},
    new int[] {11, 22}
};
```

The elements of Jag are arrays and thus are reference types, so they are initialized to null. Individual array elements can be accessed like this:

```
Jag[0][1] = 77;
// Assign 77 to the second element ([1]) of the first array
   ([0])
```

Example

```
1  public class JaggedArrayExample
2  {
3      public static void Main()
4      {
5          // Declare the array of two elements
6          int[][] arr = new int[2][];
7
8          // Initialize the elements
9          arr[0] = new int[] { 11, 21, 56, 78 };
10         arr[1] = new int[] { 42, 61, 37, 41, 59, 63 };
11
12         // Display the array elements
13         for (int i = 0; i < arr.Length; i++)
14         {
15             for (int j = 0; j < arr[i].Length; j++)
16             {
17                 System.Console.Write(arr[i][j] + " ");
18             }
19             System.Console.WriteLine();
20         }
21     }
```

```
22 }
23
24 /* output:
25 11 21 56 78
26 42 61 37 41 59 63
27 */
```

Listing 2.1: Example of jagged array

2.7 Procedure and Functions

2.7.1 Procedure

A procedure is a block of Visual Basic statements enclosed by a declaration statement (`Function, Sub, Operator, Get, Set`) and a matching `End` declaration. All executable statements in Visual Basic must be within some procedure.

A function allows you to encapsulate a piece of code and call it from other parts of your code. Functions are useful to improve the code reusability by reducing the code duplication.

Types of Procedures

Visual Basic uses several types of procedures:

Sub Procedures Perform actions but do not return a value to the calling code.

Event-handling procedures Are `Sub` procedures that execute in response to an event raised by user action or by an occurrence in a program.

Function Procedures Return a value to the calling code. They can perform other actions before returning.

Property Procedures Return and assign values of properties on objects or modules.

Operator Procedures Define the behavior of a standard operator when one or both of the operands is a newly-defined class or structure.

Generic Procedures in Visual Basic define one or more type parameters in addition to their normal parameters, so the calling code can pass specific data types each time it makes a call.

Sub Procedure

A Sub procedure is a series of Visual Basic statements enclosed by the `Sub` and `End Sub` statements. The Sub procedure performs a task and then *returns control to the calling code*, but it *does not return a value* to the calling code.

We can define a Sub procedure in *modules, classes, and structures*. By default, it is `Public`.

The term *method* describes a `Sub` or `Function` procedure that is accessed from outside its defining module, class, or structure.

A Sub procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

Syntax

```
' Syntax: Sub procedure
[modifiers] Sub SubName[(parameterList)]
' Statements of the Sub procedure.
End Sub
```

Where,

- `Modifiers`: specify the access level of the procedure; possible values are - Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
- `SubName`: indicates the name of the Sub.
- `ParameterList`: specifies the list of the parameters.

Example

Example shown in Listing 2.2 demonstrates a Sub procedure `CalculatePay` that takes two parameters `hours` and `wages` and displays the total pay of an employee.

```
1 Module mysub
2    Sub CalculatePay(ByRef hours As Double, ByRef wage As Decimal
     )
3       'local variable declaration
4       Dim pay As Double
5       pay = hours * wage
```

```
6         Console.WriteLine("Total Pay: Rs {0}", pay)
7     End Sub
8     Sub Main()
9         'calling the CalculatePay Sub Procedure
10        CalculatePay(25, 10)
11        CalculatePay(40, 20)
12        CalculatePay(30, 27.5)
13        Console.ReadLine()
14    End Sub
15 End Module
```

Listing 2.2: Example of sub procedure.

2.7.2 Function

A Function procedure is a series of Visual Basic statements enclosed by the `Function` and `End Function` statements. The Function procedure performs a task and then returns control to the calling code. When it returns control, it also *returns a value* to the calling code.

We can define a Function procedure in a module, class, or structure. It is Public by default.

A Function procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

Syntax

```
1 [Modifiers] Function FunctionName [(ParameterList)] As
     ReturnType
2   [Statements]
3 End Function
```

Where,

- `Modifiers`: specify the access level of the function; possible values are: Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
- `FunctionName`: indicates the name of the function.
- `ParameterList`:specifies the list of the parameters.
- `ReturnType`: specifies the data type of the variable the function returns.

Example

Example shown in Listing 2.3 demonstrates the example of function in VB.

```
Module myfunctions
   Function FindMax(ByVal num1 As Integer, ByVal num2 As Integer
   ) As Integer
      Dim result As Integer
      If (num1 > num2) Then
          result = num1
      Else
          result = num2
      End If
      FindMax = result
   End Function
   Sub Main()
      Dim a As Integer = 2076
      Dim b As Integer = 2077
      Dim res As Integer
      res = FindMax(a, b)
      Console.WriteLine("Max value is : {0}", res)
      Console.ReadLine()
   End Sub
End Module
```

Listing 2.3: Use of VB function to return max number

CHAPTER

3

DEVELOPING CONSOLE APPLICATION

A console application is an application that takes input and displays output at a command line console with access to three basic data streams: standard input, standard output and standard error.

A console application facilitates the reading and writing of characters from a console — either individually or as an entire line. It is the simplest form of a C# program and is typically invoked from the Windows command prompt. A console application usually exists in the form of a standalone executable file with minimal or no graphical user interface (GUI).

The program structure of a console application facilitates a sequential execution flow between statements. Designed for the keyboard and display screen, a console application is driven by keyboard and system events generated by network connections and objects.

A console application is primarily designed for the following reasons:

- To provide a simple user interface for applications requiring little or no user interaction, such as samples for learning C# language features and command-line utility programs.
- Automated testing, which can reduce automation implementation resources.

Console applications have one main entry point (static main method) of execution, which takes an optional array of parameters as its only argument for command-line parameter representation.

The `.NET` Framework provides library classes to enable rapid console application development with output display capability in different formats. `System.Console` (a sealed class) is one of the main classes used in the development of console applications.

3.1 Entry Point Method

The `Main` method is the entry point of a C# application. (Libraries and services do not require a Main method as an entry point.) When the application is started, the Main method is the first method that is invoked. Main is declared inside a class or struct. Main must be static and it need not be public.

There can only be one entry point in a C# program. If you have more than one class that has a Main method, you must compile your program with the main compiler option to specify which Main method to use as the entry point.

```
// entry point method
class EntryPointMethod {
static void Main(string[] args) {
// Display the number of command line arguments.
  Console.WriteLine(args.Length);
  }
}
```

Where,

- `static`: It means Main Method can be called without an object.
- `public`: It is access modifiers which means the compiler can execute this from anywhere.
- `void`: The Main method doesn't return anything.
- `Main()`: It is the configured name of the Main method.
- `string[] args`: For accepting the zero-indexed command line arguments. `args` is the user-defined name. So you can change it by a valid identifier. `[]` must come before the args otherwise compiler will give errors.

3.2 Command Line Parameters

In some situations, you have to send command line parameters to your application. When you pass arguments, the command line input, read from the standard entry point as string array. You can send arguments to the Main method by defining the method in one of the following ways:

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

OR

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

The arguments of the Main method is a String array that represents the command-line arguments. Usually you determine whether arguments exist by testing the `Length` property.

```
if (args.Length == 0) {
  System.Console.WriteLine("No arguments found!!");
  return 1;
}
```

Listing 3.1 shows the use of command line argument.

```
1  using System;
2  namespace BoCA
3  {
4      class Program
5      {
6          // Main function, execution entry point of the program
7          static void Main(string[] args) // string type
       parameters
8          {
9              // Command line arguments
10             Console.WriteLine("Arg length: " + args.Length);
11             Console.WriteLine("Given arguments:");
12             foreach (Object obj in args)
13             {
14                 Console.WriteLine(obj);
15             }
16         }
17     }
18 }
```

Listing 3.1: Example of command line argument

3.3 Compiling and Building Projects

The code below will demonstrate a C# program written in a simple text editor. Start by saving the following code to a text file called `hello.cs`:

```
namespace CompilingCsharpCode
{
    class CompileCsharpCommandLine
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Hello, World!");
            Console.ReadLine();
        }
    }
}
```

Listing 3.2: Compiling Csharp program

To compile `hello.cs`, run the following from the command line:

- For standard Microsoft installations of . NET 4.7, run

  `c:\windows\microsoft.net\framework\v4.7.0\csc.exe hello.cs`

- For Mono run `mcs hello.cs`

Doing so will produce hello.exe. The following command will run hello.exe:

- On Windows, use `hello.exe` using CMD/Powershell
- On Linux, use `mono hello.exe` using Terminal

3.4 Using control Statements in Console Application

```
using System;

public class Functions {
  public static long Factorial(int n) {
    // Test for invalid input.
    if ((n < 0) || (n > 20)) {
      return -1;
    }
```

```
9
10      // Calculate the factorial iteratively rather than
        recursively.
11      long tempResult = 1;
12      for (int i = 1; i <= n; i++) {
13        tempResult *= i;
14      }
15      return tempResult;
16    }
17  }
18
19  class MainClass {
20    static int Main(string[] args) {
21      // Test if input arguments were supplied.
22      if (args.Length == 0) {
23        Console.WriteLine("Please enter a numeric argument.");
24        Console.WriteLine("Usage: Factorial <num>");
25        return 1;
26      }
27
28      /* Try to convert the input arguments to numbers. This will
        throw an exception if the argument is not a number.
29      num = int.Parse(args[0]); */
30      int num;
31      bool test = int.TryParse(args[0], out num);
32      if (!test) {
33        Console.WriteLine("Please enter a numeric argument.");
34        Console.WriteLine("Usage: Factorial <num>");
35        return 1;
36      }
37
38      // Calculate factorial.
39      long result = Functions.Factorial(num);
40
41      // Print result.
42      if (result == -1)
43        Console.WriteLine("Input must be >= 0 and <= 20.");
44      else
45        Console.WriteLine("The Factorial of {num} is {result}."
        num, result);
46
47      return 0;
48    }
49  }
```

Listing 3.3: Using control statements in console application

CHAPTER

4

ESSENTIALS OF OBJECT–ORIENTED PROGRAMMING

4.1 Object and Class Definition Working

4.1.1 Class

A class is a group of different data members or objects with the same properties, processes, events of an object, and general relationships to other member functions. Furthermore, we can say that it is like a template or architect that tells what data and function will appear when it is included in a class object. For example, it represents the method and variable that will work on the object of the class.

Class Creation

C#

```
// Syntax of class creation in C#
access_modifier class class_name {
```

```
  // fields
}
```

VB

We can create a class using the `Class` keyword, followed by the class name. And the body of the class ended with the statement `End Class`.

```
[Access_Specifier] [Shadows] [MustInherit | NotInheritable] [
    Partial] Class ClassName
  ' Data Members
  ' Methods
  ' Statements
End Class
```

Where,

- `Access_Specifier`: It defines the access levels of the class, such as Public, Private or Friend, Protected, Protected Friend, etc. to use the method. (It is an optional parameter).
- `Shadows`: It is an optional parameter. It represents the re-declaration of variables and hides an identical element name or set of overloaded elements in a base class.
- `MustInherit`: It is an optional parameter that specifies that the class can only be used as a base class, and the object will not directly access the base class or the abstract class.
- `NotInheritable`: It is also an optional parameter that representing the class not being used as a base class.
- `Partial`: As the name defines, a Partial represents the partial definition of the class (optional).
- `Implements`: It is used to specify interfaces from which the class inherits (optional).

4.1.2 Object

Objects are the basic run-time units of a class. Once a class is defined, we can create any number of objects related to the class to access the defined properties and methods.

Synatx:

```
Dim Obj_Name As Class_Name = New Class_Name() ' Declaration of
  object
Obj_Name.Method_Name() 'Access a method
```

An object is basically a block of memory that has been allocated and configured according to the blueprint. 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. Client code is the code that uses these variables to call the methods and access the public properties of the object. In an object-oriented language such as C#, a typical program consists of multiple objects interacting dynamically. 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.

Instances of classes are created by using the new operator. In the following example, `Person` is the type and `person1` is instances, or object, of that type.

```
public class Person {
  public string Name {
    get; set;
  }
  public int Age {
    get; set;
  }
  public Person(string name, int age) {
    Name = name;
    Age = age;
  }
}

class Program {
  static void Main() {
    Person person1 = new Person("JP", 40);
    Console.WriteLine("person1 Name = {0} Age = {1}", person1.
   Name, person1.Age);
  }
}
```

Listing 4.1: Example of class using C#

```
Imports System
Module Class_Example
    Sub Main()
        Dim rect As Rectangle = New Rectangle() 'create an
   object
```

```
5         Dim rect2 As Rectangle = New Rectangle() 'create an
    object
6         Dim area, para As Integer
7
8         rect.setLength = (60)
9         rect.setBreadth= (50)
10
11        rect2.setLength = (1000)
12        rect2.setBreadth = (600)
13
14        area = rect.length * rect.Breadth
15
16        Console.WriteLine(" Area of Rectangle is {0}", area)
17
18        para = 2 (rect2.length + rect.Breadth)
19
20        Console.WriteLine(" Parameter of Rectangle is {0}", para
    )
21        Console.WriteLine(" Press any key to exit...")
22        Console.ReadKey()
23    End Sub
24
25    Public Class Rectangle
26        Public length As Integer
27        Public Breadth As Integer
28
29        Public Sub setLength(ByVal len As Integer)
30            length = len
31        End Sub
32
33        Public Sub setBreadth(ByVal bre As Integer)
34            Breadth = bre
35        End Sub
36        Public Function GetArea() As Integer
37            Return length * Breadth
38        End Function
39
40        Public Function GetParameter() As Integer
41            Return 2 * (length + Breadth)
42        End Function
43    End Class
44 End Module
```

Listing 4.2: Example of class using VB

4.2 Understanding Identity, State and Behavior

All objects have three essential features:

1. State
2. Behavior
3. Identity

States

States are the conditions in which objects exist. An object's state is defined by its attributes. An object's attributes are usually static, and the values of the attributes are usually dynamic.

Behavior

The term *behavior* refers to how objects interact with each other, and it is defined by the operations an object can perform.

Identity

Identity is an uniqueness of an object from other objects. No matter what its attributes and operations are, an object is always uniquely itself. It retains its identity regardless of changes to its state or behavior.

Together, state and behavior define the roles that an object may play. And an object may play many roles during its lifetime.

For example, an object in the bank's Employee class could be involved with the payroll system, with the customer database, or with the command hierarchy.

The functions you require an object to perform are its responsibilities. And an object's responsibilities are fulfilled by its roles — by its state and behavior combined.

So, you can capture all the meaningful functionality of an object by specifying its state and behavior. Objects are characterized by a third feature in addition to state and behavior — identity.

Table 4.1: Comparison between class and object

Object	Class
Object is an instance of a class	Class is a blueprint or template from which objects are created
Object is a real world entity	Class is a group of similar objects
Object is a physical entity	Class is a logical entity
Object is created through `new` keyword mainly `Student student= new Student();`	Class is declared using `class` keyword `class Student{}`
Object is created many times as per requirement	Class is declared once
Object allocates memory when it is created	Class doesn't allocated memory when it is created

4.3 Using Encapsulation to Combine Methods and Data in a Single Class

Encapsulation is the first pillar or principle of object-oriented programming. In simple words, "Encapsulation is a process of binding data members (variables, properties) and member functions (methods) into a single unit". And Class is the best example of encapsulation.

Important points

- Through encapsulation a class can hide the internal details of how an object does something. Encapsulation solves the problem at the implementation level.
- A class or structure can specify how accessible each of its members (variables, properties, and methods) are from outside the class or structure. Encapsulation simplifies the interaction between objects. An object can use another object without knowing all its data or how its data is maintained. For example, a Client object might have name, address, company, and department properties. If a Bank object wants

to use a Client object, it can request the name and address for the bank without needing to know the company and department details of the Client object.

- With the help of encapsulation, a class can change the internal implementation without hurting the overall functionality of the system.
- Encapsulation protects abstraction.

Need or purpose of encapsulation

- To hide and prevent code (data) from the outside world (here the world means other classes and assemblies).
- To prevent code (data) from accidental corruption due to programming errors so that we can deliver expected output. Due to programming mistakes, code may not behave properly and it has an effect on data and then it will affect the functionality of the system. With encapsulation, we can make variables, properties, and methods private so it is not accessible to all but accessible through proper channels only to protect it from accidental corruption from other classes.
- To have a class better control over its fields (validating values etc).

Ways to achieve encapsulation with code example

We can achieve encapsulation by the following ways. Take a look at the methods to achieve encapsulation with code example:

By using the get and set methods (Accessors and Mutators)

```
public class Account {
  private string accoutName;
  // get methods
  public string GetAccount() {
    return accoutName;
  }
  // Set method
  public void SetAccount(string name) {
    accoutName = name;
  }
}
static void Main() {
  string name = "SAVING_ACCOUNT";
```

```
14    Account account = new Account();
15    account.SetAccount(name);
16    name = string.Empty;
17    name = account.GetAccount();
18  }
```

Listing 4.3: Encapsulation using get and set methods

In the above example we use the get and set methods (`GetAccount` and `SetAccount`) to return account and set account name. We use the private variable `accountName` and as it is not accessible directly, to use this variable, we use the get and set methods.

By using properties (read only properties, write only properties)

Like the above example we can also achieve encapsulation using properties . We can use a property (which has a get and set part), or we can use a read only property (which has only a get part) or we can also use a write only property (which has only a set part). But in all cases we can achieve encapsulation.

Have a look at the following example using properties.

```
1  // Encapsulation using properties
2  public class Account {
3    private string accoutName = "SAVING_ACCOUNT";
4
5    // property which has get and set
6    public string AccoutName {
7      get {
8        return accoutName;
9      }
10     set {
11       accoutName = value;
12     }
13   }
14   private string address = "NPL";
15
16   // readonly property
17   public string Address {
18     get {
19       return address;
20     }
21   }
22
23   private string phone = "1234567890";
24
25   // writeonly property
26   public string Phone {
```

```
    set {
      phone = value;
    }
  }
}
static void Main() {

  // Encapsulation using properties
  string name = string.Empty;
  Account account = new Account();

  // call get part
  name = account.AccoutName;

  // change the value
  name = "CURRENT_ACCOUNT";

  // call set part
  account.AccoutName = name;
  string address = string.Empty;

  // call readonly property
  address = account.Address;  // now address has value "NPL"

  string phone = "1234567890";

  // call writeonly property
  account.Phone = phone; // now account.Phone has value
    "1234567890"
}
```

Listing 4.4: Encapsulation using properties

Here, when we create a new instance of the account class, all the private variables in the account class (account name, address, and phone) are assigned with values. In the main class we can skip the variables (name, address, and phone) and directly use `System.Console` to write the output.

Using an interface

```
IAccount myAccount = new AsianAccount();
IAccount myAccount = new EuropeanAccount();
IAccount myAccount = new USAAccount();
```

Now, based on the current location (for which we can have variables), whenever we want to view the balance information of a specific account, we can use the `IAccount` interface and we can see how `AsianAccount`,

`EuropeanAccount`, and `USAAccount` hide information from each other, getting the balance details through the `IAccount` interface.

4.4 Inheritance and Polymorphism With Interface

Inheritance

Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.

The class which inherits the members of another class is called derived class and the class whose members are inherited is called base class.

Syntax

```
<acess-specifier> class <base_class> {
    ...
 }

class <derived_class> : <base_class> {
    ...
}

```

Example

```
using System;

namespace InheritanceApplication {
  class Shape {
    public void setWidth(int w) {
      width = w;
    }
    public void setHeight(int h) {
      height = h;
    }
    protected int width;
    protected int height;
  }

  // Derived class
  class Rectangle: Shape {
    public int getArea() {
```

```
18        return (width * height);
19      }
20    }
21    class RectangleTester {
22      static void Main(string[] args) {
23        Rectangle Rect = new Rectangle();
24
25        Rect.setWidth(5);
26        Rect.setHeight(7);
27
28        // Print the area of the object.
29        Console.WriteLine("Total area: {0}", Rect.getArea());
30        Console.ReadKey();
31      }
32    }
33  }
```

Listing 4.5: Example of inheritance

Interface

Interface is a blueprint of a class. It is like abstract class because all the methods which are declared inside the interface are abstract methods. It cannot have method body and cannot be instantiated.

It is used to achieve multiple inheritance which can't be achieved by class. It is used to achieve fully abstraction because it cannot have method body.

Its implementation must be provided by class or struct. The class or struct which implements the interface, must provide the implementation of all the methods declared inside the interface.

Following example show the use of interface which has `draw()` method. Its implementation is provided by two classes: Rectangle and Circle.

```
1  using System;
2  public interface Drawable {
3    void draw();
4  }
5  public class Rectangle: Drawable {
6    public void draw() {
7      Console.WriteLine("drawing rectangle...");
8    }
9  }
10 public class Circle: Drawable {
11   public void draw() {
12     Console.WriteLine("drawing circle...");
13   }
14 }
```

```
15 public class TestInterface {
16   public static void Main() {
17     Drawable d;
18     d = new Rectangle();
19     d.draw();
20     d = new Circle();
21     d.draw();
22   }
23 }
```

Listing 4.6: Example of interface

Polymorphism

There are two types of polymorphism in:

- Compile time polymorphism

  Compile time polymorphism is achieved by method overloading and operator overloading. It is also known as static binding or early binding.

- Runtime polymorphism

  Runtime polymorphism in achieved by method overriding which is also known as dynamic binding or late binding.

4.4.1 Inheritance With Interface

Polymorphism is often referred to as the third pillar of object-oriented programming, after encapsulation and inheritance. Polymorphism is a Greek word that means “many form” and it has two distinct aspects:

- At run time, objects of a derived class may be treated as objects of a base class.

- Base classes may define and implement virtual methods, and derived classes can override them, which means they provide their own definition and implementation. At run-time, when client code calls the method, the CLR looks up the run-time type of the object, and invokes that override of the virtual method.

Virtual methods enable you to work with groups of related objects in a uniform way. For example, suppose you have a drawing application that enables a user to create various kinds of shapes on a drawing surface. You do

not know at compile time which specific types of shapes the user will create. However, the application has to keep track of all the various types of shapes that are created, and it has to update them in response to user mouse actions. You can use polymorphism to solve this problem in two basic steps:

- Create a class hierarchy in which each specific shape class derives from a common base class
- Use a virtual method to invoke the appropriate method on any derived class through a single call to the base class method.

First, create a base class called `Shape`, and derived classes such as Rectangle, Circle, and Triangle. Give the Shape class a virtual method called Draw, and override it in each derived class to draw the particular shape that the class represents. Create a `List<Shape>` object and add a Circle, Triangle and Rectangle to it. To update the drawing surface, use a foreach loop to iterate through the list and call the Draw method on each Shape object in the list. Even though each object in the list has a declared type of Shape, it is the run-time type (the overridden version of the method in each derived class) that will be invoked.

```
using System;
using System.Collections.Generic;
public class Shape {
  // A few example members
  public int X {
    get;
    private set;
  }
  public int Y {
    get;
    private set;
  }
  public int Height {
    get;
    set;
  }
  public int Width {
    get;
    set;
  }
  // Virtual method
  public virtual void Draw() {
    Console.WriteLine("Performing base class drawing tasks");
  }
}
```

```
26 class Circle: Shape {
27   public override void Draw() {
28     // Code to draw a circle...
29     Console.WriteLine("Drawing a circle");
30     base.Draw();
31   }
32 }
33 class Rectangle: Shape {
34   public override void Draw() {
35     // Code to draw a rectangle...
36     Console.WriteLine("Drawing a rectangle");
37     base.Draw();
38   }
39 }
40 class Triangle: Shape {
41   public override void Draw() {
42     // Code to draw a triangle...
43     Console.WriteLine("Drawing a triangle");
44     base.Draw();
45   }
46 }
47 class Program {
48   static void Main(string[] args) {
49 /* Polymorphism at work #1: a Rectangle, Triangle and Circle can
       all be used where ever a Shape is expected. No cast is
     required because an implicit conversion exists from a derived
       class to its base class.
50     */
51     var shapes = new List < Shape > {
52       new Rectangle(),
53       new Triangle(),
54       new Circle()
55     };
56     /* Polymorphism at work #2: the virtual method Draw is
     invoked on each of the derived classes, not the base class.*/
57
58     foreach(var shape in shapes) {
59       shape.Draw();
60     }
61     // Keep the console open in debug mode.
62     Console.WriteLine("Press any key to exit.");
63     Console.ReadKey();
64   }
65 }
66 /* Output:
67 Drawing a rectangle
68 Performing base class drawing tasks
69 Drawing a triangle
70 Performing base class drawing tasks
```

```
Drawing a circle
Performing base class drawing tasks
*/
```

Listing 4.7: Example of inheritance with polymorphism

Inheritance With Interface

We can implement the interface for those classes where behavior is same but definitions are different to achieve polymorphism. For example, We are having 3 classes — Human, Fish and Car, Practically, these Human, Fish and Car can move but way/ definition of moving is not identical.

Benefits

- We are clearly able to group by the fact that they share common behavior.
- You need only single list to hold movable object, not multiple list of each type of object.

```
Using System;
using System.Collections.Generic;
namespace InheritanceInterface {
  interface IMovable {
    void Move();
  }
  class Human: IMovable {
    public void Move() // code that defines how human moves
    {
      Console.WriteLine("I am a Human, I move by Walking");
    }
  }
  class Fish: IMovable {
    public void Move() // code that defines how fish moves
    {
      Console.WriteLine("I am a Fish, I move by swimming");
    }
  }
  class Car: IMovable {
    public void Move() // code that defines how car moves
    {
      Console.WriteLine("I am Car, I move By wheel");
    }
  }
  class Program {
```

```
26      static void Main(string[] args) {
27        List < IMovable > lstMovable = new List < IMovable > {
28          new Human(),
29          new Fish(),
30          new Car()
31        };
32        foreach(IMovable movableType in lstMovable) {
33          movableType.Move();
34        }
35        Console.ReadLine();
36      }
37    }
38  }
```

Listing 4.8: Example of inheritance with interface

CHAPTER

5

WINDOWS FORMS AND STANDARD COMPONENTS

5.1 Introduction

- Windows Forms is a Graphical User Interface (GUI) class library which is bundled in . Net Framework.
- Its main purpose is to provide an easier interface to develop the applications for desktop, tablet, PCs.
- It is also termed as the *WinForms*.
- The applications which are developed by using Windows Forms or WinForms are known as the Windows Forms Applications that runs on the desktop computer.
- `WinForms` can be used only to develop the Windows Forms Applications not web applications.
- `WinForms` applications can contain the different type of controls like labels, list boxes, tooltip etc.

5.2 Basic controls

Windows Forms controls are reusable components that encapsulate user interface functionality and are used in client side Windows applications.

5.2.1 Label Control

- Displays text in a set location on the page.
- Can also be used to add descriptive text to a Form to provide the user with helpful information.
- The Label class is defined in the `System.Windows.Forms` namespace.

```
label1.Text = "This is my first Label";
```

Label control can also display an image using the Image property, or a combination of the `ImageIndex` and `ImageList` properties.

```
label1.Image = Image.FromFile("C:\\testimage.jpg");
```

The following C# source code shows how to set some properties of the Label through coding.

```
1  using System;
2  using System.Drawing;
3  using System.Windows.Forms;
4
5  namespace WindowsFormsApplication1 {
6    public partial class Form1: Form {
7      public Form1() {
8        InitializeComponent();
9      }
10     private void Form1_Load(object sender, EventArgs e) {
11       label1.Text = "This is my first label";
12       label1.BorderStyle = BorderStyle.FixedSingle;
13       label1.TextAlign = ContentAlignment.MiddleCenter;
14     }
15   }
16 }
```

Listing 5.1: Set properties of label control

5.2.2 Button Control

- A button is a control, which is an interactive component that enables users to communicate with an application.
- The Button class inherits directly from the `ButtonBase` class.
- A Button can be clicked by using the mouse, ENTER key, or SPACE-BAR if the button has focus.

When you want to change display text of the Button , you can change the Text property of the button.

```
button1.Text = "Click Here";
```

Similarly, to load an Image to a Button control

```
button1.Image = Image.FromFile("C:\\testimage.jpg");
```

Call a Button's Click Event

The Click event is raised when the Button control is clicked. This event is commonly used when no command name is associated with the Button control. Raising an event invokes the event handler through a delegate.

```
private void Form1_Load(object sender, EventArgs e)
```

The following C# source code shows how to change the button Text property while Form loading event and to display a message box when pressing a Button Control.

```
1  using System;
2  using System.Drawing;
3  using System.Windows.Forms;
4
5  namespace WindowsFormsApplication1 {
6    public partial class Form1: Form {
7      public Form1() {
8        InitializeComponent();
9      }
10     private void Form1_Load(object sender, EventArgs e) {
11       button1.Text = "Click Here";
12     }
13     private void button1_Click(object sender, EventArgs e) {
14       MessageBox.Show("BCA-VIII::2077");
15     }
16   }
17 }
```

Listing 5.2: Button control

5.2.3 TextBox Control

- A TextBox control is used to display, or accept as input, a single line of text.
- This control has additional functionality that is not found in the standard Windows text box control, including multi-line editing and password character masking.

For displaying a text in a TextBox control:

```
textBox1.Text = "Dot net Programming";
```

You can also collect the input value from a TextBox control to a variable like this way.

```
string var;
```

From the following C# source code you can see some important property settings to a TextBox control.

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsFormsApplication1 {
  public partial class Form1: Form {
    public Form1() {
      InitializeComponent();
    }
    private void Form1_Load(object sender, EventArgs e) {
      textBox1.Width = 250;
      textBox1.Height = 50;
      textBox1.Multiline = true;
      textBox1.BackColor = Color.Blue;
      textBox1.ForeColor = Color.White;
      textBox1.BorderStyle = BorderStyle.Fixed3D;
    }
    private void button1_Click(object sender, EventArgs e) {
      string val;
      val = textBox1.Text;
      MessageBox.Show(val);
    }
  }
```

```
24 }
```

Listing 5.3: Setting some textbox control property

5.2.4 ComboBox Control

- C# controls are located in the Toolbox of the development environment, and you use them to create objects on a form with a simple series of mouse clicks and dragging motions.
- A `ComboBox` displays a text box combined with a `ListBox`, which enables the user to select items from the list or enter a new value.
- The user can type a value in the text field or click the button to display a drop down list. You can add individual objects with the `Add` method.
- You can delete items with the `Remove` method or clear the entire list with the `Clear` method.

Add item to combobox

```
comboBox1.Items.Add("Hami yahan chhaun");
```

Remove item from combobox

You can remove items from a combobox in two ways. You can remove item at the specified index or giving a specified item by name.

```
// removes second item from the combobox
comboBox1.Items.RemoveAt(1);

// removes "Hami yahan chhaun" from the combobox
comboBox1.Items.Remove("Hami yahan chhaun");
```

DropDownStyle

The DropDownStyle property specifies whether the list is always displayed or whether the list is displayed in a drop-down. The DropDownStyle property also specifies whether the text portion can be edited.

```
comboBox1.DropDownStyle = ComboBoxStyle.DropDown;
```

ComboBox Example

The following C# source code add four districts to a combo box while load event of a Windows Form and int Button click event it displays the selected text in the Combo Box.

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsFormsApplication1 {
  public partial class Form1: Form {
    public Form1() {
      InitializeComponent();
    }
    private void Form1_Load(object sender, EventArgs e) {
      comboBox1.Items.Add("Taplejung");
      comboBox1.Items.Add("Panchthar");
      comboBox1.Items.Add("Ilam");
      comboBox1.Items.Add("Jhapa");
      comboBox1.SelectedIndex = comboBox1.FindStringExact("
    Taplejung");

    }
    private void button1_Click(object sender, EventArgs e) {
      string var;
      var = comboBox1.Text;
      MessageBox.Show(var);
    }
  }
}
```

Listing 5.4: Combobox control

5.2.5 PictureBox Control

The Windows Forms PictureBox control is used to display images in bitmap, GIF, icon, or JPEG formats.

You can set the Image property to the Image you want to display, either at design time or at run time. You can programmatically change the image displayed in a picture box, which is particularly useful when you use a single form to display different pieces of information.

```
  pictureBox1.Image = Image.FromFile("c:\\testImage.jpg");
```

The SizeMode property, which is set to values in the PictureBoxSizeMode enumeration, controls the clipping and positioning of the image in the display area.

```
pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
```

There are five different PictureBoxSizeMode is available to PictureBox control.

- `AutoSize` - Sizes the picture box to the image.
- `CenterImage` - Centers the image in the picture box.
- `Normal` - Places the upper-left corner of the image at upper left in the picture box
- `StretchImage` - Allows you to stretch the image in code

The PictureBox is not a selectable control, which means that it cannot receive input focus. The following C# program shows how to load a picture from a file and display it in streach mode.

5.3 Menu and Context Menus

5.3.1 Menu

- The Menu control presents a list of items that specify commands or options for an application.
- Typically, clicking an item on a menu opens a submenu or causes an application to carry out a command.
- A Menu on a Windows Form is created with a `MainMenu` object, which is a collection of `MenuItem` objects.
- `MainMenu` is the container for the Menu structure of the form and menus are made of MenuItem objects that represent individual parts of a menu.
- You can add menus to Windows Forms at design time by adding the `MainMenu` component and then appending menu items to it using the Menu Designer.

5.3.2 Context Menus

- The `ContextMenu` class represents the element that exposes functionality by using a context specific Menu.
- Typically, a user exposes the `ContextMenu` in the user interface (UI) by right clicking the mouse button.
- A `ContextMenu` is attached to a specific control.
- The `ContextMenu` element enables you to present users with a list of items that specify commands or options that are associated with a particular control, for example, a Button.
- Users right-click the control to make the menu appear.
- Typically, clicking a `MenuItem` opens a submenu or causes an application to carry out a command.
- Context menu is dependent on context. For example, right clicking on desktop displays different menu whereas right clicking on image in image viewer displays different menus.

5.4 Menu Strip, Toolbar Strip

5.4.1 Menu Strip

- The `MenuStrip` control is the container for the menu structure of an application.
- `MenuStrip` is derived from the `ToolStrip` class.
- The menu system is built by adding `ToolStripMenu` objects to the `MenuStrip`.

Use the `MenuStrip` control to:

- Create easily customized, commonly employed menus that support advanced user interface and layout features, such as text and image ordering and alignment, drag-and-drop operations, MDI, overflow, and alternate modes of accessing menu commands.
- Support the typical appearance and behavior of the operating system.
- Handle events consistently for all containers and contained items, in the same way you handle events for other controls.

5.4.2 ToolStrip

- `ToolStrip` is the base class for `MenuStrip`, `StatusStrip`, and `ContextMenuStrip`.
- The `ToolStrip` control is a container control used to create toolbars, menu structures, and status bars.
- While used as a toolbar, the `ToolStrip` control uses a set of controls based on the abstract `Tool.StripItem` class.

5.5 Group box and Panel

5.5.1 Group box

GroupBox represents a control that creates a container that has a border and a header for user interface (UI) content.

5.5.2 Panel

- Windows Forms `Panel` controls are used to provide an identifiable grouping for other controls.
- Typically, you use panels to subdivide a form by function.
- The `Panel` control is similar to the `GroupBox` control; however, only the Panel control can have scroll bars, and only the `GroupBox` control displays a caption.
- Panels do not show a border by default.
- `Panel` is the base class for the `FlowLayoutPanel`, `TableLayoutPanel`, `TabPage`, and `SplitterPanel`.

Panel	**GroupBox**
It does not have the `Text` property	It has the `Text` property

Panel	GroupBox
We can display scroll bars on Panel if the height/width of the child controls exceeds that of the Panel. For that set `AutoScroll` property to true	We cannot display scroll bars on `GroupBox`
It has the click event and other events like other events `MouseMove`, `MouseDown`, `MouseUp`	It does not have the click event that are missing include MouseMove, MouseDown, MouseUp events
To display border, we have to use the `BorderStyle` property	`BorderStyle` property is not there as Border or Frame is there by default

Table 5.1: Comparison between panel and group box

5.6 ListBox

- A Windows Forms `ListBox` control displays a list from which the user can select one or more items.
- If the total number of items exceeds the number that can be displayed, a scroll bar is automatically added to the `ListBox` control.
- When the `MultiColumn` property is set to true, the list box displays items in multiple columns and a horizontal scroll bar appears.
- When the `MultiColumn` property is set to false, the list box displays items in a single column and a vertical scroll bar appears.
- When `ScrollAlwaysVisible` is set to true, the scroll bar appears regardless of the number of items.
- The `SelectionMode` property determines how many list items can be selected at a time.
- The `SelectedIndex` property returns an integer value that corresponds to the first selected item in the list box.
- If no item is selected, the `SelectedIndex` value is -1. If

the first item in the list is selected, the `SelectedIndex` value is 0.

- When multiple items are selected, the `SelectedIndex` value reflects the selected item that appears first in the list.
- The `SelectedItem` property is similar to `SelectedIndex`, but returns the item itself, usually a string value.
- The `Count` property reflects the number of items in the list, and the value of the Count property is always one more than the largest possible `SelectedIndex` value because `SelectedIndex` is zero-based.
- To add or delete items in a `ListBox` control, use the `Add`, `Insert`, `Clear` or `Remove` method.
- Alternatively, you can add items to the list by using the `Items` property at design time.

Add Items in a Listbox

```
public int Add (object item);

listBox1.Items.Add("Sunday");
```

If the Sorted property of the C# ListBox is set to true, the item is inserted into the list alphabetically. Otherwise, the item is inserted at the end of the ListBox.

Insert Items in a Listbox

```
public void Insert (int index, object item);

// inserts an item into the list box at the specified index
listBox1.Items.Insert(0, "First");
```

Listbox Column

A multicolumn ListBox places items into as many columns as are needed to make vertical scrolling unnecessary. The user can use the keyboard to navigate to columns that are not currently visible. First of all, you have Gets or sets a value indicating whether the ListBox supports multiple columns

Remove item from Listbox

```
public void RemoveAt (int index);
```

Listbox Vs ListView Vs GridView

C# ListBox has many similarities with `ListView` or `GridView` (they share the parent class `ItemsControl`), but each control is oriented towards different situations. `ListBox` is best for general UI composition, particularly when the elements are always intended to be selectable, whereas `ListView` or `GridView` are best for data binding scenarios, particularly if virtualization or large data sets are involved. One most important difference is `listview` uses the extended selection mode by default.

Checked ListBox Control

The CheckedListBox control gives you all the capability of a list box and also allows you to display a check mark next to the items in the list box.

The user can place a check mark by one or more items and the checked items can be navigated with the `CheckedListBox.CheckedItemCollection` and `CheckedListBox.CheckedIndexCollection`.

Add items

```
public int Add (object item, bool isChecked);
```

You can add individual items to the list with the Add method. The CheckedListBox object supports three states through the CheckState enumeration: Checked, Indeterminate, and Unchecked.

```
checkedListBox1.Items.Add("Sunday", CheckState.Checked);
```

If you want to add objects to the list at run time, assign an array of object references with the AddRange method. The list then displays the default string value for each object.

```
string[] days = new[] { "Sunday", "Monday", "Tuesday" };
```

By default checkedlistbox items are unchecked .

Check all items

If you want to check an item in a Checkedlistbox, you need to call SetItemChecked with the relevant item.

```
public void SetItemChecked (int index, bool value);
```

Parameters:

- `index(Int32)` - The index of the item to set the check state for.
- `value(Boolean)` - true to set the item as checked; otherwise, false.

5.7 RadioButton adn CheckBox

5.7.1 RadioButton

- A radio button or option button enables the user to select a single option from a group of choices when paired with other `RadioButton` controls.
- When a user clicks on a radio button, it becomes checked, and all other radio buttons with same group become unchecked.
- The `RadioButton` control can display text, an Image, or both.

Use the Checked property to get or set the state of a `RadioButton`.

```
radioButton1.Checked = true;
```

The radio button and the check box are used for different functions. Use a radio button when you want the user to choose only one option. When you want the user to choose all appropriate options, use a check box. Like check boxes, radio buttons support a Checked property that indicates whether the radio button is selected.

```
1  using System;
2  using System.Drawing;
3  using System.Windows.Forms;
4
5  namespace WindowsFormsApplication1 {
6    public partial class Form1: Form {
7      public Form1() {
8        InitializeComponent();
9      }
10     private void Form1_Load(object sender, EventArgs e) {
11       radioButton1.Checked = true;
```

```
12      }
13      private void button1_Click(object sender, EventArgs e) {
14        if (radioButton1.Checked == true) {
15          MessageBox.Show("You've selected Taplejung !! ");
16          return;
17        } else if (radioButton2.Checked == true) {
18          MessageBox.Show("You've selected Jhapa !! ");
19          return;
20        } else {
21          MessageBox.Show("You've selected Sunsari !! ");
22          return;
23        }
24      }
25    }
26  }
```

Listing 5.5: Radio button control

5.7.2 CheckBox

- `CheckBox` allow the user to make multiple selections from a number of options.
- `CheckBox` gives the user an option, such as true/false or yes/no.
- You can click a check box to select it and click it again to deselect it.
- The CheckBox control can display an image or text or both.

Usually CheckBox comes with a caption, which you can set in the Text property.

```
checkBox1.Text = "Java";
```

You can use the `CheckBox` control `ThreeState` property to direct the control to return the `Checked`, `Unchecked`, and `Indeterminate` values. You need to set the check box `ThreeState` property to True to indicate that you want it to support three states.

```
checkBox1.ThreeState = true;
```

The radio button and the check box are used for different functions. Use a radio button when you want the user to choose only one option. When you want the user to choose all appropriate options, use a check box. The following C# program shows how to find a checkbox is selected or not.

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsFormsApplication1 {
  public partial class Form1: Form {
    public Form1() {
      InitializeComponent();
    }
    private void button1_Click(object sender, EventArgs e) {
      string msg = "";

      if (checkBox1.Checked == true) {
        msg = "Java => SpringBoot";
      }

      if (checkBox2.Checked == true) {
        msg = msg + "C# => .NET";
      }

      if (checkBox3.Checked == true) {
        msg = msg + "Python => Django";
      }

      if (msg.Length > 0) {
        MessageBox.Show(msg + " selected ");
      } else {
        MessageBox.Show("No checkbox selected");
      }

      checkBox1.ThreeState = true;
    }
  }
}
```

Listing 5.6: Checkbox control

5.8 DateTimePicker

- The `DateTimePicker` control allows you to display and collect date and time from the user with a specified format.
- The `DateTimePicker` control has two parts, a label that displays the selected date and a popup calendar that allows users to select a new date.

- The most important property of the `DateTimePicker` is the `Value` property, which holds the selected date and time.

```
dateTimePicker1.Value = DateTime.Today;
```

- The Value property contains the current date and time the control is set to.
- You can use the Text property or the appropriate member of Value to get the date and time value.

```
DateTime iDate;
```

- The control can display one of several styles, depending on its property values.
- The values can be displayed in four formats, which are set by the Format property:
  - Long,
  - Short,
  - Time, or
  - Custom.

```
dateTimePicker1.Format = DateTimePickerFormat.Short;
```

The following C# program shows how to set and get the value of a DateTimePicker1 control.

```
1  using System;
2  using System.Drawing;
3  using System.Windows.Forms;
4
5  namespace WindowsFormsApplication1 {
6    public partial class Form1: Form {
7      public Form1() {
8        InitializeComponent();
9      }
10     private void Form1_Load(object sender, EventArgs e) {
11       dateTimePicker1.Format = DateTimePickerFormat.Short;
12       dateTimePicker1.Value = DateTime.Today;
13     }
14     private void button1_Click(object sender, EventArgs e) {
15       DateTime iDate;
16       iDate = dateTimePicker1.Value;
17       MessageBox.Show("Selected date is " + iDate);
18     }
```

```
19   }
20 }
```

Listing 5.7: DateTimePicker example

5.9 TabControl

- `TabControl` presents a tabbed layout in the user interface.
- The . NET Framework provides this versatile and easy-to-use control.
- We add the control, change its pages, manipulate it in C# code, and change its visual settings.
- The `TabControl` in the . NET Framework and Windows Forms is a powerful and easy-to-use layout control.
- The Windows Forms `TabControl` displays multiple tabs, like dividers in a notebook or labels in a set of folders in a filing cabinet.
- The tabs can contain pictures and other controls.
- You can use the tab control to produce the kind of multiple-page dialog box that appears many places in the Windows operating system, such as the Control Panel Display Properties.
- Additionally, the `TabControl` can be used to create property pages, which are used to set a group of related properties.
- The most important property of the `TabControl` is `TabPages`, which contains the individual tabs.
- Each individual tab is a `TabPage` object.
- When a tab is clicked, it raises the Click event for that `TabPage` object.

5.10 RichTextBox

- The Windows Forms `RichTextBox` control is used for displaying, entering, and manipulating text with formatting.
- The `RichTextBox` control does everything the `TextBox` control does, but it can also display fonts, colors, and links;

load text and embedded images from a file; and find specified characters.

- The `RichTextBox` control is typically used to provide text manipulation and display features similar to word processing applications such as Microsoft Word.
- Like the `TextBox` control, the `RichTextBox` control can display scroll bars; but unlike the `TextBox` control, its default setting is to display both horizontal and vertical scroll bars as needed, and it has additional scroll bar settings.
- The RichTextBox control has numerous properties to format text.
- To manipulate files, the `LoadFile` and `SaveFile` methods can display and write multiple file formats including plain text, Unicode plain text, and Rich Text Format (RTF).
- The possible file formats are listed in `RichTextBoxStreamType`.
- You can also use a `RichTextBox` control for Web-style links by setting the `DetectUrls` property to true and writing code to handle the `LinkClicked` event.
- You can prevent the user from manipulating some or all of the text in the control by setting the `SelectionProtected` property to true.
- You can undo and redo most edit operations in a `RichTextBox` control by calling the `Undo` and `Redo` methods.
- The `CanRedo` method enables you to determine whether the last operation the user has undone can be reapplied to the control.

5.11 ProgressBar

- A progress bar is a control that an application can use to indicate the progress of a lengthy operation such as calculating a complex result, downloading a large file from the Web etc.
- ProgressBar indicates visually the progress of an operation.
- It is best used on a long-running computation or task. And the `BackgroundWorker` is often used to perform that task—it does not block the interface.

- `ProgressBar` controls are used whenever an operation takes more than a short period of time.
- The `Maximum` and `Minimum` properties define the range of values to represent the progress of a task.
  - `Minimum` : Sets the lower value for the range of valid values for progress.
  - `Maximum` : Sets the upper value for the range of valid values for progress.
  - `Value` : This property obtains or sets the current level of progress.

By default, `Minimum` and `Maximum` are set to 0 and 100 respectively. As the task proceeds, the `ProgressBar` fills in from the left to the right. To delay the program briefly so that you can view changes in the progress bar clearly.

The following C# program shows a simple operation in a `progressbar`.

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsFormsApplication1 {
  public partial class Form1: Form {
    public Form1() {
      InitializeComponent();
    }
    private void button1_Click(object sender, EventArgs e) {
      int i;

      progressBar1.Minimum = 0;
      progressBar1.Maximum = 200;

      for (i = 0; i <= 200; i++) {
        progressBar1.Value = i;
      }

    }

  }
}
```

Listing 5.8: ProgressBar example

5.12 ImageList

- An ImageList component is exactly what the name implies — a list of images.
- Typically, this component is used for holding a collection of images that are used as toolbar icons or icons in a `TreeView` control.
- Many controls have an `ImageList` property.
- The `ImageList` property typically comes with an `ImageIndex` property.
- The `ImageList` property is set to an instance of the `ImageList` component, and the `ImageIndex` property is set to the index in the `ImageList` that represents the image that should be displayed on the control.
- You add images to the `ImageList` component by using the Add method of the `ImageList`.
- The control is not visible directly.

In this example, we have a list of file names, and then add each as an Image object using the `Image.FromFile` method to read the data. The `Form1_Load` event handler is used to make sure the code is run at startup of the application.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            // Add these file names to the ImageList on load.
            string[] files = {
        "image.png",
        "logo.jpg"
      };
            var images = imageList1.Images;
            foreach (string file in files)
            {
```

```
22                  // Use Image.FromFile to load the file.
23                  images.Add(Image.FromFile(file));
24              }
25          }
26      }
27  }
```

Listing 5.9: C# example that uses ImageList

5.13 HelpProvider

The Windows Forms HelpProvider component is used to associate an HTML Help `1.x` Help file (either a `.chm` file, produced with the HTML Help Workshop, or an `.htm` file) with your Windows application. You can provide help in a variety of ways:

- Provide context-sensitive Help for controls on Windows Forms.
- Provide context-sensitive Help on a particular dialog box or specific controls on a dialog box.
- Open a Help file to specific areas, such as the main page of a Table of Contents, the Index, or a search function.

Adding a `HelpProvider` component to your Windows Form allows the other controls on the form to expose the Help properties of the `HelpProvider` component. This enables you to provide help for the controls on your Windows Form. You can associate a Help file with the `HelpProvider` component using the `HelpNamespace` property. You specify the type of Help provided by calling `SetHelpNavigator` and providing a value from the `HelpNavigator` enumeration for the specified control. You provide the keyword or topic for Help by calling the `SetHelpKeyword` method.

Optionally, to associate a specific Help string with another control, use the `SetHelpString` method. The string that you associate with a control using this method is displayed in a pop-up window when the user presses the F1 key while the control has focus.

5.14 Error Provider

- `ErrorProvider` simplifies and streamlines error presentation.
- It is an abstraction that shows errors on form.

- It does not require a lot of work on your part. This is its key feature.
- The `ErrorProvider` control in the Windows Forms framework provides a simple-to-understand abstraction for presenting errors on input.
- Importantly, it reduces the frustration associated with dialog window use.
- The `ErrorProvider` can also streamline error corrections by presenting many errors at once.

In the example, we have a `TextBox` control and a `TextChanged` event handler on that control. When the `TextChanged` event handler is triggered, we check to see if a digit character is present.

And: If one is not, we activate an error through the `ErrorProvider`. Otherwise we clear the `ErrorProvider`.

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApplication8 {
  public partial class Form1: Form {
    public Form1() {
      InitializeComponent();
    }
    private void textBox1_TextChanged(object sender, EventArgs e
    ) {
      // Determine if the TextBox has a digit character.
      string text = textBox1.Text;
      bool hasDigit = false;
      foreach(char letter in text) {
        if (char.IsDigit(letter)) {
          hasDigit = true;
          break;
        }
      }
      // Call SetError or Clear on the ErrorProvider.
      if (!hasDigit) {
        errorProvider1.SetError(textBox1, "Needs to contain a
    digit");
      } else {
        errorProvider1.Clear();
      }
    }
  }
}
```

Listing 5.10: ErrorProvider example

The first argument of `SetError` is the identifier of the `TextBox` control. The second argument is the error message itself. The first argument tells the `ErrorProvider` where to draw the error sign.

5.15 Graphics and GDI

The Graphics Device Interface (GDI) is a Microsoft Windows application programming interface and core operating system component responsible for representing graphical objects and transmitting them to output devices such as monitors and printers.

GDI is responsible for tasks such as drawing lines and curves, rendering fonts and handling palettes. It is not directly responsible for drawing windows, menus, etc.; that task is reserved for the user subsystem, which resides in `user32.dll` and is built atop GDI. Other systems have components that are similar to GDI, for example macOS' Quartz and `X` Window System's `Xlib/XCB`.

GDI's most significant advantages over more direct methods of accessing the hardware are perhaps its scaling capabilities and its abstract representation of target devices. Using GDI, it is very easy to draw on multiple devices, such as a screen and a printer, and expect proper reproduction in each case. This capability is at the center of most *WYSIWYG* applications for Microsoft Windows.

Simple games that do not require fast graphics rendering may use GDI. However, GDI is relatively hard to use for advanced animation, and lacks a notion for synchronizing with individual video frames in the video card, lacks hardware rasterization for 3D, etc. Modern games usually use `DirectX` or `OpenGL` instead, which let programmers exploit the features of modern hardware.

5.16 Timer

- This class regularly invokes code. Every several seconds or minutes, it executes a method.
- This is useful for monitoring the health of a program, as with diagnostics.
- The `System.Timers` namespace proves useful.
- With a `Timer`, we can ensure nothing unexpected has happened.

- We can also run a periodic update (to do anything).
- MSDN states that `System.Timers` "allows you to specify a recurring interval at which the Elapsed event is raised in your application."
- You could create a service that uses a `Timer` to periodically check the server and ensure that the system is up and running.

Example: This example is a static class, meaning it cannot have instance members or fields. It includes the `System.Timers` namespace and shows the Elapsed event function.

Note: It appends the current `DateTime` to a List every three seconds.

```
using System;
using System.Collections.Generic;
using System.Timers;
public static class TimerExample {
  static Timer _timer; // From System.Timers
  static List < DateTime > _l; // Stores timer results
  public static List < DateTime > DateList // Gets the results
  {
    get {
      if (_l == null) // Lazily initialize the timer
      {
        Start(); // Start the timer
      }
      return _l; // Return the list of dates
    }
  }
  static void Start() {
    _l = new List < DateTime > (); // Allocate the list
    _timer = new Timer(3000); // Set up the timer for 3 seconds
    //
    // Type "_timer.Elapsed += " and press tab twice.
    //
    _timer.Elapsed += new ElapsedEventHandler(_timer_Elapsed);
    _timer.Enabled = true; // Enable it
  }
  static void _timer_Elapsed(object sender, ElapsedEventArgs e)
   {
    _l.Add(DateTime.Now); // Add date on each timer event
  }
}
/*
This class has no output. It must be called elsewhere in your
   program.
*/
```

Listing 5.11: Timer example

Timer Control

The Timer Control plays an important role in the development of programs both Client side and Server side development as well as in Windows Services. With the Timer Control we can raise events at a specific interval of time without the interaction of another thread.

We require Timer Object in many situations on our development environment. We have to use Timer Object when we want to set an interval between events, periodic checking, to start a process at a fixed time schedule, to increase or decrease the speed in an animation graphics with time schedule etc.

A Timer control does not have a visual representation and works as a component in the background.

We can control programs with Timer Control in millisecond, seconds, minutes and even in hours. The Timer Control allows us to set Interval property in milliseconds. That is, one second is equal to 1000 milliseconds.

By default, the `Enabled` property of Timer Control is `False`. So, before running the program we have to set the `Enabled` property is `True`, then only the Timer Control starts its function.

Timer example

In the following program we display the current time in a Label Control. In order to develop this program, we need a Timer Control and a Label Control. Here, we set the timer interval as 1000 milliseconds, that means one second, for displaying current system time in Label control for the interval of one second.

```
using System;
```

Start and Stop Timer Control

The Timer control have included the Start and Stop methods for start and stop the Timer control functions. The following C# program shows how to use a timer to write some text to a text file each seconds. The program has two buttons, Start and Stop. The application will write a line to a text file every 1 second once the Start button is clicked. The application stops writing to the text file once the Stop button is clicked.

Timer Tick Event

Timer Tick event occurs when the specified timer interval has elapsed and the timer is enabled.

```
myTimer.Tick += new EventHandler(TimerEventProcessor);
```

Timer Elapsed Event

Timer Elapsed event occurs when the interval elapses. The Elapsed event is raised if the Enabled property is true and the time interval (in milliseconds) defined by the Interval property elapses.

```
MyTimer.Elapsed += OnTimedEvent;
```

Timer Interval Property

Timer Interval property gets or sets the time, in milliseconds, before the Tick event is raised relative to the last occurrence of the Tick event.

Timer Reset Property

Timer `AutoReset` property gets or sets a Boolean indicating whether the Timer should raise the Elapsed event only once (false) or repeatedly (true).

```
MyTimer.AutoReset = false;
```

TimerCallback Delegate

Callback represents the method that handles calls from a Timer. This method does not execute in the thread that created the timer; it executes in a separate thread pool thread that is provided by the system.

Timer Class

System.Timers.Timer fires an event at regular intervals. This is a somewhat more powerful timer. Instead of a Tick event, it has the Elapsed event. The Start and Stop methods of `System.Timers.Timer` which are similar to changing the Enabled property. Unlike the `System.Windows.Forms.Timer`, the events are effectively queued — the timer doesn't wait for one event to have completed before starting to wait again and then firing off the next event. The class is intended for use as a server-based or service component

in a multithreaded environment and it has no user interface and is not visible at runtime.

The following example instantiates a `System.Timers.Timer` object that fires its `Timer.Elapsed` event every two seconds sets up an event handler for the event, and starts the timer.

- Create a timer object for one seconds interval.

```
myTimer = new System.Timers.Timer(1000);
```

- Set elapsed event for the timer. This occurs when the interval elapses.

```
myTimer.Elapsed += OnTimedEvent;
```

- Finally, start the timer.

```
myTimer.Enabled = true;
```

5.17 SDI and MDI Applications

5.17.1 SDI

SDI stands for Single Document Interface. It is an interface design for handling documents within a single application. SDI exists independently of others and thus is a standalone window. SDI supports one interface means you can handle only one application at a time. For grouping SDI uses special window managers.

5.17.2 MDI

MDI stands for Multiple Document Interface. It is an interface design for handling documents within a single application. When application consists of an MDI parent form containing all other window consisted by app, then MDI interface can be used. Switch focus to specific document can be easily handled in MDI. For maximizing all documents, parent window is maximized by MDI.

MDI Form

A Multiple Document Interface (MDI) programs can display multiple child windows inside them. This is in contrast to single document interface (SDI)

applications, which can manipulate only one document at a time. Visual Studio Environment is an example of Multiple Document Interface (MDI) and notepad is an example of an SDI application. MDI applications often have a Window menu item with submenus for switching between windows or documents.

Any windows can become an MDI parent, if you set the `IsMdiContainer` property to True.

```
IsMdiContainer = true;
```

The following C# program shows a MDI form with two child forms. Create a new C# project, then you will get a default form Form1. Then add two more forms in the project (Form2 , Form 3).

NOTE: If you want the MDI parent to auto-size the child form you can code like this.

```
form.MdiParent = this;
```

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsFormsApplication1 {
  public partial class Form1: Form {
    public Form1() {
      InitializeComponent();
    }
    private void Form1_Load(object sender, EventArgs e) {
      IsMdiContainer = true;
    }
    private void menu1ToolStripMenuItem_Click(object sender,
    EventArgs e) {
      Form2 frm2 = new Form2();
      frm2.Show();
      frm2.MdiParent = this;
    }
    private void menu2ToolStripMenuItem_Click(object sender,
    EventArgs e) {
      Form3 frm3 = new Form3();
      frm3.Show();
      frm3.MdiParent = this;
    }
  }
}
```

Listing 5.12: MDI example

Main Difference

MDI and SDI are interface designs for handling documents within a single application. MDI stands for "Multiple Document Interface" while SDI stands for "Single Document Interface". Both are different from each other in many aspects. One document per window is enforced in SDI while child windows per document are allowed in MDI. SDI contains one window only at a time but MDI contain multiple document at a time appeared as child window. MDI is a container control while SDI is not container control. MDI supports many interfaces means we can handle many applications at a time according to user's requirement. But SDI supports one interface means you can handle only one application at a time.

Key Differences

- MDI stands for "Multiple Document Interface" while SDI stands for "Single Document Interface".
- One document per window is enforced in SDI while child windows per document are allowed in MDI.
- MDI is a container control while SDI is not container control.
- SDI contains one window only at a time but MDI contain multiple document at a time appeared as child window.
- MDI supports many interfaces means we can handle many applications at a time according to user's requirement. But SDI supports one interface means you can handle only one application at a time.
- For switching between documents MDI uses special interface inside the parent window while SDI uses Task Manager for that.
- In MDI grouping is implemented naturally but in SDI grouping is possible through special window managers.
- For maximizing all documents, parent window is maximized by MDI but in case of SDI it is implemented through special code or window manager.
- Switch focus to specific document can be easily handled while in MDI but it is difficult to implement in SDI.

To create MDI child forms

In the Properties Windows for the form, set its `IsMdiContainer` property to true, and its `WindowsState` property to `Maximized`. This designates the form as an MDI container for child windows.

5.18 DialogBox (Modal and Modeless)

- A dialog box is a type of window, which is used to enable common communication or dialog between a computer and its user.
- A dialog box is most often used to provide the user with the means for specifying how to implement a command or to respond to a question.
- `Windows.Form` is a base class.
- Dialog boxes are used to interact with the user and retrieve information.

In simple terms, a dialog box is a form with its `FormBorderStyle` enumeration property set to `FixedDialog`. You can construct your own custom dialog boxes, add controls such as `Label`, `Textbox`, and `Button` to customize dialog boxes specific needs.

The . NET Framework also includes predefined dialog boxes, such as File Open and message boxes. Dialog boxes are special forms that are non-re sizable. They are also used to display messages to the user. The messages can be error messages, confirmation of the password, confirmation for the deletion of a particular record, Find-Replace utility etc. There are standard dialog boxes to open and save a file, select a folder, print the documents, set the font or color for the text, etc.

`MessageBox` class is used to display messages to the user. The `show()` method is used to display a message box with the specified text, caption, buttons and icon.

For example:

```
DialogResult res = MessageBox.Show("Are you sure you want to
 Delete "," Confirmation ", MessageBoxButtons.OKCancel,
 MessageBoxIcon.Information);

if (res == DialogResult.OK) {
  MessageBox.Show("You have clicked Ok Button");
  //Some task...
}
if (res == DialogResult.Cancel) {
  MessageBox.Show("You have clicked Cancel Button");
```

```
    //Some task...
  }
```

Dialog boxes are of two types:

1. Modal dialog box
2. Modeless dialog box

5.18.1 Modal dialog box

A dialog box that temporarily halts the application and the user cannot continue until the dialog has been closed is called *modal dialog box*. The application may require some additional information before it can continue or may simply wish to confirm that the user wants to proceed with a potentially dangerous course of action. The application continues to execute only after the dialog box is closed; until then the application halts. For example, when saving a file, the user gives a name of an existing file; a warning is shown that a file with the same name exists, whether it should be overwritten or be saved with different name. The file will not be saved unless the user selects "OK" or "Cancel".

5.18.2 Modeless dialog box

Another type of dialog box, which is used is a modeless dialog box. It is used when the requested information is not essential to continue, so the Window can be left open, while work continues somewhere else. For example, when working in a text editor, the user wants to find and replace a particular word. This can be done, using a dialog box, which asks for the word to be found and replaced. The user can continue to work, even if this box is open.
Note: *Model dialog is displayed, using* `ShowDialog()` *method. Modeless dialog boxes are displayed, using* `Show()` *method.*

5.19 Form Inheritance

Form inheritance, a feature of .NET that lets you create a base form that becomes the basis for creating more advanced forms. The new "derived" forms automatically inherit all the functionality contained in the base form. This design paradigm makes it easy to group common functionality and, in the process, reduce maintenance costs. When the base form is modified, the "derived" classes automatically follow suit and adopt the changes. The same concept applies to any type of object.

In order to inherit from a form, the file or namespace containing that form must have been built into an executable file or `DLL`. To build the project, choose Build from the Build menu. Also, a reference to the namespace must be added to the class inheriting the form.

To inherit a form programmatically

- In your class, add a reference to the namespace containing the form you wish to inherit from.
- In the class definition, add a reference to the form to inherit from. The reference should include the namespace that contains the form, followed by a period, then the name of the base form itself.

```
// Syntax:
public class CourseBCA : Faculty.ScienceAndTechnology
```

When inheriting forms, keep in mind that issues may arise with regard to event handlers being called twice, because each event is being handled by both the base class and the inherited class.

5.20 Developing Custom, Composite Controls

Composite Controls

A composite control is a collection of Windows Forms controls encapsulated in a common container. This kind of control is sometimes called a user control. The contained controls are called constituent controls.

A composite control holds all the inherent functionality associated with each of the contained Windows Forms controls and enables you to selectively expose and bind their properties. A composite control also provides a great deal of default keyboard handling functionality with no extra development effort on your part.

For example, a composite control could be built to display customer address data from a database. This control would include a `DataGridView` control to display the database fields, a `BindingSource` to handle binding to a data source, and a `BindingNavigator` control to move through the records. You could selectively expose data binding properties, and you could package and reuse the entire control from application to application.

Custom Controls

Another way to create a control is to create one substantially from the beginning by inheriting from Control. The Control class provides all the basic functionality required by controls, including mouse and keyboard handling events, but no control-specific functionality or graphical interface.

Creating a control by inheriting from the Control class requires much more thought and effort than inheriting from `UserControl` or an existing Windows Forms control. Because a great deal of implementation is left for you, your control can have greater flexibility than a composite or extended control, and you can tailor your control to suit your exact needs.

5.21 Field Validator Control

When users enter data into your application, you may want to verify that the data is valid before your application uses it. You may require that certain text fields not be zero-length, that a field formatted as a telephone number, or that a string doesn't contain invalid characters. Windows Forms provides several ways for you to validate input in your application.

MaskedTextBox Control

If you need to require users to enter data in a well-defined format, such as a telephone number or a part number, you can accomplish this quickly and with minimal code by using the `MaskedTextBox` control. A *mask* is a string made up of characters from a masking language that specifies which characters can be entered at any given position in the text box. The control displays a set of prompts to the user. If the user types an incorrect entry, for example, the user types a letter when a digit is required, the control will automatically reject the input.

Event-driven validation

Allows control over validation, complex validation checks. Each control that accepts free-form user input has a `Validating` event that will occur whenever the control requires data validation.

Event-driven validation data-bound controls

Validation is useful when you have bound your controls to a data source, such as a database table.

5.22 Delegates in C#

C# delegates are similar to pointers to functions, in C or C++. A delegate is a reference type variable that holds the reference to a method. The reference can be changed at runtime.

Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the `System.Delegate` class.

Declaring Delegates

Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which has the same signature as that of the delegate. For example, consider a delegate:

```
1  public delegate int MyDelegate (string s);
```

The preceding delegate can be used to reference any method that has a single string parameter and returns an int type variable. Syntax for delegate declaration is:

```
1  delegate <return type> <delegate name> <parameter list>
```

Instantiating Delegates

Once a delegate type is declared, a delegate object must be created with the new keyword and be associated with a particular method. When creating a delegate, the argument passed to the new expression is written similar to a method call, but without the arguments to the method. For example

```
1  public delegate void printString(string s);
2  ...
3  printString ps1 = new printString(WriteToScreen);
4  printString ps2 = new printString(WriteToFile);
```

Following example demonstrates declaration, instantiation, and use of a delegate that can be used to reference methods that take an integer parameter and returns an integer value.

```
1 using System;
2 delegate int NumberChanger(int n);
3 namespace DelegateAppl {
4   class TestDelegate {
5     static int num = 10;
6     public static int AddNum(int p) {
```

```
7         num += p;
8         return num;
9       }
10      public static int MultNum(int q) {
11        num *= q;
12        return num;
13      }
14      public static int getNum() {
15        return num;
16      }
17      static void Main(string[] args) {
18        //create delegate instances
19        NumberChanger nc1 = new NumberChanger(AddNum);
20        NumberChanger nc2 = new NumberChanger(MultNum);
21        //calling the methods using the delegate objects
22        nc1(25);
23        Console.WriteLine("Value of Num: {0}", getNum());
24        nc2(5);
25        Console.WriteLine("Value of Num: {0}", getNum());
26        Console.ReadKey();
27      }
28    }
29  }
30  /* output:
31  Value of Num: 35
32  Value of Num: 175
33  */
```

Listing 5.13: Delegate Example

5.23 Events – Types and Handling

Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications. Applications need to respond to events when they occur. For example, interrupts. Events are used for inter-process communication. Events can be marked as public, private, protected, internal, protected internal or private protected. These access modifiers define how users of the class can access the event.

Using Delegates with Events (Handling Events)

The events are declared and raised in a class and associated with the event handlers using delegates within the same class or some other class. The class containing the event is used to publish the event. This is called the publisher

class. Some other class that accepts this event is called the subscriber class. Events use the publisher-subscriber model.

A publisher is an object that contains the definition of the event and the delegate. The event- delegate association is also defined in this object. A publisher class object invokes the event and it is notified to other objects.

A subscriber is an object that accepts the event and provides an event handler. The delegate in the publisher class invokes the method (event handler) of the subscriber class.

Declaring Events

To declare an event inside a class, first a delegate type for the event must be declared. For example,

```
public delegate string MyDel(string str);
```

Next, the event itself is declared, using the event keyword -

```
event MyDel MyEvent;
```

Example

```
using System;
namespace SampleApp {
  public delegate string MyDel(string str);
  class EventProgram {
    event MyDel MyEvent;
  }
  public EventProgram() {
    this.MyEvent += new MyDel(this.WelcomeUser);
  }
  public string WelcomeUser(string username) {
    return "Welcome " + username;
  }
  static void Main(string[] args) {
    EventProgram obj1 = new EventProgram();
    string result = obj1.MyEvent("Mr User");
    Console.WriteLine(result);
  }
}

/* output:
    Welcome Mr User
*/
```

Listing 5.14: Events Example

5.24 Exception Handling

An exception is a problem that arises during the execution of a program. A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: try, catch, finally, and throw.

- **try** - A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
- **catch** - A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- **finally** - The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **throw** - A program throws an exception when a problem shows up. This is done using a throw keyword.

Assuming a block raises an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

C#

```
// Syntax
try {
  // statements causing exception
} catch (ExceptionName e1) {
  // error handling code
} catch (ExceptionName e2) {
  // error handling code
} catch (ExceptionName eN) {
  // error handling code
} finally {
  // statements to be executed
}
```

VB

```
Try
[ tryStatements ]
[ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
[ catchStatements ]
[ Exit Try ] ]
[ Catch ... ]
[ Finally
[ finallyStatements ] ]
End Try
```

You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

Handling Exceptions (Example)

C# provides a structured solution to the exception handling in the form of try and catch blocks. Using these blocks the core program statements are separated from the error-handling statements.

These error handling blocks are implemented using the try, catch, and finally keywords. Following is an example of throwing an exception when dividing by zero condition occurs

```
using System;
namespace ErrorHandlingApplication {
  class DivNumbers {
    int result;
    DivNumbers() {
      result = 0;
    }
    public void division(int num1, int num2) {
      try {
        result = num1 / num2;
      } catch (DivideByZeroException e) {
        Console.WriteLine("Exception caught: {0}", e);
      } finally {
        Console.WriteLine("Result: {0}", result);
      }
    }
    static void Main(string[] args) {
      DivNumbers d = new DivNumbers();
      d.division(25, 0);
      Console.ReadKey();
    }
```

```
  }
}
/* output:
  Exception caught: System.DivideByZeroException: Attempted to
    divide by zero
  at ...
  Result: 0
*/
```

Listing 5.15: Divide by zero exception using C#

VB

```
Module ExceptionExample
   Sub division(ByVal num1 As Integer, ByVal num2 As Integer)
      Dim result As Integer
      Try
         result = num1 \ num2
      Catch e As DivideByZeroException
         Console.WriteLine("Exception caught: {0}", e)
      Finally
         Console.WriteLine("Result: {0}", result)
      End Try
   End Sub
   Sub Main()
      division(50, 0)
      Console.ReadKey()
  End Sub
End Module
```

Listing 5.16: Divide by zero exception using VB

Creating User-Defined Exceptions (Example)

You can also define your own exception. User-defined exception classes are derived from the Exception class. The following example demonstrates this

```
using System;
namespace UserDefinedException {
  class TestTemperature {
    static void Main(string[] args) {
      Temperature temp = new Temperature();
      try {
        temp.showTemp();
      } catch (TempIsZeroException e) {
        Console.WriteLine("TempIsZeroException: {0}", e.Message)
    ;
```

```
      }
      Console.ReadKey();
    }
  }
}
public class TempIsZeroException: Exception {
  public TempIsZeroException(string message): base(message) {}
}
public class Temperature {
  int temperature = 0;
  public void showTemp() {
    if (temperature == 0) {
      throw (new TempIsZeroException("Zero Temperature found"));
    } else {
      Console.WriteLine("Temperature: {0}", temperature);
    }
  }
}

/* output:
    TempIsZeroException: Zero Temperature found
 */
```

Listing 5.17: User defined exception example using C#

VB

```
Module CustomExceptionVB
   Public Class TempIsZeroException : Inherits
   ApplicationException
      Public Sub New(ByVal message As String)
         MyBase.New(message)
      End Sub
   End Class
   Public Class Temperature
      Dim temperature As Integer = 0
      Sub showTemp()
         If (temperature = 0) Then
            Throw (New TempIsZeroException("Zero Temperature
   found"))
         Else
            Console.WriteLine("Temperature: {0}", temperature)
         End If
      End Sub
   End Class
   Sub Main()
      Dim temp As Temperature = New Temperature()
      Try
```

```
20          temp.showTemp()
21       Catch e As TempIsZeroException
22          Console.WriteLine("TempIsZeroException: {0}", e.Message
    )
23       End Try
24       Console.ReadKey()
25    End Sub
26 End Module
```

Listing 5.18: User defined exception example using VB

In the . Net Framework, exceptions are represented by classes. The exception classes in . Net Framework are mainly directly or indirectly derived from the System.Exception class. Some exception classes derived from the `System.Exception` class are the `System.ApplicationException` and `System.SystemException` classes.

The `System.ApplicationException` class supports exceptions generated by application programs. So, the exceptions defined by the programmers should derive from this class.

The `System.SystemException` class is the base class for all predefined system exception.

CHAPTER

6

DATA ACCESS WITH ADO (ACTIVEX DATA OBJECT) . NET

ADO . NET provides consistent access to data sources such as SQL Server and XML, and to data sources exposed through OLEDB and ODBC. Data-sharing consumer applications can use ADO . NET to connect to these data sources and retrieve, handle, and update the data that they contain.

6.1 Comparison Between ADO and ADO . NET

Table 6.1: Comparison between ADO and ADO . NET

ADO	**ADO . NET**
ADO is based on Component Object Modeling (COM)	ADO . Net is based on Common Language Runtime (CLR)

Continued on next page

Table 6.1 – *Continued from previous page*

ADO	ADO . NET
Stores data in binary format	Stores data in `XML` format i. e. parsing of data
Can't be integrated with `XML` because ADO have limited access of `XMLs`	Can be integrated with `XML` as having robust support of `XML`
In ADO, data is provided by `RecordSet`	In ADO . Net data is provided by `DataSet` or `DataAdapter`
ADO is connection oriented means it requires continuous active connection	ADO . NET is disconnected, does not need continuous connection
ADO gives rows as single table view, it scans sequentially the rows using `MoveNext` method	ADO . NET gives rows as collections so you can access any record and also can go through a table via loop
In ADO, You can create only Client side cursor	In ADO . NET, You can create both Client & Server side cursor
Using a single connection instance, ADO cannot handle multiple transactions	Using a single connection instance, ADO . NET can handle multiple transactions

***Note**: A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the **active set**.*

6.2 ADO . NET Concept and Overview

ADO . NET provides a bridge between the front end controls and the back end database. The ADO . NET objects encapsulate all the data access operations and the controls interact with these objects to display data, thus hiding the details of movement of data.

ADO . NET includes . NET Framework data providers for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, placed in an ADO . NET `DataSet`.

The ADO . NET classes are found in `System.Data.dll`, and are integrated with the `XML` classes found in `System.Xml.dll`.

Benefits of ADO . NET

- **Interoperability**

  Uses `XML` to transfer data across a network. This is not necessary that the destination component should be a . NET application. Any receiving component that is able to read XML can read the data specified in the `XML` format.

- **Maintainability**

  Since ADO . NET is different layer and independent of database, it is loosely coupled with business logic, it enables easy transform of the architectural changes in a deployed application.

- **Performance**

  Provides fast execution of disconnected applications, over the disconnected `RecordSets` in classic ADO.

- **Scalability**

  Provides the conservation of resources, such as database locks and database connections. The applications that are based on ADO.NET support disconnected across to data. So, that the databases are not locked by user queries for long duration.

The Figure 6.1 shows the ADO . NET objects at a glance:

6.2.1 ADO . NET Architecture

The ADO.NET architecture has two main parts:

1. `DataProvider` (Connected Objects or Connection oriented objects)
2. `DataSet` (Disconnected objects or Connection-less objects)

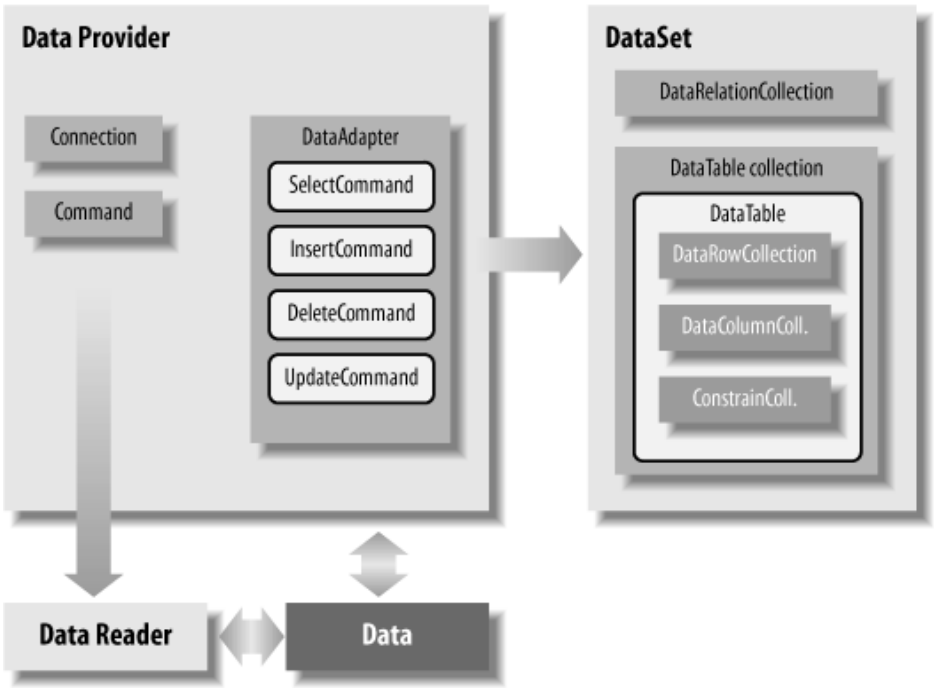

Figure 6.1: ADO . NET Architecture.

6.2.1.1 DataProvider

- The . NET framework Data Provider is component that has been explicitly designed for data manipulation and fast, forward-only, read-only access to data.
- . NET Framework data provider is used for connecting to a database, executing commands, and retrieving results.
- The Data Provider has four core objects:
  1. **Connection**: Establishes a connection to a specific data source
  2. **Command**: Executes a command against a data source
  3. **DataReader**: Reads a stream of data from a data source
  4. **DataAdapter**: Populates a data set and resolves updates with the data source

An ADO . NET data provider is a software component that interacts with a data source. ADO . NET data providers are analogous to ODBC drivers, JDBC drivers, and OLE DB providers.

***Note**: ODBC (Open Database Connectivity) is designed to provide access primarily to SQL data in a multi- platform environment. OLE DB (Object Linking and Embedding Database) is designed to provide access to all types of data in an OLE Component Object Model (COM) environment. OLE DB includes the SQL functionality defined in ODBC but also defines interfaces suitable for gaining access to data other than SQL data.*

Connection

- The Connection object is the first component of ADO . NET.
- The Connection objects provider connectivity to a data source.
- It establishes a connection to a specific data source.
- Connection object helps in accessing and manipulating a database.
- The base class for all Connection objects in the `DbConnection` class.
- Example: See Section 6.3.1.

Command

- The Command object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.
- It executes a command against a data source. Exposes Parameters and can execute in the scope of a Transaction from a Connection.
- You can execute SQL queries to return data in a `DataSet` or a `DataReader` object.
- Command object performs the standard Select, Insert, Delete and Update T-SQL operations.
- The base class for all Command objects is the `DbCommand` class.
- Example: See Section 6.3.2.

DataReader

- The Data Reader provides a high-performance stream of data from the data source.
- It reads a forward-only, read-only stream of data from a data source.
- `DataReader` object works in connected model.
- The base class for all `DataReader` objects is the `DbDataReader` class.
- Example: See Section 6.3.3.

DataAdapter

- The Data Adapter provides the bridge between the Data Set object and the data source.
- The Data Adapter uses command object to execute SQL commands at the data source to both load the Data Set with data and reconcile changes that were made to the data in the dataset back to the data source.
- It populates a `DataSet` and resolves updates with the data source.
- The base class for all `DataAdapter` objects is the `DbDataAdapter` class.

The following lists the data providers that are included in the . NET framework.

1. SQL Server
    - Provides data access for Microsoft SQL server.
    - Uses the `System.Data.SqlClient` namespace.
2. OLEDB
    - For data sources exposed by using OLEDB.
    - Uses the `System.Data.OleDb` namespace
3. ODBC
    - For data sources exposed by using ODBC.
    - Uses the `System.Data.Odbc` namespace.

4. Oracle
    - For Oracle data sources.
    - Uses the `System.Data.OracleClient` namespace.

Following Code demonstration of binding `GridView` control using Connection Oriented Objects of ADO . Net.

```
//Step 1: Prepare Connection
SqlConnection objConnection = new SqlConnection();
objConnection.ConnectionString = @ "Data Source=ComputerName\
    SQLInstance;InitialCatalog = DatabaseName;Integrated Security
     = False;User ID = username;Password = 123;";
objConnection.Open();
//Step 2: Prepare & Execute Command
  SqlCommand objCommand = new SqlCommand();
  objCommand.Connection = objConnection;
  objCommand.CommandType = CommandType.Text;
  objCommand.CommandText = "SELECT CountryID, CountryName FROM
    Country ORDER BY CountryName";
//Step 3: Collect Data to DataReader object which has been
    received as a result of Command
    SqlDataReader objSDR = objCommand.ExecuteReader();
    gvCountry.DataSource = objSDR;
    gvCountry.DataBind();
objConnection.Close();
```

Listing 6.1: Example of GridView control using connection oriented objects

6.2.1.2 DataSet

- The dataset represents a subset of the database.
- The dataset is a memory-resident representation of data that provides consistent relational programming
- It does not have a continuous connection to the database.
- To update the database a re-connection is required.
- The dataset represents a complete set of data, including related tables, constraints, and relationship among the table.
- The `DataSet` contains:
    1. `DataTable` objects and
    2. `DataRelation` objects.

DataRelation

- The `DataRelation` objects represent the relationship between two tables.
- A relationship represented by the Data relation object, associated rows in one Data table with rows in another Data table.
- A relationship is analogous to a join path that might exist between primary and foreign key columns in a relational database.
- A data relation identifies matching columns in two tables of a dataset.
- The essential element of a data relation are:
  - The name of the relationship
  - The name of the tables being related
  - The related column in each table
- Relationship can be built with more than one column per table by specifying an array of Data Column objects as the key columns.
- Optionally, a Unique key constraint can be added to enforce integrity constraints when changes are made to related column values.

Listing 6.2 example demonstrate disconnected objects.

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace DataTable_Datarow_DataColumn_Example {
  class Program {
    static void Main(string[] args) {
      //Step 1: Prepare Connection
      SqlConnection objConnection = new SqlConnection();
      objConnection.ConnectionString = @"Data Source=
   ComputerName\SQLInstance;Initial
Catalog=DatabaseName;Integrated Security=False; User ID=username
   ;
Password=123;";
      objConnection.Open();
      //Step 2: Prepare & Execute Command
      SqlCommand objCommand = new SqlCommand();
      objCommand.Connection = objConnection;
      objCommand.CommandType = CommandType.Text;
      objCommand.CommandText = "SELECT CountryID, CountryName
   FROM Country ORDER BY
```

```
18 CountryName";
19       SqlDataAdapter sda = new SqlDataAdapter(objCommand);
20       DataTable dt = new DataTable();
21       sda.Fill(dt);
22       objConnection.Close();
23     }
24   }
25 }
```

Listing 6.2: Example of disconnected objects

DataTable

- The `DataTable` class represents the tables in the database.
- A Data table is defined in the `System.Data` namespace and represents a single table of memory-resident data.
- It has the important properties as shown in Table 6.2; most of these properties are read only properties except the `PrimaryKey` property.

Table 6.2: DataTable properites

Properties	Description
`ChildRelations`	Returns the collection of child relationship
`Columns`	Returns the Columns collection
`Constraints`	Returns the Constraints collection
`DataSet`	Returns the parent `DataSet`
`DefaultView`	Returns a view of the table
`ParentRelations`	Returns the `ParentRelations` collection
`PrimaryKey`	Gets or sets an array of columns as the primary key for the table
`Rows`	Returns the Rows collection

DataRow The `DataRow` object represents a row in a table. It has the important properties as shown in Table 6.3:

Table 6.3: DataRow properites

Properties	Description
`HasErrors`	Indicates if there are any errors relationship
`Items`	Gets or sets the data stored in a specific column collection
`ItemArrays`	Gets or sets all the values for the row
`Table`	Returns the parent table

6.3 Working with Connection, Command, DataReader

An application needs to perform following steps to retrieve data from database:

1. *Connect to the Database*
2. *Prepare an SQL Command*
3. *Execute the Command*
4. *Retrieve the results and display in the application*

- The combination of `Connection`, `Command` and `DataReader` object gives us connected approach of connecting to database.
- Notice that we are using `SQLConnection`, `SQLCommand` and `SQLDataReader` classes. All the objects are prefixed with the word SQL. All these classes are present in `System.Data.SqlClient` namespace. So, we can say that the .NET data provider for SQL Server is `System.Data.SqlClient`.
- Sample ADO . NET code to connect to SQL Server Database and retrieve data using connection, command and `DataReader`:

```
1 string ConnectionString = ConfigurationManager.
    ConnectionStrings["DBConnectionString"].ConnectionString
    ;
2 using (SqlConnection connection = new SqlConnection(
    ConnectionString)){
```

```
connection.Open();
SqlCommand command = new SqlCommand("Select * from
    tblProductInventory", connection);
using (SqlDataReader reader = command.ExecuteReader()){
    ProductsGridView.DataSource = reader;
    ProductsGridView.DataBind();
    }
}
```

Listing 6.3: Example to connect sql server database and retrieve data using connection, command and reader

- Sample ADO . NET code to connect to Oracle Database and retrieve data using connection, command and `DataReader`:

```
  OracleConnection con = new OracleConnection("Oracle db
    Connection String");
  OracleCommand cmd = new OracleCommand("Select * from
    tblProduct", con);
  con.Open();
  OracleDataReader rdr = cmd.ExecuteReader();
  GridView1.DataSource = rdr;
  GridView1.DataBind();
  con.Close();
```

  Notice that we are using `OracleConnection`, `OracleCommand` and `OracleDataReader` classes. All the objects are prefixed with the word Oracle. All these classes are present in `System.Data.OracleClient` namespace. So, we can say that the . NET data provider for Oracle is `System.Data.OracleClient`.

- Another example of using `DataReader` to load data:

```
string ConnectionString = ConfigurationManager.
    ConnectionStrings["DBConnectionString"].ConnectionString
    ;
using(SqlConnection connection = new SqlConnection(
    ConnectionString)) {
  connection.Open();
  SqlCommand command = new SqlCommand("Select * from
    tblProductInventory", connection);
  using(SqlDataReader reader = command.ExecuteReader()) {
    // Create the DataTable and columns. This will
    // be used as the datasource for the GridView
    DataTable sourceTable = new DataTable();
    sourceTable.Columns.Add("ID");
    sourceTable.Columns.Add("Name");
    sourceTable.Columns.Add("Price");
```

```
12      sourceTable.Columns.Add("DiscountedPrice");
13      while (reader.Read()) {
14        //Calculate the 10% discounted price
15        int OriginalPrice = Convert.ToInt32(reader["UnitPrice
      "]);
16        double DiscountedPrice = OriginalPrice * 0.9;
17        // Populate datatable column values from the
      SqlDataReader
18        DataRow datarow = sourceTable.NewRow();
19        datarow["ID"] = reader["ProductId"];
20        datarow["Name"] = reader["ProductName"];
21        datarow["Price"] = OriginalPrice;
22        datarow["DiscountedPrice"] = DiscountedPrice;
23        //Add the DataRow to the DataTable
24        sourceTable.Rows.Add(datarow);
25      }
26      // Set sourceTable as the DataSource for the GridView
27      ProductsGridView.DataSource = sourceTable;
28      ProductsGridView.DataBind();
29    }
30  }
```

Listing 6.4: DataReader example

If for some reason, you want to loop through each row in the `SqlDataReader` object, then use the `Read()` method, which returns true as long as there are rows to read. If there are no more rows to read, then this method will return false. In the following example, we loop through each row in the `SqlDataReader` and then compute the 10% discounted price.

If we want to connect to OLEDB datasources like Excel, Access etc, we can use OleDbConnection, OleDbCommand and OleDbDataReader classes. So, ṄET data provider for OLEDB is ***`System.Data.OleDb`****.*

Different .NET Data Providers

- Data Provider for SQL Server - `System.Data.SqlClient`
- Data Provider for Oracle - `System.Data.OracleClient`
- Data Provider for OLEDB - `System.Data.OleDb`
- Data Provider for ODBC - `System.Data.Odbc`

Depending on the provider, the following ADO . NET objects have a different prefix:

1. *Connection*:
   - `SQLConnection`,
   - `OracleConnection`,
   - `OleDbConnection`,
   - `OdbcConnection`, etc.
2. *Command*:
   - `SQLCommand`,
   - `OracleCommand`,
   - `OleDbCommand`,
   - `OdbcCommand`, etc.
3. *DataReader*:
   - `SQLDataReader`,
   - `OracleDataReader`,
   - `OleDbDataReader`,
   - `OdbcDataReader`, etc.
4. *DataAdapter*:
   - `SQLDataAdapter`,
   - `OracleDataAdapter`,
   - `OleDbDataAdapter`,
   - `OdbcDataAdapter`, etc.

6.3.1 Connection

- It is used to establish an open connection to the SQL Server database.
- It is a sealed class. So, it cannot be inherited.
- `SqlConnection` class uses `SqlDataAdapter` and `SqlCommand` classes together to increase performance when connecting to a Microsoft SQL Server database.
- Connection does not close explicitly even it goes out of scope. Therefore, you must explicitly close the connection by calling `Close()` method.
- `Using` block is used to close the connection automatically.
- We don't need to call `Close()` method explicitly, `Using` block do this for ours program implicitly when the code exits the block.

Example

```
using (SqlConnection con = new SqlConnection(connectionString))
    {
  con.Open();
}
```

```
using System;
using System.Data.SqlClient;
namespace SqlConnectionExample {
  class Program {
    static void Main(string[] args) {
      new Program().Connecting();
    }
    public void Connecting() {
      using(SqlConnection con = new SqlConnection("data source
   =.; database=bca; integrated security=SSPI")) {
        con.Open();
        Console.WriteLine("Connection to database successful.");
      }
    }
  }
}
```

Listing 6.5: Example of sqlConnection

6.3.2 Command

- This class is used to store and execute SQL statement for SQL Server database.
- It is a sealed class. So, it cannot be inherited.

Example

```
using System;
using System.Data.SqlClient;
namespace SqlCommandExample {
  class Program {
    static void Main(string[] args) {
      new Program().CreateTable();
    }
    public void CreateTable() {
      SqlConnection con = null;
      try {
        // Creating Connection
        con = new SqlConnection("data source=.; database=bca;
   integrated security=SSPI");
        // writing sql query
        SqlCommand cm = new SqlCommand("select * from student",
   con);
        // Opening Connection
        con.Open();
```

```
17          // Executing the SQL query
18          SqlDataReader sdr = cm.ExecuteReader();
19          while (sdr.Read()) {
20            Console.WriteLine(sdr["name"] + " " + sdr["address"]);
21          }
22        } catch (Exception e) {
23          Console.WriteLine("Exception!" + e);
24        }
25        // Closing the connection
26        finally {
27          con.Close();
28        }
29      }
30    }
31  }
```

Listing 6.6: Example of sqlCommand

6.3.3 DataReader

- This class is used to read data from SQL Server database.
- It reads data in forward-only stream of rows from a SQL Server database.
- It is sealed class so that cannot be inherited.
- It inherits `DbDataReader` class and implements `IDisposable` interface.

```
1  using System;
2  using System.Data.SqlClient;
3  namespace SqlDataReaderExample {
4    class Program {
5      static void Main(string[] args) {
6        new Program().GetData();
7      }
8      public void GetData() {
9        SqlConnection con = null;
10       try {
11         // Creating Connection
12         con = new SqlConnection("data source=.; database=bca;
       integrated security=SSPI");
13         // writing sql query
14         SqlCommand cm = new SqlCommand("select * from stu", con)
       ;
15         // Opening Connection
16         con.Open();
```

```
17          // Executing the SQL query
18          SqlDataReader sdr = cm.ExecuteReader();
19          while (sdr.Read()) {
20            Console.WriteLine(sdr["name"] + " " + sdr["address"]);
21          }
22        } catch (Exception e) {
23          Console.WriteLine("Exception!" + e);
24        }
25        // Closing the connection
26        finally {
27          con.Close();
28        }
29      }
30    }
31  }
```

Listing 6.7: Example of sqlDataReaderExample.

6.4 Working With Dataset

- The ADO . NET `DataSet` is a memory-resident representation of data that provides a consistent relational programming model regardless of the source of the data it contains.
- A `DataSet` represents a complete set of data including the tables that contain, order, and constrain the data, as well as the relationships between the tables.
- There are several ways of working with a DataSet, which can be applied independently or in combination. We can:
  - Programmatically create a `DataTable`, `DataRelation`, and Constraint within a `DataSet` and populate the tables with data.
  - Populate the `DataSet` with tables of data from an existing relational data source using a `DataAdapter`.
  - Load and persist the `DataSet` contents using `XML`.

Creating DataSet Programatically

- In this program, we create two `DataTables`.
- One stores two rows of patient information.
- And the second stores two rows of medication information.

- We create a `DataSet` with the `DataSet` constructor.
- Then we add the two `DataTables` to the `DataSet` instance.
- Finally we print the `XML`.

```
using System;
using System.Data;
class Program {
  static void Main() {
    // Create two DataTable instances.
    DataTable table1 = new DataTable("patients");
      table1.Columns.Add("name");
      table1.Columns.Add("id");
      table1.Rows.Add("ram", 1);
      table1.Rows.Add("shyam", 2);
    DataTable table2 = new DataTable("medications");
      table2.Columns.Add("id");
      table2.Columns.Add("medication");
      table2.Rows.Add(1, "massage");
      table2.Rows.Add(2, "therapy");
    // Create a DataSet and put both tables in it.
    DataSet set = new DataSet("office");
      set.Tables.Add(table1);
      set.Tables.Add(table2);
    // Visualize DataSet.
    Console.WriteLine(set.GetXml());
  }
}
```

Listing 6.8: Creating DataSet

```
<office>
    <patients>
        <name>ram</name>
        <id>1</id>
    </patients>
    <patients>
        <name>shyam</name>
        <id>2</id>
    </patients>
    <medications>
        <id>1</id>
        <medication>massage</medication>
    </medications>
    <medications>
        <id>2</id>
        <medication>therapy</medication>
    </medications>
```

```
</office>
```

Listing 6.9: Output from Listing 6.8

Populating DataSet with tables from database

- Following program send query to stored procedure in database and loads the table data into dataset.
- The stored procedure for this would be:

```
Create procedure spGetProductAndCategoriesData
as
  Begin
  Select ProductId, ProductName, UnitPrice
  from tblProductInventory
  Select CategoryId, CategoryName
  from tblProductCategories
End
```

And the program is illustrated in Listing 6.10 calls stored procedure and loads data into dataset.

```
string ConnectionString = ConfigurationManager.ConnectionStrings
    ["DBConnectionString"].ConnectionString;
using(SqlConnection connection = new SqlConnection(
    ConnectionString)) {
  SqlDataAdapter dataAdapter = new SqlDataAdapter("
    spGetProductAndCategoriesData", connection);
  dataAdapter.SelectCommand.CommandType = CommandType.
    StoredProcedure;
  DataSet dataset = new DataSet();
  dataAdapter.Fill(dataset);
  GridViewProducts.DataSource = dataset.Tables[0];
  GridViewProducts.DataBind();
  GridViewCategories.DataSource = dataset.Tables[1];
  GridViewCategories.DataBind();
}
```

Listing 6.10: Loading data into DataSet using

6.5 Adding, Deleting and Modifying Records in Dataset

You edit data in data tables (`DataSet`) much like you edit the data in a table in any database. The process can include inserting, updating, and deleting

records in the table.

Editing/Modifying records:

To edit an existing row in a `DataTable`, you need to locate the `DataRow` you want to edit, and then assign the updated values to the desired columns. If you don't know the index of the row you want to edit, use the `FindBy` method to search by the primary key.

Adding records:

To add new records to a dataset, create a new data row by calling the method on the `DataTable`. Then add the row to the `DataRow` collection (Rows) of the `DataTable`.

Deleting records:

Call the Delete method of a DataRow.

Example

```
class Program {
  static void Main(string[] args) {
    DataTable table1 = new DataTable("patients");
      table1.Columns.Add("name");
      table1.Columns.Add("id");
      table1.Rows.Add("ram", 1);
      table1.Rows.Add("shyam", 2);

    DataSet hospital = new DataSet("Hospital");
          hospital.Tables.Add(table1);

    //Editing/updating record in DataSet
    DataRow patient1 = table1.Rows[0];
          patient1["name"] = "khyam";

    //Adding new record in DataSet
    DataRow patient3 = table1.NewRow();
      patient3["name"] = "dam";
      patient3["id"] = 3;
      table1.Rows.Add(patient3);

    //Deleting record in DataSet
     table1.Rows[2].Delete();

```

```
25      Console.WriteLine(hospital.GetXml());
26      Console.ReadLine();
27    }
28  }
```

Listing 6.11: CRUD operation in dataset

6.6 Using DataView

- A `DataView` enables you to create different views of the data stored in a `DataTable`, a capability that is often used in data-binding applications.
- Using a `DataView`, you can expose the data in a table with different sort orders, and you can filter the data by row state or based on a filter expression.
- A `DataView` provides a dynamic view of data in the underlying `DataTable`:
  - the content, ordering, and membership reflect changes as they occur.
- This behavior differs from the Select method of the `DataTable`, which returns a `DataRow` array from a table based on a particular filter and/or sort order:
  - this content reflects changes to the underlying table, but its membership and ordering remain static.
- The dynamic capabilities of the `DataView` make it ideal for data-binding applications.
- A `DataView` provides you with a dynamic view of a single set of data, much like a database view, to which you can apply different sorting and filtering criteria. Unlike a database view, however, a `DataView` cannot be treated as a table and cannot provide a view of joined tables. You also cannot exclude columns that exist in the source table, nor can you append columns, such as computational columns, that do not exist in the source table.
- You can use a `DataViewManager` to manage view settings for all the tables in a `DataSet`.

- The `DataViewManager` provides you with a convenient way to manage default view settings for each table. When binding a control to more than one table of a `DataSet`, binding to a `DataViewManager` is the ideal choice.

Creating DataView

There are two ways to create a `DataView`. You can use the `DataView` constructor, or you can create a reference to the `DefaultView` property of the DataTable. The `DataView` constructor can be empty, or it can take either a `DataTable` as a single argument, or a `DataTable` along with filter criteria, sort criteria, and a row state filter.

The following code example demonstrates how to create a `DataView` using the `DataView` constructor. A `RowFilter`, `Sort` column, and `DataViewRowState` are supplied along with the `DataTable`.

```
DataView custDV = new DataView(custDS.Tables["Customers"], "
    Country = 'Nepal'", "ContactName", DataViewRowState.
    CurrentRows);
```

The following code example demonstrates how to obtain a reference to the default DataView of a DataTable using the DefaultView property of the table.

```
DataView custDV = custDS.Tables["Customers"].DefaultView;
```

Example of DataView

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace WindowsApplication1 {
  public partial class Form1: Form {
    public Form1() {
      InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e) {
      string connetionString = null;
      SqlConnection connection;
      SqlCommand command;
      SqlDataAdapter adapter = new SqlDataAdapter();
      DataSet ds = new DataSet();
```

```
18        DataView dv;
19        string sql = null;
20        connetionString = "Data Source=ServerName;Initial Catalog=
      DatabaseName;User ID=UserName;Password=Password";
21        sql = "Select * from product";
22        connection = new SqlConnection(connetionString);
23        try {
24          connection.Open();
25          command = new SqlCommand(sql, connection);
26          adapter.SelectCommand = command;
27          adapter.Fill(ds, "Create DataView");
28          adapter.Dispose();
29          command.Dispose();
30          connection.Close();
31
32          dv = ds.Tables[0].DefaultView;
33
34          dataGridView1.DataSource = dv;
35        }
36        catch(Exception ex) {
37          MessageBox.Show(ex.ToString());
38        }
39      }
40    }
41  }
```

Listing 6.12: Example of DataView

6.7 Working With DataGridVeiw

DataGridView displays data from SQL databases. This example takes a specific table from a database and display it on a DataGridView. This is done with a DataAdapter and data table. A visual representation of data is the end result.

```
1  void FillData() {
2    // 1 Open connection
3    using(SqlConnection c = new SqlConnection( Properties.Settings
       .Default.DataConnectionString)) {
4      c.Open();
5
6      // 2 Create new DataAdapter
7      using(SqlDataAdapter a = new SqlDataAdapter( "SELECT * FROM
       Animals", c)) {
8        // 3 Use DataAdapter to fill DataTable
9        DataTable t = new DataTable();
10       a.Fill(t);
```

```
11
12        // 4  Render data onto the screen
13        dataGridView1.DataSource = t;
14      }
15    }
16 }
```

Listing 6.13: Working with DataGridView.

Example of DataGridView

```
1  using System;
2  using System.Data;
3  using System.Windows.Forms;
4  using System.Data.SqlClient;
5
6  namespace WindowsFormsApplication1 {
7    public partial class Form1: Form {
8      SqlCommand sCommand;
9      SqlDataAdapter sAdapter;
10     SqlCommandBuilder sBuilder;
11     DataSet sDs;
12     DataTable sTable;
13
14     public Form1() {
15       InitializeComponent();
16     }
17
18     private void button1_Click(object sender, EventArgs e) {
19       string connectionString = "Data Source=.;Initial Catalog=
       pubs;Integrated Security=True";
20       string sql = "SELECT * FROM Stores";
21       SqlConnection connection = new SqlConnection(
       connectionString);
22       connection.Open();
23       sCommand = new SqlCommand(sql, connection);
24       sAdapter = new SqlDataAdapter(sCommand);
25       sBuilder = new SqlCommandBuilder(sAdapter);
26       sDs = new DataSet();
27       sAdapter.Fill(sDs, "Stores");
28       sTable = sDs.Tables["Stores"];
29       connection.Close();
30       dataGridView1.DataSource = sDs.Tables["Stores"];
31       dataGridView1.ReadOnly = true;
32       save_btn.Enabled = false;
33       dataGridView1.SelectionMode = DataGridViewSelectionMode.
       FullRowSelect;
34     }
```

```
35
36      private void new_btn_Click(object sender, EventArgs e) {
37        dataGridView1.ReadOnly = false;
38        save_btn.Enabled = true;
39        new_btn.Enabled = false;
40        delete_btn.Enabled = false;
41      }
42
43      private void delete_btn_Click(object sender, EventArgs e) {
44        if (MessageBox.Show("Do you want to delete this row ?", "
      Delete", MessageBoxButtons.YesNo) == DialogResult.Yes) {
45          dataGridView1.Rows.RemoveAt(dataGridView1.SelectedRows
      [0].Index);
46          sAdapter.Update(sTable);
47        }
48      }
49
50      private void save_btn_Click(object sender, EventArgs e) {
51        sAdapter.Update(sTable);
52        dataGridView1.ReadOnly = true;
53        save_btn.Enabled = false;
54        new_btn.Enabled = true;
55        delete_btn.Enabled = true;
56      }
57    }
58  }
```

Listing 6.14: Example of DataGridView CRUD

6.8 Calling Stored procedure

First, create the following stored procedure in MSSQL Server.

```
1  Create procedure spGetProductAndCategoriesData
2  as
3  Begin
4    Select ProductId, ProductName, UnitPrice
5    from tblProductInventory
6    Select CategoryId, CategoryName
7    from tblProductCategories
8  End
```

Listing 6.15: Stored procedure

And the program below calls stored procedure and loads data into dataset.

```
1  string ConnectionString = ConfigurationManager.ConnectionStrings
       ["DBConnectionString"].ConnectionString;
```

```
using(SqlConnection connection = new SqlConnection(
    ConnectionString)) {
  SqlDataAdapter dataAdapter = new SqlDataAdapter("
    spGetProductAndCategoriesData", connection);
    dataAdapter.SelectCommand.CommandType = CommandType.
    StoredProcedure;
    DataSet dataset = new DataSet();
    dataAdapter.Fill(dataset);
    GridViewProducts.DataSource = dataset.Tables[0];
    GridViewProducts.DataBind();
    GridViewCategories.DataSource = dataset.Tables[1];
    GridViewCategories.DataBind();
}
```

Listing 6.16: Calling stored procedure

CHAPTER

7

WEB APPLICATION

7.1 Basic Concepts of Web Application Development

A web application (or web app) is application software that runs on a web server, unlike software programs that are run locally on the operating system (OS) of the device. Web applications are accessed by the user through a web browser with an active internet connection. These applications are programmed using a client–server modeled structure—the user ("client") is provided services through an off-site server that is hosted by a third party. Examples of commonly-used web applications include: web-mail, online retail sales, online banking, and online auctions.

A Static Web Application is any web application that can be delivered directly to an end user's browser without any server-side alteration of the HTML, CSS, or JavaScript content. While this can encompass very flat, unchanging sites like a corporate web site, static web applications generally refer to rich sites that utilize technologies in the browser instead of on the server to deliver dynamic content.

A dynamic web application generates the pages/data in real time, as per the request, a respective response will trigger from the server end and will reach the client end. Depending upon the response the client side code will

take action as it's supposed to.

7.2 Implementing Session and Cookies

7.2.1 Cookies

A cookie is a token that the Web server embeds in a user's Web browser to identify the user. The next time the same browser requests a page, it sends the cookie it received from the Web server. Cookies allow a set of information to be associated with a user. ASP scripts can both get and set the values of cookies by using the `Response.Cookies` Collection collection of the **Response** and **Request** objects. Once a cookie has been set, all page requests that follow return the cookie name and value. A cookie can only be read from the domain that it has been issued from. If a cookie contains a collection of multiple values, we say that the cookie has Keys.
Note: *The maximum cookie size is 4KB.*

7.2.1.1 Types of Cookies

1. **Persistence Cookie**: Has an expiry time.
2. **Non-Persistence Cookie**: Maintains user information as long as the user accesses the same browser. It will be discarded when user closes the browser.

7.2.1.2 Set/Create Cookies

- To set the value of a cookie, use `Response.Cookies`
- If the cookie does not already exist, `Response.Cookies` creates a new one
- Cookie can have multiple values; such a cookie is called an indexed cookie.
- The `Response.Cookies` command must appear before the `<html>` tag.

```
<%
  // create a cookie named 'semester' with the value '8'
  Response.Cookies("semester")= 8

  // create a cookie named 'name' with the value 'Jeevan'
  Response.Cookies("name")="Jeevan"

```

```
  // assign 'Expires' property to a cookie
  Response.Cookies("name").Expires= "Jan 20,2021"
%>
```

A Cookie with Keys

```
<%
  /* create a cookie collection named 'student' */
  Response.Cookies("student")("firstname")="Jeevan"
  Response.Cookies("student")("lastname")="P"
  Response.Cookies("student")("district")="Taplejung"
%>
```

7.2.1.3 Get/Retrieve Cookies

- The `Request.Cookies` command is used to retrieve a cookie value
- In the example below, we retrieve the value of the cookie named 'name' and display it on a page:

```
<%
  name=Request.Cookies("name")
  response.write("Name=" & name)
  // output: Name=Jeevan
%>
```

```
<!DOCTYPE html>
<html>
<body>

<%
dim x,y
for each x in Request.Cookies
  response.write("<p>")
  if Request.Cookies(x).HasKeys then
    for each y in Request.Cookies(x)
      response.write(x & ":" & y & "=" & Request.Cookies(x)
    (y))
      response.write("<br>")
    next
  else
    Response.Write(x & "=" & Request.Cookies(x) & "<br>")
  end if
  response.write "</p>"
next
%>
```

```
<!-- /*
    student:firstname=Jee
    student:lastname=P
    student:district=Taplejung
    */
-->
</body>
</html>
```

Listing 7.1: Reading all cookies

7.2.2 Session

The web server does not know who you are and what you do, because the HTTP address doesn't maintain state.

ASP solves this problem by creating a unique cookie for each user. The cookie is sent to the user's computer and it contains information that identifies the user. This interface is called the Session object.

The Session object stores information about, or change settings for a user session.

Variables stored in a Session object hold information about one single user, and are available to all pages in one application. Common information stored in session variables are name, id, and preferences. The server creates a new Session object for each new user, and destroys the Session object when the session expires.

A session starts when:

- A new user requests an ASP file, and the Global.asa file includes a `Session_OnStart` procedure
- A value is stored in a Session variable
- A user requests an ASP file, and the `Global.asa` file uses the `<object>` tag to instantiate an object with session scope

A session ends if a user has not requested or refreshed a page in the application for a specified period. By default, this is 20 minutes.

If you want to set a timeout interval that is shorter or longer than the default, use the Timeout property.

The example below sets a timeout interval of 5 minutes:

```
<%
  Session.Timeout=5
%>
```

Use the `Abandon` method to end a session immediately:

```
<%
  Session.Abandon
%>
```

Note: The main problem with sessions is WHEN they should end. We do not know if the user's last request was the final one or not. So we do not know how long we should keep the session "alive". Waiting too long for an idle session uses up resources on the server, but if the session is deleted too soon the user has to start all over again because the server has deleted all the information. Finding the right timeout interval can be difficult!

Store and Retrieve Session Variables

The most important thing about the Session object is that you can store variables in it.

The example below will set the Session variable username to "Donald Duck" and the Session variable age to "50":

```
<%
  Session("username")="Donald Duck"
  Session("age")=50
%>
```

When the value is stored in a session variable it can be reached from ANY page in the ASP application:

```
Welcome <%Response.Write(Session("username"))%>
```

The line above returns: "Welcome Donald Duck".

You can also store user preferences in the Session object, and then access that preference to choose what page to return to the user.

The example below specifies a text-only version of the page if the user has a low screen resolution:

```
<%If Session("screenres")="low" Then%>
  This is the text version of the page
<%Else%>
  This is the multimedia version of the page
<%End If%>
```

Remove Session Variables

The Contents collection contains all session variables.

It is possible to remove a session variable with the Remove method.

The example below removes the session variable "sale" if the value of the session variable "age" is lower than 18:

```
<%
If Session.Contents("age")<18 then
  Session.Contents.Remove("sale")
End If
%>
```

To remove all variables in a session, use the RemoveAll method:

```
<%
  Session.Contents.RemoveAll()
%>
```

Loop Through The Contents Collection

The Contents collection contains all session variables. You can loop through the Contents collection, to see what's stored in it:

```
<%
  Session("username")="Donald Duck"
  Session("age")=50

  dim i
  For Each i in Session.Contents
  Response.Write(i & "<br>")
  Next

  /*  output:
    username
    age
  */
%>
```

If you do not know the number of items in the Contents collection, you can use the Count property:

```
<%
  dim i
  dim j
  j=Session.Contents.Count
  Response.Write("Session variables: " & j)
  For i=1 to j
  Response.Write(Session.Contents(i) & "<br>")
  Next

  /* output:
    Session variables: 2
```

```
    Donald Duck
    50
  */
%>
```

Loop Through The StaticObjects Collection

You can loop through the StaticObjects collection, to see the values of all objects stored in the Session object:

```
<%
  dim i
  For Each i in Session.StaticObjects
  Response.Write(i & "<br>")
  Next
%>
```

7.3 Client and Server Side Validation

Validation is a set of rules that you apply to the data you collect. These rules can be many or few and enforced either strictly or in a lax manner: It really depends on you. No perfect validation process exists because some users may find a way cheat to some degree, no matter what rules you establish. The trick is to find the right balance of the fewest rules and the proper strictness, without compromising the usability of the application.

The data you collect for validation comes from the Web forms you provide in your applications. Web forms are made up of different types of HTML elements that are constructed using raw HTML form elements, ASP . NET HTML server controls, or ASP . NET Web Form server controls. In the end, your forms are made up of many types of HTML elements, such as text boxes, radio buttons, check boxes, drop-down lists, and more.

7.3.1 Client Side Validation

Client-side validation is quick and responsive for the end user. It is something end users expect of the forms that they work with. If something is wrong with the form, using client-side validation ensures that the end user knows this as soon as possible. Client-side validation also pushes the processing power required of validation to the client meaning that you don't need to spin CPU cycles on the server to process the same information because the client can do the work for you.

With this said, client-side validation is the more insecure form of validation. When a page is generated in an end user's browser, this end user can look at the code of the page quite easily (simply by right-clicking his mouse in the browser and selecting View Code). When he does this, in addition to seeing the HTML code for the page, he can also see all the JavaScript that is associated with the page. If you are validating your form client-side, it doesn't take much for the crafty hacker to repost a form (containing the values he wants in it) to your server as valid. There are also the cases in which clients have simply disabled the client-scripting capabilities in their browsers - thereby making your validations useless. Therefore, client-side validation should be looked on as a convenience and a courtesy to the end user and never as a security mechanism

7.3.2 Server Side Validation

The more secure form of validation is server-side validation. Server-side validation means that the validation checks are performed on the server instead of on the client. It is more secure because these checks cannot be easily bypassed. Instead, the form data values are checked using server code (C# or VB) on the server. If the form isn't valid, the page is posted back to the client as invalid. Although it is more secure, server-side validation can be slow. It is sluggish simply because the page has to be posted to a remote location and checked. Your end user might not be the happiest surfer in the world if, after waiting 20 seconds for a form to post, he is told his e-mail address isn't in the correct format.

The best approach is always to perform client-side validation first and then, after the form passes and is posted to the server, to perform the validation checks again using server-side validation This approach provides the best of both worlds. It is secure because hackers can't simply bypass the validation. They may bypass the client-side validation, but they quickly find that their form data is checked once again on the server after it is posted.

7.3.3 ASP . NET Validation Server Controls

In the classic ASP days, developers could spend a great deal of their time dealing with different form validation schemes. For this reason, with the initial release of ASP . NET, the ASP . NET team introduced a series of validation server controls meant to make it a snap to implement sound validation for forms.

ASP . NET not only introduces form validations as server controls, but it also makes these controls rather smart. As stated earlier, one of the tasks

of classic ASP developers was to determine where to perform form validation — either on the client or on the server. The ASP . NET validation server controls eliminate this dilemma because ASP . NET performs browser detection when generating the ASP . NET page and makes decisions based on the information it gleans.

This means that if the browser can support the JavaScript that ASP . NET can send its way, the validation occurs on the client-side. If the client cannot support the JavaScript meant for client-side validation, this JavaScript is omitted and the validation occurs on the server.

The best part about this scenario is that even if client-side validation is initiated on a page, ASP . NET still performs the server-side validation when it receives the submitted page, thereby ensuring security won't be compromised. This decisive nature of the validation server controls means that you can build your ASP . NET Web pages to be the best they can possibly be.

The available validation controls include:

1. RequiredFieldValidator
2. CompareValidator
3. RangeValidator
4. RegularExpressionValidator
5. CustomValidator
6. ValidationSummary

The Table 7.1 describes the functionality of each of the available validation server controls.

7.3.3.1 The RequiredFieldValidator Server Control

- The `RequiredFieldValidator` control simply checks to see if something was entered into the HTML form element.
- It is a simple validation control, but it is one of the most frequently used.
- You must have a `RequiredFieldValidator` control for each form element on which you wish to enforce a *value-required* rule.

VB Listing 7.2 shows a simple use of the `RequiredFieldValidator` server control using VB.

```
1 <%@ Page Language="VB" %>
2 <script runat="server">
3     Protected Sub Button1_Click(ByVal sender As Object, ByVal e
    As System.EventArgs)
4     Label1.Text = "Page is valid!"
```

Table 7.1: Validation Server Controls.

Validation Server Control	Description
`RequiredFieldValidator`	Ensures that the user does not skip a form entry field
`CompareValidator`	Allows for comparisons between the user's input and another item using a comparison operator (equals, greater than, less than, and so on)
`RangeValidator`	Checks the user's input based upon a lower- and upper level range of numbers or characters
`RegularExpressionValidator`	Checks that the user's entry matches a pattern defined by a regular expression. This is a good control to use to check e-mail addresses and phone numbers
`CustomValidator`	Checks the user's entry using custom-coded validation logic
`ValidationSummary`	Displays all the error messages from the validators in one specific spot on the page

```
5       End Sub
6   </script>
7   <html xmlns="http://www.w3.org/1999/xhtml">
8   <head runat="server">
9       <title>RequiredFieldValidator</title>
10  </head>
11  <body>
12      <form id="form1" runat="server">
13          <div>
14              <asp:TextBox ID="TextBox1" Runat="server"></asp:
    TextBox>
15              <asp:RequiredFieldValidator ID="
    RequiredFieldValidator1" Runat="server" ErrorMessage="
    Required!" ControlToValidate="TextBox1">
16          </asp:RequiredFieldValidator>
```

```
        <br />
        <asp:Button ID="Button1" Runat="server" Text="Submit"
        OnClick="Button1_Click" />
        <br />
        <br />
        <asp:Label ID="Label1" Runat="server"></asp:Label>
    </div>
</form>
</body>
</html>
```

Listing 7.2: A simple use of the RequiredFieldValidator server control using VB.

C# Listing 7.3 shows a simple use of the `RequiredFieldValidator` server control using C#.

```
<%@ Page Language="C#" %>
<script runat="server">
    protected void Button1_Click(Object sender, EventArgs e) {
        Label1.Text = "Page is valid!";
    }
</script>
```

Listing 7.3: A simple use of the RequiredFieldValidator server control using C#.

7.3.3.2 The CompareValidator Server Control

- The `CompareValidator` control allows you to make comparisons between two form elements as well as to compare values contained within form elements to constants that you specify.
- For instance, you can specify that a form element's value must be an integer and greater than a specified number.
- You can also state that values must be strings, dates, or other data types that are at your disposal.

VB Listing 7.4 shows the use of `CompareValidator`.

```
<%@ Page Language="VB" %>
<script runat="server">
   Protected Sub Button1_Click(sender As Object, e As EventArgs)
   Label1.Text = "Passwords match"
   End Sub
```

```
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>CompareFieldValidator</title>
   </head>
   <body>
      <form runat="server">
         <p>
            Password<br>
            <asp:TextBox ID="TextBox1" Runat="server"
               TextMode="Password"></asp:TextBox>
             
            <asp:CompareValidator ID="CompareValidator1"
               Runat="server" ErrorMessage="Passwords do not
   match!"
               ControlToValidate="TextBox2"
               ControlToCompare="TextBox1"></asp:
   CompareValidator>
         </p>
         <p>
            Confirm Password<br>
            <asp:TextBox ID="TextBox2" Runat="server"
               TextMode="Password"></asp:TextBox>
         </p>
         <p>
            <asp:Button ID="Button1" OnClick="Button1_Click"
               Runat="server" Text="Login"></asp:Button>
         </p>
         <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
         </p>
      </form>
   </body>
</html>
```

Listing 7.4: Using the CompareValidator to test values against other control values

C# Listing 7.5 shows the use of `CompareValidator` with C#.

```
   <%@ Page Language="C#" %>
   <script runat="server">
      protected void Button1_Click(Object sender, EventArgs e) {
         Label1.Text = "Passwords match";
      }
   </script>
```

Listing 7.5: Using the CompareValidator to test values against other control values with C#

7.3.3.3 The RangeValidator Server Control

- The `RangeValidator` control is quite similar to that of the `CompareValidator` control, but it makes sure that the end user value or selection provided is between a specified range as opposed to being just greater than or less than a specified constant.
- For an example of this, go back to the text-box element that asks for the age of the end user and performs a validation on the value provided.

```
Age:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
 
<asp:RangeValidator ID="RangeValidator1" Runat="server"
ControlToValidate="TextBox1" Type="Integer"
ErrorMessage="You must be between 30 and 40"
MaximumValue="40" MinimumValue="30"></asp:RangeValidator>
```

Listing 7.6: Using the RangeValidator control to test an integer value.

VB Listing 7.7 shows the use of `RangeValidator` control to test a string date value with VB.

```
1  <%@ Page Language="VB" %>
2  <script runat="server">
3      Protected Sub Page_Load(ByVal sender As Object, ByVal e As
       System.EventArgs)
4          RangeValidator1.MinimumValue = DateTime.Now.
       ToShortDateString()
5          RangeValidator1.MaximumValue = DateTime.Now.AddDays(14).
       ToShortDateString()
6      End Sub
7
8      Protected Sub Calendar1_SelectionChanged(ByVal sender As
       Object, ByVal e As System.EventArgs)
9          TextBox1.Text = Calendar1.SelectedDate.ToShortDateString
       ()
10     End Sub
11
12     Protected Sub Button1_Click(ByVal sender As Object, ByVal e
       As System.EventArgs)
13         If Page.IsValid Then
14             Label1.Text = "You are set to arrive on: " &
       TextBox1.Text
15         End If
16     End Sub
17 </script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Date Validation Check</title>
</head>
<body>
    <form id="form1" runat="server">
        Arrival Date:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox
    > 
        <asp:RangeValidator ID="RangeValidator1" runat="server"
        ErrorMessage="You must only select a date within the
    next two weeks."
        ControlToValidate="TextBox1" Type="Date"></asp:
    RangeValidator><br />
        <br />
        Select your arrival date:<br />
        <asp:Calendar ID="Calendar1" runat="server"
        OnSelectionChanged="Calendar1_SelectionChanged"></asp:
    Calendar>
         
        <br />
        <asp:Button ID="Button1" runat="server" Text="Button"
        OnClick="Button1_Click" />
        <br />
        <br />
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </form>
</body>
</html
```

Listing 7.7: Using the RangeValidator control to test a string date value with VB

C# Listing 7.8 shows the use of `RangeValidator` control to test a string date value with C#.

```
<%@ Page Language="C#" %>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e) {
        RangeValidator1.MinimumValue = DateTime.Now.
    ToShortDateString();
        RangeValidator1.MaximumValue =
        DateTime.Now.AddDays(14).ToShortDateString();
    }
    protected void Calendar1_SelectionChanged(object sender,
    EventArgs e){
        TextBox1.Text = Calendar1.SelectedDate.ToShortDateString
    ();
```

```
    }
    protected void Button1_Click(object sender, EventArgs e){
        if (Page.IsValid){
            Label1.Text = "You are set to arrive on: " +
   TextBox1.Text.ToString();
        }
    }
</script>
```

Listing 7.8: Using the RangeValidator control to test a string date value with C#

7.3.3.4 The RegularExpressionValidator Server Control

- This control offers a lot of flexibility when you apply validation rules to your Web forms.
- Using the `RegularExpressionValidator` control, you can check a user's input based on a pattern that you define using a regular expression.
- This means that you can define a structure that a user's input will be applied against to see if its structure matches the one that you define.
- For instance, you can define that the structure of the user input must be in the form of an e-mail address or an Internet `URL`; if it doesn't match this definition, the page is considered invalid.

Listing 7.9 shows how to validate what is input into a text box by making sure it is in the form of an e-mail address.

```
Email:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
 
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
Runat="server" ControlToValidate="TextBox1"
ErrorMessage="You must enter an email address"
ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)
    *">
</asp:RegularExpressionValidator>
```

Listing 7.9: Making sure the text-box value is an e-mail address

Just like the other validation server controls, the `RegularExpressionValidator` control uses the `ControlToValidate` property to bind itself to the `TextBox` control, and it includes an

`ErrorMessage` property to push out the error message to the screen if the validation test fails. The unique property of this validation control is the `ValidationExpression` property. This property takes a string value, which is the regular expression you are going to apply to the input value.

7.3.3.5 The CustomValidator Server Control

- Validation controls described above are enough for us but if none of those works for us then we can use *CustomValidator* control.
- The `CustomValidator` control allows you to build your own client-side or server-side validations that can then be easily applied to your Web forms.
- Doing so allows you to make validation checks against values or calculations performed in the data tier (for example, in a database), or to make sure that the user's input validates against some arithmetic validation (for example, determining if a number is even or odd).

Using Client Side Validation

- One of the worthwhile functions of the `CustomValidator` control is its capability to easily provide custom client-side validations.
- Many developers have their own collections of JavaScript functions they employ in their applications, and using the `CustomValidator` control is one easy way of getting these functions implemented.
- For example, look at a simple form that asks for a number from the end user.
- This form uses the `CustomValidator` control to perform a custom client-side validation on the user input to make sure that the number provided is divisible by 5.
- This is illustrated in Listing 7.10.

VB Listing 7.10 illustrates the CustomValidator control to perform client-side validations with VB.

```
1 <%@ Page Language="VB" %>
2 <script runat="server">
3   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
      System.EventArgs)
```

```
      Label1.Text = "VALID NUMBER!"
  End Sub
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>CustomValidator</title>
    <script language="JavaScript">
      function validateNumber(oSrc, args) {
          args.IsValid = (args.Value % 5 == 0);
      }
    </script>
  </head>
  <body>
    <form id="form1" runat="server">
      <div>
        <p>
          Number:
          <asp:TextBox ID="TextBox1"
            Runat="server"></asp:TextBox>
           
          <asp:CustomValidator ID="CustomValidator1"
            Runat="server" ControlToValidate="TextBox1"
            ErrorMessage="Number must be divisible by 5"
            ClientValidationFunction="validateNumber">
          </asp:CustomValidator>
        </p>
        <p>
          <asp:Button ID="Button1" OnClick="Button1_Click"
            Runat="server" Text="Button"></asp:Button>
        </p>
        <p>
          <asp:Label ID="Label1" Runat="server"></asp:Label>
        </p>
      </div>
    </form>
  </body>
</html>
```

Listing 7.10: Using the CustomValidator control to perform client-side validations with VB.

C# Listing 7.11 illustrates the CustomValidator control to perform client-side validations with C#.

```
<%@ Page Language="C#" %>
<script runat="server">
  protected void Button1_Click(Object sender, EventArgs e) {
    Label1.Text = "VALID NUMBER!";
```

```
5   }
6 </script>
```

Listing 7.11: Using the CustomValidator control to perform client-side validations with C#.

Looking over this Web form, you can see a couple of things happening. First, it is a simple form with only a single text box requiring user input. The user clicks the button that triggers the `Button1_Click` event that, in turn, populates the Label1 control on the page. It carries out this simple operation only if all the validation checks are performed and the user input passes these tests.

Using Server-Side Validation

- Now let's move this same validation check from the client to the server.
- The `CustomValidator` control allows you to make custom server-side validations a reality as well.
- `CustomValidator` for server-side validations is something you do if you want to check the user's input against dynamic values coming from XML files, databases, or elsewhere.
- For an example of using the `CustomValidator` control for some custom server-side validation, you can work with the same example as you did when creating the client-side validation.
- Now, create a server-side check that makes sure a user-input number is divisible by 5.
- This is illustrated in Listing 7.12 and 7.13.

VB Using the CustomValidator control to perform server-side validations with VB.

```
1 <%@ Page Language="VB" %>
2 <script runat="server">
3   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
      System.EventArgs)
4       If Page.IsValid Then
5           Label1.Text = "VALID ENTRY!"
6       End If
7   End Sub
8   Sub ValidateNumber(sender As Object, args As
     ServerValidateEventArgs)
```

```
9       Try
10          Dim num As Integer = Integer.Parse(args.Value)
11          args.IsValid = ((num mod 5) = 0)
12      Catch ex As Exception
13          args.IsValid = False
14      End Try
15  End Sub
16 </script>
17 <html xmlns="http://www.w3.org/1999/xhtml" >
18   <head runat="server">
19     <title>CustomValidator</title>
20   </head>
21   <body>
22     <form id="form1" runat="server">
23       <div>
24         <p>
25           Number:
26           <asp:TextBox ID="TextBox1"
27             Runat="server"></asp:TextBox>
28            
29           <asp:CustomValidator ID="CustomValidator1"
30             Runat="server" ControlToValidate="TextBox1"
31             ErrorMessage="Number must be divisible by 5"
32             OnServerValidate="ValidateNumber"></asp:
    CustomValidator>
33         </p>
34         <p>
35           <asp:Button ID="Button1" OnClick="Button1_Click"
36             Runat="server" Text="Button"></asp:Button>
37         </p>
38         <p>
39           <asp:Label ID="Label1" Runat="server"></asp:Label>
40         </p>
41       </div>
42     </form>
43   </body>
44 </html>
```

Listing 7.12: Using the CustomValidator control to perform server-side validations with VB.

C# Using the CustomValidator control to perform server-side validations with C#.

```
1 <%@ Page Language="C#" %>
2 <script runat="server">
3   protected void Button1_Click(Object sender, EventArgs e) {
4     if (Page.IsValid) {
```

```
      Label1.Text = "VALID ENTRY!";
    }
  }
  void ValidateNumber(object source, ServerValidateEventArgs
   args) {
    try {
      int num = int.Parse(args.Value);
      args.IsValid = ((num%5) == 0);
    }
    catch(Exception ex) {
      args.IsValid = false;
    }
  }
</script>
```

Listing 7.13: Using the CustomValidator control to perform server-side validations with C#.

Instead of a client-side JavaScript function in the code, this example includes a server-side function — `ValidateNumber`. The `ValidateNumber` function, as well as all functions that are being constructed to work with the `CustomValidator` control, must use the `ServerValidateEventArgs` object as one of the parameters in order to get the data passed to the function for the validation check. The `ValidateNumber` function itself is nothing fancy. It simply checks to see if the provided number is divisible by 5.

7.3.3.6 The ValidationSummary Server Control

- The `ValidationSummary` control is not a control that performs validations on the content input into your Web forms.
- Instead, this control is the reporting control, which is used by the other validation controls on a page.
- You can use this validation control to consolidate error reporting for all the validation errors that occur on a page instead of leaving this up to each and every individual validation control.
- By default, the `ValidationSummary` control shows the list of validation errors as a bulleted list.
- This is illustrated in Listing 7.14.

```
<p>
    First name
    <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
```

```
4      
5     <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
6       Runat="server" ErrorMessage="You must enter your first
      name"
7       ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
8   </p>
9   <p>
10    Last name
11    <asp:TextBox ID="TextBox2" Runat="server"></asp:TextBox>
12     
13    <asp:RequiredFieldValidator ID="RequiredFieldValidator2"
14      Runat="server" ErrorMessage="You must enter your last name
      "
15      ControlToValidate="TextBox2"></asp:RequiredFieldValidator>
16  </p>
17  <p>
18    <asp:Button ID="Button1" OnClick="Button1_Click" Runat="
      server"
19      Text="Submit"></asp:Button>
20  </p>
21  <p>
22    <asp:ValidationSummary ID="ValidationSummary1" Runat="server
      "
23      HeaderText="You received the following errors:"></asp:
      ValidationSummary>
24  </p>
25  <p>
26    <asp:Label ID="Label1" Runat="server"></asp:Label>
27  </p>
```

Listing 7.14: A partial page example of the ValidationSummary control.

This example asks the end user for her first and last name. Each text box in the form has an associated `RequiredFieldValidator` control assigned to it. When the page is built and run, if the user clicks the Submit button with no values placed in either of the text boxes, it causes both validation errors to fire.

As in earlier examples of validation controls on the form, these validation errors appear next to each of the text boxes. You can see, however, that the `ValidationSummary` control also displays the validation errors as a bulleted list in red at the location of the control on the Web form. In most cases, you do not want these errors to appear twice on a page for the end user.

7.4 Building Web Application

Hosting

Internet Information Services (IIS)

Stands for "Internet Information Services". IIS is a web server software package designed for Windows Server. It is used for hosting websites and other content on the Web.

Microsoft's Internet Information Services provides a graphical user interface (GUI) for managing websites and the associated users. It provides a visual means of creating, configuring, and publishing sites on the web. The IIS Manager tool allows web administrators to modify website options, such as default pages, error pages, logging settings, security settings, and performance optimizations.

IIS can serve both standard HTML webpages and dynamic webpages, such as ASP . NET applications and PHP pages. When a visitor accesses a page on a static website, IIS simply sends the HTML and associated images to the user's browser. When a page on a dynamic website is accessed, IIS runs any applications and processes any scripts contained in the page, then sends the resulting data to the user's browser.

While IIS includes all the features necessary to host a website, it also supports extensions (or "modules") that add extra functionality to the server. For example, the WinCache Extension enables PHP scripts to run faster by caching PHP processes. The URL Rewrite module allows webmasters to publish pages with friendly URLs that are easier for visitors to type and remember. A streaming extension can be installed to provide streaming media to website visitors.

IIS is a popular option for commercial websites, since it offers many advanced features and is supported by Microsoft. However, it also requires requires a commercial license and the pricing increases depending on the number of users. Therefore, Apache HTTP Server, which is open source and free for unlimited users, remains the most popular web server software.

Features of IIS

Application pools Application pools form an important part of an IIS server system. An individual application pool could have zero or many IIS worker processes running. These worker processes are responsible for running application instances.

Authentication IIS server features authentication options, including Windows auth, Basic, and ASP . NET. If you use Windows Active Directory, Windows auth is especially useful, because it lets you sign in to web apps automatically via your domain account.

Security IIS comes with security features, like utilities for managing TLS certificates, binding so SFTP and HTTPS can be enabled, and the ability to filter requests so you can effectively whitelist and blacklist traffic. You can implement authorization and permission rules and log requests and access a suite of FTP security functions.

Remote management Remote management utilities allow IIS to be managed through the CLI or via PowerShell. You can create the script yourself, which many IT administrators value because it offers ultimate flexibility and control.

PURBANCHAL UNIVERSITY

2018/ २०७५

4 Years Bachelor of Computer Application (BCA/Eighth Semester/Final)

Time: 3.00 hrs. Full Marks: 60 / Pass Marks: 24

BCA453CO, Dot Net Programming

Candidates are required to give their own answers in their own words as far as practicable. Figure in the margin indicate full marks.

Group A

Answer TWO questions. **$2 \times 12 = 24$**

1. What are the components of NET framework? How do you implement OOP in C#? Explain with example.
2. What is ADO.NET? Explain ADO.NET architecture. Write a program to insert, delete and update employee records (Name, Age, Address, and Basic Salary) in database with proper validation. 2+4+6
3. (a) What is hosting? Write about IIS Web Server and its features
   (b) What is the use of data bound control in web application? Give example.

Group B

Answer SIX questions. **$6 \times 6 = 36$**

4. What are feature of VB.net? Explain event handling in C#with example.
5. What is a constructor? How do you overload methods in VB.NET?
6. What do you mean by concurrency control? How is concurrency control resolved and maintained in ADO.NET? Explain with example.
7. Write a program to demonstrate the use of smart tags.
8. Explain data grid control used in .NET programming with example.
9. Write a VB.NET client application and incorporate the necessary validations.
10. Write notes on any TWO:
    (a) Cross language architecture
    (b) Namespace
    (c) .NET class libraries

PURBANCHAL UNIVERSITY

2019/ २०७६

4 Years Bachelor of Computer Application (BCA/Eighth Semester/Final)

Time: 3.00 hrs. Full Marks: 60 / Pass Marks: 24

BCA453CO, Dot Net Programming

Candidates are required to give their own answers in their own words as far as practicable. Figure in the margin indicate full marks.

Group A

Answer TWO questions. **$2 \times 12 = 24$**

1. Describe Architecture of NET Framework. Explain Just in Time compilation of managed code.

2. What do you mean by polymorphism? Describe with its types and example.

3. Write a program for Electricity Bill Statement (EBS). The EBS takes units consumed from consumer and calculates; Electricity Charges (EC) using provided criteria:
   (i) Units less than 100 Rs 500 minimum charge.
   (ii) Units 100 above 300 below Rs 15 (per unit + minimum charge).
   (iii) Units above 300 Rs 25 (per unit + minimum charge).

Group B

Answer SIX questions. **$6 \times 6 = 36$**

4. What is the difference between a class and an object, and how do these terms relate to each other? Explain briefly with examples?

5. List different features of . NET Technology. What does 'managed code' means in . NET context?

6. What are the difference between Radio Button control and Check Box control? Explain with an example.

7. Why do we use String Builder in place of String in C#. NET? Differentiate between string and String Builder.

8. Explain ADO.NET, its features and architecture.

9. Print the following pattern:

   2
   3 5
   7 11 13
   17 19 23 29
   31 37 41 43 47

10. What is array? Explain types of array with example.

11. Write notes on any TWO:
    (a) Base Class Library
    (b) Boxing and Unboxing
    (c) Inheritance

REFERENCES

Uprety, M. (2020). Dot Net Programming (C#, VB & ASP): BCA-VIII. *Gomendra Multiple College*. http://madan.surge.sh/

Bill Evjen, Scott Hanselman, Farhan Muhammad, Srinivasa Sivakumarm, & Devin Rader. (n.d.). *Professional asp.net 2.0*. Wiley Publishing, Inc. https://www.wiley.com/en-us/Professional+ASP+NET+2+0+-p-9780764576102

Difference between object and class - javatpoint. (n.d.). Retrieved February 21, 2021, from https://www.javatpoint.com/difference-between-object-and-class

C# Interface - javatpoint. (n.d.). Retrieved February 21, 2021, from https://www.javatpoint.com/c-sharp-interface

VB.NET Classes and Object - Javatpoint. (n.d.). Retrieved February 21, 2021, from https://www.javatpoint.com/vb-net-classes-and-object

Command-Line Arguments - C# Programming Guide. (n.d.). https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/main-and-command-args/command-line-arguments

Objects - C# Programming Guide. (n.d.). https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/objects

C# Label Control. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/label.htm

C# Button Control. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/button.htm

C# TextBox Control. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/cs-textbox.htm

C# ComboBox. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/cs-combobox.htm

C# Checked ListBox Control. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/cs-checkedlistbox.htm

C# RadioButton Control. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/cs-radiobutton.htm

C# CheckBox Control. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/cs-checkbox.htm

C# ProgressBar Control. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/cs-progressbar.htm

C# DateTimePicker Control. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/cs-datetimepicker.htm

C# Menu Control. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/cs-menu.htm

C# MDI Form. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/cs-mdi-form.htm

C# Timer Control. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/timer-cs.htm

C# PictureBox Control. (n.d.). Retrieved January 4, 2021, from http://csharp.net-informations.com/gui/cs-picturebox.htm

Standard Controls and Components C# Assignment Help, Online C Sharp Project & Homework Help. (2015). Retrieved January 4, 2021, from https://csharpaid.com/standard-controls-and-components-14171

Procedures - Visual Basic. (n.d.). Retrieved January 21, 2021, from https://docs.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/procedures/

KathleenDollard. (n.d.). Sub procedures - Visual Basic. Retrieved January 21, 2021, from https://docs.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/procedures/sub-procedures

VB.Net Programming Tutorial. (2015). Retrieved January 26, 2021, from https://www.tutorialspoint.com/vb.net/

Ultimate Guide to IIS Server: What Is IIS? IIS Tutorial - DNSstuff. (2020). Retrieved January 28, 2021, from https://www.dnsstuff.com/windows-iis-server-tools

What is Microsoft .Net Framework. (n.d.). Retrieved January 28, 2021, from http://vb.net-informations.com/framework/what_is_net_framework.htm

How to Managed Code - Microsoft .Net Framework. (n.d.). Retrieved January 28, 2021, from http://vb.net-informations.com/framework/managed_code.htm

What is Microsoft Just In Time Compiler. (n.d.). Retrieved January 28, 2021, from http://vb.net-informations.com/framework/just_in_time_compiler.htm

What is Microsoft Intermediate Language. (n.d.). Retrieved January 28, 2021, from http://vb.net-informations.com/framework/microsoft_intermediate_language.htm

What is Common Type System. (n.d.). Retrieved January 28, 2021, from http://vb.net-informations.com/framework/common_type_system.htm

What is Common Language Specification. (n.d.). Retrieved January 28, 2021, from http://vb.net-informations.com/framework/common_language_specification.htm

What is .Net Framework Class Library. (n.d.). Retrieved January 28, 2021, from http://vb.net-informations.com/framework/framework_class_library.htm

How to Common Language Runtime. (n.d.). Retrieved January 28, 2021, from http://vb.net-informations.com/framework/functions_of_common_language_runtime.htm

What are the functions of microsoft .net framework? (n.d.). Retrieved January 28, 2021, from http://vb.net-informations.com/framework/How_to_net_framework.htm

Advantages of ADO.Net over ADO. (n.d.). Retrieved January 28, 2021, from http://vb.net-informations.com/ado.net/ado.net-advantages.htm

Andy De George. (n.d.). Types of custom controls - Windows Forms .NET. Retrieved January 28, 2021, from https://docs.microsoft.com/en-us/dotnet/desktop/winforms/controls/custom

ADO.Net Connection - javatpoint. (n.d.). Retrieved January 28, 2021, from https://www.javatpoint.com/ado-net-connection

ADO.Net Command - javatpoint. (n.d.). Retrieved January 28, 2021, from https://www.javatpoint.com/ado-net-command

ADO.Net Datareader - javatpoint. (n.d.). Retrieved January 28, 2021, from https://www.javatpoint.com/ado-net-datareader

adegeo. (n.d.). User input validation - Windows Forms .NET. Retrieved February 21, 2021, from https://docs.microsoft.com/en-us/dotnet/desktop/winforms/input-keyboard/validation

C# DataGridView Database Operations. (n.d.). Retrieved February 22, 2021, from http://csharp.net-informations.com/datagridview/csharp-datagridview-database-operations.htm

How to create a DataView. (n.d.). Retrieved February 22, 2021, from http://csharp.net-informations.com/dataview/create-dataview.htm