Mahindra Satyam

Workbook for C#.Net

A beginner's guide to effective programming

Why This Module?

Many programming languages suffice to support the programmers. Earlier with C programming language numerous applications has been created successfully. However, C is a structured language, so the application is deprived of the benefits provided by object oriented approach which could be better implemented with the same. C++ as a programming language benefits the programmer with its object-oriented approach like encapsulation, inheritance and polymorphism; however they are still the painful aspects of the C language like manual memory management, ugly pointer arithmetic, and ugly syntactical constructs. VB6 is popular due to its ability to build complex user interfaces, code libraries, etc. Again VB6 is not fully object-oriented.

It is very much obvious that programmers need to be self-inflicted to stay tune with the current technologies. C# is new programming language designed specifically to work with .NET. Using C# you can, for example, develop a dynamic Web page, an XML web services, a component of a distributed application, a database access component, or a classic windows desktop application. C# and .Net both are set to revolutionize the way that you write programs, and to programming on windows and console much easier than it has ever been.. It also provides automatic memory management through garbage collection. Most important point to understand about the C# language is it can only produce code which can be executed within .Net runtime.

In this module, participant will understand to build generic types and generic members which build very efficient and type-safe code. They will learn about the concepts of object oriented that can be implemented while making complex applications. Then, the participants will learn to define single type across multiple code files using partial keyword. With the knowledge of assembly, they can version their application as well as reuse their code for perfection.

Contents

1. Introduction to .NET Framework 8-13

1.1 .NET Framework 9
1.2 .NET My Services 9
1.3 .Net Enterprise Server 9
1.4 Visual Studio .Net 9
1.5 Overview of the .Net Application 9
1.6 Benefits of the .Net Framework 9
1.7 Crossword 13

2. .Net Framework Components and Languages 14-23

2.1 Basic Input/Output Operation 15
2.2 Working with Files 17
2.3 Structure of C# program 21
2.4 Crossword 23

3. Using Value Type Variables 24-37

3.1 Common Type System 25
3.2 Naming Variable 29
3.3 Built in Data Types 32
3.4 Creating user Defined Data Types 32
3.5 Converting Data Types 32
3.6 Crossword 37

4. Statements and Exception 38-48

4.1 Introduction to Statements 39
4.2 Exception Handling 44
4.3 Crossword 48

5. Methods and Parameters 49-61

5.1 Methods 50
5.2 Parameters 53
5.3 Overloading Parameters 57
5.4 Crossword 61

6. Arrays 62-69

6.1 Overview of Arrays 63
6.2 Jagged Arrays 66
6.3 Crossword 69

7. Essential of OOP 70-81

7.1 Classes and Objects 71
7.2 Encapsulation 75
7.3 Object Oriented Features 78
7.4 Crossword 81

8. Using Reference type Variable 82-94

8.1 Usage of Reference type variables 83
8.2 Namespaces in the .NET framework 87
8.3 Data Conversions 90
8.4 Crossword 94

9. Creating and Destroying Objects 95-104

9.1 Constructors 96
9.2 Resource Management 100
9.3 Crossword 104

10. Inheritance 105-120

10.1 Deriving Classes 106
10.2 Using Interfaces and Abstract Classes 109
10.3 Structured Exception Handling 117
10.4 Crossword 120

11. Aggregation, Namespaces and Advanced Scopes 121-134

11.1 Aggregation 122
11.2 Modules and Assemblies 125

11.3 Collections and Generics 128
11.4 Crossword 134

12. Operator Delegates and Events 135-144
12.1 Operator Overloading 136
12.2 Creating and using Delegates 141
12.3 Crossword 144

13. Properties and Indexers 145-153
13.1 Using Properties 146
13.2 Indexers 149
13.3 Crossword 153

14. Attributes 155-159
14.1 Overview of Attributes 155
14.2 Retrieving attribute Values 155
14.3 Crossword 159

**Answers For Crosswords 160-161
**Contributors 162-163

Guide to Use this Workbook

Conventions Used

Convention	Description

Topic	Indicates the Topic to be discussed.
Estimated Time	Overview of estimated time needed to understand the Topic and complete the Practice session.
Presentation	A brief introduction about the Topic.
Scenario	An idea on real time situation where topics are used.
Demonstration/Code Snippet	Topics implemented in real time code along with screenshots
`Code in Italic`	Italic framed lines are generated by the System related to that particular event.
// OR ‘	Topic is being described from complete program as code snippet
Context	When the Topic should be used in an application.
Practice Session	Participant should be able to implement the Topic to develop an Application in practice sessions.
Check list	List of brief content of the Topic.
Common Errors	List of common errors occured while developing an Application.
Exceptions	List of exceptions resulted from the execution of an Application.
Lessons Learnt	Lessons learnt from the article of the workbook.
Best Practices	Efficient development of the an Application through best ways.

Notes	Important information related to the Topic

1.0 Introduction to .NET Platform

Topics

- 1.1 .Net Framework
- 1.2 .Net My Services
- 1.3 The .Net Enterprise Servers
- 1.4 Visual Studio .Net
- 1.5 Overview of the .Net Applications
- 1.6 Benefits of the .NET Framework
- 1.7 Crossword

Topic: Microsoft .NET Platform

Estimated Time: 40 mins.

Objectives: After the completion of module, participant should know

- Benefits of .Net
- Components of .Net Framework

Presentation :

Brief of .Net Framework

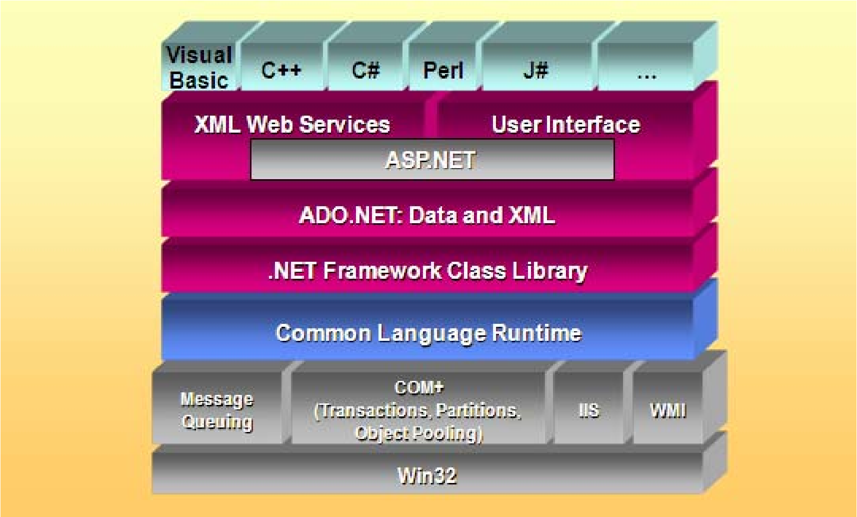


Figure 1-1: .Net Framework

Benefits of .Net Framework

- Based on Web standards and practices
- Designed using unified application models
- Easy for developers to use
- Extensible classes

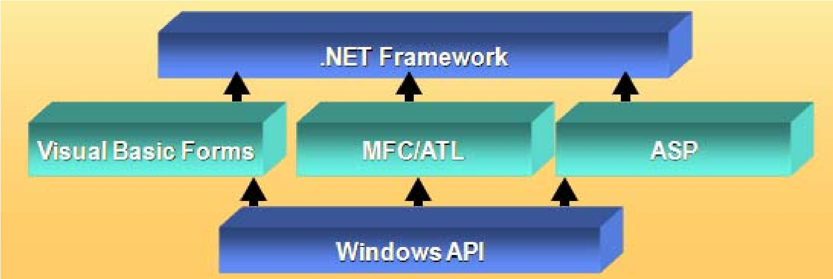


Figure 1-2: Interaction with Windows and .Net Framework

Components of .Net Framework

- Common Language Runtime.

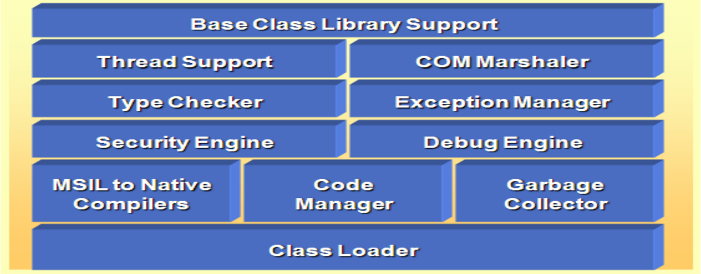


Figure 1-3: Common Language Runtime

- .NET Framework Class Library.

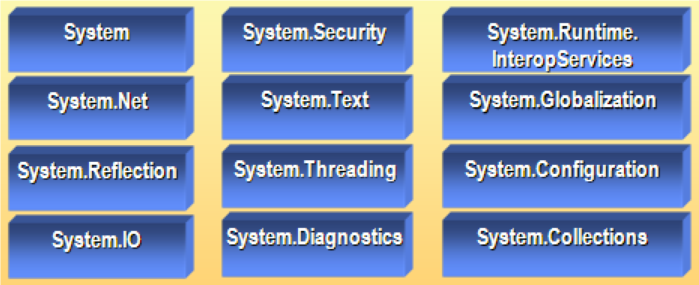


Figure 1-3: .Net Framework Class Library

- ADO.NET: Data and XML.

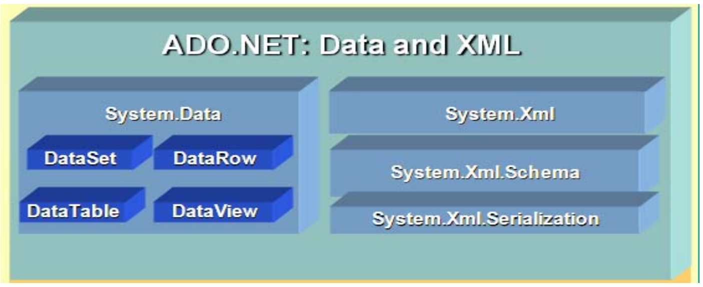


Figure 1-4: Data and XML

- Web Forms and XML Web Services.

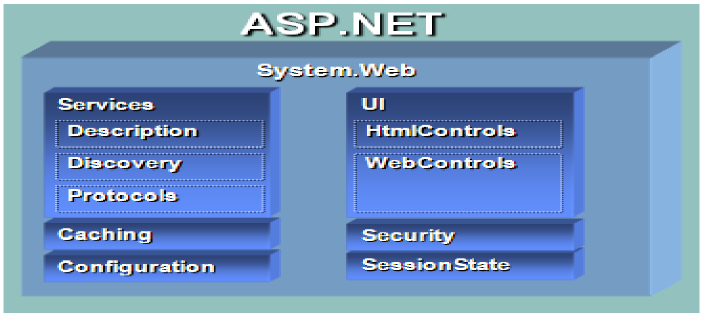


Figure 1-5: Web Forms and XML Web Services

- User Interface for Windows
  - ➢ System.Windows.Forms
  - ➢ System. Drawing

Scenario:

Satyam provides services to clients and deals with applications like banking, medical, insurance, etc. Application oriented services includes authorized login, web services, etc which is supported by .Net framework.

Demonstration/Code Snippet :

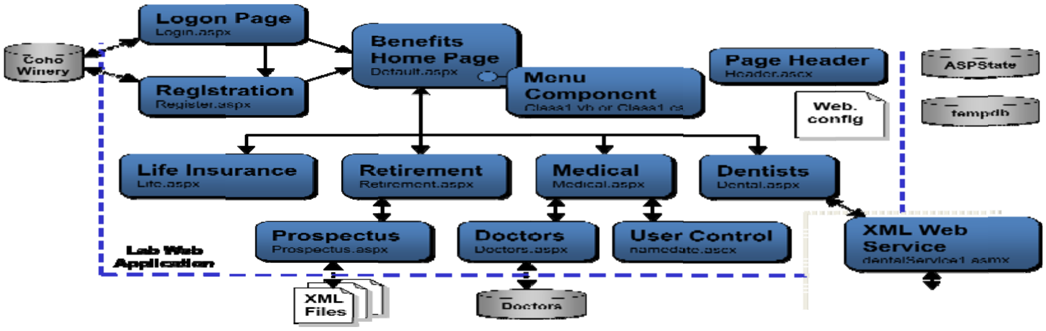


Figure 1-6:Using .NET framework

.Net framework provides code-based security, where as windows provide Role-based security, code based security which reduces risk associated with code because it's based on what the code actually does and how much code is trusted.

CLR translates code into Microsoft Intermediate Language and this MSIL code quickly translates into machine code.

Context:

- For managing the program's runtime requirements
- To implement powerful, cost-effective applications
- To build windows and web applications

Practice Session:

- List down the classes available in the namespace System.IO, Identify the tools to free the unused resources

Check List:

- The Benefits of .NET Framework components
- The languages supported by .NET Framework

Common Errors:

- Implementing low level API without checking for the availability of functionality in .Net
- Usage of VB.Net for making software drivers.
- Assuming .Net to be a language.

Exception :

- Trying to access services provided by framework 2.0, where as the system supports .Net version 1.1

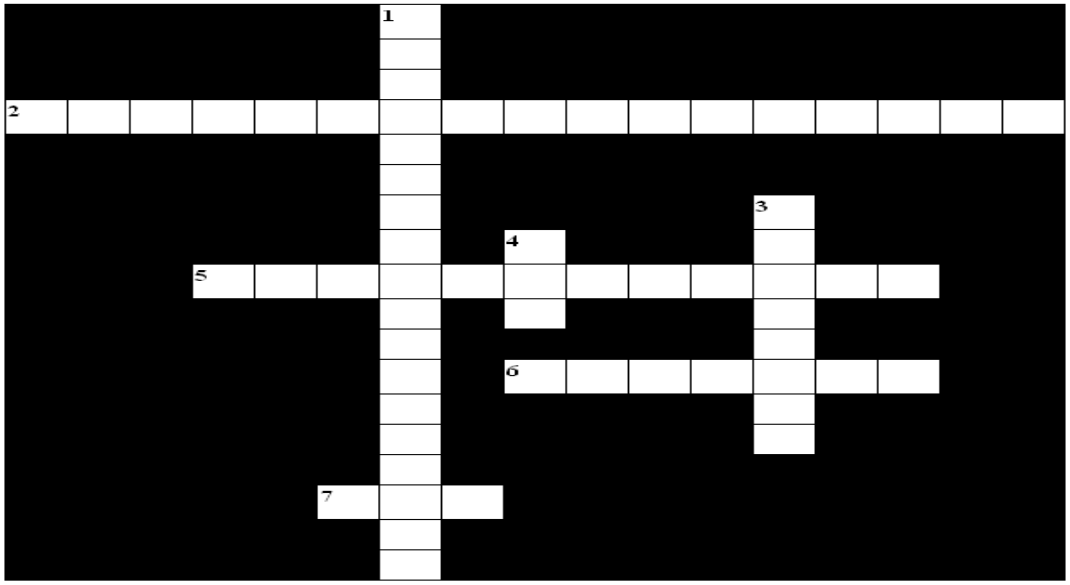
Lessons Learnt :

- .Net framework helps to build web applications not necessary to have prior knowledge of networking protocols.

Best Practices :

- Organize code into hierarchical namespaces, and classes within the namespaces.
- Language independent implementation of classes.
- To provide runtime error checking, security and managed execution.

Crossword: Unit-1 **Estimated Time: 10 mins.**



Across:

2 Function that manages the allocation and release of memory for an application(17)

7 A folder that contains files (3)

5 Library that contains classes and value types that provides the functionality(9)

6 The code that runs directly on the users computer(15)

Down:

4 Functionality that makes easy to design components and applications whose objects interact across languages.(3)

3 Environment that provides the essential components(8)

1 Security mechanisms give rights to users based on their logon credentials(18)

2.0 .Net Framework Components

Topics

- 2.1 Basic Input\Output Operations
- 2.2 Working with Files
- 2.3Structure of C# program
- 2.4 Crossword

Topic: Basic Input/Output Operations

Estimated Time: 30 mins.

Objectives: After the completion of module, participant should know

- Write () and WriteLine () methods
- Read () and ReadLine () methods

Presentation:

Console Class

- Provides access to the standard input, output, and error streams

Write () and WriteLine () Methods

- Displays information on the console screen
- WriteLine () outputs a line feed/carriage return
- Both these methods are overloaded
- Text and Numeric formatting can be used with mentioned methods

Read () and ReadLine () Methods

- Read () :Reads the next character
- ReadLine ():Reads the entire input line

Scenario :

Mr. Thomas is a publisher, preparing journal for “Satyam Computer Services Ltd”. Satyam wants the name of the chairman to get printed on the cover page of this journal.

Demonstration\Code Snippet :

[Note: Numbering represents the sequence in which the program flows]

Step 1: Use following Namespaces to import base class library

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

Step 2: Import System.IO namespace for basic input/output operations

```
using System.IO; //Default namespace
namespace ConsoleApplication1
```

```
class test
{
[1]  static void Main(string[] args){
[2]        string name;
[3]        Console.WriteLine("Enter CEO name:");
[4]        name = Console.ReadLine();
[5]        Console.Write("CEO:", + name);}
}
```

Context :

- To accept and print necessary data from/to the console or input/output files

Practice Session:

- Write a program to accept the name and address from the user and display it

- Write a program to display all the ASCII values (Hint : use ToChar())

Check List :

- Basic Input/Output Operations

Common Errors :

- Syntax Errors

Exception:

- . Net accepts input in the form of a string, so it is essential to convert the string into required data type

Lessons Learnt :

☑ WriteLine () : Outputs a line feed/carriage return

☑ ReadLine (): To read entire line

Best Practices:

- Suggest to use ReadLine () rather than Read ()

Topic: Working with Files

Estimated Time:40 mins

Objectives: After the completion of module, participant should Know

- Different operations performed on the files
- Stream-related classes

Presentation:

Streams

- A Stream is an object used to transfer data. The data can be transferred in one or two directions:
  - ➢ Outside source to program – *Reading* from the stream
  - ➢ Program to some outside source – *Writing* to the stream

Classes used for Reading and Writing operations

- FileStream:-Reading and Writing binary data in a binary file. It can also be used to read from and write to any files
- Enumerations of FileStream Class:

Enumeration	**Values**
FileMode	Append, Create, CreateNew, Open, OpenOrCreate or Truncate
FileAccess	Read, ReadWrite or Write
FileShare	Inheritance, None, Read, ReadWrite, or Write

- StreamReader :-Reading from text files
- StreamWriter :- Writing to text files

Hierarchy of stream-related classes in System.IO Namespace

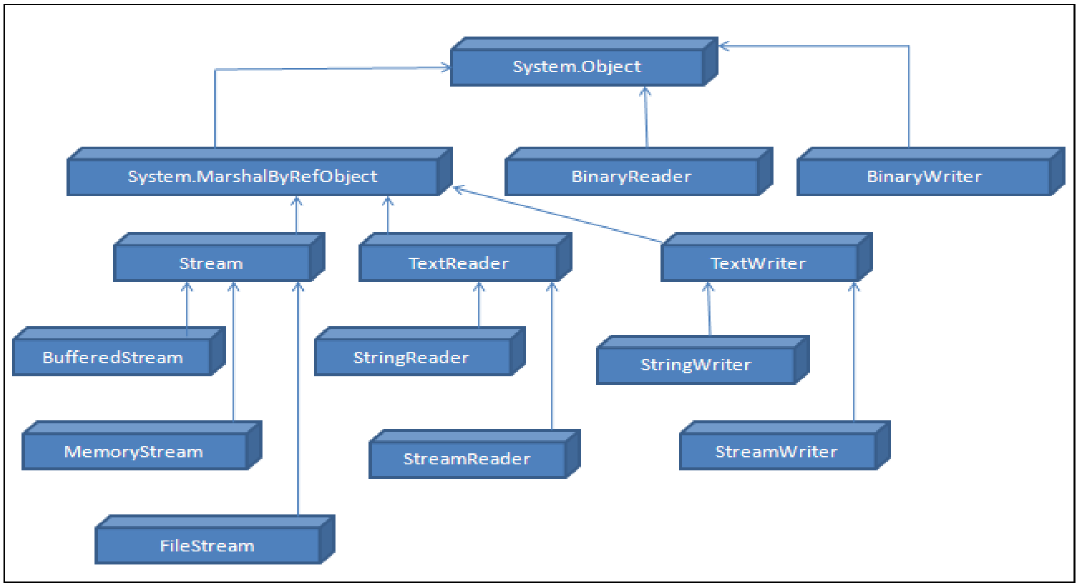


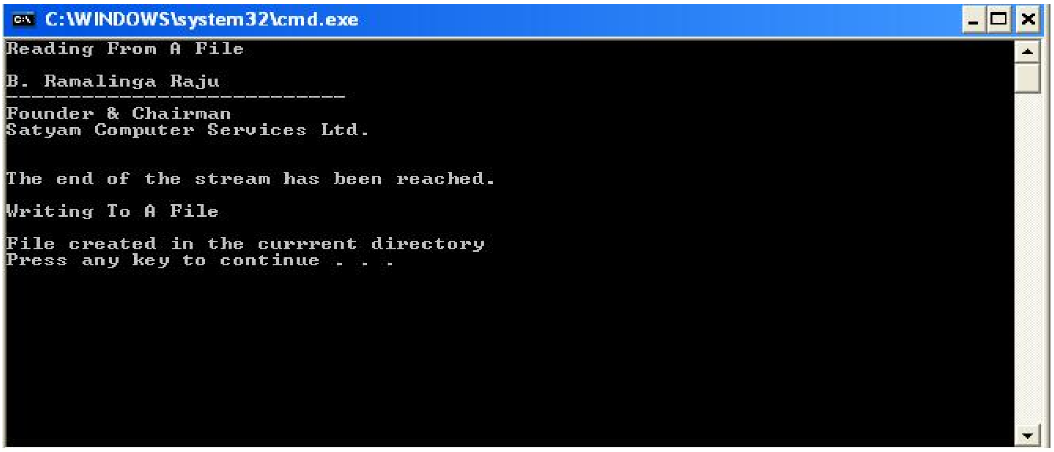
Figure 2.1-1: Stream Related Classes

Scenario:

Mr. Thomas is a publisher, preparing journal for "Satyam Computer Services Ltd". Satyam wants the name and designation of the Chairman stored in a file to get dsiplayed on the cover page of this journal. Name of the Chairman is stored in Profile.txt in the current directory but the designation is missing. Make an application that will read the contents from Profile.txt and append Chairman's designation and store back the formatted output in another file in the current directory.

Demonstration\Code Snippet :

[Note: Numbering represents the sequence in which the program flows]



```
Reading From A File

B. Ramalinga Raju
---------------------------
Founder & Chairman
Satyam Computer Services Ltd.


The end of the stream has been reached.

Writing To A File

File created in the currrent directory
Press any key to continue . . .
```

Figure 2.1-2: Output

Step 1: Use following Namespaces to import base class library

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

Step 2: Import System.IO namespace for basic Input/Output operations and define a class: TextFromToFile to perform read and write operations

```
using System.IO;
namespace ConsoleApplication
{
```

Step 3: In class define StreamReader class to read string from file and StreamWriter class to write string into file

```
    public class TextFromToFile
    {
         //File Profile.txt is stored in current directory
         private const string FILE_NAME = "Profile.txt";
         //File to be written in the current directory
         private const string FILE_OUTPUT="FormattedProfile.txt";
```

```
[1]      public static void Main(String[] args)
         {
            //Instantiating StringBuilder class.
[2]         StringBuilder formatString = new StringBuilder();
[3]         String input;
[4]         String lines = "\n---------------------------\n";
[5]         String designation = "Founder & Chairman";
            //Reading From A File
[6]         Console.WriteLine("Reading From A File\n");

[7]         if (!File.Exists(FILE_NAME))
            {Console.WriteLine("{0} does not exist", FILE_NAME);
             return;}
            //StreamReader reads character from a byte stream in
            //a particular encoding
[8]         using (StreamReader sr = File.OpenText(FILE_NAME))
            {
[9]               input = sr.ReadLine();
```

```
                     //Appends a string represntation of a specified
                     //object to the end of this instance.
[10]                  formatString.Append(input);
[11]                  formatString.Append(lines);
[12]                  formatString.Append(designation+"\n");
[13]                  input = sr.ReadToEnd();
[14]                  formatString.Append(input);
[15]                  Console.WriteLine(formatString);
[16]                  Console.WriteLine("The end of the stream has"+
                     "been reached.\n");
[17]                  sr.Close(); //File Closed
              }
              //Writing To A File
[18]          Console.WriteLine("Writing To A File\n");
[19]          if (!File.Exists(FILE_OUTPUT))
              {
                    //StreamWriter writes characters to a stream in
                    //a particular encoding
[20]                using (StreamWriter sw =
                    File.CreateText(FILE_OUTPUT))
                    {
[21]                      sw.WriteLine(formatString);
[22]                      sw.Close();
                    }
[23]                Console.WriteLine("File created in the currrent
                    directory");
              }
              else
              {   Console.WriteLine("File:-  {0}  with  appropriate
              content  already  exist  in  the"+ "current  directory"
              ,FILE_OUTPUT);
              }
          }
      }
```

Filename declared as constant, remains unchanged throughout the program.

StringBuilder class represents mutable strings of characters.

Check for the existence of file before performing any operation on the file.

The “Using” keyword calls Dispose() method when an object goes out of scope.

Context:

- To perform off-line processing on the data
- To read, process and write the data from/to a file

Practice Session:

- Implement the above scenario with FileStream class
- Identify properties of StreamReader and StreamWriter Class

Check List:

- Reading and Writing from/to a file
- Attributes, properties of the file
- The methods of StreamReader and StreamWriter classes

Common Errors:

- Reading from a non-existing file
- Invalid operations on the file due to the access permissions

Exception:

- Opening a file that is not closed

Lessons Learnt:

☑ FileStream class is used to transfer binary data
☑ StreamReader class is used to read from a file
☑ StreamWriter class is used to write to a file

Best Practices:

- Always close an open file
- Always check whether the file exist or not before opening or creating it

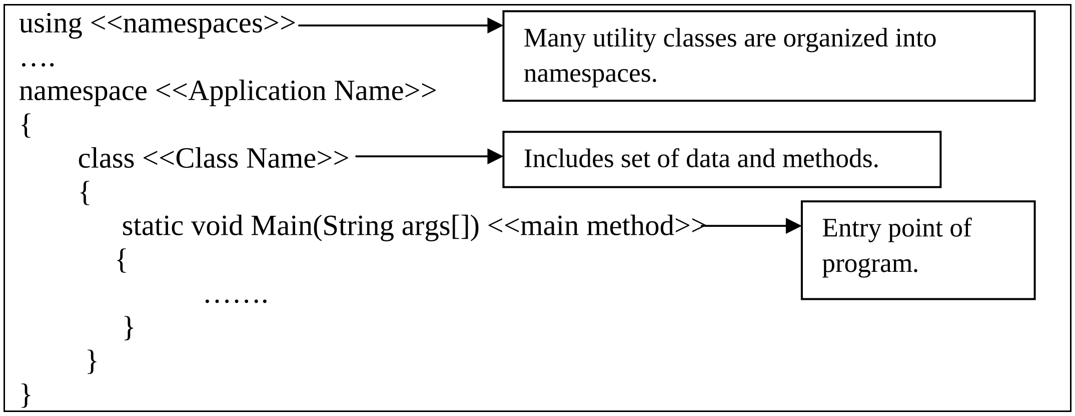
Topic: Structure of C# Program

Estimated Time: 25 mins.

Objectives: After the completion of module, participant should **know**

- To declare and define a Class
- The importance of Main() method and using directives
- The Namespace of System

Presentation:



```
using <<namespaces>>
….
namespace <<Application Name>>
{
        class <<Class Name>>
        {
              static void Main(String args[]) <<main method>>
              {
                        …….
               }
         }
}
```

Many utility classes are organized into namespaces.
Includes set of data and methods.
Entry point of program.

Figure 2.2-1: Structure of C# Program

Scenario:

Mr.Tsang is an interior decorator. He needs to buy a carpet for his client's office for which he is suppose to do the interior work. So he is required to calculate the area of the floor whose length and breadth are 12 units and 30 units respectively.

Demonstration/Code Snippet :

[Note: Numbering represents the sequence in which the program flows]

Step 1: Add variables i and j of type int and calculate method to calculate area

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

class Program
{
        int i,j;
[4]     public int calculate(int length,int breadth)
        {
[5]         return length*breadth;
        }
[1]     static void Main(string[] args)
        {
              //Instantiating object of class
[2]          Program p=new Program();
[3]          int  area= p.calculate(12,30);
             //Tostring() converts int to string
[6]          Console.Write("Area:",+area.ToString());
        }
}
```

Context:

- To encapsulate and abstract objects data and behavior

Practice Session:

- Write a program which includes classes, main method, using directive and system namespaces
- Identify at least three namespaces that goes with using directive

Check List:

- Declaration of Variables, Classes and Methods
- Instantiating an object of the Class
- Invocation of methods

Common Errors:

- Syntax Errors
- Type mismatch

Exception:

- The Main() method doesn’t exists in class libraries

Lessons Learnt :

☑ Implementation of a Class

☑ A Static Main() method is encapsulated as part of the Class

Best Practices:

- Use Namespaces to avoid name collisions of types (classes, structures, enumerations, interfaces).

Crossword: Unit-2 **Estimated Time: 10 mins.**

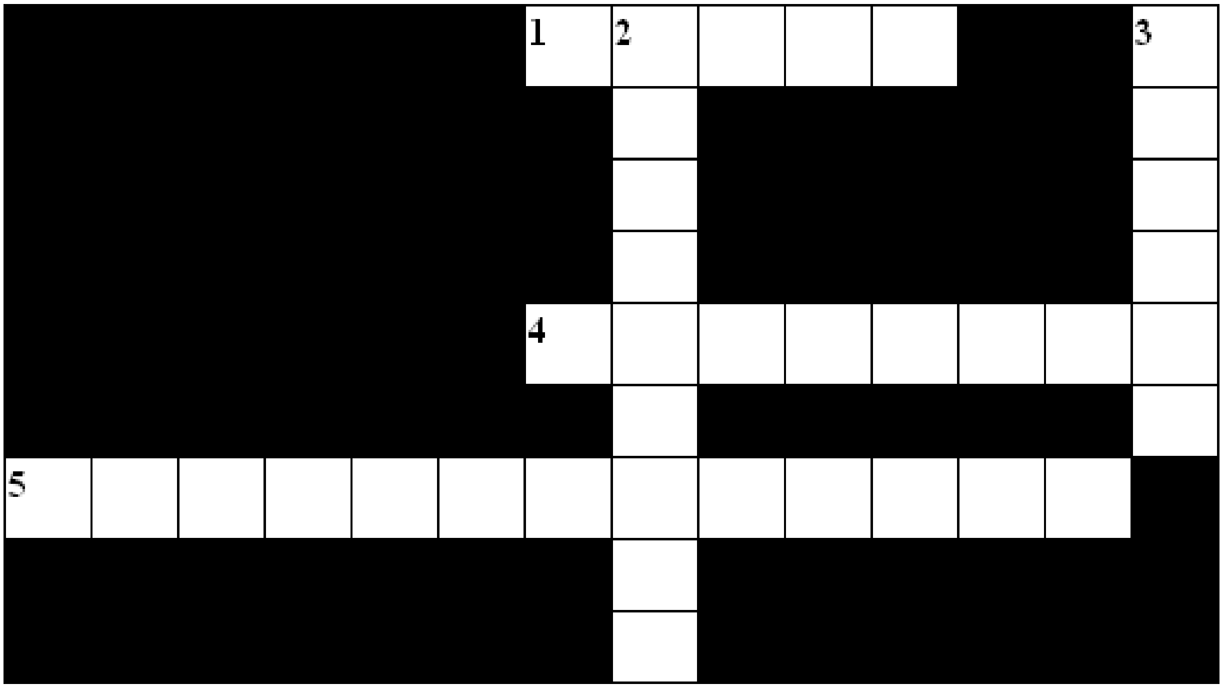


Fig. 2-1

Across:

1. The namespace can be included into the application by the _____ keyword.(5)

4. The method used to read the next line in console application.(8)

5. Method used to display messages on a console screen.(13)

Down:

2. What is the namespace used for importing IO class library.(9)

3. Which is the most commonly used namespace in console applications.(6)

3.0 Using Value Type Variables

Topics

- 3.1 Common Type System
- 3.2 Name Variables
- 3.3 Using Built in Data types
- 3.4 Creating User Defined Data Types
- 3.5 Converting Data Types
- 3.6 Crossword

Topic: Common Type System

Estimated Time: 40 mins.

Objective : After the completion of module, participant should know

- Value and Reference Types
- Built-in and User-Defined Value Types
- Simple Types

Presentation :

- CTS define the rules concerning data types

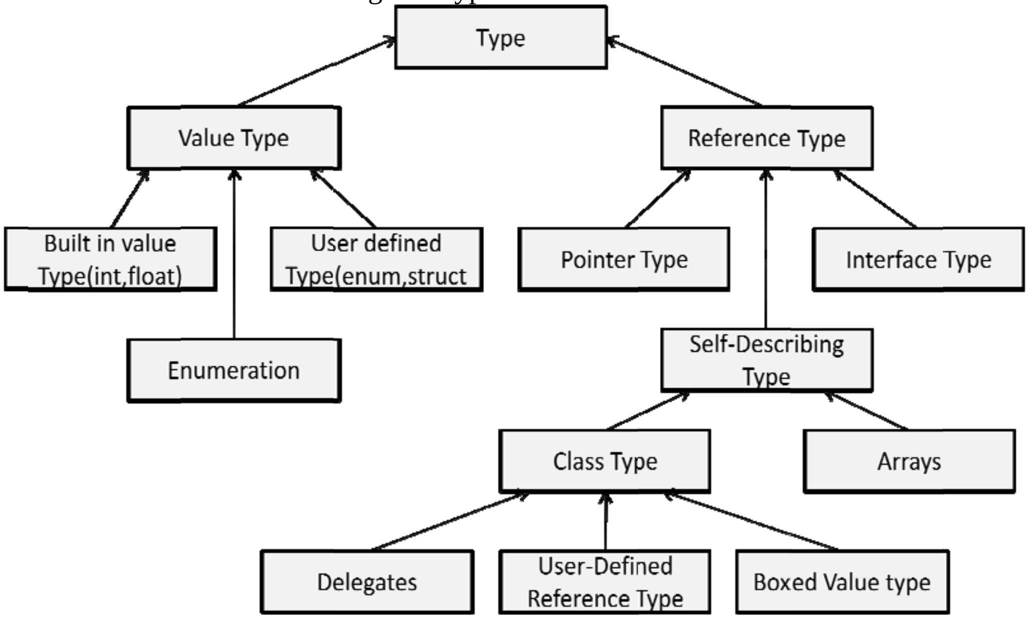


Figure 3.1-1: Hierarchical structure of common type system

- Value type Vs Reference types

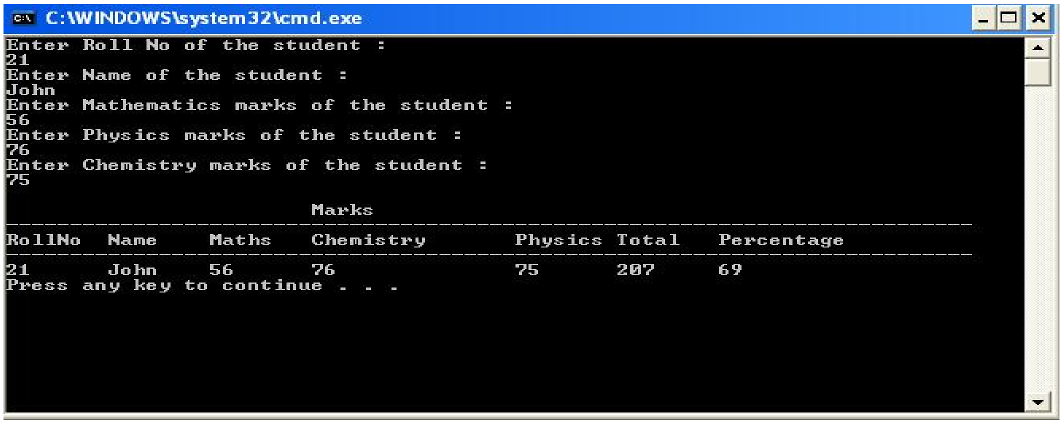
Value Type	**Reference Type**
Holds the value of data	Gets stored as reference
Object has its own copy of data	Two or more references can be set to the same object
Operation on Value Type object doesn't affect another object	Operation on one object can affect another object reference set to it

Scenario :

Mr. Adison is required to maintain records of the student's marks. Create a console application to accept the roll number, name and marks obtained in mathematics, physics and chemistry .Calculate the total marks and their percentage.

Demonstration/Code Snippet :

[Note: Numbering represents the sequence in which the program flows]



```
Enter Roll No of the student :
21
Enter Name of the student :
John
Enter Mathematics marks of the student :
56
Enter Physics marks of the student :
76
Enter Chemistry marks of the student :
75
                        Marks
----------------------------------------------------------------------------
RollNo  Name    Maths   Chemistry       Physics Total   Percentage
----------------------------------------------------------------------------
21      John    56      76              75      207     69
Press any key to continue . . .
```

Figure 3.1-2: Output

```
using System;
```

Step1: Defining a class named Student and varibles and its type

```
class Student
{
      Int32 rollno; //Int32 is a CTS equivalent of int
      string name;
      float marks1;
      float marks2;
      float marks3;
      float total;
      float percent;
```

Step2: Declare display as function and take required information from the user as input

```
[4]    public void Display()
       {
[5]          StringBuilder format = new StringBuilder("------"+
             "-------------------------------------------------");

[6]          Console.WriteLine(“Enter Roll No of the student :”);
             //parse() converts accepted string into an int type
[7]          rollno = Int32.Parse(Console.ReadLine());

[8]          Console.WriteLine(“Enter Name of the student :”);
[9]          name = Console.ReadLine();
```

```
[10]         Console.WriteLine(“Enter Mathematics marks of the
             student :”);
             //parse() converts accepted string into an float type
[11]         marks1 = float.Parse(Console.ReadLine());

[12]         Console.WriteLine(“Enter Physics marks of the
             student :”);
[13]         marks2 = float.Parse(Console.ReadLine());

[14]         Console.WriteLine(“Enter Chemistry marks of the
             student :”);
```

```
            marks3 = float.Parse(Console.ReadLine());

            total = marks1 + marks2 + marks3;
            percent = total / 3;

            StringBuilder student = new StringBuilder("RollNo");

            //Appends the string representation of a specified
            //object to the end of this instance.
            student.Append("\tName\tMaths\tChemistry\t”+
            "Physics\tTotal\tPercentage");

            Console.WriteLine();
            Console.WriteLine("\t\t\tMarks");
            Console.WriteLine(format);
            Console.WriteLine(student);
            Console.WriteLine(format);
            Console.WriteLine(rollno+"\t"+name+"\t"+marks1+"\t"
            +marks2+"\t"+"\t"+marks3+"\t"+total+"\t"+percent );
      }
```

Step3: In the main method create an object of the class and call the function

```
      static void Main(string[] args)
      {
            Student stud = new Student();
            stud.Display();
      }
}
```

CTS primitive data type is syntactically available as classes that support methods.

Example: int32 i;

String s = i.ToString (); //ToString () is a method

Context:

- Use Data Type for the declaration of variables

Practice session:

- Identify other CTS types
- In given scenario add new feature of assigning grades to students according to percentage
  1. If >85% -A grade
  2. If <85% and >70% -B grade
  3. If <70% and >60% -C grade
  4. If <60% and >50% -D grade
  5. Else Fail

Check List :

- Value Type Vs Reference Type
- In-built, CTS and user defined data types

Common Errors:

- Defining value out of range
- Type mismatch

Exception:

- The Nullable type feature for value type enables a variable to hold an undefined value

Lessons Learnt:

☑ Data Types makes the language strong

Best Practices:

- Strongly checking of data type prevents errors and enhances reliability.
- CTS provide the correct scoping, and advantages over the basic Built-in type.

Topic: Naming Variables

Estimated Time:25 mins.

Objectives:

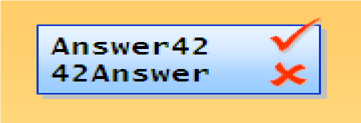
After the completion of module, participant should know

- Rules and Recommendations (also called as Guidelines) for naming variables.
- About C# keywords and their usage.

Presentation:

Rules

- Use letters, the underscore, and digits and avoid using numeric before alphabets

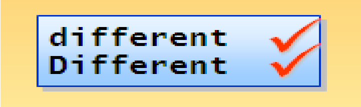


```
Answer42   ✓
42Answer   ✗
```

Figure 3.2-1

Recommendations

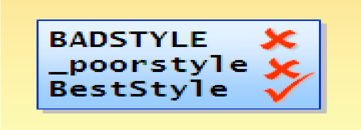
- Avoid using all uppercase letters



```
different  ✓
Different  ✓
```

Figure 3.2-2

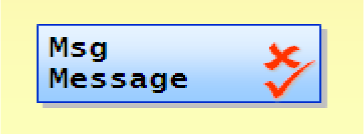
- Avoid starting with an underscore



```
BADSTYLE    ✗
_poorstyle  ✗
BestStyle   ✓
```

Figure 3.2-3

- Avoid using abbreviations



```
Msg      ✗
Message  ✓
```

Figure 3.2-4

- Use Pascal Casing naming in multiple-word names

C# Keyword

- Keywords are reserved identifiers

```
abstract, base, bool, default, if, finally
```

- Not to use keywords by changing their case sensitivity

```
int INT;   // Poor style
```

Scenario :

Mr. Allen is an interior decorator. He needs to buy a carpet for his client's office to do the interior work. So he requires calculating the area of the floors of different length and breadth.

Demonstration/Code Snippet :

[Note: Numbering represents the sequence in which the program flows]

```
using System;
```

Step 1: Define a class named Interior containing Main() method (entry point of execution), declaring length and breadth variable of float data type and display area

```
Class Interior
{
      static void Main(string[] args)
      {
            float length, breadth, area;
            Console.WriteLine("Enter Length. ::");
            length = float.Parse (Console.ReadLine());
            Console.WriteLine("Enter Breadth. ::");
            breadth = float.Parse (Console.ReadLine());
            area = length * breadth;
            Console.WriteLine("area is {0}:",area);
      }
}
```

Variable can also declare as constant using const keyword.Its value is assigned during declaration and the declared value can’t change.

Variable can be also declared as read_only using readonly keyword. Its value can assigned during runtime using constructor and assigned value can’t change.

Context:

- Proper naming convention provides an opportunity to debug and maintain the code

Practice Session:

- Spot the disallowed variable names?
  A. int 12count;
  B. char $diskPrice;
  C. char middleInitial;
  D. float this;
  E. int __identifier;

Checklist:

- Variable naming conventions, and appropriate naming prefixes.
- Ease of maintenance of code using naming conventions.

Common Errors:

- Naming variables same as Keywords.
- Syntax Errors are raised when naming conventions are not followed.
- An initialized variable before it is used.

Exception:

- If a pointer to int, p is declared and assigned the address of an integer variable, number. The variable number is initialized as a result of the assignment to *p.

Commenting this assignment statement will remove the initialization of the variable number, but no compile-time error is issued.

```
Ex:   int number
  // Assign the address of number to a pointer:
  int* p = &number;
   // commenting the following statement will remove the initialization of number
  *p = 0xffff;
```

Lessons Learnt :

- Variable can also be defined dynamically.
- Variable scope.

Best Practices:

- Create an instance of a class, before referenced

Topic: Built-In Data Types

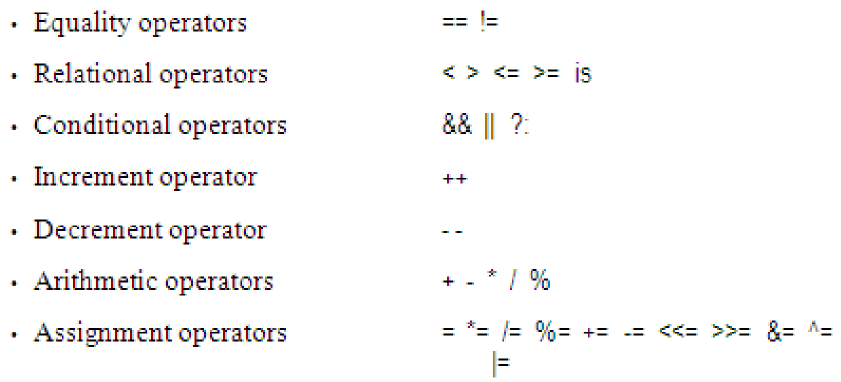
Estimated Time: 35 mins.

Objective: After the completion of module, participant should know

- To declare and assign values to Variables
- Compound Assignment & Common Operators
- Operator Precedence
- Declaration of enum ,struct and user defined data type
- Implicit and Explicit conversion of data type

Presentation:

- Declaring Local Variables: int itemcount;
- Assigning Values to Variables: int itemcount=23;
- Compound Assignment
  - ➢ Adding a value to a variable: itemcount = itemcount + 40;
  - ➢ Convenient shorthand: itemcount += 40;
- Common Operators



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

- Increment and Decrement
  - ➢ Changing a value by one: itemcount += 1; itemcount -= 1;
  - ➢ Convenient shorthand: itemcount++; itemcount--;
- Operator Precedence
  - ✓ All binary operators are left-associative
  - ✓ Assignment operators and conditional operators are right-associative
- Numerical conversion

```
int a=123;
long b=a;      //Implicit conversion from int to long
int c= (int) b;   //Explicit conversion from long to int
```

Scenario :

Mr. David wants to maintain the records for percentage of marks obtained by the student in coaching institute. He requires the information of student (Student name and roll no, marks scored in physics, chemistry and mathematics) and calculate the total marks and percentage of the student with appropriate user defined data types.

Demonstration\Code Snippet :

[Note: Numbering represents the sequence in which the program flows]

```
using System;
```

Step 1: Declare struct type

```
[9]   public struct FullName
    {
[10]  public string FName;
[11]  public string LName;
[12]  public string MName;
    }
```

Step 2: Declare enum datatype

```
[41] enum grade{Distinction,FirstClass,SecondClass,Pass,Fail }
```

Step 3: Declare user defined datatype

```
      class Mark
      {
            private float _marks1, _marks2, _marks3;
[31]        public void SetData(float m1,float m2,float m3)
            {
[32]              _marks1 = m1;
[33]              _marks2 = m2;
[34]              _marks3 = m3;
            }
      }
```

Step 4: Define the main () method to determine the total marks, percentage

```
class Student
{
[1]   static void Main(string[] args)
      {
[2]         Int32 rollno;           //CTS type
[3]         string Name;            //In built datatype
[4]         String Address;         //String class
[5]         float marks1, marks2, marks3, total, percentage;
[6]         Console.WriteLine("Enter Roll No of the Student:");
[7]         rollno = Int32.Parse(Console.ReadLine());
            //Initialize struct object
[8]         FullName n;
[13]        Console.WriteLine("Enter FirstName of the Student:");
[14]        n.FName = Console.ReadLine();
[15]        Console.WriteLine("Enter MiddleName of Student:");
[16]        n.MName = Console.ReadLine();
[17]        Console.WriteLine("Enter LastName of the Student:");
[18]        n.LName = Console.ReadLine();
            //Use of String class and its method.
[19]        Console.WriteLine("Enter Address of the Student:");
[20]        Address = Console.ReadLine();
            if (Address.Length > 30)
                  Console.WriteLine("Too Lengthy…Not Allowed");
[21]        else
                    Console.WriteLine("Allowed");
            //Instantiating object
[23]        Mark m = new Mark();
[24]        Console.WriteLine("Enter Mathematics marks:");
[25]        marks1 = float.Parse(Console.ReadLine());
[26]        Console.WriteLine("Enter Physics marks:");
[27]        marks2 = float.Parse(Console.ReadLine());
[28]        Console.WriteLine("Enter Chemistry marks:");
[29]        marks3 = float.Parse(Console.ReadLine());
[30]        m.SetData(marks1, marks2, marks3);
            //Arithmetic Operators
[35]        total = marks1 + marks2 + marks3;
[36]        Console.WriteLine("Total marks" + total);
            //Explicit conversion
[37]          percentage = (float)(total*100) / 300;
[38]          Console.WriteLine("Percentage is" + percentage +"%");
            //Calling enum datatype.
[39]          if (percentage >= 70)
            {
            //GetName is method to retrieve value of particular
            //enumarted constant.
[40]                Console.WriteLine("Grade is:{0}",
                  Enum.GetName(typeof(grade), 1));
            }
            else if (percentage < 70 && percentage >= 60)
            {
                  Console.WriteLine("Grade is:{0}",
                  Enum.GetName(typeof(grade), 2));
            }
            else if (percentage < 60 && percentage >= 50)
            {
                  Console.WriteLine("Grade is:{0}",
                  Enum.GetName(typeof(grade), 3));   }
            else if (percentage < 50 && percentage >= 40)
            {
                  Console.WriteLine("Grade is:{0}",
                  Enum.GetName(typeof(grade), 4));
            }
            else if (percentage < 40)
            {
                  Console.WriteLine("Grade is:{0}",
                  Enum.GetName(typeof(grade), 5));
            }
      }
}
```

Enumeration DataType is used to create a set of symbolic names that map to known numerical values (specifically integer format).

Structure data_type contains number of data fields and methods that operate on these fields.

Struct can contain constructors, interface, properties, method, events, overloaded operators.

Structure can-not contain Destructors.

Structures are the best examples for Value Type under .Net Framework.

Context :

- To build abstract or user defined types
- For efficient storage management
- To handle type compatibility issues

Practice session :

- Write a program to print all even numbers from 1 to 50
- Write a program to calculate the area of triangle whose dimensions are as follows: Breadth=17, Height=12 (Hint: area of triangle=1\2*b*h)

Check List :

- Declare and assign values to variables
- User defined data type
- Use of various operators

Common Errors :

- Issues with respect to type compatibilities and casting
- Unable to distinguish between the CTS type String and Built-in type string.

Exceptions:

- An integer overflow may occur during calculations. Sometimes the application needs to be continued while discarding significant bits involved in the calculations, however due to overflow exception, the application halts as it is not handled appropriately

Lessons Learnt:

☑ Assignment operators and Conditional operators are right-associative

Best Practices:

- Declare variables before they are used.
- Better use parenthesis, if unsure of the order of precedence of the operators.

Crossword: Unit-3 **Estimated Time: 10 mins.**

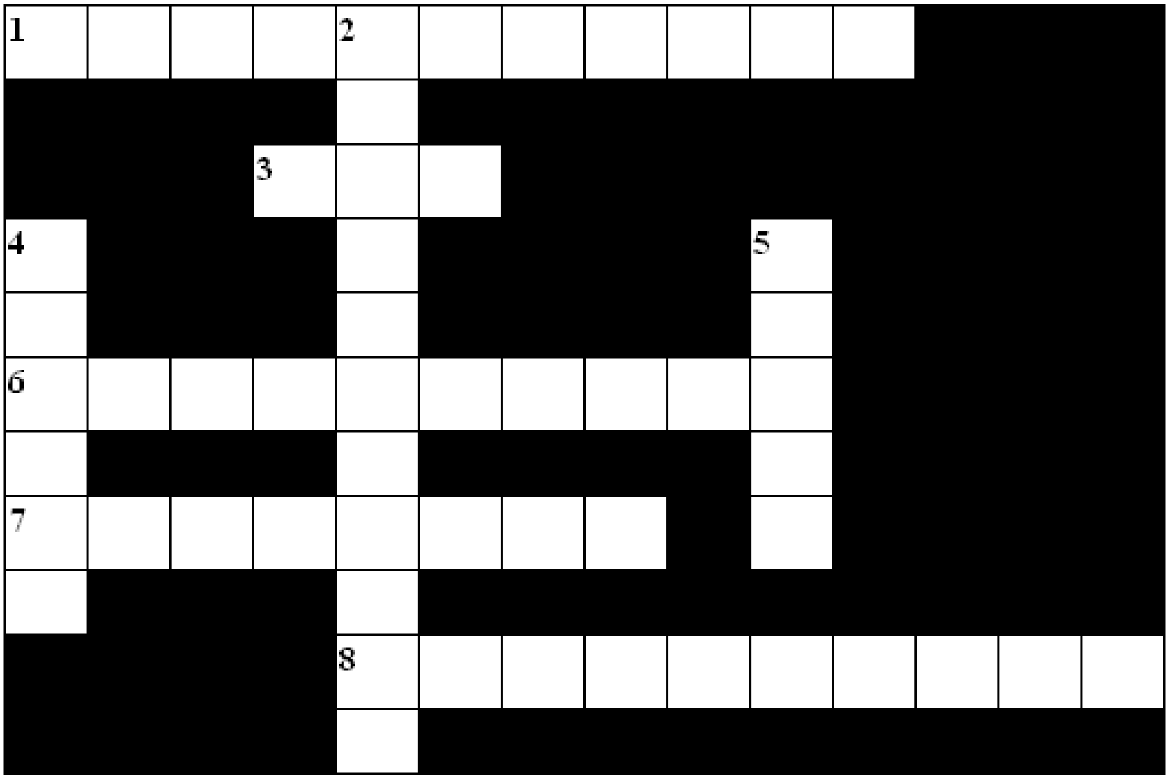


Fig.3-1

Across:

1. rollno =Int32.Parse(Console.ReadLine());
What operation is performed on the data types in the example above?(11)

3. What is the example of built in value type.(3)

6. >= <= are example of ______ operators.(10)

Down:

2. &&, || are examples of _______ operators.(11)

4. Example of user defined value type.(6)

5. Built in type is an example of

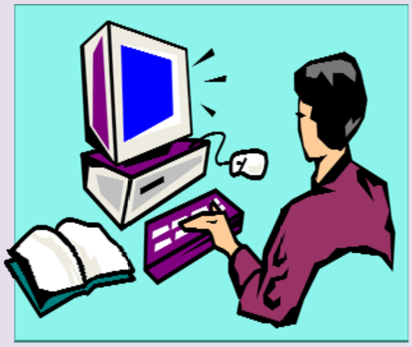
7. itemCount += 40; is a type of ______ assignment.(8) ______ type.(5)

8. All binary operators are left associative, except for ______ operators.(10)

4.0 Statements and Exceptions

Topics

- 4.1 Introduction to Statements
- 4.2 Exception Handling
- 4.3 Crossword



Topic: Statements

Estimated Time: 45 mins.

Objectives: After the completion of module , participant should know

- Types of statements and their implementation

Presentation:

- Statements are program instructions executed in sequence
- C# statement ends with terminator **‘;’**
- C# has the following categories of statements

Category	C# keywords
Selection Statement	if, else, switch, case
Iteration Statement	do-while, for, foreach, while
Jump Statement	break, continue, default, goto, return, yield
Exception handling Statement	throw, try-catch, try-finally, try-catch-finally

Selection Statement

- It transfers the control of a program to a specific flow based upon whether a certain condition is true or not.

Iteration Statement

- Iteration statements execute the embedded statements number of times, subject to the loop-termination criteria.
- Iteration statements are executed in order, except when a jump statement is encountered.

Jump Statement

- Jump Statement performs branching without condition, which causes immediate transfer of the control of program

Scenario:

A XMart Shopping Mall is offering a special end of season offer for its customers. In order to make this offer functional, the management has decided to add a separate application, which accepts from Billing Executive, the Customer Name,Total Amount of Purchase and provides an option for customer to select among three choices as given below:-

(1.1) 12% discount for a total purchase greater than 2000 and less than 3000
(1.2) 15% discount for a total purchase greater than or equal to 3000 and less than 4000
(1.3) 20% discount for a total purchase greater than or equal to 4000
(2) Free Gifts
(3) Member ship Card

Demonstration\Code Snippet :

[Note: Numbering represents the sequence in which the program flows]

```
using System;
```

Step 1: Define a class,declare its members and accept required data from the user

```
class Discount_offer
{
[1]   static void Main(string[] args)
      {
           String custname;
           float amntpurchase;
           float amtpaid;
           float discountoffer;
           int choice;
[2]        Console.WriteLine("Enter the Customer Name and Amount
           of Purchase made");
[3]        custname=Console.ReadLine();
[4]        amntpurchase =Convert.ToString(Console.ReadLine());
```

Step 2: Apply the If condition

```
[5]         if (amntpurchase >= 2000 && amntpurchase < 3000)
            {
[6]               Console.WriteLine("(1) 12% discount.");
[7]               Console.WriteLine("(2) Free Gifts upto 20% of
                  Total purchase.");
[8]               Console.WriteLine("(3) Member ship Card.");
[9]               Console.WriteLine("Enter your choice");
```

Step 3: Different Switch cases

```
[10]              choice=Convert.ToInt32(Console.ReadLine());
[11]              switch (choice)
                  {
[12]                    case 1:
                              discountoffer = Convert.ToSingle
                              (amntpurchase * (0.12));
[13]                          amtpaid=amntpurchase-discountoffer;
[14]                          Console.WriteLine("Amount to be
                              paid after discount is $ " +
                              amtpaid);
[15]                          break;
                        case 2:
                              Console.WriteLine("Collect ur free
                              Gift From the Counter");
                              break;
                        case 3:
                              Console.WriteLine("Collect your
                              Membership card ");
                              break;

                        default:
                              Console.WriteLine("Invalid Choice")
                              break;
                        }
                  }
```

Step 4: Applying if else Condition

```
            else if (amntpurchase >= 3000 && amntpurchase < 4000)
            {
                  Console.WriteLine("(1) 15% discount.");
                  Console.WriteLine("(2)Free Gifts upto 20% of
                  Total purchase.");
                  Console.WriteLine("(3) Member ship Card.");
                  Console.WriteLine("Enter your choice");
                  choice= Convert.ToInt32(Console.ReadLine());
                  switch (choice)
                  {
                        case 1:
                              discountoffer = Convert.ToSingle
                              (amntpurchase *(0.15));
                              amtpaid=amntpurchase-discountoffer;
                              Console.WriteLine("Amount to be
                              paid after discount is" + amtpaid);
                              break;
                        case 2:
                              Console.WriteLine("Collect ur free
                              Gift From the Counter");
                              break;
                        case 3:
                              Console.WriteLine("Collect your
                              Member ship card ");
                              break;
                        default:
                              Console.WriteLine("Invalid
                              Choice");
                              break;
                  }
            }
            else if (Amnt_Purchase >= 4000)
            {
                  Console.WriteLine("(1) 20% discount.");
                  Console.WriteLine("(2) Free Gifts upto 20% of
                  Total purchase.");
                  Console.WriteLine("(3) Member ship Card.");
                  Console.WriteLine("Enter your choice");
                  choice = Convert.ToInt32(Console.ReadLine());
                  switch (choice)
                  {
                        case 1:
                              Discount_offer = Convert.ToSingle
                              (Amnt_Purchase * (0.20));
                              amtpaid=amntpurchase-discountoffer;
                              Console.WriteLine("Amount to be
                              paid after discount is" + amtpaid);
                              break;
                        case 2:
                              Console.WriteLine("Collect ur free
                              Gift From the Counter");
                              break;
                        case 3:
                              Console.WriteLine("Collect your
                              Member ship card ");
                              break;
                        default:
                              Console.WriteLine("Invalid Choice")
                              break;
                  }
            }
      } //end of Main() method
};//end of class
```

Context:

- For handling multiple conditions and different cases

Practice Session:

- Create a console application for providing grades to the students depending on overall performance and marks in examination
  - Grade A+ for marks>=90
  - Grade A for 80<=Marks<90
  - Grade B for 70<=marks<80
  - Grade C for 60<=Marks<70
  - Grade D for <60

Check List :

- Categories of Statements and their Classification.

Common Errors :

- Conditions in ‘if’ not mentioned.
- Using Single ‘=’ in condition.
- ‘break’ statement not written at the end of each case of switch statement.
- ‘default’ case not mentioned in switch statement.

- Errors in Boolean expressions.

Exception :

- Explicitly need to convert ‘int’ to ‘bool’ type
  For example:
  In ‘C’ language int a=1; if (a)   // statement is valid
  In ‘C#’, if (a==1) //statement has to be explicitly stated

Lessons Learnt :

- Use of different statements for ease of implementation.
- To implement different looping constructs.
- GOTO statement helps to jump to any line of code within a program.

Best Practices:

- To use switch in place of if-else.
- Don’t use Jump Statements.

Topic: Basic Exceptions

Estimated Time: 30 mins.

Objectives: After the completion of module, participant should know

- Processing of handled and unhandled exceptions
- Implementations of try, catch, throw and finally blocks

Presentation:

Exception Handling

- It is an built in mechanism to detect and handle runtime errors
- It uses the try, catch**,** throw and finally keywords
- Exceptions can be generated by
  - Common Language Runtime (CLR)
  - The .NET Framework
  - Any third-party libraries
  - Application code

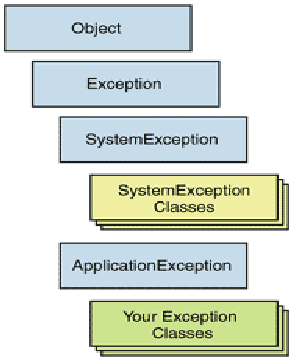


Figure 4.4&4.5-1: Exception Handling Hierarchy

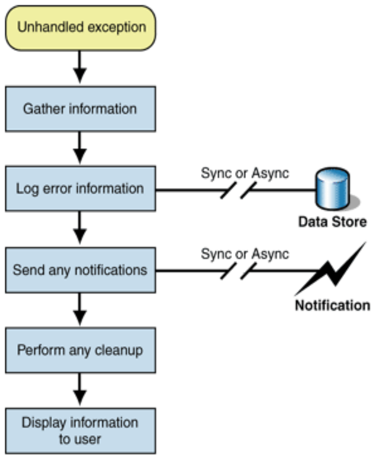


Figure 4.4&4.5-3: Processing of Unhandled Exceptions

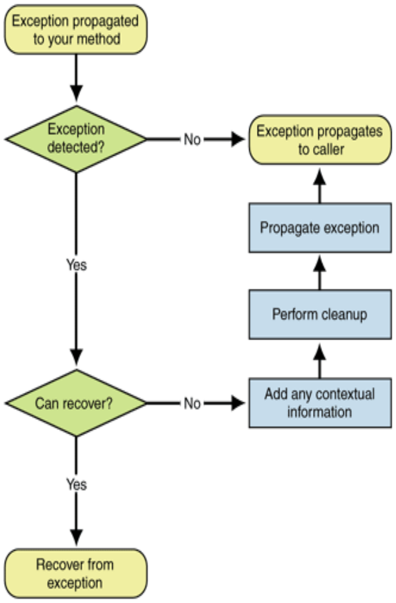


Figure 4.4&4.5-2: Exception Handling Processes

Scenario :

GK Constructions Private Ltd constructed a 5-storey building consisting a lift. The lift can accommodate only 5 persons at a time. The company wants to keep a track of maximum persons accommodated in the lift with the help of software.

.

Demonstration\Code Snippet :

[Note: Numbering represents the sequence in which the program flows]

```
using System;
```

Step 1 : Declare a class and define its members

```
 class Program
{
[1]    static void Main(string[] args)
       {
             int[] array=new int[5];//maximum lift size defined
```

Step2: Implementing Try ,Catch and Finally blocks

```
[2]          try
             {
[3]                Console.WriteLine("Persons in the Lift ");
[4]                for (int i = 0; i < 10; i++)
                   {
[5]                      array[i] = i;
[6]                      Console.WriteLine("Person[{0}]:{1}", i,
                         array[i]);
```

```
                        }
                  }
[7]               catch (IndexOutOfRangeException ex) {
[8]                     Console.WriteLine("Warning..!!! More than 5
                        persons "+ex.Message); }
[9]               finally{
[10]                    Console.WriteLine("Only 5 persons can be
                        accomodated at a time"); }
[11]              Console.ReadLine();
            }
      }
```

Always order, exceptions in catch blocks from the most specific to the least specific. This technique handles the specific exception before it is passed to a more general catch block.

Context:

- Handling expected or unexpected situation.
- Error free execution.

Practice Session:

- Can multiple catch blocks be executed for a single try statement?
- List the advantages of using multiple catch blocks.
- Identify other basic exceptions and implement programs with FileNotFoundException, DivideByZeroException, OverflowException etc

Check List:

- Exception Handling Technique
- Try, Catch and Finally Blocks
- Multiple Catch Blocks

Common Errors:

- ‘try’ block must be implemented in exception handling
- There should be only one try block above the mentioned multiple catch blocks.

Exception:

- If finally block is not written after exception handlers then it is a possibility that lines of code (resource cleanup, closing a file) you expect to be called are not executed.

Lessons Learnt :

☑ To catch the raised exceptions.

☑ Finally block gets executed definitely, and at the end, irrespective of an exception being caught or not.

Best Practices :

- In most cases, use the predefined exception types.
- Include a localized description string in every exception.
- Write the code for clean up resources in ‘finally’ block.

Crossword: Unit-4 **Estimated Time: 10 mins.**

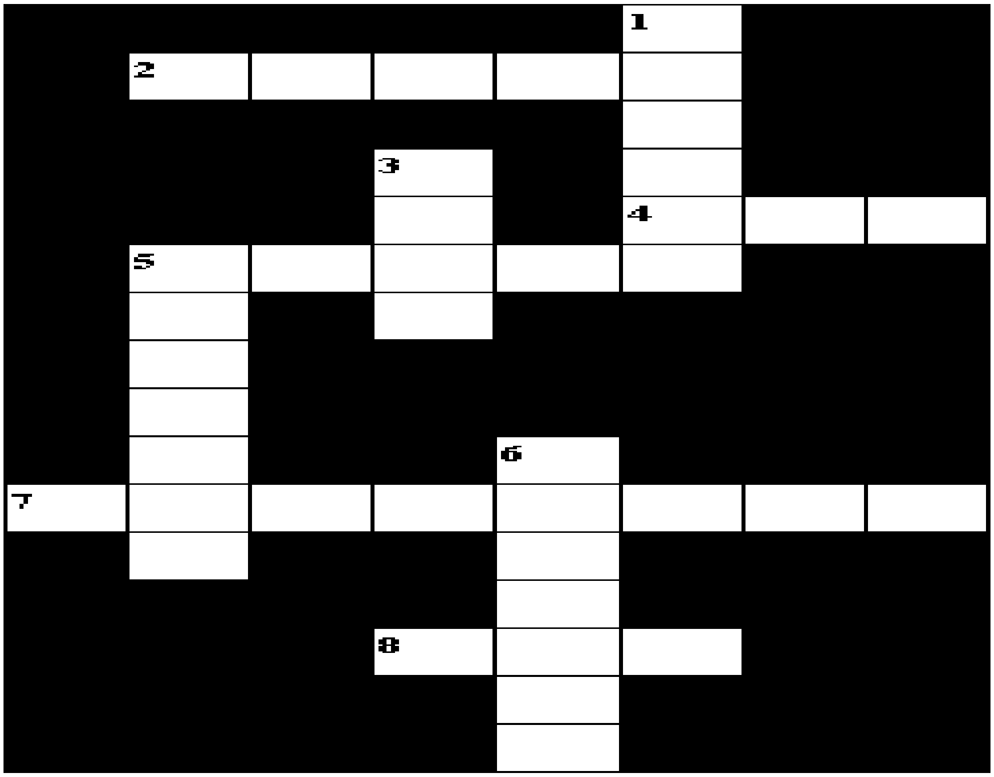


Fig. 4-1

Across:

2. Exceptions are thrown using ______ keyword.(5)
4. Exceptions can be generated by __________ .(3)
5. ___________ keyword handles failures when it is reasonable to do so.(5)
7. The _________ statement passes control to the next iteration of the enclosing iteration statement in which it appears.(8)
8. All exception classes should derive from the Exception base class within the System namespace for compliance with _____________.(3)

Down:

1. The __________ statement is a control statement that handles multiple selections and enumerations by passing control to one of the case statements within its body otherwise it will go to the default one.(6)
3. The _____ statement transfers the program control directly to a label.(4)
5. The switch statement is a ________ statement that handles multiple selections and enumerations by passing control to one of the case statements within its body otherwise it will go to the default one.(7)
6. The ________ keyword encloses the block of code that is executed irrespective of whether the try or catch blocks are executed or not.(7)

5.0 Methods and Parameters

Topics

- 5.1 Using Methods
- 5.2 Using Parameters
- 5.3 Overloading Methods
- 5.4 Crossword

Topic: Methods

Estimated Time: 25 mins.

Objectives : After the completion of module, participant should know

- To identify the signature of the methods
- Invocation of the methods

Presentation :

Methods

- Methods are place holders to write actual business logic or functionality
- Represents the behavior/functionality of the object
- Provides with Syntactical Representation
  Return-type <<*method-name*>> (parameters-list) {//code}
- Example: - *string WhatIsYourName() { …} h*ere string is the return-type and *WhatIsYourName* () is the method name and at last this method does not contain any parameters

Scenario :

Spokesperson of McKinsey Inc did a wonderful job. The company needs to highlight his profile and display his name.

Demonstration/Code Snippet :

[Note: Numbering represents the sequence in which the program flows]

```
using System;
```

Step 1: Consider following members of the Class McKinsey

- _m_No_Years
- _m_Name

```
public class McKinsey
{
      private int _m_No_Years=30;
      private string _m_Name=”Alex”;
}
```

Step 2: Create following methods in the Class:

- Name_Person() – Returns the name of the person to the console/screen
- Service_Years() – Returns the age of the person to the console/screen
  - ➢ These methods state the behavior of the objects _m_Name and _m_No_Years
  - ➢ Return-type is string and int. It could be any generic C# type

```
[5] public string Name_Person()
    {
[6]   Console.WriteLine("Name of the Spokesperson : "+ _m_Name);
[7]   return _m_Name;
    }
[9] public int Service_Years()
    {
```

```
[10]  Console.WriteLine("No. of Years Serviced :" + _m_No_Years);
[11]  return _m_No_Years;
    }
```

Step 3: Calling the methods

- An Object need to be created in order to call the methods
- Here *mc_obj* object is created to call the methods

```
class Profile_Man
{
[1]   static void Main(string[] args)
      {
[2]         try
            {
[3]               McKinsey mc_obj=new McKinsey();
[4]               mc_obj.Name_Person();
[8]               mc_obj.Service_Years();
            }
            catch(Exception ex)
            {
                  Console.WriteLine(ex.Message);
            }
      }
}
```

Context :

- To define business logic as a unit of code which can be reused.

Practice Session :

- Is it mandatory to specify the return-type for a method? True or False

Check List:

- Method call parameters should match the method declared parameters.
- To invoke a method from outside the class in which it is declared, an instance of the class must be defined.

Common Errors :

- Return-Type of the method not matching with the type it is assigned at the time of invocation

Exception :

- From the above demonstration let us consider the following situation
  `private int _m_No_Years=30;`

- Now define the method body

```
public String Service_Years() //Error-Return type should
be ‘int’
{
    Console.WriteLine(" : " + _m_No_Years);
     return _m_No_Years;
}
```

- The following exception will be raised
  - Cannot implicitly convert ‘int’ type to ‘string’

Lessons Learnt:

☑ A method declared as public can be used in more than one class.

Best Practices:

- Use Methods to demonstrate a particular situation and long lines of codes need to be encapsulated in a particular unit.
- Limit a method, specific to its functionality.

Topic: Parameters

Estimated Time: 40 mins.

Objectives:

After the completion of module, participant should know

- Implementation of Value and Reference parameters

Presentation:

- Parameters are passed to be used along with the methods. There are four different types of parameters in C#:

Types	Significance	Example

Value	• By default, parameters are value parameters
	• A copy of the object is passed instead of the object itself
	• Thus, value is passed but not the same object
	• A new storage location is created for the variable
		void Foo (int x)
		{ x=10; }
		...
		int y = new int ();
		y=5;
		Foo (y);
		Console.WriteLine (y);
		Output: 5

Reference	• *ref* modifier can be used with reference parameters. Here, a new storage location is not created like value parameters
	• Same storage location is used, so the value of the variable in the member function and the value of the reference parameter will always be the same
		void Foo (ref int x)
		{ x=10; }
		int y = new int ();
		y=5;
		Foo (ref y);
		Console.WriteLine (y);
		Output: 10

Output	• Output parameters need the *out* modifier as part of both the declaration and invocation
	• Output Parameters don't create a new storage location, however uses the storage location of the variable specified on the invocation
		void Foo (out int x){
		// Can't read x here - it's considered unassigned
		// Assignment - this must happen before the method can complete
		x = 10;
		// The value of x can now be read
		int a = x;}
		// Declare a variable but don't assign a value to it
		int y;
		// Pass it in as an output parameter, even though its value is unassigned
		Foo (out y);
		// It's now assigned a value, so we can write it out:
		Console.WriteLine (y);
		Output: 10

Parameter Arrays	• It uses *params* modifier.
	• Parameter arrays allow a variable number of arguments to be passed into a function member
		void ShowNumbers (params int[] numbers){
		foreach (int x in numbers){
		Console.Write (x+" ");} }
		...
		int[] x = {1, 2, 3};
		// The variable x is passed by value, as it's just an array~~.~~
		ShowNumbers (x);
		// A new array of int is created containing the two values specified,

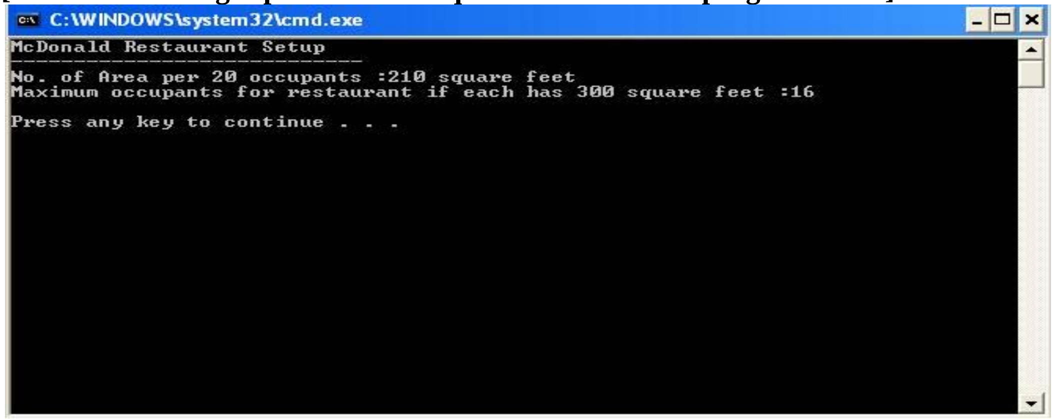
		and a reference to this array is passed. ShowNumbers (4, 5); Output: 1 2 3 4 5

Scenario:

McDonalds setup a branch office at Hyderabad. Company needs some excellent site to build a restaurant at a stylish place. Chairperson invites the members of McDonalds to discuss upon maximum occupant space in a specific area. He orders his engineers to come up with a formidable solution that could be implemented**.**

Demonstration:

[Note: Numbering represents the sequence in which the program flows]



```
McDonald Restaurant Setup
-----------------------------
No. of Area per 20 occupants :210 square feet
Maximum occupants for restaurant if each has 300 square feet :16

Press any key to continue . . .
```

Figure 5.2-1: Output

```
using System;
```

Step 1: Create a class McDonald. Define two methods AreaPerPerson () and other MaxOccupant ()

```
class McDonald
{
      public float floors;
      public float area;
      //Pass by value
[6]   public float AreaPerPerson(int occupants) {
[7]               return area/occupants; }
      //Pass by reference
[12]  public float maxOccupant(ref int minArea){
[13]        minArea = 250;
[14]        return area / minArea;}
 }
```

Step 2: Define the object of the class McDonald and call appropriate methods

```
class Program
{
```

```
[1]     static void Main(string[] args)
        {
              float areapp ,minrefArea=300;
              try
              {
[2]                 McDonald restaurant = new McDonald();
                    //assign value to fields in restaurant
[3]                 restaurant.area = 4200;
[4]                 restaurant.floors = 3;
                    //AreaPerPerson called
[5]                 areapp = restaurant.AreaPerPerson(20);
[8]                 Console.WriteLine("McDonald Restaurant Setup");
[9]                 Console.WriteLine("-------------------------");
[10]                Console.WriteLine("No. of Area per 20 occupants
                    :"+areapp+ " square feet");
                    //Pass  By  Reference  ref  modifier  used  to
                    //calculate the max.occupants
[11]                Console.WriteLine("Maximum occupants for
                    restaurant if each has " + minrefArea + "
                    square feet :" + restaurant.maxOccupant(ref
                    minrefArea));
                    Console.WriteLine();
              }
              catch(Exception ex)
              { Console.WriteLine(ex.Message); }
        }
}
```

Context:

- Parameters can be used to pass /exchange data between methods. Pass by reference can be used to obtain more than one value from a method.

Practice Session:

- McDonalds setup a branch office at Hyderabad. Chairperson invites the members of McDonalds to discuss that he wants a printed bill to handover the customer which contains name and the price of beverage ordered and also contain the date inclusive of taxes .Perform this task by using pass by value and pass by reference.

Check List :

- Pass By Value
- Pass By Reference

Common Errors :

- The number of arguments declared doesn’t match with method definition.
- Mismatch with respect to the data types.
- References to the constants cannot be set.

Exceptions :

- Arrays by default are passed as references.

Lessons Learnt:

☑ ‘*ref*’ modifier should be prefixed at the time of definition and invocation.

Best Practices:

- Pass parameters as reference rather than value.
- Provide default parameters wherever necessary.
- Pass parameters whenever they are more likely subjected for a change

Topic: Overloaded Methods

Estimated Time: 35 mins.

Objective:

After the completion of module, participant should Know

- Implementation of overloaded methods

Presentation:

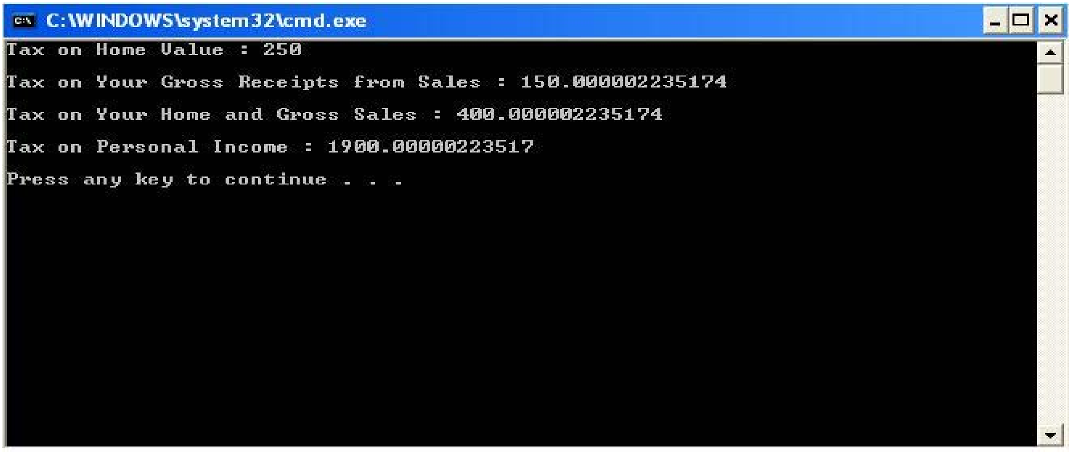
- Simplifies code reusability and clarity
- Two or more methods are said to be overloaded, if they have the same name, but differ in either the number of parameters or the type of parameters.
- Overloading is not defined on the return type.
- To demonstrate the difference in the number of parameters
    1) public static double TaxCalc (double pamt1, double prate1, double pamt2, double prate2)
    2) public static double TaxCalc (double pamt1, double prate1)

Scenario:

Demonstrate a hypothetical example of computing various taxes on a person's property, sales, and income.

Demonstration/Code Snippet:

[Note: Numbering represents the sequence in which the program flows]



```
Tax on Home Value : 250
Tax on Your Gross Receipts from Sales : 150.000002235174
Tax on Your Home and Gross Sales : 400.000002235174
Tax on Personal Income : 1900.00000223517
Press any key to continue . . .
```

Figure 5.3-1: Output

```
using System;
```

Step 1: Create methods to calculate different taxes

```
class Program
{
      // This method takes four arguments: two amounts and  two
      //rates
      public static double TaxCalc(double pamt1, double prate1,
      double pamt2, double prate2)
      {
            double taxamt;
            taxamt = (pamt1 * prate1) + (pamt2 * prate2);
            return taxamt;
      }
      // This method only takes two arguments: an amount and a
      //rate
[3],[8]public static double TaxCalc(double pamt1, double prate1)
      {
            double taxamt;
[4],[9]     taxamt = pamt1 * prate1;
[5],[10]    return taxamt;
      }
      // This method only takes two argument, but datatype for 2nd
      //argument is float
[14]  public static double TaxCalc(double pgrossamt, float
      pgrossrate)
      {
[15]        double taxamt = 0;
[16]        taxamt = pgrossamt * pgrossrate;
[17]        return taxamt;
      }
      // This method only takes one argument
[21]  public static double TaxCalc(double pamt)
      {
```

```
[22]          double taxrate = 0.15;
[23]          double taxamt = 0;
[24]          taxamt = pamt * taxrate;
[25]          return taxamt;
        }
```

Step 2: Encode main program and call appropriate methods with correct arguments

Here, HV=Home_Value; HR=Home_Tax_Rate; GS=Gross_Sales; GR=Gross_Sales_Tax_Rate; PI=Personal_Income

```
[1]     static void Main(string[] args)
        {
              double Tax;
              double HV = 2500;
              double HR = 0.10;
              double GS = 3000;
              float GR = 0.05F;
              double PI = 10000;
              try
              {
[2]                 Tax=TaxCalc(HV,HR);//Calculate tax on the home
                    value
[6]                 Console.WriteLine("Tax  on  Home  Value  :  "  +
                    Tax);
[7]                 Tax=TaxCalc(GS,GR);
                    //Calculate  tax  on  your  gross  receipts  from
                    sales
[11]                Console.WriteLine();
[12]                Console.WriteLine("Tax  on  Your  Gross  Receipts
                    from Sales : " + Tax);
[13]                Tax=TaxCalc(HV, HR, GS, GR);// Calculate tax on
                    both //your home and gross receipts
[18]                Console.WriteLine();
[19]                      Console.WriteLine("Tax  on  Your  Home  and
                    Gross Sales : " + Tax);
[20]                Tax = Tax + TaxCalc(PI);
                          // And everyone gets a tax calculation on
                          personal //income
[26]                Console.WriteLine();
[27]                Console.WriteLine("Tax on Personal Income :" +
                    Tax );
[28]                Console.WriteLine();
              }
              catch(Exception ex)
              {
                    Console.WriteLine(ex.Message);
              }
        }
}
```

Context:

- Overloaded methods provide different functionality with the same name.

Practice Session:

- Write a code to overload ‘+’ operator for addition, as well as to concatenate two words.

Check List:

- Importance of signatures

Common Errors:

- Proper arguments are not passed when the methods are invoked.
- Type of arguments does not match.

Exceptions:

- Should not use overloads when two methods really perform different tasks - otherwise confusion regarding user of the classes will take place.

Lessons Learnt:

☑ Use Overloaded Methods to satisfy different purpose.

Best Practices:

- To write overloaded methods with same names with same purpose, but differing inputs
- Not to overuse, as it may be difficult to debug and maintain.

Crossword: Unit-5 **Estimated Time: 10 mins.**

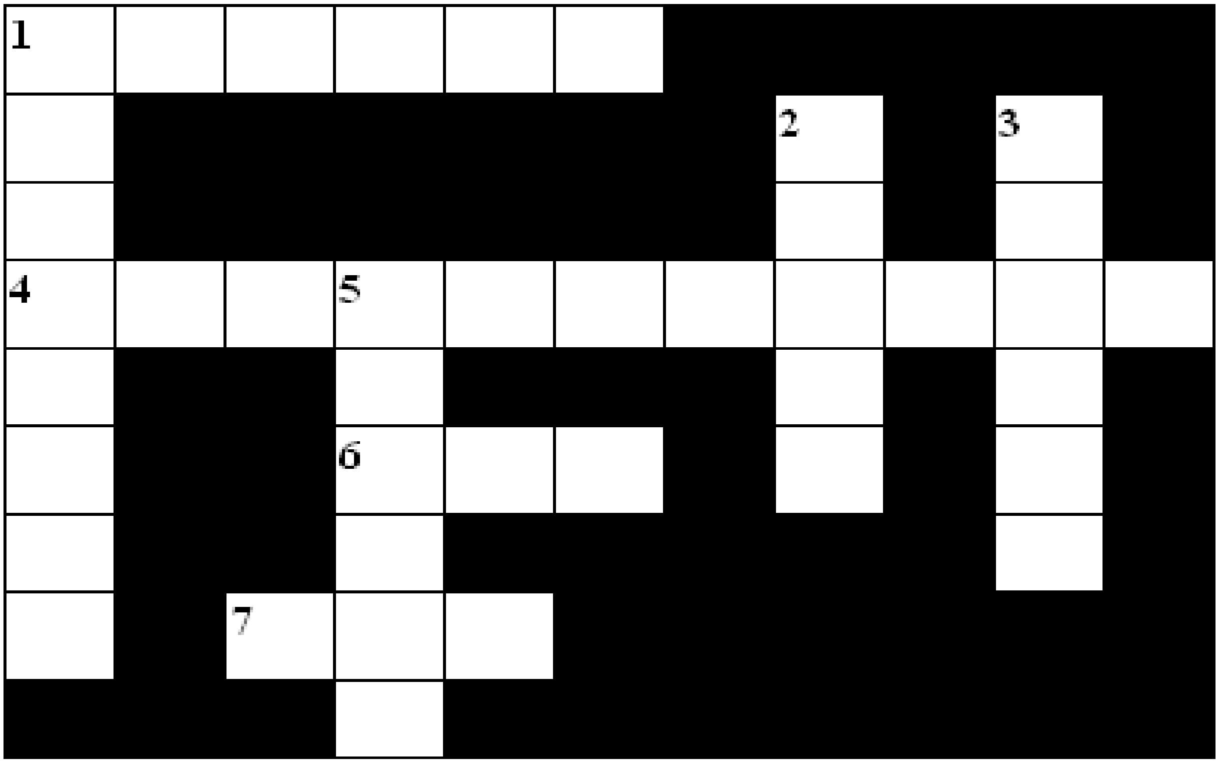


Fig. 5-1

Across:

1. What is required to call the methods?(6)

4. Overloading helps in code __________.(11)

6. _________ modifier can be used with reference parameters.(3)

7. Output parameters need the ___ modifier as part of both the declaration and invocation.(3)

Down:

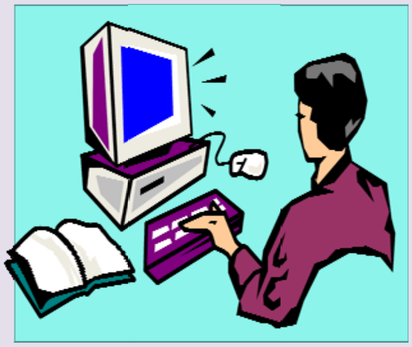
1. Overloading " +" is called _________ overloading.(8)

2. By default parameters are ______ type parameters.(5)

3. __________ represents the behavior/functionality of the object.(6)

5. String What IsYourName (). Here ________ is the return-type.(6)

6.0 Arrays

Topics	
- 6.1 Overview of Arrays - 6.2 Jagged Arrays - 6.3 Crossword	

Topic: Arrays

Estimated Time: 30 mins.

Objectives: After the completion of module, participant should know

- Different types of arrays
- Declaration and definition of array

Presentation:

- An Array is a collection of similar data types stored in adjacent memory locations.
- In C#, an array falls in category of reference type and used to store objects of the same type and provides access to the objects using an index.
- There are mainly three types of arrays
  - ➢ Single- Dimensional Array
  - ➢ Two-Dimensional Array
  - ➢ Multidimensional Array

Creating an Array:

- Three step for creating an array (all these steps can be combined)

1. The array must be declared

```
int [] numbers; //single dimension array
//multi dimension array syntax is as follows
<Type> [,…,] name= new <type> [size1, size2,size3,…….size[N];
```

2. Instantiate the array

```
numbers = new int [3];
```

3. Initialize the array with values

```
numbers [0] =1;
```

[OR]

A combined step could be as follows

```
int [] numbers = new int [] {3,1,4};
```

- Initializing a two dimensional Array

```
int [,] table=new int [10, 20];
```

Scenario

Mr. Johnson took a challenge to create software for junior section of the school. He needs multiplication table of a number to be displayed when the student enters the number. So create a console application to print multiplication table of that number entered by the student.

Demonstration/Code Snippet:

[Note: Numbering represents the sequence in which the program flows]

```
using System;
```

Step 1: Declaring,initiating and initializing the array within a class

```
class Program
{
[1]   static void Main(string[] args)
      {
            int[,] array = new int[5, 2];
            int[,] table = new int[5, 2];
            int i, j;
            int c = 0;
```

Step2: Declaring the variable for input

```
            int ch;
[2]         Console.WriteLine("Enter the no. whose table U
            want");
[3]         ch = Convert.ToInt32(Console.ReadLine());
```

Step3: Applying For loop for multiplication and second For loop for displaying output

```
[4]         try
            {
[5]               for (i = 0; i < 5; i++)
                  {
[6]                     for (j = 0; j < 2; j++)
                        {
[7]                           array[i, j] = c;
[8]                           c++;
                        }
                  }
[9]               for (i = 0; i < 5; i++)
                  {
[10]                    for (j = 0; j < 2; j++)
                        {
[11]                          table[i, j] = ch * array[i, j];
[12]                          Console.WriteLine(ch + "*" +
                              array[i, j] + "=" + table[i, j]);
                        }
                  }
[13]              Console.ReadLine();
            }
            catch(Exception e1)
            {
                  Console.WriteLine(e1.Message());
            }
     }
}
```

Context

- To store offset of values of the same data type with a common name

Practice Session

- Declare an array of student's names where a name itself can be an array of three strings - first name, middle name and last name

Check List

- What is Array in C#?
- Accessing of array elements.
- Initialize and create arrays.

Common Errors

- The number of inputs passed should not exceed the size of the array.
- When we explicitly specify the array size during initialization, the size must agree with the number of initializes.

Exceptions

- In C#, arrays are actually objects, not just addressable regions of contiguous memory as in C and C++.

Lessons Learnt

☑ To declare and initialize arrays in a single statement.

Best Practices

- Use arrays to sort, search, get and set items of same data type.

Topic: Jagged Arrays

Estimated Time: 45 mins.

Objectives: After the completion of module, participant should know

- Creating jagged arrays
- Declaring, initiating and initializing jagged arrays

Presentation:

- Jagged arrays are nothing but array of arrays.
- The elements of a jagged array can be of different dimensions and sizes.
- Elements of jagged arrays are reference types and can be initialized to null.

Creating an Array:

- Declares a variable to hold an array of arrays with elements of the Data Type, creates the array, and assigns it to the variable.
  int [][] jaggedArray = new int[3][];

Scenario :

Mr. Johnson wants to group student in his class based on their Last name. Create a console application using jagged array.

Demonstration/Code Snippet:

[Note: Numbering represents the sequence in which the program flows]

```
using System;
```

Step 1: Declaring,initiating and initializing the array within a class

```
class Program
{
[1]    static void Main(string[] args)
       {
             try{
             String lastname;
             //Declaring the Jagged array
[2]          String [][]stud=new String[4][];
[3]          stud[0]=new String[3];
[4]          stud[1]=new String[2];
[5]          stud[2]=new String[4];
[6]          stud[3]=new String[5];
```

Step 2: Assigning the Values to the Array

```
[7]          for(int i=0 ; i < stud.Length ; i++)
             {
                  Console.WriteLine("Enter the Last name");
                  lastname = Console.ReadLine();
                  Console.WriteLine("Enter the name of students
                  whose last name is "+ lastname);

                  for(int j=0 ; j < stud[i].Length ; j++)
                  {
                        String firstname= Console.ReadLine();
                        stud[i][j]=firstname;
                  }
            }
                  Console.WriteLine("");
```

Step 3: Printing the values assigned to an array

```
            for(int i=0 ; i < stud.Length ; i++)
            {
                  for(int j=0 ; j < stud[i].Length ; j++)
                  {
                        Console.Write(arr[i][j]);
                        Console.Write("\0");
                  }
                  Console.WriteLine("");
            }
            Console.ReadLine();}
            catch(Exception e1)
            {
                  Console.WriteLine(e1.Message());
            }
      }
}
```

Context :

- It contains some number of inner arrays, each of which may have a unique upper limit.

Practice Session:

- Declare an array of student's names where a name itself can be an array of three strings First Name, Middle Name and Last Name.

Check List:

- Jagged Array in C#.
- Accessing of array elements.
- Create and Initialize jagged arrays.

Common Errors :

- Number of inputs passed should not exceed the size of an array.
- Compilation errors can arise from misunderstanding of the rules for declaring, creating, and initializing arrays.

Exceptions :

- Every access to an element of the array must specify a valid index, or subscript, for every dimension. If any index is out of bounds results in, an IndexOutOfRangeException.

Lessons Learnt :

☑ To declare and initialize jagged arrays in a single statement.

Best Practices :

- Use jagged arrays instead of Multi Dimensional array when required.

Crossword: Unit-6 **Estimated Time: 10 mins.**

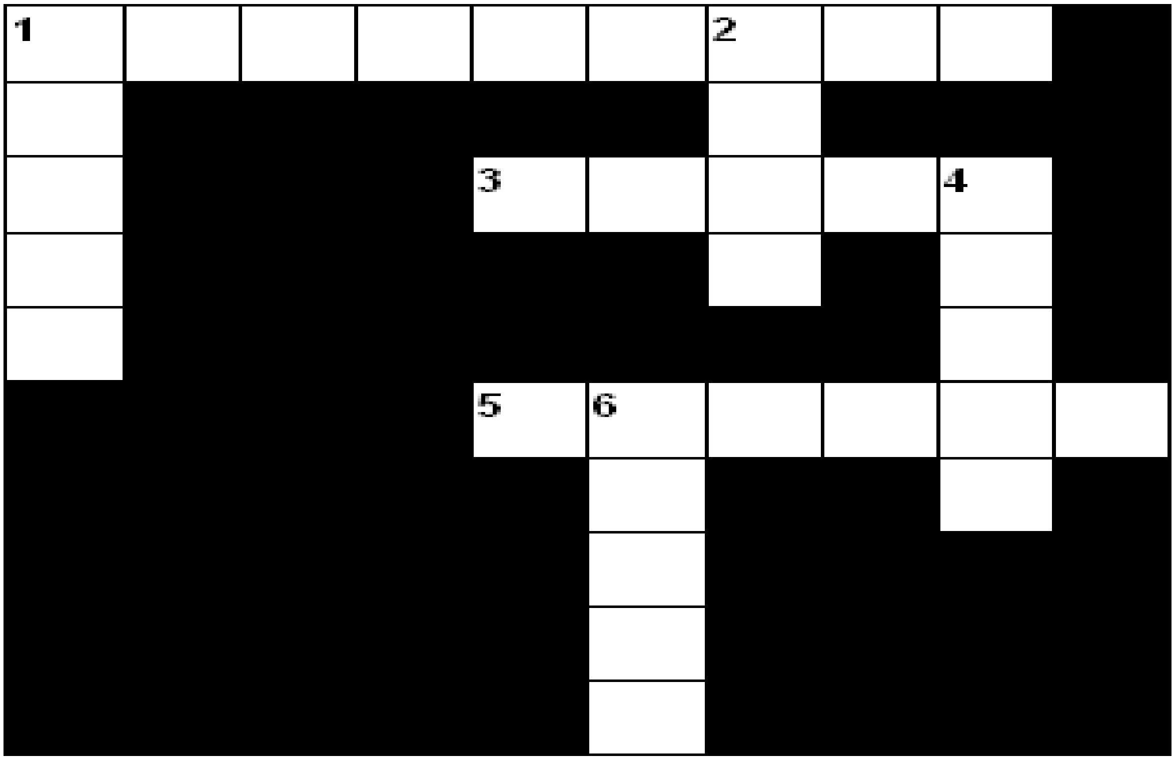


Fig.6-1

Across:

1. Elements of jagged arrays are _______ types.(9)

3. Types of arrays one dimensional, two dimensional and _______ dimensional.(5)

5. An array with arrays for elements is called an array of arrays, or a _______ array. (6)

Down:

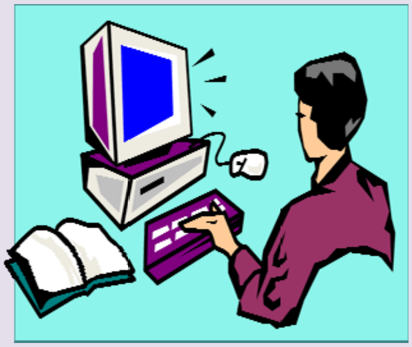
1. Initializing a two dimensional Array, Ex: int [,] table=new int [10,20]; //Mention whether the statement is Right or Wrong (5)
2. Elements of jagged arrays are initialized to _____.(4)

4. In C#, an array falls in category of reference type and used to store objects of the same type and provides access to the objects using an ______. (5)
6. An ______ is a collection of similar data types stored in adjacent memory locations.(5)

7.0 Essentials of Object Oriented Programming

Topics

- 7.1 Classes and Objects
- 7.2 Encapsulation
- 7.3 Defining of Object Oriented Systems
- 7.4 Crossword



Topic: Classes and Objects

Estimated Time: 20 mins.

Objectives: After the completion of module, participant should know

- Concepts of Class and Object.
- Implement Class and Object in C# applications.

Presentation:

Class

- It is a collection of properties and functionality that describes a group of objects.
- For example, human being is object of a class called Person.
- There are two types of Classes: The built- in classes that comes along with Framework Class Library and the user defined classes.
- Object is an instance of a class.
- Object exhibits
  - ➢ Identity: A distinguishable feature from one another.
  - ➢ Behavior: Which describes the task to be performed
  - ➢ State: To store information.

Class v/s Structure

Class	**Structure**
1. They are reference type	1. They are value type
2. By default members have public accessibility	2. By default members have private accessibility
3. Supports inheritance and polymorphism.	3. Does not support inheritance and polymorphism.
4. Supports the special methods like constructors and destructor.	4. Supports only constructors.

Partial Modifier

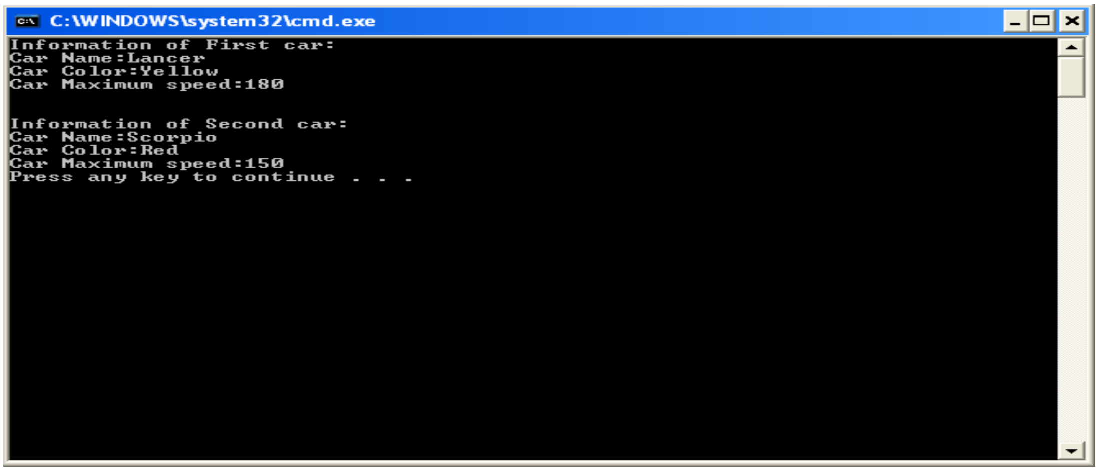
- Structures, Classes and Interfaces are defined with a type modifier named **partial,** so as to define a type across multiple *.cs files.
- Different parts of the class, struct, or interface can be defined within the namespace.
- The following are merged together from all the partial-type definitions:
  - ➢ XML comments
  - ➢ Interfaces
  - ➢ Generic-type parameter attributes
  - ➢ Class attributes
  - ➢ Members

Scenario:

Mr. George has bought a new car. He wants to send car information to his friends, so display information of cars on screen and maintain that information in one class so it can use later on.

Demonstration/Code Snippet :

[Note: Numbering represents the sequence in which the program flows]



```
Information of First car:
Car Name:Lancer
Car Color:Yellow
Car Maximum speed:180

Information of Second car:
Car Name:Scorpio
Car Color:Red
Car Maximum speed:150
Press any key to continue . . .
```

Figure 7.1-1: Output

```
using System;
using ...
```

Step 1: Create a Class named Car. Declare the properties of Car

```
class Car
{
      public string car_name;
      public string car_color;
      public int car_maxspeed;
}
```

Step 2: In Main() method declare two objects of class Car

```
[1]   class Program
      {
      static void Main(string[] args)
[2]   {
[3]   try
[4]   {
[5]       Car car1, car2; //Declaring objects
[6]       car1 = new Car();//Initializing memory to objects
[7]       car2 = new Car();
```

Step 3: Initializing car1 and car2 data members

```
[8]       car1.car_name = "Lancer";
[9]       car1.car_color = "Yellow";
[10]      car1.car_maxspeed = 180;
[11]      car2.car_name = "Scorpio";
[12]      car2.car_color = "Red";
[13]      car2.car_maxspeed = 150;
```

Step 4: Displaying the information on Console for both objects of class Car

```
          Console.WriteLine("Information of First car:");
          Console.Write("Car Name:");
          Console.WriteLine(car1.car_name);
          Console.Write("Car Color:");
          Console.WriteLine(car1.car_color);
          Console.Write("Car Maximum speed:");
          Console.WriteLine(car1.car_maxspeed);
          Console.WriteLine("\n");
          Console.WriteLine("Information of Second car:");
          Console.Write("Car Name:");
          Console.WriteLine(car2.car_name);
          Console.Write("Car Color:");
          Console.WriteLine(car2.car_color);
          Console.Write("Car Maximum speed:");
          Console.WriteLine(car2.car_maxspeed);
      }
      catch (Exception ex)
      {
           Console.WriteLine(ex.Message);
      }
   }
}
```

Context:

- Encapsulating different properties of same type into single object.
- To build modularity into the programs.
- When working on large projects, spreading a class over separate files allows multiple programmers to work on it simultaneously.

Practice Session:

- Create a class named person having properties hair color, eye color, height and weight. Make two objects named Michael and Sam and define different properties for both the objects.

Check List:

- Class
- Object

Common Errors:

- Trying to access private variables outside the class.

Exception:

- Trying to access the object for which memory is already deallocated using destructor i.e. NullReferenceException.

- The ‘partial’ modifier is not available on delegate or enumeration declarations.

Lessons Learnt :

☑ Allocate memory using ‘new’ operator to instantiate an Object.

Best Practices:

- Class is an object oriented approach better than procedural oriented approach.

Topic: Encapsulation

Estimated Time: 30 mins.

Objectives:

After the completion of module, participant should know

- The concepts of encapsulation.
- Defining the properties using ‘get’ and ‘set’ methods.
- Assigning and accessing class variables using Mutator and Accessor.

Presentation:

Encapsulation

- It is a mechanism for hiding instance variables. Only the methods necessary to use with an object of the class, are exposed.

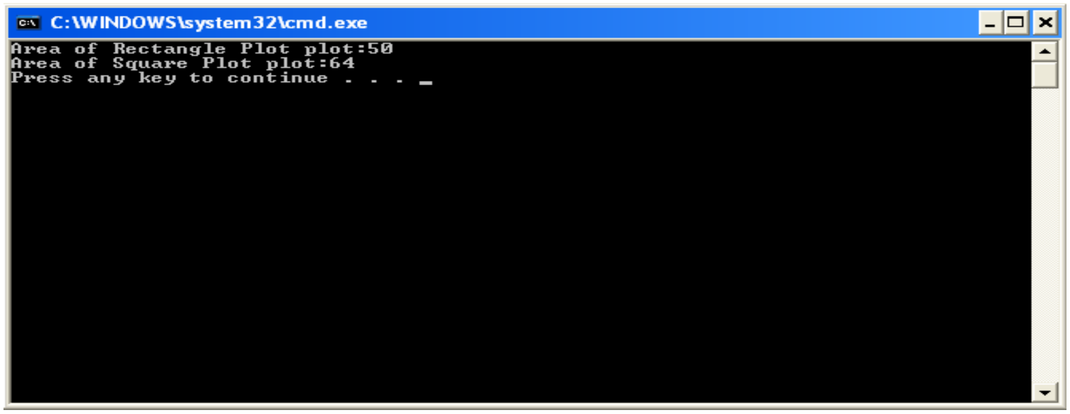
- A powerful mechanism for reducing complexity and protecting data of individual objects at the programming level.
- It is a mechanism to combine data and the actions (methods) acting on the data into a single entity.
- It can be implemented using the following
    1. **Properties**
    2. **Accessor (get method)** and **Mutator (set method)** methods

Scenario :

Mr. Watson desires to build two multi-cuisine restaurants in Newyork city and he needs to buy two plots with sufficient area.

Demonstration\Code Snippet :

[Note: Numbering represents the sequence in which the program flows]



```
Area of Rectangle Plot plot:50
Area of Square Plot plot:64
Press any key to continue . . . _
```

```
using System;
using ...
```

Step 1: Create a Class named Shape

```
class Shape
{
      public string name;
      //Declaring length and breadth as private variable
      private int length;
      private int breadth;
```

Step 2: Make a property to get and set the length. Accessor and Mutator methods to get and set the breadth

```
       //Encapsulation Using Property
              public int Length
              {
              get{
                  return length;   }
              set{
[11],[17]         length = value;  }
```

```
                 // value keyword refers to the value assigned
                 }
               //Encapsulation using Accessor and Mutator method
[20]           public int GBtbreadth()       //Accessor method
[21]           {return breadth;}
[13],[19]      public void SetBreadth(int value)   //Mutator Method
[14],[20]      {breadth = value;}
```

Step 3: Make a method to calculate area

```
[22],[27]      public int Calculate_Area()
[23],[28]      {return length * breadth;}
}
```

Step 4: In the main method declare two objects for class shape and calculate area of plot

```
class Program
{
[1]         static void Main(string[] args)
[2]         {
[3]             try
[4]             {
[5]                   Shape rect, square;
[6]                   int area;
[7]                   rect=new Shape();
[8]                   square=new Shape();
[9]                   rect.name = "Rectangle Plot";
[10]                  rect.Length = 10;//set length in property
[12]                  rect.SetBreadth(5);//set breadth in mutator
[15]                  square.name="Square Plot";
[16]                  square.Length = 8;
[18]                  square.SetBreadth(square.Length);
[21]                  area = rect.Calculate_Area();
[24]                  Console.Write("Area of "+rect.name+" plot:");
[25]                  Console.WriteLine(area);
[26]                  area = square.Calculate_Area();
[29]                  Console.Write("Area of " + square.name + "
                      plot:");
[30]                  Console.WriteLine(area);
                }
[31]            catch (Exception ex)
                {  Console.WriteLine(ex.Message);   }
      }
 }
```

Context:

- Encapsulation allows control by keeping related things together viz. methods and Properties.
- Encapsulation allows changes to the methods and properties of the class, without making changes to the method calls.

Practice Session:

- Create a Class with two private members and using any of the encapsulation method access it in the main method.

Check List :

- Description of Encapsulation.
- Implementation of Encapsulation.

Common Error :

- If properties are not implemented properly expected outputs will be different.

Exception :

- Trying to access private members of class outside the Class

Lessons Learnt:

☑ We cannot directly access private member out of the class, we need to implement Accessor and Mutator methods or implement property to access it.

☑ Encapsulation provides a way to protect data from unintended corruption. Rather than defining the data in the form of public, we can declare those fields as private.

Best Practices:

- Always try to use private members in class as it provides loose coupling with the Object.

Topic: Object Oriented Features

Estimated time: 35 mins.

Objectives:

After the completion of module, participant should know

- Inheritance Concept in OOP.
- Implementation of polymorphism.
- Abstract Class and Interface.

Presentation:

Inheritance

- In object-oriented programming, inheritance is a way to form new classes using already defined classes.
- Inheritance allows reusing some portion of code.
- The new classes, known as Derived Classes, inherit attribute and behavior of the pre-existing classes, which are referred to as Base Classes.

Polymorphism

- Polymorphism means having more than one form.
- Polymorphism treats derived class members just like their parent class members.

- It is the ability of objects belonging to different types to respond to method calls of the same name, each one according to an appropriate type-specific behavior.
- Polymorphism can be performed in two ways

  1. Overloading

  ➢ Overloading Polymorphism is the use of one method signature, or one operator such as "+", to perform several different functions depending on the implementation.

  2. Overriding

  ➢ Overriding is a language feature that allows a derived class to provide a specific implementation of a method that is already provided by one of its base class.

  ➢ The function which is overridden is called **Virtual Function.**

Abstract Class:

- The abstract keyword enables you to create classes and its members exclusively for the purpose of inheritance. It provides a common definition of a base class that multiple derived classes can share.
- An abstract class must be inherited, but cannot be instantiated.
- Derived classes of the abstract class must implement all abstract methods.

Interface:

- Interfaces describe a group of related behaviors which belongs to any class or struct.
- An interface cannot contain fields.
- Interfaces members are automatically public.
- Classes and Structures can inherit from Interfaces similar to how classes can inherit base class or structure, with two exceptions:
  1. Class or Structure can inherit more than one interface
  2. When a Class or Structure inherits an Interface, it inherits member definitions but not implementations.

Scenario:

Write down a sample code to demonstrate concept of overloading and overriding.

Demonstration/Code Snippet:

Overloading

```
using System;
using……
```

Step 1: Create a Class with two methods having same name but different parameters

```
class demo
{
  public void one()
  {
   Console.WriteLine("Method without parameter called");
  }
  public void one(int n)
  {
   Console.WriteLine("Method called without parameter:" + n);
  }
}
```

Step 2: Declare Object of Class demo in Main () method

```
demo obj=new demo();
obj.one();            //first method invoked
obj.one(1);           //second method invoked
```

Overriding

Step 1: Create a class and derive another class from it. Then create a method with same name and same signature in both classes

```
 class A
 {
   public virtual void Demo()
   {
    Console.WriteLine("Demo Method in class A called");
   }
 }
 class B:A
 {
   public override void Demo()
   {
      Console.WriteLine("Demo Method in class B called");
   }
}
```

Step 2: Now call the methods of both the class using respective objects

```
   A objA=new A();
   B objB=new B();
   objA.Demo();          //Demo method in class A is called
   objB.Demo();         //Demo method in class B is called
```

Context:

- Use Inheritance when many classes with same root have different implementation.
- When same method performs different function, use Polymorphism.

Practice Session :

- Do the function having same name and different return type provides function overloading?

Check List :

- Inheritance
- Polymorphism using Overloading and Overriding
- Abstract Class
- Interface

Common Error :

- Changing the signature of method in overriding.
- Instantiating an abstract class.

Exception :

- Ambiguity error in function overloading.

Lessons Learnt :

☑ Number of parameters or type of parameters must be different for method overloading.

☑ Use interface in place of abstract class when required in all methods in class to be abstract.

☑ In overloading, at compile time it is decided which version of method to call while in overriding it is decided at run time.

Best Practices:

- Inheritance increases code reusability.

Crossword: Unit-7 **Estimated Time: 10 mins.**

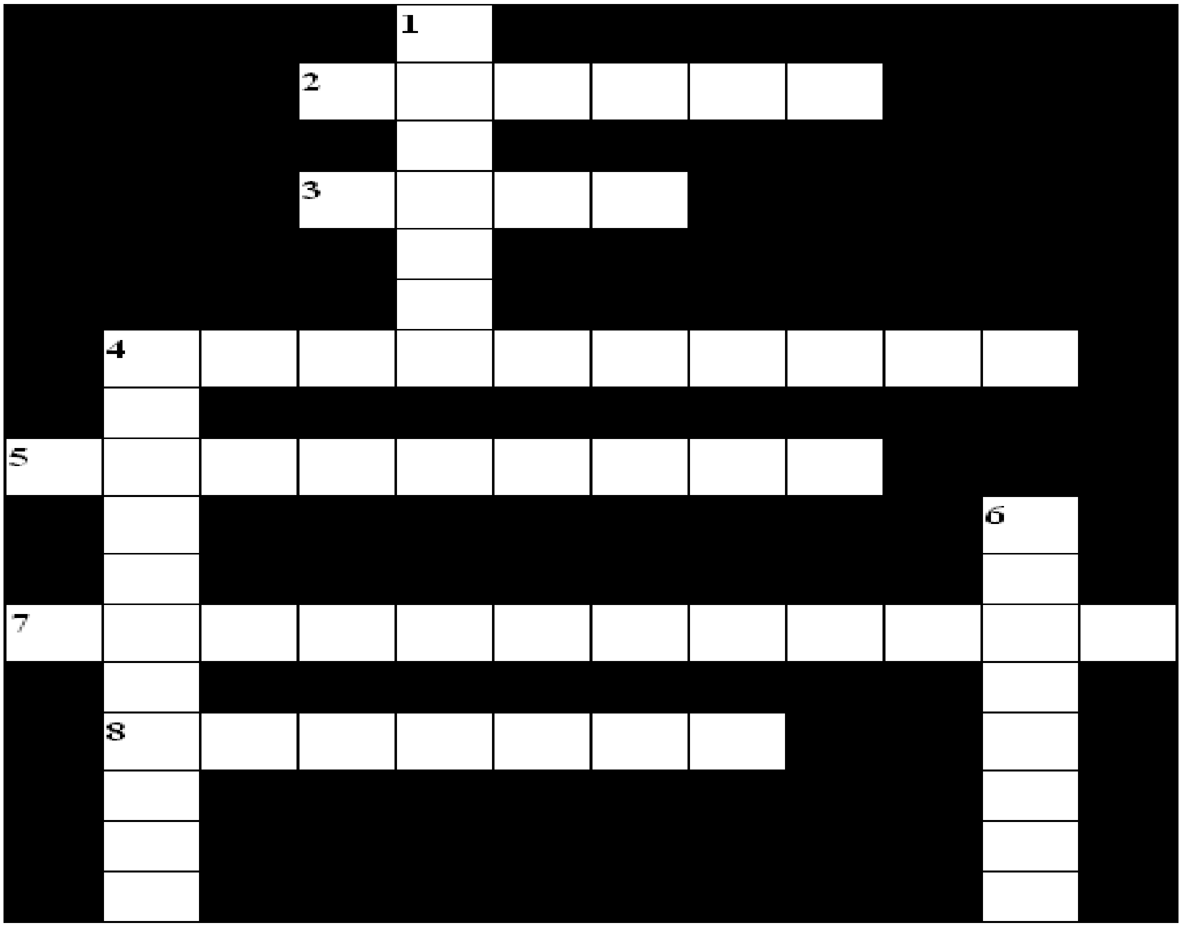


Fig. 7-1

Across:

2. Interfaces members are automatically ________.(6)

3. The new classes, known as derived classes, inherits attribute and behavior of the pre-existing classes, which are referred to as _______ classes.(4)

4. Polymorphism can be achieved using ______ .(10)

5. Classes are ________ type and Structures are value type.(9)

7. Classes support _________ while structures don't.(12)

8. The new classes, known as ________ classes, inherit attribute and behavior of the pre-existing classes, which are referred to as base classes.(7)

Down:

1. Encapsulation can be implemented using the following methods in C#
1. Using Properties
2. Using Accessor and ________ methods(7)

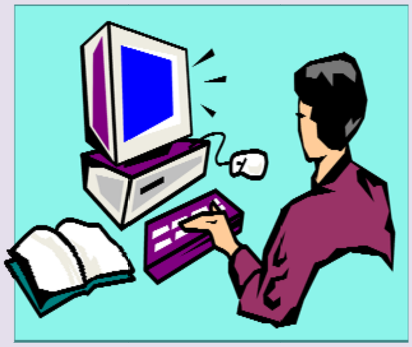
4. Polymorphism can be achieved using _______ .(11)

6. The _______ keyword enables you to create classes and its members solely for the purpose of inheritance.(8)

8.0 Using Reference Type Variables

Topics

- 8.1 Usage of Reference Type Variables
- 8.2 Namespaces in the .NET framework
- 8.3 Data Conversions
- 8.4 Crossword



Topic: Reference Type Variables

Estimated Time: 40 mins.

Objectives: After the completion of module, participant should know

- Significance of reference type variable.
- Declaration and implementation of reference type variable.

Presentation:

Need of reference type variable

- Reference Type stores a reference to the value's memory address, and is allocated on the heap. It can be self-describing type, pointer type, or interface type.
- Copying reference type variable just makes a copy of reference to the variable. No direct copy of value is made.

Significance of reference type variable

- Operations on one reference type variable can affect the same object referred to by another variable.

Declaration of reference type variable

Example: coordinate c1 =new coordinate ();

Common reference type

1. Exception Class
   - ➢ Exception objects are used to raise exception.
   - ➢ Create an exception object by new and throw object by throw.
2. String Class
   - ➢ Use to store multiple characters Unicode data.
   - ➢ Its immutable.
3. Object Type
   - ➢ Its base class for all classes

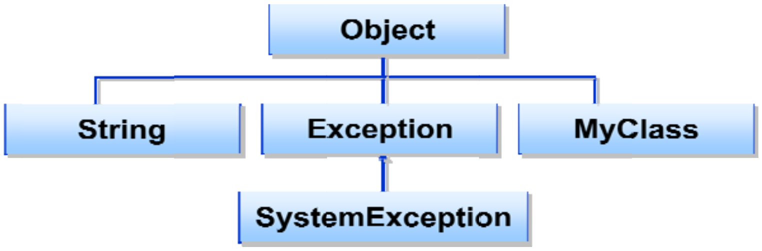


Figure 8.1-1: Hierarchy of Classes

Properties

- Properties are used to set and get the value
  Example:

```
public int Number
{
            get { return n;}
            set { n=value; }
}
```

String Comparisons

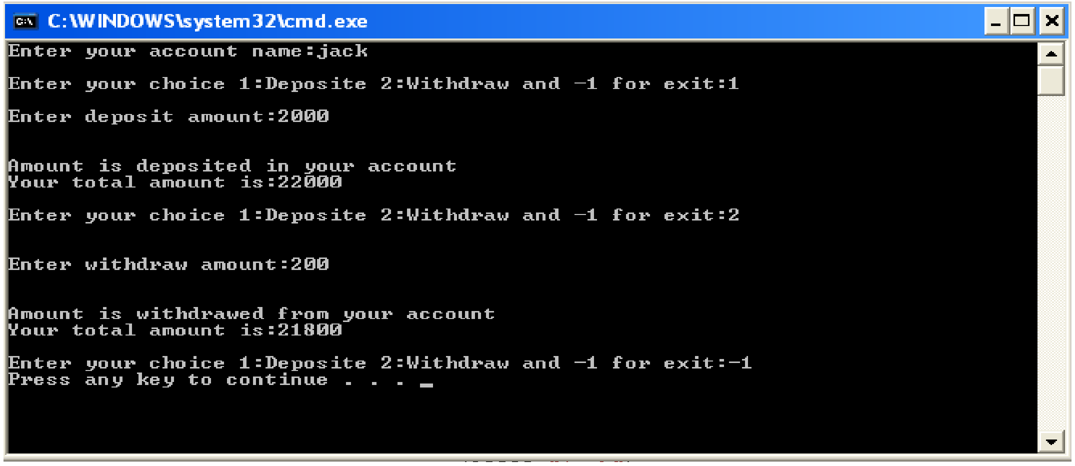
- Two methods are used for string comparison
  1. Equals() Method
  2. Compare() Method
- == and != operators are overloaded for string comparison.

Scenario :

Mr. Jack who is a businessman uses internet banking facility provided by Barclays bank. He requires transaction through internet banking.

Demonstration/Code Snippet :

[Note: Numbering represents the sequence in which the program flows]



```
Enter your account name:jack

Enter your choice 1:Deposite 2:Withdraw and -1 for exit:1

Enter deposit amount:2000

Amount is deposited in your account
Your total amount is:22000

Enter your choice 1:Deposite 2:Withdraw and -1 for exit:2

Enter withdraw amount:200

Amount is withdrawed from your account
Your total amount is:21800

Enter your choice 1:Deposite 2:Withdraw and -1 for exit:-1
Press any key to continue . . . _
```

Figure 8.1-2: Output

```
using System;
using...
```

Step 1: Make one class having account no and account amount

```
      public class account
      {
              public string accountNo;
              public int Amount;
      }
      class test
      {
[1]          static void Main(string[] args)
              {
[2]                  string jackAcc;
[3]                  int amt,i,j=1;
```

Step 2: Declarig refrence variable

```
[4]                  account acc = new account();
[5]                  acc.accountNo = "jack";
[6]                  acc.Amount = 20000;
```

Step 3: Using string comparison method find valid account

```
[7]                  Console.Write("Enter your account name:");
[8]                  jackAcc = Console.ReadLine();
[9]                  i = String.Compare("jack", jackAcc);
[10]                 account deposit;
```

Step 4: Multiple refrences to same object

```
[11]            deposit = acc;
[12]            if (i == 0)
                {
[13]                  while (j > 0)
                      {
[14]                        Console.Write("\nEnter your choice
                            1:Deposite
                            2:Withdraw and -1 for exit:");
[15]                        j = int.Parse(Console.ReadLine());
[16*]                       switch (j)
                            {
                            case 1:
                             Console.Write("\nEnter deposit
                            amount:");
                            amt= int.Parse(Console.ReadLine());
                            deposit.Amount=deposit.Amount+amt;
                            Console.WriteLine("\n\nAmount is
                            deposited in account");
                            Console.WriteLine("Your total
                            amount is:{0}", +deposit.Amount);
                            Break;
                            case 2:
                            Console.Write("\n\Enter
                            withdrawamount:");
                            amt= int.Parse(Console.ReadLine());
                            deposit.Amount=deposit.Amount-amt;
                            Console.WriteLine("\n\nAmount is
                            withdrawn from your
                            account");
                            Console.WriteLine("Your total
                            amount is:{0}",+deposit.Amount);
                            break;
                            }
                      }
                }
[12*]           else
[13*]                 Console.WriteLine("You are not authorized
                      user");
         }
}
*: Conditional statement.
```

Context :

- Use reference type variable to refer same object.
- Overload string operator for string comparison.

Practice Session :

- Difference between Value Type and Reference Type.
- Using property ‘set’ or ‘get’ value of any class variable.

Check List :

- Use of reference type variable
- Property
- String comparison method

Common Errors:

- Uninitialized references

Exception:

- Invalid reference at runtime gives an exception error.

Lessons Learnt :

☑ Using reference type, operations on one variable can affect the same object referred to by another variable.

☑ Class type, interface type, delegate type and array type are all reference type.

Best Practices:

- A reference type is a type having its value as reference to the appropriate data rather than the data itself.

Topic: Namespaces in the .Net Framework.

Estimated Time -30 mins

Objectives: After the completion of module, participant should know

- Implementation of user-defined namespace.
- Different types of namespaces in .Net Framework.

Presentation:

- Namespace is a way to keep one set of types separate from another.
- It organizes code and gives a way to create globally-unique types.
- It logically arranges classes, structures, interfaces, enumerations, delegates or another namespace.

- Common namespace used by the .Net Framework library is **System**
  In earlier chapters, using System; ….is used
- User-defined namespace can also be created:

  ```
  namespace <namespace_name>
  {
   // Classes and/or structures and/or enumerations and/or interfaces and/or delegates
  etc.
  }
  ```
- A default namespace is created even if one is not created.
- Namespace Alias Qualifier is used to access the namespace member outside the defined namespace
  General Form: namespace-alias: identifier
- **Namespaces in .Net Framework**
  - System.IO Namespace : Access to File System input/output
    - File, Directory
    - StreamReader, StreamWriter
    - FileStream
    - BinaryReader, BinaryWriter
  - System.Xml Namespace
    - Xml support and various-Xml related standards
  - System.Data Namespace
    - System.Data.SqlClient – SQL Server .Net Data Provider
    - System.Data – Consists mostly of the classes of ADO.Net Architecture
  - Other Useful Namespaces
    - System.Net
    - System.Net.Sockets

Scenario:

Mr. Lalit Modi, Chairman of IPL needs a new format to launch for next year IPL. He wants his engineers to work on a new design that could be implemented.
[Engineers can thus create a separate namespace for their new design].

Demonstration/Code Snippet :

```
using ……….
```

Step 1: Define a Class IPL under defined namespace

```
namespace @namespace1
{

    class IPL
    {
        public void Design()
        {
            Console.WriteLine("This is design method called in
            namespace1 ");
        }
    }
}
```

Step 2: Make Object of Class IPL

```
class Program
{
    static void Main(string[] args)
    {
        //Object of class IPL of namespace1 created.
        namespace1.IPL ipl_obj = new namespace1.IPL();
        ipl_obj.Design();
    }
}
```

Context:

- Use namespace to avoid name collision

Practice Session:

- List the namespace which is used to create network connection

Check List:

- User-defined namespace
- Different namespaces
- Namespace Alias Qualifier

Common Errors:

- If a namespace is misspelled, then it gives compiler-error

Exception:

- When same name is declared within two different namespaces, it causes an ambiguity, when tried to access the namespace
  Example:
  MyClass is declared within two different namespace, say namespace1 and namespace2. It may show ambiguity when following statement is used:
  using namespace1;

//which MyClass will be called. MyClass declared in namespace1 or namespace2

Lessons Learnt :

☑ Namespaces prevent name conflicts

Best Practices:

- Avoid creating more number of user-defined namespaces as it leads to ambiguity.

Topic : Data Conversions

Estimated Time- 30 mins

Objectives: After the completion of module, participant should know

- Operator Conversions
- Boxing and Unboxing
- Conversions and Interfaces

Presentation:

- Conversion is a process of converting one DataType or object type into another.
- Two types of conversion are:
  - Implicit Conversions - One type of data is automatically converted into another type of data and there is no data loss.
  - Explicit Conversion is a forced conversion and there may be a data loss.

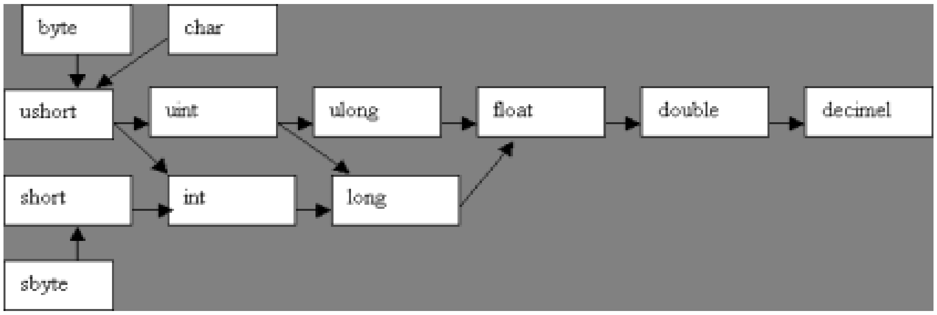


Fig: 8.4-1 Various Data Types and Possible Conversions

Other type of Conversions

- Implicit Enumeration Conversion
- Implicit Reference Conversion
- Boxing Conversions
- Explicit Conversions in C#
- Explicit Numerical Conversions
- Explicit Enumeration Conversions
- Explicit Reference Conversions
- Un-boxing Conversions
- User-Defined Conversions
- Arithmetic Conversions

Boxing - Conversion of Value Type to Reference Type. It is of implicit type
Unboxing- Conversion of Reference Type to Value Type. Boxing is necessary for Unboxing

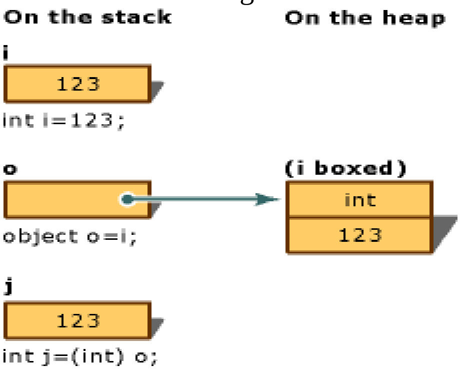


Figure 8.4-2: Boxing and Unboxing

- Interfaces describe a group of related behaviors that belongs to any class or structure.

Scenario:

Mr. John is a Bank Manager and he wants to maintain a record of all accountholder of his bank whether account holder are employee of that bank or not.

Demonstration/Code Snippet :

```
using System;
using …
```

Step1: Create Class Person and Employee

```
class Person
{
      private string name;
      private int age;

[5]   public string propName
      {
            get { return name; }
[6]         set { name = value; }
      }

[7]   public int propAge
      {
            get { return age; }
[8]         set { age = value; }
      }
}

public class Employee : Person
{
            private string designation;
            private int salary;
[10]        public string propDesignation
            {
                  get { return designation; }
[11]              set { designation = value; }
            }
[13]        public int propSalary
            {
                  get { return salary; }
[14]              set { salary = value; }
            }
      }
```

Step2: In main method typecast employee object to Person object

```
class Program
{
[1]   static void Main(string[] args)
      {
[2]         Person obj_person = new Person();
[3]         Employee obj_emp  = (Person)obj_person;//Type Casting
[4]         obj_emp.propName = "Andrew";
[6]         obj_emp.propAge = 28;
[9]         obj_emp.propDesignation = "HR Manager";
[12]        obj_emp.propSalary = 3000;
      }
}
```

Context :

- Data Conversions is used for application interoperability and using new features.

Practice Session :

- Find the differences between Interfaces and Classes.
- Demonstrate a simple code for short to long and long to short file names conversion.

Check List :

- Operator Conversions
- Boxing and Unboxing
- Interfaces and Conversions

Common Errors :

- Updating an instance of a value type does not actually update the instance at all.
- Unboxing requires a *cast* for determining the type of an object. If the object is not the correct type, it will cause an exception during the cast operation.

Exception :

- Casting does not physically move or operate on an object. Casting merely changes the way objects are treated in a program by altering their reference type.

Lessons Learnt:

☑ Different type of conversion techniques.
☑ Uses of Boxing and Unboxing.

Best Practices:

- Explicitly implement interface member with inheritance.

Crossword: Unit-8 **Estimated Time: 10 mins.**

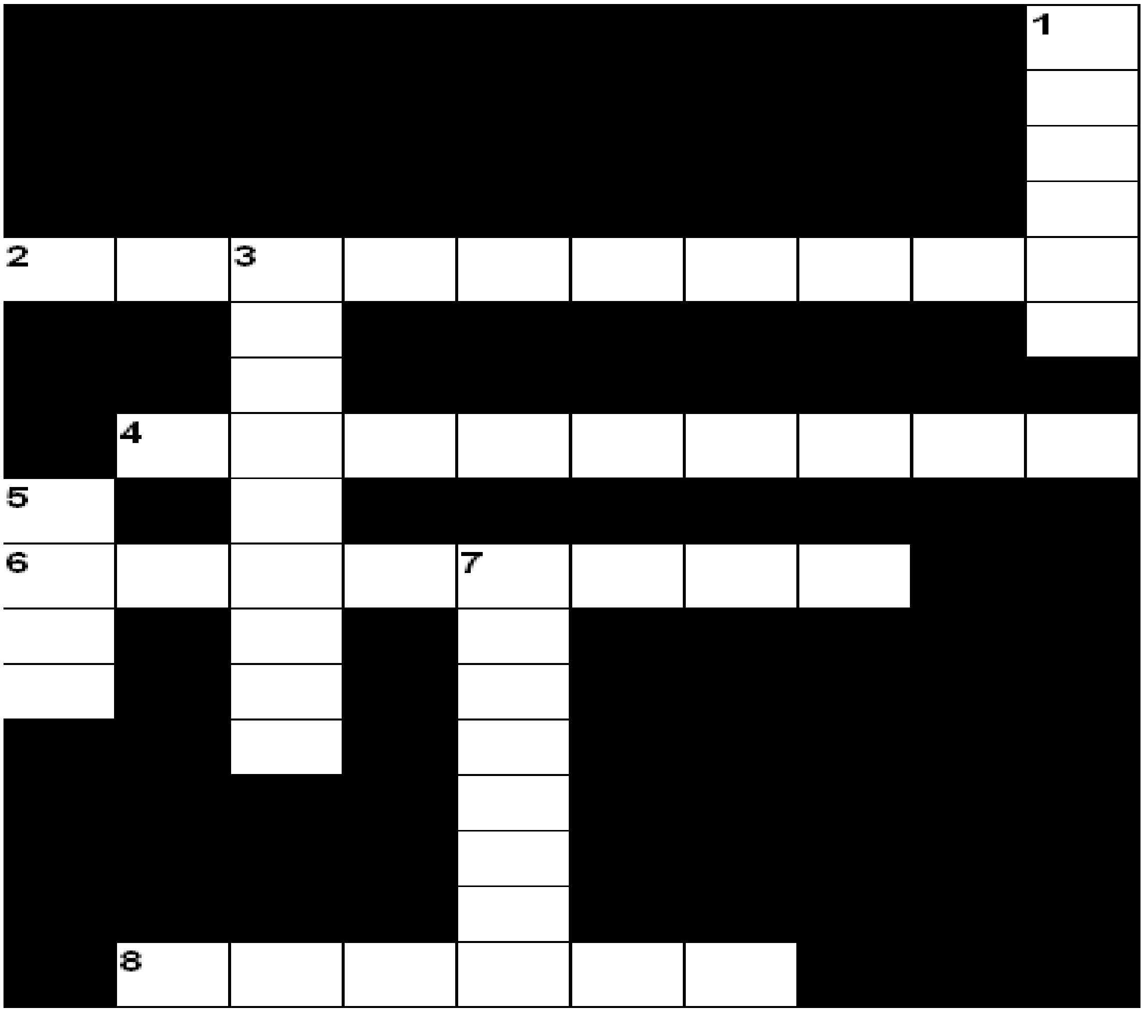


Fig 8-1

Across:

2. _______ is a process of converting one DataType or object type into another.(10)

4. Conversion of Value Types to _______ Types is Boxing.(9)

6. ________ conversion is a forced conversion and there may be a data loss.(8)

8. Namespace used by the .Net Framework library is ______. (6)

Down:

1. ________ is an implicit conversion.(6)

3. ________ is a way to keep one set of names separate from another.(9)

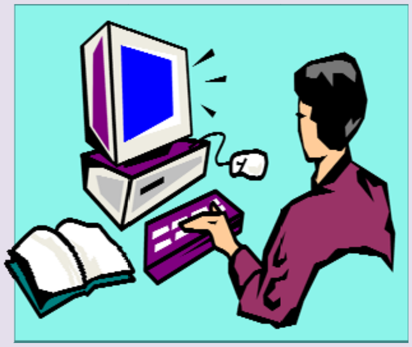
5. Reference types store a reference to the value's memory address, and are allocated on the _____. (4)

7. If one type of data is automatically converted into another type of data and there is no data loss then the conversion is called ___________ conversions.(8)

9.0 Creating and Destroying Objects

Topics

- 9.1 Constructors
- 9.2 Object and Memory
- 9.3 Resource Management
- 9.4 Crossword



Topic: Constructors

Estimated Time: 45 mins.

Objectives: After the completion of module, participant should know

- To Create a Constructor.
- Understand static and instance based Constructor.
- Implementing Constructor Overloading.

Presentation:

Constructor

- Constructor is a method of class, gets executed when instantiated object.
- It contains initialization
- It has the same name as its containing class
- It does not have a return type
- It can be Static or Instance based
    - Static Constructor
        - This is a special constructor and gets called before the class first object is created
        - Declaration: static <Class_Name>()
        - Only one static constructor in the class
        - It should be without parameters
        - It can only access the static members
        - No access modifier for defining
    - Instance Based Constructor
        - ✓ It gets called when class object is created
        - ✓ Declaration: public <Class_Name>()
- Constructor can be declared as public, private or protected

  public – Can be instantiated by any class
  private – Cannot be instantiated by any class
  protected – Can be instantiated by that class or derived class
- Constructor can take parameters

Default Constructor

- Every inbuilt class has default constructor and used to assign default values to class data members.
- It is invoked using new operator

  For Example: int myInt = new int();
- Some compilers automatically add a default constructor in class, however adding it explicitly makes code maintenance easier.

Constructor Overloading

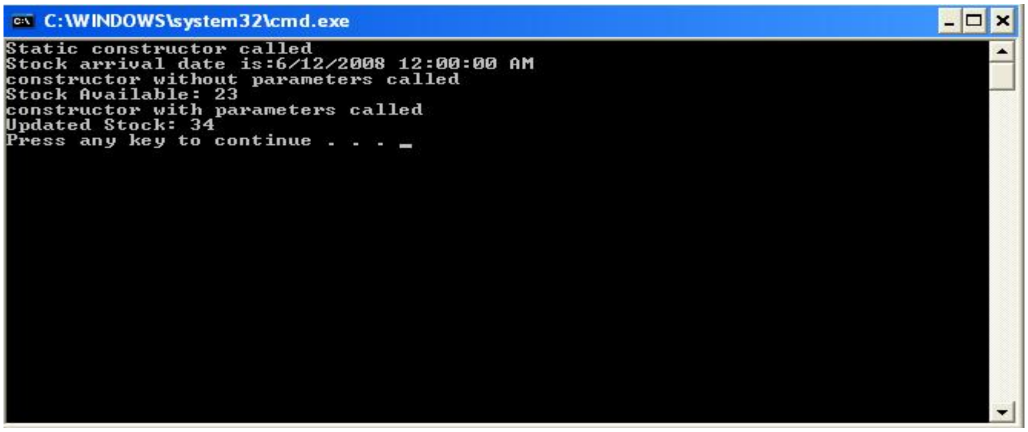
- Constructors with different set of parameters.
- One constructor can be called from another constructor.

Scenario :

Mr. Donald works for supermarket. He needs to update the available stock when new stock comes. So, build an application which display previous stock value and then show updated stock value.

Demonstration\Code Snippet :

[Note: Numbering represents the sequence in which the program flows]



```
Static constructor called
Stock arrival date is:6/12/2008 12:00:00 AM
constructor without parameters called
Stock Available: 23
constructor with parameters called
Updated Stock: 34
Press any key to continue . . . _
```

Figure 9.1-1: Output

using System;
using ...

Step 1: Declare a constructor with and without parameters and static constructor

```
public class StockInitialize
{
      public int stock;
       //static constructor
[1]   static StockInitialize()
      {
[2]         Console.WriteLine("Static constructor called");
            //Default constuctor of DateTime Class invoked
[3]         DateTime date = new DateTime();
[4]         Console.WriteLine("Stock arrival date is:" +
            DateTime.Today);
      }
      //constructor without parameters
[8]   public StockInitialize()
      {
[9]         stock = 23;
[10]        Console.WriteLine("constructor called");
      }
       //consturctor with parameters
       //constructor overloading
[13]  public StockInitialize(int s)
      {
[14]        stock = s;
[15]        Console.WriteLine("constructor called");
      }
}
```

Step 2:Initiate object using without and with parameter constructor

```
public class StockUpdate
{
[5]   static void Main(string[] args)
```

```
            {
[6]               try
                  {
[7]               StockInitialize objStockInitialize = new
                  StockInitialize ();
[11]              Console.WriteLine("Already available stock: " +
                  objStockInitialize.stock);
[12]              StockInitialize objStockInitialize1 = new
                  StockInitialize (34);
[16]              Console.WriteLine("Updated Stock: " +
                                    objStockInitialize1.stock);
                  catch (Exception ex)
                  {
                        Console.WriteLine(ex.Message);
                  }
            }
}
```

The class `StockInitialize` contains public data members. This is not a recommended programming practice because it allows any method anywhere in a program unrestricted and unverified access to an object's inner workings. Data members should generally be private, and should be accessed only through class methods and properties.

Context:

- To initialize members of class.
- Overload constructor, if there is more than one way of instantiating an object.

Practice Session:

- Extend above scenario by adding new functionality of calculating a total amount to be paid to supplier when new stock comes. Accept price of the product from user with a discount of 5% and pass it as an argument to constructor.

Check List:

- Use of Constructor
- Static Constructor and Instance Based Constructor
- Default Constructor
- Constructor Overloading [OR] Parameterized Constructors.

Common Errors:

- The Constructor and Class name may differ.
- Defining Return-type for constructors.
- Declaring Modifiers for static constructor.

Exception:

- Instantiating an object without parameter for the class defined with a parameterized constructor results in an error.

Lessons Learnt :

☑ Use static constructor to initialize default values before an object of the class is created.

☑ Avoid having default constructors on structures.

☑ Many compilers, including the C# compiler, do not support parameter less constructors on structures.

Best Practices:

- Use Constructor for default initialization purpose.
- Do minimal work in the constructor.
- Throw exceptions from instance constructors, if appropriate.
- Explicitly define a default constructor, if class supports it.

Topic: Resource Management for Objects

Estimated Time: 45 mins.

Objectives: After the completion of module, participant should know

- To destroy the object
- Memory management life cycle
- Working of garbage collector
- Use of Destructor

Presentation:

Memory Management

- C# employs automatic memory management that allocates and frees the memory occupied by objects.

- Memory Management Life Cycle:
  - When the object is created, memory is allocated for it.
  - If the object cannot be accessed by any execution then it is eligible for destruction.
  - After becoming eligible for collection. GC frees the memory associated with that object.

Garbage Collector

- It is a.Net Framework thread, which runs when required or when other threads are in suspended mode.

Lifetime of Object/Scope of Object/Object Clean Up

- Runtime environment reclaim memory when it is not being used:
  - For managed resource: Automatically GC releases memory
  - For Unmanaged Resources:
    - ✓ Object references resources outside .NET framework
    - ✓ Dispose method control explicitly
    - ✓ The using statement defines scope for an object lifetime
  - By using System.GC.Collect() method

Destructor

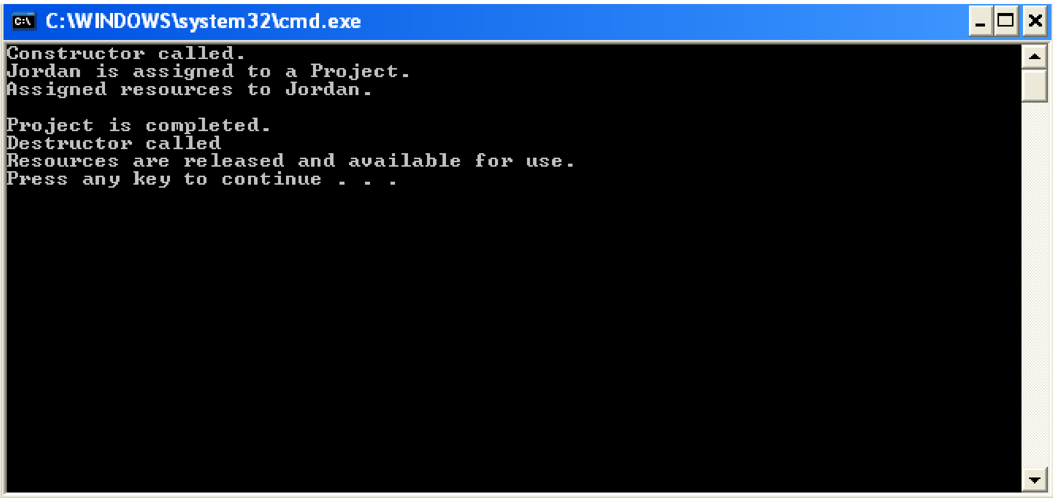
- It destructs instance of class when it is not used.
- It enables the runtime system to recover the heap space and terminate file I/O.
- Garbage Collector (GC) has the control over the destructor.
- They are invoked automatically.
- They cannot be used with structures.

Scenario:

Jordan is an employee of a software company. He is assigned to a project. The company allocates some resource for him. He stays on the project until he finishes it. After the completion of project he will be released and the resources allocated to him will be available for others.

Demonstration/Code Snippet :

[Note: Numbering represents the sequence in which the program flows]



```
Constructor called.
Jordan is assigned to a Project.
Assigned resources to Jordan.

Project is completed.
Destructor called
Resources are released and available for use.
Press any key to continue . . .
```

Figure 9.2&9.3-1:Output

using System;
using ...

Step 1: Declare a destructor in a class

```
public class resource
{
[3]   public resource()
      {
[4]         Console.WriteLine("Constructor called.");
[5]         Console.WriteLine("Jordan is assigned to a project");
      }
      ~resource()
      {
            Console.WriteLine("Destructor called");
            Console.WriteLine("Resources are released and
            available for use.");
      }
[7]   public void getResource() {
[8]         Console.WriteLine("Assigned resources to Jordan.");}
[10]  public void projectComplete() {
[11]        Console.WriteLine("Project is completed.");
      }
}
```

Step 2:Instantiate object and allocate resources using getResource() method

```
public class resourceUpdate
{
[1]   static void Main(string[] args)
      {
[2]         resource objResource = new resource();
[6]         objResource.getResource();
```

Step 3:Release resources using projectComplete() method

```
[9]         objResource.projectComplete();
      }
}
```

In this scenario destructor will call when Main() method execution completes means object goes out of scope. Then, garbage collector call and it call finalize method to free the memory space.

Context:

- Use destructor to release unused resources.
- Use GC to free the memory space.

Practice Session:

- Identify the ways to release unmanaged resources.
- Differentiate between Finalize() and Dispose() method.
- In above scenario release resources using Dispose() method.

Check List:

- Memory Management Life Cycle
- Working of GC and Destructor
- Use of dispose method

Common Errors:

- Syntax Errors.
- Trying to release the object which is already released.
- Defining return type in destructor.
- Calling Dispose() without implementing Idisposable interface results in error like: *expected enum, delegate, interface, struct.*

Exception :

- Imagine a server component which uses a database connection and does not have a **Dispose ()** method. Here, garbage collection might not destroy the components for some time after the references to them are released. So new connection can't be established and unable to respond to the user request even though enough memory space is available.

Lessons Learnt :

☑ Use Dispose() method if resources are unmanaged.
☑ When released, resources using Dispose() method, provide implicit cleanup using the Finalize().
☑ Destructor control is done by GC.
☑ Destructor cannot be overloaded and inherited and cannot be used with structure.

Best Practices:

- Use Destructor to free the memory space.
- Use System.GC.Collect() method to release resources directly.

Crossword: Unit-9 **Estimated Time: 10 mins.**

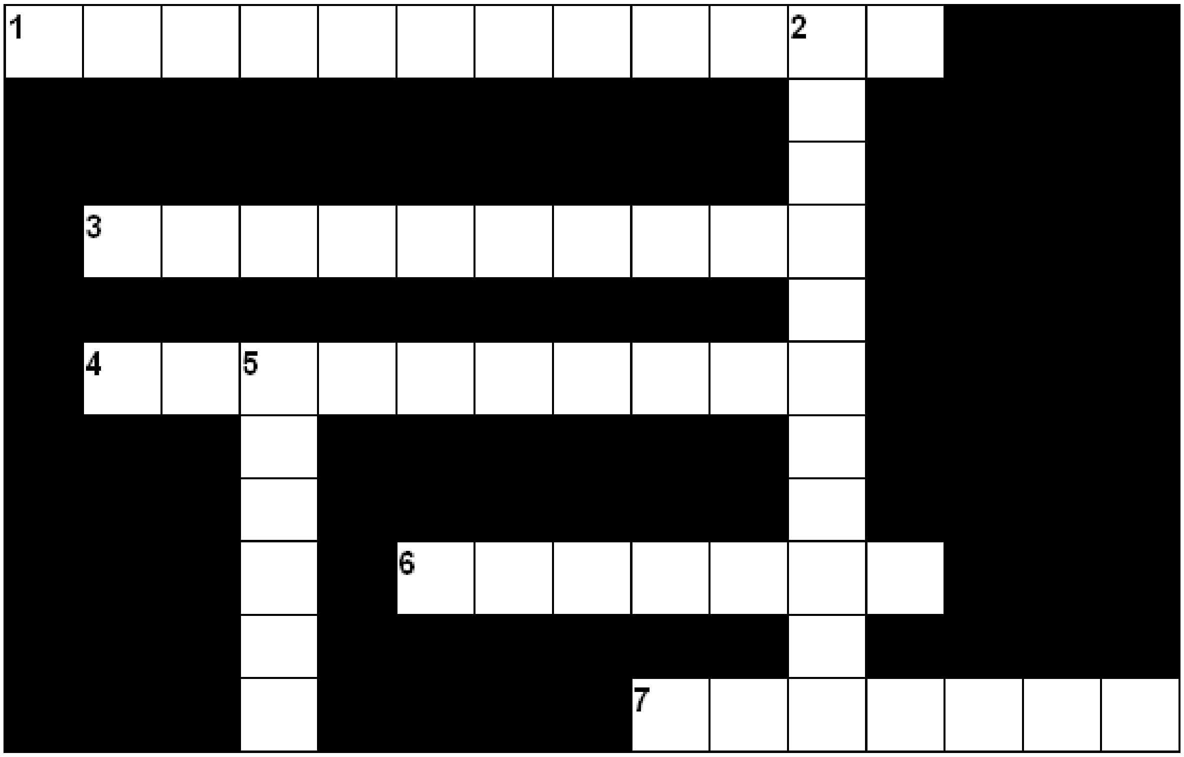


Fig. 9-1

Across:

1. _________ can be made easier by calling constructors, instead of calling them automatically as is the norm in most compilers.(12)

3. Destructors cannot be used with _________.(10)

4. It destructs instance of class when the instance is not useful. ___________ .(10)

6. Constructor can be declared as public, _________ or protected.(7)

7. C# employs automatic memory management that allocates and frees the memory occupied by objects. For the purpose it uses __________ collector.(7)

Down:

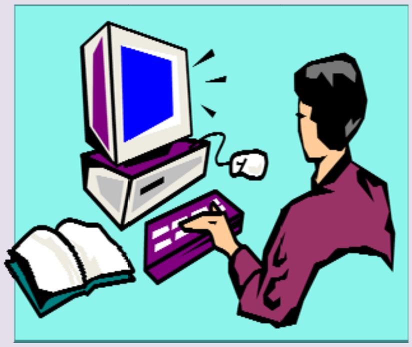
2. ____________ has the same name as its initializing class.(11)

5. Constructor can be ______ or instance based.(6)

10.0 Inheritance

Topics

- 10.1 Deriving Classes
- 10.2 Using Interfaces and Abstract Classes
- 10.3 Structured Exception Handling
- 10.4 Crossword



Topic: Derived Classes

Estimated Time: 30 mins.

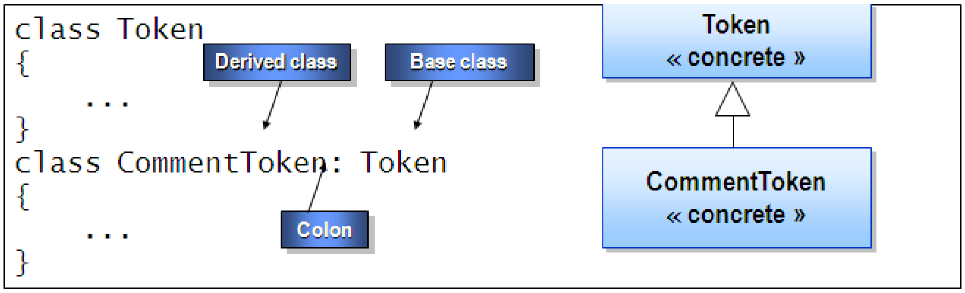
Objectives: After the completion of module, participant should know:

- Base class and access its members
- Use of Derived Class

Presentation:

Base Class and Derived Class

- In OOPs Concepts, a parent class can inherit its behavior and state to children classes.
- Inheritance is the process of creating new classes from Base class .The inheritance feature reuses some parts of code.
- **Derived classes**, inherit attribute and behavior of the **Base Classes.**
- Derived class member can be accessed using object of its own class while base class members other than private can be accessed using object of derived class as well as its own class.



```
class Token
{
    ...
}
class CommentToken: Token
{
    ...
}
```

Figure 10.1-1: Base Class and Derived Class

Implementing Methods:

- Method signature in C#
  <return_type> methodname (Parameterslist) { ….}

Scenario:

Mr. Alex is working for finance department of McAfee. He has to calculate salary of employees at the end of the month. For that he needs employee’s personal information and number of working days. This information is provided by HR department of McAfee.

Demonstration/Code Snippet:

[Note: Numbering represents the sequence in which the program flows]

```
using System;
```

Step 1: Create a base class with employee personal information

```
      public class deptHR
      {
            public int empID;
            public string empName;
            public int noDays;
[7]         public deptHR(int id, string name, int days){
[8]                     empID = id;
[9]                     empName = name;
[10]                    noDays = days; }
      }
```

Step 2: Create a derived class that inherits base class using (: base class name)

```
      public class deptFinance: deptHR
      {
            public int empSalary;
[5]         public deptFinance(int salary,int id,string name,int
            days):base(id,name,days)
            {
[6]               empSalary = salary;
            }
[12]         public void GetInfo()
            {
[13]        Console.WriteLine("Employee id is: {0}", empID);
[14]        Console.WriteLine("Employee name is: {0}", empName);
[15]        Console.WriteLine("Employee working day is: {0}",
                        noDays);
            }
      }
```

Step3: Access the members of base and derived class in the main method

```
       public class Test
      {
[1]         public static void Main()
            {
[2]               try{
[3]               int Salary;
[4]               deptFinance finance = new
                  deptFinance(12000,6,"Sam",5);
                  //creating an object of class
[11]              finance.GetInfo();
[16]              Salary = finance.noDays * finance.empSalary;
[17]              Console.WriteLine("Employee salary is: {0}",
                                    Salary);
            }
            catch (Exception ex){
                   Console.WriteLine(ex.Message); }
            }
       }
```

Context:

- To reuse the base class code in the inherited classes.
- Derived class inherits the attributes and behavior of base class.

Practice Session:

- In the above scenario take one private variable designation in class deptHR and apply any of the encapsulation method to access it in GetInfo() method where it should display designation on the console.

Checklist:

- Inheriting derived class from the base class.
- Accessing base class and derived class members.

Common Errors:

- Accessing private variable of base class in derived class.
- Accessing derived class members using base class object.

Exception:

- ‘sealed’ keyword prevents inheritance from occurring.

Lessons Learnt:

☑ Any public members that you add to a derived class can be accessed only by using a reference of the derived class type.

Best Practices:

- By using the concept of inheritance, it is possible to create a new class from an existing one and add new features to it. Thus inheritance provides a mechanism for class level reusability.

Topic: Interface and Abstract Class

Estimated Time: 60 mins.

Objectives : After the completion of module, participant should know

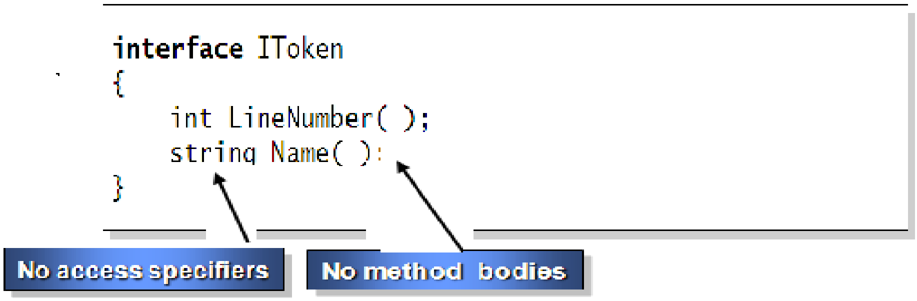
- Implementing Interface
- Multiple Inheritance
- Abstract Class Implementation

Presentation :

Interface Methods

- An **Interface** is a *reference type* and it *contains only abstract members*
- An interface has no implementation; it only has the definition of the methods without the body.
- **Interface**'s members can be **Events**, **Methods**, **Properties** and **Indexers.**

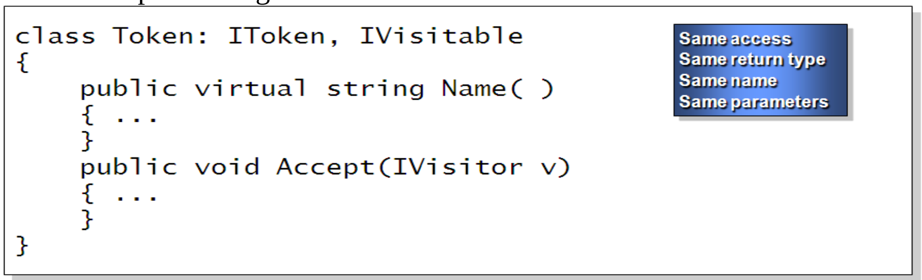
- All the member declarations inside interface are implicitly public



```
interface IToken
{
    int LineNumber( );
    string Name( );
}
```

Figure 10.3-1: Declaring Interface

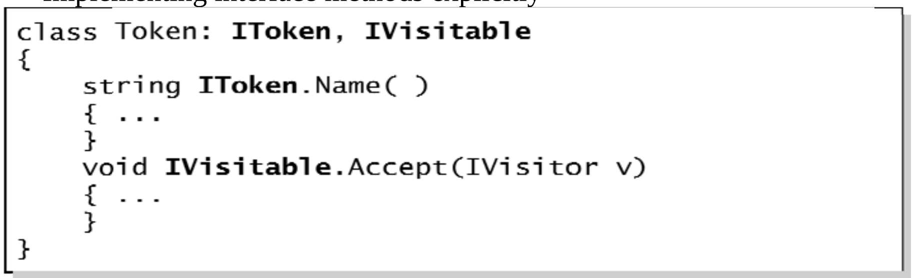
- **Implementing Interface Method**
  - ➢ The Implementing Methods must be same as the Interface Method
  - ➢ The Implementing Method can be virtual or Non-virtual



```
class Token: IToken, IVisitable
{
    public virtual string Name( )
    { ...
    }
    public void Accept(IVisitor v)
    { ...
    }
}
```

Figure 10.3-2: Implementing Interface

- ➢ Implementing interface methods explicitly

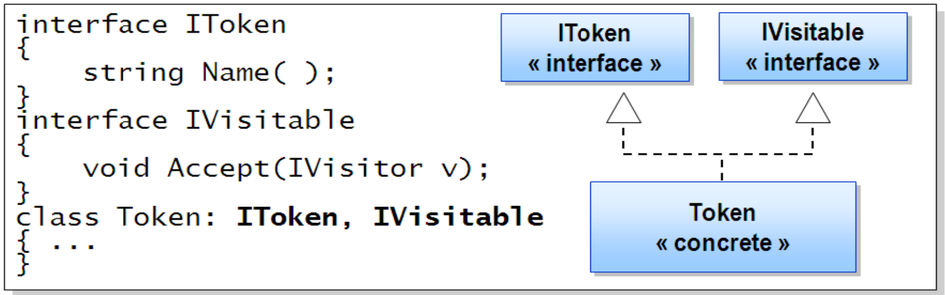


```
class Token: IToken, IVisitable
{
    string IToken.Name( )
    { ...
    }
    void IVisitable.Accept(IVisitor v)
    { ...
    }
}
```

Figure 10.3-3: Implementing Interface Explicitly

Multiple Interfaces

- A class can implement zero or multiple interface



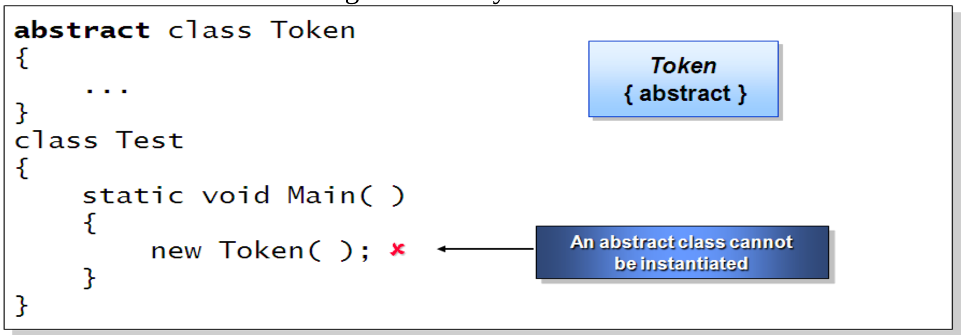
```
interface IToken
{
    string Name( );
}
interface IVisitable
{
    void Accept(IVisitor v);
}
class Token: IToken, IVisitable
{ ...
}
```

Figure 10.3-4: Implementing Multiple Interface

- A class must implement all inherited interface methods.

Declaring Abstract Class

- Abstract class is declared using abstract keyword



```
abstract class Token
{
    ...
}
class Test
{
    static void Main( )
    {
        new Token( ); ×
    }
}
```

Token
{ abstract }

An abstract class cannot be instantiated

Figure 10.3-5: Declaring Abstract Class

Comparison between Interface and Abstract Class

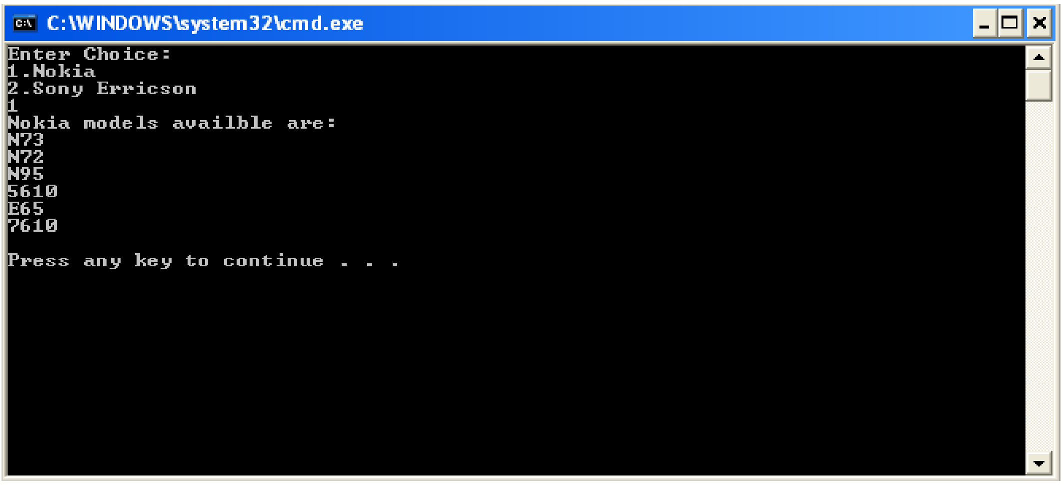
- **Similarities**
  - ➢ Neither can be instantiated.
  - ➢ Neither can be sealed.
- **Differences**
  - ➢ Interfaces cannot contain any implementation.
  - ➢ Interfaces cannot declare non-public members.
  - ➢ Interfaces cannot extend non-interfaces.

Scenario :

Mr. Austin has a mobile shop in Newyork city. He requires software which should facilitate to find which models of mobiles are present in the shop.

Demonstration/Code Snippet :

[Note: Numbering represents the sequence in which the program flows]



```
Enter Choice:
1.Nokia
2.Sony Erricson
1
Nokia models availble are:
N73
N72
N95
5610
E65
7610

Press any key to continue . . .
```

Figure 10.3-5: Output

Step 1: Create an interface with one method

```
using System;

namespace ConsoleApplication
{
    interface iMobile
    {
       void models_available();
    }
```

Step 2: Create two classes which implements interface and provide definition for interface method

```
           class Nokia : iMobile
           {
[10]        public void models_available() {
[11]        StringBuilder str = new
            StringBuilder("N73\nN72\nN95\n5610\nE65\n7610\n");

[12]       Console.WriteLine("Nokia models availble are:");
[13]       Console.WriteLine(str);
        }
    }
    class Sonyerricson : iMobile
    {
        public void models_available()
        {
            StringBuilder str = new
            StringBuilder("K750\nK700\nW800\nW900\nJ230i\nW550\n"
            );

            Console.WriteLine("Sony Erricsom models available
            are:");
           Console.WriteLine(str);
        }
     }
```

Step 3: In Main(), based upon condition call the respective methods.

```
    class Program
    {
[1]        static void Main(string[] args)
           {
[2]                try{
[3]                Nokia obj_Nokia = new Nokia();
[4]                Sonyerricson obj_sony = new Sonyerricson();

[5]                Console.WriteLine("Enter Choice:");
[6]                Console.WriteLine("1.Nokia\n2.Sony Erricson");
[7]                int choice = int.Parse(Console.ReadLine());

                  //runtime deciding which method to call
[8]               if (choice == 1)
[9]                    obj_Nokia.models_available();
                  else if (choice == 2)
                    obj_sony.models_available();
```

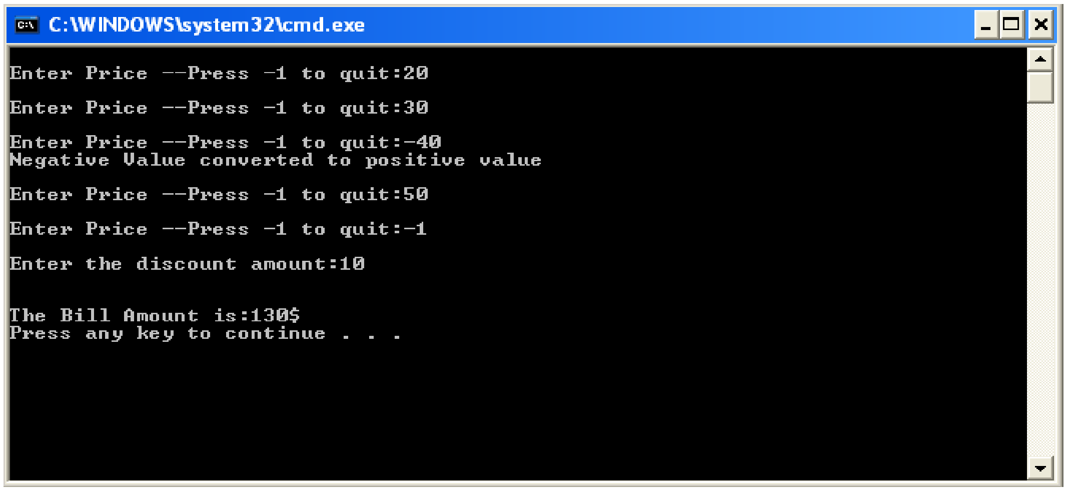
```
                         else
                           Console.WriteLine("Invalid Choice");
[14]                      }
                         catch (Exception ex){
                              Console.WriteLine(ex.Message); }
                }
            }
        }
```

Scenario: (Usage of Abstract Class)

Mr. Smith runs departmental store. He requires an application which calculates the bill along with discount.

Demonstration/Code Snippet:

[Note: Numbering represents the sequence in which the program flows]



```
Enter Price --Press -1 to quit:20

Enter Price --Press -1 to quit:30

Enter Price --Press -1 to quit:-40
Negative Value converted to positive value

Enter Price --Press -1 to quit:50

Enter Price --Press -1 to quit:-1

Enter the discount amount:10


The Bill Amount is:130$
Press any key to continue . . .
```

Figure 10.3-6: Output

Step 1: Create an abstract class which defines one method and declares one method as abstract

```
using System;

namespace ConsoleApplication8
{
    public abstract class Bill
    {
        public int check(int n)
        {
[11*]       if (n < 0)
            {
            Console.WriteLine("Negative Value converted to
                                   positive value");
            return -n;
            }
            else
              return n;
        }

        public abstract int calculate(int n, int m);
    }
```

Step 2: Create two classes that inherits the abstract class and defines the abstract class

```
     public class Addition : Bill
     {
[13]           public override int calculate(int n, int m)
[14]           {
[15]               return n + m;
               }
     }

     public class Discount : Bill
     {
[19]        public override int calculate(int n, int m)
        {
[20]            return n - m;
        }
     }
```

Step 1: In Main() method, call calculate() method with respective of class object

```
        class Program
        {
 [1]        static void Main(string[] args)
            {
 [2]            try{
 [3]            Addition obj_add = new Addition();
 [4]            Discount obj_disc=new Discount();
 [5]            int num1=0, num2;

 [6]           while (true) {
 [7]            Console.Write("\nEnter Price [Press -1 to
                             Quit]:");
  [8]             num2 = int.Parse(Console.ReadLine());
  [9]             if (num2 != -1)
                  {
  [10]                num2 = obj_add.check(num2);
  [12]                num1 = obj_add.calculate(num1, num2);
                      //calculate() of Addition class is called
               }
                 else
                 break;
            }

  [16]          Console.Write("\nEnter the discount amount:");
  [17]          num2 = int.Parse(Console.ReadLine());
  [18]          num1=obj_disc.calculate(num1, num2);
            //calculate() of Discount class is called


  [21]      Console.WriteLine("\n\nThe Bill Amount is:"+
                             num1.ToString()+"$");
            }
            }
            catch (Exception ex){
                Console.WriteLine(ex.Message);}
        }
```

Context :

- Use interface, when need to hide the processing of complex method definition in an application.
- Use abstract class when you do not want to create an object of class.

Practice Session:

- Fix the bugs

```
1.  class Base
    {
        public void Alpha( ) { ... }
        public virtual void Beta( ) { ... }
    }
    class Derived: Base
    {
        public override void Alpha( ) { ... }
        protected override void Beta( ) { ... }
    }
2.  interface IToken
    {
        string Name( );
        int LineNumber( ) { return 42; }
        string name;
    }
    class Token
    {
        string IToken.Name( ) { ... }
        static void Main( )
        {
                        IToken t = new IToken( );
        }
}
```

Check List:

- Implementing Interface
- Implementing Multiple Interfaces
- Implementing Abstract Class

Common Errors:

- Providing access specifier for interface members.
- Providing access specifier other than public when implementing interface methods.
- Implementing method in interface.
- Creating instance of abstract class.

Exception:

- No security permissions can be attached to members or to the interface itself.

Lessons Learnt :

☑ Cannot declare virtual methods as ‘static’ and ‘private’.
☑ Only override identical inherited virtual methods.
☑ A class must implement all inherited interface methods.

Best Practices:

- In concept of inheritance, it is possible to create a new class from an existing one and add new features to it. Thus, inheritance provides a mechanism for class level re-usability.
- Use abstract class because it enforces certain hierarchies for all the subclasses.

Topic: Structured Exception Handling **Estimated Time: 35 mins.**

Objectives: After the completion of module, participant should know

- Distinction between Application-level and System-level Exceptions.
- Role of System.Exception class.
- Building Custom Exception.

Presentation:

System Level Exception:

- Are thrown by .NET platform.
- Non recoverable, fatal errors.
- Derived from System.SystemException namespace.
- Example: NullReferenceException, DivideByZeroException, OverflowException, IndexOutofBoundException, FormatException etc

Application Level Exception

- Are thrown by user program.
- Derived from System.ApplicationException.

Steps to create Custom Exception

1. Derive class from System.ApplicationException.
2. Define Constructor.
3. Override Exception. Message property.
4. Throw custom exception.
5. In Main() method, catch exception using try and catch block.

Scenario :

Mr. Henry working for a software company, got a new project. To interact with client, he requires a valid email address of the person.

Demonstration\Code Snippet :

[Note: Numbering represents the sequence in which the program flows]

```
using System;
```

Step 1: Declare a class which inherits Application Exception class

```
public class TextException : ApplicationException
```

```
{
      private string messageDetails;
      //Define constructor
      public TextException(){}
[8]   public TextException(string message)
      {
[9]         messageDetails=message;
      }
      //override the Exception.Message property
[10]  public override string Message
      {
[11]        get {
[12]            return string.Format("Error Message:{0}"
                ,messageDetails); }
      }
}
```

Step 2: Declare a class with TestEnteredMail which find whether @symbol is there or not and mailaddress is given or not

```
public class MailValidator
{
        private static char symbol = '@';
[5]     public static void TestEnteredMail(string mailAddress)
        {
[*6]        if (mailAddress == "")
[7]               throw new NullReferenceException("Enter mail
                  address");
[*6]        else if (mailAddress.IndexOf(symbol)==-1)
[7]               throw new TextException(string.Format("The
                  string entered is not a valid email address"));
        }
[1]     static void Main(string[] args)
        {
[2]         try
            {
[3]                Console.WriteLine("Enter email address:");
[4]                MailValidator.TestEnteredMail
                   (Console.ReadLine());
[*8]               Console.WriteLine("Valid email address:");
            }
            //Using application level exception class
[13]        catch (TextException e)
            {
[14]               Console.WriteLine(e.Message);
            }
            //Using system level exception class
            catch(NullReferenceException e)
            {
                   Console.WriteLine(e.Message);
            }
            //catch any other exception
            catch (Exception e)
            {
                   Console.WriteLine(e.Message);
            }
```

```
          }
      }
```

.

String. Format method used to format numeric results.

String.IndexOf method returns index position of value if that particular character is found or 1 if it is not.

Context:

- Need to create Custom Exception when error is tightly bound to class issuing the error.

Practice Session:

- In above scenario, extend the method TestEnteredMail () by checking whether @ is present or not. If not, generate a custom exception and handle the exception.

Check List:

- System Level Exception and Application Level Exception.
- Building of Custom Exception.
- Handling multiple exceptions.

Common Errors:

- Custom Exception classes should be defined as public type because exceptions are passed outside of assembly boundaries which should be accessible to calling code base.

Exception:

- Allows exceptions to be thrown across process and even machine boundaries.

Lessons Learnt:

- Do create and throw custom exceptions if you have an error condition that can be programmatically handled in a different way than any other existing exceptions. Otherwise, throw one of the existing exceptions.

Best Practices:

- Best to use application specific exception for creating custom exception rather than system level exception.
- Avoid deep exception hierarchies.
- Make exceptions serializable.

Crossword: Unit-10 **Estimated Time: 10 mins.**

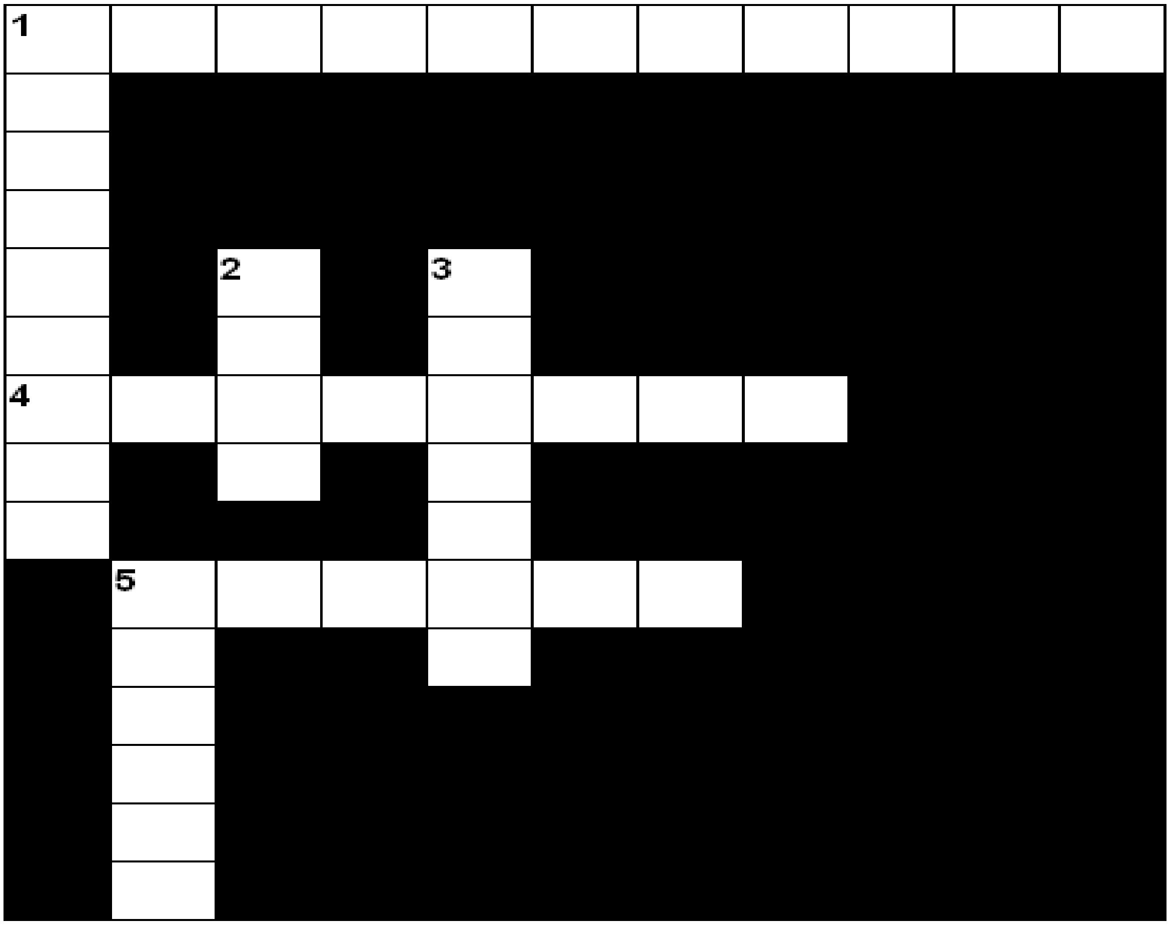


Fig. 10-1

Across:

1. __________ is the process of creating new classes from Base class.(11)

4. An Interface is a reference type and it contains only _________ members.(8)

5. In OOP, a ________ class can inherit its behavior and state to children classes.(6)

Down:

1. The implementing methods for implementing an interface must be same as the _________ method.(9)

2. Derived classes, inherit attribute and behavior of the ____ classes.(4)

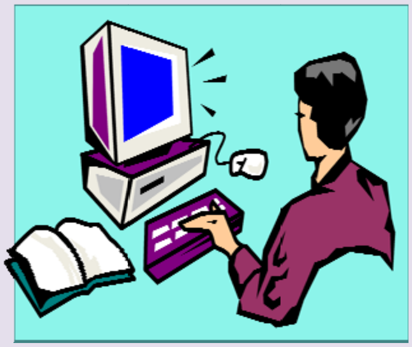
3. ____________ class member can be accessed using object of its own class while base class members other than private can be accessed using object of derived class as well as its own class.(7)

5. All the member declarations inside interface are implicitly __________. (6)

11.0 Aggregation, Namespaces and Advanced Scopes

Topics

- 11.1 Aggregation
- 11.2 Modules and Assemblies
- 11.3 Collections and Generics
- 11.4 Crossword



Topic: Aggregation

Estimated Time: 30 mins.

Objectives: After the completion of module, participant should know

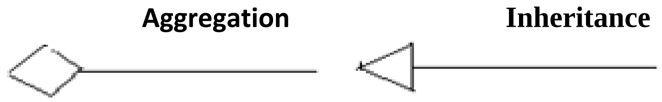
- Concept of Aggregation.
- Difference between Aggregation and Inheritance.

Presentation:

- Aggregation is a new abstraction which turns a relationship between objects into an aggregate object.
- It put objects together to build larger objects and to control the resulting behavior of these complex objects.
- It is used as an object structuring mechanism i.e. Objects within object
  - ➢ Complex objects are built from simpler objects.
  - ➢ Simpler objects are parts of complex whole objects.
  - ➢ This is called aggregation.

Aggregation Vs Inheritance

Aggregation	Inheritance
Object relationship	Class relationship
Weak whole to part dependency	Strong derived to base dependency
Dynamically flexible	Statically inflexible



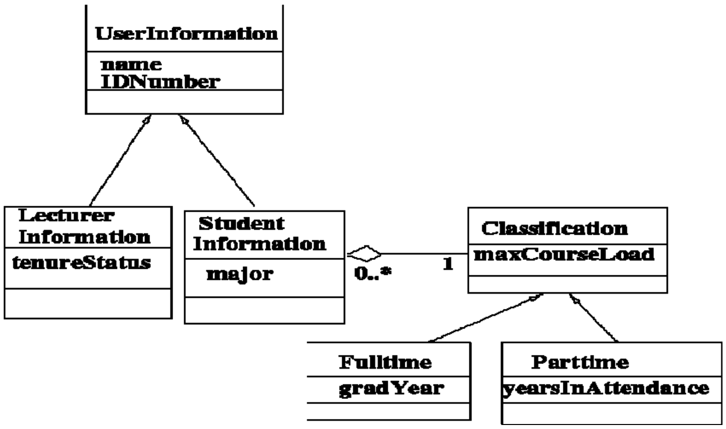
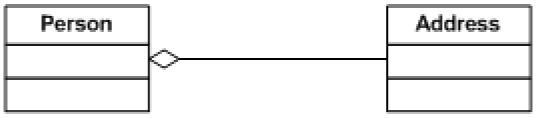


Figure 11.1-1: Example showing Aggregation and Inheritance

Scenario:

The CRM system has a database of customers and a separate database that holds all addresses within a geographic area. As the customer does not manage the lifetime of address, it exists even though the customers are no more.



Aggregation

Demonstration\Code Snippet :

Step 1: Make a class

```
public class Address
{
      String Address
}
```

Step 2: Make another class named Person

```
public class Person
{
      private Address address;
      public Person(Address address)
      {
            this.address = address;
      }
}
```

Step 3: Person would then be used as follows:

```
static void Main(string[] args)
{
      Address address = new Address();
      Person person = new Person(address);
      //or
      Person person = new Person( new Address() );
}
```

Context:

- To know better the use of ‘part of’ and ‘has a’ relationship.

Practice Session:

- An engine is part of a car and Car model engine ENG01 is part of a car model CM01. Find out the appropriate relationship for the above two representation.

- **We have two relationships**
  - ➢ University and Professors
  - ➢ University and Departments

Check List:

- Difference between Aggregation and Inheritance.

Common Errors:

- Use Aggregation as a Composition.
- An Aggregation may not involve into more than two classes.

Exception:

- In aggregation relationship, it is possible to have a single object that is partly implemented in managed code (the inner portion) and partly in unmanaged code (the outer portion).

Lessons Learnt:

☑ Aggregation is different from composition and inheritance.
☑ Aggregation is of ‘has a’ type.
☑ In object modeling, it should be a matter of 'part-of' or 'have-a'.

Best Practices :

- Clarify the relationship whether it is of ‘part-of’ or ‘has-a’ type.
- Aggregation should be used whenever their exist a dependency between objects.

Topic: Assemblies

Estimated Time: 60 mins.

Objectives:

After the completion of module, participant should know

- Concept Of Assembly and Manifest
- Creating an Assembly
- Concepts of Dynamic Linking
- Global Assembly Cache

Presentation:

- The entire .NET code on compilation gets converted into an Intermediate Language (IL) code and gets stored as an Assembly.
- An Assembly is a self-describing unit of reuse, versioning, security and deployment.
- There are two types of assembly **Private Assembly** and **Public Assembly.**
- A Private Assembly deployed with application, and stored in the application's directory. It is used by only one application.
- A shared assembly is stored in the global assembly cache (GAC), maintained by the .NET runtime. It is shared among different application.
- One more type of assemblies called **satellite assemblies** are used for localization. (Partition the resources based on a specific culture)
- **Assembly Manifest**
  - ➢ Assembly Manifest contains the metadata for the assembly. It contains :
    - ✓ Name and Version Information
    - ✓ Files that are made up of the assembly
    - ✓ Set of permissions required for the assembly to run properly
    - ✓ Other details depends on assembly
- Assembly avoids DLL HELL problem

Demonstration\Code Snippet:

Create an Assembly

Step1: Choose File/New Project
Choose Visual C# from project type
Choose Class Library from templates
Choose location as C:\SSIT
Provide File name as “MyFirstLibrary”
Choose Submit Button
Step2: A file with Class1.cs gets created
Step3: Provide a function as follows as part of the class “Class1”

```
public int square(int x)
{
      return x * x;
}
```

The class1.cs appears as follows:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace MyLibrary
{
public class Class1
{
      public int square(int x)
      {
            return x * x;
      }
}
```

Step 4: Choose Visual Studio 2005 command prompt from the start menu as follows
Start/all programs/Microsoft Visual studio 2005/visual studio tools/visual studio 2005 command prompt

Step 5: Change the drive and directory to the root directory of c: drive, as follows

```
C:\.........> cd \
```

Step 6: Create a strong name key with an extension of. snk (For example: one.snk)

```
C:\> sn –k  one.snk
```

Step 7: Open AssemblyInfo.cs from the solution explorer and add the following lines

```
[assembly:AssemblyKeyFile("c://one.snk")]
```

Step 8: Choose Build/Build solution, to create a .dll (an assembly) in the debug/bin folder of the current application folder

```
C:\........\MyLibrary\Debug\bin
```

Step 9: The assembly is placed into Global Assembly Cache and made as a shared library

```
C:\>Gacutil   /i   c:\......\debug\bin\MyLibrary.dll
```

Build application using the assembly

Step 1: Choose Windows Applications from the Visual Studio

Step 2: Choose Project/Add Reference/Browse to select the assembly (MyLibrary.dll) from
C:\.......\MyLibrary\Debug\bin folder

Step 3: Go to the design view, place a button and associate the following code in the Button_Click() event in code behind

```
MyLibrary.Class1  ob = new MyLibrary.Class1();
MessageBox.Show(ob.square(10).ToString());
```

Context:

- ➢ To inform whether all the types defined in the assembly are visible to other assemblies or private to one.
- ➢ A hash of all files in the assembly and details of any security permissions that clients need to have in order to be able to run the assembly.

Practice Session:

- Assign strong name to assembly.
- Describe the functions performed by the assembly manifest.
- What is GAC? And describe- the ways to deploy an assembly into the GAC.
- Create assemblies
    - ➢ Using development tools, such as Visual Studio 2005
    - ➢ Using tools provided in the .NET Framework SDK
    - ➢ Using common language runtime APIs

Check List:

- Assembly types

- Assembly Manifest
- Create an Assembly
- Application using Assembly

Common Errors:

- Forgot to add library reference of private assembly.
- Each assembly can have only one entry point (that is, **DllMain** or **WinMain** or **Main**)

Exception:

- System not having administrative credentials.

Lessons Learnt:

☑ Creating a strong name
☑ Building an Assembly
☑ Building the keys

Best Practices:

- Specify a set of permissions that is required to run.
- Simplify application deployment.
- Solve versioning problems that can occur with component-based applications.
- For side-by-side execution that gives more control over versions of a component and the version of the runtime an application uses.

Topic: Collections and Generics

Estimated Time: 45 mins.

Objectives: After the completion of module, participant should know

- Basic Collection Classes and Interfaces.
- Defining and Implementing Generic Base Classes and Interfaces.
- Implementing Custom Generic Methods, Structures and Classes.

Presentation:

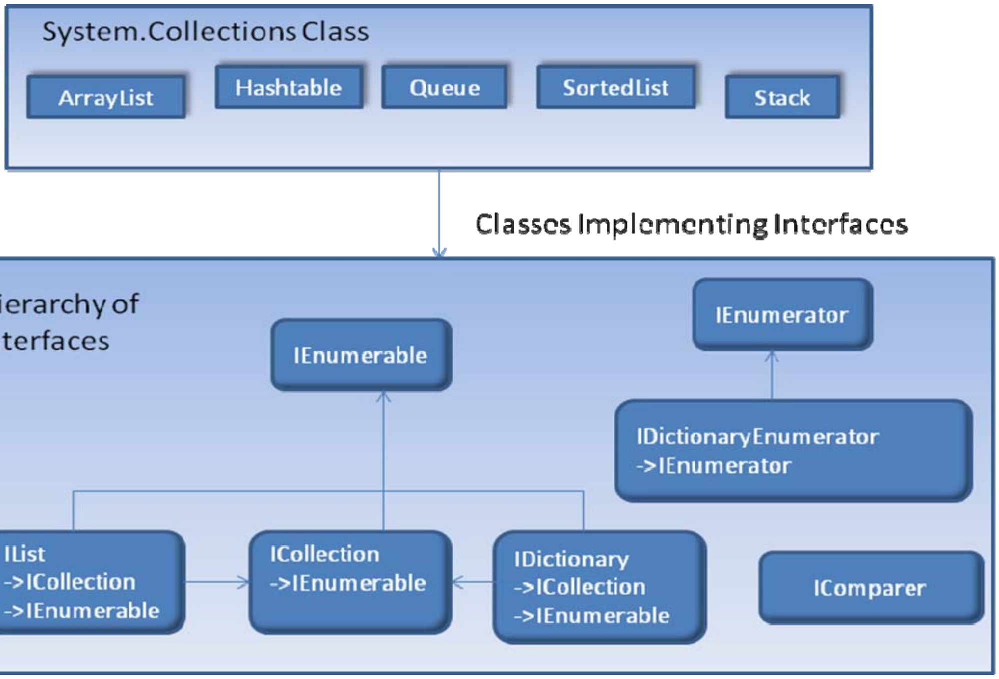

System.Collections Class
ArrayList
Hashtable
Queue
SortedList
Stack
Classes Implementing Interfaces
Hierarchy of Interfaces
IEnumerable
IEnumerator
IDictionaryEnumerator ->IEnumerator
IList ->ICollection ->IEnumerable
ICollection ->IEnumerable
IDictionary ->ICollection ->IEnumerable
IComparer

Figure 11.3-1: Collection Classes and Interfaces

Type Safety and Strongly Typed Collections

- Majority of the type of System. Collections can typically hold anything.
- Most of the times *type-safe* container is recommended to operate on a particular type of data point.
  For Example: A specific container to hold only database connections or bitmaps
- Custom Collection or Strongly Typed Collections is built to make a *type-safe* container.

> While Custom Collections ensure type-safety, it needs to be created for each type required.
> It is better to use for System.Collections.Generic Namespace.

System.Collections.Generic Namespace

System.Collections.Generic defines number of classes that implement many interfaces.

Non-Generic Class	Non-Generic Class in System.Collections
Collection<T>	CollectionBase
Comparer<T>	Comparer
Dictionary<Tkey,Tvalue>	Hashtable
List<T>	ArrayList
Queue<T>	Queue
SortedDictionary<Tkey, TValue>	SortedList
Stack<T>	Stack
LinkedList<T>	N/A

ReadOnlyCollection<T>	ReadOnlyCollectionBase

> T- Represent Types, Tkey - Represent Keys, TValues -Represent Values

Custom Generic Methods

- These methods can operate on any possible data-type (value-based or reference-based) using a single parameter type.
  Example: At the time of specification: static void Swap<T> (ref T a, ref T b)
  At the time of invocation: Swap<int> (ref a, ref b).//Pass by reference.
  Here T represents int type.

Custom Generic Structures and Classes

- The process of building generic structure and classes are identical
  Example: Consider Point<T> types as follows:

```
//Point using ints.
Point<int> p=new Point<int>(10,10);
//Generic point structure.
public struct Point<T>
{
        private T xPos; //Generic state date.
        private T yPos;
        ….
        public void ResetPoint()
        {
                xPos=default(T);
                yPos=default(T)
        }
}
```

> “default” keyword is overloaded in C#. Represents the default value of a type parameter.
> Fields of structure are set *0* for value types and *null* for reference types.

Generic Base Classes

- Generic classes can be the base class to other classes, thus, it can define any number of virtual or abstract methods.
- Derived methods must substitute the type parameters used in the parent class.

```
Example: //Custom Generic List Class with a virtual method.
public class MyList<T>
{
        private List<T>  listofData=new List<T>();
        public virtual void PrintList(T data){}
}
//Deriving from a Generic Base Class
public class MyStringList : MyList<string>
{
        public override void PrintList(string data){ }
}
```

Generic Interfaces

- Custom Generic Interfaces can be defined according to the need

Ex: //Defining Interfaces

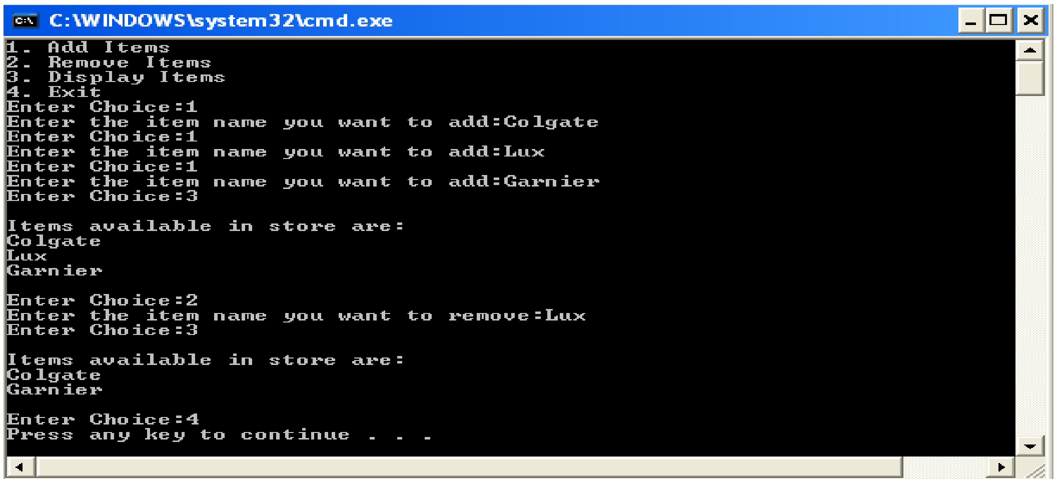
```
 public interface IBinaryOperations<T> where T: struct
{
      T Add(T arg1, T arg2);
}
// Implementing Interfaces.
public class BasicMath : IBinaryOperations<int>
{
      public int Add(int arg1, int arg2)
      {
            return  arg1+ arg2;
      }
}
```

Scenario:

Mr. George owns a departmental store. He requires keeping record of incoming and sold out stocks.

Demonstration\Code Snippet:

[Note: Numbering represents the sequence in which the program flows]



```
1. Add Items
2. Remove Items
3. Display Items
4. Exit
Enter Choice:1
Enter the item name you want to add:Colgate
Enter Choice:1
Enter the item name you want to add:Lux
Enter Choice:1
Enter the item name you want to add:Garnier
Enter Choice:3

Items available in store are:
Colgate
Lux
Garnier

Enter Choice:2
Enter the item name you want to remove:Lux
Enter Choice:3

Items available in store are:
Colgate
Garnier

Enter Choice:4
Press any key to continue . . .
```

Figure 11.3-2: Output

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

Step 1: Define a generic base class and declare a generic list

```
namespace ConsoleApplication14
{
    public class Department<T>  //generic base class
    {
        public List<T> mylist = new List<T>();
    }
    public class Items : Department<string>
```

```
            {
[19]        public void addItems(string item)
            {
[20]           mylist.Add(item);
            }
[25]        public void removeItems(string item)
            {
[26]            mylist.Remove(item);
            }
```

Step 2: Define a generic method to display list of items

```
            //Generic method
[29]        public void DisplayItems<T>(IList<T> coll)
            {
[30]        Console.WriteLine("\nItems available in store are:");
            foreach (T item in coll)
[31]            System.Console.Write(item.ToString() + "\n");
[32]            System.Console.WriteLine();
            }
      }
```

Step 3: Define the Object of the class Items in the Main() method and invoke the methods.

```
class Program
{
[1]   static void Main()
      {
[2]         try{
[3]            int choice;
[4]            string item;
[5]            Items obj_items = new Items();

[6]            Console.WriteLine("1. Add Items");
[7]            Console.WriteLine("2. Remove Items");
[8]            Console.WriteLine("3. Display Items");
[9]            Console.WriteLine("4. Exit");

[10]            while (true)
[11]            {
[12]                Console.Write("Enter Choice:");
[13]                choice = int.Parse(Console.ReadLine());
[14*]               switch (choice)
                    {
[15]                case 1:
[16]                Console.Write("Enter the item name you want
                                   to add:");
[17]                item = Console.ReadLine();
[18]                obj_items.addItems(item);
                    break;

[21]                case 2:
[22]                Console.Write("Enter the item name you want
                                  to remove:");
[23]                item = Console.ReadLine();
[24]                obj_items.removeItems(item);
                    break;

[27]                case 3:
[28]                obj_items.DisplayItems(obj_items.mylist);
                    break;

                    case 4:
                    Environment.Exit(0);  //Exit from program
                    break;

                    default:
                    Console.WriteLine("\nInvalid Choice");
                    break;
               }
           }
           catch (Exception ex)
           { Console.WriteLine(ex.Message)}
      }
}
```

Context:

- Closely related data can be handled more efficiently when grouped together into a collection.
- To gain the immediate benefit of type safety without having to derive from a base collection type and implement type-specific members.

Practice Session:

- Write a program to swap two integer numbers using custom generic method.
- Identify the interfaces implemented by ArrayList Class.

Check List:

- Collection Classes and Interfaces.
- Type Safety and Strongly Typed Collections.
- Custom Generic Methods, Generic Structures and Classes.
- Generic Base Classes and Generic Interfaces.

Common Errors:

- Type parameter not substituted appropriately in the derived methods from the parent class.
- Type parameter not specified when the method does not take parameters. It produces complier error .

Exception:

- Building Custom Collection may raise an exception as it is not type-safe.

Lessons Learnt :

☑ Generics are more type safe.
☑ Interfaces must be implemented by a Class or Structure.

Best Practices:

- Generic collection types generally perform better than the non generic collection types, as they do not result in boxing or unboxing penalties.
- Use Generics rather than building Custom Collection types.
- Use Generics to maximize code reuse, type safety, and performance.

Crossword: Unit-11 **Estimated Time: 10 mins.**

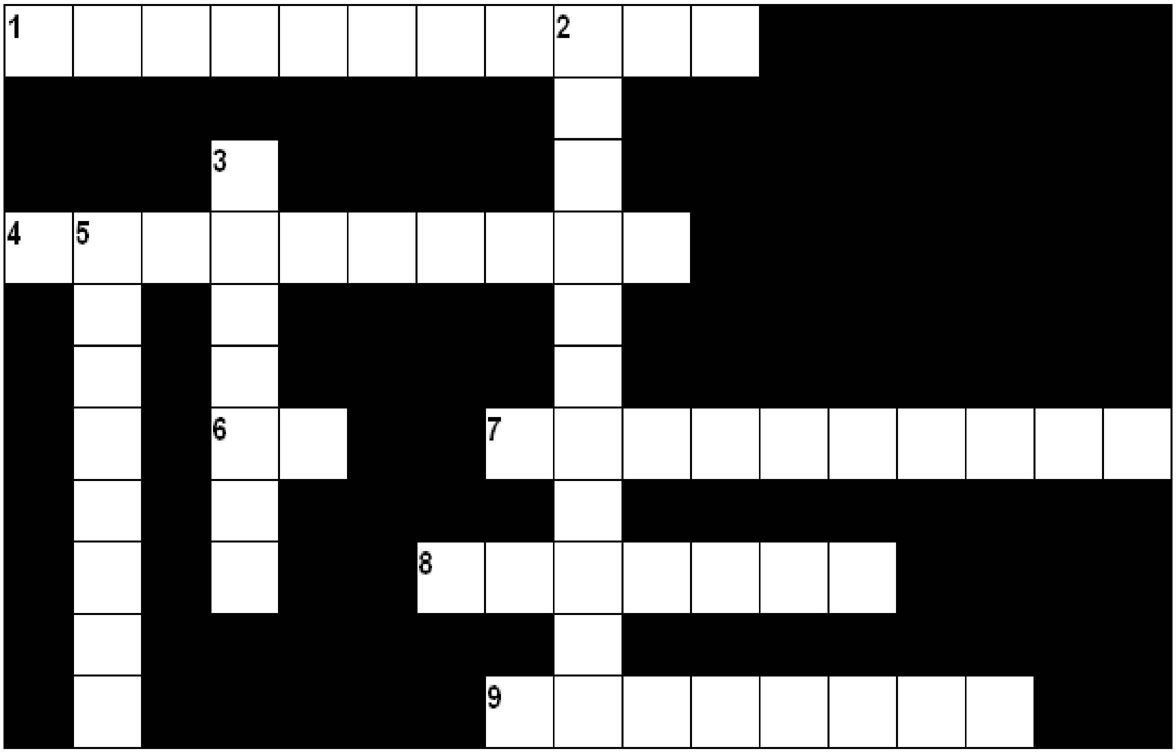


Fig.11-1

Across:

1. _____________ specifies an object relationship.(11)

4. _____________ are hierarchical.(10)

6. The entire .NET code on compilation gets converted into an ___ code and gets stored as an assembly. (2)

7. To build public and private keys we have to use the __________ utility. (10)

8. The types of assemblies are - Single-File and Multifile Assemblies, Private and Shared Assemblies, Satellite and Resource-only Assemblies, Static and _________ Assemblies.(7)

9. Assembly Manifest contains the _______ for the assembly.(8)

Down:

2. ____________ specifies a class relationship. (11)

3. Assembly Manifest contains the name and ______ Information of Assembly.(7)

5. ___________ is a self-describing unit of reuse, versioning, security and deployment.(8)

12.0 Operators, Delegates and Events

Topics

- 12.1 Operator Overloading
- 12.2 Creating and Using Delegates
- 12.3 Crossword

Topic: Operator Overloading

Estimated Time: 35 mins.

Objectives: After the completion of module, participant should know:

- Use of Operator
- Operator overloading and its use

Presentation:

Operator

- C# operators in precedence wise:

Operator category	Operators
Primary	x.y, f(x), a[x], x++, x--, new, typeof, checked, unchecked, ->

Unary	+, -, !, ~, ++x, --x, (T)x, true, false &, sizeof
Multiplicative	*, /, %
Additive	+ ,-
Shift	<< ,>>
Relational and type testing	< ,>, <=, >=, is, as
Equality	== ,!=
Logical AND and XOR and OR	& and ^ and |
Conditional AND and OR and Condition	&& and || and ?:
Assignment	= ,+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, ??

Operator Overloading

- It permits user-defined operator implementations to do specified operations
- Operators and their overloadability in C#

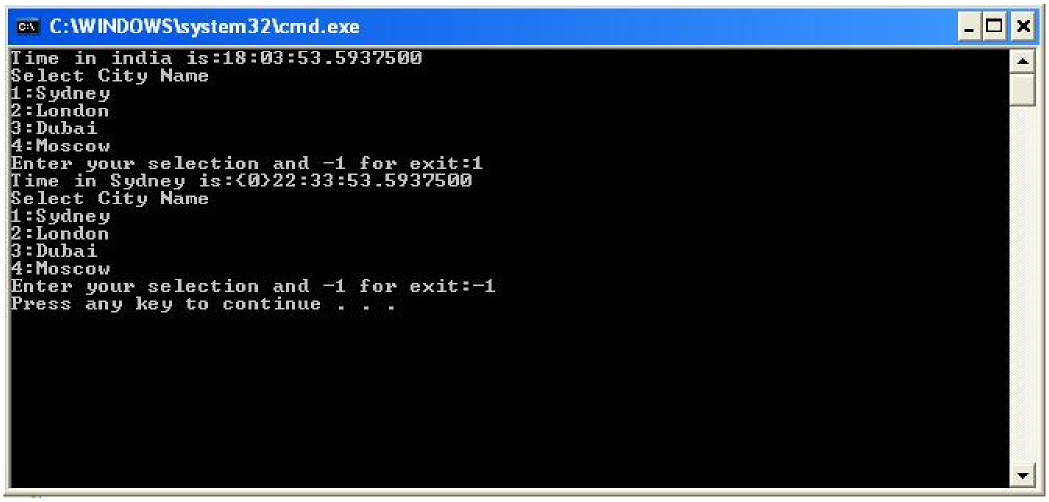
Operators	Overloadability
+, -, !, ~, ++, --, true, false	These unary operators can be overloaded.
+, -, *, /, %, &, |, ^, <<, >>	These binary operators can be overloaded.
==, !=, <, >, <=, >=	The comparison operators can be overloaded, but also overload matching operator.
&&, ||	The conditional logical operators cannot be overloaded, but they are evaluated using **&** and |, which can be overloaded.
[]	The array indexing operator cannot be overloaded, but you can define indexers.
( )	The cast operator cannot be overloaded, but you can define new conversion operators (explicit and implicit).
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	Assignment operators cannot be overloaded, but +=, for example, is evaluated using +, which can be overloaded.
=, ., ?:, ->, new, is, sizeof, typeof	These operators cannot be overloaded.

- In C#, a special function operator function is used for overloading purpose
- These methods must be public, static, have value arguments and can’t return void
- The general form of an operator function is as follows
  Example: `public static return_type operator op (argument list)`
- op-keyword Unary operators-one argument Binary operator-two arguments.

Scenario:

Mr. Alex owns a software company and his branches are available in many countries. So to arrange meeting and keep track of progress of their work he needs to know timing of respective countries.

Demonstration/Code Snippet:



```
Time in india is:18:03:53.5937500
Select City Name
1:Sydney
2:London
3:Dubai
4:Moscow
Enter your selection and -1 for exit:1
Time in Sydney is:(0)22:33:53.5937500
Select City Name
1:Sydney
2:London
3:Dubai
4:Moscow
Enter your selection and -1 for exit:-1
Press any key to continue . . .
```

Figure: 12.1-3: Output

[Note: Numbering represents the sequence in which the program flows]
using System;

Step 1: Declare class which overload + and – operator

```
public class opOverloadTime
{
            //Declare a one timespan for taking time
            public TimeSpan ts;
[9]         public opOverloadTime(){}
[6]         public opOverloadTime(TimeSpan t)
            {
[7]               ts=t;
            }
            //Overload + operator
[17]        public static opOverloadTime operator
            +(opOverloadTime   o1,TimeSpan t)
            {
[18]              opOverloadTime temp = new opOvrloadTime();
[19]              temp.ts = o1.ts+t;
[20]              return temp;
            }
           //Overload - operator
[17]        public static opOverloadTime operator -
            (opOverloadTime o1,TimeSpan t)
            {
[18]              opOverloadTime temp = new opOverloadTime();
[19]              temp.ts = o1.ts - t;
[20]              return temp;
```

```
            }
}
```

Step 2: Make another class in which operators are overloaded and time in different countries are calculated

```
class Program
{
[1]   static void Main(string[] args)
      {
[2]         int cityname=1;
            //Using datetime class now method get time
[3]         TimeSpan ts = DateTime.Now.TimeOfDay;
[4]         Console.WriteLine("Time in india is:" + ts);
            //Intantiating object
[5]         opOverloadTime o1 = new opOverloadTime(ts);
[8]         opOverloadTime o2 = new opOverloadTime();
[10]        while (cityname>0)
            {
[11]              Console.Write("Select City Name \n1:Sydney
                  \n2:London \n3:Dubai\n4:Moscow\nEnter your
                  selection and -1 for exit:");
[12]              cityname = int.Parse(Console.ReadLine());
[*13]             switch (cityname)
                  {
[14]              case 1:
[15]              TimeSpan t1 = new TimeSpan(4, 30, 00);
                  //Call overloaded operator method
[16]              o2 = o1 + t1;
[21]              Console.WriteLine("Time in Sydney is:{0}" +
                  o2.ts);
[22]              break;
[14]              case 2:
[15]              TimeSpan t2 = new TimeSpan(6, 20, 00);
[16]              o2 = o1 + t2;
[21]              Console.WriteLine("Time in London is:{0}" +
                  o2.ts);
[22]              break;
[14]              case 3:
[15]              TimeSpan t3 = new TimeSpan(1, 20, 00);
[16]              o2 = o1 - t3;
[21]              Console.WriteLine("Time in Dubai is:{0}" +
                  o2.ts);
[22]              break;
[14]              case 4:
[15]              TimeSpan t4 = new TimeSpan(1, 20, 00);
[16]              o2 = o1 - t4;
[21]              Console.WriteLine("Time in Moscow is:{0}" +
                  o2.ts);
[22]              break;
                  }
           }
     }
}
```

When o2 = o1 + t1 is executed then + overloaded method is called and when o2=o1-t3 is ececuted then – overloded method is called.

Operator keyword can use only with static methods.

Context:

- Operator is used to define which operations to perform in an expression.
- Operator overloading permits user-defined operator implementations.

Practice Session:

- How many types of operators are there?
- Fix the bugs in the following code snippets
  1. public bool operator != (Time t1, Time t2) { ... }
  2. public static operator float(Time t1) { ... }
  3. public static Time operator += (Time t1, Time t2){ ... }
  4. public static bool Equals(Object obj) { ... }
  5. public static int operator implicit(Time t1){ ...}

Check List:

- Use of Operator Overloading.
- Unary and Binary operator overloading.

Common Errors:

- Return type of binary operator overloading is void.
- Operator overloading methods must be declared with public and static access modifiers.

Exception:

- If we are trying to Overload Comparison operator without matching operator then we get exception with the message “**matching operator is missing**”

Lessons Learnt:

☑ Overloaded operators useful in building utility types, Strings, Points, rectangles, fractions, hexagons.

☑ Use explicit and implicit forecast operator ().

☑ Numerous types in base class library have already overloaded operators.

Best Practices:

- Instead of using Explicit Method calls it is preferred to use Operator Overloading.
- Use this feature intelligently as it makes harder for user to understand type functionality.

Topic: Delegates

Estimated Time: 45 mins.

Objectives: After the completion of module, participant should know:

- Use of Delegate and way of Delegate implementation.
- Use of Events.

Presentation:

Delegate

- A Delegate in C# is similar to a function pointer in C or C++.
- It is an object that refers to a method.
- It passes the methods as parameters. Any delegate can be used to call different methods during runtime but remember that method argument type and return type must match.
- Method argument and return type must be same.
- Three steps in defining and using delegates are:
  - ➢ Declaration
  - ➢ Instantiation
  - ➢ Invocation
- Two types of Delegates:
  - ➢ Single Cast Delegate: point to one function at a time.
  - ➢ Multi Cast Delegate: point to more than one function at a time.

Events

- It is a member that enables an object or class to provide notifications.

- Event handlers are represented by delegates.
- Event declaration: event <event-delegate> <object-name>;

Scenario:

Mr. Steve is working for Bookshop. He is working for stock department. When enough stock is not available then he has to meet the supplier and place order. So he wants to find whether available quantity of book is enough or not.

Demonstration/Code Snippet :

[Note: Numbering represents the sequence in which the program flows]

using System;

Step 1: Declare Delegate

```
[7],[9],[11] public delegate void intQty(int i);
```

Step 2: Make two method which signature are same as delegate

```
class delegateTest
{
      int quantity;

[17]  static void lessQty(int a)
      {
[18]        Console.WriteLine("Place  new  order  because  quantity
            for book is not enough.");
      }
[9]   static void enoughQty(int b)
      {
[10]        Console.WriteLine("Quantity for book is enough.");
      }
[19]  static void placeQty(int c)
      {
[20]        quantity=c;
[21]        Console.WriteLine("Oreder place for book and quantity
            asked is:"+quantity);
      }
[1]   static void Main(string[] args)
      {
[2]         int j;
[3]         Console.Write("Enter Quantity for book:");
[4]         j = Convert.ToInt32(Console.ReadLine());
[5]         if (j <= 0)
            {
                  //construct delegate
[6]               intQty test = new intQty(lessQty);
                  //call methods through delegate
[8]               intQty test1 = new intQty(placeQty);
[10]              intQty test2;
```

```
                        //multicast  delegate  two  delegates  test  and
                        //test1 are composed to form test2
[12]                    test2=test+test1;
[13]                    int qty;
[14]                    Console.WriteLine("Enter quantity:");
[15]                    qty= int.parse(Console.ReadLine());
[16]                    test2(qty);
                  }
[*5]              else
                  {
[6]                     intQty test3 = new intQty(enoughQty);
[8]                     test3(j);
                  }
            }
      }
```

When test2(qty) executed then two methods lessQty() and placeQty() are call. And test3(j) call enougQty() method.

Context:

- Use delegate to create an object that refers to a method.
- Use delegate to support event.

Practice Session:

- Enlist differences between Delegate and Classes.
- In above scenario, add new method calculatePrice() to calculate price for order that placed so that bill can paid to supplier.

Check List:

- Uses of Delegates and Events.
- Implementation of Delegates.
- Multicast Delegates.

Common Errors:

- If delegates has no arguments and it passes argument in function it gives compile error with the following description :

delegate doesn't take arguments

Exception:

- Delegate is similar like function pointer but delegate can represent both static and instance method.

Lessons Learnt:

☑ The method's argument type and return type have to match with delegate.

☑ A delegate can either be called synchronously or asynchronously by using BeginInvoke() and EndInvoke() methods

☑ Windows Microsoft Paint can easily made by delegate.

Best Practices:

- Effective use of Delegate improves the performance of application.
- Once a delegate is created, the method it is associated will never changes because delegate objects are immutable.
- Delegate gives a way to execute methods at runtime.
- Delegate chains allow defining set of methods that executed as one unit.

Crossword: Unit-12 **Estimated Time: 10 mins.**

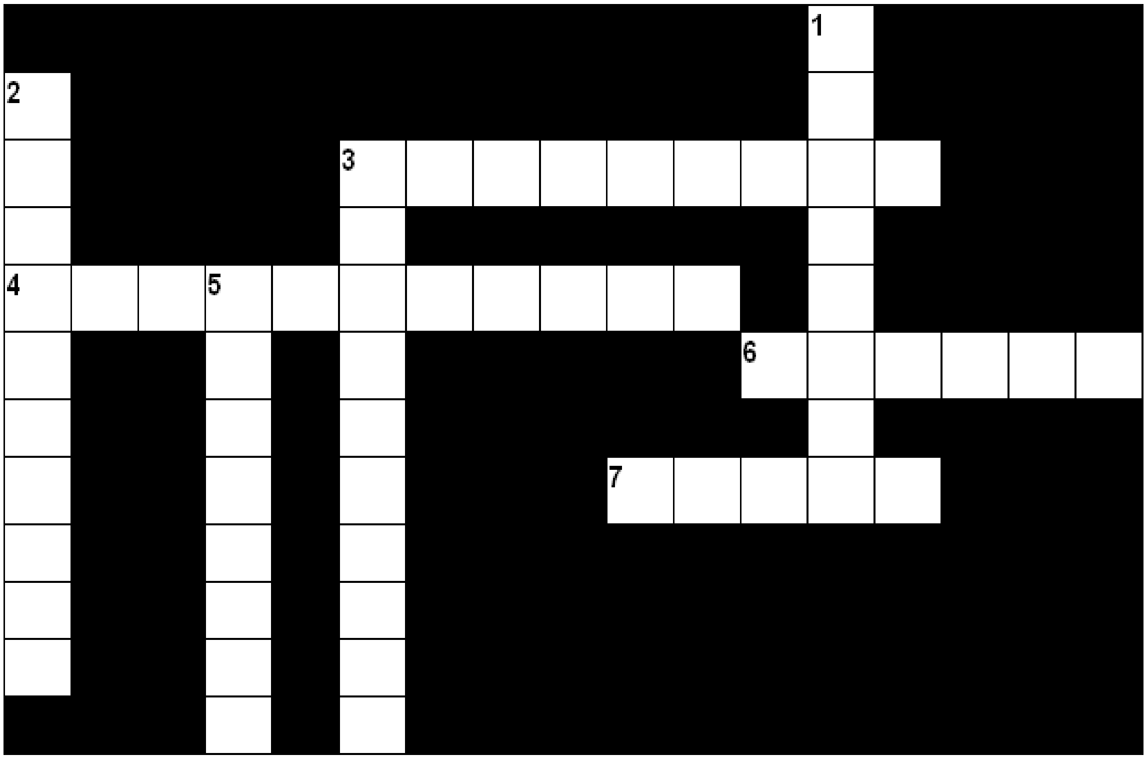


Fig.12-1

Across:

3. The ________ determines when an event is raised. (9)

4. Events can be used to __________ threads.(3)

6. In C#, a special function called operator function is used for overloading purpose. These special function or method must be public and ______. (6)

7. The _______ determine what action is taken in response to the event. (5)

Down:

1. The mechanism of giving a special meaning to a standard C# operator with respect to a user defined data type such as classes or structures is known as ________ overloading. (8)

2. The ________ determine what action is taken in response to the event.(10)

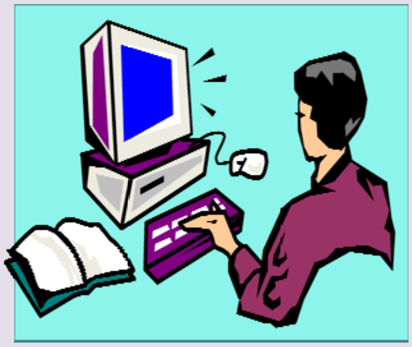
3. Delegates allow methods to be passed as __________.(10)

5. This ability to refer to a method as a parameter makes delegates ideal for defining ______methods. (8)

13.0 Properties and Indexers

Topics

- 13.1 Using Properties
- 13.2 Indexers
- 13.3 Crossword



Topic: Properties

Estimated Time: 30 mins.

Objectives: After the completion of module, participant should know

- Implementation of Properties
- Property Accessors
- Property Types

Presentation:

Properties:

- It provides flexible mechanism to read, write, or compute the values of private fields.
- It provides a useful way to encapsulate information inside a class.
- Accesors are the special methods of properties used as public data member.

- A ‘get’ property accessor returns the property value and provide read access.
- A ‘set’ accessor assigns a new value and provide write access.

Types of Properties

1. Static
   - ➢ It belongs to a class rather than to the object of the class.
   - ➢ set and get Accessor can access only other static members of the class.
2. Abstract
   - ➢ ‘abstract’ keyword is used to declare the property as abstract.
   - ➢ ‘abstract’ property carries no code at all. The get or set accessors are simply represented with a semicolon and are implemented in derived class.
3. Read/Write Properties have both get and set accessors
   - ➢ Read-Only Properties have get accessor only. They are not constants.
   - ➢ Write-Only Properties have set accessor only.

Scenario:

Mr. John owns a Watch Emporium. Time is in seconds and for customer convenience he wants to display time in hours. So he needs one property which converts seconds into hours and display time in hours in his digital clock.

Demonstration/Code Snippet:

[Note: Numbering represents the sequence in which the program flows]

```
using System;
```

Step 1: Create a Class TimePeriod which takes a time period in seconds~~.~~

```
namespace PropertiesSample
{
      class TimePeriod
      {
            private double seconds;
```

Step 2: Define a property called Hours which specifies time in Hours. Accessors for the Hours property perform the conversion between Hours and Seconds~~.~~

```
[4]         public double Hours
            {
[9]               get { return seconds / 3600; }
[5]               set { seconds = value * 3600; }// Value Keyword
                          defines the value being assigned by set
                          accessor.
            }
       }
```

Step 3: Define the main class and assign some value to Hours~~.~~

```
      class Program
      {
```

```
[1]          static void Main(string[] args)
             {
[2]               TimePeriod t = new TimePeriod();
                  // Assigning the Hours property causes the
                      'set'accessor to be called.
[3]               t.Hours = 24;
                  // Evaluating the Hours property causes the
                    'get' accessor to be called.
[6]               System.Console.WriteLine("John Clock Shop");
[7]               System.Console.WriteLine("----------------");
[8]               System.Console.WriteLine("Time in hours: " +
                  t.Hours);
             }
       }
}
```

Context:

- Properties can be used while implementing Data Encapsulation and Hiding.

Practice Session:

- Is it possible to have different access modifiers on the get/set methods of a property?
- Write a demo code to illustrate the use of static and abstract properties.

Checklist:

- Accessors
- Static Properties
- Abstract Properties

Common Errors:

- While using static properties, members of the class are not declared as static.

Exception

- We cannot declare any variable with the name value inside the set accessor.

Lessons Learnt:

☑ Properties that do not implement set method are read only.

☑ If the abstract class contains only set accessor then implement only set in the derived class.

Best Practices:

- It is a good programming practice to provide a set of public SET and GET methods to access the private data fields of a particular class, since they cannot be directly accessed outside the class.

Topic: Indexer

Estimated Time:40 mins

Objectives: After completion of this module, participant should know

- Implement Indexers.
- Compare Indexers to Arrays and Properties.

Presentation:

An Indexer

- Enables to use index on an object.
- Provides array-like access to an object.
- To define an indexer create a property called *this* and specify the index type.

INDEXERS	ARRAYS
Uses array notation	Uses array notation
Can use non-integer subscripts	Cannot use non-integer subscripts
Can be overloaded	Cannot be overloaded
These are not variables	These are variables

Fig: 13.2-1

INDEXERS	PROPERTIES
Uses get and set accessors	Uses get and set accessors
Have no address	Have no address
Cannot be void	Cannot be void
Specific data member is accessed. Obtain value from the object itself.	Specific data member is not accessed
Can be overloaded	Cannot be overloaded
Cannot be static	Can be static

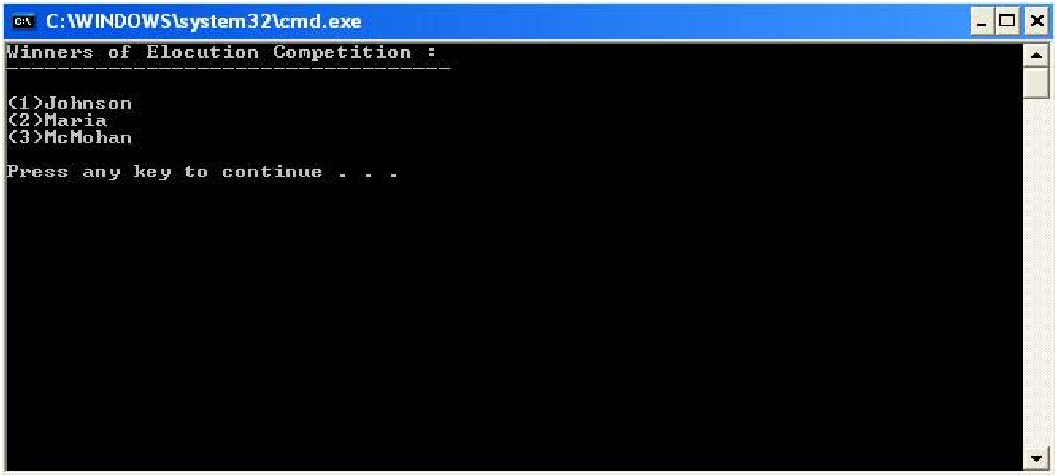
Fig: 13.2-2

Scenario:

Leeds University hosted an Inter-College competition. University Chairman requires list of top 3 winners of the competition to be displayed on the Notice Board.

Demonstration/Code Snippet:

[Note: Numbering represents the sequence in which the program flows]



```
Winners of Elocution Competition :
-----------------------------------

(1)Johnson
(2)Maria
(3)McMohan

Press any key to continue . . .
```

```
using ...
```

Step 1: Create a class Day to return a day's value

```
// Using a string as an indexer value
class Date
{
      string[] days = { "Sun", "Mon", "Tues", "Wed", "Thurs",
      "Fri", "Sat" };
      // This method finds the day or returns -1
[7]    private int GetDate(string testDay)
       {
[8]          int i = 0;
[9]          foreach (string day in days)
             {
[10]               if (day == testDay)
```

```
                  {
[11]                    if (day == "Sun")
[12]                          return 25;
                        else if (day == "Mon")
                              return 26;
                        else if (day == "Tue")
                              return 27;
                        else if (day == "Wen")
                              return 28;
                        else if (day == "Thur")
                              return 29;
                        else if (day == "Fri")
                              return 30;
                        else if (day == "Sat")
                              return 31;
[13]              }
                  i++;
            }
            return -1;
       }
       // The get accessor returns an integer for a given string
[4]    public int this[string day]
       {
[5]         get
[6]               return (GetDate(day));
       }
}
```

Step 2: Declare a string variable to hold the names of winners.

```
// Using a integer as an indexer value
class Elocution
{
      private string[] data = new string[3];
[22]   public string this[int index]
      {
[28],[31],[34]  get{
[29],[32],[35]       return data[index];}
[16],[19],[22]  set{
[17],[20],[23]       data[index] = value;}
      }
}
```

Step 4: Define main class and list the winners.

```
class Leeds
{
[1]   public static void Main()
      {
[2]         Date dat=new Date();
[3]         Console.WriteLine("Date of elocution competition:",
            +dat["Sun"]);
[14]        Elocution eloc_obj = new Elocution();
[15]        eloc_obj[0] = "Johnson";
[18]        eloc_obj[1] = "Maria";
[21]        eloc_obj[2] = "McMohan";
```

```
            Console.WriteLine("Winners of Elocution competition:")
            Console.WriteLine("--------------------------------");
            Console.WriteLine();
            Console.WriteLine("(1){0}", eloc_obj[0]);
            //1st index element is accessed
            Console.WriteLine("(2){0}", eloc_obj[1]);
            //2nd index element is accessed
            Console.WriteLine("(3){0}", eloc_obj[2]);
            //3rd index element is accessed
            Console.WriteLine();
      }
}
```

Context:

- Indexers can be used as an alternative to arrays as it can be overloaded.

Practice Session:

- Explain the participation of Indexer in Inheritance.
- Integer can use only integer subscripts. True or False
- Create a stack that can have items placed on it and taken off of it. Using an indexer, access items within the stack without needing to remove items.

Check List:

- Indexer can be overloaded.
- Indexer can use non-integer subscripts.

Common Errors:

- Indexer declared as Static shows compilation error.
- Use of braces.

Exception:

- In the above example the statement:
  `Console.WriteLine("(1){1}",eloc_obj[3]);`
  Raises System.FormatException depicting the index size to be less than the size of argument list.

Lessons Learnt:

☑ Indexer can be used alike properties, but it can be overloaded.
☑ Indexers cannot be static.
☑ Use of Indexer instead of properties.

Best Practices:

- Use of indexer makes the code more readable and easier to understand.

Crossword: Unit-13 **Estimated Time: 10 mins.**

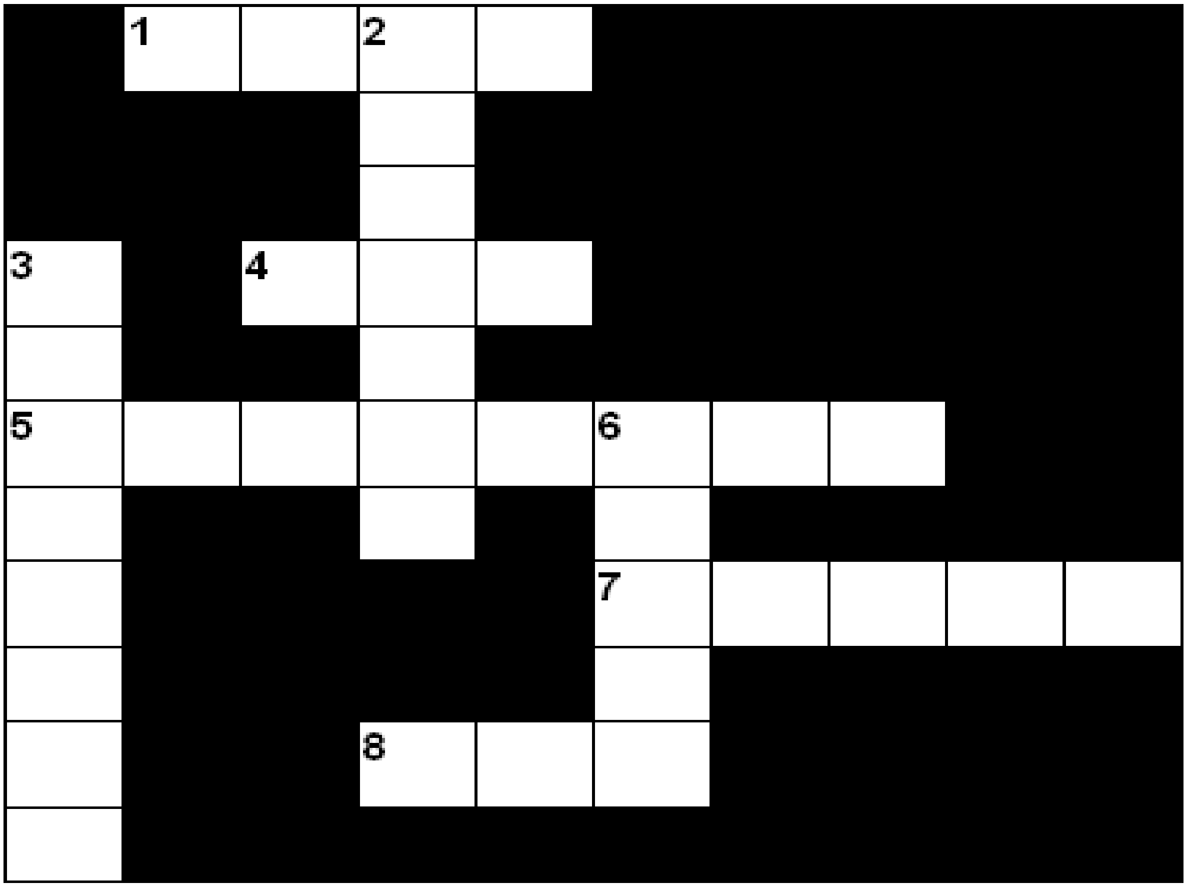


Fig.13-1

Across:

1. With indexers, instead of creating a name as you do with properties, you use the ____ keyword, which refers to the object instance and thus the object name is used.(4)

4. A _____ accessor is used to assign a new value.(3)

5. Properties can be used as though they are public data members, but they are actually special methods called _______. This enables data to be accessed easily while still providing the safety and flexibility of methods. (8)

7. In essence indexer enables you to treat an object like an ______.(5)

8. A ____ property accessor is used to return the property value.(3)

Down:

2. An ________ enables you to use an index on an object to obtain values stored within the object.(7)

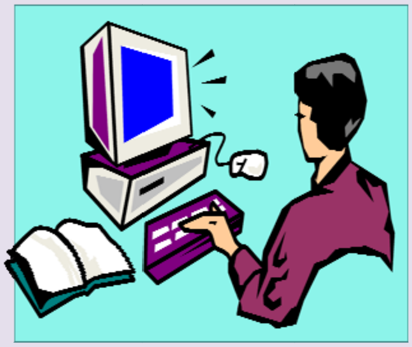
3. Properties that do not implement a set method are _________. (8)

6. The indexers are usually known as ________ arrays in C# community.(5)

14.0 Attributes

Topics

- 14.1 Overview of Attributes
- 14.2 Retrieving Attribute Values
- 14.3 Crossword



Topic: Attributes

Estimated Time: 45 mins.

Objectives: After the completion of this module, participant should know

- Attributes and their uses.
- Attributes with multiple or no parameters.

Presentation:

- Attributes are elements that add declarative information to programs.
- They are placed in square brackets [].
- A program's attributes can be retrieved from its assembly metadata with the process named reflection or with ildasm.exe tool
- Two types of Attributes:
  1. Predefined
     - ➢ Conditional
       - ✓ Allows to create conditional methods
       - ✓ Its only invoked when symbol defined via #define keyword
       - ✓ It must return void and members of class or structure.
     - ➢ Obsolete
       - ✓ Display message at compiled time.
       - ✓ In [obsolete(message,error)] if error value is true then it generates compilation error rather than warning so program cannot be compiled into executable program.
  2. Custom
     - ➢ Steps to create Custom Attributes:
       1. Define the attribute's usage.
       2. Extend our class with Attribute Class.
       3. Define the behaviors to the class.
     - ➢ AttributeUsage used to create custom attribute and its properties are:
       1. AttributeTarget
          - ✓ Values-All, Class, Method, Assembly, Enum.

- ✓ Specifies programming entity.
- ✓ Multiple Attribute targets are declared by using "|" between different Targets.

2. Inherited
   - ✓ Values-true and false.
   - ✓ True then the Attribute gets attached to the subclasses of the class where it’s used.
3. AllowMultiple
   - ✓ Values-True and False.
   - ✓ It denotes whether the Attribute can be used multiple times in the target element.
   - ✓ Allows multiple=false means multiple attributes can be used.

Scenario:

Explain how predefined attribute and custom attribute are used in a program to give productive information.

Demonstration/Code Snippet:

[Note: Numbering represents the sequence in which the program flows]

```
//Define conditional attribute ALLOW
#define ALLOW

using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;  //Use to define conditional attribute.

namespace ConsoleApplication8
{
      //Defining AttributeUsage
      [AttributeUsage(AttributeTargets.All)]
      //Extend class with attribute class
[7]   public class AttributecustDemo : Attribute
      {
            //Define behaviors of class
            string firstName;
            //Constructor
[8]         public AttributecustDemo(string fname){
[9]               firstName = fname;}
            //Property to return name
[11]        public string Name{
[12]              get{
[13]                    return firstName; }    }
            [Conditional("ALLOW")]   //Conditional attribute
[15]        public void allow(){
[16]              Console.WriteLine("Allow to diaplayname"); }
            [Conditional("DISALLOW")]
            public void disallow()
                  Console.WriteLine("Not allow to display name");
            [Obsolete("Display name")]  //Obsolete attribute
[19]        public void displayName(){
[20]              Console.WriteLine("Name is display using
                  method:" + this.firstName);  }
      }
      //Attaching attribute with class
      [AttributecustDemo("Hiral")]
      class att{ }
      public class test
      {
[1]         static void Main()
            {
                  //Retrieve attribute with GetCustAttribute
                  //method,Which defined by MemberInfo and
                  //inherited by Type
[2]               Type t = typeof(att);
[3]               Console.Write("Name display using custom
                  attribute:");
                  //Retrieve the custom attribute
                  //AttributecustDemo
[4]               Type t1 = typeof(AttributecustDemo);
                  //Give refrence to AttributecustDemo
[5]               AttributecustDemo cust =
                  (AttributecustDemo)
                  Attribute.GetCustomAttribute(t,t1);
                  //Get member of AttributecustDemo
[10]              Console.WriteLine(cust.Name);
                  //called because ALLOW is defined
[14]              cust.allow();
                  //Not called because DISALLOW is not defined
[17]              cust.disallow();
                  //During compilation at this point message
                  //diaplayed and give compilation warning.
[18]              cust.displayName();
            }
      }
}
```

When defining a custom attribute, you must derive from **Attribute** class.

ObsoleteAttribute used to mark a code element as obsolete. This informs users of the code that they should not rely on its availability in future versions.

Context:

- Use Attribute to give declarative information.

Practice Session:

- Describe the fundamental difference between Elements and Attributes.
- Can attributes be placed on specific parameters to a method?

Checklist:

- Use of Attributes
- Custom attribute
- Predefined Attribute

Common Errors:

- Attribute class must be preceded by attribute called AttributeUsage.

Exception:

- The Obsolete attribute marks a program entity to no longer recommend for use. Use of an entity marked obsolete will subsequently generate a warning or an error, depending on how the attribute is configured.

Lessons Learnt:

☑ An attribute must be defined by a class which derives from System.Attribute.

☑ Conditional attribute cannot be member of interface and cannot be preceded with override keyword.

Best Practices:

- Attributes are C# language elements that decorate program elements with additional metadata that describes the program. This metadata is then evaluated at different places, such as runtime or design time for various purposes.

Crossword: Unit-14 **Estimated Time: 10 mins.**

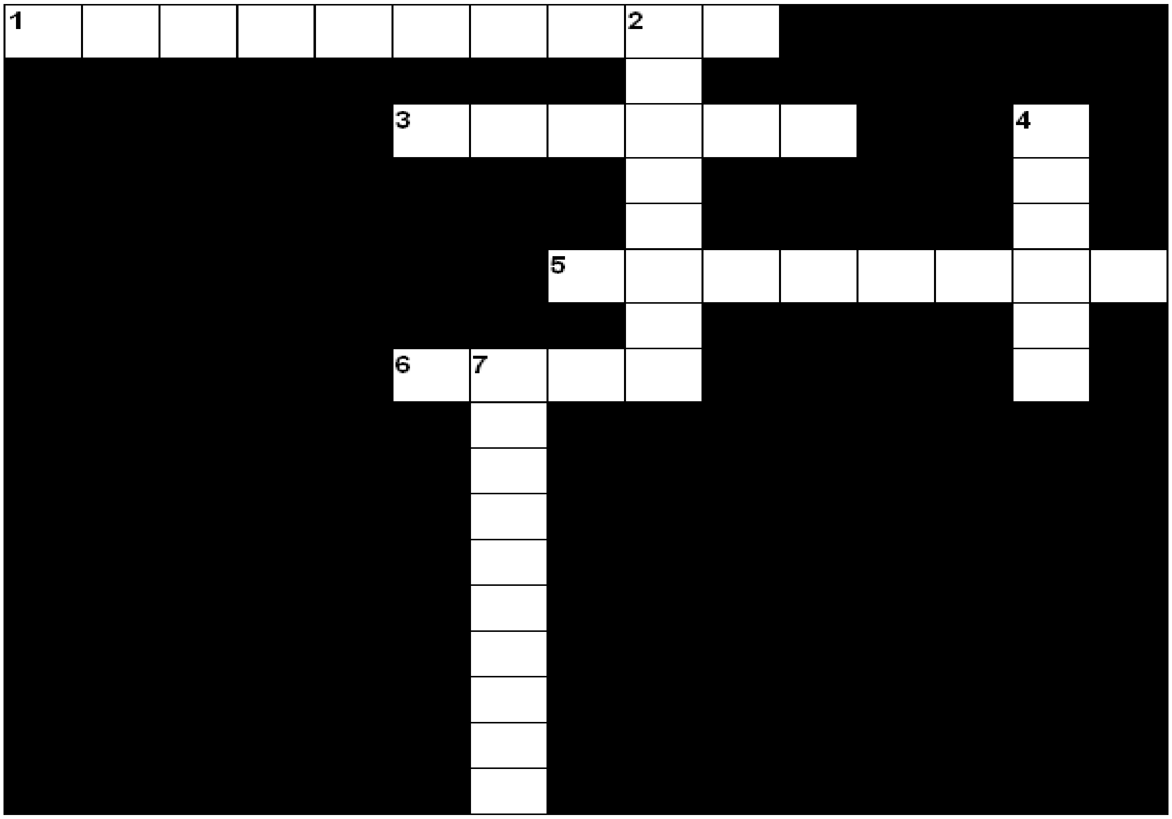


Fig.14-1

Across:

1. C# provides a mechanism for defining declarative tags, called attributes, which you can place on certain entities in your source code to specify additional information. The information that it contain can be retrieved at run time through __________. (10)

3. System.ObsoleteAttribute, which marks code as obsolete. The string "Will be removed in next version" is passed to the attribute. This attribute causes a compiler warning that displays the _________ string when code that the attribute describes is called.(6)

5. In component-based programming versus source code based programming, the __________ replaces the source file as the method of reuse.(8)

6. [In] or [InAttribute] have the _____ meaning.(4)

Down:

2. [Obsolete] can be used to mark an Attribute Target as _________. (8)

4. ___________attributes are extensions of the metadata that survive compilation and are therefore visible by the client even if he doesn't have access to the source code.(6)

7. Declarative tags that convey information to the runtime are ________.(10)

Answers for Crosswords

Unit-1

Across	**2) Garbage collection 5) Class Library 6)Application Code 7)Bin**
Down	**1) Code Access Security 3) Platform 4)CLR**

Unit-2

Across	**2) Garbage collection 5)Class Library 6)Application Code 7)Bin**
Down	**1)Code Access Security 3)Platform 4)CLR**

Unit-3

Across	**1)Using 4)Readline 5) Console.Write**
Down	**2)System.IO 3)System**

Unit-4

Across	**2)Throw 4)CLR 5) Catch 7) Continue 8) CLS**
Down	**1)Switch 3)Goto 4) Control 6) Finally**

Unit-5

Across	**1)Object 4)Reusability 6) Ref 7) End**
Down	**1)Operator 2)Value 3) Method 5) String**

Unit-6

Across	**1)Reference 3)Multi 5) Jagged**

Down	1)Right 2) Null 4) Index 6) Array

Unit-7

Across	2)Public 3)Base 4) Overriding 5) Reference 7) Polymorphism 8) Base
Down	1) Mutator 4)Overloading 6) Abstract

Unit-8

Across	2)Conversion 4) Reference 6) Explicit 8) System
Down	1) Boxing 3) Namespace 5) Heap 7) Implicit

Unit-9

Across	1)Maintainence 3) Structures 4) Destructor 6) Private 7) Garbage
Down	2) Constructor 5) Static

Unit-10

Across	1)Inheritance 4) Abstract 5) Parent
Down	1) Interface 2) Base 3) Derived 5) Public

Unit-11

Across	1)Aggregation 4) Namespaces 6) IL 7) Strongname 8) Dynamic 9) Metadata
Down	2) Inheritance 3) Version 5) Assembly

Unit-12

Across	3) Publisher 4)Synchronize 6) Static 7) Subscribers
Down	1) Operator 2) Subscriber 3) Parameters 5) Callback

Unit-13

Across	1)This 4)Set 5) Accessors 7) Array 8) Get
Down	2) Indexer 3) ReadOnly 6) Smart

Unit-14

Across	1)Reflection 3)Passed 5) Metadata 6) Same
Down	2) Obsolete 4) Custom 7) Attributes

Team Members

Srikanth Nivarthi
47275

Sreenivas Ram
66223

Seshu Babu Barma
56150

Veerendra Kumar Ankem
77964

Teena Arora
74572

Contributors

Nimesh Abhinav
68484

Sonia Gupta
68532

Vinit Naik
72118

Himanshu Raj
72084

Hiral Rathod
72120

Janit Vora
72121

Praveen Nagumantri
66774