Q2 (a) Describe the building blocks of .NET Platform with the help of a diagram.

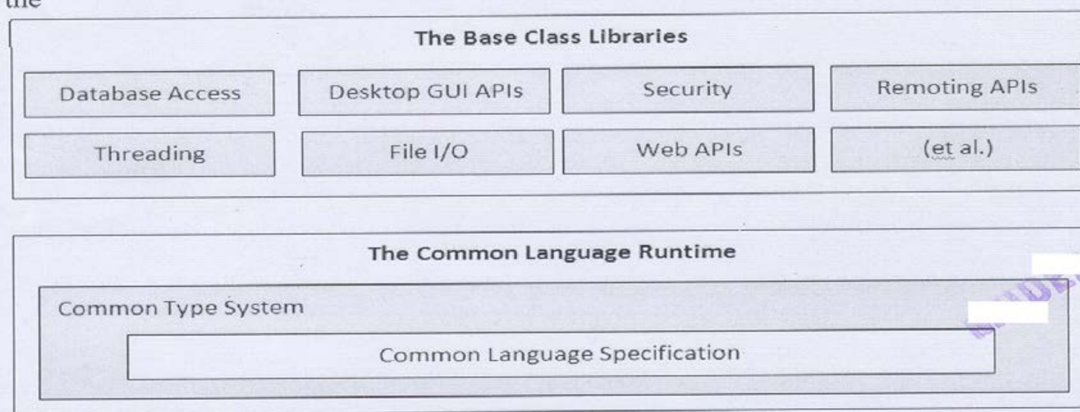
Answer

Introducing the Building Blocks of the .NET Platform (the CLR, CTS, and CLS)Now that you know some of the major benefits provided by .NET, let's preview three key (andinterrelated) topics that make it all possible: the CLR, CTS, and CLS. From a programmer's point of view,.NET can be understood as a runtime environment and a comprehensive base class library. The runtimelayer is properly referred to as the *Common Language Runtime*, or *CLR*. The primary role of the CLR is tolocate, load, and manage .NET objects on your behalf. The CLR also takes care of a number of low-level details such as memory management, application hosting, coordinating threads, and performingsecurity checks (among other low-level details).Another building block of the .NET platform is the *Common Type System*, or *CTS*. The CTSspecification fully describes all possible data types and all programming constructs supported by the runtime, specifies how these entities can interact with each other, and details how they are representedin the .NET metadata format Understand that a given .NET-aware language might not support each and every feature defined by the CTS. The *Common Language Specification, or CLS,* is a related specification that defines a subset ofcommon types and programming constructs that all .NET programming languages can agree on. Thus, if you build .NET types that expose only CLS-compliant features, you can rest assured that all .NET-awarelanguages can consume them. Conversely, if you make use of a data type or programming construct thatis outside of the bounds of the CLS, you cannot guarantee that every .NET programming language can interact with your .NET code library.

The Role of the Base Class Libraries

In addition to the CLR and CTS/CLS specifications, the .NET platform provides a base class library that isavailable to all .NET programming languages. Not only does this base class library encapsulate variousprimitives such as threads, file input/output (I/O), graphical rendering systems, and interaction with various external hardware devices, but it also provides support for a number of services required by most real-world applications.

The base class libraries define types that can be used to build any type of software application. For example, you can use ASP.NET to build web sites, WCF to build networked services, WPF to build desktop GUI applications, and so forth. As well, the base class libraries provide types to interact with XML documents, the local directory and file system on a given computer, communicate with a relational databases (via ADO.NET), and so forth. From a high level, you can visualize the relationship between the



Q2 (b) Describe briefly any four .NET namespaces.

Answer

.NET Namespace Meaning in Life

System Within System, you find numerous useful types dealing with intrinsic data, mathematical computations, random number generation, environment variables, and garbage collection, as well as a number of commonly used exceptions and attributes.

System.Collections
System.Collections.Generic
These namespaces define a number of stock container types, as well as base types and interfaces that allow you to build customized collections.

System.Data
System.Data.Common
System.Data.EntityClient
System.Data.SqlClient
These namespaces are used for interacting with relational databases using ADO.NET.

System.IO
System.IO.Compression
System.IO.Ports
These namespaces define numerous types used to work with file I/O, compression of data, and port manipulation.

System.Reflection
System.Reflection.Emit
These namespaces define types that support runtime type discovery as well as dynamic creation of types.

Q3 (a) What is the significance of pre-processor directive? Explain any two C# pre-processor directives.

Answer

- a #define and #undef: To define and undefine conditional compilation symbols, respectively. These symbols could be checked during compilation and the required section of source code can be compiled. The scope of a symbol is the file in which it is defined.
- #if, #elif, #else, and #endif: To skip part of source code based on conditions. Conditional sections may be nested with directives forming complete sets.
- #line: To control line numbers generated for errors and warning. This is mostly used by meta-programming tools to generate C# source code from some text input. It is generally used to modify the line numbers and source file names reported by the compiler in its output.
- #error and #warning : To generate errors and warnings, respectively. #error is used to stop compilation, while #warning is used to continue compilation with messages in the console.
- #region and #endregion :To explicitly mark sections of source code. These allow expansion and collapse inside Visual Studio for better readability and reference.

When the C# compiler encounters an **#if** directive, followed eventually by an#endif directive, it will compile the code between the directives only if the specified symbol is defined. Unlike C and C++, you cannot assign a numeric value to a symbol; the #if statement in C# is Boolean and only tests whether the symbol has been defined or not. For example,

```
#define DEBUG
// ...
#if DEBUG
   Console.WriteLine("Debug version");
#endif
```

You can use the operators == (equality), != (inequality) only to test for true orfalse . True means the symbol is defined. The statement #if DEBUG has the same meaning as #if (DEBUG == true). You can use the operators && (and), ||(or), and ! (not) to evaluate whether multiple symbols have been defined. You can also group symbols and operators with parentheses.

```
#define VC7
//...
#if debug
   Console.Writeline("Debug build");
#elif VC7
   Console.Writeline("Visual Studio 7");
#endif
```

You can use the operators == (equality), != (inequality), **&&** (and), and ||(or), to evaluate multiple symbols. You can also group symbols and operators with parentheses.

#line lets you modify the compiler's line number and (optionally) the file name output for errors and warnings. This example shows how to report two warnings associated with line numbers. The #line 200 directive forces the line number to be 200 (although the default is #7) and until the next #line directive, the filename will be reported as "Special". The #line default directive returns the line numbering to its default numbering, which counts the lines that were renumbered by the previous directive.

```
class MainClass
{
  static void Main()
  {
#line 200 "Special"
     int i;   // CS0168 on line 200
     int j;   // CS0168 on line 201
#line default
     char c;  // CS0168 on line 9
     float f; // CS0168 on line 10
#line hidden // numbering not affected
     string s;
     double d; // CS0168 on line 13
  }
}
```

#endif specifies the end of a conditional directive, which began with the #ifdirective. For example,

```
#define DEBUG
// ...
#if DEBUG
  Console.WriteLine("Debug version");
#endif
```

Q3 (b) Describe System Environment class. Illustrate with the help of a program.

Answer

The Environment class exposes a number of extremely helpful methods beyond GetCommandLineArgs().
Specifically, this class allows you to obtain a number of details regarding the operating system currentlyhosting your .NET application using various static members. To illustrate the usefulness ofSystem.Environment, update your Main() method to call a helper method namedShowEnvironmentDetails().

```
static int Main(string[] args)
{
...
// Helper method within the Program class.
ShowEnvironmentDetails();

Console.ReadLine();
return -1;
}
static void ShowEnvironmentDetails()
{
// Print out the drives on this machine,
// and other interesting details.
foreach (string drive in Environment.GetLogicalDrives())
Console.WriteLine("Drive: {0}", drive);
Console.WriteLine("OS: {0}", Environment.OSVersion);
Console.WriteLine("Number of processors: {0}",
Environment.ProcessorCount);
Console.WriteLine(".NET Version: {0}",
Environment.Version);
}
```

Q4 (a) Differentiate between the value type and reference type. Is there any mechanism in C# to convert between value types and reference types? Explain.

Answer Page Number 128 of Text Book

Q4 (b) Explain four methods of System .Array & System. String

Answer

Clear() This static method sets a range of elements in the array to empty values (0 for numbers, null for object references, false for booleans).

CopyTo() This method is used to copy elements from the source array into the destination array.

Length This property returns the number of items within the array.

Rank This property returns the number of dimensions of the current array.

Reverse() This static method reverses the contents of a one-dimensional array.

Sort() This static method sorts a one-dimensional array of intrinsic types. If the elements in the array implement the IComparer interface

String Member Meaning in Life

Length This property returns the length of the current string.

Compare() This static method compares two strings.

Contains() This method determines whether a string contains a specific substring.

Equals() This method tests whether two string objects contain identical character data.

Format() This static method formats a string using other primitives (e.g., numerical data, other strings) and the {0} notation examined earlier in this chapter.

Insert() This method inserts a string within a given string.

Q5 (a) What is the significance of encapsulation? How it is ensured using class properties? Describe it with examples

Answer

Encapsulation Using .NET Properties

Although you can encapsulate a piece of field data using traditional get and set methods, .NET languages prefer to enforce data encapsulation state data using *properties*. First of all, understand that properties are just a simplification for "real" accessor and mutator methods. Therefore, as a class designer, you are still able to perform any internal logic necessary before making the value assignment
(e.g., uppercase the value, scrub the value for illegal characters, check the bounds of a numerical value,

Here is the updated Employee class, now enforcing encapsulation of each field using property syntax
rather than traditional get and set methods:

```
class Employee
{
// Field data.
private string empName;
private int empID;
private float currPay;
// Properties!
public string Name
{
get { return empName; }
set
{
if (value.Length > 15)
Console.WriteLine("Error! Name must be less than 16 characters!");
else
empName = value;
}
}
// We could add additional business rules to the sets of these properties;
// however, there is no need to do so for this example.
public int ID
{
get { return empID; }
set { empID = value; }
}
public float Pay
{
get { return currPay; }
set { currPay = value; }
}
...
```

```
}
public string Name
{
get { return empName; }
set
{
// Here, value is really a string.
if (value.Length > 15)
Console.WriteLine("Error! Name must be less than 16 characters!");
else
empName = value;
}
}
```

After you have these properties in place, it appears to the caller that it is getting and setting a *public*
point of data; however, the correct get and set block is called behind the scenes to preserve
encapsulation:

```
static void Main(string[] args)
{
Console.WriteLine("***** Fun with Encapsulation *****\n");
Employee emp = new Employee("Marvin", 456, 30000);
emp.GiveBonus(1000);
emp.DisplayStats();
// Set and get the Name property.
emp.Name = "Marv";
Console.WriteLine("Employee is named: {0}", emp.Name);
Console.ReadLine();
}
```

Q5 (b) What are the significance of virtual and override function? Demonstrate C# Polymorphic support with an example.

Answer

The virtual and override Keywords
Polymorphism provides a way for a subclass to define its own version of a method defined by its base class, using the process termed *method overriding*. To retrofit your current design, you need to understand the meaning of the virtual and override keywords. If a base class wants to define a method that *may be* (but does not have to be) overridden by a subclass, it must mark the method with the

virtual keyword:

```
partial class Employee
{
// This method can now be "overridden" by a derived class.
public virtual void GiveBonus(float amount)
{
        Pay += amount;
        }
        ...
        }
        class SalesPerson : Employee
        {
        ...
        // A salesperson's bonus is influenced by the number of sales.
        public override void GiveBonus(float amount)
        {
        int salesBonus = 0;
        if (SalesNumber >= 0 && SalesNumber <= 100)
        salesBonus = 10;
        else
        {
        if (SalesNumber >= 101 && SalesNumber <= 200)
        salesBonus = 15;
        else
        salesBonus = 20;
        }
        base.GiveBonus(amount * salesBonus);
        }
        }
        class Manager : Employee
        {
        public override void GiveBonus(float amount)
        {
        base.GiveBonus(amount);
        Random r = new Random();
        StockOptions += r.Next(500);
        }
        }
        static void Main(string[] args)
        {
        Console.WriteLine("***** The Employee Class Hierarchy *****\n");
        // A better bonus system!
        Manager chucky = new Manager("Chucky", 50, 92, 100000, "333-23-2322", 9000);
        chucky.GiveBonus(300);
        chucky.DisplayStats();
        Console.WriteLine();
        SalesPerson fran = new SalesPerson("Fran", 43, 93, 3000, "932-32-3232", 31);
        fran.GiveBonus(200);
fran.DisplayStats();

Console.ReadLine();

}
```

Q6 (a) Define errors and exceptions. Explain any four members of System.Exception Type.

Answer

• *User errors:* User errors, on the other hand, are typically caused by the individual running your application, rather than by those who created it. For example, an end user who enters a malformed string into a text box could very well generate an error *if* you fail to handle this faulty input in your code base.

• *Exceptions:* Exceptions are typically regarded as runtime anomalies that are difficult, if not impossible, to account for while programming your application.
Possible exceptions include attempting to connect to a database that no longer exists, opening a corrupted XML file, or trying to contact a machine that is currently offline. In each of these cases, the programmer (or end user) has little control over these "exceptional" circumstances

Core Members of the System.Exception Type
System.Exception Property Meaning in Life
Data This read-only property retrieves a collection of key/value pairs (represented by an object implementing IDictionary) that provide additional, programmer-defined information about the exception. By default, this collection is empty.

HelpLink This property gets or sets a URL to a help file or web site describing the error in full detail.

InnerException This read-only property can be used to obtain information about the previous exception(s) that caused the current exception to occur. The previous exception(s) are recorded by passing them into the constructor of the most current exception.

Message This read-only property returns the textual description of a given error. The error message itself is set as a constructor parameter.

Source This property gets or sets the name of the assembly, or the object, that threw the current exception.

StackTrace This read-only property contains a string that identifies the sequence of calls that triggered the exception. As you might guess, this property is very useful during debugging or if you wish to dump the error to an external error log.
TargetSite This read-only property returns a MethodBase object, which describes numerous details about the method that threw the exception (invoking ToString() will identify the method by name).

Q6 (b) With the help of an example, describe building custom exceptions.

Answer

While you can always throw instances of System.Exception to signal a runtime error (as shown in the first example), it is sometimes advantageous to build a *strongly typed exception* that represents the unique details of your current problem. For example, assume you want to build a custom exception (named CarIsDeadException) to represent the error of speeding up a doomed automobile. The first step is to derive a new class from System.Exception/System.ApplicationException (by convention, all exception classes end with the "Exception" suffix; in fact, this is a .NET best practice).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
   public class CarisdeadException : System.Exception
   {
      private string carname;
      public CarisdeadException() { }

      public CarisdeadException(string carname)
      {
         this.carname = carname;
      }
      public override string Message
      {
         get
         {
            string msg = base.Message;

  if (carname == null)
               msg += carname + "has null value";
            return msg;
```

```
        }
    }
    public class Car
    {
        private int curspeed;
        private int maxspeed;
        private string petname;
        bool carisdead = false;
        public Car()
        { maxspeed = 100; }

        public Car(string name, int max, int cur)
        {
            curspeed = cur;
            maxspeed = max;
            petname = name;
        }
public void speedup(int delta)
{

 if (delta==0)
        throw new ArgumentOutOfRangeException("speed<0");

if (carisdead)
        throw new CarisdeadException(this.petname);

 else
                {
                    curspeed += delta;
                    if (curspeed >= maxspeed)
                    {
        Console.WriteLine("sorry {0} has overheated", petname);
                        carisdead = true;
                    }
                    else
        Console.WriteLine("=> currspeed={0}", curspeed);
                }

            }
        }
static int Main(string[] args)
        {
            Car Buddha = new Car("Buddha", 100, 20);
            try
            {
```

```
        for (int i = 0; i < 10; i++)
            Buddha.speedup(10);
    }

    catch (CarisdeadException e)
    {
        Console.WriteLine("\n**** error***");
        Console.WriteLine("Messsge:{0}", e.Message);
       // throw e;
    }

    catch (ArgumentOutOfRangeException e)
    {
        Console.WriteLine("Messsge:{0}", e.Message);
    }

    Console.WriteLine("\n out of exception");
    Console.ReadLine();
    return 0;
  }
}
```

Q7 (a) Explain how can you implement standard IEnumerable and IEnumerator interfaces on a custom type.

Answer Page Number 293 from Text Book

Q7 (b) Explain the significance of interface. How the interfaces can be used to have Multiple Base interfaces?

Answer

In C#, when two interfaces have functions with the same name and a class implements these interfaces, then we have to specifically handle this situation. We have to tell the compiler which class function we want to implement. For such cases, we have to use the name of the interface during function implementation. Have a look at the following example:

Blocks of code should be set as style Formatted like this:

```
/// <summary />
    /// Interface 1
    /// </summary />
    public interface Interface1
    {
```

```
    /// <summary />
    /// Function with the same name as Interface 2
    /// </summary />
    void MyInterfaceFunction();
}

/// <summary />
/// Interface 2
/// </summary />
public interface Interface2
{
    /// <summary />
    /// Function with the same name as Interface 1
    /// </summary />
    void MyInterfaceFunction();
}

/// <summary />
/// MyTestBaseClass Implements the two interfaces Interface1 and Interface2
/// </summary />
public class MyTestBaseClass:Interface1,Interface2
{
    #region Interface1 Members

    void Interface1.MyInterfaceFunction()
    {
        MessageBox.Show("Frm MyInterface1 Function()");
        return;
    }

    #endregion

    #region Interface2 Members

    void Interface2.MyInterfaceFunction()
    {
        MessageBox.Show("Frm MyInterface2 Function()");
        return;
    }

    #endregion
}
```

In the above example, we are implementing the function MyInterfaceFunction() by using its interface name. In this case if we create the object of MyTestBaseClass and check for MyInterfaceFunction(), it won't be directly available. Look at the following code:

```
MyTestBaseClass obj = new MyTestBaseClass();
//Following code would give an error saying that
//class does not have a definition for MyInterfaceFunction.
obj.MyInterfaceFunction();
```

Q8 (a) Describe call back interface and write a program to demonstrate the same.

Answer

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

namespace ConsoleApplication1
{  public interface IEngineEvents
   {
      void AboutToBlow(string msg);
```

```
void Exploded(string msg);

public class careventsink:IEngineEvents
{
   private string name;
   public careventsink(){}
   public careventsink(string sinkname)
   {name=sinkname;}
   public void AboutToBlow(string msg)
   {Console.WriteLine("{0} reporting:{1}",name,msg);}
   public void Exploded(string msg)
   { Console.WriteLine("{0} reporting :{1}", name, msg);}
}

class car
{
   private int curspeed;
   private int maxspeed;
   private string petname;
   bool carisdead=false;

   public car()
   {maxspeed=100;}

   public car(string name, int max, int cur)
   {
      curspeed=cur;
      maxspeed=max;
      petname=name;
   }

   ArrayList itfconnections = new ArrayList();

   public void Advise(IEngineEvents itfclientimpl)
  { itfconnections.Add(itfclientimpl);}

   public void unadvise(IEngineEvents itfclientimpl)
   { itfconnections.Remove(itfclientimpl);}

   public void speedup(int delta)
   {
      if(carisdead)
      {
         foreach(IEngineEvents e in itfconnections)
            e.Exploded("sorry,this car is dead");
      }
      else{
         curspeed+=delta;
         if(10 == maxspeed - curspeed)
         {
            foreach(IEngineEvents e in itfconnections)
```

© IETE

Q8 (b) Describe the concept of multicast delegate using a program demonstrating it.

Answer

Multicast delegates provide functionality to execute more than one method.Internally, a linked list of delegates (called Invocation List) is stored, and when the multicast delegate is invoked, the list of delegates will be executed in sequence.

```
eclares the multicast delegate
public delegate void myDel();

static public void Main(string[] args)
{
   /// Declares the single delegate that points to MethodA
   myDel myDelA = new myDel(MethodA);
   /// Declares the single delegate that points to MethodA
   myDel myDelB = new myDel(MethodB);
   /// Declares the multicast delegate combining both delegates A and B
   myDel myMultiCast = (myDel)Delegate.Combine(myDelA, myDelB);
   /// Invokes the multicast delegate
   myMultiCast.Invoke();
}

static void MethodA()
{
   Console.WriteLine("Executing method A.");
}

static void MethodB()
{
   Console.WriteLine("Executing method B.");
}
```

o/p

Executing method A.

Executing method B.

Deriving a class from the MulticastDelegate class

The Delegate and MulticastDelegate classes cannot be derived explicitly, but there's a way to do that.The following example defines three classes named Order, Stock, and Receipt beyond the main method, from a console application, for example:

The `Order` class holds the delegate that defines the methods signature, and provides methods to add product items to the order, and a checkout method that will use a delegate as a parameter to define what method to call. It can be used for single delegates or a multicast delegate.
The `Stock` class has an operation to remove the products from the stock.
The `Receipt` class has an operation to print an item to the receipt.
The `Main` method creates an instance of the `Order` type, adding product items to the order, and creates the multicast delegate based on a combination of two derived delegates.

```
class Order
{
   /// Main order delegate
   public delegate void myOrderDel(int prodId, int quantity);

   /// Stores a dictionary of products ids and respective quantities
   private static HybridDictionary prodList = new HybridDictionary();

   public void AddItem(int prodId, int quantity)
   {
      /// Add products and quantitites to the dictionary.
      prodList.Add(prodId, quantity);
   }

   public static void Checkout(myOrderDel multicastDelegate)
   {
      /// Loop through all products in the dictionary
      foreach (DictionaryEntry prod in prodList)
      {
         /// Invoke the multicast delegate
         multicastDelegate.Invoke(Convert.ToInt32(prod.Key),
                        Convert.ToInt32(prod.Value));
      }
   }
}

class Stock
{
   public static void Remove(int prodId, int quantity)
   {
      Console.WriteLine("{0} unit(s) of the product {1} has/have" +
                  " been removed from the stock.", quantity, prodId);
   }
}

class Receipt
{
   public static void PrintItem(int prodId, int quantity)
   {
      Console.WriteLine("{0} unit(s) of the product {1} has/have been" +
                  " printed to the receipt.", quantity, prodId);
   }
```

```
static public void Main(string[] args)
{
  /// Create the order object
  Order myOrder = new Order();

  /// Add products and quantities to the order
  myOrder.AddItem(1, 2);
  myOrder.AddItem(2, 3);
  myOrder.AddItem(3, 1);
  myOrder.AddItem(4, 1);
  myOrder.AddItem(5, 4);

  /// Order delegate instance pointing to Stock class.
  Order.myOrderDel myStockDel = new Order.myOrderDel(Stock.Remove);

  /// Receipt delegate instance pointing to Receipt class.
  Order.myOrderDel myReceiptDel = new Order.myOrderDel(Receipt.PrintItem);

  /// Combine the two previous delegates onto the multicast delegate.
  Order.myOrderDel myMulticastDel =
   (Order.myOrderDel)Delegate.Combine(myStockDel, myReceiptDel);

  /// Invoke the checkout method passing the multicast delegate
  Order.Checkout(myMulticastDel);
}
```

Output

```
2 unit(s) of the product 1 has/have been removed from the stock.
2 unit(s) of the product 1 has/have been printed to the receipt.
3 unit(s) of the product 2 has/have been removed from the stock.
3 unit(s) of the product 2 has/have been printed to the receipt.
1 unit(s) of the product 3 has/have been removed from the stock.
1 unit(s) of the product 3 has/have been printed to the receipt.
1 unit(s) of the product 4 has/have been removed from the stock.
1 unit(s) of the product 4 has/have been printed to the receipt.
4 unit(s) of the product 5 has/have been removed from the stock.
4 unit(s) of the product 5 has/have been printed to the receipt.
```

Q9 (a) Explain the steps to build a shared assembly.

Answer

Step 1: Create a class file

```
using System;
namespace SharedAssembly
{
    public class Bike
    {
        public void start()
        {
            Console.WriteLine("kick start ");
        }
    }
}
```

Step 2: Generate the token

This token known as strong name key. It is the string that is large collection of alphabet and numeric value, it is very big-string. It is also so long so that no one can access without any permission.

--> open command prompt of visual studio

D:\shared assembly> sn -k shrd.snk

This will create a key having name shrd.snk(128 bit key).

Note: To generate the key it use RSA2 (Rivest Shamir Alderman) algorithm.

Step 3: Apply the key on our class file by written the code in .dll source file

```
// add this code to your class file
using System.Reflection;
[assembly:AssemblyKeyFile("shrd.snk")]
```

Class file

```
using System;
using System.Reflection;
[assembly:AssemblyKeyFile("shrd.snk")]
namespace SharedAssembly
{
    public class Bike
    {
        public void start()
        {
            Console.WriteLine("kick start ");
        }
    }
}
```

Step 4: Complied the code file & Create a .dll file of Bike class

Step 5: Now register/install .dll into GAC. GAC is the database of CLR in case of .NET.After installing this .dll in GAC, any file can use it with the help of CLR.

To install

D:\shared assembly>gacutil /i sharedAss.dll

Step 6: The Client Application are as follow

```
using System;
using SharedAssembly;
public class MainP
```

```
{
  public static void Main(string []args)
  {
   Bike bk=new Bike();
   bk.start();
   Console.Read();
  }
}
```

1. Step 7: Compiled the whole application

```
D:\> csc /r:d:\shared assembly\sharedAss.dll MainP.cs
```

Step 8: Run your program by using the command given below:

```
D:\>MainP
```

Q9 (b) Write short notes on any <u>TWO</u>:

(i) VS .NET Add References Dialog Box

(ii) GAC Internals

(iii) Versioning shared Assemblies

Answer

(i) Page Number 449 of Text Book

(ii) Page Number 440 of Text Book

(iii) Page Number 437 of Text Book

<u>Text Book</u>

C# and the .NET Platform, Andrew Troelsen, II Edition 2003, Dreamtech Press