

Topics:

- What is a Class?
- What are Members?
- What is Constructor?
- Constructor Overloading
- Method Overloading
- Inheritance
- Visual Inheritance
- Method Overriding
- Abstract Class
- Final class
- Interfaces
- Encapsulation

Object Oriented Programming introduced to overcome the drawback that exists in structured programming language.

The drawback of Structured programming languages are:

Error debugging because a difficult task one it exceeds certain lines of codes.

Lack of security for members

Lack of reusability.

C# is pure object oriented that supports all OOP features.

Class

A class is an abstract data type that represents a real-world entity. A class can be defined in the C# using the following syntax.

```
<access-specifier> class <classname>
{
    Members(we'll discuss in coming session)
}
```

Comment [S1]: Predefined keyword

Comment [S2]: Where classname is the name of the class

Where **access-specifier** specify how the class can be accessed within the application. **Classname** is the name of the class. Always the classname should maintain in **pascal case** is the good programming practice. Once the class can be defined a new object from the class is defined using the following syntax:

Comment [S3]: Examples: TestClass
SampleTest

Type 1

```
<classname> obj=new <classname>
```

Example: Student s=new Student();

Type 2

```
<classname> obj; Example: Student s;
```

Obj=new <classname> **Example:** s=new Student();

A class can be defined in C# using any one of the following

Define the class within the windows application forms class definition.

Project> Add class

In the dialog select class template and click "Add" button.

Example:

Create a class called student and invoke the student object from the form.

Step 1: Project> Add class> Student.cs> Add

Step 2: Add the following namespace.

Using System.Windows.Form

Step 3: Add the following information inside class block.

```

public int stid, fee;
public string name, course;
public void Display()
{
    String s="";
    s=s+"Student Id:"+stid+"\n";
    s=s+"Name:"+name+"\n";
    s=s+"Course:"+course+"\n";
    s=s+"Fee:"+fee+"\n";
    MessageBox.Show(s,"Student Details", MessageBoxButtons.OK, MessageBoxIcon.Information);
}

```

Step 4: create the following from

Step 5: Add the following event logic

```

//Display
Student st = new Student();
st.stid = int.Parse(textBox1.Text);
st.name = textBox2.Text;
st.course = textBox3.Text;
st.fee = int.Parse(textBox4.Text);
st.Display();

//Clear
foreach (Control ctrl in this.Controls)
    if (ctrl is TextBox)
        ctrl.Text = "";
textBox1.Focus();

//close
this.Close();

```

Comment [S4]: It is a specialized loop that is used to access the collections (this.controls)

Comment [S5]: A Method that is used to set the input focus for the textbox

Members

A class can contain members that can be defined by the user. Based on the functionality of the member it can be classified into any one of the following.

1. Function
2. Property
3. Event
4. Enumerator

1. Function:

This function is a set of statements that can be used for achieving a specific task. In general it returns a value. If the function does not return a value its return type is specified as **void** and is called void function.

Syntax:

```
<access-specifier> <returntype> <method name>([parameter1],[parameter2],....)
{
}
```

2. Property:

A property is a set of statements that is used to define the properties for object. The advantage of the property procedure is that it can be customized according to the requirements.

Syntax:

```
<access-specifier> <return-type> <propertyname>
{
    get
    {
        statement(s) //return property value
    }
    set
    {
        statement(s) //Initializes the property
    }
}
```

3. Event:

An event is a user generated action. These procedures are generally associated with a control. The procedure gets executed in response to the user interaction with the control. These procedures do not return any value.

Syntax:

```
<access-specifier> void <controlname>-<Eventname>(Parameter1, parameter2,....)
{
    statement(s)
}
```

4. Enumerator

An enumerator is a list of constants that can be utilized as required. It does not return any value.

Syntax:

```
<access-specifier> enum <Enumerator-name>
{
    constants
}
```

Example:

Create a class called student and invoke the student object from the form.

Step 1: Project> Add class> StudentResults.cs> Add

Step 2: Add the following namespace.
Using System.Windows.Form

Step 3: Add the following information inside class block.

```

int stid, m1, m2, m3;
string name;
public int StudentId
{
    get{return stid;}
    set{stid=value;}
}
public string sname
{
    get{return name;}
    set{name=value;}
}
public int MathsMarks
{
    get { return m1; }
    set { m1 = value; }
}
public int PhysicsMarks
{
    get { return m2; }
    set { m2 = value; }
}
public int ChemistryMarks
{
    get { return m3; }
    set { m3 = value; }
}
public int Total()
{
    return (m1 + m2 + m3);
}
public float Average()
{
    return (Total() / 3.0f);
}
public string Result()
{
    if (m1 > 34 && m2 > 34 && m3 > 34)
    {
        return("Pass");
    }
    else
    {
        return("Fali");
    }
}
public string Division()
{
    if (Result() == "Pass")
    {
        if (Average() >= 75)
            return ("Distintion");
        else if (Average() >= 60)
            return ("First Class");
        else if (Average() >= 50)
            return ("Second Class");
        else
            return ("Third");
    }
    else
    {

```

```

        return ("Nil1");
    }
}
public void Display()
{
    string s = "";
    s = s+"Student Id:" + StudentId + "\n";
    s = s+"Name:" + sname + "\n";
    s = s+"Marks in Maths:" + MathsMarks + "\n";
    s = s+"Marks in Physics:" + PhysicsMarks+ "\n";
    s = s+"Marks in Chemistry:" + ChemistryMarks+ "\n";
    s = s+"Total:" + Total() + "\n";
    s = s+"Average:" + Average() + "\n";
    s = s+"Result:" + Result() + "\n";
    s = s+"Division:" + Division() + "\n";
    MessageBox.Show(s, "Student Results", MessageBoxButtons.OK, MessageBoxIcon.Information);
}

```

Step 4: create the following from

Step 5: Add the following event logic

```

//Display
StudentResults sr = new StudentResults();
sr.StudentId=int.Parse(textBox1.Text);
sr.sname=textBox2.Text;
sr.MathsMarks=int.Parse(textBox3.Text);
sr.PhysicsMarks=int.Parse(textBox4.Text);
sr.ChemistryMarks=int.Parse(textBox5.Text);

textBox6.Text=sr.Total().ToString();
textBox7.Text=sr.Average().ToString();
textBox8.Text=sr.Result().ToString();
textBox9.Text=sr.Division().ToString();

sr.Display();

//Clear
foreach (Control ctrl in this.Controls)
    if (ctrl is TextBox)
        ctrl.Text = "";
textBox1.Focus();
//close
this.Close();

```

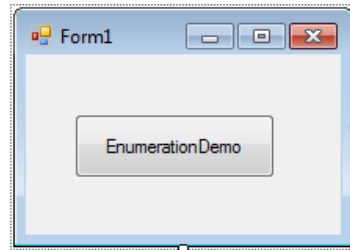
Comment [S6]: It is a specialized loop that is used to access the collections (this.controls)

Comment [S7]: A Method that is used to set the input focus for the textbox

Enumerator Example

Nagendra Prasad

Step 1: Create a form as following



```
enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };  
//EnumerationDemo  
int WeekdayStart = (int)Days.Mon;  
int WeekdayEnd = (int)Days.Fri;  
MessageBox.Show(WeekdayStart.ToString());  
MessageBox.Show(WeekdayEnd.ToString());
```

Define a constructor

A constructor is a special method of a class that is used to initialize the instance variables of a class. A default constructor is automatically created when a class is defined. The default constructor does not any parameters.

- A Constructor never returns any value
- Constructors can be overloaded.
- A constructor has the same name as the classname.

Syntax:

```
<access-specifier> <classname>(<parameter1,parameter2...>)  
{  
    statement(s)  
}
```

Example: Create a class called employee with constructor to invoke through a form.

Step 1: Project> Add class> Employee.cs> Add

Step 2: Add the following namespace.
Using System.Windows.Form

Step 3: Add the following information inside class block.

```

int eid, sal;
string name, job;
public Employee(int Empid, string Name, string Job, int salary)
{
    eid = Empid;
    name = Name;
    job = Job;
    sal = salary;
}
public void Display()
{
    string s = "";
    s = s + "Employee Id:" + eid + "\n";
    s = s + "Name:" + name + "\n";
    s = s + "Job:" + job + "\n";
    s = s + "Salary:" + sal + "\n";
    MessageBox.Show(s, "Employee Details", MessageBoxButtons.OK, MessageBoxIcon.Information);
}

```

Step 4: create the following from

Step 5: Add the following event logic

```

//Display
int eid = int.Parse(textBox1.Text);
int sal = int.Parse(textBox4.Text);
string name = textBox2.Text;
string job = textBox3.Text;
Employee emp = new Employee(eid, name, job, sal);
emp.Display();

//Clear
foreach (Control ctrl in this.Controls)
    if (ctrl is TextBox)
        ctrl.Text = "";
        textBox1.Focus();
//close
this.Close();

```

Comment [S8]: It is a specialized loop that is used to access the collections (this.controls)

Comment [S9]: A Method that is used to set the input focus for the textbox

Constructor overloaded

A constructor can be overloaded when it needs the following conditions.

1. It has the same name as a classname
2. Two or more constructor with the same name but with different parameters.
3. Two or more constructor with the same parameter but different data types.
4. Overloading is an indirect way of implementing polymorphism.

Example Create an interface to demonstrate constructor overloaded

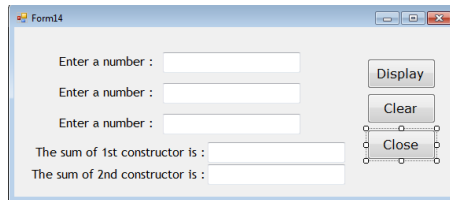
Step 1: Project> Add class> `ConstructorOverloadDemo.cs`> Add

Step 2: Add the following namespace.
`Using System.Windows.Forms`

Step 3: Add the following information inside class block.

```
int x, y, z;
public ConstructorOverloadDemo(int a, int b)
{
    x = a;
    y = b;
}
public ConstructorOverloadDemo(int a, int b, int c)
{
    x = a; y = b; z = c;
}
public int sum()
{
    return (x + y + z);
}
```

Step 4: create the following from



```
//display
int x, y, z;
x = int.Parse(textBox1.Text);
y = int.Parse(textBox2.Text);
z = int.Parse(textBox3.Text);

ConstructorOverloadDemo co = new ConstructorOverloadDemo(x, y);
ConstructorOverloadDemo co1 = new ConstructorOverloadDemo(x, y, z);
textBox4.Visible = true;
textBox5.Visible = true;
textBox4.Text = co.sum().ToString();
textBox5.Text = co1.sum().ToString();

//Clear
foreach (Control ctrl in this.Controls)
    if (ctrl is TextBox)
        ctrl.Text = "";
textBox1.Focus();

//close
this.Close();
```

Method Overloading

Two or more methods with the same name but with different parameters. Two or more methods with the same name and same parameters but different return type are called method overloading. C# compiler identity the method name uniquely based on its signature includes, return type, method name and parameters.

Create an interface to demonstrate method overloading

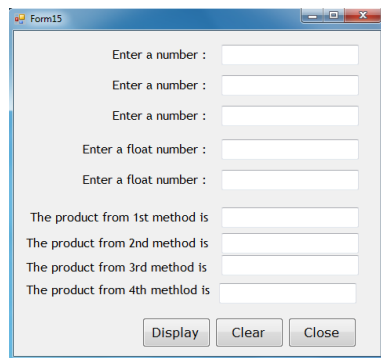
Step 1: Project> Add class> `ConstructorOverloadDemo.cs`> Add

Step 2: Add the following namespace.
Using System.Windows.Forms

Step 3: Add the following information inside class block.

```
public int product(int x)
{
    return(x*x);
}
public int product(int x,int y)
{
    return(x*y);
}
public int product(int x,int y,int z)
{
    return(x*y*z);
}
public float product(float x,float y)
{
    return(x*y);
}
```

Step 4: create the following from



//Display

```
MethodOverloadDemo mod = new MethodOverloadDemo();
int x, y, z;
float p,q;
x = int.Parse(textBox1.Text);
y = int.Parse(textBox2.Text);
z = int.Parse(textBox3.Text);
p = float.Parse(textBox7.Text);
q = float.Parse(textBox6.Text);
textBox4.Text = mod.product(x).ToString();
textBox5.Text = mod.product(x,y).ToString();
textBox9.Text = mod.product(x,y,z).ToString();
textBox8.Text = mod.product(p,q).ToString();
```

//Clear

```
foreach (Control ctrl in this.Controls)
    if (ctrl is TextBox)
        ctrl.Text = "";
textBox1.Focus();
```

//close

```
this.Close();
```

Inheritance

Inheritance is a mechanism by which the properties and methods of an existing class can be accessed by another class. The class that is already existing and it is above to be inherited by other classes is called main/parent/root/base class. The class that is going to acquire the properties of existing class is called a child/sub/derived class. Inheritance is implemented to colon (:).

Ex: class DerivedClass : BaseClass

In C# at a time only one class can be inherited that is it does not support multiple inheritances.

C#
VB.NET
Java

Base
Mybase
Super

Step 1: Project> Add class> A.cs> Add

Step 2: Add the following namespace.
Using System.Windows.Forms

Step 3: Add the following information inside class block.

```
class A
{
    public void Display()
    {
        MessageBox.Show("This is Base Class-A");
    }
}
```

Step 4: Project> Add class> B.cs> Add

Step 5: Add the following namespace.
Using System.Windows.Forms

Step 6: Add the following information inside class block.

```
class B : A
{
    public void Show()
    {
        base.Display();
        MessageBox.Show("This is Derived Class-B");
    }
}
```

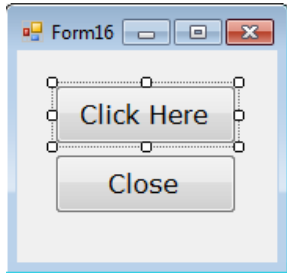
Step 6: Project> Add class> C.cs> Add

Step 7: Add the following namespace.
Using System.Windows.Forms

Step 8: Add the following information inside class block.

```
class C : B
{
    public void Print()
    {
        base.Display();
        MessageBox.Show("This is Derived Class-C");
    }
}
```

Step 4: create the following from



```
//Click Here
A a = new A();
B b = new B();
C c = new C();

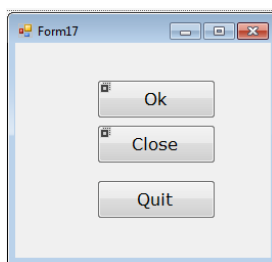
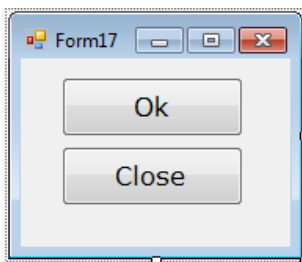
/* a.Display(); b.Display();
b.Show(); c.Show();*/
c.Print(); //c.Display();

//close
this.Close();
```

Visual Inheritance

Java does not support visual inheritance.

Visual Inheritance is a new feature that supports the extended inheritance to a graphical user interface. This helps us to build complex applications. Visual inheritance can be implemented in .NET(Network Enabled Technology).



```
//Ok
MessageBox.Show("This is a Base form method calling");
//close
this.Close();
//Quit
this.Close();
```

Observations

- The scopes of the controls that are placed within the form are private by default. This can be changed by setting the modifier property. All forms defined within the application are public by default. They inherit System.Windows.Forms.Form by default.
- When the inherited form is applied from the user defined form then the derived class will inherit the user defined form that is already inherited from System.Windows.Forms.Form.

Method Overriding

A method defined in the base class can be changed with regard to its form the base class to the sub-class is method overriding.

- The method contains the same name & same parameters both in the main class and the subclass.

- The method should be defined in the virtual base class.
- The method should be override in the subclass.

Method overriding is implemented only through inheritance

Step 1: Project> Add class> `PartTimeEmployee.cs`> Add

Step 2: Add the following namespace.
Using System.Windows.Forms

Step 3: Add the following information inside class block.

```
class PartTimeEmployee
{
    public virtual int payment(int NoOfHours, int Rate)
    {
        return (NoOfHours * Rate);
    }
}
```

Step 4: Project> Add class> `FullTimeEmployee.cs`> Add

Step 5: Add the following namespace.
Using System.Windows.Forms

Step 6: Add the following information inside class block.

```
class FullTimeEmployee : PartTimeEmployee
{
    public override int payment(int NoOfHours, int Rate)
    {
        //return(30* base.payment(NoOfHours,Rate));
        return (30 * NoOfHours * Rate);
    }
}
```

Design the interface as follows:

```
//Display
PartTimeEmployee pte = new PartTimeEmployee();
FullTimeEmployee fte = new FullTimeEmployee();
int NoOfHours, Rate;
NoOfHours = int.Parse(textBox1.Text);
Rate = int.Parse(textBox2.Text);
textBox3.Text = pte.payment(NoOfHours,
Rate).ToString();
textBox4.Text = fte.payment(NoOfHours,
Rate).ToString();
//Clear
foreach (Control ctrl in this.Controls)
    if (ctrl is TextBox)
        ctrl.Text = "";
textBox1.Focus();
//close
this.Close();
```

Abstract Class

An abstract class provides higher security for the members by hiding actual details of the respective class. Abstract class can be defined by using the abstract keyword. An abstract class should contain atleast one abstract method. These abstract methods are to be overridden in the subclasses. An abstract method should be inherited by other class.

Create an interface to be defined abstract class and implements its number.

Step 1: Project> Add class> `Shape.cs`> Add

Step 2: Add the following namespace.
Using System.Windows.Forms

Step 3: Add the following information inside class block.

```
abstract class Shape
{
    public abstract void FindArea();
}
```

Step 4: Project> Add class> `Circle.cs`> Add

Step 5: Add the following namespace.
Using System.Windows.Forms

Step 6: Add the following information inside class block.

```
class Circle:Shape
{
    public double radius;
    public override void FindArea()
    {
        double area = (Math.PI*radius * radius);
        MessageBox.Show("The area of circle is " + area.ToString(), "Circle Area",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

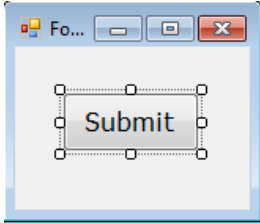
Step 7: Project> Add class> `Rectangle.cs`> Add

Step 8: Add the following namespace.
Using System.Windows.Forms

Step 9: Add the following information inside class block.

```
class Rectangle:Shape
{
    public int l, b;
    public override void FindArea()
    {
        int area = (l * b);
        MessageBox.Show("The area of rectangle is " + area.ToString(), "Rectangle Area",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

Design the interface as follows:



```
//submit

//Shape s = new Shape();
Shape s;
Circle c = new Circle();
c.radius=12.765;
c.FindArea();

Rectangle r = new Rectangle();
r.l = 10;
r.b = 20;
r.FindArea();
```

Note:

- It is not possible to create an object instance for an Abstract class.
- A reference can be created for abstract class.

Final Class

Class functionality should not be extended to another class then be declared as final class. Once the class is declared final we cannot be inherited by any other class. In c#, a final class is declared using sealed keyword.

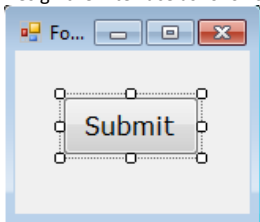
Step 1: Project> Add class> `FinalClassDemo.cs`> Add

Step 2: Add the following namespace.
Using System.Windows.Forms

Step 3: Add the following information inside class block.

```
sealed class FinalClassDemo
{
    public void Display()
    {
        MessageBox.Show("This is a final class", "FinalClass", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
    }
}
```

Design the interface as follows:



```
//submit

FinalClassDemo fcd = new FinalClassDemo();
fcd.Display();
```

Abstract

Abstract classes contains abstract methods
It is not possible to define an object for abstract class.
This class can be inherited by other class
The members of abstract class are overridden in the class that inherits the abstract class

Final

A final class cannot contain abstract methods.
It allows to create object.
This class cannot be inherited by other classes.
The members of final class cannot be overridden.

Interfaces

An interface provides highest level of abstraction that contains only the abstract methods.

1. Interfaces are implemented to a class using the colon operator (:)
2. An interface contains only method declarations.
3. The class that implement interface should define all the methods.
4. The interface & its members are public by default. You cannot change these access specifiers.

Syntax:

```
Interface <interface_name>
{
    members
}
```

Step 1: Project> Add interface> `IShape.cs`> Add

Step 2: Add the following namespace.
`Using System.Windows.Forms`

Step 3: Add the following information inside class block.

```
interface IShape
{
    int Rectangle(int lenght, int breadth);
    double CircleArea(float radius);
    int BoxArea(int length, int breadth, int height);
}
```

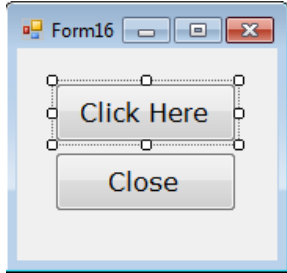
Step 1: Project> Add Class> `InterfaceDemo.cs`> Add

Step 2: Add the following namespace.
`Using System.Windows.Forms`

Step 3: Add the following information inside class block.

```
public int Rectangle(int length, int breadth)
{
    return(length * breadth);
}
public double CircleArea(float radius)
{
    return (Math.PI * radius * radius);
}
public int BoxArea(int length, int breadth, int height)
{
    return (length * height * breadth);
}
```

Design the interface as follows:



```
//Click Here
InterfaceDemo id = new InterfaceDemo();
string s = "";
s = s+"The area of rectangle is " + id.Rectangle(10,
20) +"\n";
s = s+"The area of circle is " +
id.CircleArea(12.6755f) + "\n";
s = s+"The area of BoxArea is " + id.BoxArea(10,
20,30) +"\n";
MessageBox.Show(s, "Interface Demo",
MessageBoxButtons.OK, MessageBoxIcon.Information);
//close
this.Close();
```

Encapsulation

Encapsulation is defined 'as the process of enclosing one or more items within a physical or logical package'. Encapsulation, in object oriented programming methodology, prevents access to implementation details.

Abstraction and encapsulation are related features in object oriented programming. Abstraction allows making relevant information visible and encapsulation enables a programmer to *implement the desired level of abstraction*.

Encapsulation is implemented by using **access specifiers**. An **access specifier** defines the scope and visibility of a class member. C# supports the following access specifiers:

- Public
- Private
- Protected
- Internal
- Protected internal

Public Access Specifier

Public access specifier allows a class to expose its member variables and member functions to other functions and objects. Any public member can be accessed from outside the class.

Private Access Specifier

Private access specifier allows a class to hide its member variables and member functions from other functions and objects. Only functions of the same class can access its private members. Even an instance of a class cannot access its private members.

Protected Access Specifier

Protected access specifier allows a child class to access the member variables and member functions of its base class.

Internal Access Specifier

The internal access specifier hides its member variables and methods from other classes and objects, that is resides in other namespace. The variable or classes that are declared with internal can be access by any member within application. It is the default access specifiers for a class in C# programming.

Protected Internal Access Specifier

The protected internal access specifier allows its members to be accessed in derived class, containing class or classes within same application. However, this access specifier rarely used in C# programming but it becomes important while implementing inheritance.

Example

Step 1: Create the following 5 classes using the following code.

```

PublicAccess.cs
// String Variable declared as public
public string name;
// Public method
public void print()
{
    MessageBox.Show("Welcome :"+name);
}

PrivateAccess.cs
// String Variable declared as public
private string name="";
// Public method
public void print()
{
    MessageBox.Show("Welcome :" + name);
}

ProtectedAccess.cs
// String Variable declared as public
protected string name;
// Public method
public void print()
{
    MessageBox.Show("Welcome :" + name);
}

InternalAccess.cs
// String Variable declared as internal
internal string name;
// Public method
public void print()
{
    MessageBox.Show("Welcome :" + name);
}

ProtectedInternalAccess.cs
// String Variable declared as ProtectedInternalAccess
protected internal string name;
// Public method
public void print()
{
    MessageBox.Show("Welcome :" + name);
}

//Public
PublicAccess ac = new PublicAccess();
// Accepting value in public variable that is outside the class
ac.name = textBox1.Text;
ac.print();

//Private
/*PrivateAccess ac = new PrivateAccess();
// Accepting value in private variable that is outside the class
ac.name = textBox1.Text;
ac.print();*/

//Protected
ProtectedAccess ac = new ProtectedAccess();
// Accepting value in protected variable that is outside the class
ac.name = textBox1.Text;
ac.print();

//Internal

```

```
InternalAccess ac = new InternalAccess();  
// Accepting value in internal variable that is outside the class  
ac.name = textBox1.Text;  
ac.print();  
//Protected internal  
ProtectedInternalAccess ac = new ProtectedInternalAccess();  
// Accepting value in internal variable that is outside the class  
ac.name = textBox1.Text;  
ac.print();
```