C# - Encapsulation

Previous Page Next Page

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.

The following example illustrates this:

```
using System;
namespace RectangleApplication
{
   class Rectangle
   {
      //member variables
      public double length;
      public double width;

      public double GetArea()
      {
         return length * width;
      }
      public void Display()
      {
         Console.WriteLine("Length: {0}", length);
         Console.WriteLine("Width: {0}", width);
         Console.WriteLine("Area: {0}", GetArea());
      }
   }//end class Rectangle

   class ExecuteRectangle
   {
      static void Main(string[] args)
      {
         Rectangle r = new Rectangle();
         r.length = 4.5;
         r.width = 3.5;
         r.Display();
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
Length: 4.5
Width: 3.5
Area: 15.75
```

In the preceding example, the member variables length and width are declared **public**, so they can be accessed from the function Main() using an instance of the Rectangle class, named **r**.

The member function *Display()* and *GetArea()* can also access these variables directly without using any instance of the class.

The member functions *Display()* is also declared **public**, so it can also be accessed from *Main()* using an instance of the Rectangle class, named **r**.

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.

The following example illustrates this:

```
using System;
namespace RectangleApplication
{
   class Rectangle
   {
      //member variables
      private double length;
      private double width;

      public void Acceptdetails()
      {
         Console.WriteLine("Enter Length: ");
         length = Convert.ToDouble(Console.ReadLine());
         Console.WriteLine("Enter Width: ");
         width = Convert.ToDouble(Console.ReadLine());
      }
      public double GetArea()
      {
         return length * width;
      }
      public void Display()
      {
         Console.WriteLine("Length: {0}", length);
         Console.WriteLine("Width: {0}", width);
         Console.WriteLine("Area: {0}", GetArea());
      }
   }//end class Rectangle

   class ExecuteRectangle
   {
      static void Main(string[] args)
      {
         Rectangle r = new Rectangle();
         r.Acceptdetails();
         r.Display();
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
Enter Length:
4.4
Enter Width:
3.3
Length: 4.4
Width: 3.3
Area: 14.52
```

In the preceding example, the member variables length and width are declared **private**, so they cannot be accessed from the function Main(). The member functions *AcceptDetails()* and *Display()* can access these variables. Since the member functions *AcceptDetails()* and *Display()* are declared **public**, they can be accessed from *Main()* using an instance of the Rectangle class, named **r**.

Protected Access Specifier

Protected access specifier allows a child class to access the member variables and member functions of its base class. This way it helps in implementing inheritance. We will discuss this in more details in the inheritance chapter.

Internal Access Specifier

Internal access specifier allows a class to expose its member variables and member functions to other functions and objects in the current assembly. In other words, any member with internal access specifier can be accessed from any class or method defined within the application in which the member is defined.

The following program illustrates this:

```
using System;
namespace RectangleApplication
{
   class Rectangle
   {
      //member variables
      internal double length;
      internal double width;

      double GetArea()
      {
         return length * width;
      }
      public void Display()
      {
         Console.WriteLine("Length: {0}", length);
         Console.WriteLine("Width: {0}", width);
         Console.WriteLine("Area: {0}", GetArea());
      }
   }//end class Rectangle

   class ExecuteRectangle
   {
      static void Main(string[] args)
      {
         Rectangle r = new Rectangle();
         r.length = 4.5;
         r.width = 3.5;
         r.Display();
         Console.ReadLine();
      }
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
Length: 4.5
Width: 3.5
Area: 15.75
```

In the preceding example, notice that the member function *GetArea()* is not declared with any access specifier. Then what would be the default access specifier of a class member if we don't mention any? It is **private**.

Protected Internal Access Specifier

The protected internal access specifier allows a class to hide its member variables and member functions from other class objects and functions, except a child class within the same application. This is also used while implementing inheritance.

Previous Page Next Page