Access modifiers specify the accessibility of an object and all of its members in the C# project. Moreover, all the C# types have access modifiers implemented, even if they are not stated (default access modifier is applied then).

Even though this topic is more related to the object-oriented concept, we will talk about it now, thus making easier to understand the next article about methods, that strongly relies on access modifiers.

Access Modifiers Types

C# provides four types of access modifiers: private, public, protected, internal, and two combinations: protected-internal and private-protected.

Private Access Modifier

Objects that implement private access modifier are accessible only inside a class or a structure. As a result, we can't access them outside the class they are created:

```
class NumberClass
{
   private int number = 10;
}
class Program
{
   static void Main(string[] args)
   {
      NumberClass num = new NumberClass();
      Console.WriteLine(num.number); // Error. We can't access the number variable
because
      // it has the private access modifier and its only accessible in the NumberClass class
   }
}
```

Public Access Modifier

Objects that implement public access modifiers are accessible from everywhere in our project. Therefore, there are no accessibility restrictions:

```
class NumberClass
{
   public int number = 10;
}
class Program
{
   static void Main(string[] args)
   {
      NumberClass num = new NumberClass();
      Console.WriteLine(num.number); // This is OK. The number variable has the public
access modifier.
   }
```

Protected

Variables, methods & constructors which are declared protected in a superclass can be accessed only by the subclass in other package or any class within the package of protected members class.

Protected Access Modifier

The protected keyword implies that the object is accessible inside the class and in all classes that derive from that class. We will talk in more detail about inheritance in module 2 about object-oriented programming. But for now, we are going to take a look at this example to understand the behavior of the protected members:

```
class NumberClass
{
   protected int number = 10; //we can access this variable inside this class
}
class DerivedClass: NumberClass //this is inheritance. DerivedClass derives from the
NumberClass class
{
   void Print()
   {
      Console.WriteLine(number); //we can access it in this class as well because it derives
from the NumberClass class
   }
}
class Program
{
   void Print()
   {
      NumberClass num = new NumberClass();
      Console.WriteLine(num.number); // Error. The number variable is inaccessible due to its
protection level.
                        // The Program class doesn't derive from the NumberClass
   }
}
```

Internal Access Modifier

The internal keyword specifies that the object is accessible only inside its own assembly but not in other assemblies:

```
//First Project (ASSEMBLY)
public class NumberClassInFirstProject
{
   internal int number = 10; //we can access this variable inside this class
}
class ProgramInFirstProject
{
   void Print()
   {
      NumberClassInFirstProject num = new NumberClassInFirstProject();
      Console.WriteLine(num.number); // This is OK. Anywhere in this project (assembly)
```

```
                                    // we can access the number variable.
    }
}
//Second project (ASSEMBLY)
class Program
{
    void Print()
    {
        NumberClassInFirstProject num = new NumberClassInFirstProject();
        Console.WriteLine(num.number); // Error. The number variable is inaccessible due to its
protection level.
                                //The Program class in second project can't access the internal members
from another project
    }
}
```

Protected Internal Access Modifier

The protected internal access modifier is a combination of protected and internal. As a result, we can access the protected internal member only in the same assembly or in a derived class in other assemblies (projects):

```
//First Project (ASSEMBLY)
public class NumberClassInFirstProject
{
    protected internal int number = 10; //we can access this variable inside this class
}
class ProgramInFirstProject
{
    void Print()
    {
        NumberClassInFirstProject num = new NumberClassInFirstProject();
        Console.WriteLine(num.number); // This is OK. Anywhere in this project (assembly) we
can access the number variable.
    }
}
//Second project (ASSEMBLY)
class Program: NumberClassInFirstProject //Inheritance
{
    void Print()
    {
        Console.WriteLine(number); //This is OK as well. The class Program derives from the
NumberClassInFirstProject clas.
    }
}
```

Private Protected Access Modifier

The private protected access modifier is a combination of the private and protected keywords. We can access members inside the containing class or in a class that derives

from a containing class, but only in the same assembly(project). Therefore, if we try to access it from another assembly, we will get an error.

private ⇒ Member is accessible only within the class containing the member

Public ⇒ Member is accessible from everywhere outside the class as well as. It is also accessible in derived classes.

protected ⇒ Member is visible only to its own class and its derived classes

internal ⇒ Member is available within the assembly or component that is being created but not to the clients of that component.

protected internal ⇒ Available in the containing program or assembly in the derived classes.