Access Modifiers

This lesson discusses access modifiers like public, private, protected and protected internal

WE'LL COVER THE FOLLOWING

- public
- private
- protected internal
- protected

public

The `public` *keyword* makes a *class* (including *nested classes*), *property*, *method* or *field* available to every consumer:

```
using System;

public class Dog
{
   //these public attributes of class are available in other methods and classes
    public string name = "lucy";
    public string gender  = "female";
    public int age = 5;
    public int size = 5;
    public bool healthy = true;
};

class PublicExample
{
    static void Main()
    {
      Dog dogObj = new Dog(); //making object of Dog class

      //the public members of class Dog can be accessed directly using dot operator
      Console.WriteLine("Doggo's name is: {0}",dogObj.name);
      Console.WriteLine("Doggo's gender is: {0}",dogObj.gender);
      Console.WriteLine("Doggo's age is: {0}",dogObj.age);
      Console.WriteLine("Doggo's size is: {0}",dogObj.size);
      Console.WriteLine("Is Doggo healthy? {0}",dogObj.healthy);
    }
}
```

private

The `private` keyword marks *properties*, *methods*, *fields* and *nested classes* for use inside the *class* only.

In order to *access* or *set* the values of `private` *variables* outside of the *class* `get` and `set` *methods* are used. This will be discussed in an upcoming lesson.

> **Note:** Since `private` members cannot be accessed outside of the *class* using `.` operator the code below will give run time **errors**.

```
using System;

class Dog
{
 //class members without the keyword public are by default considered private
    string name = "lucy";
    string gender = "female";
    int size = 5;
    bool healthy = true;
    int age = 2;
};

class PrivateExample
{
    static void Main()
    {
      Dog dogObj = new Dog();  //making object of Dog class

      //this block of code will give errors as all the class members being accessed are priva
      //private members CANNOT be accessed directly using dot operator outside of the class i
      Console.WriteLine("Doggo's age is: ",dogObj.age);
      Console.WriteLine(dogObj.name);
      Console.WriteLine(dogObj.gender);
      Console.WriteLine(dogObj.size);
      Console.WriteLine(dogObj.healthy);
    }
}
```

protected internal

The `protected internal` *keyword* marks *field*, *methods*, *properties* and *nested classes* for use inside the same assembly or derived classes in another

assembly:

```
using System;

public class Example
{
    protected internal int value = 5;
    protected internal class MyProtectedInternalNestedClass //nested class
    {
        public int temp = 4;
    }
}

class ProtectedInternalExample
{
    static void Main()
    {
        Example Obj = new Example();

      //the protected internal class and its member is accessible by the current assembly
        var obj2 = new Example.MyProtectedInternalNestedClass();
        Console.WriteLine("value is: {0}",Obj.value);
        Console.WriteLine("temp is: {0}", obj2.temp);

    }
}
```

protected

The `protected` *keyword* marks *field, methods properties* and *nested classes* for use inside the same *class* and *derived classes* only.

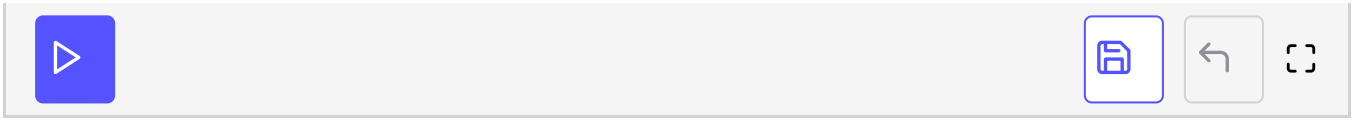
> **Note:** You will get runtime **errors** when you run the code below as the *variables* in *class* will be *inaccessible* outside of it due to their **protection** level.

```
// This code will give error as
//protected members of the class Dog cannot be accessed outside of the class
using System;

public class Dog
{
  int Age=5;
  protected string gender="female";
 protected void getAge()
 {
   Console.WriteLine("Doggo's age is: {0}", Age);
 }
```

```
}

class ProtectedExample
{
    static void Main()
    {
      Dog dogobj = new Dog();
      dogobj.getAge();
      Console.WriteLine(dogobj.gender);
    }
}
```



This marks the end of our discussion on the four major *access modifiers*. In the next, lesson we will discuss *constructors* of *classes*.