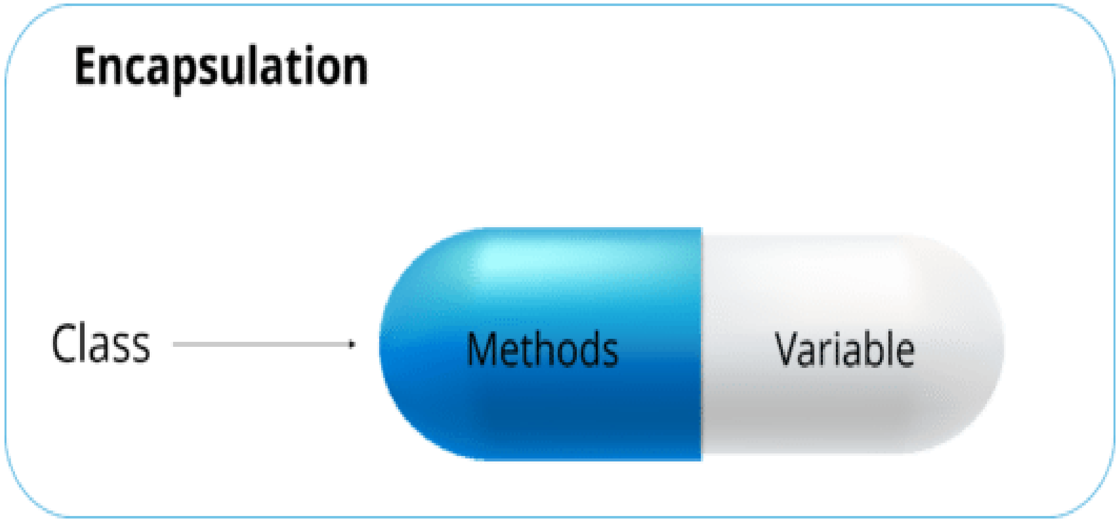
Object Oriented Programming

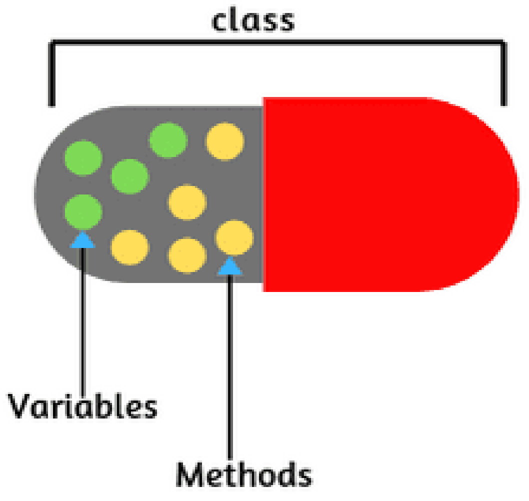
Encapsulation 1



```
class
{

  data members
        +
  methods (behavior)

}
```



- “Mechanism that associates the **code** and the **data** it manipulates into a single unit and keeps them safe from external interference and misuse.”
- The data can only accessible only through the function otherwise it is hidden from user
- This insulation of data from direct access by the program is called data hiding
- Encapsulation = **Data Hiding + Abstraction**

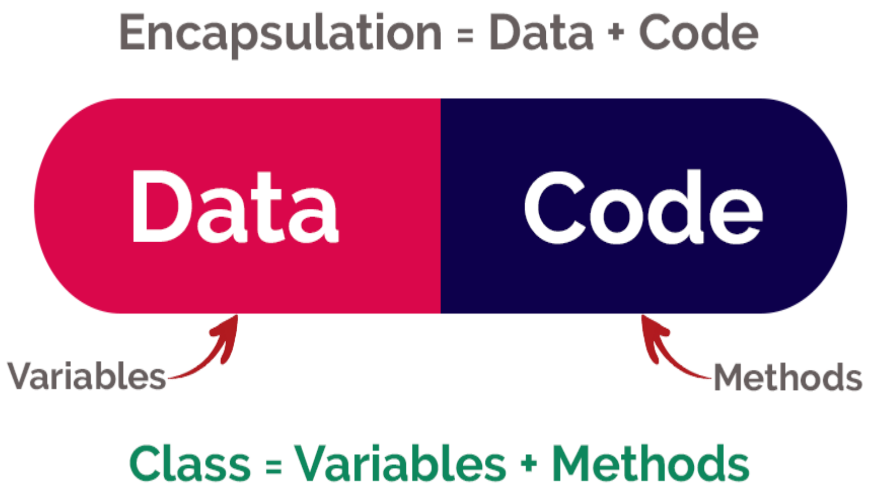
Data Abstraction

“A data abstraction is a **simplified view** of an object that includes only features one is **interested** in while **hides** away the **unnecessary** details.”

“Data abstraction becomes an **abstract data type** (ADT)or a user-defined type.”

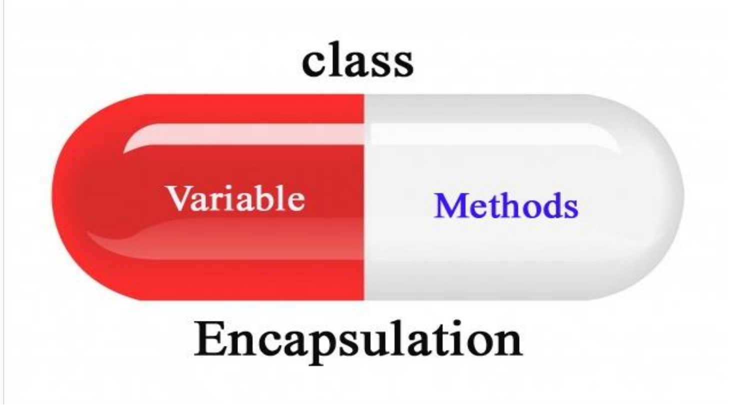
- Encapsulation is the process of hiding irrelevant data from the user.
- Abstraction is mechanism to show only relevant data to the user.

Encapsulation is the process of combining data and code into a single unit (object / class). In OOP, every object is associated with its data and code. In programming, data is defined as variables and code is defined as methods.



Encapsulation

- Is the inclusion of property & method within a class/object in which it needs to function properly.
- Also, enables reusability of an *instant* of an already implemented class within a new class while ***hiding & protecting*** the method and properties from the client classes.



- The wrapping up of data and function into a single unit (called class). Incorporation into a class of data & operations in one package
- Data can only be accessed through that package . The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.
- “Information Hiding”
- These functions provide the interface between the object’s data and the program.
- This insulation of the data from direct access by the program is called *data hiding or information hiding*

Encapsulation – Benefits

- Ensures that structural changes remain local:
  - Changing the class internals does not affect any code outside of the class
  - Changing methods' implementation does not reflect the clients using them
- Encapsulation allows adding some logic when accessing client's data
  - E.g. validation on modifying a property value
- Hiding implementation details reduces complexity → easier maintenance

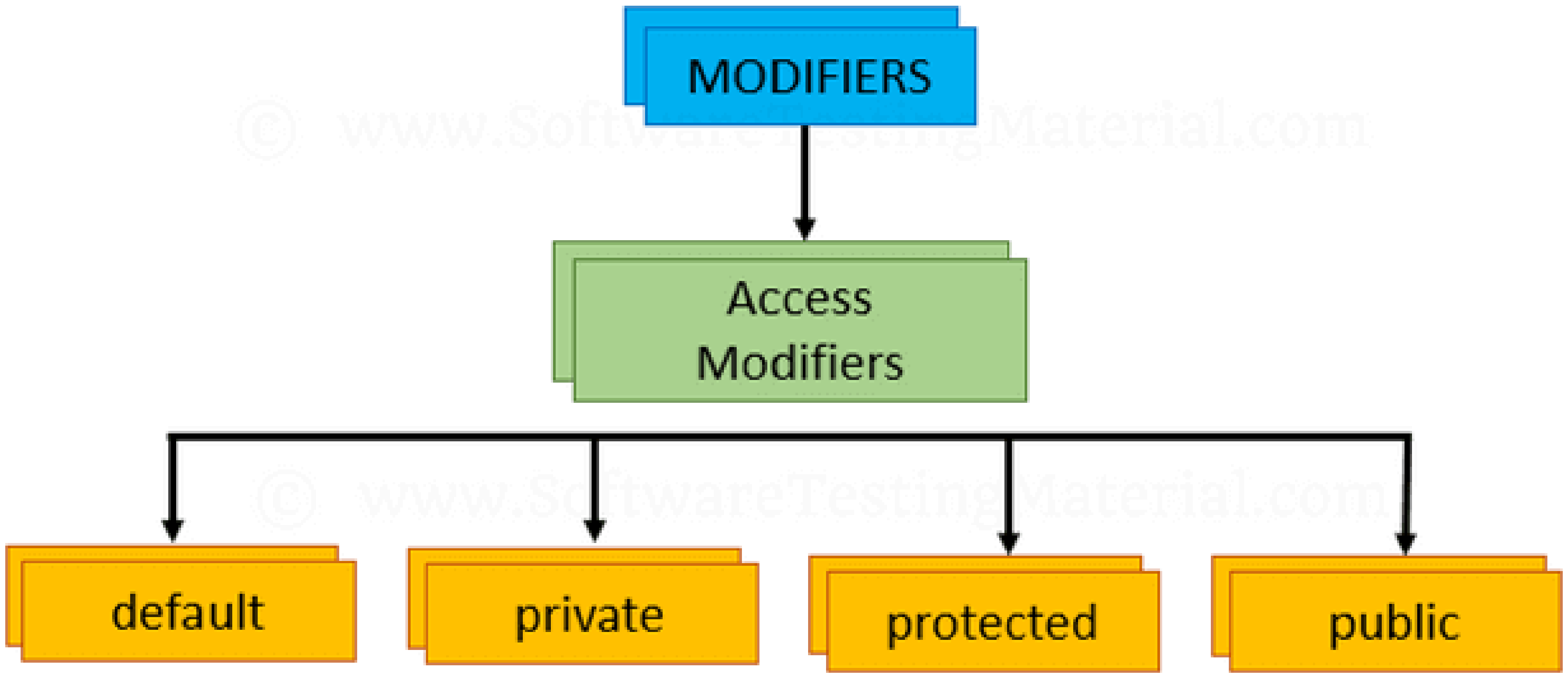
Access Modifiers

keywords that are used to specify accessibility or scope of variables and functions in the C# application.

Access modifiers are an integral part of object-oriented programming. Access modifiers are used to implement encapsulation of OOP. Access modifiers allow you to define who does or who doesn't have access to certain features.

Access modifiers used to specify the scope of accessibility of a member of a class or type of the class itself. For example, a public class is accessible to everyone without any restrictions, while an internal class may be accessible to the assembly only.

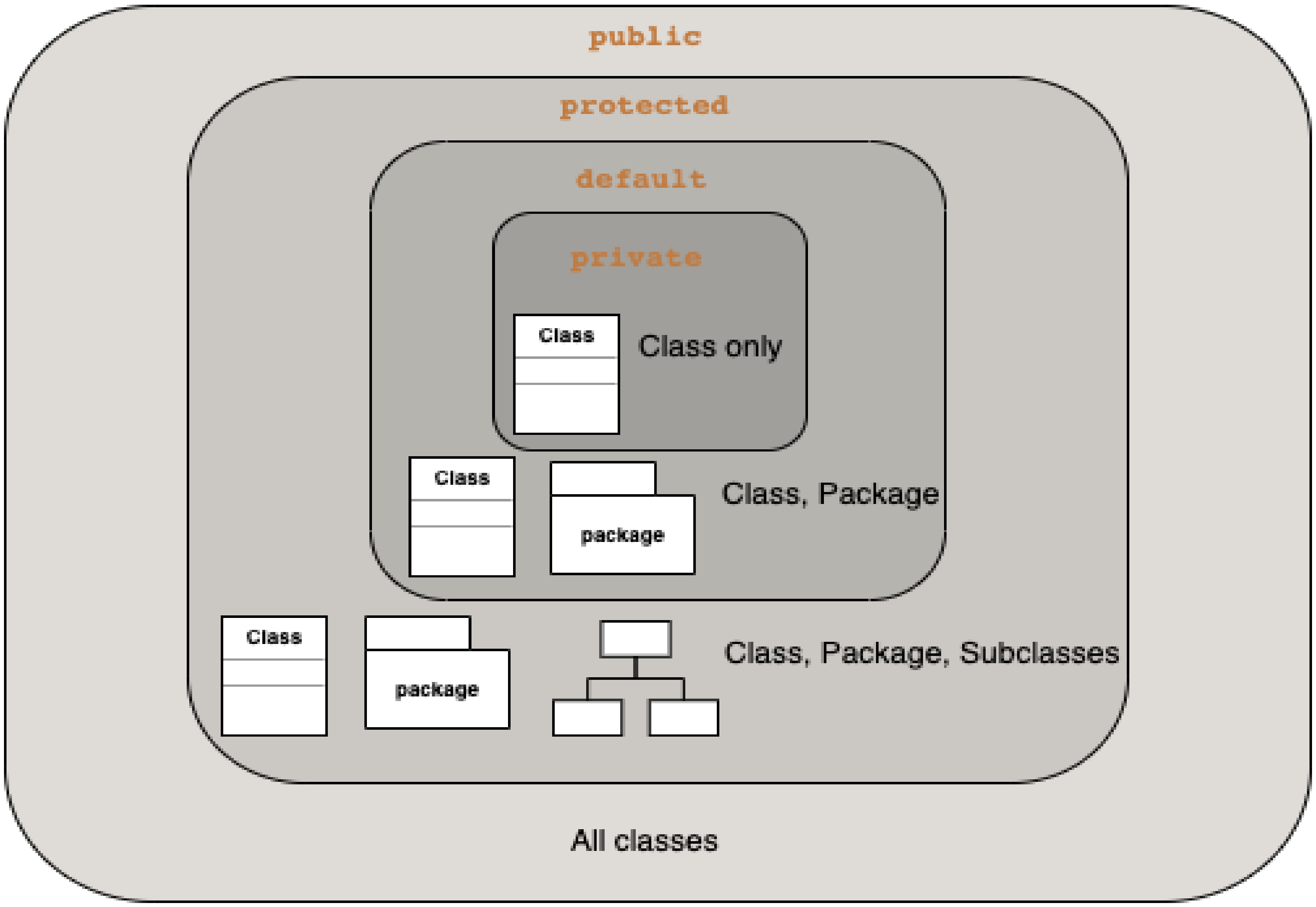
Access Modifiers

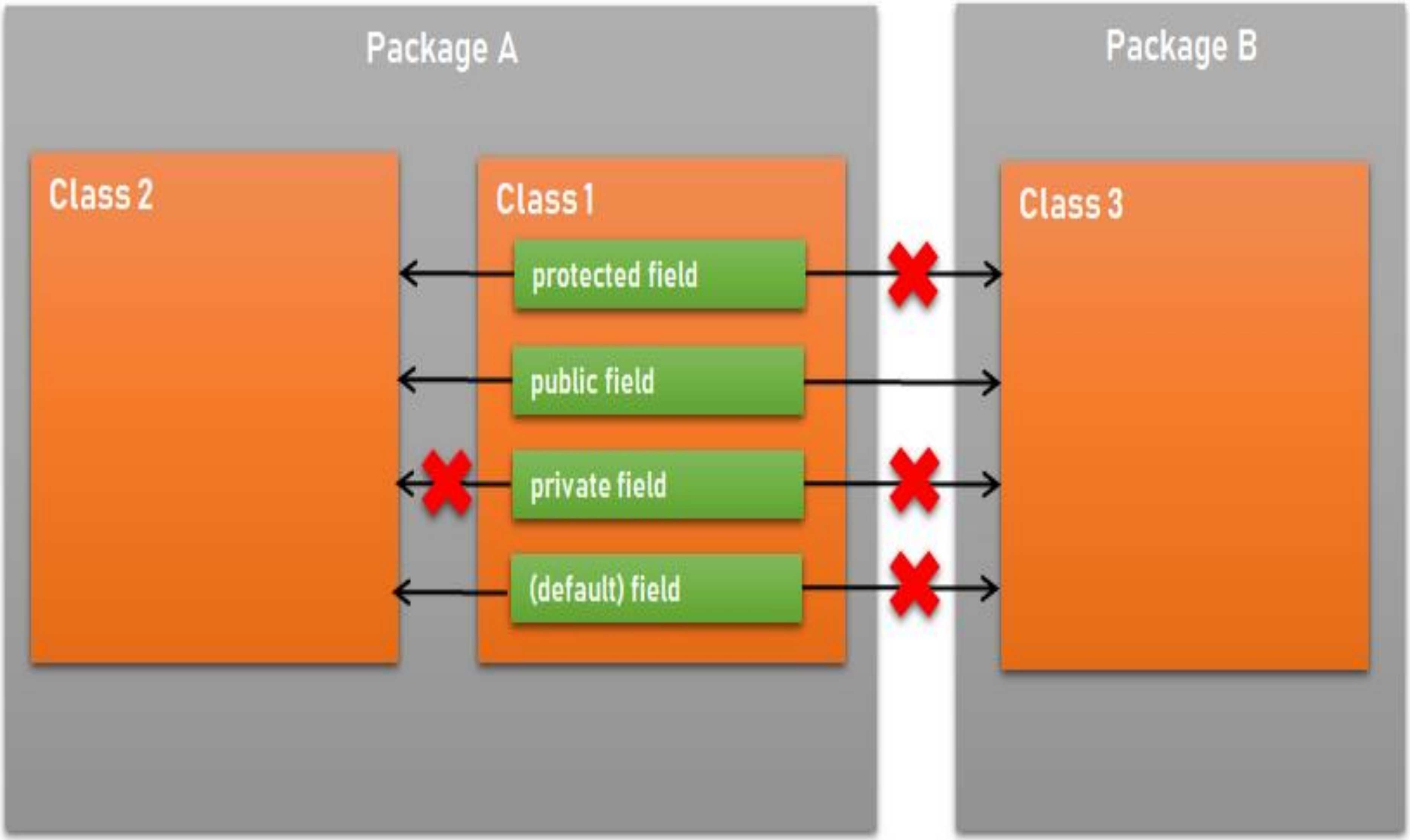


The public keyword is an **access modifier**, which is used to set the access level/visibility for classes, fields, methods and properties.
C# has the following access modifiers:

Modifier	Description
public	The code is accessible for all classes
private	The code is only accessible within the same class
protected	The code is accessible within the same class, or in a class that is inherited from that class. You will learn more about inheritance in a later chapter
internal	The code is only accessible within its own assembly, but not from another assembly. You will learn more about this in a later chapter

There's also two combinations: protected internal and private protected.


public
protected
default
private
Class
Class only
Class
package
Class, Package
Class
package
Class, Package, Subclasses
All classes


Package A
Package B
Class 2
Class 1
Class 3
protected field
public field
private field
(default) field

	Inside same class	Inside default class	Other code	Inside derived class	Other code
+Public	✓	✓	✓	✓	✓
-Private	✓	✗	✗	✗	✗
~Internal	✓	✓	✓	✗	✗
Protected	✓	✗	✗	✓	✗

Default access

A default access level is used if no access modifier is specified in a member declaration. The following list defines the default access modifier for certain C# types:

enum: The default and only access modifier supported is public.
class: The default access for a class is private. It may be explicitly defined using any of the access modifiers.
interface: The default and only access modifier supported is public.

The default access may suffice for a given situation, but you should specify the access modifier you want to use to ensure proper application behavior.
Note: Interface and enumeration members are always public and no access modifiers are allowed.

Private Modifier

If you declare a field with a private access modifier, it can only be accessed within the same class:

```
class Car
{
 private string model;
 static void Main(string[] args)
{

 Car Ford = new Car("Mustang");
 Console.WriteLine(Ford.model);
}
}
```

The output will be: Mustang

If you try to access it outside the class, an error will occur:

```
class Car
{
  private string model = "Mustang";
}
class Program
{
static void Main(string[] args)
{
Car myObj = new Car();
Console.WriteLine(myObj.model);
}
}
```

```
The output will be:
 'Car.model' is inaccessible due to its protection level
The field 'Car.model' is assigned but its value is never used
```

Public Modifier

If you declare a field with a public access modifier, it is accessible for all classes:

```
class Car
{
 public string model = "Mustang";
}
class Program
{
static void Main(string[] args)
{
 Car myObj = new Car();
Console.WriteLine(myObj.model);
}
}
```

```
The output will be:
 Mustang
```

Why Access Modifiers?

To control the visibility of class members (the security level of each individual class and class member).
To achieve "**Encapsulation**" - which is the process of making sure that "sensitive" data is hidden from users. This is done by declaring fields as private.
Note: By default, all members of a class are private if you don't specify an access modifier

To achieve "**Encapsulation**" - which is the process of making sure that "sensitive" data is hidden from users. This is done by declaring fields as private.
To control the visibility of class members (the security level of each individual class and class member).
We are using access modifiers for providing security of our applications.

1. **Public:** As the name says, members can be accessed from any class and any assembly.

Example

```
namespace Sample1
{
  public class Father
  {
      public int Id;
  }

   public class Program
  {
       public static void Main(string[] args)
       {
           Father myfather = new Father();

           myfather.Id = 10;
       }
   }
}
```

Output :

في كل مكان من الكلاس والبرنامج

2. Private: Members can be accessed within the class only.

Example

```
namespace Sample2
{
  public class Father
  {
      private int Id;
  }

   public class Program
  {
       public static void Main(string[] args)
       {
           Father myfather = new Father();

           myfather.Id = 10;
       }
   }
}
```

Output*Error 1 "Sample.ABC.Id" is inaccessible due to its protection level*

في كل مكان من الكلاس فقط

3. Protected: Members can be accessed within its class and derived class of the same assembly.
Protected members are also accessible outside the assembly provided it should be derived.

Example

```
namespace Sample3
{
  public class Father
  {
      protected int Id;
  }

   public class Program
  {
       public static void Main(string[] args)
       {
           Father myfather = new Father();

           myfather.Id = 10;
       }
   }
}
```

في كل مكان من الكلاس و وريثه

Output*Error 1 "Sample.ABC.Id" is inaccessible due to its protection level*

Example 2

```
namespace Sample3
{
    public class Father
    {
        protected int Id;
    }
    public class Son : Father
    {
        public void Print()
        {
            Id = 5;
            Console.WriteLine(Id);
        }
    }
    public class Program
    {
        public static void Main(string[] args)
        {
            Son s = new Son();
            s.Print();
            Console.ReadLine();
        }
    }
```

Output5

4. Internal: Members can be accessed from anywhere within the same assembly.

Create a Class library

```
namespace Sample4
{
    public class Class1
    {
        internal int a;
    }
}
```

Now create a console application, take a reference of the preceding library.

```
namespace Sample44
{
    public class Program
   {
        public static void Main(string[] args)
        {
            Sample4.Class1 cls = new Class1();
            cls.a = 6; // error
        }
    }
}
```

Output*Error 1 'ClassLibrary1.Class1' does not contain a definition for 'a' and no extension method 'a' accepting a first argument of type 'ClassLibrary1.Class1' could be found (are you missing a using directive or an assembly reference?*

في كل مكان من البرنامج

5. Protected Internal: Members can be accessed anywhere in the same assembly and also accessible by inheriting that class.
It can be accessible outside the assembly in the derived class only.
Protected Internal member works as Internal within the same assembly and works as Protected outside the assembly.

```
namespace Sample5
{
    public class Class1
    {
        protected internal int a;
    }
}
```

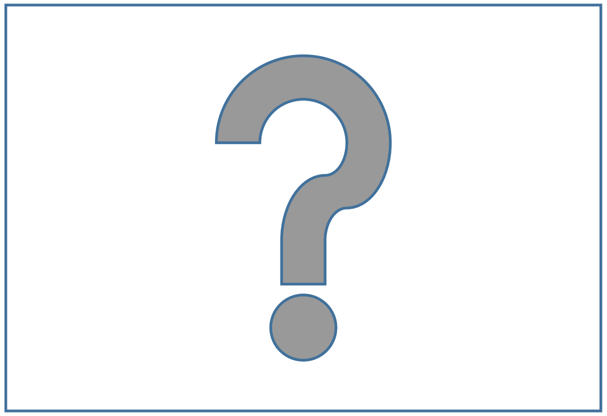
Now create a console application and make a reference of the preceding library.

في كل مكان من البرنامج و خارجه

```
namespace Sample55
{
  public class ABC:Class1
  {
      private int Id;
      public void XYZ()
      {
          a = 5;
          Console.WriteLine(a);
        }
  }

   public class Program
  {
       public static void Main(string[] args)
       {
           ABC abc = new ABC();
           abc.XYZ();
           Console.ReadKey();
       }
  }
}
```

QUESTION



Google Classroom :