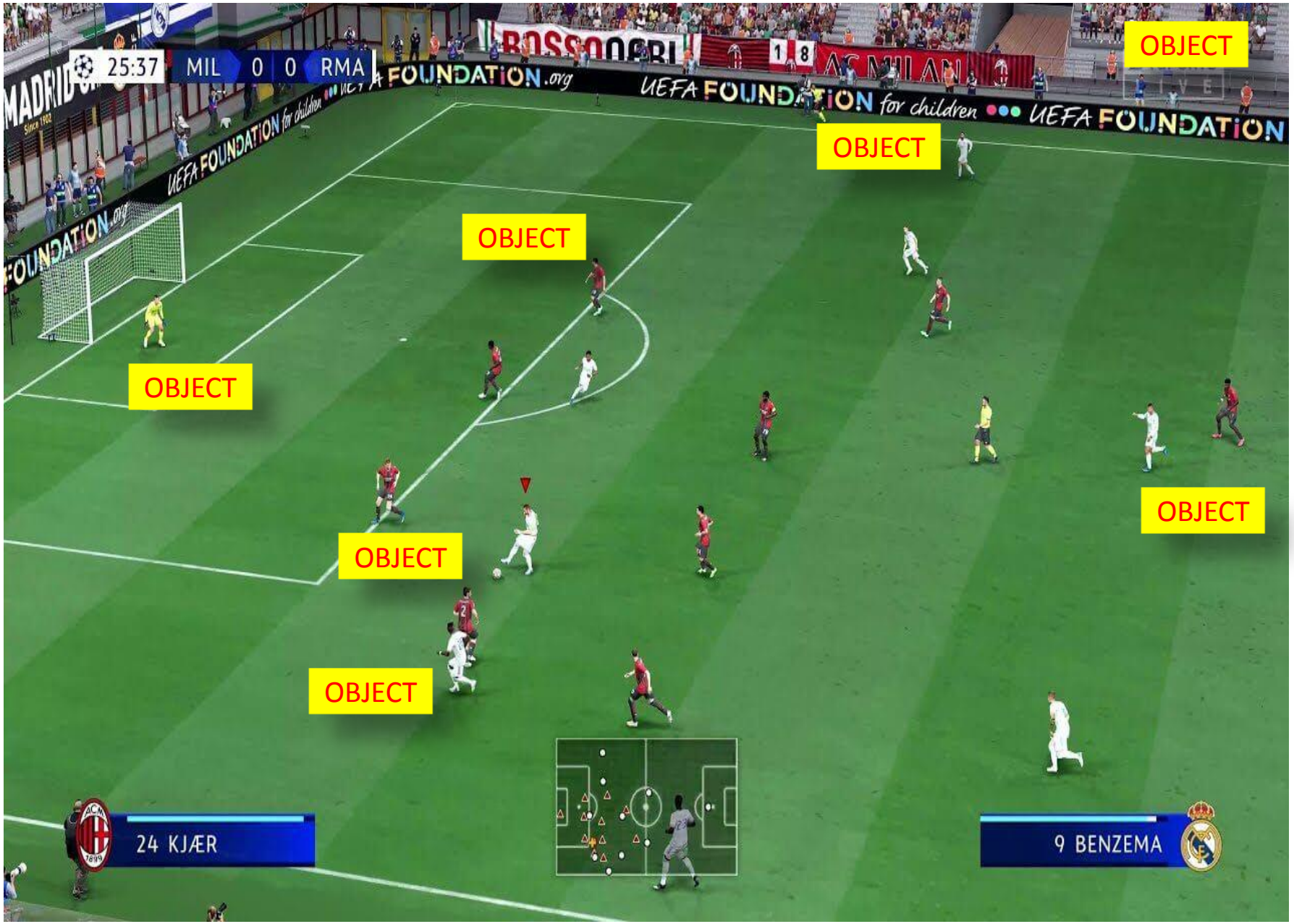
OOP

OBJECT-ORIENTED PROGRAMMING

5

ENCAPSULATION
ACCESS MODIFIERS



PLAYER
Name Team Position Age Goals salary
Attack Kick Pass Jump Out Offside

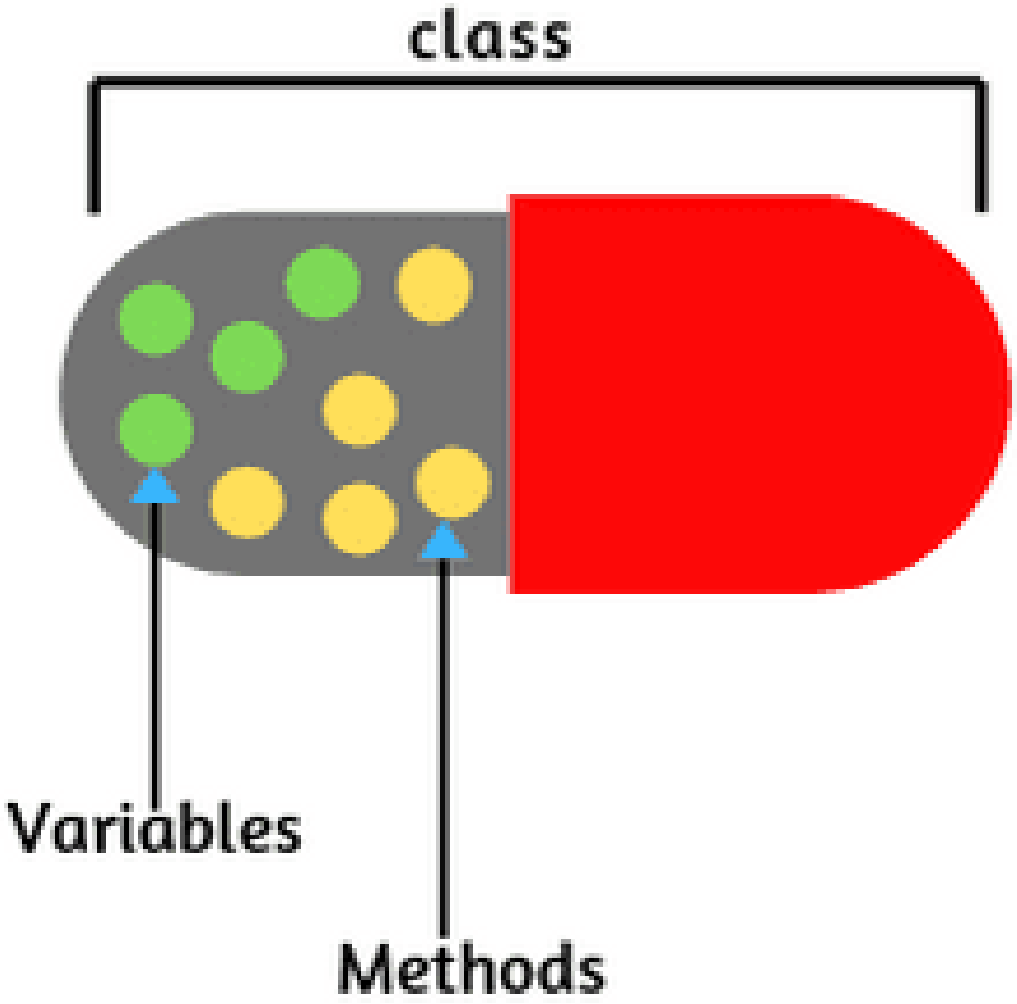
Attributes

Methods

```
class
{
   attributes
   methods
}
```

data members

(behavior) function



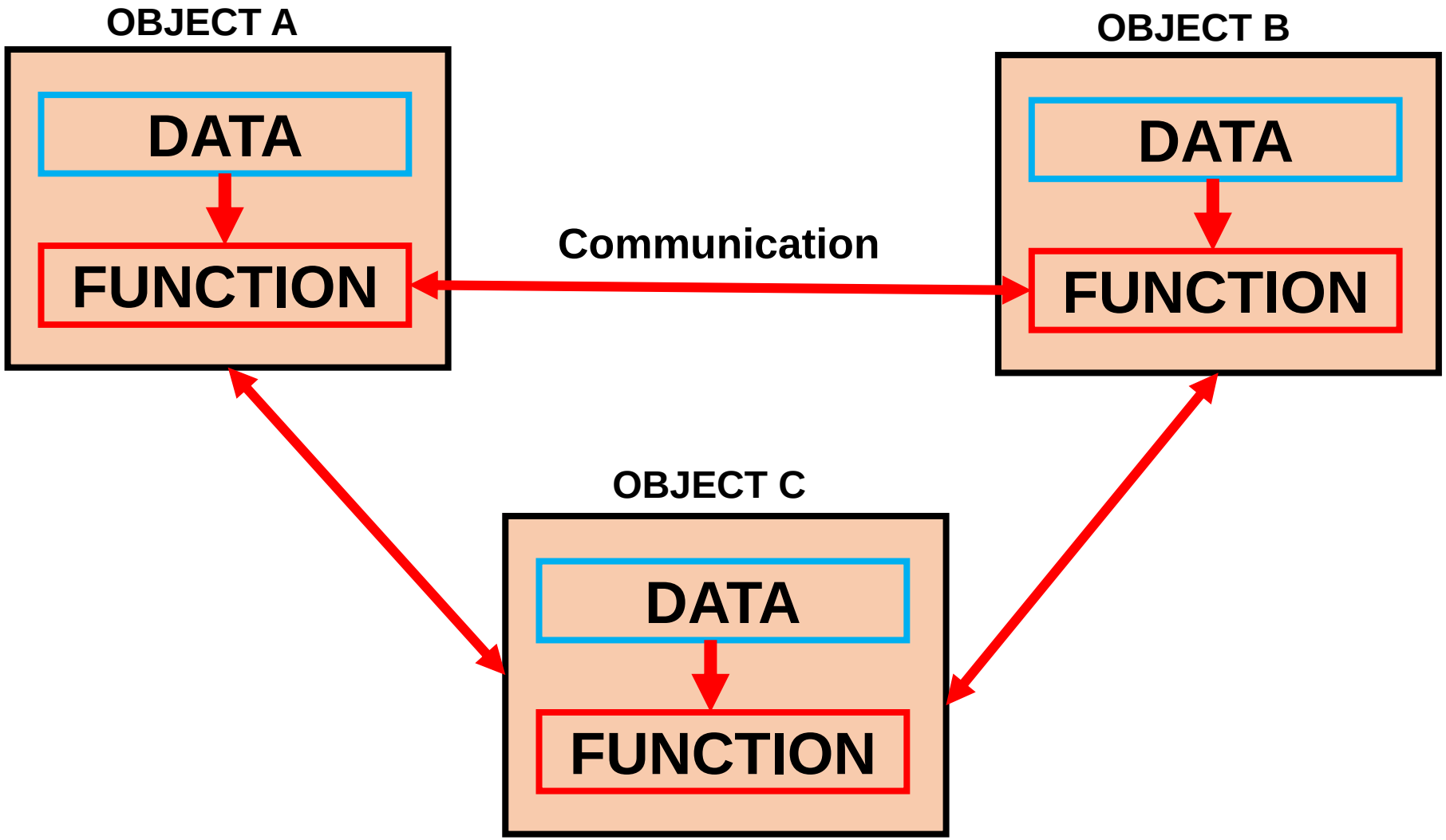
Encapsulation is the process of combining data and code into a single unit (object / class). And keep them safe from external interference and misuse

In OOP, every object is associated with its data and code.

The data can be accessible only through the function otherwise it is hidden from user

This isolation of data from direct access by the program is called data hiding

Encapsulation = Data hiding + Abstraction


OBJECT A
DATA
FUNCTION
OBJECT B
DATA
FUNCTION
Communication
OBJECT C
DATA
FUNCTION

Encapsulation - Benefits

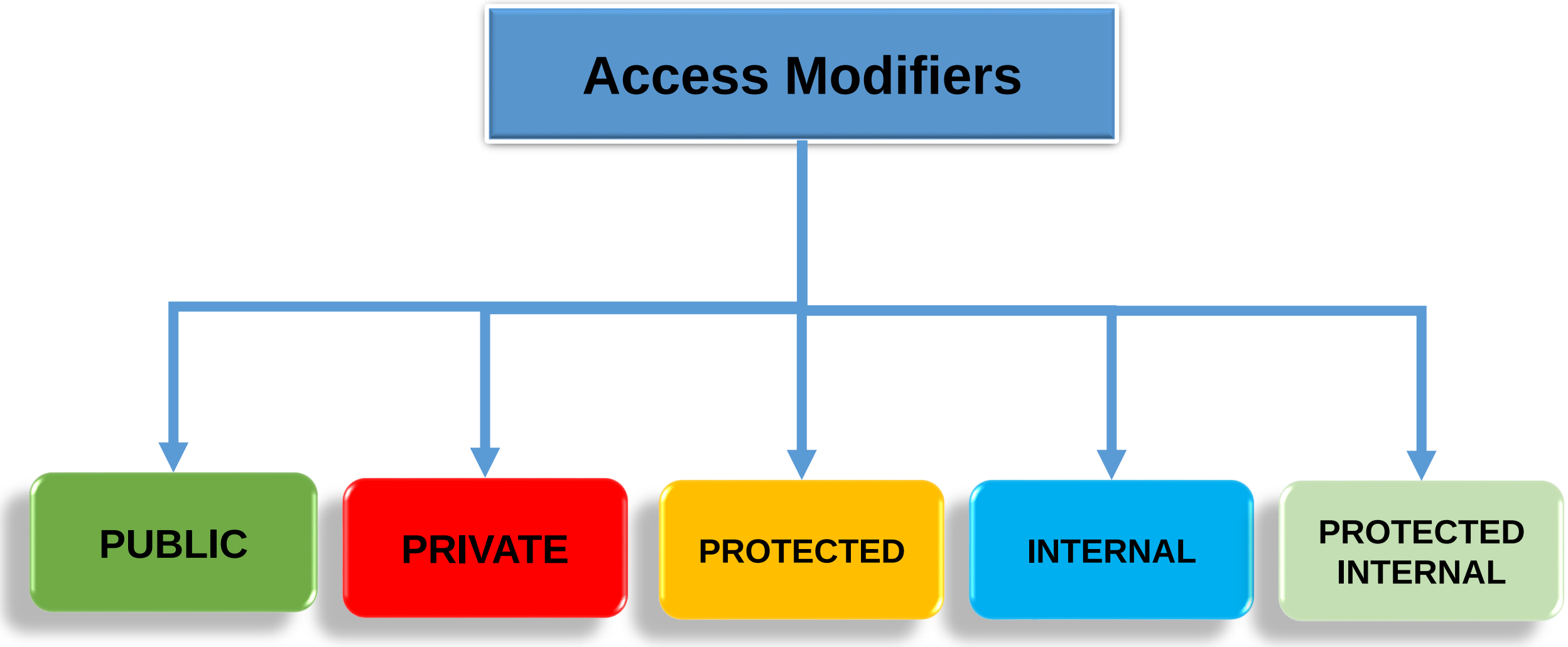
- Encapsulation protects an object from unwanted access by clients.
- Encapsulation allows access to a level without revealing the complex details below that level.
- It reduces human errors.
- Simplifies the maintenance of the application
- Makes the application easier to understand.

Access Modifiers

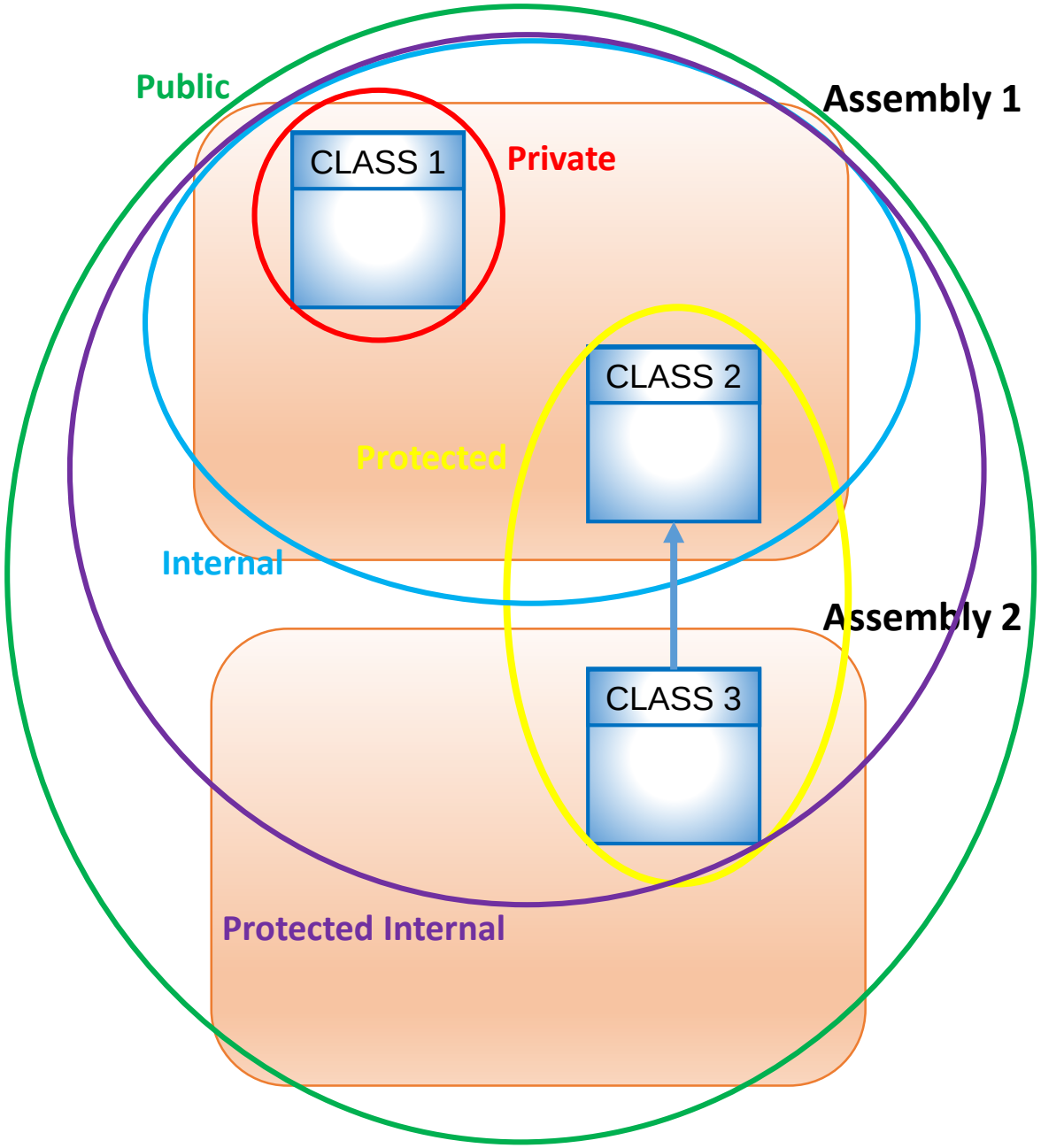
specify accessibility or scope of variables and functions in the C# application.

Access modifiers :

- an integral part of object-oriented programming.
- used to implement encapsulation of OOP. (process of making sure that "sensitive" data is hidden from users).
- allow define who does or who doesn't have access to certain features.
- control the visibility of class members (the security level of each individual class and class member).


Access Modifiers
PUBLIC
PRIVATE
PROTECTED
INTERNAL
PROTECTED INTERNAL

Modifier	Description
+public	All objects can access the data.
-private	The code is only accessible within the same class Access is limited to members of the same class.
~protected	The code is accessible within the same class, or in a class that is inherited from that class. Access is limited to members of the same class or descendants.
#internal	The code is only accessible within the current assembly but not from another assembly. Access is limited to the current assembly. (C# only)
Protected Internal	Access is limited to the current assembly or types derived from the containing class. (C# only)


Public
Assembly 1
CLASS 1
Private
CLASS 2
Protected
Internal
Assembly 2
CLASS 3
Protected Internal

Public it is accessible for all classes

```
class Car
{
  public string model;
  public void info()
  {
     model = "Corolla";
    Console.WriteLine("THE MODEL : " + model);
  }
}
 static void Main(string[] args)
{
  Car Toyota = new Car();
  Toyota.info();

  Toyota.model="Camry";
  Console.WriteLine(Toyota.model);
```

The output will be:

Private it can only be accessed within the same class:

```
class Car
{
  private string model;
  public void info()
  {
     model = "Corolla";
    Console.WriteLine("THE MODEL : " + model);
  }
}
 static void Main(string[] args)
{
  Car Toyota = new Car();
  Toyota.info();

  //Toyota.model="Camry";
  //Console.WriteLine(Toyota.model);
```

✓

X

```
class AccessMod
{
    public int num1;
    int num2;
}
static void Main(string[] args)
{
    AccessMod ob1 = new AccessMod();

    //Direct access to public members
    ob1.num1 = 100;
    ob1.
    C                                    ne("Number one value in
    C                                    e();
}
```

Equals
GetHashCode
GetType
num1
ToString

Protected

is accessible from within the class in which it is declared, and from within any class derived from the class that declared this member.(inheritance)

```
class Car
{
  protected string model;
  protected string color;
  public void info()
  {
    model = "Corolla";
    Console.WriteLine("THE MODEL : " + model);
  }
}
class Vehicle : Car
{
  private int     year;
  private double price;
  public void printinfo()
  {
   Console.WriteLine("THE COLOR : " + color );
   Console.WriteLine("THE YEAR : " + year );
  }
}
```

وراثة

```
static void Main(string[] args)
{
  Car Toyota = new Car();
  Vehicle opel = new Vehicle();

  Toyota.info();

  //Toyota.model="Camry";
  //Console.WriteLine(Toyota.model);

  opel .model="Astra";
  Console.WriteLine(opel.model);
}
```

Internal (== public)

Members can be accessed from anywhere within the same assembly. accessible only within files in the same assembly (.dll).

```
namespace Sample1
{
    internal class Class1
   {
       internal int age=20;
   }
}
```

✓

```
Using sample1;
namespace Sample2
{
   class Class2
  {
      public static void Main(string[] args)
      {
         Class1 myclass = new Class1();
         Console.WriteLine(myclass.age);
      }
  }
}
```

X

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.

```
namespace Sample1
{
     class Class1
   {
       protected internal int age=20;
   }
}
```

```
Using sample1;
namespace Sample2
{
    class Class2  : Class1
  {
       public static void Main(string[] args)
       {
          Class2 myclass = new Class2();
          Console.WriteLine(myclass.age);
       }
  }
}
```

Access Modifiers

	Containing Classes	Derived Classes	Containing Assembly	Anywhere outside the containing assembly
Public	Yes	Yes	Yes	Yes
Private	Yes	No	No	No
Protected	Yes	Yes	No	No
Internal	Yes	No	Yes	No
Protected Internal	Yes	Yes	Yes	No

```
class Shape
{
  private int height;
  private int base;

  public void Readinfo()
  {
    height = 10;
    base   = 12;
  }
  public void Printinfo()
  {
    Console.WriteLine("THE HEIGHT = " + height);
    Console.WriteLine("THE BASE   = " + base);
  }
  public double FindArea()
  {
    double a;
    a = 0.5 * height * base ;
    return a;
  }
}
```

Attributes === privet

Methods === public

Default Access Modifier

A default access level is used if no access modifier is specified in a member declaration. The default access for a class is private. Method is public

```
class Shape
{
  private int height;
  private int base;

  public void Readinfo()
  {
    height = 10;
    base   = 12;
  }
  public void Printinfo()
  {
    Console.WriteLine("THE HEIGHT = " + height);
    Console.WriteLine("THE BASE   = " + base);
  }
  public double FindArea()
  {
    double a;
    a = 0.5 * height * base ;
    return a;
  }
}
```

```
class Shape
{
    int height;
    int base;

    void Readinfo()
  {
    height = 10;
    base   = 12;
  }
    void Printinfo()
  {
    Console.WriteLine("THE HEIGHT = " + height);
    Console.WriteLine("THE BASE   = " + base);
  }
    double FindArea()
  {
    double a;
    a = 0.5 * height * base ;
    return a;
  }
}
```

```
using System;
namespace Lecture5
{
 class patient
 {
   private string  name;
   private int age;
   private string diag;
   private string med;
   private double fees;

   public void Readinfo()
   {
     name = Console.ReadLine();
     age  = int.Parse(Console.ReadLine());
     diag = Console.ReadLine();
     med  = Console.ReadLine();
     fees = double.Parse(Console.ReadLine());
   }
   public void Printinfo()
   {
     Console.WriteLine("NAME      = " + name);
     Console.WriteLine("AGE       = " + age);
     Console.WriteLine("DIAGNOSIS = " + diag);
     Console.WriteLine("MEDICINE  = " + med);
     Console.WriteLine("FEES      = " + fees);
   }
 }
```

```
static void Main(string[] args)
{
  patient p1 = new patient();
  patient p2 = new patient();

  p1.name = "Ahmad";
  p2.name = "Ali";

  Console.WriteLine("NAME1=" + p1.name);
  Console.WriteLine("NAME2=" + p2.name);

  Console.WriteLine("NAME=" + p1.name + p2.name);


  Console.ReadKey();
}

}
}
```

```
using System;
namespace Lecture5
{
 class patient
 {
   private string  name;
   private int age;
   private string diag;
   private string med;
   private double fees;

   public void Readinfo()
   {
     name = Console.ReadLine();
     age  = int.Parse(Console.ReadLine());
     diag = Console.ReadLine();
     med  = Console.ReadLine();
     fees = double.Parse(Console.ReadLine());
   }
   public void Printinfo()
   {
     Console.WriteLine("NAME      = " + name);
     Console.WriteLine("AGE       = " + age);
     Console.WriteLine("DIAGNOSIS = " + diag);
     Console.WriteLine("MEDICINE  = " + med);
     Console.WriteLine("FEES      = " + fees);
   }
 }
```

```
static void Main(string[] args)
{
  patient p1 = new patient();
  patient p2 = new patient();

  p1.Readinfo();
  p2.Readinfo();

  p1.Printinfo();
  p2.Printdinfo();

  Console.ReadKey();
}
}
```

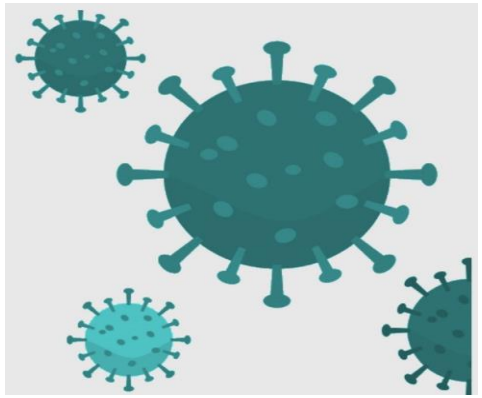
```
using System;
namespace Lecture5
{
 class patient
 {
   private string  name;
   private int age;
   private string diag;
   private string med;
   private double fees;

   public void Readinfo()
   {
     name = Console.ReadLine();
     age  = int.Parse(Console.ReadLine());
     diag = Console.ReadLine();
     med  = Console.ReadLine();
     fees = double.Parse(Console.ReadLine());
   }
   public string rname()
   {
     return name;
   }

 }
```

```
static void Main(string[] args)
{
  patient p1 = new patient();
  patient p2 = new patient();

  // p1.name = “Ahmad”;
  // p2.name = “Ali”;

  // Console.WriteLine(“NAME1=” + p1.name);
  // Console.WriteLine(“NAME2=” + p2.name);

  Console.WriteLine(“NAME=” + p1.rname + p2.rname);


  Console.ReadKey();
}

}
}
```

```
using System;
namespace Lecture3
{
  class Shape
  {
    private int height;
    private int width;

    public void Readinfo()
    {
      Console.WriteLine(“INPUT HEIGHT ");
      height = int.Parse(Console.ReadLine());
      Console.WriteLine(“INPUT WIDTH ");
      width  = int.Parse(Console.ReadLine());
    }
    public void Printinfo()
    {
      Console.WriteLine("THE HEIGHT = " + height);
      Console.WriteLine("THE BASE   = " + width);
    }
    public double FindArea()
    {
      double ar;
      ar = height * width ;
      return ar;
    }
  }
```

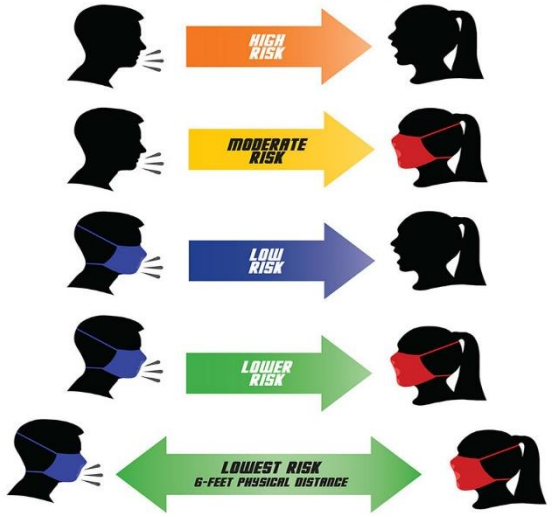
```
  static void Main(string[] args)
  {
    Shape tringle = .....

    ..................

    ..................

    Console.WriteLine(“AREA=” + ..................);

    Shape square = new Shape();

    ..................

    ..................

    Console.WriteLine(“AREA=” + ..................);

    Console.ReadKey();
  }

  }
```

H.W.



Name1
Name2
Distance

Mask1
Mask2
Distance



Attributes

Methods