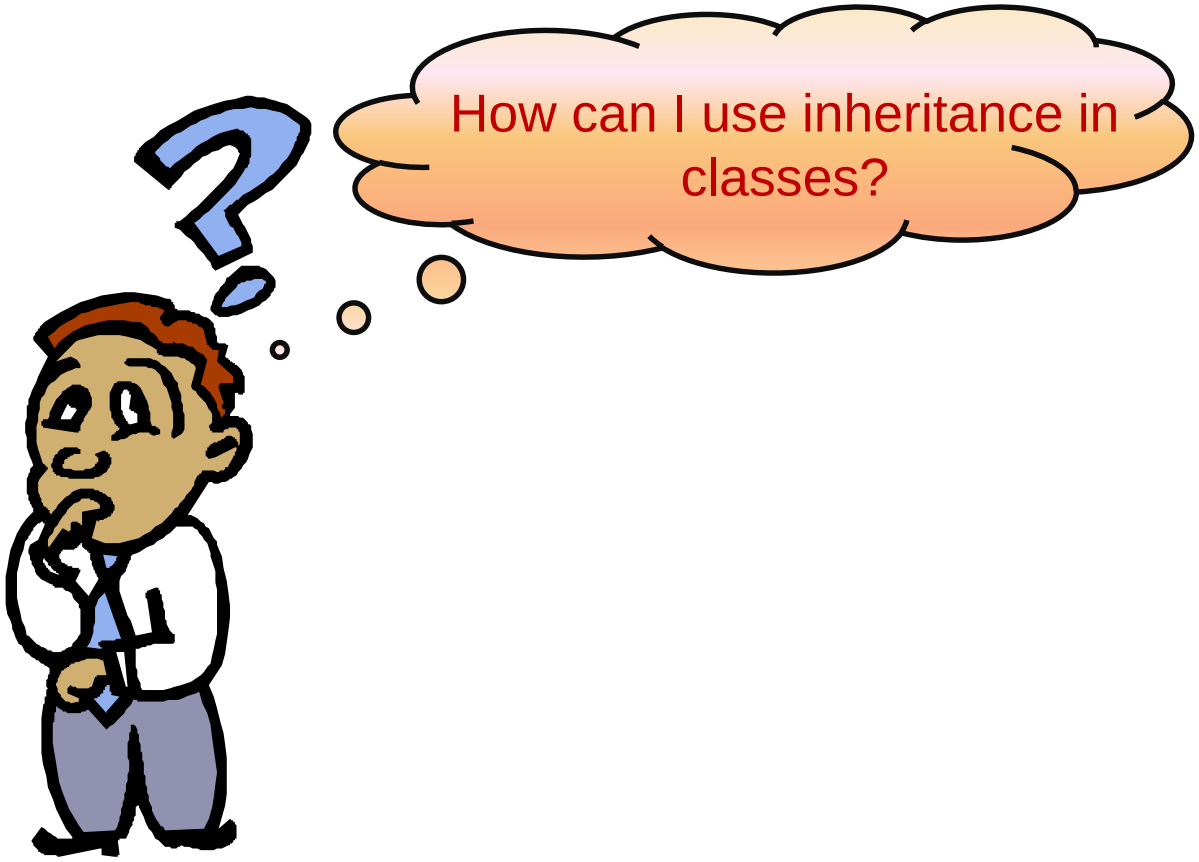
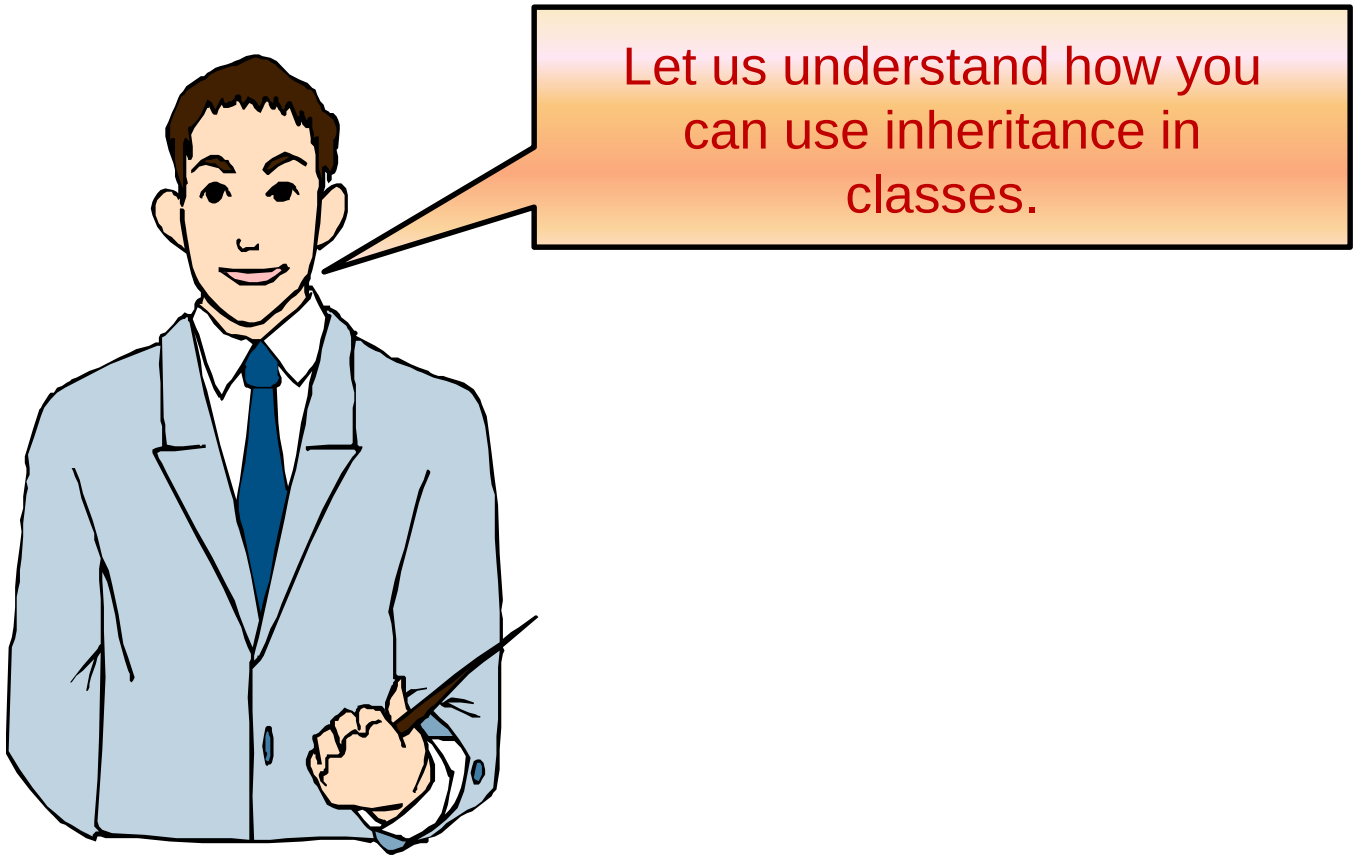
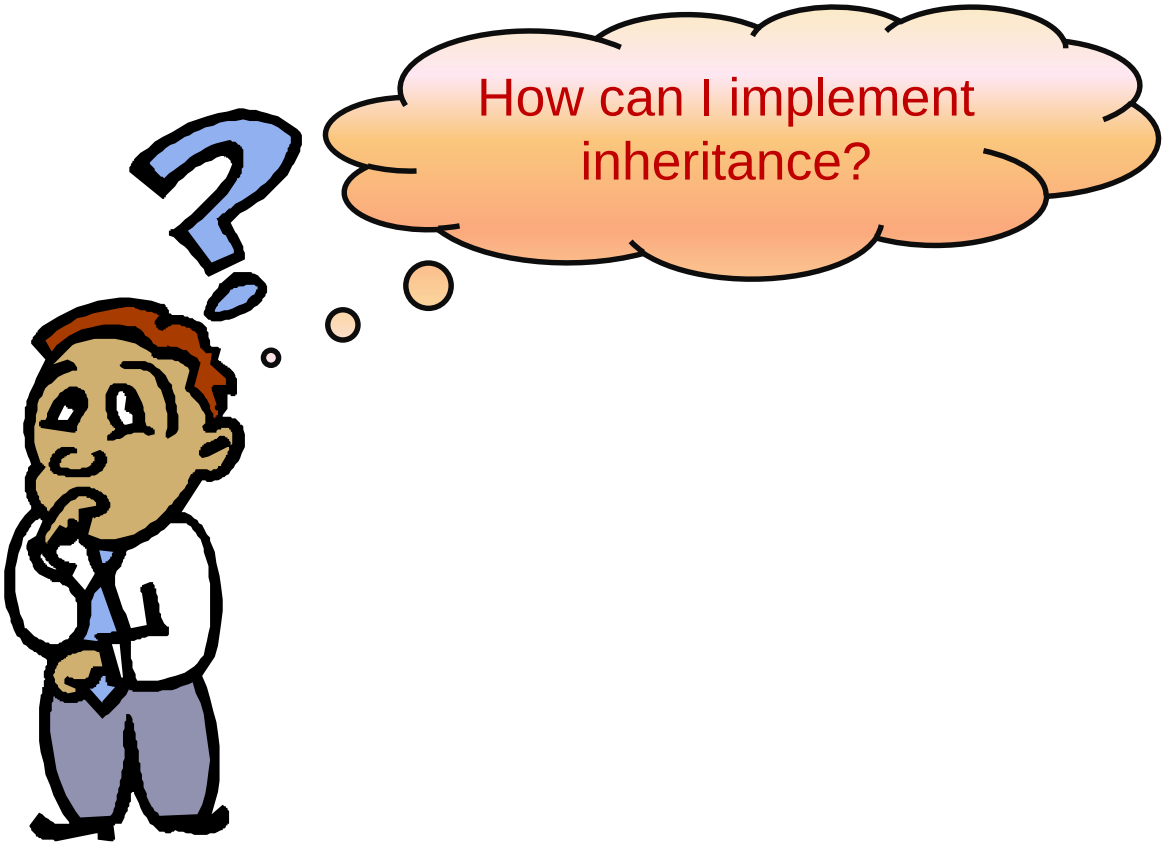
AlmaMate

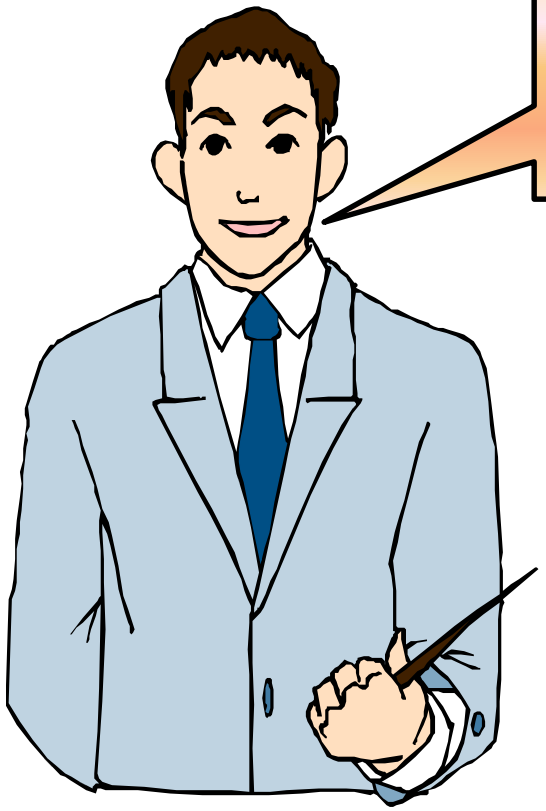
- In this session, you will learn to:
  - Use classes and inheritance
  - Implement encapsulation by using access specifiers


How can I use inheritance in classes?


Let us understand how you can use inheritance in classes.

- In C#, the objects of a derived class get the copy of the data members and member functions of the base class by using inheritance.
- A class that inherits or derives attributes from another class is called the derived class.
- The class from which attributes are derived is known as base class.
- In OOP, the base class is actually a superclass and the derived class is a subclass.


How can I implement inheritance?

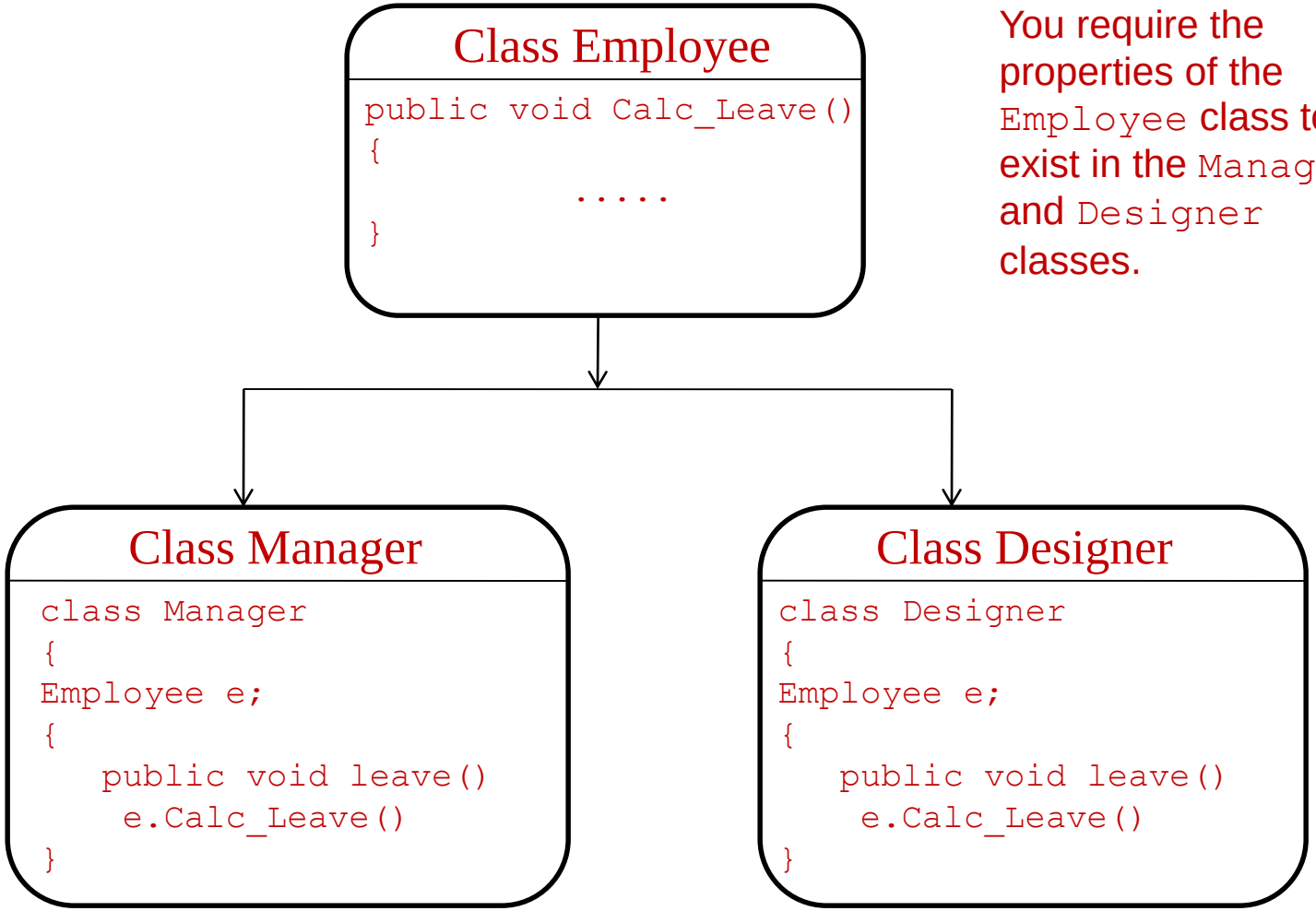


Let us understand how you can implement inheritance.

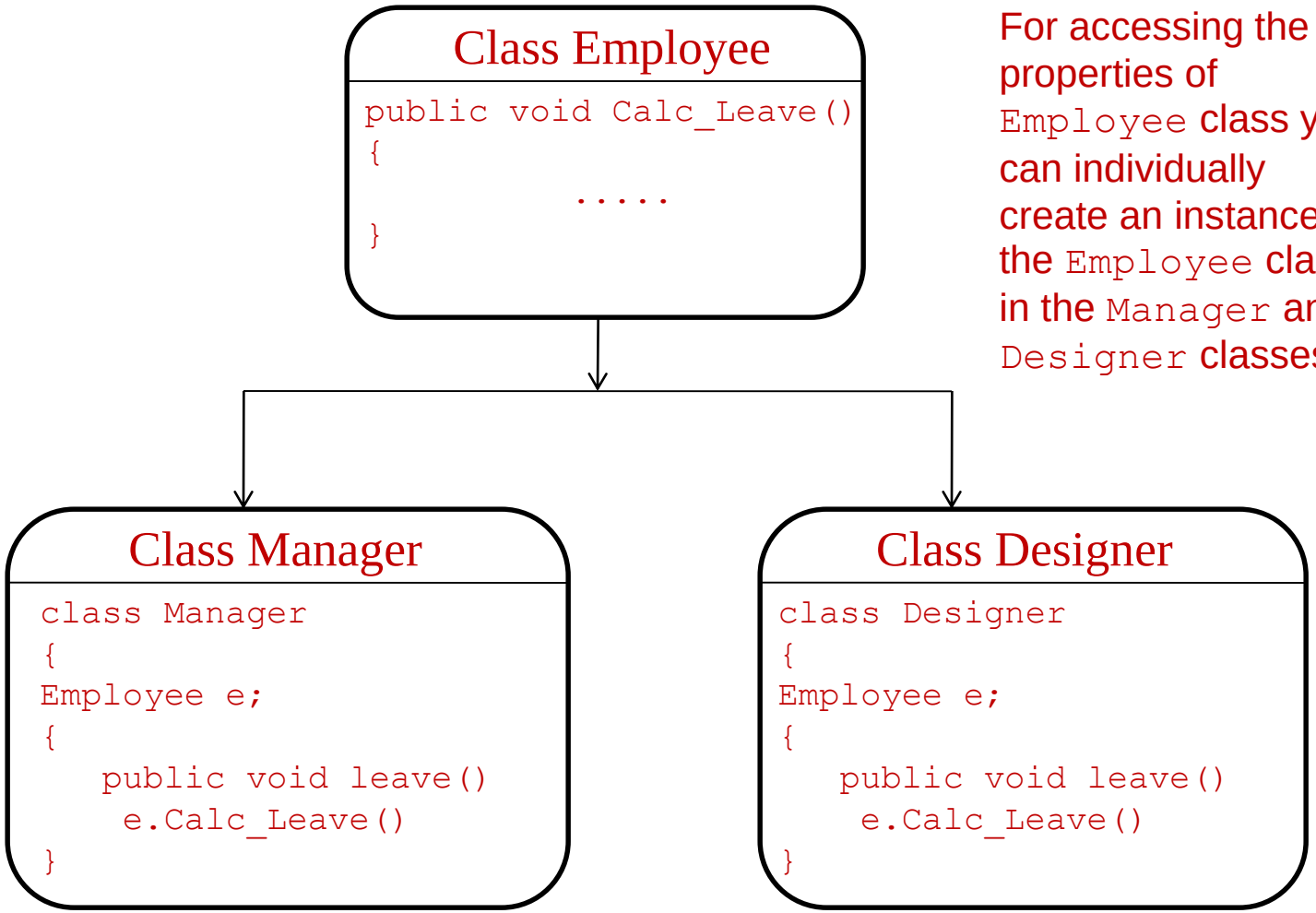
- Each instance of the derived class includes its own attributes and all the attributes of the base class.
- Any change made to the base class automatically changes the behavior of its subclasses.
- The following syntax is used in C# for creating derived classes:

```
<access-specifier> class <base_class>
   {
       ...
   }
       class <derived_class> : <base_class>
   {
       ...
   }
```

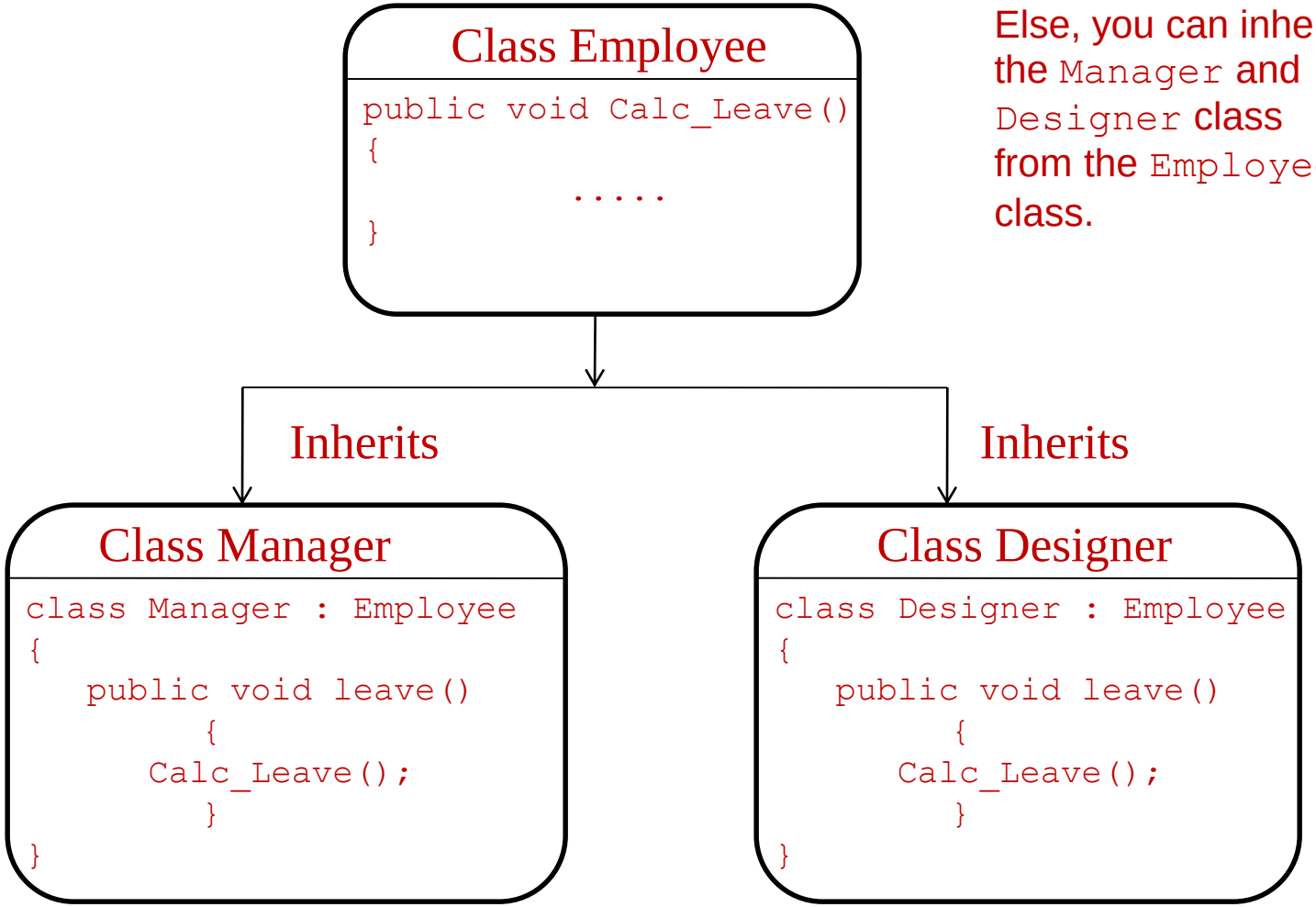
- To determine inheritance hierarchies, you must check the kind of relationship between the derived classes and the base class.



You require the properties of the `Employee` class to exist in the `Manager` and `Designer` classes.



For accessing the properties of `Employee` class you can individually create an instance of the `Employee` class in the `Manager` and `Designer` classes.



Else, you can inherit the `Manager` and `Designer` class from the `Employee` class.

Class Employee

```
public void Calc_Leave()
{
        .....
}
```

Inherits

Inherits

As both the Manager and Designer are employees, therefore, it will be more appropriate to use inheritance.

AlmaMate

- Constructors are called in the order of base-to-derived.
- The following code shows the use of constructors with inherited classes:

```
using System;
class Base
{
     public Base()
     {
            Console.WriteLine("Constructor of Base");
     }
     ~Base()
     {
            Console.WriteLine("Destructor of Base");
     }
}
```

```
class Derived : Base
{
     public Derived()
     {
      Console.WriteLine("Constructor of Derived");
     }
     ~Derived()
     {
      Console.WriteLine("Destructor of Derived");
      }
}
```

```
class BaseDerived
{
  static int Main(string[] args)
  {
  Derived obj = new Derived();
  return 0;
  }
}
```

- The base class can be initialized by calling the constructor of the base class when creating an instance of the derived class.
- The `base` keyword is used to access methods and properties of the base class from a method of the derived class.

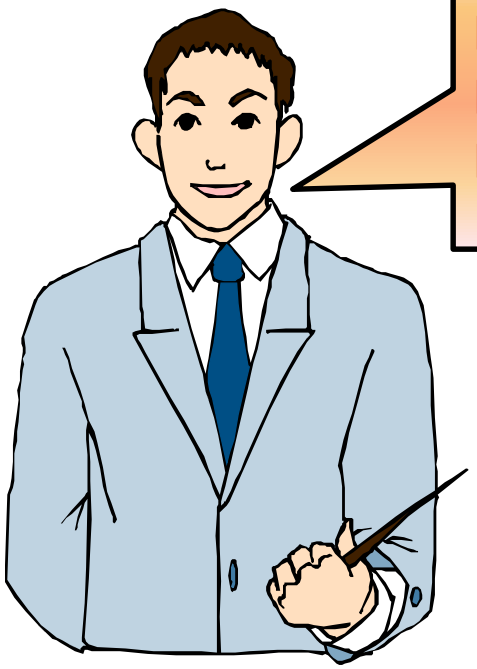
AlmaMate

- The following code shows the use of the `base` keyword:

```
using System;
class Point
{
        private int x, y;
        public Point(int a , int b )
        {
                x = a;
                y = b;
        }
        void setx(int a)
        {
                x = a;
        }
```

```
    void sety(int b)
    {
         y = b;
    }
    int getx()
    {
         return x;
    }
    int gety()
    {
         return y;
    }
    ~Point()
    {
          x = y = 0;
    }
}
```

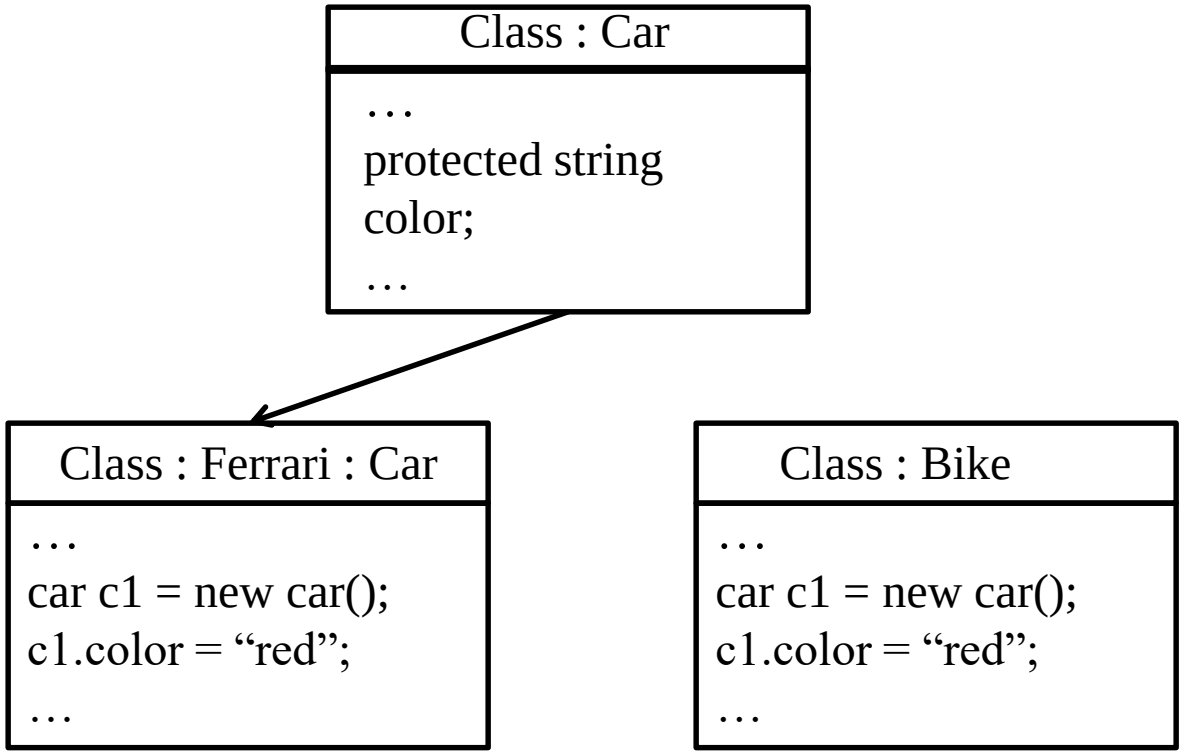
```
class Line : Point
 {
 // calling the base class constructor
 public Line(int x1 ,int y1 ,int x2 ,int y2):base(10,20)
 {
 }
 pubic static void main()
 {
    Line l1= new Line(); // calling the derived
    class constructor
 }
 }
```



Let us now understand the following access specifiers:

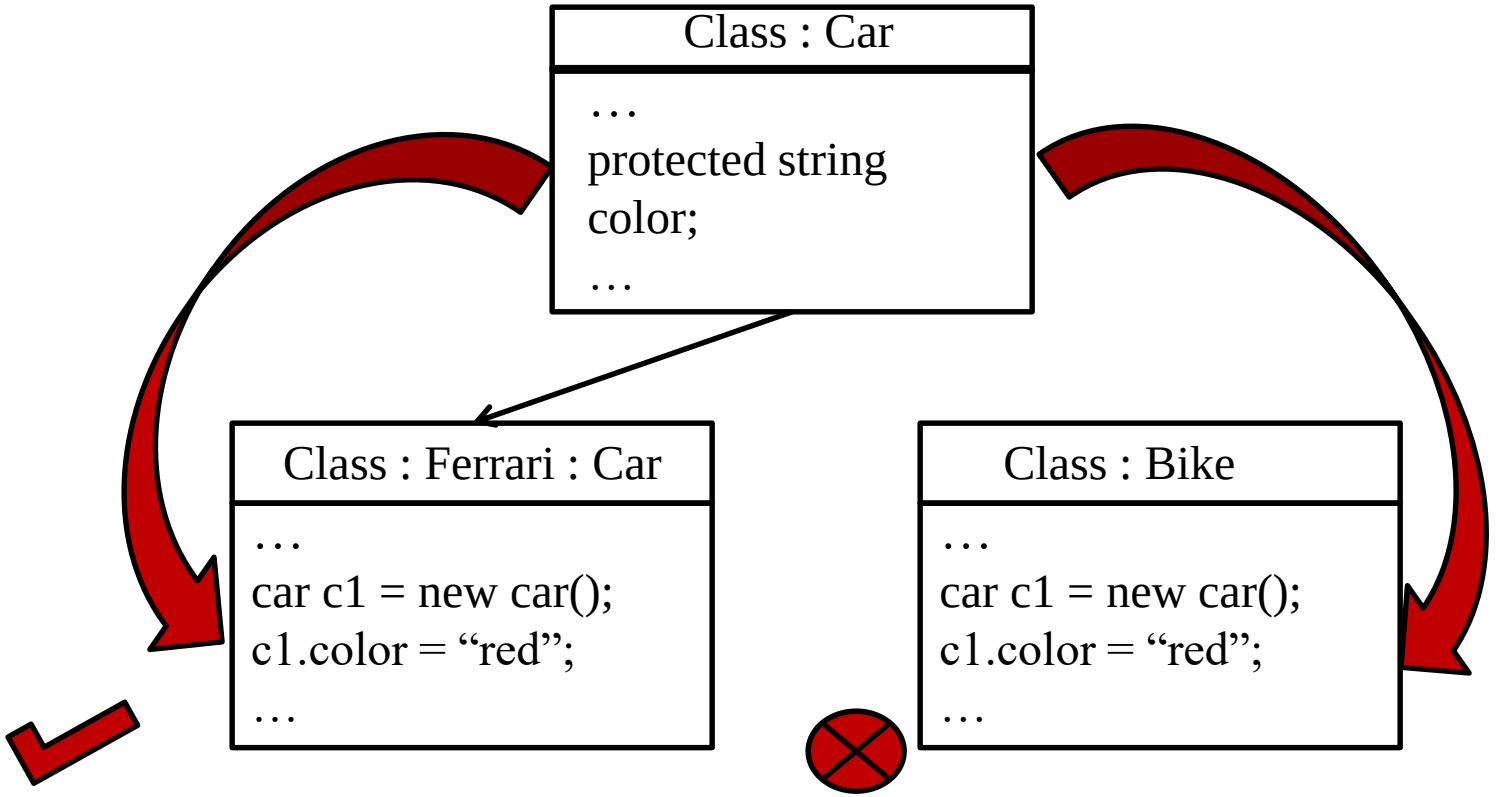
- protected
- internal
- protected internal

- `protected`: Allows a class to hide its member variables and member functions from other class objects and functions, except the child class.



`Ferrari` class is a child class of the `Car` class and wants to access the `color` variable.

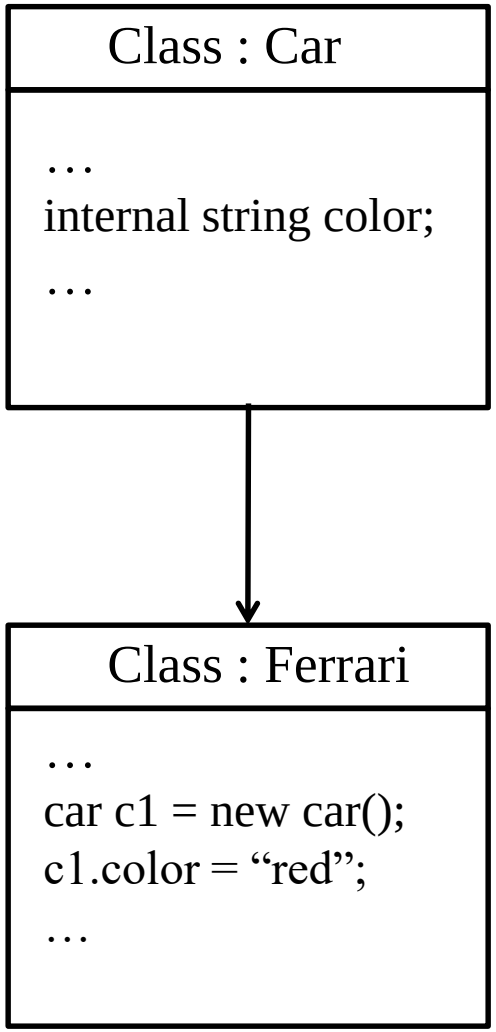
`Bike` class is another class, which wants to access the `color` variable.



Since the `color` variable is declared `protected`, it can be accessed by `Ferrari` class, which is a child class of `Car` class.

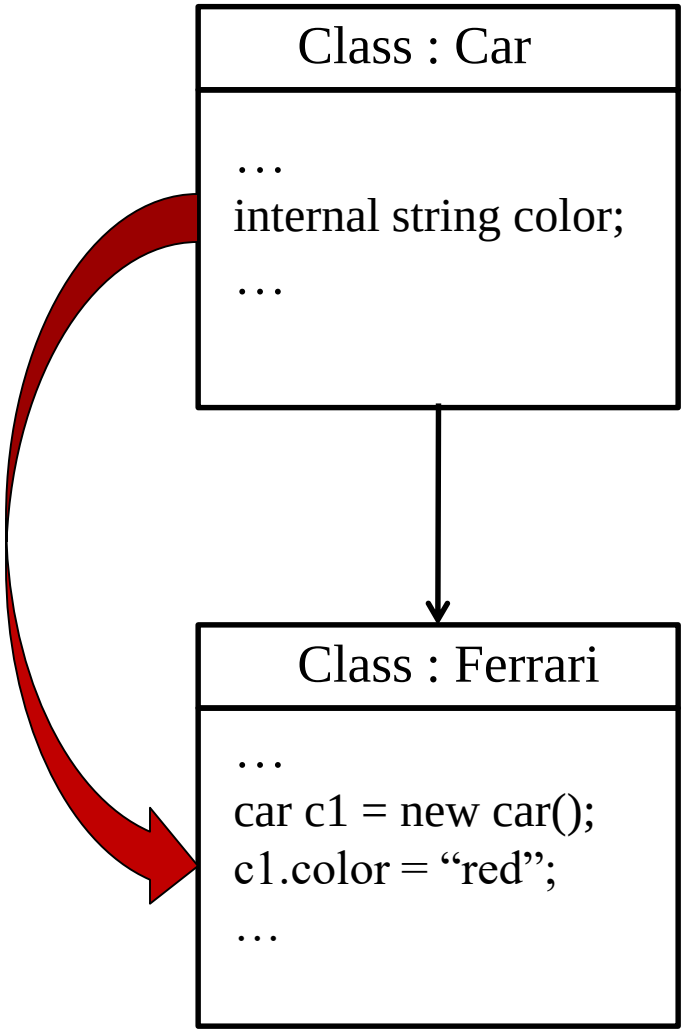
Since the `color` variable is declared `protected`, it cannot be accessed by `Bike` class .

- `internal`: Allows a class to expose its member variables and member functions to other class functions and objects.



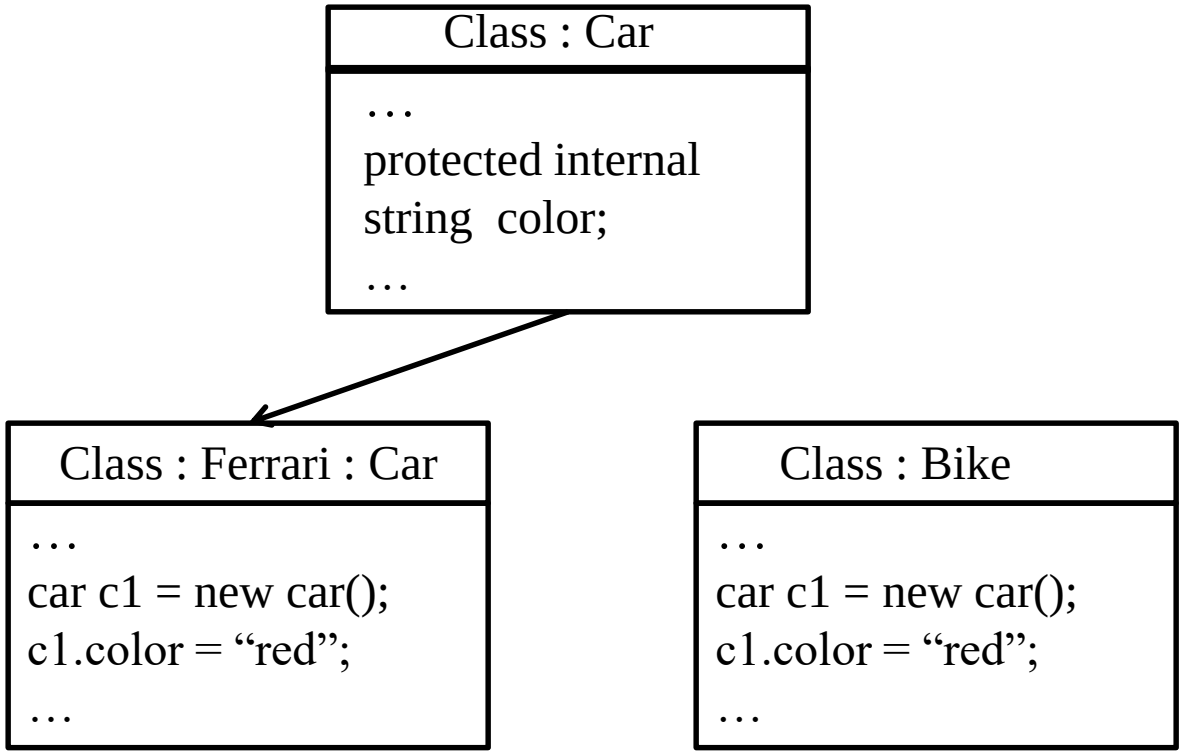
`Car` class contains a `color` variable, which is declared as `internal`.

`Ferrari` class, which is defined within the same application in which `Car` class is defined, wants to access the `color` variable.



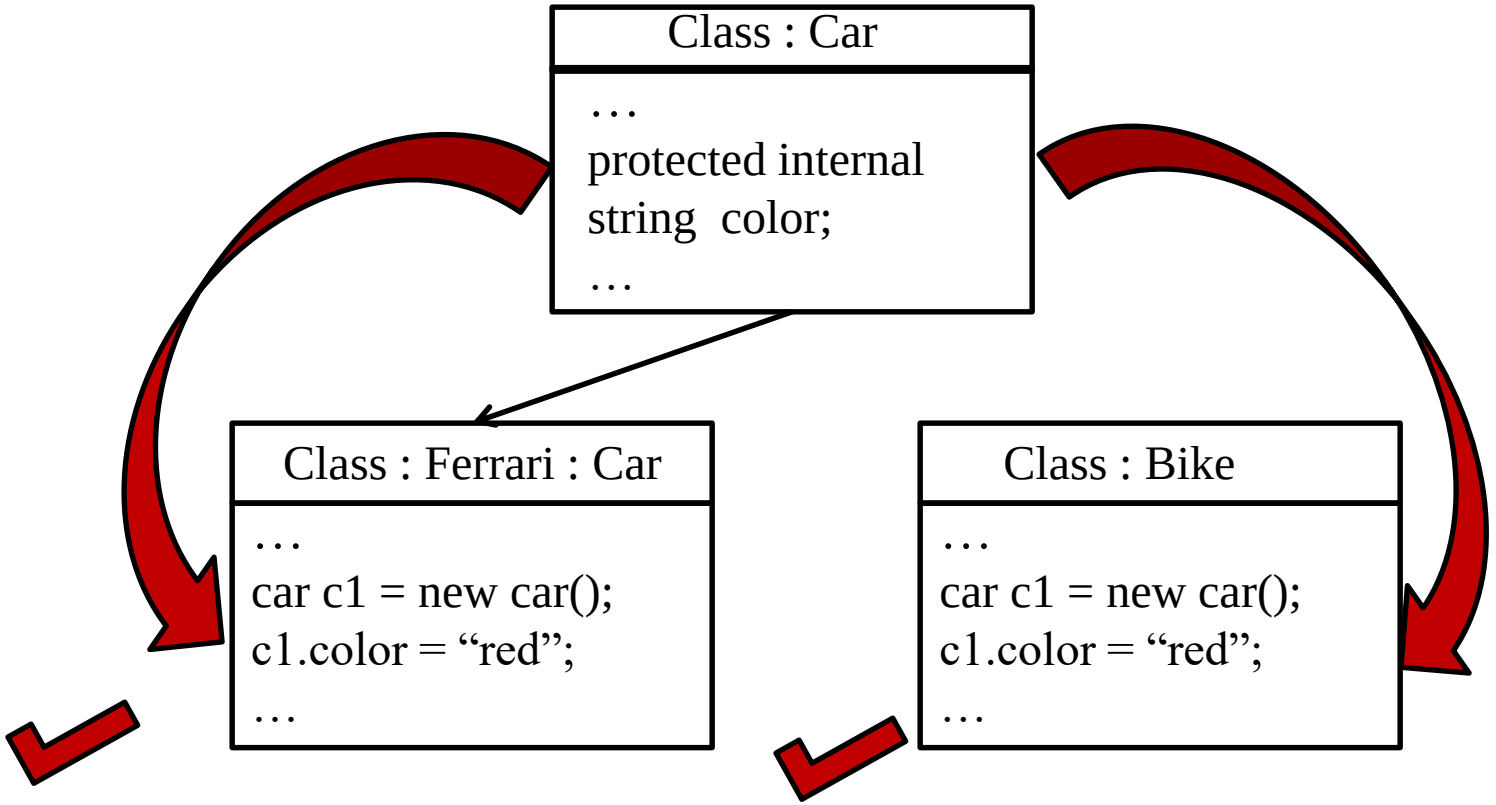
Since the `color` variable is declared `internal`, it can be accessed from any class or method defined within the application in which the member is defined.

- `protected internal`: Allows a class to expose its member variables and member functions to the containing class, child classes, or classes within the same application.



`Ferrari` class is a child class of `Car` class, which wants to access the `color` variable.

`Bike` class is another class, which is defined within the same application in which `Car` class is defined. It also wants to access the `color` variable.



Since the `color` variable is declared `protected internal`, it can be accessed by `Ferrari` class, which is a child class of `Car` class.

Since `color` variable is declared `protected internal`, it can be accessed by `Bike` class, which is defined within the same application.

- In this session, you learned that:
  - A class that inherits or derives attributes from another class is called the derived class and the class from which it is derived is called the base class.
  - Inheritance avoids redundancy in code and enables easy maintenance of code.
  - Constructors are called in the order of base-to-derived.
  - The `protected` access specifier allows a class to hide its member variables and member functions from other class objects and functions, just like the `private` access specifier while implementing inheritance.