Access Inherited Members with Base

To construct a Sedan, we must first construct an instance of its superclass Vehicle.

We can refer to a superclass inside a subclass with the base keyword.

For example, in Sedan:

```
base.SpeedUp();
```

refers to the SpeedUp() method in Vehicle.

There's special syntax for calling the superclass constructor:

```
class Sedan : Vehicle
{
  public Sedan (double speed) : base(speed)
  {
  }
}
```

The above code shows a Sedan that inherits from Vehicle. The Sedan constructor calls the Vehicle constructor with one argument, speed. This works as long as Vehicle has a constructor with one argument of type double.

Even if we don't use base() in Sedan, it will call a Vehicle constructor. Without an explicit call to base(), any subclass constructor will implicitly call the default parameterless constructor for its superclass. For example, this code implicitly calls Vehicle():

```
class Sedan : Vehicle
{
   // Implicitly calls base(), aka Vehicle()
   public Sedan (double speed)
   {
   }
}
```

The above code is equivalent to this:

public Sedan (double speed) : base()
{
}
}

This code will ONLY work if the constructor Vehicle() exists. If it doesn't, then an error will be thrown.

Instructions

1.

Currently, the Sedan constructor implicitly calls the Vehicle default parameterless constructor, also known as Vehicle().

Let's explicitly define a constructor in Vehicle. It should look similar to the Sedan constructor, with a few differences:

It has one parameter, double speed

Within the constructor, it sets Speed and LicensePlate

After doing this you may see the error below, which is good! It proves to us that the Sedan constructor is calling the parameterless constructor Vehicle(). Now that we have explicitly defined a constructor Vehicle(double speed), there is no more Vehicle(), hence the error.

error CS7036: There is no argument given that corresponds to the required formal parameter 'speed' of 'Vehicle.Vehicle(double)'

Hint



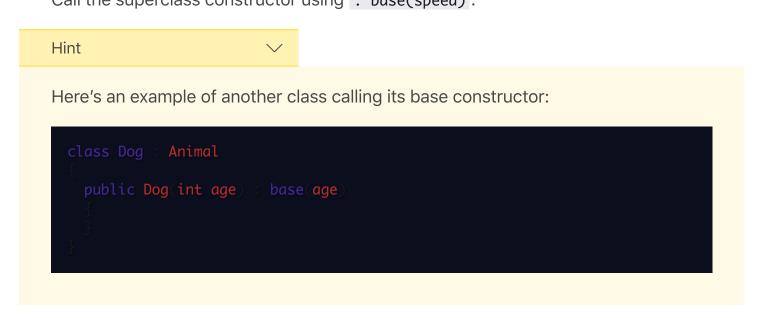
Remember that a class has a default parameterless constructor if no constructor is explicitly defined. This is the case with Vehicle.

When we define a constructor, like Vehicle(double speed), that parameterless constructor is no longer available.

2. Back in **Sedan.cs**:

Delete the lines in the constructor that set Speed and LicensePlate

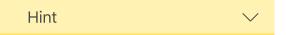
Call the superclass constructor using : base(speed).



3. Repeat the process in **Truck.cs**:

Delete the lines in the constructor that set Speed and LicensePlate

Call the superclass constructor using : base(speed)



At this point, the Truck constructor body should only set the Weight and Wheels properties. The Speed and LicensePlate properties are set in the Vehicle constructor, which is called via base(speed).

4. Since the LicensePlate and Speed properties defined in Vehicle are no longer accessed in Sedan or Truck, they no longer need to be protected. Switch those two setters to private.

