# Exercise: Defining Classes

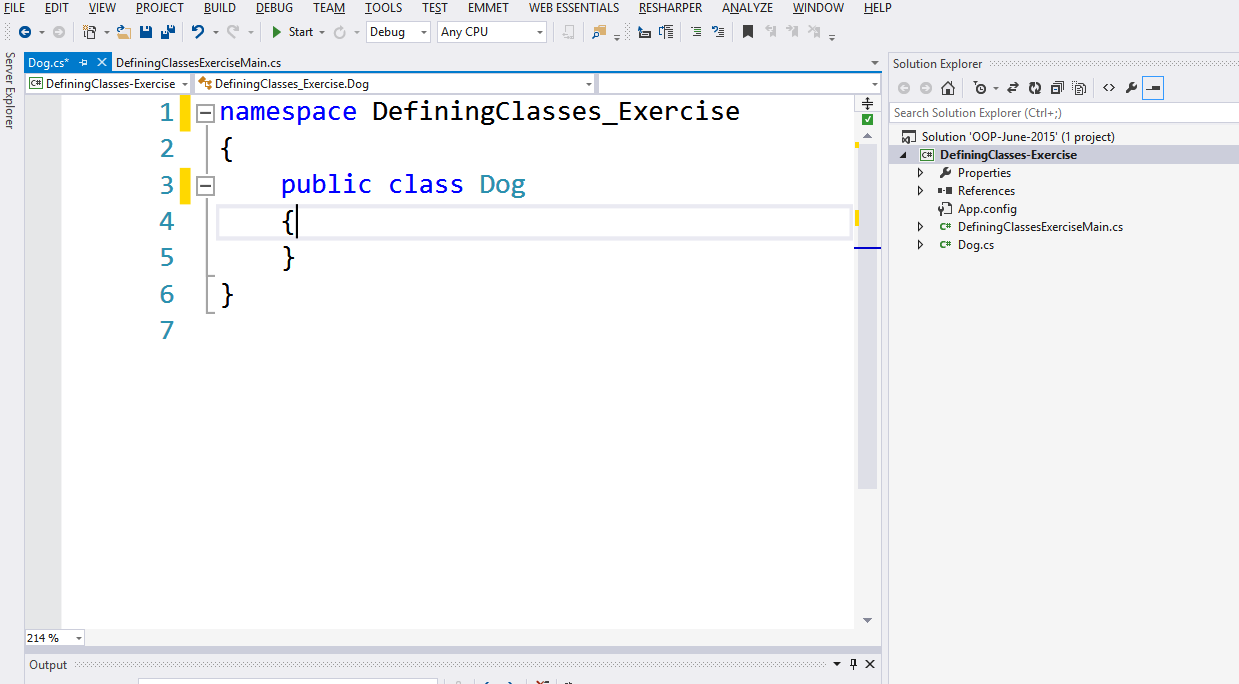
This document defines an in-class exercise from the ["OOP" Course @ Software University](https://softuni.bg/courses/oop/).

## Define a Class "Dog"

Your task is to define a simple class Dog. It should have **properties** **name** and **breed** (both **optional**). The class should allow you to **view** and **modify** the name and breed at any time. Finally, the dog should be able to Bark().

### Step 1. Create a Class

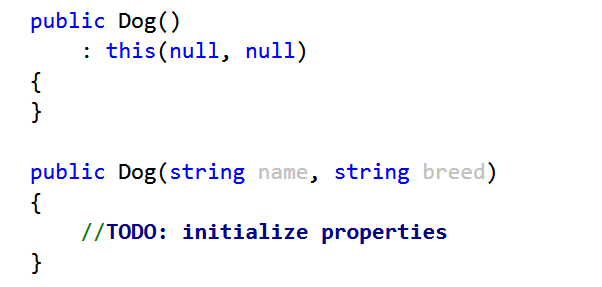
Create a new project in Visual Studio. Keep the class containing the Main() method unchanged for now. The class Dog should be in its own file (as a rule, there is a separate .cs file for each class). In Solution Explorer, right-click the project name and select **[Add] ? [Class]**. Name your class "Dog". Here is what the result should be:



### Step 2. Add Constructors

Constructors are special methods which **initialize** new objects of the given class. They are placed in the class body after all fields and before all properties and methods. They are usually declared with **public** access and their name is the same as the name of the class.

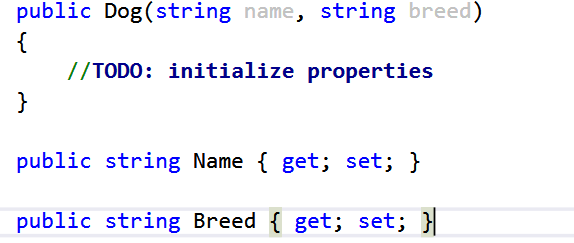
Add **two constructors**. The first should hold **no parameters** (since both the name and breed are optional, a user should be able to create a dog without entering them). The second constructor should have **both parameters**. Use constructor **chaining** – the first constructor should call the second one. A shortcut for creating a constructor is ctor **+ [TAB] + [TAB]**. Here is an example:



### Step 3. Add Properties

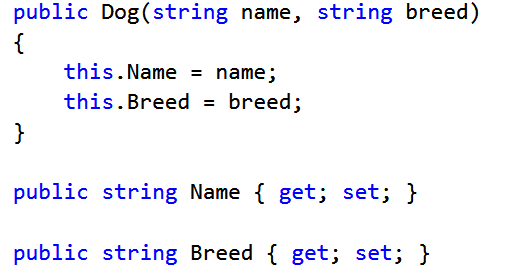
Properties allow you to **view** and **modify** the data kept in (private) fields. They are placed after fields and constructors, but before methods.

For this task, use automatic properties (no validation of data is required). A shortcut for adding a property is prop **+ TAB + TAB**. Both properties should be of type string. A property has the same name as the underlying field, but is Pascal-case.



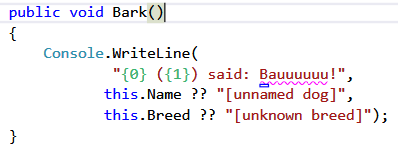
### Step 4. Assign the Properties in the Constructors

When you have properties, constructors should use them instead of initializing the fields directly. This is important in case the properties **validate** the data. Use the keyword "this" to refer to the current instance being created. A sample code:



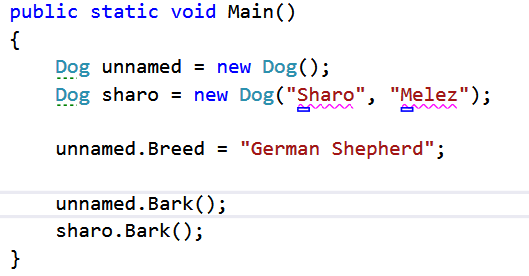
### Step 5. Add a Method Bark()

After all fields, constructors and properties come the methods. Add a **public** method Bark(), which should print some info on the console. It should have **no parameters and no return value (void)**. When accessing data from the current instance, prefer using the **properties** instead of the underlying fields directly.



### Step 6. Create a Few Dogs in the Main Class and Bark Them

The Dog class is now complete. It’s time to test it in the main method of your program. Create two dogs, one without name and breed and the other with both properties set. **Set** the breed of the first dog through the Breed property. Call the Bark() method on both dogs. Here is an example:



The result might look like this:

