

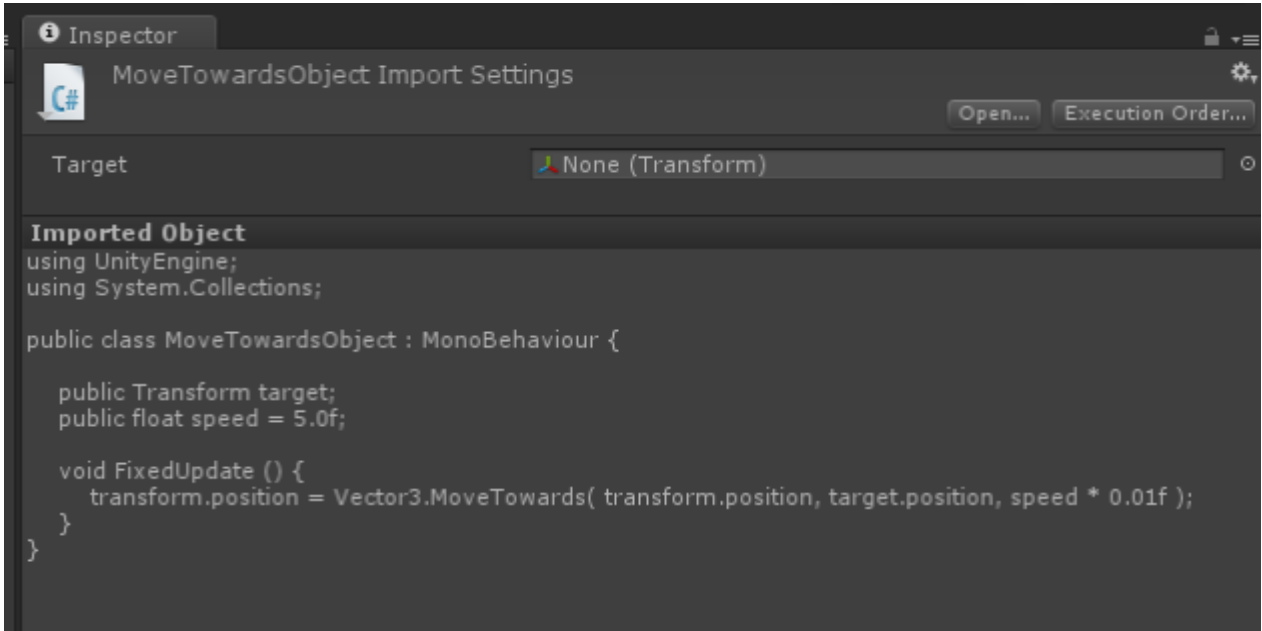
## Classes in Unity C#

### What is a class?

In Unity, a class is a text file\*, also referred to as a Script. Unity has a built-in compiler to interpret the file as programming instructions. Here is a C# script in the Project view:

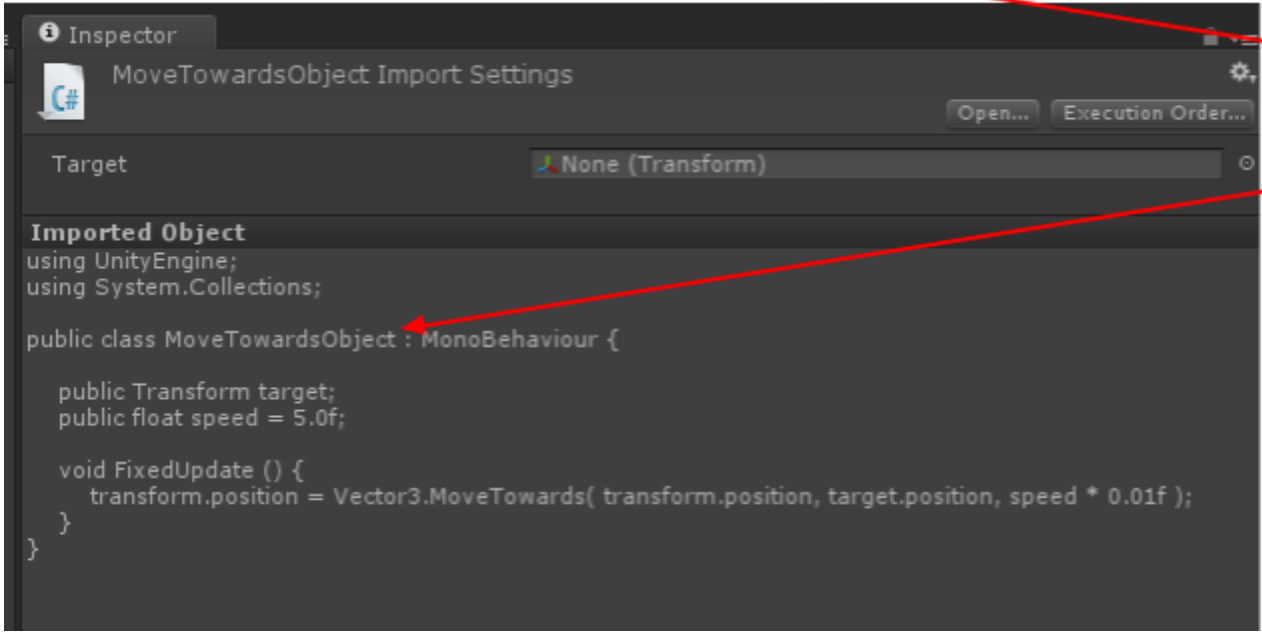


When selecting the Script, a preview of the code is show in in the Inspector:



\*Sometimes the Script or text file may contain more than one class, this is quite rare, but acceptable to Unity’s script compiler

**IMPORTANT!** name your file **exactly** the same as your class name, don’t start the name with a number, or have spaces in the name.



Name your class the same  
as your file

# Class structure

A class is structured in a particular way, which encapsulates its functionality between curly braces “{}”  
The code that Unity uses goes between the curly braces

```
public class MyClass {  
  
}
```

The Unity Game Engine has “hooks” or “event functions” to interact with its code, these can be accessed with the “using” statement.  
These are always entered **outside** the class definition, often before it:

```
using UnityEngine;  
using System.Collections;  
  
public class MyClass {  
  
}
```

## Setting up a class to use as a Component in the Unity Editor

To use a class as a Component in the Unity Editor, you need to extend the MonoBehaviour class.  
MonoBehaviour is a class contained in the Unity Engine with “hooks” or “event functions” for code to use.  
To enable this, simply add a colon “:” followed by “MonoBehaviour” to your class definition.  
Check you are “using” the “UnityEngine” also!

```
using UnityEngine;  
using System.Collections;  
  
public class MyClass : MonoBehaviour {  
  
}
```

If you have this in your class, you will be able to drag the file onto a GameObject in Unity’s Inspector

The code inside a class will contain:

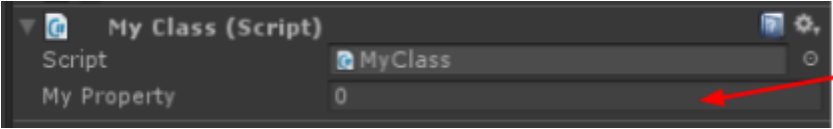
- **Properties** or
- **Methods**

## Properties

Properties are variables the class will use in its code.  
Properties are typically declared at the top of the class definition, inside the curly braces:

```
using UnityEngine;  
using System.Collections;  
  
public class MyClass : MonoBehaviour {  
  
    public int myProperty = 0;  
    private int myOtherProperty = 0;  
  
}
```

MyClass component in the Unity Editor



Note the Private property “myOtherProperty” **does not** appear for Editing in Unity

**IMPORTANT!** like classes, property names in a class have **no spaces**, and **no number at the beginning** of the name

## Access Modifiers

Properties can be used outside of the class by other classes, or the Unity Editor.  
To do this, they need to be declared as **Public**.  
If you only want to use a property inside the class, use **Private**.  
These are known as **Access Modifiers**.

Methods

Methods (sometimes called functions) are the lines of code that do the things in your game, like shooting, moving exploding etc. A method is typically a list of code to do one particular thing e.g. “TakeDamage” could make the player’s health go down, then check if any health is left. Methods encapsulate their lines of code between curly braces, like a class does:

```
using UnityEngine;
using System.Collections;

public class MyClass {

    public bool TakeDamage( int damageAmount ) {
        bool isZombieDead = false;

        return isZombieDead;
    }

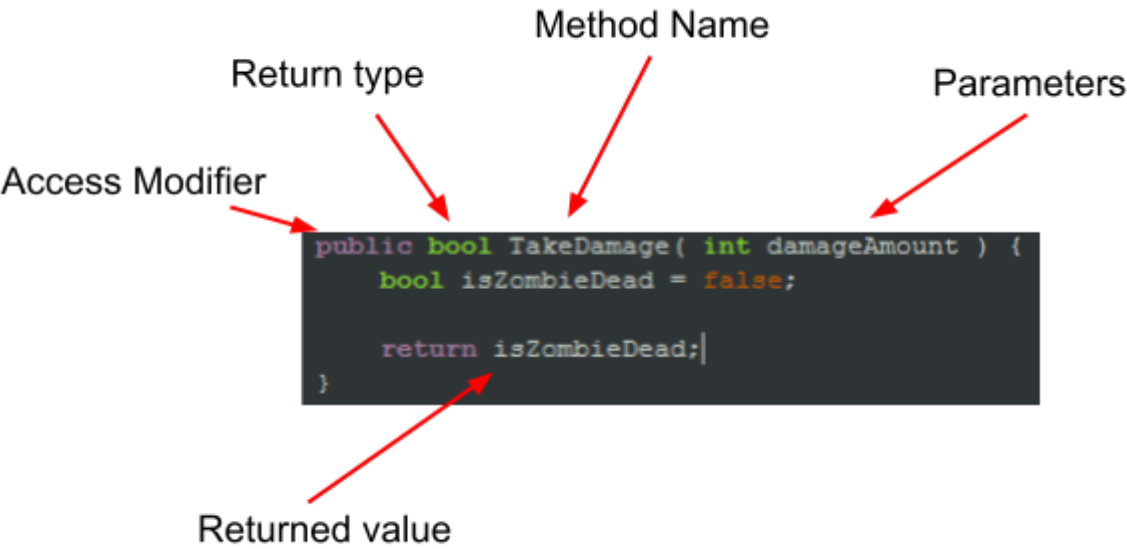
}
```

Method signatures

Methods have a “signature”, which refers to their first line.

A signature has 4 parts:

- Access Modifier
- Return type
- Name
- Parameters



Access Modifier

Like Properties, access to run the method from outside the class is controlled here. **Public** means other classes can use the method **Private** means they can’t The above example is accessible from other classes.

Return Type

When a method runs, the code that called that method may want a value returned. For example, if a Zombie took some damage which made its health go down to zero, the “TakeDamage” method returns a bool (true/false) to indicate if the zombie was killed. Another class or method could run the method and assign the return type to a variable:

```
bool isZombieDead = TakeDamage( 10 );
```

To not return anything after running the method, use the “**void**” keyword If the return type is not “void”, you must use the “**return**” keyword at the end of your method\* to return the value

**IMPORTANT!** The “return” keyword will cause the script **skip to the end of the method**, missing any code after

\*you can use the “return” keyword in several places, when using “if” or “switch” statements

Method Name

The name of the method used. This should be as descriptive as possible, remembering:

- No numbers at the beginning
- No spaces in the name
  - A standard way to write names is to start with a capital letter and capitalize each new word
    - e.g. TakeDamage, or DoSomethingInteresting

Parameters

Parameters are values the method needs up front in order to run.

If the “TakeDamage” method were used it would need an int inside its braces “()”:

```
TakeDamage( 10 );
```

If no number were entered, or a different type of value were entered, Unity would give an error.

```
TakeDamage(); // THIS WOULD GIVE AN ERROR
```

Using Parameters

If my method had a Parameter, the code inside the method can use that Parameter.

Here, the Parameter “damageAmount” is used to change the “isZombieDead” variable to true if “damageAmount” is greater than “>” a value of 100

```
void bool TakeDamage( int damageAmount ) {  
    bool isZombieDead = false;  
  
    if( damageAmount > 100 ) {  
        isZombieDead = true;  
    }  
  
    return isZombieDead  
}
```

Using this method from somewhere else, the code would read:

```
bool zombieWasKilled = TakeDamage( 23 ); // would be false  
bool zombieWasKilled = TakeDamage( 101 ); // would be true
```

# Class layout

other code you wish to use in this class

Class definition

Properties

Method

End of class

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class MyClass : MonoBehaviour {
5
6     public int myProperty = 0;
7     private int muOtherProperty = 0;
8
9     public bool TakeDamage( int damageAmount ) {
10         bool isZombieDead = false;
11
12         return isZombieDead;
13     }
14 }
15
```

## Gotchas

**Don't** create **methods** inside other **methods**:

```
using UnityEngine;
using System.Collections;

public class MyClass : MonoBehaviour {

    public bool DoSomething() {

        public void DoSomethingElse() {

        }

    }

}
```

The method “DoSomethingElse” is INSIDE the method “DoSomething”  
This will give an error

**Don't** create **properties** inside **methods**:

```
using UnityEngine;
using System.Collections;

public class MyClass : MonoBehaviour {

    public bool DoSomething() {
        public int myValue = 0;

    }

}
```

The property “myValue” is declared INSIDE the method “DoSomething”  
This will give an error

**Don't** create **methods** outside the **class definition**

```
using UnityEngine;
using System.Collections;

public class MyClass : MonoBehaviour {

}

public bool DoSomething() {

}
```

The method “DoSomething” is declared OUTSIDE the class definition  
This will give an error

**Don't** create **properties** outside the **class definition**

```
using UnityEngine;
using System.Collections;

public class MyClass : MonoBehaviour {

}

public int myValue = 0;
```

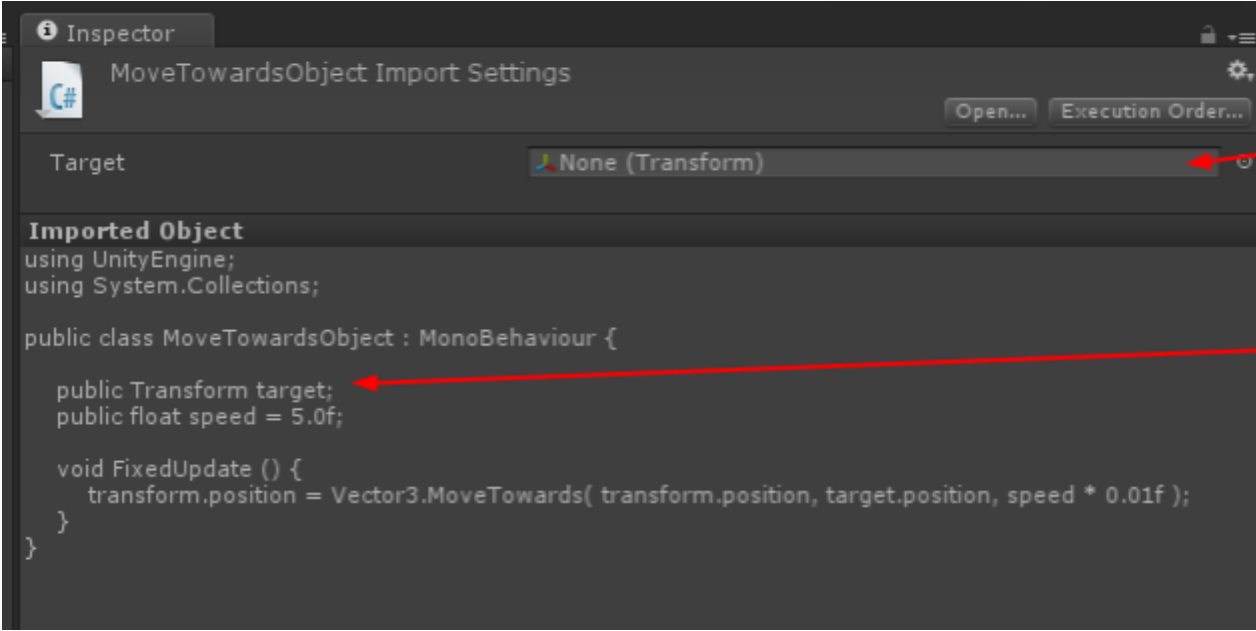
The property “myValue” is declared OUTSIDE the class definition  
This will give an error

# Using classes as components in the Unity Editor

## Scripts in the Inspector

**IMPORTANT!** If your script has errors, Unity will not show your script in the Inspector, or allow you to attach it to Components. Fix the errors first

Note the “Target” variable with inlet at the top.  
Note the line of code with the lower case “target”.



## Your class attached to a GameObject

