**C# Programming**

**CLASSES & OBJECTS**

**Orlando Unity3d Development Group**

**Date: 19-Apr-22**

**All Copyright GAMEDEVREJECTS©2022**

Contents

[**4.0**  **Classes & Objects** 3](#_Toc101255514)

[**4.1**  **What is a Class ?** 3](#_Toc101255515)

[**4.1.1** **Class Explainer Demo** 3](#_Toc101255516)

[**4.2** **The Class Declaration** 4](#_Toc101255517)

[**4.3** **When and Why We Use Custom Classes ?** 5](#_Toc101255518)

[**4.4**  **Create a Custom class** 5](#_Toc101255519)

[**4.5**  **Create an Object** 6](#_Toc101255520)

[**4.6** **Class Members** 6](#_Toc101255521)

[**4.7** **Class Fields** 7](#_Toc101255522)

[**4.8** **Creating Multiple Objects** 9](#_Toc101255523)

[**Glossary** 11](#_Toc101255524)

[**Resources** 12](#_Toc101255525)

# **4.0 Classes & Objects**

Classes and Objects.

# **4.1 What is a Class ?**

A **Class is a blueprint** or template from which Objects are created.

A **Class aka "Object Constructor" is used to instantiate or Create Objects** => Objects are the "building blocks" for programs.

A **Class is a container for a "block of code"**, that contains all your variables, your functions, your events etc. for your script.

# **4.1.1 Class Explainer Demo**

Now in unity in the Scene View,…at a very general high level. If I create a gameobject, let’s say a cube. The cube has a certain size with a specific a length, a height, a width, a weight, and a colour characteristic of grey.

So if I create a 2nd GameObject say a sphere, then add a material to it,…would you agree, we could that its different, and looks distinct its “round it’s not squarish” in comparison to the 1st gameobject the cube we just created ?

Now stay with me, what if I created a 3rd cube, with the exact same size = length x height x width, weight, and characteristics like the 1st gameobject a cube ?…you would say now that it looks different to the 2nd GameObject but at the same time it looks the same as the 1st gameObject. In fact there is now doubt in our minds that the 3rd object and 1st object are the same, right ? In fact you could say the "process" or "template" used to create the 1st and 3rd Game Object is the same….so that template we used to create two or more of the same objects, is what we then call the **class.** In other words a class is a template for the 2 game objects we just created.

Graphical user interface

Description automatically generated

…does that make sense now what a class is ?, Clear as crystal ? So the reasoning is that…when you believe you need to create two or more objects that are likely to be very similar or the same type, you create a class, a new template.

Otherwise you end up in a situation creating lots, and lots of individual scripts and programs to create the same thing, which is inefficient, and increase the probability . For example what if in unity you were creating a Car Racing game, with 10 different racing cars like in the cartoon "Whacky Races". You would have 2 basic options 1)Create scripts for 10 to represent the different Car types for make, model, different car features etc.. OR acknowledge they are all cars of a certain type of vehicle, create 1 custom car class to cover all the cars

Obviously the 2nd option is more efficient to use, and so what we do in Unity is create a specific custom class for Cars.

# **4.2 The Class Declaration**

(NavigateTo).SCENE -- (IN).HIERARCHY -- (Right Click).Create Empty => / GameObject /

(IN).PROJECT -- Assets -- \_SCRIPTS -- (Right Click).Create.C# Script => / NewBehaviourScript -- (Rename).NameClass /

(IN).\_SCRIPTS -- (Select).NameClass.(Drag and Drop) => / (Onto).HIERARCHY.GameObject /

(IN).\_SCRIPTS -- (Select).NameClass.(Double Left Click) => / VSCODE /

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| using System.Collections; |  |  |  |  |  |  |  |  |  |
| using System.Collections.Generic; |  |  |  |  |  |  |  |  |  |
| using UnityEngine; |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| public class NameClass : MonoBehaviour |  |  |  |  |  |  |  |  |  |
| { |  |  |  |  |  |  |  |  |  |
| // Start is called before the first frame update | | | |  |  |  |  |  |  |
| void Start() |  |  |  |  |  |  |  |  |  |
| { |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| } |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| // Update is called once per frame |  |  |  |  |  |  |  |  |  |
| void Update() |  |  |  |  |  |  |  |  |  |
| { |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| } |  |  |  |  |  |  |  |  |  |
| } |  |  |  |  |  |  |  |  |  |

In Unity when you create a new script. By default the name of the script is the name of the class.

Now when you open the script in VSCODE or VS. automatically above the name of the class, you have three using statements:

|  |
| --- |
| using System.Collections; |
| using System.Collections.Generic; |
| using UnityEngine; |

…these (above) are called namespaces, which give you access to the code libraries:

* **Using Unity Engine gives you access to the Unity API's i.e. Unity API (Application Programming Interface)**
* **Using System.Collections = gives you access to the Microsoft C# code libraries**

Now If we "analyze" the class statement

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| public class NameClass : MonoBehaviour |  |  |  |  |  |  |  |  |  |

* **public class NameOfClass : MonoBehaviour**

Adding the public before the class is the access modifier means this class is not private, not hidden, in other words the **NameOfClass** **class** is accessible to all other classes in the world !

Then the class keyword declares the class inside of the script followed by the "name of our class" i.e. **NameOfClass**

Then following the **name we use a semi-colon ":" - which stands for "Inheritance"** => which means this named class “inherits” all the code, keywords, libraries from MonoBehaviour which uses the Unity Engine library.

**Monobehaviour** = basically allows you to add scripts to gameObjects

***Monobehaviour*** *is the parent or base class; where you inherit the Unity APIs to access all the (code) methods, keywords, and commands you need to use in your script*.

If we comment out //Using UnityEngine, then we get an error with Monobehaviour => then void Start and Void Update don't work

# **4.3 When and Why We Use Custom Classes ?**

I want to take a moment to explain when and why we use Custom Classes.

Let’s consider an example if we were building a typical RPG (Role-Playing Game), and you need to create some items for the player character in the game to collect.

Say you wanted to have a sword in your game, and you created a sword script containing all the attributes and behaviour related to the sword item.

And let's say you want to add a shield item, and a helmet item.

As you can see I have a "bunch" of scripts for each item in the game that defines their characteristics, traits etc.

Well answer me this,...if I have 50 items, or 100 items in my game, does it make sense to create,…50-100 C# scripts, and add them to my game ?

Obviously Not ! Also what do all the items I've just created have in common ? Well they all have names, they all have item IDs, they probably all have shared stats amongst each other - which allows us to couple or aggregate them into 1 single class called item, AND then define them, make sense ?

Also we use Custom Classes, because if you try to contain all your code for calculations in the main class, it could get very messy and you increase the probability of introducing bugs into your script.

We often use Custom Classes for better organization of our scripts (one class has all the fields and methods, while the other class inherits from Monobehaviour (i.e. the code to be executed)).

# **4.4 Create a Custom class**

In Unity a custom class is just a Class that does not inherit from Monobehaviour.

We use Custom Classes, because if you try to contain all your code for calculations in the main class, it’s going to be messy, and you will increase the probability of introducing bugs into your script

We often use Custom Classes for better organization of classes (one class has all the fields and methods, while the other class inherits from Monobehaviour (i.e. the code to be executed).

Again to create a class, we use the **class** keyword. So let’s create a class named "Spaceship" with a variable color red as follows:

(NavigateTo).SCENE -- (IN).HIERARCHY -- (Right Click).Create Empty => / GameObject /

(IN).PROJECT -- Assets -- \_SCRIPTS -- (Right Click).Create.C# Script => / NewBehaviourScript -- (Rename).**Spaceship** /

(IN).\_SCRIPTS -- (Select).**Spaceship**.(Drag and Drop) => / (Onto).HIERARCHY.GameObject /

(IN).\_SCRIPTS -- (Select).**Spaceship**.(Double Left Click) => / VSCODE /

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| using System.Collections; |  |  |  |  |  |  |  |  |  |
| using System.Collections.Generic; |  |  |  |  |  |  |  |  |  |
| using UnityEngine; |  |  |  |  |  |  |  |  |  |
| public class Spaceship |  |  |  |  |  |  |  |  |  |
| { |  |  |  |  |  |  |  |  |  |
| public string \_color = "red"; |  |  |  |  |  |  |  |  |  |
| } |  |  |  |  |  |  |  |  |  |

Let’s consider some example’s so we can get to grips with the syntax.

(NavigateTo).SCENE -- (IN).HIERARCHY -- (Right Click).Create Empty => / GameObject /

(IN).PROJECT -- Assets -- \_SCRIPTS -- (Right Click).Create.C# Script => / NewBehaviourScript -- (Rename).CoroutineEx2/

(IN).\_SCRIPTS -- (Select).CoroutineEx2.(Drag and Drop) =>/ (Onto).HIERARCHY.GameObject/

(IN).\_SCRIPTS =>/(Select).CoroutineEx2.(Double Left Click) -- VSCODE/

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

# 

# **4.5 Create an Object**

An object is created from a class. We've already created the class named Spaceship, so now we can use this class to create objects (i.e. different "types" of Spaceships). Therefore to create an object of Spaceship Type, we specify the class name, followed by the object name, and must use the keyword new:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| using System.Collections; |  |  |  |  |  |  |  |  |  |
| using System.Collections.Generic; |  |  |  |  |  |  |  |  |  |
| using UnityEngine; |  |  |  |  |  |  |  |  |  |
| public class Main Class: MonoBehaviour |  |  |  |  |  |  |  |  |  |
| { |  |  |  |  |  |  |  |  |  |
| // Start is called before the first frame update | | | |  |  |  |  |  |  |
| void Start() |  |  |  |  |  |  |  |  |  |
| { |  |  |  |  |  |  |  |  |  |
| Spaceship rocket1 = new Spaceship(); | |  |  |  |  |  |  |  |  |
| Debug.Log($"This Spaceship is {rocket1.\_color}"); | | | | |  |  |  |  |  |
| } |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| // Update is called once per frame |  |  |  |  |  |  |  |  |  |
| void Update() |  |  |  |  |  |  |  |  |  |
| { |  |  |  |  |  |  |  |  |  |
| // |  |  |  |  |  |  |  |  |  |
| } |  |  |  |  |  |  |  |  |  |
| } |  |  |  |  |  |  |  |  |  |

# **4.6 Class Members**

**Fields and methods inside classes** are often referred to as "Class Members". For example, in unity let’s create a Spaceship class with 3 class members: 2 fields and 1 method, as follows

(NavigateTo).SCENE -- (IN).HIERARCHY -- (Right Click).Create Empty => / GameObject /

(IN).PROJECT -- Assets -- \_SCRIPTS -- (Right Click).Create.C# Script => / NewBehaviourScript -- (Rename).**Spaceship** /

(IN).\_SCRIPTS -- (Select).**Spaceship**.(Drag and Drop) => / (Onto).HIERARCHY.GameObject /

(IN).\_SCRIPTS -- (Select).**Spaceship**.(Double Left Click) => / VSCODE /

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Spaceship//The Class

{

    public string color = "Red"; //field

    public int maxSpeed = 200; //field

    public void WarpSpeed() //method

    {

        Debug.Log("This Spaceship is going WarpSpeed fast !!");

    }

}

# **4.7 Class Fields**

In the previous section, you learned that **variables inside a class are called fields**, and that you can access them by creating an object of the class, then use dot syntax (.) notation to access them. In the following example we will create an object of the Spaceship class, with the name rocket. Then output the print value of the fields **color and maxSpeed** to the Console.

[ NavigateTo).SCENE -- (IN).HIERARCHY -- (Right Click).Create Empty => / GameObject /

(IN).PROJECT -- Assets -- \_SCRIPTS -- (Right Click).Create.C# Script => / NewBehaviourScript -- (Rename).**MainClass** /

(IN).\_SCRIPTS -- (Select).**MainClass**.(Drag and Drop) => / (Onto).HIERARCHY.GameObject /

(IN).\_SCRIPTS -- (Select).**MainClass**.(Double Left Click) => / VSCODE /

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MainClass : MonoBehaviour

{

    // Start is called before the first frame update

    void Start()

    {

        Spaceship rocket = new Spaceship();

        Debug.Log($"The rocket is painted {rocket.color}");

        Debug.Log($"The Rocket Max Speed is {rocket.maxSpeed}");

        rocket.WarpSpeed();

    }

    // Update is called once per frame

    void Update()

    {

    }

}

public class Spaceship //Class

{

    public string color = "RED"; //field

    public int maxSpeed = 200; //field

    public void WarpSpeed() //method

    {

        Debug.Log("This Spaceship is going very fast !!");

    }

}

Graphical user interface, application

Description automatically generated

You can also leave the fields blank, and modify them when creating the object:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MainClass : MonoBehaviour

{

    // Start is called before the first frame update

    void Start()

    {

        Spaceship rocket = new Spaceship();

        rocket.color = "BLUE";

        rocket.maxSpeed = 500;

        Debug.Log($"The rocket is painted {rocket.color}");

        Debug.Log($"The Rocket Max Speed is {rocket.maxSpeed}");

        rocket.WarpSpeed("very fast");

    }

    // Update is called once per frame

    void Update()

    {

    }

}

public class Spaceship //Class

{

    public string color; //field is blank

    public int maxSpeed; //field is blank

    public void WarpSpeed(string txt) //method

    {

        Debug.Log($"This Spaceship is going {txt} !!");

    }

}

# **4.8 Creating Multiple Objects**

Creating multiple objects of one class. E.g. Using the Spaceship class, we’ll create 1 x **rocket** and 1 x **DeathStar** as follows:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MainClass : MonoBehaviour

{

    // Start is called before the first frame update

    void Start()

    {

        Spaceship rocket = new Spaceship(); //1st object

        rocket.color = "BLUE";

        rocket.maxSpeed = 500;

        Debug.Log($"The rocket is painted {rocket.color}");

        Debug.Log($"The Rocket Max Speed is {rocket.maxSpeed}");

        rocket.WarpSpeed("very fast");

        Spaceship deathStar = new Spaceship(); //2nd object

        deathStar.color = "BLACK";

        deathStar.maxSpeed = 1000;

        Debug.Log($"The rocket is painted {deathStar.color}");

        Debug.Log($"The Rocket Max Speed is {deathStar.maxSpeed}");

        deathStar.WarpSpeed("extremely fast");

    }

    // Update is called once per frame

    void Update()

    {

    }

}

public class Spaceship //Class

{

    public string color; //field is blank

    public int maxSpeed; //field is blank

    public void WarpSpeed(string txt) //method

    {

        Debug.Log($"This Spaceship is going {txt} !!");

    }

}Graphical user interface

Description automatically generated

# **Glossary**

Class a class is a blueprint or template for Objects aka the “Object Constructor”; it contains all your variables, methods for your program script

Monobehaviour Monobehaviour is the parent or base class; where you inherit the Unity APIs to access all the (code) methods, keywords, and commands you need to use in your script

Namespace Namespaces are used to avoid naming conflicts. When you start to add libraries from other programmers its highly likely we are going to use the same names. Therefore, we create a namespace as a unique group name identifier or reference for the collection of classes that it contains.

# **Resources**

/End