C#

(C Sharp)

Classes and Instances

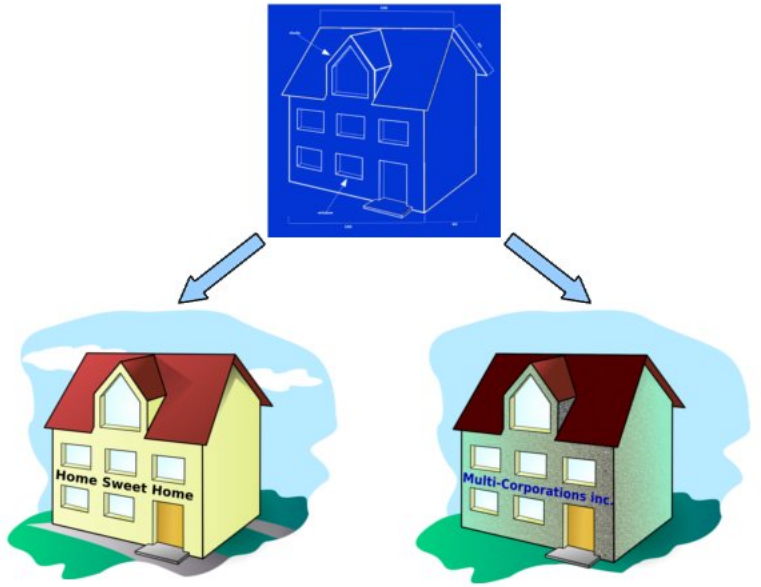
(OOP)

Classes & Instances

- Encapsulation: organize variables and functions together
- Code reuse

Analogy Time

- Blueprint -> House
- Cookie Cutter -> Cookie
- Person -> Bob

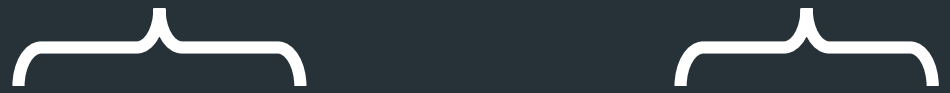


http://processing.lyndondaniels.com/53blueprint.php

Class

ACCESS MODIFIER

CLASS NAME



```
public class Enemy {
    // Fields and methods go inside brackets
}
```

Fields

Fields

```
public class Enemy {
    // Fields
    public string Name;
}
```

ACCESS MODIFIER

VARIABLE TYPE

VARIABLE NAME

INSTANCE

CLASS

```
public class ClassDemo : MonoBehaviour {

    void Start () {
        Enemy enemy1 = new Enemy();
    }

}

public class Enemy {
    // Fields
    public string Name;
}
```

VARIABLE NAME

CONSTRUCTOR

```
Enemy enemy1 = new Enemy();
```

VARIABLE TYPE

KEYWORD

INSTANCE

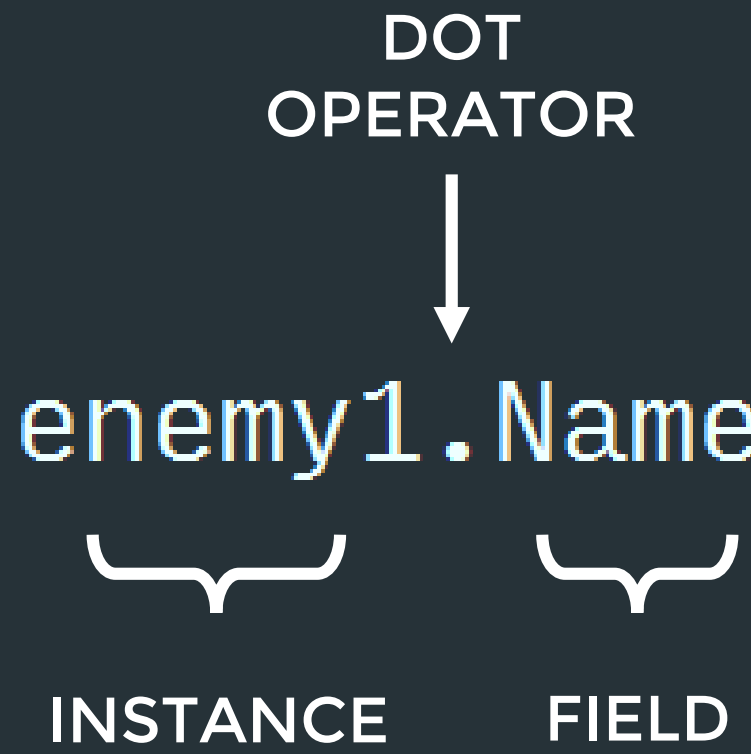
CLASS

```
public class ClassDemo : MonoBehaviour {

    void Start () {
        Enemy enemy1 = new Enemy();
        enemy1.Name = "Carl The Goblin";
        Debug.Log("Monster 1 is: " + enemy1.Name);
    }

}

public class Enemy {
    // Fields
    public string Name;
}
```


DOT
OPERATOR
enemy1.Name = "Carl the Goblin";
INSTANCE
FIELD

INSTANCE

INSTANCE

CLASS

```
public class ClassDemo : MonoBehaviour {

    void Start () {
        Enemy enemy1 = new Enemy();
        enemy1.Name = "Carl The Goblin";
        Debug.Log("Monster 1 is: " + enemy1.Name);

        Enemy enemy2 = new Enemy();
        enemy2.Name = "Radcliff";
        Debug.Log("Monster 2 is: " + enemy2.Name);
    }

}

public class Enemy {
    // Fields
    public string Name;
}
```

Constructors

```
public class Enemy {
    // Fields
    public string Name;

    // Constructor
    public Enemy(string name) {
        Name = name;
    }
}
```

```
public class ClassDemo : MonoBehaviour {
    void Start () {
        Enemy enemy1 = new Enemy("Carl The Goblin");
        Debug.Log("Enemy 1 is: " + enemy1.Name);
    }
}

public class Enemy {
    // Fields
    public string Name;

    // Constructor
    public Enemy(string name) {
        Name = name;
    }
}
```

Methods

```
public class Enemy {
    // Fields
    public string Name;

    // Constructor
    public Enemy(string name) {
        Name = name;
    }

    // Methods
    public void Speak() {
        Debug.Log("Hello, I am " + Name + ".");
    }
}
```

```
Enemy enemy1 = new Enemy("Carl The Goblin");
enemy1.Speak();
```

More Methods

```
public class Enemy {
    // Fields
    public string Name;
    private int Health;

    // Constructor
    public Enemy(string name, int health) {
        Name = name;
        Health = health;
    }

    // Methods
    public void Speak() {
        Debug.Log("Hello, I am " + Name + ".");
    }
    public void TakeDamage(int damage) {
        Health = Health - damage;
    }
    public void SayHealth() {
        if (Health < 0) {
            Debug.Log(Name + ": I am dead :(");
        } else {
            Debug.Log(Name + ": I have " + Health + " health");
        }
    }
}
```

```
// Create a Rectangle class:
//  - It should have public fields for Width & Height
//  - It should have a constructor that can initialize the Width and Height
//  - It should have a method GetArea that returns a float
//  - It should have a method GetPerimeter that returns a float
//
// Test it by creating a couple Rectangle instances and calculating the area
// and perimeter
```

Scripts are Classes

```
public class Rotator : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

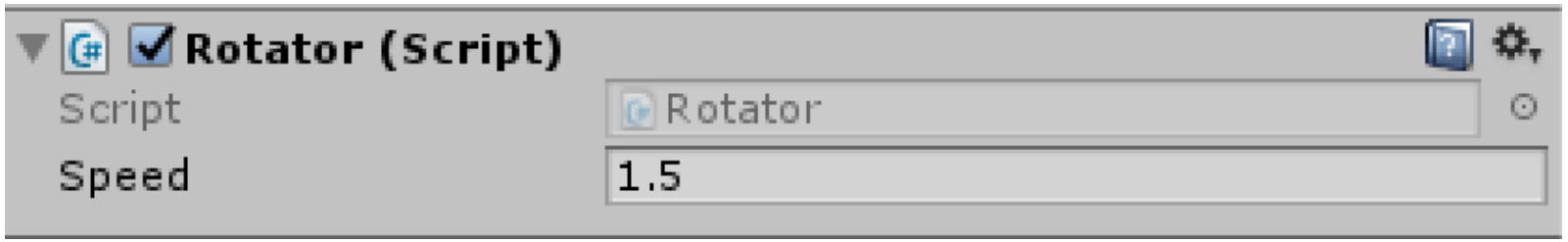
```
public class Rotator : MonoBehaviour {

    public float speed;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        float rotationAmount = speed * Time.deltaTime;
        transform.Rotate(rotationAmount, 0f, 0f);
    }
}
```




```
public class Rotator : MonoBehaviour {

    public float speed = 10;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        float rotationAmount = speed * Time.deltaTime;
        transform.Rotate(rotationAmount, 0f, 0f);
    }
}
```

Accessing Components

Via Inspector

```
public class LightColorSwitcher : MonoBehaviour {

    public Light LightComponent;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

Variables

areaSize	The size of the area light. Editor only.
bakedIndex	A unique index, used internally for identifying lights contributing to lightmaps and/or light probes.
bounceIntensity	The multiplier that defines the strength of the bounce lighting.
color	The color of the light.
commandBufferCount	Number of command buffers set up on this light (Read Only).
cookie	The cookie texture projected by the light.
cookieSize	The size of a directional light's cookie.
cullingMask	This is used to light certain objects in the scene selectively.
flare	The flare asset to use for this light.
intensity	The Intensity of a light is multiplied with the Light color.
isBaked	Is the light contribution already stored in lightmaps and/or lightprobes (Read Only).
range	The range of the light.
renderMode	How to render the light.
shadowBias	Shadow mapping constant bias.
shadowCustomResolution	The custom resolution of the shadow map.
shadowNearPlane	Near plane value to use for shadow frustums.
shadowNormalBias	Shadow mapping normal-based bias.
shadowResolution	Control the resolution of the ShadowMap.
shadows	How this light casts shadows
shadowStrength	Strength of light's shadows.
spotAngle	The angle of the light's spotlight cone in degrees.
type	The type of the light.

http://docs.unity3d.com/ScriptReference/Light.html

Via Scripting

```
public class LightColorSwitcher : MonoBehaviour {

    private Light LightComponent;

    // Use this for initialization
    void Start () {
        LightComponent = GetComponent<Light>();
    }

    // Update is called once per frame
    void Update () {

    }
}
```

Generic Method

```
LightComponent = GetComponent<Light>();
```

TYPE OF
COMPONENT

RequireComponent

class in UnityEngine

Description

The RequireComponent attribute automatically adds required components as dependencies.

When you add a script which uses RequireComponent to a GameObject, the required component will automatically be added to the GameObject. This is useful to avoid setup errors. For example a script might require that a Rigidbody is always added to the same GameObject. Using RequireComponent this will be done automatically, thus you can never get the setup wrong. Note that RequireComponent only checks for missing dependencies during the moment the component is added to a GameObject. Existing instances of the component whose GameObject lacks the new dependencies will not have those dependencies automatically added.

```
using UnityEngine;

// PlayerScript requires the GameObject to have a Rigidbody component
[RequireComponent (typeof (Rigidbody))]
public class PlayerScript : MonoBehaviour {
        Rigidbody rb;

        void Start() {
                rb = GetComponent<Rigidbody>();
        }
        void FixedUpdate()  {
                rb.AddForce(Vector3.up);
        }
}
```

https://docs.unity3d.com/ScriptReference/RequireComponent.html

Mathf

http://docs.unity3d.com/ScriptReference/Mathf.html

Mathf.Repeat

public static float **Repeat**(float **t**, float **length**);

Parameters

Description

Loops the value t, so that it is never larger than length and never smaller than 0.

This is similar to the modulo operator but it works with floating point numbers. For example, using 3.0 for `t` and 2.5 for `length`, the result would be 0.5. With `t` = 5 and `length` = 2.5, the result would be 0.0. Note, however, that the behaviour is not defined for negative numbers as it is for the modulo operator.

In the example below the value of time is restricted between 0.0 and just under 3.0. This is then used to keep the x position in this range.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        transform.position = new Vector3(Mathf.Repeat(Time.time, 3), transform.position.y, transform.position.z);
    }
}
```

http://docs.unity3d.com/ScriptReference/Mathf.Repeat.html

Mathf.PingPong

public static float **PingPong**(float **t**, float **length**);

Parameters

Description

PingPongs the value t, so that it is never larger than length and never smaller than 0.

The returned value will move back and forth between 0 and `length`.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    void Update() {
        transform.position = new Vector3(Mathf.PingPong(Time.time, 3), transform.position.y, transform.position.z);
    }
}
```

http://docs.unity3d.com/ScriptReference/Mathf.PingPong.html

Color

http://docs.unity3d.com/ScriptReference/Color.html

Color

struct in UnityEngine

Description

Representation of RGBA colors.

This structure is used throughout Unity to pass colors around. Each color component is a floating point value with a range from 0 to 1.

Components (r,g,b) define a color in RGB color space. Alpha component (a) defines transparency - alpha of one is completely opaque, alpha of zero is completely transparent.

https://docs.unity3d.com/ScriptReference/Color.html

Color Constructor

public **Color**(float **r**, float **g**, float **b**, float **a**);

Parameters

r	Red component.
g	Green component.
b	Blue component.
a	Alpha component.

Description

Constructs a new Color with given r,g,b,a components.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Color color = new Color(0.2F, 0.3F, 0.4F, 0.5F);
}
```

https://docs.unity3d.com/ScriptReference/Color-ctor.html

Color.Lerp

public static Color **Lerp**(Color **a**, Color **b**, float **t**);

Parameters

a	Color a
b	Color b
t	Float for combining a and b

Description

Linearly interpolates between colors a and b by t.

t is clamped between 0 and 1. When t is 0 returns a. When t is 1 returns b.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public Color lerpedColor = Color.white;
    void Update() {
        lerpedColor = Color.Lerp(Color.white, Color.black, Mathf.PingPong(Time.time, 1));
    }
}
```

http://docs.unity3d.com/ScriptReference/Color.Lerp.html