유니티 스터디

2018 – 05 – 25

CONTENTS

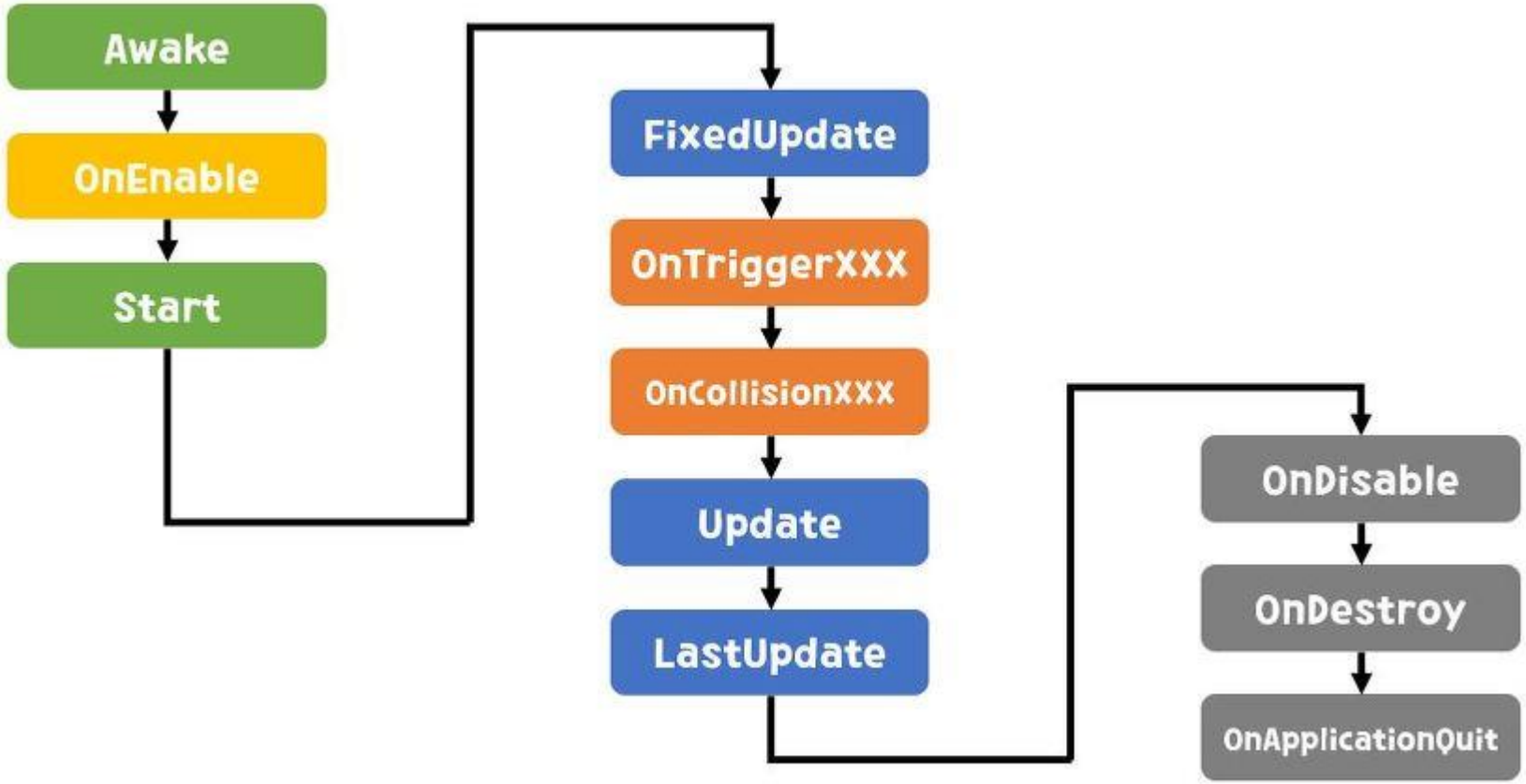
01

Flow

01. Flow

알아둬야 할 것들

Unity Monobehaviour Flow



01. Flow

알아둬야 할 것들

Unity Monobehaviour Flow

Update() - 스크립트가 enabled 상태일때, 매 프레임마다 호출됩니다.
일반적으로 가장 빈번하게 사용되는 함수이며,
물리 효과가 적용되지 않은 오브젝트의 움직임이나 단순한 타이머,
키 입력을 받을 때 사용됩니다.

FixedUpdate() - 프레임을 기반으로 호출되는 Update 와
달리 Fixed Timestep에 설정된 값에 따라 일정한 간격으로 호출됩니다.
물리 효과가 적용된(Rigidbody) 오브젝트를 조정할 때 사용됩니다

(Update는 불규칙한 호출임으로 물리엔진 충돌검사 등이 제대로 안될 수 있음).

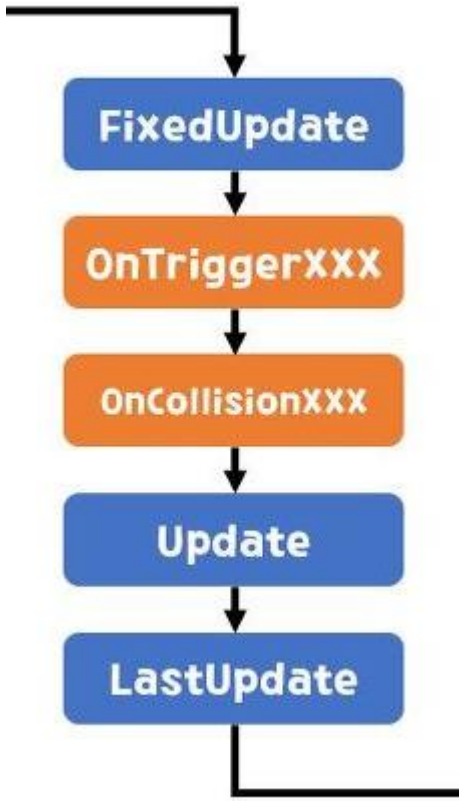
LateUpdate() - 모든 Update 함수가 호출된 후, 마지막으로 호출됩니다.
주로 오브젝트를 따라가게 설정한 카메라는 LateUpdate 를 사용합니다

(카메라가 따라가는 오브젝트가 Update함수 안에서 움직일 경우가 있기 때문).

01. Flow

알아둬야 할 것들

Unity Monobehaviour Flow




```
void Update()
{
    if (Input.GetKeyDown(KeyCode.R))
    {
        GetComponent<Renderer> ().material.color = Color.red;
    }
    if (Input.GetKeyDown(KeyCode.G))
    {
        GetComponent<Renderer>().material.color = Color.green;
    }
    if (Input.GetKeyDown(KeyCode.B))
    {
        GetComponent<Renderer>().material.color = Color.blue;
    }
```

02

Review

02. Review

function

```
int myInt = 5;



void Start ()
{
    myInt = MultiplyByTwo(myInt);
    Debug.Log (myInt);
}



int MultiplyByTwo (int number)
{
    int ret;
    ret = number * 2;
    return ret;
}
```

02. Review

if & update

```
float coffeeTemperature = 85.0f;
float hotLimitTemperature = 70.0f;
float coldLimitTemperature = 40.0f;


void Update ()
{
    if(Input.GetKeyDown(KeyCode.Space))
        TemperatureTest();

    coffeeTemperature -= Time.deltaTime * 5f;
}


void TemperatureTest ()
{
    // If the coffee's temperature is greater than the hottest drinking temperature...
    if(coffeeTemperature > hotLimitTemperature)
    {
        // ... do this.
        print("Coffee is too hot.");
    }
    // If it isn't, but the coffee temperature is less than the coldest drinking temperature...
    else if(coffeeTemperature < coldLimitTemperature)
    {
        // ... do this.
        print("Coffee is too cold.");
    }
    // If it is neither of those then...
    else
    {
        // ... do this.
        print("Coffee is just right.");
    }
}
```

02. Review

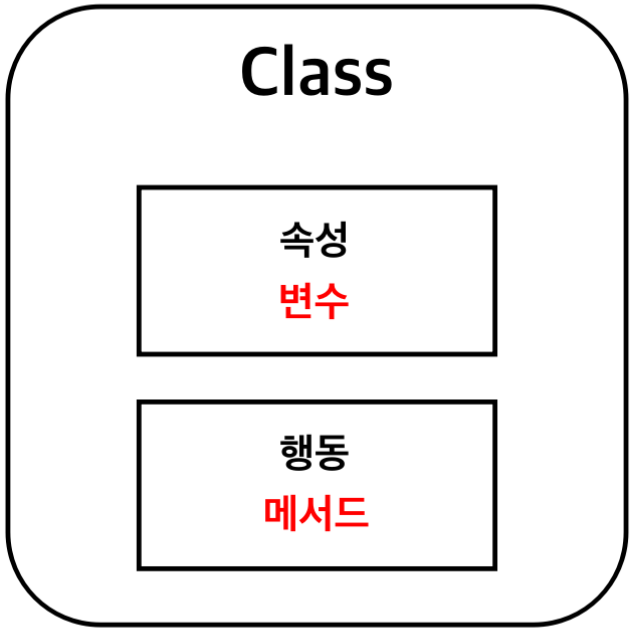
while

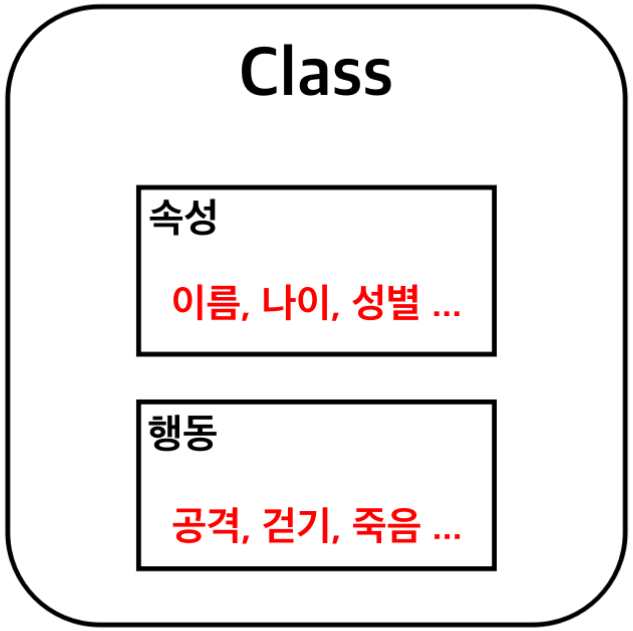
```
int cupsInTheSink = 4;



void Start ()
{
    while(cupsInTheSink > 0)
    {
        Debug.Log ("I've washed a cup!");
        cupsInTheSink--;
    }
}
```

03

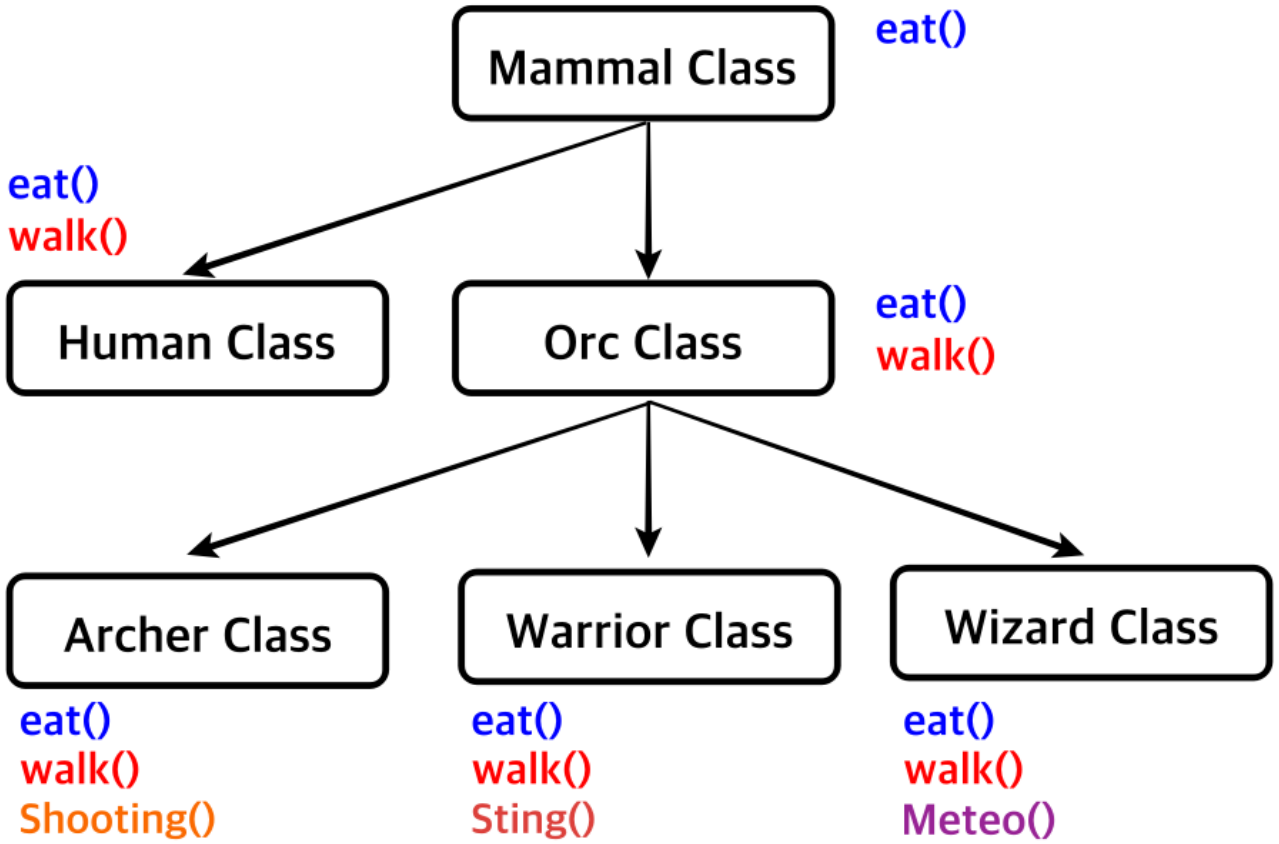
Class

03. Class





03. Class


Mammal Class
eat()
eat()
walk()
Human Class
Orc Class
eat()
walk()
Archer Class
Warrior Class
Wizard Class
eat()
walk()
Shooting()
eat()
walk()
Sting()
eat()
walk()
Meteo()

03. Class

```
public class AnotherClass
{
    public int apples;
    public int bananas;


    private int stapler;
    private int sellotape;


    public void FruitMachine (int a, int b)
    {
        int answer;
        answer = a + b;
        Debug.Log("Fruit total: " + answer);
    }


    private void OfficeSort (int a, int b)
    {
        int answer;
        answer = a + b;
        Debug.Log("Office Supplies total: " + answer);
    }
}
```

```
public class ScopeAndAccessModifiers : MonoBehaviour
{
    public int alpha = 5;


    private int beta = 0;
    private int gamma = 5;


    private AnotherClass myOtherClass;


    void Start ()
    {
        alpha = 29;

        myOtherClass = new AnotherClass();
        myOtherClass.FruitMachine(alpha, myOtherClass.apples);
    }


    void Example (int pens, int crayons)
    {
        int answer;
        answer = pens * crayons * alpha;
        Debug.Log(answer);
    }


    void Update ()
    {
        Debug.Log("Alpha is set to: " + alpha);
    }
}
```

03. Class

```
// 이 튜토리얼 코드는, Derek Benas 의 C# 교육 코드를 유니티 스크립트 교육 맞춰 간단하게 고친것임을 밝힌다.

class Animal
{

    // public : 접근 제한이 없다
    // protected : 현재 클래스와 자식 클래스에서만 접근 가능
    // private : 현재 클래스에서만 접근 가능

    // 클래스 내부의 변수를 필드라고 부른다.
    public double height;
    public double weight;
    public string sound;


    // 바깥에서 변수에 마음대로 접근하지 않도록 접근자 메소드와 설정자 메소드를 따로 만들거나
    // 프로퍼티를 사용하여 스스로 데이터를 검증하게 할 수 있다
    private string name;
    // 이것이 프로퍼티. 필드처럼 동작하지만, 내부에 따로 값을 반환할 때와 받아들일 때의 처리를 만들 수 있다.
    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    // 모든 오브젝트들은 입력을 받지 않는 기본 생성자를 가지고 있다
    // 생성자는 오브젝트가 생성될때 마다
    // this 키워드는 현재 자신의 오브젝트의 필드 값들을 가져올때 사용된다. 왜냐면 오브젝트에 특정한 이름이 없어 스스로를 지칭할 이름이 필요하기 때문.

    // 기본 생성자는, 다른 생성자들을 따로 만들어줄 경우 자동으로 만들어지지 않는다.
    public Animal()
    {
        this.height = 0;
        this.weight = 0;
        this.name = "No Name";
        this.sound = "No Sound";

        numOfAnimals++;
    }
```

03. Class

```
// 커스텀 생성자도 만들어 줄수 있다
public Animal(double height, double weight, string name, string sound)
{
    this.height = height;
    this.weight = weight;
    this.name = name;
    this.sound = sound;

    numOfAnimals++;
}

// static 필드는 Animal 클래스의 모든 오브젝트들이 공유한다
// static 은 클래스와 관련은 있으나, 개별 오브젝트가 가지고 있기에는 어색한 기능이나 값에 사용한다. 각각의 동물들이 자기 외의 전체 동물들의 수를
static int numOfAnimals = 0;

// static 메소드는 static 이 아닌 멤버에 접근 할수 없다
public static int getNumOfAnimals()
{
    return numOfAnimals;
}

// 메소드를 선언
public string Explain()
{
    return String.Format("{0} 은 {1} 인치 높이고, 무게는 {2} 파운드 이며, 울음소리가 {3} 이다.", name, height, weight, sound);
}
```

04

Report

04. Report

Thank you

질문