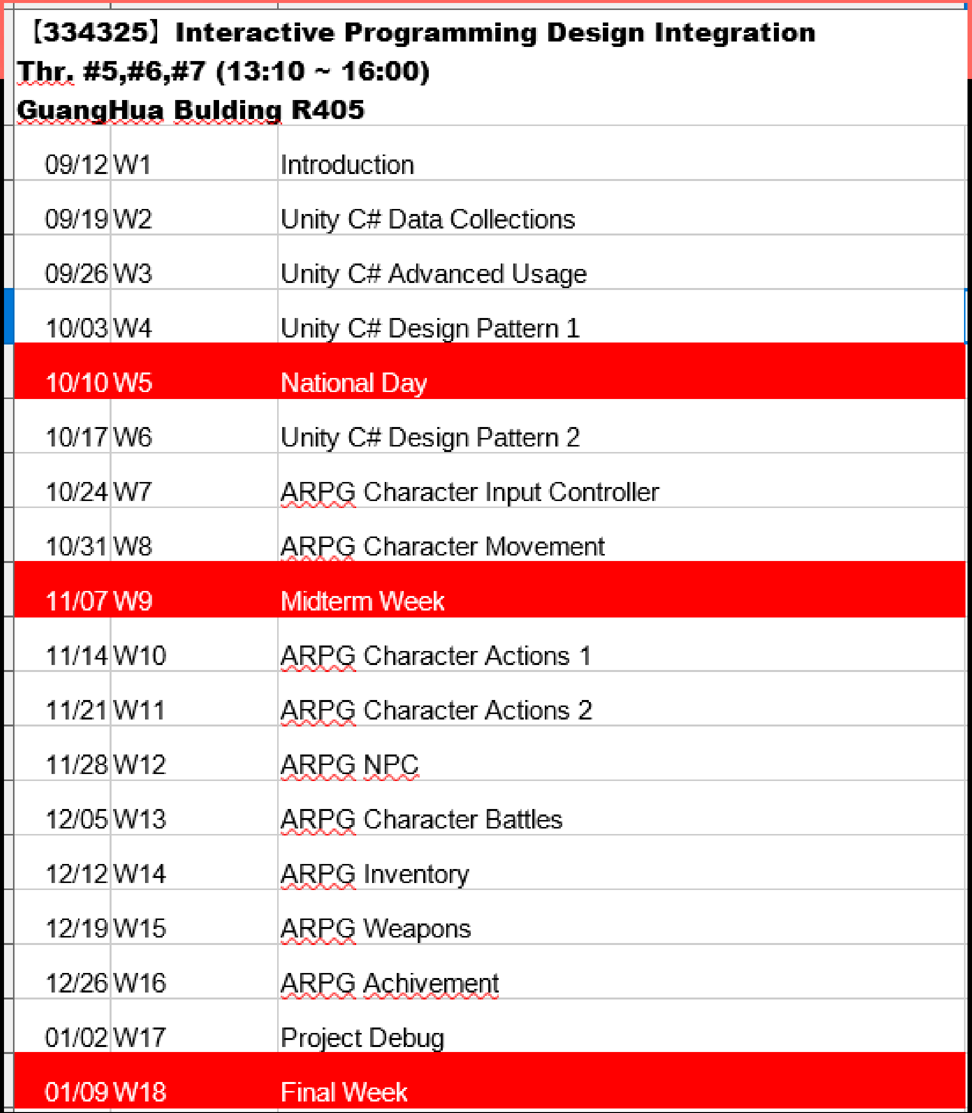
互動程式設計III

Interactive Programming Design Integration

Syllabus

- Participation: 10%
- Mid-term project: 40%
- Final project: 50%
- You can use ChatGPT or other LLM to write all the codes in this class.
    - But.. luckily thay are not familiar with Unity3D programming! 😂



[334325] Interactive Programming Design Integration
Thr. #5,#6,#7 (13:10 ~ 16:00)
GuangHua Bulding R405

09/12	W1	Introduction
09/19	W2	Unity C# Data Collections
09/26	W3	Unity C# Advanced Usage
10/03	W4	Unity C# Design Pattern 1
10/10	W5	National Day
10/17	W6	Unity C# Design Pattern 2
10/24	W7	ARPG Character Input Controller
10/31	W8	ARPG Character Movement
11/07	W9	Midterm Week
11/14	W10	ARPG Character Actions 1
11/21	W11	ARPG Character Actions 2
11/28	W12	ARPG NPC
12/05	W13	ARPG Character Battles
12/12	W14	ARPG Inventory
12/19	W15	ARPG Weapons
12/26	W16	ARPG Achivement
01/02	W17	Project Debug
01/09	W18	Final Week

- Data Collections
  - Enumeration
  - List
  - Dictionary
  - Class Object
  - JSON Object

互動程式設計III

Interactive Programming Design Integration

Introduction to Data Collections

- In this lesson you'll learn:
  - A comprehensive overview of various data collection types and their usage in C#.
  - Understanding how to manage and manipulate collections effectively.
- Learning Objectives
  - Gain a solid understanding of different data collection types in C#.
  - Learn how to efficiently manage and manipulate data collections.
  - Understand the practical applications and performance considerations of each collection type.
- Why This Chapter is Important
  - Data Management:
    - Effective data management is crucial in software development.
    - Collections provide powerful tools to handle data efficiently and intuitively.
  - Performance:
    - Choosing the right collection type can significantly impact the performance and scalability of your applications.
  - Real-World Applications:
    - Collections are used extensively in various domains, including web development, data processing, and application development.

- Data Collections
  - Enumeration
  - List
  - Dictionary
  - Class Object
  - JSON Object

互動程式設計III

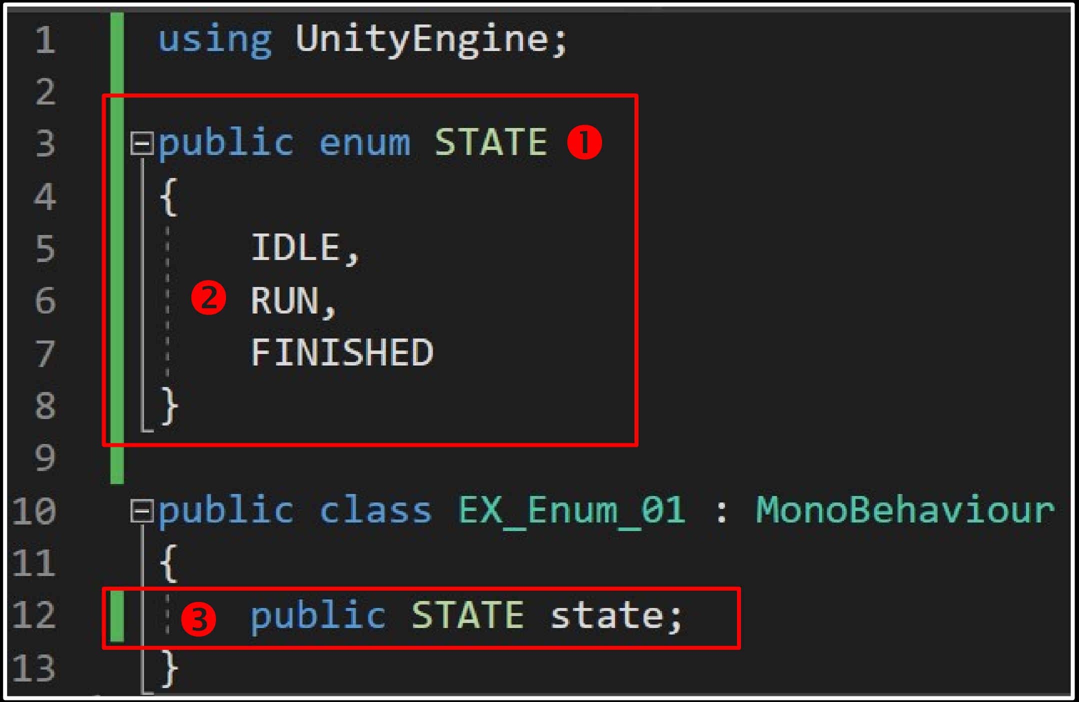
Interactive Programming Design Integration

What is Enumeration

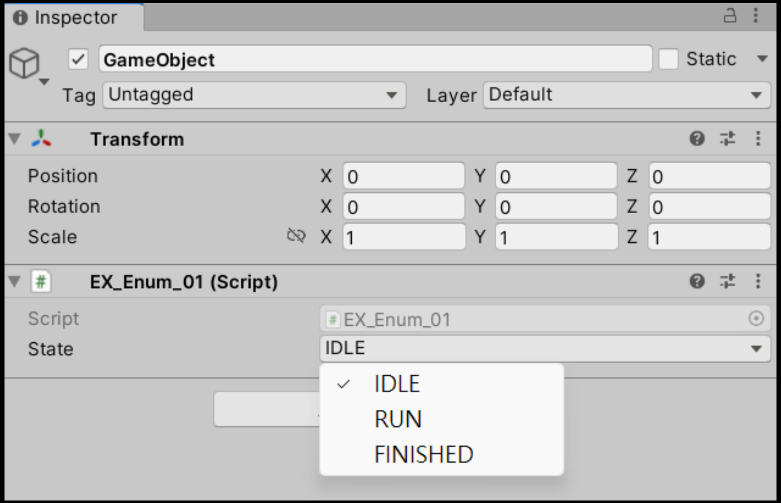
- Definition and Usage
  - Definition: An enumeration type (or enum type) is a value type defined by a set of named constants of the underlying integral numeric type.
  - Usage: Enums are used to represent a collection of related constants, such as states, categories, or modes, that can be assigned to variables for improved code readability and maintainability.
  - Example: Enums are commonly used in game development to define game states, player actions, or item types.
- Benefits of Using Enums
  - Improved Code Readability: Enums provide meaningful names to integral constants, making the code easier to understand and maintain.
  - Type Safety: Enums restrict variables to have one of the predefined values, reducing bugs caused by invalid values.
  - Code Organization: Enums group related constants together, improving the organization and structure of the code.
  - IntelliSense Support: Enums enhance the development experience by providing IntelliSense support in IDEs, helping developers to avoid typos and discover available options.

EX: C# Enum

- Define an enum type❶ with enum members❷.
- Declare an enum variable.❸



```
using UnityEngine;

public enum STATE
{
    IDLE,
    RUN,
    FINISHED
}

public class EX_Enum_01 : MonoBehaviour
{
    public STATE state;
}
```



☑ GameObject □ Static

Tag Untagged Layer Default

Transform

Position X 0 Y 0 Z 0

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1

EX_Enum_01 (Script)

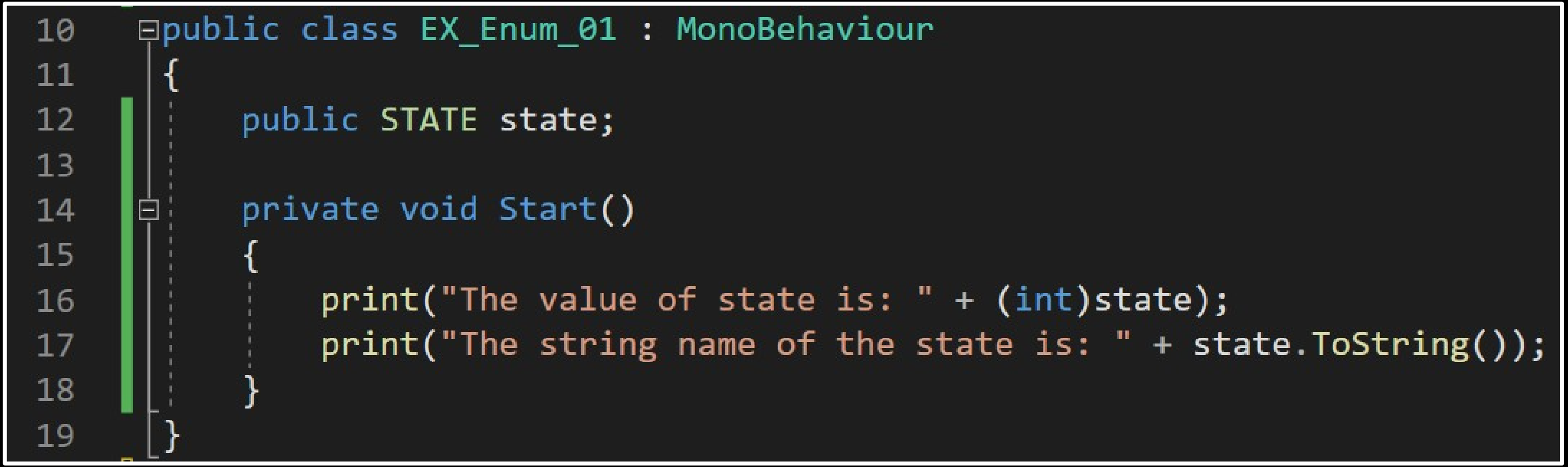
Script EX_Enum_01

State IDLE

✓ IDLE
RUN
FINISHED

EX: Read and Print the Value of the Enum

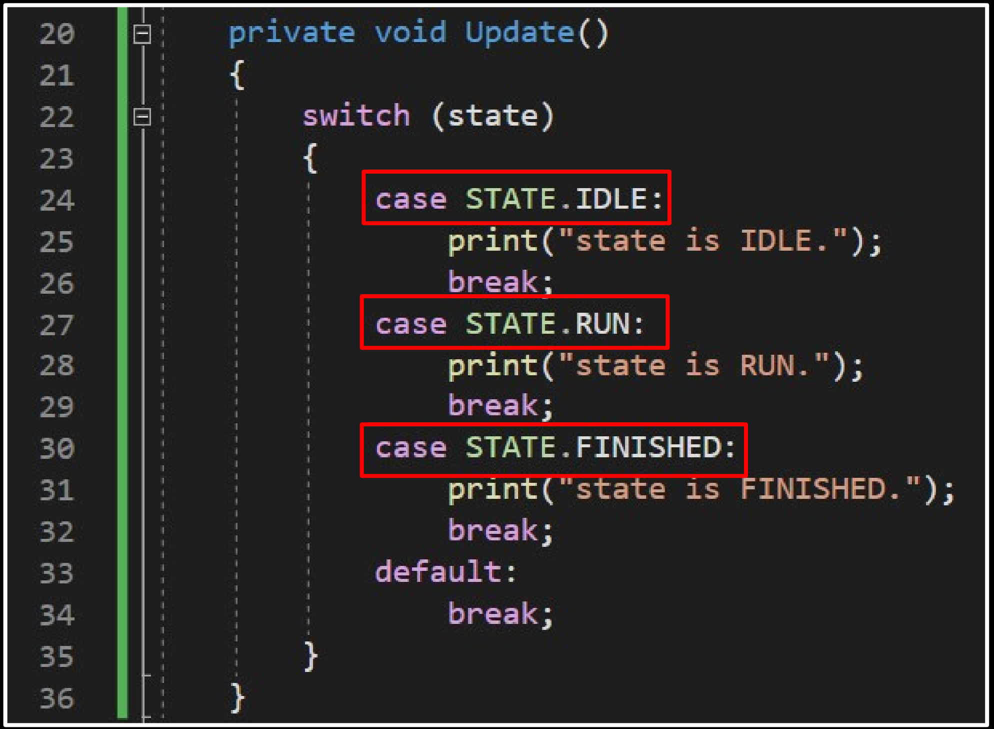
- Show the integer value of the enum variable.❶
- Show the string name of the enum variable.❷



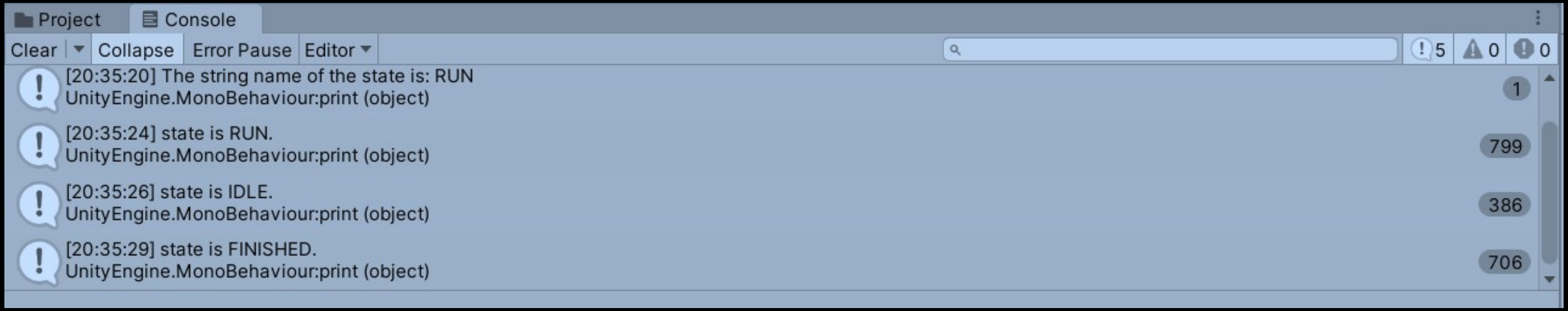
```
public class EX_Enum_01 : MonoBehaviour
{
    public STATE state;

    private void Start()
    {
❶       print("The value of state is: " + (int)state);
❷       print("The string name of the state is: " + state.ToString());
    }
}
```

EX: Check Enum Value in a Switch

- You don't need to memorize each of the integer value for an enum member.



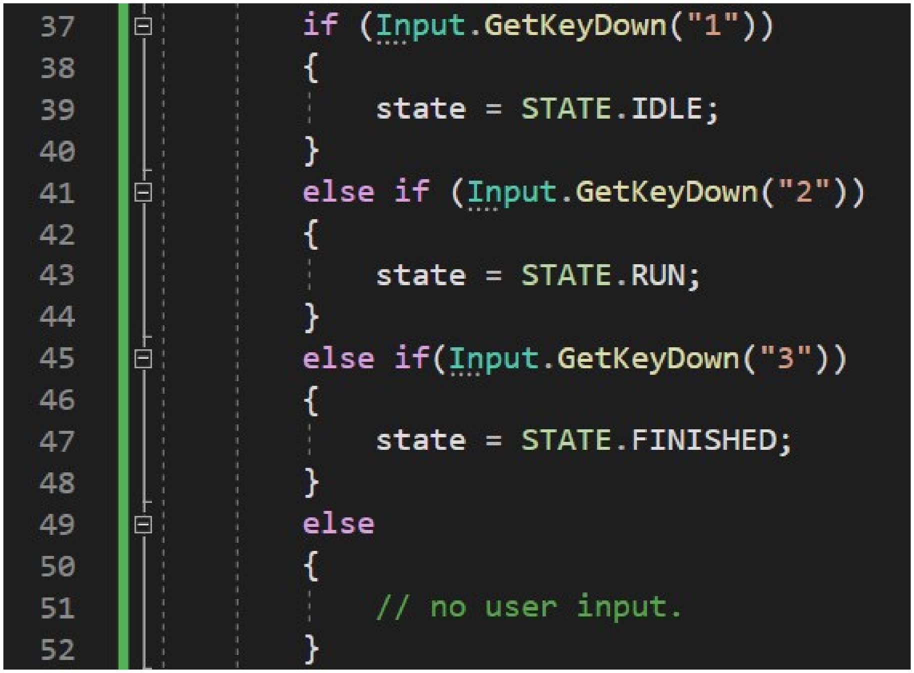
```
    private void Update()
    {
        switch (state)
        {
            case STATE.IDLE:
                print("state is IDLE.");
                break;
            case STATE.RUN:
                print("state is RUN.");
                break;
            case STATE.FINISHED:
                print("state is FINISHED.");
                break;
            default:
                break;
        }
    }
```



```
[20:35:20] The string name of the state is: RUN          1
UnityEngine.MonoBehaviour:print (object)
[20:35:24] state is RUN.                               799
UnityEngine.MonoBehaviour:print (object)
[20:35:26] state is IDLE.                              386
UnityEngine.MonoBehaviour:print (object)
[20:35:29] state is FINISHED.                          706
UnityEngine.MonoBehaviour:print (object)
```

EX: Write Enum Value

- You can assign an enum member(like STATE.IDLE) to an enum variable.



```
if (Input.GetKeyDown("1"))
{
    state = STATE.IDLE;
}
else if (Input.GetKeyDown("2"))
{
    state = STATE.RUN;
}
else if(Input.GetKeyDown("3"))
{
    state = STATE.FINISHED;
}
else
{
    // no user input.
}
```

互動程式設計III

Interactive Programming Design Integration

- Data Collections
  - Enumeration
  - List
  - Dictionary
  - Class Object
  - JSON Object

What is List

- Usage and Performance Considerations
  - Usage: Lists are dynamic collections that allow you to store elements of the same type. They are useful when the number of elements is not fixed and you need the ability to add, remove, or modify elements.
  - Performance Considerations:
    - Dynamic Resizing: Lists automatically resize themselves as elements are added or removed. This resizing can be costly in terms of performance, especially if it happens frequently.
    - Memory Allocation: Lists allocate memory in chunks to reduce the frequency of resizing, but this can lead to underutilized memory.
  - Iteration: Iterating over a List is generally fast and comparable to arrays.

What is List

- Methods and Properties of List
  - Properties:
    - Count: Gets the number of elements in the list.
    - Capacity: Gets or sets the number of elements the list can hold before resizing is required.
    - Methods:
      - Add(T item): Adds an element to the end of the list.
      - Remove(T item): Removes the first occurrence of a specific object from the list.
      - RemoveAt(int index): Removes the element at the specified index.
      - Insert(int index, T item): Inserts an element into the list at the specified index.
      - Contains(T item): Determines whether the list contains a specific element.
      - Clear(): Removes all elements from the list.
      - Find(Predicate<T> match): Searches for an element that matches the conditions defined by the specified predicate and returns the first occurrence.
      - Sort(): Sorts the elements in the list.

The Difference Between List and Array in C#

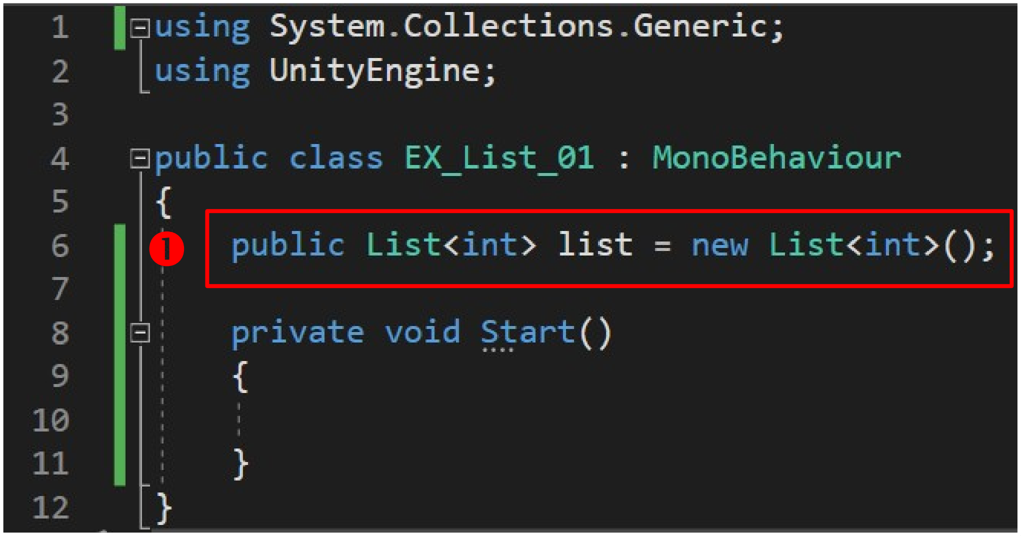
- Array:
  - Fixed size: Once created, the size of an array cannot be changed.
  - Performance: Accessing elements in an array is slightly faster due to fixed size and contiguous memory allocation.
  - Use case: Suitable for scenarios where the number of elements is known and fixed.

  -
    ```
    int[] numbers = new int[3] { 1, 2, 3 };
    ```
- List:
  - Dynamic size: Lists can grow or shrink dynamically as elements are added or removed.
  - Performance: Slightly slower than arrays for element access due to dynamic nature, but provides flexibility.
  - Use case: Suitable for scenarios where the number of elements can change.

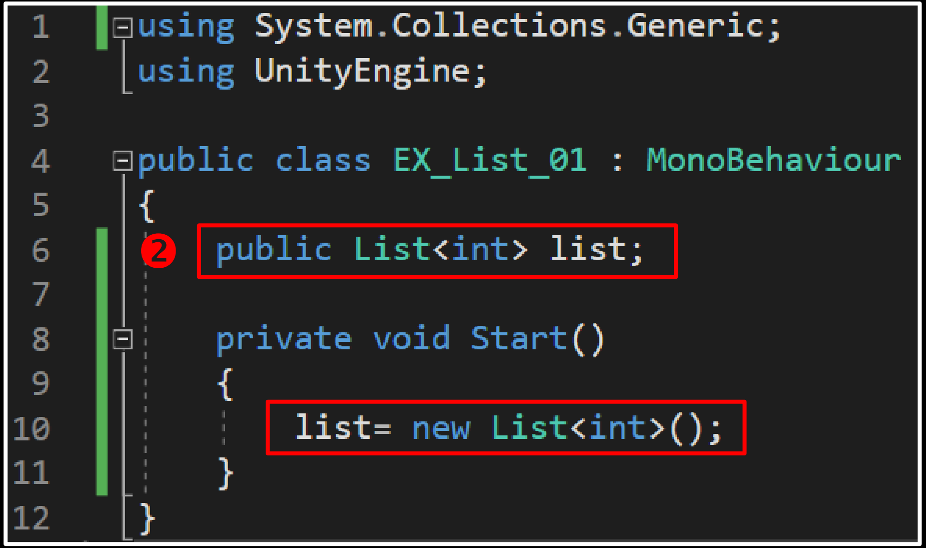
  -
    ```
    List<int> numbers = new List<int> { 1, 2, 3 };
    ```

EX: Initialize a List

- Initialize the list in declaration. ❶
- Declare first, then initialize it in definition. ❷



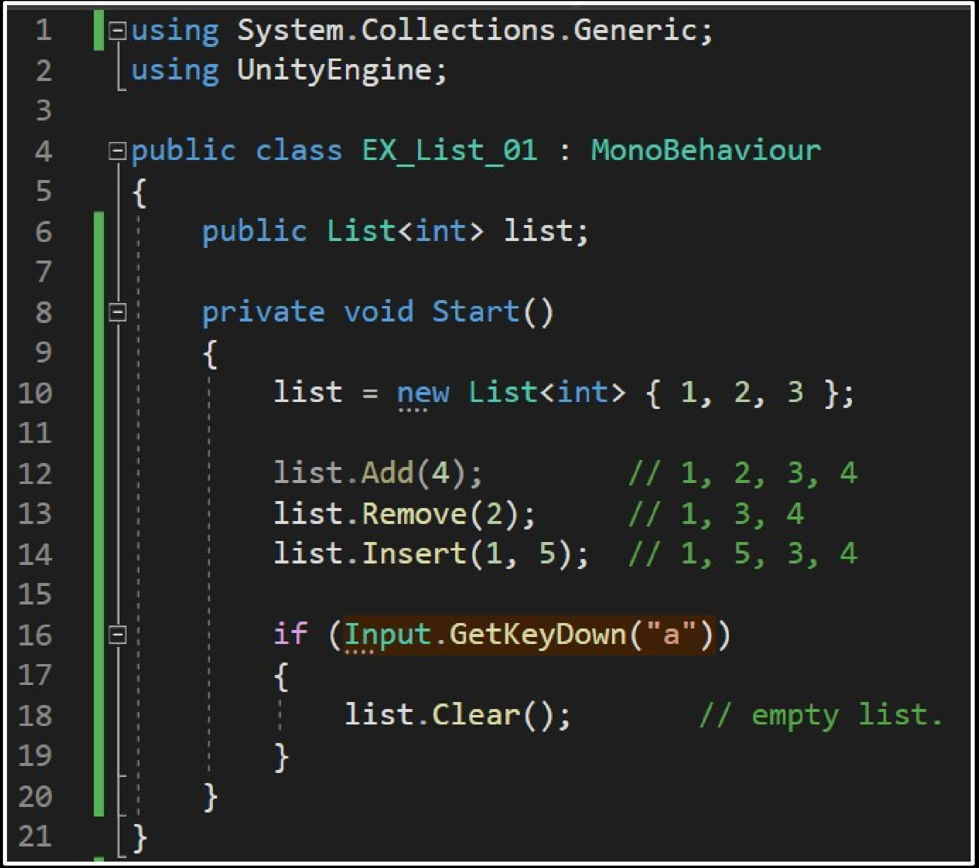
```
using System.Collections.Generic;
using UnityEngine;

public class EX_List_01 : MonoBehaviour
{
    public List<int> list = new List<int>();

    private void Start()
    {

    }
}
```



```
using System.Collections.Generic;
using UnityEngine;

public class EX_List_01 : MonoBehaviour
{
    public List<int> list;

    private void Start()
    {
        list= new List<int>();
    }
}
```

EX: Add, Remove, Insert and Clear a List

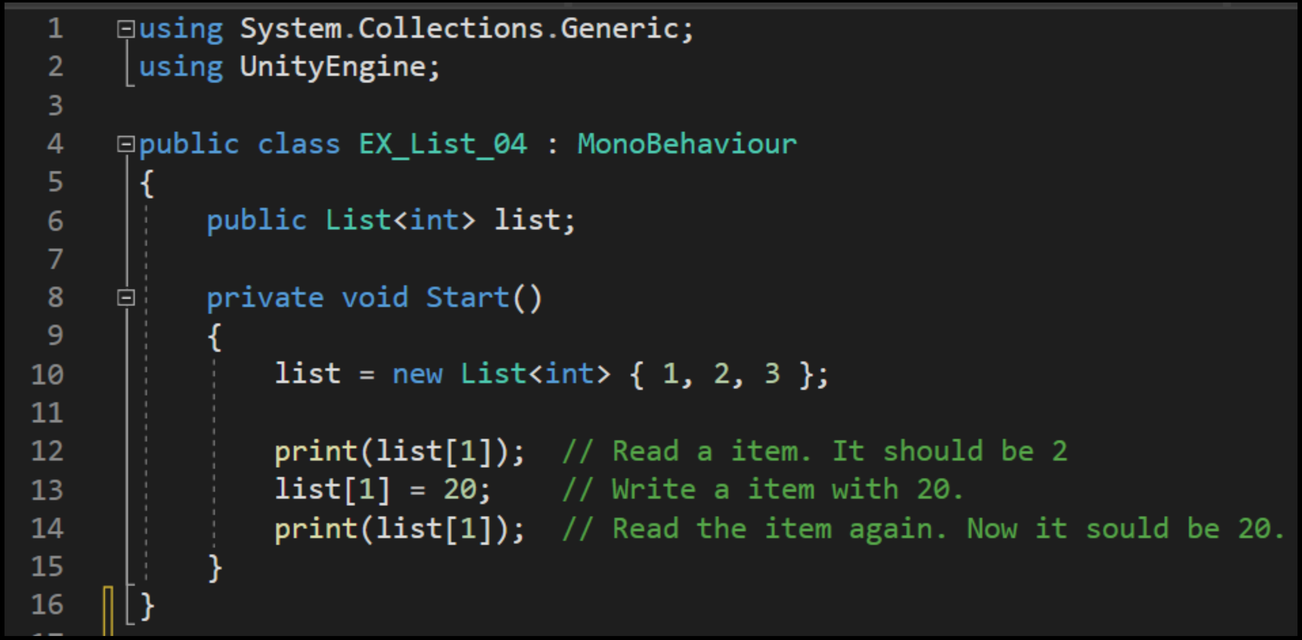
- Add(T item): Adds an element to the end of the list.
- Remove(T item): Removes the first occurrence of a specific object from the list.
- RemoveAt(int index): Removes the element at the specified index.
- Insert(int index, T item): Inserts an element into the list at the specified index.
- Clear(): Removes all elements from the list.



```
using System.Collections.Generic;
using UnityEngine;

public class EX_List_01 : MonoBehaviour
{
    public List<int> list;

    private void Start()
    {
        list = new List<int> { 1, 2, 3 };

        list.Add(4);         // 1, 2, 3, 4
        list.Remove(2);      // 1, 3, 4
        list.Insert(1, 5);   // 1, 5, 3, 4

        if (Input.GetKeyDown("a"))
        {
            list.Clear();        // empty list.
        }
    }
}
```

EX: Read, Write a List Item

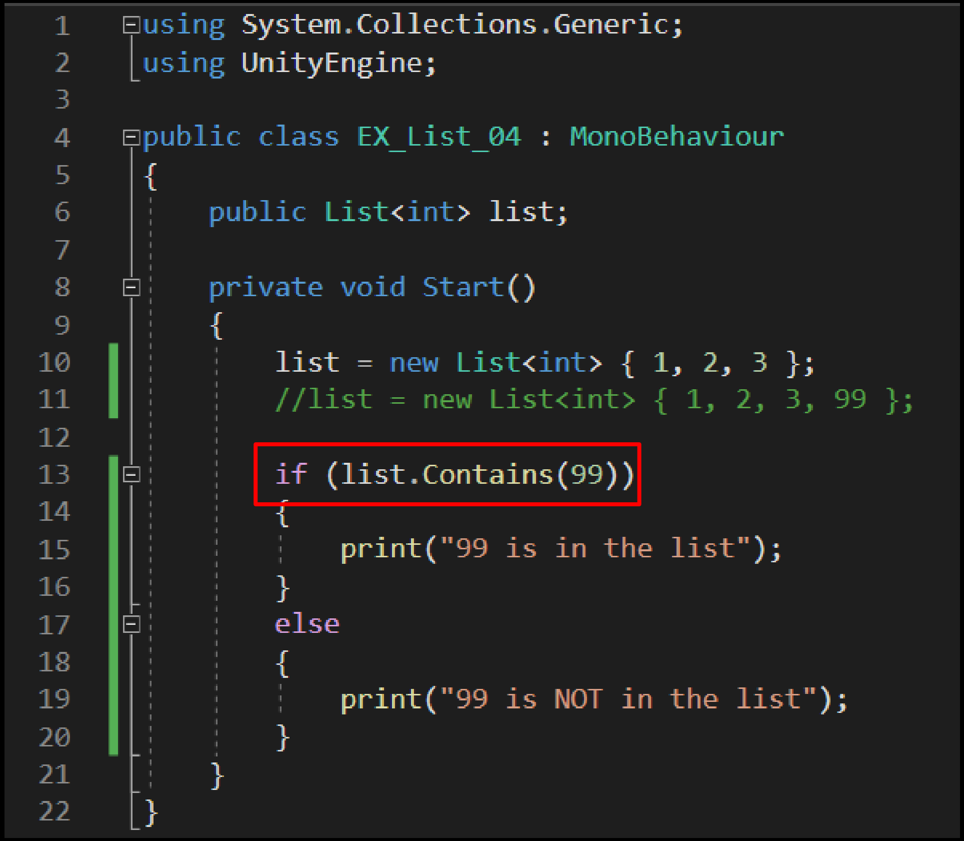
- You can access an item in a list with respect to index.
  - List is 0 index: the initial index number is 0 (NOT 1).



```
using System.Collections.Generic;
using UnityEngine;

public class EX_List_04 : MonoBehaviour
{
    public List<int> list;

    private void Start()
    {
        list = new List<int> { 1, 2, 3 };

        print(list[1]);  // Read a item. It should be 2
        list[1] = 20;    // Write a item with 20.
        print(list[1]);  // Read the item again. Now it sould be 20.
    }
}
```

EX: Contains

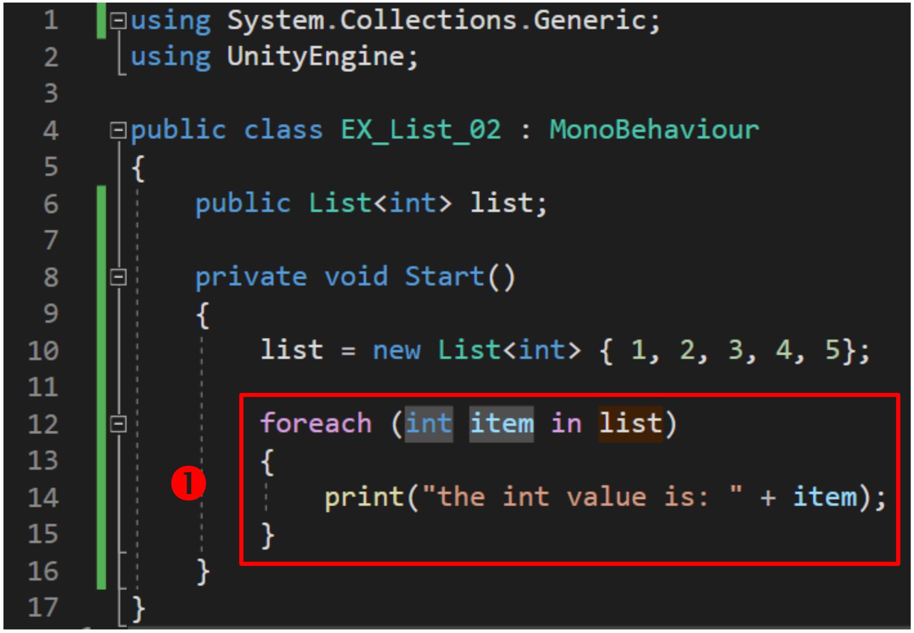
- Use Contain() to check whether target value is in the list.



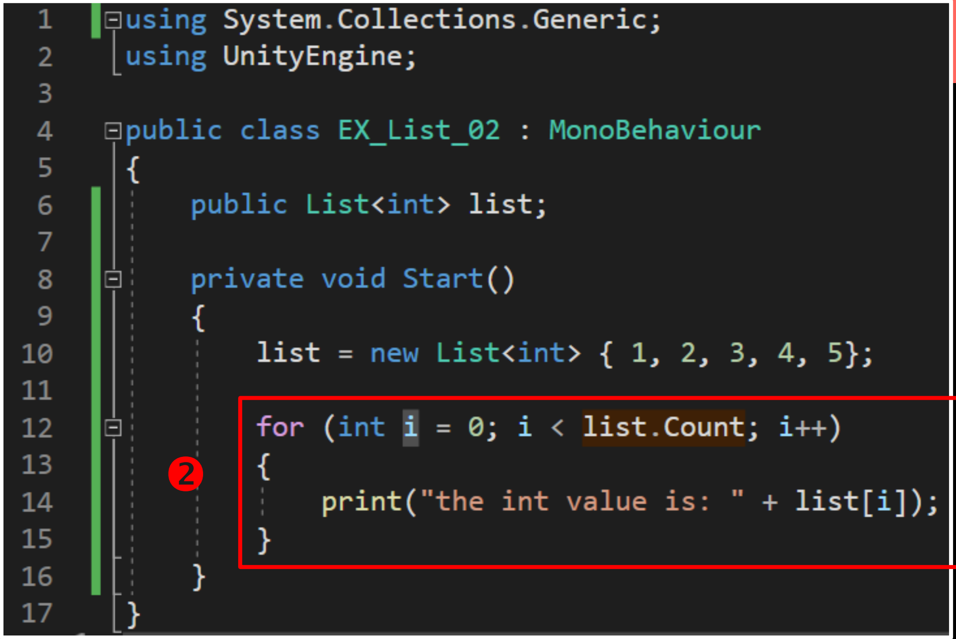
```
using System.Collections.Generic;
using UnityEngine;

public class EX_List_04 : MonoBehaviour
{
    public List<int> list;

    private void Start()
    {
        list = new List<int> { 1, 2, 3 };
        //list = new List<int> { 1, 2, 3, 99 };

        if (list.Contains(99))
        {
            print("99 is in the list");
        }
        else
        {
            print("99 is NOT in the list");
        }
    }
}
```

EX: Iterate a List

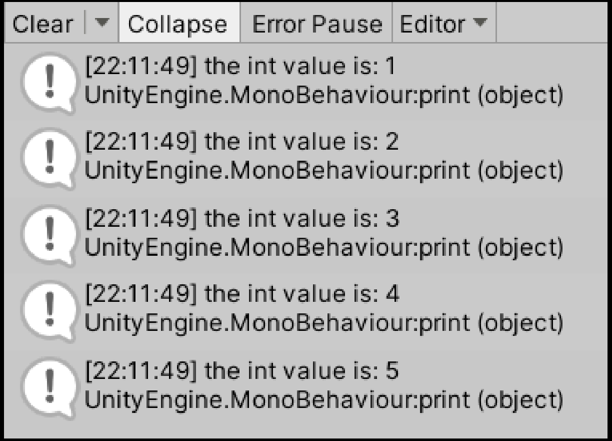
- Use foreach loop to iterate a list. ❶
- Use for loop to iterate a list. ❷



```
using System.Collections.Generic;
using UnityEngine;

public class EX_List_02 : MonoBehaviour
{
    public List<int> list;

    private void Start()
    {
        list = new List<int> { 1, 2, 3, 4, 5};

        foreach (int item in list)
        {
            print("the int value is: " + item);
        }
    }
}
```



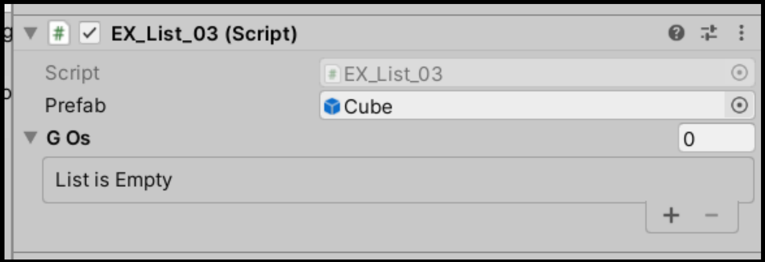
```
using System.Collections.Generic;
using UnityEngine;

public class EX_List_02 : MonoBehaviour
{
    public List<int> list;

    private void Start()
    {
        list = new List<int> { 1, 2, 3, 4, 5};

        for (int i = 0; i < list.Count; i++)
        {
            print("the int value is: " + list[i]);
        }
    }
}
```



```
[22:11:49] the int value is: 1
UnityEngine.MonoBehaviour:print (object)
[22:11:49] the int value is: 2
UnityEngine.MonoBehaviour:print (object)
[22:11:49] the int value is: 3
UnityEngine.MonoBehaviour:print (object)
[22:11:49] the int value is: 4
UnityEngine.MonoBehaviour:print (object)
[22:11:49] the int value is: 5
UnityEngine.MonoBehaviour:print (object)
```

EX: A List of GameObjects

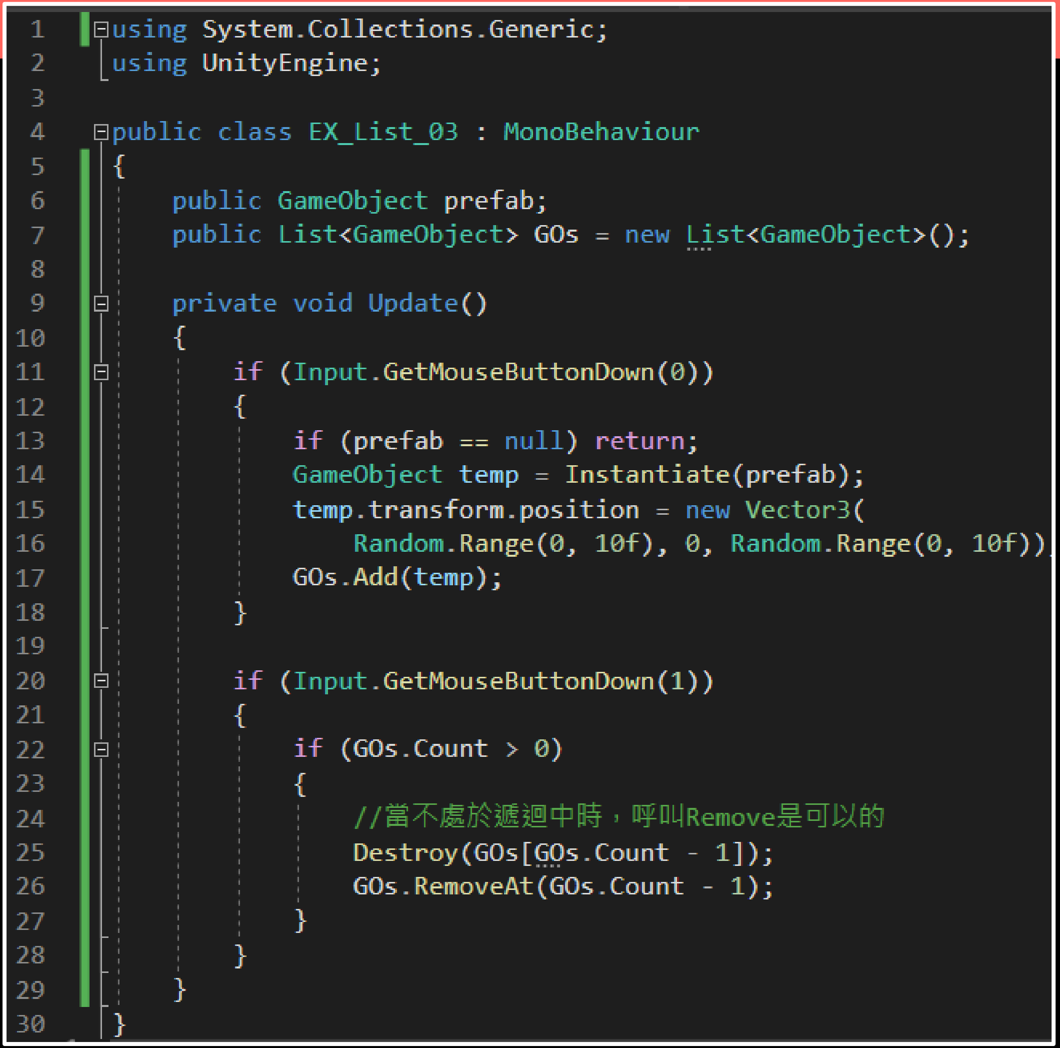
- Assign a prefab for cloning.
- Press 0 to add a new clone to the list.
- Press 1 to delete the last clone in the list.



EX_List_03 (Script)

Script	EX_List_03
Prefab	Cube
G Os	0

List is Empty



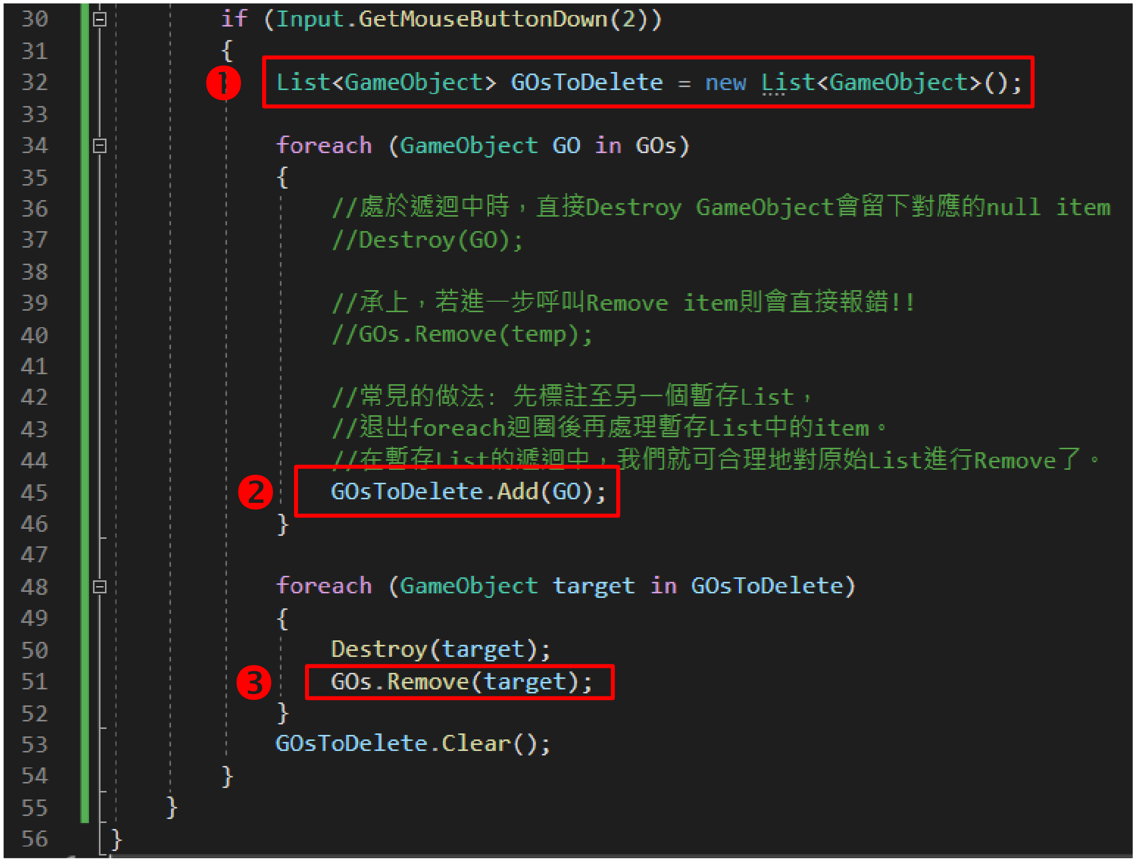
```csharp
using System.Collections.Generic;
using UnityEngine;

public class EX_List_03 : MonoBehaviour
{
    public GameObject prefab;
    public List<GameObject> GOs = new List<GameObject>();

    private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            if (prefab == null) return;
            GameObject temp = Instantiate(prefab);
            temp.transform.position = new Vector3(
                Random.Range(0, 10f), 0, Random.Range(0, 10f))
            GOs.Add(temp);
        }

        if (Input.GetMouseButtonDown(1))
        {
            if (GOs.Count > 0)
            {
                //當不處於遞迴中時，呼叫Remove是可以的
                Destroy(GOs[GOs.Count - 1]);
                GOs.RemoveAt(GOs.Count - 1);
            }
        }
    }
}
```

EX: Delete List Items in a Loop

- Create a 2nd list – GOsToDelete. ❶
- Copy the items to delete when iterating the 1st list GOs. ❷
- Now you can iterate the 2nd list and remove the items in 1st list. ❸

You are not allowed to remove items in ListA when you are iterating ListA.

So Let's create a ListB and then remove items in ListA while iterating ListB!



```
        if (Input.GetMouseButtonDown(2))
        {
            List<GameObject> GOsToDelete = new List<GameObject>();

            foreach (GameObject GO in GOs)
            {
                //處於遞迴中時，直接Destroy GameObject會留下對應的null item
                //Destroy(GO);

                //承上，若進一步呼叫Remove item則會直接報錯!!
                //GOs.Remove(temp);

                //常見的做法: 先標註至另一個暫存List，
                //退出foreach迴圈後再處理暫存List中的item。
                //在暫存List的遞迴中，我們就可合理地對原始List進行Remove了。
                GOsToDelete.Add(GO);
            }

            foreach (GameObject target in GOsToDelete)
            {
                Destroy(target);
                GOs.Remove(target);
            }
            GOsToDelete.Clear();
        }
    }
}
```

- Data Collections
  - Enumeration
  - List
  - Dictionary
  - Class Object
  - JSON Object

互動程式設計III

Interactive Programming Design Integration

What is Dictionary

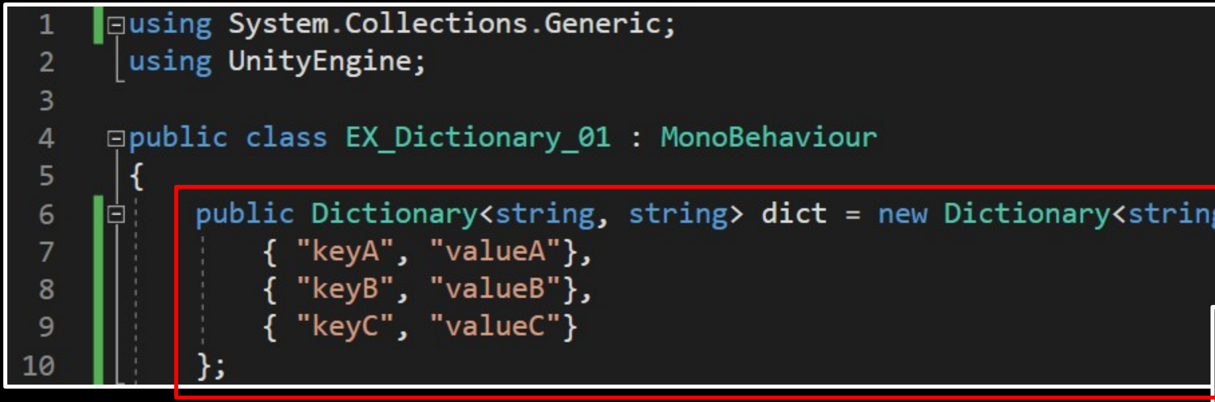
- Key-Value Pair
  - The key (usually a string) is used to uniquely identify the value. The value is the data associated with the key.
- Dictionary is a Key-Value Pair Storage
  - Definition: A Dictionary is a collection of key-value pairs where each key is unique, and each key maps to exactly one value.
  - Usage: Dictionaries are used when you need to store data that can be quickly retrieved using a unique key. They are ideal for scenarios where look-up operations are frequent.
  - Example Scenarios:
    - Storing and retrieving player scores by player name.
    - Mapping item IDs to item descriptions in an inventory system.
    - Associating configuration settings with their values.

What is Dictionary

- Performance Benefits
  - Fast Look-Up: Dictionaries provide very fast look-up times (average O(1) time complexity) for accessing values by their keys, making them highly efficient for search operations.
  - Efficient Insertion and Deletion: Adding and removing elements in a dictionary is generally efficient, with average time complexity of O(1) for insertion and deletion operations.
  - Dynamic Sizing: Similar to Lists, Dictionaries dynamically resize themselves as elements are added, although they manage their internal structure to maintain performance.
  - Use Case Considerations: Dictionaries are particularly beneficial when the primary operations are look-up, addition, and removal by key. They are not as efficient for scenarios requiring frequent enumeration of all elements or ordered operations.

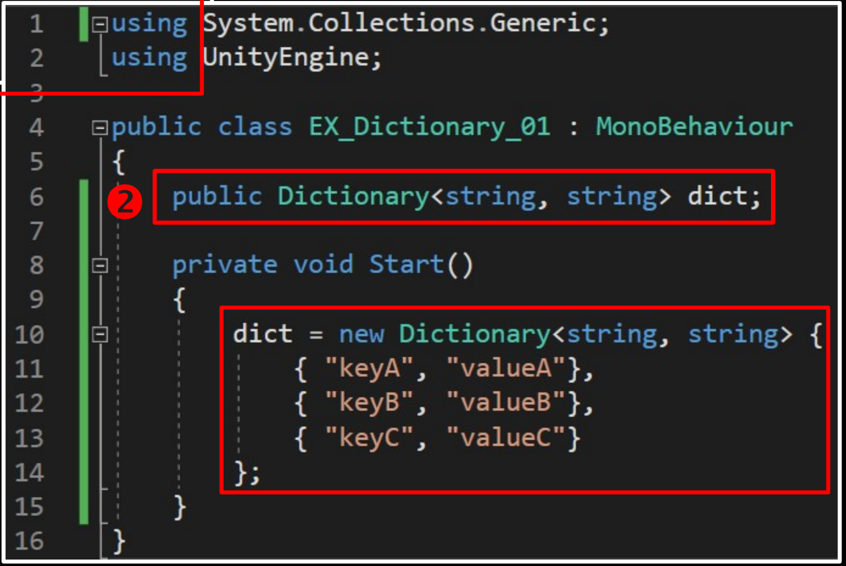
EX: Initialize a Dictionary

- Initialize the dictionary in declaration. ❶
- Declare first, then initialize it in definition. ❷



❶

```
using System.Collections.Generic;
using UnityEngine;

public class EX_Dictionary_01 : MonoBehaviour
{
    public Dictionary<string, string> dict = new Dictionary<string, string> {
        { "keyA", "valueA"},
        { "keyB", "valueB"},
        { "keyC", "valueC"}
    };
```

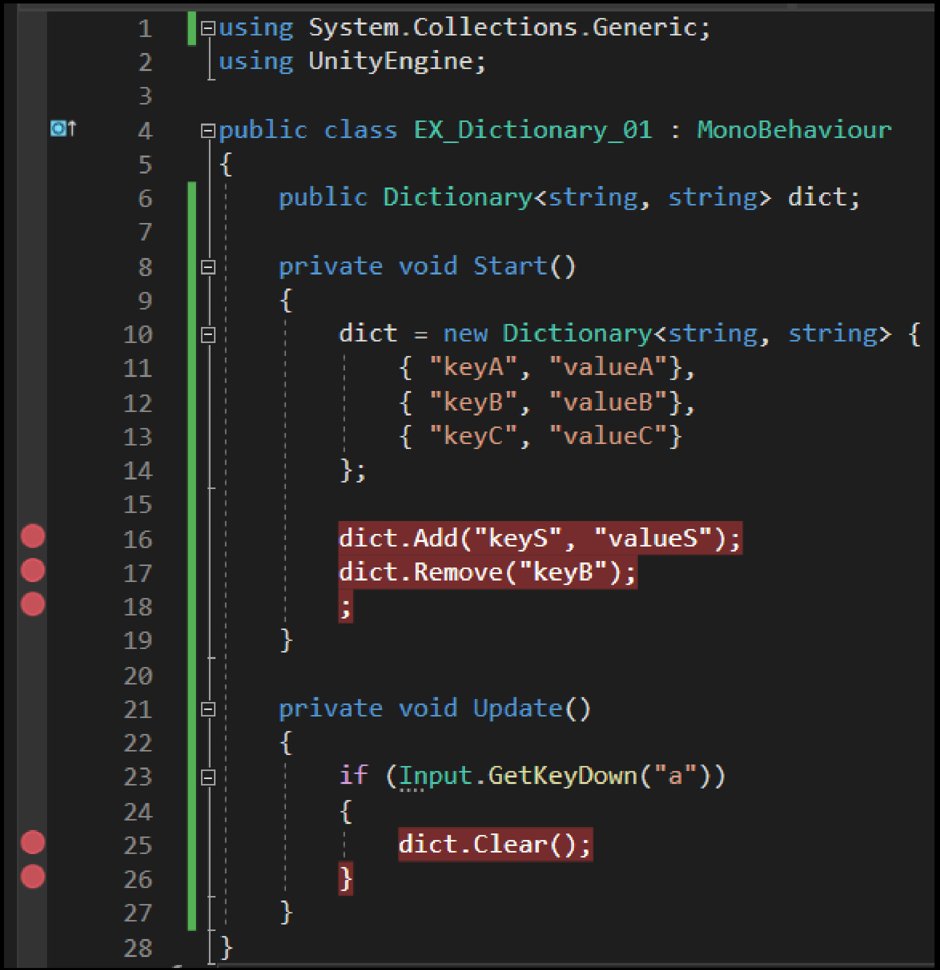


❷

```
using System.Collections.Generic;
using UnityEngine;

public class EX_Dictionary_01 : MonoBehaviour
{
    public Dictionary<string, string> dict;

    private void Start()
    {
        dict = new Dictionary<string, string> {
            { "keyA", "valueA"},
            { "keyB", "valueB"},
            { "keyC", "valueC"}
        };
    }
}
```

EX: Add and Remove an Item in Dictionary

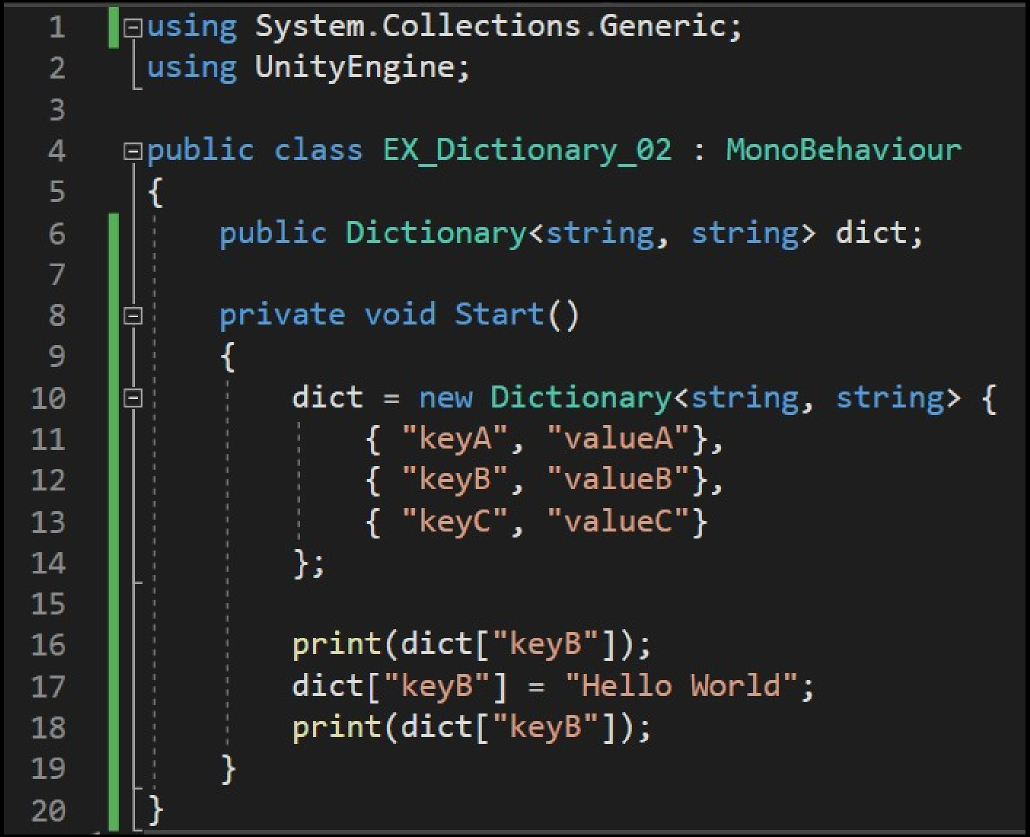
- Add(TKey key, TValue value): Adds an element to the dictionary.
- Remove(TKey key): Removes the value with the specified key from the dictionary.
- Remove(TKey key, out TValue value): Removes the value with the specified key from the dictionary, and copies the element to the value parameter.
- Clear(): Removes all element from the dictionary.
- C# Dictionary is not visible in Unity Editor!
  - So we need to add breakpoints in order to look inside the dictionary.
  - Later we will install a plugin to solve this problem.



```
using System.Collections.Generic;
using UnityEngine;

public class EX_Dictionary_01 : MonoBehaviour
{
    public Dictionary<string, string> dict;

    private void Start()
    {
        dict = new Dictionary<string, string> {
            { "keyA", "valueA"},
            { "keyB", "valueB"},
            { "keyC", "valueC"}
        };

        dict.Add("keyS", "valueS");
        dict.Remove("keyB");
        ;
    }

    private void Update()
    {
        if (Input.GetKeyDown("a"))
        {
            dict.Clear();
        }
    }
}
```

EX: Read, Write a Dictionary Item

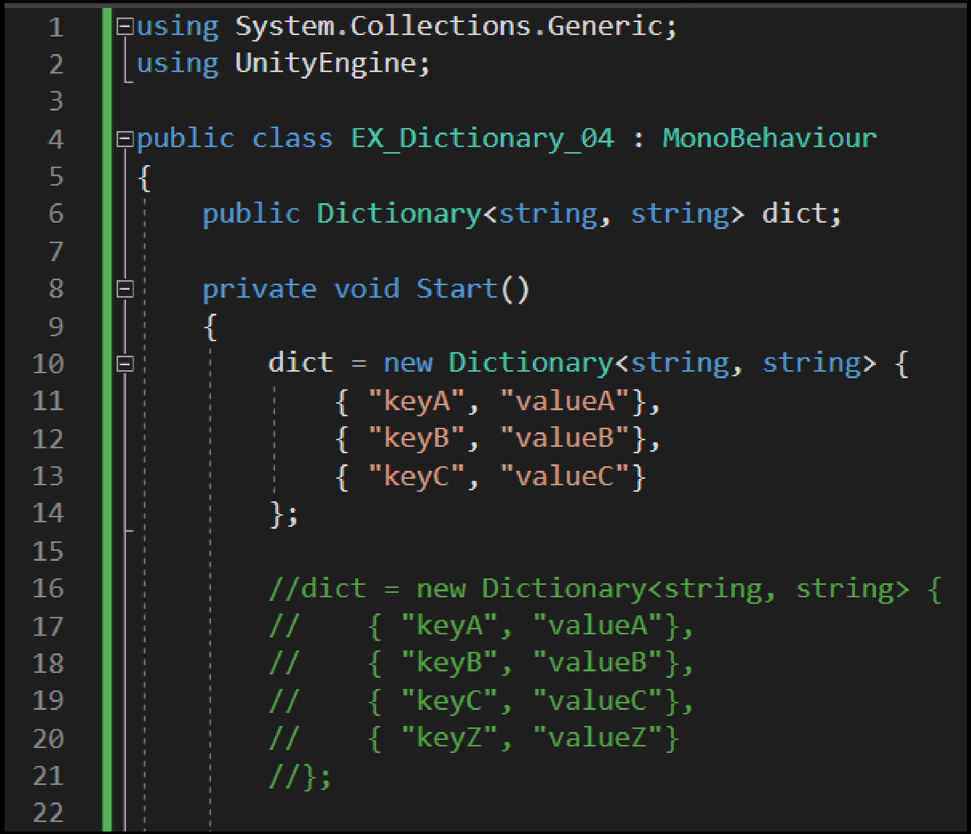
- You can access an item in a dictionary with respect to key.
  - Like dict[”keyB”]



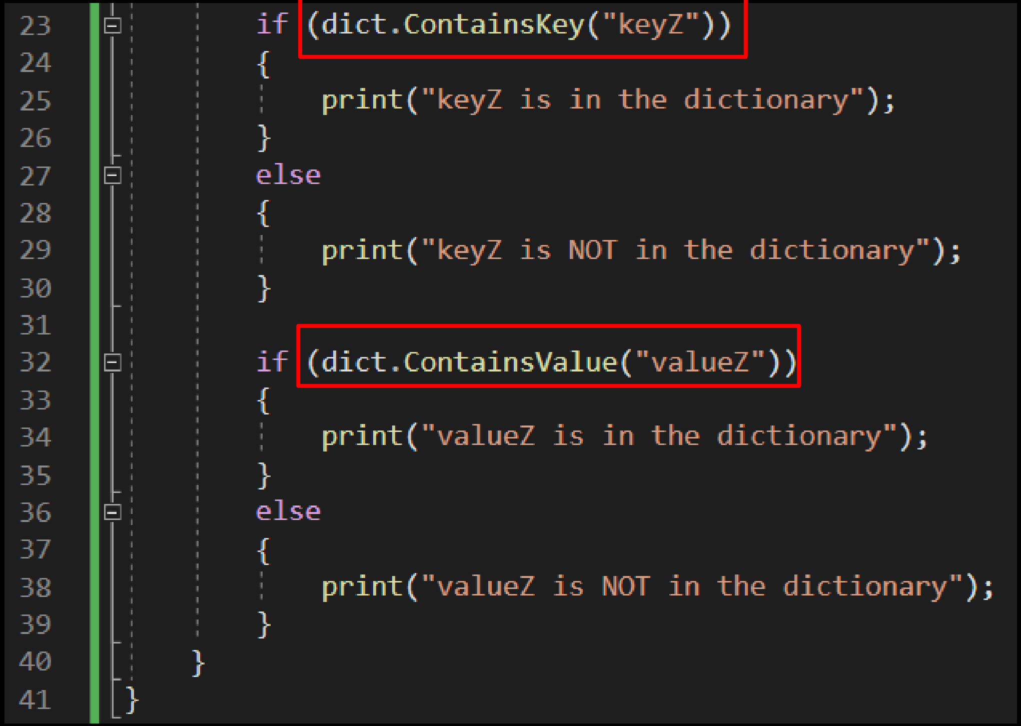
```csharp
using System.Collections.Generic;
using UnityEngine;

public class EX_Dictionary_02 : MonoBehaviour
{
    public Dictionary<string, string> dict;

    private void Start()
    {
        dict = new Dictionary<string, string> {
            { "keyA", "valueA"},
            { "keyB", "valueB"},
            { "keyC", "valueC"}
        };

        print(dict["keyB"]);
        dict["keyB"] = "Hello World";
        print(dict["keyB"]);
    }
}
```

EX: Contains

- Use ContainsKey() and ContainsValue() to check whether target value is in the dictionary.



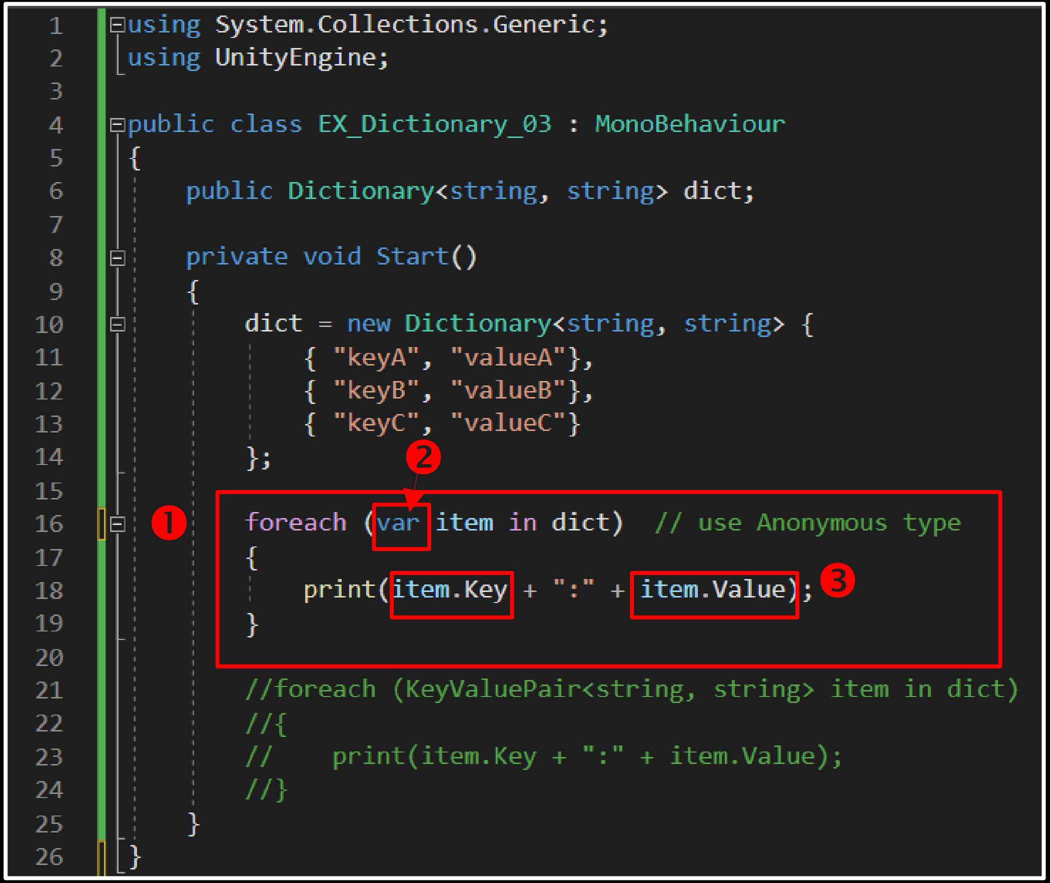
```
using System.Collections.Generic;
using UnityEngine;

public class EX_Dictionary_04 : MonoBehaviour
{
    public Dictionary<string, string> dict;

    private void Start()
    {
        dict = new Dictionary<string, string> {
            { "keyA", "valueA"},
            { "keyB", "valueB"},
            { "keyC", "valueC"}
        };

        //dict = new Dictionary<string, string> {
        //    { "keyA", "valueA"},
        //    { "keyB", "valueB"},
        //    { "keyC", "valueC"},
        //    { "keyZ", "valueZ"}
        //};

```



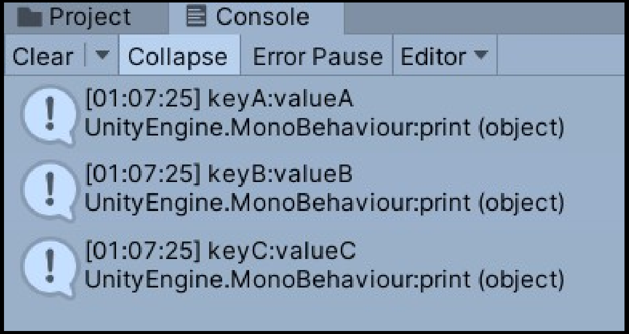
```
        if (dict.ContainsKey("keyZ"))
        {
            print("keyZ is in the dictionary");
        }
        else
        {
            print("keyZ is NOT in the dictionary");
        }

        if (dict.ContainsValue("valueZ"))
        {
            print("valueZ is in the dictionary");
        }
        else
        {
            print("valueZ is NOT in the dictionary");
        }
    }
}
```

EX: Iterate a Dictionary

- Use foreach loop to iterate a list. ❶
  - You may use anonymous type in foreach. ❷
  - You can read/write the Key and Value properties in each iteration. ❸



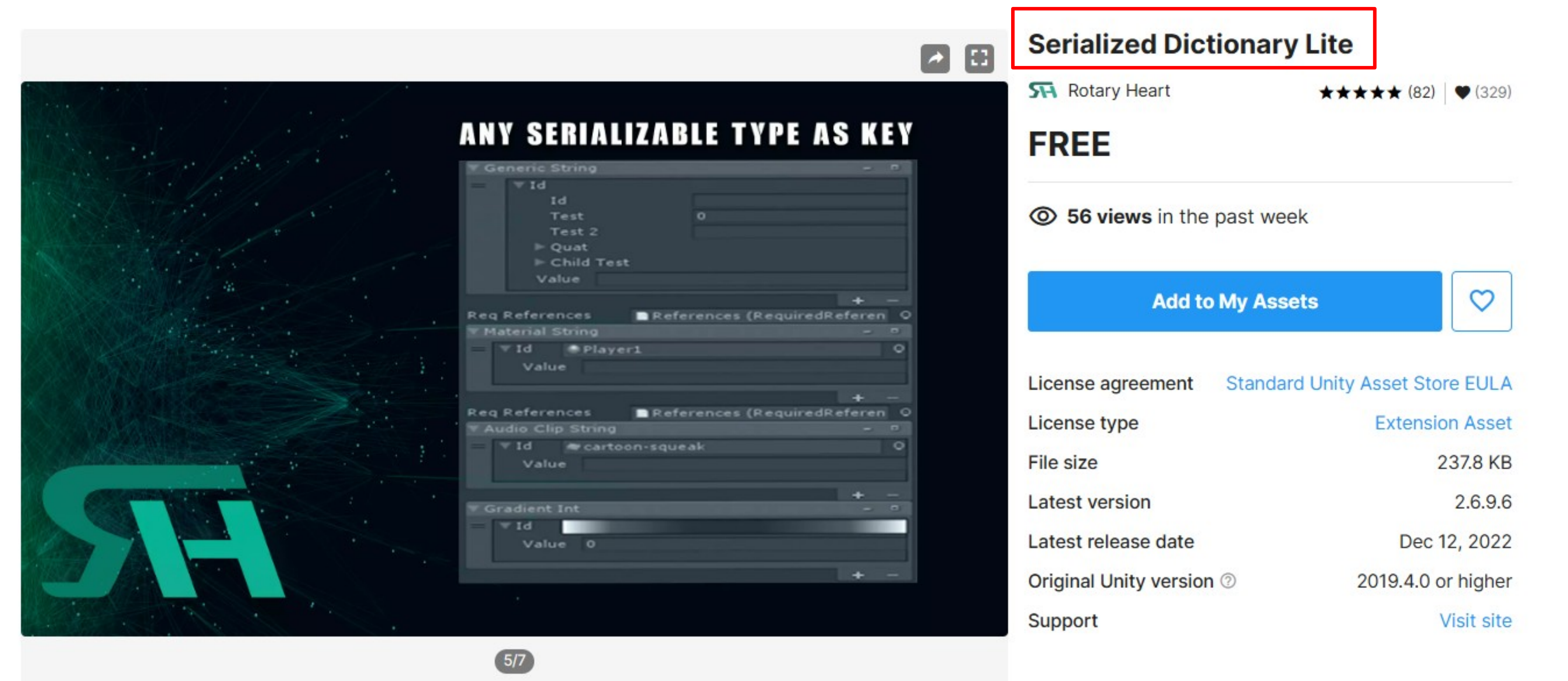
```
using System.Collections.Generic;
using UnityEngine;

public class EX_Dictionary_03 : MonoBehaviour
{
    public Dictionary<string, string> dict;

    private void Start()
    {
        dict = new Dictionary<string, string> {
            { "keyA", "valueA"},
            { "keyB", "valueB"},
            { "keyC", "valueC"}
        };

        foreach (var item in dict)  // use Anonymous type
        {
            print(item.Key + ":" + item.Value);
        }

        //foreach (KeyValuePair<string, string> item in dict)
        //{
        //    print(item.Key + ":" + item.Value);
        //}
    }
}
```



```
[01:07:25] keyA:valueA
UnityEngine.MonoBehaviour:print (object)
[01:07:25] keyB:valueB
UnityEngine.MonoBehaviour:print (object)
[01:07:25] keyC:valueC
UnityEngine.MonoBehaviour:print (object)
```

Plugin to Show Dictionary in Unity Editor

- Unity Editor doesn't support exposing a dictionary.
- You need some 3rd party plugin.



Serialized Dictionary Lite

Rotary Heart ★★★★★ (82) | ♥ (329)

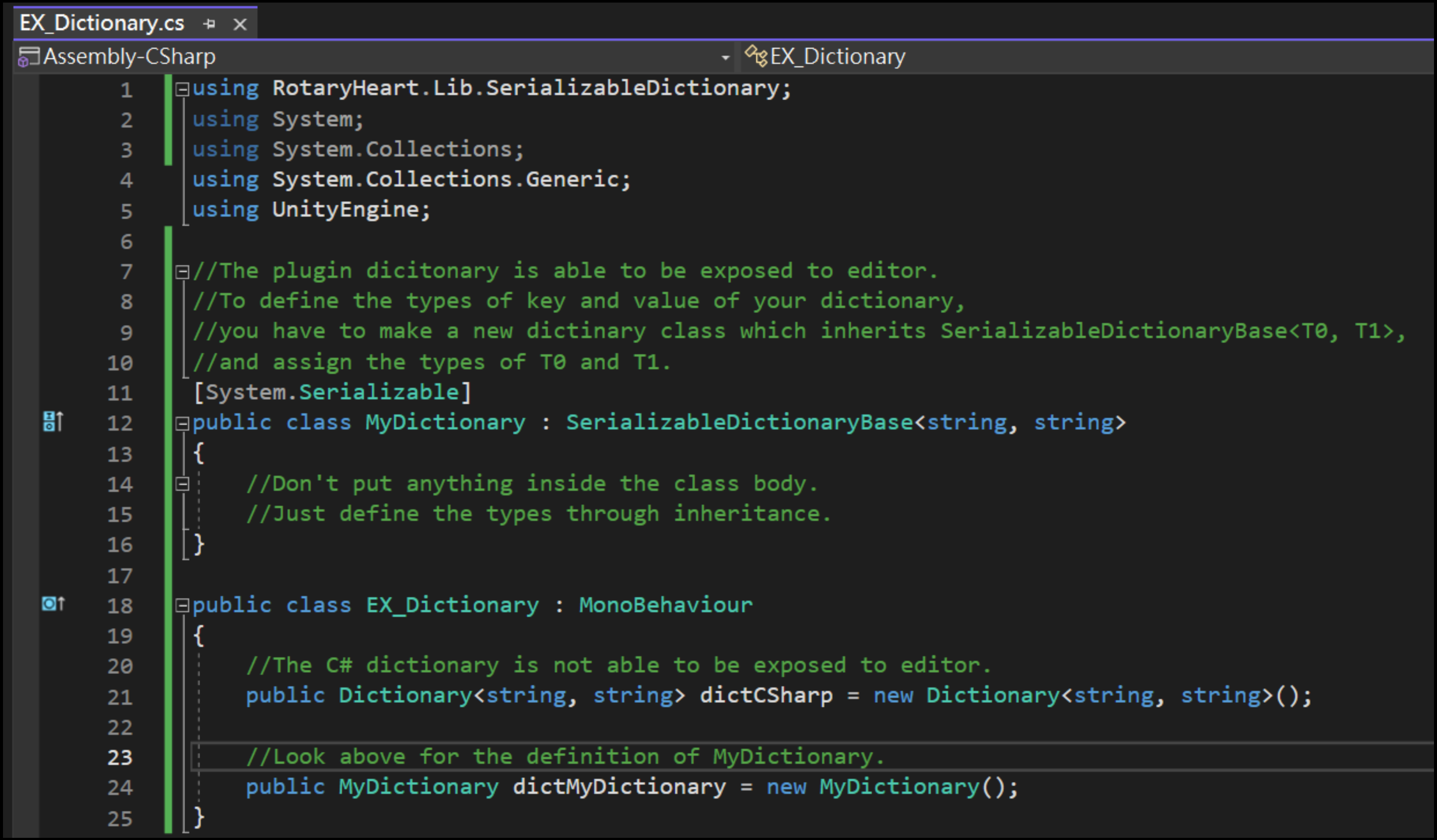
FREE

56 views in the past week

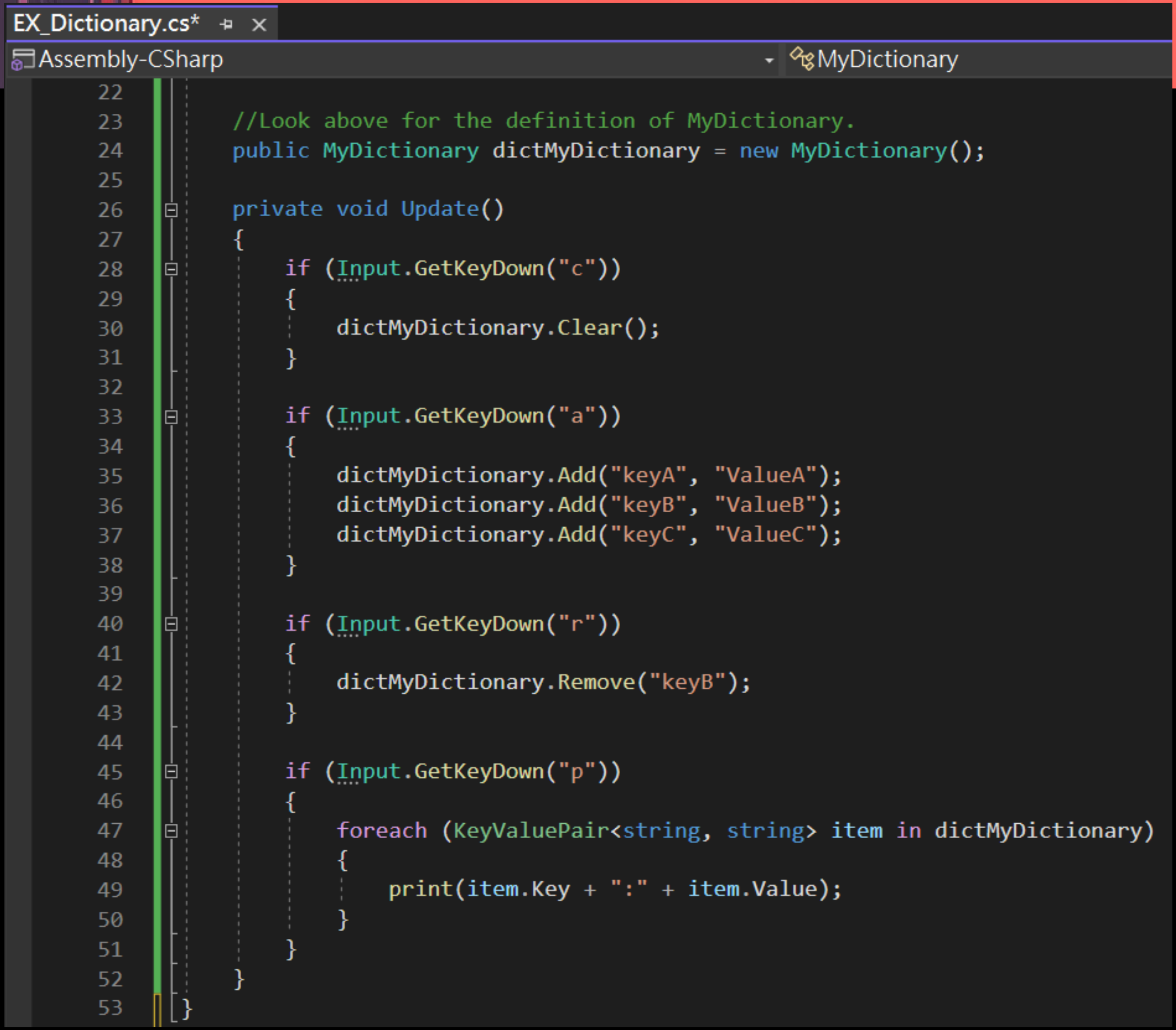
Add to My Assets

License agreement	Standard Unity Asset Store EULA
License type	Extension Asset
File size	237.8 KB
Latest version	2.6.9.6
Latest release date	Dec 12, 2022
Original Unity version	2019.4.0 or higher
Support	Visit site

Plugin to Show Dictionary in Unity Editor

- 

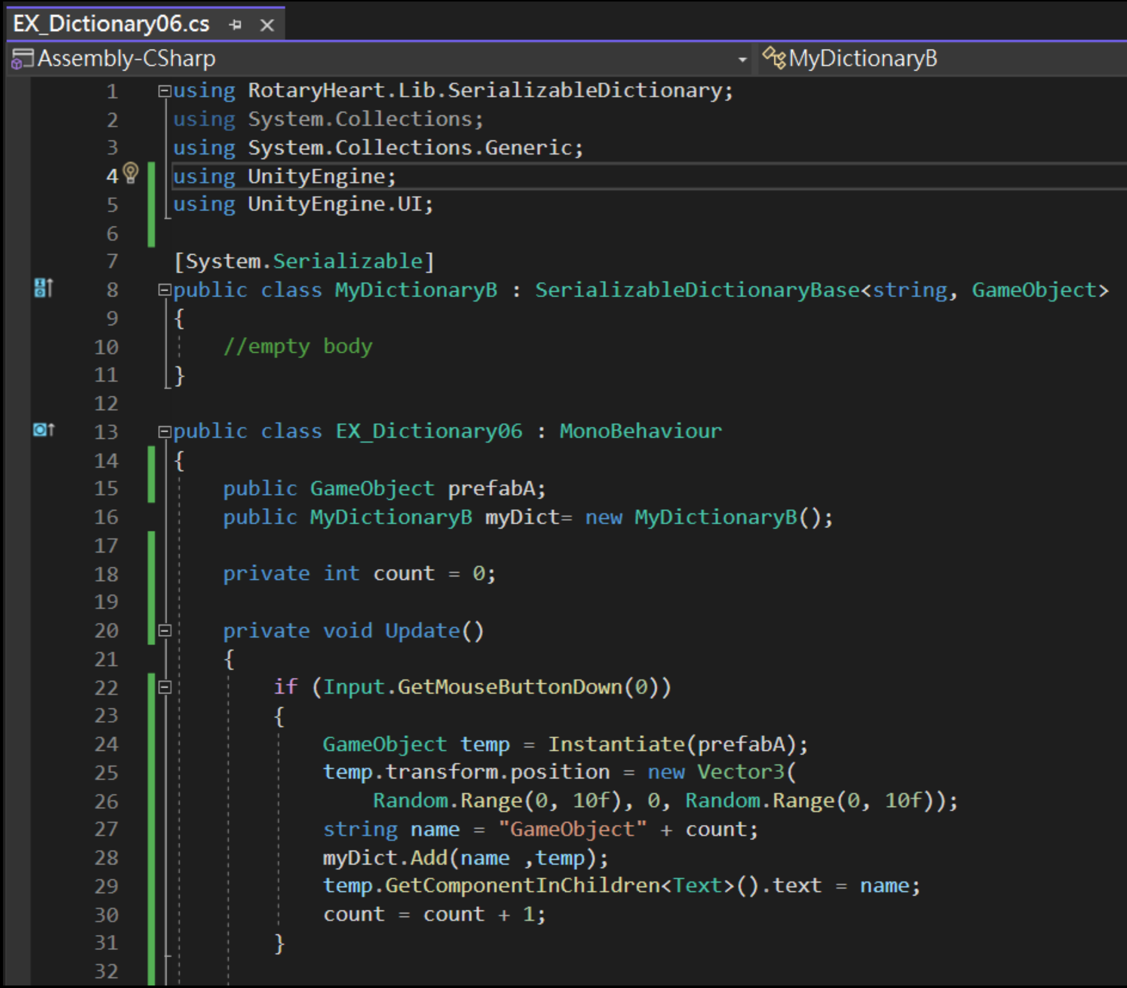
```
using RotaryHeart.Lib.SerializableDictionary;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

//The plugin dicitonary is able to be exposed to editor.
//To define the types of key and value of your dictionary,
//you have to make a new dictinary class which inherits SerializableDictionaryBase<T0, T1>,
//and assign the types of T0 and T1.
[System.Serializable]
public class MyDictionary : SerializableDictionaryBase<string, string>
{
    //Don't put anything inside the class body.
    //Just define the types through inheritance.
}

public class EX_Dictionary : MonoBehaviour
{
    //The C# dictionary is not able to be exposed to editor.
    public Dictionary<string, string> dictCSharp = new Dictionary<string, string>();

    //Look above for the definition of MyDictionary.
    public MyDictionary dictMyDictionary = new MyDictionary();
}
```



```
    //Look above for the definition of MyDictionary.
    public MyDictionary dictMyDictionary = new MyDictionary();

    private void Update()
    {
        if (Input.GetKeyDown("c"))
        {
            dictMyDictionary.Clear();
        }

        if (Input.GetKeyDown("a"))
        {
            dictMyDictionary.Add("keyA", "ValueA");
            dictMyDictionary.Add("keyB", "ValueB");
            dictMyDictionary.Add("keyC", "ValueC");
        }

        if (Input.GetKeyDown("r"))
        {
            dictMyDictionary.Remove("keyB");
        }

        if (Input.GetKeyDown("p"))
        {
            foreach (KeyValuePair<string, string> item in dictMyDictionary)
            {
                print(item.Key + ":" + item.Value);
            }
        }
    }
}
```

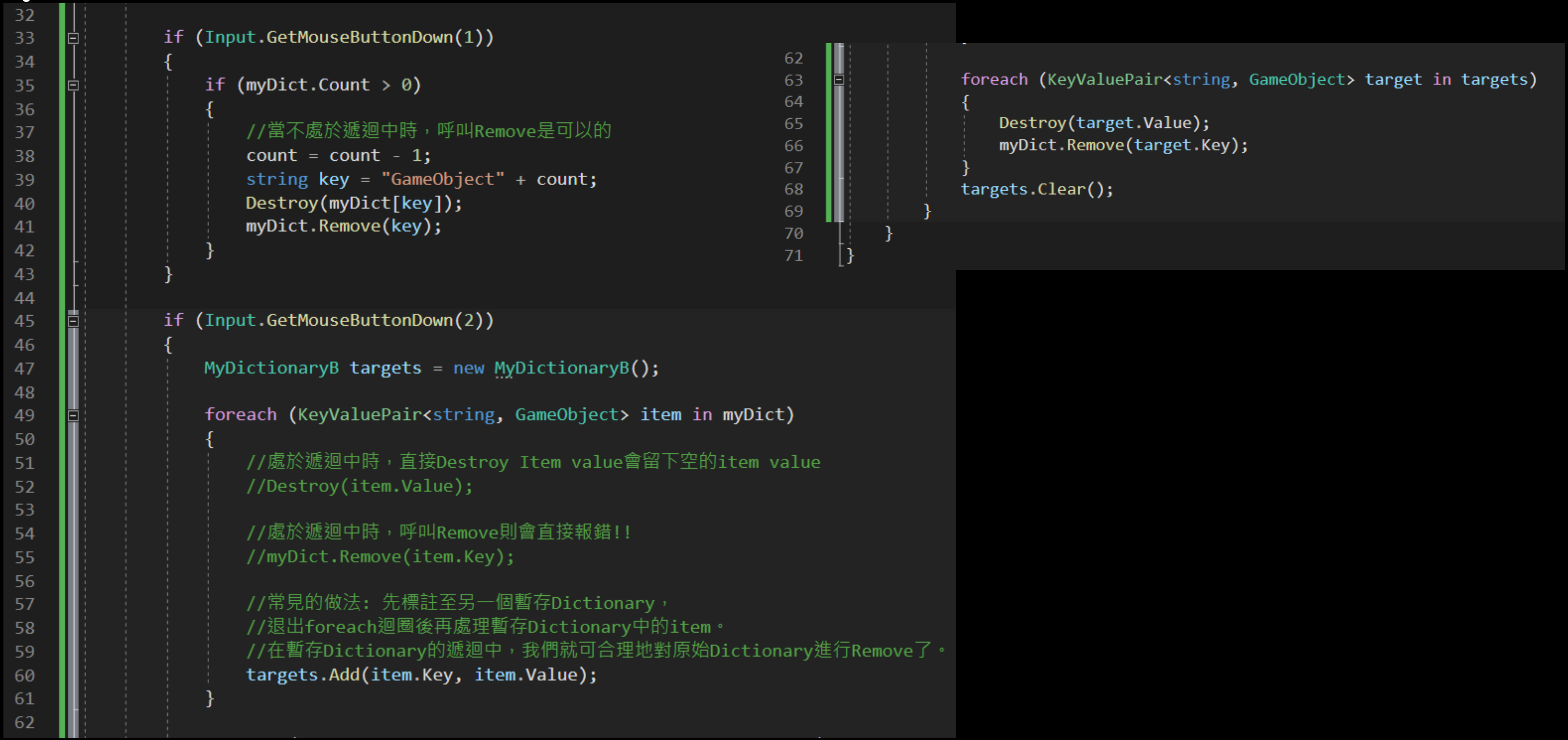
EX: Delete Dictionary Items in a Loop

- You can't delete a dictionary item in a foreach loop.
  - Just like you can't do this to a list item neither.
- Let's remake the deleting example by replacing the List with a Dictionary.



```csharp
using RotaryHeart.Lib.SerializableDictionary;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

[System.Serializable]
public class MyDictionaryB : SerializableDictionaryBase<string, GameObject>
{
    //empty body
}

public class EX_Dictionary06 : MonoBehaviour
{
    public GameObject prefabA;
    public MyDictionaryB myDict= new MyDictionaryB();

    private int count = 0;

    private void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            GameObject temp = Instantiate(prefabA);
            temp.transform.position = new Vector3(
                Random.Range(0, 10f), 0, Random.Range(0, 10f));
            string name = "GameObject" + count;
            myDict.Add(name ,temp);
            temp.GetComponentInChildren<Text>().text = name;
            count = count + 1;
        }

```

EX: Delete Dictionary Items in a Loop



```csharp

        if (Input.GetMouseButtonDown(1))
        {
            if (myDict.Count > 0)
            {
                //當不處於遞迴中時，呼叫Remove是可以的
                count = count - 1;
                string key = "GameObject" + count;
                Destroy(myDict[key]);
                myDict.Remove(key);
            }
        }

        if (Input.GetMouseButtonDown(2))
        {
            MyDictionaryB targets = new MyDictionaryB();

            foreach (KeyValuePair<string, GameObject> item in myDict)
            {
                //處於遞迴中時，直接Destroy Item value會留下空的item value
                //Destroy(item.Value);

                //處於遞迴中時，呼叫Remove則會直接報錯!!
                //myDict.Remove(item.Key);

                //常見的做法：先標註至另一個暫存Dictionary，
                //退出foreach迴圈後再處理暫存Dictionary中的item。
                //在暫存Dictionary的遞迴中，我們就可合理地對原始Dictionary進行Remove了。
                targets.Add(item.Key, item.Value);
            }

            foreach (KeyValuePair<string, GameObject> target in targets)
            {
                Destroy(target.Value);
                myDict.Remove(target.Key);
            }
            targets.Clear();
        }
    }
}
```

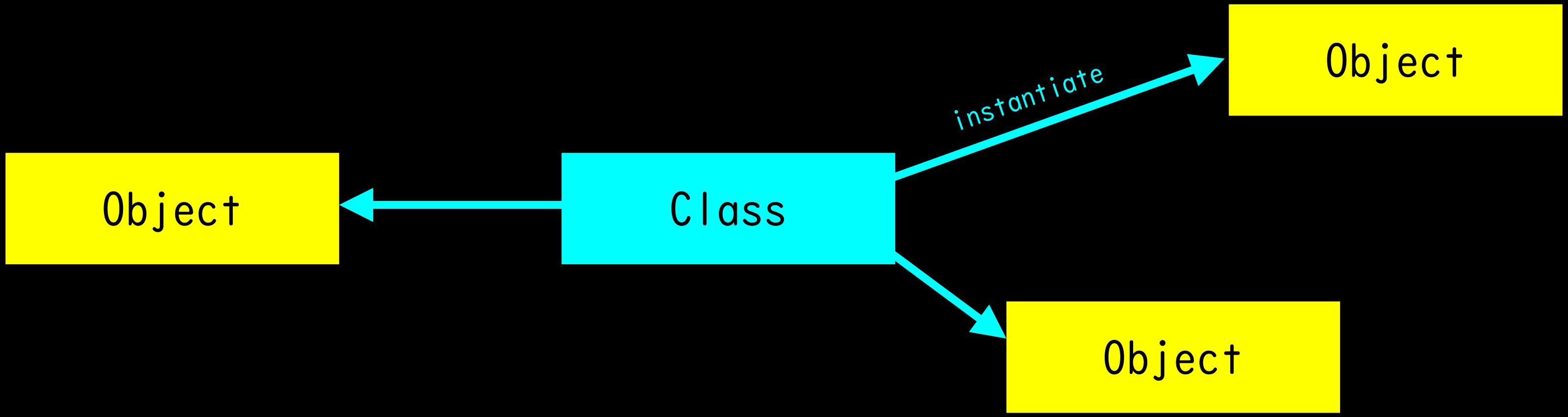
互動程式設計III

Interactive Programming Design Integration

- Data Collections
  - Enumeration
  - List
  - Dictionary
  - Class Object
  - JSON Object

What is Class Object

- Definition and Usage:
  - Definition: A Class in C# is a blueprint for creating objects. It encapsulates data for the object and methods to manipulate that data.
  - Usage: Classes are used to model complex data structures and behaviors in object-oriented programming (OOP). They provide a way to create objects with properties, methods, and events.
  - Example: Classes are commonly used to represent entities in a game, such as players, enemies, and items.



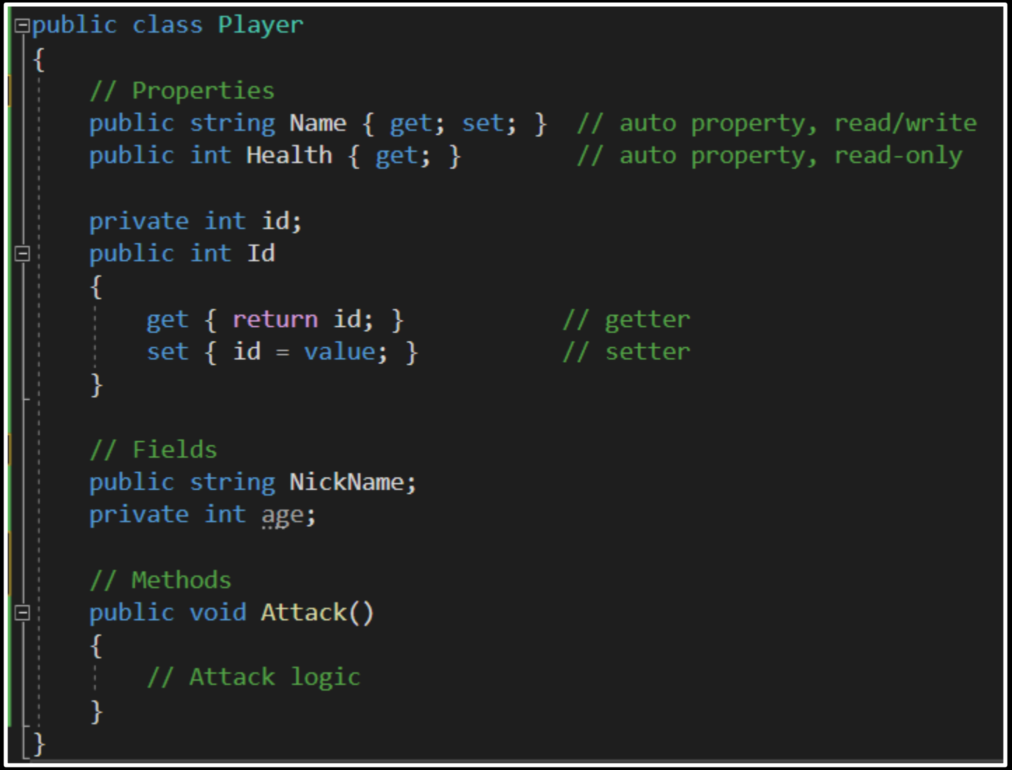
Class VS. Struct

- Comparison Between C# Class Object and Struct Object:
  - Class:
    - Definition: A class is a reference type that supports inheritance, polymorphism, and encapsulation.
    - Memory Allocation: Objects created from a class are stored on the heap, and variables hold references to these objects.
    - Usage: Ideal for complex data structures that require behavior, inheritance, and identity.
  - Struct:
    - Definition: A struct is a value type that is typically used for small, simple objects.
    - Memory Allocation: Struct instances are stored on the stack (if they are local variables) or inline (if they are fields in a class), and variables hold the actual data.
    - Usage: Suitable for lightweight objects that do not require inheritance and are often short-lived.
      - It is generally recommended to keep structs small, ideally under 16 bytes. This helps ensure they remain efficient and do not incur excessive copying overhead.

Class VS. Struct

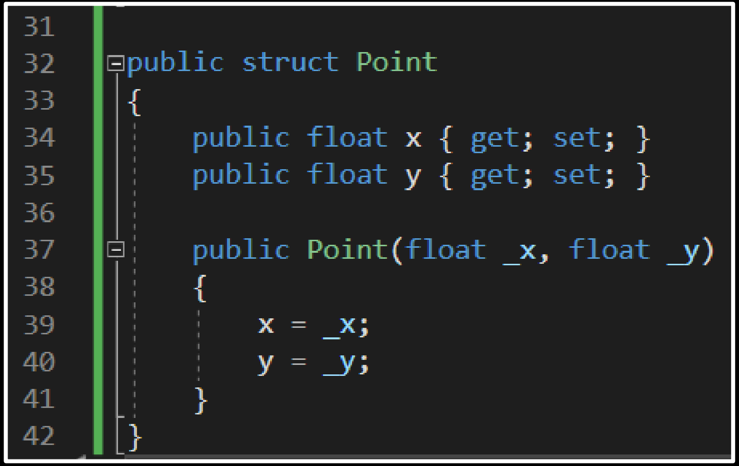
-

Class



```
public class Player
{
    // Properties
    public string Name { get; set; }  // auto property, read/write
    public int Health { get; }        // auto property, read-only

    private int id;
    public int Id
    {
        get { return id; }           // getter
        set { id = value; }          // setter
    }

    // Fields
    public string NickName;
    private int age;

    // Methods
    public void Attack()
    {
        // Attack logic
    }
}
```

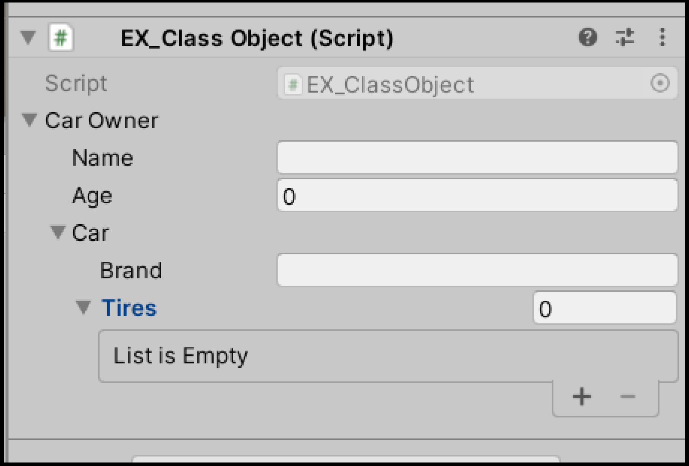
Struct



```

public struct Point
{
    public float x { get; set; }
    public float y { get; set; }

    public Point(float _x, float _y)
    {
        x = _x;
        y = _y;
    }
}
```

EX: Class Object

- An object in another object: A person owns a car.



Script: EX_ClassObject

Car Owner

Name:

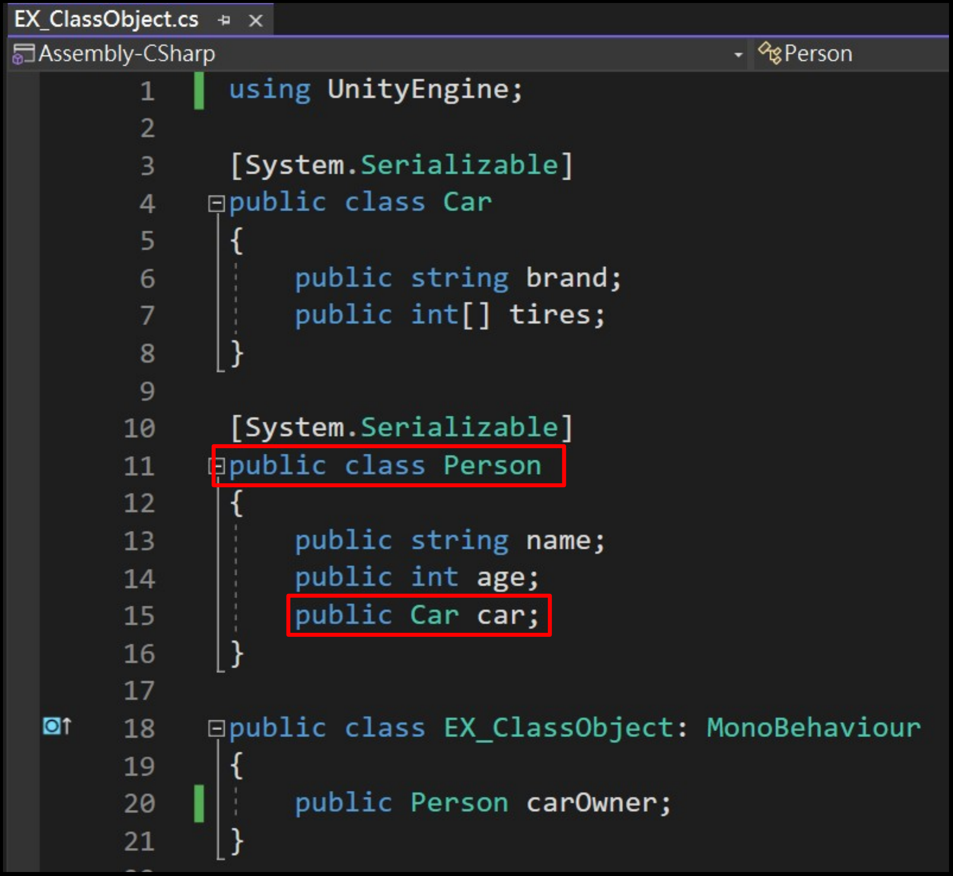
Age: 0

Car

Brand:

Tires: 0

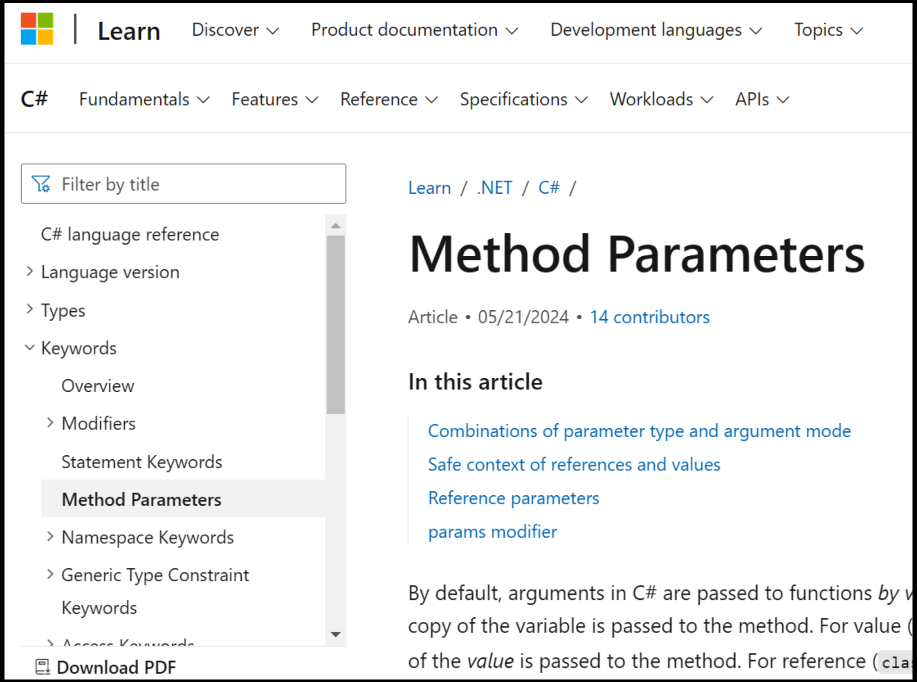
List is Empty



```
using UnityEngine;

[System.Serializable]
public class Car
{
    public string brand;
    public int[] tires;
}

[System.Serializable]
public class Person
{
    public string name;
    public int age;
    public Car car;
}

public class EX_ClassObject: MonoBehaviour
{
    public Person carOwner;
}
```

Pass by Value VS. Pass by Reference

- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/method-parameters#in-parameter-modifier



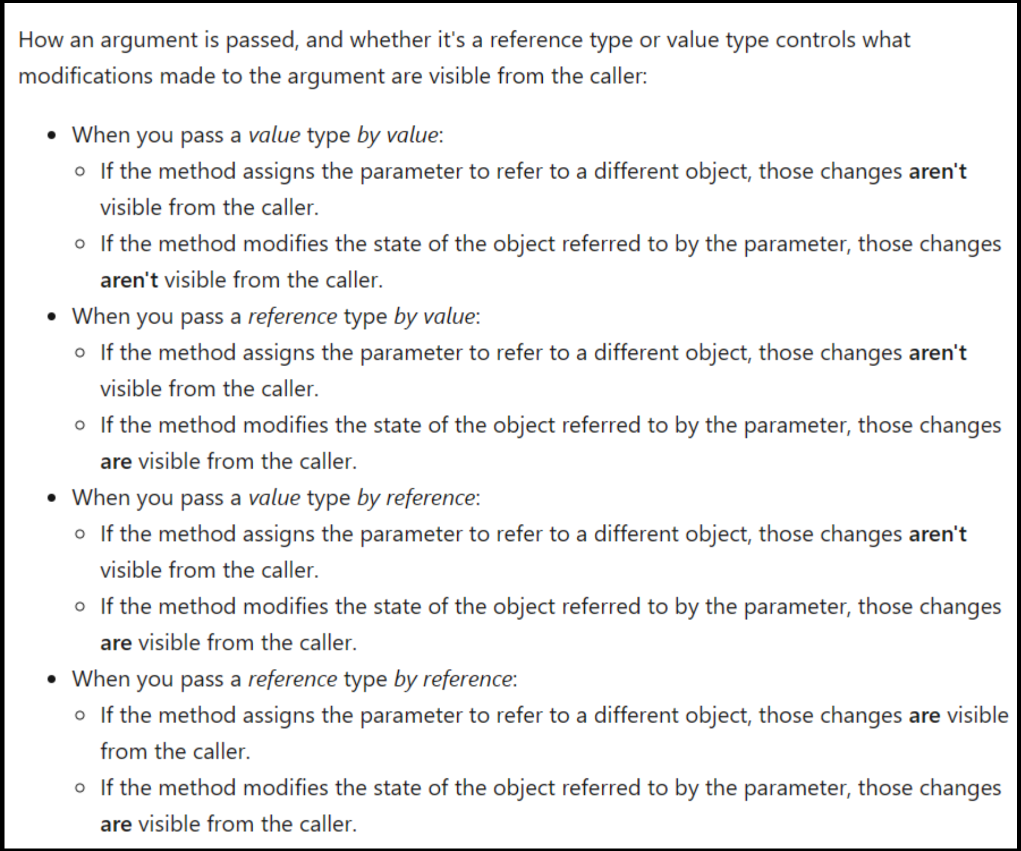
Method Parameters

Article • 05/21/2024 • 14 contributors

In this article

Combinations of parameter type and argument mode
Safe context of references and values
Reference parameters
params modifier

By default, arguments in C# are passed to functions *by v*
copy of the variable is passed to the method. For value (
of the *value* is passed to the method. For reference (`cla`



How an argument is passed, and whether it's a reference type or value type controls what modifications made to the argument are visible from the caller:

- When you pass a *value* type *by value*:
  - If the method assigns the parameter to refer to a different object, those changes **aren't** visible from the caller.
  - If the method modifies the state of the object referred to by the parameter, those changes **aren't** visible from the caller.
- When you pass a *reference* type *by value*:
  - If the method assigns the parameter to refer to a different object, those changes **aren't** visible from the caller.
  - If the method modifies the state of the object referred to by the parameter, those changes **are** visible from the caller.
- When you pass a *value* type *by reference*:
  - If the method assigns the parameter to refer to a different object, those changes **aren't** visible from the caller.
  - If the method modifies the state of the object referred to by the parameter, those changes **are** visible from the caller.
- When you pass a *reference* type *by reference*:
  - If the method assigns the parameter to refer to a different object, those changes **are** visible from the caller.
  - If the method modifies the state of the object referred to by the parameter, those changes **are** visible from the caller.

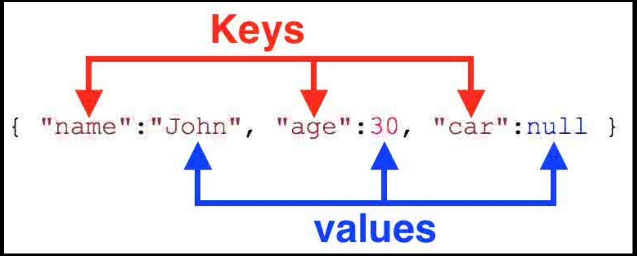
互動程式設計III

Interactive Programming Design Integration

- Data Collections
  - Enumeration
  - List
  - Dictionary
  - Class Object
  - JSON Object

What is JSON

- Definition and Usage
  - Definition: JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is text-based and uses key-value pairs to represent structured data.
  - Usage: JSON objects are commonly used for:
    - Storing configuration settings.
    - Exchanging data between a web server and a client.
    - Persisting data in files.
    - Interoperability between different systems and programming languages.
  - Structure:
    - Key-Value Pairs: JSON objects are collections of key-value pairs separated by commas, where keys are strings and values can be strings, numbers, arrays, booleans, or other JSON objects.



JSON Objects Explained! [*]

What is JSON

- Advantages
  - Human-Readable and Easy to Write:
    - JSON is straightforward and easy to understand, making it accessible for humans to read and write. Its syntax is simple and clean, which helps in quickly comprehending and generating data structures.
  - Language Independence:
    - JSON is language-independent but uses conventions familiar to programmers of the C family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. This makes it versatile and widely applicable.
  - Flexibility:
    - JSON supports a variety of data types, including strings, numbers, arrays, objects, booleans, and null values. This flexibility allows it to represent complex data structures effectively.
  - Interoperability:
    - JSON’s standardized format ensures compatibility across different systems and platforms. It facilitates data exchange between disparate systems, such as between a web server and a mobile app, or between different microservices.

What is JSON

- Objects:
  - Objects are enclosed in curly braces {}.
  - Each name is followed by a colon : and the name/value pairs are separated by commas.
  - The value can be another JSON object or JSON array.
- Arrays:
  - Arrays are enclosed in square brackets [].
  - Values are separated by commas.
- Values:
  - String, Number, Boolean, Null, JSON object, JSON array.

```
{
  "name": "John",
  "age": 30,
  "isStudent": false
}
```

JSON Object

```
[
  "apple",
  "banana",
  "cherry"
]
```

JSON Array

```
{
  "name": "John",
  "age": 30,
  "married": true,
  "children": null,
  "pets": ["dog", "cat"]
}
```

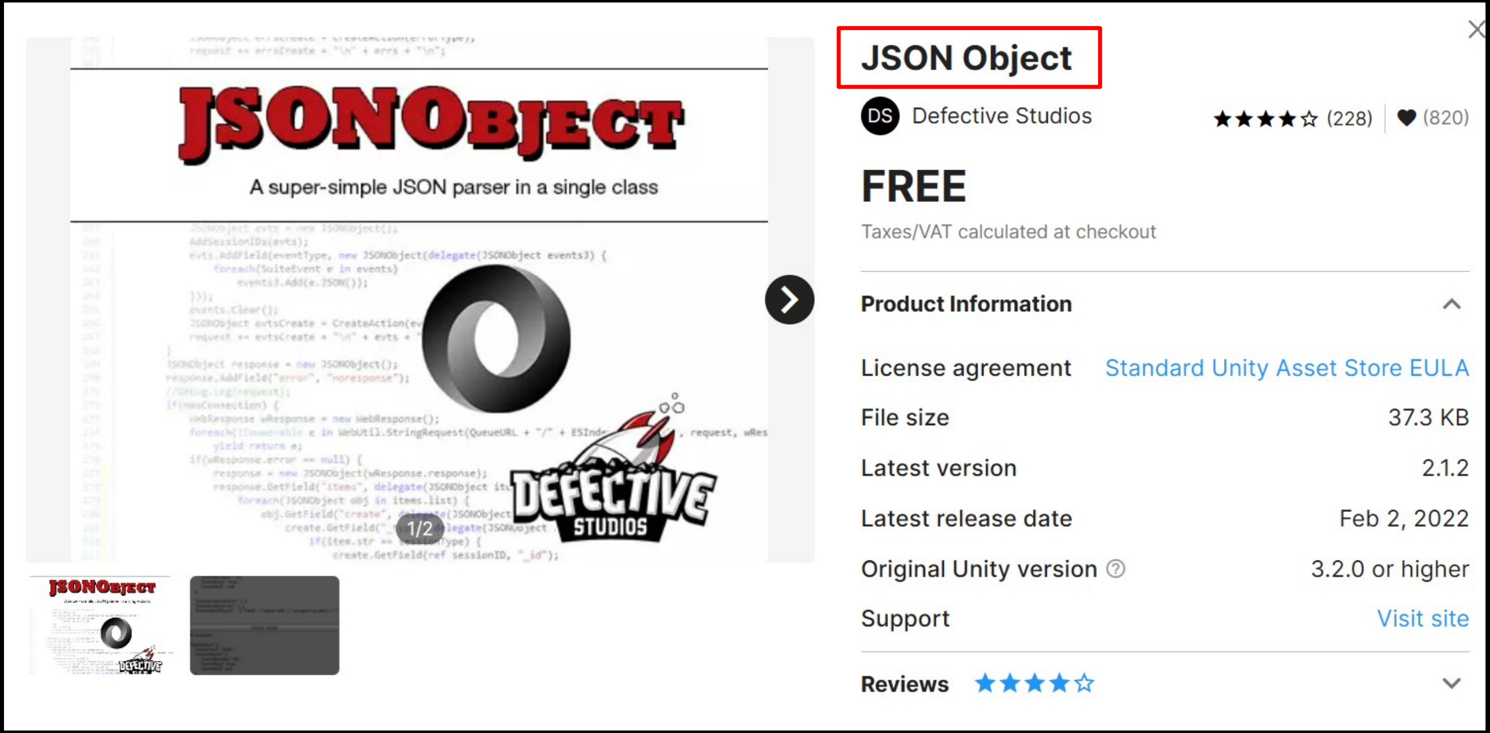
JSON Array in JSON Object

What is JSON Object

- Definition and Usage
  - Definition: JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is primarily used for transmitting data in web applications between a server and a client.
  - Usage: JSON objects are commonly used for:
    - Storing configuration settings.
    - Exchanging data between a web server and a client.
    - Persisting data in files.
    - Interoperability between different systems and programming languages.
- Structure of JSON Object
  - Key-Value Pairs: JSON objects are collections of key-value pairs, where keys are strings and values can be strings, numbers, arrays, booleans, or other JSON objects.

Unity Plugin for JSON

- Unity Editor doesn’t support exposing a JSON object.
  - You need some 3rd party plugin.



JSON Object

DS Defective Studios ★★★★☆ (228) | ♥ (820)

FREE

Taxes/VAT calculated at checkout

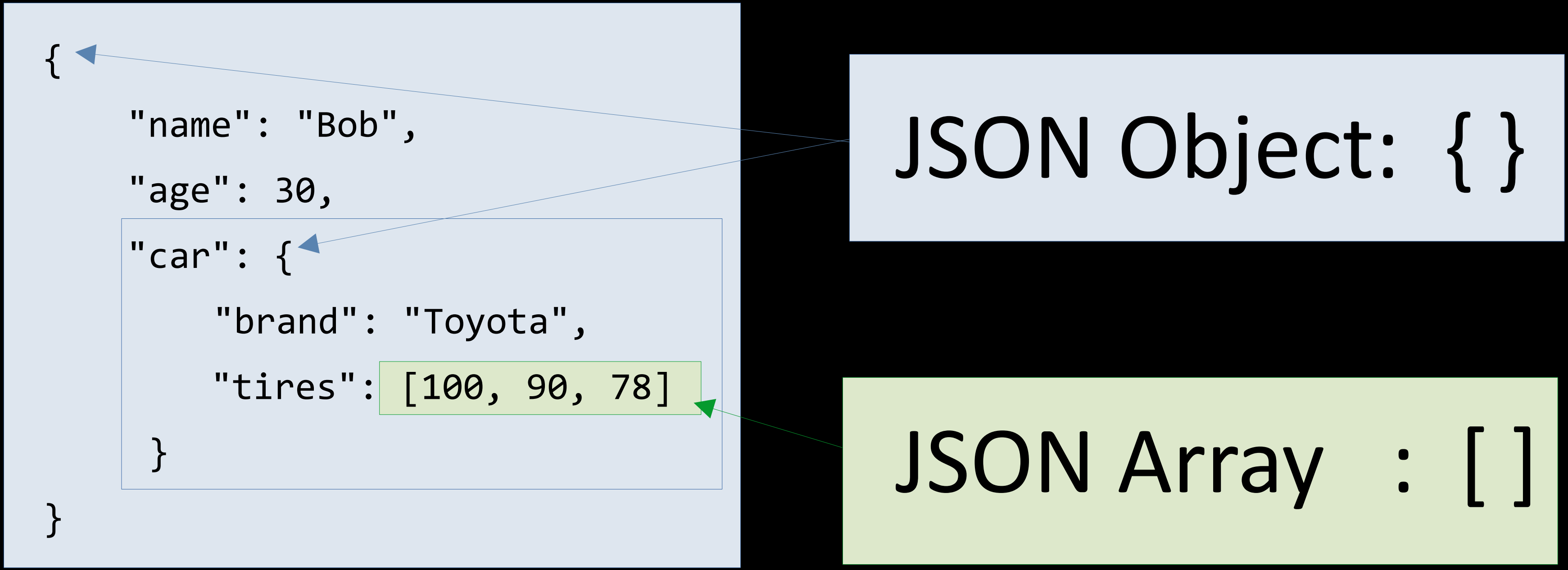
Product Information

License agreement	Standard Unity Asset Store EULA
File size	37.3 KB
Latest version	2.1.2
Latest release date	Feb 2, 2022
Original Unity version	3.2.0 or higher
Support	Visit site

Reviews ★★★★☆

Unity Plugin for JSON

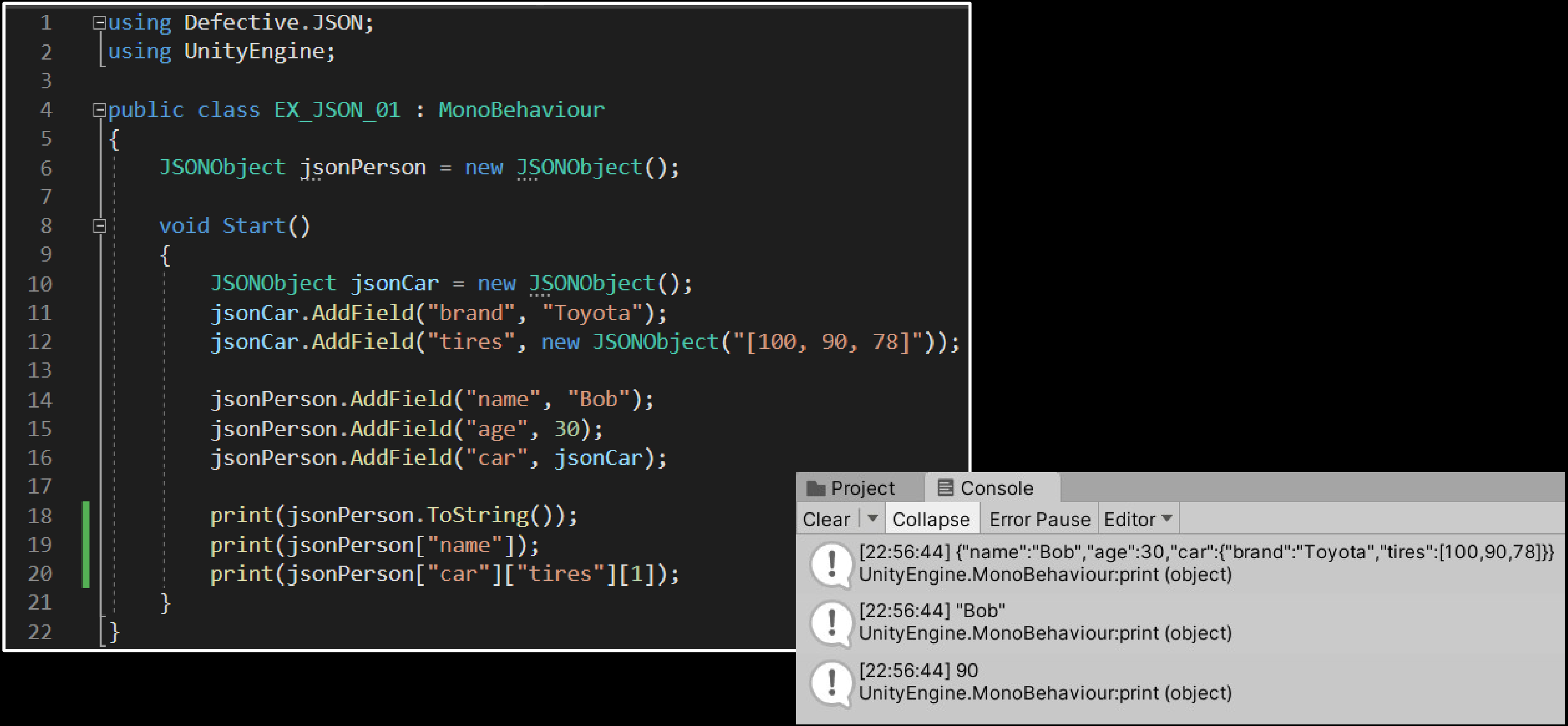
- JSON in JSON



```
{
     "name": "Bob",
     "age": 30,
     "car": {
          "brand": "Toyota",
          "tires": [100, 90, 78]
      }
}
```

EX: JSON Object in JSON Object

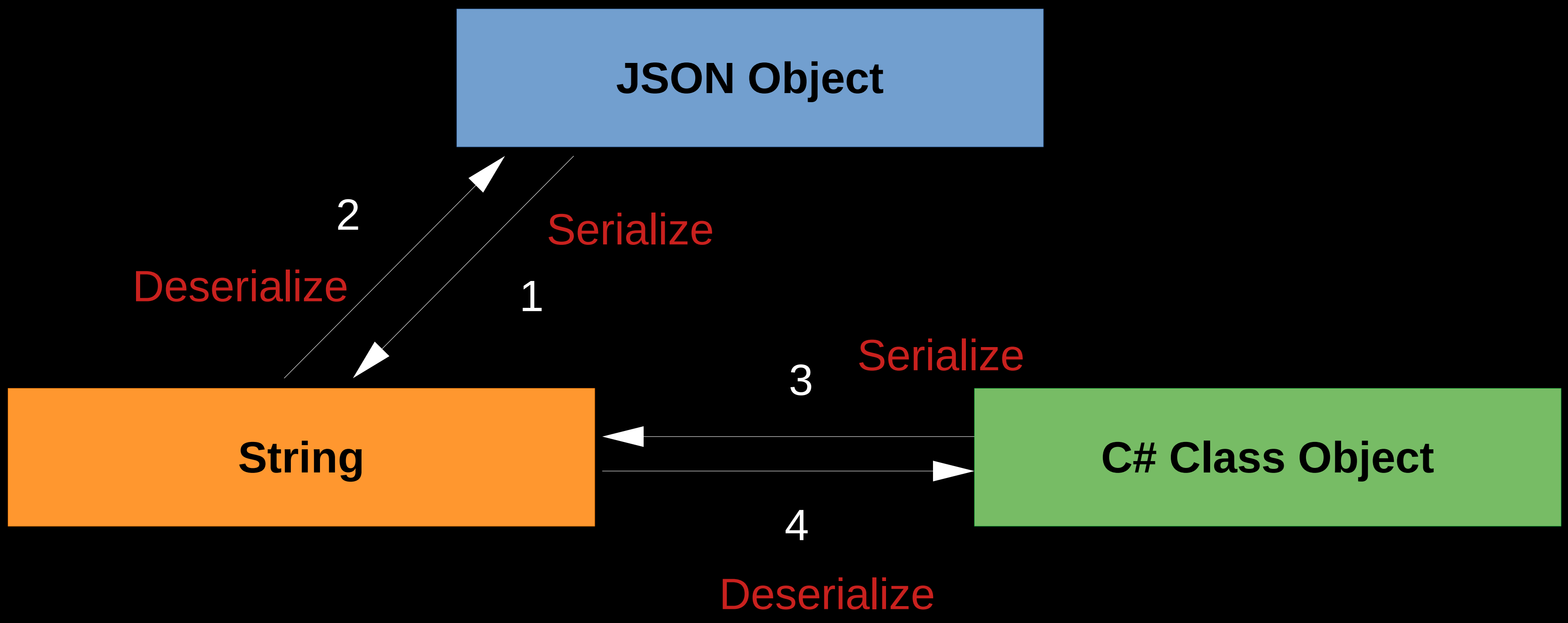
-



```csharp
using Defective.JSON;
using UnityEngine;

public class EX_JSON_01 : MonoBehaviour
{
    JSONObject jsonPerson = new JSONObject();

    void Start()
    {
        JSONObject jsonCar = new JSONObject();
        jsonCar.AddField("brand", "Toyota");
        jsonCar.AddField("tires", new JSONObject("[100, 90, 78]"));

        jsonPerson.AddField("name", "Bob");
        jsonPerson.AddField("age", 30);
        jsonPerson.AddField("car", jsonCar);

        print(jsonPerson.ToString());
        print(jsonPerson["name"]);
        print(jsonPerson["car"]["tires"][1]);
    }
}
```

```
[22:56:44] {"name":"Bob","age":30,"car":{"brand":"Toyota","tires":[100,90,78]}}
UnityEngine.MonoBehaviour:print (object)
[22:56:44] "Bob"
UnityEngine.MonoBehaviour:print (object)
[22:56:44] 90
UnityEngine.MonoBehaviour:print (object)
```

Conversion Between JSON Object, String and C# Class Object



JSON Object

2

Serialize

Deserialize

1

Serialize

3

String

C# Class Object

4

Deserialize

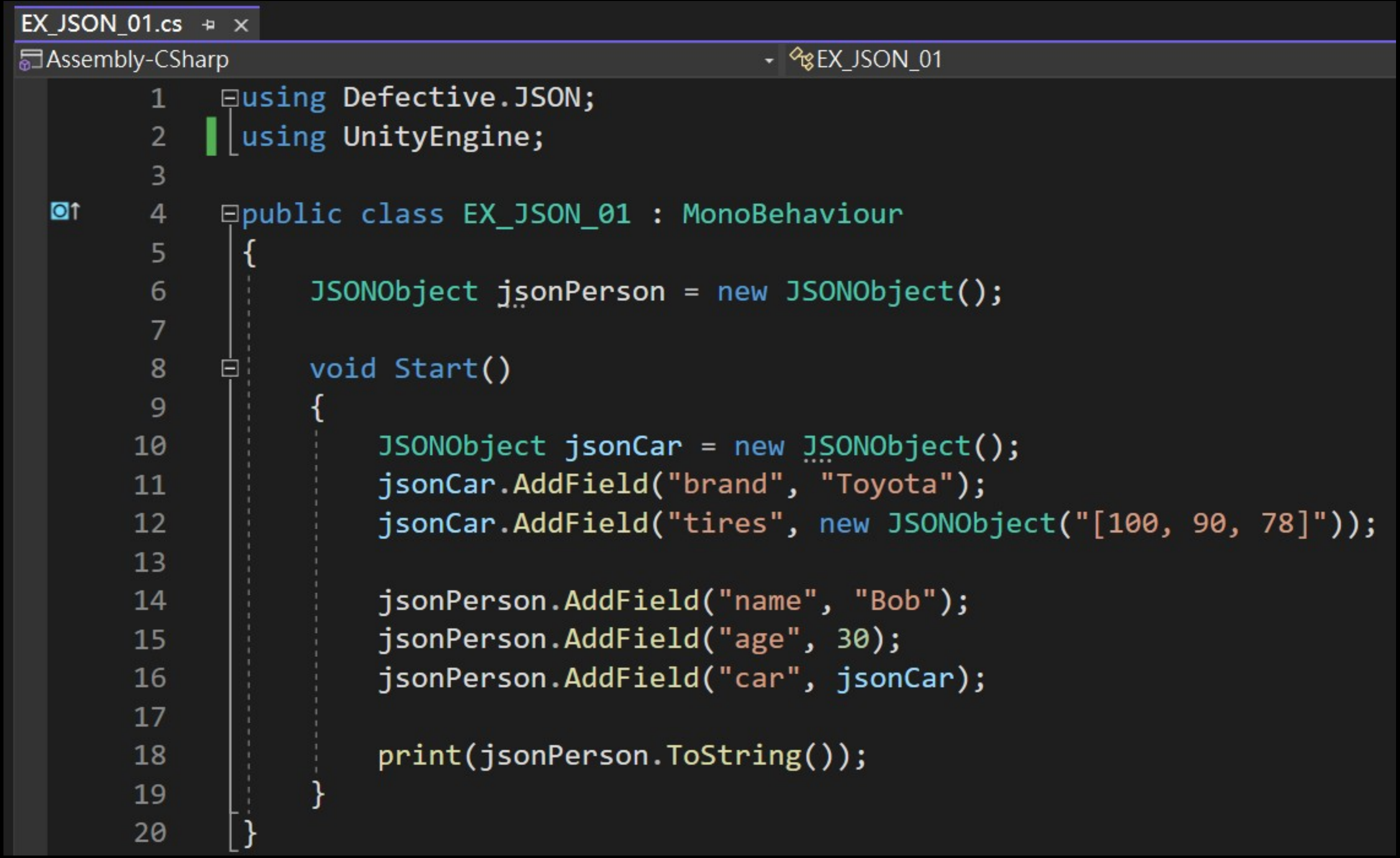
Serialize/Deseirialize

- Serialize: Transfer N dimensional Data into 1D series.
- Deserialize: Transfer 1D series back to N dimensional Data.

Bob
atoyoT63h.B
Bob

EX: Write JSON Object to String (1)

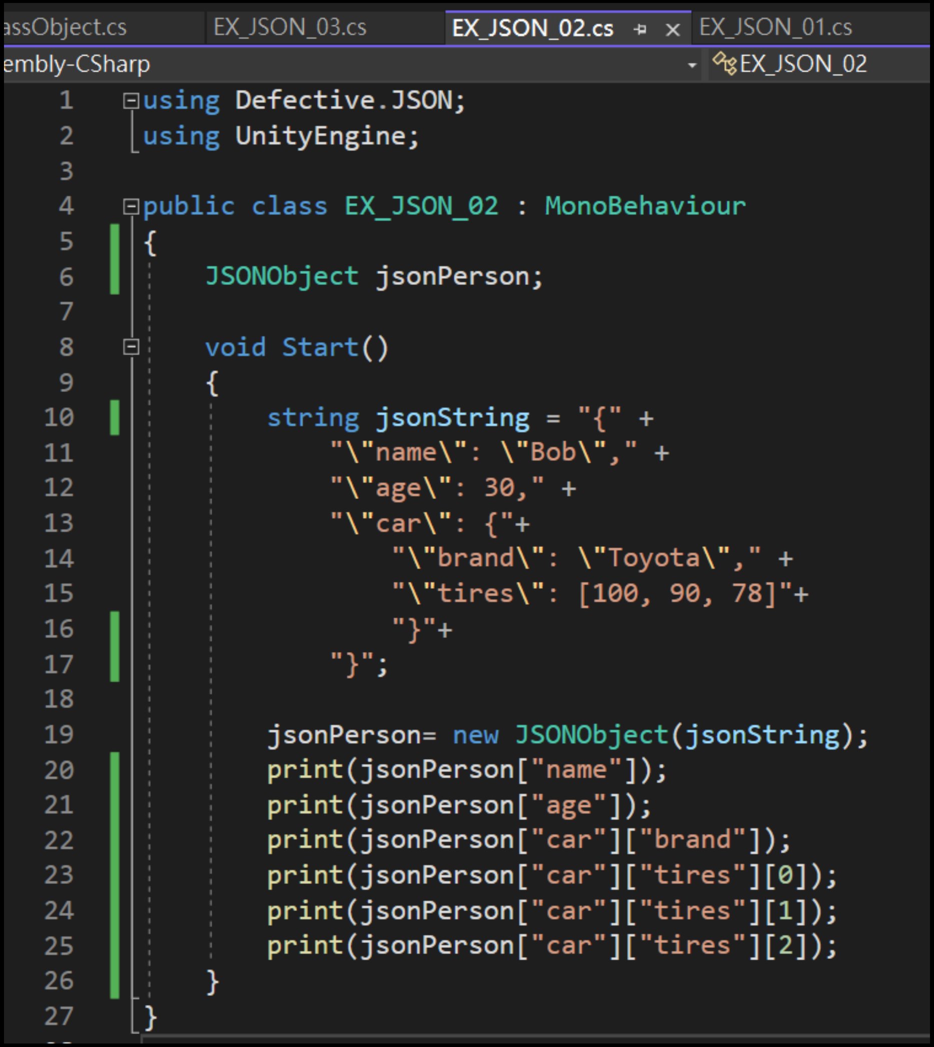
- Making a JSON Object to be a string is also called serialization of a JSON object.



```
using Defective.JSON;
using UnityEngine;

public class EX_JSON_01 : MonoBehaviour
{
    JSONObject jsonPerson = new JSONObject();

    void Start()
    {
        JSONObject jsonCar = new JSONObject();
        jsonCar.AddField("brand", "Toyota");
        jsonCar.AddField("tires", new JSONObject("[100, 90, 78]"));

        jsonPerson.AddField("name", "Bob");
        jsonPerson.AddField("age", 30);
        jsonPerson.AddField("car", jsonCar);

        print(jsonPerson.ToString());
    }
}
```

EX: Read JSON Object from String (2)

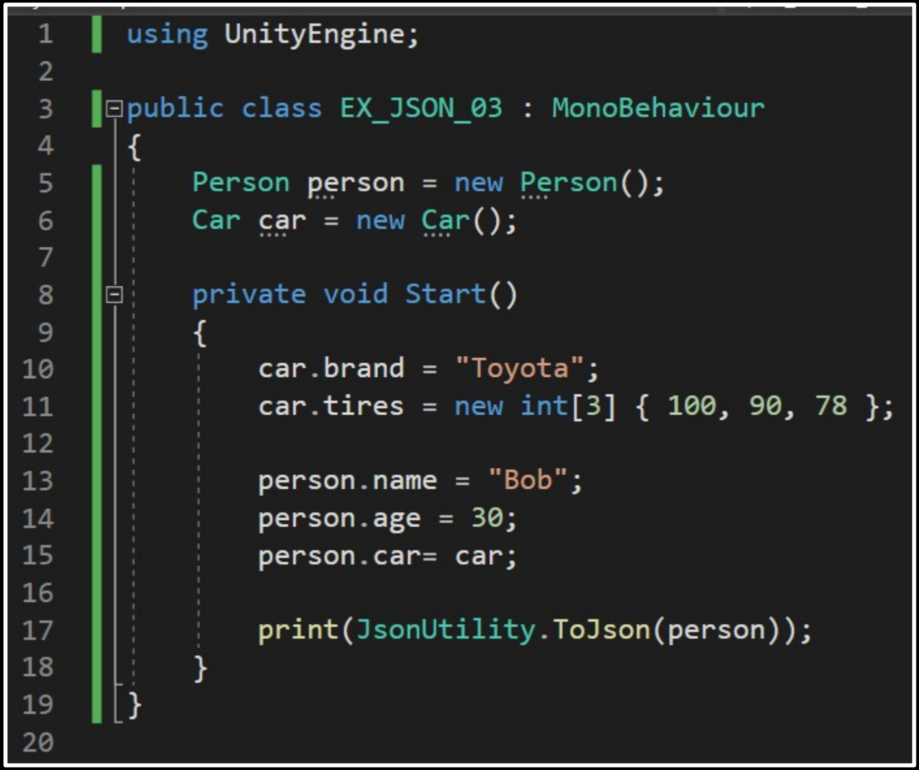
- Load a JSON Object from a string is also called deserialization of a JSON string.



```
using Defective.JSON;
using UnityEngine;

public class EX_JSON_02 : MonoBehaviour
{
    JSONObject jsonPerson;

    void Start()
    {
        string jsonString = "{" +
            "\"name\": \"Bob\"," +
            "\"age\": 30," +
            "\"car\": {"+
                "\"brand\": \"Toyota\"," +
                "\"tires\": [100, 90, 78]"+
                "}"+
            "}";

        jsonPerson= new JSONObject(jsonString);
        print(jsonPerson["name"]);
        print(jsonPerson["age"]);
        print(jsonPerson["car"]["brand"]);
        print(jsonPerson["car"]["tires"][0]);
        print(jsonPerson["car"]["tires"][1]);
        print(jsonPerson["car"]["tires"][2]);
    }
}
```

EX: Write C# Class Object to String (3)

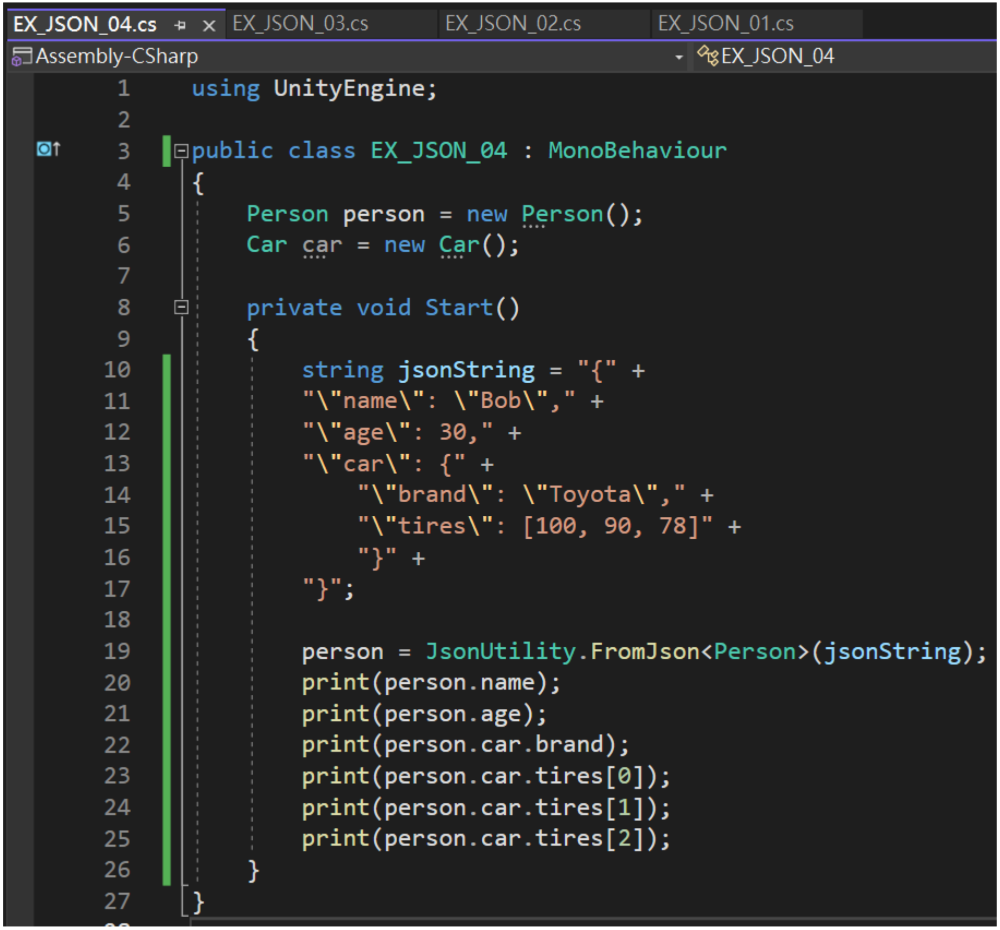
- Unity provides a JsonUtility class to facilitate some helpful conversion APIs.
  - public static T FromJson(string json);
  - public static void FromJsonOverwrite(string json, object objectToOverwrite);
  - public static string ToJson(object obj);



```
using UnityEngine;

public class EX_JSON_03 : MonoBehaviour
{
    Person person = new Person();
    Car car = new Car();

    private void Start()
    {
        car.brand = "Toyota";
        car.tires = new int[3] { 100, 90, 78 };

        person.name = "Bob";
        person.age = 30;
        person.car= car;

        print(JsonUtility.ToJson(person));
    }
}
```

EX: Read C# Class Object from String (4)

- Unity provides a JsonUtility class to facilitate some helpful conversion APIs.
  - public static T FromJson(string json);
  - public static void FromJsonOverwrite(string json, object objectToOverwrite);
  - public static string ToJson(object obj);



```csharp
using UnityEngine;

public class EX_JSON_04 : MonoBehaviour
{
    Person person = new Person();
    Car car = new Car();

    private void Start()
    {
        string jsonString = "{" +
        "\"name\": \"Bob\"," +
        "\"age\": 30," +
        "\"car\": {" +
            "\"brand\": \"Toyota\"," +
            "\"tires\": [100, 90, 78]" +
            "}" +
        "}";

        person = JsonUtility.FromJson<Person>(jsonString);
        print(person.name);
        print(person.age);
        print(person.car.brand);
        print(person.car.tires[0]);
        print(person.car.tires[1]);
        print(person.car.tires[2]);
    }
}
```

Quiz

1. What is Enum primarily used for?
   - A. Storing large amounts of data
   - B. Representing a collection of related constants
   - C. Performing mathematical operations
   - D. Storing dynamic data
   - **Answer: B**

2. Which of the following is not a primary benefit of using Enum?
   - A. Improved code readability
   - B. Enhanced type safety
   - C. Reduced memory usage
   - D. Improved code organization
   - **Answer: C**

Quiz

3. Which method in a List is used to insert an element?

- A. Add()
- B. Insert()
- C. Remove()
- D. Contains()
- **Answer: B**

4. What is the main difference between a List and an Array in C#?

- A. List is of fixed size, Array is dynamic
- B. Array is of fixed size, List is dynamic
- C. Both List and Array can adjust size dynamically
- D. Neither List nor Array can adjust size dynamically
- **Answer: B**

Quiz

5. The primary feature of a Dictionary is:

- A. Elements are ordered
- B. Keys must be unique
- C. Values must be unique
- D. Both keys and values must be unique
- **Answer: B**

6. Which method can be used to check if a specific key exists in a Dictionary?

- A. Contains()
- B. ContainsKey()
- C. ContainsValue()
- D. Exists()
- **Answer: B**

Quiz

7. What does JSON stand for?

- A. Java Standard Object Notation
- B. JavaScript Object Notation
- C. JavaScript Open Notation
- D. Java Open Notation
- **Answer: B**

8. Which of the following is not a data type in JSON?

- A. String
- B. Number
- C. List
- D. Function
- **Answer: D**

Quiz

9. Which class is commonly used in Unity for handling JSON data?

- A. System.Collections
- B. System.IO
- C. UnityEngine.UI
- D. UnityEngine.JsonUtility
- **Answer: D**

10. Which method is used to read a C# class object from a JSON string in Unity using UnityEngine.JsonUtility?

- A. DeserializeObject()
- B. SerializeObject()
- C. FromJson()
- D. ToJson()
- **Answer: C**

Project

11. Mary wants to design an application which lets her teacher to calculate the average test score of the students in her class. Please help Mary to define, design and develop the application.

- Define: What is the key feature (or goal) of this application?
- Design: What are the operation scenarios to enable the feature (or achieve the goal)?
- Develop: How to implement the system to support required operations?

Can I use a JSON array?

Can I use a List?

Can I use a Dictionary?

Can I use...

互動程式設計III

Interactive Programming Design Integration

Q&A