

**Task3.1**

**(1) what’s the operator ‘using’ used for?**

The 'using' operator in C# is used for two different purposes in the context of Unity development:

- Namespace Import: The 'using' keyword is used to import namespaces in C#. By using the 'using' directive at the beginning of a code file, you can bring in the types and members of a specific namespace into the current scope. For example, 'using UnityEngine;' allows you to access Unity's built-in classes and functionalities without specifying the full namespace each time.

- Resource Cleanup: In Unity, the 'using' statement is used for resource management and cleanup. It allows you to declare and define a variable within the 'using' block, and this variable is automatically disposed of when it goes out of scope. This is particularly useful for objects that implement the IDisposable interface, as it ensures their resources are properly released when no longer needed.

**(2) which class does the class ‘MonoBehaviour’ inherit? What are the new/overridden members (attributes and functions) defined?**

The class 'MonoBehaviour' in Unity inherits from the class 'Behaviour'. The 'Behaviour' class is part of the UnityEngine namespace and serves as the base class for all Unity components that can be attached to GameObjects.

When deriving from 'MonoBehaviour', you can override several members, such as:

- Functions:

- Start(): This function is called once, just before the first frame update, and is commonly used for initialization tasks.

- Update(): This function is called once per frame and is often used for updating game logic, input handling, or animation.

- FixedUpdate(): This function is called at fixed time intervals and is used for physics-related calculations.

- LateUpdate(): This function is called after all Update functions and is useful for actions that need to happen after other updates.

- Attributes:

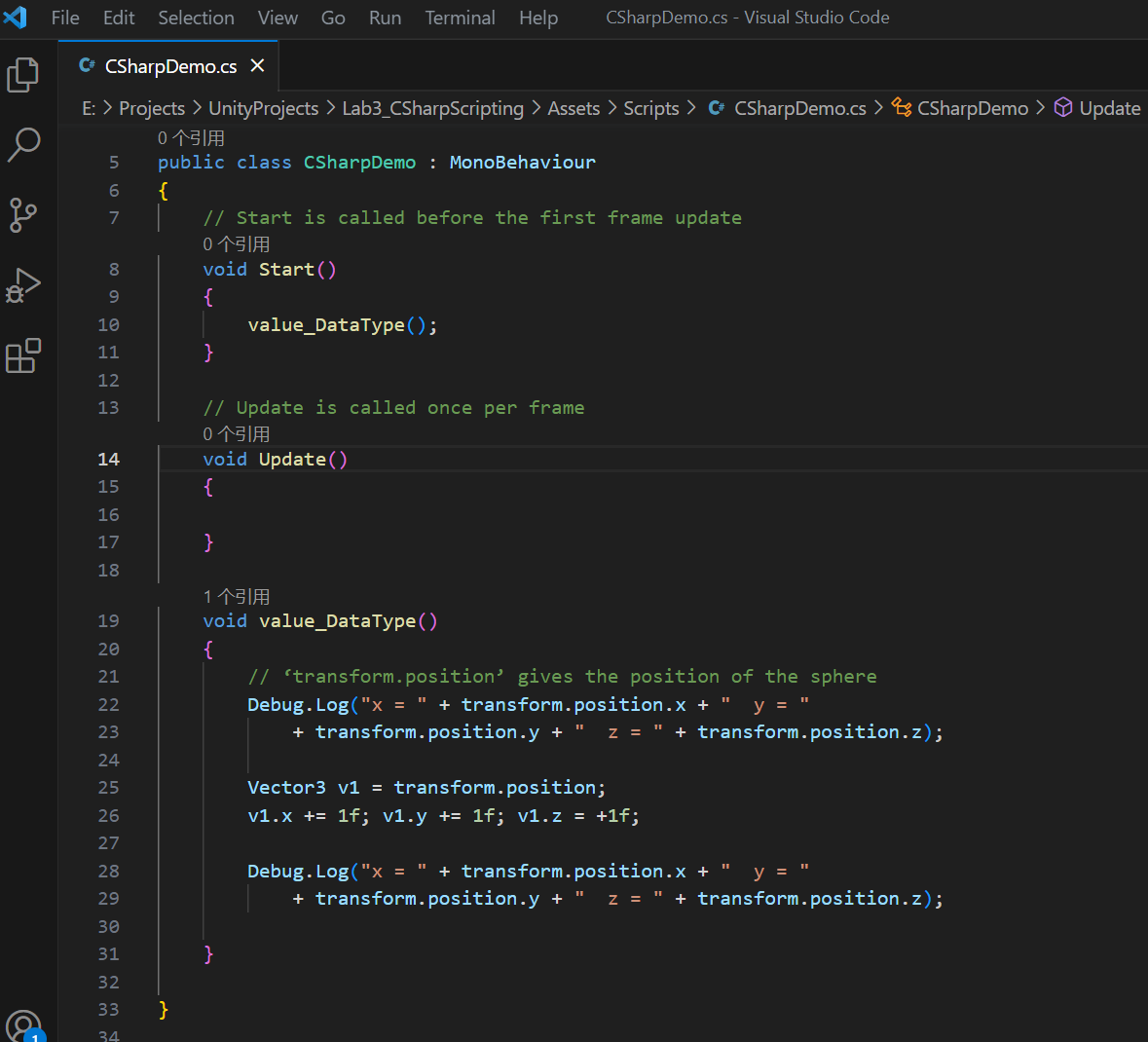
- [SerializeField]: This attribute allows private fields to be serialized and displayed in the Unity Inspector for editing.

- [Range(min, max)]: This attribute allows you to specify a minimum and maximum value for a float or int field in the Inspector.

- [HideInInspector]: This attribute hides a serialized field from the Inspector, while still allowing it to be edited in code.

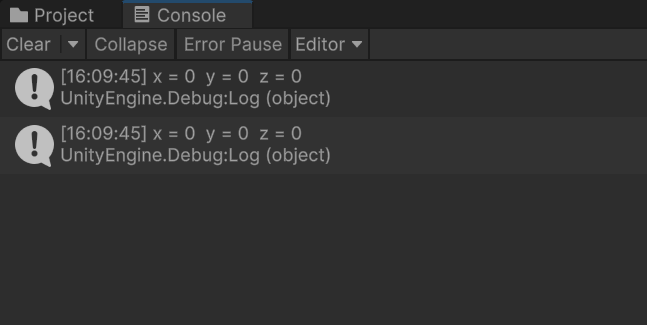
These are just a few examples, and there are many more members that can be overridden or used with the 'MonoBehaviour' class in Unity.

**Task3.2**



**(1) What is the printout in the Console window? Does the sphere change its position?**

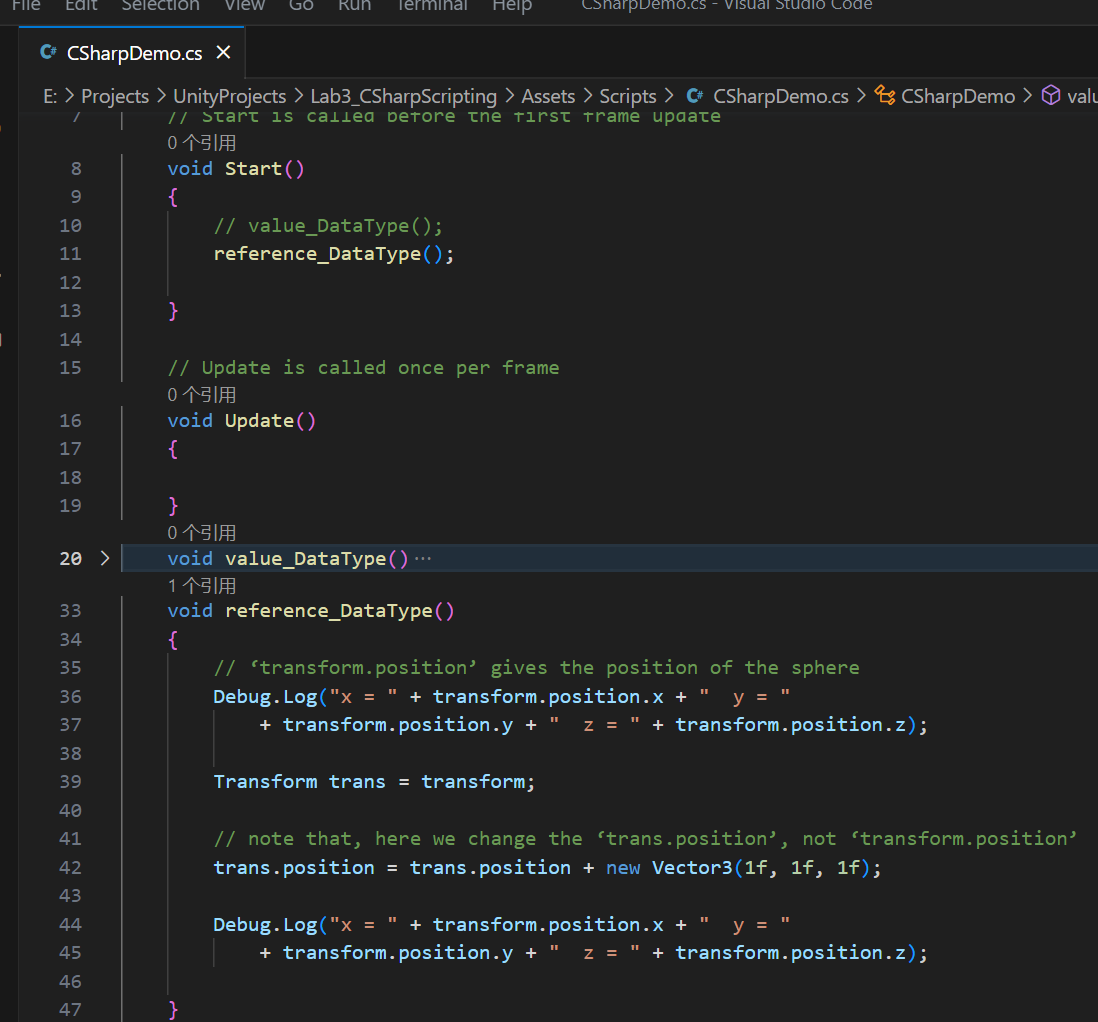
The following figure shows the printout in the Console window. The sphere does not change its position.



**(2) Is the Vector3 a value type or reference type? When we amend ‘v1’, does the ‘transform.position’ change as well? Why?**

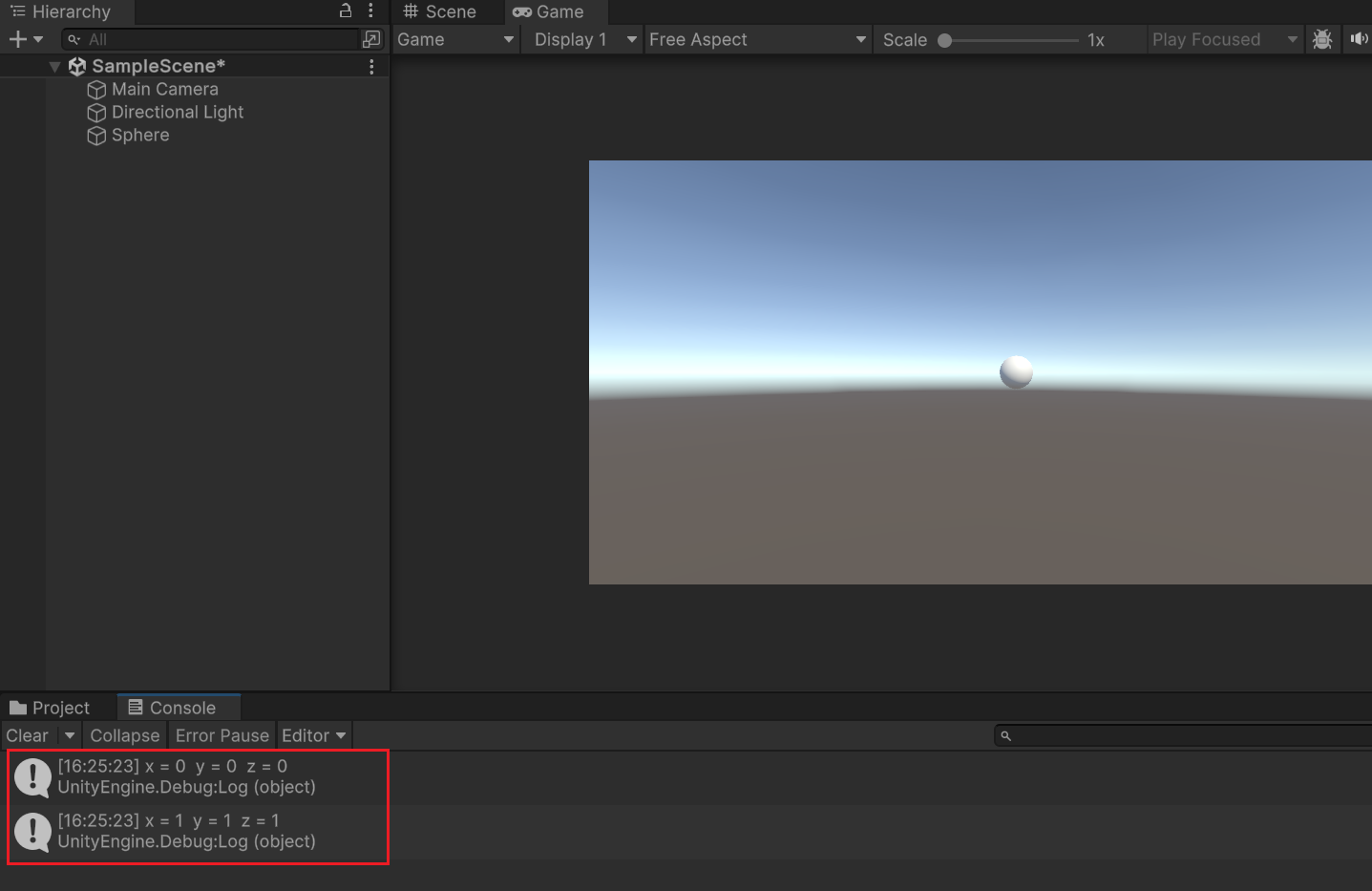
The Vector3 is a value type. When we amend ‘v1’, the ‘transform.position’ does not change as well. Because the Vector3 is a value type, these two variables correspond to different storage spaces.

**Task3.3**



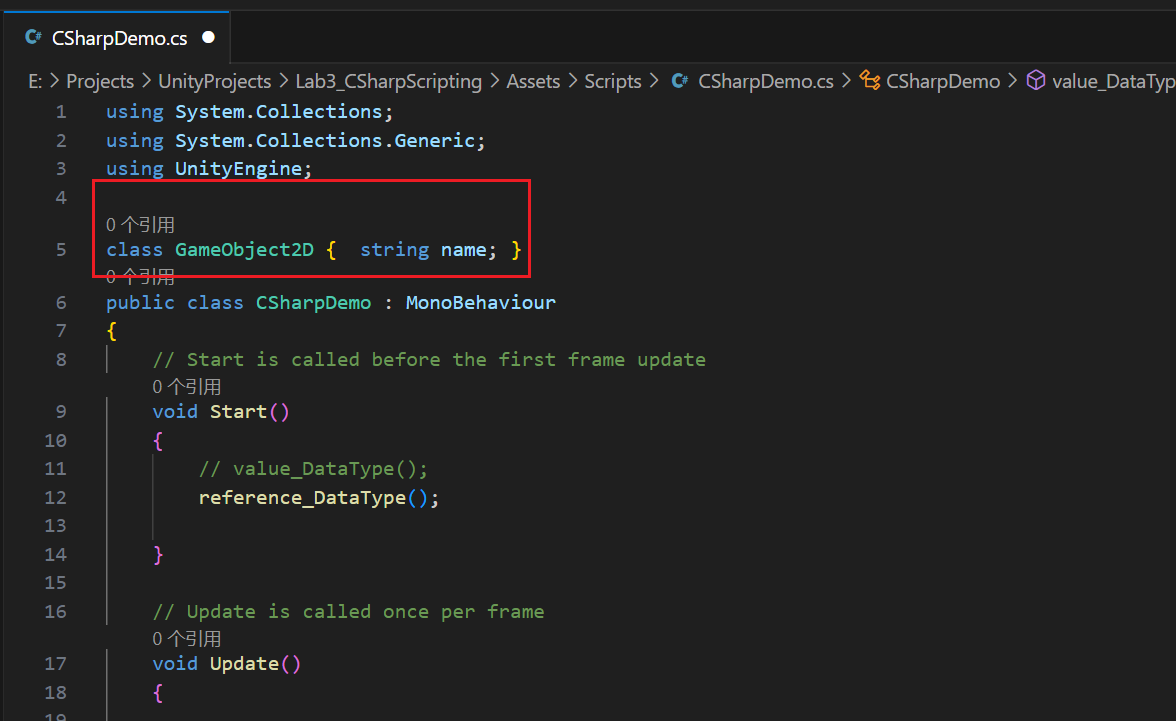
The following figure shows the printout in the Console window. The sphere changes its position.

Because, the ‘Transform’ is a reference type. These two variables, ‘transform’ and ‘trans’ correspond to the same storage space. When we amend ‘trans’ (by changing its ‘position), the ‘transform.position’ changes as well.

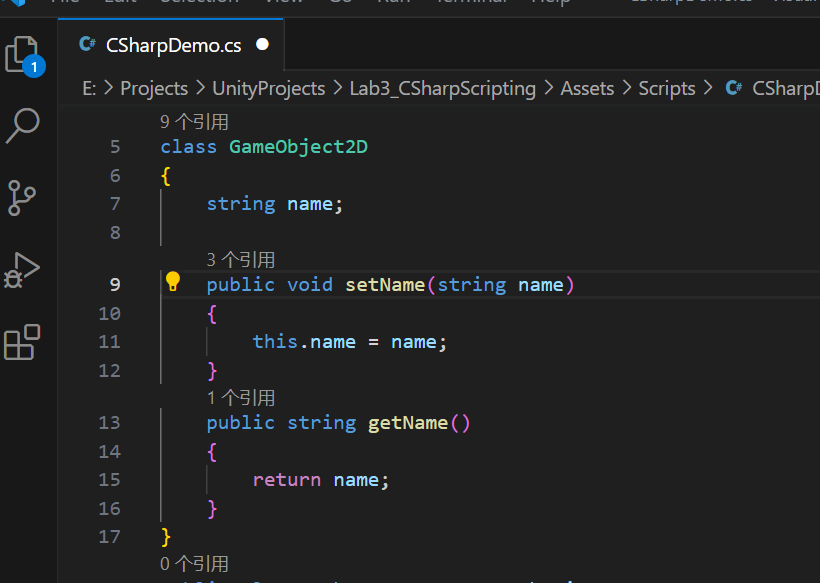


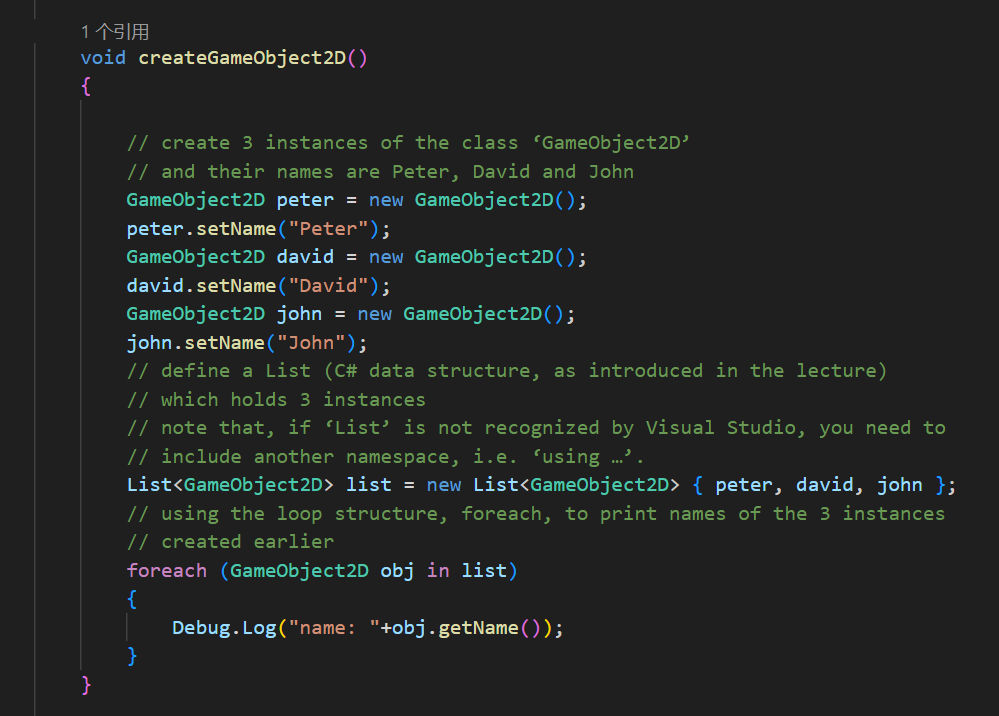
**Task3.4**

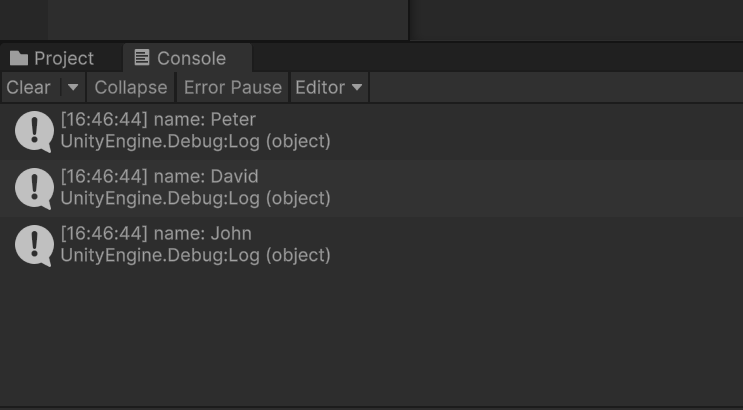
The attribute 'name' in the class 'GameObject2D' has default accessibility, which is private.



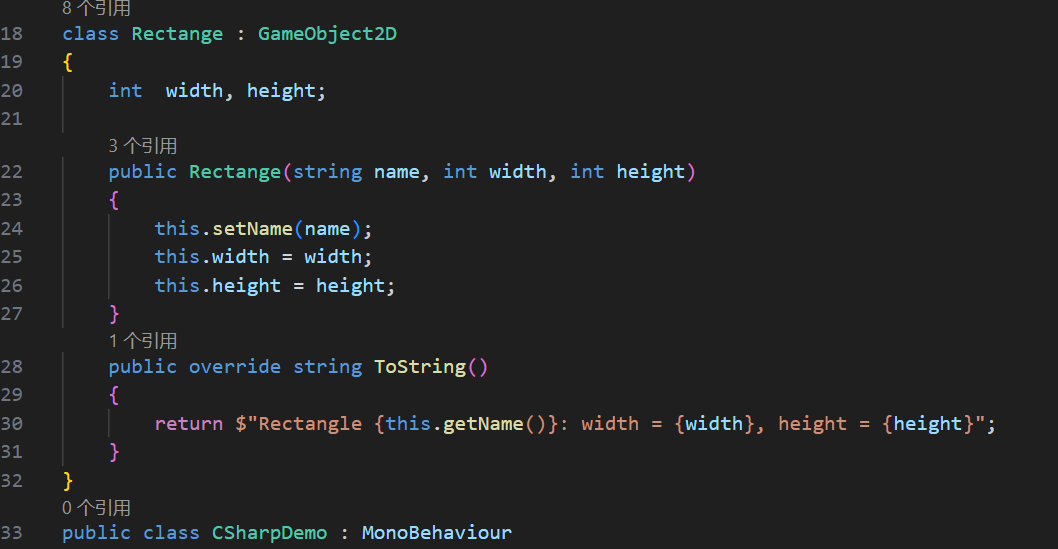
**Task3.5**







**Task3.6**



**Task3.7**



