2D Programming
GPR103
Pattern Example : Singleton

Singleton Pattern

The **singleton pattern** is a design **pattern** that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system.

- Similar to static variables.
- The singleton pattern gives us the benefit of global access to variables and the simplicity of a single copy of the class.
- Commonly used for scripts that manage data and don't require multiple instances.
- Like a game manager that holds the player score.

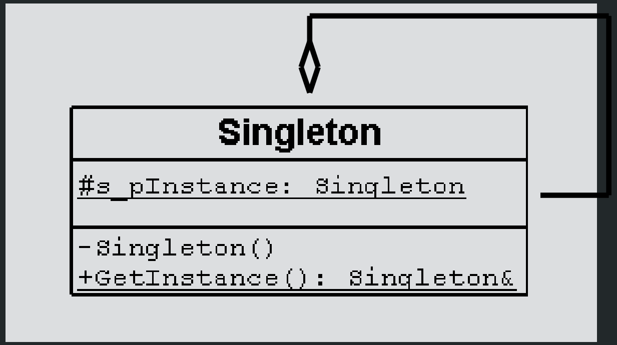
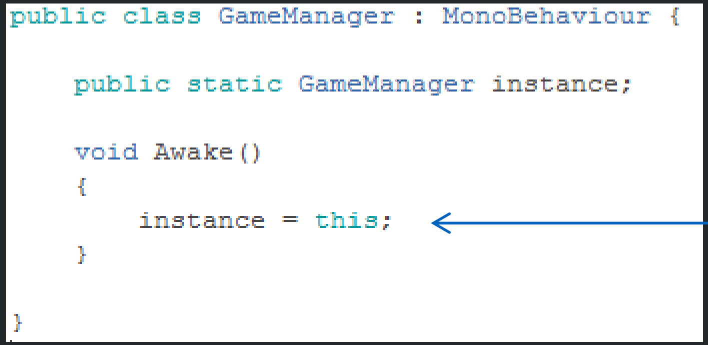


Fig. 9: UML representation of a singleton.

Singleton Pattern

- With a singleton pattern in Unity, we can only have 1 instance of a class.
- Because this is a design pattern, Unity won't be upset if we have more then 1 copy, but it will cause us grief if we do.

- A simple implementation of the pattern. (A proper, more complicated but **safer** implementation is on next slide.)
- If we put the following script on a gameObject in our scene.

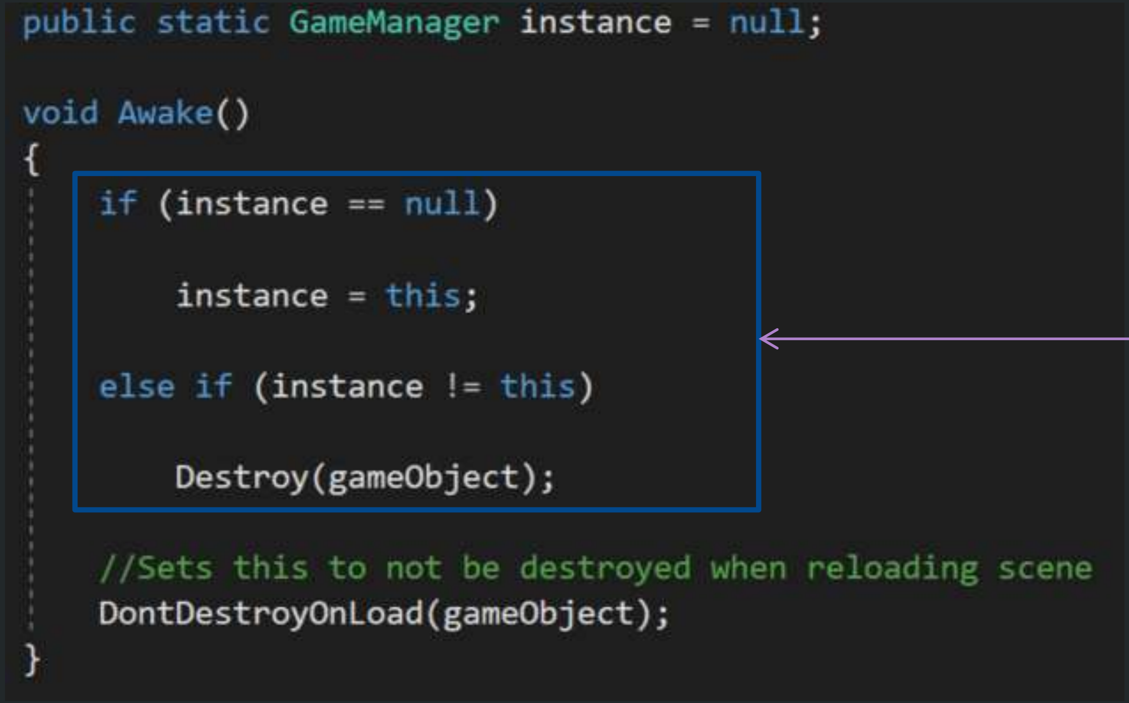


```
public class GameManager : MonoBehaviour {

    public static GameManager instance;

    void Awake()
    {
        instance = this;
    }

}
```

this – refers to 'this' instance of the script.

Singleton Pattern

A safer implementation.

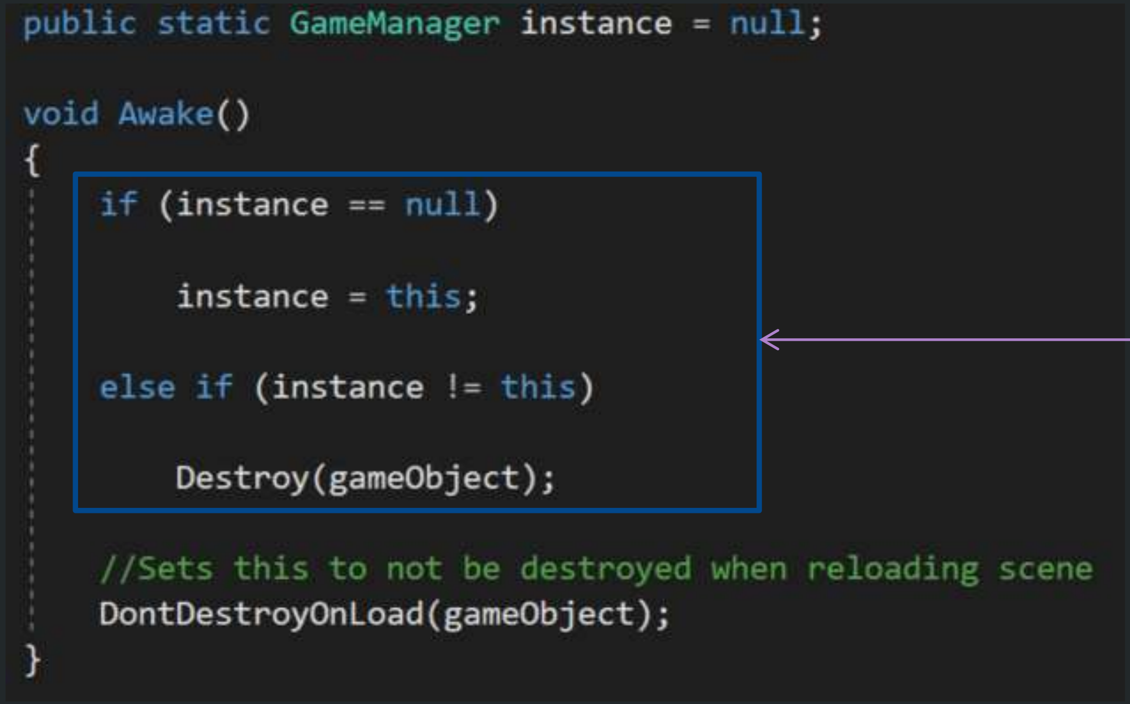


```
public static GameManager instance = null;

void Awake()
{
    if (instance == null)

        instance = this;

    else if (instance != this)

        Destroy(gameObject);

    //Sets this to not be destroyed when reloading scene
    DontDestroyOnLoad(gameObject);
}
```

Only allows 1 instance of the script to be in the Unity scene.

Singleton Pattern

A safer implementation.



```
public static GameManager instance = null;

void Awake()
{
    if (instance == null)

        instance = this;

    else if (instance != this)

        Destroy(gameObject);

    //Sets this to not be destroyed when reloading scene
    DontDestroyOnLoad(gameObject);
}
```

Only allows 1 instance of the script to be in the Unity scene.

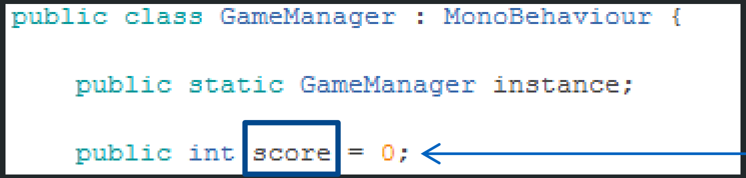
See if you can modify this to hide the public instance.

Singleton Pattern Usage

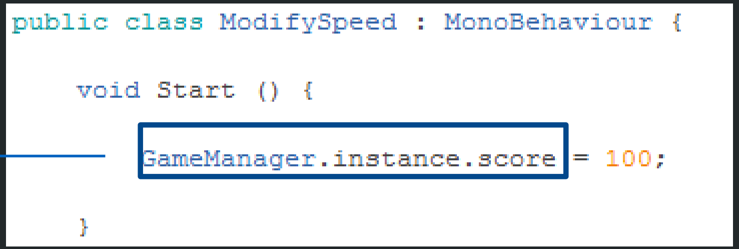
We can use the **instance** reference to access the script in the scene from any other script.

GameManger.instance

So if we added a public score variable, we could then access that from another script.



```
public class GameManager : MonoBehaviour {

    public static GameManager instance;

    public int score = 0;
```



```
public class ModifySpeed : MonoBehaviour {

    void Start () {

        GameManager.instance.score = 100;

    }
```