NAMING RULES:

1.Do use PascalCasing for class names and method names

```
public class CodeStyle : MonoBehaviour
{
    Ссылок: 0
    private void CheckCodeStyle()
    {
        Debug.Log("Code style is correct");
    }
}
```

2.Do use camelCasing for method arguments and local variables

```
public class CodeStyle : MonoBehaviour
{
    Ссылок: 0
    private void CheckCodeStyle(CurrentStyle currentStyle)
    {
        int linesCount = currentStyle.LinesCount;
    }
}
```

3.Do not use Hungarian notation or any other type identification in identifiers

```
//Correct
private int _itemsAmount;
private string _playerName;

//Avoid
private int _iCounter;
private int _strPlayerName;
```

4.Do not use Screaming Caps for constants or readonly variables, name it in same way with common fields, but always with the first letter in

uppercase

```
//Correct
public readonly string CompanyName = "Nesquik";
private const string _ShippingType = "DropShipping";
private readonly string _ShippingDate = "Today";

//Avoid
public readonly string COMPANYNAME = "Nesquik";
private const string SHIPPINGTYPE = "DropShipping";
private readonly string SHIPPINGDATE = "Today";
```

5.Avoid using Abbreviations. Exceptions: abbreviations commonly used as names, such as Id, Xml, Ftp, Uri.

```
//Correct
private int _usersAmount;
private string _employeeAssignment;

//Avoid
private int _usrsAmnt;
private string _empAssignment;

//Exceptions
private int _usersId;
private string _resourceUri;
```

6.Do use PascalCasing for abbreviations 3 characters or more (2 chars are both uppercase)

```
//3 characters abbreviations
private int _htmlHelper;
private string _ftpTransfer;

//2 characters abbreviations
public int UIController;
private int _uiController;
```

7.Do not use Underscores in identifiers. Exception: prefix private fields with an underscore

```
//Correct
private DateTime _appearanceDate;
private TimeSpan _timeLeft;

//Avoid
private DateTime appearance_Date;
private TimeSpan time_Left;
```

8.Avoid of using public fields, better to use properties

```
//Correct
Ссылок: 0
public DateTime AppearanceDate { get; set; }
Ссылок: 0
public int AnimationDuration => 5;
private string _userName;

//Avoid
public TimeSpan TimeLeft;
public string TeamName;
```

9.Do use implicit type var for local variable declarations when it use word new or Factory Method . Exception: primitive types (int, string, double, etc) use predefined names

```
//Correct
var stream = File.Create(_path);
var style = new CurrentStyle();

//Avoid
var fullness = GetSomething();

//Exceptions
int index = 100;
string name = "Android";
bool isCompleted = true;
```

10.Do use noun or noun phrases to name a class

```
Ссылок: 0
public class Employee
{
}

Ссылок: 0
public class BusinessLocation
{
}

Ссылок: 0
public class Documentation
{
}
```

11.Do prefix interfaces with the letter I. Interface names are noun (phrases) or adjectives.

```
Ссылок: 0
public interface IEnemy
{
}

Ссылок: 0
public interface IDestroyable
{
}

Ссылок: 0
public interface IShapeCollection
{
}
```

12.Do name source files according to their main classes.

```
public class Rules
{
}
```

13.Do not use partial classes.

```
//Avoid
Ссылок: 0
public partial class Rules
{
}
```

14.Do organize namespaces with a clearly defined structure

```
namespace Company.Product.Module.SubModule
{
}

namespace EJawSDK.Authorization.Data
{
}

namespace EJawSDK.Competition.View
{
}
```

15.Do vertically align curly brackets

```
public class CodeStyle : MonoBehaviour
{
    private string _path;

    Ссылок: 0
    public partial class Rules
    {
    }

    Ссылок: 0
    public interface IEnemy
    {
    }
```

16.Do put the curly braces on a separate line. Here's a sample

```
public void DoSomething(bool isFinished)
{
    if (isFinished)
    {
        //code
    }
    else
    {
        //code
    }
}
```

17.Do add the curly braces even if you have only one line run.

```
//Correct
if (isFinished)
{
    //code
}

for (int i = 0; i < 1; i++)
{
    //code
}

//Avoid
if (isFinished)
    Debug.Log("here");

for (int i = 0; i < 1; i++)
    Debug.Log("here");
```

18. Group the code blocks by logic, separating them from the rest with an empty line

```
string userName = data.username;
//One empty line
if (data.id == ApplicationManager.NewUserData.id)
{
    if (data.already_changed_username == false)
    {
        userName = "you";
    }
}
//One empty line
_name.text = $"{userName}";
_starsAmount.text = $"{data.stars}";
_place.text = $"{place}";
```

19. Group fields by access modifiers

```
public int _amount;
public int _duration;

private string _companyName;
private string _sourceName;

[SerializeField] private Transform _root;
[SerializeField] private GameObject _prefab;
```

20. Separate methods with an empty line

```
Ссылок: 0
protected void Get()
{
}

Ссылок: 0
private void Set()
{
}

Ссылок: 0
private void Take()
{
}
```

21. One file - one class

22.Sort by access modifiers public events ,public, internal, protected, private,

[SerializeField] private, constructors, properties

```
public event Action LevelFinished;

public int Amount;
internal int Duration;
protected int _startIndex;

private const int _speedMultiplier = 2;
private readonly int _name;

private int _currentDay;

[SerializeField] private GameObject _prefab;

Ссылок: 0
public CodeStyle()
{
}

Ссылок: 0
public int CurrentDay { get; set; }
```

23.Do not create names of parameters in methods (or constructors) which differ only by the register:

```
public void DoSomething(bool isFinished, bool IsFinished)
{
}
```

24.Do use suffix Exception at creation of the new classes comprising the information on exception:

```
public class BarcodeReadException : System.Exception
{
}
```

25.Events naming rules:
-1.Do name events with a verb or a verb phrase.

```
public event Action Finished;
public event Action Painting;
public event Action DroppedDown;
```

-2.Do NOT use "Before" or "After" prefixes or postfixes to indicate pre- and post-events. Use present and past tenses as just described.

3.Do give events names with a concept of before and after, using the present and past tenses.

```
public event Action Finished;
public event Action Finishing;
```

26.Do use prefix On when handle event

```
public event Action Finished;

Сообщение Unity | Ссылок: 0
private void OnEnable()
{
    Finished += OnFinished;
}

Сообщение Unity | Ссылок: 0
private void OnDisable()
{
    Finished -= OnFinished;
}

Ссылок: 2
public void OnFinished()
{
}
```

27.Delete all unused code and usings, use usings sorting
28.Explicitly specify access modifiers

```
//Correct
private int _speedMultiplier = 2;
private int _name;

//Avoid
int _currentDay;
int _startIndex;
```

29.Save comments ONLY if you will need it later, otherwise delete it. Your code should be self - documented
30. Use sealed if class will not be inherited

31. Do use prefix On and postfix Click when handle button click event

```
[SerializeField] private Button _startLevelButton;

Сообщение Unity | Ссылок: 0
private void OnEnable()
{
    _startLevelButton.onClick.AddListener(OnStartLevelButtonClick);
}

Сообщение Unity | Ссылок: 0
private void OnDisable()
{
    _startLevelButton.onClick.RemoveListener(OnStartLevelButtonClick);
}

Ссылок: 2
public void OnStartLevelButtonClick()
{
}
```