
# OFFICIAL CODING STANDARDS FOR TEAM DEEPPURPLE

---

## 1. Naming:

### i) Classes and Methods:

a. Class and Method names should follow standard PascalCasing notation.
b. Class and Method names should not be abbreviated or shortened.

Example:

```
public class AnimationController
{
    public void CalculateFrames()
    {
        //...
    }
    public void CalculateAcceleration()
    {
       //...
    }
}
```

### ii) Variables and Arguments:

a. Local variables, static variables and Arguments should use camelCasing notation.
b. Constant variable names should all be uppercase.
c. If two or more words are used for naming a constant, they should be separated by an underscore '_'.
d. Any kind of abbreviation is not allowed.

Example:

```
public class PlayerStats
{
    public void GetDamage(int damageHits)
    {
        int damageCount = damageHits.Count;
        static int healthCounter = 6;
```

```
        // ...
    }
}

public class Calculator
{
    public const double PI = 3.14159;
    public const double GOLDEN_RATIO = 1.618;
    // ...
}
```

iii) **Files**:

a. File names should follow PascalCasing notation followed by the uppercased initials of team leads and an underscore '_'.
b. No abbreviation of actual filename is allowed.

Example:

*GH_PlayerHud.cs*

*MA_ClassDiagram.png*

*KH_GanttChart.xlsx*

---

## 2. Commenting:

a. For every script there should be a multi-line comments on top stating file name, name of programmer, and a brief description of script.

Example:

```
/* GH_InventorySlot.cs
 * Programmer: Gabriel Hasen
 * Description: Adds and removes item to and from the inventory,
   updates inventory items.
 */

using UnityEngine;
using UnityEngine.UI;

public class InventorySlot : MonoBehaviour
{
    //...
}
```

b. Single line comments should be used inside classes and methods.

Example:

```
public void UseItem()
{
    // Check if the item is null
    if(item != null)
    {
        item.Use();
    }
    //...
}
```

c. Trailing single or multiple line comments are not allowed inside a method or a function.

Example:

```
public void UseItem()
{
   if(item != null)      // Check if the item is null
   {
       item.Use();
   }
   //...
}
```

---

## 3. Alignment and Brackets:

*a. Vertically aligned curly brackets should be used to declare scope of code.*
*b. There should not be more than two lines of spaces between lines of code.*
*c. Tab size should be set to 4.*

*Example:*

```
if(True)
{
    //...
}
else
{
    //...
}
```

## 4. Exception Handling:

Error handling blocks should be implemented using the try, catch, and finally keywords. At the end we throw the exception. C# provides a structured solution to the exception handling in the form of try and catch blocks. Using these blocks the core program statements are separated from the error-handling statements.

Example:

```
try
{
   // statements causing exception
}
catch( ExceptionName e1 )
{
   // error handling code
}
catch( ExceptionName e2 )
{
   // error handling code
}
catch( ExceptionName eN )
{
   // error handling code
}
finally
{
   // statements to be executed
}
throw (new ExceptionName("Exception found"))
```

References:

https://www.tutorialspoint.com/csharp/csharp_exception_handling.htm