**Одиночка** — это порождающий паттерн, который гарантирует существование только одного объекта определённого класса, а также позволяет достучаться до этого объекта из любого места программы.

Одиночка имеет такие же преимущества и недостатки, что и глобальные переменные. Его невероятно удобно использовать, но он нарушает модульность вашего кода.

**Применимость:** Многие программисты считают Одиночку антипаттерном, поэтому его всё реже и реже можно встретить в C#-коде.

**Признаки применения паттерна:** Одиночку можно определить по статическому создающему методу, который возвращает один и тот же объект.

using System;

namespace Singleton

{

// Класс Одиночка предоставляет метод `GetInstance`, который ведёт себя как

// альтернативный конструктор и позволяет клиентам получать один и тот же

// экземпляр класса при каждом вызове.

class Singleton

{

// Конструктор Одиночки всегда должен быть скрытым, чтобы предотвратить

// создание объекта через оператор new.

private Singleton() { }

// Объект одиночки храниться в статичном поле класса. Существует

// несколько способов инициализировать это поле, и все они имеют разные

// достоинства и недостатки. В этом примере мы рассмотрим простейший из

// них, недостатком которого является полная неспособность правильно

// работать в многопоточной среде.

private static Singleton \_instance;

// Это статический метод, управляющий доступом к экземпляру одиночки.

// При первом запуске, он создаёт экземпляр одиночки и помещает его в

// статическое поле. При последующих запусках, он возвращает клиенту

// объект, хранящийся в статическом поле.

public static Singleton GetInstance()

{

if (\_instance == null)

{

\_instance = new Singleton();

}

return \_instance;

}

// Наконец, любой одиночка должен содержать некоторую бизнес-логику,

// которая может быть выполнена на его экземпляре.

public static void someBusinessLogic()

{

// ...

}

}

class Program

{

static void Main(string[] args)

{

// Клиентский код.

Singleton s1 = Singleton.GetInstance();

Singleton s2 = Singleton.GetInstance();

if (s1 == s2)

{

Console.WriteLine("Singleton works, both variables contain the same instance.");

}

else

{

Console.WriteLine("Singleton failed, variables contain different instances.");

}

}

}

}