ООП в C#

(Распространенные интерфейсы .NET, статические методы и классы, синглтон)

Артём Трофимушкин

Интерфейс IDisposable

Интерфейс IDisposable предоставляет механизм для освобождения неуправляемых ресурсов. Например, при работе с файловой системой.

```
public class DisposableSample : IDisposable
{
    // ... some logic here

    public void Dispose()
    {
        // TODO: free unmanaged resources (unmanaged objects)
        // TODO: set large fields to null.
    }
}
```

Конструкция using

Вместо явного использования метода Dispose() можно воспользоваться конструкцией using

```
var sample = new DisposableSample ();
// some work with sample object here
sample.SomeMethod();
// disposing after finish
sample.Dispose();
```

Лучше использовать конструкцию using:

```
using (var sample = new DisposableSample())
{
    // some work with sample object here
    sample.SomeMethod();
} // Dispose() will be automatically called!
```

Самостоятельная работа

Написать класс ErrorList, который будет хранить тип ошибок определённой категории, реализующий интерфейс IDisposable. Он должен содержать следующие публичные члены:

- Свойство Category типа string - категория ошибок, свойство read-only, задаётся из конструктора,
- Свойство Errors типа List<string> - собственно список ошибок,
- Конструктор, в котором будут инициализироваться свойство Category (через параметр category) и пустой список строк Errors,
- Метод void Dispose(), в котором будет происходить 1) очистка списка методом Clear() и 2) приравнивание свойства Errors к null.

В основном потоке программы создать экземпляр объекта ErrorList используя конструкцию using, и написать пару ошибок.
Затем вывести их на экран в формате: “категория: ошибка”.

Интерфейс IEnumerable<T>

Интерфейс IEnumerable<T> предоставляет возможность получить доступ для чтения типизированной коллекции.

```
public class EnumberableSample : IEnumerable<int>
{
    // some logic here

    public IEnumerator<int> GetEnumerator()
    {
        throw new NotImplementedException();
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        throw new NotImplementedException();
    }
}
```

Интерфейс IEnumerable<T>

Предоставляет перечислитель, который поддерживает простой перебор элементов в указанной коллекции.

Требует реализации двух перегрузок метода GetEnumerator().

Благодаря его реализации мы можем перебирать значения последовательности в цикле foreach:

```
var sample = new EnumerableSample();
foreach(int number in sample)
{
    Console.WriteLine(number);
}
```

Самостоятельная работа

Добавить к классу ErrorList реализацию интерфейса IEnumerable<string>.
Также добавить метод void Add(string errorMessage)
Изменить публичное свойство Errors, сделав его внутренним полем класса, с соответствующими переименованиями.

В основном потоке программы обновить код добавления новых ошибок через новый метод Add. Обновить вывод ошибок на экран в формате: “категория: ошибка” через использование foreach по экземпляру класса.

Другие распространённые интерфейсы

ICloneable
IComparable<T>
ICollection<T>
IEquatable<T>
IList<T>
IReadOnlyCollection<T>
IReadOnlyList<T>
IServiceProvider

И много чего ещё можно найти внутри пространств имен System и System.Collections.Generic

Статические члены в обычных классах

Кроме обычных полей, методов, свойств класс может иметь статические поля, методы, свойства. Статические поля, методы, свойства относятся ко всему классу.

Для обращения к подобным членам класса не обязательно создавать экземпляр класса.

Статические члены могут иметь доступ к другим членам класса, только если они тоже статические.

На уровне памяти для статических полей будет создаваться участ ок в памяти, который будет общим для всех объектов класса. При этом память для статических переменных выделяется даже в том случае, если не создано ни одного объекта этого класса.

Статические конструкторы

Кроме обычных конструкторов у класса также могут быть статические конструкторы. Статические конструкторы имеют следующие отличительные черты:

- Статические конструкторы не должны иметь модификатор доступа и не принимают параметров,
- Как и в статических методах, в статических конструкторах нельзя использовать ключевое слово this для ссылки на текущий объект класса и можно обращаться только к статическим членам класса,
- Статические конструкторы нельзя вызвать в программе вручную. Они выполняются автоматически при самом первом создании объекта данного класса или при первом обращении к его статическим членам (если таковые имеются)

Статические члены в обычных классах

```
public class StaticMembersContainer
{
      private static int staticField;

      public static int StaticProperty
      {
            get { return staticField; }
            set { staticField = value; }
      }

      public int RegularProperty { get; set; }

      public static void StaticMethod(int valueToAdd)
      {
            StaticProperty += valueToAdd;
      }

      public int RegularMethod()
      {
            return StaticProperty + RegularProperty;
      }
}
```

```
class Program
{
      static void Main(string[] args)
      {
            StaticMembersContainer.StaticProperty = 10;

            var regularInstance =
                  new StaticMembersContainer();
            regularInstance.RegularProperty = 20;

            Console.WriteLine(
                  regularInstance.RegularMethod());

            StaticMembersContainer.StaticMethod(15);

            Console.WriteLine(
                  regularInstance.RegularMethod());
      }
}
```

Самостоятельная работа

Добавьте в класс ErrorList

- публичное статическое свойство OutputPrefixFormat типа string, которое будет позволять просматривать и задавать префиксную строку для вывода
- статический конструктор в котором бы определялось начальное значение свойства OutputPrefixFormat как дата и время по заданному формату согласно примеру:
  - `April 7, 2019 (7:55 PM)`
- обычный публичный метод void WriteToConsole(), который бы выводил в консоль все сообщения об ошибках вместе с префиксом, который бы формировался через DateTime.Now.ToString(OutputPrefixFormat)

Статические классы

Статические классы объявляются с модификатором static и могут содержать только статические поля, свойства и методы.

Например, если бы класс ErrorList имел бы только статические переменные, свойства и методы, то его можно было бы объявить как статический.

В C# показательными примерами статических классов являются классы

- Console, который используется для вывода данных в консоль
- Math, который применяется для различных математических операций

Статические классы

```
public static class StaticSample
{
      public static string StaticPropertySample
      {
            get; set;
      }

      static StaticSample()
      {
            StaticPropertySample = "Test String";
      }

      public static void StaticMethod()
      {
            InternalStaticMethod();
      }

      public static void InternalStaticMethod()
      {
            Console.WriteLine(StaticPropertySample);
      }
}
```

```
static void Main(string[] args)
{
      StaticSample.StaticMethod();

      StaticSample.StaticPropertySample =
            "New String!";
      StaticSample.StaticMethod();
}
```

Паттерн Синглтон (Singleton)

Одиночка (Singleton, Синглтон) - порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа.

Когда надо использовать Синглтон? Когда необходимо, чтобы для класса существовал только один экземпляр

Синглтон позволяет создать объект только при его необходимости. Если объект не нужен, то он не будет создан. В этом отличие синглтона от глобальных переменных.

Классическая реализация синглтона

```
public class SingletonDemo
{
      private static SingletonDemo instance;

      private SingletonDemo()
      { }

      public static SingletonDemo GetInstance()
      {
            return instance ??
                (instance = new SingletonDemo());
      }
}
```

```
static void Main(string[] args)
{
      SingletonDemo singleton =
            SingletonDemo.GetInstance();

      SingletonDemo singleton2 =
            SingletonDemo.GetInstance();

      Console.WriteLine(
            singleton2.Equals(singleton));

      // True
}
```

Реализация паттерна Синглтон

Чтобы реализовать данный паттерн необходимо выполнить три задачи:

1. Закрыть возможность создавать экземпляры класса через конструктор
   - Для этого мы определяем единственный конструктор с сигнатурой такой же, как и у конструктора по-умолчанию, но сделав его private.
2. Определить закрытый (внутренний) статический член того же типа, что и сам класс
   - Это и будет наш единственный экземпляр, который мы будем всем отдавать через метод, реализованный ниже.
3. Реализовать публичный статический метод, который будет возвращать наш единственный внутренний экземпляр класса
   - Поскольку при самом первом обращении внутренний экземпляр ещё не создан, необходимо там проверять его на null и создавать его (там-то конструктор будет работать). Уже после этого возвращать

Домашнее задание

Реализовать паттерн синглтон в классах с прошлого домашнего задания:

- ConsoleLogWriter
- FileLogWriter
- MultipleLogWriter

Спасибо за внимание.