ООП в C#

(абстрактные классы, интерфейсы)

Андрей Голяков

Абстрактные классы

Если базовый класс используется лишь для унификации между собой дочерних классов, но не планируется быть использованным в логике программы, то его можно сделать абстрактным.

Это запретит создавать экземпляры таких классов, сохранив все прелести наследования.

Можно и не делать класс абстрактным, а просто не создавать его экземпляров, в результате мы получим одно и то же, однако явное определение класса как абстрактного более четко описывает назначение этого класса – быть базовым для других классов.

Пример выделения абстрактного класса

Рассмотрим выделение абстрактного класса на примере заготовленного когда для классной работы L13_C01_abstract_source. В нём есть три основных функциональных класса и один вспомогательный для работы с консолью.

Мы последовательно проведём анализ кода и выделим общие методы в базовый класс. В результате мы получим дополнительный класс, который содержит общую для производных классов логику.

Поскольку этот класс не является функциональным, т.е. мы не планируем создавать его экземпляры, мы можем подчеркнуть его назначение как абстрактного с помощью ключевого слова abstract.

Результат, к которому мы должны прийти: L13_C02_abstract_class

Самостоятельная работа

Создайте два класса: **Helicopter** (вертолёт) и **Plane** (самолёт).

Самостоятельно определив пересечение членов и оформив его в виде абстрактного класса.

Задание выдаётся в распечатанном виде, так как оно достаточно объёмно для единственного слайда.

Пример решения находится в проекте классной работы с названием L13_C02_abstract_class_SW.

Абстрактные члены классов

Кроме обычных свойств и методов абстрактный класс может иметь абстрактные члены классов, которые определяются с помощью ключевого слова abstract и не имеют никакого функционала. В частности, абстрактными могут быть как методы, так и свойства.

Абстрактный метод создается с помощью указываемого модификатора типа abstract.

У абстрактного метода отсутствует тело, и поэтому он не реализуется в базовом классе.

Это означает, что он должен быть переопределен в производном классе с помощью модификатора override.

Нетрудно догадаться, что абстрактный метод автоматически становится виртуальным и не требует указания модификатора virtual. В действительности совместное использование модификаторов virtual и abstract считается ошибкой. Схема определения абстрактного метода:

```
abstract тип имя(список_параметров);
```

Неабстрактные члены (абстрактных классов)

В абстрактные классы вполне допускается (и часто практикуется) включать конкретные методы, которые могут быть использованы в своем исходном виде в производном классе, а переопределению в производных классах подлежат только те методы, которые объявлены как abstract.

Самостоятельная работа

На примере классов Helicopter и Plane выделите абстрактный метод, который можно вынести в абстрактный класс.

Также в базовом классе создайте новую версию этого метода с суффиксом “2” и сделайте его не абстрактным, а как виртуальным.
Переопределите его реализацию в производных классах на использование метода базового класса (с добавлением дополнительных действий).

Пример решения находится в проекте классной работы с названием L13_C03_abstract_members_SW.

Интерфейсы в наследовании

Интерфейс (interface) представляет собой не более чем просто именованный набор абстрактных членов.

Абстрактные методы являются чистым протоколом, поскольку не имеют никакой стандартной реализации. Конкретные члены, определяемые интерфейсом, зависят от того, какое поведение моделируется с его помощью.

Интерфейс описывает поведение, которое данный класс или структура поддерживает.

Каждый класс (или структура) может поддерживать столько интерфейсов, сколько необходимо, и, следовательно, тем самым поддерживать множество поведений.

Интерфейсы в наследовании

Пример описания интерфейса

```
public interface IDemoExample
{
    string ExampleProperty { get; set; }

    void ExampleMethod();
}
```

Пример имплементации интерфейса

```
public class Demo: IDemoExample
{
    public string ExampleProperty { get; set; }

    public void ExampleMethod() { Console.WriteLine("Demo"); }
}
```

Абстрактные классы vs Интерфейсы

Вся разница между интерфейсами и абстрактными классами заключается в том, что абстрактные классы могут содержать логику “по умолчанию” которую наследуют дочерние классы (и могут переопределить).

Самостоятельная работа

Какие интерфейсы мы можем выделить внутри классов Plane и Helicopter?

Самостоятельная работа

IFlyingObject

- MaxHeight
- CurrentHeight
- TakeUpper
- TakeLower

IHelicopter

- BladesCount

IPlane

- EnginesCount

IPropertiesWriter

- WriteAllProperties()

Самостоятельная работа

Напишем интерфейсы и обновим классы с учетом реализуемых интерфейсов.

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.

Домашнее задание

Описать интерфейс **ILogWriter** с методами

- void **LogInfo**(string message)
- void **LogWarning**(string message)
- void **LogError**(string message)

Написать два класса, реализующих этот интерфейс и имеющие необходимые конструкторы:

- **FileLogWriter** - для записи логов в файл
- **ConsoleLogWriter** - для вывода логов в консоль

Формат одной записи лога - в одной строке:

- YYYY-MM-DDTHH:MM:SS+0000 <tab> MessageType <tab> Message

, где MessageType может быть “Info”, “Warning” или “Error”.

Написать класс третий класс: **MultipleLogWriter**, также унаследованный от ILogWriter, который бы принимал в конструкторе коллекцию классов, реализующих интерфейс ILogWriter, и мог бы писать логи сразу всеми переданными в конструктор log-writer-ами одновременно.

В основном потоке программы:

- Создать по одному экземпляру класса FileLogWriter и ConsoleLogWriter.
- Затем создать экземпляр класса MultipleLogWriter, который бы принял в конструктор созданные выше экземпляры FileLogWriter и ConsoleLogWriter.
- Сделать по одной записи логов каждого типа, чтобы убедиться, что они одновременно пишутся и в консоль и в файл.

Домашнее задание* (сложное)

Выделите общие части реализации интерфейса и вынесите их в абстрактный класс AbstractLogWriter: ILogWriter.

Добавьте его как базовый класс для всех трёх классов

- ConsoleLogWriter,
- FileLogWriter,
- MultipleLogWriter.

Реализация этих классов должна существенно сократиться.

Спасибо за внимание.