Темы урока

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.

Обычно абстрактный класс выявляется в процессе анализа сущностей, которыми оперирует программа. Становится видно, что для описания нескольких сущностей имеют что-то общее, однако логически не могут выступить в качестве базового класса друг для друга. В таком случае, чтобы не унифицировать код и не нарушать принцип DRY, можно вынести общую часть в базовый абстрактный класс.

Пример

Рассматриваем проект **L13_C01_abstract_source**

В нём есть три основных функциональных класса, один вспомогательный для работы с консолью.

У них у всех есть нечто их объединяющее, а именно:

- private поле _writer
- public свойство MusicSource
- public метод PlayMusic()

Конструктор похож, но не в точности такой же, внутри там разный текст.

Но обычное наследование здесь трудно применить, так как эти классы с точки зрения логики не могут являться наследниками друг друга.

Однако, мы видим, что общая часть есть - это публичные члены, если попробовать осмыслить эту группу членов как сущность, то получится что-то вроде “Музыкальный проигрыватель”. Т.е. функция воспроизведения музык из какого либо источника есть как в телефоне, так и в видеоплеере, и в радио-записывающем устройстве.

Выделим общие члены в нашу общую сущность (думаю, вы догадались, к чему я клоню :) назовём её сразу AbstractMusicPlayer.

Конструктор можно также сразу перенести, так как он есть у всех, только удалим уникальную часть – оставим только логику инициализации ConsoleWriter, ну и получается, сам закрытый _writer тоже там будет, так как он необходим функции PlayMusic().

Проставим наш новый класс как базовый для наших трёх классов.

(можно заметить, что пока мы не удалили публичные члены, они выглядят как сокрытые). Удаляем из этих классов публичные члены, которые есть в базовом, наследуем вызов конструктора.

Запускаем, результат работы ничем не отличается от первоначального, однако мы сократили количество повторяющегося кода и обеспечили единое место для внесения исправлений в функции музыкального плеера.

По логике программы нам не нужны экземпляры класса AbstractMusicPlayer и чтобы подчеркнуть это, мы можем добавить ему ключевое слово abstract.

Результат можно посмотреть в проекте **L13_C02_abstract_class**.

Самостоятельная работа (на печать)

Создайте два класса: Helicopter (вертолёт) и Plane (самолёт), самостоятельно определив пересечение членов и оформив его в виде абстрактного класса.

У классов Helicopter и Plane будет следующий набор одинаковых членов:

Тип члена	Сигнатура члена	Детали реализации
Свойство	int MaxHeight	*Максимально возможная высота над землёй в условных единицах высоты.* { get; private set }
Свойство	int CurrentHeight	*Текущая высота над землёй в условных единицах высоты.* { get; private set }
Метод	void TakeUpper(int delta)	Набрать высоту на определённое число единиц. Если delta <= 0 бросаем исключение ArgumentOutOfRangeException Если CurrentHeight + delta > MaxHeight, устанавливаем текущую высоту полёта в значение MaxHeight Если нет, обновляем текущую высоту полёта как сумму CurrentHeight и delta
Метод	void TakeLower(int delta)	Сбросить высоту на определённое число единиц. Если delta <= 0 бросаем исключение ArgumentOutOfRangeException Если CurrentHeight - delta > 0, устанавливаем текущую высоту полёта в значение CurrentHeight равное значению новой высоты Если CurrentHeight - delta = 0, устанавливаем текущую высоту полёта в 0 Если < 0, бросаем исключение типа InvalidOperationException с текстом “Crash!”

Уникальные члены вертолёта:

Тип члена	Сигнатура члена	Детали реализации
Свойство	byte BladesCount	*Число лопастей винта вертолёта.* { get; private set }
Конструктор	Helicopter(int maxHeight, byte bladesCount)	*Конструктор нового вертолёта.* Свойство MaxHeight инициализируется параметром maxHeight, Свойство BladesCount инициализируется параметром bladesCount, Свойство CurrentHeight инициализируется нулём, В консоль выводится строка “It’s a helicopter, welcome aboard!”
Метод	void WriteAllProperties()	*Вывод текущих параметров воздушного судна в формате:* Properties of Helicopter: BladesCount: XXX CurrentHeight: YYY MaxHeight: ZZZ

Уникальные члены самолёта:

Тип члена	Сигнатура члена	Детали реализации
Свойство	byte EnginesCount	*Число двигателей самолёта.* { get; private set }
Конструктор	Plane(int maxHeight, byte enginesCount)	*Конструктор нового вертолёта.* Свойство MaxHeight инициализируется параметром maxHeight, Свойство EnginesCount инициализируется параметром enginesCount, Свойство CurrentHeight инициализируется нулём, В консоль выводится строка “It’s a plane, welcome aboard!”
Метод	void WriteAllProperties()	*Вывод текущих параметров воздушного судна в формате:* Properties of Helicopter: EnginesCount: XXX CurrentHeight: YYY MaxHeight: ZZZ

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:

- Методы
- Свойства

Фактически, эти методы и свойства являются неким контрактом, который все дочерние классы обязаны

Абстрактные члены классов **не должны иметь модификатор private**. При этом производный класс **обязан переопределить и реализовать все абстрактные методы и свойства**, которые имеются в базовом абстрактном классе.

При переопределении в производном классе такой метод или свойство также объявляются с модификатором **override** (как и при обычном переопределении виртуальных методов и свойств).

Также следует учесть, что если класс имеет хотя бы один абстрактный метод (или абстрактные свойство, индексатор, событие), то этот класс должен быть определен как абстрактный.

Пример

В нашем примере мы можем выделить один метод, который также есть у всех дочерних классов, но не имеет смысла быть описанным в базовом классе обычным способом.

Это метод Restart(), и это хороший кандидат, чтобы сделать такой метод абстрактным, обязуя все дочерние классы переопределить его с помощью ключевого слова **override**.

Также, если продолжать рефакторинг этого класса, можно улучшить ситуацию с закрытым членом типа ConsoleWriter. Сейчас он определен в абстрактном базовом классе классе как private, таким образом, мы не можем получить к нему доступ из класса наследника и нам приходится создавать свой собственный такой же и у наследника.

Если мы поменяем модификатор доступа к этому полю с **private** на **protected**, мы разрешим дочерним классам обращаться к нему изнутри.
После этого можно удалить скрывающие определения **_writer** в дочерних классах, а также убрать ставшую ненужной инициализацию в конструкторе.

Результат можно посмотреть в проекте **L13_C03_abstract_members**.

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

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.

Пример

Разделим по функциям все действия наших устройств и соответствующие им зависимые свойства по соответствующим интерфейсам.

Для определения интерфейсов:
- Используют ключевое слово interface,
- Членам не указывают модификаторы доступа,
- Методы и свойства не реализуются.

Определим интерфейсы с этими элементарными функциями.
- ICaller,
- IMusicPlayer,
- IMusicRecorder,
- IVideoPlayer,
- IRestartable

Теперь применим наследование интерфейсов к нашим классам.
- Реализовывать интерфейсы могут как обычные классы, так и абстрактные
- Можно реализовывать члены интерфейса абстрактными членами с последующей реализацией уже в наследниках.

После реализации интерфейсов в классе, можно создавать переменные типа интерфейса.
Присваивать таким переменным в качестве значений можно любые объекты, реализующие данный интерфейс.

Также можно применять к объектам операторы is и as.

Самостоятельная работа

Какие интерфейсы мы можем выделить внутри классов Plane и Helicopter?

- IFlyer
  - MaxHeight
  - CurrentHeight
  - TakeUpper
  - TakeLower
- IPropertiesWriter
  - GetAllProperties()

Напишем интерфейсы и обновим классы с учетом реализуемых интерфейсов.

Распространенные интерфейсы .NET

IDisposable

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

Пример: **L13_C05_interface_IDisposable**.

Конструкция using

Вместо явного использования метода Dispose() можно воспользоваться конструкцией using.

Пример: **L13_C06_interface_IDisposable_using**

Самостоятельная работа

Показать задание (на слайде).
Решение: **L13_C07_interface_IDisposable_using_SW**

IEnumerable

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.

Пример: **L13_C08_interface_IEnumerable**

Самостоятельная работа

Показать задание (на слайде).
Решение: **L13_C09_interface_IEnumerable_SW**

Домашнее задание

Базовый вариант

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.

Сложный вариант

Выделите общие части реализации интерфейса и вынесите их в абстрактный класс **AbstractLogWriter**: ILogWriter. Добавьте его как базовый для классов **ConsoleLogWriter**, **FileLogWriter**, **MultipleLogWriter**. Реализация конечных классов должна существенно сократиться.