**Группа** ИС 3-1

**ФИО:** Мозыркий А.Д.

**Курс:** C# Essetrional

**Урок 4:** Абстрактные классы и интерфейсы

**Теория**

* Абстракция — в объектно-ориентированном программировании это придание объекту характеристик, которые отличают его от всех других объектов, четко определяя его концептуальные границы. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации, в виде более простых объектов, подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации, в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня.
* Абстрактный класс в объектно-ориентированном программировании — базовый класс, который не предполагает создания экземпляров. Абстрактные классы реализуют на практике один из принципов ООП - полиморфизма. Абстрактный класс может содержать (и не содержать) абстрактные методы и свойства. Абстрактный метод не реализуется для класса, в котором описан, однако должен быть реализован для его неабстрактных потомков. Абстрактные классы представляют собой наиболее общие абстракции, то есть имеющие наибольший объем и наименьшее содержание.
* Ключевое слово abstract может использоваться с классами, методами, свойствами, индексаторами и событиями.
* Ключевое слово abstract при создании класса указывает, что класс предназначен только для использования в качестве базового класса для других классов. Члены, помеченные как абстрактные или включенные в абстрактный класс, должны быть реализованы с помощью классов, производных от абстрактных классов.
* Возможности абстрактных классов:
  1. Экземпляр абстрактного класса создать нельзя
  2. Абстрактные классы могут содержать как абстрактные, так и обыкновенные (неабстрактные) члены.
  3. Неабстрактный класс, являющийся производным от абстрактного, должен содержать фактические реализации всех наследуемых абстрактных методов и методов доступа.
* Возможности абстрактных методов:
  1. Абстрактный метод является неявным виртуальным методом.
  2. Создание абстрактных методов допускается только в абстрактных классах
  3. Тело абстрактного метода отсутствует; создание метода просто заканчивается двоеточием, а после сигнатуры ставить фигурные скобки ({ }) не нужно
  4. Реализация предоставляется методом переопределения override, который является членом неабстрактного класса.
* Абстрактный класс должен предоставлять реализацию для всех членов интерфейса.
* Абстрактный класс, реализующий интерфейс, может отображать методы интерфейса в абстрактных методах.
* Преимущества использования абстрактных классов:
  1. Общий код в одной реализации в виде конкретных и абстрактных членов
  2. Изменение значения полей или неабстрактных членов абстрактного класса приводит к соответствующему изменению во всех его производных классах.
  3. Наличие реализации по умолчанию.
* Интерфейс (от лат. inter — «между», и face — «поверхность») — семантическая и синтаксическая конструкция в коде программы, используемая для специфицирования услуг, предоставляемых классом или компонентом. Интерфейс определяет границу взаимодействия между классами или компонентами, специфицируя определенную абстракцию, которую осуществляет реализующая сторона. В отличие от многих других видов интерфейсов, интерфейс в ООП является строго формализованным элементом объектно-ориентированного языка и, в качестве семантической конструкции, широко используется кодом программы.
* Невозможно создать экземпляр интерфейса.
* Интерфейсы и члены интерфейсов являются абстрактными. Интерфейсы не имеют реализации по умолчанию.
* Интерфейс содержит только сигнатуры методов, свойств, событий или индексаторов.
* Члены интерфейсов автоматически являются открытыми, абстрактными, и они не могут иметь модификаторы доступа.
* Интерфейсы не могут содержать константы, поля, операторы, конструкторы экземпляров, деструкторы или типы.
* Класс или структура, которые реализуют интерфейс, должны реализовать члены этого интерфейса, указанные при его создании.
* Однако если базовый класс реализует интерфейс, производный также класс наследует эту реализацию.
* Интерфейс может наследоваться от одного или нескольких базовых интерфейсов.
* Базовый класс также может реализовать члены интерфейса с помощью виртуальных членов. В этом случае производный класс может изменить поведение интерфейса путем переопределения виртуальных членов.
* Если класс реализует два интерфейса, содержащих член с одинаковой сигнатурой, то при реализации этого члена в классе оба интерфейса будут использовать этот член для своей реализации.
* Если члены двух интерфейсов с одинаковой сигнатурой методов должны выполнять различные
* действия при их реализации, необходимо воспользоваться явной реализацией члена интерфейса — техникой явного указания в имени члена имени интерфейса, которому принадлежит данный член. Это достигается путем включения в имя члена класса имени интерфейса с точкой. Данный член в производном классе будет помечен по умолчанию как скрытый.
* Преимущество использования интерфейсов:
  1. Класс или структура может реализовать несколько интерфейсов.
  2. Если класс или структура реализует интерфейс, она получает только имена и сигнатуры метода
  3. Интерфейсы определяют поведение экземпляров производных классов
  4. Базовый класс может обладать ненужным функционалом, полученным от других его базовых классов, чего можно избежать, применяя интерфейсы
* Внедрение зависимостей (Dependency Injection, DI) — превосходная методика для создания слабосвязанных приложений. Она предоставляет возможности для упрощения кода, извлечения и обработки зависимостей между объектами и автоматического создания экземпляров зависимого объекта
* Внедрение зависимостей описывает процесс разработки приложений — вместо указания конкретных зависимостей в приложении во время разработки и создания необходимых объектов в коде во время выполнения приложение решает, какие объекты ему требуются, а потом создает и внедряет их в приложение.
* Использование внедрения зависимостей предоставляет несколько преимуществ:
  1. Ослабление связи между классами
  2. Создание кода, который лучше поддается проверке.
  3. Упрощение тестирования.

**Практика**

**Задание 1**

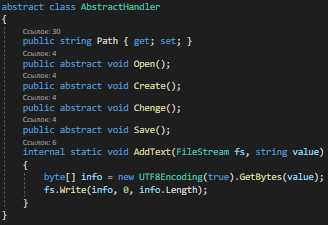


Рис.1.1 Реализация абстрактного класса AbstractHandler

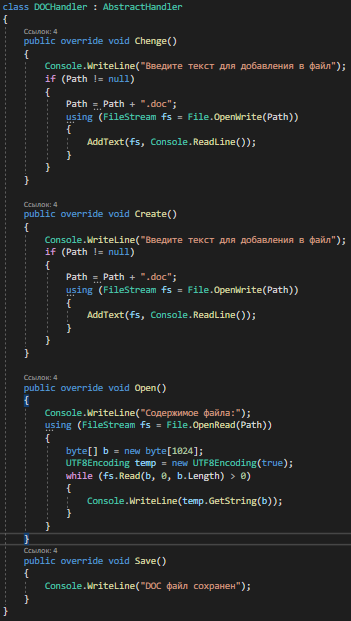
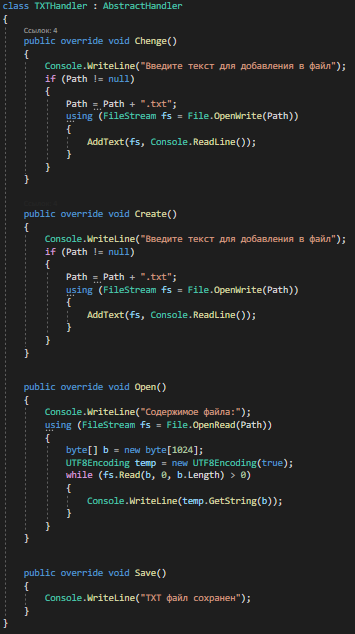
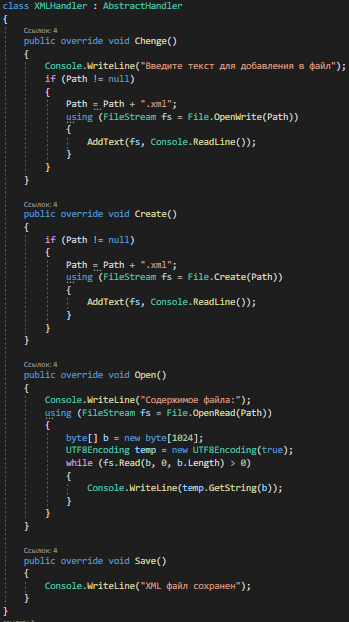


Рис.1.2 Реализация классов, реализующих абстрактный класс



Рис.1.3 Реализация кода программы

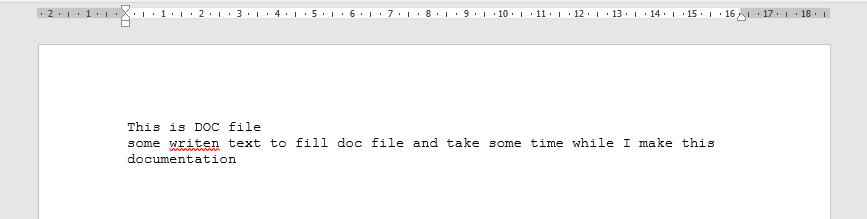
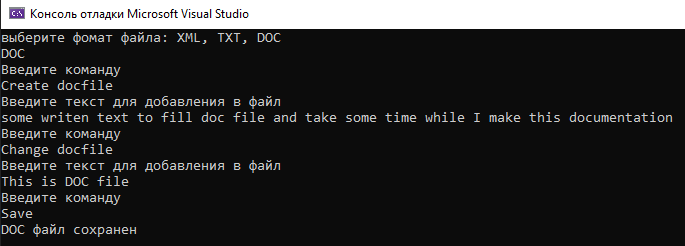


Рис.1.4 Результат работы программы

Результат работы программы соответствует ожидаемому.

**Задание 2**

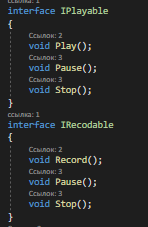
****

Рис.2.1 Создание интерфейсов IPlayable и IRecordable

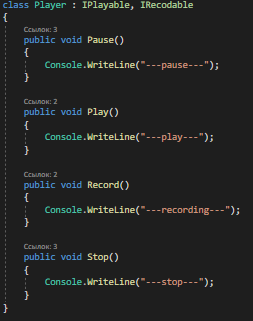


Рис.2.2 Реализация класса Player с использованием ранее созданных интерфейсов

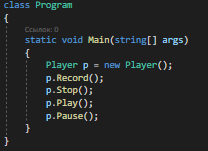


Рис.2.3 Реализация кода программы

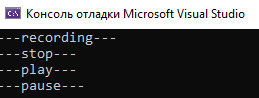


Рис.2.4 Результат работы программы

Результат работы программы соответствует ожидаемому.