# Лабораторная работа № 5 – Интерфейсы

**1 Цель работы**

Изучить основные принципы разработки интерфейсов в C#.

1. **Порядок выполнения работы**
   * прочесть краткие теоретические сведения;
   * выполнить задания раздела;
   * составить отчет по лабораторной работе и защитить его у преподавателя.
2. **Общая часть**

## 3.1 Основные понятия интерфейса

Интерфейс представляет ссылочный тип, который определяет набор методов и свойств, но не реализует их. Затем этот функционал реализуют классы и структуры, которые применяют данные интерфейсы.

В языке С++ есть возможность множественного наследования. Разработчики C# решили отказаться от этого и придумали интерфейсы. Получается, что класс не может быть унаследован от нескольких классов, но при этом он может унаследовать несколько интерфейсов.

Часто бывает необходимо реализовать несколько классов, при этом у них будут одинаковые методы (по названию!), но они по-разному должны быть реализованы.

В терминах ООП .NET интерфейс – это просто перечисление методов, которые должны быть обязательно реализованы у класса.

Интерфейс в практическом смысле дает возможность указать из чего именно должен состоять тот или иной объект разрабатываемой модели *без описания поведения объекта*.

То есть интерфейс позволяет в начале разработки описать основу разрабатываемой модели, определиться в понятиях, где и что должно быть, без описания поведения того или иного свойства или метода.

## Общий синтаксис интерфейса

Интерфейсы объявляются с помощью ключевого слова *interface*. Ниже приведена упрощенная форма объявления интерфейса:

**interface имя{    возвращаемый\_тип имя\_метода\_1 (список\_параметров);    возвращаемый\_тип имя\_метода\_2 (список\_параметров);    // …    возвращаемый\_тип имя\_метода\_N (список\_параметров);}**

где***имя*** — это конкретное имя интерфейса. В объявлении методов интерфейса используются только их возвращаемый тип и сигнатура. Они, по существу, являются абстрактными методами.

## Свойства интерфейсов

Интерфейс 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 выражает поведение, которое данный класс или структура может избрать для поддержки. Более того, каждый класс (или структура) может поддерживать столько интерфейсов, сколько необходимо, и, следовательно, тем самым поддерживать множество поведений.

В интерфейсе ни у одного из методов не должно быть тела. Это означает, что в интерфейсе вообще не предоставляется никакой реализации. В нем указывается только, что именно следует делать, но не как это делать. Как только интерфейс будет определен, он может быть реализован в любом количестве классов. Кроме того, в одном классе может быть реализовано любое количество интерфейсов.

Для реализации интерфейса в классе должны быть предоставлены тела (т.е. конкретные реализации) методов, описанных в этом интерфейсе. Каждому классу предоставляется полная свобода для определения деталей своей собственной реализации интерфейса. Следовательно, один и тот же интерфейс может быть реализован в двух классах по-разному. Тем не менее, в каждом из них должен поддерживаться один и тот же набор методов данного интерфейса. А в том коде, где известен такой интерфейс, могут использоваться объекты любого из этих двух классов, поскольку интерфейс для всех этих объектов остается одинаковым.

Благодаря поддержке интерфейсов в C# может быть в полной мере реализован главный принцип полиморфизма: *один интерфейс — множество методов*.

## Примеры реализации интерфейсов на C#

Начнем с классического примера о геометрических фигурах. Создадим интерфейс фигура с методом «Рассчитать площадь» и реализуем для разных типов фигур (прямоугольник, трапеция):

public interface IFigure

{

double GetArea();

}

public class Rectangle : IFigure

{

public int A;//длина

public int B;//ширина

public Rectangle(int a, int b)

{

A = a;

B = b;

}

public double GetArea()

{

return A \* B;

}

}

public class Trapeze : IFigure

{

public int A;

public int B;

public int h;//высота

public Trapeze(int a, int b, int h)

{

A = a;

B = b;

this.h = h;

}

public double GetArea()

{

return 0.5 \* (A + B) \* h;

}

}

Теперь можно создать метод, принимающий в качестве аргумента экземпляр любого класса-наследника:

public static void PrintArea(IFigure figure)

{

Console.WriteLine(figure.GetArea());

}

Интерфейсы удобны, например, когда заранее неизвестно, какой именно тип понадобится в будущем:

IFigure figure1;

if (...) figure1 = new Rectangle(4, 6);

else figure1 = new Trapeze(3, 5, 4);

Можно пойти дальше и создать коллекцию таких объектов:

var figureArray = new IFigure[2];

figureArray[0] = new Rectangle(4, 6);

figureArray[1] = new Trapeze(3, 5, 4);

В следующем примере вывернем предыдущий наизнанку, создав два интерфейса для разных фигур с одинаковыми методами (не забываем, что помимо методов, интерфейс может содержать и свойства). Применим множественное наследование, а также используем возможность реализовать в одном классе два интерфейса с одинаковыми по сигнатуре, но разными по содержанию методами:

public interface IRectangle

{

int A { get; set;}

int B { get; set;}

double GetArea();

}

public interface ITrapeze

{

int h { get; set; }

double GetArea();

}

public class AnyFigure : IRectangle, ITrapeze

{

public int A { get; set; }

public int B { get; set; }

public int h { get; set; }

double IRectangle.GetArea()

{

return A \* B;

}

double ITrapeze.GetArea()

{

return 0.5 \* (A + B) \* h;

}

}

Это называется явной реализацией интерфейса. Теперь можно построить такую конструкцию:

var figure1 = new AnyFigure {A = 3, B = 4, h = 3};

var s1 = ((IRectangle)figure1).GetArea();

var s2 = ((ITrapeze)figure1).GetArea();

Или использовать приведение типа для улучшения читаемости кода:

var figure = new AnyFigure {A = 3, B = 4, h = 3};

var case1 = (IRectangle)figure;

var case2 = (ITrapeze)figure;

var s1 = case1.GetArea();

var s2 = case2.GetArea();

1. **Общее задание для выполнения на паре**

Разработать приложение, реализующее логику компьютерной игры, в которой:

- Имеется базовый класс "Существо" для любого персонажа/существа (с полями "Количество здоровья", "Переносимый вес", "Кошелек", "Инвентарь" и т.д);

- Имеется базовый класс "Предмет" для любого предмета (с полями "Вес", "Цена" и т.д.);

- Интерфейс «Оружие» для предметов, имеющий метод «Получить урон» и возвращающий количество снимаемого за одну атаку здоровья;

- Имеется интерфейс "Боец" с методом "Атаковать", принимающим в качестве аргументов существо, на которое направлена атака и предмет, реализующий интерфейс «Оружие», которым производится атака;

- Имеется интерфейс "Торговец" с методами "Получить"/"Отдать", принимающими в качестве аргументов другого торговца-участника сделки и предмет.

При помощи комбинаций обозначенных сущностей реализовать несколько классов-наследников.

## Индивидуальные задания для выполнения работы

Доработать приложение из общего задания, добавив:

- Интерфейс «Помощь» для предметов, имеющий метод «Получить эффект» и возвращающий количество восстанавливаемого за одну сессию лечения здоровья;

- Интерфейс «Лекарь» с методом «Лечить», принимающим в качестве аргумента существо, на которое направлено лечение, и предмет, реализующий интерфейс «Помощь», которым ведется лечение;

- Интерфейс «Вор» с методом «Получить», имеющий такое же название и сигнатуру, что и в интерфейсе «Торговец», но иную реализацию.

При помощи комбинаций обозначенных сущностей реализовать несколько классов-наследников.