Лабораторная работа №4

РАСШИРЕННЫЕ ВОЗМОЖНОСТИ ПРОГРАММИРОВАНИЯ НА C#

Цель работы: познакомиться с расширенными возможностями языка программирования C#, такими как интерфейсы и делегаты.

Краткие теоретические сведения

Используя механизм наследования, возможно дополнять и переопределять общий функционал базовых классов в классах-наследниках. Однако, напрямую можно наследовать только от одного класса, в отличие, например, от языка C++, где имеется множественное наследование.

В языке C# подобную проблему позволяют решить интерфейсы. Они играют важную роль в системе ООП. Интерфейсы позволяют определить некоторый функционал, не имеющий конкретной реализации. Затем этот функционал реализуют классы, применяющие данные интерфейсы.

Для определения интерфейса используется ключевое слово **interface**. Как правило, названия интерфейсов в C# начинаются с заглавной буквы **«I»**, например, **IComparable**, **IEnumerable**, однако это не обязательное требование, а больше стиль программирования. Интерфейсы также, как и классы, могут содержать свойства, методы и события, только без конкретной реализации.

Определим следующий интерфейс **IAccount**, который будет содержать методы и свойства для работы со счётом клиента. Для добавления интерфейса в проект нужно нажать правой кнопкой мыши на проект и в появившемся контекстном меню выбрать **Add → New Item** и в диалоговом окне добавления нового компонента выбрать **Interface**.

Изменим пустой код интерфейса **IAccount** на следующий:

```
interface IAccount
{
    // Текущая сумма на счету
    int CurrentSum { get; }
    // Положить деньги на счёт
    void Put(int sum);
    // Взять со счёта
    void Withdraw(int sum);
    // Процент начислений
    int Percentage { get; }
}
```

У интерфейса методы и свойства не имеют реализации, в этом они сближаются с абстрактными методами абстрактных классов. Сущность данного интерфейса проста: он определяет два свойства для текущей суммы денег на счёте и ставки процента по вкладам и два метода для добавления денег на счёт и их изъятия.

При объявлении интерфейса все его члены (методы и свойства) не имеют модификаторов доступа, но по умолчанию у них доступ **public**, так как целью является определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации.

Применение интерфейса аналогично наследованию класса:

```
class Client : IAccount
{
    // реализация методов и свойств интерфейса
}
```

Так как клиент обладает счётом, реализуем интерфейс в классе **Client**:

```
class Client : IAccount
{
    int _sum; // Переменная для хранения суммы
    int _percentage; // Переменная для хранения процента

    public string Name { get; set; }
    public Client(string name, int sum, int percentage)
    {
        Name = name;
        _sum = sum;
        _percentage = percentage;
    }
    public int CurrentSum
    {
        get { return _sum; }
    }
    public void Put(int sum)
    {
        _sum += sum;
    }
    public void Withdraw(int sum)
    {
        if (sum <= _sum)
        {
            _sum -= sum;
        }
    }
```

```
    public int Percentage
    {
        get { return _percentage; }
    }
    public void Display()
    {
        Console.WriteLine("Клиент " + Name + " имеет счёт на сумму
" + _sum);
    }
}
```

Как и в случае с абстрактными методами абстрактного класса класс **Client** реализует все методы интерфейса. Поскольку все методы и свойства интерфейса являются публичными, то при их реализации в классе к ним можно применять только модификатор **public**. Поэтому если класс должен иметь метод с каким-то другим модификатором, например **protected**, то интерфейс не подходит для определения подобного метода.

Применение класса в программе:

```
Client client = new Client("Tom", 200, 10);
client.Put(30);
Console.WriteLine(client.CurrentSum); //230
client.Withdraw(100);
Console.WriteLine(client.CurrentSum); //130
```

Интерфейсы, как и классы, могут наследоваться:

```
interface IDepositAccount : IAccount
{
    void GetIncome(); // начисление процентов
}
```

При применении этого интерфейса класс **Client** должен будет реализовать как методы и свойства интерфейса **IDepositAccount**, так и базового интерфейса **IAccount**.

Задания на лабораторную работу

Реализуйте для иерархии из лабораторной работы №3 механизм интерфейсов, при этом один из классов должен реализовывать как минимум два интерфейса. Используйте для проверки всех методов данного класса многоадресный делегат.

Контрольные вопросы

1. Что понимается под термином «интерфейс»?
2. Чем отличается синтаксис интерфейса от синтаксиса абстрактного класса?

3. Какое ключевое слово языка C# используется для описания интерфейса?

4. Поддерживают ли реализацию методы интерфейса?

5. Какие объекты языка C# могут быть членами интерфейсов?

6. Каким количеством классов может быть реализован интерфейс?

7. Может ли класс реализовывать множественные интерфейсы?

8. Необходима ли реализация методов интерфейса в классе, включающем этот интерфейс?

9. Какой модификатор доступа соответствует интерфейсу?

10. Допустимо ли явное указание модификатора доступа для интерфейса?

11. Приведите синтаксис интерфейса в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.

12. Возможно ли создание ссылочной переменной интерфейсного типа?

13. Возможно ли наследование интерфейсов?

14. Насколько синтаксис наследования интерфейсов отличается от синтаксиса наследования классов?

15. Необходимо ли обеспечение реализации в иерархии наследуемых интерфейсов?

16. Что понимается под термином «делегат»?

17. В чём состоят преимущества использования делегатов?

18. В какой момент осуществляется выбор вызываемого метода в случае использования делегатов?

19. Что является значением делегата?

20. Какое ключевое слово языка C# используется для описания делегатов?

21. Приведите синтаксис делегата в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.

22. Возможно ли использование делегата для вызова метода, соответствующего подписи делегата?

23. Возможен ли вызов метода в том случае, если его подпись не соответствует подписи делегата?

24. Что понимается под термином «многоадресность»?

25. В чём состоит практическое значение многоадресности?