

**УО «МОГИЛЕВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ А.А. КУЛЕШОВА»
СОЦИАЛЬНО-ГУМАНИТАРНЫЙ КОЛЛЕДЖ**



**Дисциплина
«Конструирование программ и языки программирования»**

**Разработка программ, реализующих механизм наследования.
Полиморфизм
(4 часа)**

Методические рекомендации к лабораторной работе №10

Могилев 2017

Базовые понятия по объектно-ориентированному программированию. Понятия «наследование», «полиморфизм», «абстрактный класс/метод», «виртуальный метод». Методические указания по лабораторной работе №10 «Конструирование программ и языки программирования». Для учащихся очной формы обучения специальности 1–40 01 01 «Программное обеспечение информационных технологий».

Оглавление

1 Цель работы.....	4
2 Краткие теоретические сведения	5
2.1 Наследование	5
2.1.1 Доступ к членам базового класса из класса-наследника	6
2.1.2 Ключевое слово base.....	7
2.2 Конструкторы в производных классах	8
2.3 Полиморфизм. Переопределение методов	9
2.3.1 Ключевое слово base.....	12
2.3.2 Запрет переопределения методов.....	13
2.4 Абстрактные классы, методы и свойства в Си-шарп	13
2.4.1 Абстрактные свойства	14
3 Задания.....	16
4 Контрольные вопросы.....	21

1 Цель работы

1. изучение структуры иерархии классов и понятий наследования и полиморфизма в объектно-ориентированном программировании.
2. изучить описание и возможности абстрактных классов и функций в C#, их создание в C# и некоторые алгоритмы их обработки.

2 Краткие теоретические сведения

2.1 Наследование

Наследование (inheritance) является одним из ключевых моментов ООП. Его смысл состоит в том, что мы можем расширить функциональность уже существующих классов за счет добавления нового функционала или изменения старого. Пусть у нас есть следующий класс `Person`, описывающий отдельного человека:

```
class Person
{
    private string _firstName;
    private string _lastName;

    public string FirstName
    {
        get { return _firstName; }
        set { _firstName = value; }
    }
    public string LastName
    {
        get { return _lastName; }
        set { _lastName = value; }
    }
    public void Display()
    {
        Console.WriteLine(FirstName + " " + LastName);
    }
}
```

Но вдруг нам потребовался класс, описывающий сотрудника предприятия – класс `Employee`. Поскольку этот класс будет реализовывать тот же функционал, что и класс `Person`, так как сотрудник - это также и человек, то было бы рационально сделать класс `Employee` производным (или наследником, или подклассом) от класса `Person`, который, в свою очередь, называется базовым классом или родителем (или суперклассом):

```
class Employee:Person
{
}
```

После двоеточия мы указываем базовый класс для данного класса. Для класса `Employee` базовым является `Person`, и поэтому класс `Employee` наследует все те же свойства, методы, поля, которые есть в классе `Person`. Единственное, что не передается при наследовании, это конструкторы базового класса.

Таким образом, наследование реализует отношение *is-a* (является), объект класса `Employee` также является объектом класса `Person`:

```
static void Main(string[] args)
{
    Person p = new Person { FirstName = "Bill", LastName = "Gates" };
    p.Display();
    p = new Employee { FirstName = "Denis", LastName = "Ritchi" };
    p.Display();
    Console.Read();
}
```

И поскольку объект `Employee` является также и объектом `Person`, то мы можем так определить переменную: `Person p = new Employee()`.

Все классы по умолчанию могут наследоваться. Однако здесь есть ряд ограничений: Не поддерживается множественное наследование, класс может наследоваться только от одного класса. Хотя проблема множественного наследования реализуется с помощью концепции интерфейсов.

При создании производного класса надо учитывать тип доступа к базовому классу – тип доступа к производному классу должен быть таким же, как и у базового класса, или более строгим. То есть, если базовый класс у нас имеет тип доступа `internal`, то производный класс может иметь тип доступа `internal` или `private`, но не `public`.

Если класс объявлен с модификатором `sealed`, то от этого класса нельзя наследовать и создавать производные классы. Например, следующий класс не допускает создание наследников:

```
sealed class Admin
{
}
```

2.1.1 Доступ к членам базового класса из класса-наследника

Вернемся к нашим классам `Person` и `Employee`. Хотя `Employee` наследует весь функционал от класса `Person`, посмотрим, что будет в следующем случае:

```
class Employee : Person
{
    public void Display()
    {
        Console.WriteLine(_firstName);
    }
}
```

Этот код не сработает и выдаст ошибку, так как переменная `_firstName` объявлена с модификатором `private` и поэтому к ней доступ имеет только класс `Person`. Но зато в классе `Person` определено общедоступное свойство `FirstName`, которое мы можем использовать, поэтому следующий код у нас будет работать нормально:

```
class Employee : Person
```

```

{
    public void Display()
    {
        Console.WriteLine(FirstName);
    }
}

```

Таким образом, производный класс может иметь доступ только к тем членам базового класса, которые определены с модификаторами `public`, `internal`, `protected` и `protected internal`.

2.1.2 Ключевое слово `base`

Теперь добавим в наши классы конструкторы:

```

class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public Person(string fName, string lName)
    {
        FirstName = fName;
        LastName = lName;
    }
    public void Display()
    {
        Console.WriteLine(FirstName + " " + LastName);
    }
}
class Employee : Person
{
    public string Company { get; set; }

    public Employee(string fName, string lName, string comp):base(fName,
lName)
    {
        Company = comp;
    }
}

```

Класс `Person` имеет стандартный конструктор, который устанавливает два свойства. Поскольку класс `Employee` наследует и устанавливает те же свойства, что и класс `Person`, то логично было бы не писать по сто раз код установки, а как-то вызвать соответствующий код класса `Person`. К тому же свойств, которые надо установить, и параметров может быть гораздо больше.

С помощью ключевого слова `base` мы можем обратиться к базовому классу. В нашем случае в конструкторе класса `Employee` нам надо установить имя, фамилию и компанию. Но имя и фамилию мы передаем на установку в конструктор базового класса, то есть в кон-

структур класса `Person`, с помощью выражения `base (fName, lName)` .

```
static void Main(string[] args)
{
    Person p = new Person("Bill", "Gates");
    p.Display();
    Employee emp = new Employee ("Tom", "Simpson","SHC");
    emp.Display();
    Console.Read();
}
```

2.2 Конструкторы в производных классах

Конструкторы не передаются производному классу при наследовании. И если в базовом классе не определен конструктор по умолчанию без параметров, а только конструкторы с параметрами (как в случае с базовым классом `Person`), то в производном классе мы обязательно должны вызвать один из этих конструкторов через ключевое слово `base`. Например, из класса `Employee` уберем определение конструктора:

```
class Employee : Person
{
    public string Company { get; set; }
}
```

В данном случае мы получим ошибку, так как класс `Employee` не соответствует классу `Person`, а именно не вызывает конструктор базового класса. Даже если бы мы добавили какой-нибудь конструктор, который бы устанавливал все те же свойства, то мы все равно бы получили ошибку:

```
public Employee(string fName, string lName, string comp)
{
    FirstName = fName;
    LastName = lName;
    Company = comp;
}
```

То есть в классе `Employee` через ключевое слово `base` надо явным образом вызвать конструктор класса `Person`:

```
public Employee(string fName, string lName, string comp):base(fName,
lName)
{
    Company = comp;
}
```

Либо в качестве альтернативы мы могли бы определить в базовом классе конструктор

без параметров:

```
class Person
{
    // остальной код класса
    // конструктор по умолчанию
    public Person()
    {
        FirstName = "Tom";
        LastName = "Johns";
        Console.WriteLine("Вызов конструктора без параметров");
    }
}
```

Тогда в любом конструкторе производного класса, где нет обращения конструктору базового класса, все равно неявно вызывался бы этот конструктор по умолчанию. Например, следующий конструктор

```
public Employee(string comp)
{
    Company = comp;
}
```

Фактически был бы эквивалентен следующему конструктору:

```
public Employee(string comp):base()
{
    Company = comp;
}
```

2.3 Полиморфизм. Переопределение методов

Полиморфизм является третьим ключевым аспектом объектно-ориентированного программирования и предполагает способность к изменению функционала, унаследованного от базового класса. Полиморфизм предполагает определение полиморфного интерфейса в базовом классе – набор членов класса, которые могут быть переопределены в классе-наследнике. Методы, которые мы хотим сделать доступными для переопределения, в базовом классе помечаются модификатором `virtual`. Такие методы называют виртуальными. Они и представляют полиморфный интерфейс (также частью полиморфного интерфейса могут быть абстрактные члены класса).

При определении класса-наследника и наследовании методов базового класса мы можем выбрать одну из следующих стратегий:

- Обычное наследование всех членов базового класса в классе-наследнике
- Переопределение членов базового класса в классе-наследнике
- Скрытие членов базового класса в классе-наследнике

Первая стратегия довольно проста. Допустим, есть следующая пара классов `Person` и


Employee:

```
class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public Person(string lName, string fName)
    {
        FirstName = fName;
        LastName = lName;
    }
    public virtual void Display()
    {
        Console.WriteLine(FirstName + " " + LastName);
    }
}
class Employee : Person
{
    public string Company { get; set; }
    public Employee(string lName, string fName, string comp)
        :base(fName, lName)
    {
        Company = comp;
    }
}
```

В базовом классе Person метод Display() определен с модификаторами virtual, поэтому данный метод может быть переопределен. Но класс Employee наследует его как есть:

```
class Program
{
    static void Main(string[] args)
    {
        Person p1 = new Person("Bill", "Gates");
        p1.Display(); // вызов метода Display из класса Person
        Person p2 = new Employee("Tom", "Johns", "UnitBank");
        p2.Display(); // вызов метода Display из класса Person
        Employee p3 = new Employee("Sam", "Toms", "CreditBank");
        p3.Display(); // вызов метода Display из класса Person
        Console.Read();
    }
}
```

Консольный вывод:



```
Bill Gates
Tom Johns
Sam Toms
```

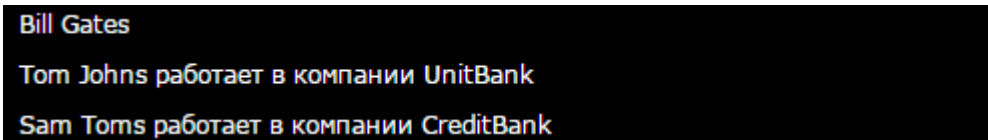
Вторая стратегия – переопределение методов базового класса в классе-наследнике предполагает использование ключевого слова `override`:

```
class Employee : Person
{
    public string Company { get; set; }
    public Employee(string lName, string fName, string comp)
        :base(fName, lName)
    {
        Company = comp;
    }
    public override void Display()
    {
        Console.WriteLine(FirstName + " " + LastName + " работает в ком-
пании "+ Company);
    }
}
```

Класс `Person` остается тем же, в нем так же метод `Display` объявляется как виртуальный. В этом случае поведение объекта `Employee` изменится:

```
Person p1 = new Person("Bill", "Gates");
p1.Display(); // вызов метода Display из класса Person
Person p2 = new Employee("Tom", "Johns", "UnitBank");
p2.Display(); // вызов метода Display из класса Employee
Employee p3 = new Employee("Sam", "Toms", "CreditBank");
p3.Display(); // вызов метода Display из класса Employee
```

Консольный вывод:



```
Bill Gates
Tom Johns работает в компании UnitBank
Sam Toms работает в компании CreditBank
```

При третьей стратегии можно просто определить в классе-наследнике метод с тем же именем, без переопределения с помощью слова `override`:

```
class Employee : Person
{
    public string Company { get; set; }
    public Employee(string lName, string fName, string comp)
        :base(fName, lName)
```

```

    {
        Company = comp;
    }
    public new void Display()
    {
        Console.WriteLine(FirstName + " " + LastName + " работает в ком-
пании "+ Company);
    }
}

```

В этом случае метод `Display()` в `Employee` скрывает метод `Display()` из класса `Person`. Чтобы явно скрыть метод из базового класса, используется ключевое слово `new`, хотя в принципе оно необязательно, по умолчанию система это делает неявно.


Использование в программе:

```

Person p1 = new Person("Bill", "Gates");
p1.Display(); // вызов метода Display из класса Person
Person p2 = new Employee("Tom", "Johns", "UnitBank");
p2.Display(); // вызов метода Display из класса Person
Employee p3 = new Employee("Sam", "Toms", "CreditBank");
p3.Display(); // вызов метода Display из класса Employee

```

Консольный вывод:



```

Bill Gates
Tom Johns
Sam Toms работает в компании CreditBank

```

2.3.1 Ключевое слово `base`

Кроме конструкторов, мы можем обратиться с помощью ключевого слова `base` к другим членам базового класса. В нашем случае вызов `base.Display()`; будет обращением к методу `Display()` в классе `Person`:

```

class Employee : Person
{
    public string Company { get; set; }
    public Employee(string lName, string fName, string comp)
        :base(fName, lName)
    {
        Company = comp;
    }
    public override void Display()
    {
        base.Display();
        Console.WriteLine("Место работы : " + Company);
    }
}

```

```
}  
}
```

2.3.2 Запрет переопределения методов

Также можно запретить переопределение методов и свойств. В этом случае их надо объявлять с модификатором `sealed`:

```
class Employee : Person  
{  
    public string Company { get; set; }  
  
    public Employee(string lName, string fName, string comp)  
        :base(fName, lName)  
    {  
        Company = comp;  
    }  
    public override sealed void Display()  
    {  
        base.Display();  
        Console.WriteLine("Место работы : " + Company);  
    }  
}
```

При создании методов с модификатором `sealed` надо учитывать, что `sealed` применяется в паре с `override`, то есть только в переопределяемых методах.

И в этом случае мы не сможем переопределить метод `Display` в классе, унаследованном от `Employee`.

2.4 Абстрактные классы, методы и свойства в Си-шарп

Абстрактный класс – это класс объявленный с ключевым словом `abstract`:

```
abstract class [имя_класса]  
{  
    //тело  
}
```

Такой класс имеет следующие особенности:

- нельзя создавать экземпляры (объекты) абстрактного класса;
- абстрактный класс может содержать как абстрактные методы/свойства, так и обычные;
- в классе наследнике должны быть реализованы все абстрактные методы и свойства, объявленные в базовом классе.

В самом по себе абстрактном классе, от которого никто не наследуется, смысла нет, так как нельзя создавать его экземпляры. В абстрактном классе обычно реализуется некоторая общая часть нескольких сущностей или другими словами - абстрактная сущность, которая, как объект, не может существовать, и эта часть необходима в классах наследниках. Конкретные примеры будут дальше.

Объявление абстрактного метода происходит при помощи ключевого слова `abstract`, и

при этом фигурные скобки опускаются, точка с запятой ставится после заголовка метода:

```
[модификатор доступа] abstract [тип] [имя метода] ([аргументы]);
```

Реализация абстрактного метода в классе наследнике происходит так же, как и переопределение метода – при помощи ключевого слова `override`:

```
[модификатор доступа] override [тип] [имя метода] ([аргументы])  
{  
    // реализация метода  
}
```

2.4.1 Абстрактные свойства

Создание абстрактных свойств не сильно отличается от методов:

```
protected [тип] [поле, которым управляет свойство];  
[модификатор доступа] abstract [тип] [имя свойства] { get; set; }
```

Реализация в классе-наследнике:

```
[модификатор доступа] override [тип] [имя свойства]  
{  
    get { тело аксесора get }  
    set { тело аксесора set }  
}
```

В качестве примера, приведу программу похожую на ту, которая была в предыдущем уроке о виртуальных методах, где выводилась информация о человеке/студенте/школьнике. Сейчас уже будут животные. Тогда мы могли создать человека без статуса (не студент, не школьник), у которого была некоторая информация, а сейчас у нас будет абстрактная сущность Животное, объект которой создавать нельзя и нет смысла, так как каждое животное будет конкретного подцарства – млекопитающее, рыба, птица:

```
abstract class Animal  
{  
    public string Name { get; set; }  
    public string Type { get; protected set; }  
  
    public abstract void GetInfo(); // объявление абстрактного  
    метода  
}  
class Parrot : Animal  
{  
    public Parrot(string name)  
    {  
        Name = name;  
        Type = "Птица";  
    }  
    public override void GetInfo() // реализация абстрактного  
    метода  
    {  
        Console.WriteLine("Тип: " + Type + "\n" + "Имя: " + Name +  
        "\n");  
    }  
}  
class Cat : Animal  
{  
    public Cat(string name)  
    {
```

```

        Name = name;
        Type = "Млекопитающее";
    }
    public override void GetInfo() // реализация абстрактного
метода
    {
        Console.WriteLine("Тип: " + Type + "\n" + "Имя: " + Name +
"\n");
    }
}
class Tuna : Animal
{
    public Tuna(string name)
    {
        Name = name;
        Type = "Рыба";
    }
    public override void GetInfo() // реализация абстрактного
метода
    {
        Console.WriteLine("Тип: " + Type + "\n" + "Имя: " +
Name+"\n");
    }
}
class Program
{
    static void Main(string[] args)
    {
        List<Animal> animals = new List<Animal>(); //список типа Animal
animals.Add(new Parrot("Кеша"));
animals.Add(new Cat("Пушок"));
animals.Add(new Tuna("Тёма"));

        foreach (Animal animal in animals)
            animal.GetInfo();

        Console.ReadKey();
    }
}

```

В итоге, мы все так же работаем с одним списком животных, и, вызывая один метод `GetInfo()`, мы получаем информацию о соответствующем животном.

При попытке создать объект абстрактного класса мы получим ошибку "Cannot create an instance of the abstract class or interface 'ConsoleApplication1.Animal'":

```
Animal animal = new Animal(); // ошибка
```

3 Задания

В заданиях требуется описать абстрактный базовый класс и производные от него и создать параметризованную коллекцию объектов производных классов. Обеспечить читабельный вывод полей классов на экран.

Используя механизм виртуальных методов, продемонстрировать единообразную работу с элементами коллекции.

Вариант 1

Создать абстрактный класс File, инкапсулирующий в себе методы Open, Close, Seek, Read, Write, GetPosition и GetLength. Создать производные классы MyDataFile1 и MyDataFile2— файлы, содержащие в себе данные некоторого определенного типа MyData1 и MyData2, а также заголовки, облегчающие доступ к этим файлам.

Создать класс Folder, содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода списка имен и длин файлов. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 2

Создать абстрактный класс Point (точка). На его основе создать классы ColoredPoint и Line. На основе класса Line создать класс ColoredLine и класс PolyLine (многоугольник). Все классы должны иметь виртуальные методы установки и получения значений всех координат, а также изменения цвета и получения текущего цвета.

Создать класс Picture, содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 3

Создать абстрактный класс Vehicle. На его основе реализовать классы Car (автомобиль), Bicycle (велосипед) и Lorry (грузовик). Классы должны иметь возможность задавать и получать параметры средств передвижения (цена, максимальная скорость, год выпуска и т.д.). Наряду с общими полями и методами, каждый класс должен содержать и специфичные для него поля.

Создать класс Garage, содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 4

Создать абстрактный класс Figure. На его основе реализовать классы Rectangle (прямоугольник), Circle (круг) и Trapezium (трапеция) с возможностью вычисления площади, центра тяжести и периметра.

Создать класс Picture, содержащий массив/параметризованную коллекцию объек-

тов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 5

Создать абстрактный класс Number с виртуальными методами, реализующими арифметические операции. На его основе реализовать классы Integer и Real.

Создать класс Series (набор), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 6

Создать абстрактный класс Body. На его основе реализовать классы Parallelepiped (прямоугольный параллелепипед), Cone (конус) и Ball (шар) с возможностью вычисления площади поверхности и объема.

Создать класс Series (набор), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 7

Создать абстрактный класс Currency для работы с денежными суммами. Определить в нем методы перевода в рубли и вывода на экран. На его основе реализовать классы Dollar, Euro и Pound (фунт стерлингов) с возможностью пересчета в центы и пенсы соответственно.

Создать класс Purse (кошелек), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода общей суммы, переведенной в рубли, и суммы по каждой из валют. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 8

Создать абстрактный класс Triangle (треугольник), задав в нем длину двух сторон, угол между ними, методы вычисления площади и периметра. На его основе создать классы, описывающие равносторонний, равнобедренный и прямоугольный треугольники со своими методами вычисления площади и периметра.

Создать класс Picture, содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка и получения суммарной площади. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 9

Создать абстрактный класс Solution (решение) с виртуальными методами вычисления корней уравнения и вывода на экран. На его основе реализовать классы Linear (линейное уравнение) и Square (квадратное уравнение).

Создать класс Series (набор), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 10

Создать абстрактный класс Function (функция) с виртуальными методами вычисления значения функции $y = f(x)$ в заданной точке x и вывода результата на экран. На его основе реализовать классы Ellipse, Hiperbola и Parabola.

Создать класс Series (набор), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 11

Создать абстрактный класс Triad (тройка) с виртуальными методами увеличения на 1. На его основе реализовать классы Date (дата) и Time (время).

Создать класс Memories, содержащий массив/параметризованную коллекцию пар (дата-время) объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка и выборки самого раннего и самого позднего событий. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 12

Создать абстрактный класс Sorting (сортировка) с идентификатором последовательности, виртуальными методами сортировки, получения суммы и вывода на экран. На его основе реализовать классы Choice (метод выбора) и Quick (быстрая сортировка).

Создать класс Series (набор), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода идентификаторов и сумм элементов каждого объекта списка, а также вывода общей суммы всех значений. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 13

Создать абстрактный класс Worker с полями, задающими фамилию работника, фамилии руководителя и подчиненных и виртуальными методами вывода списка обязанностей и списка подчиненных на экран. На его основе реализовать классы Manager (руководитель проекта), Developer (разработчик) и Coder (младший программист).

Создать класс Group (группа), содержащий массив/параметризованную коллекцию

объектов этих классов в динамической памяти. Предусмотреть возможность вывода всех объектов списка и выборки по фамилии с выводом всего дерева подчиненных. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 14

Создать абстрактный класс Progression (прогрессия) с виртуальными методами вычисления заданного элемента и суммы прогрессии. На его основе реализовать классы Linear (арифметическая) и Exponential (геометрическая).

Создать класс Series (набор), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка и вывода общей суммы всех прогрессий. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 15

Создать абстрактный класс Pair (пара значений) с виртуальными методами, реализующими арифметические операции, и методом вывода на экран. На его основе реализовать классы Money (деньги) и Complex (комплексное число).

В классе Money денежная сумма представляется в виде двух целых, в которых хранятся рубли и копейки соответственно. При выводе части числа снабжаются словами «руб.» и «коп.». В классе Complex предусмотреть при выводе символ мнимой части (i).

Создать класс Series (набор), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода объектов списка. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 16

Создать абстрактный класс Pair (пара значений) с виртуальными методами, реализующими арифметические операции. На его основе реализовать классы Fractional (дробное) и LongLong (длинное целое).

В классе Fractional вещественное число представляется в виде двух целых, в которых хранятся целая и дробная часть числа соответственно. В классе LongLong длинное целое число хранится в двух целых полях в виде старшей и младшей части.

Создать класс Series (набор), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможность вывода характеристик объектов списка и вывода общей суммы всех значений. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 17

Создать абстрактный класс Integer (целое) с символьным идентификатором, виртуальными методами, реализующими арифметические операции, и методом вывода на экран. На его основе реализовать классы Decimal (десятичное) и Binary (двоичное). Число представить в виде массива цифр.

Создать класс Series (набор), содержащий массив/параметризованную коллекцию

объектов этих классов в динамической памяти. Предусмотреть возможность вывода значений и идентификаторов всех объектов списка и вывода общей суммы всех десятичных значений. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 18

Описать абстрактный класс *Element* (элемент логической схемы), задав в нем числовой идентификатор, количество входов, идентификаторы присоединенных к нему элементов (до 10) и двоичные значения на входах и выходе. На его основе реализовать классы *AND* и *OR* — двоичные вентили, которые могут иметь различное количество входов и один выход и реализуют логическое умножение и сложение соответственно.

Создать класс *Scheme* (схема), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможности вывода характеристик объектов списка и вычисление значений, формируемых на выходах схемы по заданным значениям входов. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 19

Описать абстрактный класс *Trigger* (триггер), задав в нем идентификатор и двоичные значения на входах и выходах. На его основе реализовать классы *RS* и *JK*, представляющие собой триггеры соответствующего типа.

Создать класс *Register* (регистр), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможности вывода характеристик объектов списка, общего сброса и установки значений каждого триггера по заданным значениям входов. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

Вариант 20

Описать абстрактный класс *Element* (элемент логической схемы) задав в нем символьный идентификатор, количество входов, идентификаторы присоединенных к нему элементов (до 10) и двоичные значения на входах и выходе. На его основе реализовать классы *AND_NOT* и *OR_NOT* — двоичные вентили, которые могут иметь различное количество входов и один выход и реализуют логическое умножение с отрицанием и сложение с отрицанием соответственно.

Создать класс *Scheme* (схема), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти. Предусмотреть возможности вывода характеристик объектов списка и вычисление значений, формируемых на выходах схемы по заданным значениям входов. Написать демонстрационную программу, в которой будут использоваться все методы классов.

Дополнительное задание: дополнить класс методами сортировки по некоторому критерию, вывода в файл и считывания из файла.

4 Контрольные вопросы

1. Что понимается под термином «наследование»?
2. Какая классификация объектов соответствует наследованию?
3. Что общего имеет дочерний класс с родительским?
4. В чем состоит различие между дочерним и родительским классами?
5. Приведите синтаксис описания наследования классов в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.
6. Что понимается под термином «полиморфизм»?
7. Как можно запретить переопределение методов и свойств?
8. Что понимается под термином «полиморфизм»?
9. В чем состоит основной принцип полиморфизма?
10. В чем состоит значение основного принципа полиморфизма?
11. Какие механизмы используются в языке C# для реализации концепции полиморфизма?
12. Что понимается под термином «виртуальный метод»?
13. Какое ключевое слово языка C# используется для определения виртуального метода?
14. В чем состоит особенность виртуальных методов в производных (дочерних) классах?
15. Какие условия определяют выбор версии виртуального метода?
16. Какое ключевое слово (модификатор) языка C# используется для определения виртуального метода в базовом (родительском) классе?
17. Какое ключевое слово (модификатор) языка C# используется для определения виртуального метода в производном (дочернем) классе?
18. Какие модификаторы недопустимы для определения виртуальных методов?
19. Что означает термин «переопределенный метод»?
20. Что понимается под термином «абстрактный класс»?
21. В чем заключаются особенности абстрактных классов?
22. Какой модификатор языка C# используется при объявлении абстрактных методов?
23. Являются ли абстрактные методы виртуальными?
24. Возможно ли создание иерархии классов посредством абстрактного класса?
25. Возможно ли создание объектов абстрактного класса?
26. Приведите синтаксис абстрактного класса в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.