

Лабораторная работа №3.

Наследование. Инкапсуляция. Полиморфизм.

Цель: практически освоить основные принципы ООП.

Ход работы:

1 часть. Изучение принципов наследования и инкапсуляции.

1. Создать новое консольное приложение под именем: MyFruits.

2. В приложении создать класс «Fruct». Для этого в Solution Explorer на имени проекта нажать правую кнопку мыши → последовательно выбрать пункты Add → Class → дать имя новому классу Fruct.

3. Описание класса будет выглядеть следующим образом:

```
class Fruct
{
    //фрукт для продавца характеризуется:
    //где растёт
    private string where;
    //названием
    private string name;
    //ценой
    private double cost;
}
```

4. Опишем два класса – наследника (яблоко и ягода, у которых появляются дополнительные характеристики):

```
class Apple :Fruct
{
    //характеризуется сортом
    private string Sort;
}
class Berry:Fruct
{
    //характеризуется цветом
    private enum ColorBerry {Red = 1, Black = 2, Blue = 3, Creen = 4};
}
```

5. Применим принцип инкапсуляции для ограничения доступа к методам: для этого в классе фруктов допишем методы:

```
//установка имени фрукта
public void SetName(string name)
{
    this.name = name;
}
//установка цены фрукта
private void SetCost(double cost)
{
    this.cost = cost;
}
//установка поля: где растёт
```

```

protected void SetWhere(string where)
{
    this.where = where;
}
//вывод результата установок
public string WRTLN()
{
    return String.Format("name = {0}  where = {1}",name, where);
}
}

```

6. В классах-наследниках вызовем эти методы соответствующим образом (в конструкторах):

```

public Apple(string nameApp, /* double cost, */ string whereApp)
{
    SetName(nameApp);
    // при вызове этого метода будет ошибка, т.к. он недоступен
    SetCost(5.00);
    SetCommonName(whereApp);
}

public Berry(string name, string where)
{
    SetName("Это ягода");
    SetCommonName("куст");
}

```

7. В основной программе, в методе Main() создаем объект класса Apple:
 - набираем слово Apple,
 - на нем вызываем контекстное меню (правой кнопкой мыши)→ Resolve → using MyFructs.

8. Далее дописываем следующий код:

```

Apple a1 = new Apple("Это яблоко", "дерево");
Console.WriteLine(a1.WRTLN());
Berry b1 = new Berry("bb1", "bb2");
Console.WriteLine(b1.WRTLN());

```

9. Запускам проект.

10. Сделать выводы.

2 часть. Изучение принципа полиморфизма.

1. На примере использования виртуальных классов и спецификаторов: **virtual**, **override**, **new** (см. пример Virtuality).

2. На примере использования интерфейсов (см. примеры UsingInheritance, IStudent).

Теоретические сведения.

ООП поддерживает три основных принципа:

- инкапсуляция,
- полиморфизм,
- наследование.

Инкапсуляция

Инкапсуляция — основной принцип ООП.

Состоит из двух аспектов:

- объединение данных и функций по работе с этими данными в единую сущность (описание класса),

- контроль доступа к элементам этой сущности (использование модификаторов доступа).

Оба этих аспекта были подробно рассмотрены в предыдущей лабораторной работе.

Единственное, что осталось отметить – предоставление контроля над поведением объекта. Для этого вернемся к описанной абстракции «велосипед».

Если вы едите на велосипеде, то вам нужно поворачивать руль и крутить педали, и вы вряд ли знаете, какие подшипники используются для вращения колес (это тоже одно из свойств инкапсуляции).

Инкапсуляция дает контроль над поведением объекта, но не более. Изменить атрибуты объекта возможно через предназначенный для этого интерфейс. Обычно, если вы хотите поменять колесо на велосипеде, то вы можете использовать для этого мастерскую.

Обычно считается правильным все данные в классе делать закрытыми извне, и если их необходимо изменять, то для этого предоставляют специальные методы.

Наследование

Предположим, вы описываете все тот же велосипед. А через некоторое время потребовалось описание абстракции «горный велосипед». У вас есть два варианта: либо написать все описание с нуля (используя предыдущие описания), либо сказать, что горный велосипед – это в первую очередь велосипед, но с характерными деталями (см. рис. 1.).

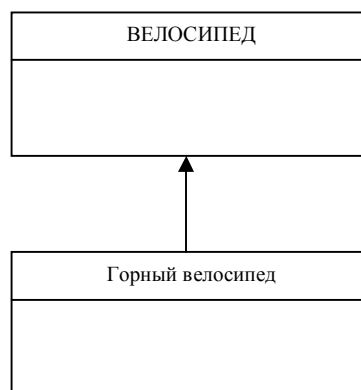


Рис. 1. Принцип наследования.

В C# не поддерживается множественное наследование! (см. пример InherSchema). Однако, сюда добавлены интерфейсы, что позволяет создавать свой класс, реализовав в нем более чем один интерфейс.

Полиморфизм

Полиморфизм (от греческого слова *polymorphism*, означающего "много форм") — это качество, которое позволяет одному интерфейсу получать доступ к целому классу действий.

Концепцию полиморфизма часто выражают такой фразой: "один интерфейс — много методов". Это означает, что для выполнения группы подобных действий можно разработать общий интерфейс. Полиморфизм позволяет понизить степень сложности программы, предоставляя программисту возможность использовать один и тот же интерфейс для задания *общего класса действий*. Конкретное (нужное в том или ином случае) действие (метод) выбирается компилятором. Программисту нет необходимости делать это вручную. Его задача — правильно использовать общий интерфейс.

Предположим, что мы создали класс музыканта, на основе которого построили два класса: гитарист и виолончелист (см. рис. 2.).

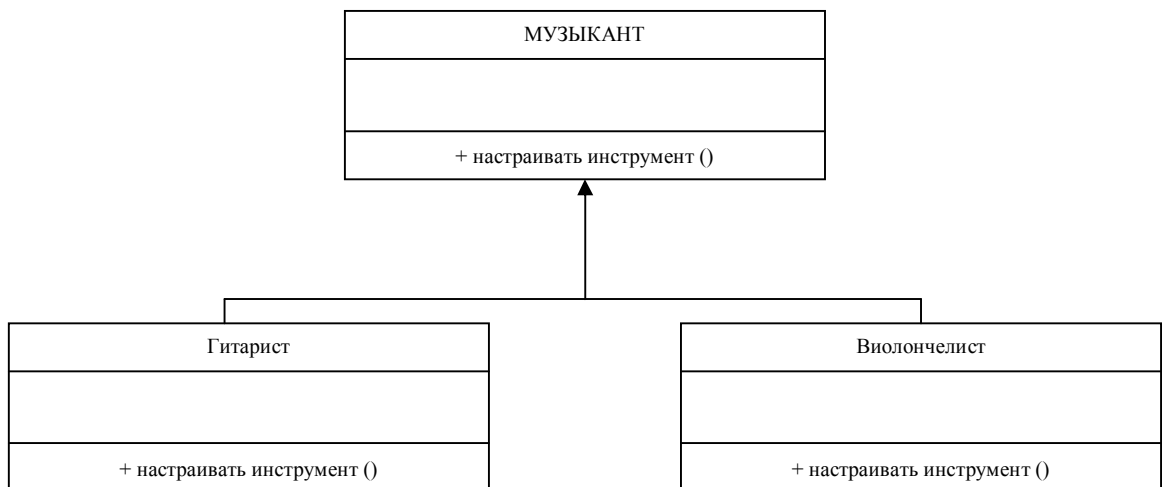


Рис. 2. Принцип полиморфизма

Все музыканты умеют настраивать свои инструменты. Если мы хотим целый оркестр настроить свои инструменты, то нам проще обратиться ко всем музыкантам одновременно, а не отдельно ко всем гитаристам и ко всем виолончелистам. Полиморфизм позволяет вызвать именно ту функцию, которая необходима. То есть все музыканты начинают настраивать свои инструменты тем способом, который подходит для их инструментов, а не тем, которым умеет настраивать сам музыкант.

Виртуальные методы и работа с ними в производных классах.

В том случае, когда необходимо поддерживать концепцию полиморфизма при наследовании, выполняются два этапа:

- 1) определяется метод в базовом классе с ключевым словом **virtual**;
- 2) переопределить функцию в производном классе с ключевым словом **override** (при перегрузке метода в производном классе) или **new** (при скрытии реализации метода базового класса).

Другими словами, создадим базовый класс «Графический объект», методами которого являются его рисование и заливка (см. пример Virtuality):

```
class GraphObject
{
    public void DrawSth()
    {
```

```

        Console.WriteLine("Draw object");
    }
    public virtual void FillSth()
    {
        Console.WriteLine("Fill object");
    }
}

```

Будем наследовать этот класс в классе «Точка»:

```

class Point : GraphObject
{
    public new void DrawSth()
    {
        Console.WriteLine("Draw point");
    }
    public override void FillSth()
    {
        Console.WriteLine("Fill point");
    }
}

```

А в основной программе создаем экземпляр базового класса и класса – наследника:

```

static void Main(string[] args)
{
    GraphObject a = new Point();
    a.DrawSth();
    a.FillSth();
    Point b = new Point();
    b.DrawSth();
    b.FillSth();
}

```

Результатом выполнения этой программы будет:

```

C:\WINDOWS\system32\cmd.exe
Draw object
Fill point
Draw point
Fill point
Для продолжения нажмите любую клавишу . . . _

```

Следовательно, полиморфизм поддерживается в случае заливки FillSth(), а в случае рисования DrawSth() - нет.

Для наглядности сведем выводы в таблицу на дополнительном примере:

	Полиморфизм поддерживается	Полиморфизм не поддерживается
использование	override	new
код программы	<pre> class A { public virtual void Display() { </pre>	<pre> class A { public void Display() { </pre>

	<pre> Console.WriteLine("A"); } } class B : A { public override void Display() { Console.WriteLine("B"); } } class Program { static void Main(string[] args) { B b = new B(); A a = b; b.Display(); a.Display(); } } </pre>	<pre> Console.WriteLine("A"); } } class B : A { public new void Display() { Console.WriteLine("B"); } } class Program { static void Main(string[] args) { B b = new B(); A a = b; b.Display(); a.Display(); } } </pre>
результат	BB	BA

Модификатор **new** с виртуальными методами работает аналогично. Разница состоит в том, что при вызове обычных методов компилятор подставляет их вызов на этап компиляции, при вызове же виртуальных методов выбор метода происходит только на этапе выполнения.

При определении виртуального метода необходимо придерживаться некоторых правил:

- 1) метод должен содержать тело,
- 2) нельзя определить виртуальный метод как **static**,
- 3) виртуальный метод не может быть **private**, иначе его нельзя будет переопределить в производном классе.

При перегрузке метода тоже поддерживаются правила:

- 1) можно перегрузить только соответствующие методы с ключевым словом **virtual**,
- 2) перегруженный метод обязательно должен иметь реализацию в производном классе,
- 3) можно перегружать перегруженные методы, причем слово **virtual** указывается только в базовом классе,
- 4) нельзя определить метод со спецификатором **override** как **virtual** (перегруженный метод является виртуальным по умолчанию),
- 5) перегруженный метод не может быть определен как **static** или **private**.

Интерфейсы.

Если класс должен служить лишь для описания набора методов, которые должен поддерживать любой производный от него класс, для этой цели используются так называемые **интерфейсы** (пример UsingInheritance).

Объявление интерфейса подобно классу, за исключением того, что вместо ключевого слова `class` используется **interface**.

Можно выделить три основные особенности интерфейса:

1) при объявлении интерфейса все его методы неявно имеют модификатор `public`, и явное указание модификатора доступа приведет к ошибке.

2) методы не содержат тела с реализацией. После объявления метода следует точка с запятой.

3) интерфейс не может содержать данных, в нем могут быть объявлены только методы, события, свойства и индексы, которые объявляются, но не имеют реализации.

Создадим интерфейс `ITest`, который будет хранить метод создания теста:

```
interface ITest
{
    void MakeTest();
}
```

Два класса будут наследовать этот интерфейс, и хранить собственную реализацию. Отметим, что для наглядности в одном из классов при реализации метода будет стоять спецификатор **virtual**:

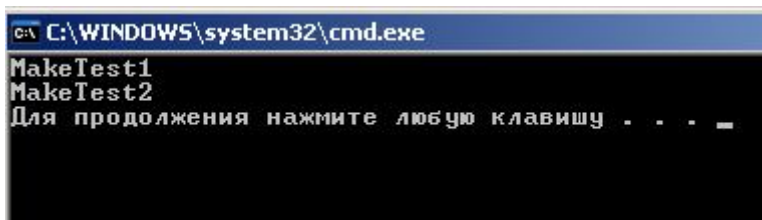
```
class Tester1 : ITest
{
    public void MakeTest()
    {
        Console.WriteLine("MakeTest1");
    }
}
class Tester2 : ITest
{
    public virtual void MakeTest()
    {
        Console.WriteLine("MakeTest2");
    }
}
```

В основной программе допишем объявление экземпляров этих классов и вызов методов для них:

```
static void Main(string[] args)
{
    ITest a = new Tester1();
    ITest b = new Tester2();
    a.MakeTest();
    b.MakeTest();

}
```

В результате выполнения данного кода получим окно:



```
C:\WINDOWS\system32\cmd.exe
MakeTest1
MakeTest2
Для продолжения нажмите любую клавишу . . . _
```

Выводы:

1) автоматически поддерживается полиморфизм (каждый класс-предок реализует метод интерфейса своим способом). Однако, если планируется класс, реализующий интерфейс, делать базовым и поддерживающим полиморфизм, необходимо явно указать спецификатор **virtual**:

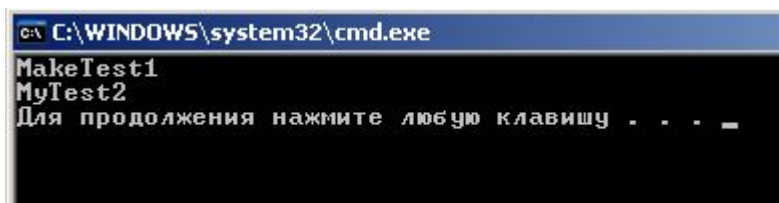
```
// создаем наследника для реализации метода без спецификатора virtual
class MyTest1 : Tester1
{
    public void MakeTest()
    {
        Console.WriteLine("MyTest1");
    }
}

// создаем наследника для реализации метода с virtual
class MyTest2 : Tester2
{
    public void MakeTest()
    {
        Console.WriteLine("MyTest2");
    }
}

...

static void Main(string[] args)
{
    // вызываем методы наследников
    Tester1 t1 = new MyTest1();
    Tester2 t2 = new MyTest2();
    t1.MakeTest();
    t2.MakeTest();
}
```

В результате выполнения данного кода получим окно:



```
C:\WINDOWS\system32\cmd.exe
MakeTest1
MyTest2
Для продолжения нажмите любую клавишу . . . _
```

При этом в самой среде в окне Error List появится предупреждение, что для первого класса нарушен полиморфизм.

2) несмотря на то, что модификатор доступа у всех интерфейсных методов **public**, все равно явно необходимо указывать этот модификатор в классе, если необходимо сделать реализацию метода открытым. В противном случае методы будут иметь модификатор **private** по умолчанию:

```
interface ITest
{
    void MakeTest();
}

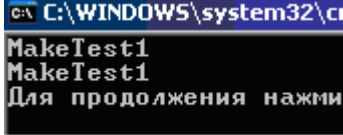
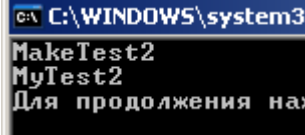
class Tester1 : ITest
{
    void MakeTest()
    {
        Console.WriteLine("MakeTest1");
    }
}

...

static void Main(string[] args)
{
    Tester1 a = new Tester1();
    a.MakeTest();           //ошибка, метод недоступен
}
```

Для наглядности сведем результаты UsingInheritance в таблицу:

	Полиморфизм не поддерживается	Полиморфизм поддерживается
интерфейс	<pre>interface ITest { void MakeTest(); }</pre>	
базовый класс	<pre>class Tester1 : ITest { public void MakeTest() { Console.WriteLine("MakeTest1"); } }</pre>	<pre>class Tester2 : ITest { public virtual void MakeTest() { Console.WriteLine("MakeTest2"); } }</pre>
вызов в Main()	<pre>ITest a = new Tester1(); a.MakeTest();</pre>	<pre>ITest b = new Tester2(); b.MakeTest();</pre>
класс-наследник	<pre>class MyTest1 : Tester1 { public void MakeTest() { Console.WriteLine("MyTest1"); } }</pre>	<pre>class MyTest2 : Tester2 { public override void MakeTest() { Console.WriteLine("MyTest2"); } }</pre>

вызов Main()	в	<code>Tester1 t1 = new MyTest1(); t1.MakeTest();</code>	<code>Tester2 t2 = new MyTest2(); t2.MakeTest();</code>
результат			

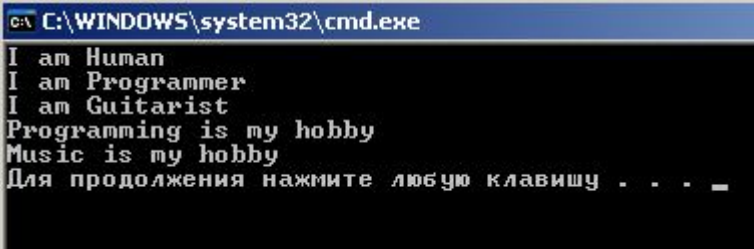
Напомним, что C# позволяет наследовать класс только от одного класса. Но класс может реализовывать несколько интерфейсов (в таком случае они перечисляются через запятую при объявлении класса). Вышесказанное демонстрирует пример IStudent.

При реализации методов явным образом есть некоторые ограничения:

- 1) при доступе к методу необходимо явно приводить ссылку к типу интерфейса, метод которого мы вызываем, или вызывать через ссылку на этот интерфейс,
- 2) нельзя указать для реализуемого метода модификатор доступа,
- 3) нельзя объявить метод как виртуальный.

Для наглядности сведем результаты IStudent в таблицу:

	Полиморфизм не поддерживается	Полиморфизм поддерживается
	<pre>//опишем интерфейс программиста interface IProgrammer { //умеет программировать void Programming(); //является ли это хобби void Hobby(); }</pre>	<pre>//опишем интерфейс музыканта interface IMusician { //умеет настраивать инструмент void Tune(); //является ли это хобби void Hobby(); }</pre>
базовый класс	<pre>//создадим класс Человек class Human { //конструктор класса public Human() { Console.WriteLine("I am Human"); } }</pre>	
класс-наследник	<pre>//создадим класс-предок Студент, который наследуется от Человека //и реализует интерфейсы программиста и музыканта class Student: Human, IMusician, IProgrammer { public void Programming() { Console.WriteLine("I am Programmer"); } public void Tune() { </pre>	

	<pre>Console.WriteLine("I am Guitarist"); } //реализация методов интерфейсов с одинаковыми именами void IProgrammer.Hobby() { Console.WriteLine("Programming is my hobby"); } void IMusician.Hobby() { Console.WriteLine("Music is my hobby"); } }</pre>	
вызов в Main()	<pre>Student st = new Student(); st.Programming(); st.Tune();</pre>	
явное преобразование при вызове метода	<pre>((IProgrammer)st).Hobby();</pre>	<pre>((IMusician)st).Hobby();</pre>
результат		

Задания для лабораторной работы

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

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.

2. 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.

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

Вариант 2

1. 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.

2. 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.

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

Вариант 3

1. 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.

2. 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.

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

Вариант 4

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

2. 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.

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

Вариант 5

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

2. 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.

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

Вариант 6

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

2. 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.

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

Вариант 7

1. 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.

2. 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.

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

Вариант 8

1. 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.

2. 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.

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

Вариант 9

1. 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).

2. 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.

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

Вариант 10

1. 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.

2. 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.

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

Вариант 11

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

2. Создать класс Memogies, 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.

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

Вариант 12

1. 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.

2. 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.

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

Вариант 13

1. 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.

2. 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.

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

Вариант 14

1. 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.

2. 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.

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

Вариант 15

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

2. 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.

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

Вариант 16

1. 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.

2. 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.

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

Вариант 17

1. 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.

2. 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.

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

Вариант 18

1. 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).

2. 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.

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

Вариант 19

1. 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).

2. 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.

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

Вариант 20

1. 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).

2. 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.

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