

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Кафедра «Вычислительные системы и информационная безопасность»

**Методические указания к контрольной работе по дисциплине «Объектно-ориентированное программирование»**

**для студентов 3-го курса**

**направления подготовки 09.03.02 Информационные системы и технологии**

**заочной формы обучения**

Ростов-на-Дону

2021 г.

Составитель к.т.н., доцент, доцент А.Р. Айдинян,

**УДК 681.3**

Подготовлено на кафедре «Вычислительные системы и информационная безопасность»

Методические рекомендации  
к контрольной работе по дисциплине   
«Объектно-ориентированное программирование»  
/ ДГТУ, Ростов-на-Дону, 2021, 96 с.

Методические рекомендации для студентов заочной формы обучения представляют собой комплекс рекомендаций и разъяснений, позволяющих студенту оптимальным образом организовать процесс выполнения контрольной работы.

1. **Алгоритм выбора варианта контрольной работы**

Контрольная работа заключается в выполнении заданий, приведенных в данных методических указаниях. При выполнении каждого задания необходимо привести:

— формулировку задания;

— описание алгоритма;

— листинг программы;

— введенные и полученные данные.

Номер варианта выбирается по следующему алгоритму:

Для выбора варианта необходимо взять предпоследнюю и последнюю цифры номера зачетной книжки. Номер варианта находится на пересечении соответствующей строки и столбца.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Последняя цифра номера зачетной книжки | | | | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Предпоследняя цифра номера зачетной книжки | 0 | 1 | 2 | 5 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 3 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 4 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 5 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 6 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 7 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 9 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |

Например, для зачетки с номером 123456 необходимо взять номер варианта из 5-ой строки и 6-го столбца (вариант 17).

1. **Среда и язык программирования**

Наиболее предпочтительными для выполнения контрольной работы являются объектно-ориентированные языки программирования Java, C#, C++, Object Pascal. Рекомендуется использовать ту же среду разработки и язык программирования, которые используете или будете использовать для выполнения контрольной работы по дисциплине «Межплатформенное программирование».

В данных методических указаниях рассматриваются примеры выполнения заданий на языке C#.

**Задание 1. Классы**

# Теоретические сведения

Объектно-ориентированное (ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

Класс представляет собой шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными. В спецификация класса С# используется для построения объектов, которые являются экземплярами класса. Следовательно, класс, по существу, представляет собой ряд схематических описаний способа построения объекта. При этом очень важно подчеркнуть, что класс является логической абстракцией. Физическое представление класса появится в оперативной памяти лишь после того, как будет создан объект этого класса.

В общем виде объявление класса на языке C# выглядит следующим образом:

class имя\_класса {

// Объявление переменных (полей класса).

доступ тип переменная1;

доступ тип переменная2;

//...

доступ тип переменнаяN;

// Объявление методов.

доступ возращаемый\_тип метод1 (параметры) {

// тело метода

}

доступ возращаемый\_тип метод2 (параметры). {

// тело метода

}

//...

доступ возращаемый\_тип методы (параметры) {

// тело метода

}

}

Доступ определяется одним из следующих ключевых слов:

* private — поле или метод могут быть использован только внутри тела класса, в котором объявлены;
* public — поле или метод могут быть использован из любого места в программе.

Для создания объекты из класса используется оператор new:

Имя\_класса название\_объекта = new Имя\_класса();

Например, для класса Student:

Student s1 = new Student();

Эта строка объявления выполняет три функции. Во-первых, объявляется переменная s1, относящаяся к классу Student. Сама эта переменная не является объектом, а лишь переменной, которая может ссылаться на объект. Во-вторых, создается конкретная, физическая, копия объекта. Это делается с помощью оператора new. И наконец, переменной s1 присваивается ссылка на данный объект. Таким образом, после выполнения анализируемой строки объявленная переменная s1 ссылается на объект типа Student.

Класс представляет собой объявленный программистом составной тип данных, имеющий в составе:

* Поля данных — параметры объекта, задающие его состояние (свойства объекта предметной области). Иногда поля данных объекта называют свойствами объекта, из-за чего возможна путаница. Физически поля представляют собой значения (переменные, константы), объявленные как принадлежащие классу.
* Методы — процедуры и функции, связанные с классом. Они определяют действия, которые можно выполнять над объектом такого типа, и которые сам объект может выполнять.

Методы представляют собой подпрограммы, которые манипулируют данными, определенными в классе, а во многих случаях они предоставляют доступ к этим данным. Как правило, другие части программы взаимодействуют с классом посредством его методов.

Метод состоит из одного или нескольких операторов. В грамотно написанном коде С# каждый метод выполняет только одну функцию. У каждого метода имеется свое имя, по которому он вызывается. В общем случае, методу в качестве имени можно присвоить любой действительный идентификатор. Следует, однако, иметь в виду, что идентификатор Main() зарезервирован для метода, с которого начинается выполнение программы. Кроме того, в качестве имен методов нельзя использовать ключевые слова С#.

Общая форма определения метода:

доступ тип имя\_метода(тип\_параметра имя\_параметра, тип\_параметра имя\_параметра, …)

{

// тело метода

}

Для возврата значения из метода в вызывающую часть программы служит следующая форма оператора return:

return значение;

где значение — это конкретное возвращаемое значение.

Если метод не возвращает никакого значения, то указывается тип void.

# Задание

1. Разработать класс на языке C# в соответствии с вариантом задания.
2. Написать программу, демонстрирующую работу указанных выше элементов класса. В программе должен быть создан массив объектов разработанного класса, реализован ввод и вывод объектов, содержащихся в массиве.

## Вариант 1

Разработать класс Worker (сотрудник), содержащий следующие поля:

personID (табельный номер сотрудника),

family (фамилия сотрудника),

name (имя сотрудника),

dolgnost (должность),

date (дата приёма на работу);

методы:

CalculateSalary() — посчитать зарплату (зарплата зависит от должности).

## Вариант 2

Разработать класс Worker (сотрудник), содержащий следующие поля:

personID (табельный номер сотрудника),

family (фамилия сотрудника),

name (имя сотрудника),

dolgnost (должность),

date (дата приёма на работу);

методы:

CalculateSalary() — посчитать зарплату (зарплата зависит от должности).

## Вариант 3

Разработать класс Jobless (безработный), содержащий следующие поля:

joblessID (регистрационный номер безработного),

lastName (фамилия безработного),

firstName (имя безработного),

birth (год рождения);

методы:

GetAge() — получить возраст.

## Вариант 4

Разработать класс Jobless (безработный), содержащий следующие поля:

joblessID (регистрационный номер безработного),

lastName (фамилия безработного),

firstName (имя безработного),

birth (год рождения);

методы:

GetAge() — получить возраст.

## Вариант 5

Разработать класс Passport (паспорт), содержащий следующие поля:

serial (серия документа),

number (номер документа),

date (дата выдачи),

who (кем выдан);

методы:

GetChangeTime() — получить время следующего обмена.

## Вариант 6

Разработать класс Passport (паспорт), содержащий следующие поля:

serial (серия документа),

number (номер документа),

date (дата выдачи),

who (кем выдан);

методы:

GetChangeTime() — получить время следующего обмена.

## Вариант 7

Разработать класс Route (туристический маршрут), содержащий следующие поля:

routeID (идентификатор маршрута),

name (название маршрута),

period (время в пути),

сost (стоимость одного дня);

методы:

GetPrice() — получить цену за весь срок прохождения маршрута.

## Вариант 8

Разработать класс Route (туристический маршрут), содержащий следующие поля:

routeID (идентификатор маршрута),

name (название маршрута),

period (время в пути),

сost (стоимость одного дня);

методы:

GetPrice() — получить цену за весь срок прохождения маршрута.

## Вариант 9

Разработать класс Cust (арендатор), содержащий следующие поля:

INN (ИНН арендатора),

name (название арендатора),

address (адрес арендатора),

room (номер кабинета),

chief (фамилия руководителя);

методы:

GetFloor() — получить номер этажа (зависит от номера кабинета, кабинеты, номера которых начинаются на 1 находятся на 1-м этаже, номера которых начинаются на 2 находятся на 2-м этаже и т.д.).

## Вариант 10

Разработать класс Cust (арендатор), содержащий следующие поля:

INN (ИНН арендатора),

name (название арендатора),

address (адрес арендатора),

room (номер кабинета),

chief (фамилия руководителя);

методы:

GetFloor() — получить номер этажа (зависит от номера кабинета, кабинеты, номера которых начинаются на 1 находятся на 1-м этаже, номера которых начинаются на 2 находятся на 2-м этаже и т.д.).

## Вариант 11

Разработать класс TelephonNumber (телефонный номер), содержащий следующие поля:

id (идентификатор клиента),

family (фамилия клиента),

phoneAddress (адрес клиента),

phoneNumber (номер телефона),

tariffType (вид тарифа — безлимитный, поминутный и т.д.);

методы:

GetCost() — получить плату за телефон (зависит от вида тарифа).

## Вариант 12

Разработать класс TelephonNumber (телефонный номер), содержащий следующие поля:

id (идентификатор клиента),

family (фамилия клиента),

phoneAddress (адрес клиента),

phoneNumber (номер телефона),

tariffType (вид тарифа — безлимитный, поминутный и т.д.);

методы:

GetCost() — получить плату за телефон (зависит от вида тарифа).

## Вариант 13

Разработать класс Count (счет), содержащий следующие поля:

сountNumber (номер счета),

date (дата выписки счета),

value (сумма к уплате),

count (количество товара);

методы:

GetAverage() — получить среднюю стоимость товара в счёте.

## Вариант 14

Разработать класс Count (счет), содержащий следующие поля:

сountNumber (номер счет-фактуры),

date (дата выписки счет-фактуры),

value (сумма к уплате),

count (количество товара);

методы:

GetAverage() — получить среднюю стоимость товара в счёте.

## Вариант 15

Разработать класс Buy (покупка), содержащий следующие поля:

number (номер покупки),

sum (сумма покупки),

otdel (название отдела магазина, где совершена покупка),

otdel (название отдела магазина, где совершена покупка);

методы:

GetSumSoSkidkoi(double skidka) — посчитать сумму со скидкой.

## Вариант 16

Разработать класс Buy (покупка), содержащий следующие поля:

number (номер покупателя),

sum (сумма покупки),

otdel (название отдела магазина, где совершена покупка);

методы:

GetSumSoSkidkoi(double skidka) — посчитать сумму со скидкой.

## Вариант 17

Разработать класс Spec (специальность), содержащий следующие поля:

kod (код специальности),

name (название специальности),

balls (количество баллов, необходимых для поступления);

методы:

Check(int mathBall, int phisicsBall, int rusBall) — проверить достаточно ли баллов для поступления на данную специальность.

## Вариант 18

Разработать класс Spec (специальность), содержащий следующие поля:

kod (код специальности),

name (название специальности),

balls (количество баллов, необходимых для поступления);

методы:

Check(int mathBall, int phisicsBall, int rusBall) — проверить достаточно ли баллов для поступления на данную специальность.

## Вариант 19

Разработать класс Bilet (билет), содержащий следующие поля:

time1 (время вылета),

time2 (время прилета),

cost (цена билета),

name (владелец билета),

nomer (номер места);

методы:

GetDuration() — посчитать время в пути.

## Вариант 20

Разработать класс Bilet (билет), содержащий следующие поля:

time1 (время вылета),

time2 (время прилета),

cost (цена билета),

name (владелец билета),

nomer (номер места);

методы:

GetDuration() — посчитать время в пути.

# Порядок выполнения работы

Ниже приведены варианты заданий.

Рассмотрим порядок выполнения работы на примере класса Bilet.

1. Создать класс Bilet и добавить в него поля:

class Bilet

{

public int time1;

public int time2;

public int cost;

public string name;

public int nomer;

}

1. Создать 2 объекта класса Bilet с помощью оператора new и занести его в ссылку b1 и b2:

Bilet b1=new Bilet();

Bilet b2=new Bilet();

1. После этого в поля класса занести значения, вводимые с клавиатуры пользователем.

static void Main(string[] args)

{

Bilet b1=new Bilet();

//Ввод первого билета

Console.WriteLine("введите число вылета, число прилета, цену билета, имя владельца билета, номер места - каждое поле с новой строчки");

b1.time1 = int.Parse(Console.ReadLine());

b1.time2 = int.Parse(Console.ReadLine());

b1.cost = int.Parse(Console.ReadLine());

b1.name = Console.ReadLine();

b1.nomer = int.Parse(Console.ReadLine());

Bilet b2=new Bilet();

//Ввод второго билета

Console.WriteLine("введите число вылета, число прилета, цену билета, имя владельца билета, номер места - каждое поле с новой строчки");

b2.time1 = int.Parse(Console.ReadLine());

b2.time2 = int.Parse(Console.ReadLine());

b2.cost = int.Parse(Console.ReadLine());

b2.name = Console.ReadLine();

b2.nomer = int.Parse(Console.ReadLine());

}

1. Желательно код для считывания данных одного объекта оформить в виде метода (этот метод можно назвать Read()) внутри класса Bilet:

например,

class Bilet

{

public int time1;

public int time2;

public int cost;

public string name;

public int nomer;

public void Read()

{

Console.WriteLine("введите число вылета, число прилета, цену билета, имя владелеца билета, номер места - каждое поле с новой строчки");

time1 = int.Parse(Console.ReadLine());

time2 = int.Parse(Console.ReadLine());

cost = int.Parse(Console.ReadLine());

name = Console.ReadLine();

nomer = int.Parse(Console.ReadLine());

}

}

1. Разработать метод в соответствии со своим заданием

public int GetDuration()

{

return b-a;

}

1. Добавить метод для преобразования объекта в строку:

//Стандартная запись для метода, который преобразует объект в строку

public override string ToString()

{

return “Билет 1 :” + time1 + time2 + “Цена, ” + cost +   
“ Владелец: “ + name + “ Место: nomer”;

}

1. Написать метод Main

static void Main(string[] args)

{

Bilet b1 = new Bilet ();

Bilet b2 = new Bilet ();

b1.Read();

b2.Read();

Console.WriteLine(b1.ToString());

Console.WriteLine(b2.ToString());

}

# Контрольные вопросы

1. Что такое класс?
2. Что такое объект?
3. Что такое поля класса?
4. Что такое методы класса?
5. Как описать класс на языке C#?
6. Какие бывают модификаторы доступа к членам класса?

Задание 2. Методы для доступа к полям, свойства класса.

# Теоретические сведения

Пусть имеется класс

class Student

{

public string fam;

public int gr;

}

На год рождения студента имеется ограничение, в диапазоне 1970-2010. Public-поле не позволяет учесть ограничения в связи с тем, что к такому полю открыт доступ из-вне класса и в него можно записать любое значение. Пока поле является public, в него можно записать что угодно. Поэтому доступ к полям обычно закрывается и обращение к полям идет через методы.

Поэтому для работы с полем класса gr был добавлен метод, чтобы учесть это ограничение диапазоне 1970-2010 и сделаем доступ к полю gr с модификатором доступа private:

private int gr;

и добавим 2 метода:

public int GetGr () { return gr; }

public void SetGr (int gr)

{

this.gr = 2000;

if (gr>=1970 & gr <=2010)

this.gr = gr;

}

Слово this используется перед именем поля для того, чтобы отличить поле и параметр метода, которые имеют одинаковые имена.

Таким образом, в поле gr не попадет никакое другое значение вне интервала [1970, 2010]. При попытке записать туда число вне интервала, в поле будет записано 2000.

Некоторые языки программирования, например, C# или Object Pascal, позволяют вместо двух методов GetGr или SetGr использовать «свойство», которое имеет более компактную и удобную форму:

class Student

{

private int gr;

// добавим свойство:

public int Gr

{

get { return gr; }

set {

this.gr = 2000;

if (value >=1970 & value <=2010)

this.gr = value;

}

}

Ключевое слово **value** определяет входной параметр для аксессора **set**, то есть новое значение поля.

В результате можно переделать конструкторы.

class Student

{

public Student (string fam, int gr)

{

this.fam = fam;

Gr = gr;

}

Обращение к свойствам объекта осуществляется следующим образом:

Student s = new Student();

s.Fam = “Козлов”;

s.Gr = 1990;

string fam = s.Fam;

# Задание

Изменить класс, разработанный в предыдущей работе, следующим образом:

1. Все поля класса, для которых необходимо ограничение на значение поля, описать с модификатором **private**.
2. Добавить методы для считывания и записи значений одного из полей с модификатором доступа **private**. В метод для записи значения в поле установить ограничение на значение.
3. Добавить два свойства (если позволяет язык программирования) для считывания и записи значений двух полей с модификатором доступа **private** с учетом ограничений на значение.
4. Переделать главную функцию программы Main таким образом, чтобы программа компилировалась, работала корректно и демонстрировала использование добавленных методов и свойств.

# Контрольные вопросы

1. Что такое свойство класса?
2. Что означает зарезервированное слово **value**?
3. Как выглядит синтаксис методов для записи и считывания значения поля?
4. Почему поля предпочтительнее описывать с модификатором доступа **private**?

**Задание 4. Конструкторы. Операторы класса.**

# Теоретические сведения

Конструктор — метод, вызываемый при создании объекта для инициализации значений его полей. У конструктора такое же имя, как и у его класса. Тип значения у конструктора не указывается.

Перегрузка операторов позволяет описывать и применять к созданным программистом типам данных (классам) операции, например, + \* / и.т.д., по смыслу эквивалентные уже имеющимся в языке. Как только для класса определяются операторы, появляется возможность оперировать объектами этого класса, используя обычный синтаксис выражений в С#. Однако не все языки поддерживают перегрузку операторов.

Общая форма перегрузки унарного оператора:

public static возвращаемый\_тип operator op(тип\_параметра операнд)

{

// операции

}

Общая форма перегрузки бинарного оператора:

public static возвращаемый\_тип operator op(тип\_параметра1 операнд1, тип\_параметра1 операнд2)

{

// операции

}

Возвращаемый\_тип может быть любым типом, но обычно он соответствует типу класса, для которого перегружается оператор.

Тип\_параметра для унарных операторов должен быть таким же, как и у класса, для которого перегружается оператор. А в бинарных операторах хотя бы один из операндов должен быть такого же типа, как и у его класса.

Пусть имеется массив Студентов m, которым присвоены какие-то значения

int n =10;

Student [] m = new Student [n];

for (int i=0; i<n; i++)

{

//Ввод значений для i-го студента

Console.WriteLine(“Введите для студента ” + i + “ Фамилию”);

String fam = Console.ReadLine();

Console.WriteLine(“Введите для студента ” + i + “ Год рождения”);

int gr = int.Parse(Console.ReadLine());

m[i] = new Student(fam, gr);

}

Сортировку массива объектов можно выполнить алгоритмом сортировки.

for (int i=0; i<n-1; i++)

for (int j=i+1; j<n; j++)

if (m[i] < m[j]) // здесь вызывается Student.operator <(m[i], m[j])

{

Student s = m[i];

m[i] = m[j];

m[j] = s;

}

Разработаем конструкторы:

public Student ()

{

fam = “Иван”; gr = 1990;

}

public Student (string fam, int gr)

{

this.fam = fam;

this.gr = gr;

}

Обратите внимание, что в этом конструкторе в поле gr при создании объекта может быть записано любое значение, так как осуществляется запись непосредственно в поле. На год рождения студента имеется ограничение, в диапазоне 1970-2010. Учтем это ограничение:

public Student (string fam, int gr)

{

this.fam = fam;

if (gr>=1970 & gr <=2010)

this.gr = gr;

}

Это требует повтора проверки gr>=1970 & gr <=2010, которая уже была прописана в методе SetGr или свойстве Gr (как показано в предыдущем задании). Поэтому лучше изменить конструктор следующим образом:

public Student (string fam, int gr)

{

this.fam = fam;

SetGr(gr); // или Gr = gr;

}

public Student (Student st)

{

this.fam = st.fam;

this.gr = st.gr;

}

Здесь копируется поле из другого студента, который не может содержать неверное значение поля, поэтому можно не использовать SetGr или Gr.

# Задание

Изменить класс, разработанный в предыдущем задании следующим образом:

1. Добавить в класс конструктор по умолчанию, конструктор с параметрами и конструктор копирования с учетом ограничений на соответствующие поля.
2. Реализовать операторы сравнения (> и <) для класса.
3. В программе выполнить сортировку массива объектов.

# Контрольные вопросы

1. Что такое модификатор доступа? Какие бывают модификаторы доступа?
2. Что такое конструктор?
3. Как на языке C# создать конструктор класса?
4. Что такое перегрузка операторов?
5. Какие операторы можно перегружать?

Задание 4. Наследование.

**Теоретические сведения**

Наследование — механизм объектно-ориентированного программирования, позволяющий описать новый класс на основе уже существующего. Оно является одним из основополагающих принципов объектно-ориентированного программирования. Благодаря наследованию можно создать общий класс, в котором определяются характерные особенности, присущие множеству связанных элементов. От этого класса могут затем наследовать другие, более конкретные классы, добавляя в него свои индивидуальные особенности.

В языке С# класс, который наследуется, называется базовым, а класс, который наследует — производным. Следовательно, производный класс представляет собой специализированный вариант базового класса. Он наследует все поля, методы и свойства, определяемые в базовом классе, добавляя к ним свои собственные элементы.

Для любого производного класса можно указать только один базовый класс. Если ClassC является производным от ClassB, и ClassB является производным от ClassA, ClassC наследует члены, объявленные в ClassB и ClassA.

Поддержка наследования в С# состоит в том, что в объявление одного класса разрешается вводить другой класс. Для этого при объявлении производного класса через двоеточие указывается базовый класс:

class имя\_производного\_класса : имя\_базового\_класса {

// тело класса

}

Для любого производного класса можно указать только один базовый класс.

Наследование класса не отменяет ограничения, накладываемые на доступ к закрытым членам класса. Поэтому если в производный класс и входят все члены его базового класса, в нем все равно оказываются недоступными те члены базового класса, которые являются закрытыми. Для того чтобы доступ к полю или методы класса можно было получать из объектов этого класса и из объектов классов-наследников, но не из каких-либо других объектов или классов, используется модификатор доступа **protected**.

В иерархии классов допускается, чтобы у базовых и производных классов были свои собственные конструкторы, при этом конструктор базового класса конструирует базовую часть объекта, а конструктор производного класса — производную часть этого объекта.

С помощью формы расширенного объявления конструктора производного класса и ключевого слова **base** в производном классе может быть вызван конструктор, определенный в его базовом классе. Ниже приведена общая форма этого расширенного объявления:

конструктор\_производного\_класса(параметры) : base (аргументы)

{

//тело конструктора

}

где список\_аргументов обозначает любые аргументы, необходимые конструктору в базовом классе.

Виртуальный метод отличается от обычного тем, что он может быть переопределен в одном или нескольких производных классах. Следовательно, у каждого производного класса может быть свой вариант виртуального метода. Кроме того, виртуальные методы интересны тем, что именно происходит при их вызове по ссылке на базовый класс. В этом случае средствами языка С# определяется именно тот вариант виртуального метода, который следует вызывать, исходя из типа объекта, к которому происходит обращение по ссылке, причем это делается во время выполнения программы, а не во время компиляции. Поэтому при ссылке на разные типы объектов выполняются разные варианты виртуального метода.

Метод объявляется как виртуальный в базовом классе с помощью ключевого слова **virtual**, указываемого перед его именем. Когда же виртуальный метод переопределяется в производном классе, то для этого используется модификатор **override**. А сам процесс повторного определения виртуального метода в производном классе называется переопределением метода.

Иногда требуется создать базовый класс, в котором определяется лишь самая общая форма для всех его производных классов, а наполнение ее деталями предоставляется каждому из этих классов. В таком классе определяется лишь характер методов, которые должны быть конкретно реализованы в производных классах, а не в самом базовом классе. Подобная ситуация возникает, например, в связи с невозможностью получить содержательную реализацию метода в базовом классе.

Абстрактный метод создается с помощью указываемого модификатора типа **abstract**. У абстрактного метода отсутствует тело, и поэтому он не реализуется в базовом классе. Это означает, что он должен быть переопределен в производном классе, поскольку его вариант из базового класса просто непригоден для использования. Нетрудно догадаться, что абстрактный метод автоматически становится виртуальным и не требует указания модификатора **virtual**.

Класс, содержащий один или больше абстрактных методов, должен быть также объявлен как абстрактный, и для этого перед его объявлением class указывается модификатор **abstract**. А поскольку реализация абстрактного класса не определяется полностью, то у него не может быть объектов. Следовательно, попытка создать объект абстрактного класса с помощью оператора **new** приведет к ошибке во время компиляции.

Когда производный класс наследует абстрактный класс, в нем должны быть реализованы все абстрактные методы базового класса. В противном случае производный класс должен быть также определен как **abstract**. Таким образом, атрибут **abstract** наследуется до тех пор, пока не будет достигнута полная реализация класса.

**Задание**

**Вариант 1**

Разработать класс Docent, являющийся наследником класса Worker, добавив следующие поля:

godZ (год присвоения звания),

uchStep (ученая степень).

При создании объекта класса Docent в соответствующее поле базового класса должна автоматически заноситься должность «доцент», которую невозможно изменить в дальнейшем.

**Вариант 2**

Разработать класс Voditel, являющийся наследником класса Worker (сотрудник), добавив следующие поля:

prava (номер водительских прав),

category (категория прав, например, A, B, C, D).

При создании объекта класса Voditel в соответствующее поле базового класса должна автоматически заноситься должность «водитель», которую невозможно изменить в дальнейшем.

**Вариант 3**

Разработать класс HighJobless, являющийся наследником класса Jobless, добавив следующие поля:

institut (оконченный институт),

date (дата постановки на учёт),

godEnd (год окончания).

Добавить в класс метод EverWork() для определения, работал ли человек вообще — если дата постановки на учёт совпадает с датой окончания института, можно считать, что он не работал никогда.

**Вариант 4**

Разработать класс LongJobless, являющийся наследником класса Jobless (безработный), добавив следующие поля:

date (дата постановки на учёт),

reason (причина отсутствия работы),

lastPlace (последнее место работы).

Добавить в класс методы:

GetDuration(), определяющий как долго человек является безработным.

GetJoblessPercent(), определяющий какую часть жизни человек является безработным.

**Вариант 5**

Разработать класс ZarganPassport, являющийся наследником класса Passport, добавив следующие поля:

country (страна, выдавшая паспорт),

lastBorderCrossDate (дата последнего пересечения границы),

lastBorderCrossState (государство, чья граница была пересечена).

Добавить в класс метод Cross(string state), в котором реализовать автоматическую установку текущей даты в поле lastBorderCrossDate и запись в поле lastBorderCrossState параметра state.

**Вариант 6**

Разработать класс OldPassport, являющийся наследником класса Passport, добавив следующие поля:

dateChange (дата обмена),

reason (причина обмена).

Причиной обмена могут быть только истечение срока действия, утеря и изменение личных данных.

**Вариант 7**

Разработать класс ZagranRoute, являющийся наследником класса Route, добавив следующие поля:

country (страна),

needVisa (необходимость получения визы).

Необходимость получения визы зависит от страны — для поездки в некоторые страны получать визу не обязательно. Это поле должно заполняться автоматически только в результате указания страны, и ни в каком другом случае.

**Вариант 8**

Разработать класс HardRoute, являющийся наследником класса Route, добавив следующие поля:

skills (необходимые навыки),

place (местность, по которой проходит маршрут).

Необходимые навыки зависят от местности, по которой проходит маршрут и должны устанавливаться автоматически только при указании местности.

**Вариант 9**

Разработать класс PhisCust, являющийся наследником класса Cust, добавив следующие поля:

passport (номер паспорта),

snils (страховой номер индивидуального лицевого счёта).

Для физического лица поля название и фамилия руководителя должны всегда совпадать.

**Вариант 10**

Разработать класс UrCust, являющийся наследником класса Cust, добавив следующие поля:

OGRN (основной государственный регистрационный номер),

type (тип юридического лица),

fullAddress (полный адрес) — должен включать в себя адрес арендуемого помещения и номер кабинета и формироваться автоматически.

**Вариант 11**

Разработать класс TelephonWitnInternet, являющийся наследником класса TelephonNumber, добавив следующие поля:

tarifName (название тарифа),

tarifCost (стоимость тарифа),

speed (ограничение скорости интернета).

Метод GetCost() вызванный для объекта класса TelephonWitnInternet должен при расчёте платы за телефон учитывать также плату за интернет.

**Вариант 12**

Разработать класс MobileNumber, являющийся наследником класса TelephonNumber, добавив следующие поля:

operator (оператор связи),

region (домашний регион),

balans (текущий баланс счёта).

Добавить в класс метод Call(int minutes) отвечающий за снятие со счёта денег за звонок, длительность которого задаётся параметром minutes. Снятие должно осуществляться только в случае, если в поле базового класса tariffType указано «поминутный».

**Вариант 13**

Разработать класс PayedCount, являющийся наследником класса Count, добавив следующие поля:

datePayed (дата оплаты),

who (кто оплатил).

Добавить в базовый класс Count метод Pay(string who) реализующий оплату счёта и возвращающий в качестве результата объект класса PayedCount. Дата оплаты должны устанавливаться автоматически на основе текущей даты.

**Вариант 14**

Разработать класс UnpayedCount, являющийся наследником класса Count, добавив следующие поля:

srok (срок оплаты),

days (количество просроченных дней).

При получении значения поля сумма для неоплаченного счёта оно должно быть увеличено на некоторую сумму, составляющую пеню за просрочку и зависящую от количества дней просрочки.

**Вариант 15**

Разработать класс CardBuy, являющийся наследником класса Buy (покупка), добавив следующие поля:

creditNumber (номер кредитной карты),

creditCode (код кредитной карты).

Скидки не должны распространяться на покупки, совершенные по кредитной карте, соответственно метод базового класса GetSumSoSkidkoi, если он вызван из класса CardBuy, должен выдавать соответствующее сообщение и возвращать исходное значение суммы, независимо от того, какая величина скидки была указана в качестве параметра.

**Вариант 16**

Разработать класс InternetBuy, являющийся наследником класса Buy (покупка), добавив следующие поля:

paymentSystem (платёжная система),

commission (величина комиссии).

При получении поля sum (сумма) базового класса необходимо прибавлять к ней величину комиссии.

**Вариант 17**

Разработать класс BakalSpec, являющийся наследником класса Spec (специальность), добавив следующие поля:

srok (срок обучения),

magistr (возможность дальнейшего поступления в магистратуру).

Разработать класс MagSpec, являющийся наследником класса Spec (специальность), добавив следующие поля:

srok (срок обучения).

**Вариант 18**

Разработать класс SpecSpec, являющийся наследником класса Spec (специальность), добавив следующие поля:

qualification (получаемая квалификация),

yearClose (год окончания приёма на данную специальность).

Если год окончания приёма больше текущего года, то при попытке проверить достаточно ли баллов для поступления на данную специальность должно быть выдано сообщение о том, что на данную специальность больше нельзя поступить.

**Вариант 19**

Разработать класс LocalBilet, являющийся наследником класса Bilet (билет), добавив следующие поля:

town1 (пункт отправления),

town2 (пункт назначения).

Добавить в класс метод ReturnTicket(int nomer) — обратный билет, для которого необходимо указать только номер места, и поменять местами пункт отправления и пункт назначения.

**Вариант 20**

Разработать класс ForeignBilet, являющийся наследником класса Bilet (билет), добавив следующие поля:

country (страна назначения),

airport (название аэропорта).

**Контрольные вопросы**

1. Что такое наследование?
2. Как на языке C# при описании класса указать, что он является наследником от другого класса?
3. Как работает модификатор доступа protected?
4. Что такое виртуальный метод?
5. Что такое абстрактный класс?

Задание 5. Иерархия классов.

Разработать самостоятельно 2 класса, в соответствии с заданием и программу, демонстрирующую работу с экземплярами этих классов.

Определить иерархию наследования из двух классов в соответствии с номером задания. Каждый класс снабдить свойствами и методами в соответствии с предметной областью, указанной в варианте задания.

В базовом классе предусмотреть метод info() (или ToString() для C#), возвращающий информацию об объекте в виде строки. Предусмотреть конструкторы, инициализирующие свойства объектов переданными данными либо значениями по умолчанию. Написать демонстрационную программу, создающую 4-5 объектов и выводящую на экран информацию о них.

Варианты классов:

1. «Водный транспорт», «Грузовое судно»

2. «Летательный аппарат», «Дирижабль»

3. «Здание», «Коттедж»

4. «Двигатель», «Двигатель внутреннего сгорания»

5. «Устройство печати», «Струйный принтер»

6. «Устройство ввода», «Цифровая камера»

7. «Растровое изображение», «Репродукция картины»

8. «Млекопитающее», «Собака»

9. «Транспортное средство», «Легковой автомобиль»

10. «Печатное издание», «Номер журнала»

11. «Документ», «Квитанция об оплате»

12. «Пищевой продукт», «Йогурт»

13. «Корпусная мебель», «Книжный шкаф»

14. «Проверка знаний», «Экзамен»

15. «Носитель информации», «Компакт-диск»

16. «Аудиозапись», «файл в формате MP3»

17. «Видеозапись», «Художественный фильм»

18. «Транспортное средство», «Маршрутный автобус»

19. «Удостоверение», «Паспорт»

20. «Сотрудник компании», «Начальник отдела»

Примеры выполнения заданий

Пример задания 1

1. Описать класс с именем AEROFLOT, содержащий следующие поля:

* название пункта назначения рейса;
* номер рейса;
* тип самолета.

1. Написать программу, выполняющую следующие действия:

* ввод с клавиатуры данных в массив, состоящий из семи объектов класса AEROFLOT; записи должны быть упорядочены по возрастанию номера рейса;
* вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры;
* если таких рейсов нет, выдать на дисплей соответствующее сообщение.

**Решение**

**Листинг (исходный код программы)**

Листинг файла Aeroflot.cs

using System;

namespace \_5

{

class Aeroflot

{

public string punkt;

public int number;

public string type;

public void Init()

{

Console.Write("Пункт назначения рейса = ");

punkt = Console.ReadLine();

Console.Write("Номер рейса = ");

number = Convert.ToInt32(Console.ReadLine());

Console.Write("Тип самолета = ");

type = Console.ReadLine();

}

public override string ToString()

{

return string.Format("Рейс №{0}, {1}, {2}", number, punkt, type);

}

}

}

Листинг файла Program.cs

using System;

using System.Collections.Generic;

namespace \_5

{

class Program

{

public class AeroflotComparer : Comparer<Aeroflot>

{

public override int Compare(Aeroflot first, Aeroflot second)

{

return (first.number < second.number) ? -1 : ((first.number > second.number) ? 1 : 0);

}

}

static void Main(string[] args)

{

Aeroflot[] aero = new Aeroflot[7];

for (int i = 0; i < aero.Length; ++i)

{

aero[i] = new Aeroflot();

aero[i].Init();

}

AeroflotComparer comparer = new AeroflotComparer();

Array.Sort(aero, comparer);

Console.WriteLine("\nВведенные данные:");

for (int i = 0; i < aero.Length; ++i)

Console.WriteLine(aero[i].ToString());

Console.WriteLine();

Console.Write("Пункт назначения для поиска = ");

string punkt = Console.ReadLine();

int cnt = 0;

Console.WriteLine("Подходящие рейсы:");

for (int i = 0; i < aero.Length; ++i)

if (aero[i].punkt.Equals(punkt))

Console.WriteLine("{0}. {1}, {2}", ++cnt, aero[i].number, aero[i].type);

if (cnt == 0)

Console.WriteLine("Подходящих рейсов нет!");

Console.Read();

}

}

}

**Результат работы программы**

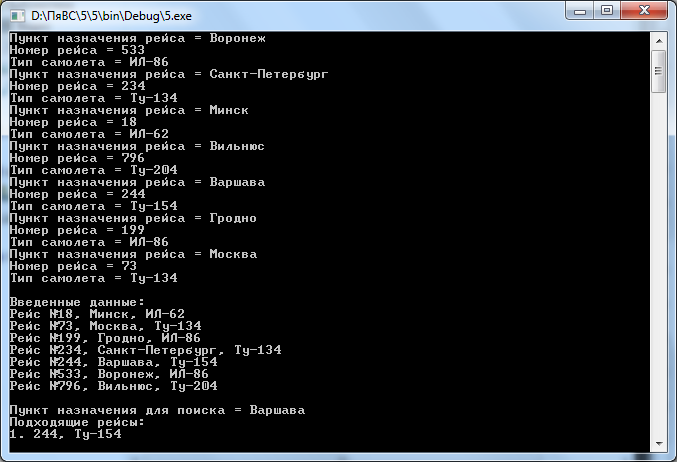


Рисунок 12 – Результат работы программы

**Пример выполнения задание 2.**

Разработать класс Jobless (безработный), содержащий следующие поля:

* joblessID (регистрационный номер безработного),
* lastName (фамилия безработного),
* firstName (имя безработного),
* birth (возраст безработного);

методы:

* GetAge() — получить возраст.

Все поля класса должны быть скрытыми (объявленными как private). Для доступа к полям классам должны быть предусмотрены соответствующие свойства.

Реализовать для класса конструктор по умолчанию, конструктор с параметрами и конструктор копирования.

Перегрузить операторы == и !=. Перегрузить операторы > и <, считая, что один безработный больше другого, если родился раньше (если позволяет выбранный язык программирования).

Отсортировать элементы массива по выбранному параметру, используя операторы < и >, или если не позволяет язык программирования, то сравнивать объекты без использования перегруженных операторов.

Написать программу, демонстрирующую работу указанных выше элементов класса.

**Решение**

**Листинг (исходный код программы)**

Листинг файла Jobless.cs

using System;

namespace \_6

{

class Jobless

{

private int joblessID;//регистрационный номер

private string lastName;//фамилия

private string firstName;//имя

private int birth;//возраст

public Jobless()

{

lastName = "Фамилия";

firstName = "Имя";

joblessID = birth = 0;

}

public Jobless(string lastName, string firstName, int joblessID, int birth)

{

this.joblessID = joblessID;

this.lastName = lastName;

this.firstName = firstName;

this.birth = birth;

}

public Jobless(Jobless p)

{

this.lastName = p.LastName;

this.firstName = p.FirstName;

this.birth = p.Birth;

this.joblessID = p.JoblessID;

}

public string LastName

{

get { return lastName; }

}

public string FirstName

{

get { return firstName; }

}

public int JoblessID

{

get { return joblessID; }

}

public int Birth

{

get { return birth; }

}

public int GetAge()

{

return birth;

}

public static bool operator >(Jobless l, Jobless r)

{

return l.birth > r.birth;

}

public static bool operator <(Jobless l, Jobless r)

{

return l.birth < r.birth;

}

public static bool operator ==(Jobless l, Jobless r)

{

return ((l.Birth == r.Birth) &&

(l.FirstName == r.FirstName) &&

(l.LastName == r.LastName) &&

(l.JoblessID == r.JoblessID));

}

public static bool operator !=(Jobless l, Jobless r)

{

return !(l == r);

}

public override string ToString()

{

return string.Format("{0}: {1} {2}, {3} лет", joblessID, lastName, firstName, birth);

}

}

}

Листинг файла Program.cs

using System;

namespace \_6

{

class Program

{

static void Main(string[] args)

{

DateTime now = DateTime.Now;

Jobless[] obj = new Jobless[4];

obj[0] = new Jobless();

obj[1] = new Jobless("Тесла", "Никола", 356, 33);

obj[2] = new Jobless(obj[1]);

obj[3] = new Jobless("Маяковский", "Владимир", 17, 29);

for (int i = 0; i < obj.Length; ++i)

Console.WriteLine("obj[{0}]: {1}", i, obj[i].ToString());

Console.WriteLine();

for (int i = 0; i < obj.Length - 1; ++i)

{

Console.WriteLine("(obj[{0}] == obj[{1}]) = {2}", i, i + 1, (obj[i] == obj[i + 1]));

Console.WriteLine("(obj[{0}] != obj[{1}]) = {2}", i, i + 1, (obj[i] != obj[i + 1]));

Console.WriteLine("(obj[{0}] > obj[{1}]) = {2}", i, i + 1, (obj[i] > obj[i + 1]));

Console.WriteLine("(obj[{0}] < obj[{1}]) = {2}", i, i + 1, (obj[i] < obj[i + 1]));

Console.WriteLine("obj[{0}].GetAge() = {1}", i, obj[i].GetAge());

Console.WriteLine();

}

Console.WriteLine("obj[3].GetAge() = {0}", obj[3].GetAge());

Console.WriteLine();

//Сортировка безработных по возрастанию

for (int i = 0; i < obj.Length - 1; ++i)

for (int j = i+1; j < obj.Length; ++j)

{

if(obj[i]>obj[j])

{

Jobless t = obj[i];

obj[i] = obj[j];

obj[j] = t;

}

}

Console.WriteLine("Массив объектов после сортировки");

for (int i = 0; i < obj.Length; ++i)

Console.WriteLine("obj[{0}]: {1}", i, obj[i].ToString());

Console.Read();

}

}

}

**Результат работы программы**

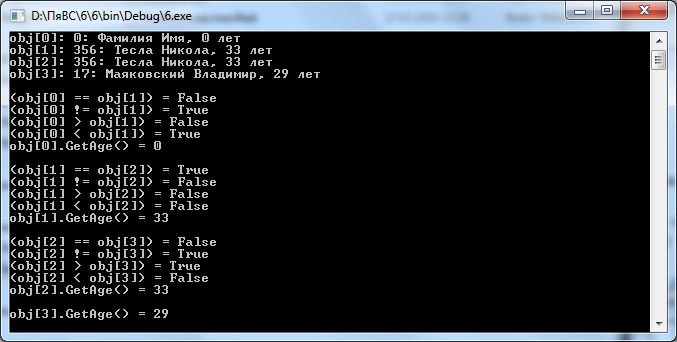


Рисунок 13 – Результат работы программы

# Задание 4

**Задание**

Разработать класс LongJobless, являющийся наследником класса Jobless (безработный), добавив следующие поля:

* date (дата постановки на учёт),
* reason (причина отсутствия работы).

**Решение**

Листинг класса Jobless.cs представлен в предыдущем задании. Его исходный код не был изменен.

Листинг файла LongJobless.cs

using System;

using \_6;

namespace \_7

{

class LongJobless : Jobless

{

private DateTime date;//(дата постановки на учёт),

private string reason;//(причина отсутствия работы)

public LongJobless() : base()

{

date = DateTime.Now;

reason = "";

}

public LongJobless(LongJobless lj) : base(lj)

{

date = lj.Date;

reason = lj.Reason;

}

public LongJobless(string lastName, string firstName, int joblessID, int birth, DateTime date, string reason) : base(lastName, firstName, joblessID, birth)

{

this.date = date;

this.reason = reason;

}

public DateTime Date

{

get { return date; }

}

public string Reason

{

get { return reason; }

}

public override string ToString()

{

return string.Format("{0}. Дата: {1}, причина: {2}", base.ToString(), date.ToString("dd.MM.yyyy"), reason);

}

}

}

Листинг файла Program.cs

using System;

namespace \_7

{

class Program

{

static void Main(string[] args)

{

DateTime now = DateTime.Now;

LongJobless[] obj = new LongJobless[4];

obj[0] = new LongJobless();

obj[1] = new LongJobless("Тесла", "Никола", 356, 33, DateTime.Now.AddYears(-1), "Безработица");

obj[2] = new LongJobless(obj[1]);

obj[3] = new LongJobless("Маяковский", "Владимир", 17, 29, DateTime.Now.AddMonths(-6), "Кризис");

for (int i = 0; i < obj.Length; ++i)

Console.WriteLine("obj[{0}]: {1}", i, obj[i].ToString());

Console.WriteLine();

for (int i = 0; i < obj.Length - 1; ++i)

{

Console.WriteLine("(obj[{0}] == obj[{1}]) = {2}", i, i + 1, (obj[i] == obj[i + 1]));

Console.WriteLine("(obj[{0}] != obj[{1}]) = {2}", i, i + 1, (obj[i] != obj[i + 1]));

Console.WriteLine("(obj[{0}] > obj[{1}]) = {2}", i, i + 1, (obj[i] > obj[i + 1]));

Console.WriteLine("(obj[{0}] < obj[{1}]) = {2}", i, i + 1, (obj[i] < obj[i + 1]));

Console.WriteLine("obj[{0}].GetAge() = {1}", i, obj[i].GetAge());

Console.WriteLine();

}

Console.WriteLine("obj[3].GetAge() = {0}", obj[3].GetAge());

Console.WriteLine();

Console.Read();

}

}

}

**Результат работы программы**

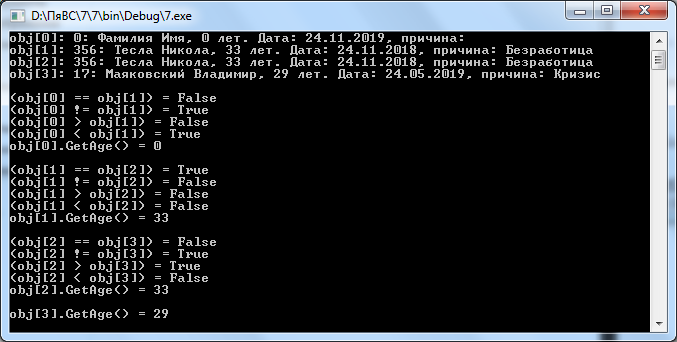


Рисунок 14 – Результат работы программы