Контрольная работа.
Часть 1
Основные языковые конструкции языка C#

Требуется разработать набор консольных приложений на языке C#.
В контрольную занести код с комментариями, таблицы тестов и результатов выполнения программ.
Разработка проводится в IDE Microsoft Visual Studio, .Net Framework , язык разработки — C#.
Названия давать таким образом, чтобы по названию было понятно, для чего предназначена переменная, метод и т.д.

1 задание
Оператор Switch Case
Составить программу, которая рассчитывает значение y(x), согласно введенному вещественному числу x с помощью конструкции Switch Case. Кроме этого, программа должна выводить текст сработавшего условия.

Значение y(x)	Условие
x - 100,	при x = -2;
x - 200,	при x = 4, -5, 11;
x - 300,	при x < -5;
x - 400,	при x от 32 до 70;
x - 500.	при других значениях.

Оператор switch обеспечивает многонаправленное ветвление программы. Следовательно, этот оператор позволяет сделать выбор среди нескольких альтернативных вариантов дальнейшего выполнения программы. Несмотря на то что многонаправленная проверка может быть организована с помощью последовательного ряда вложенных операторов if, во многих случаях более эффективным оказывается применение оператора switch. Этот оператор действует следующим образом. Значение выражения последовательно сравнивается с константами выбора из заданного списка. Как только будет обнаружено совпадение с одним из условий выбора, выполняется последовательность операторов, связанных с этим условием. Ниже приведена общая форма оператора switch:

```
switch(выражение) {
        case константа1:
                последовательность операторов
                break;
        case константа2:
                последовательность операторов
                break;
        case константа3:
                последовательность операторов
                break;
                ...
        default:
                последовательность операторов
                break;
}
```

Пример 2

```
using System;

namespace ConsoleApplication1
{
   class Program
   {
```

```
        static void Main(string[] args)
        {
            Console.WriteLine("Введите язык (C#, VB или C++)");
            string myLanguage = Console.ReadLine();

            sw1(myLanguage);

            Console.ReadLine();
        }

        // Данный метод выводит выбор пользователя
        static void sw1(string s)
        {
            switch (s)
            {
                case "C#":
                    Console.WriteLine("Вы выбрали язык C#");
                    break;
                case "VB":
                    Console.WriteLine("Вы выбрали язык Visual Basic");
                    break;
                case "C++":
                    Console.WriteLine("Вы выбрали язык C++");
                    break;
                default:
                    Console.WriteLine("Такой язык я не знаю");
                    break;
            }
        }
    }
}
```

2 задание
Цикл с постусловием, цикл с предусловием
Написать программу, вычисляющую частичную сумму ряда Sn с заданной пользователем точностью.

Значение Sn
$S(n) \quad \sum_{i=1}^{n} \frac{\sqrt{i+1}}{(4*i-3)*a^i}$

Подобно for, while также является циклом с предварительной проверкой. Синтаксис его аналогичен, но циклы while включают только одно выражение:

while(условие)
оператор (операторы);
где оператор — это единственный оператор или же блок операторов, а условие означает конкретное условие управления циклом и может быть любым логическим выражением. В этом цикле оператор выполняется до тех пор, пока условие истинно. Как только условие становится ложным, управление программой передается строке кода, следующей непосредственно после цикла.

Как и в цикле for, в цикле while проверяется условное выражение, указываемое в самом начале цикла. Это означает, что код в теле цикла может вообще не выполняться, а также избавляет от необходимости выполнять отдельную проверку перед самим циклом.

Пример 4.1

```
using System;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         // Пример возведения числа в несколько степеней
         byte l = 2, i = 0;
         int result = 1;

         while (i < 10)
         {
            i++;
            result *= l;
            Console.WriteLine("{0} в степени {1} равно {2}",l,i,result);
         }

         Console.ReadLine();
      }
   }
}
```

Цикл do...while в C# — это версия while с постпроверкой условия. Это значит, что условие цикла проверяется после выполнения тела цикла. Следовательно, циклы do...while удобны в тех ситуациях,

когда блок операторов должен быть выполнен как минимум однажды. Ниже приведена общая форма оператора цикла do-while:

```
do {
операторы;
} while (условие);
```

При наличии лишь одного оператора фигурные скобки в данной форме записи необязательны. Тем не менее они зачастую используются для того, чтобы сделать конструкцию do-while более удобочитаемой и не путать ее с конструкцией цикла while. Цикл do-while выполняется до тех пор, пока условное выражение истинно.

Пример 4.2

```
using System;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            // Вычисляем факториал числа
            int i, result = 1, num = 1;

            Console.WriteLine("Введите число:");
            i = int.Parse(Console.ReadLine());

            Console.Write("\n\nФакториал {0} = ", i);
            do
            {
               result *= num;
               num++;
            } while (num <= i);

            Console.Write(result);
         }
         catch (FormatException ex)
         {
            Console.WriteLine("Вы ввели не число. {0}",ex.Message);
         }
         finally
         {
            Console.ReadLine();
         }
      }
   }
}
```

Часть 2
Основы объектно-ориентированного программирования

Требуется разработать приложение из нескольких классов и перечисления и протестировать приложение.
Разработка проводится в IDE Microsoft Visual Studio, .Net Framework 3.5 или 4.5, язык разработки — C#.
Разработка может проводиться при помощи диаграммы классов.
Названия давать таким образом, чтобы по названию было понятно, для чего предназначена переменная, метод и т.д.
В контрольной работе должны быть приведены таблицы для описания членов классов, код и диаграмма классов.

0. Общее задание (изучить и перенести код в свое приложение).
Определить класс Person, который имеет
– закрытое поле типа string, в котором хранится имя;
– закрытое поле типа string, в котором хранится фамилия;
– закрытое поле типа DateTime для даты рождения.
В классе Person определить конструкторы:
– конструктор с тремя параметрами типа string, string, DateTime для инициализации всех полей класса;
– конструктор без параметров, инициализирующий все поля класса некоторыми значениями по умолчанию.
В классе Person определить свойства с методами get и set:
– свойство типа string для доступа к полю с именем;
– свойство типа string для доступа к полю с фамилией;
– свойство типа DateTime для доступа к полю с датой рождения;
– свойство типа int с методами get и set для получения информации (get) и изменения (set) года рождения в закрытом поле типа DateTime, в котором хранится дата рождения.
В классе Person определить:
– перегруженную (override) версию виртуального метода string ToString() для формирования строки со значениями всех полей класса;
– виртуальный метод string ToShortString(), который возвращает строку, содержащую только имя и фамилию.

Таблица 1 — Члены класса Person

Член класса	Название	Тип	Модификатор доступа	Параметры	Комментарий
Поле	_name	string	private		Имя
Поле	_surname	string	private		Фамилия
Поле	_birth	DateTime	private		Дата рождения
Конструктор	Person		public	string name, string surname, DateTime birth	Конструктор с параметрами
Конструктор	Person		public		Конструктор без параметров
Свойство	Name	string	public		Имя
Свойство	Surname	string	public		Фамилия
Свойство	Birth	DateTime	public		Дата рождения
Свойство	Year	int	public		Год рождения
Метод	ToString	string	public		Возвращает все поля класса
Метод	ToShortString	string	public		Возвращает имя и фамилию

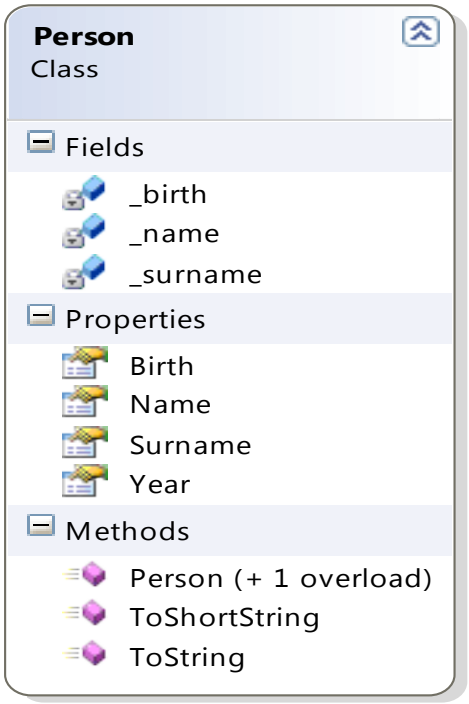

Рисунок 1 — Класс Person в диаграмме классов

Листинг 1. Класс Person с комментариями:

```
using System;

namespace KR2
{
    /// <summary>
    /// Класс, предназначенный для хранения персональных данных
    /// </summary>
    public class Person
    {
        private string _name;    // закрытое поле типа string, в котором хранится имя
        private string _surname; // закрытое поле типа string, в котором хранится фамилия
        private DateTime _birth; // закрытое поле типа DateTime для даты рождения

        /// <summary>
        /// Конструктор с тремя параметрами для инициализации всех полей класса
        /// </summary>
        /// <param name="name">Имя</param>
        /// <param name="surname">Фамилия</param>
        /// <param name="birth">Дата рождения</param>
        public Person(string name, string surname, DateTime birth)
        {
            _name = name;
            _surname = surname;
            _birth = birth;
        }

        /// <summary>
        /// Конструктор без параметров, инициализирующий все поля класса
        /// некоторыми значениями по умолчанию.
        /// Используется ссылка на конструктор с параметрами this(параметры).
        /// </summary>
        public Person()
            : this("UnknounName", "UnknownSurname", DateTime.Now)
```

```
{
}

/*/// <summary>
/// Конструктор без параметров, инициализирующий все поля класса
/// можно реализовать без ссылки на конструктор с параметрами.
/// Результаты будут идентичны
/// </summary>
public Person()
{
_name = "UnknounName";
_surname = "UnknownSurname";
_birth = DateTime.Now;
}*/

/// <summary>
/// Свойство типа DateTime для доступа к полю с датой рождения
/// </summary>
public DateTime Birth
{
    get
    {
        return _birth;
    }
    set
    {
        _birth = value;
    }
}

/// <summary>
/// Свойство типа string для доступа к полю с именем
/// </summary>
public string Name
{
    get
    {
        return _name;
    }
    set
    {
        _name = value;
    }
}

/// <summary>
/// Свойство типа string для доступа к полю с фамилией
/// </summary>
public string Surname
{
    get
    {
        return _surname;
    }
    set
    {
```

```
            _surname = value;
        }
    }

    /// <summary>
    /// Свойство типа int с методами get и set
    /// для получения информации (get) и изменения (set) года рождения
    /// в закрытом поле типа DateTime, в котором хранится дата рождения
    /// </summary>
    public int Year
    {
        get
        {
            return _birth.Year;
        }
        set
        {
            // т.к. свойство Year в переменных типа DateTime только на чтение,
            // изменяем год рождения, создавая дату заново, указывая вместо года value
            _birth = new DateTime(value, _birth.Month, _birth.Day);
        }
    }

    /// <summary>
    /// Перегруженная версия виртуального метода string ToString()
    /// для формирования строки со значениями всех полей класса
    /// </summary>
    /// <returns>Отформатированная строка</returns>
    public override string ToString()
    {
        return String.Format("Имя: {0}\nФамилия: {1}\nДата рождения: {2}",
            _name, _surname, _birth.ToShortDateString());
    }

    /// <summary>
    /// Виртуальный метод string ToShortString(),
    /// который возвращает строку, содержащую только имя и фамилию
    /// </summary>
    /// <returns>Отформатированная строка</returns>
    public virtual string ToShortString()
    {
        return String.Format("Имя: {0}\nФамилия: {1}", _name, _surname);
    }
  }
}
```

Листинг 2. Создание экземпляров класса Person в методе Main

```
using System;

namespace KR2
{
  class Program
  {
    static void Main(string[] args)
    {
      // Объявили экземпляр класса и вызвали конструктор по умолчанию
```

```
            Person pers1 = new Person();
            Console.WriteLine("Экземпляр класса с полями, заполненными по умолчанию:");
            Console.WriteLine(pers1.ToString());
            Console.WriteLine(pers1.ToShortString());

            // Объявили экземпляр класса
            Person pers2;
            Console.WriteLine();
            Console.WriteLine("Введите имя, фамилию, дату рождения через Enter");
            // Вызвали конструктор с параметрами, в которые считываем данные
            try
            {
                pers2 = new Person(
                    Console.ReadLine(),             // считываем имя
                    Console.ReadLine(),             // считываем фамилию
                    DateTime.Parse(Console.ReadLine()));// считываем дату рождения
            }
            catch (Exception ex)
            {
                Console.WriteLine("Возникла следующая ошибка: {0}", ex.Message);
                pers2 = new Person();
            }
            Console.WriteLine("\nРезультат заполнения:\n{0}", pers2.ToString());

            // Изменяем поля имя, фамилия, год, используя свойства
            pers2.Name = "Полиграф";
            pers2.Surname = "Шариков";
            pers2.Year = 1925;
            Console.WriteLine("\nИзмененные свойства объекта pers2:");
            Console.WriteLine(pers2.ToString());
        }
    }
}
```

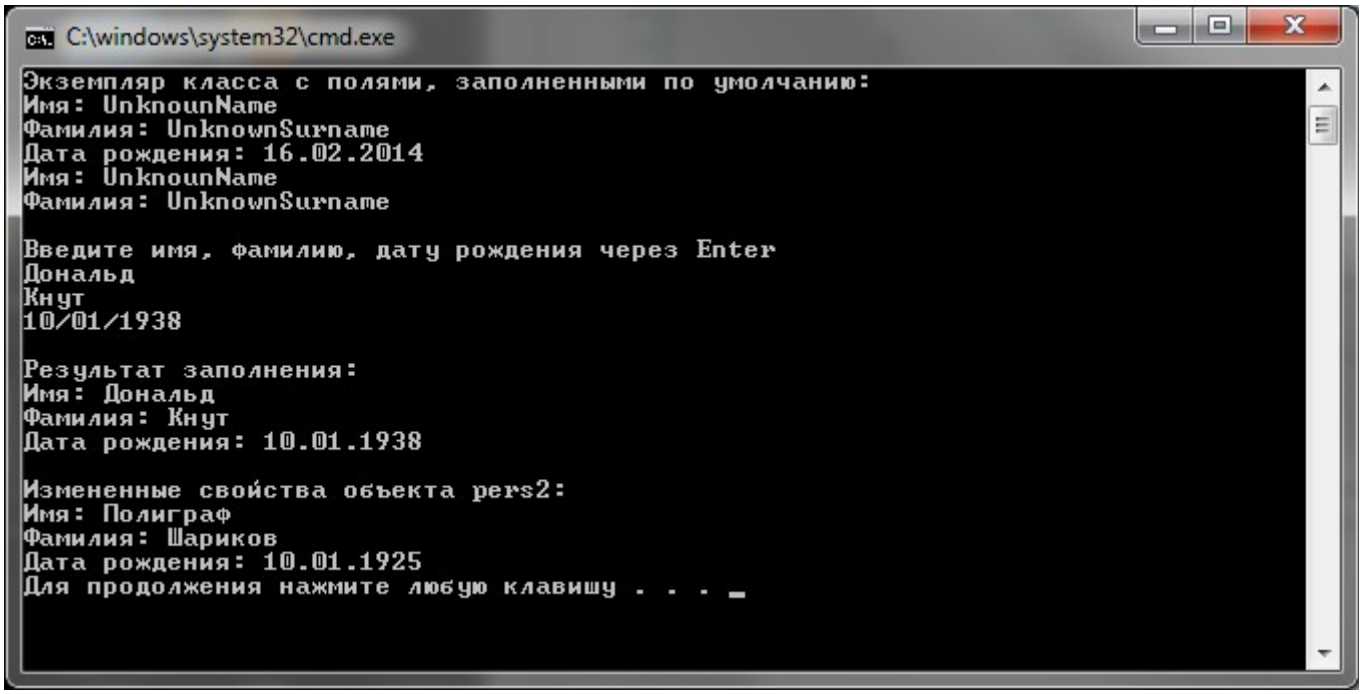

Рисунок 1 — Результат выполнения программы

Задание

1 Определить тип Form – перечисление (enum) со значениями OOO, OAO, ZAO для хранения информации о правовой форме предприятия.

2 Определить класс Employee, который имеет три открытых автореализуемых свойства, доступных для чтения и записи:
– свойство типа Person для информации о сотруднике;
– свойство типа string, в котором хранится должность;
– свойство типа DateTime с датой принятия на работу.
В классе Employee определить
– конструктор с параметрами типа Person, string, DateTime для инициализации всех свойств класса;
– конструктор без параметров, инициализирующий все свойства класса некоторыми значениями по умолчанию;
– перегруженную (override) версию виртуального метода string ToString() для формирования строки со значениями всех полей класса.

3 Определить класс Organization, который имеет
– закрытое поле типа string с названием организации;
– закрытое поле типа int – регистрационный номер;
– закрытое поле типа Form – правовая форма организации;
– закрытое поле типа Employee[], в котором хранится список работников.
В классе Organization определить конструкторы:
– конструктор с параметрами типа string, int, Form для инициализации соответствующих полей класса;
– конструктор без параметров, инициализирующий поля класса значениями
по умолчанию.
В классе Organization определить свойства с методами get и set:
– свойство типа string для доступа к полю с названием темы исследований;
– свойство типа int для доступа к полю с номером регистрации;
– свойство типа Form для доступа к полю с правовой формой организации;
– свойство типа Employee[] для доступа к полю со списком работников.
В классе Organization определить
– свойство типа Person (только с методом get), которое возвращает ссылку на рабочего с самой ранней датой приема на работу; если список рабочих пустой, свойство возвращает значение null;
– индексатор булевского типа (только с методом get) с одним параметром типа Form; значение индексатора равно true, если значение поля с информацией о правовой форме совпадает со значением индекса, и false в противном случае;
– метод void AddEmployee(params Employee[] employees) для добавления элементов в список рабочих;
– перегруженную версию виртуального метода string ToString() для формирования строки со значениями всех полей класса, включая список рабочих;
– виртуальный метод string ToShortString(), который формирует строку со значениями всех полей класса без списка рабочих.

4 В методе Main()
4.1 Создать один объект типа Organization, преобразовать данные в текстовый вид с помощью метода ToShortString() и вывести данные.
4.2 Вывести значения индексатора для значений индекса Form.OOO, Form.OAO, Form.ZAO.
4.3 Присвоить значения всем определенным в типе Organization свойствам, преобразовать данные в текстовый вид с помощью метода ToString() и вывести данные.
4.4 С помощью метода AddEmployee(params Employee[] employees) добавить элементы в список публикаций и вывести данные объекта Organization.
4.5 Вывести значение свойства, возвращающего ссылку на рабочего с самой ранней датой приема на работу.