

## Оглавление

1. Задание .....	2
2. Результат работы программы .....	6
3. Листинг .....	8
Файл Program.cs .....	8
Файл IDateAndCopy.cs .....	9
Файл StudentEnumerator.cs .....	10
Файл Student.cs .....	11
Файл Test.cs .....	16
Файл Exam.cs .....	17
Файл Person.cs .....	19

# 1. Задание

Определить интерфейс

```
interface IDateAndCopy
{
    object DeepCopy();
    DateTime Date { get; set; }
}
```

Определить новые версии классов **Exam**, **Person** и **Student** из лабораторной работы 1. В классы **Exam**, **Person** и **Student** добавить реализацию интерфейса **IDateAndCopy**. Новую версию класса **Student** определить как класс, производный от класса **Person**.

Все поля новой версии класса **Person** определить с доступом `protected`, сохранить все свойства, определенные в первой версии класса.

В новой версии класса **Person** дополнительно

- переопределить метод `virtual bool Equals(object obj)` и определить операции `==` и `!=` так, чтобы равенство объектов типа **Person** трактовалось как совпадение всех данных объектов, а не ссылок на объекты **Person**;
- переопределить виртуальный метод `int GetHashCode()`;
- определить виртуальный метод `object DeepCopy()`;
- реализовать интерфейс **IDateAndCopy**.

Определить класс **Test**, который имеет два открытых автореализуемых свойства, доступных для чтения и записи:

- свойство типа `string`, в котором хранится название предмета;
- свойство типа `bool` для информации о том, сдан зачет или нет.

В классе **Test** определить:

- конструктор с параметрами типа `string` и `bool` для инициализации 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 свойств класса.

Класс **Student** определить как производный от класса **Person**.

Новая версия класса **Student** имеет следующие поля:

- закрытое поле типа `Education` для информации о форме обучения;
- закрытое поле типа `int` для номера группы;
- закрытое поле типа `System.Collections.ArrayList`, в котором хранится список зачетов (объекты типа **Test**);
- закрытое поле типа `System.Collections.ArrayList` для списка экзаменов (объекты типа **Exam**).

Класс **Student** определить как производный от класса **Person**.

Новая версия класса **Student** имеет следующие поля:

- закрытое поле типа **Education** для информации о форме обучения;
- закрытое поле типа **int** для номера группы;
- закрытое поле типа **System.Collections.ArrayList**, в котором хранится список зачетов (объекты типа **Test**);
- закрытое поле типа **System.Collections.ArrayList** для списка экзаменов (объекты типа **Exam**).

Код следующих конструкторов, методов и свойств из старой версии класса **Student** необходимо изменить с учетом того, что часть полей класса перемещена в базовый класс **Person**, и в новой версии класса **Student** список экзаменов хранится в коллекции **System.Collections.ArrayList**:

- конструктор с параметрами типа **Person**, **Education**, **int** для инициализации соответствующих полей класса;
- конструктор без параметров для инициализации по умолчанию;
- свойство типа **Person**; метод **get** свойства возвращает объект типа **Person**, данные которого совпадают с данными подобъекта базового класса, метод **set** присваивает значения полям из подобъекта базового класса;
- свойство типа **double** (только с методом **get**), в котором вычисляется средний балл как среднее значение оценок в списке сданных экзаменов;
- свойство типа **System.Collections.ArrayList** с методами **get** и **set** для доступа к полю со списком экзаменов;
- метод **void AddExams (params Exam[] )** для добавления элементов в список экзаменов;
- перегруженная версия виртуального метода **string ToString()** для формирования строки со значениями всех полей класса, включая список зачетов и экзаменов;
- виртуальный метод **string ToShortString()**, который формирует строку со значениями всех полей класса без списка зачетов и экзаменов, но со значением среднего балла.

Дополнительно в новой версии класса **Student**

- определить перегруженную версию виртуального метода **object DeepCopy()**;
- реализовать интерфейс **IDateAndCopy**;
- определить свойство типа **int** с методами **get** и **set** для доступа к полю с номером группы. В методе **set** бросить исключение, если присваиваемое значение меньше или равно 100 или больше 599. При создании объекта-исключения использовать один из определенных в библиотеке CLR классов-исключений, инициализировать объект-исключение с помощью конструктора с параметром типа **string**, в сообщении передать информацию о допустимых границах для значения свойства.

В новой версии класса **Student** определить

- итератор для последовательного перебора всех элементов (объектов типа `object`) из списков зачетов и экзаменов (объединение);
- итератор с параметром для перебора экзаменов (объектов типа `Exam`) с оценкой больше заданного значения.

В методе **Main()**

1. Создать два объекта типа `Person` с совпадающими данными и проверить, что ссылки на объекты не равны, а объекты равны, вывести значения хэш-кодов для объектов.
2. Создать объект типа `Student`, добавить элементы в список экзаменов и зачетов, вывести данные объекта `Student`.
3. Вывести значение свойства типа `Person` для объекта типа `Student`.
4. С помощью метода `DeepCopy()` создать полную копию объекта `Student`. Изменить данные в исходном объекте `Student` и вывести копию и исходный объект, полная копия исходного объекта должна остаться без изменений.
5. В блоке `try/catch` присвоить свойству с номером группы некорректное значение, в обработчике исключения вывести сообщение, переданное через объект-исключение.
6. С помощью оператора `foreach` для итератора, определенного в классе `Student`, вывести список всех зачетов и экзаменов.
7. С помощью оператора `foreach` для итератора с параметром, определенного в классе `Student`, вывести список всех экзаменов с оценкой выше 3.



## Дополнительное задание:

### В классе **Student**

- реализовать интерфейс `System.Collections.IEnumerable` для перебора названий всех предметов (объектов типа `string`), которые есть как в списке зачетов, так и в списке экзаменов (пересечение). Для этого определить вспомогательный класс `StudentEnumerator`, реализующий интерфейс `System.Collections.IEnumerable`.
- определить итератор для перебора сданных зачетов и экзаменов (объектов типа `object`), для этого определить метод, содержащий блок итератора и использующий оператор `yield`; сданный экзамен - экзамен с оценкой больше 2;
- определить итератор для перебора всех сданных зачетов (объектов типа `Test`), для которых сдан и экзамен, для этого определить метод, содержащий блок итератора и использующий оператор `yield`.

### В методе **Main()**

8. С помощью оператора `foreach` для объекта типа `Student` вывести список предметов, которые есть как в списке зачетов, так и в списке экзаменов.
9. С помощью оператора `foreach` для итератора, определенного в классе `Student`, вывести список всех сданных зачетов и сданных экзаменов.
10. С помощью оператора `foreach` для итератора, определенного в классе `Student`, вывести список сданных зачетов, для которых сдан и экзамен.

## 2. Результат работы программы

Выполнение задания по вариантам:

1. -----

```
is Equals(p1,p2): True
is ==: True
GetHashCode p1: 27252167
GetHashCode p2: 43942917
```

-----

2. -----

```
Student ToString: forename surname 01.01.0001 0:00:00 Specialist 407, AvgRat: 3,25 Exams:
- subject1 1, Date:13.05.2024 12:05:16
- subject5 5, Date:13.05.2024 12:05:16
- subject4 4, Date:13.05.2024 12:05:16
- subject3 3, Date:13.05.2024 12:05:16
```

-----

3. -----

```
Student Person: forename surname 01.01.0001 0:00:00 Specialist 407, AvgRat: 3,25 Exams:
- subject1 1, Date:13.05.2024 12:05:16
- subject5 5, Date:13.05.2024 12:05:16
- subject4 4, Date:13.05.2024 12:05:16
- subject3 3, Date:13.05.2024 12:05:16
```

-----

4. -----

```
Student ToString: forename surname 13.05.2024 12:05:16 Specialist 407, AvgRat: 3 Exams:
- subject1 1, Date:13.05.2024 12:05:16
- subject5 5, Date:13.05.2024 12:05:16
- subject4 4, Date:13.05.2024 12:05:16
- subject3 3, Date:13.05.2024 12:05:16
- subject2 2, Date:13.05.2024 12:05:16
```

```
StudentCopy ToString: forename surname 01.01.0001 0:00:00 Specialist 407, AvgRat: 3,25 Exams:
- subject1 1, Date:13.05.2024 12:05:16
- subject5 5, Date:13.05.2024 12:05:16
- subject4 4, Date:13.05.2024 12:05:16
- subject3 3, Date:13.05.2024 12:05:16
```

-----

5. -----

System.Exception: Недопустимый номер группы, границы значений от 101 до 599  
at LR2.Student.set\_NumberGroup(Int32 value) in C:\Users\iDemetr\source\repos\MIET\Course C#\LR 2\Student.cs:line 59  
at LR2.Program.Main(String[] args) in C:\Users\iDemetr\source\repos\MIET\Course C#\LR 2\Program.cs:line 58

-----

6. -----

- subject1 1, Date:13.05.2024 12:05:16
- subject5 5, Date:13.05.2024 12:05:16
- subject4 4, Date:13.05.2024 12:05:16
- subject3 3, Date:13.05.2024 12:05:16
- subject2 2, Date:13.05.2024 12:05:16
- subject1 сдан.
- subject5 не сдан.
- subject4 сдан.
- subject3 сдан.
- subject7 сдан.
- subject2 не сдан.

-----

7. -----

- subject5 5, Date:13.05.2024 12:05:16
- subject4 4, Date:13.05.2024 12:05:16
- subject3 3, Date:13.05.2024 12:05:16

-----

### Выполнение дополнительного задания:

8. -----

- subject1
- subject5
- subject4
- subject3
- subject2

-----

9. -----

- subject5 5, Date:13.05.2024 12:05:16
- subject4 4, Date:13.05.2024 12:05:16
- subject3 3, Date:13.05.2024 12:05:16
- subject1 сдан.
- subject4 сдан.
- subject3 сдан.
- subject7 сдан.

-----

10. -----

- subject4 сдан.
- subject3 сдан.

-----

### 3. ЛИСТИНГ

```
//-----||
// Файл Program.cs
//-----||

namespace LR2
{
    internal class Program
    {
        static void Main(string[] args)
        {
            var p1 = new Person("forename", "surname", DateTime.MinValue);
            var p2 = new Person("forename", "surname", DateTime.MinValue);

            Console.WriteLine("\n1. ----- \n");
            Console.WriteLine("    is Equals(p1,p2): " + p1.Equals(p2));
            Console.WriteLine("    is ==: " + (p1 == p2));
            Console.WriteLine("    GetHashCode p1: " + p1.GetHashCode());
            Console.WriteLine("    GetHashCode p2: " + p2.GetHashCode());
            Console.WriteLine("\n ----- \n");

            var student = new Student(p1, Education.Specialist, 407);
            student.AddExams(new System.Collections.ArrayList {
                new Exam("subject1", 1, DateTime.Now),
                new Exam("subject5", 5, DateTime.Now),
                new Exam("subject4", 4, DateTime.Now),
                new Exam("subject3", 3, DateTime.Now)
            });

            student.Subjects = (new System.Collections.ArrayList {
                new Test("subject1", true),
                new Test("subject5", false),
                new Test("subject4", true),
                new Test("subject3", true),
                new Test("subject7", true)
            });

            Console.WriteLine("\n2. ----- \n");
            Console.WriteLine("    Student ToString: " + student.ToString());
            Console.WriteLine("\n ----- \n");

            Console.WriteLine("\n3. ----- \n");
            Console.WriteLine("    Student Person: " + (student as Person).ToString());
            Console.WriteLine("\n ----- \n");

            var studentCopy = student.DeepCopy() as Student;

            student.BirthData = DateTime.Now;
            student.Subjects.Add(new Test("subject2", false));
            student.AddExams(new System.Collections.ArrayList { new Exam("subject2", 2,
DateTime.Now) });

            Console.WriteLine("\n4. ----- \n");
            Console.WriteLine("    Student ToString: " + student.ToString());
            Console.WriteLine("    StudentCopy ToString: " + studentCopy?.ToString());
            Console.WriteLine("\n ----- \n");

            try
            {
                student.NumberGroup = 1;
            }
            catch (Exception ex)
            {
                Console.WriteLine("\n5. ----- \n");
                Console.WriteLine(ex.ToString());
                Console.WriteLine("\n ----- \n");
            }
        }
    }
}
```



```

        Console.WriteLine("\n6. ----- \n");
        foreach (var item in student.GetSubjects())
        {
            Console.WriteLine(" - " + item.ToString());
        }

        Console.WriteLine("\n ----- \n");
        Console.WriteLine("\n7. ----- \n");

        foreach (var item in student.GetExam(3))
        {
            Console.WriteLine(" - " + item.ToString());
        }

        Console.WriteLine("\n ----- \n");
        Console.WriteLine("\n8. ----- \n");

        foreach (var item in student)
        {
            Console.WriteLine(" - " + item.ToString());
        }

        Console.WriteLine("\n ----- \n");
        Console.WriteLine("\n9. ----- \n");

        foreach (var item in student.GetPassedSubjectsAndExams())
        {
            Console.WriteLine(" - " + item.ToString());
        }

        Console.WriteLine("\n ----- \n");
        Console.WriteLine("\n10. ----- \n");

        foreach (var item in student.GetPassedSubjects())
        {
            Console.WriteLine(" - " + item.ToString());
        }

        Console.WriteLine("\n ----- \n");
    }
}

```

```

//----- ||
// Файл IDateAndCopy.cs
//----- ||

```

```

namespace LR2
{
    internal interface IDateAndCopy
    {
        object DeepCopy();
        DateTime Date { get; set; }
    }
}

```

```

//-----||
// Файл StudentEnumerator.cs
//-----||

using System.Collections;

namespace LR2
{
    /// <summary>
    /// По доп требованиям:
    /// Реализовать интерфейс System.Collections.IEnumerable для перебора
    /// названий всех предметов(объектов типа string), которые есть как в списке
    /// зачетов, так и в списке экзаменов(пересечение). Для этого определить
    /// вспомогательный класс StudentEnumerator, реализующий интерфейс
    /// System.Collections.IEnumerator.
    /// </summary>
    internal class StudentEnumerator : IEnumerator
    {
        Student _student;

        int position = -1;

        public object Current => (_student.ClosedExams[position] as Exam).NameSubject;

        public StudentEnumerator(Student student)
        {
            _student = student;
        }

        public bool MoveNext()
        {
            // Поиск индексов
            while (++position < _student.ClosedExams.Count)
            {
                Exam exam = _student.ClosedExams[position] as Exam;
                if (exam != null)
                {
                    foreach (Test subject in _student.Subjects)
                    {
                        if (subject.SubjectName == exam.NameSubject)
                        {
                            return true;
                        }
                    }
                }
            }

            return false;
        }

        public void Reset() => position = -1;
    }
}

```

```
//-----||
// Файл Student.cs
//-----||

using System.Collections;

namespace LR2
{
    internal class Student : Person, IDateAndCopy, IEnumerable
    {
        #region --- Переменные ---
        Person info => this;

        /// <summary>
        /// Форма обучения
        /// </summary>
        Education education;

        /// <summary>
        /// Номер группы
        /// </summary>
        int numberGroup;

        /// <summary>
        /// Закрытые экзамены
        /// </summary>
        ArrayList closedExams;

        /// <summary>
        ///
        /// </summary>
        ArrayList subjects;

        #endregion

        #region --- Свойства ---

        /// <summary>
        /// Средний балл
        /// </summary>
        double AvgRating {
            get {
                if (closedExams == null || closedExams.Count == 0) return 0;

                return (double)(closedExams.ToArray()?.Average((ex) => { return (ex as
Exam)?.Rating ?? 0.0; }));
            }
        }

        internal Education Education
        {
            get { return education; }
            set { education = value; }
        }

        internal int NumberGroup
        {
            get { return numberGroup; }
            set {
                if (value <= 100 || value > 599)
                    throw new Exception("Недопустимый номер группы, границы значений от 101 до
599");
                numberGroup = value;
            }
        }

        internal ArrayList ClosedExams
        {
            get { return closedExams; }
            set { closedExams = value; }
        }
    }
}
```

```

internal ArrayList Subjects
{
    get { return subjects; }
    set { subjects = value; }
}

#endregion

/// <summary>
/// Итератор для последовательного перебора всех элементов (объектов типа object)
/// из списков зачетов и экзаменов(объединение)
/// </summary>
/// <returns></returns>
public IEnumerable<object> GetSubjects()
{
    foreach (Exam ex in closedExams)
    {
        yield return ex;
    }

    foreach (Test subject in subjects)
    {
        yield return subject;
    }
}

/// <summary>
/// Итератор с параметром для перебора экзаменов (объектов типа Exam)
/// с оценкой больше заданного значения.
/// </summary>
/// <param name="targetRating"></param>
/// <returns></returns>
public IEnumerable<Exam> GetExam(int targetRating)
{
    foreach (Exam ex in closedExams)
    {
        if (ex.Rating >= targetRating)
            yield return ex;
    }
}

/// <summary>
/// По доп требованиям:
/// определить итератор для перебора сданных зачетов и экзаменов
/// (объектов типа object), для этого определить метод, содержащий блок
/// итератора и использующий оператор yield; сданный экзамен - экзамен с
/// оценкой больше 2;
/// </summary>
/// <param name="targetRating"></param>
/// <returns></returns>
public IEnumerable<object> GetPassedSubjectsAndExams()
{
    var iter = closedExams.GetEnumerator();
    while (iter.MoveNext())
    {
        if ((iter.Current as Exam).Rating > 2)
            yield return iter.Current;
    }

    iter = Subjects.GetEnumerator();
    while(iter.MoveNext())
    {
        if ((iter.Current as Test).isPassed)
            yield return iter.Current;
    }
}

```

```

/// <summary>
/// По доп требованиям:
/// определить итератор для перебора всех сданных зачетов (объектов
/// типа Test), для которых сдан и экзамен, для этого определить метод,
/// содержащий блок итератора и использующий оператор yield.
/// </summary>
/// <returns></returns>
public IEnumerable<Test> GetPassedSubjects()
{
    foreach (Exam exam in closedExams)
    {
        foreach (Test subject in Subjects)
        {
            if (subject.SubjectName == exam.NameSubject && exam.Rating > 2 &&
subject.isPassed)
            {
                yield return subject;
            }
        }
    }
}

/// <summary>
///
/// </summary>
/// <param name="education"></param>
/// <returns></returns>
internal bool this[Education education]
{
    get => education == this.education;
}

/// <summary>
///
/// </summary>
internal Student() {

    education = new ();
    numberGroup = 0;
    closedExams = new ArrayList();
}

/// <summary>
///
/// </summary>
/// <param name="info"></param>
/// <param name="education"></param>
/// <param name="numberGroup"></param>
internal Student(Person person, Education education, int numberGroup )
{
    base._BirthData = person.BirthData;
    base._Surname = person.Surname;
    base._Forename = person.Forename;

    this.education = education;
    this.numberGroup = numberGroup;
}

/// <summary>
/// Добавляет перечень экзаменов в коллекцию закрытых экзаменов
/// </summary>
/// <param name="exams">Закрытые экзамены</param>
internal void AddExams(ArrayList exams)
{
    if (closedExams != null)
    {
        if (exams != null)
            closedExams.AddRange((exams).ToArray());
    }
    else
        closedExams = exams;
}

```



```

public override string ToString()
{
    string exams = "";

    if (closedExams != null)
        foreach (var ex in ClosedExams)
        {
            exams += " - " + ex.ToString() + "\n";
        }

    return ToShortString() + " Exams:\n" + exams;
}

public virtual string ToShortString()
{
    return base.ToString() + " " + education.ToString() + " " + numberGroup + ", AvgRat:
" + AvgRating;
}

/// <summary>
///
/// </summary>
/// <param name="obj"></param>
/// <returns></returns>
public override bool Equals(object obj)
{
    Student tmp = obj as Student;
    return tmp != null &&
        base.Equals(tmp) &&
        this.education == tmp.education &&
        this.closedExams == tmp.closedExams &&
        this.AvgRating == tmp.AvgRating &&
        this.subjects == tmp.subjects &&
        this.numberGroup == tmp.numberGroup;
}

public static bool operator ==(Student p1, Student p2)
{
    if (p1 is null || p2 is null)
        return false;

    return p1.Equals(p2);
}

public static bool operator !=(Student p1, Student p2)
{
    return !(p1 == p2);
}

/// <summary>
///
/// </summary>
/// <returns></returns>
public override int GetHashCode()
{
    return info.GetHashCode() + education.GetHashCode() + numberGroup +
closedExams.GetHashCode() + subjects.GetHashCode();
}

```

```

/// <summary>
/// Нельзя перегрузить, ошибка CS0506
/// </summary>
/// <returns></returns>
public new object DeepCopy()
{
    var tmp = new Student((Person)info.DeepCopy(), education, numberGroup);
    tmp.closedExams = new ArrayList();
    tmp.subjects = new ArrayList();

    foreach (Exam exam in closedExams)
    {
        tmp.closedExams.Add(exam.DeepCopy());
    }

    foreach (Test subject in subjects)
    {
        tmp.subjects.Add(subject);
    }

    return tmp;
}

/// <summary>
/// По доп. требованиям
/// </summary>
/// <returns></returns>
IEnumerator IEnumerable.GetEnumerator()
{
    return new StudentEnumerator(this);
}
}
}

```

```
//-----||
// Файл Test.cs
//-----||

namespace LR2
{
    internal class Test
    {
        /// <summary>
        ///
        /// </summary>
        internal string SubjectName { get; set; }

        /// <summary>
        ///
        /// </summary>
        internal bool isPassed { get; set; }

        public Test() {
            SubjectName = string.Empty;
            isPassed = false;
        }

        public Test(string subjectName, bool isPassed)
        {
            SubjectName = subjectName;
            this.isPassed = isPassed;
        }

        public override string ToString()
        {
            return SubjectName + (isPassed ? " сдан." : " не сдан.");
        }
    }
}
```

```
//-----||  
// Файл Exam.cs  
//-----||
```

```
namespace LR2  
{  
    internal class Exam : IDateAndCopy  
    {  
        /// <summary>  
        /// Название предмета  
        /// </summary>  
        internal string NameSubject { get; set; }  
  
        /// <summary>  
        /// Оценка  
        /// </summary>  
        internal int Rating { get; set; }  
  
        /// <summary>  
        /// Дата экзамена  
        /// </summary>  
        internal DateTime DateExam { get; set; }  
  
        /// <summary>  
        ///  
        /// </summary>  
        public DateTime Date {  
            get => throw new NotImplementedException();  
            set => throw new NotImplementedException();  
        }  
  
        /// <summary>  
        /// Конструктор по умолчанию  
        /// </summary>  
        internal Exam() {  
            NameSubject = "noData";  
            Rating = 0;  
            DateExam = DateTime.MinValue;  
        }  
  
        /// <summary>  
        ///  
        /// </summary>  
        /// <param name="nameSubject"></param>  
        /// <param name="rating"></param>  
        /// <param name="dateExam"></param>  
        internal Exam(string nameSubject, int rating, DateTime dateExam) {  
            NameSubject = nameSubject;  
            Rating = rating;  
            DateExam = dateExam;  
        }  
  
        /// <summary>  
        ///  
        /// </summary>  
        /// <returns></returns>  
        public override string ToString()  
        {  
            return NameSubject + " " + Rating + ", Date:" + DateExam.ToString();  
        }  
    }  
}
```

```

    /// <summary>
    ///
    /// </summary>
    /// <param name="obj"></param>
    /// <returns></returns>
    public override bool Equals(object obj)
    {
        Exam tmp = obj as Exam;
        return tmp != null &&
            this.NameSubject == tmp.NameSubject &&
            this.Rating == tmp.Rating &&
            this.DateExam.Equals(tmp.DateExam);
    }

    public static bool operator ==(Exam p1, Exam p2)
    {
        if (p1 is null || p2 is null)
            return false;

        return p1.Equals(p2);
    }
    public static bool operator !=(Exam p1, Exam p2)
    {
        return !(p1 == p2);
    }

    /// <summary>
    ///
    /// </summary>
    /// <returns></returns>
    public override int GetHashCode()
    {
        return base.GetHashCode();
    }

    public object DeepCopy()
    {
        return new Exam(NameSubject, Rating, DateExam);
    }
}
}
}

```



```

//-----||
// Файл Person.cs
//-----||

namespace LR2
{
    internal class Person : IDateAndCopy
    {
        /// <summary>
        /// Имя
        /// </summary>
        protected string _Forename;
        /// <summary>
        /// Фамилия
        /// </summary>
        protected string _Surname;
        /// <summary>
        /// День рождения
        /// </summary>
        protected DateTime _BirthData;

        internal string Forename { get { return _Forename; } }
        internal string Surname { get { return _Surname; } }
        internal DateTime BirthData {
            get { return _BirthData; }
            set { _BirthData = value; }
        }

        public DateTime Date {
            get => throw new NotImplementedException();
            set => throw new NotImplementedException();
        }

        /// <summary>
        /// Конструктор по умолчанию
        /// </summary>
        internal Person()
        {
            _Forename = "";
            _Surname = "";
            _BirthData = DateTime.MinValue;
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="forename"></param>
        /// <param name="surname"></param>
        /// <param name="birthData"></param>
        internal Person(string forename, string surname, DateTime birthData)
        {
            _Forename = forename;
            _Surname = surname;
            _BirthData = birthData;
        }

        /// <summary>
        ///
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            return _Forename + " " + _Surname + " " + _BirthData.ToString();
        }
    }
}

```

```

    /// <summary>
    ///
    /// </summary>
    /// <returns></returns>
    public virtual string ToShortString()
    {
        return _Forename + " " + _Surname + " ";
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="obj"></param>
    /// <returns></returns>
    public override bool Equals(object obj)
    {
        Person tmp = obj as Person;
        return tmp != null &&
            this._Forename == tmp._Forename &&
            this._Surname == tmp._Surname &&
            this._BirthData == tmp._BirthData;
    }

    public static bool operator ==(Person p1, Person p2)
    {
        if(p1 is null || p2 is null)
            return false;

        return p1.Equals(p2);
    }
    public static bool operator !=(Person p1, Person p2)
    {
        return !(p1 == p2);
    }

    /// <summary>
    ///
    /// </summary>
    /// <returns></returns>
    public override int GetHashCode()
    {
        return base.GetHashCode();
    }

    public object DeepCopy()
    {
        return new Person(Forename, Surname, BirthData);
    }
}
}

```