Отчёт ЛР№3 Понкращенков Д.Б. ПИН-21Д.

Вариант №1.

Оглавление

1. Задание

Определить новые версии классов **Exam** и **Student** из лабораторной работы 2.

В класс **Exam** добавить реализацию интерфейсов

- System.IComparable для сравнения объектов типа Exam по названию предмета;
- System.Collections.Generic.IComparer<Exam> для сравнения объектов типа Exam по оценке.

Определить **вспомогательный класс**, реализующий интерфейс System.Collections.Generic.IComparer<Exam>, который можно использовать для сравнения объектов типа Exam по дате экзамена.

В новой версии класса **Student** для списков зачетов и экзаменов использовать типы

- System.Collections.Generic.List<Test> для списка зачетов;
- System.Collections.Generic.List<Exam> для списка экзаменов.

В новой версии класса **Student** сохранить все остальные поля, свойства и методы из предыдущей версии класса, внести необходимые исправления в код свойств и методов из-за изменения типов полей для списков.

В классе **Student** определить методы для сортировки списка экзаменов

- по названию предмета;
- по оценке;
- по дате экзамена.

Определить универсальный делегат

```
delegate TKey KeySelector<TKey>(Student st);
```

Определить универсальный класс **StudentCollection<TKey>**, содержащий коллекцию объектов Student, в котором для хранения коллекции используется тип System.Collections.Generic.Dictionary<TKey, Student>. Типовой параметр TKey универсального класса StudentCollection<TKey> определяет тип ключа в коллекции Dictionary<TKey, Student>.

Метод, который используется для вычисления ключа при добавлении элемента Student в коллекцию класса StudentCollection<TKey>, отвечает делегату KeySelector<TKey> и передается StudentCollection<TKey> через параметр единственного конструктора класса.

Класс StudentCollection<TKey> содержит

- закрытое поле типа System.Collections.Generic.Dictionary<TKey, Student>;
- закрытое поле типа KeySelector<TKey> для хранения экземпляра делегата с методом, вычисляющим ключ для объекта Student;
- конструктор с одним параметром типа KeySelector<TKey> ;
- метод void AddDefaults (), с помощью которого можно добавить некоторое число элементов типа Student для инициализации коллекции по умолчанию;
- метод void AddStudents (params Student[]) для добавления элементов в коллекцию Dictionary<TKey, Student>;
- перегруженную версию виртуального метода string ToString() для формирования строки, содержащей информацию обо всех элементах коллекции Dictionary<TKey, Student>, в том числе значения всех полей класса Student, включая список зачетов и экзаменов;
- метод string ToShortString(), который формирует строку с информацией обо всех элементах коллекции Dictionary<TKey, Student>, состоящую из значений всех полей, среднего балла, числа зачетов и экзаменов для каждого элемента Student, но без списка зачетов и экзаменов.

В классе **StudentCollection<TKey>** определить свойства и методы, выполняющие операции со словарем Dictionary<TKey,Student> с использованием методов расширения класса System.Linq.Enumerable и статические методы-селекторы, которые необходимы для выполнения соответствующих операций с коллекцией:

- свойство типа double (только с методом get), возвращающее максимальное значение среднего балла для элементов Dictionary<TKey,Student>; если в коллекции нет элементов, свойство возвращает некоторое значение по умолчанию; для поиска максимального значения среднего балла надо использовать метод Max класса System.Linq.Enumerable;
- метод IEnumerable<KeyValuePair<TKey,Student>>EducationForm(Education value), возвращающий подмножество элементов коллекции Dictionary<TKey,Student> с заданной формой обучения; для формирования подмножества использовать метод Where класса System.Linq.Enumerable;
- свойство типа IEnumerable<IGrouping<Education,KeyValuePair<TKey,Student>>> (только с методом get), выполняющее группировку элементов коллекции Dictionary<TKey, Student> в зависимости от формы обучения студента с помощью метода Group класса System.Linq.Enumerable.

В методе **Main()**

1. Создать объект Student и вызвать методы, выполняющие сортировку списка экзаменов List<Exam> по разным критериям, после каждой сортировки вывести данные объекта. Выполнить сортировку
   - по названию предмета;
   - по оценке;
   - по дате экзамена.
2. Создать объект типа StudentCollection<string>. Добавить в коллекцию несколько разных элементов типа Student и вывести объект StudentCollection<string>.
3. Вызвать методы класса StudentCollection<string>, выполняющие операции с коллекцией-словарем Dictionary<TKey, Student>, и после каждой операции вывести результат операции. Выполнить
   - вычисление максимального значения среднего балла для элементов коллекции; вывести максимальное значение;
   - вызвать метод EducationForm для выбора студентов с заданной формой обучения, вывести результат фильтрации;
   - вызвать свойство класса, выполняющее группировку элементов коллекции по форме обучения; вывести все группы элементов.
4. Создать объект типа TestCollection<Person, Student>. Ввести число элементов в коллекциях и вызвать метод для поиска первого, центрального, последнего и элемента, не входящего в коллекции. Вывести значения времени поиска для всех четырех случаев.

2. Результат работы программы

Выполнение задания по вариантам:

```
1. ----------------------------------------------------------

 Sort by Name:   01.01.0001 0:00:00 Specialist 0, AvgRat: 3
 Exams:
 - subject0 1, Date:14.05.2024 14:03:22
 - subject1 5, Date:01.01.0001 0:00:00
 - subject2 2, Date:14.05.2024 14:03:22
 - subject3 3, Date:14.05.2024 14:03:22
 - subject4 4, Date:31.12.9999 23:59:59

 Tests:

 Sort by Rating:   01.01.0001 0:00:00 Specialist 0, AvgRat: 3
 Exams:
 - subject0 1, Date:14.05.2024 14:03:22
 - subject2 2, Date:14.05.2024 14:03:22
 - subject3 3, Date:14.05.2024 14:03:22
 - subject4 4, Date:31.12.9999 23:59:59
 - subject1 5, Date:01.01.0001 0:00:00

 Tests:

 Sort by Date:   01.01.0001 0:00:00 Specialist 0, AvgRat: 3
 Exams:
 - subject4 4, Date:31.12.9999 23:59:59
 - subject3 3, Date:14.05.2024 14:03:22
 - subject2 2, Date:14.05.2024 14:03:22
 - subject0 1, Date:14.05.2024 14:03:22
 - subject1 5, Date:01.01.0001 0:00:00

 Tests:


   ----------------------------------------------------------
```

```
2. ------------------------------------------------------------

||--------------------------------------------------------------||
 0 -   01.01.0001 0:00:00 Specialist 0, AvgRat: 0
 Exams:

 Tests:

||--------------------------------------------------------------||
||--------------------------------------------------------------||
 1 -   01.01.0001 0:00:00 SecondEducation 0, AvgRat: 1
 Exams:
 - BBE20X 0, Date:01.01.1601 3:00:07
 - 2SV5LX 2, Date:01.01.1601 3:02:40
 - TO9DOB 0, Date:01.01.1601 3:02:10
 - 5P4V8D 3, Date:01.01.1601 3:00:52
 - N6DGS2 0, Date:01.01.1601 3:01:56

 Tests:

||--------------------------------------------------------------||
||--------------------------------------------------------------||
 2 -   01.01.0001 0:00:00 Specialist 0, AvgRat: 1,4
 Exams:
 - U9TDEV 1, Date:01.01.1601 3:03:08
 - 3HWAJW 2, Date:01.01.1601 3:01:56
 - ZJ4B67 1, Date:01.01.1601 3:02:09
 - U7PK66 2, Date:01.01.1601 3:00:42
 - Y1H0O0 1, Date:01.01.1601 3:02:48

 Tests:

||--------------------------------------------------------------||
||--------------------------------------------------------------||
 3 -   01.01.0001 0:00:00 SecondEducation 0, AvgRat: 2,4
 Exams:
 - BWPGM3 1, Date:01.01.1601 3:00:44
 - R8SGE3 2, Date:01.01.1601 3:01:14
 - 5WTMAT 4, Date:01.01.1601 3:01:09
 - LA1J39 0, Date:01.01.1601 3:02:03
 - YFVCCA 5, Date:01.01.1601 3:00:01

 Tests:

||--------------------------------------------------------------||
||--------------------------------------------------------------||
 4 -   01.01.0001 0:00:00 Specialist 0, AvgRat: 2,2
 Exams:
 - 655ADM 3, Date:01.01.1601 3:02:44
 - N1CCJR 1, Date:01.01.1601 3:01:34
 - B2GVM1 2, Date:01.01.1601 3:01:47
 - 5I0MFE 3, Date:01.01.1601 3:01:36
 - 79CDJT 2, Date:01.01.1601 3:02:46

 Tests:

||--------------------------------------------------------------||
```

```
||-------------------------------------------------------------||
 5 -   01.01.0001 0:00:00 Bachelor 0, AvgRat: 4,2
 Exams:
 - VTJTZL 4, Date:01.01.1601 3:02:46
 - O51GF9 5, Date:01.01.1601 3:01:03
 - GBQ0DE 4, Date:01.01.1601 3:01:43
 - F3LHSB 4, Date:01.01.1601 3:03:03
 - 3JY3SP 4, Date:01.01.1601 3:03:32

 Tests:

||-------------------------------------------------------------||


   ------------------------------------------------------


3. ------------------------------------------------------

 Max AVG Rating: 4,2

   ------------------------------------------------------

 Students - Bachelor:
 -  5 -   01.01.0001 0:00:00 Bachelor 0, AvgRat: 4,2

   ------------------------------------------------------

 Grouped students:
 Specialist:
 -  0 -   01.01.0001 0:00:00 Specialist 0, AvgRat: 0
 -  2 -   01.01.0001 0:00:00 Specialist 0, AvgRat: 1,4
 -  4 -   01.01.0001 0:00:00 Specialist 0, AvgRat: 2,2
 SecondEducation:
 -  1 -   01.01.0001 0:00:00 SecondEducation 0, AvgRat: 1
 -  3 -   01.01.0001 0:00:00 SecondEducation 0, AvgRat: 2,4
 Bachelor:
 -  5 -   01.01.0001 0:00:00 Bachelor 0, AvgRat: 4,2

   ------------------------------------------------------
```

```
 Введите число элементов:
5
------------------------ TestListKey ------------------------
 Ticks contains First:3371 True
 Ticks contains Middle:24 True
 Ticks contains Last:18 True
 Ticks contains Outsider:18 False
------------------------ TestListString ------------------------
 Ticks contains First:134 True
 Ticks contains Middle:58 True
 Ticks contains Last:62 True
 Ticks contains Outsider:58 False
------------------------ TestDictionary ------------------------
 Ticks contains First:324 False
 Ticks contains Middle:44 False
 Ticks contains Last:41 False
 Ticks contains Outsider:42 False
------------------------ TestDictionaryString ------------------------
 Ticks contains First:283 True
 Ticks contains Middle:88 True
 Ticks contains Last:64 True
 Ticks contains Outsider:101 False
```

3. Листинг

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

using LR2;
using System.Security.Cryptography;

namespace LR3
{
    internal class Program
    {
        private static Random random = new Random();

        static void Main(string[] args)
        {
            T1();

            T2_3();

            T4();

            Console.ReadKey();
        }

        static void T1()
        {
            var student = new Student();

            student.AddExams(new List<Exam>{
                    new Exam("subject0", 1, DateTime.Now),
                    new Exam("subject2", 2, DateTime.Now),
                    new Exam("subject1", 5, DateTime.MinValue),
                    new Exam("subject3", 3, DateTime.Now),
                    new Exam("subject4", 4, DateTime.MaxValue)
            });

            Console.WriteLine("\n1. ------------------------------------------------------\n");
            student.SortExamByName();
            Console.WriteLine(" Sort by Name: " + student.ToString());
            student.SortExamByRating();
            Console.WriteLine(" Sort by Rating: " + student.ToString());
            student.SortExamByDate();
            Console.WriteLine(" Sort by Date: " + student.ToString());
            Console.WriteLine("\n   ------------------------------------------------------\n");
        }

        static void T2_3()
        {
            int countElem = 0;
            var StudentCollection = new StudentCollection<string>((Student st) => { return
st.Forename + countElem++; });
            StudentCollection.AddDefaults();
            StudentCollection.AddStudents([
                getStudentRND(),
                getStudentRND(),
                getStudentRND(),
                getStudentRND(),
                getStudentRND()
            ]);

            Console.WriteLine("\n2. ------------------------------------------------------\n");
            Console.WriteLine(StudentCollection.ToString());
            Console.WriteLine("\n   ------------------------------------------------------\n");

            Console.WriteLine("\n3. ------------------------------------------------------\n");
            Console.WriteLine( " Max AVG Rating: " + StudentCollection.MaxAvgRating);
            Console.WriteLine("\n   ------------------------------------------------------\n");
```

```
    var coolect = StudentCollection.EducationForm(Education.Bachelor);

    Console.WriteLine(" Students - Bachelor:");

    foreach (var kvp in coolect)
    {
        Console.WriteLine(" - " + $" {kvp.Key} - {kvp.Value.ToShortString()}");
    }

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

    Console.WriteLine(" Grouped students:");

    foreach (var group in StudentCollection.group)
    {
        Console.WriteLine(" " + group.Key.ToString() + ":");
        foreach (var kvp in group)
        {
            Console.WriteLine(" - " + $" {kvp.Key} - {kvp.Value.ToShortString()}");
        }
    }

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

static void T4()
{
    Console.WriteLine(" Введите число элементов:");

    int count = 0;
    while (count < 1)
    {
        try
        {
            count = int.Parse(Console.ReadLine());
        }
        catch (Exception)
        {
            Console.WriteLine(" Ошибка! Введите положительное число элементов:");
        }
    }

    GenerateElement<Person,Student> initDelegate = (int num) =>
    {
        var key = new Person(num.ToString(), "", DateTime.MinValue);
        var value = new Student(key, Education.Bachelor,0);

        return KeyValuePair.Create(key, value);
    };

    var TestCollection = new TestCollection<Person, Student>(initDelegate, count);

    TestCollection.TestListKey();
    TestCollection.TestListString();
    TestCollection.TestDictionary();
    TestCollection.TestDictionaryString();
}

#region --- Рандомайзеры ---

static Student getStudentRND()
{
    var student = new Student();
    student.AddExams(new List<Exam>{
            new Exam(getStringRND(), getRatRND(), DateTime.FromFileTime(getDateRND())),
            new Exam(getStringRND(), getRatRND(), DateTime.FromFileTime(getDateRND())),
            new Exam(getStringRND(), getRatRND(), DateTime.FromFileTime(getDateRND())),
            new Exam(getStringRND(), getRatRND(), DateTime.FromFileTime(getDateRND())),
            new Exam(getStringRND(), getRatRND(), DateTime.FromFileTime(getDateRND()))
    });
    student.Education = (Education)RandomNumberGenerator.GetInt32(0, 3);
```

```
            return student;
        }

        static int getRatRND()
        {
            return RandomNumberGenerator.GetInt32(0, 6);
        }

        static int getDateRND()
        {
            return RandomNumberGenerator.GetInt32(0, int.MaxValue);
        }

        static string getStringRND()
        {
            int length = 6;
            const string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
            return new string(Enumerable.Repeat(chars, length)
                .Select(s => s[random.Next(s.Length)]).ToArray());
        }

        #endregion
    }

}
```

```
//-----------------------------------------------------------------------------------------||
// Файл TestCollection.cs
//-----------------------------------------------------------------------------------------||

using System.Diagnostics;


namespace LR3
{
    delegate KeyValuePair<TKey, TValue> GenerateElement<TKey, TValue>(int j);

    internal class TestCollection<TKey, TValue>
    {
        int SizeCollections;

        List<TKey> LKeys;
        List<string> LStrings;
        Dictionary<TKey, TValue> Dictionary;
        Dictionary<string, TValue> DictionarySTR;

        GenerateElement<TKey, TValue> InitDelegate;

        internal TestCollection(GenerateElement<TKey, TValue> deleg, int countElem)
        {
            InitDelegate = deleg;
            SizeCollections = countElem;

            LKeys = new List<TKey>(SizeCollections);
            for (int i =0; i< SizeCollections;i++)
            {
                LKeys.Add(InitDelegate.Invoke(i).Key);
            }

            LStrings = new List<string>(SizeCollections);
            for (int i = 0; i < SizeCollections; i++)
            {
                LStrings.Add(InitDelegate.Invoke(i).Key.ToString());
            }

            Dictionary = new Dictionary<TKey, TValue>(SizeCollections);
            for (int i = 0; i < SizeCollections; i++)
            {
                var kvp = InitDelegate.Invoke(i);
                Dictionary.Add(kvp.Key, kvp.Value);
            }

            DictionarySTR = new Dictionary<string, TValue>(SizeCollections);
            for (int i = 0; i < SizeCollections; i++)
            {
                var kvp = InitDelegate.Invoke(i);
                DictionarySTR.Add(kvp.Key.ToString(), kvp.Value);
            }
        }

        internal void TestListKey()
        {
            bool isFind;
            var watcher = new Stopwatch();

            Console.WriteLine("---------------------- TestListKey ----------------------");

            var tmp = InitDelegate(0);
            watcher.Start();
            isFind = LKeys.Contains(tmp.Key);
            watcher.Stop();
            Console.WriteLine(" Ticks contains First:" + watcher.ElapsedTicks + " " + isFind);

            tmp = InitDelegate(SizeCollections/2);
            watcher.Restart();
            isFind = LKeys.Contains(tmp.Key);
            watcher.Stop();
            Console.WriteLine(" Ticks contains Middle:" + watcher.ElapsedTicks + " " + isFind);
```

```
            tmp = InitDelegate(SizeCollections-1);
            watcher.Restart();
            isFind = LKeys.Contains(tmp.Key);
            watcher.Stop();
            Console.WriteLine(" Ticks contains Last:" + watcher.ElapsedTicks + " " + isFind);

            tmp = InitDelegate(SizeCollections);
            watcher.Restart();
            isFind = LKeys.Contains(tmp.Key);
            watcher.Stop();
            Console.WriteLine(" Ticks contains Outsider:" + watcher.ElapsedTicks + " " +
isFind);
        }

        internal void TestListString()
        {

            bool isFind;
            var watcher = new Stopwatch();

            Console.WriteLine("---------------------- TestListString ----------------------");

            var tmp = InitDelegate(0);
            watcher.Start();
            isFind = LStrings.Contains(tmp.Key.ToString());
            watcher.Stop();
            Console.WriteLine(" Ticks contains First:" + watcher.ElapsedTicks + " " + isFind);

            tmp = InitDelegate(SizeCollections / 2);
            watcher.Restart();
            isFind = LStrings.Contains(tmp.Key.ToString());
            watcher.Stop();
            Console.WriteLine(" Ticks contains Middle:" + watcher.ElapsedTicks + " " + isFind);

            tmp = InitDelegate(SizeCollections - 1);
            watcher.Restart();
            isFind = LStrings.Contains(tmp.Key.ToString());
            watcher.Stop();
            Console.WriteLine(" Ticks contains Last:" + watcher.ElapsedTicks + " " + isFind);

            tmp = InitDelegate(SizeCollections);
            watcher.Restart();
            isFind = LStrings.Contains(tmp.Key.ToString());
            watcher.Stop();
            Console.WriteLine(" Ticks contains Outsider:" + watcher.ElapsedTicks + " " +
isFind);
        }

        internal void TestDictionary()
        {
            bool isFind;
            var watcher = new Stopwatch();

            Console.WriteLine("---------------------- TestDictionary ----------------------");

            var tmp = InitDelegate(0);
            watcher.Start();
            isFind = Dictionary.ContainsKey(tmp.Key);
            watcher.Stop();
            Console.WriteLine(" Ticks contains First:" + watcher.ElapsedTicks + " " + isFind);

            tmp = InitDelegate(SizeCollections / 2);
            watcher.Restart();
            isFind = Dictionary.ContainsKey(tmp.Key);
            watcher.Stop();
            Console.WriteLine(" Ticks contains Middle:" + watcher.ElapsedTicks + " " + isFind);

            tmp = InitDelegate(SizeCollections - 1);
            watcher.Restart();
            isFind = Dictionary.ContainsKey(tmp.Key);
            watcher.Stop();
```

```
            Console.WriteLine(" Ticks contains Last:" + watcher.ElapsedTicks + " " + isFind);

            tmp = InitDelegate(SizeCollections);
            watcher.Restart();
            isFind = Dictionary.ContainsKey(tmp.Key);
            watcher.Stop();
            Console.WriteLine(" Ticks contains Outsider:" + watcher.ElapsedTicks + " " +
isFind);
        }

        internal void TestDictionaryString()
        {
            bool isFind;
            var watcher = new Stopwatch();
           Console.WriteLine("---------------------- TestDictionaryString -------------------");

            var tmp = InitDelegate(0);
            watcher.Start();
            isFind = DictionarySTR.ContainsKey(tmp.Key.ToString());
            watcher.Stop();
            Console.WriteLine(" Ticks contains First:" + watcher.ElapsedTicks + " " + isFind);

            tmp = InitDelegate(SizeCollections / 2);
            watcher.Restart();
            isFind = DictionarySTR.ContainsKey(tmp.Key.ToString());
            watcher.Stop();
            Console.WriteLine(" Ticks contains Middle:" + watcher.ElapsedTicks + " " + isFind);

            tmp = InitDelegate(SizeCollections - 1);
            watcher.Restart();
            isFind = DictionarySTR.ContainsKey(tmp.Key.ToString());
            watcher.Stop();
            Console.WriteLine(" Ticks contains Last:" + watcher.ElapsedTicks + " " + isFind);

            tmp = InitDelegate(SizeCollections);
            watcher.Restart();
            isFind = DictionarySTR.ContainsKey(tmp.Key.ToString());
            watcher.Stop();
            Console.WriteLine(" Ticks contains Outsider:" + watcher.ElapsedTicks + " " +
isFind);
        }
    }
}
```

```
//-----------------------------------------------------------------------------------------||
// Файл StudentCollection.cs
//-----------------------------------------------------------------------------------------||

using LR2;
using System.Text;

namespace LR3
{
    /// <summary>
    ///
    /// </summary>
    /// <typeparam name="TKey"></typeparam>
    /// <param name="st"></param>
    /// <returns></returns>
    delegate TKey KeySelector<TKey>(Student st);

    internal class StudentCollection<TKey>
    {
        internal double MaxAvgRating {
            get {
                if (Dictionary.Count == 0)
                    return 0;
                else
                    return Dictionary.Max(kvp=>kvp.Value.AvgRating);
            }
        }

        /// <summary>
        /// Выполняет группировку элементов коллекции
        /// Dictionary<TKey, Student> в зависимости от формы обучения студента с
        /// помощью метода Group класса System.Linq.Enumerable.
        /// </summary>
        internal IEnumerable<IGrouping<Education, KeyValuePair<TKey, Student>>> group
        {
            get
            {
                return Dictionary.GroupBy((kvp) => { return kvp.Value.Education; });
            }
        }

        /// <summary>
        /// Кеш инициализированных студентов
        /// </summary>
        Dictionary<TKey, Student> Dictionary;

        /// <summary>
        /// Делегат формирования ключей словаря
        /// </summary>
        KeySelector<TKey> keySelector;

        internal StudentCollection(KeySelector<TKey> deleg)
        {
            keySelector = deleg;

            Dictionary = new Dictionary<TKey, Student>();
        }

        /// <summary>
        /// Возвращает подмножество элементов коллекции
        /// Dictionary<TKey, Student> с заданной формой обучения;
        /// для формирования подмножества использовать метод Where класса
System.Linq.Enumerable;
        /// </summary>
        /// <param name="value"></param>
        /// <returns></returns>
        internal IEnumerable<KeyValuePair<TKey, Student>> EducationForm(Education value)
        {
            return Dictionary.Where((kvp)=>kvp.Value.Education == value);
        }
```

```
        /// <summary>
        /// Для добавления некоторого числа элементов типа Student для инициализации коллекции
        /// по умолчанию;
        /// </summary>
        internal void AddDefaults()
        {
            for (int i = 0; i < 1; i++) {
                var st = new Student();
                var key = keySelector.Invoke(st);
                Dictionary.TryAdd(key, st);
            }
        }

        /// <summary>
        /// Для добавления элементов в коллекцию Dictionary<TKey, Student>
        /// </summary>
        /// <param name="students"></param>
        internal void AddStudents(params Student[] students)
        {
            foreach (var student in students) {
                var key = keySelector.Invoke(student);
                Dictionary.TryAdd(key, student);
            }
        }

        /// <summary>
        /// Формирует строку, содержащую информацию обо всех элементах
        /// коллекции Dictionary<TKey, Student>, в том числе значения всех полей
        /// класса Student, включая список зачетов и экзаменов;
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            StringBuilder stringBuilder = new StringBuilder();

            foreach (var kvp in Dictionary)
            {
                stringBuilder.AppendLine("||-------------------------------------------------
------------||");
                stringBuilder.AppendLine($" {kvp.Key} - {kvp.Value.ToString()}");
                stringBuilder.AppendLine("||-------------------------------------------------
------------||");
            }
            return stringBuilder.ToString();
        }

        /// <summary>
        /// Формирует строку с информацией обо всех элементах коллекции Dictionary<TKey,
Student>, состоящую из
        /// значений всех полей, среднего балла, числа зачетов и экзаменов для
        /// каждого элемента Student, но без списка зачетов и экзаменов.
        /// </summary>
        /// <returns></returns>
        public virtual string ToShortString()
        {
            StringBuilder stringBuilder = new StringBuilder();

            foreach (var kvp in Dictionary)
            {
                stringBuilder.AppendLine("||-------------------------------------------------
------------||");
                stringBuilder.AppendLine($" {kvp.Key} - {kvp.Value.ToShortString()} " +
                    $"Exams: {kvp.Value.ClosedExams.Count()} Tests:
{kvp.Value.Subjects.Count()}");
                stringBuilder.AppendLine("||-------------------------------------------------
------------||");
            }
            return stringBuilder.ToString();
        }
    }
}
```

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

using LR2;

namespace LR3
{
    internal class Exam : IDateAndCopy, IComparable, IComparer<Exam>
    {
        #region --- Параметры ---

        /// <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 { return DateTime.MinValue; }//throw new NotImplementedException();
            set { }//throw new NotImplementedException();
        }

        #endregion

        #region --- Компараторы ---

        /// <summary>
        /// Сравнение объектов типа Exam по оценке.
        /// </summary>
        public int Compare (Exam? x, Exam? y)
        {
            if (x.Rating<y.Rating) return -1;
            else if (x.Rating > y.Rating) return 1;
            else return 0;
        }

        /// <summary>
        /// Сравнение объектов типа Exam по названию предмета
        /// </summary>
        /// <param name="obj"></param>
        /// <returns></returns>
        /// <exception cref="TypeAccessException"></exception>
        public int CompareTo(object? obj)
        {
            var y = obj as Exam;

            if (y == null)
                throw new TypeAccessException();

            return this.NameSubject.CompareTo(y.NameSubject);
        }

        #endregion
```

```
        #region --- Конструкторы ---
        /// <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;
        }
        #endregion

        #region --- Virtuals ---
        /// <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();
        }
        #endregion

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

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

using System.Collections;
using System.Text;
using LR2;

namespace LR3
{
    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>
        List<Exam> closedExams;

        /// <summary>
        ///
        /// </summary>
        List<Test> subjects;

        #endregion

        #region --- Свойства ---

        /// <summary>
        /// Средний балл
        /// </summary>
        internal 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 List<Exam> ClosedExams
        {
            get { return closedExams; }
            set { closedExams = value; }
        }
```

```
internal List<Test> Subjects
{
    get { return subjects; }
    set { subjects = value; }
}

#endregion

#region --- Итераторы ---

/// <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 iterExam = closedExams.GetEnumerator();
    while (iterExam.MoveNext())
    {
        if ((iterExam.Current as Exam).Rating > 2)
            yield return iterExam.Current;
    }

    var iterTest = Subjects.GetEnumerator();
    while(iterTest.MoveNext())
    {
        if ((iterTest.Current as Test).isPassed)
            yield return iterTest.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>
        /// По доп. требованиям
        /// </summary>
        /// <returns></returns>
        IEnumerator IEnumerable.GetEnumerator()
        {
            return null;
            //return new StudentEnumerator(this);
        }

        #endregion

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

        #region --- Конструкторы ---

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

            education = new ();
            numberGroup = 0;
            closedExams = new List<Exam>();
            Subjects = new List<Test>();
        }
```

```
/// <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;

    closedExams = new List<Exam>();
    Subjects = new List<Test>();
}

#endregion

#region --- Сортировщики ---

/// <summary>
///
/// </summary>
/// <returns></returns>
internal void SortExamByName()
{
    closedExams.Sort();
}

/// <summary>
///
/// </summary>
/// <returns></returns>
internal void SortExamByRating()
{
    closedExams.Sort(new Exam());
}

/// <summary>
///
/// </summary>
/// <returns></returns>
internal void SortExamByDate()
{
    closedExams.Sort(new CompareExam());
}

#endregion
```

```
        #region --- Virtuals ---

        public override string ToString()
        {
            StringBuilder exams = new StringBuilder();

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

            StringBuilder tests = new StringBuilder();

            if (subjects != null)
                foreach (var ex in subjects)
                {
                    tests.AppendLine(" - " + ex.ToString());
                }

            return ToShortString() + "\n Exams:\n" + exams.ToString() + "\n Tests:\n" +
tests.ToString();
        }

        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();
        }

        #endregion
```

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

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

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

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

            return tmp;
        }

    }
}
```