Лабораторная работа 3

Универсальные типы. Классы-коллекции. Методы расширения класса System.Linq.Enumerable

Требования к программе, общие для всех вариантов

Во всех вариантах лабораторной работы 3 требуется определить класс TestCollections, который содержит поля следующих типов

• System.Collections.Generic.List ;

• System.Collections.Generic.List ;

•System.Collections.Generic.Dictionary ;

• System.Collections.Generic.Dictionary .

Конкретные значения типовых параметров TKey и TValue зависят от варианта. Во всех вариантах тип ключа TKey и тип значения TValue связаны отношением базовый-производный. Во всех вариантах в классе TValue определено свойство, которое возвращает ссылку на объект типа TKey с данными, совпадающими с данными подобъекта базового класса (это свойство должно возвращать ссылку на объект типа TKey, а не ссылку на вызывающий объект TValue).

В конструкторе класса TestCollections создаются коллекции с заданным числом элементов. Надо сравнить время поиска элемента в коллекцияхсписках List и время поиска элемента по ключу и элемента по значению в коллекциях-словарях Dictionary.

Для автоматической генерации элементов коллекций в классе TestCollections надо определить статический метод, который принимает один целочисленный параметр типа int и возвращает ссылку на объект типа TValue.

Каждый объект TValue содержит подобъект базового класса TKey. Соответствие между значениями целочисленного параметра метода и подобъектами TKey класса TValue должно быть взаимно-однозначным – равным значениям параметра должны отвечать равные объекты TKey и наоборот. Равенство объектов типа TKey трактуется так же, как это было сделано в лабораторной работе 2 при определении операций равенства объектов.

Все четыре коллекции содержат одинаковое число элементов. Каждому элементу из коллекции List должен отвечать элемент в коллекции Dictionary с равным значением ключа. Список List состоит из строк, которые получены в результате вызова метода ToString() для объектов TKey из списка List. Каждому элементу списка List отвечает элемент в коллекции-словаре Dictionary с равным значением ключа типа string.

Число элементов в коллекциях вводится пользователем в процессе работы приложения. Если при вводе была допущена ошибка, приложение должно обработать исключение, сообщить об ошибке ввода и повторить прием ввода до тех пор, пока не будет правильно введено целочисленное значение.

Для четырех разных элементов – первого, центрального, последнего и элемента, не входящего в коллекцию – надо измерить время поиска

• элемента в коллекциях List и List с помощью метода Contains;

• элемента по ключу в коллекциях Dictionary< TKey, TValue> и Dictionary с помощью метода ContainsKey;

• значения элемента в коллекции Dictionary< TKey, TValue > с помощью метода ContainsValue.

Так как статический метод для автоматической генерации элементов должен обеспечивать взаимно-однозначное соответствие между значением целочисленного параметра метода и объектами TKey, этот метод можно использовать как при создании коллекций с большим числом элементов, так и для генерации элемента для поиска.

Вариант 1.

Требования к программе

Определить новые версии классов Person и Student из лабораторной работы 2.

В класс Person добавить реализацию интерфейсов

• System.IComparable для сравнения объектов типа Person по полю с фамилией;

• System.Collections.Generic.IComparer для сравнения объектов типа Person по дате рождения.

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

• System.Collections.Generic.List для списка зачетов;

• System.Collections.Generic.List 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 зачетов и экзаменов.

Определить вспомогательный класс, реализующий интерфейс System.Collections.Generic.IComparer, который можно использовать для сравнения объектов типа Student по среднему баллу.

Определить класс StudentCollection, который содержит

• закрытое поле типа System.Collections.Generic.List;

• метод void AddDefaults(), c помощью которого можно добавить некоторое число элементов типа Student для инициализации коллекции по умолчанию;

• метод void AddStudents (params Student[] ) для добавления элементов в список List;

• перегруженную версию виртуального метода string ToString() для формирования строки c информацией обо всех элементах списка List, включающую значения всех полей, список зачетов и экзаменов для каждого элемента Student;

• метод string ToShortString(), который формирует строку c информацией обо всех элементах списка List, содержащую значения всех полей, средний балл, число зачетов и число экзаменов для каждого элемента Student, но без списков зачетов и экзаменов.

В классе StudentCollection определить методы, выполняющие сортировку списка List

• по фамилии студента с использованием интерфейса IComparable, реализованного в классе Person;

• по дате рождения студента с использованием интерфейса IComparer, реализованного в классе Person;

• по среднему баллу с использованием интерфейса IComparer, реализованного во вспомогательном классе.

В классе StudentCollection определить свойства и методы, выполняющие операции со списком List с использованием методов расширения класса System.Linq.Enumerable, и статические методы-селекторы, которые необходимы для выполнения соответствующих операций со списком:

• свойство типа double (только с методом get), возвращающее максимальное значение среднего балла для элементов списка List; если в коллекции нет элементов, свойство возвращает некоторое значение по умолчанию; для поиска максимального значения среднего балла использовать метод Max класса System.Linq.Enumerable;

• свойство типа IEnumerable (только с методом get), возвращающее подмножество элементов списка List с формой обучения Education.Specialist; для формирования подмножества использовать метод Where класса System.Linq.Enumerable;

• метод List AverageMarkGroup(double value), который возвращает список, в который входят элементы Student из списка List с заданным значением среднего балла; для формирования списка использовать методы Group и ToList класса System.Linq.Enumerable.

Определить класс TestCollections, в котором в качестве типа TKey используется класс Person, а в качестве типа TValue - класс Student. Класс содержит закрытые поля с коллекциями типов

• System.Collections.Generic.List<Person>;

• System.Collections.Generic.List<string>;

• System.Collections.Generic.Dictionary <Person, Student>;

• System.Collections.Generic.Dictionary <string, Student>.

В классе TestCollections определить

• статический метод с одним целочисленным параметром типа int, который возвращает ссылку на объект типа Student и используется для автоматической генерации элементов коллекций;

• конструктор c параметром типа int (число элементов в коллекциях) для автоматического создания коллекций с заданным числом элементов;

• метод, который вычисляет время поиска элемента в списках List<Person> и List<string>, время поиска элемента по ключу и время поиска элемента по значению в коллекциях-словарях Dictionary<Person, Student> и Dictionary<string, Student>.

В методе Main()

1. Создать объект типа StudentCollection. Добавить в коллекцию несколько различных элементов типа Student и вывести объект StudentCollection.

2. Для созданного объекта StudentCollection вызвать методы, выполняющие сортировку списка List по разным критериям, и после каждой сортировки вывести данные объекта. Выполнить сортировку

• по фамилии студента;

• по дате рождения;

• по среднему баллу.

3. Вызвать методы класса StudentCollection, выполняющие операции со списком List, и после каждой операции вывести результат операции. Выполнить

• вычисление максимального значения среднего балла для элементов списка;

• фильтрацию списка для отбора студентов с формой обучения Education.Specialist;

• группировку элементов списка по значению среднего балла; вывести все группы элементов.

4. Создать объект типа TestCollections. Вызвать метод для поиска в коллекциях первого, центрального, последнего и элемента, не 30 входящего в коллекции. Вывести значения времени поиска для всех четырех случаев. Вывод должен содержать информацию о том, к какой коллекции и к какому элементу относится данное значение.

Код программы (описание в комментариях):

1. Ниже преведен метод Main откуда начинается программа

static void Main(string[] args)

{

//Создание коллекции студентов

var studentCollection = new StudentCollection();

//Добавление стуентов по умолчанию

studentCollection.AddDefaults(10);

//Вывод

Console.WriteLine(studentCollection.ToString() + "\n");

Console.ResetColor();

//Сортировка по фамилии

studentCollection.SortByLastName();

Console.WriteLine("Сортировка по фамилии:" + studentCollection.ToString());

//Сортировка по дате

studentCollection.SortByDate();

Console.WriteLine("Сортировка по дате: " + studentCollection.ToString());

//Сортировка по оценке

studentCollection.SortByGrage();

Console.WriteLine("Сортировка по оценке:" + studentCollection.ToString());

//Получение результатов

Console.WriteLine();

Console.WriteLine("Максимальный средний балл: " +

studentCollection.MaxGrade);

Console.WriteLine("Специалисты: \n" + String.Join("\n",

studentCollection.StudentsWithSpecialist.Select(c => c.ToString())));

Console.WriteLine("Группировка по сред баллу: \n" + String.Join("\n",

studentCollection.AverageMarkGroup(3).Select(c => c.ToString())));

//Тестирование коллекций элементов

var count = 1000000;

var test = new TestCollections(count);

Console.WriteLine("\n\n" + test.GetTimeForFindItem(count - 1));

Console.WriteLine("\n\n" + test.GetTimeForFindItem(0));

Console.WriteLine("\n\n" + test.GetTimeForFindItem(count / 2));

Console.WriteLine("\n\n" + test.GetTimeForFindItem(count));

}

1. Определяем новую версию класса Person

/// <summary>

/// Класс человека

/// </summary>

public class Person : IDateAndCopy, IComparer<Person>, IComparable

{

/// <summary>

/// Имя

/// </summary>

protected string \_name;

/// <summary>

/// Фамилия

/// </summary>

protected string \_lastName;

/// <summary>

/// Имя

/// </summary>

public string Name

{

get => \_name;

set => \_name = value;

}

/// <summary>

/// Фамилия

/// </summary>

public string LastName

{

get => \_lastName;

set => \_lastName = value;

}

/// <summary>

/// Дата рождения

/// </summary>

public DateTime DateOfBirth

{

get => Date;

set => Date = value;

}

/// <summary>

/// Год рождения

/// </summary>

public int CustomizeDateOfBirthYear

{

get

{

return Date.Year;

}

set

{

Date = Date.AddYears(-Date.Year).AddYears(value);

}

}

public DateTime Date { get; set; }

/// <summary>

/// Конструктор заполняющий дефолтными значениями

/// </summary>

public Person()

{

\_name = ProgramConsts.DefaultName;

\_lastName = StringExtension.GetRandom(10);

Date = DateTime.Now.AddYears(-(new Random().Next(1, 20)));

}

/// <summary>

/// Конструктор с параметрами

/// </summary>

public Person(string name, string lastName, DateTime datetime)

{

\_name = name;

\_lastName = lastName;

Date = datetime;

}

/// <summary>

/// Получение строки со всеми полями класса

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Имя: {\_name}, Фамилия: {\_lastName}, Дата рождения {Date}";

}

/// <summary>

/// Получение строки с именем и фамилией

/// </summary>

/// <returns></returns>

public virtual string ToShortString()

{

return $"Имя: {\_name}, Фамилия: {\_lastName}";

}

/// <summary>

/// Сравнение оьъектов

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

public override bool Equals(object obj)

{

if (Object.ReferenceEquals(this, obj))

return true;

if (this.GetType() != obj.GetType())

return false;

var person = (Person?)obj;

return this.Name == person.Name && LastName == person.LastName &&

DateOfBirth == person.DateOfBirth;

}

/// <summary>

/// Проверка на равенство

/// </summary>

/// <param name="person1"></param>

/// <param name="person2"></param>

/// <returns></returns>

public static bool operator == (Person person1, Person person2)

{

if(person1 is null)

{

if(person2 is null)

{

return true;

}

return false;

}

return person1.Equals(person2);

}

/// <summary>

/// Проверка на неравенство

/// </summary>

/// <param name="person1"></param>

/// <param name="person2"></param>

/// <returns></returns>

public static bool operator != (Person person1, Person person2)

{

return !(person1 == person2);

}

/// <summary>

/// Получение хэш кода

/// </summary>

/// <returns></returns>

public override int GetHashCode()

{

return (Name, LastName, DateOfBirth).GetHashCode();

}

/// <summary>

/// Создание полной копии

/// </summary>

/// <returns></returns>

public virtual object DeepCopy()

{

return new Person(this.Name, this.LastName, this.DateOfBirth);

}

/// <summary>

/// Сравнение даты

/// </summary>

/// <param name="x"></param>

/// <param name="y"></param>

/// <returns></returns>

/// <exception cref="Exception"></exception>

public int Compare(Person? x, Person? y)

{

if(x is null || y is null)

throw new Exception("Объект равен null");

if (x.DateOfBirth > y.DateOfBirth)

{

return 1;

}

else if (x.DateOfBirth < y.DateOfBirth)

return -1;

else

return 0;

}

/// <summary>

/// Сравнение объекта по фамилии

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

/// <exception cref="Exception"></exception>

public int CompareTo(object? obj)

{

if (obj is Person person)

return LastName.CompareTo(person.LastName);

throw new Exception("Объект не является типом " + nameof(Person));

}

}

1. Определяем новую версию класса Student

/// <summary>

/// Класс студента

/// </summary>

public class Student : Person, IDateAndCopy

{

/// <summary>

/// Образование

/// </summary>

private EducationEnum \_education;

/// <summary>

/// Номер группы

/// </summary>

private int \_groupNumber;

/// <summary>

/// Экзамены

/// </summary>

private List<Exam> \_exams;

/// <summary>

/// Зачеты

/// </summary>

private List<Test> \_tests;

/// <summary>

/// Образование

/// </summary>

public EducationEnum Education

{

get => \_education;

set => \_education = value;

}

/// <summary>

/// Номер группы

/// </summary>

public int GroupNumber

{

get

{

return \_groupNumber;

}

set

{

if (value <= 100 || value > 599)

throw new Exception("Значение не соответствует условию > 100 и

<= 599");

\_groupNumber = value;

}

}

/// <summary>

/// Экзамены

/// </summary>

public List<Exam> Exams

{

get => \_exams;

set => \_exams = value;

}

/// <summary>

/// Тесты

/// </summary>

public List<Test> Tests

{

get => \_tests;

set => \_tests = value;

}

/// <summary>

/// Средняя оценка

/// </summary>

public double AvgGrade { get => \_exams == null ? 0 :

\_exams.ToArray().Average(c => ((Exam)c).Grade); }

/// <summary>

/// Дата

/// </summary>

public DateTime Date { get; set; }

/// <summary>

/// Индексатор

/// </summary>

/// <param name="index"></param>

/// <returns></returns>

public bool this[EducationEnum education]

{

get => \_education == education ? true : false;

}

public Person Person

{

get

{

return (Person)base.MemberwiseClone();

}

set

{

this.Name = value.Name;

this.LastName = value.LastName;

this.DateOfBirth = value.DateOfBirth;

}

}

/// <summary>

/// Конструктор с параметрами

/// </summary>

/// <param name="personInfo"></param>

/// <param name="education"></param>

/// <param name="groupNumber"></param>

public Student(Person person, EducationEnum education, int groupNumber) :

base(person.Name, person.LastName, person.DateOfBirth)

{

\_education = education;

\_groupNumber = groupNumber;

}

/// <summary>

/// Конструктор без параметров

/// </summary>

public Student() : base()

{

\_groupNumber = new Random().Next(100, 401);

\_education = (EducationEnum)new Random().Next(0,3);

}

/// <summary>

/// Добавление экзаменов

/// </summary>

/// <param name="exams"></param>

public void AddExams(List<Exam> exams)

{

if (\_exams == null)

\_exams = exams;

else

\_exams.AddRange(exams);

}

/// <summary>

/// Добавление тестов

/// </summary>

/// <param name="tests"></param>

public void AddTests(List<Test> tests)

{

if (\_tests == null)

\_tests = tests;

else

\_tests.AddRange(tests);

}

/// <summary>

/// Получение строки со всеми полями класса

/// </summary>

/// <returns></returns>

public override string ToString()

{

var str = $"\n{base.ToString()}\nОбразование: {\_education}, Группа:

{\_groupNumber} Средний балл: {AvgGrade} \n";

if (\_exams != null && \_exams.Count > 0)

str += string.Join(",\n", \_exams.ToArray().Select(c =>

c.ToString()).ToList());

if(\_tests != null && \_tests.Count > 0)

str += "\n" + string.Join(",\n", \_tests.ToArray().Select(c =>

c.ToString()).ToList());

return str;

}

/// <summary>

/// Получение строки с именем и фамилией

/// </summary>

/// <returns></returns>

public virtual string ToShortString()

{

return $"{base.ToString()}\nОбразование: {\_education}, Группа:

{\_groupNumber}, Средний балл: {AvgGrade}, Кол-во экзаменов:

{\_exams.Count}, Кол-во зачетов: {\_tests.Count}\n";

}

/// <summary>

/// Получение полной копии

/// </summary>

/// <returns></returns>

public override object DeepCopy()

{

var newStudent = new Student((Person)base.DeepCopy(), Education,

GroupNumber);

var newExams = new List<Exam>();

var newTests = new List<Test>();

foreach(var item in Exams)

{

newExams.Add((Exam)item.DeepCopy());

}

foreach(var item in Tests)

{

newTests.Add((Test)item.DeepCopy());

}

newStudent.AddExams(newExams);

newStudent.AddTests(newTests);

return newStudent;

}

/// <summary>

/// Итератор

/// </summary>

/// <returns></returns>

public IEnumerable GetEnumerator(double? minGrade)

{

if (minGrade == null)

{

foreach (var item in Exams)

{

yield return item;

}

foreach(var item in Tests)

{

yield return item;

}

}

else

foreach (var item in Exams)

{

if (((Exam)item).Grade > minGrade)

yield return (Exam)item;

}

}

}

1. Определяем новый класс StudentCollection для создания коллекции студентов.

/// <summary>

/// Коллекция студентов

/// </summary>

public class StudentCollection : IEnumerable<Student>

{

/// <summary>

/// Студенты

/// </summary>

private List<Student> \_students = new List<Student>();

/// <summary>

/// Максимальная оценка

/// </summary>

public double MaxGrade

{

get

{

if (\_students == null || \_students.Count == 0)

return 0;

return \_students.Max(c=> c.AvgGrade);

}

}

/// <summary>

/// Получение специалистов

/// </summary>

/// <returns></returns>

public IEnumerable<Student> StudentsWithSpecialist

{

get

{

foreach (var student in \_students.Where(c => c.Education ==

Practice.Models.Enums.EducationEnum.Specialist))

{

yield return student;

}

}

}

/// <summary>

/// Получение студентов со средним баллом

/// </summary>

/// <param name="value"></param>

/// <returns></returns>

public List<Student> AverageMarkGroup(double value)

{

return \_students.GroupBy(c => c.AvgGrade).FirstOrDefault(c => c.Key ==

value)?.ToList() ?? new List<Student>();

}

/// <summary>

/// Добавление студентов

/// </summary>

public void AddStudents(params Student[] students)

{

\_students.AddRange(students);

}

/// <summary>

/// Добавление дефолтных значений

/// </summary>

public void AddDefaults(int count)

{

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

{

var stud = new Student();

stud.Exams = new List<Exam>();

for(int j = 0; j < 5; j++)

{

stud.Exams.Add(new Exam());

}

\_students.Add(stud);

}

}

/// <summary>

/// Получение строки объекта

/// </summary>

/// <returns></returns>

public override string ToString()

{

return String.Join("", \_students.Select(c => c.ToString()));

}

/// <summary>

/// Возвращает короткую версию строки элемента

/// </summary>

/// <returns></returns>

public string ToShortString()

{

return String.Join("\n\n", \_students.Select(c => c.ToShortString()));

}

/// <summary>

/// Сортировка по фамилии

/// </summary>

public void SortByLastName()

{

\_students.Sort();

}

/// <summary>

/// Сортировка по дате

/// </summary>

public void SortByDate()

{

\_students.Sort(new Person());

}

/// <summary>

/// Сортировка по оценкам

/// </summary>

public void SortByGrage()

{

\_students.Sort(new StudentComparer());

}

/// <summary>

/// Полученре перечисления

/// </summary>

/// <returns></returns>

public IEnumerator<Student> GetEnumerator()

{

return \_students.GetEnumerator();

}

/// <summary>

/// Получение перечисления

/// </summary>

/// <returns></returns>

IEnumerator IEnumerable.GetEnumerator()

{

return \_students.GetEnumerator();

}

}

1. Определяем класс StudentComparer реализующего интерфейс Icomparer для сортировки списка студентов

public class StudentComparer : IComparer<Student>

{

public int Compare(Student? x, Student? y)

{

if (x is null || y is null)

throw new Exception("Объект равен null");

//return (int)(x.AvgGrade - y.AvgGrade);

if (x.AvgGrade > y.AvgGrade)

{

return 1;

}

else if (x.AvgGrade < y.AvgGrade)

return -1;

else

return 0;

}

}

1. Определяем класс TestCollections, предназначенный для тестирования коллекций

/// <summary>

/// Тест коллекций

/// </summary>

public class TestCollections

{

/// <summary>

/// Список для теста

/// </summary>

public List<Person> TestTKeys;

/// <summary>

/// Список для теста

/// </summary>

public List<string> TestTString;

/// <summary>

/// Словарь для теста где ключом выступает класс пользователя

/// </summary>

public Dictionary<Person, Student> TestDictionaryTKey;

/// <summary>

/// Словарь для теста где ключом выступает строка класса

/// </summary>

public Dictionary<string, Student> TestDictionaryString;

/// <summary>

/// Тестовый студент

/// </summary>

private static Student \_setedStudent;

/// <summary>

/// Генерация случайных чисел

/// </summary>

private static Random \_random = new Random(DateTime.Now.Millisecond);

public TestCollections(int count)

{

TestTKeys = new List<Person>();

TestTString = new List<string>();

TestDictionaryTKey = new Dictionary<Person, Student>();

TestDictionaryString = new Dictionary<string, Student>();

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

{

TestTKeys.Add(new Person(i.ToString(), i.ToString(),

DateTime.UtcNow));

TestTString.Add(new Person(i.ToString(), i.ToString(),

DateTime.UtcNow).ToString());

var newPerson = new Person(i.ToString(), i.ToString(),

DateTime.UtcNow);

TestDictionaryString.Add(newPerson.ToString(), new

Student(newPerson, Practice.Models.Enums.EducationEnum.SecondEducation,

\_random.Next(1, 200)));

var newPerson2 = new Person(i.ToString(), i.ToString(),

DateTime.UtcNow);

TestDictionaryTKey.Add(newPerson2, new Student(newPerson2,

Practice.Models.Enums.EducationEnum.SecondEducation, \_random.Next(1,

200)));

}

}

public static Student SetStudentForTestCollection(int count)

{

\_setedStudent = new Student();

return \_setedStudent;

}

/ /// <summary>

/// Получение времени на поиск элемента в коллекциях

/// </summary>

/// <param name="index"></param>

/// <returns></returns>

public string GetTimeForFindItem(int index)

{

var response = "";

var item\_1 = TestTKeys.ElementAtOrDefault(index) ?? new Person();

var startTest1 = Environment.TickCount64;

var firstItem = TestTKeys.Contains(item\_1);

var endTest1 = Environment.TickCount64;

response += "\nЭлемент типа List<Person> был найден за " + (endTest1 –

startTest1) + "ms";

var item\_2 = TestTString.ElementAtOrDefault(index) ?? "NotContainsItem";

var startTest2 = Environment.TickCount64;

var firstItem2 = TestTString.Contains(item\_2);

var endTest2 = Environment.TickCount64;

response += "\nПервый элемент типа List<string> был найден за " +

(endTest2 - startTest2) + "ms";

var testItem3 = TestDictionaryTKey.Keys.ElementAtOrDefault(index) ?? new

Person();

var startTest3 = Environment.TickCount64;

var firstItem3 = TestDictionaryTKey.ContainsKey(testItem3);

var endTest3 = Environment.TickCount64;

response += "\nПервый элемент типа Dictionary<Person, Student> был

найден по ключу за " + (endTest3 - startTest3) + "ms";

var testItem4 = TestDictionaryString.Keys.ElementAtOrDefault(index) ??

"NotContainsItem";

var startTest4 = Environment.TickCount64;

var firstItem4 = TestDictionaryString.ContainsKey(testItem4);

var endTest4 = Environment.TickCount64;

response += "\nПервый элемент типа Dictionary<string, Student> был

найден по клюу за " + (endTest4 - startTest4) + "ms";

var testItem5 = TestDictionaryTKey.Values.ElementAtOrDefault(index) ??

new Student();

var startTest5 = Environment.TickCount64;

var firstItem5 = TestDictionaryTKey.ContainsValue(testItem5);

var endTest5 = Environment.TickCount64;

response += "\nПервый элемент типа Dictionary<Person, Student> был

найден по значению за " + (endTest5 - startTest5) + "ms";

var testItem6 = TestDictionaryString.Values.ElementAtOrDefault(index) ??

new Student();

var startTest6 = Environment.TickCount64;

var firstItem6 = TestDictionaryString.ContainsValue(testItem6);

var endTest6 = Environment.TickCount64;

response += "\nПервый элемент типа Dictionary<string, Student> был

найден по значению за " + (endTest6 - startTest6) + "ms";

return response;

}

}

1. Определили класс Exam для описания экзаменов студентов.

/// <summary>

/// Экзамен

/// </summary>

public class Exam : IDateAndCopy

{

/// <summary>

/// Предмет

/// </summary>

public string Subject { get; set; }

/// <summary>

/// Оценка

/// </summary>

public int Grade { get; set; }

/// <summary>

/// Дата экзамена

/// </summary>

public DateTime Date { get; set; }

public Exam()

{

Subject = StringExtension.GetRandom(7);

Grade = new Random().Next(1, 6);

Date = DateTime.UtcNow;

}

public Exam(string subject, int grade, DateTime date)

{

Subject = subject;

Grade = grade;

Date = date;

}

/// <summary>

/// Получение строки со всеми полями

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Предмет: {Subject}, Оценка: {Grade}, Дата начала: {Date}";

}

/// <summary>

/// Полуение полнй копии

/// </summary>

/// <returns></returns>

/// <exception cref="NotImplementedException"></exception>

public object DeepCopy()

{

return new Exam(Subject, Grade, Date);

}

}

1. Определяем класс Test для описания зачетов у студентов.

public class Test : IDateAndCopy

{

/// <summary>

/// Название

/// </summary>

public string Name { get; set; }

/// <summary>

/// Сдан ли

/// </summary>

public bool IsPassed { get; set; }

/// <summary>

/// Дата рождения

/// </summary>

public DateTime Date { get; set; }

public Test(string name, bool isPassed)

{

Name = name;

IsPassed = isPassed;

}

public Test()

{

Name = StringExtension.GetRandom(10);

IsPassed = false;

}

/// <summary>

/// Получение строки объекта

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Имя: {Name}, Зачет: {IsPassed}";

}

/// <summary>

/// Полная копия объекта

/// </summary>

/// <returns></returns>

public object DeepCopy()

{

return new Test(Name, IsPassed);

}

}

1. Определяем интерфейс IdateAndCopy из второй лабораторной работы.

public interface IDateAndCopy

{

/// <summary>

/// Копирование

/// </summary>

/// <returns></returns>

object DeepCopy();

/// <summary>

/// Дата

/// </summary>

DateTime Date { get; set; }

}

1. Для описания образования студентов был определен во второй лабораторной перечисление EducationEnum

public enum EducationEnum

{

/// <summary>

/// Специалист

/// </summary>

Specialist,

/// <summary>

/// Балаклавр

/// </summary>

Вachelor,

/// <summary>

/// Второе образование

/// </summary>

SecondEducation

}

Консольный вывод:

Создаем коллекцию элементов Student и заполняем ее случайными студентами. Выводим на экран что получилось.

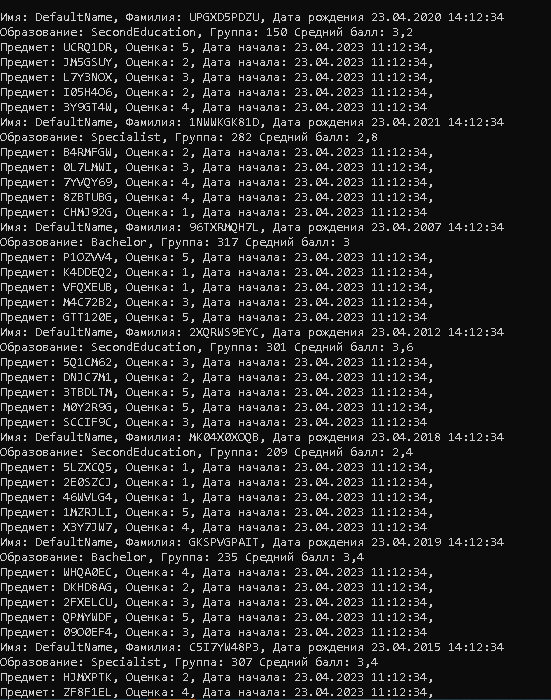


Рисунок 1 Первоначальный вывод списка студентов

Далее применяем сортировку по фамилии студентов

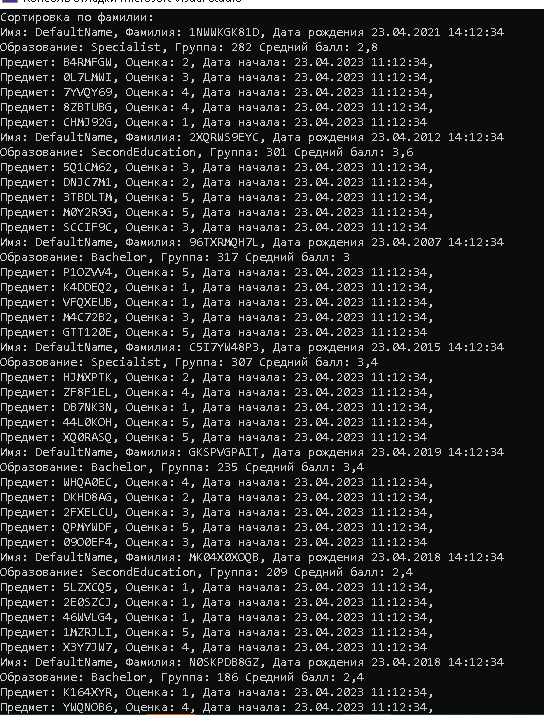


Рисунок 2 Применениие сортировки по фамилии

Применяем сортировку по дате и выводим результат.

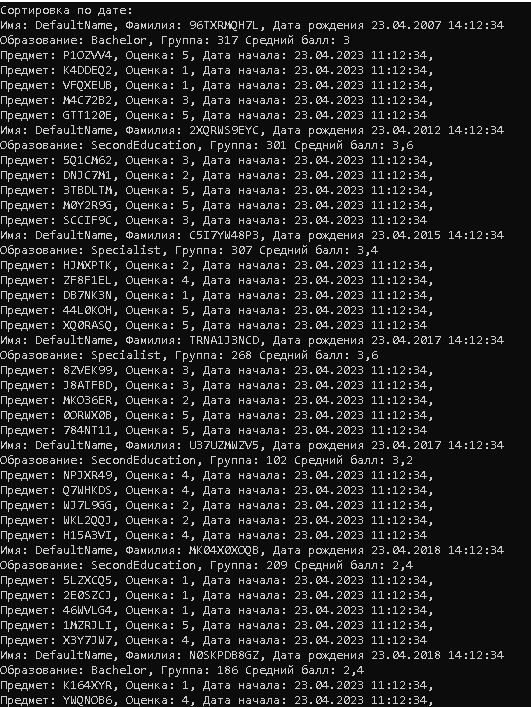


Рисунок 3 Применение сортировки по дате

Применяем сортировку по средней оценке и выводим результат

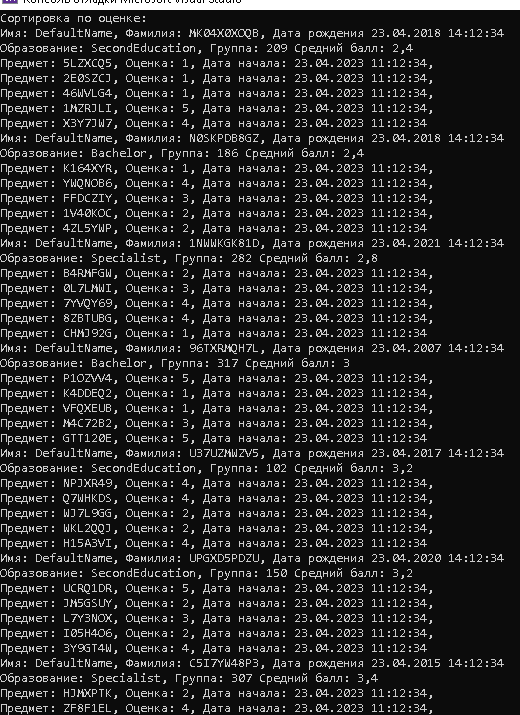


Рисунок 4 Применяем сортировку по среднему баллу

Далее выводим максимальный средний балл среди студентов, ищем среди списка специалистов и выводим сгруппированных студентов по оценке 3

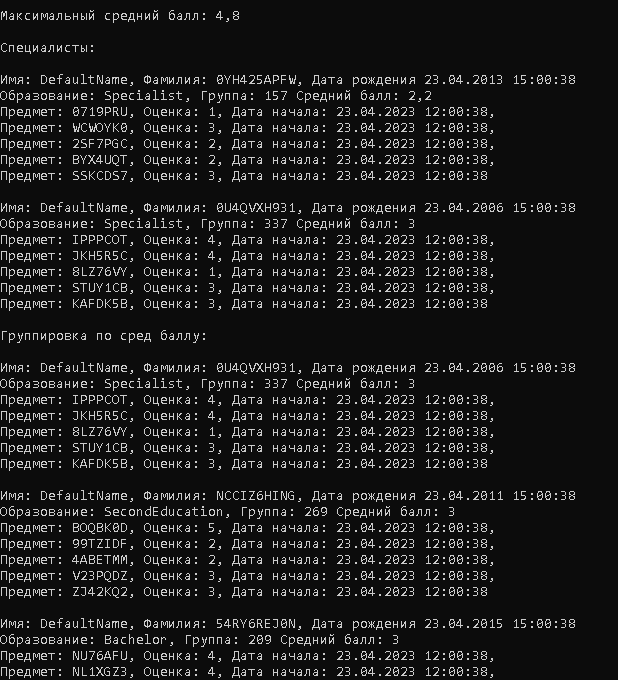


Рисунок 5 Получение максимального балла, сгруппированного списка по оценке и списка специалистов

Создаем тестовый класс содержащий коллекции элементов над которыми будет проведено тестирование. Выводим результаты тестирования

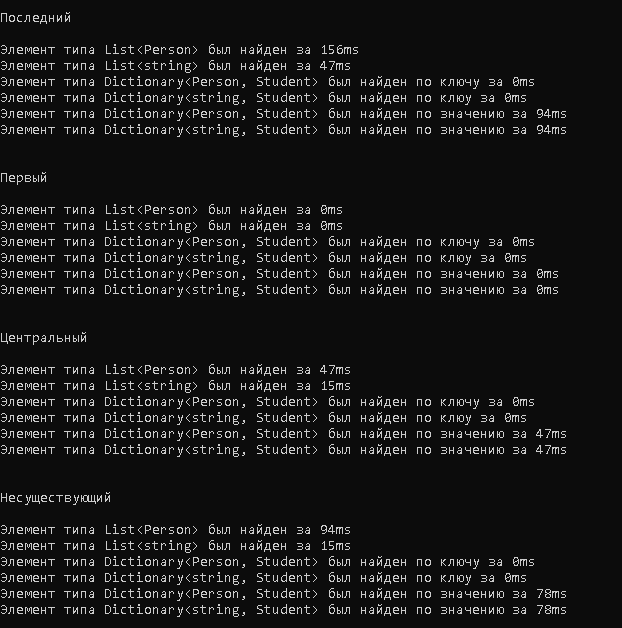


Рисунок 6 Тестирование коллекций элементов