Лабораторная работа 3

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

Требования к программе, общие для всех вариантов

Во всех вариантах второго уровня требуется определить универсальный делегат

delegate System.Collections.Generic.KeyValuePair GenerateElement(int j);

и универсальный класс TestCollections, который содержит закрытые поля следующих типов

• System.Collections.Generic.List;

• System.Collections.Generic.List ;

• System.Collections.Generic.Dictionary ;

• System.Collections.Generic.Dictionary;

• GenerateElement.

Конкретные значения типовых параметров TKey и TValue зависят от варианта.

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

Для автоматической генерации элементов коллекций надо определить метод, который принимает один целочисленный параметр типа int и возвращает ссылку на объект типа KeyValuePair. Метод должен инициализировать объекты KeyValuePair так, чтобы соответствие между номером элемента и объектом TKey в паре ключ-значение было взаимно-однозначным.

Метод для автоматической генерации элементов коллекций передается в класс TestCollections через параметр конструктора класса. Для этого в классе TestCollections надо определить конструктор c двумя параметрами, имеющими тип int и GenerateElement. Через целочисленный параметр объектам класса передается число элементов в коллекциях, через экземпляр делегата GenerateElement – метод, который используется для автоматической генерации пары ключ-значение в виде объекта KeyValuePair.

Число элементов в коллекциях пользователь вводит 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, его можно использовать как при создании коллекций, так и для генерации элемента для поиска.

Вариант 1.

Требования к программе

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 для сравнения объектов типа Exam по оценке.

Определить вспомогательный класс, реализующий интерфейс System.Collections.Generic.IComparer, 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 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.

В классе Student определить методы для сортировки списка экзаменов

• по названию предмета;

• по оценке;

• по дате экзамена.

Определить универсальный делегат

delegate TKey KeySelector(Student st);

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

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

Класс StudentCollection содержит

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

• закрытое поле типа KeySelector для хранения экземпляра делегата с методом, вычисляющим ключ для объекта Student;

• конструктор c одним параметром типа KeySelector ;

• 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 коллекцию Dictionary;

• перегруженную версию виртуального метода string ToString() для формирования строки, содержащей информацию обо всех элементах коллекции Dictionary, в том числе значения всех полей класса Student, включая список зачетов и экзаменов;

• метод string ToShortString(), который формирует строку c информацией обо всех элементах коллекции Dictionary, состоящую из значений всех полей, среднего балла, числа зачетов и экзаменов для каждого элемента Student, но без списка зачетов и экзаменов.

В классе StudentCollection определить свойства и методы, выполняющие операции со словарем Dictionary 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; если в коллекции нет элементов, свойство 39 возвращает некоторое значение по умолчанию; для поиска максимального значения среднего балла надо использовать метод Max класса System.Linq.Enumerable;

• метод IEnumerable<KeyValuePair>EducationForm(Education value), возвращающий подмножество элементов коллекции Dictionary с заданной формой обучения; для формирования подмножества использовать метод Where класса System.Linq.Enumerable;

• свойство типа IEnumerable<IGrouping<Education,KeyValuePair>> (только с методом get), выполняющее группировку элементов коллекции Dictionary 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 по разным критериям, после каждой сортировки вывести данные объекта. Выполнить сортировку

• по названию предмета;

• по оценке;

• по дате экзамена.

2. Создать объект типа StudentCollection. Добавить в коллекцию несколько разных элементов типа Student и вывести объект StudentCollection.

3. Вызвать методы класса StudentCollection, выполняющие операции с коллекцией-словарем Dictionary, 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. 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.

Код программы (описание в комментариях):

1. Ниже преведен класс Program и метод Main откуда начинается программа. В соответствии с заданием так же были добавлены некоторые статические методы.

public class Program

{

static void Main(string[] args)

{

//Создание коллекции студентов

var student = new Student();

//Добавление стуентов по умолчанию

student.AddExams(new List<Exam> { new Exam(), new Exam(), new Exam()}) ;

//Вывод

WriteInfo("Получение первоначальных данных студента:", ConsoleColor.Green);

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

//Сортировка по названию

student.SortByNameSubject();

WriteInfo("Сортировка по названию: ", ConsoleColor.Green);

Console.WriteLine(student.ToString());

//Сортировка по дате

student.SortByDate();

WriteInfo("Сортировка по дате: ", ConsoleColor.Green);

Console.WriteLine(student.ToString());

//Сортировка по оценке

student.SortByGrade();

WriteInfo("Сортировка по оценке: ", ConsoleColor.Green);

Console.WriteLine(student.ToString());

var collection = new StudentCollection<string>(GetRandomValue);

collection.AddDefaults(10);

WriteInfo("Вывод списка студентов: ", ConsoleColor.Green);

Console.WriteLine(collection.ToString());

WriteInfo("Максимальный балл: ", ConsoleColor.Green);

Console.WriteLine(collection.MaxGrade);

WriteInfo("Получение студентов по образованию: ", ConsoleColor.Green);

Console.Write("Введите номер образования для вывода студентов с таким же

номером: ");

EducationEnum education =(EducationEnum)Convert.ToInt32(Console.ReadLine());

Console.WriteLine(String.Join('\n',

collection.EducationForm(education).Select(c => c.ToString() + "\n")));

WriteInfo("Получение студентов сгруппированных по образованиям: ",

ConsoleColor.Green);

foreach(var item in collection.GetEducationGroup)

{

Console.WriteLine($"Группа {item.Key}\n" + String.Join('\n',

item.Select(h => h.Value.ToString())));

}

//Тестирование коллекций элементов

WriteInfo("Тестирование коллекция элементов: ", ConsoleColor.Green);

Console.Write("Введите число для определения размера коллекций: ");

var count = Convert.ToInt32(Console.ReadLine());

var test = new TestCollections<Person, Student>(count, GetRandomPerson);

WriteInfo("\n\nПоиск последнего: ", ConsoleColor.Green);

Console.WriteLine(test.GetTimeForFindItem(count - 1));

WriteInfo("\n\nПоиск первого: ", ConsoleColor.Green);

Console.WriteLine( test.GetTimeForFindItem(0));

WriteInfo("\n\nПоиск Центрального: ", ConsoleColor.Green);

Console.WriteLine(test.GetTimeForFindItem(count / 2));

WriteInfo("\n\nПоиск не существующего: ", ConsoleColor.Green);

Console.WriteLine(test.GetTimeForFindItem(count));

}

/// <summary>

/// Создание ключа

/// </summary>

/// <param name="student"></param>

/// <returns></returns>

private static string GetRandomValue(Student student)

{

return $"{student.Name} {student.LastName}";

}

/// <summary>

/// Получение клю-значение

/// </summary>

/// <param name="i"></param>

/// <returns></returns>

private static KeyValuePair<Person, Student> GetRandomPerson(int i)

{

return new KeyValuePair<Person, Student>(new Person($"{i}", $":{i}",

DateTime.Now), new Student());

}

/// <summary>

/// Оформленный вывод информации на экран

/// </summary>

/// <param name="str"></param>

/// <param name="color"></param>

private static void WriteInfo(string str, ConsoleColor color)

{

Console.ForegroundColor = color;

Console.WriteLine(str);

Console.ResetColor();

}

}

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;

}

}

/// <summary>

/// Сортировка по названию предмета

/// </summary>

public void SortByNameSubject()

{

\_exams.Sort(new Exam());

}

/// <summary>

/// Собтировка по оценке

/// </summary>

public void SortByGrade()

{

\_exams.Sort();

}

/// <summary>

/// Собтировка по дате

/// </summary>

public void SortByDate()

{

\_exams.Sort(new ExamComparer());

}

}

1. Определяем новую версию класса Exam

/// <summary>

/// Экзамен

/// </summary>

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

{

/// <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);

}

/// <summary>

/// Реализация метода compareto для сортировки по оценке

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

public int CompareTo(Exam? other)

{

return Grade.CompareTo(other?.Grade);

}

/// <summary>

/// Реализация метода compare для сортировки по названию предмета

/// </summary>

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

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

/// <returns></returns>

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

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

{

if (x is null || y is null)

throw new Exception("Объект равен null");

return String.Compare(x.Subject, y.Subject);

}

}

1. Определяем новый класс StudentCollection

/// <summary>

/// Коллекция студентов

/// </summary>

public class StudentCollection<TKey> : IEnumerable<Student>

{

/// <summary>

/// Делегат селектора ключа

/// </summary>

/// <typeparam name="TKey"></typeparam>

/// <param name="st"></param>

/// <returns></returns>

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

/// <summary>

/// Селектор

/// </summary>

private readonly KeySelector<TKey> \_keySelector;

/// <summary>

/// Студенты

/// </summary>

private Dictionary<TKey,Student> \_students = new Dictionary<TKey,Student>();

public StudentCollection(KeySelector<TKey> selector)

{

\_students = new Dictionary<TKey, Student>();

\_keySelector = selector;

}

/// <summary>

/// Максимальная оценка

/// </summary>

public double MaxGrade

{

get

{

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

return 0;

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

}

}

/// <summary>

/// Получение студентов с определенным типом образования

/// </summary>

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

/// <returns></returns>

public List<Student> EducationForm(EducationEnum value)

{

return \_students.Where(c => c.Value.Education == value).Select(c =>

c.Value).ToList();

}

/// <summary>

/// Группировка по образованию

/// </summary>

public IEnumerable<IGrouping<EducationEnum, KeyValuePair<TKey, Student>>>

GetEducationGroup

{

get

{

return \_students.GroupBy(c => c.Value.Education);

}

}

/// <summary>

/// Получение специалистов

/// </summary>

/// <returns></returns>

public IEnumerable<Student> StudentsWithSpecialist

{

get

{

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

Practice.Models.Enums.EducationEnum.Specialist))

{

yield return student.Value;

}

}

}

/// <summary>

/// Получение студентов со средним баллом

/// </summary>

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

/// <returns></returns>

public List<Student> AverageMarkGroup(double value)

{

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

c.Key == value)?.Select(c => c.Value).ToList() ?? new List<Student>();

}

/// <summary>

/// Добавление студентов

/// </summary>

public void AddStudents(params Student[] students)

{

foreach(var item in students)

{

\_students.Add(\_keySelector(item), item);

}

}

/// <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(\_keySelector(stud),stud);

}

}

/// <summary>

/// Получение строки объекта

/// </summary>

/// <returns></returns>

public override string ToString()

{

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

}

/// <summary>

/// Возвращает короткую версию строки элемента

/// </summary>

/// <returns></returns>

public string ToShortString()

{

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

c.Value.ToShortString()));

}

/// <summary>

/// Полученре перечисления

/// </summary>

/// <returns></returns>

public IEnumerator<Student> GetEnumerator()

{

return \_students.Values.GetEnumerator();

}

/// <summary>

/// Получение перечисления

/// </summary>

/// <returns></returns>

IEnumerator IEnumerable.GetEnumerator()

{

return \_students.GetEnumerator();

}

}

1. Определяем новый класс для тестирования работы коллекций TestCollections

/// <summary>

/// Тест коллекций

/// </summary>

public class TestCollections<TKey, TValue>

{

public delegate KeyValuePair<TKey, TValue> GenerateElement(int j);

/// <summary>

/// Список для теста

/// </summary>

public List<TKey> TestTKeys;

/// <summary>

/// Список для теста

/// </summary>

public List<string> TestTString;

/// <summary>

/// Словарь для теста где ключом выступает класс пользователя

/// </summary>

public Dictionary<TKey, TValue> TestDictionaryTKey;

/// <summary>

/// Словарь для теста где ключом выступает строка класса

/// </summary>

public Dictionary<string, TValue> TestDictionaryString;

/// <summary>

/// Клю-значение для теста

/// </summary>

private GenerateElement \_testGenerateElement;

/// <summary>

/// Генерация случайных чисел

/// </summary>

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

public TestCollections(int count, GenerateElement gen)

{

\_testGenerateElement = gen;

TestTKeys = new List<TKey>();

TestTString = new List<string>();

TestDictionaryTKey = new Dictionary<TKey, TValue>();

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

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

{

TestTKeys.Add(\_testGenerateElement(i).Key);

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

DateTime.UtcNow).ToString());

var newPerson = \_testGenerateElement(i);

TestDictionaryString.Add(newPerson.Key.ToString(), newPerson.Value);

var newPerson2 = \_testGenerateElement(i);

TestDictionaryTKey.Add(newPerson2.Key, newPerson2.Value);

//TestGenerateElement.Add();

}

}

/// <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) ??

\_testGenerateElement(1).Key;

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) ??

\_testGenerateElement(1).Key;

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) ??

\_testGenerateElement(1).Value;

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) ??

\_testGenerateElement(1).Value;

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. Определяем класс реализующий интерфейс Icomparer для сортировки экзаменов

public class ExamComparer : IComparer<Exam>

{

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

{

if (x is null || y is null)

throw new Exception("Объект равен null");

if (x.Date > y.Date)

{

return 1;

}

else if (x.Date < y.Date)

return -1;

else

return 0;

}

}

1. Определяем класс Test для описания зачетов 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. Создаем интерфейс с прошлой лабораторной работы IdateAndCopy

public interface IDateAndCopy

{

/// <summary>

/// Копирование

/// </summary>

/// <returns></returns>

object DeepCopy();

/// <summary>

/// Дата

/// </summary>

DateTime Date { get; set; }

}

Консольный вывод:

Создаем одного студента и проводим операции сортировки по списку экзаменов

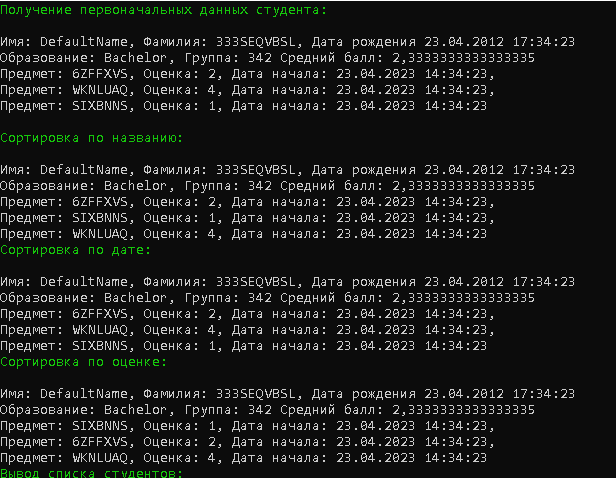


Рисунок 1 Сортировка списка экзаменов студента

Создаем список студентов и среди них ищем максимальный балл.После выводим список студентов найденных по образованию

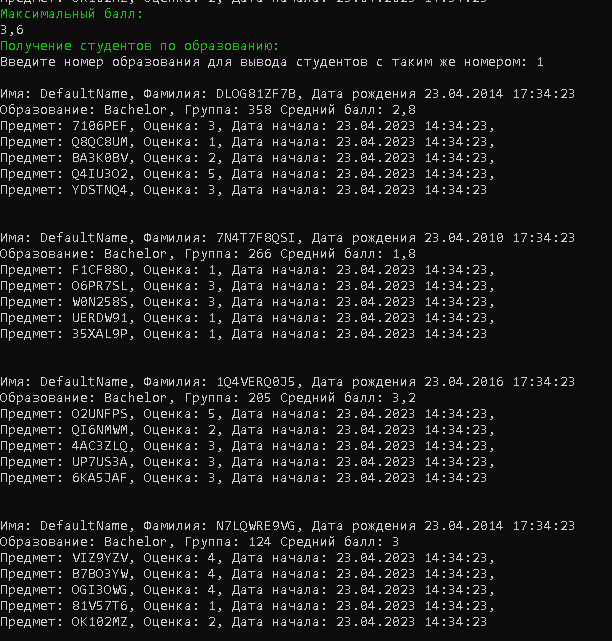


Рисунок 2 Получение максимального балла и списка студентов по образованию

Следующим шагом получаем список сгруппированных студентов по образованию

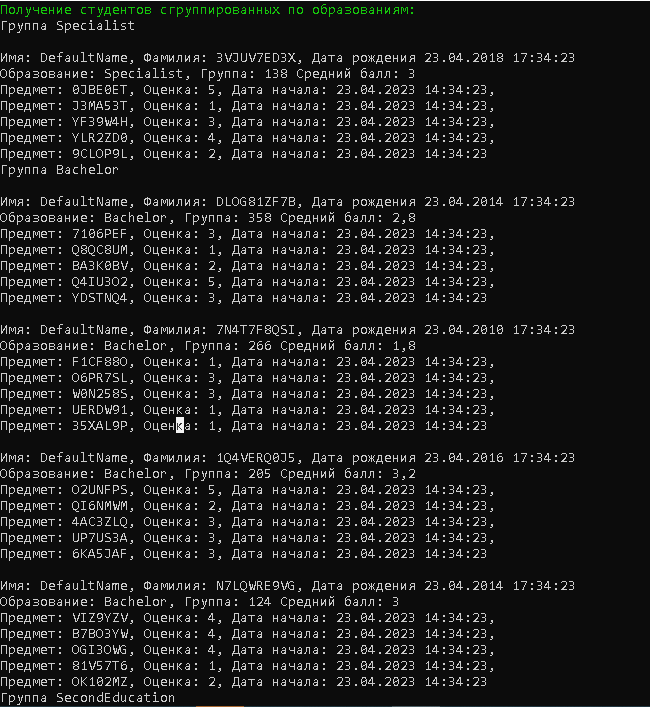


Рисунок 3 Получение сгруппированых студентов по образованию

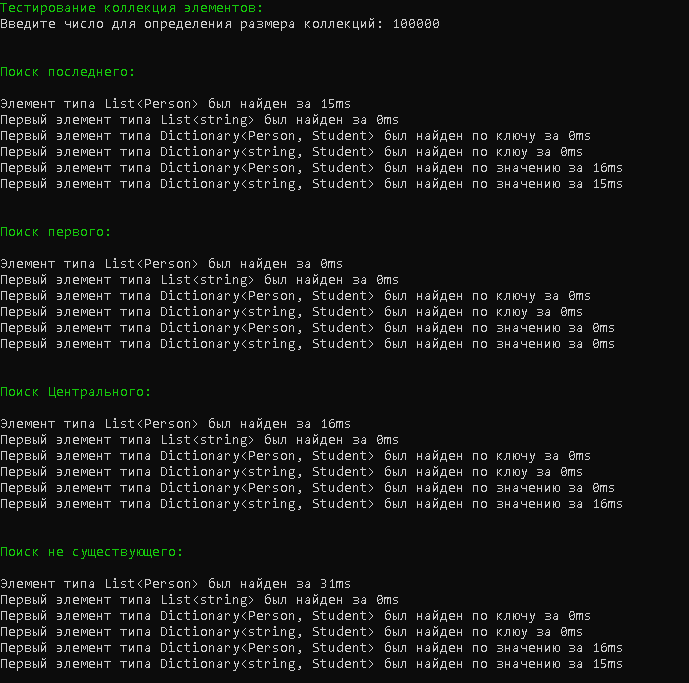


Рисунок 4 Тестирование поиска элементов коллекций