Лабораторная работа 4. Варианты первого уровня

Делегаты. События.

Требования к программе, общие для всех вариантов

В лабораторной работе требуется определить класс, содержащий типизированную коллекцию, который с помощью событий извещает об изменениях в коллекции.

Коллекция состоит из объектов ссылочных типов. Коллекция изменяется при удалении/добавлении элементов или при изменении одной из входящих в коллекцию ссылок, например, когда одной из ссылок присваивается новое значение. В этом случае в соответствующих методах или свойствах класса бросаются события.

При изменении данных объектов, ссылки на которые входят в коллекцию, значения самих ссылок не изменяются. Этот тип изменений не порождает событий.

Для событий, извещающих об изменениях в коллекции, определяется свой делегат. События регистрируются в специальных классах-слушателях.

Вариант 1.

Требования к программе

Определить новую версию класса StudentCollection из лабораторной работы 3, которая с помощью событий сообщает об изменениях в коллекции.

Для событий определить делегат StudentListHandler с сигнатурой:

void StudentListHandler (object source, StudentListHandlerEventArgs args);

Класс StudentListHandlerEventArgs, производный от класса System.EventArgs, содержит

• открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие;

• открытое автореализуемое свойство типа string с информацией о типе изменений в коллекции;

• открытое автореализуемое свойство типа Student для ссылки на объект, с которым связаны изменения;

• конструкторы для инициализации класса;

• перегруженную версию метода string ToString() для формирования строки с информацией обо всех полях класса.

В новую версию класса StudentCollection добавить

• открытое автореализуемое свойство типа string с названием коллекции;

• метод bool Remove (int j) для удаления элемента с номером j из списка List; если в списке нет элемента с номером j, метод возвращает значение false;

• индексатор типа Student (с методами get и set) с целочисленным индексом для доступа к элементу списка List с заданным номером.

В новую версию класса StudentCollection добавить два события типа StudentListHandler

• StudentsCountChanged, которое происходит при добавлении нового элемента в коллекцию или при удалении элемента из коллекции; через объект StudentListHandlerEventArgs cобытие передает имя коллекции, строку с информацией о том, что в коллекцию был добавлен новый элемент или из нее был удален элемент, ссылку на добавленный или удаленный элемент Student;

• StudentReferenceChanged, которое происходит, когда одной из ссылок, входящих в коллекцию, присваивается новое значение; через объект StudentListHandlerEventArgs событие передает имя коллекции, строку с информацией о том, что был заменен элемент в коллекции, и ссылку на новый элемент Student.

Событие StudentsCountChanged бросают следующие методы класса StudentCollection

• AddDefaults();

• AddStudents (params Student[] ) ;

• Remove (int j).

Событие StudentReferenceChanged бросает метод set индексатора, определенного в классе StudentCollection.

Определить класс Journal, который можно использовать для накопления информации об изменениях в коллекциях типа StudentCollection. Класс Journal хранит информацию в списке объектов типа JournalEntry. Каждый элемент списка содержит информацию об отдельном изменении, которое произошло в коллекции.

Класс JournalEntry содержит

• открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие;

• открытое автореализуемое свойство типа string с информацией о типе изменений в коллекции;

• открытое автореализуемое свойство типа string c данными объекта Student, с которым связаны изменения в коллекции; • конструктор для инициализации полей класса;

• перегруженную версию метода string ToString().

Класс Journal содержит

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

•обработчики событий StudentsCountChanged и StudentReferenceChanged, которые добавляют новый элемент JournalEntry в список List; для инициализации JournalEntry используется информация из объекта StudentListHandlerEventArgs, который передается вместе с событием;

• перегруженную версию метода string ToString() для формирования строки с информацией обо всех элементах списка List.

В методе Main()

1. Создать две коллекции StudentCollection.

2. Создать два объекта типа Journal, один объект Journal подписать на события StudentsCountChanged и StudentReferenceChanged из первой коллекции StudentCollection, другой объект Journal подписать на события StudentReferenceChanged из обеих коллекций StudentCollection. 3. Внести изменения в коллекции StudentCollection • добавить элементы в коллекции;

• удалить некоторые элементы из коллекций;

• присвоить некоторым элементам коллекций новые значения.

4. Вывести данные обоих объектов Journa

Код программы (описание в комментариях):

1. Ниже преведен класс Program и метод Main откуда начинается программа.

public class Program

{

static void Main(string[] args)

{

//Создание двух коллекций студентов

var collection1 = new StudentCollection();

var collection2 = new StudentCollection();

//Присваивание имен коллекциям

collection1.CollectionName = "TestListCollection1";

collection2.CollectionName = "TestListCollection2";

//Создание двух журналов

var journal1 = new Journal();

var journal2 = new Journal();

//Подписки на события обновления списков

collection1.StudentsCountChanged += journal1.StudentsCountChanged;

collection1.StudentReferenceChanged += journal1.StudentReferenceChanged;

collection2.StudentReferenceChanged += journal2.StudentReferenceChanged;

collection1.StudentReferenceChanged += journal2.StudentReferenceChanged;

//Добавление студентов

collection1.AddStudents(new Practice.Models.Student(), new Student());

collection2.AddStudents(new Practice.Models.Student(), new Student());

//Удаленпие

collection1.Remove(0);

collection2.Remove(0);

//Обновление существующего

collection1[0] = new Student();

collection2[0] = new Student();

///Вывод на экран

Console.WriteLine(journal1.ToString());

Console.WriteLine(journal2.ToString());

}

}

1. Воссоздаем класс Student из прежних лабораторных работ

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 рождения

/// </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.Date = value.Date;

}

}

/// <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.Date)

{

\_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. Воссоздаем класс Person из прежних лабораторных работ

/// <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>

/// Год рождения

/// </summary>

public int CustomizeDateOfBirthYear

{

get

{

return Date.Year;

}

set

{

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

}

}

/// <summary>

/// Дата рождения

/// </summary>

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 && Date

== person.Date;

}

/// <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, Date).GetHashCode();

}

/// <summary>

/// Создание полной копии

/// </summary>

/// <returns></returns>

public virtual object DeepCopy()

{

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

}

/// <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.Date > y.Date)

{

return 1;

}

else if (x.Date < y.Date)

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. Воссоздаем класс Exam из прежних лабораторных работ

/// <summary>

/// Экзамен

/// </summary>

public class Exam : IDateAndCopy

{

/// <summary>

/// Предмет

/// </summary>

public string Subject { get; set; }

/// <summary>

/// Оценка

/// </summary>

public int Grade { get; set; }

/// <summary>

/// Начало экзамена

/// </summary>

public DateTime DateStart { get; set; }

/// <summary>

/// Дата

/// </summary>

public DateTime Date { get; set; }

public Exam()

{

Subject = StringExtension.GetRandom(7);

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

DateStart = DateTime.UtcNow;

}

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

{

Subject = subject;

Grade = grade;

DateStart = date;

}

/// <summary>

/// Получение строки со всеми полями

/// </summary>

/// <returns></returns>

public override string ToString()

{

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

}

/// <summary>

/// Полуение полнй копии

/// </summary>

/// <returns></returns>

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

public object DeepCopy()

{

return new Exam(Subject, Grade, DateStart);

}

}

1. Воссоздаем класс Test из прежних лабораторных работ

public class Test : IDateAndCopy

{

/// <summary>

/// Название

/// </summary>

public string Name { get; set; }

/// <summary>

/// Сдан ли

/// </summary>

public bool IsPassed { get; set; }

/// <summary>

/// Дата сдачи

/// </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>

/// Преобразование объекта к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

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

}

/// <summary>

/// Полное копирование

/// </summary>

/// <returns></returns>

public object DeepCopy()

{

return new Test(Name, IsPassed);

}

}

1. Определяем новую версию класса StudentCollection для описания коллекции студентов.

public class StudentCollection : IEnumerable<Student>

{

/// <summary>

/// Студенты

/// </summary>

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

/// <summary>

/// Название коллекции

/// </summary>

public string CollectionName { get; set; } = "ListCollection";

/// <summary>

/// Делегат

/// </summary>

/// <param name="sourse"></param>

/// <param name="args"></param>

public delegate void StudentListHandler(object sourse,

StudentListHandlerEventArgs args);

/// <summary>

/// Событие вызываеморе при изменении размера коллекции

/// </summary>

public event StudentListHandler StudentsCountChanged;

/// <summary>

/// Вызываемое при изменении ссылки объекта

/// </summary>

public event StudentListHandler StudentReferenceChanged;

/// <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);

if(StudentsCountChanged != null)

foreach(var item in students)

{

StudentsCountChanged.Invoke(null, new

StudentListHandlerEventArgs

{

Name = CollectionName,

Student = item,

OperationName = "Add"

});

}

}

/// <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);

StudentsCountChanged.Invoke(null, new StudentListHandlerEventArgs

{

Name = CollectionName,

Student = stud,

OperationName = "Add"

});

}

}

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>

/// Получение перечисления

/// </summary>

/// <returns></returns>

public IEnumerator<Student> GetEnumerator()

{

return \_students.GetEnumerator();

}

/// <summary>

/// Получение перечисления

/// </summary>

/// <returns></returns>

IEnumerator IEnumerable.GetEnumerator()

{

return \_students.GetEnumerator();

}

/// <summary>

/// Метод удаления элемента

/// </summary>

/// <param name="j"></param>

/// <returns></returns>

public bool Remove(int j)

{

try

{

var stud = \_students[j];

\_students.RemoveAt(j);

if(StudentsCountChanged != null)

StudentsCountChanged.Invoke(null, new

StudentListHandlerEventArgs

{

Name = CollectionName,

Student = stud,

OperationName = "Remove"

});

return true;

}

catch(Exception ex)

{

return false;

}

}

/// <summary>

/// индексатор

/// </summary>

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

/// <returns></returns>

public Student this[int index]

{

get => \_students[index];

set

{

\_students[index] = value;

StudentReferenceChanged.Invoke(null, new StudentListHandlerEventArgs

{

Student = value,

Name = CollectionName,

OperationName = "Update"

});

}

}

}

1. Определяем класс StudentListHandlerEventArgs для описания модели передаваемой в обработчик события

public class StudentListHandlerEventArgs : EventArgs

{

/// <summary>

/// Название теста

/// </summary>

public string Name { get; set; }

/// <summary>

/// Название операции проводимой над коллекций

/// </summary>

public string OperationName { get; set; }

/// <summary>

/// Объект студента

/// </summary>

public Student Student { get; set; }

public StudentListHandlerEventArgs(Student student, string name)

{

Student = student;

Name = name;

}

public StudentListHandlerEventArgs() { }

/// <summary>

/// Получение строки с объекта

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Название коллекции: {Name}\nНазвание операции:

{OperationName}";

}

}

1. Создаем класс Journal для описания журнала событий

public class Journal

{

/// <summary>

/// Список записей журнала

/// </summary>

private List<JournalEntry> \_journalsEntry;

public Journal()

{

\_journalsEntry = new List<JournalEntry>();

}

/// <summary>

/// Обработчик обновления студента

/// </summary>

/// <param name="sourse"></param>

/// <param name="args"></param>

public void StudentReferenceChanged(object sourse,

StudentListHandlerEventArgs args)

{

\_journalsEntry.Add(new JournalEntry

{

CollectionName = args.Name,

OperationName = args.OperationName,

Student = args.Student.ToString()

});

}

/// <summary>

/// Обработчик обновления списка студентов

/// </summary>

/// <param name="sourse"></param>

/// <param name="args"></param>

public void StudentsCountChanged(object sourse, StudentListHandlerEventArgs

args)

{

\_journalsEntry.Add(new JournalEntry

{

CollectionName = args.Name,

OperationName = args.OperationName,

Student = args.Student.ToString()

});

}

/// <summary>

/// Конверт к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

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

c.ToString()));

}

}

1. Создаем класс JournalEntry для описания элемента журнала событий

public class JournalEntry

{

/// <summary>

/// Название коллекции запись которой создана

/// </summary>

public string CollectionName { get; set; }

/// <summary>

/// Название операции

/// </summary>

public string OperationName { get; set; }

/// <summary>

/// Информация о студенте

/// </summary>

public string Student { get; set; }

public JournalEntry(string collectionName, string operationName, string

student)

{

CollectionName = collectionName;

OperationName = operationName;

Student = student;

}

public JournalEntry() { }

/// <summary>

/// Преобразование к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

return $"Название коллекции: {CollectionName}\nНазвание операции:

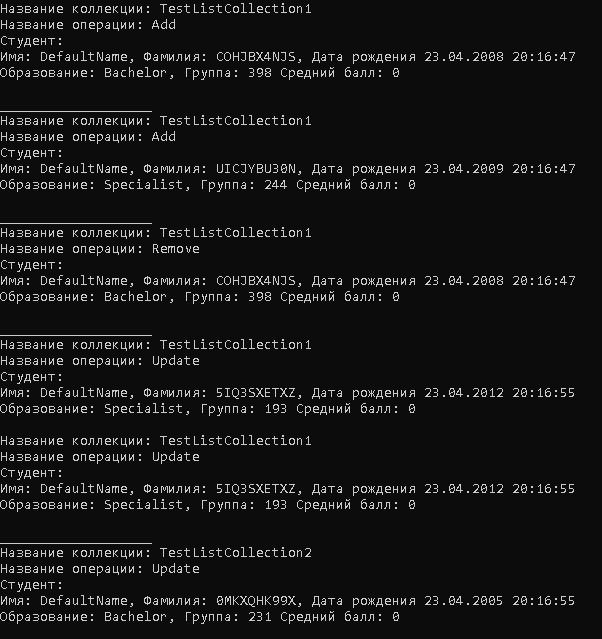
{OperationName}\nСтудент:{Student}";

}

}

Консольный вывод:

С начала мы создаем две коллекции StudentCollection элементов Student. Присваиваем им соответствующие названия для того чтобы можно было их различать при просмотре журнала. После создали 2 журнала и подписались на события коллекций. Все проводимые операции разделяются знаком «\_». Сначала мы добавляем новых студентов в коллекции, срабатывает событие и выводит сообщение на консоль. Далее операции удаления и обновления с выводом на экран



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

Следующим шагом получаем список сгруппированных студентов по образованию