

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра информационных технологий и компьютерных систем

ПРОГРАММИРОВАНИЕ. ЯЗЫКИ ВЫСОКОГО УРОВНЯ

Методические указания
к лабораторным работам по дисциплине
для студентов направления
44.03.05 - Педагогическое образование
(с двумя профилями подготовки. Профиль: математика и информатика)
дневной формы обучения

В пяти частях

Часть 5

**Использование коллекций и карт отображений
при разработке java-программ**

Севастополь
СевГУ
2023

УДК 004.432(076.5)
ББК 32.973-018.1я73
П784

Рецензент:

В. Ю. Карлусов - канд. техн. наук, доцент кафедры информационных систем

Составители: Т. В. Волкова, Е. С. Владимирова

П784 Программирование. Языки высокого уровня. [В 5-ти частях].
Часть 5. Использование коллекций и карт отображений при разработке
java-программ : методические указания к лабораторным работам по
дисциплине для студентов направления 44.03.05 – Педагогическое
образование (с двумя профилями подготовки. Профиль: математика и
информатика) дневной формы обучения / Севастопольский
государственный университет, Институт информационных технологий,
кафедра информационных технологий и компьютерных систем ; сост.
Т. В. Волкова, Е. С. Владимирова. – Севастополь : СевГУ, 2023. – 87 с.
– Текст : электронный.

Целью методических указаний является оказание помощи в подготовке к лабораторным работам по дисциплине «Программирование. Языки высокого уровня», предусматривающим разработку и выполнение программ с использованием мощного средства библиотеки Java – Java Collection Framework (JCF), предназначенного для обработки списков, множеств и карт отображений объектов.

Предназначены для студентов первого курса дневной формы обучения направления 44.03.05 – Педагогическое образование (с двумя профилями подготовки. Профиль: математика и информатика).

УДК 004.432(076.5)
ББК 32.973-018.1я73

Рассмотрены и рекомендованы к изданию на заседании кафедры информационных технологий и компьютерных систем, протокол № 2 от 14 сентября 2022 г.

© Волкова Т. В., Владимирова Е. С., сост., 2023
© ФГАОУ ВО «Севастопольский
государственный университет», 2023

СОДЕРЖАНИЕ

1. ОБЩИЕ ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ	4
1. ЛАБОРАТОРНАЯ РАБОТА № 12. ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ JAVA COLLECTION FRAMEWORK (JCF). СПИСКИ И МНОЖЕСТВА (НАБОРЫ).....	6
2.ЛАБОРАТОРНАЯ РАБОТА № 13. ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ JAVA COLLECTION FRAMEWORK (JCF). КАРТЫ ОТОБРАЖЕНИЙ.....	62
БИБЛИОГРАФИЧЕСКИЙ СПИСОК	88

1. ОБЩИЕ ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Для освоения основ программирования на языке Java предлагается серия лабораторных работ. Каждая работа требует изучения методических указаний и рекомендованной литературы, подготовку текста программы, компиляцию программы некоторое ее исследование.

Все действия, указанные в методических указаниях к отдельной работе (освоение практических приемов использования системы разработки программ, объяснение примененных элементов языка Java, ответы на вопросы, задания) обязательно должны быть выполнены до следующего лабораторного занятия.

По выполненной лабораторной работе составляется отчет. Оформление отчета должно соответствовать требованиям к текстовым учебным документам соответствующих ГОСТов [Электронный фонд правовой и нормативно-технической документации. <http://docs.cntd.ru>].

Отчет представляется на листах формата А4 и в электронном виде (файл Microsoft Word). Первый лист отчета – титульный. На нем указывается название вуза, название выпускающей кафедры, название работы, ФИО и группа исполнителя, ФИО принимающего работу, город (Севастополь) и год. Пример оформления титульного листа приведен в приложении А. Следующие листы являются собственно отчетом и должны содержать следующие разделы:

- 1) цель работы
- 2) постановка задачи;
- 3) анализ задачи, выявляющий связи между элементами задачи (обоснование типов входных и выходных данных, описание реализуемых функций);
- 4) схема и описание алгоритма решения задачи;
- 5) тестовые примеры и результаты их обработки вручную;
- 6) текст Java-программы, заданной вариантом задания;
- 7) сведения об отладке программы и проверке ее работоспособности (описание ошибок (синтаксических и логических), выявленных на этапе отладки программы, результаты работы (в виде скриншотов), сравнение результатов работы программы на тестовых примерах с результатами ручных расчетов);
- 8) выводы (констатирует решение всех задач, описанных в разделе «Постановка задачи»).

Студент обязан завести рабочую тетрадь, в которой должны фиксироваться инструкции к работе, вопросы, возникающие при выполнении работы и ответы на них, черновики и фрагменты программ. В рабочей тетради можно представлять отчеты по работам. При этом титульный лист

каждого отчета должен находиться на правой странице разворота. Рабочую тетрадь обязательно иметь на каждом лабораторном занятии – будут оцениваться качество и полнота выполненных заданий.

Студент допускается к защите отчета по лабораторной работе после того, как он продемонстрирует преподавателю выполнение поставленной задачи на компьютере (результаты работы программы и/или другое задание, например, работу с окном кода BlueJ) и предоставит папку с разработанным java-проектом (в электронном виде). Рекомендуемое имя папки: Лаб_п_м_Фамилия_Группа_Год, где п – номер лабораторной работы, m – номер проекта (используется если задание по лабораторной работе предусматривает разработку нескольких проектов). Текстовый файл с отчетом по работе рекомендуется поместить в папку проекта.

Защита лабораторных работ состоит из доклада студента о проделанной работе с объяснением содержания отчета. Студент должен ответить на контрольные вопросы к соответствующей лабораторной работе, приведенные в методических указаниях, а также другие вопросы преподавателя, касающиеся поставленной задачи, показать работоспособность подготовленной программы и навыки работы с программной системой. При защите текущей работы возможны вопросы по темам предыдущих лабораторных работ. Результат защиты оценивается по шкале 0 – 100 баллов (ESTC).

Для подготовки программ студентам рекомендуется установить на своих компьютерах последнюю версию системы разработки программ BlueJ и соответствующую версию комплекта разработчика Java Development Kit (<http://www.bluej.org>).

Выполнив несложную настройку BlueJ, можно выбрать наиболее удобный для Вас язык интерфейса.

Рекомендуется иметь съемный носитель информации (флэш-диск), на котором будут храниться проекты лабораторных работ и соответствующие отчеты.

2. ЛАБОРАТОРНАЯ РАБОТА № 1. ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ JAVA COLLECTION FRAMEWORK (JCF). СПИСКИ И МНОЖЕСТВА (НАБОРЫ)

2.1. Цель работы

Целью данной работы является получение навыков использования набора классов и интерфейсов **JAVA COLLECTION FRAMEWORK** для создания коллекций (списков или множеств) объектов.

2.2. Постановка задачи

1) Разработать программу, реализующую взаимодействие объектов, образующих иерархическую структуру данных (группу объектов с заданными свойствами и функциями). Для создания объектов, определенных вариантом задания разработать соответствующие классы. Для манипулирования группой объектов использовать возможности интерфейса **List**.

2) Разработать программу, реализующую тот же алгоритм взаимодействия объектов. Для манипулирования группой объектов использовать возможности интерфейса **Set**.

3) Сравнить две реализации группы объектов.

2.3. Внеаудиторная подготовка

Для подготовки к лабораторной работе следует изучить краткие теоретические сведения, приведенные в пункте 2.4 методических указаний. Рекомендуются также ознакомиться с [1](с.413 – 433).

2.4. Краткие теоретические сведения

2.4.1. Набор классов и интерфейсов Java Collection Framework

Java содержит средства для единообразной работы с группами объектов (коллекциями) – набор классов **JCF – Java Collection Framework** (Framework (англ.) – каркас, остов, структура). Классы JCF обеспечивают работу с наборами объектов произвольной природы так, словно они находятся в массиве, но с возможностью изменения количества элементов массива.

JCF состоит из таких частей:

- интерфейсы, которые определяют функциональность коллекции;
- реализации – классы, реализующие методы интерфейсов;
- алгоритмы – методы решения типовых задач с коллекциями [1-3].

Классы и интерфейсы **JCF** определены в пакете **java.util** [4–6].

В лабораторной работе исследуются основные типы коллекций – множество и список. Множество (Set) соответствует математическому понятию «множество» с такими его характеристиками, как отсутствие копий объектов и неупорядоченность. Список (List) имеет все черты связанных списков, допускает дублирование объектов, определяет дополнительные методы, упрощающие работу со списком.

Иерархия интерфейсов:

Collection → **List**
→ **Set** → **SortedSet**

Интерфейс **SortedSet** предполагает в множестве «естественную» упорядоченность объектов, например, множество слов может быть упорядочено по алфавиту.

Универсальность интерфейсов обеспечивается типами-параметрами Java. Типы-параметры конкретизируют тип элементов коллекции и указываются в виде **<Type>** при создании коллекций.

Перемещение по коллекции от элемента к элементу обеспечивают методы интерфейса **Iterator**: **hasNext()** проверяет наличие следующего элемента, **next()** возвращает следующий элемент, **remove()** удаляет текущий элемент. Если удаление элементов не требуется, перемещение по коллекции можно производить с помощью цикла **for-each**.

Интерфейс **Collection** – базовый для интерфейсов **List** (списки), **Set** (множества) и **SortedSet** (упорядоченные множества). Интерфейсы **List** и **SortedSet** содержат дополнительные методы, упрощающие некоторые действия с коллекциями соответствующих типов.

Методы базового интерфейса Collection перечислены ниже (полный список – в документации).

boolean add(E e) добавляет объект **e** типа **E** в коллекцию, возвращает **true**, если добавление удалось.

boolean addAll(Collection<? extends E> c) добавляет в данную коллекцию все элементы коллекции **c** (**this = this ∪ c**). Тип элементов из **C** должен быть расширением типа **E**.

void clear() удаляет все элементы коллекции.

boolean contains(Object o) проверка принадлежности объекта **o** коллекции (**o ∈ this**).

boolean containsAll(Collection<?> c) проверка включения коллекции **c** в данную коллекцию (**c ⊆ this**).

boolean equals(Object o) проверяет равенство элемента коллекции объекту **o**.

boolean isEmpty() возвращает **true**, если коллекция пуста.

Iterator<E> iterator() создает итератор для перебора элементов коллекции.

boolean remove(Object o) удаляет элемент **o** из коллекции, возвращает true, если удаление удалось.

boolean removeAll(Collection<?> c) удаляет из данной коллекции все элементы коллекции **c** (**this = this \ c**).

boolean retainAll(Collection<?> c) оставляет в данной коллекции только те элементы, которые содержатся в коллекции **c** (**this = this ∩ c**).

int size() возвращает количество элементов коллекции.

Object[] toArray() возвращает массив, содержащий все элементы коллекции.

Дополнительные методы интерфейса List:

add(index, element) – добавить элемент **element** в заданной позиции списка **index**;

get(index) – получить копию элемента находящегося в заданной позиции списка;

set(index,element) – изменить элемент в заданной позиции списка;

indexOf(element) – возвращает позицию элемента в списке (просмотр слева);

lastIndexOf(element) – возвращает позицию элемента в списке (просмотр справа);

remove(index) – удалить элемент из заданной позиции списка;

subList(start,end) – создать подсписок;

listIterator(index) – вернуть итератор с возможностью перемещения с заданного элемента в прямом или обратном направлении.

Интерфейс Set расширяет интерфейс **Collection** и объявляет поведение коллекции, не допускающей дублирование элементов. Поэтому метод **add()** возвращает **false** при попытке добавить в набор дублирующие элементы. В **Set** не определяется никаких дополнительных собственных методов.

Дополнительные методы интерфейса SortedSet:

comparator() –возвращает **null** или компаратор, задающий естественный порядок объектов в наборе;

first() – возвращает копию первого (наименьшего) элемента множества;

last() – возвращает копию последнего (наибольшего) элемента множества;

headSet(e) – возвращает часть множества **this** от наименьшего объекта до **e** (исключительно);

tailSet(e) – возвращает часть множества **this**, начиная с объекта **e** и до конца;

subSet(e1, e2) – часть множества **this**, начиная с объекта **e1** (включительно) и до **e2** (исключительно).

Изменения множеств, полученных методами **headSet**, **tailSet**, **subSet**, изменяют множество **this**, и наоборот.

Примеры создания переменных-коллекций:

List<Integer> A – переменная **A**, список с элементами типа Integer;

Set<String> B – переменная **B**, множество с элементами типа String;

SortedSet<Character> C – переменная **C**, упорядоченное множество СИМВОЛОВ.

Реализации.

Классы, реализующие интерфейс List:

- ArrayList (на основе массива с возможностью увеличения его длины);
- LinkedList (на основе связанного списка).

Примеры создания пустого списка с элементами типа String на основе массива и на основе связанного списка:

List <String> A=new ArrayList <String>();

List <String> B=new LinkedList <String>();

Классы, реализующие интерфейс Set: LinkedHashSet, HashSet, TreeSet.

Пример создания пустых множеств с элементами типа Integer:

Set<Integer> C =new HashSet<Integer>();

Set<Integer> D=new TreeSet<Integer>();

В классах имеются также конструкторы, создающие коллекцию из имеющейся коллекции C, например, следующим образом можно создать набор из списка:

List <тип элементов C> C = new ArrayList <тип элементов C> ();

... // добавление элементов в список C

Set <тип элементов C> D = new TreeSet <тип элементов C>(C);

Алгоритмы.

Алгоритмы для решения типовых задач, в которых данными являются коллекции, содержатся в классе java.util.Collections. Алгоритмы обеспечивают сортировку данных (sort), перемешивание данных (shuffle), поиск элементов (binarySearch), поиск экстремальных элементов (min, max). Имеется также ряд методов для дополнительных действий с коллекциями [1].

2.4.2. Пример создания и манипулирования группой объектов на основе интерфейса List

В качестве примера рассмотрим задачу, которая решалась в ранее выполненных лабораторных работах: организация взаимодействия студентов и студенческих групп. Постановка задачи будет расширена за счет добавления новых возможностей объекта Студенческая группа, реализованного средствами JCF. В частности, в проекте появятся дополнительные методы сортировки.

Код класса Студент усовершенствован (по сравнению с используемым в предыдущих лабораторных работах) с учетом основополагающих принципов философии Java.

1) Согласно принципу инкапсуляции все поля класса сделаны недоступными (private). Для работы с полями (получения и установки значений переменных экземпляра) предусмотрены public-методы (геттеры и сеттеры).

2) С целью оптимизации изменен код метода toString: вместо операции конкатенации значений полей объекта, применение которой на неизменяемом классе String чревато загрязнением памяти «мусорными» объектами, применяется функция форматирования возвращаемой строки при помощи строки формата записи о студенте, которая задана как статическое поле класса.

3) Для задания правила сравнения экземпляров студентов на равенство, в классе Student переопределен метод equals родительского класса Object (класс Object является родителем (суперклассом) для всех других классов Java). Напомним, что поле «Номер зачетной книжки» по смыслу сущности «Студент» является ключевым. Это значит, что для каждого студента это поле должно иметь уникальное значение (номера зачетных книжек не могут повторяться у разных студентов, студента можно четко идентифицировать по номеру зачетной книжки). Переопределенный метод equals возвращает true, если передаваемая в качестве параметра ссылка не равна null, указывает на объект того же класса, что и запускающий объект, номера зачетных книжек у запускающего объекта и объекта-параметра совпадают, а также, если вызывающая ссылка и ссылка-параметр указывают на один и тот же объект.

4) В связи с переопределением метода equals, переопределен метод hashCode класса Object, который возвращает хэш-код объекта. Данный метод переопределяется всякий раз, когда переопределяется метод equals, т.к. существует требование: у равных объектов должны быть равные хэш-коды;

5) для задания естественного порядка перечисления (сортировки) объектов класса Student, реализован метод compareTo() интерфейса Comparable (экземпляры студентов сортируются по возрастанию номера зачетной книжки).

```
public class Student implements Comparable <Student> {
    // Сущность - Студент
    //поля - скрыты в классе
    private final static String STUD_FORMAT_STRING =
        "Студент: %-7d | %-10s | %-8.2f |"; //формат записи о студенте
    private int id; //номер зачетной книжки, ключевое поле
    private String name; //фамилия
    private float ball; //балл на вступительных экзаменах
    // конструктор без параметров
    public Student(){
        id = 0; name = "";
        ball = 0;
    }
}
```

```

// конструктор с параметрами
public Student(int id, String name, float ball){
    this.id=id; this.name=name;
    this.ball=ball;
}
//методы-геттеры
public int getId(){return id;}
public String getName(){return name;}
public float getBall(){return ball;}
//методы-сеттеры
public void setId(int id){this.id = id;}
public void setName(String name){this.name = name;}
public void setBall(float ball){this.ball = ball;}
//Переопределяется метод toString класса Object
 //(возвращает строку описания объекта)
public String toString(){
    return String.format(STUD_FORMAT_STRING,id,name,ball);
}
//Переопределяется метод equals класса Object
 //(задает способ сравнения объектов на равенство,
 //возвращает true, если запускающий объект
 //равен объекту-параметру)
public boolean equals (Object ob){
    if (ob==this) return true; // ссылки равны – один
                                // и тот же объект
    if (ob==null) return false; // в метод передана null-ссылка
    if (getClass()!=ob.getClass())return false; // объекты разных классов
    Student st=(Student)ob; // преобразуем Object в Student
    return id == st.id;      // id – ключевое поле объекта
}
//Переопределяется метод hashCode класса Object
// Возвращает хэш-код объекта
// (у равных объектов должны быть равные hash-коды)
public int hashCode(){
    return 7* (new Integer(id)).hashCode();
}
// Определяем метод CompareTo интерфейса Comparable
// Для определения естественного порядка перечисления элементов
public int compareTo(Student stud){
    if (id < stud.id) return -1;
    else if (id == stud.id) return 0;
    else return 1;
}
}

```

Код класса Группа студентов (**StudGroup**) также изменен с учетом того, что список студентов в нем реализуется на основе инструмента коллекций. Кроме того, в класс добавлены новые возможности поведения (методы) студенческой группы.

Определение полей класса **StudGroup**:

- 1) **private String name;** *//название группы*
 - 2) **private List <Students> students;** *// список (набор) студентов*
- Перечень возможностей (методов) класса **StudGroup**:

конструкторы:

- 1) сконструировать пустую Группу студентов без названия (конструктор без параметров);
- 2) сконструировать Группу студентов с заданным названием и пустым списком студентов;
- 3) сконструировать Группу студентов с заданным названием на основе готового списка студентов;

геттеры для private-полей:

- 4) вернуть название Группы студентов – **getGroupName**;
- 5) вернуть Группу студентов (всех студентов) – **getStudSet**;

сеттер для private-поля:

- 6) изменить название Группы студентов – **setGroupName**;

формирование строки описания объекта Группа студентов:

- 7) **toString**;

запросы на вставку и удаление данных (в теории баз данных – insert, delete):

- 8) добавить студента в Группу студентов – **addStud**;
- 9) удалить студента из Группы студентов – **delStud**;

запросы на выборку данных (в теории баз данных – select):

- 10) вернуть студента с заданным номером зачетки – **getStud**;
- 11) вернуть количество студентов в группе – **getStudNum**;
- 12) вернуть средний балл для группы – **avgBall**;
- 13) вернуть Группу студентов, у которых балл выше среднего для группы – **aboveAvgBall**;
- 14) вернуть Группу студентов, у которых балл находится в заданном диапазоне значений – **betweenBall**;

запросы на сортировку данных:

- 15) вернуть группу студентов, отсортированную в естественном порядке – **sort()**;
- 16) вернуть Группу студентов, отсортированную по одному из нижеперечисленных правил, задаваемых соответствующими объектами, реализующими интерфейс **Comparator**, – **sort (Comparator comp)**:
 - а) по баллу в порядке возрастания – компаратор **CompBallAsc**;
 - б) по баллу в порядке убывания – компаратор **CompBallDesc**;

- в) по фамилии в порядке возрастания, внутри одинаковых фамилий по баллу в порядке убывания – компаратор **CompNameAscBallDesc**;
 17) вывести студентов Группы в окно терминала – **putStudGroup**.

Заметим, что в списках (List) или наборах (Set) хранятся не сами объекты (в данном примере – объекты класса Student), а ссылки на них, поэтому создание новых списков студентов (в методах, возвращающих тип Группа студентов) не увеличивает количество самих объектов класса Students. Другими словами, если мы создаем список на базе другого списка, то в этих списках будут храниться ссылки на одни и те же объекты. Этот принцип был проиллюстрирован рисунками в лабораторных работах №3 и №4.

Структура проекта, реализующего взаимодействие экземпляров сущностей «Студент» и «Студенческая группа» изображена на рисунке 2.1.

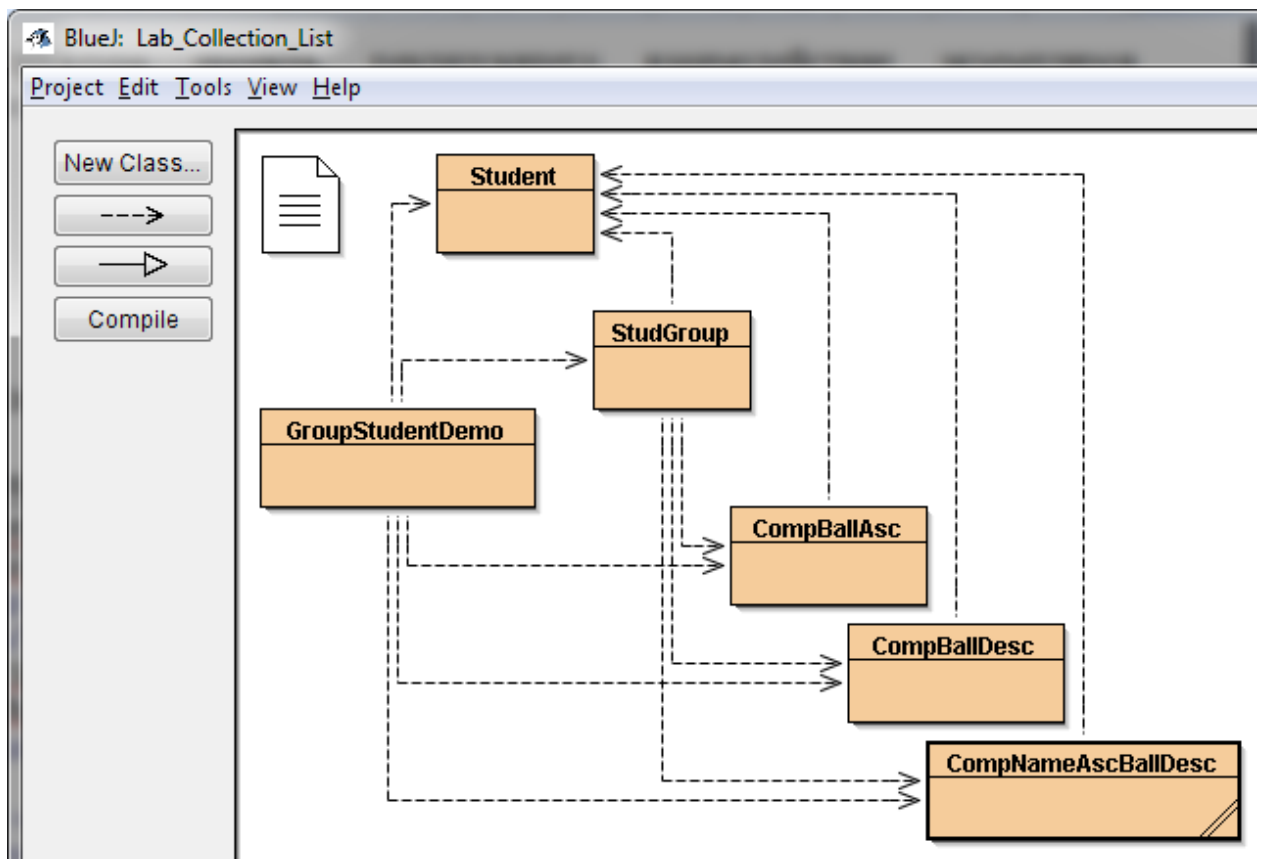


Рисунок 2.1 – Структура проекта, обеспечивающего взаимодействие экземпляров студентов и студенческих групп, использующего концепцию списка (List)

Требуемые постановкой задачи правила сортировки задаются тремя классами, реализующими интерфейс Comparator: **CompBallAsc**, **CompBallDesc**, **CompNameAscBallDesc**.

Класс **StudentGroupDemo** включает в себя метод `main()`, который является стартовой точкой программы. В методе `main()` создаются 10 экземпляров студентов. Значения полей для студентов передаются через параметр `String[] args` (массив строк) метода `main()` при его запуске, как это делалось в лабораторных работах №2, №3 и №4. Для удобства соответствующие данные можно подготовить в текстовом редакторе и сохранить в файле, чтобы потом скопировать их в окно запуска метода `main()`.

Кроме экземпляров студентов, в методе `main()` создается два экземпляра класса `Group` (две студенческих группы).

Затем делается попытка добавить в группу `gr1` студента с уже существующим в группе номером зачетной книжки. Такая попытка заканчивается неудачей, что мы и наблюдаем при выводе соответствующих списков.

Для каждой группы выводится полный список студентов, средний балл по группе, список студентов, у которых балл выше среднего, список студентов, у которых балл находится в заданном диапазоне. Все выводимые списки заданным образом упорядочиваются.

Из первой группы исключается студент, после чего (для контроля) выводится еще раз список группы.

Осуществляются проверки на наличие в группах студентов с заданными номерами зачетных книжек и выдаются соответствующие сообщения.

Текст этих методов приведен ниже.

```
import java.util.*;
public class CompBallAsc implements Comparator{
    public int compare (Object ob1, Object ob2){
        Student stud1 = (Student) ob1;
        Student stud2 = (Student) ob2;
        if (stud1.getBall() < stud2.getBall()) return -1;
        else if (stud1.getBall() == stud2.getBall()) return 0;
        else return 1;
    }
    //нет необходимости переопределять метод equals
}
```

```
import java.util.*;
public class CompBallDesc implements Comparator{
    public int compare (Object ob1, Object ob2){
        Student stud1 = (Student) ob1;
        Student stud2 = (Student) ob2;
        if (stud1.getBall() < stud2.getBall()) return 1;
        else if (stud1.getBall() == stud2.getBall()) return 0;
        else return -1;
    }
    //нет необходимости переопределять метод equals
}
```

```

import java.util.*;
public class CompNameAscBallDesc implements Comparator{
    public int compare (Object ob1, Object ob2){
        Student stud1 = (Student) ob1;
        Student stud2 = (Student) ob2;
        if (stud1.getName().compareTo(stud2.getName())<0) return -1;
        else if (stud1.getName().compareTo(stud2.getName())> 0) return 1;
        //фамилии одинаковые
        else if (stud1.getBall() < stud2.getBall()) return 1;
        else if (stud1.getBall() == stud2.getBall()) return 0;
        else return -1;
    }
    //нет необходимости переопределять метод equals
}

import java.util.*;
public class StudGroup { // Сущность – Группа студентов
    //поля (скрыты в классе)
    private final static String GROUP_FORMAT_STRING =
        "Группа: %-s, %-5d студентов"; //формат записи о группе
    private String name; //название группы
    private List <Student> students; // список (набор) студентов
    // конструкторы
    public StudGroup() {
        name=""; //без названия
        students = new ArrayList <Student>(); //создается пустой список
    }
    public StudGroup(String name){
        this.name=name; //задается название группы
        students = new ArrayList <Student>(); //создается пустой список
    }
    public StudGroup(String name, List list){
        this.name=name; //задается название группы
        students = new ArrayList <Student>(list); //создается на основе
        // существующего списка
    }
    //метод-сеттер для private-поля
    public void setGroupName(String name)
        {this.name = name;}
    //методы-геттеры для private-полей
    public String getGroupName(){return name;}
    public List <Student> getStudents(){
        return students;
    }
    //Переопределяем метод toString класса Object
    //возвращает строку описания объекта Группа студентов:
    public String toString(){
        return String.format(GROUP_FORMAT_STRING,name,getStudNum());
    }
    //Запросы на вставку, удаление, изменение данных:
    public boolean addStud(Student stud){

```

```

        //Добавить студента в список группы
        //(студента нельзя добавить, если в группе уже есть студент
        //таким же id)
        if (getStud(stud.getId())!=null) return false; //дополнительная
                                                    // программная проверка
        if (students.add(stud)) return true;
        else return false;
    }
    public boolean delStud(int id){
        //Удалить студента из списка группы
        if (students.remove(new Student(id,"",0))) return true;
        else return false;
    }
    // запросы на выборку данных:
    public Student getStud (int id){
        // Возвращает студента с заданным номером зачетки (id):
        for (Student stud:students)
            if (stud.getId() == id) return stud; // если студент найден
        return null; // студент не найден
    }
    public int getStudNum(){
        //Возвращает число студентов в группе
        return students.size();
    }
    public float avgBall(){
        //Возвращает средний балл по группе
        int num=students.size();
        if (num==0) return 0;
        float avg=0;
        for (Student stud:students)
            avg=avg+stud.getBall();
        return avg/num;
    }

    public StudGroup aboveAvgBall(){
        //Вернуть студентов, у которых балл выше среднего
        float avg=avgBall();
        StudGroup group = new StudGroup (name +
            ": студенты, у которых балл выше среднего - " + avg);
        // конкатенация на неизменяемых строках - плохое решение,
        // покажем, как избавиться от нее, в методе between;
        //для просмотра (перечисления) объектов списка
        //используем цикл for-each
        for (Student stud:students)
            if (stud.getBall()>avg)group.addStud(stud);
        return group;
    }

    public StudGroup betweenBall(float b1, float b2){
        //Вернуть студентов с баллом в диапазоне [b1,b2]
        //Избавляемся от конкатенации при формировании имени группы
        StudGroup group = new StudGroup (

```

```

        String.format (
            "%s: студенты, у которых балл в диапазоне от %.2f до %.2f",
            name,b1,b2));
//для просмотра (перечисления) объектов списка
//используем итератор
Iterator <Student> iter = students.iterator();
while (iter.hasNext()){
    Student stud = iter.next();
    if ((stud.getBall() >= b1)&&(stud.getBall() <= b2))group.addStud(stud);
}
return group;
}
//Запросы на сортировку данных
//Для сортировки списков List используются статические
//методы sort, определенные в классе Collections
public StudGroup sort(){
    //Сортировка студентов в естественном порядке
    //Естественный порядок задает метод CompareTo,
    //переопределенный в классе Student
    StudGroup group = new StudGroup (name, students);
    Collections.sort(group.students);
    return group;
}

public StudGroup sort(Comparator comp){ //сортировка студентов
    //по правилу, задаваемому компаратором comp
    StudGroup group = new StudGroup (name, students);
    Collections.sort(group.students, comp);
    return group;
}
// Запрос на вывод данных
public void putStudGroup(){
    // Вывод Группы студентов в окно терминала
    System.out.println(name); //имя группы
    System.out.printf("%5s%17s%11s%17s\n",
        "Номер","Номер зач. кн. ","ФИО","Балл"); //заголовки столбцов
    int i=1;
    for (Student stud:students){
        System.out.printf(" %-7d %-15d %-16s %-10.2f\n",
            i, stud.getId(), stud.getName(), stud.getBall());
        i=i+1;
    }
} // putStudGroup
} //class

public class GroupStudentDemo{ // Взаимодействие студентов и групп
    public static void main (String[ ] args){
        //Создаем студенческие группы
        StudGroup gr1=new StudGroup ("ИВТБ-11д");
        StudGroup gr2=new StudGroup ("ИВТБ-12д");
        // в массив args при запуске метода main (String[ ] args)
        // ввели 10 студентов: по 5 для каждой группы.
    }
}

```

```

int j=0; //индекс массива args
// добавление студентов в первую группу
for (int i=0; i<5; i++){
    int id=Integer.valueOf(args[j]); j++; // получен int-эквивалент из String
    String name=args[j]; j++;
    //получаем float-эквивалент из String
    float ball=Float.valueOf(args[j]); j++;
    //добавить студента
    gr1.addStud(new Student(id, name, ball));
}

// добавление студентов во вторую группу
for (int i=0; i<5; i++){
    int id=Integer.valueOf(args[j]); j++;
    String name=args[j]; j++;
    float ball=Float.valueOf(args[j]); j++;
    //добавить студента
    gr2.addStud(new Student(id, name, ball));
}

//Выборка данных
// Выводим всевозможные списки студентов
// Выводим основной список без сортировки
System.out.println("Группа студентов (без сортировки)");
    gr1.putStudGroup();
    // пытаемся вставить студента с тем же номером зачетки
    System.out.println("Попытка добавить студента: 32348, Козлов,5.0");
    gr1.addStud(new Student (32348, "Козлов",5.0f));
    // снова выводим основной список
// (теперь с естественным порядком сортировки)
    System.out.println(
        "Группа студентов (с естественным порядком сортировки)");
    gr1.sort().putStudGroup();
    // другие списки:
    System.out.println(
        "Группа студентов (с сортировкой по возрастанию балла)");
    gr1.aboveAvgBall().sort(new CompBallAsc()).putStudGroup();
    System.out.println(

```

```

        "Группа студентов (с сортировкой по убыванию балла)");
gr1.betweenBall(3.2f,4.2f).sort(new CompBallDesc()).putStudGroup();
// удаление студента по id
gr1.delStud(12345);
//выводим список группы gr1 после удаления студента
System.out.println ("После удаления студента с id=12345:");
System.out.println(
        "Группа студентов (с естественным порядком сортировки)");
gr1.putStudGroup();
// Выводим списки студентов для второй группы
System.out.println(
        "Группа студентов (с сортировкой по возрастанию имени и");
System.out.println(" убыванию балла)");
gr2.sort(new CompNameAscBallDesc()).putStudGroup();
// другие списки для второй группы:
System.out.println(
        "Группа студентов (с сортировкой по возрастанию балла)");
gr2.aboveAvgBall().sort(new CompBallAsc()).putStudGroup();
System.out.println(
        "Группа студентов (с сортировкой по убыванию балла)");
gr2.betweenBall(3.2f,4.2f).sort(new CompBallDesc()).putStudGroup();
//Проверяем,есть ли во первой группе студент
// с заданным номером зачетки
int n=23345;
Student s1=gr1.getStud(n);
if (s1==null) System.out.println ("В группе "+gr1.getGroupName()+
        " нет студента с номером зачетки "+n);
else System.out.println(s1);
n=70000;
s1=gr1.getStud(n);
if (s1==null) System.out.println ("В группе "+gr1.getGroupName()+
        " нет студента с номером зачетки "+n);
else System.out.println(s1);
//Проверяем,есть ли во второй группе студент с заданным
// номером зачетки

```

```

n=53349;
s1=gr2.getStud(n);
if (s1==null) System.out.println ("В группе "+gr1.getGroupName()+
    " нет студента с номером зачетки "+n);
else System.out.println(s1);
}}

```

Результат работы программы представлен на рисунках 2.2, 2.3. Для того, чтобы наблюдать все выводимые данные, включайте в окне терминала опции «Очищать окно терминала перед вызовом метода» и «Неограниченная буферизация».

```

Options
Группа студентов (без сортировки)
ИВТБ-11д


| Номер | Номер зач. кн. | ФИО         | Балл |
|-------|----------------|-------------|------|
| 1     | 23345          | Иванов      | 4,50 |
| 2     | 23346          | Петров      | 4,20 |
| 3     | 12345          | Александров | 3,80 |
| 4     | 23347          | Орлов       | 5,00 |
| 5     | 32348          | Воробьев    | 3,40 |


Попытка добавить студента: 32348, Козлов, 5.0
Группа студентов (с естественным порядком сортировки)
ИВТБ-11д


| Номер | Номер зач. кн. | ФИО         | Балл |
|-------|----------------|-------------|------|
| 1     | 12345          | Александров | 3,80 |
| 2     | 23345          | Иванов      | 4,50 |
| 3     | 23346          | Петров      | 4,20 |
| 4     | 23347          | Орлов       | 5,00 |
| 5     | 32348          | Воробьев    | 3,40 |


Группа студентов (с сортировкой по возрастанию балла)
ИВТБ-11д: студенты, у которых балл выше среднего - 4.18


| Номер | Номер зач. кн. | ФИО    | Балл |
|-------|----------------|--------|------|
| 1     | 23346          | Петров | 4,20 |
| 2     | 23345          | Иванов | 4,50 |
| 3     | 23347          | Орлов  | 5,00 |


Группа студентов (с сортировкой по убыванию балла)
ИВТБ-11д: студенты, у которых балл в диапазоне от 3.2 до 4.2


| Номер | Номер зач. кн. | ФИО         | Балл |
|-------|----------------|-------------|------|
| 1     | 23346          | Петров      | 4,20 |
| 2     | 12345          | Александров | 3,80 |
| 3     | 32348          | Воробьев    | 3,40 |


```

Рисунок 2.2 – Результаты работы программы, использующей List

```

После удаления студента с id=12345:
Группа студентов (с естественным порядком сортировки)
ИБТБ-11д


| Номер | Номер зач. кн. | ФИО      | Балл |
|-------|----------------|----------|------|
| 1     | 23345          | Иванов   | 4,50 |
| 2     | 23346          | Петров   | 4,20 |
| 3     | 23347          | Орлов    | 5,00 |
| 4     | 32348          | Воробьев | 3,40 |


Группа студентов (с сортировкой по возрастанию имени и убыванию балла)
ИБТБ-12д


| Номер | Номер зач. кн. | ФИО       | Балл |
|-------|----------------|-----------|------|
| 1     | 53347          | Афанасьев | 3,10 |
| 2     | 53346          | Ливанов   | 4,50 |
| 3     | 53348          | Морев     | 5,00 |
| 4     | 53345          | Федоров   | 4,10 |
| 5     | 53349          | Федоров   | 3,40 |


Группа студентов (с сортировкой по возрастанию балла)
ИБТБ-12д: студенты, у которых баллы выше среднего - 4.02


| Номер | Номер зач. кн. | ФИО     | Балл |
|-------|----------------|---------|------|
| 1     | 53345          | Федоров | 4,10 |
| 2     | 53346          | Ливанов | 4,50 |
| 3     | 53348          | Морев   | 5,00 |


Группа студентов (с сортировкой по убыванию балла)
ИБТБ-12д: студенты, у которых балл в диапазоне от 3.2 до 4.2


| Номер | Номер зач. кн. | ФИО     | Балл |
|-------|----------------|---------|------|
| 1     | 53345          | Федоров | 4,10 |
| 2     | 53349          | Федоров | 3,40 |


Студент: 23345 | Иванов | 4,50 |
В группе ИБТБ-11д нет студента с номером зачетки 70000
Студент: 53349 | Федоров | 3,40 |

```

Рисунок 2.3 – Результаты работы программы, использующей List (продолжение)

2.4.3. Пример создания и манипулирования группой объектов на основе интерфейса Set

Структура проекта изображена на рисунке 2.4. Данный проект состоит из тех же классов, что и предыдущий. После тщательного анализа текстов программ, объясните отсутствие в структуре, приведенной на рисунке 2.4, некоторых связей между классами, присутствующих на рисунке 2.1.

Учитывая, что каждый студент имеет уникальный номер зачетной книжки (ключевое свойство), а, значит, двух одинаковых экземпляров класса

Student быть не может, в данном проекте Группа студентов реализована при помощи упорядоченного набора объектов (интерфейс Set, реализующий класс TreeSet). Как уже говорилось выше, множество (набор) не допускает дублирования в нем объектов. Проверка уникальности объекта осуществляется автоматически при его добавлении в набор методом add интерфейса Set. При этом объект добавляется только в том случае, если он не совпадает ни с одним из объектов, уже присутствующих в наборе. Правило сравнения объектов на равенство задает метод equals, переопределенный в классе Student.

Естественный порядок перечисления объектов в упорядоченном наборе TreeSet задает метод CompareTo, реализованный в классе Student. Все методы класса StudGroup, являющиеся запросами на сортировку данных в другом порядке, используют компараторы, задающие правила сортировки в возвращаемых наборах. Компараторы передаются в конструкторы в качестве параметров при создании этих наборов.

Ниже приводятся тексты классов **StudGroup** и **GroupStudentDemo**. Тексты классов, задающих **компараторы**, и класса **Student** скопированы в данный проект из предыдущего проекта без изменений.

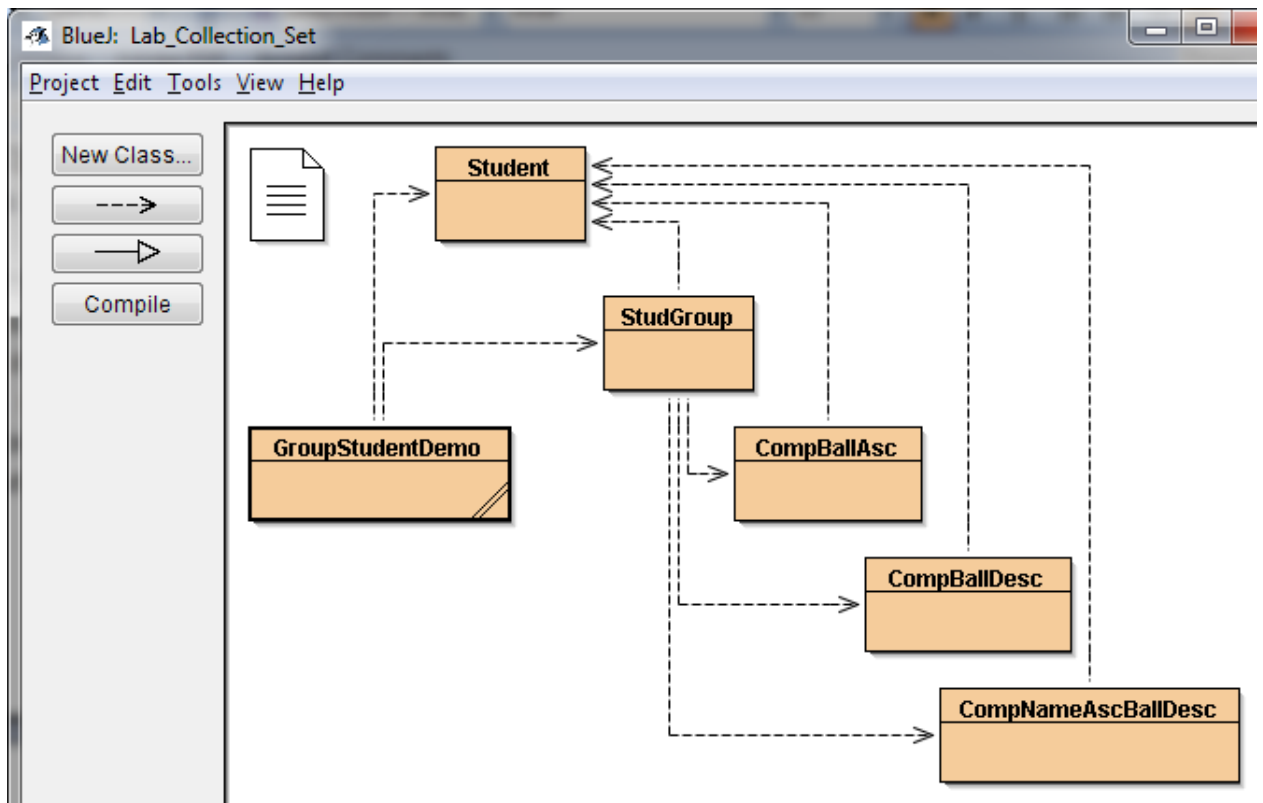


Рисунок 2.4 – Структура проекта, обеспечивающего взаимодействие экземпляров студентов и студенческих групп, использующего концепцию множества (Set)

Текст программы.

```
import java.util.*;
public class StudGroup { // Сущность - Студенческая группа
    //поля
    private final static String GROUP_FORMAT_STRING =
        "Группа: %-s, %-5d студентов"; //формат записи о группе
    private String name; //название группы
    private Set <Student> students; // список (набор) студентов
    // конструкторы
    public StudGroup(){
        name=""; //без названия
        students = new TreeSet <Student>(); //создается пустой набор
    }
    public StudGroup(String name){
        this.name=name; //задается название группы
        students = new TreeSet <Student>(); //создается пустой набор
    }
    public StudGroup(String name, Comparator comp){
        this.name=name; //задается название группы
        students = new TreeSet <Student>(comp); //создается пустой набор,
        // в котором ссылки на добавляемые объекты будут
        // упорядочиваться по правилам компаратора comp
    }
    public StudGroup(String name, Set set){
        this.name=name; //задается название группы
        students = new TreeSet <Student>(set); //создается набор на основе
        // существующего набора set
    }
    //метод-сеттер для private-поля
    public void setGroupName(String name)
    {this.name=name;}
    //методы-геттеры для private-полей
    public String getGroupName(){return name;}
    public Set <Student> getStudents(){
        return students;
    }
    //Переопределяем метод toString класса Object
    //возвращает строку описания объекта Группа студентов:
    public String toString(){
        return String.format(GROUP_FORMAT_STRING,name,getStudNum());
    }
    //Запросы на вставку и удаление данных:
    public boolean addStud(Student stud){
        //Добавить студента в список группы
        //студент не будет вставлен, если в наборе уже есть студент
        //таким же id
        if (students.add(stud)) return true;
        else return false;
    }
}
```

```

public boolean delStud(int id){
    //Удалить студента из списка группы
    if (students.remove(new Student(id,"",0))) return true;
    else return false;
}
// Запросы на выборку данных:
public Student getStud (int id){
    // возвращает студента с заданным номером зачетки (id):
    for (Student stud:students)
        if (stud.getId() == id) return stud; // если студент найден
    return null; // студент не найден
}
public int getStudNum(){
    //Возвращает число студентов в группе
    return students.size();
}
public float avgBall(){
    //Возвращает средний балл по группе
    int num=students.size();
    if (num==0) return 0;
    float avg=0;
    for (Student stud:students)
        avg=avg+stud.getBall();
    return avg/num;
}

public StudGroup aboveAvgBall(){
    //Возвращает студентов, у которых балл выше среднего
    float avg=avgBall();
    StudGroup group = new StudGroup (name+
        ": Студенты, у которых балл выше среднего - " + avg);
    // конкатенация на неизменяемых строках - плохое решение,
    // покажем, как избавиться от нее, в методе between
    //для просмотра (перечисления) объектов списка
    //используем цикл for-each
    for (Student stud:students)
        if (stud.getBall()>avg) group.addStud(stud);
    return group;
}

public StudGroup betweenBall(float b1, float b2){// запрос:
    //Возвращает студентов с баллом в диапазоне [b1,b2]
    //Избавляемся от конкатенации при формировании имени группы
    StudGroup group = new StudGroup (
        String.format (
            "%s: студенты, у которых балл в диапазоне от %4.2f до %4.2f",
            name,b1,b2));
    //для просмотра (перечисления) объектов списка
    // используем итератор
    Iterator <Student> iter=students.iterator();
    while (iter.hasNext()){
        Student stud=iter.next();
    }
}

```

```

        if ((stud.getBall()>=b1)&&(stud.getBall()<=b2))group.addStud(stud);
    }
    return group;
}
//запросы на сортировку данных
public StudGroup sortBallAsc(){
    StudGroup group = new StudGroup(name+
        " (сортировка по возрастанию балла)", new CompBallAsc());
    for (Student stud:students) group.addStud(stud); // при добавлении
    // в набор производится упорядочивание ссылок на объекты по
    // правилу, задаваемому компаратором (сортировка)
    return group;
}
public StudGroup sortBallDesc(){
    StudGroup group = new StudGroup(name+
        " (сортировка по убыванию балла)", new CompBallDesc());
    for (Student stud:students) group.addStud(stud); // при добавлении
    // в набор производится упорядочивание ссылок на объекты по
    // правилу, задаваемому компаратором (сортировка)
    return group;
}
public StudGroup sortNameAscBallDesc (){
    StudGroup group = new StudGroup(name+
        " (сортировка по возрастанию имени и убыванию балла)",
        new CompNameAscBallDesc());
    for (Student stud:students) group.addStud(stud); // при добавлении
    // в набор производится упорядочивание ссылок на объекты по
    // правилу, задаваемому компаратором (сортировка)
    return group;
}
// Запрос на вывод данных
public void putStudGroup(){
    //Вывод Группы студентов в окно терминала
    System.out.println(name); //имя группы
    System.out.printf("%5s%17s%11s%17s\n",
        "Номер","Номер зач. кн.,","ФИО","Балл"); //заголовки столбцов
    int i=1;
    for (Student stud:students){
        System.out.printf(" %-7d %-15d %-16s %-10.2f\n",
            i, stud.getId(), stud.getName(), stud.getBall());
        i=i+1;
    }
} // putStudGroup
} //class

public class GroupStudentDemo{ // Взаимодействие студентов и групп
    public static void main (String[ ] args){
        //Создаем студенческие группы
        StudGroup gr1=new StudGroup ("ИВТБ-11д");
        StudGroup gr2=new StudGroup ("ИВТБ-12д");
        // в массив args при запуске метода main (String[] args)
        // ввели 10 студентов: по 5 для каждой группы.

```

```

int j=0; //индекс массива args
// добавление студентов в первую группу
for (int i=0; i<5; i++){
    int id=Integer.valueOf(args[j]); j++; // получен int-эквивалент из String
    String name=args[j]; j++;
    float ball=Float.valueOf(args[j]); j++; //получен float-эквивалент из String
    //добавить студента
    gr1.addStud(new Student(id, name, ball));
}
// добавление студентов во вторую группу
for (int i=0; i<5; i++){
    int id=Integer.valueOf(args[j]); j++;
    String name=args[j]; j++;
    float ball=Float.valueOf(args[j]); j++;
    //добавить студента
    gr2.addStud(new Student(id, name, ball));
}
//Выборка данных
// Выводим всевозможные списки студентов
// Выводим основной список без сортировки
gr1.putStudGroup();
// пытаемся вставить студента с тем же номером зачетки
System.out.println("Попытка добавить студента: 32348, Козлов,5.0");
gr1.addStud(new Student (32348, "Козлов",5.0f));
// снова выводим основной список без сортировки
gr1.putStudGroup();
// другие списки:
gr1.aboveAvgBall().sortBallAsc().putStudGroup();
gr1.betweenBall(3.2f,4.2f).sortBallDesc().putStudGroup();
// удаление студента по id
gr1.delStud(12345);
//выводим список группы gr1 после удаления студента
System.out.println ("После удаления студента с id=12345:");
gr1.putStudGroup();

// выводим списки студентов для второй группы
// основной список группы
gr2.putStudGroup(); // без сортировки
gr2.sortNameAscBallDesc().putStudGroup(); // с сортировкой по
                                           // возрастанию фамилии и убыванию балла
// другие списки:
gr2.aboveAvgBall().sortBallAsc().putStudGroup();
gr2.betweenBall(3.2f,4.2f).sortBallDesc().putStudGroup();

//Проверяем,есть ли во первой группе студент
// с заданным номером зачетки
int n=23345;
Student s1=gr1.getStud(n);
if (s1==null) System.out.println ("В группе "+gr1.getGroupName()+
    " нет студента с номером зачетки "+n);
else System.out.println(s1);
n=70000;

```

```

s1=gr1.getStud(n);
if (s1==null) System.out.println ("В группе "+gr1.getGroupName()+
    " нет студента с номером зачетки "+n);
else System.out.println(s1);
//Проверяем, есть ли во второй группе студент
// с заданным номером зачетки
n=53349;
s1=gr2.getStud(n);
if (s1==null) System.out.println ("В группе "+gr1.getGroupName()+
    " нет студента с номером зачетки "+n);
else System.out.println(s1);

}}

```

Результаты работы программы представлены на рисунках 2.5, 2.6.

```

Options
ИВТБ-11д
Номер  Номер зач. кн.      ФИО      Балл
1       12345      Александров  3,80
2       23345      Иванов      4,50
3       23346      Петров      4,20
4       23347      Орлов       5,00
5       32348      Воробьев    3,40
Попытка добавить студента: 32348, Козлов,5.0
ИВТБ-11д
Номер  Номер зач. кн.      ФИО      Балл
1       12345      Александров  3,80
2       23345      Иванов      4,50
3       23346      Петров      4,20
4       23347      Орлов       5,00
5       32348      Воробьев    3,40
ИВТБ-11д: Студенты, у которых балл выше среднего - 4.18 (сортировка по возрастанию балла)
Номер  Номер зач. кн.      ФИО      Балл
1       23346      Петров      4,20
2       23345      Иванов      4,50
3       23347      Орлов       5,00
ИВТБ-11д: студенты, у которых балл в диапазоне от 3,20 до 4,20 (сортировка по убыванию балла)
Номер  Номер зач. кн.      ФИО      Балл
1       23346      Петров      4,20
2       12345      Александров  3,80
3       32348      Воробьев    3,40
После удаления студента с id=12345:
ИВТБ-11д
Номер  Номер зач. кн.      ФИО      Балл
1       23345      Иванов      4,50
2       23346      Петров      4,20
3       23347      Орлов       5,00
4       32348      Воробьев    3,40

```

Рисунок 2.5 – Результаты работы программы, использующей Set

ИВТЕ-12д			
Номер	Номер зач. кн.	ФИО	Балл
1	53345	Федоров	4,10
2	53346	Ливанов	4,50
3	53347	Афанасьев	3,10
4	53348	Морев	5,00
5	53349	Федоров	3,40
ИВТЕ-12д (сортировка по возрастанию имени и убыванию балла)			
Номер	Номер зач. кн.	ФИО	Балл
1	53347	Афанасьев	3,10
2	53346	Ливанов	4,50
3	53348	Морев	5,00
4	53345	Федоров	4,10
5	53349	Федоров	3,40
ИВТЕ-12д: Студенты, у которых балл выше среднего – 4.02 (сортировка по возрастанию балла)			
Номер	Номер зач. кн.	ФИО	Балл
1	53345	Федоров	4,10
2	53346	Ливанов	4,50
3	53348	Морев	5,00
ИВТЕ-12д: студенты, у которых балл в диапазоне от 3,20 до 4,20 (сортировка по убыванию балла)			
Номер	Номер зач. кн.	ФИО	Балл
1	53345	Федоров	4,10
2	53349	Федоров	3,40
Студент: 23345 Иванов 4,50			
В группе ИВТЕ-11д нет студента с номером зачетки 70000			
Студент: 53349 Федоров 3,40			

Рисунок 2.6 – Результаты работы программы, использующей Set (продолжение)

2.5. Порядок выполнения работы

- 1) Внимательно прочитайте п. 1.4 и разберите тексты программ-примеров.
- 2) По аналогии с примером (п. 1.4.2) разработайте программу, реализующую группу объектов на основе интерфейса List для заданной вариантной предметной области.
- 3) По аналогии с примером (п. 1.4.3) разработаете программу, реализующую группу объектов на основе интерфейса Set для заданной вариантной предметной области.
- 4) Избавьтесь от применения в программе операции конкатенации на неизменяемых строках, вместо конкатенации для сборки строки из частей применяйте функцию String.format().
- 4) Разработайте тексты дополнительных методов, предусмотренных вариантной заданием, и добавьте их в соответствующие программы. Для работы со списками и множествами максимально используйте методы соответствующих интерфейсов.

5) Проведите отладку программ и проверку их работы на тестовых примерах.

6) Сравните две разработанные программы, проанализируйте различия в программах, вызванные различиями в способах организации группы объектов. Оцените, какие достоинства и недостатки, на ваш взгляд, имеют разработанные реализации групп объектов. Результаты анализа отразите в отчете.

2.6. Варианты заданий

Варианты заданий представлены в таблицах 2.1 и 2.2. Вариант студенту выдается преподавателем.

Таблица 1.1 – Варианты заданий (предметные области)

№	Сущность 1 (класс1) Все поля – public	Сущность 2 (класс 2) Все поля – private
1	2	3
1	Автомобиль <u>Поля (private):</u> 1) String номер_автомобиля (ключевое), 2) String марка_автомобиля, 3) String тип_топлива, 4) double пробег_в_км, 5) double стоимость_часа, 6) int максимальное_число_часов_проката. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Прокатная_фирма (выдает автомобили напрокат) <u>Поля (private):</u> 1) String название_фирмы; 2) Автомобиль [] список_автомобилей, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий автомобиль с заданным значением ключевого поля из списка фирмы, 4) геттер, возвращающий число автомобилей в списке фирмы, 5) сеттер для поля название_фирмы, 6) добавить автомобиль в список с учетом уникальности его id, 7) удалить автомобиль с заданным id, 8) вернуть среднюю стоимость_часа проката автомобиля для салона, 9) вернуть список автомобилей, стоимость_часа проката которых ниже средней, 10) вернуть список автомобилей, у

1	2	3
		которых пробег находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id_автомобиля), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список автомобилей в окно терминала, 13) toString() (переопределение).
2	Офис. <u>Поля (private):</u> 1) int id_офиса (ключевое), 2) String адрес, 3) double площадь, 4) int число_комнат. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Фирма. <u>Поля (private):</u> 1) String название_фирмы, 2) Офис [] список_офисов. <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий офис с заданным значением ключевого поля из списка офисов, 4) геттер, возвращающий число офисов в списке фирмы, 5) сеттер для поля название_фирмы, 6) добавить офис с учетом уникальности id_офиса, 7) удалить офис с заданным id_офиса, 8) вернуть среднюю площадь для списка офисов, 9) вернуть список офисов, в которых площадь превышает среднюю, 10) вернуть список офисов, у которых площадь находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).

1	2	3
3	Сотрудник. <u>Поля (private):</u> 1) int id_сотрудника (ключевое), 2) String фамилия, 3) int год_рождения, 4) double зарплата. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Отдел. <u>Поля (private):</u> 1) String название_отдела, 2) Сотрудник [] список_сотрудников. <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий сотрудника с заданным значением ключевого поля из списка сотрудников, 4) геттер, возвращающий число сотрудников в списке отдела, 5) сеттер для поля название_отдела, 6) добавить сотрудника с учетом уникальности id_сотрудника, 7) удалить сотрудника с заданным id_сотрудника, 8) вернуть среднюю зарплату сотрудников по отделу, 9) вернуть список сотрудников, у которых зарплата ниже средней, 10) вернуть список сотрудников, у которых зарплата находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).
4	Автомобиль. <u>Поля (private):</u> 1) int id_автомобиля (ключевое), 2) String марка, 3) int год_выпуска, 4) double стоимость. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение),	Автопарк. <u>Поля (private):</u> 1) String название_автопарка, 2) Автомобиль [] список_автомобилей, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий автомобиль с заданным значением ключевого поля из списка автомобилей,

1	2	3
	4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	4) геттер, возвращающий число автомобилей в списке автопарка, 5) сеттер для поля название_автопарка, 6) добавить автомобиль с учетом уникальности его id_автомобиля, 7) удалить автомобиль с заданным id_автомобиля, 8) вернуть среднюю стоимость автомобиля в автопарке, 9) вернуть список автомобилей, стоимость которых выше средней, 10) вернуть список автомобилей, стоимость которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).
5	Товар. <u>Поля (private):</u> 1) int id_товара (ключевое), 2) String название, 3) String производитель, 4) double цена. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Склад: <u>Поля (private):</u> 1) int номер_склада, 2) Товар [] список_товаров, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий товар с заданным значением ключевого поля из списка товаров, 4) геттер, возвращающий число товаров в списке склада, 5) сеттер для поля номер_склада, 6) добавить товар с учетом уникальности id_товара, 7) удалить товар с заданным id_товара, 8) вернуть среднюю цену товара на складе, 9) вернуть список товаров, цена которых ниже средней,

1	2	3
		10) вернуть список товаров, цена которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).
6	Инженер. <u>Поля (private):</u> 1) int id_инженера (ключевое), 2) String фамилия, 3) int стаж, 4) double оклад. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Выч_центр. <u>Поля (private):</u> 1) int номер_выч_центра, 2) Инженер [] список_инженеров, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий инженера с заданным значением ключевого поля из списка инженеров, 4) геттер, возвращающий число инженеров в списке вычислительного центра, 5) сеттер для поля номер_выч_центра, 6) добавить инженера с учетом уникальности id_инженера, 7) удалить инженера с заданным id_инженера, 8) вернуть средний оклад инженера, 9) вернуть список инженеров, оклад которых выше среднего, 10) вернуть список инженеров, оклад которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно

1	2	3
		терминала, 13) toString() (переопределение).
7	Специальность. <u>Поля (private):</u> 1) int id_специальности (ключевое), 2) String название, 3) int срок обучения, 4) double проходной_балл. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Университет. <u>Поля (private):</u> 1) String название_университета, 2) Специальность [] список_специальностей, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий специальность с заданным значением ключевого поля из списка специальностей, 4) геттер, возвращающий число специальностей в списке университета, 5) сеттер для поля название_университета, 6) добавить специальность с учетом уникальности id_специальности, 7) удалить специальность с заданным id_специальности, 8) вернуть средний проходной балл для университета, 9) вернуть список специальностей, проходной балл на которые выше среднего, 10) вернуть список специальностей, проходной балл на которые находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).
8	Дисциплина. <u>Поля (private):</u> 1) int id_дисциплины (ключевое),	Кафедра. <u>Поля (private):</u> 1) String название_кафедры,

1	2	3
	2) String название, 3) int номер семестра, 4) float число_кредитов. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация). <u>Примечание.</u> Число кредитов не может быть меньше 3.0 и больше 7.0	2) Дисциплина [] список_дисциплин, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий дисциплину с заданным значением ключевого поля из списка дисциплин, 4) геттер, возвращающий число дисциплин в списке кафедры, 5) сеттер для поля название_кафедры, 6) добавить дисциплину с учетом уникальности id_дисциплины, 7) удалить дисциплину с заданным id_дисциплины, 8) вернуть среднее число кредитов для списка дисциплин кафедры, 9) вернуть список дисциплин, число кредитов для которых выше среднего, 10) вернуть список дисциплин, число кредитов для которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).
9	Преподаватель <u>Поля (private):</u> 1) int id_преподавателя (ключевое), 2) String фамилия, 3) String должность, 4) double размер_ставки. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Кафедра. <u>Поля (private):</u> 1) String название_кафедры, 2) Преподаватель [] список_преподавателей, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий преподавателя с заданным значением ключевого поля из списка преподавателей, 4) геттер, возвращающий число

1	2	3
	<u>Примечание.</u> Преподаватель не может работать менее, чем на 0.25 ставки и более, чем на 1.75 ставки.	преподавателей в списке кафедры, 5) сеттер для поля название_кафедры, 6) добавить преподавателя с учетом уникальности id_преподавателя, 7) удалить преподавателя с заданным id_преподавателя, 8) вернуть средний размер ставки преподавателя на кафедре, 9) вернуть список преподавателей, размер ставки у которых выше среднего, 10) вернуть список преподавателей, размер ставки которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).
10	Книга <u>Поля (private):</u> 1) int id_книги (ключевое), 2) String автор, 3) String название_книги, 4) int число страниц, 5) double цена. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Каталог. <u>Поля (private):</u> 1) String название_каталога, 2) Книга [] список_книг, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий книгу с заданным значением ключевого поля из списка книг, 4) геттер, возвращающий число книг в списке каталога, 5) сеттер для поля название_каталога, 6) добавить книгу с учетом уникальности id_книги, 7) удалить книгу с заданным id_книги, 8) вернуть среднюю цену книги для каталога, 9) вернуть список книг, цена которых выше средней,

1	2	3
		10) вернуть список книг, цена которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).
11	Спортсмен. <u>Поля (private):</u> 1) int id_спортсмена (ключевое), 2) String фамилия, 3) String роль, 4) double зарплата. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Команда. <u>Поля (private):</u> 1) String название_команды, 2) Спортсмен [] список_спортсменов, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий спортсмена с заданным значением ключевого поля из списка спортсменов, 4) геттер, возвращающий число спортсменов в списке команды, 5) сеттер для поля название_команды, 6) добавить спортсмена с учетом уникальности id_спортсмена, 7) удалить спортсмена с заданным id_спортсмена, 8) вернуть среднюю зарплату спортсмена в команде, 9) вернуть список спортсменов, зарплата которых ниже средней, 10) вернуть список спортсменов, зарплата которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно

1	2	3
		терминала, 13) toString() (переопределение).
12	Кинофильм. <u>Поля (private):</u> 1) int id_кинофильма (ключевое), 2) String название, 3) String год выпуска, 4) double бюджет. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Кинотеатр. <u>Поля (private):</u> 1) String название_кинотеатра, 2) Кинофильм [] список_кинофильмов, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий кинофильм с заданным значением ключевого поля из списка кинофильмов, 4) геттер, возвращающий число кинофильмов в списке кинотеатра, 5) сеттер для поля название_кинотеатра, 6) добавить кинофильм с учетом уникальности id_кинофильма, 7) удалить кинофильм с заданным id_кинофильма, 8) вернуть средний бюджет для фильмов, демонстрирующихся в кинотеатре, 9) вернуть список фильмов, бюджет которых выше среднего, 10) вернуть список кинофильмов, бюджет которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).
13	Партия_деталей <u>Поля (private):</u> 1) int id_детали (ключевое), 2) String название_детали, 3) String количество_деталей, 4) double цена_детали.	Поставка <u>Поля (private):</u> 1) String шифр_поставки, 2) Партия_деталей [] список_партий, <u>Методы:</u> 1) конструкторы,

1	2	3
	<u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	2) геттеры для всех полей, 3) геттер, возвращающий партию деталей с заданным значением ключевого поля из списка партий, 4) геттер, возвращающий число партий деталей в списке поставки, 5) сеттер для поля шифр_поставки, 6) добавить партию_деталей с учетом уникальности id_детали, 7) удалить партию деталей с заданным id_детали, 8) вернуть среднюю цену детали для поставки, 9) вернуть список партий, цена детали в которых ниже средней, 10) вернуть список партий, цена детали в которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).
14	Секция <u>Поля (private):</u> 1) int id_секции (ключевое), 2) String название_секции, 3) int раз_в_неделю, 4) double цена_за_месяц. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Спортивная_школа. <u>Поля (private):</u> 1) String название_школы, 2) Секция [] список_секций, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий секцию с заданным значением ключевого поля из списка секций, 4) геттер, возвращающий число секций в списке спортивной школы, 5) сеттер для поля название_школы, 6) добавить секцию с учетом уникальности id_секции, 7) удалить секцию с заданным

1	2	3
		id_секции, 8) вернуть среднюю цену_за_месяц для спортивной школы, 9) вернуть список секций, цена_за_месяц в которых ниже средней, 10) вернуть список секций, цена_за_месяц в которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).
15	Клиент <u>Поля (private):</u> 1) int id_клиента(ключевое), 2) String имя_клиента, 3) String мобильный_телефон, 5) int возраст, 6) double общая_стоимость_услуг. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Салон красоты <u>Поля (private):</u> 1) String название_салона; 2) Клиент [] список_клиентов, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий клиента с заданным значением ключевого поля из списка салона красоты, 4) геттер, возвращающий число клиентов в списке салона красоты, 5) сеттер для поля название_салона, 6) добавить клиента с учетом уникальности id_клиента, 7) удалить клиента с заданным id_клиента, 8) вернуть средний возраст клиентов салона красоты, 9) вернуть список клиентов, возраст которых выше среднего, 10) вернуть список клиентов, возраст которых находится в заданном интервале, 11) методы, возвращающие

1	2	3
		отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список клиентов в окно терминала, 13) toString() (переопределение).
16	Надбавка_к_ зарплате <u>Поля (private):</u> 1) int id_надбавки (ключевое), 2) String название_надбавки, 3) float величина_в_ процентах_ от_ основного оклада, 4) int число_раз_в_ году. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Должность <u>Поля (private):</u> 1) String название_должности; 2) float базовая зарплата; 3) Надбавка_к_ зарплате [] список_надбавок, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий надбавку с заданным значением ключевого поля из списка надбавок для должности, 4) геттер, возвращающий число надбавок в списке для должности, 5) сеттер для поля название_должности 6) добавить надбавку с учетом уникальности id_надбавки, 7) удалить надбавку с заданным id_надбавки, 8) вернуть среднюю величину надбавки для должности, 9) вернуть список надбавок, величина которых выше средней, 10) вернуть список надбавок, число_раз_в_ году которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список надбавок в окно терминала, 13) toString() (переопределение).

1	2	3
17	Игрушка <u>Поля (private):</u> 1) int инвентарный_номер(ключевое), 2) String название_игрушки, 3) String год_покупки, 4) float цена. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Группа детсада <u>Поля (private):</u> 1) String шифр_группы; 2) Игрушка [] список_игрушек, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий игрушку с заданным значением ключевого поля из списка группы детсада, 4) геттер, возвращающий число игрушек в списке группы детсада, 5) сеттер для поля шифр_группы, 6) добавить игрушку с учетом уникальности ее инвентарного номера, 7) удалить игрушку с заданным инвентарным номером, 8) вернуть среднюю цену игрушки для группы детсада, 9) вернуть список игрушек, цена которых ниже средней, 10) вернуть список игрушек, год_покупки которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию инвентарного номера), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список клиентов в окно терминала, 13) toString() (переопределение).
18	Терминал <u>Поля (private):</u> 1) int id_терминала (ключевое), 2) String адрес, 3) int время_работы_за_сутки, 4) float поступление_средств, 5) float выдача средств. <u>Методы:</u> 1) конструкторы,	Банк <u>Поля (private):</u> 1) String название_банка; 2) Терминал [] список терминалов, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий терминал с заданным значением ключевого поля из списка банка,

1	2	3
	2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	4) геттер, возвращающий число терминалов в списке банка, 5) сеттер для поля название_банка, 6) добавить терминал с учетом уникальности его id_терминала, 7) удалить терминал с заданным id_терминала, 8) вернуть среднюю сумму поступления_средств для терминалов банка, 9) вернуть список терминалов, поступление_средств на которые выше средней суммы поступления средств, 10) вернуть список терминалов, время_работы которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id_терминала), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список терминалов банка в окно терминала BlueJ, 13) toString() (переопределение).
19	Точка_остановки (населенный пункт или достопримечательность) <u>Поля (private):</u> 1) int id_точки (ключевое), 2) String название_точки, 3) int число_дней, 6) float стоимость_посещения. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Туристическая_путевка <u>Поля (private):</u> 1) String шифр_путевки; 2) Точка_остановки [] список_точек, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий точку остановки с заданным значением ключевого поля из списка путевки, 4) геттер, возвращающий число точек остановки в списке путевки, 5) сеттер для поля шифр_путевки, 6) добавить точку остановки с учетом уникальности id_точки, 7) удалить точку остановки с заданным id_точки, 8) вернуть среднюю стоимость

1	2	3
		посещения точки остановки, 9) вернуть список точек, стоимость посещения которых ниже средней, 10) вернуть список точек остановки, число дней остановки в которых находится в заданном диапазоне, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список точек остановки в окно терминала, 13) toString() (переопределение).
20	Планета <u>Поля (private):</u> 1) String шифр_планеты(ключевое), 2) String название_планеты, 3) double масса, 4) double расстояние_до_звезды, 5) boolean наличие_атмосферы. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Планетная система <u>Поля (private):</u> 1) String шифр_звезды; 2) Планета [] список_планет, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий планету с заданным значением ключевого поля из списка планетной системы, 4) геттер, возвращающий число планет в списке планетной системы, 5) сеттер для поля шифр_звезды, 6) добавить планету с учетом уникальности ее шифра, 7) удалить планету с заданным шифром, 8) вернуть среднюю массу планеты для планетной системы, 9) вернуть список планет, масса которых выше средней, 10) вернуть список планет, у которых расстояние до звезды которых находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию шифра_планеты), в порядке

1	2	3
		возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список планет в окно терминала, 13) toString() (переопределение).
21	Цех <u>Поля (private):</u> 1) String id_цеха(ключевое), 2) String название_цеха, 3) int число_ставок_в_штатном_расписании, 4) int число_рабочих, 5) double зарплата. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Фабрика <u>Поля (private):</u> 1) String название_фабрики; 2) Цех [] список_цехов, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий цех с заданным значением ключевого поля из списка фабрики, 4) геттер, возвращающий число цехов в списке фабрики, 5) сеттер для поля название_фабрики, 6) добавить цех в список с учетом уникальности его id, 7) удалить цех с заданным id, 8) вернуть среднюю зарплату для фабрики, 9) вернуть список цехов, зарплата в которых выше средней, 10) вернуть список цехов, в которых число незанятых ставок находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id_цеха), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список цехов в окно терминала, 13) toString() (переопределение).
22	Горючее <u>Поля (private):</u> 1) String шифр(ключевое), 2) String название_горючего, 3) boolean есть_в_наличии, 4) int min_порция,	Автозаправка <u>Поля (private):</u> 1) String название_заправки; 2) Горючее [] список_наименований_горючего, <u>Методы:</u>

1	2	3
	5) double цена_за_литр. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий горючее с заданным значением ключевого поля из списка автозаправки, 4) геттер, возвращающий число наименований горючего в списке автозаправки, 5) сеттер для поля название_заправки, 6) добавить горючее в список с учетом уникальности его шифра, 7) удалить горючее с заданным шифром, 8) вернуть среднюю цену за литр горючего для фабрики, 9) вернуть список наименований горючего, стоимость которых ниже средней, 10) вернуть список наименований горючего, в которых минимально отпускаемая порция находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию шифра_горючего), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список наименований горючего в окно терминала, 13) toString() (переопределение).
23	Фрукт <u>Поля (private):</u> 1) int шифр_фрукта (ключевое), 2) String название_фрукта, 3) String вид_упаковки, 4) double вес упаковки, 4) double цена_за_кг. 5) boolean есть_в_наличии. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение),	Фруктовый магазин <u>Поля (private):</u> 1) String название_магазина; 2) Фрукт [] список_наименований_фруктов, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий наименование фрукта с заданным значением ключевого поля из списка магазина,

1	2	3
	4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	4) геттер, возвращающий число наименований фруктов в списке магазина, 5) сеттер для поля название_магазина, 6) добавить наименование фрукта в список с учетом уникальности его шифра, 7) удалить наименование фрукта с заданным шифром, 8) вернуть средний вес упаковки фрукта для магазина, 9) вернуть список наименований фруктов, вес упаковки которых выше среднего, 10) вернуть список наименований фруктов, у которых цена за килограмм находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию шифра_фрукта), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список наименований фруктов в окно терминала, 13) toString() (переопределение).
24	Косметическая_процедура <u>Поля (private):</u> 1) int id_процедуры (ключевое), 2) String название_процедуры, 3) int курс (число раз), 4) double цена_за_одну, 5) int скидка_за_приобретение_абонемента_на_курс. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Салон_красоты. <u>Поля (private):</u> 1) String название_салона; 2) Косметическая_процедура [] список_процедур, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий наименование процедуры с заданным значением ключевого поля из списка магазина, 4) геттер, возвращающий число процедур в списке салона красоты, 5) сеттер для поля название_салона, 6) добавить процедуру в список с учетом уникальности ее id,

1	2	3
		7) удалить процедуру с заданным id, 8) вернуть среднюю цену процедуры для салона, 9) вернуть список процедур, цена которых ниже средней, 10) вернуть список процедур, у которых скидка находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id_процедуры), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список процедур в окно терминала, 13) toString() (переопределение).
25	Самолет <u>Поля (private):</u> 1) int id_самолета (ключевое), 2) String название_самолета, 3) String тип_самолета, 4) String авиакомпания, 4) int год выпуска, 5) double оценочная_стоимость, <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Аэропорт <u>Поля (private):</u> 1) String название_аэропорта; 2) Самолет [] список_самолетов, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий наименование самолета с заданным значением ключевого поля из списка аэропорта, 4) геттер, возвращающий число самолетов в списке аэропорта, 5) сеттер для поля название_аэропорта, 6) добавить самолет в список с учетом уникальности его id, 7) удалить самолет с заданным id, 8) вернуть среднюю оценочную_стоимость самолета для аэропорта, 9) вернуть список самолетов, оценочная_стоимость которых ниже средней, 10) вернуть список самолетов, у которых год выпуска находится в заданном интервале,

1	2	3
		11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id_самолета), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список самолетов в окно терминала, 13) toString() (переопределение).
26	Объект_банка (банкомат, терминал, отделение, главный офис) <u>Поля (private):</u> 1) int id_объекта (ключевое), 2) String название_объекта, 3) String адрес, 4) double стоимость_содержания, 5) int персонал (число обслуживающих работников). <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Банк <u>Поля (private):</u> 1) String название_банка; 2) Объект_банка [] список_объектов, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий объект с заданным значением ключевого поля из списка банка, 4) геттер, возвращающий число объектов в списке банка, 5) сеттер для поля название_банка, 6) добавить объект в список с учетом уникальности его id, 7) удалить объект с заданным id, 8) вернуть среднюю стоимость_содержания объекта для банка, 9) вернуть список объектов, стоимость_содержания которых выше средней, 10) вернуть список объектов, у которых персонал (число обслуживающих работников) находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id_объекта), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список объектов в окно терминала, 13) toString() (переопределение).

1	2	3
27	Маршрут_городского_транспорта <u>Поля (private):</u> 1) String код_маршрута (ключевое), 2) String название_маршрута, 3) String вид транспорта, 4) double протяженность_в_км, 5) int число остановок. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Предприятие_городского_транспорта <u>Поля (private):</u> 1) String название_предприятия; 2) Маршрут_городского_транспорта [] список_маршрутов, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий маршрут с заданным значением ключевого поля из списка предприятия, 4) геттер, возвращающий число маршрутов в списке предприятия, 5) сеттер для поля название_предприятия, 6) добавить маршрут в список с учетом уникальности его id, 7) удалить маршрут с заданным id, 8) вернуть среднюю протяженность маршрута для предприятия, 9) вернуть список маршрутов, протяженность которых выше средней, 10) вернуть список маршрутов, у которых число остановок находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id_маршрута), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список маршрутов в окно терминала, 13) toString() (переопределение).
28	Компьютерный_сервис <u>Поля (private):</u> 1) String id_сервиса (ключевое), 2) String название_сервиса, 3) String тип (онлайн-консультация, консультация на месте, установка ПО, апгрейд, обучение и т.п.). 4) double стоимость, 5) int число_сотрудников.	IT-салон <u>Поля (private):</u> 1) String название_салона; 2) Компьютерный_сервис [] список_сервисов, <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий сервис с

1	2	3
	<u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	заданным значением ключевого поля из списка салона, 4) геттер, возвращающий число сервисов в списке салона, 5) сеттер для поля название_салона, 6) добавить сервис в список с учетом уникальности его id, 7) удалить сервис с заданным id, 8) вернуть среднюю стоимость сервиса для салона, 9) вернуть список сервисов, стоимость которых ниже средней, 10) вернуть список сервисов, у которых число сотрудников (способных их выполнять) находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id_сервиса), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список сервисов в окно терминала, 13) toString() (переопределение).
29	Фирма. <u>Поля (private):</u> 1) int id_фирмы (ключевое), 2) String название, 3) int число_сотрудников, 4) double базовая_зарплата. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Проект. <u>Поля (private):</u> 1) String название_проекта, 2) Фирма [] список_фирм. <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий фирму с заданным значением ключевого поля из списка фирм, 4) геттер, возвращающий число фирм в списке проекта. 5) сеттер для поля название_проекта, 6) добавить фирму с учетом уникальности id_фирмы, 7) удалить фирму с заданным id_фирмы, 8) вернуть среднюю базовую

1	2	3
		зарплату для списка фирм, 9) вернуть список фирм, в которых базовая зарплата ниже средней, 10) вернуть список фирм, у которых базовая зарплата находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания числового вещественного поля, по двум полям), 12) вывести список фирм в окно терминала, 13) toString() (переопределение).
30	Штучный товар. <u>Поля (private):</u> 1) int id_товара (ключевое), 2) String название_товара, 3) String тип_товара, 4) int число_штук, 5) float цена_за_штуку. <u>Методы:</u> 1) конструкторы, 2) геттеры и сеттеры для всех полей, 3) toString() (переопределение), 4) equals () (переопределение), 5) hashCode() (переопределение), 6) compareTo() (реализация).	Накладная. Проект. <u>Поля (private):</u> 1) String шифр_накладной, 2) Товар [] список_товаров. <u>Методы:</u> 1) конструкторы, 2) геттеры для всех полей, 3) геттер, возвращающий товар с заданным значением ключевого поля из списка накладной, 4) геттер, возвращающий число товаров в списке накладной. 5) сеттер для поля шифр_накладной, 6) добавить товар с учетом уникальности id_товара, 7) удалить товар с заданным id_товара, 8) вернуть среднюю цену за штуку товара для списка товаров накладной, 9) вернуть список товаров, у которых цена за штуку выше средней, 10) вернуть список товаров, у которых число штук находится в заданном интервале, 11) методы, возвращающие отсортированный список (в естественном порядке (по возрастанию id), в порядке возрастания и убывания

1	2	3
		числового вещественного поля, по двум полям), 12) вывести список товаров в окно терминала, 13) toString() (переопределение).

Таблица 2.2 – Дополнительные методы класса, описывающего группу объектов, подлежащие обязательному программированию

Номер варианта	Дополнительные методы класса, описывающего группу объектов
1	2
1	1) Обновить (в теории БД – update) id у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых начинается с заданного буквосочетания (для перечисления объектов использовать итератор).
2	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 3) Фильтрация данных: вернуть группу объектов, String-поле которых заканчивается заданным буквосочетанием (для перечисления объектов использовать итератор).
3	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых содержит заданное буквосочетание (для перечисления объектов использовать итератор).
4	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых начинается с заданного буквосочетания (для перечисления объектов использовать итератор).

1	2
5	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых заканчивается заданным буквосочетанием (для перечисления объектов использовать итератор).
6	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 3) Фильтрация данных: вернуть группу объектов, String-поле которых содержит заданное буквосочетание (для перечисления объектов использовать итератор).
7	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых выше среднего в группе. 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых начинается с заданного буквосочетания (для перечисления объектов использовать итератор).
8	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 3) Фильтрация данных: вернуть группу объектов, String-поле которых заканчивается заданным буквосочетанием (для перечисления объектов использовать итератор).
9	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых выше среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых содержит заданное буквосочетание (для перечисления объектов использовать итератор).
10	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 3) Фильтрация данных: вернуть группу объектов, String-поле которых начинается с заданного буквосочетания (для

1	2
	перечисления объектов использовать итератор). 3) Фильтрация данных: вернуть группу объектов, String-поле которых содержит заданное буквосочетание (для перечисления объектов использовать итератор).
11	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых выше среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых заканчивается заданным буквосочетанием (для перечисления объектов использовать итератор).
12	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 3) Фильтрация данных: вернуть группу объектов, String-поле которых содержит заданное буквосочетание (для перечисления объектов использовать итератор).
13	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых начинается с заданного буквосочетания (для перечисления объектов использовать итератор).
14	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых выше среднего в группе. 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых заканчивается заданным буквосочетанием (для перечисления объектов использовать итератор).
15	1) Обновить (в теории БД – update) id у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых начинается с заданного буквосочетания (для перечисления объектов использовать итератор).
16	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id).

1	2
	2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 3) Фильтрация данных: вернуть группу объектов, String-поле которых заканчивается заданным буквосочетанием (для перечисления объектов использовать итератор).
17	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых содержит заданное буквосочетание (для перечисления объектов использовать итератор).
18	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых начинается с заданного буквосочетания (для перечисления объектов использовать итератор).
19	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых заканчивается заданным буквосочетанием (для перечисления объектов использовать итератор).
20	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 3) Фильтрация данных: вернуть группу объектов, String-поле которых содержит заданное буквосочетание (для перечисления объектов использовать итератор).
21	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых выше среднего в группе. 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых начинается с заданного буквосочетания (для перечисления объектов использовать итератор).

1	2
22	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 3) Фильтрация данных: вернуть группу объектов, String-поле которых заканчивается заданным буквосочетанием (для перечисления объектов использовать итератор).
23	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых выше среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых содержит заданное буквосочетание (для перечисления объектов использовать итератор).
24	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 3) Фильтрация данных: вернуть группу объектов, String-поле которых начинается с заданного буквосочетания (для перечисления объектов использовать итератор). 3) Фильтрация данных: вернуть группу объектов, String-поле которых содержит заданное буквосочетание (для перечисления объектов использовать итератор).
25	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых выше среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых заканчивается заданным буквосочетанием (для перечисления объектов использовать итератор).
26	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 3) Фильтрация данных: вернуть группу объектов, String-поле которых содержит заданное буквосочетание (для перечисления объектов использовать итератор).
27	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле

1	2
	которых начинается с заданного буквосочетания (для перечисления объектов использовать итератор).
28	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых выше среднего в группе. 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых заканчивается заданным буквосочетанием (для перечисления объектов использовать итератор).
29	1) Обновить (в теории БД – update) id у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых ниже среднего в группе. 3) Фильтрация данных: вернуть группу объектов, String-поле которых начинается с заданного буквосочетания (для перечисления объектов использовать итератор).
30	1) Обновить id (в теории БД – update) у объекта с заданным id (предусмотреть проверку на уникальность нового id). 2) Удалить из группы все объекты, значение числового вещественного поля у которых принадлежит заданному интервалу. 3) Фильтрация данных: вернуть группу объектов, String-поле которых заканчивается заданным буквосочетанием (для перечисления объектов использовать итератор).

Кроме методов, обязательных для программирования, перечисленных в таблицах 2.1 и 2.2 предлагается (по желанию) разработать следующие методы:

запросы на выборку (select):

- 1) вернуть Группу объектов, состоящую из n объектов, выбранных случайным образом – случайная выборка;
- 2) вернуть Группу из n объектов с самым высоким значением числового вещественного поля – вернуть лучших;
- 3) вернуть Группу из n объектов с самым низким значением числового вещественного поля – вернуть худших;

запрос на обновление (update):

- 4) увеличить на заданное значение числовое вещественное поле первым n объектам с самым высоким значением этого поля;

запрос на удаление (delete):

- 5) удалить из группы все объекты с длиной String-поля, превышающей n символов или не включающие заданное буквосочетание.

Кроме того, можно разработать методы, реализующие операции алгебры множеств, например:

- 1) вернуть группу объектов, являющуюся результатом пересечения двух групп объектов (запускающей группы и группы-параметра);
- 2) вернуть группу объектов, являющуюся результатом объединения двух групп объектов (запускающей группы и группы-параметра);
- 3) вернуть группу объектов, являющуюся результатом вычитания группы параметра из запускающей группы объектов (результатом операции вычитания множеств является множество объектов, которые присутствуют в множестве-уменьшаемом, но которых нет в множестве-вычитаемом).

2.7. Содержание отчета

- 1) Цель работы.
- 2) Постановка задачи с указанием данных варианта задания.
- 3) Структуры проектов.
- 4) Описание классов для каждого проекта (полей и методов, инкапсулированных в классе).
- 5) Тексты программ.
- 6) Протоколы отладки и тестирования программ.
- 7) Результаты выполнения программ.
- 8) Результаты сравнения программ.
- 9) Вывод.

2.8. Контрольные вопросы

- 1) Перечислите основные типы коллекций, имеющиеся в Java Collection Framework
- 2) Укажите назначения таких частей JCF как интерфейсы, реализации, алгоритмы. Приведите примеры методов, реализаций и алгоритмов.
- 3) Укажите преимущества, получаемые от применения JCF
- 4) Поясните различие между списками и множествами. Приведите пример, когда один из этих типов не может быть использован вместо другого.
- 5) Как создать список или набор на основе существующего списка или набора?
- 6) Что на самом деле помещается в список (ссылка или объект)?
Дублируются ли объекты, из которых состоит список (набор) при создании списка (набора) из готового списка (набора) или в обоих списках хранятся ссылки на одни и те же объекты? Придумайте, как это можно проверить?
- 7) Что такое итератор? Как его получить и использовать? Какие методы имеет объект-итератор?

- 8) Когда можно и когда нельзя применять цикл `for-each` для перечисления элементов коллекции?
- 9) Где и для чего в вашей программе используется рекурсия?
- 10) Как применение рекурсии в классе, описывающем группу объектов, позволяет избежать применения статических методов обработки списков?
- 11) Почему не рекомендуется применять операцию конкатенации на неизменяемых строках? Чем ее заменить?
- 12) В чем состоит правильный подход к созданию классов с точки зрения безопасности данных и кода?
- 13) Сформулируйте принцип инкапсуляции.
- 14) В чем состоит принцип наследования? Какие методы и какого суперкласса переопределяются в классах программы? Для чего служат эти методы?
- 15) Что такое компаратор? Назовите методы интерфейса `Comparator`. Какой из них нужно реализовать в классе, включающем интерфейс `Comparator`?
- 16) Как создать набор, который будет использовать компаратор для упорядочивания объектов?
- 17) Для чего служит интерфейс `Comparable` и его метод `compareTo()`? Где и для чего вы применили его в своей программе? Проведите исследование: вставьте в код метода операторы, выводящие в окно терминала строки описания запускающего объекта и объекта-параметра. Проанализируйте результаты в окне терминала и сделайте выводы.
- 18) Какой принцип сортировки является определяющим для набора, созданного при помощи реализующего класса `TreeSet`: принцип, определяемый интерфейсом `Comparable` для объекта, входящего в набор, или принцип, определяемый интерфейсом `Comparator` для набора.

3. ЛАБОРАТОРНАЯ РАБОТА № 2. ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ JAVA COLLECTION FRAMEWORK (JCF). КАРТ ОТОБРАЖЕНИЙ

3.1. Цель работы

Целью данной работы является получение навыков использования набора классов и интерфейсов JAVA COLLECTION FRAMEWORK для создания карт отображений объектов-ключей в объекты-значения.

3.2. Постановка задачи

1) Разработать программу, реализующую взаимодействие объектов, образующих иерархическую структуру данных (группу объектов с заданными свойствами и функциями). Для создания объектов, определенных вариантом задания разработать соответствующие классы. Для манипулирования группой объектов использовать возможности интерфейса Map.

2) Сравнить возможности коллекций (списков и множеств) и карт отображений.

3.3. Внеаудиторная подготовка

Для подготовки к лабораторной работе следует изучить краткие теоретические сведения, приведенные в пункте 3.4 методических указаний. Рекомендуются также ознакомиться с [1](с.436 – 448).

3.4. Краткие теоретические сведения

3.4.1. Набор классов и интерфейсов Java Collection Framework

Java обеспечивает работу с отображениями (англ. Map) при помощи интерфейсов Map и SortedMap и реализующих классов – HashMap, TreeMap [4] и др. Отображение, или словарь – это коллекция, каждый элемент которой содержит две части – ключ (key) и значение (value). Ключ используется для доступа к информации, представленной значением, поэтому дублирование ключей запрещено – значения ключей должны быть уникальными. Отображение реализует математическую идею однозначной функции

$$value = f(key).$$

Методы интерфейса Map.

Отображение Map может рассматриваться как множество ключей (множество кружков на оси key, рис.6.1, слева) или как множество пар «ключ,значение» (множество из шести точек на плоскости, там же) или как

коллекция значений (набор точек на оси value, там же) [1-3]. В коллекцию значений каждое из значений приведенного отображения попадет дважды. Ниже приведены основные методы интерфейса Map<K,V>, часть из которых использована в программе-шаблоне 1.

```
public Set<K> keySet( ); // получить множество ключей
public Set<Map.Entry<K,V>> entrySet( ); // получить множество пар
public Collection<V> values( ); // получить коллекцию значений
V put(K key, V value); // задает f(key) для данного key
V get(Object key); // f(key)
V remove(Object key); // удаляет пару (key,value) для данного key
boolean containsKey(Object key); // проверка, есть ли заданный ключ
boolean containsValue(Object value); // проверка, есть ли заданное значение
void putAll(Map<? extends K, ? extends V> m); // добавить все пары m
// другой карты Map
void clear( ); // уничтожить все пары ключ-значение (входы карты)
int size( ); // вычисление размера карты – количества пар
boolean isEmpty( ); // проверка, пуста ли карта
```

Для работы с отдельными парами отображения предусмотрен интерфейс **Entry** (подинтерфейс **Map**) с методами **K getKey()** (получить из пары ключ), **V getValue()** (получить из пары значение) и **V setValue(V value)** (задать значение в паре).

Интерфейс SortedMap и методы сравнения элементов.

Интерфейс SortedMap и реализующий класс TreeMap предполагают, что пары «ключ,значение» упорядочены по ключам *естественным* образом или с учетом заданного программистом отношения порядка.

Естественное сравнение используется по умолчанию в ряде методов (например, в сортировке sort) и в конструкторах, т.е. без дополнительных указаний на метод сравнения. Естественное отношение порядка сравнивает числа по положению на числовой оси, символы – по числовому значению кода, строки – лексикографически. Программист может указать, что следует использовать другой метод сравнения. Для этого он должен реализовать интерфейс Comparator, т.е. запрограммировать его метод

int Compare(T a, T b).

Этот метод сравнивает элементы a и b типа T. Некоторые методы и конструкторы требуют дополнительный параметр, чтобы указать необходимость именно такого сравнения.

3.4.2. Пример создания и манипулирования группой объектов на основе интерфейса Map

В теории баз данных под ключом некоторой сущности, понимается поле или совокупность полей, которая однозначно определяет конкретный экземпляр этой сущности. Т. е. различные экземпляры одной и той же сущности должны отличаться значениями хотя бы в одном поле ключа.

Ключ, состоящий из одного поля, называется простым. Такой ключ использовался в предыдущей работе (id).

Ключ, состоящий из совокупности полей называется сложным или составным. Составной ключ должен быть избыточным. Это значит, что ни одно поле не может быть удалено из ключа без нарушения функции ключа (ключ однозначно определяет экземпляр сущности, а, значит, значения всех неключевых полей экземпляра).

Представляется наиболее интересным исследовать возможности интерфейса Мар для работы с группами объектов, ключи которых состоят из нескольких полей (составные ключи).

Рассмотрим выделенный фрагмент знакомой нам уже предметной области (базы данных – БД), который представляет собой сущность Экзаменационная ведомость (группа экзаменов) – рисунок 3.1. Пример экзаменационной ведомости изображен на рисунке 3.2.

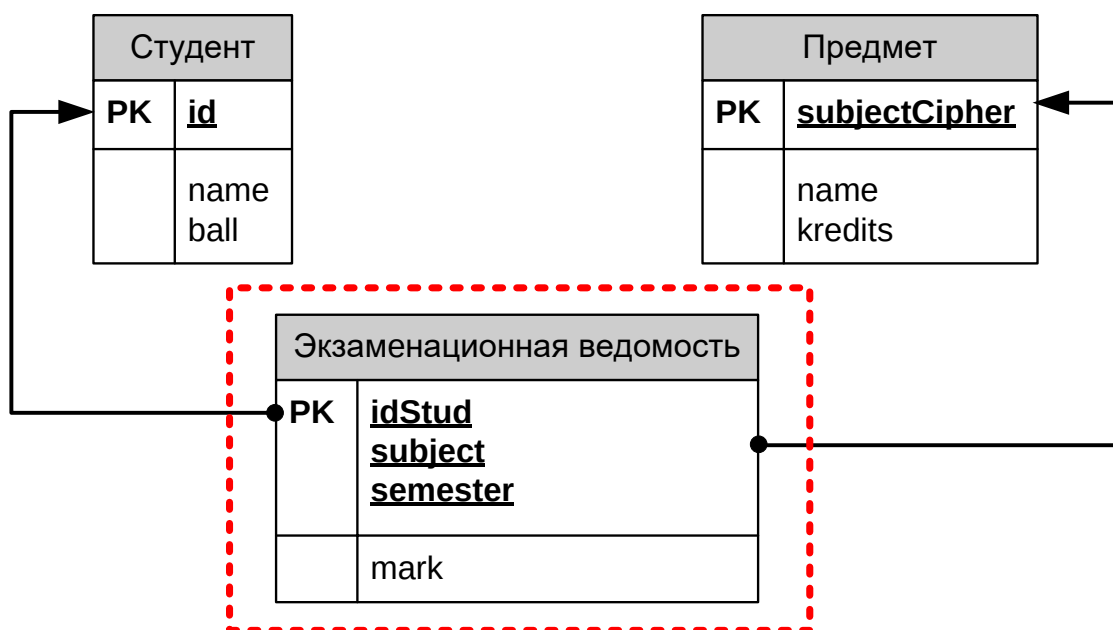


Рисунок 3.1 – Фрагмент схемы БД «Университет»

Экзаменационная ведомость кафедры ИТиКС			
Номер_зач_кн (idStud)	Шифр_предмета (subject)	Номер_семестра (semester)	Оценка (mark)
125550	ДМ	1	отлично
125550	ВМ	1	хорошо
125550	Физика	1	отлично
125551	ДМ	1	удовлетворительно
...	...	...	...
125550	ДМ	2	неудовлетворительно
...	...	...	...

Экземпляр записи об экзамене – запись о конкретном экзамене (ключ → значение)

Рисунок 3.2 – Пример экзаменационной ведомости кафедры

Как видно из рисунка 3.1, сущность «Экзаменационная ведомость» связывает между собой две другие сущности: «Студент» и «Предмет», которые останутся за пределами нашего рассмотрения в данной лабораторной работе.

Экземпляр сущности «Экзаменационная ведомость» (рисунок 3.2) – это данные о конкретном экзамене (т.е. данные о том, какой студент, по какому предмету в каком семестре, какую получил оценку). Сущность имеет ключ (Primary Key – PK), состоящий из трех полей: номер зачетной книжки (idStud), шифр предмета (subject), номер семестра (semester). Совокупность значений полей, составляющих ключ, однозначно определяет экземпляр сущности «Экзаменационная ведомость» (запись об экзамене). У различных экземпляров значения ключей должны отличаться хотя бы в одном поле. Каждому ключу соответствует строго одно значение оценки в неключевом поле mark. Примеры различных экземпляров (ключ → значение) сущности «Экзаменационная ведомость»:

(12345, "ДМ", 1)	→	("хорошо");
(12345, "КЛ", 1)	→	("хорошо");
(12345, "ДМ", 2)	→	("отлично");
(12346, "ДМ", 1)	→	("хорошо");
(12347, "Программирование", 2)	→	("удовлетворительно");
(12346, "ДМ", 2)	→	("неудовлетворительно").

В предлагаемом проекте сущность «Экзаменационная ведомость» (группа экзаменов) реализуется на основе интерфейса Map). Одним из достоинств такой реализации является осуществление автоматической проверки уникальности составного ключа при добавлении очередной записи об экзамене в ведомость (карту отображений).

Структура проекта изображена на рисунке на рисунке 3.3.

При запуске метода main() данные в массив args нужно загрузить из файла экзамены.txt, находящегося в папке проекта.

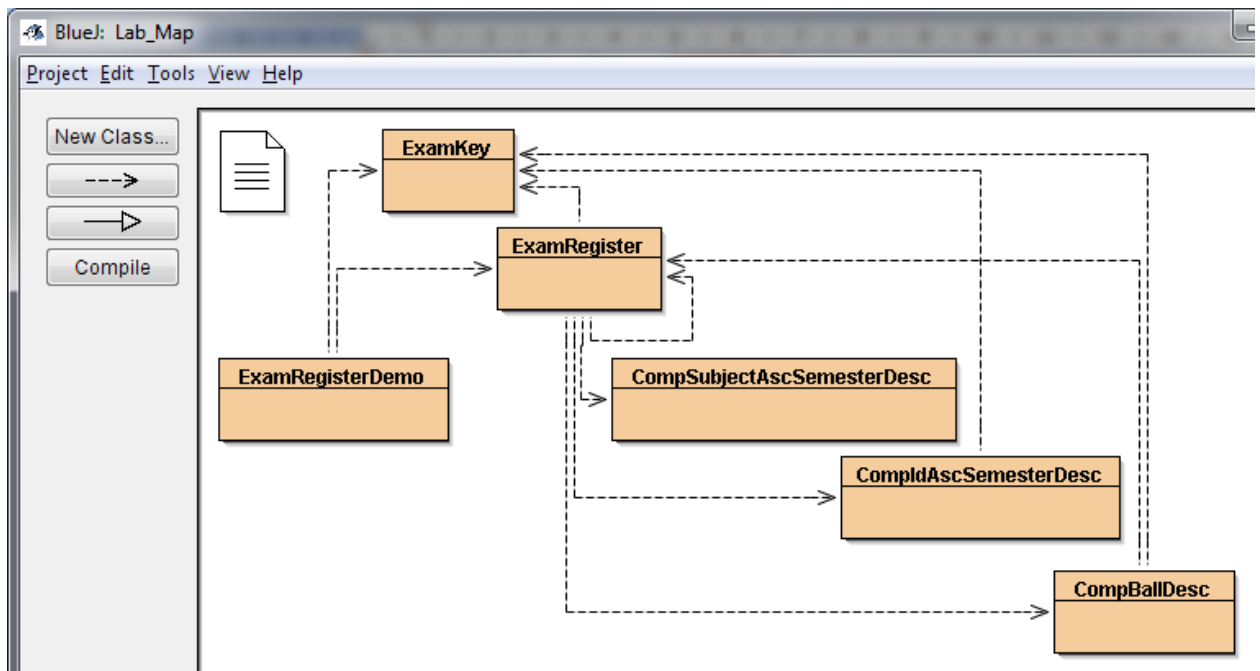


Рисунок 3.3 – Структура проекта

Класс ExamKey описывает ключ записи об экзамене, содержит private-поля (idStud, subject, semester), конструкторы для создания объектов, методы-геттеры и сеттеры для обращения к private-полям, переопределяет методы toString(), equals() и hashCode() суперкласса Object, реализует метод CompareTo() интерфейса Comparable (для задания естественного порядка сортировки объектов).

Класс ExamRegister() описывает сущность «Экзаменационная ведомость», т.е группу (карту) экзаменов. В статической области класса определена и инициализирована ссылка на список возможных оценок MARKS (final-переменная), а также определен метод transMark, возвращающий int-эквивалент String-оценки. Эти члены класса разделяются всеми объектами данного класса (являются принадлежностью не объекта, а класса).

Класс содержит следующие переменные экземпляра (нестатические поля):

name – название ведомости (группы экзаменов);

register – интерфейсная ссылка на объект типа Map – карту отображений ключей экзаменов, включенных в данную ведомость, на соответствующие значения полученных студентами оценок.

Класс имеет несколько конструкторов, в том числе конструктор, позволяющий задать компаратор для карты register. Карты отображений во всех конструкторах создаются на основе класса TreeMap.

Другие методы класса:

setRegisterName – сеттер для поля названия ведомости;

getRegisterName(), getRegister() – геттеры для полей;

toString() – возвращает строку описания объекта;

запрос на вставку данных:

addExam() – добавляет экзамен (ключ и значение оценки) в ведомость;

запросы на удаление данных:

delExam(ExamKey ek) – исключает запись об экзамене, заданную ключом ek из ведомости;

delExam(int idStudent) – исключает все записи об экзаменах заданного студента;

запрос на обновление данных:

updateExam() – меняет на заданную оценку экзамена с заданным ключом;

запросы на выборку данных:

size() – вычисляет размер экзаменационной ведомости (количество записей об экзаменах в ней);

selectSumBallStud() – вычисляет суммарный балл конкретного студента (по результатам всех его экзаменов в ведомости);

selectStudentData() – возвращает подведомость (выборку экзаменов) для заданного студента;

selectSubjectData() – возвращает подведомость (выборку экзаменов по заданному предмету);

selectStudentSemestrData() – возвращает подведомость (выборку экзаменов для заданного студента в заданном семестре);

запросы на сортировку данных:

sortIdAscSemesterDesc() – возвращает ведомость, являющуюся результатом сортировки основной ведомости по возрастанию номера зачетки студента и убыванию номера семестра;

sortSubjectAscSemesterDesc() – возвращает ведомость, являющуюся результатом сортировки основной ведомости по возрастанию шифра предмета (лексикографически) и убыванию номера семестра;

sortBallDesc() – возвращает ведомость, являющуюся результатом сортировки основной ведомости по баллу;

запрос на вывод данных:

putExamRegister() – выводит основную ведомость (текущий объект) в окно терминала.

Названия методов, возвращающих ссылки на ведомости (объекты определяемого класса ExamRegister) выделены полужирным шрифтом. Можно сказать, что эти методы возвращают ссылки на некоторые подмножества основной ведомости (текущего объекта), которые, в свою очередь, являются ведомостями и, следовательно, могут выполнять все вышеперечисленные методы, определенные в описываемом классе. Мы наблюдаем рекурсию при определении данных.

Кроме описанных выше классов, проект содержит три класса, реализующих интерфейс Comparator: CompIdAscSemesterDesc, CompSubjectAscSemesterDesc, CompBallDesc. Принцип их действия описан в предыдущей лабораторной работе. Наиболее интересен класс CompBallDesc, с помощью которого может быть реализована сортировка не по ключам (как это принято в TreeMap), а по значениям (балльным эквивалентам оценок). Обратите внимание на следующие моменты.

1) Для получения значений в методе compare, мы все равно используем ключи, а также ссылку на карту, являющуюся предметом сортировки, передаваемую в конструктор компаратора (конструктор с параметром);

2) Значения (оценки) в карте не уникальны, в отличие от ключей, а результат, возвращенный методом compare используется системой для принятия решения о добавлении (или запрете добавления) объекта в набор TreeMap. Если при сравнении добавляемого объекта с объектами, добавленными в карту ранее, метод compare на одинаковых значениях оценки будет возвращать 0 (признак равенства объектов), то добавляемый объект не будет добавлен в сортируемый набор. Поэтому возможность возвращения нуля методом compare компаратора CompBallDesc закрыта. Проанализируйте код метода, измените его так, чтобы он при равных значениях оценок сравниваемых объектов возвращал 0, запустите проект и посмотрите, как изменился вывод программы.

Класс ExamRegisterDemo включает метод main, в котором осуществляется демонстрация основных возможностей, предусмотренных в проекте для работы с экзаменационными ведомостями.

Текст программы.

```
public class ExamKey implements Comparable <ExamKey> {
    // Сущность - Ключ экзамена
    //Именованная константа -
    //формат записи о Ключе экзамена:
    private final static String EXKEY_FORMAT_STRING =
        "Студент: %1d | Предмет: %1s | Семестр: %1d |";
    //Переменные экземпляра - скрыты в классе
    private int idStudent; //номер зачетной книжки студента
    private String subject; //шифр предмета
```

```

private int semester; //номер семестра
// конструктор без параметров
public ExamKey(){
    idStudent = 0; subject = ""; semester = 0; }
// конструктор с параметрами
public ExamKey(int idStudent, String subject, int semester){
    this.idStudent = idStudent; this.subject = subject;
    this.semester = semester;
}
//методы-геттеры
public int getIdStudent(){return idStudent;}
public String getSubject(){return subject;}
public int getSemester(){return semester;}
//методы-сеттеры
public void setIdStudent(int idStudent){this.idStudent = idStudent;}
public void setSubject(String subject){this.subject = subject;}
public void setSemester(int semester){this.semester = semester;}
//Переопределяется метод toString класса Object
//(возвращает строку описания объекта)
public String toString(){
    return
        String.format(EXKEY_FORMAT_STRING, idStudent, subject, semester);
}
//Переопределяется метод equals класса Object
//(задает способ сравнения объектов на равенство,
//возвращает true, если запускающий объект равен объекту-параметру)
public boolean equals (Object ob) {
    if (ob == this) return true; // ссылки равны – один
                                // и тот же объект
    if (ob == null) return false; // в метод передана null-ссылка
    if (getClass() != ob.getClass())return false; //объекты разных классов
    ExamKey ek = (ExamKey)ob; // преобразуем Object в Student
    return (idStudent == ek.idStudent)&&(subject.equals(ek.subject))
        &&(semester == ek.semester); //true, если каждое из полей
                                //текущего (запускающего) объекта равно
                                //соответствующему полю /объекта-параметра
}
//Переопределяется метод hashCode класса Object
//Возвращает хэш-код объекта
//(у равных объектов должны быть равные hash-коды)
public int hashCode(){
    return 7* (new Integer(idStudent)).hashCode()+
        11 * subject.hashCode()+
        13 * (new Integer(semester)).hashCode();
}

//Определяем метод CompareTo интерфейса Comparable
//Для определения порядка (естественного) перечисления элементов
public int compareTo(ExamKey ek){
    //по возрастанию номера зачетки,
    //если номера зачетки одинаковые -
    //по возрастанию шифра предмета,

```

```

//если шифры предмета одинаковые -
//по возрастанию семестра
if (idStudent < ek.idStudent) return -1;
if (idStudent > ek.idStudent) return 1;
//номера зачетов равны (один и тот же студент)
if (subject.compareTo(ek.subject)<0) return -1;
if (subject.compareTo(ek.subject)>0) return 1;
//предметы равны (один и тот же предмет)
if (semester < ek.semester) return -1;
if (semester > ek.semester) return 1;
return 0; //семестры равны
}
} // ExamKey

import java.util.*;
public class ExamRegister { // Сущность - Зачетно-экзаменационная
    //ведомость (Группа экзаменов)
    //Поля
    //Статические элементы разделяются всеми объектами данного
    // класса т.е. являются общими для всех объектов
    //Инициализируем список возможных оценок (класс Arrays
    //находится в пакете java.util) и является членом
    //Java Collections Framework
    private final static List <String> MARKS =
        Arrays.asList("отлично","хорошо",
            "удовлетворительно","неудовлетворительно");
    //формат записи о группе экзаменов:
    private final static String EXREG_FORMAT_STRING =
        "\"Экзамены: %s, количество: %d\"";
    //переменные экземпляра:
    private String name; //название группы экзаменов (ведомости)
    private Map <ExamKey, String> register; // ссылка на карту (ведомость)
        //экзаменов

    // конструкторы
    public ExamRegister(){
        name = ""; //без названия
        register = new TreeMap <ExamKey, String>(); //создается пустая карта
    }

    public ExamRegister(String name) {
        this.name = name; //задается название ведомости
        register = new TreeMap <ExamKey, String>(); //создается пустая карта
    }

    public ExamRegister(String name, Comparator comp){
        this.name = name; //задается название ведомости
        register = new TreeMap <ExamKey, String> (comp); //создается пустая
            // карта, в которой ссылки на добавляемые объекты-ключи будут
            // упорядочиваться по правилам компаратора comp
    }

    //метод-сеттер для private-поля
    public void setRegisterName(String name)
        {this.name = name;}
    //методы-геттеры для private-полей

```

```

public String getRegisterName(){return name;}
public Map <ExamKey, String> getRegister(){
    return register;
}
//Переопределяем метод toString класса Object
//возвращает строку описания объекта Группа студентов:
public String toString(){
    return String.format(EXREG_FORMAT_STRING,name,size());
}

//Запросы на вставку и удаление данных:
public boolean addExam(ExamKey ek, String mark){
    //добавить экзамен (ключ и значение(оценку)) в карту
    //экзамен не будет вставлен, если в наборе уже есть экзамен
    //с таким же ключом
    if (register.containsKey(ek)) return false;
    if (!MARKS.contains(mark)) return false;
    register.put(ek,mark);
    return true;
}

public boolean delExam(ExamKey ek){
    //удалить экзамен из ведомости по ключу
    if (register.containsKey(ek)){register.remove(ek); return true;}
    else return false;
}

public boolean delExam(int idStudent){
    //удалить все экзамены конкретного студента
    Set <Map.Entry <ExamKey, String>> setE = register.entrySet();
    Iterator <Map.Entry <ExamKey,String>> it = setE.iterator();
    int i = 0; //счетчик найденных экзаменов
    while (it.hasNext()){
        Map.Entry <ExamKey,String> keyVal = it.next();
        if (keyVal.getKey().getIdStudent() == idStudent){
            it.remove();
            i = i+1;
        }
    }
    if (i == 0) return false;
    return true;
    //Удаление входа (пары ключ-значение)из набора setE методом
//remove() итератора it влечет за собой удаление этого входа
//из карты register (формируется неявный итератор для карты).
//Любой другой способ удаления входа
//при циклическом просмотре входов карты вызовет ошибку
//типа ConcurrentModificationException.

public boolean updateExam(ExamKey ek, String mark){
    //обновить оценку на ключе (студент пересдал экзамен)
    if (!register.containsKey(ek)) return false;
    if (!MARKS.contains(mark)) return false;

```

```

        register.put(ek, mark);
        return true;
    }

    // Запросы на выборку данных:
    public int size(){
        //Возвращает число экзаменов в ведомости
        return register.size();
    }

    public ExamRegister selectStudentData(int idStud){
        //Возвращает карту экзаменов заданного студента
        ExamRegister subExReg = new ExamRegister ( String.format(
            "%s: выборка по студенту %7d",
            name,idStud)); // создаем (под)ведомость
        Set <Map.Entry <ExamKey, String>> setE = register.entrySet();
        for (Map.Entry <ExamKey, String> keyVal:setE)
            if (keyVal.getKey().getIdStudent() == idStud)
                subExReg.register.put(keyVal.getKey(),keyVal.getValue());
        return subExReg;
    }

    public ExamRegister selectSubjectData(String subject){
        //Возвращает карту экзаменов по заданному предмету
        ExamRegister subExReg = new ExamRegister ( String.format(
            "%s: выборка по предмету %s",
            name,subject)); // создаем (под)ведомость
        Set <Map.Entry <ExamKey, String>> setE = register.entrySet();
        for (Map.Entry <ExamKey, String> keyVal: setE)
            if (keyVal.getKey().getSubject().equals(subject))
                subExReg.register.put(keyVal.getKey(), keyVal.getValue());
        return subExReg;
    }

    public ExamRegister selectStudentSemestrData(int idStud, int semester){
        //Возвращает карту экзаменов заданного студента
        //в заданном семестре
        ExamRegister subExReg = new ExamRegister (
            String.format("%s: выборка по студенту %1d и семестру %1d",
                name,idStud, semester)); // создаем (под)ведомость
        Set <Map.Entry <ExamKey, String>> setE = register.entrySet();
        for (Map.Entry <ExamKey, String> keyVal:setE)
            if ((keyVal.getKey().getIdStudent() ==
                idStud)&&(keyVal.getKey().getSemester() == semester))
                subExReg.register.put(keyVal.getKey(), keyVal.getValue());
        return subExReg;
    }

    public static int transMark(String mark){
        //Перевод оценки в числовую форму
        if (mark=="отлично") return 5;
        if (mark=="хорошо") return 4;
        if (mark=="удовлетворительно") return 3;
    }

```

```

    return 0;
}

public int selectSumBallStud(int idStud){
    //Возвращает суммарный балл для конкретного студента
    int sum = 0;
    //Получаем выборку(карту) по заданному студенту
    ExamRegister subReg = selectStudentData(idStud);
    //Получаем из карты набор значений (оценок)
    Collection <String> v = register.values();
    //Суммируем оценки
    for (String mark: v) sum += transMark(mark);
    return sum;
}
//запросы на сортировку данных
public ExamRegister sortIdAscSemesterDesc(){
    //По возрастанию номера зачетки
    ExamRegister subExReg = new ExamRegister (String.format(
"%s:\nИт сортировка по возрастанию номера зачетки и убыванию семестра",
    name), new CompIdAscSemesterDesc()); // создаем новую ведомость
    Set <Map.Entry <ExamKey, String>> setE = register.entrySet();
    for (Map.Entry <ExamKey, String> keyVal:setE)
        subExReg.addExam(keyVal.getKey(), keyVal.getValue()); // при добавлении
// в карту производится упорядочивание ссылок на объекты-ключи по
// правилу, задаваемому компаратором (сортировка)
    return subExReg;
}

public ExamRegister sortSubjectAscSemesterDesc(){
    //По возрастанию шифра предмета
    ExamRegister subExReg = new ExamRegister(String.format(
"%s:\nИт сортировка по возрастанию шифра предмета и убыванию семестра",
    name), new CompSubjectAscSemesterDesc()); // создаем новую ведомость
    Set <Map.Entry <ExamKey, String>> setE = register.entrySet();
    for (Map.Entry <ExamKey, String> keyVal:setE)
        subExReg.addExam(keyVal.getKey(), keyVal.getValue());
    return subExReg;
}

public ExamRegister sortBallDesc (){
    //По убыванию балла
    ExamRegister subExReg = new ExamRegister(String.format(
"%s:\nИт сортировка по убыванию балла",
    name), new CompBallDesc(this.register)); // создаем новую ведомость
    Set <Map.Entry <ExamKey, String>> setE = register.entrySet();
    for (Map.Entry <ExamKey, String> keyVal:setE)
        subExReg.addExam(keyVal.getKey(), keyVal.getValue());
    return subExReg;
}

// Запрос на вывод данных

```

```

public void putExamRegister(){
    //Вывод экзаменационной ведомости в окно терминала
    System.out.println(name); //название ведомости
    //заголовки столбцов
    System.out.printf("%5s%17s%19s%15s%15s\n",
        "Номер","Номер зач. кн.","Шифр предмета","Семестр","Оценка");
    int i = 1;
    Set <Map.Entry <ExamKey, String>> setE = register.entrySet();
    for (Map.Entry <ExamKey, String> keyVal:setE){
        ExamKey key = keyVal.getKey(); //получили ключ очередного входа карты
        System.out.printf(" %-7d %-15d %-16s %-7d %-20s\n",
            i, key.getIdStudent(), key.getSubject(), key.getSemester(),
            keyVal.getValue());
        i = i+1;
    }
} // putStudGroup
} // ExamRegister

import java.util.*;
//компаратор
public class CompSubjectAscSemesterDesc
    implements Comparator <ExamKey> {
    public int compare (ExamKey ex1, ExamKey ex2){
        String subj1 = ex1.getSubject();
        String subj2 = ex2.getSubject();
        int sem1 = ex1.getSemester();
        int sem2 = ex2.getSemester();
        if (subj1.compareTo(subj2) < 0)return -1;
        if (subj1.compareTo(subj2) > 0)return 1;
        //шифры предметов равны (один и тот же предмет)
        if (sem1 > sem2) return -1;
        if (sem1 < sem2) return 1;
        return 0;
    }
    //нет необходимости переопределять метод equals
} // CompSubjectAscSemesterDesc

import java.util.*;
//компаратор
public class CompIdAscSemesterDesc implements Comparator <ExamKey>{
    public int compare (ExamKey ex1, ExamKey ex2){
        int id1 = ex1.getIdStudent();
        int id2 = ex2.getIdStudent();
        int sem1 = ex1.getSemester();
        int sem2 = ex2.getSemester();
        if (id1<id2) return -1;
        if (id1>id2) return 1;
        //номера зачеток совпадают (один и тот же студент)
        if (sem1>sem2) return -1;
        if (sem1<sem2) return 1;
        return 0;
    }
}

```

```

//нет необходимости переопределять метод equals
} //CompIdAscSemesterDesc

```

```

import java.util.*;
//компаратор
public class CompBallDesc implements Comparator <ExamKey>{
    // Как же организовать сортировку карты по значениям,
    // если (по определению) она сортируется по ключам?
    // Нужно получить ссылку на сортируемый набор, чтобы
    // через его ключи получить доступ к значениям!
    Map <ExamKey, String> registr;
    public CompBallDesc (Map <ExamKey, String> registr){ //конструктор
        this.registr = registr;
    }
    public int compare (ExamKey ex1, ExamKey ex2){
        int ball1 = ExamRegister.transMark(registr.get(ex1));
        int ball2 = ExamRegister.transMark(registr.get(ex2));
        if (ball1 > ball2) return -1;
        return 1;
        //Метод не должен никогда возвращать ноль.
        //В случае, когда он вернул бы ноль при одинаковых значениях
        //анализируемых неключевых полей, соответствующая запись об
        // экзамене не была бы добавлена в карту, являющуюся результатом
        // сортировки, т.к. считалось бы, что такой объект там уже есть.
    }
    //нет необходимости переопределять метод equals
} // CompBallDesc

```

```

public class ExamRegisterDemo{ // Взаимодействие студентов и групп
    public static void main (String[] args){
        //Создаем зачетно-экзаменационную ведомость
        ExamRegister reg=new ExamRegister ("Ведомость кафедры ИТиКС");
        // в массив args при запуске метода main (String[] args)
        // введем данные о 14 экзаменах
        int j = 0; //индекс массива args
        // добавление студентов в первую группу
        for (int i = 0; i < 14; i++){
            // получаем int-эквивалент из String:
            int idStud = Integer.valueOf(args[ j ]); j++;
            String subject = args[ j ]; j++;
            //получен float-эквивалент из String:
            int semester = Integer.valueOf(args[ j ]); j++;
            String mark = args[ j ]; j++;
            //добавить данные об экзамене в карту (ключ и значение)
            reg.addExam(new ExamKey(idStud, subject, semester), mark);
        }
        // вывод карты (естественное упорядочивание:
        //по возрастанию номера зачетки,
        //если номера зачетки одинаковые -
        //по возрастанию шифра предмета,

```

```

//если шифры предмета одинаковые -
//по возрастанию семестра
System.out.println(
    String.format("Начальное состояние ведомости %s:", reg));
reg.putExamRegister();
//запросы на добавление и обновление (изменение) записей об экзаменах
System.out.println(
    "\n-----1.Запрос на добавление записи об экзамене.-----");
//попытка добавить экзамен с ключом, который уже присутствует
//в ведомости
ExamKey ek = new ExamKey(55999,"Программирование",1);
if (!reg.addExam(ek, "хорошо"))
    System.out.println(
        String.format(
            "Запись %s\n\t нарушает правило уникальности ключа",ek));
System.out.println(
    "\n-----2.Запрос на обновление записи об экзамене.-----");
//Изменить оценку на существующем в карте ключе - студент
//пересдал на 5
ek = new ExamKey(15000,"Математика",1);
reg.updateExam(ek,"отлично");
//Выводим ведомость после обновления записи
System.out.println(
    String.format("После обновления оценки в записи:\n\t %s",ek));
reg.putExamRegister();

//Запросы на выборку данных
System.out.println(
    "\n-----3.Запросы на выборку записей об экзаменах.-----");
// Выводим выборку из карты экзаменов для заданного студента,
// упорядоченную по названию предмета и семестру
reg.selectStudentData(15002).sortSubjectAscSemesterDesc().putExamRegister();
// Выводим суммарный балл для заданного студента
System.out.println(
    String.format("Суммарный балл студента 15002: %5d:",
        reg.selectSumBallStud(15002)));
// Выводим выборку из карты экзаменов для заданного студента в
// заданном семестре, упорядоченную по баллу
reg.selectStudentSemestrData(15002,1).sortBallDesc ().putExamRegister();
// Выводим выборку из карты экзаменов для заданного предмета,
// упорядоченную по номеру зачетки и семестру
reg.selectSubjectData("Математика").sortIdAscSemesterDesc().putExamRegister();

// Запросы на удаление записей об экзаменах
System.out.println(
    "\n-----4.Запросы на удаление записей об экзаменах.-----");
//удаляем из ведомости
//запись о сдаче программирования студентом 16090 в четвертом
// семестре
ek = new ExamKey(15000,"Математика",1);
reg.delExam(ek);
//Выводим ведомость после удаления записи

```

```

System.out.println(
    String.format("После удаления записи:\n\t %s",ek));
reg.putExamRegister();

//пытаемся удалить запись, которой нет в ведомости
ek = new ExamKey(75000,"Математика",1);
if (!reg.delExam(ek))
    System.out.println(
        String.format("Попытка удаления несуществующей записи:\n\t %s",ek));
//Удаляем из ведомости все экзамены студента 15002
reg.delExam(15002);
//Выводим ведомость после удаления записей
System.out.println(
    "После удаления всех записей об экзаменах студента 15002");
reg.putExamRegister();
//пытаемся удалить экзамены студента, которого нет в ведомости
if (!reg.delExam(75000))
    System.out.println(
        String.format(
            "Попытка удаления экзаменов несуществующего студента:%7d",75000));
System.out.println(
    "\n Рассмотренная структура программы обработки базы данных (БД):");
System.out.println("код и вызовы запросов встроены в код программы,");
System.out.println("может применяться, если список запросов к БД");
System.out.println("ограничен и заранее известен.");
System.out.println("В противном случае, нужно использовать
возможности Java");
    System.out.println("по применению встроенного динамического SQL.");
}
} //ExamRegisterDemo

```

Результаты выполнения программы изображены на рисунке 3.4.

3.5. Порядок выполнения работы

- 1) Внимательно прочитайте п. 3.4 и разберите текст программы-примера.
- 2) По аналогии с примером (п. 3.4.2) разработайте программу, реализующую группу объектов на основе интерфейса Map для заданной вариантном предметной области.
- 3) Проведите отладку программы и проверку ее работы на тестовых примерах.

3.6. Варианты заданий

Варианты заданий представлены в таблице 3.1.

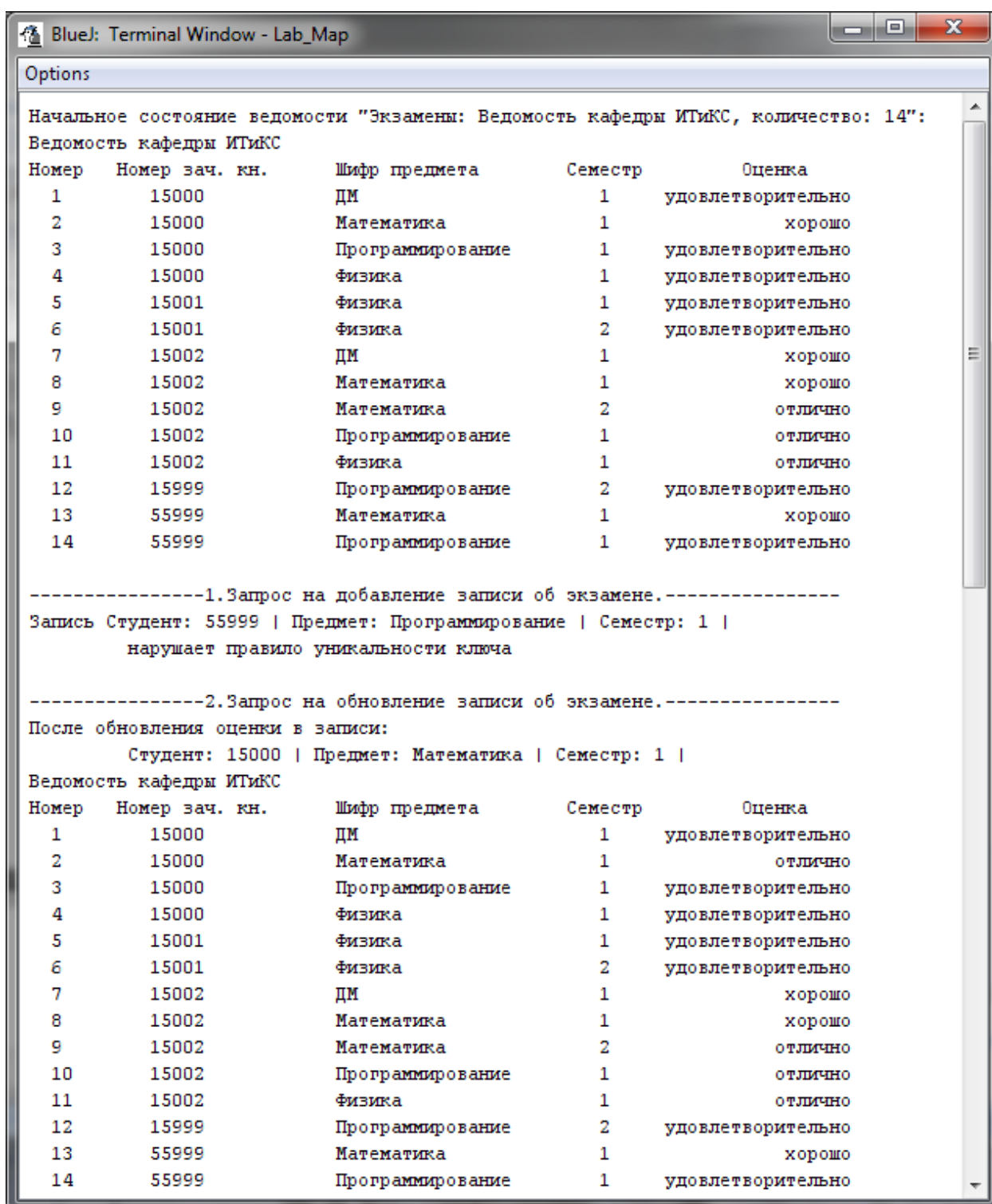


Рисунок 3.4 – Результаты работы программы

-----3.Запросы на выборку записей об экзаменах.-----

Суммарный балл студента 15002: 53:

Ведомость кафедры ИТиКС: выборка по студенту 15002 и семестру 1:

сортировка по убыванию балла

Номер	Номер зач. кн.	Шифр предмета	Семестр	Оценка
1	15002	Программирование	1	отлично
2	15002	Физика	1	отлично
3	15002	ДМ	1	хорошо
4	15002	Математика	1	хорошо

-----4.Запросы на удаление записей об экзаменах.-----

После удаления записи:

Студент: 15000 | Предмет: Математика | Семестр: 1 |

Ведомость кафедры ИТиКС

Номер	Номер зач. кн.	Шифр предмета	Семестр	Оценка
1	15000	ДМ	1	удовлетворительно
2	15000	Программирование	1	удовлетворительно
3	15000	Физика	1	удовлетворительно
4	15001	Физика	1	удовлетворительно
5	15001	Физика	2	удовлетворительно
6	15002	ДМ	1	хорошо
7	15002	Математика	1	хорошо
8	15002	Математика	2	отлично
9	15002	Программирование	1	отлично
10	15002	Физика	1	отлично
11	15999	Программирование	2	удовлетворительно
12	55999	Математика	1	хорошо
13	55999	Программирование	1	удовлетворительно

Попытка удаления несуществующей записи:

Студент: 75000 | Предмет: Математика | Семестр: 1 |

После удаления всех записей об экзаменах студента 15002

Ведомость кафедры ИТиКС

Номер	Номер зач. кн.	Шифр предмета	Семестр	Оценка
1	15000	ДМ	1	удовлетворительно
2	15000	Программирование	1	удовлетворительно
3	15000	Физика	1	удовлетворительно
4	15001	Физика	1	удовлетворительно
5	15001	Физика	2	удовлетворительно
6	15999	Программирование	2	удовлетворительно
7	55999	Математика	1	хорошо
8	55999	Программирование	1	удовлетворительно

Попытка удаления экзаменов несуществующего студента: 75000

Рассмотренная структура программы обработки базы данных (БД):

код и вызовы запросов встроены в код программы,

может применяться, если список запросов к БД

ограничен и заранее известен.

В противном случае, нужно использовать возможности Java

по применению встроеного динамического SQL.

Продолжение рисунка 3.4

Таблица 3.1 – Варианты заданий

Но- мер вари- анта	Описание предметной области	Название группы записей	Ключ записи	Значение, определяемое ключом
1	2	3	4	5
1	Фирмы выполняют поставки деталей	Журнал учета поставок	Идентификационный номер фирмы (int), шифр детали (String), дата (String).	Количество деталей в поставке (int).
2	В офисах проводится уборочные работы в определенные дни недели	График клининга	Идентификационный номер офиса (int), шифр уборочной работы (String), день недели (int).	Выделяемое время в минутах (int)
3	Сотрудники повышают квалификацию на курсах 1 раз в квартал.	Журнал учета профес- сиональ- ного роста	Идентификационный номер сотрудника (int), шифр курсов (String), номер квартала (int).	Результат (диплом, свидетельств о, сертификат - String)
4	Автомобили арендуются фирмами для перевозок в определенном месяце	Журнал учета аренды	Идентификационный номер автомобиля (int), шифр фирмы (String), номер месяца (int).	Сумма договора (int)
5	Производится ежеквартальная проверка качества товаров	Журнал проверки качества товаров	Идентификационный номер товара (int), шифр инспектирующей фирмы (String), номер квартала (int).	Результат (String)
6	Инженеры дежурят в компьютерных залах по определенным числам	Журнал учета дежурств	Идентификационный номер инженера (int), шифр компьютерного класса, например, А-003 (String), дата (String).	Количество обнаруженны х неполадок (int)
7	В зависимости от формы обучения (очная, заочная, вечерняя) и	Паспорт специ- альности	Идентификационный код специальности (int), шифр формы	Срок обучения (int)

1	2	3	4	5
	ступени подготовки (среднее специальное образование, бакалавриат, магистратура) на каждой специальности различаются сроки обучения.		обучения (String), шифр ступени подготовки (String)	
8	Преподаватели дистанционно консультируют по дисциплинам в определенном семестре.	Учебный план	Идентификационный номер дисциплины (int), логин преподавателя (String), номер семестра (int).	Количество запланированных часов (int)
9	Преподаватель может поехать в командировку в определенный вуз раз в квартал	График командировок	Идентификационный номер преподавателя (int), шифр вуза (String), номер квартала (int).	Срок в днях (int)
10	Книги покупаются библиотекой у поставщиков раз в месяц в определенном количестве	Журнал закупок	Идентификационный номер книги (int), шифр поставщика (String), номер месяца (int).	Количество экземпляров (int)
11	Спортсмен один раз в месяц может участвовать в игре чужой команды (стажировка).	Журнал учета стажировок.	Идентификационный номер спортсмена (int), шифр команды (String), номер месяца (int).	Число забитых мячей (int)
12	Кинофильм участвует в кинофестивалях в определенных номинациях	Журнал учета результатов кинофестивалей	Идентификационный номер фильма (int), шифр кинофестиваля (String), шифр номинации (String).	Занятое место (int)

1	2	3	4	5
13	Детали поставлялись для проектов различными фирмами	Отчет о поставках	Идентификационный номер детали (int), шифр фирмы-поставщика (String), шифр проекта (String).	Количество деталей в поставке (int).
14	Спортивные секции участвуют в соревнованиях, которые проводятся в определенные дни (даты).	Журнал спортивных достижений.	Идентификационный номер секции (int), шифр соревнования (String), дата (String).	Количество побед, одержанных членами секции (int).
15	Сотрудники могут работать в нескольких отделах одновременно (совместительство). Ежемесячно они получают зарплату и премиальные.	Ведомость начисления зарплаты	Номер сотрудника (int), шифр отдела (String), месяц (String).	Сумма (float).
16	Турагентство предлагает путевки в разные страны.	Прайс-лист турагентства	Страна (String), количество дней (int), класс обслуживания (int).	Стоимость путевки (float).
17	Ежедневный учет прихода в банке	Ведомость прихода денег	шифр отправителя (номер карты) (String), шифр получателя (номер карты) (String), дата (String)	Сумма (float)
18	Оценки по предметам за семестр выставляются студентам группы по балльно-рейтинговой	Балльно-рейтинговая ведомость	Номер зачетной книжки (int), название предмета (String), номинация (например, посещаемость, изучение	Балл (int)

1	2	3	4	5
	системе		электронных лекций, выполнение лаб.работ, тестирование, реферат) (String).	
19	Ведение учета начислений за коммунальные услуги жилищно- коммунальным хозяйством (ЖКХ)	Журнал учета комму- нальных услуг	Код клиента (int), шифр услуги (String), месяц (String).	Сумма (float)
20	Фирмы участвуют в тендерах на оказание ремонтных услуг муниципальным предприятиям	Конкурс- ный журнал	Код фирмы (int), код предприятия(int), название услуги (String).	Сумма (float)
21	Учет динамики строительства в городе	Журнал прогресса строитель- ных работ	Код подрядчика (int), название объекта (String), дата (String).	Процент готовности (int)
22	Учет активности студента в системе дистанционного обучения	Журнал учета активно- сти студента	Аккаунт студента (String), название форума (String), семестр (int).	Число сообщений в обсуждениях (int)
23	Приобретение билета на поезд	Справо- чник цен на билеты	Номер поезда (int), тип вагона (String), класс обслуживания (int)	Цена билета (float)
24	Отдых на базах отдыха предприятия. Стоимость путевки отличается для отдыхающих с разным статусом (сотрудников, членов семей	Прейску- рант	Название базы отдыха (String), месяц (int), статус отдыхающего (int)	Цена путевки (float)

1	2	3	4	5
	сотрудников, сторонних отдыхающих) и зависит от месяца.			
25	Фирма по доставке грузов	Журнал учета длительности перевозок	Пункт отправки (String), пункт прибытия (String), вид транспорта (String).	Количество дней (int)
26	Калькуляция стоимости театральных билетов. Театр имеет несколько зданий в разных районах города. Каждое здание имеет две сцены – малую и большую. Спектакль имеет один из трех статусов: 1 – премьеры, 2 – до двух лет в репертуаре, 3 – более двух лет в репертуаре.	Прейскурант	Номер здания (int), тип сцены (String), статус спектакля (int).	Минимальная (базовая) цена билета (float)
27	Летний отдых в студенческом оздоровительном лагере «Горизонт» СевГУ. Путевки типа «Отдых выходного дня», «Двухнедельный отдых», «Полная смена». Отдыхать могут студенты,	Прейскурант	Код типа путевки (int), шифр типа отдыхающего (String), шифр типа жилья (String).	Цена путевки.

1	2	3	4	5
	преподаватели, посторонние граждане. Жилье обеспечивается в одноэтажных коттеджах или многоэтажном корпусе.			
28	Преподаватели (профессора, доценты, старшие преподаватели, ассистенты) работают на кафедрах университета и в филиалах.	Штатное расписание профессорско-преподавательского состава.	Шифр кафедры (String), номер филиала (у основного вуза – 0) (int), название должности (String)	Количество ставок (int)
29	Фирма осуществляет автоперевозки грузов. Для груза определяется граничный вес (например, 5 – груз весом до 5 кг) и уровень ценности (например, существует 5 уровней ценности груза (от 1(самый ценный) до 5)).	Прейскурант	Граничный вес (float), длительность доставки в днях (int), уровень ценности груза(int)	Стоимость доставки (float)
30	Калькуляция стоимости билетов в зрительный зал театра. В зале есть следующие локации: партер,	Прейскурант	Шифр локации (String), ряд (int), премьера?(boolean).	Цена билета (float)

1	2	3	4	5
	амфитеатр, балкон первого яруса, балкон второго яруса. Стоимость билетов в каждой локации зависит от ряда. Стоимость билетов зависит также от того, является ли спектакль премьерным.			

Перечень запросов – по аналогии с образцом.

Творческое задание: придумайте и запрограммируйте дополнительно один запрос на выборку данных, аналога которому нет в образце.

3.7. Содержание отчета

- 1) Цель работы.
- 2) Постановка задачи с указанием данных варианта задания.
- 3) Структура проекта.
- 4) Описание классов проекта (полей и методов, инкапсулированных в классах).
- 4) Текст программы с комментариями.
- 5) Протокол отладки и тестирования программы (с подробным описанием всех ошибок процесса выполнения, в том числе, вызванных некорректной работой со списками, множествами и картами отображений).
- 6) Результат выполнения программы.
- 7) Сравнительный анализ возможностей инструментария коллекций и карт отображений (по итогам двух лабораторных работ).
- 8) Вывод.

3.8. Контрольные вопросы

- 1) Что такое карта отображения?
- 2) Какие методы задает библиотечный интерфейс Map?
- 3) Какие библиотечные классы реализуют интерфейс Map?
- 4) Поведение каких объектов определяет интерфейс Map.Entry? Какие методы он содержит?

- 5) Что такое компаратор и для чего он используется?
- 6) В конструктор какого реализующего класса можно передать компаратор?
- 7) Как задать компаратор?
- 8) Как сделать сортировку объектов карты по неключевому полю? Какие «подводные камни» можно встретить на этом пути?
- 9) Как переделать карту в коллекцию? Какие коллекции можно получить из карты?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ноутон П. Java™ 2 / П. Ноутон, Г. Шилдт. – СПб. : БХВ-Петербург, 2008. – 1072 с. — Текст : непосредственный.
2. Шилдт Г. Java 8. Полное руководство/ Г. Шилдт. – М. : Вильямс, 2015 – 1376 с. — Текст : непосредственный.
3. Эккель Б. Философия Java / Б. Эккель. – СПб : Питер, 2015 – 1168 с. — Текст : непосредственный.
4. Java™ 2 Platform Standard Ed. 5.0. – URL : <https://docs.oracle.com/javase/1.5.0/docs/api/index.html>. (дата обращения 10.11.2022) — Текст : электронный.
5. Гуськова, О. И. Объектно ориентированное программирование в Java : учебное пособие / О. И. Гуськова. - Москва : МПГУ, 2018. - 240 с. - URL: <https://znanium.com/catalog/product/1020593> (дата обращения: 10.11.2022). – Режим доступа: по подписке. - ISBN 978-5-4263-0648-6. - Текст : электронный.
6. Пономарчук, Ю. В. Программирование на языке Java : учебное пособие / Ю. В. Пономарчук, И. В. Кузнецов. — Хабаровск : ДВГУПС, 2021. — 103 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/259451> (дата обращения: 09.11.2022). — Режим доступа: для авториз. пользователей

ПРОГРАММИРОВАНИЕ. ЯЗЫКИ ВЫСОКОГО УРОВНЯ

Методические указания

Часть 5

**Использование коллекций и карт отображений
при разработке java-программ**

Составители: **Волкова** Татьяна Викторовна,
Владимирова Елена Сергеевна

В авторской редакции

Изд. № 64/2023. Объем 5,5 п.л. Усл. печ. л. 5,11. Уч.-изд. л. 5,39.
РИИЦМ ФГАОУ ВО «Севастопольский государственный университет»