1. ПРАКТИЧЕСКАЯ РАБОТА № 1. ИССЛЕДОВАНИЕ

ВОЗМОЖНОСТЕЙ JAVA COLLECTION FRAMEWORK (JCF).

СПИСКИ И МНОЖЕСТВА (НАБОРЫ)

**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. Постановка задачи**

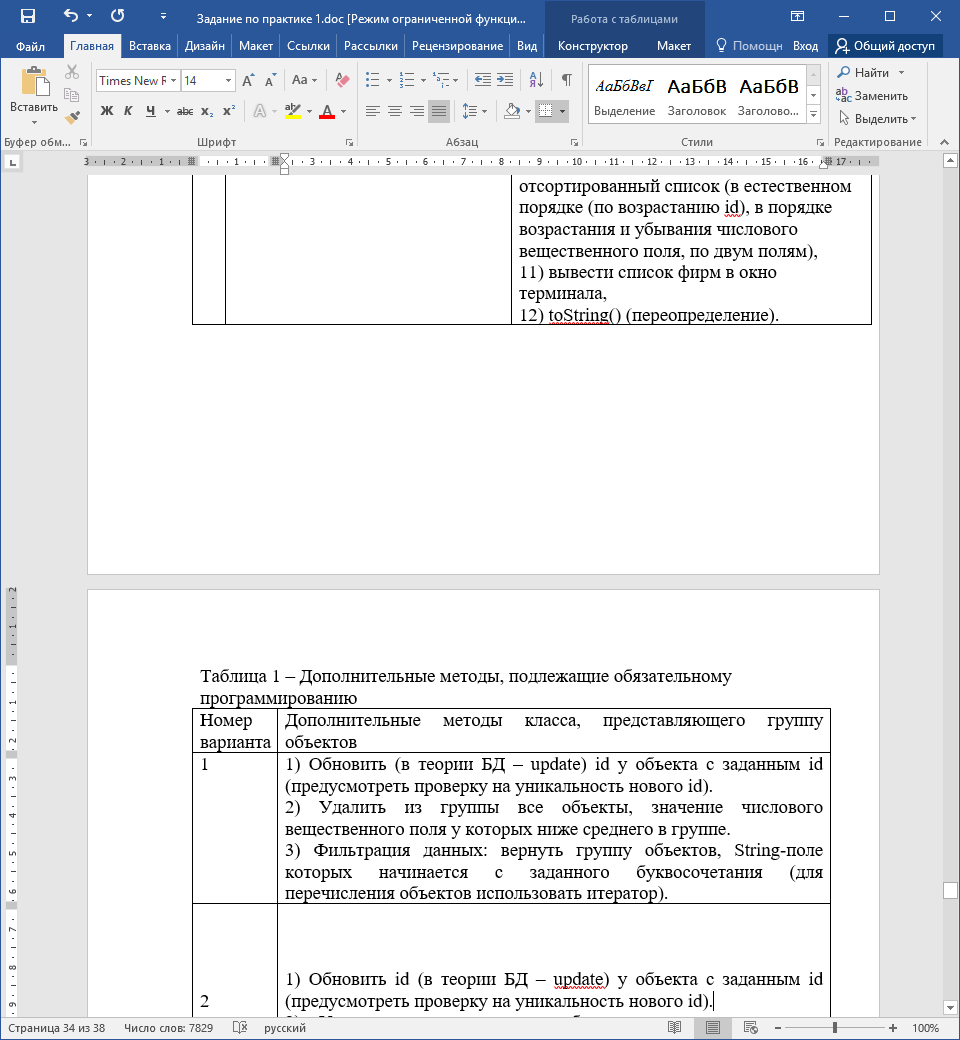
1) Разработать программу, реализующую взаимодействие объектов, образующих иерархическую структуру данных (группу объектов с заданными свойствами и функциями). Для создания объектов, определенных вариантом задания разработать соответствующие классы. Для манипулирования группой объектов использовать возможности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

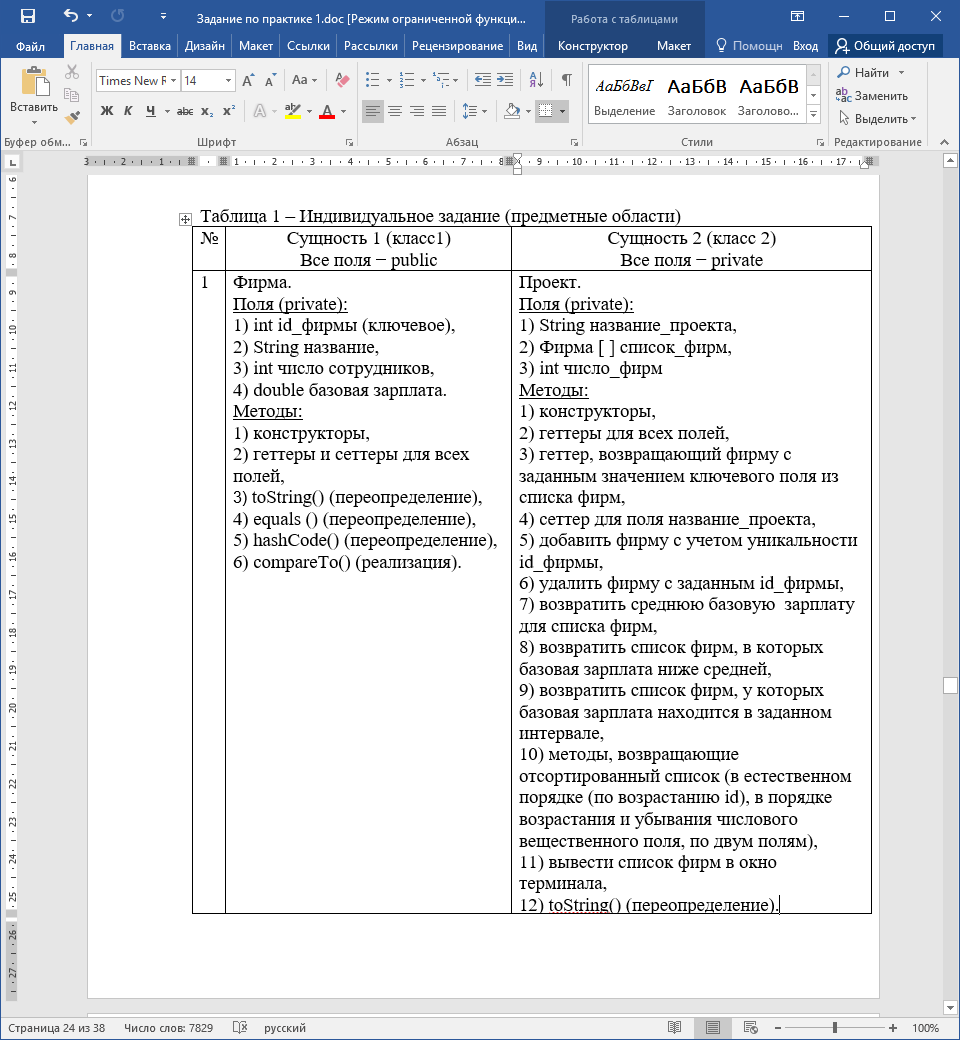
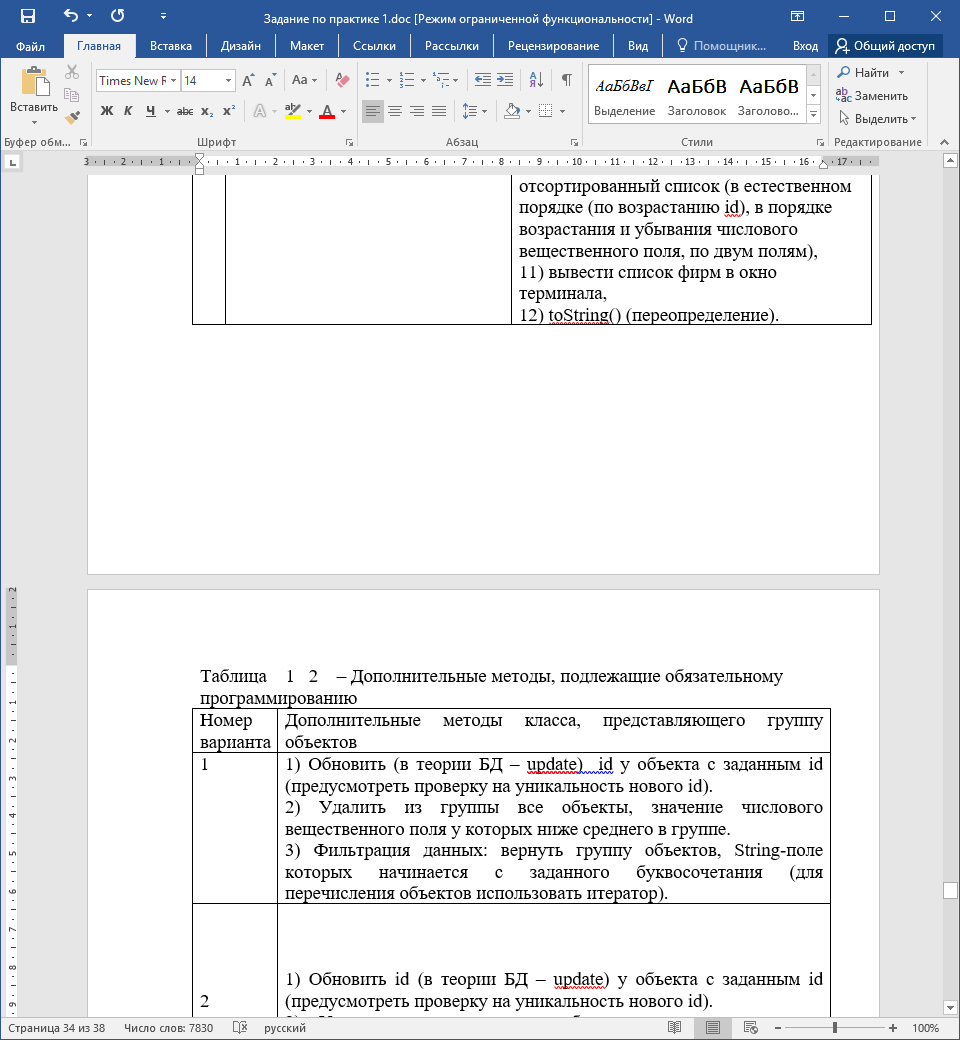
2) Разработать программу, реализующую тот же алгоритм взаимодействия объектов. Для манипулирования группой объектов использовать возможности интерфейса Set.

3) Сравнить две реализации группы объектов.

Индивидуальное задание (вариант №1): представлено в таблице 2.

Дополнительные методы: представлены в таблице 1.





**3/4. Структуры проектов. Описание классов для каждого проекта (полей и методов, инкапсулированных в классе).**

Структура проекта, реализующего взаимодействие экземпляров сущностей «Фирма» и «Проект» изображена на рисунке 1.

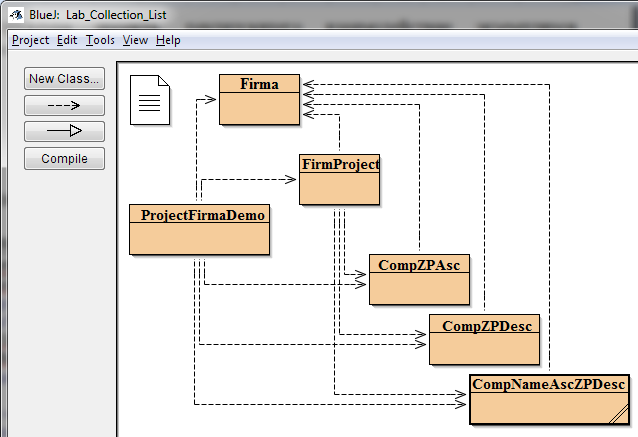


Рисунок 1 – Структура проекта, обеспечивающего

взаимодействие экземпляров фирм и проектов,

использующего концепцию списка (List)

Код класса Фирма проектов (**FirmProject**) также изменен с учетом того, что список фирм в нем реализуется на основе инструмента коллекций. Кроме того, в класс добавлены новые возможности поведения (методы) фирм проекта.

Определение полей класса **FirmProject**:

1) **private String name; *//название проекта***

2) **private List <** **Firma > firms; *// список (набор) фирм***

Перечень возможностей (методов) класса **FirmProject**:

конструкторы:

1) сконструировать пустой проект фирм без названия (конструктор без параметров);

2) cконструировать Проект фирм с заданным названием и пустым списком фирм;

3) сконструировать Проект фирм с заданным названием на основе готового списка фирм;

геттеры для private-полей:

1. вернуть название Проекта фирм – **getProjectName**;
2. вернуть Проект фирм (всех фирм) – **getFirmSet**;

сеттер для private-поля:

1. изменить название Проекта фирм – **setProjectName**;

формирование строки описания объекта Проект фирм:

7) **toString;**

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

1. добавить фирму в Проект фирм – **addFirm**;
2. удалить фирму из Проекта фирм – **delFirm**;

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

1. вернуть фирму с заданным номером ID – **getFirm**;
2. вернуть количество фирм в проекте – **getFirmNum**;
3. вернуть среднюю зарплату для проекта – **avgZP**;
4. вернуть фирмы, у которых зарплата выше среднего для проекта – **aboveAvgZP;**
5. вернуть фирмы, у которых зарплата находится в заданном диапазоне значений – **betweenZP;**

запросы на сортировку данных:

1. вернуть фирмы, отсортированные в естественном порядке – **sort();**
2. вернуть фирмы, отсортированные по одному из нижеперечисленных правил, задаваемых соответствующими объектами, реализующими интерфейс Comparator, –

**sort (Comparator comp):**

a) по зарплате в порядке возрастания – компаратор **CompBallAsc**;

б) по зарплате в порядке убывания – компаратор **CompBallDesc**;

в) по имени в порядке возрастания, внутри одинаковых названий по зарплате в порядке убывания – компаратор **CompNameAscZPDesc**;

1. вывести Проект фирм в окно терминала – **putFirmProject**.

**Заметим, что в списках (List) или наборах (Set) хранятся не сами объекты (в данном примере − объекты класса Firma), а ссылки на них, поэтому создание новых списков студентов (в методах, возвращающих тип Группа студентов) не увеличивает количество самих объектов класса Firma. Другими словами, если мы создаем список на базе другого списка, то в этих списках будут храниться ссылки на одни и те же объекты.**

Требуемые постановкой задачи правила сортировки задаются тремя классами, реализующими интерфейс Comparator: **CompZPAsc, CompZPDesc, CompNameAscZPDesc.**

Класс **ProjectFirmaDemo** включает в себя метод main(), который является стартовой точкой программы. В методе main() создаются 10 экземпляров фирм. Значения полей для фирм 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 фирм, в методе main() создается два экземпляра класса Project (два проекта содержащих список фирм).

Затем делается попытка добавить в проект pr1 фирму c уже существующим в проекте номером ID. Такая попытка заканчивается неудачей, что мы и наблюдаем при выводе соответствующих списков.

Для каждого проекта выводится полный список фирм, средняя зарплата по фирме, список фирм, у которых зарплата выше среднего, список фирм, у которых зарплата находится в заданном диапазоне. Все выводимые списки заданным образом упорядочиваются.

Из первого проекта исключается фирма, после чего (для контроля) выводится еще раз список группы.

Осуществляются проверки на наличие в проектах фирм с заданными номерами ID и выдаются соответствующие сообщения.

Cоздание и манипулирование группой объектов на основе интерфейса Set.

Структура проекта изображена на рисунке 2. Данный проект состоит из тех же классов, что и предыдущий.

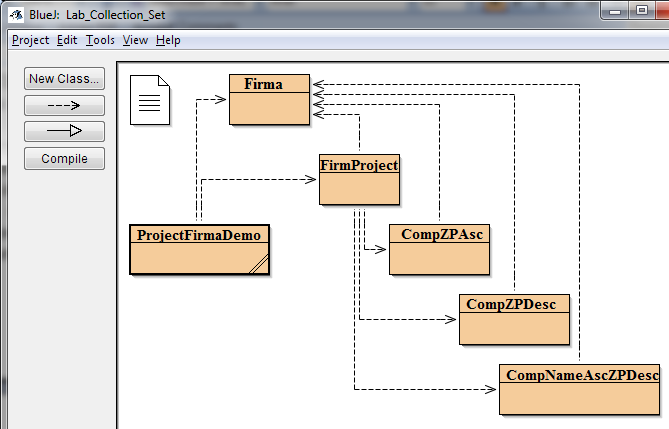


Рисунок 2 – Структура проекта, обеспечивающего

взаимодействие экземпляров фирм и проектов,

использующего концепцию множества (Set)

Учитывая, что каждая фирма имеет уникальный номер (ключевое свойство), двух одинаковых экземпляров класса Firma быть не может, в данном задании Проект фирм реализован 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 Firma.

Естественный порядок перечисления объектов в упорядоченном наборе TreeSet задает метод CompareTo, реализованный в классе Firma. Все методы класса FirmaProject, 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.

**5. Тексты программ**

**Реализация на основе списка (List):**

**public class ProjectFirmaDemo { // Взаимодействие фирм и проектов**

public static void main(String[] args) {

// Создаем проекты фирм

FirmProject pr1 = new FirmProject("Festo");

FirmProject pr2 = new FirmProject("Kepler");

// в массив args при запуске метода main (String[ ] args)

// ввели 10 фирм: по 5 для каждой группы.

int j = 0; // индекс массива args

// добавление фирм 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++;

int rab = Integer.valueOf(args[j]);

j++;

// получаем double-эквивалент из String

double ZP = Double.valueOf(args[j]);

j++;

// добавить фирм

pr1.addFirm(new Firma(id, name, rab, ZP));

}

// добавление фирм во вторую группу

for (int i = 0; i < 5; i++) {

int id = Integer.valueOf(args[j]);

j++;

String name = args[j];

j++;

int rab = Integer.valueOf(args[j]);

j++;

double ZP = Double.valueOf(args[j]);

j++;

// добавить фирм

pr2.addFirm(new Firma(id, name, rab, ZP));

}

// Выборка данных

// Выводим всевозможные списки фирм

// Выводим основной список без сортировки

System.out.println("Список фирм участвующих в проекте (без сортировки)");

pr1.putFirmProject();

// пытаемся вставить фирм с тем же номером id

System.out.println("Попытка добавить фирму: 3234, Скворцово, 76, 5000.0");

pr1.addFirm(new Firma(3234, "Скворцово", 76, 5000.0f));

// снова выводим основной список

// (теперь с естественным порядком сортировки)

System.out.println("Список фирм участвующих в проекте (с естественным порядком сортировки)");

pr1.sort().putFirmProject();

// другие списки:

System.out.println("Список фирм участвующих в проекте (с сортировкой по возрастанию зарплаты)");

pr1.aboveAvgZP().sort(new CompZPAsc()).putFirmProject();

System.out.println("Список фирм участвующих в проекте (с сортировкой по убыванию зарплаты)");

pr1.betweenZP(3000f, 6000f).sort(new CompZPDesc()).putFirmProject();

// удаление фирм по id

pr1.delFirm(3234);

// выводим список группы pr1 после удаления фирм

System.out.println("После удаления фирмы c id=3234:");

System.out.println("Список фирм участвующих в проекте (с естественным порядком сортировки)");

pr1.putFirmProject();

// Выводим списки фирм для второй группы

System.out.println("Список фирм участвующих в проекте (с сортировкой по возрастанию названия и");

System.out.println(" убыванию зарплаты)");

pr2.sort(new CompNameAscZPDesc()).putFirmProject();

// другие списки для второй группы:

System.out.println("Список фирм участвующих в проекте (с сортировкой по возрастанию зарплаты)");

pr2.aboveAvgZP().sort(new CompZPAsc()).putFirmProject();

System.out.println("Список фирм участвующих в проекте (с сортировкой по убыванию зарплаты)");

pr2.betweenZP(30f, 8000f).sort(new CompZPDesc()).putFirmProject();

// Проверяем,есть ли во первой проекте фирма

// с заданным номером id

int n = 34;

Firma s1 = pr1.getFirm(n);

if (s1 == null)

System.out.println("В проекте " + pr1.getProjectName() + " нет фирмы с номером " + n);

else

System.out.println(s1);

n = 354;

s1 = pr1.getFirm(n);

if (s1 == null)

System.out.println("В проекте " + pr1.getProjectName() + " нет фирмы с номером " + n);

else

System.out.println(s1);

// Проверяем,есть ли во второй проекте фирма с заданным

// номером id

n = 53349;

s1 = pr2.getFirm(n);

if (s1 == null)

System.out.println("В проекте " + pr1.getProjectName() + " нет фирмы с номером " + n);

else

System.out.println(s1);

pr1.UpdateId(34, 24); //1й метод индивидуального задания

pr1.Filtr("Бе").putFirmProject();//3й метод индивидуального задания

pr1.DelObject().putFirmProject();//2й метод индивидуального задания

}

}

**import java.util.\*;**

**public class FirmProject { // Сущность – Проект фирмы**

// поля (скрыты в классе)

private final static String PROJECT\_FORMAT\_STRING = "Проект: %-s, %-5d фирмы"; // формат записи о проекте

private String name; // название проекта

private List<Firma> firms; // список (набор) фирм

// конструкторы

public FirmProject() {

name = ""; // без названия

firms = new ArrayList<Firma>(); // создается пустой список

}

public FirmProject(String name) {

this.name = name; // задается название проекта

firms = new ArrayList<Firma>(); // создается пустой список

}

public FirmProject(String name, List list) {

this.name = name; // задается название проекта

firms = new ArrayList<Firma>(list); // создается на основе

// существующего списка

}

// метод-сеттер для private-поля

public void setProjectName(String name) {

this.name = name;

}

// методы-геттеры для private-полей

public String getProjectName() {

return name;

}

public List<Firma> getFirm() {

return firms;

}

// Переопределяем метод toString класса Object

// возвращает строку описания объекта проект фирмы:

public String toString() {

return String.format(PROJECT\_FORMAT\_STRING, name, getFirmNum());

}

// Запросы на вставку, удаление, изменение данных:

public boolean addFirm(Firma firm) {

// Добавить фирму в список проектов

// (фирму нельзя добавить, если в проекте уже есть фирма

// с таким же id)

if (getFirm(firm.getId()) != null)

return false; // дополнительная

// программная проверка

if (firms.add(firm))

return true;

else

return false;

}

public boolean delFirm(int id) {

// Удалить фирму из списка проекта

if (firms.remove(new Firma(id, "", 0, 0)))

return true;

else

return false;

}

// запросы на выборку данных:

public Firma getFirm(int id) {

// Возвращает фирму с заданным номером (id):

for (Firma firm : firms)

if (firm.getId() == id)

return firm; // если фирма найдена

return null; // фирма не найдена

}

public int getFirmNum() {

// Возвращает число фирм в проекте

return firms.size();

}

public double avgZP() {

// Возвращает среднюю заработную плату по проекту

int num = firms.size();

if (num == 0)

return 0;

double avg = 0;

for (Firma firm : firms)

avg = avg + firm.getZP();

return avg / num;

}

public FirmProject aboveAvgZP() {

// Вернуть фирмы, у которых зарплата выше среднего

double avg = avgZP();

FirmProject project = new FirmProject(name + ": фирмы, у которых зарплата ниже 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 (Firma firm : firms)

if (firm.getZP() < avg)

project.addFirm(firm);

return project;

}

public FirmProject betweenZP(double b1, double b2) {

// Вернуть фирмы с зарплатой в диапазоне [b1,b2]

// Избавляемся от конкатенации при формировании имени проекта

FirmProject project = new FirmProject(

String.format("%s: фирмы, у которых зарплата в диапазоне от %4.2f до %4.2f", name, b1, b2));

// для просмотра (перечисления) объектов списка

// используем итератор

Iterator<Firma> iter = firms.iterator();

while (iter.hasNext()) {

Firma firm = iter.next();

if ((firm.getZP() >= b1) && (firm.getZP() <= b2))

project.addFirm(firm);

}

return project;

}

// Запросы на сортировку данных

// Для сортировки списков List используются статические

// методы sort, определенные в классе Collections

public FirmProject sort() {

// Cортировка фирм в естественном порядке

// Естественный порядок задает метод CompareTo,

// переопределенный в классе Firma

FirmProject project = new FirmProject(name, firms);

Collections.sort(project.firms);

return project;

}

public FirmProject sort(Comparator comp) { // coртировка фирм

// по правилу, задаваемому компаратором comp

FirmProject project = new FirmProject(name, firms);

Collections.sort(project.firms, comp);

return project;

}

// Запрос на вывод данных

public void putFirmProject() {

// Вывод проекта фирм в окно терминала

System.out.println(name); // имя проекта

System.out.printf("%3s%11s%23s%21s%22s\n", "№", "ID", "Название", "Число сотрудников", "Базовая зарплата"); // заголовки

// столбцов

int i = 1;

for (Firma firm : firms) {

System.out.printf(" %-7d %-15d %-18s %-18d %-10.2f\n", i, firm.getId(), firm.getName(), firm.getRab(), firm.getZP());

i = i + 1;

}

}

//Обновить (в теории БД – update) id у объекта с заданным id (предусмотреть проверку на уникальность нового id)

public boolean UpdateId(int id1, int newId){ // 1й метод, обновление Id

if (getFirm(id1)!=null && !firms.contains(new Firma(newId,"",0,0))){

getFirm(id1).setId(newId);

System.out.println("Id "+id1+" был обновлён на "+newId);

return true;

}

else{System.out.println("Id не был обновлён"); return false; }

}

//2й метод, удалить фирмы у которых значение числового вещественного поля ниже среднего

public FirmProject DelObject(){

double avg = avgZP();// Возвращает среднюю заработную плату по проекту

FirmProject project = new FirmProject(name + ": фирмы, у которых зарплата выше среднего среднего: " + avg+" остальные удаляем");

for (Firma firm : firms){

project.addFirm(firm);

if (firm.getZP() < avg){

int c = firm.getId();

project.delFirm(c);

}

}

return project;

}

public FirmProject Filtr (String str){ //3й метод, вывод слов начинающихся заданным буквосочетанием

FirmProject project = new FirmProject (

String.format ("%s: Фирмы начинающиеся заданным буквосочетанием: %s", name, str));

Iterator <Firma> iter=firms.iterator();

while (iter.hasNext()){

Firma firm = iter.next ();

if (firm.getName().indexOf(str)==0){

project.addFirm(firm);

}

}

return project;

}

} // class

**public class Firma implements Comparable<Firma> {**

**// Сущность - Фирма**

// поля - скрыты в классе

private final static String FIRM\_FORMAT\_STRING = "Фирма: %-7d | %-10s | %-7d | %-8.2f |"; // формат записи о фирме

private int id; // id фирмы, ключевое поле

private String name; // название

private int rab; // число\_сотрудников

private double zp; // базовая\_зарплата

// конструктор без параметров

public Firma() {

this.id = 0;

this.name = "";

this.rab = 0;

this.zp = 0;

}

// конструктор с параметрами

public Firma(int id, String name, int rab, double zp) {

this.id = id;

this.name = name;

this.rab = rab;

this.zp = zp;

}

// методы-геттеры

public int getId() {

return id;

}

public String getName() {

return name;

}

public int getRab() {

return rab;

}

public double getZP() {

return zp;

}

// методы-сеттеры

public void setId(int id) {

this.id = id;

}

public void setName(String name) {

this.name = name;

}

public void setRab(int rab) {

this.rab = rab;

}

public void setZP(double zp) {

this.zp = zp;

}

// Переопределяется метод toString класса Object

// (возвращает строку описания объекта)

public String toString() {

return String.format(FIRM\_FORMAT\_STRING, id, name, rab, zp);

}

// 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

Firma fr = (Firma) ob; // преобразуем Object в Firma

return id == fr.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 СоmpareTo интерфейса Сomporable

// Для определения естественного порядка перечисления элементов

public int compareTo(Firma firm) {

if (id < firm.id)

return -1;

else if (id == firm.id)

return 0;

else

return 1;

}

}

**import java.util.\*;**

**public class CompZPDesc implements Comparator {**

public int compare(Object ob1, Object ob2) {

Firma firm1 = (Firma) ob1;

Firma firm2 = (Firma) ob2;

if (firm1.getZP() < firm2.getZP())

return 1;

else if (firm1.getZP() == firm2.getZP())

return 0;

else

return -1;

}

// нет необходимости переопределять метод equels

}

**import java.util.\*;**

**public class CompZPAsc implements Comparator {**

public int compare(Object ob1, Object ob2) {

Firma firm1 = (Firma) ob1;

Firma firm2 = (Firma) ob2;

if (firm1.getZP() < firm2.getZP())

return -1;

else if (firm1.getZP() == firm2.getZP())

return 0;

else

return 1;

}

// нет необходимости переопределять метод equels

}

**import java.util.\*;**

**public class CompNameAscZPDesc implements Comparator{**

public int compare (Object ob1, Object ob2){

Firma firm1 = (Firma) ob1;

Firma firm2 = (Firma) ob2;

if (firm1.getName().compareTo(firm2.getName())<0) return -1;

else if (firm1.getName().compareTo(firm2.getName())> 0) return 1;

//фамилии одинаковые

else if (firm1.getZP() < firm2.getZP()) return 1;

else if (firm1.getZP() == firm2.getZP()) return 0;

else return -1;

}

//нет необходимости переопределять метод equels

}

**Реализация на основе набора объектов (Set):**

**public class ProjectFirmaDemo { // Взаимодействие фирм и проектов**

**public static void main(String[] args) {**

// Создаем проекты фирм

FirmProject pr1 = new FirmProject("Festo");

FirmProject pr2 = new FirmProject("Kepler");

// в массив args при запуске метода main (String[ ] args)

// ввели 10 фирм: по 5 для каждой группы.

int j = 0; // индекс массива args

// добавление фирм в первый проект

for (int i = 0; i < 5; i++) {

int id = Integer.valueOf(args[j]);

j++; // получен int-эквивалент из String

String name = args[j];

j++;

int rab = Integer.valueOf(args[j]);

j++;

// получаем double-эквивалент из String

double ZP = Double.valueOf(args[j]);

j++;

// добавить фирм

pr1.addFirm(new Firma(id, name, rab, ZP));

}

// добавление фирм во второй проект

for (int i = 0; i < 5; i++) {

int id = Integer.valueOf(args[j]);

j++;

String name = args[j];

j++;

int rab = Integer.valueOf(args[j]);

j++;

double ZP = Double.valueOf(args[j]);

j++;

// добавить фирм

pr2.addFirm(new Firma(id, name, rab, ZP));

}

// Выборка данных

// Выводим всевозможные списки фирм

// Выводим основной список без сортировки

System.out.println("Список фирм участвующих в проекте (без сортировки)");

pr1.putFirmProject();

// пытаемся вставить фирм с тем же номером id

System.out.println("Попытка добавить фирму: 3234, Скворцово, 76, 5000.0");

pr1.addFirm(new Firma(3234, "Скворцово", 76, 5000.0f));

// снова выводим основной список

// (теперь с естественным порядком сортировки)

System.out.println("Список фирм участвующих в проекте (с естественным порядком сортировки)");

pr1.putFirmProject();

// другие списки:

pr1.aboveAvgZP().sortZPAsc().putFirmProject();

pr1.betweenZP(3.2f,4.2f).sortZPDesc().putFirmProject();

// удаление студента по id

pr1.delFirm(12345);

//выводим список группы pr1 после удаления студента

System.out.println ("После удаления студента c id=12345:");

pr1.putFirmProject();

// выводим списки студентов для второй группы

// основной список группы

pr2.putFirmProject(); // без сортировки

pr2.sortNameAscZPDesc().putFirmProject(); // с сортировкой по

// возрастанию фамилии и убыванию балла

// другие списки:

pr2.aboveAvgZP().sortZPAsc().putFirmProject();

pr2.betweenZP(3.2f,4.2f).sortZPDesc().putFirmProject();

//Проверяем,есть ли во первой группе студент

// c заданным номером зачетки

int n=23345;

Firma s1=pr1.getFirm(n);

if (s1==null) System.out.println ("В группе "+pr1.getProjectName()+

" нет студента с номером зачетки "+n);

else System.out.println(s1);

n=70000;

s1=pr1.getFirm(n);

if (s1==null) System.out.println ("В группе "+pr1.getProjectName()+

" 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=pr2.getFirm(n);

if (s1==null) System.out.println ("В группе "+pr1.getProjectName()+

" нет студента с номером зачетки "+n);

else System.out.println(s1);

pr1.UpdateId(34, 24); //1й метод индивидуального задания

pr1.Filtr("Бе").putFirmProject();//3й метод индивидуального задания

pr1.DelObject().putFirmProject();//2й метод индивидуального задания

} }

**import java.util.\*;**

**public class FirmProject { // Сущность - Проект фирмы**

//поля

private final static String project\_FORMAT\_STRING =

"Проект: %-s, %-5d фирмы"; //формат записи о проекте

private String name; //название проекта

private Set <Firma> firms; // список (набор) фирм

// конструкторы

public FirmProject(){

name=""; //без названия

firms = new TreeSet <Firma>(); //создается пустой набор

}

public FirmProject(String name){

this.name=name; //задается название проекта

firms = new TreeSet <Firma>(); //создается пустой набор

}

public FirmProject(String name, Comparator comp){

this.name=name; //задается название проекта

firms = new TreeSet <Firma>(comp); //создается пустой набор,

// в котором ссылки на добавляемые объекты будут

// упорядочиваться по правилам компаратора comp

}

public FirmProject(String name, Set set){

this.name=name; //задается название проекта

firms = new TreeSet <Firma>(set); //создается набор на основе

// существующего набора set

}

//метод-сеттер для private-поля

public void setProjectName(String name)

{this.name=name;}

//методы-геттеры для private-полей

public String getProjectName(){return name;}

public Set <Firma> getFirms(){

return firms;

}

//Переопределяем метод toString класса Object

//возвращает строку описания объекта Группа фирм:

public String toString(){

return String.format(project\_FORMAT\_STRING,name,getFirmNum());

}

//Запросы на вставку и удаление данных:

public boolean addFirm(Firma firm){

//Добавить фирму в список проекта

//фирма не будет вставлен, если в наборе уже есть фирма

//таким же id

if (firms.add(firm)) return true;

else return false;

}

public boolean delFirm(int id){

//Удалить фирму из списка проекта

if (firms.remove(new Firma(id,"",0,0))) return true;

else return false;

}

// Запросы на выборку данных:

public Firma getFirm (int id){

// возвращает фирму с заданным номером зачетки (id):

for (Firma firm:firms)

if (firm.getId() == id) return firm; // если фирма найден

return null; // фирма не найден

}

public int getFirmNum(){

//Возвращает число фирм в группе

return firms.size();

}

public double avgZP(){

//Возвращвет средний зарплата по группе

int num=firms.size();

if (num==0) return 0;

double avg=0;

for (Firma firm:firms)

avg=avg+firm.getZP();

return avg/num;

}

public FirmProject aboveAvgZP(){

//Возвращает фирм, у которых зарплата выше среднего

double avg=avgZP();

FirmProject project = new FirmProject (name+

": Фирмы, у которых зарплата 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 (Firma firm:firms)

if (firm.getZP()>avg) project.addFirm(firm);

return project;

}

public FirmProject betweenZP(double b1, double b2){// запрос:

//Возвращает фирм с зарплатаом в диапазоне [b1,b2]

//Избавляемся от конкатенации при формировании имени проекта

FirmProject project = new FirmProject (

String.format("%s: фирмы, у которых зарплата в диапазоне от %4.2f до %4.2f", name, b1, b2));

// для просмотра (перечисления) объектов списка

// используем итератор

Iterator <Firma> iter=firms.iterator();

while (iter.hasNext()){

Firma firm=iter.next();

if ((firm.getZP()>=b1)&&(firm.getZP()<=b2))project.addFirm(firm);

}

return project;

}

//запросы на сортировку данных

public FirmProject sortZPAsc(){

FirmProject project = new FirmProject(name+

" (сортировка по возрастанию зарплаты)", new CompZPAsc());

for (Firma firm:firms) project.addFirm(firm); // при добавлении

// в набор производится упорядочивание ссылок на объекты по

// правилу, задаваемому компаратором (сортировка)

return project;

}

public FirmProject sortZPDesc(){

FirmProject project = new FirmProject(name+

" (сортировка по убыванию зарплатаа)", new CompZPDesc());

for (Firma firm:firms) project.addFirm(firm); // при добавлении

// в набор производится упорядочивание ссылок на объекты по

// правилу, задаваемому компаратором (сортировка)

return project;

}

public FirmProject sortNameAscZPDesc (){

FirmProject project = new FirmProject(name+

" (сортировка по возрастанию имени и убыванию зарплаты)",

new CompNameAscZPDesc());

for (Firma firm:firms) project.addFirm(firm); // при добавлении

// в набор производится упорядочивание ссылок на объекты по

// правилу, задаваемому компаратором (сортировка)

return project;

}

// Запрос на вывод данных

public void putFirmProject(){

//Вывод проекта фирм в окно терминала

System.out.println(name); // имя проекта

System.out.printf("%3s%11s%23s%21s%22s\n", "№", "ID", "Название", "Число сотрудников", "Базовая зарплата"); // заголовки

int i = 1;

for (Firma firm : firms) {

System.out.printf(" %-7d %-15d %-18s %-18d %-10.2f\n", i, firm.getId(), firm.getName(), firm.getRab(), firm.getZP());

i = i + 1;

}

} // putFirmProject

//Обновить (в теории БД – update) id у объекта с заданным id (предусмотреть проверку на уникальность нового id)

public boolean UpdateId(int id1, int newId){ // 1й метод, обновление Id

if (getFirm(id1)!=null && !firms.contains(new Firma(newId,"",0,0))){

getFirm(id1).setId(newId);

System.out.println("Id "+id1+" был обновлён на "+newId);

return true;

}

else{System.out.println("Id не был обновлён"); return false; }

}

//2й метод, удалить фирмы у которых значение числового вещественного поля ниже среднего

public FirmProject DelObject(){

double avg = avgZP();// Возвращает среднюю заработную плату по проекту

FirmProject project = new FirmProject(name + ": фирмы, у которых зарплата выше среднего среднего: " + avg+" остальные удаляем");

for (Firma firm : firms){

project.addFirm(firm);

if (firm.getZP() < avg){

int c = firm.getId();

project.delFirm(c);

}

}

return project;

}

public FirmProject Filtr (String str){ //3й метод, вывод слов начинающихся заданным буквосочетанием

FirmProject project = new FirmProject (

String.format ("%s: Фирмы начинающиеся заданным буквосочетанием: %s", name, str));

Iterator <Firma> iter=firms.iterator();

while (iter.hasNext()){

Firma firm = iter.next ();

if (firm.getName().indexOf(str)==0){

project.addFirm(firm);

}

}

return project;

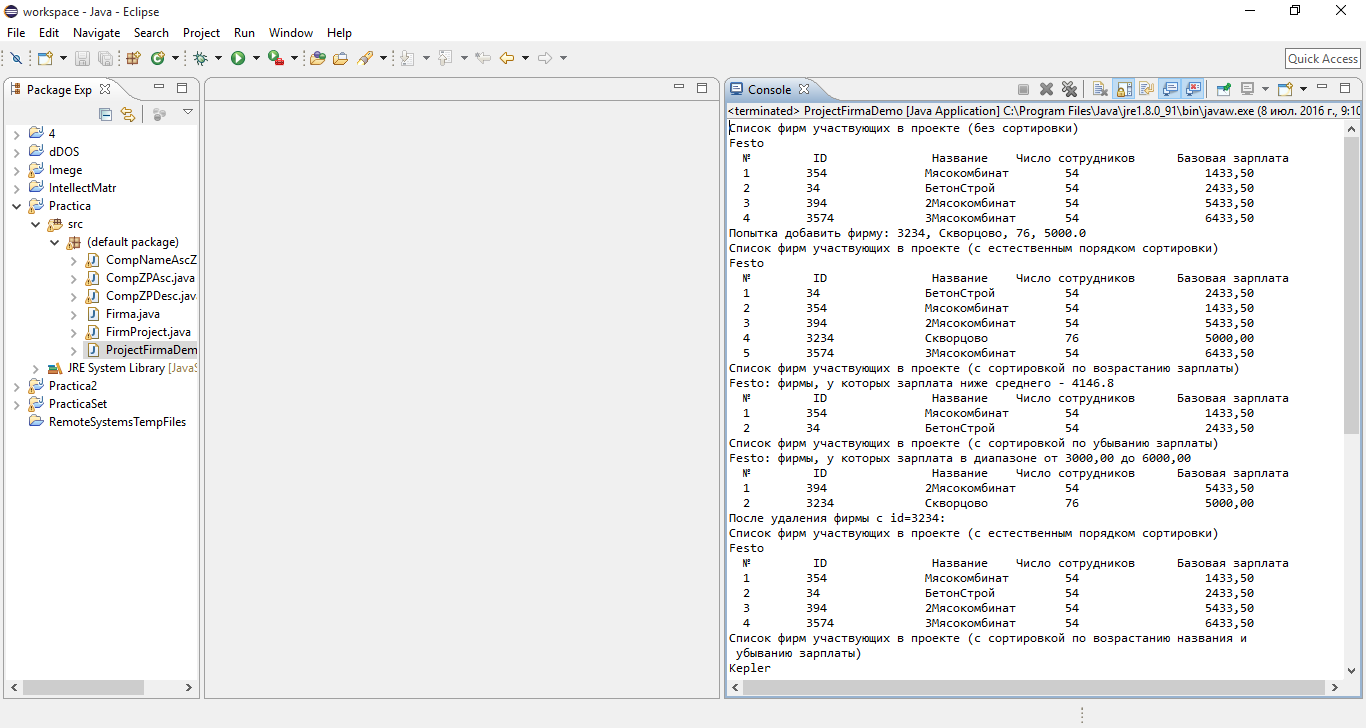
}

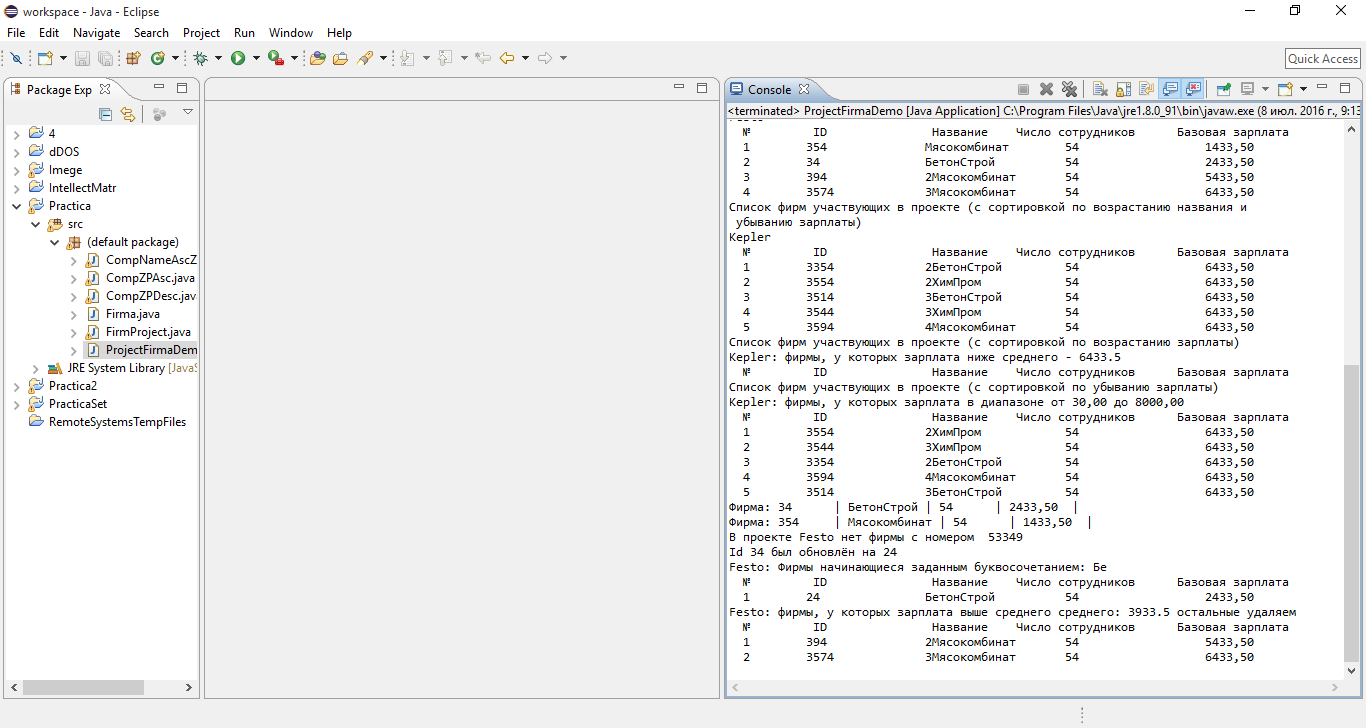
} //class

**6. Протоколы отладки и тестирования программ**

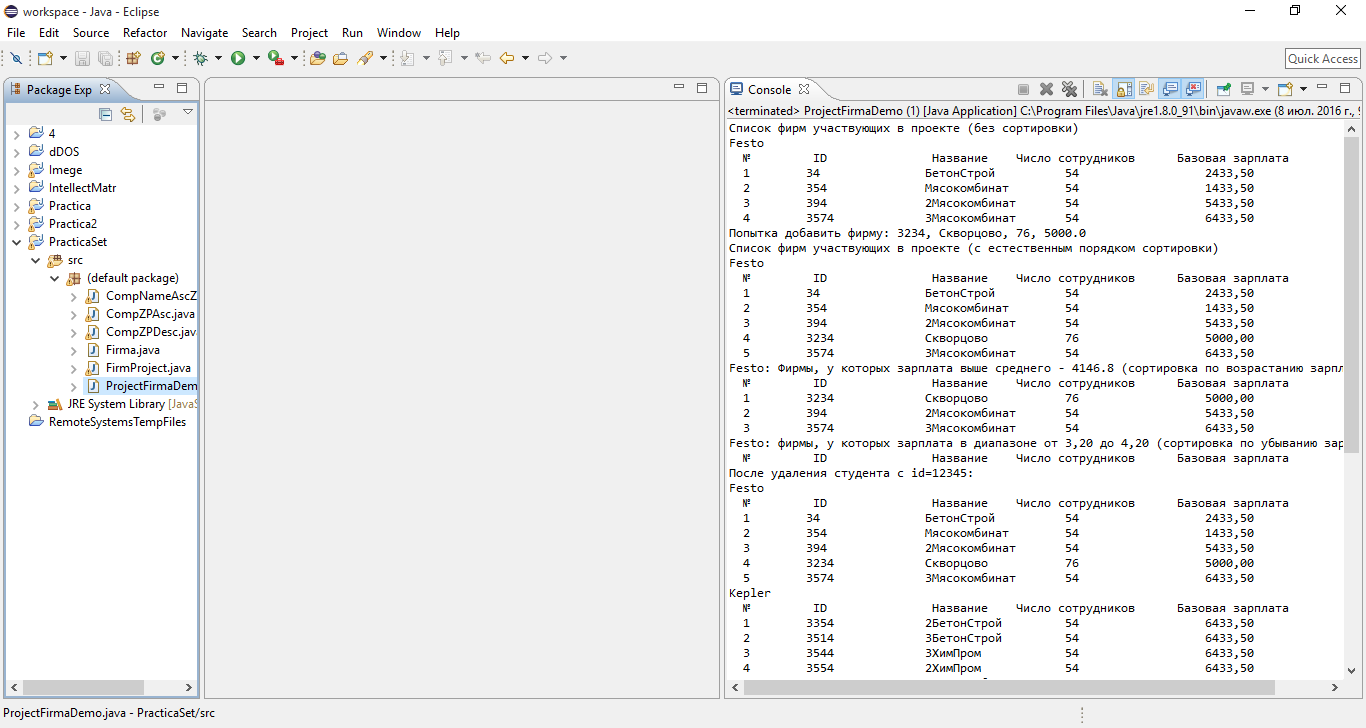
**7. Результаты работы программ**

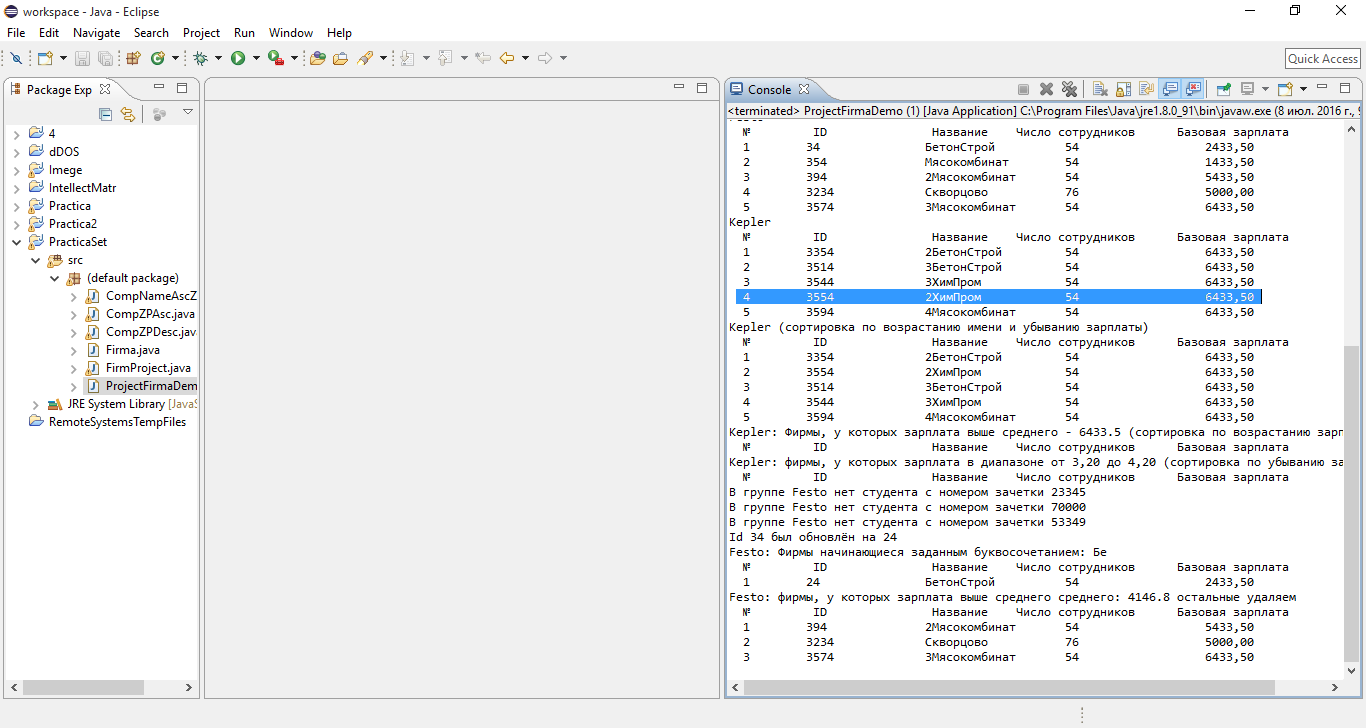
**Результат работы программы на основе списка (List):**





**Результат работы программы на основе набора объектов (Set):**





**8. Результаты сравнения программ (List/Set)**

Оба унаследованы от Collection, а значит имеют одинаковый набор и сигнатуры методов. List хранит объекты в порядке вставки, элемент можно получить по индексу. Set не может хранить одинаковых элементов.

**9. Вывод**

В ходе выполнения практического задания №1, были получены навыки использования набора классов и интерфейсов JAVA COLLECTION FRAMEWORK для создания коллекций (списков или множеств) объектов.