

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

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

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
И УПРАВЛЕНИЯ В ТЕХНИЧЕСКИХ СИСТЕМАХ**

Кафедра информационных технологий и компьютерных систем

**ПРОГРАММИРОВАНИЕ.
ЯЗЫКИ ВЫСОКОГО УРОВНЯ**

Часть 2

**Создание списков. Использование рекурсии при определении данных.
Реализация принципа полиморфизма с помощью интерфейсов Java**

Методические указания
к лабораторным работам по дисциплине
для студентов направлений
09.03.01 – Информатика и вычислительная техника и
09.03.04 – «Программная инженерия» дневной формы обучения

**Севастополь
СевГУ
2020**

УДК 004.42/.432(076.5)
ББК 32.973-018.1я7
П784

Рецензент:

Ю.В. Доронина - докт. техн. наук, профессор кафедры информационных систем

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

П784 Программирование. Языки высокого уровня. [В 3-х частях]. Ч. 2. Создание списков. Использование рекурсии при определении данных. Реализация принципа полиморфизма с помощью интерфейсов Java : методические указания к лабораторным работам по дисциплине «Программирование. Языки высокого уровня» для студентов направлений 09.03.01 – Информатика и вычислительная техника и 09.03.04 – Программная инженерия дневной формы обучения / Сост. Т.В. Волкова, Е.С. Владимирова ; Министерство образования и науки Российской Федерации, Севастопольский государственный университет, Институт информационных технологий и управления в технических системах, кафедра информационных технологий и компьютерных систем. – Севастополь : СевГУ, 2020. – 78 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, использующих принципы инкапсуляции и полиморфизма в рамках объектно-ориентированного программирования на языке Java в системе программирования BlueJ. Методические указания предназначены для студентов первого курса дневной формы обучения направлений 09.03.01 – Информатика и вычислительная техника и 09.03.04 – «Программная инженерия».

УДК 004.42/.432(076.5)
ББК 32.973-018.1я7

Методические указания рассмотрены и утверждены на заседании кафедры информационных технологий и компьютерных систем, протокол № 3 от 6 ноября 2019 г.

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

СОДЕРЖАНИЕ

1. ОБЩИЕ ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ	4
2. ЛАБОРАТОРНАЯ РАБОТА № 3. ПРОГРАММИРОВАНИЕ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ, СВЯЗАННЫХ ИЕРАРХИЧЕСКИ. РЕАЛИЗАЦИЯ СПИСКА ОБЪЕКТОВ НА ОСНОВЕ МАССИВА ФИКСИРОВАННОЙ ДЛИНЫ	6
4. ЛАБОРАТОРНАЯ РАБОТА № 4. ПРОГРАММИРОВАНИЕ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ, СВЯЗАННЫХ ИЕРАРХИЧЕСКИ. РЕАЛИЗАЦИЯ СПИСКА ОБЪЕКТОВ НА ОСНОВЕ ДИНАМИЧЕСКОГО СВЯЗНОГО СПИСКА	41
5. ЛАБОРАТОРНАЯ РАБОТА № 5. ИСПОЛЬЗОВАНИЕ ИНТЕРФЕЙСОВ ДЛЯ УНИФИКАЦИИ РАБОТЫ С ПОЛИМОРФНЫМИ ОБЪЕКТАМИ, ИМЕЮЩИМИ РАЗНУЮ ВНУТРЕНнюю ПРИРОДУ	64
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	78

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_n_m_Фамилия_Группа_Год, где n – 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 № 3. ПРОГРАММИРОВАНИЕ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ, СВЯЗАННЫХ ИЕРАРХИЧЕСКИ. РЕАЛИЗАЦИЯ СПИСКА ОБЪЕКТОВ НА ОСНОВЕ МАССИВА ФИКСИРОВАННОЙ ДЛИНЫ

2.1. Цель работы

Целью данной работы является получение навыков программирования иерархических структур данных на Java, закрепление навыков программирования классов и взаимодействия объектов.

2.2. Постановка задачи

Разработать программу, реализующую взаимодействие объектов, образующих иерархическую структуру данных. Для создания объектов, определенных вариантом задания разработать соответствующие классы. Список объектов реализовать на основе массива фиксированной длины.

2.3. Внеаудиторная подготовка

Для подготовки к лабораторной работе следует изучить краткие теоретические сведения, приведенные в пункте 2.4 методических указаний. Рекомендуется также ознакомиться с [1] (с.133-153).

2.4. Краткие теоретические сведения

Во всех современных языках программирования предусмотрены возможности для программирования иерархических структур, объединяющих разнотипные данные. В языке Pascal для этой цели предусмотрен тип «запись» (record), в языках C/C++ – тип «структура» (struct). Это структурированные комбинированные типы данных, состоящие из фиксированного числа полей (переменных) разного типа. Если объединяемые в запись (структуру) переменные имеют простые типы или типы массивов, то такая структура представляет собой простейшее двухуровневое дерево. На рисунке 2.1 изображено двухуровневое дерево (иерархическая структура), представляющее сущность «Студент».

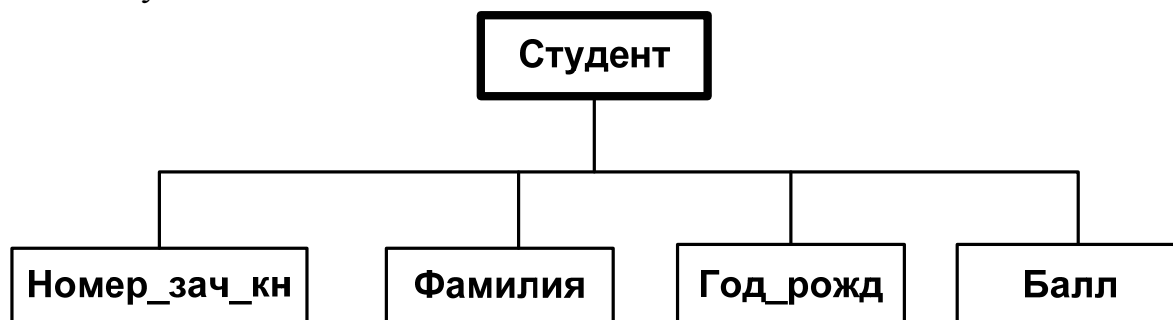


Рисунок 2.1 – Двухуровневое дерево для сущности «Студент»

Как видно из рисунка, запись о студенте, объединяет в себе поля различных типов: номер зачетной книжки и год рождения – целчисленные, фамилия – строка символов, балл – вещественное. В Java не предусмотрены специальные типы для представления подобных структур. Такие структуры реализуются на основе класса – базовой концепции Java. Создается соответствующий класс, определяющий поля (переменные экземпляра, свойства) и, при необходимости, – поведение (методы) сущности «Студент». На основе такого класса создаются экземпляры конкретных студентов – объекты. Класс может не иметь методов или иметь один конструктор.

Наличие конструктора с параметрами в классе «Студент» позволит создавать экземпляры студентов с конкретными значениями полей.

Заметим, 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).

Если по условию задачи студентов предстоит объединять в группы, то число уровней в древовидной структуре увеличится (рисунок 2.2)

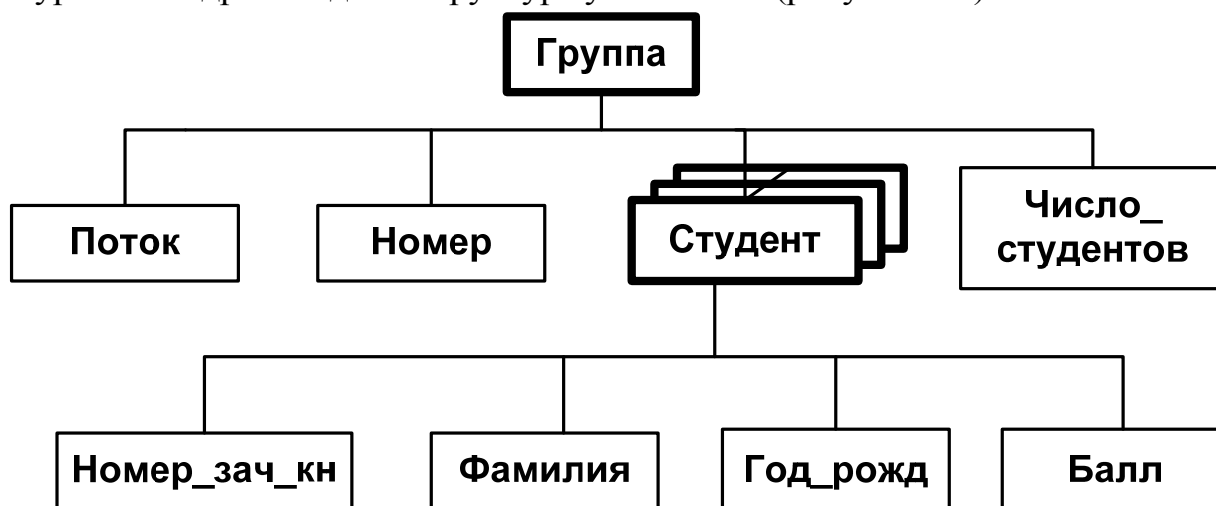


Рисунок 2.2 – Трехуровневая древовидная структура данных для сущности «Студенческая группа»

Заметим, что одним из полей (свойств) сущности «Студенческая группа» является массив студентов этой группы. Таким образом, если одним из полей класса является объект (массив объектов), созданный на основе другого класса, число уровней иерархии в древовидной структуре увеличивается.

Очевидно, что сущность «Студенческая группа» должна иметь методы для включения в свой состав и исключения из своего состава студентов. Метод, включающий очередного студента в группу должен увеличивать число студентов в группе, добавлять новый объект класса «Студент» к массиву студентов, проверяя при этом номер его зачетной книжки на несовпадение с номерами зачетных книжек студентов, уже включенных в группу ранее. Метод исключения студента должен находить требуемый экземпляр студента по

какому-либо условию, например, по номеру зачетной книжки, удалять его из массива студентов, уменьшать число студентов в группе. Кроме описанных методов в класс «Студенческая группа» можно включить и другие полезные методы, например, метод, рассчитывающий средний балл для группы.

Структура проекта, реализующего взаимодействие экземпляров сущностей «Студент» и «Студенческая группа» изображена на рисунке 2.3.

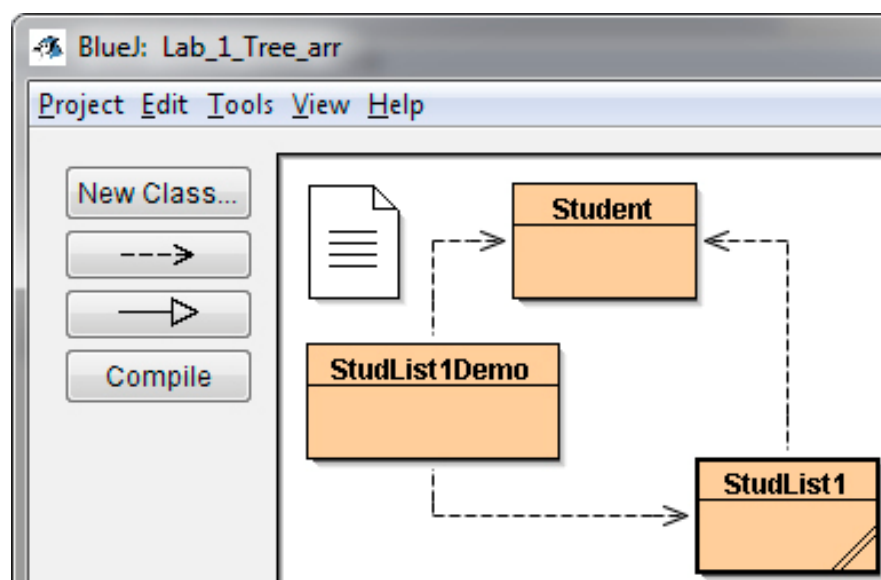


Рисунок 2.3 – Структура проекта, обеспечивающего взаимодействие экземпляров студентов и студенческих групп

В классе Student описана сущность «Студент», в классе StudList1 описана сущность «Студенческая группа», класс StudList1Demo (содержит метод main()) демонстрирует взаимодействие объектов (студентов и студенческих групп).

Члены класса Student описаны в таблице 2.1

Таблица 2.1 – Члены класса Student

Переменные экземпляра	Методы
<pre>private int id; /*номер зачетной книжки*/ private String name; //фамилия private float ball; /*балл на вступительных экзаменах*/ /*Все поля – private, доступ к ним осуществляется через public- методы класса*/ /*Поле «год рождения» в проекте не использовалось*/</pre>	<pre>// конструкторы public Student(); public Student(int id, String name, float ball); public Student (Student stud); //геттеры и сеттеры public int getId(); public String getName(); public float getBall(); public void setId(int id); public void setName(String name); public void setBall(float ball) // возвращает строку описания объекта public String toString()</pre>

Помимо переменных экземпляра в классе Student можно задать статическую final-переменную (именованную константу) STUDENT_FORMAT_STRING, которая будет задавать формат строки описания объекта и разделяться всеми объектами класса Student.

Члены класса StudList1 («Студенческая группа») описаны в таблице 2.2.

Таблица 2.2 – Члены класса StudList1

Переменные экземпляра	Методы
<pre>private String name; //шифр группы private Student[] studArr; /* ссылка на список студентов*/ private int num; /*количество студентов в группе (реальное) */ /*Все поля – private, доступ к ним осуществ- ляется через public- методы класса*/</pre>	<pre>//конструкторы public StudList1() /* создает группу с именем "" (пустая строка), числом студентов, равным нулю, выделяет память под минимально возможное число студентов в группе - MIN_NUM*/ public StudList1(String name, int numMax) /* создает группу с заданным именем name, числом студентов, равным нулю, выделяет память под максимально возможное число студентов в группе numMax */ public void setName(String name) /* метод-сеттер – меняет имя студенческой группы на значение параметра name */ public String getName() /* метод-геттер – возвращает имя студенческой группы */ public int getNum() /* метод-геттер – возвращает число студентов в группе */ public Student getStud (int id) /* метод-геттер – возвращает студента с заданным номером зачетки (id) */ public boolean addStud(Student stud) /*добавляет студента stud в список группы, учитывая следующее: студента нельзя добавить, если в список уже добавлено максимально возможное число студентов или id добавляемого студента уже упомянут в списке; если студент успешно добавлен метод возвращает true, иначе -false*/ public boolean delStud(int id) /* удаляет студента с заданным id (номером зачетной книжки) из списка группы */ public float avgBall(/*возвращает усредненный балл по группе*/ public Student[] aboveAvgBall() /*возвращает ссылку на список(массив) студентов, у которых балл выше среднего*/ public Student[] betweenBall(float b1, float b2) /* возвращает ссылку на список (массив) студентов с баллом в диапазоне [ball1,ball2]*/ putStudList() /* выводит в окно терминала список (массив) студентов группы (на него ссылается переменная экземпляра studArr) */</pre>

Помимо переменных экземпляра в классе StudList1 будут заданы две именованные константы: `private final static int MIN_NUM` (минимально возможное число студентов в группе; используется конструктором без параметров), `private final static String GROUP_FORMAT_STRING` (формат строки описания объекта). Эти static-переменные будут общими (разделяться) для всех объектов класса.

Следует отметить, что список студентов, являющийся одним из свойств сущности «Студенческая группа» может быть реализован несколькими способами. В данном проекте используется организация списка с помощью массива фиксированной длины: в классе StudList1 задана переменная экземпляра `private Student[ ] studArr`. Тип этой переменной – ссылочный. Значением этой переменной будет ссылка (адрес) на массив ссылок на объекты класса Student (рисунок 2.4).

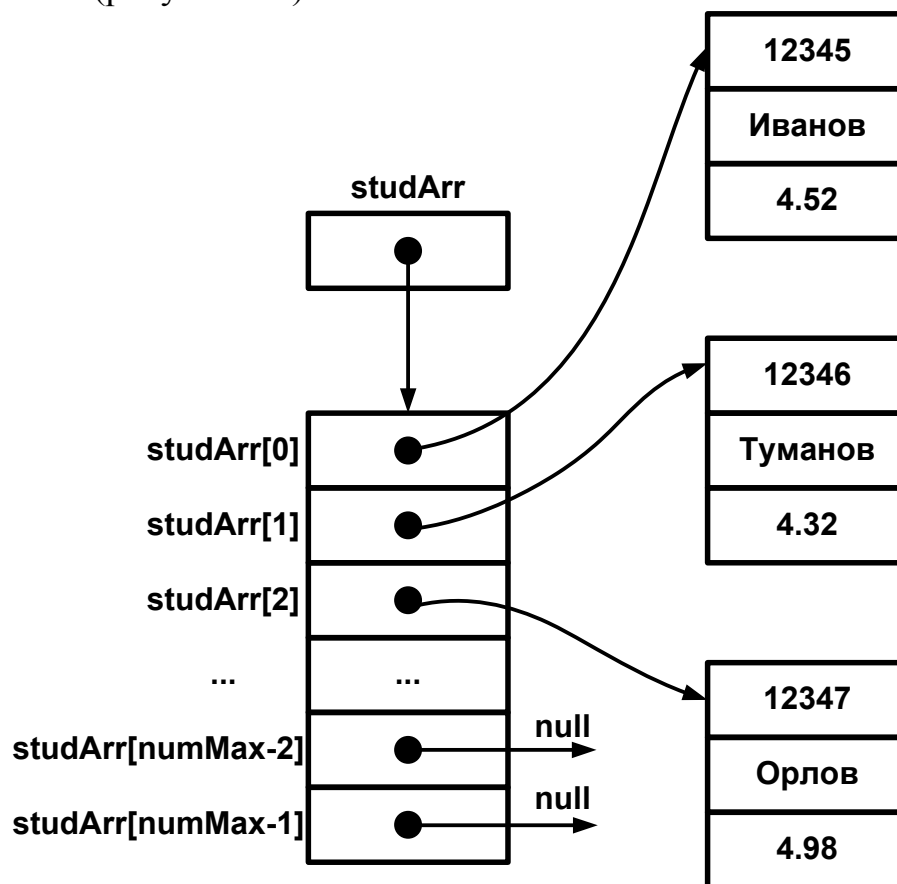


Рисунок 2.4 – Структура списка студентов

При создании массива функции `new` должна быть сообщена его длина. Поэтому одним из параметров конструктора класса StudList1 является переменная `int numMax` (максимально возможное количество студентов в группе). В теле конструктора присутствует оператор `studArr = new Student [numMax];`, который создает массив ссылок, состоящий из `numMax` элементов. Конструктор без параметров в качестве длины создаваемого массива ссылок использует значение именованной константы `MIN_NUM`. В классе StudList1 задана также переменная экземпляра `num` – реальное число студентов в группе (ее значение не может быть больше длины

созданного массива ссылок). После создания экземпляра группы значение **num** равно 0, значения всех ссылок в массиве **studArr** равны null (в группе пока нет студентов). После добавления в список очередного студента методом **addStud()**, очередной элемент массива станет ссылаться на конкретный экземпляр студента.

Если после добавления всех студентов в группу значение **num** будет меньше, чем длина массива ссылок, то ряд ссылок в массиве **studArr** останутся равными null, т.е. не будут использоваться (рисунок 2.4). Это является недостатком описанной реализации списка студентов.

Достоинством данной реализации является возможность прямого доступа к элементам списка, т.к. доступ к элементу массива осуществляется по его индексу. В задачах, предполагающих использование списков, в которых максимальное число элементов заранее неизвестно, можно строить списки на основе динамических структур данных с последовательным доступом и переменной длиной. Это так называемые связанные списки, которые будут рассматриваться нами в следующей лабораторной работе.

Прямой доступ к элементам списка обеспечивает более высокую скорость обработки элементов списка, чем последовательный.

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. Для удобства соответствующие данные можно подготовить в текстовом редакторе и сохранить в файле (рисунок 2.5), чтобы потом скопировать их в окно запуска метода **main()** (рисунок 2.6).

Позже мы изучим более удобные способы вводить данные из файла в программу.

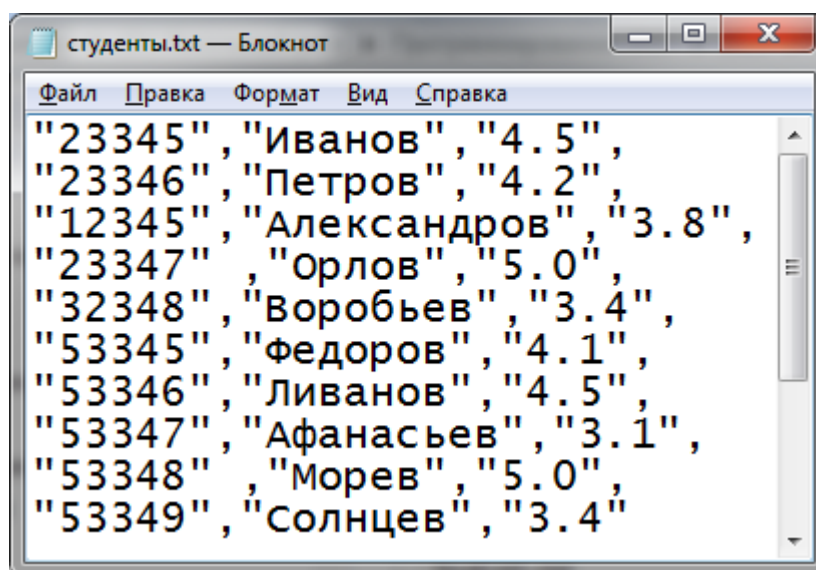


Рисунок 2.5 – Для метода **main()** подготовлены данные о 10 студентах

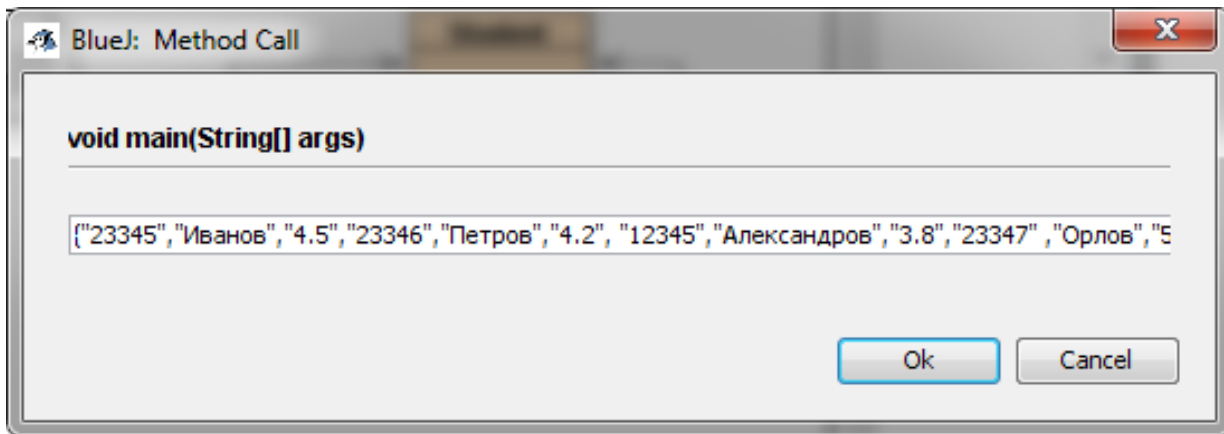


Рисунок 2.6 – Данные о студентах скопированы из файла студенты.txt и вставлены в окно запуска метода main() (между двумя фигурными скобками)

Кроме экземпляров студентов, в методе main() создается два экземпляра класса StudList1 (две студенческие группы). При создании группы gr1 число студентов ограничивается десятью (используется конструктор с параметрами), а у группы gr2 – пятью (используется конструктор без параметров). В каждую из групп включаются по пять студентов.

Затем делаются попытки добавить в группу gr1 студента с уже существующим в группе номером зачетной книжки, а в группу gr2 – шестого студента (лишнего). Попытки заканчиваются неудачно, о чем выдаются соответствующие сообщения.

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.

Затем из каждой группы исключается по одному студенту, после чего выводятся списки групп.

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

Текст программы.

```
public class Student { //Сущность - Студент
    // статическая переменная класса задает формат строки
    // описания объекта (общая для всех объектов класса)
    private final static String STUDENT_FORMAT_STRING =
        "Студент: %8d, %15s, %5.2f";
    // поля - переменные экземпляра (скрыты в классе)
    private int id; //номер зачетной книжки - ключевое
    private String name; // фамилия
    private float ball; // средний балл по последней сессии
    //методы
    // конструктор без параметров
    public Student() {
        id = 0; name = ""; ball = 0.0f;
```

```

}
// конструктор с параметрами
// инициализирует поля значениями параметров
public Student(int id,String name, float ball){
    this.id = id; this.name = name; this.ball = ball;
}
// конструктор с параметрами
// строит клон объекта, переданного как параметр
public Student (Student stud) {
    id=stud.id; name=stud.name; //здесь можно обойтись без геттеров
    ball=stud.ball;
}
//public-интерфейс доступа к полям:
//геттеры
public int getId() {return id;}
public String getName() {return name;}
public float getBall() {return ball;}
//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;}
//возвращает строку описания объекта
//(экземпляра класса Student)
public String toString(){
    return String.format(STUDENT_FORMAT_STRING,id,name,ball);
}
} //Student

```

```

public class StudList1 { /*Сущность - Студенческая группа.
    В данном классе список студентов реализован на основе
    массива фиксированной длины*/

    // статическая переменная класса задает
    // минимально возможное число студентов в группе
    // (общая для всех объектов класса)нужна
    // для конструктора без параметров
    private final static int MIN_NUM = 5;
    // статическая переменная класса задает формат строки
    // описания объекта (общая для всех объектов класса)
    private final static String GROUP_FORMAT_STRING =
        "Группа: %s, число студентов: %3d";
    //поля - переменные экземпляра (свойства группы)
    private String name; //шифр группы
    private Student[ ] studArr; //ссылка на список студентов
    private int num; //количество студентов в группе
    // конструкторы
    public StudList1(){
        name = "";
        num = 0; //пока список не заполнен
        studArr = new Student [MIN_NUM]; // выделение памяти под
        // максимально возможное в создаваемой группе число
        // студентов - MIN_NUM
    }
}

```

```

}
public StudList1(String name, int numMax){
    //задается название группы и максимально возможное
    //числа студентов в группе
    this.name = name;
    num = 0; //пока список не заполнен
    studArr = new Student [numMax]; //выделение памяти под
    // максимально возможное в создаваемой группе число
    //студентов - numMax
}
//метод-сеттер – меняет шифр группы
public void setName(String name) {this.name = name;}
/*для остальных переменных экземпляра сеттеры не предусмотрены,
их будут изменять по определенным правилам методы addStud() и
delStud() */

//методы-геттеры
// возвращает шифр группы:
public String getName() {return name;}
// возвращает число студентов в группе:
public int getNum(){return num;}
// возвращает студента с заданным номером зачетки (id):
public Student getStud (int id){
    for (int i=0; i<num; i++)
        if (studArr[i].getId() == id) return studArr[i];
    return null;
}

// сформировать строку описания объекта
public String toString(){
    return String.format(GROUP_FORMAT_STRING,name,num);
}

//добавить студента в список группы:
public boolean addStud(Student stud){
    //студента нельзя вставить, если
    //уже вставлено максимально возможное
    //количество студентов или
    //его id уже упомянут в списке
    if (num == studArr.length) return false;
    for (int i = 0; i < num; i++)
        if (studArr[i].getId() == stud.getId()) return false;
    studArr[num] = stud;
    num++;
    return true;
}

//удалить студента из списка группы:
public boolean delStud(int id){
    int i;
    for (i = 0; i < num; i++) //ищем студента в списке
        if (studArr[i].getId() == id) break;

```

```

    if (i == num) //прошли весь список
        return false; // студента с таким id нет
//студент найден, i – его индекс
//удаляем из списка:
    for (i = i+1; i < num; i++) //i-ая ссылка будет затерта (i+1)-ой
        studArr[i-1] = studArr[i]; //смещение ссылок
    studArr[i-1] = null;
    // в массиве стало на 1 студента меньше
    num--;
    return true;
}
//запрос:
//вернуть средний балл по группе
public float avgBall() {
    if (num==0) return 0; // если в списке нет студентов
    float avg=0;
    for (int i = 0; i < num; i++)
        avg = avg + studArr[i].getBall();
    return avg / num;
}
//запрос:
//вернуть список (массив) студентов, у которых балл выше среднего
public Student[ ] aboveAvgBall(){
    if (num==0) return null; // список группы пуст - возврат null
    //создаем вспомогательный массив ссылок на студентов
    Student [ ]X = new Student[num];
    int j = 0; //индекс вспомогательного массива
    float avg = avgBall();
    for (int i = 0; i < num; i++)
        if (studArr[ i ].getBall() > avg){
            X[ j ] = studArr[ i ];
            //ссылка на студента X[j] теперь равна ссылке studArr[i];
            //число объектов типа Student не увеличивается,
            //т.к. работаем со ссылками
            j = j+1;
        } //if
    // j содержит количество ссылок на студентов в массиве X
    //остальные (num-j)ссылок в массиве X равны null
    if (j == 0) return null; // не найдено элементов с баллом
    // выше среднего – когда баллы у всех студентов равны
    Student [ ] Y = new Student[ j ]; // в массиве Y не будет
    //null-ссылок
    for (int i=0; i<j; i++) Y[ i ] = X[ i ];
    return Y;
}
// запрос:
//вернуть список (массив) студентов с баллом в диапазоне [ball1,ball2]
public Student[ ] betweenBall(float b1, float b2){
    //алгоритм аналогичен использованному в предыдущем методе
    if (num == 0) return null; //список группы пуст - возврат null
    // создаем вспомогательный массив ссылок на студентов

```

```

Student[ ]X = new Student [num];
int j=0; //индекс вспомогательного массива
for (int i=0; i<num; i++){
    float b = studArr[ i ].getBall();
    if ((b >= b1)&&(b <= b2)){
        X[ j ] = studArr[ i ];
        //ссылка X[j] теперь равна ссылке studArr[i];
        //число объектов типа Student не увеличивается,
        //т.к. работаем со ссылками
        j = j+1;
    } //if
} //for
//j содержит количество ссылок на студентов в массиве X
//остальные (num-j)ссылок в массиве X равны null
if (j == 0) return null; // не найдено элементов, входящих в диапазон
Student [ ] Y = new Student[ j ]; // в массиве Y не будет
                                // null-ссылок
for (int i=0; i<j; i++) Y[ i ]=X[ i ];
return Y;
}
//выводит список студентов в окно терминала
public void putStudList() {
    System.out.println(name);
    if (num !=0 ) {
        System.out.printf ("%5s%17s%11s%17s\n",
                            "Номер","Номер зач. кн.","ФИО","Балл");
        for (int i=0; i<num; i++)
            System.out.printf (" %-7d %-15d %-16s %-10.2f\n",
                                i+1, studArr[i].getId(), studArr[i].getName(), studArr[i].getBall());
    } //if
} // putStudList
} //StudList1

public class StudList1Demo{ // Взаимодействие студентов и групп
    public static void putStudArr(Student [ ] arr){
        // вывести список студентов
        if (arr == null) return;
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[ i ]); // используется метод toString() студента
    }
    public static void main (String[ ] args) {
        StudList1 gr1 = new StudList1 ("ИБТБ-11д",10);
        StudList1 gr2 = new StudList1 (); //в группе может быть до 5 человек
        gr2.setName("ИБТБ-12д"); // назначаем группе название

        // 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));
}
System.out.println ("Сформированы следующие группы:");
System.out.printf ("%s,\n%s\n\n", gr1,gr2);
// пытаемся вставить в gr1 студента с тем же номером зачетки
if (!gr1.addStud(new Student (32348, "Козлов",5.0f))) {
    System.out.printf ("В группу %s добавить студента 32348 не удалось:\n",
        gr1.getName());
    System.out.println(" нарушена уникальность ключа.");
}
//пытаемся вставить в gr2 лишнего студента
if (!gr2.addStud(new Student (60538, "Акимов",4.3f))) {
    System.out.printf ("В группу %s добавить студента 60538 не удалось:\n",
        gr2.getName());
    System.out.println(" превышен лимит.");
}
System.out.println();
//выборка данных
// выводим списки студентов
gr1.putStudList();
System.out.printf("Средний балл группы: %.2f\n", gr1.avgBall());
System.out.println ("Студенты, у которых балл выше среднего в группе:");
putStudArr(gr1.aboveAvgBall());
System.out.println("Студенты, у которых балл в диапазоне [3.2,4.2]:");
putStudArr(gr1.betweenBall(3.2f,4.2f));
// удаление студента по id
gr1.delStud(12345);
//выводим список группы после удаления студента
System.out.printf("Состав группы %s после удаления студента:\n",
    gr1.getName());
gr1.putStudList();
System.out.println();
// то же для второй группы:
//выборка данных
// выводим списки студентов
gr2.putStudList();
System.out.printf("Средний балл группы: %.2f\n", gr2.avgBall());
System.out.println("Студенты, у которых балл выше среднего в группе:");
putStudArr(gr2.aboveAvgBall());
System.out.println("Студенты, у которых балл в диапазоне 3.2,4.2]:");
putStudArr(gr2.betweenBall(3.2f,4.2f));

```

```

// удаление студента по id
gr2.delStud(53347);
//выводим список группы после удаления студента
System.out.printf ("Состав группы %s после удаления студента:\n",
                    gr2.getName());
gr2.putStudList();
System.out.println();
//Проверяем,есть ли в группах студенты с заданным номером зачетки
System.out.println("Проверка наличия студентов:");
int n = 23345;
Student s1 = gr1.getStud(n);
System.out.printf ("В группе %s:\n",gr1.getName());
if (s1 ==null ) System.out.printf ("Нет студента с номером зачетки %d\n",n);
else System.out.println(s1);
n=70000;
s1=gr1.getStud(n);
if (s1==null) System.out.printf ("Нет студента с номером зачетки %d\n",n);
else System.out.println(s1);
n=53349;
s1=gr2.getStud(n);
System.out.printf ("В группе %s:\n",gr2.getName());
if (s1==null) System.out.printf ("Нет студента с номером зачетки %d\n",n);
else System.out.println(s1);
} //main
} // StudList1Demo

```

Результат работы программы представлен на рисунках 2.7- 2.9. Для того, чтобы наблюдать все выводимые данные, включайте в окне терминала опции «Очищать окно терминала перед вызовом метода» и «Неограниченная буферизация».

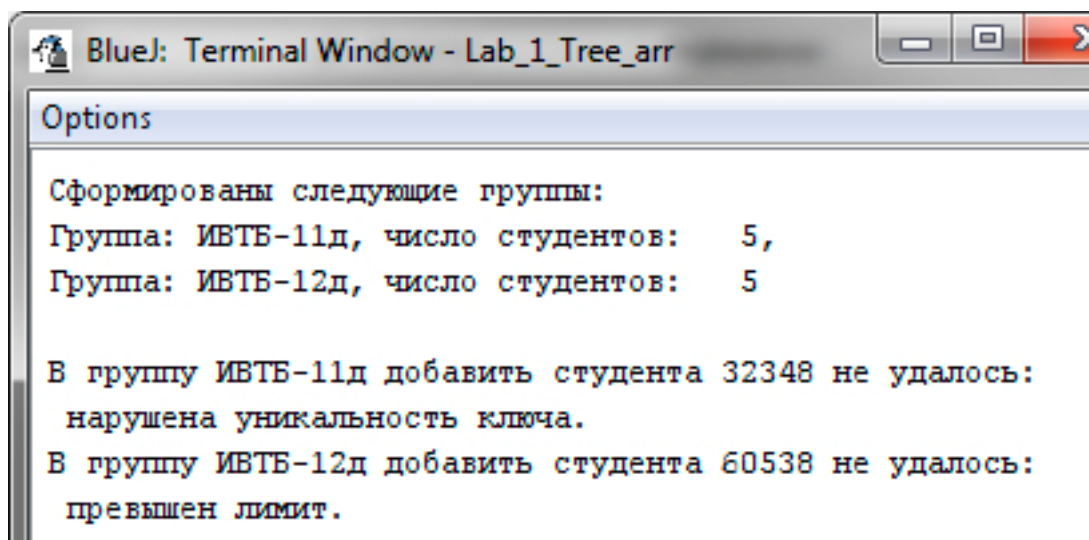


Рисунок 2.7 – Результаты работы программы

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
Средний балл группы: 4,18			
Студенты, у которых балл выше среднего в группе:			
Студент:	23345,	Иванов,	4,50
Студент:	23346,	Петров,	4,20
Студент:	23347,	Орлов,	5,00
Студенты, у которых балл в диапазоне [3.2,4.2]:			
Студент:	23346,	Петров,	4,20
Студент:	12345,	Александров,	3,80
Студент:	32348,	Воробьев,	3,40
Состав группы ИВТБ-11д после удаления студента:			
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
ИВТБ-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	Солнцев	3,40
Средний балл группы: 4,02			
Студенты, у которых балл выше среднего в группе:			
Студент:	53345,	Федоров,	4,10
Студент:	53346,	Ливанов,	4,50
Студент:	53348,	Морев,	5,00
Студенты, у которых балл в диапазоне 3.2,4.2]:			
Студент:	53345,	Федоров,	4,10
Студент:	53349,	Солнцев,	3,40
Состав группы ИВТБ-12д после удаления студента:			
ИВТБ-12д			
Номер	Номер зач. кн.	ФИО	Балл
1	53345	Федоров	4,10
2	53346	Ливанов	4,50
3	53348	Морев	5,00
4	53349	Солнцев	3,40

Рисунок 2.8 – Результаты работы программы (продолжение)

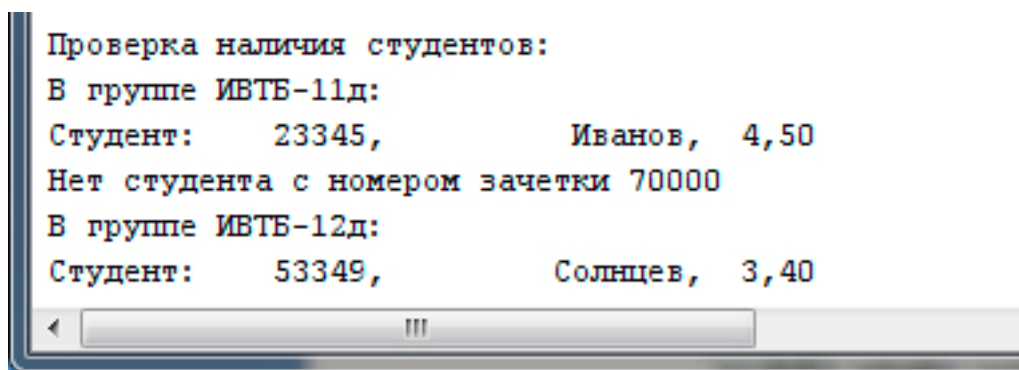


Рисунок 2.9 – Результаты работы программы (продолжение)

Для тестирования работы методов класса **StudList1** в случае отсутствия студентов в списке группы, была сделана копия проекта, в которой класс **StudList1Demo** был изменен следующим образом:

```
public class StudList1Demo{ // Взаимодействие студентов и групп
    public static void putStudArr(Student [ ] arr){
        // вывести список студентов
        if (arr == null) return;
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]); // используется метод toString() студента
    }
    public static void main (String[ ] args){
        StudList1 gr1 = new StudList1 ("ИВТБ-11д",10);
        // группа создана, но студентов в ней нет
        // выборка данных
        // выводим списки студентов
        gr1.putStudList();
        System.out.printf("Средний балл группы: %.2f\n", gr1.avgBall());
        System.out.println ("Студенты, у которых балл выше среднего в группе:");
        putStudArr(gr1.aboveAvgBall());
        System.out.println("Студенты, у которых балл в диапазоне [3.2,4.2]:");
        putStudArr(gr1.betweenBall(3.2f,4.2f));
        // удаление студента по id
        gr1.delStud(12345);
        //выводим список группы после удаления студента
        System.out.printf("Состав группы %s после удаления студента:\n",
                           gr1.getName());
        gr1.putStudList();
        System.out.println();
        //Проверяем, есть ли в группах студенты с заданным номером зачетки
        System.out.println("Проверка наличия студентов:");
        int n = 23345;
        Student s1 = gr1.getStud(n);
        System.out.printf ("В группе %s:\n", gr1.getName());
        if (s1 == null) System.out.printf ("Нет студента с номером зачетки %d\n",n);
        else System.out.println(s1);
    } //main
} // StudList1Demo
```

Результат работы программы представлен на рисунке 2.10.

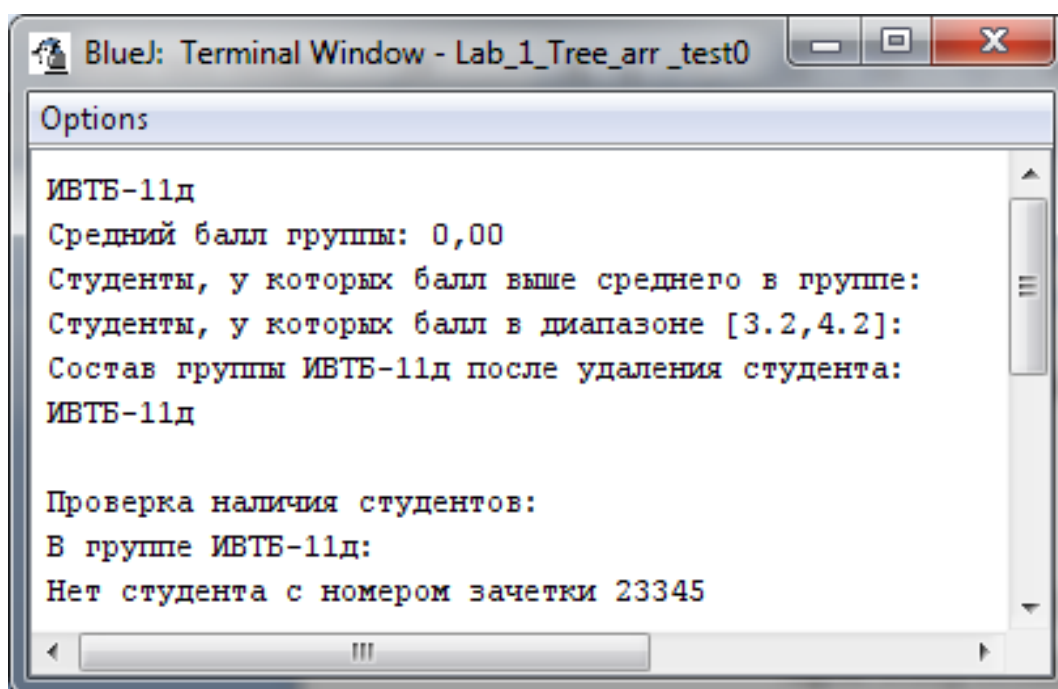


Рисунок 2.10 – Результаты тестирования методов программы при отсутствии студентов в группе

Для тестирования работы методов класса **StudList1** в случае наличия в списке группы только одного студента в классе **StudList1Demo**, текст которого приведен выше, было предусмотрено добавление одного студента в группу:

```
public class StudList1Demo{ // Взаимодействие студентов и групп
    public static void putStudArr(Student [ ] arr){
        // вывести список студентов
        if (arr == null) return;
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]); // используется метод toString() студента
    }
    public static void main (String[ ] args){
        StudList1 gr1 = new StudList1 ("ИВТБ-11д",10);
        gr1.addStud(new Student (23345, "Зайцев",4.8f));
        //группа создана и в нее добавлен один студент
        // выборка данных
        // выводим списки студентов
        gr1.putStudList();
        ...
    } //main
} // StudList1Demo
```

Результаты работы программы приведены на рисунке 2.11.

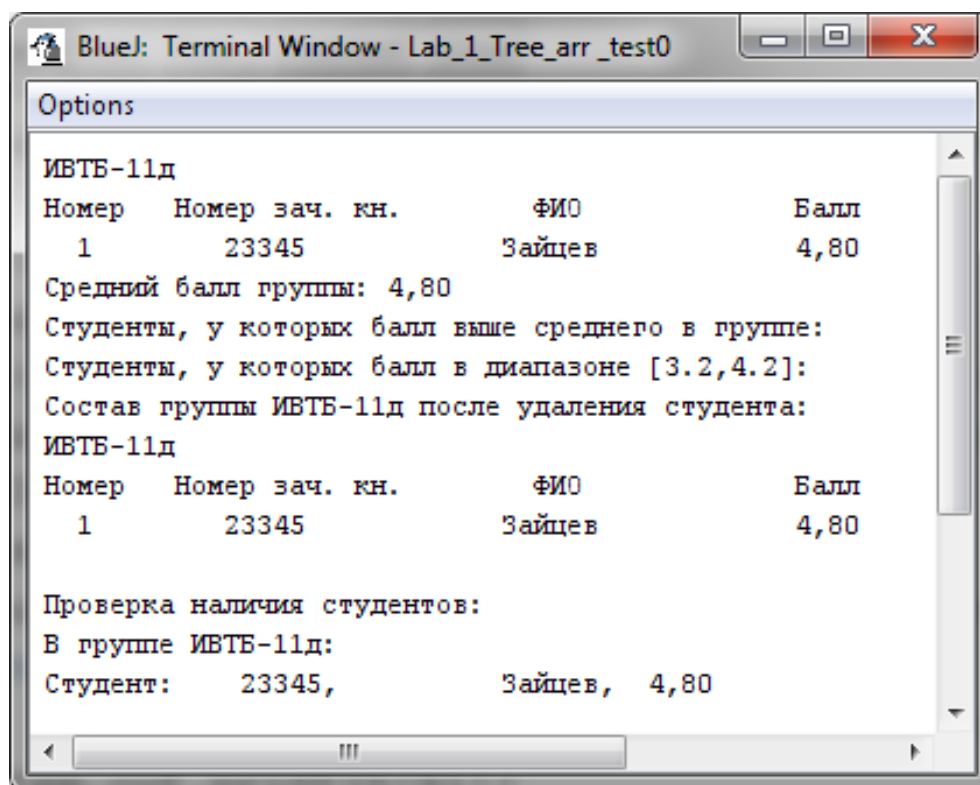


Рисунок 2.11 – Результаты тестирования методов программы при наличии в группе только одного студента

Чтобы допроверить (при других условиях) работу методов **betweenBall()** (балл находится в диапазоне) и **delStud()** (студент с указанным номером зачетки есть в группе) при наличии одного студента в группе, нужно внести в код метода **main()** следующие изменения:

```

...   System.out.println("Студенты, у которых балл в диапазоне [4.7,5.0]:");
      putStudArr(gr1.betweenBall(3.2f,4.2f));
      // удаление студента по id
      gr1.delStud(23345); ...
  
```

Результаты работы программы представлены на рисунке 2.12.

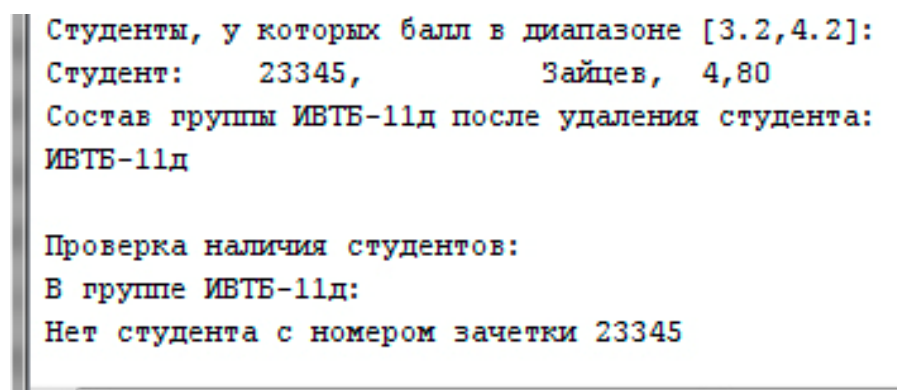


Рисунок 2.12 – Результаты работы программы с измененными значениями балла и номера зачетной книжки

Анализируя текст программы, заметим, что методы
public Student[] aboveAvgBall(),
public Student[] betweenBall(float b1, float b2),

возвращающие ссылку на список студентов (массив студентов) не увеличивают количество объектов класса «Студент» в оперативной памяти, т.к. они работают со ссылками на эти объекты.

В качестве примера на рисунке 2.13 показано размещение объектов в оперативной памяти для группы gr1 после создания группы, добавления в нее студентов и выполнения метода gr1.aboveAvgBall().

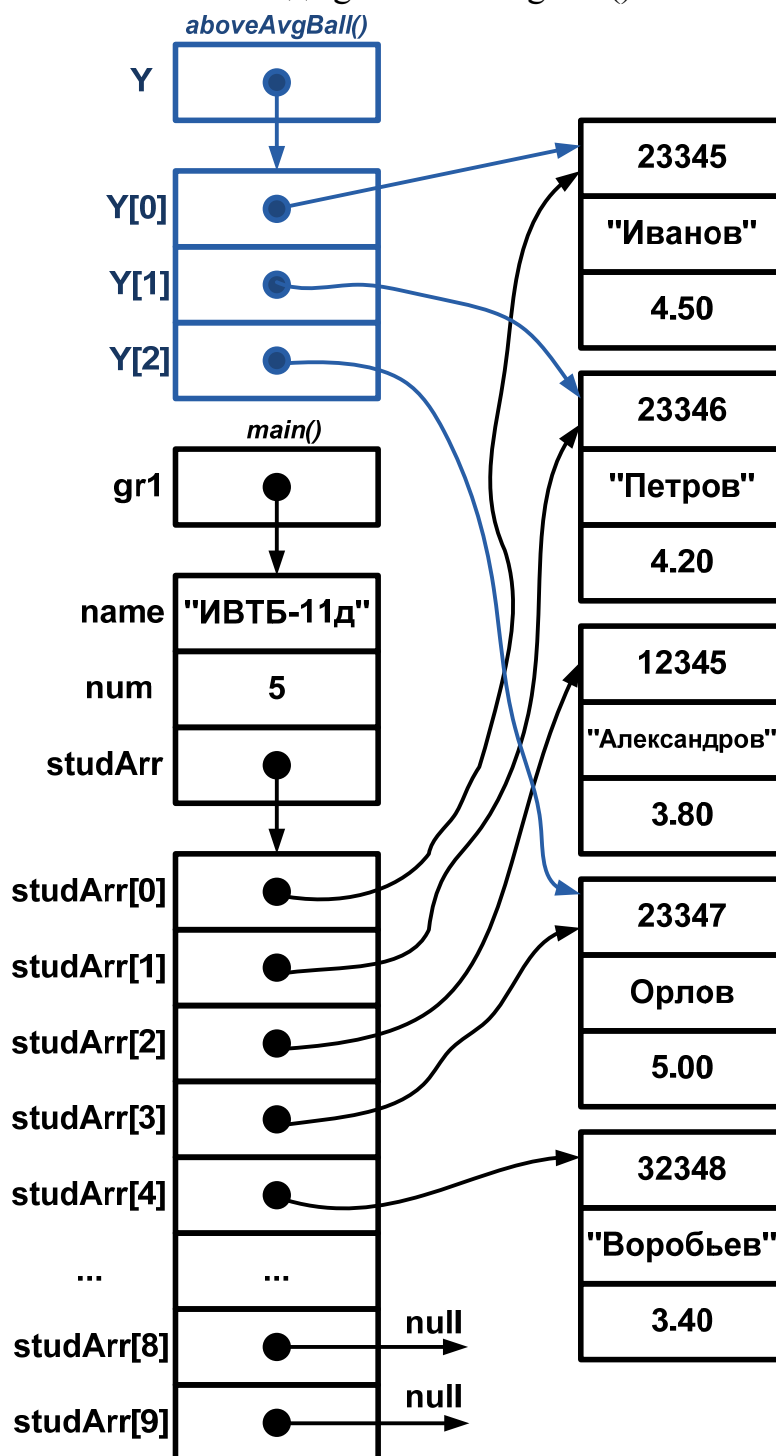


Рисунок 2.13 – Размещение объектов в оперативной памяти для группы gr1

2.5. Порядок выполнения работы

- 1) Внимательно прочитайте п. 2.4 и разберите текст программы-примера.
- 2) Разработаете описание классов, заданных вариантом задания (по примеру таблиц 2.1, 2.2, а также действий, которые будет выполнять метод `main()`).
- 3) Согласно описанию разработайте код для каждого класса. Если программирование каких-либо методов класса вызовет затруднение, составьте алгоритм метода и приведите его в отчете.
- 4) Проведите отладку программы и проверку ее работы на тестовых примерах.
- 5) Текст программы и результаты ее работы приведите в отчете.

2.6. Варианты заданий

Варианты заданий представлены в таблице 2.3.

Номер варианта задания V необходимо вычислить по формуле

$$\text{int } V = (N \leq 14) ? \text{variant} : N \% 14;$$

где N – номер студента в списке группы,

`variant` – номер варианта задания в таблице 2.3.

Внимание! программирование метода 14 Сущности 2 (отмечен * в таблице 2.3) является обязательным только для студентов, претендующих на отличную оценку.

Рекомендуется следующая сигнатура метода (на примере метода сортировки по полю `ball` списка объектов класса `Student`, описанного в разделе 2.4):

`public Student [ ] sortBall (int sortType).`

В качестве параметра в метод передается тип сортировки:

`sortType < 0` – по возрастанию;

`sortType > 0` – по убыванию;

`sortType = 0` – без сортировки.

В качестве базового можно использовать любой из известных методов сортировки одномерного массива, например, метод пузырька_2.

Подобно методам **`aboveAvgBall()`** и **`betweenBall()`** данный метод возвращает ссылку на массив ссылок на объекты класса **`Student`**. Размер массива ссылок должен быть равен значению поля **`num`**. Проверка работы метода **`sortBall()`** может быть выполнена аналогично проверкам методов **`aboveAvgBall()`** и **`betweenBall()`**: возвращаемый список студентов выводится в окно терминала с помощью метода **`putStudArr()`** класса **`StudList1Demo`**.

Метод `main()`, определенный в отдельном классе, должен демонстрировать взаимодействие объектов, заданных вариантом задания аналогично тому, как это сделано в проекте, текст которого приведен в разделе 2.4, а результаты работы – на рисунках 2.7 – 2.12.

Таблица 2.3 – Варианты заданий

№	Сущность 1 (класс1) Все поля – private	Сущность 2 (класс 2) Все поля – private
1	2	3
1	Фирма. <u>Именованные константы (static final-переменные):</u> FIRM_FORMAT_STRING <u>Поля:</u> 1) int id_фирмы (ключевое), 2) String название, 3) int число_сотрудников, 4) double базовая_зарплата. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Проект. <u>Именованные константы (static final-переменные):</u> PROJECT_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива фирм; используется в конструкторе без параметров) <u>Поля:</u> 1) String название_проекта, 2) Фирма [] список_фирм, 3) int число_фирм <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий фирму с заданным значением ключевого поля из списка фирм, 6) сеттер для поля название_проекта, 7) добавить фирму с учетом уникальности id_фирмы, 8) удалить фирму с заданным id_фирмы, 9) вернуть среднюю базовую зарплату для списка фирм, 10) вернуть список фирм, в которых базовая зарплата ниже средней, 11) вернуть список фирм, у которых базовая зарплата находится в заданном интервале, 12) вывод списка фирм в окно терминала, 13) toString(), 14)* вернуть список фирм, отсортированный по базовой_зарплате.

1	2	3
2	Офис. <u>Именованные константы (static final-переменные):</u> OFFICE_FORMAT_STRING <u>Поля:</u> 1) int id_офиса (ключевое), 2) String адрес, 3) double площадь, 4) int число_комнат. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Фирма. <u>Именованные константы (static final-переменные):</u> FIRM_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива офисов; используется в конструкторе без параметров) <u>Поля:</u> 1) String название_фирмы, 2) Офис [] список_офисов, 3) int число_офисов. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий офис с заданным значением ключевого поля из списка офисов, 6) сеттер для поля название_фирмы, 7) добавить офис с учетом уникальности id_офиса, 8) удалить офис с заданным id_офиса, 9) вернуть среднюю площадь для списка офисов, 10) вернуть список офисов, в которых площадь превышает среднюю, 11) вернуть список офисов, у которых площадь находится в заданном интервале, 12) вывод списка офисов в окно терминала. 13) toString(), 14)* вернуть список офисов, отсортированный по площади.

1	2	3
3	Сотрудник. <u>Именованные константы (static final-переменные):</u> EMPLOYEE_FORMAT_STRING <u>Поля:</u> 1) int id_сотрудника (ключевое), 2) String фамилия, 3) int год_рождения, 4) double зарплата. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Отдел. <u>Именованные константы (static final-переменные):</u> DEPARTMENT_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива сотрудников; используется в конструкторе без параметров) <u>Поля:</u> 1) String название_отдела, 2) Сотрудник [] список_сотрудников, 3) int число_сотрудников <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий сотрудника с заданным значением ключевого поля из списка сотрудников, 6) сеттер для поля название_отдела, 7) добавить сотрудника с учетом уникальности id_сотрудника, 8) удалить сотрудника с заданным id_сотрудника, 9) вернуть среднюю зарплату сотрудников по отделу, 10) вернуть список сотрудников, у которых зарплата ниже средней, 11) вернуть список сотрудников, у которых зарплата находится в заданном интервале, 12) вывод списка сотрудников в окно терминала. 13) toString(), 14)* вернуть список сотрудников, отсортированный по году рождения.

1	2	3
4	Автомобиль. <u>Именованные константы (static final-переменные):</u> CAR_FORMAT_STRING <u>Поля:</u> 1) int id_автомобиля (ключевое), 2) String марка, 3) int год_выпуска, 4) double стоимость. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Автопарк. <u>Именованные константы (static final-переменные):</u> CARPARK_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива автомобилей; используется в конструкторе без параметров) <u>Поля:</u> 1) String название_автопарка, 2) Автомобиль [] список_автомобилей, 3) int число_автомобилей. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий автомобиль с заданным значением ключевого поля из списка автомобилей, 6) сеттер для поля название_автопарка, 7) добавить автомобиль с учетом уникальности его id_автомобиля, 8) удалить автомобиль с заданным id_автомобиля, 9) вернуть среднюю стоимость автомобиля в автопарке, 10) вернуть список автомобилей, стоимость которых выше средней, 11) вернуть список автомобилей, стоимость которых находится в заданном интервале, 12) вывод списка автомобилей в окно терминала, 13) toString(), 14)* вернуть список автомобилей, отсортированный по году выпуска.

1	2	3
5	Товар. <u>Именованные константы (static final-переменные):</u> PRODUCT_FORMAT_STRING <u>Поля:</u> 1) int id_товара (ключевое), 2) String название, 3) String производитель, 4) double цена. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Склад. <u>Именованные константы (static final-переменные):</u> STOCK_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива продуктов; используется в конструкторе без параметров) <u>Поля:</u> 1) int номер_склада, 2) Товар [] список_товаров, 3) int число_товаров_в_списке. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий товар с заданным значением ключевого поля из списка товаров, 6) сеттер для поля номер_склада, 7) добавить товар с учетом уникальности id_товара, 8) удалить товар с заданным id_товара, 9) вернуть среднюю цену товара на складе, 10) вернуть список товаров, цена которых ниже средней, 11) вернуть список товаров, цена которых находится в заданном интервале, 12) вывод списка товаров в окно терминала, 13) toString(), 14)* вернуть список товаров, отсортированный по цене.

1	2	3
6	Инженер. <u>Именованные константы (static final-переменные):</u> ENGINEER_FORMAT_STRING <u>Поля:</u> 1) int id_инженера (ключевое), 2) String фамилия, 3) int стаж, 4) double оклад. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Вычислительный центр. <u>Именованные константы (static final-переменные):</u> COMPCENTER_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива инженеров; используется в конструкторе без параметров) <u>Поля:</u> 1) int номер_выч_центра, 2) Инженер [] список_инженеров, 3) int число_инженеров. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий инженера с заданным значением ключевого поля из списка инженеров, 6) сеттер для поля номер_выч_центра, 7) добавить инженера с учетом уникальности id_инженера, 8) удалить инженера с заданным id_инженера, 9) вернуть средний оклад инженера, 10) вернуть список инженеров, оклад которых выше среднего, 11) вернуть список инженеров, оклад которых находится в заданном интервале, 12) вывод списка инженеров в окно терминала, 13) toString(), 14)* вернуть список инженеров, отсортированный по стажу работы.

1	2	3
7	Специальность. <u>Именованные константы (static final-переменные):</u> SPECIALTY_FORMAT_STRING <u>Поля:</u> 1) int id_специальности (ключевое), 2) String название, 3) int срок обучения, 4) double проходной_балл. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Университет. <u>Именованные константы (static final-переменные):</u> UNIVERSITY_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива специальностей; используется в конструкторе без параметров). <u>Поля:</u> 1) String название_университета, 2) Специальность [] список_специальностей, 3) int число_специальностей. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий специальность с заданным значением ключевого поля из списка специальностей, 6) сеттер для поля название_университета, 7) добавить специальность с учетом уникальности id_специальности, 8) удалить специальность с заданным id_специальности, 9) вернуть средний проходной балл для университета, 10) вернуть список специальностей, проходной балл на которые выше среднего, 11) вернуть список специальностей, проходной балл на которые находится в заданном интервале, 12) вывод списка специальностей в окно терминала, 13) toString(), 14)* вернуть список специальностей, отсортированный по проходному баллу.

1	2	3
8	Дисциплина (учебный предмет). <u>Именованные константы (static final-переменные):</u> SUBJECT_FORMAT_STRING <u>Поля:</u> 1) int id_дисциплины (ключевое), 2) String название, 3) int номер семестра, 4) float число_кредитов. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString(). <u>Примечание.</u> Число кредитов не может быть меньше 3.0 и больше 7.0	Кафедра. <u>Именованные константы (static final-переменные):</u> DEPARTMENT_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива дисциплин; используется в конструкторе без параметров). <u>Поля:</u> 1) String название_кафедры, 2) Дисциплина [] список_дисциплин, 3) int число_дисциплин <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий дисциплину с заданным значением ключевого поля из списка дисциплин, 6) сеттер для поля название_кафедры, 7) добавить дисциплину с учетом уникальности id_дисциплины, 8) удалить дисциплину с заданным id_дисциплины, 9) вернуть среднее число кредитов для списка дисциплин кафедры, 10) вернуть список дисциплин, число кредитов для которых выше среднего, 11) вернуть список дисциплин, число кредитов для которых находится в заданном интервале, 12) вывод списка дисциплин в окно терминала, 13) toString(), 14)* вернуть список дисциплин, отсортированный по числу кредитов.

1	2	3
9	Преподаватель. <u>Именованные константы (static final-переменные):</u> LECTURER_FORMAT_STRING <u>Поля:</u> 1) int id_преподавателя (ключевое), 2) String фамилия, 3) String должность, 4) double размер_ставки. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString(). <u>Примечание.</u> Преподаватель не может работать менее, чем на 0.25 ставки и более, чем на 1.75 ставки.	Кафедра. <u>Именованные константы (static final-переменные):</u> DEPARTMENT_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива преподавателей; используется в конструкторе без параметров). <u>Поля:</u> 1) String название_кафедры, 2) Преподаватель [] список_преподавателей, 3) int число_преподавателей. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий преподавателя с заданным значением ключевого поля из списка преподавателей, 6) сеттер для поля название_кафедры, 7) добавить преподавателя с учетом уникальности id_преподавателя, 8) удалить преподавателя с заданным id_преподавателя, 9) вернуть средний размер ставки преподавателя на кафедре, 10) вернуть список преподавателей, размер ставки у которых выше среднего, 11) вернуть список преподавателей, размер ставки которых находится в заданном интервале, 12) вывод списка преподавателей в окно терминала, 13) toString(), 14)* вернуть список преподавателей, отсортированный по размеру ставки.

1	2	3
10	Книга. <u>Именованные константы (static final-переменные):</u> BOOK_FORMAT_STRING <u>Поля:</u> 1) int id_книги (ключевое), 2) String автор, 3) String название_книги, 4) int число страниц, 5) double цена. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Каталог. <u>Именованные константы (static final-переменные):</u> CATALOG_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива книг; используется в конструкторе без параметров). <u>Поля:</u> 1) String название_каталога, 2) Книга [] список_книг, 3) int число_книг. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий книгу с заданным значением ключевого поля из списка книг, 6) сеттер для поля название_каталога, 7) добавить книгу с учетом уникальности id_книги, 8) удалить книгу с заданным id_книги, 9) вернуть среднюю цену книги для каталога, 10) вернуть список книг, цена которых выше средней, 11) вернуть список книг, цена которых находится в заданном интервале, 12) вывод списка книг в окно терминала, 13) toString(), 14)* вернуть список книг, отсортированный по числу страниц.

1	2	3
11	Спортсмен. <u>Именованные константы (static final-переменные):</u> SPORTSMAN_FORMAT_STRING <u>Поля:</u> 1) int id_спортсмена (ключевое), 2) String фамилия, 3) String роль, 4) double зарплата. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Команда. <u>Именованные константы (static final-переменные):</u> TEAM_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива спортсменов; используется в конструкторе без параметров). <u>Поля:</u> 1) String название_команды, 2) Спортсмен [] список_спортсменов, 3) int число_спортсменов. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий спортсмена с заданным значением ключевого поля из списка спортсменов, 6) сеттер для поля название_команды, 7) добавить спортсмена с учетом уникальности id_спортсмена, 8) удалить спортсмена с заданным id_спортсмена, 9) вернуть среднюю зарплату спортсмена в команде, 10) вернуть список спортсменов, зарплата которых ниже средней, 11) вернуть список спортсменов, зарплата которых находится в заданном интервале, 12) вывод списка спортсменов в окно терминала, 13) toString(), 14)* вернуть список спортсменов, отсортированный по зарплате.

1	2	3
12	Кинофильм. <u>Именованные константы (static final-переменные):</u> FILM_FORMAT_STRING <u>Поля:</u> 1) int id_кинофильма (ключевое), 2) String название, 3) String год выпуска, 4) double бюджет. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Кинотеатр. <u>Именованные константы (static final-переменные):</u> CINEMA_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива кинофильмов; используется в конструкторе без параметров). <u>Поля:</u> 1) String название_кинотеатра, 2) Кинофильм [] список_кинофильмов, 3) int число_кинофильмов. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий кинофильм с заданным значением ключевого поля из списка кинофильмов, 6) сеттер для поля название_кинотеатра, 7) добавить кинофильм с учетом уникальности id_кинофильма, 8) удалить кинофильм с заданным id_кинофильма, 9) вернуть средний бюджет для фильмов, демонстрирующихся в кинотеатре, 10) вернуть список фильмов, бюджет которых выше среднего, 11) вернуть список кинофильмов, бюджет которых находится в заданном интервале, 12) вывод списка кинофильмов в окно терминала, 13) toString(), 14)* вернуть список кинофильмов, отсортированный по году выпуска.

1	2	3
13	Партия деталей. <u>Именованные константы (static final-переменные):</u> BATCH_FORMAT_STRING <u>Поля:</u> 1) int id_детали (ключевое), 2) String название_детали, 3) String количество_деталей, 4) double цена_детали. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Поставка. <u>Именованные константы (static final-переменные):</u> SUPPLY_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива партий_деталей; используется в конструкторе без параметров). <u>Поля:</u> 1) String шифр_поставки, 2) Партия_деталей [] список_партий, 3) int число_партий. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий партию деталей с заданным значением ключевого поля из списка партий, 6) сеттер для поля шифр_поставки, 7) добавить партию_деталей с учетом уникальности id_детали, 8) удалить партию деталей с заданным id_детали, 9) вернуть среднюю цену детали для поставки, 10) вернуть список партий, цена детали в которых ниже средней, 11) вернуть список партий, цена детали в которых находится в заданном интервале, 12) вывод списка партий в окно терминала, 13) toString(), 14)* вернуть список партий_деталей, отсортированный по количеству деталей.

1	2	3
14	Секция <u>Именованные константы (static final-переменные):</u> SECTION_FORMAT_STRING <u>Поля:</u> 1) int id_секции (ключевое), 2) String название_секции, 3) int раз_в_неделю, 4) double цена_за_месяц. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, инициализирующими значения полей объекта, 3) конструктор с параметром-объектом, строящий клон этого объекта, 4) геттеры и сеттеры для всех полей, 5) toString().	Спортивная_школа. <u>Именованные константы (static final-переменные):</u> SPORTSCHOOL_FORMAT_STRING, MIN_NUM = 5 (минимальная длина массива секций; используется в конструкторе без параметров). <u>Поля:</u> 1) int номер_школы, 2) Секция [] список_секций, 3) int число_секций. <u>Методы:</u> 1) конструктор без параметров, 2) конструктор с параметрами, 3-4) геттеры для первого и третьего полей, 5) геттер, возвращающий секцию с заданным значением ключевого поля из списка секций, 6) сеттер для поля номер_школы, 7) добавить секцию с учетом уникальности id_секции, 8) удалить секцию с заданным id_секции, 9) вернуть среднюю цену_за_месяц для спортивной школы, 10) вернуть список секций, цена_за_месяц в которых ниже средней, 11) вернуть список секций, цена_за_месяц в которых находится в заданном интервале, 12) вывод списка секций в окно терминала, 13) toString(), 14)* вернуть список секций, отсортированный по цене_за_месяц.

2.7. Содержание отчета

- 1) Цель работы.
- 2) Постановка задачи с указанием данных варианта задания.
- 3) Иерархическая структура данных для Сущности_2 (пример на рисунке 2.2).
- 4) Структура основного списка объектов (для экземпляра Сущности _2) после добавления в него 5 объектов и списка, созданного в результате работы метода **aboveAvgBall()** (пример изображен на рисунке 2.13).
- 5) Структура проекта (пример на рисунке 2.3).
- 6) Описание классов проекта (пример – таблицы 2.1, 2.2).
- 7) Описание действий, выполняемых методом **main()**.
- 8) Текст программы.
- 9) Протокол отладки
- 10) Результаты работы программы.
- 11) Вывод.

2.8. Контрольные вопросы

- 1) Как реализовать в Java тип данных, аналогичный типам данных **record** (запись) в языке Pascal и **struct** (структура) в языках C/C++? Какой класс в вашей программе наиболее похож на такой тип данных?
- 2) Какую структуру данных определяет класс, в котором определена переменная экземпляра, являющаяся объектом другого класса?
- 3) Как создать иерархическую структуру данных на Java?
- 4) Опишите, что определяет свойства и поведение объектов в вашей программе?
- 5) Охарактеризуйте возможности доступа к полям объектов, определенных в вашей программе. Как они заданы? Как осуществляется обращение к полям объектов в зависимости от спецификатора доступа?
- 6) Что такое состояние объекта? Охарактеризуйте состояние объекта, включающего в себя список других объектов, до и после удаления объекта из списка.
- 7) Как в вашей программе реализован список объектов? В чем недостаток и достоинство этого метода?
- 8) Какие методы в вашей программе возвращают значение простого типа?
- 9) Какие методы в вашей программе возвращают ссылку на объект?
- 10) Какие методы в вашей программе возвращают ссылку на массив объектов? Увеличивают ли они количество этих объектов в оперативной памяти?
- 11) Почему методы добавления и удаления объектов из списка возвращают тип **boolean**?
- 12) Назовите тип переменной, заданной следующим образом:
private Student[] studArr; // список студентов

- 13) При помощи какой функции можно назначить ссылочной переменной значение конкретного адреса объекта? Приведите пример.
- 14) Как осуществляется передача в методы параметров простых и структурированных типов. Объясните механизм.
- 15) Почему метод **putStudArr()** класса **StudList1Demo** объявлен статическим?
- 16) Для чего и как в данной работе использовался параметр `args` метода `main`?
- 18) При помощи каких библиотечных методов можно получить из строкового представления (`String`) `int`-значение и `double`-значение? Приведите пример из вашей программы.
- 19) Для предметной области, определенной вариантом задания, разработайте метод:
- а) добавляющий очередной объект в начало списка;
 - б) добавляющий очередной объект в позицию списка с заданным номером;
 - в) удаляющий первый объект из списка объектов;
 - г) удаляющий объект из позиции списка с заданным номером;
 - д) удаляющий из списка объектов все объекты, у которых значение неключевого числового поля меньше заданного;
 - е) возвращающий первый объект списка;
 - ж) возвращающий последний объект списка;
 - з) возвращающий объект из заданной позиции списка;
 - и) возвращающий список объектов, у которых значение неключевого числового поля больше заданного;
 - к) осуществляющий копирование списка объектов (полная копия вместе с объектами);
 - л) осуществляющий выборочное копирование списка объектов (полная копия вместе с объектами) – с указанием начальной позиции и количества элементов копируемой части;
 - м) осуществляющий перестановку объектов списка в обратном порядке – `reverse`;
 - н) осуществляющий перестановку двух объектов списка, стоящих в заданных позициях – `swap`;
 - о) осуществляющий перестановку объекта, стоящего в заданной позиции списка и следующего за ним объекта – `swapNext`;
 - п) осуществляющий сравнение двух списков – `compareList`.

3. ЛАБОРАТОРНАЯ РАБОТА № 4. ПРОГРАММИРОВАНИЕ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ, СВЯЗАННЫХ ИЕРАРХИЧЕСКИ. РЕАЛИЗАЦИЯ СПИСКА ОБЪЕКТОВ НА ОСНОВЕ ДИНАМИЧЕСКОГО СВЯЗНОГО СПИСКА

3.1. Цель работы

Целью данной работы является получение навыков программирования иерархических структур данных на Java, ознакомление с принципами создания и использования динамически изменяющихся структур данных – связанных списков, а также с использованием рекурсии при определении данных.

3.2. Постановка задачи

Разработать программу, реализующую взаимодействие объектов, образующих иерархическую структуру данных. Для создания объектов, определенных вариантом задания разработать соответствующие классы. Список объектов реализовать на основе динамического связанного списка.

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.133-153), [2] (с. 167-190) и [3].

3.4. Краткие теоретические сведения

Массивы, рассмотренные нами ранее, в определенной степени можно считать статическими, т.е. не изменяющимися во времени структурами данных. Т.е. меняться могут только значения элементов массива, а не его структура. При создании массива с помощью процедуры new или путем инициализации жестко фиксируется его длина. Список объектов в предыдущей лабораторной работе был реализован на основе такого массива фиксированной длины. Тип элементов этого массива – ссылочный (рисунок 2.4). Каждый элемент массива содержит адрес некоторого объекта или null (если он не указывает ни на какой объект). Использование такой структуры имеет ограничение, т.к. предполагает, что должно быть заранее известно максимальное число объектов в списке. Если число объектов в списке меньше максимально возможного, часть элементов массива ссылок не используется.

Если максимально возможное число элементов в списке объектов заранее не известно, можно построить такой список по принципу линейного динамического связанного списка [2, 3]. Слово «динамический» здесь подразумевает изменение структуры данных (количества элементов в списке) во времени.

Связные списки являются самым простым примером абстрактных динамических типов данных – типов данных, которые применяются при решении многих задач дискретной математики и программирования. Слово «абстрактные» здесь означает, что элементы списка могут иметь произвольную природу, слово «динамические» (изменяющиеся во времени) отмечает, что количество элементов списка может изменяться в процессе выполнения программы.

Особенностью списков являются:

- наличие у каждого элемента информационной (*inf*) и связывающей (*next*) частей, причем именно связывающая часть обеспечивает динамичность;
- малое количество *базовых* операций со списками (добавить элемент, извлечь элемент).

В списке выделяют «голову» и элементы. Если нет «головы», то нет списка. Если нет элементов, то список – пустой. Последний элемент списка называют «хвостом» списка. Обычно голова и элементы – одного типа. Из головы непосредственно доступен только один элемент. Структура простейшего линейного динамического списка объектов изображена на рисунке 3.1. Такой список называют односвязным, т.к. из каждого элемента непосредственно доступен только один (следующий) элемент.

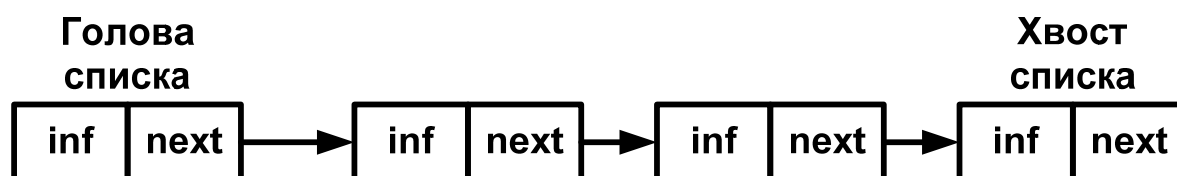


Рисунок 3.1 – Структура линейного односвязного динамического списка

Рассмотрим отдельный элемент списка с учетом механизма его реализации на Java. Информационная часть **inf** объекта типа «Элемент списка» представляет из себя ссылочную переменную, значением которой является адрес одного из объектов, из которых состоит список. Ссылка **inf** может указывать на объект любой природы, при этом все объекты одного и того же списка должны иметь одинаковую природу.

Связывающая часть **next** объекта типа «Элемент списка» также является переменной ссылочного типа. Ее значением является адрес следующего элемента списка. У элемента, являющегося «хвостом» списка, значение поля **next** равно **null**.

На рисунке 3.2 детализирована структура списка объектов некоторого класса *C*. На голову списка, которая является объектом класса «Элемент списка» указывает ссылочная переменная *A*. В момент времени, отображенный на рисунке, связный список состоит из трех элементов типа «Элемент списка». В следующие моменты времени количество элементов в списке может

увеличиваться или уменьшаться. Каждый элемент типа «Элемент списка» ссылается на объект класса *C* (напомним, что наша задача – создать список объектов именно класса *C*). Программист может использовать объект класса *C*, на который ссылается «голова» списка для хранения, например, названия списка и/или числа элементов в списке, если у объекта класса *C* есть подходящие поля (*String*, *int*). Если не предполагается использовать указанный объект, то его можно не создавать. В этом случае ссылка *inf* «головы» списка будет равна *null*, поэтому соответствующий объект класса *C* изображен пунктиром.

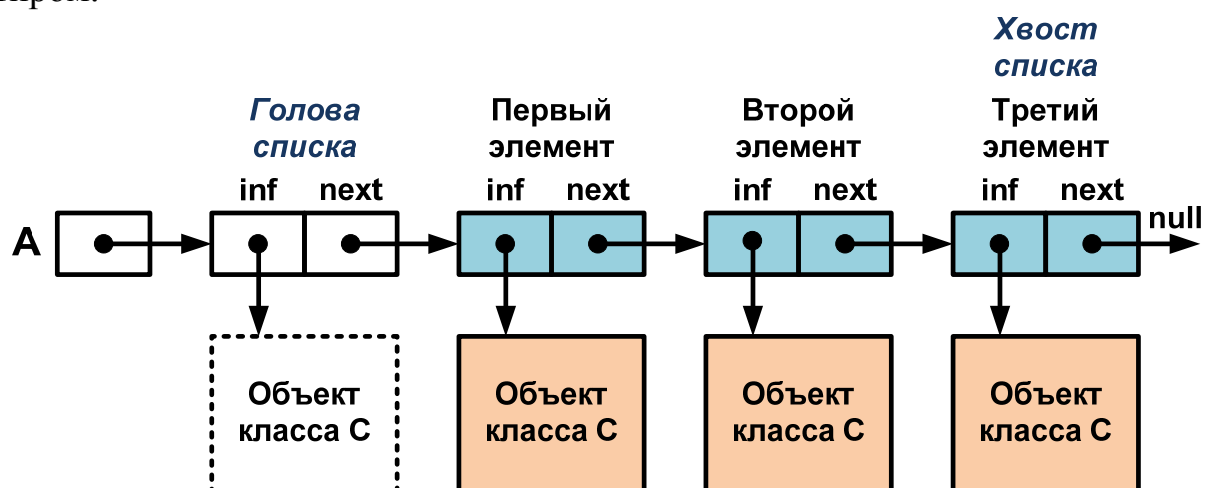


Рисунок 3.2 – Организация линейного односвязного динамического списка объектов класса *C*

Возникает вопрос, можно ли обойтись без элемента, который является головой списка? Если, например, мы реализуем стек объектов, т.е. динамическую структуру данных, реализующую принцип LIFO (Last In First Out) – «последний вошел, первый вышел», то элемент всегда будет вставляться в начало списка (кладем на «вершину» стека) и удаляться из начала списка (снимаем с «вершины» стека). Поскольку на начало списка указывает ссылочная переменная *A* (указатель стека), «голова» списка становится лишним элементом (без нее можно обойтись).

Соответствующая структура данных изображена на рисунке 3.3.

Оператор удаления элемента списка для стека будет выглядеть следующим образом: ***A=A.next***. После выполнения этого оператора структура стека будет иметь вид, изображенный на рисунке 3.4. Список потеряет связь с удаленным элементом. Указатель стека станет указывать на следующий элемент.

Процедура добавления элемента в стек иллюстрируется рисунками 3.5-3.8. На рисунке 3.5 изображено состояние памяти перед добавлением: есть стек с указателем *A* и создан новый объект класса «Элемент списка», на который указывает ссылочная переменная *e*. Вспомогательная ссылочная переменная **old** пока не указывает ни на какой объект.

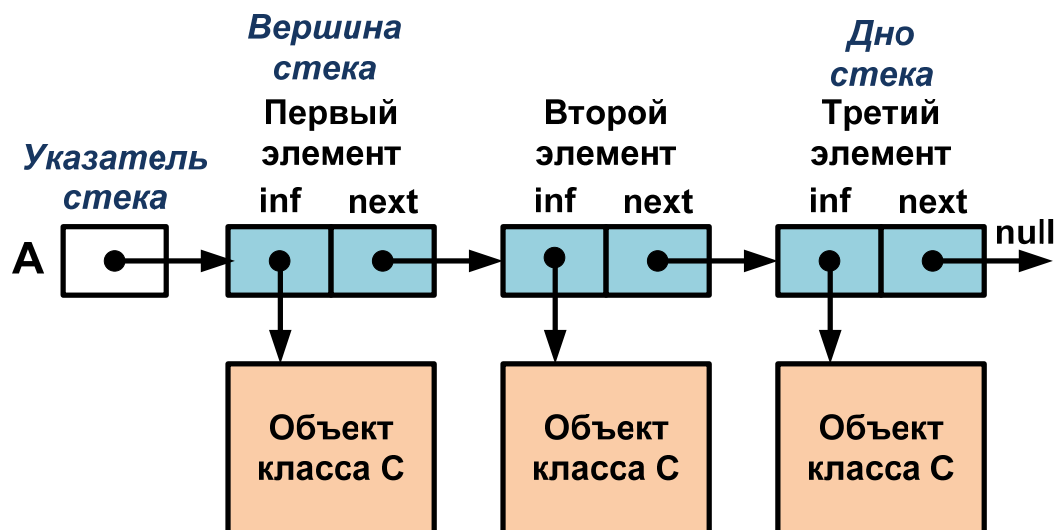


Рисунок 3.3 – Структура данных для стека объектов класса C

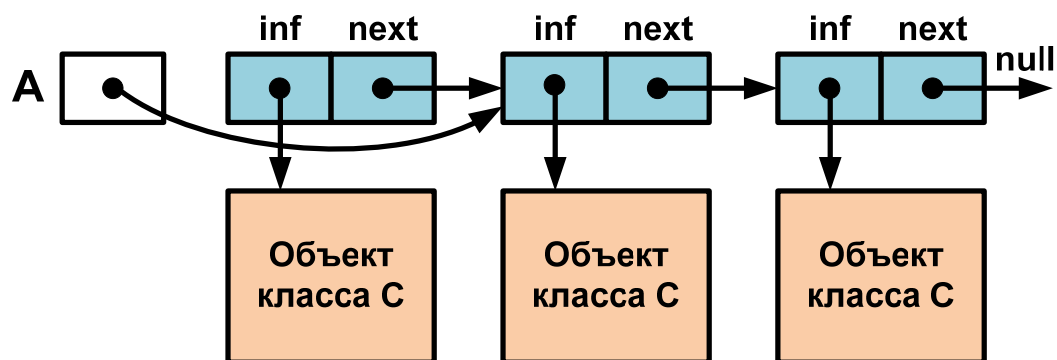


Рисунок 3.4 – Удаление элемента с вершины стека

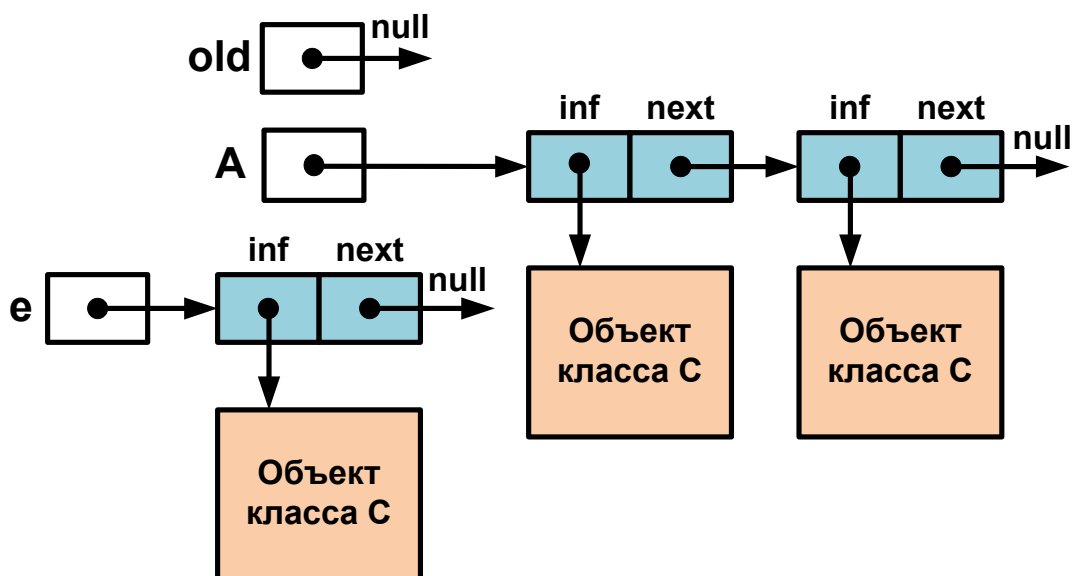


Рисунок 3.5 – Состояние ссылочных переменных и объектов перед добавлением элемента в список

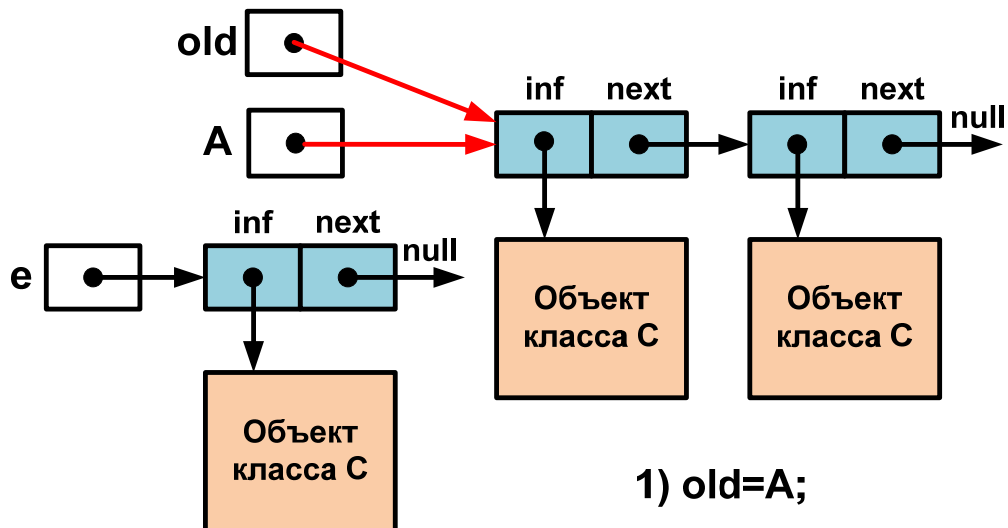


Рисунок 3.6 – Вставка элемента в стек – первое действие

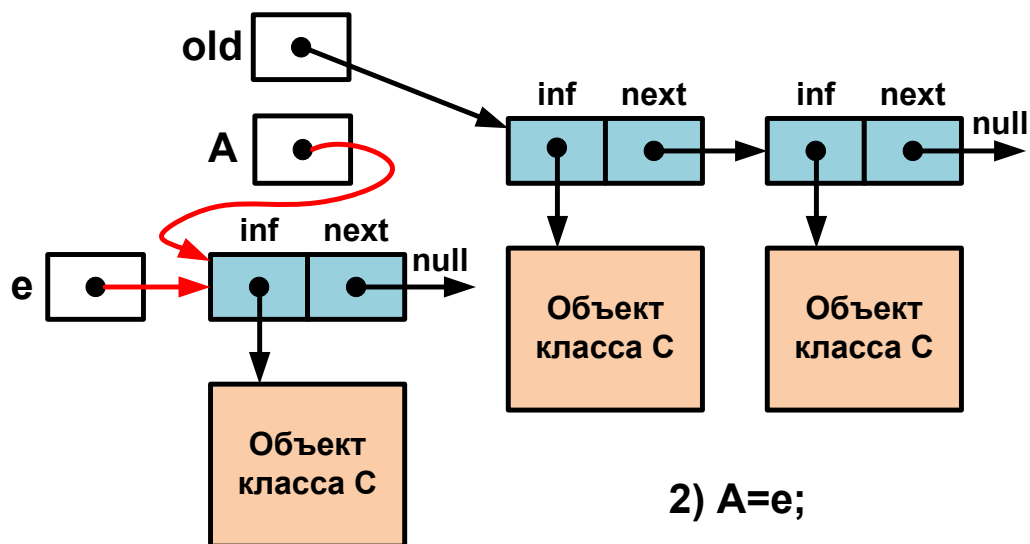


Рисунок 3.7 – Вставка элемента в стек – второе действие

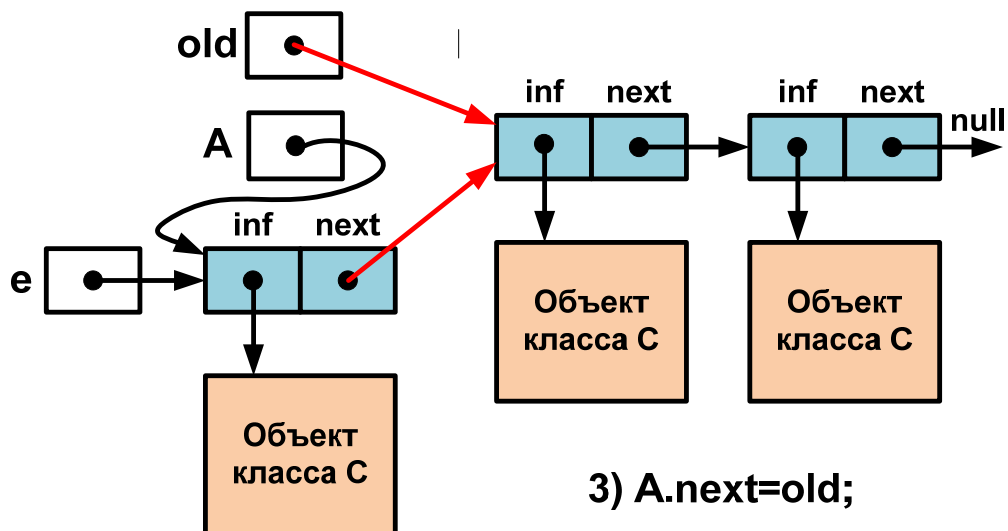


Рисунок 3.8 – Вставка элемента в стек – третье действие

Рисунки 3.6 – 3.8 иллюстрируют последовательные действия по добавлению элемента к стеку: **old=A; A=e; A.next=old;** .

Допустим, что постановка задачи предусматривает наличие операции удаления не первого, а произвольного элемента списка, например элемента, указывающего на объект класса С с определенным значением некоторого поля (например, на студента с определенным значением номера зачетной книжки). Процедура поиска такого элемента будет состоять в **последовательном** просмотре элементов от начала списка до нахождения элемента, удовлетворяющего условию поиска. Соответствующий фрагмент кода может выглядеть следующим образом:

```
e=A;
while (e!=null) {
    if (объект с адресом e.inf удовлетворяет условию поиска) break;
    e = e.next;
}
```

После нахождения элемента списка, соответствующего условию поиска, на него указывает ссылочная переменная **e**. Ссылочная переменная **A** попрежнему указывает на начало списка (рисунок 3.9).

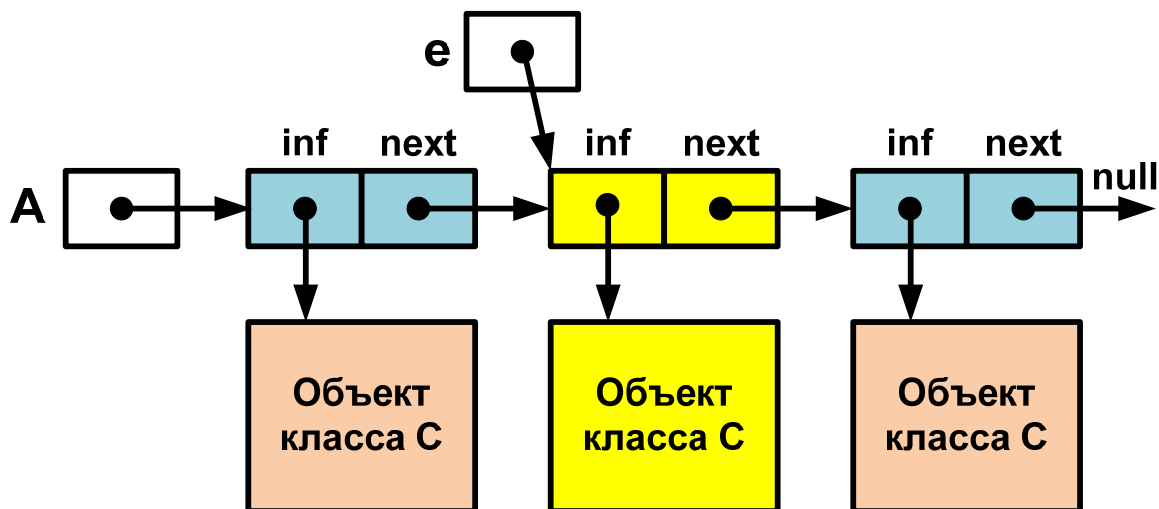


Рисунок 3.9 – Найден элемент списка, удовлетворяющий условию поиска

Удалить из списка элемент, на который указывает ссылочная переменная **e**, затруднительно, т.к. для этого нужно иметь доступ к полю **next** предыдущего в списке объекта, а связь с ним через переменную **e** уже потеряна (наш список – структура с последовательным доступом и односторонними связями).

Решить данную проблему как раз и позволяет наличие «головы» (дополнительного элемента в начале списка). Для списка, изображенного на рисунке 3.2, фрагмент кода, осуществляющий поиск элемента, удовлетворяющего условию, и его удаление из списка, может выглядеть так:

```
e=A;
```

```

while (e.next!=null) {
    if (объект с адресом e.next. inf удовлетворяет условию поиска)
        e.next=e.next.next; //удалить элемент из списка
    e = e.next;
}

```

Поскольку в момент выполнения условия поиска ссылочная переменная **e** указывает на предыдущий элемент списка (рисунок 3.10), удаление найденного элемента (элемента с адресом **e.next**) из списка будет достаточно простой задачей. Значение ссылки **e.next** после удаления элемента изображено пунктирной стрелкой.

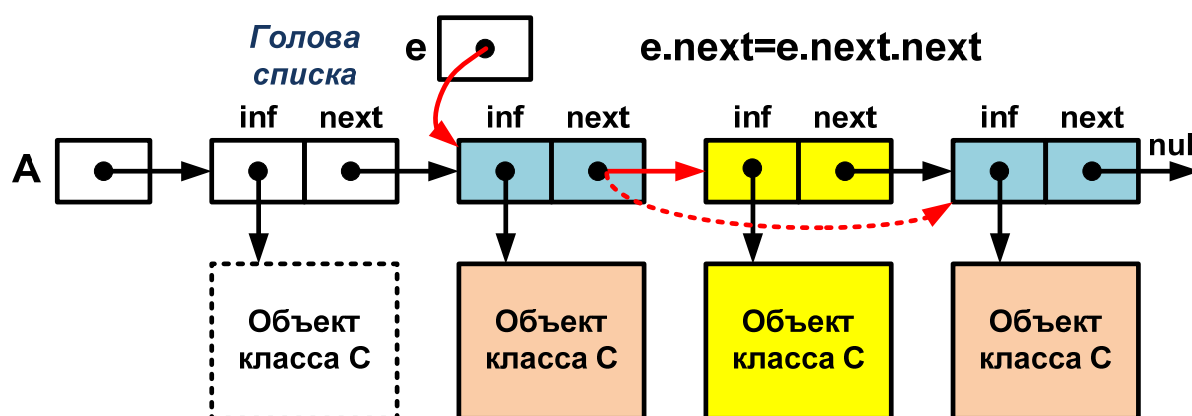


Рисунок 3.10 – Удаление элемента из списка, имеющего «голову»

Еще одна (главная) причина использования отдельного элемента списка в качестве «головы» списка состоит в том, что методы, работающие со списком, в рамках объектно-ориентированного программирования должны запускаться от имени объекта, принадлежащего классу «Элемент списка». Если список представлен его «головой», то целесообразно такие методы, как «включить новый элемент в список», «удалить элемент из списка» запускать от имени «головы» списка.

Рекурсивное определение связного списка можно представить следующим образом:

<Список объектов> = <Элемент списка объектов> <Список объектов>.

Заметим, что при рекурсивном описании списка любой элемент списка можно считать головой списка, состоящего из следующих за ним элементов списка (следует учитывать, что списки могут быть и пустыми).

Помимо рассмотренных операций добавления и удаления элементов списка, для списка могут быть определены операции, которые позволяют работать со списком, как с массивом: «получить значение поля **inf** элемента с заданным номером», «задать значение поля **inf** элемента с заданным номером».

В отличие от параллельного доступа к элементу массива, доступ к элементу списка с заданным номером осуществляется путем последовательного перемещения по списку, начиная от головы. На основе вышеперечисленных операций можно разработать другие операции. Типовой перечень операций со списком:

- определить число элементов в списке или отсутствие элементов в списке;
- проверить равенство списков;
- удалить список; скопировать список;
- добавить элемент в конец списка; удалить элемент из конца списка;
- добавить или удалить элемент в заданной позиции списка;
- расположить элементы списка в порядке, обратном исходному порядку;
- упорядочить элементы списка.

Пример. Рассмотрим задачу создания списка студентов на основе линейного динамического односвязного списка. Предполагаем, что объекты списка – студенты создаются на основе класса **Student**, текст которого рассмотрен в методических указаниях к предыдущей лабораторной работе.

Структура проекта изображена на рисунке 3.11.

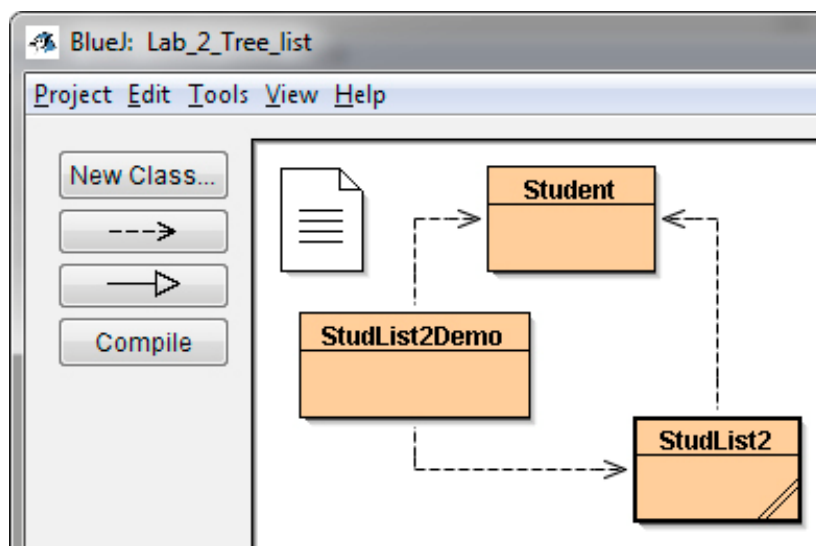


Рисунок 3.11 – Структура проекта

Класс **StudList2** реализует список студентов (студенческую группу) на основе динамического связного списка.

Фактически, класс **StudList2** определяет объект типа «Элемент списка студентов», который является началом (головой) некоторого связного списка объектов типа «Элемент списка студентов» (согласно рекурсивному определению списка). Использование рекурсии при определении элемента списка выражается в следующем: поле **next** определяемого типа само принадлежит типу, который определяется:

```
public class StudList2 { //Элемент списка студентов
    // поля
    private Student stud; //информационная часть элемента списка (объект)
    private StudList2 next; // указатель (ссылка) на следующий элемент списка
```

Полный текст класса **StudList2** приведен ниже.

```
public class StudList2 { /* Сущность - Студенческая группа
    В данном классе список студентов реализован на основе
    динамического связанного списка*/

    // Класс описывает элемент динамического связанного списка
    // Каждый экземпляр данного класса (элемент списка)
    // является началом (головой)списка
    // следующих за ним связанных элементов списка

    // статическая переменная класса задает формат строки
    // описания объекта (общая для всех объектов класса)
    private final static String GROUP_FORMAT_STRING =
        "Группа: %s, число студентов: %3d";

    // поля
    private Student stud; //информационная часть элемента списка (объект)
    //Использование рекурсии при определении данных:
    private StudList2 next; // указатель (ссылка) на следующий элемент списка

    // методы-конструкторы (используется перегрузка в
    // зависимости от указанных параметров)
    public StudList2(){ //без параметров
        stud = null; next = null;
    }
    public StudList2(String name){ //передается название списка
        //для инициализации головы списка студентов
        stud = new Student();
        stud.setName(name); stud.setId(0); stud.setBall(0);
        next = null;
    }
    public StudList2(Student st){
        // инициализируем элемент списка для студента st
        stud=st; next=null;
    }

    // методы, определяющие поведение элемента списка студентов
    public void setName(String name){ //назначить имя списку (группе)
        stud.setName(name);
    }

    public String getName(){return stud.getName();} //вернуть имя списка
    public int getNum(){ // вернуть число студентов в списке
        //головой списка является текущий объект, т.е. объект,
        // запустивший метод (this)
```

```

    int num = 0; // число элементов
    StudList2 el = next; // перешли к элементу, на который указывает
                        // голова списка
    while (el != null){ // пока в списке есть элементы
        num++;
        el = el.next;
    }
    return num;
}

```

```

public Student getStud (int id){
    // вернуть из списка студента с заданным номером зачетки
    // головой списка является текущий объект
    if (next == null) return null; //если список пуст
    StudList2 el = next; // el - ссылка на очередной элемент списка
    while (el != null){ // пока в списке есть элементы
        if (el.stud.getId() == id) return el.stud; // студент найден
        // перейти к следующему элементу списка
        el=el.next;
    }
    return null; // студент не найден
}

```

```

// сформировать строку описания объекта
public String toString() {
    return String.format(GROUP_FORMAT_STRING,stud.getName(),getNum());
}

```

```

//добавить студента в список студентов
public boolean addStud(Student stud){
    // головой списка является текущий объект
    //студента нельзя добавить, если
    //его id уже упомянут в списке
    if (getStud(stud.getId()) != null) return false;
    StudList2 el = new StudList2(stud); // создали новый элемент списка
    if (next == null) next = el; // вставили в пустой список
    else { // вставить первым в непустой список
        StudList2 old = next;
        next = el;
        el.next = old;
    }
    return true;
}

```

```

//удалить студента из списка студентов
public boolean delStud(int id){
    // головой списка является текущий объект

```

StudList2 el = this; // el.next - очередной элемент списка

```

while (el.next != null) { // пока в списке есть элементы
    if (el.next.stud.getId() == id){ // студент найден
        el.next = el.next.next; // и удален
        return true;
    }
    // перейти к следующему элементу списка
    el = el.next;
}
return false; // студент не найден
}

```

//запрос:

//вернуть средний балл для списка студентов

```

public float avgBall(){
    // головой списка является текущий объект
    if (next == null) return 0;
    float avg = 0; int num = 0;
    StudList2 el = next; // el - очередной элемент списка
    while (el != null){ // пока в списке есть элементы
        avg = avg+el.stud.getBall();
        num = num+1;
        // перейти к следующему элементу списка
        el = el.next;
    }
    return avg/num;
}

```

//запрос:

//вернуть список студентов, у которых балл выше среднего

// из списка, головой которого является текущий объект

```

public StudList2 aboveAvgBall(){
    //возвращает ссылку на объект определяемого класса -
    //рекурсия при определении данных
    float avg = avgBall();
    // формируем новый список с головой head
    // студенты не дублируются
    StudList2 head = new StudList2(String.format
        ("Студенты, у которых балл выше среднего %.2f: ", avg));
    StudList2 el = next; // el - очередной элемент основного списка
    while (el != null){ // пока в списке есть элементы
        if (el.stud.getBall() > avg)
            head.addStud(el.stud);
        // перейти к следующему элементу списка
        el=el.next;
    }
    return head;
}

```

// запрос:

//вернуть список студентов с баллом в диапазоне [ball1,ball2]

// из списка, головой которого является текущий объект

```

public StudList2 betweenBall (float b1, float b2) {
    //возвращает ссылку на объект определяемого класса -
    //рекурсия при определении данных;
    //алгоритм аналогичен использованному в предыдущем методе
    // клонов соответствующих студентов запишем в новый список head,
    // т.е. в этом методе, в отличие от предыдущего, студенты
    // дублируются (оцените оба подхода!)
    StudList2 head = new StudList2 (String.format (
        "Студенты, у которых балл в диапазоне от %.2f до %.2f: ", b1, b2));
    StudList2 el = next; // el - очередной элемент основного списка
    while (el != null) { // пока в списке есть элементы
        float b = el.stud.getBall();
        if ((b >= b1) && (b <= b2))
            head.addStud(new Student(el.stud));
        // перейти к следующему элементу списка
        el = el.next;
    }
    return head;
}

//выводит список студентов в окно терминала
public void putStudList(){
    //головой списка является текущий объект
    System.out.println(stud.getName());
    System.out.printf("%5s%17s%11s%17s\n",
        "Номер","Номер зач. кн.,""ФИО","Балл");
    StudList2 el = next; // el - очередной элемент списка
    int i = 1; // номер студента
    while (el != null){ // пока в списке есть элементы
        System.out.printf(" %-7d %-15d %-16s %-10.2f\n",
            i, el.stud.getId(), el.stud.getName(), el.stud.getBall());
        // перейти к следующему элементу списка
        el = el.next;
        i=i+1;
    } //while
} // putStudList()
} //StudList2

```

Конструктор без параметров **public StudList2()** позволяет создать пустой элемент списка, т.е. элемент, не указывающий на конкретного студента (обе переменные экземпляра равны **null**).

Чтобы создать некоторый список студентов, нужно создать его «голову» на основе класса **StudList2** и объект класса **Student**, на который указывает «голова». При этом используется конструктор **public StudList2(String name)**. Он создает объект класса **Student** при помощи соответствующего конструктора без параметров, инициализирует поле **name** созданного объекта класса **Student** значением строки, передаваемой в качестве параметра. Таким образом, мы задаем и храним имя создаваемого списка.

Третий конструктор **public StudList2(Student st)** предназначен для создания элемента списка, указывающего на конкретного студента (на которого, в свою очередь, указывает ссылочная переменная **st**). Такой элемент можно добавить к

уже существующему списку, на который указывает некая «голова», используя метод **addStud()**.

Конструкторы будут перегружаться в зависимости от параметров вызова.

Методы, возвращающие простые типы и **void**, используют простейшие алгоритмы последовательного доступа к элементам связного списка.

Например, метод **public boolean addStud (Student stud)** позволяет текущему объекту (т.е. **this** – объекту класса «Элемент списка», запустившему метод), создать и добавить в начало списка, головой которого он является, новый элемент списка, ссылающийся на объект класса **Student**, переданный в качестве параметра). Сначала создается новый элемент списка: **StudList2 el = new StudList2(stud);**. Затем, если список, на который указывает текущий объект, пуст (**next == null**), то **next** получает значение ссылки **el**. В противном случае, в ссылочной переменной **old** сохраняется старое значение ссылки **next**, затем ссылочная переменная **next** получает значение ссылки **el**, после этого ссылочной переменной **el.next** присваивается значение ссылки **old**. Таким образом, новый элемент всегда вставляется в начало списка.

Заметим, что методы **aboveAvgBall ()** и **betweenBall ()** возвращают ссылку на элемент определяемого класса (**StudList2**). Здесь мы снова наблюдаем *рекурсию при определении данных*.

При разработке указанных методов, возвращающих ссылку на список студентов: **public StudList2 aboveAvgBall ()** и **public StudList2 betweenBall (float b1, float b2)**, было учтено следующее:

1) они возвращают ссылку на голову списка студентов, а, значит, соответствующую ссылочную переменную (**head**) и элемент-«голову», на который она будет указывать, сначала нужно создать;

2) к созданному списку (**head**) можно применять все методы, определенные в классе **StudList2**, в том числе, методы **addStud()** и **putStudList()** (преимущество рекурсивного определения).

Заметим, что такая характеристика списка студентов, как число студентов в списке, в классе **StudList2** является вычисляемой, причем каждый раз заново (проанализируйте текст метода **public int getStudNum()**). Можно использовать и другой подход, а именно, число студентов в списке хранить в поле **stud.id** (имеет подходящий тип **int**) головы списка, подобно тому, как в поле **stud.name**, мы договорились хранить название списка. Тогда задачи увеличения и уменьшения числа элементов в списке нужно возложить на методы **addStud** и **delStud** соответственно (как это сделано в классе **StudList1** в предыдущей лабораторной работе), а код метода **getStudNum()** будет выглядеть следующим образом: **public int getStudNum(){ return stud.id}**.

Заметим также, что при реализации студенческой группы на основе динамического связного списка не нужно фиксировать максимально возможное количество студентов в группе, поэтому отпадает необходимость в переменных **MIN_NUM** и **numMax**, которые использовались для реализации списка студентов на основе массива фиксированной длины.

Метод **main**, определенный в классе **StudList2Demo**, организующий взаимодействие студентов и групп, выполняет те же тестирующие действия, что и в предыдущей лабораторной работе.

```
public class StudList2Demo{ //Взаимодействие студентов и групп
public static void main (String[ ] args){
    StudList2 gr1 = new StudList2 ("ИВТБ-11д");
    StudList2 gr2 = new StudList2 ("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++;
        String name = args[j]; j++;
        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));
    }
    System.out.println ("Сформированы следующие группы:");
    System.out.printf ("%s,\n%s\n\n", gr1,gr2);
    // пытаемся вставить студента с тем же номером зачетки
    if (!gr1.addStud(new Student (32348, "Козлов",5.0f))) {
        System.out.printf (
            "В группу %s добавить студента 32348 не удалось:\n",
            gr1.getName());
        System.out.println(" нарушена уникальность ключа.\n");
    }
    //выборка данных
    gr1.putStudList();
    gr1.aboveAvgBall().putStudList();
    gr1.betweenBall(3.2f,4.2f).putStudList();
    // удаление студента по id
    gr1.delStud(12345);
    //выборка данных
    System.out.println("После удаления студента:");
    gr1.putStudList();
    System.out.println();
    // то же для второй группы:
    //выборка данных
    gr2.putStudList();
```

```

gr2.aboveAvgBall().putStudList();
gr2.betweenBall(3.2f,4.2f).putStudList();
// удаление студента по id
gr2.delStud(53347);
//выборка данных
System.out.println("После удаления студента:");
gr2.putStudList();
System.out.println();
//Проверяем,есть ли в группах студенты с заданным номером зачетки
System.out.println("Проверка наличия студентов:");
int n = 23345;
Student s1 = gr1.getStud(n);
System.out.printf ("В группе %s:\n",gr1.getName());
if (s1==null) System.out.printf (
    "Нет студента с номером зачетки %d\n",n);
else System.out.println(s1);
n = 70000;
s1 = gr1.getStud(n);
if (s1 == null) System.out.printf (
    "Нет студента с номером зачетки %d\n",n);
else System.out.println(s1);
n = 53349;
s1 = gr2.getStud(n);
System.out.printf("В группе %s:\n",gr2.getName());
if (s1 == null) System.out.printf (
    "Нет студента с номером зачетки %d\n",n);
else System.out.println(s1);
} //main
} // StudList2Demo
// Отпала необходимость в методе
// public static void putStudArr (Student [ ] arr). Объясните, почему.

```

Обратите внимание на участок кода:

```

gr2.aboveAvgBall().putStudList();
gr2.betweenBall(3.2f,4.2f).putStudList(); .

```

Поскольку методы **aboveAvgBall()** и **betweenBall()** класса **StudList2** возвращают ссылку на объект того же класса **StudList2** (голову списка), от имени возвращаемого объекта может быть запущен метод **putStudList()**, определенный в том же классе.

Результаты работы программы приведены на рисунках 3.12, 3.13.

```

BlueJ: Terminal Window - Lab_2_Tree_list

Options

Сформированы следующие группы:
Группа: ИВТБ-11д, число студентов: 5,
Группа: ИВТБ-12д, число студентов: 5

В группу ИВТБ-11д добавить студента 32348 не удалось:
нарушена уникальность ключа.

ИВТБ-11д


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


Студенты, у которых балл выше среднего 4,18:


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


Студенты, у которых балл в диапазоне от 3,20 до 4,20:


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


После удаления студента:
ИВТБ-11д


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


```

Рисунок 3.12 – Результаты работы программы

```

ИВТБ-12д
Номер  Номер зач. кн.      ФИО      Балл
  1      53349      Солнцев      3,40
  2      53348      Морев      5,00
  3      53347      Афанасьев      3,10
  4      53346      Ливанов      4,50
  5      53345      Федоров      4,10
Студенты, у которых балл 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
Студенты, у которых балл в диапазоне от 3,20 до 4,20:
Номер  Номер зач. кн.      ФИО      Балл
  1      53345      Федоров      4,10
  2      53349      Солнцев      3,40
После удаления студента:
ИВТБ-12д
Номер  Номер зач. кн.      ФИО      Балл
  1      53349      Солнцев      3,40
  2      53348      Морев      5,00
  3      53346      Ливанов      4,50
  4      53345      Федоров      4,10

Проверка наличия студентов:
В группе ИВТБ-11д:
Студент:  23345,      Иванов,  4,50
Нет студента с номером зачетки 70000
В группе ИВТБ-12д:
Студент:  53349,      Солнцев,  3,40

```

Рисунок 3.13 – Результаты работы программы (продолжение)

Работа методов класса **StudList2**, как и в предыдущей лабораторной работе, была протестирована для группы без студентов.

Для этого была создана копия проекта, в которой класс **StudList2Demo** был изменен следующим образом:

```

public class StudList2Demo{ //Взаимодействие студентов и групп
public static void main (String[ ] args){
    StudList2 gr1 = new StudList2 ("ИВТБ-11д");
    // группа создана, но студентов в ней нет
    // выборка данных
    // выводим списки студентов
    gr1.putStudList();
    gr1.aboveAvgBall().putStudList();
    gr1.betweenBall(3.2f,4.2f).putStudList();
    // удаление студента по id
    gr1.delStud(12345);
    System.out.println("После удаления студента:");
    gr1.putStudList();
    System.out.println();
    //Проверяем,есть ли в группе студент с заданным номером зачетки
    System.out.println("Проверка наличия студентов:");
    int n=23345;
    Student s1=gr1.getStud(n);
    System.out.printf("В группе %s:\n",gr1.getName());
    if (s1==null) System.out.printf (
        "Нет студента с номером зачетки %d\n",n);
    else System.out.println(s1);
} //main
} //StudList2Demo

```

Результаты тестирования приведены на рисунке 3.14. **Обратите внимание**, что при формировании списка студентов, у которых балл выше среднего, для группы, в которой нет студентов, метод **aboveAvgBall()** возвращает ссылку на голову пустого списка с названием **"Студенты, у которых балл выше среднего 0,00:"**. Если бы в описанном случае указанный метод возвращал бы **null** (пустую ссылку) при выполнении участка кода **gr1.aboveAvgBall().putStudList();** возникла бы ошибка: попытка запустить метод **putStudList()** от имени пустой ссылки. Аналогичным образом ведет себя и метод **betweenBall()**.

Для тестирования работы методов класса **StudList2** в случае наличия в списке группы только одного студента в классе **StudList2Demo**, текст которого приведен выше, было предусмотрено добавление одного студента в группу:

```

public class StudList2Demo{ //Взаимодействие студентов и групп
public static void main (String[ ] args) {
    StudList2 gr1 = new StudList2 ("ИВТБ-11д");
    gr1.addStud(new Student (23345, "Зайцев",4.1f));
    //группа создана и в нее добавлен один студент
    ...
} //main
} //StudList2Demo

```

Результаты тестирования приведены на рисунке 3.15.

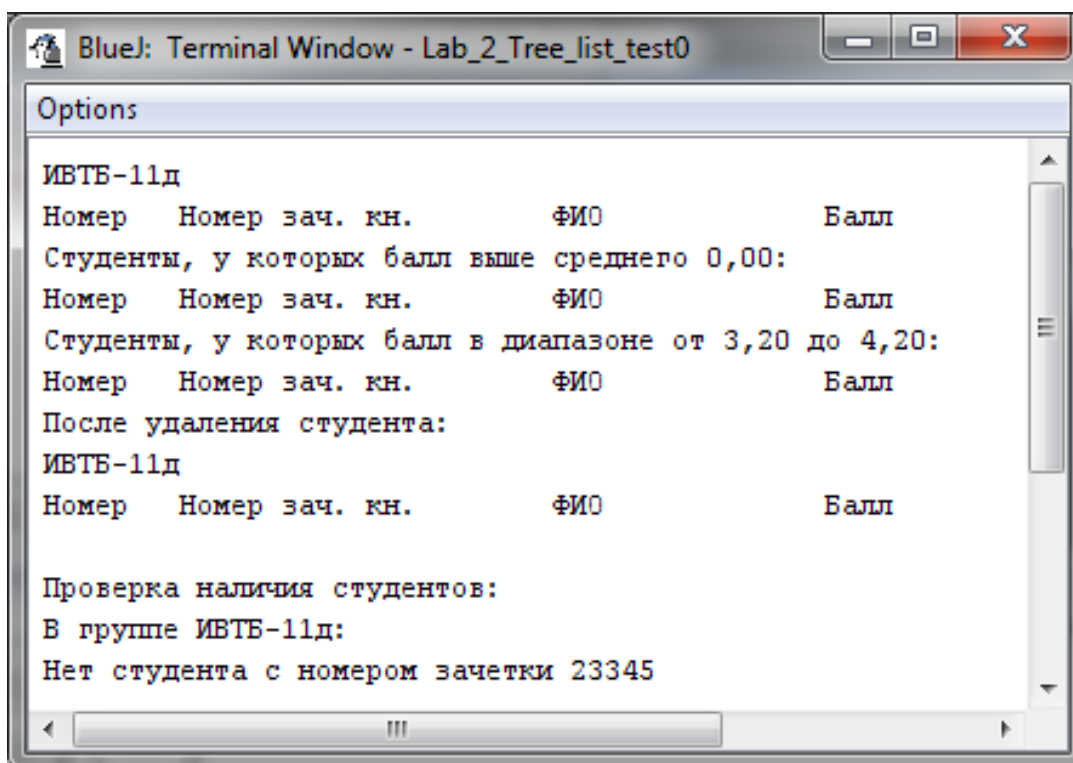


Рисунок 3.14 – Результаты тестирования программы для группы без студентов

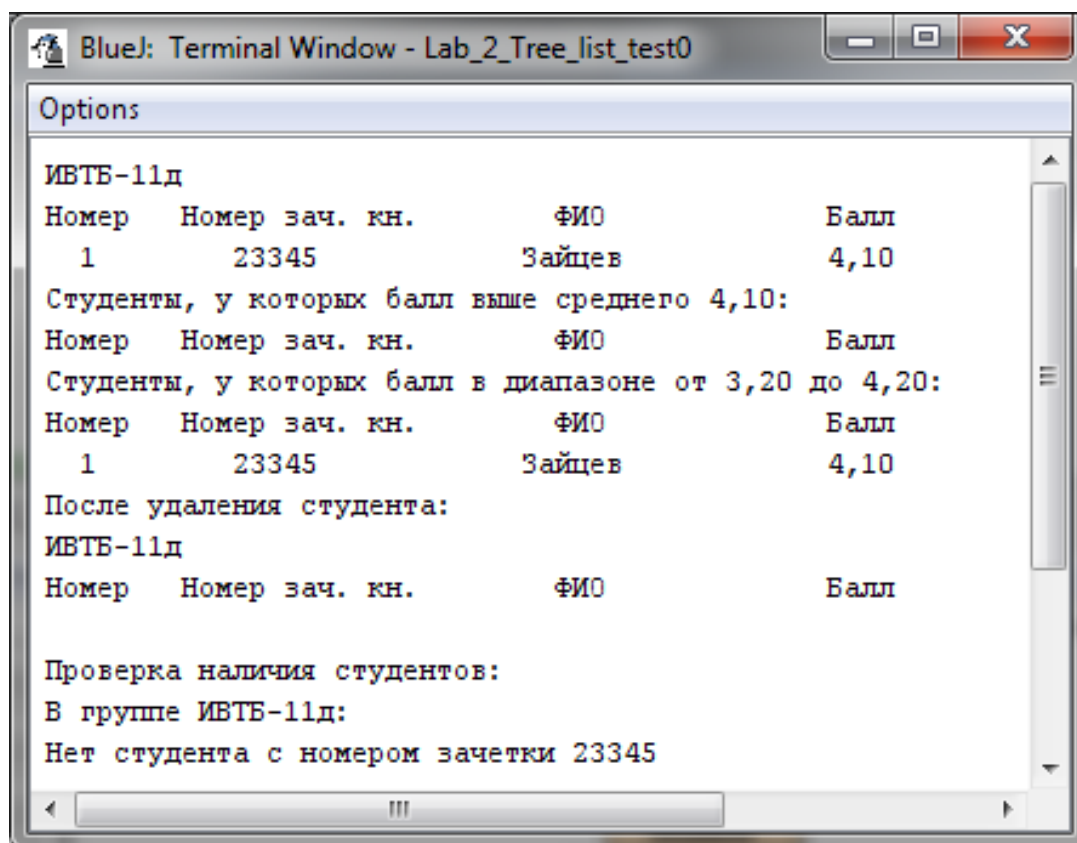


Рисунок 3.15 – Результаты тестирования программы для группы, включающей одного студента

Анализируя текст класса **StudList2**, заметим, что метод **public StudListEl aboveAvgBall()**, возвращающий ссылку на объект класса «элемент списка», являющийся «головой» соответствующего списка студентов, не увеличивает количество объектов класса **Student** в оперативной памяти (за исключением объекта, на который ссылается поле `stud` «головы» списка), т.к. он работает со ссылками на эти объекты.

Метод **public StudListEl betweenBall(float b1, float b2)**, в отличие от предыдущего, включает в создаваемый список копии объектов класса **Student** из основного списка группы, т.е. увеличивает число объектов класса **Student**.

В качестве примера на рисунке 3.16 показано размещение объектов в оперативной памяти для группы `gr1` после создания группы и выполнения метода `gr1.aboveAvgBall()`. Сравните данную структуру со структурой, изображенной на рисунке 2.13. Объясните, почему студенты присутствуют в списке группы в порядке, обратном порядку их добавления.

3.5. Порядок выполнения работы

- 1) Внимательно прочитайте п. 3.4 и разберите текст программы-примера.
- 2) Возьмите за основу описание классов проекта, выполненное в предыдущей лабораторной работе, и доработайте его с учетом того, что некоторые методы будут возвращать вместо ссылок на массивы ссылки на объекты класса «элемент списка».

Метод `main()`, определенный в отдельном классе, должен демонстрировать взаимодействие объектов, заданных вариантом задания аналогично тому, как это сделано в проекте, текст которого приведен в разделе 3.4, а результаты работы – на рисунках 3.12 – 3.15.

- 3) Согласно описанию разработайте код для каждого класса. Если программирование каких-либо методов класса вызовет затруднение, составьте алгоритм метода и приведите его в отчете.

- 4) Проведите отладку программы и проверку ее работы на тестовых примерах.

- 5) Текст программы и результаты ее работы приведите в отчете.

3.6. Варианты заданий

В качестве варианта задания возьмите вариант, который вы выполняли в предыдущей лабораторной работе.

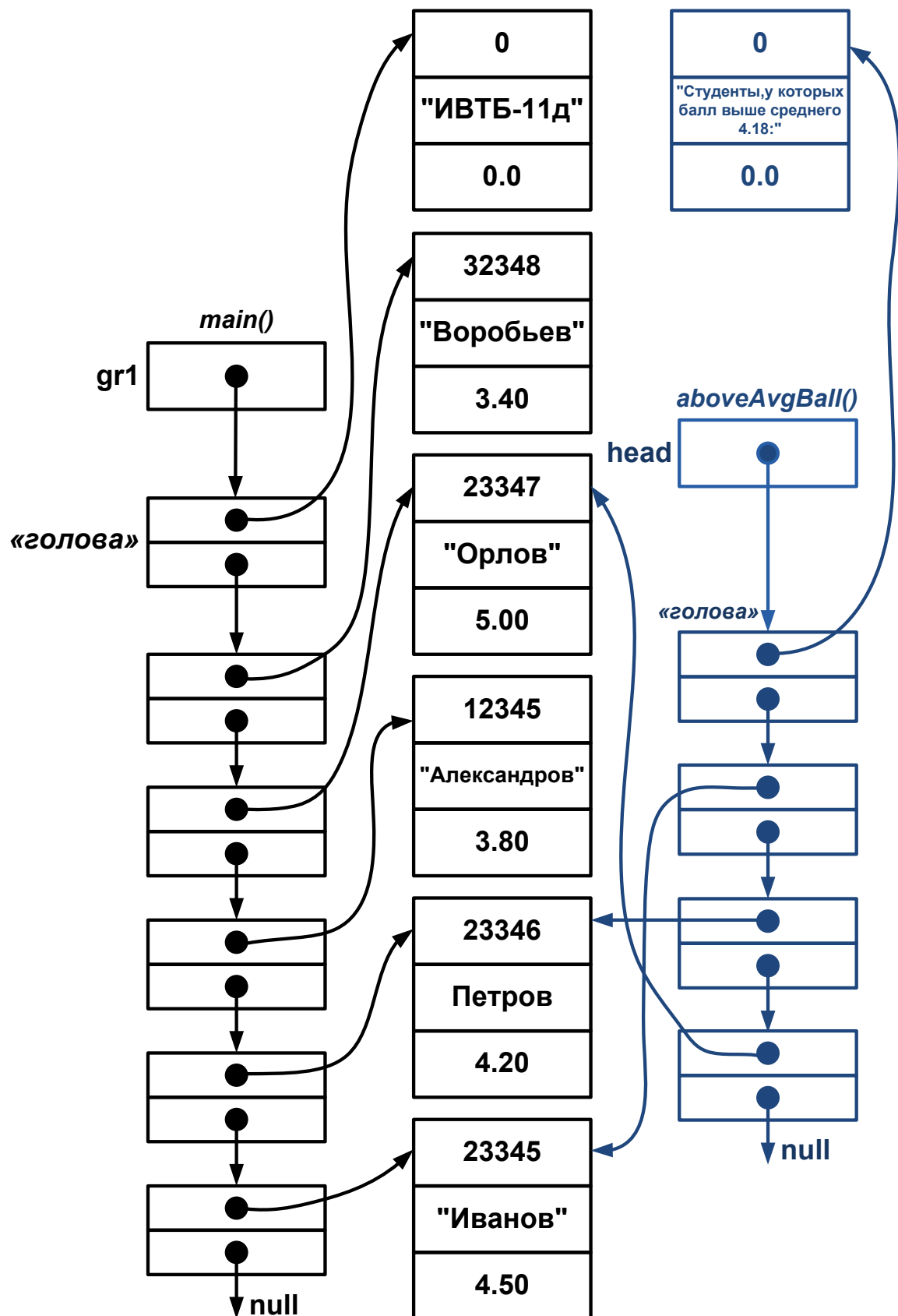


Рисунок 3.16 – Размещение объектов в оперативной памяти для группы gr1

3.7. Содержание отчета

- 1) Цель работы.
- 2) Постановка задачи с указанием данных варианта задания.
- 3) Иерархическая структура данных для сущности_2 (пример на рисунке 3.2).
- 4) Структура списка объектов после добавления в него 5 объектов (пример изображен на рисунке 4.16).
- 5) Структура проекта (пример на рисунке 3.3).
- 6) Описание классов проекта (пример – таблицы 3.1, 3.2).
- 7) Описание действий, выполняемых методом `main()`.
- 8) Текст программы.
- 9) Протокол отладки
- 10) Результаты работы программы.
- 11) Вывод.

3.8. Контрольные вопросы

- 1) Что представляет из себя линейный динамический односвязный список. В каком смысле здесь употребляется слово динамический?
- 2) Для реализации каких структур данных можно использовать связные списки?
- 3) Какие обязательные операции должны быть предусмотрены для списка?
- 3) Какие дополнительные операции могут быть предусмотрены для списка?
- 4) В чем преимущества и недостатки использования динамического списка для создания списка объектов по сравнению с использованием статического массива ссылок для той же цели? В какой из названных реализаций используется последовательный доступ к элементам списка, а в какой – прямой?
- 5) Какой принцип добавления и удаления объектов использует список, называемый стеком?
- 6) Что понимается под «головой» и «хвостом» списка? Какому списку было бы «трудно» без «головы» и почему?
- 7) Как опознать «хвост» списка?
- 8) Как найти элемент с заданным значением полей объекта в списке?
- 9) В чем состоит процедура добавления элемента к началу односвязного списка?
- 10) Как удалить из списка первый элемент?
- 11) Какие поля и методы необходимо задать в классе, определяющем элемент односвязного списка? Как используется рекурсия при определении элемента списка?
- 12) Как удалить из списка элемент с заданным значением полей объекта, на который он ссылается? Приведите пример из своего проекта.

13) Напишите методы для возврата элемента списка с заданным номером, добавления элемента в заданную позицию списка, удаления элемента из позиции с заданным номером, удаления последнего элемента списка.

14) Объясните, почему студенты присутствуют в списке группы (рисунки 4.12, 4.16) в порядке, обратном порядку их добавления?

15) Объясните, какие действия выполняет фрагмент программы из приведенного в разделе 3.4. примера:

```
gr2.aboveAvgBall().putStudList();  
gr2.betweenBall(3.2f,4.2f).putStudList();
```

и благодаря чему такая последовательность применения методов стала возможной.

16) Как бы вы определили элемент двусвязного списка, представленного на рисунке 4.17.



Рисунок 4.17 – Двусвязный список

17) Для предметной области, определенной вариантом задания, разработайте метод:

- а) добавляющий очередной объект в конец списка;
- б) добавляющий очередной объект в позицию списка с заданным номером;
- в) удаляющий первый объект из списка объектов;
- г) удаляющий последний объект из списка объектов;
- д) удаляющий объект из позиции списка с заданным номером;
- е) удаляющий из списка объектов все объекты, у которых значение неключевого числового поля меньше заданного;
- ж) возвращающий первый объект списка;
- з) возвращающий последний объект списка;
- и) возвращающий объект из заданной позиции списка;
- к) возвращающий список объектов, у которых значение неключевого числового поля больше заданного;
- л) осуществляющий копирование списка объектов (полная копия вместе с объектами);
- м) осуществляющий выборочное копирование списка объектов (полная копия вместе с объектами) – с указанием начальной позиции и количества элементов копируемой части;
- н) осуществляющий перестановку объектов списка в обратном порядке – reverse;
- о) осуществляющий перестановку двух объектов списка, стоящих в заданных позициях – swap;
- п) осуществляющий перестановку объекта, стоящего в заданной позиции списка и следующего за ним объекта – swapNext;
- р) осуществляющий сравнение двух списков – compareList.

4. ЛАБОРАТОРНАЯ РАБОТА № 5. ИСПОЛЬЗОВАНИЕ ИНТЕРФЕЙСОВ ДЛЯ УНИФИКАЦИИ РАБОТЫ С ПОЛИМОРФНЫМИ ОБЪЕКТАМИ, ИМЕЮЩИМИ РАЗНУЮ ВНУТРЕННЮЮ ПРИРОДУ

4.1. Цель работы

Целью данной работы является:

- 1) ознакомление с одним из принципов поддержки полиморфизма в Java («один интерфейс – много методов»);
- 2) получение навыков разработки и применения специальных классов Java – интерфейсов, описывающих функции некоторого семейства объектов, а также обычных классов, реализующих функции, заданные интерфейсом, для отдельных членов семейства (полиморфных объектов);
- 3) закрепление навыков использования рекурсии при определении данных в объектно-ориентированном программировании (определение в некотором классе метода, возвращающего ссылку на объект этого же класса).

4.2. Постановка задачи

Разработать программу, которая работает с полиморфными объектами, т.е. объектами, принадлежащими одному семейству, но имеющими разные реализации. Объект первого класса представляет собой группу объектов некоторого класса C, реализованную на основе статического массива ссылок, объект второго класса представляет собой группу объектов того же класса C, реализованную на основе динамического связного списка. Для работы с объектами первого и второго класса должен использоваться один и тот же интерфейс.

4.3. Внеаудиторная подготовка

Для подготовки к лабораторной работе следует изучить краткие теоретические сведения, приведенные в пункте 4.4 методических указаний. Рекомендуется также ознакомиться с [1] (с.230-242).

4.4. Краткие теоретические сведения

Применяя ключевое слово `interface`, вы можете полностью освободить интерфейс класса от его реализации. То есть, используя `interface`, вы можете указать, *что* класс должен делать, но не *как* он это делает. Интерфейсы синтаксически подобны классам, но в них нет экземплярных переменных, а их методы объявляются без тела. Как только интерфейс определен, реализовать его может любое число классов. Для реализации интерфейса класс должен создать полный набор методов, определенных интерфейсом. Однако каждый класс волен сам определять детали своей реализации. Ключевое слово `interface`

позволяет полностью использовать аспект полиморфизма, декларируемый как «один интерфейс, множественные методы». Интерфейсы разработаны для поддержки динамического вызова методов во время выполнения.

Определение интерфейса подобно определению класса. Общая форма интерфейса выглядит так:

```
access interface name {
    return_type method_name1(parameter_list);
    return_type method_name2(parameter_list);
    ...
    return_type method_nameN(parameter_list);
    type final_varname1=value;
    type final_varname2=value;
    ...
    type final_varnameM=value;
}
```

Здесь `access` – спецификатор доступа (или `public`, или не используется). Если спецификатор доступа не указан, тогда используется доступ по умолчанию, и интерфейс доступен только другим членам пакета, в котором он объявлен. При объявлении с `public` интерфейс может использоваться любым другим кодом.

`name` – имя интерфейса (любой допустимый идентификатор). Объявленные методы не имеют тел (объявление метода заканчивается точкой с запятой после списка параметров). Все методы интерфейса считаются общедоступными. Каждый класс, который включает (реализует) интерфейс, должен реализовать все его методы.

Внутри объявлений интерфейсов можно объявлять переменные. Они неявно считаются `final` и `static` (это означает, что они не могут быть изменены реализующим классом), а также должны быть инициализированы постоянными значениями. Это позволяет использовать интерфейсы для импорта разделяемых констант во множественные (реализующие интерфейс) классы.

Реализация интерфейсов.

Когда интерфейс определен, он может реализовываться одним или несколькими классами. Для реализации интерфейса в определение класса включают предложение с ключевым словом `implements` и затем создают методы, определенные в интерфейсе. Общая форма класса, который включает `implements`-предложение:

```
access class classname
    implements interface_name1 [, interface_name2 ...] {
    // тело класса;
    // в классе обязательно должны быть реализованы все методы
    // включаемых интерфейсов!
    // в заголовке этих методов должен быть употреблен
    // спецификатор доступа public!
    // в классе также могут быть реализованы и другие методы,
    // (не относящиеся к интерфейсам).
}
```

Если класс реализует более одного интерфейса, имена интерфейсов в предложении `implements` разделяются запятой. Если класс реализует два интерфейса, которые объявляют один и тот же метод, то клиенты любого интерфейса будут использовать один и тот же метод. Методы, которые реализуют интерфейс, должны быть объявлены как `public`. Кроме того сигнатура реализующего метода должна точно соответствовать сигнатуре метода, определенного в интерфейсе. Для классов, реализующих интерфейсы, обычно допустимо определять дополнительные собственные члены.

Реализация доступа к объектам через интерфейсные ссылки.

Можно объявлять ссылочные переменные, которые используют интерфейсный тип, а не тип класса. В такой переменной можно сохранять адрес объекта-экземпляра любого класса, который реализует объявленный интерфейс. Когда вы вызываете метод через ссылку такого рода, будет вызываться его правильная версия, основанная на актуальном экземпляре интерфейса. Это – одно из ключевых свойств интерфейсов. Выполняемый метод отыскивается динамически (во время выполнения), что позволяет создавать классы позже кода, который вызывает их методы (можно распараллелить процесс написания сложного проекта между несколькими программистами).

Поскольку динамический поиск метода во время выполнения приводит к существенной потере производительности по сравнению с нормальным вызовом (через ссылку на объект обычного класса), следует проявлять осторожность и беспричинно не использовать интерфейсы там, где нужна повышенная эффективность.

Пример. Покажем использование интерфейсов на примере задачи о студенческих группах, варианты решения которой рассмотрены в двух предыдущих лабораторных работах.

Напомним, что под студенческой группой понимается **список**, содержащий (в данный момент времени) некоторое число (`num`) объектов класса `Student`, имеющий заголовок (`name`). Со **списком** можно выполнять следующие операции:

- 1) получить заголовок (`name`) списка студентов;
- 2) установить заголовок (`name`) списка студентов;
- 3) получить количество студентов в списке;
- 4) получить студента с заданным `id`;
- 5) добавить студента с уникальным номером зачетки (`id`);
- 6) исключить студента с заданным `id`;
- 7) получить средний балл студента;
- 8) получить **список** (подсписок основного списка) студентов, имеющих балл выше среднего;
- 9) получить **список** (подсписок основного списка) студентов, имеющих балл, входящий в диапазон `[b1, b2]`;
- 10) вывести **список** (в окно терминала).

В предыдущих лабораторных работах рассмотрены две возможных реализации такого списка: на основе статического массива ссылок и на основе динамического связного списка.

Независимо от конкретной реализации списка группы, его функции можно задать следующим интерфейсом.

```
// Интерфейс задает методы (процедуры и функции)
// списка студентов, абстрагируясь от их реализации
public interface StudList {
    // 1) установить имя списка студентов:
    void setListName(String name);
    // 2) получить имя списка студентов:
    String getListName();
    // 3) получить количество студентов в списке:
    int getStudNum();
    // 4) получить из списка студента с заданным
    // номером зачетки id:
    Student getStud (int id);
    // 5) добавить студента stud в список:
    boolean addStud(Student stud);
    // 6) удалить из списка студента с заданным
    // номером зачетки id:
    boolean delStud(int id);
    // 7) вернуть средний балл для списка:
    float avgBall();
    // 8) вернуть список студентов (подсписок
    // основного списка), у которых балл выше
    // среднего:
    StudList aboveAvgBall();
    // 9) вернуть список студентов (подсписок
    // основного списка), у которых балл входит
    // в диапазон [b1, b2]:
    StudList betweenBall(float b1, float b2);
    // 10) вывести список студентов в окно терминала:
    void putStudList();
}
```

Заметим, что методы

StudList aboveAvgBall() и **StudList betweenBall(float b1, float b2);**

возвращают интерфейсную ссылку на объект типа «Список студентов», т.е. на объект, принадлежащий семейству объектов, функции которого мы определяем данным интерфейсом (снова наблюдаем рекурсию при определении данных). Другими словами, подсписок основного **списка студентов** группы (например, студентов, имеющих балл выше среднего по группе) в свою очередь является списком студентов, над которым можно выполнять все операции, определенные интерфейсом для списка (например, вычислять средний балл уже для этого списка студентов).

Метод (процедура) **void putStudList()** будет выводить в окно терминала запустивший его объект, принадлежащий семейству «Список студентов».

Текст классов, реализующих интерфейс приведен ниже.

Класс **StudList1** реализует список студентов (студенческую группу) на основе статического массива ссылок (массива фиксированной длины). Текст методов, возвращающих простые типы или void, взят без изменения из соответствующего класса **StudList1**, приведенного в методических указаниях к лабораторной работе №3 (п.2.4). Текст методов, возвращающих ссылку на список студентов, **public StudList1 aboveAvgBall()** и **public StudList1 betweenBall(float b1, float b2)**, несколько изменен с учетом того, что

1) они возвращают ссылку на список студентов, а, значит, соответствующую ссылочную переменную (list) и список, на который она будет указывать, сначала нужно создать;

2) к созданному списку (list) можно применять все методы, определенные в классе **StudList1**, в том числе, метод **addStud()**.

*// Класс реализует список студентов на основе статического массива
// ссылок и все методы интерфейса StudList*
public class StudList1 implements StudList {

...

//запрос:

//вернуть список студентов, у которых балл выше среднего

public StudList1 aboveAvgBall(){ *//возвращает подсписок:*

//возвращает ссылку на элемент определяемого класса -

//используется рекурсия при определении данных

float avg = avgBall();

//создаем подсписок

StudList1 list = new StudList1(String.format

("Студенты, у которых балл выше среднего %.2f: ", avg), num);

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

if (studArr[i].getBall() > avg) list.addStud(studArr[i]);

//число объектов типа Student не увеличивается,

//т.к. работаем со ссылками

return list;

}

//вернуть список студентов с баллом в диапазоне [ball1,ball2]

public StudList1 betweenBall(float b1, float b2){ *//возвращает подсписок:*

//возвращает ссылку на элемент определяемого класса -

//используется рекурсия при определении данных

//алгоритм аналогичен использованному в предыдущем методе

//создаем подсписок

StudList1 list = new StudList1(String.format(

"Студенты, у которых балл в диапазоне от %.2f до %.2f: ",b1,b2),num);

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

float b = studArr[i].getBall();

if ((b >= b1) && (b <= b2))

list.addStud(studArr[i]);

}

return list;

}

...

```
} //StudList1
```

Следующий класс **StudList2** реализует список студентов (студенческую группу) на основе динамического связанного списка. Текст класса **StudList2** подробно обсуждался в разделе 3.4 методических указаний к предыдущей лабораторной работе. Копируем его в проект и включаем в данный класс интерфейс **StudList**:

```
// Класс реализует список студентов на основе динамического
// связанного списка и все методы интерфейса StudList
public class StudList2 implements StudList {
    // Класс описывает элемент динамического связанного списка
    // Каждый экземпляр данного класса (элемент списка)
    // является началом (головой) списка
    // следующих за ним связанных элементов списка
    ...

} // StudList2
```

Структура проекта изображена на рисунке 4.1.

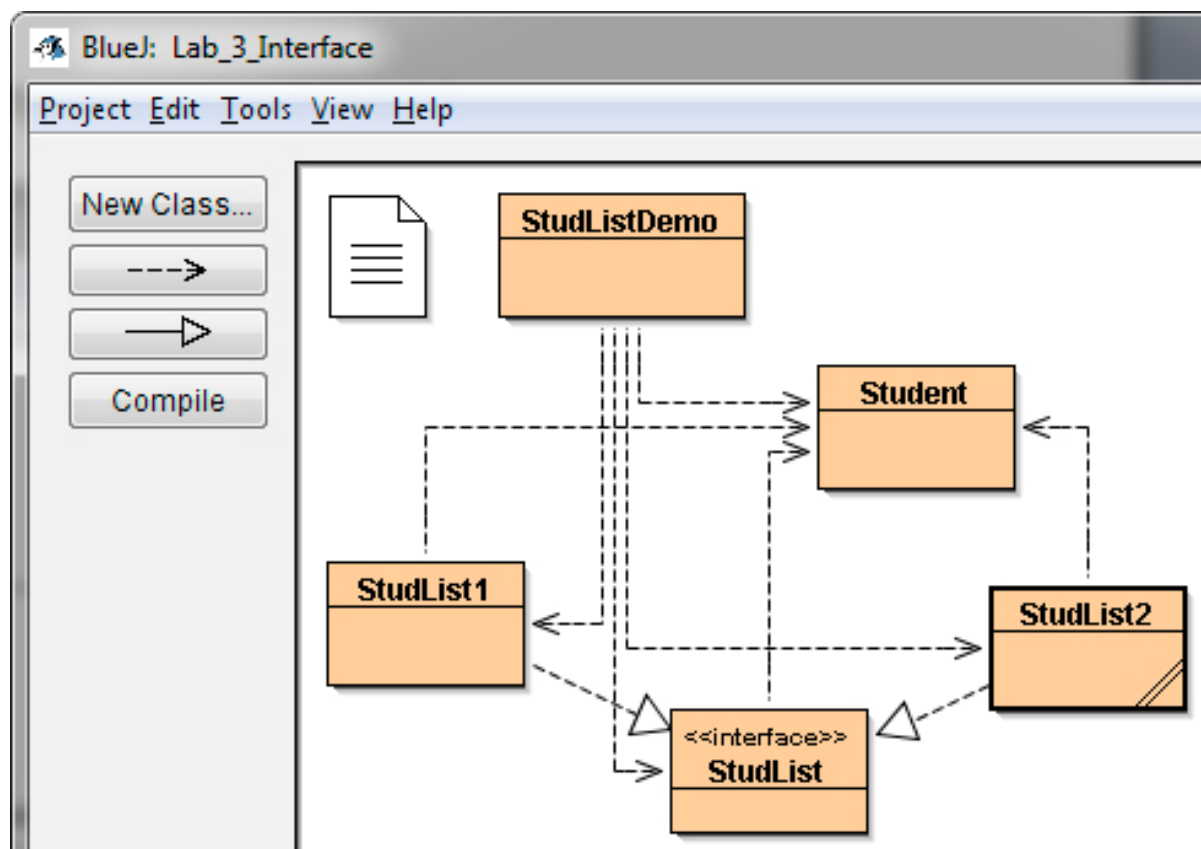


Рисунок 4.1 – Структура проекта

Текст класса **Student** приведен в методических указаниях к лабораторной работе №3 (раздел 2.4). Текст класса **StudListDemo**, осуществляющего

взаимодействие студентов и групп приведен ниже. Класс выполняет все те же тестирующие действия, что и подобный класс, приведенный в методических указаниях к предыдущей лабораторной работе. Отличие состоит в том, что в методе **main()** создаются две группы студентов, имеющих различную природу (реализацию). Первая группа **gr1** с названием "Академическая группа ИВТБ-11д" создана на основе класса **StudList1** (реализует статический массив ссылок на студентов), а вторая группа **gr2** является экземпляром класса **StudList2** (реализует динамический связный список). Ссылки **gr1** и **gr2** заданы как интерфейсные:

```
StudList gr1=new StudList1 ("Академическая группа ИВТБ-11д",5);  
StudList gr2=new StudList2 ("Спортивная секция по волейболу"); ,
```

но можно было бы их задать и как обычные ссылки на объекты соответствующих классов:

```
StudList1 gr1=new StudList1 ("Академическая группа ИВТБ-11д",5);  
StudList2 gr2=new StudList2 ("Спортивная секция по волейболу"); .
```

Достоинство интерфейсной ссылки в ее универсальности. Например, если бы мы захотели создать в программе ссылку **gr3** и определили ее как интерфейсную, то следующий код не вызвал бы ошибки:

```
StudList gr3;  
gr3 = gr1; gr3.putStudList();  
gr3 = gr2; gr3.putStudList(); .
```

В коде, приведенном ниже, компилятор обнаружил бы синтаксическую ошибку (как вы думаете, в каком операторе?):

```
StudList1 gr3;  
gr3 = gr1; gr3.putStudList();  
gr3 = gr2; gr3.putStudList(); .
```

Поскольку оба класса (**StudList1** и **StudList2**) включают (implements) интерфейс **StudList**, работа с объектами этих классов унифицирована с помощью методов, определенных интерфейсом. При этом особенности реализации этих объектов скрыты внутри реализующих классов. Пользователю (в данном случае разработчику класса **StudListDemo**) достаточно знать:

1) существует реализация интерфейса **StudList**, с помощью которой можно создавать группы с ограниченным числом студентов (класс **StudList1**) и реализация того же интерфейса, с помощью которой можно создавать группы без ограничения на число студентов (класс **StudList2**);

2) сигнатуру конструкторов классов **StudList1** и **StudList2**;

3) сигнатуру методов, определенных интерфейсом **StudList**.

Проанализируйте код метода **main()** и убедитесь в том, что вызовы методов для объекта, на который указывает ссылка **gr1**, ничем не отличаются от соответствующих вызовов для объекта, на который указывает ссылка **gr2**.

Заметим, что класс **StudListDemo** содержит еще один метод:

public static void SetGroupStatus (StudList group, String status). Этот метод позволяет приписать к названию группы студентов **group** определенный статус, заданный параметром **status**, например: "не сертифицирована", "сертификат А", "сертификат Б". Формальный параметр **group** задан как интерфейсная ссылка (т.е. ссылка на объект любого класса, реализующего интерфейс **StudList**). Это значит, что в данный метод могут быть переданы как объекты, созданные на основе класса **StudList1**, так и объекты, созданные на основе класса **StudList2**. Внутри метода **SetGroupStatus** осуществляются вызовы методов, определенных интерфейсом **StudList** для списка студентов, на который указывает интерфейсная ссылка **group**. Вызовы метода **SetGroupStatus** для групп **gr1** и **gr2** продемонстрированы в заключительной части кода метода **main()**.

Текст класса **StudListDemo**:

```
public class StudListDemo{ // Взаимодействие студентов и групп

    public static void SetGroupStatus (StudList group, String status){
        // назначить группе статус
        // метод демонстрирует использование интерфейсной ссылки
        // в качестве параметра и возможность использования методов,
        // определенных в интерфейсе, абстрагируясь от их внутренней
        // реализации
        group.setName(group.getName()+" " + status);
    }

    public static void main (String[ ] args){
        // создаем группы студентов
        StudList gr1 = new StudList1 ("Академическая группа ИВТБ-11д",5);
        StudList gr2 = new StudList2 ("Спортивная секция по волейболу");
        // Далее работаем с интерфейсными ссылками gr1 и gr2.

        // 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++;
            String name = args[ j ]; j++;
            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));
}
System.out.println ("Сформированы следующие группы:");
System.out.printf ("%s,\n%s\n\n", gr1, gr2);
//пытаемся вставить лишнего студента в первую группу
if (!gr1.addStud(new Student (60538, "Акимов",4.3f))) {
    System.out.printf (
        "В группу %s добавить студента 60538 не удалось:\n",
        gr2.getName());
    System.out.println("превышен лимит.");
}
// пытаемся вставить студента с тем же номером зачетки
// во вторую группу
if (!gr2.addStud(new Student (53347, "Козлов",5.0f))) {
    System.out.printf (
        "В группу %s добавить студента 32348 не удалось:\n",
        gr1.getName());
    System.out.println("нарушена уникальность ключа.");
}
System.out.println();
//выборка данных
// выводим списки студентов для группы gr1
gr1.putStudList();
gr1.aboveAvgBall().putStudList();
gr1.betweenBall(3.2f,4.2f).putStudList();
// удаление студента по id
gr1.delStud(12345);
//выводим список группы gr1 после удаления студента
System.out.println ("После удаления студента:");
gr1.putStudList();
System.out.println();
// то же для второй группы:
// выводим списки студентов для группы gr2
gr2.putStudList();
gr2.aboveAvgBall().putStudList();
gr2.betweenBall(3.2f,4.2f).putStudList();
// удаление студента по id
gr2.delStud(53345);
//выводим список группы gr2 после удаления студента
System.out.println ("После удаления студента:");
gr2.putStudList();
System.out.println();
//Проверяем,есть ли в группах студенты
// с заданным номером зачетки
System.out.println("Проверка наличия студентов:");
int n = 23345;
Student s1 = gr1.getStud(n);
System.out.printf("В группе %s:\n",gr1.getName());
if (s1 == null) System.out.printf (
    "Нет студента с номером зачетки %d\n",n);
else System.out.println(s1);
n = 70000;

```

```

s1 = gr1.getStud(n);
if (s1 == null) System.out.printf (
    "Нет студента с номером зачетки %d\n",n);
else System.out.println(s1);
n = 53349;
s1 = gr2.getStud(n);
System.out.printf("В группе %s:\n",gr2.getName());
if (s1 == null) System.out.printf (
    "Нет студента с номером зачетки %d\n",n);
else System.out.println(s1);
System.out.println();
//назначаем группам статус
//для групп, реализуемых разными классами, используем один и
//тот же метод SetGroupStatus(), использующий в качестве
// параметра интерфейсную ссылку
SetGroupStatus(gr1, "сертификат А");
SetGroupStatus(gr2, "не сертифицирована");
System.out.println ("После назначения группам статуса имеем:");
System.out.println (gr1.getName()+";");
System.out.println (gr2.getName()+".");

} //main
} //StudListDemo

```

Результаты работы проекта представлены на рисунках 4.2, 4.3.

4.5. Порядок выполнения работы

1) Внимательно прочитайте раздел 4.4 и разберите текст программы-примера.

2) По аналогии с программой-примером разработайте интерфейс и реализующие его классы для своего варианта задания. Текст класса, реализующего список объектов на основе статического массива ссылок, можно взять из лабораторной работы №3, внося соответствующие изменения в методы, возвращающие ссылку на список объектов. Текст класса, реализующего список объектов на основе динамического связного списка, можно взять из лабораторной работы № 4. Не забудьте включить в классы реализуемый ими интерфейс.

Метод **main()**, определенный в отдельном классе, должен демонстрировать взаимодействие объектов, заданных вариантом задания аналогично тому, как это сделано в проекте, текст которого приведен в разделе 4.4, а результаты работы – на рисунках 4.2, 4.3.

В том же классе, что и метод **main**, должен быть определен *еще один метод, демонстрирующий использование интерфейсной ссылки в качестве формального параметра* (аналогично тому, как это сделано в примере).

3) Проведите отладку программы и проверку ее работы на тестовых примерах.

5) Текст программы и результаты ее работы приведите в отчете.

```

Options

Сформированы следующие группы:
Группа: Академическая группа ИВТБ-11д, число студентов: 5,
Группа: Спортивная секция по волейболу, число студентов: 5

В группу Спортивная секция по волейболу добавить студента 60538 не удалось:
превышен лимит.
В группу Академическая группа ИВТБ-11д добавить студента 32348 не удалось:
нарушена уникальность ключа.

Академическая группа 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 |


Студенты, у которых балл выше среднего 4,18:


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


Студенты, у которых балл в диапазоне от 3,20 до 4,20:


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


После удаления студента:
Академическая группа 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 4.2 – Результаты работы программы

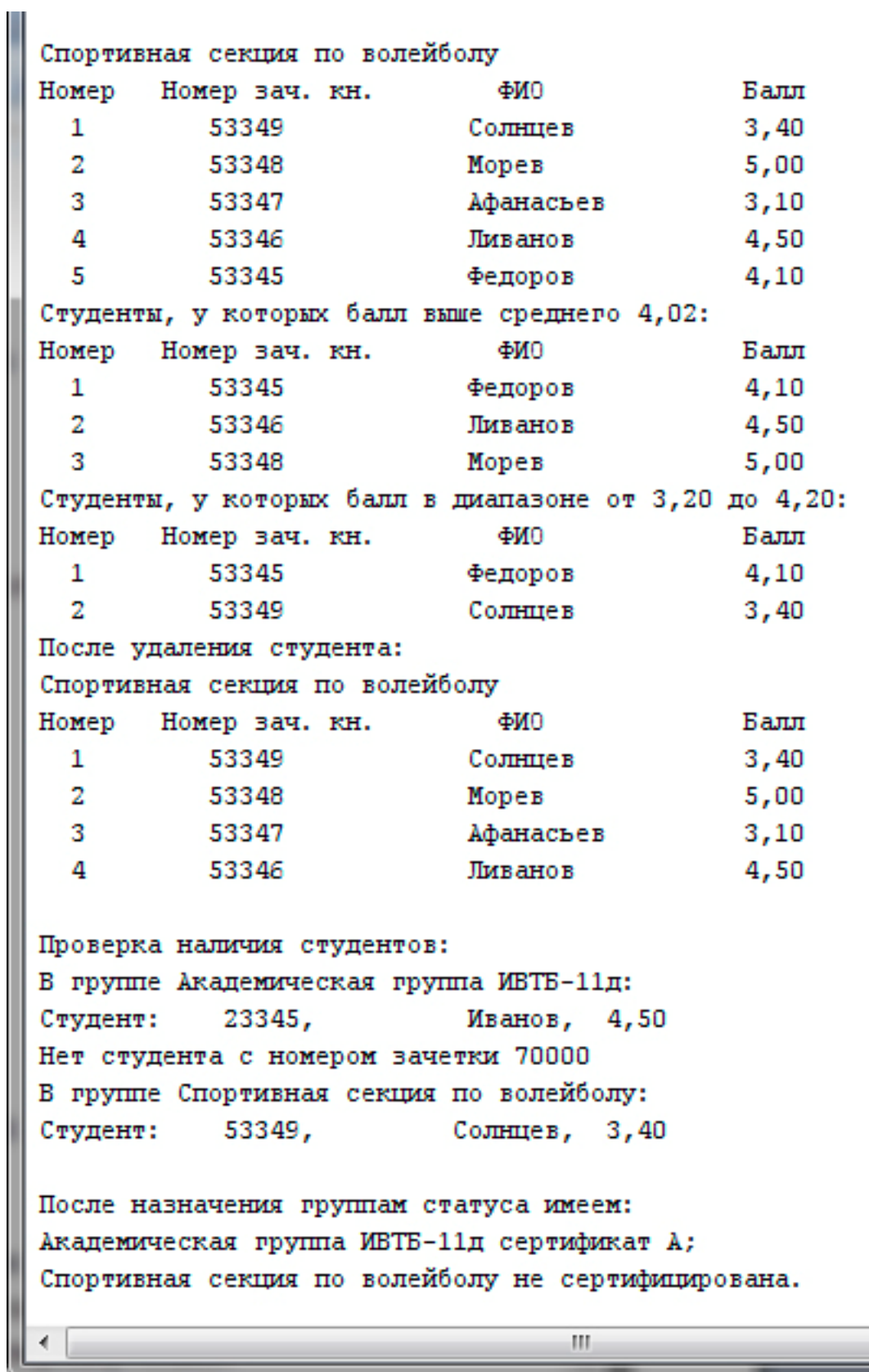


Рисунок 4.3 – Результаты работы программы (продолжение)

4.6. Варианты заданий

В качестве варианта задания возьмите вариант, который вы выполняли в предыдущей лабораторной работе.

4.7. Содержание отчета

- 1) Цель работы.
- 2) Постановка задачи с указанием данных варианта задания.
- 5) Структура проекта (пример на рисунке 4.1).
- 6) Описание интерфейса.
- 8) Текст программы.
- 9) Протокол отладки.
- 10) Результаты работы программы.
- 11) Вывод.

4.8. Контрольные вопросы

- 1) В чем состоит принцип полиморфизма в объектно-ориентированном программировании?
- 2) Для чего используют интерфейсы?
- 3) Как задать интерфейс? Какой спецификатор доступа должен быть у интерфейса?
- 4) Могут ли у интерфейса быть переменные экземпляра?
- 5) Как использовать интерфейс для задания именованных констант, разделяемых несколькими классами?
- 6) Что представляет из себя класс, реализующий (включающий) интерфейс? Как его задать? Какие нужно использовать спецификаторы доступа в реализациях методов, определенных включаемым интерфейсом.
- 7) Сколько интерфейсов может включать класс? Сколько методов из включаемых интерфейсов должно быть реализовано в классе?
- 8) Могут ли в классе, включающем интерфейс, быть реализованы методы, не определенные в интерфейсе?
- 9) Для чего вы использовали интерфейс в своей программе?
- 10) Назовите аспект полиморфизма, который в Java поддерживается за счет интерфейсов? Где вы ранее сталкивались с полиморфизмом в Java?
- 11) Что такое интерфейсная ссылка? Как ее можно использовать? Как создать объект, на который может указывать интерфейсная ссылка?
- 12) Как и когда осуществляется выбор актуального метода, вызванного через интерфейсную ссылку? Приведите пример из своей программы.
- 13) Объясните преимущества использования интерфейсной ссылки в качестве параметра метода.

- 14) Допустим, что классы проекта, описанного в разделе 5.4, параллельно разрабатывают разные программисты (сотрудники соответствующей фирмы). Что нужно знать разработчику класса **StudListDemo** о других классах проекта? Сформулируйте тот же вывод для вашего проекта.
- 15) Когда не нужно использовать интерфейсы?
- 16) Где и как в вашем проекте использована рекурсия при определении данных?
- 17) Как вы думаете, какой из проектов, посвященных спискам объектов, разработанных вами в лабораторных работах 3, 4, 5, наиболее соответствует принципам объектного программирования и почему?
- 18) Добавьте в интерфейс и реализуйте в соответствующих классах дополнительные методы для работы со списком объектов (по согласованию с преподавателем).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ноутон П. Java 2: /П.Ноутон, Г.Шилдт. – СПб.: БХВ-Петербург, 2007. – 1072 с.
2. Вирт. Н. Алгоритмы и структуры данных. Новая версия для Оберона+CD / Н. Вирт. – М.: ДМК Пресс, 2010. – 272 с.
3. Связный список. – Электронный ресурс. – Режим доступа: [http://ru.wikipedia.org/wiki/Связный список](http://ru.wikipedia.org/wiki/Связный_список).

ПРОГРАММИРОВАНИЕ. ЯЗЫКИ ВЫСОКОГО УРОВНЯ

Часть 2

**Создание списков. Использование рекурсии при определении данных.
Реализация принципа полиморфизма с помощью интерфейсов Java**

Методические указания

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

В авторской редакции

Изд. № 78/2020. Объем 5 п.л.
РИИЦМ ФГАОУ ВО «Севастопольский государственный университет»