МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра информационных технологий и компьютерных систем

ПРОГРАММИРОВАНИЕ. ЯЗЫКИ ВЫСОКОГО УРОВНЯ

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

В пяти частях

Часть 2

Использование принципа наследования – создание суперклассов и подклассов. Взаимодействие объектов разных классов в java-программе

Севастополь
СевГУ
2023

УДК 004.432(076.5)
ББК 32.973-018.1я73
П784

Р е ц е н з е н т :
В. Ю. Карлусов - канд. техн. наук, доцент кафедры информационных систем

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

П784 **Программирование. Языки высокого уровня** [В 5-ти частях]. Часть 2. Использование принципа наследования – создание суперклассов и подклассов. Взаимодействие объектов разных классов в java-программе : методические указания к лабораторным работам по дисциплине для студентов направления 44.03.05 – Педагогическое образование (с двумя профилями подготовки. Профиль: математика и информатика) дневной формы обучения. / Севастопольский государственный университет, Институт информационных технологий, кафедра информационных технологий и компьютерных систем ; сост. Т. В. Волкова, Е. С. Владимирова. – Севастополь : СевГУ, 2023. – 62 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 наследования и инкапсуляции в рамках объектно-ориентированного программирования на языке Java в системе программирования BlueJ.

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

УДК 004.432(076.5)
ББК 32.973-018.1я73

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

СОДЕРЖАНИЕ

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. РАЗРАБОТКА ПОДКЛАССОВ ПЕРЕОПРЕДЕЛЕНИЕ МЕТОДОВ В ПОДКЛАССАХ. ИСПОЛЬЗОВАНИЕ БИБЛИОТЕЧНОГО ИНТЕРФЕЙСА COMPARABLE

2.1. Цель работы

Целью данной работы является ознакомление с реализацией принципа наследования в Java, получение навыков в проектировании классов-шаблонов, позволяющих осуществлять сравнение соответствующих объектов на равенство и больше/меньше/равно.

2.2. Постановка задачи

2.2.1. Проект1: Разработать в соответствии с вариантом задания класс (Класс1), позволяющий создавать объекты, которые можно сравнивать на равенство (использовать переопределение методов equals(), hashCode() и toString() суперкласса Object) и на больше/меньше/равно. Разработать класс (Класс1Demo), демонстрирующий сравнение и упорядочивание (сортировку) объектов класса Класс1.

2.2.2. Проект 2: Разработать подкласс Класс2 класса Класс1, позволяющий создавать объекты, которые можно сравнивать на равенство (использовать переопределение методов equals(), hashCode() и toString() суперкласса Класс1) и на больше/меньше/равно. Разработать класс (Класс2Demo), демонстрирующий сравнение и упорядочивание (сортировку) объектов класса Класс2. Продемонстрировать также возможность обращения к объектам класса Класс2 с помощью ссылочной переменной суперкласса (Класс1).

2.2.2. Проект 3: Скопировать проект Проект2 и включить в классы Класс1 и Класс2 библиотечный интерфейс Comparable. Разработать класс (ComparableDemo), демонстрирующий использование интерфейсной ссылки для унификации действий при сортировке любых объектов (в данном случае объектов классов Класс1 и Класс2), включающих интерфейс Comparable.

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.189-218, 410).

2.4. Краткие теоретические сведения

2.4.1. Принцип наследования и его реализация в Java

Принцип наследования:

1) создается родительский класс, который определяет свойства, общие для нескольких подклассов;

2) на базе родительского класса создается несколько более специфических подклассов (дочерних классов), каждый из которых наследует все свойства родительского класса и добавляет к ним характерные для него свойства.

Родительский класс называется суперклассом, дочерний – подклассом.

Преимущество наследования: можно использовать готовый, качественно разработанный код, создавая на его основе дочерние классы (добавляя специфические свойства и методы) [1, 2].

Чтобы унаследовать класс B от класса A нужно включить в заголовок класса B предложение extends A:

class B extends A {…} *//класс B расширяет класс A*

Общая форма определения подкласса:

```
class subclass-name extends superclass-name {
        //тело класса
  }
```

Если вы создаете класс с нуля (предложение extends в заголовке класса отсутствует), то такой класс будет являться наследником класса Object.

Доступ к членам суперкласса

Подкласс имеет доступ ко всем public-членам своего суперкласса.

Хотя подкласс включает все элементы (члены) своего суперкласса, он не может обращаться к тем элементам суперкласса, которые были объявлены как private. Если в коде есть подобное обращение, возникнет ошибка компиляции.

Действует общее правило: член класса, который был объявлен как private доступен только членам этого класса.

Чтобы сделать член класса доступным только его прямым потомкам, нужно объявить его со спецификатором доступа protected [3 – 6].

Роль суперкласса не означает, что этот класс не может использоваться сам по себе, т.е. на базе суперкласса, как и на базе подкласса, можно создавать объекты.

Подкласс одного класса может быть суперклассом для другого класса (таким образом строятся иерархии классов).

Можно определять только один суперкласс для любого подкласса. Java, в отличие от C++ не поддерживает множественного наследования (наследования множества суперклассов одним подклассом).

Отсутствие данной возможности компенсируется тем, что класс может включать несколько интерфейсов.

Класс не может быть суперклассом для самого себя.

Ссылочная переменная суперкласса может ссылаться на объект подкласса (любого класса, производного от данного суперкласса). При этом будет возможно обращение только к тем частям объекта, которые определены суперклассом.

Ссылочной переменной класса Object можно назначать ссылку на объект любого класса, т.к. Object – суперкласс для всех классов (стоит в вершине иерархии классов).

Подкласс может обратиться к своему суперклассу с помощью ключевого слова super.

Использование ключевого слова super

1) Вызов конструктора суперкласса;
2) Решение проблемы сокрытия полей суперкласса полями подкласса с совпадающими именами (подобно использованию this для похожей цели).

Вызов конструктора суперкласса

Из конструктора подкласса можно вызывать конструктор суперкласса для инициализации полей, определенных в суперклассе.

Преимущества:

1) код конструктора суперкласса не дублируется в конструкторе подкласса;
2) Можно использовать для создания подклассов суперкласс, который хранит свои компоненты данных как private (сохраняет подробности своей реализации для себя в полном соответствии с принципом инкапсуляции).

Подкласс может вызвать метод-конструктор, определенный его суперклассом при помощи следующей общей формы оператора super:

super (parameter-list);

где parameter-list – список параметров (определяет параметры, необходимые конструктору суперкласса.

Такой оператор должен выполняться первым в конструкторе подкласса (для инициализации полей, определенных в суперклассе) . Затем должны быть инициализированы поля, определенные в подклассе.

Суперкласс может иметь несколько конструкторов, отличающихся набором параметров (вспомним перегрузку методов в Java). Метод super будет вызывать тот конструктор суперкласса, который подходит по набору параметров.

Один из возможных конструкторов – конструктор, создающий копию (клон) объекта-параметра, например:

```
//Построить клон объекта ob (Box – суперкласс, BoxWeight – подкласс)
BoxWeight (BoxWeight ob) {
    super(ob);
    weight = ob.weight;
 }
```

Можно передать в конструктор суперкласса объект подкласса, но суперкласс распознает в этом отделе только свои собственные члены.

Если слово super в начале конструктора подкласса не указано, то для инициализации полей, определенных в суперклассе используется конструктор суперкласса, заданный по умолчанию (им является конструктор без параметров).

Когда подкласс вызывает super() , он обращается к конструктору своего непосредственного суперкласса. В многоуровневой иерархии классов родительские конструкторы вызываются последовательно от низшего уровня к высшему (от подкласса к суперклассу), а выполняются, соответственно, в обратном порядке (т.е. в порядке подчиненности классов – от суперкласса к подклассу).

Если конструктор суперкласса требует наличия параметров, то все подклассы должны передать эти параметры «вверх по линии», и это не зависит от того, нуждается ли подкласс в своих собственных параметрах.

Решение проблемы сокрытия полей суперкласса полями подкласса с совпадающими именами (подобно использованию this для похожей цели).

В классе (например, в библиотечном классе Frame – окно) может быть скрыто множество полей, каждое из которых имеет имя. Нам не нужно знать все эти имена, чтобы не повторить их для полей создаваемого нами подкласса. Совпадение имен полей подкласса и его суперкласса допускается.

Чтобы поле подкласса не закрывало доступ к полю суперкласса с таким же именем, для обращения к полю суперкласса используется super.

Общий формат такого использования super:

super.member ,

где member – член суперкласса (поле или метод).

Переопределение методов родительских классов в классах-потомках

В иерархии классов, если метод в подклассе имеет такое же имя и сигнатуру типов, как метод в его суперклассе, то говорят, что метод в подклассе переопределяет (override) метод в суперклассе. Когда переопределенный метод вызывается в подклассе, он будет обращаться к версии метода, определенной подклассом. Версия метода, определенного суперклассом будет скрыта.

Если нужно обратиться из подкласса, переопределившего метод, к версии этого метода, определенной в суперклассе, то нужно использовать ключевое слово super перед именем метода.

Переопределение методов происходит только тогда, когда сигнатуры этих методов идентичны. Если у методов подкласса и суперкласса имена одинаковые, а наборы параметров и/или типы параметров разные, то происходит простая перегрузка методов. Это значит, что метод суперкласса не будет скрыт в подклассе методом с тем же именем, и из подкласса можно будет

обращаться и к методу подкласса и к методу суперкласса. Какой из них будет вызван, зависит от набора параметров вызова.

Методы **hashCode()** и **equals()** определены в классе Object, который является родительским классом для объектов java. Поэтому все java-объекты наследуют от этих методов реализацию по умолчанию.

Использование методов hashCode() и equals()

Метод hashCode() используется для получения уникального целого номера для данного объекта. Когда необходимо сохранить объект как структуру данных в некой хэш-таблице (такой объект также называют корзиной — bucket), этот номер используется для определения его местонахождения в этой таблице.

Если в Java Вы используете объект собственного класса в качестве наполнителя для коллекций HashSet, HashMap, HashTable или любых других коллекций, которые хранят объекты в группах, то Вам необходимо переопределить метод hashCode(). Это необходимо для правильной и более эффективной работы с коллекциями. Также всегда необходимо переопределять метод hashCode() если Вы переопределили метод equals().

По умолчанию, метод hashCode() для объекта возвращает номер ячейки памяти, где объект сохраняется (адрес объекта).

Метод hashCode() возвращает значение int (4 байта), которое является числовым представлением объекта. Этот хэш-код используется, например, коллекциями для более эффективного хранения данных и, соответственно, более быстрого доступа к ним.

Для одного и того же объекта метод hashCode() должен возвращать одно и то же значение в течении всей "жизни" объекта.

Также при переопределении метода equals() необходимо помнить, что для одинаковых объектов следующее условие должно быть истинно:

object1.hashCode() == object2.hashCode()

Если у разных объектов одинаковый хэш, то это может говорить о том, что вычисление хэш-кода не является эффективным. Но это не всегда истинное утверждение. Например, если у вас несколько миллионов или миллиардов разных экземпляров одного и того же класса, то вероятность того, что встретятся два объекта с одинаковым хэш-кодом очень высока.

В общем случае, разный хэш для разных объектов не является обязательным требованием. У одинаковых же объектов хэш-коды должны совпадать обязательно! [5, 6].

Следуйте следующим общепринятым правилам при переопределении метода hashCode():

1. Присвойте результирующей переменной (result) некоторое ненулевое простое число (например, 17).
2. Если поле value имеет тип boolean, вычислите (value ? 0 : 1).
3. Если поле value имеет тип byte, char, short или int, вычислите (int)value.
4. Если поле value имеет тип long, вычислите (int)(value - (value >>> 32))

5. Если поле value имеет тип float, вычислите Float.floatToIntBits(value)

6. Если поле value имеет тип double, вычислите Double.doubleToLongBits(value), а затем преобразуйте полученное значение, как указано в п.4

7. Если поле value является ссылкой на объект, вызывайте метод hashCode() этого объекта.

8. Если поле value является ссылкой на объект и равно null, используйте число 0 для представления его хэш-кода

9. Объедините полученные в п. 2 - п. 8 значения следующим образом: 37*result + value

10. Если поле является массивом, примените правило 9 для каждого элемента массива

11. Проверьте, что равные объекты возвращают одинаковый hashCode.

Метод **equals()**, как и следует из его названия, используется для проверки равенства двух объектов.

В классе Object метод equals определен так:

```
public boolean equals(Object obj) {
        return (this == obj);
}
```

Т.е. просто сравниваются две ссылки . Как правило, это не то, что нужно.

Обычно два объекта считаются равными, если основные их данные равны. Чтобы это понимали не только мы, но и компилятор, нам и нужно переопределять метод equals в своих классах.

Существует несколько свойств, которым должен удовлетворять переопределенный метод equals, а именно:

1) Сравниваться должны объекты одного класса.

2) Для любого не null объекта *x* x.equals(null) должен возвращать false.

Для не null-объектов метод должен обеспечивать:

3) Рефлексивность. Для любого объекта *x* x.equals(x) должен возвращать true.

4)Симметричность. Если для каких-либо объектов *x* и *y* x.equals(y) возвращает true, то и y.equals(x) тоже должен возвращать true.

5)Постоянство. Для любых объектов *x* и *y* x.equals(y) возвращает одно и то же значение, если информация, используемая в сравнениях, не меняется.

6)Транзитивность. Для любых объектов *x, y* и *z,* если x.equals(y) вернет true и y.equals(z) вернет true, то и x.equals(z) должен вернуть true.

Как было сказано выше, в общем случае необходимо переопределить метод hashCode всякий раз, когда переопределяется метод equals, таким образом, чтобы поддерживать общее соглашение о методе hashCode, в котором говорится, что равные объекты должны иметь равные хэш-коды.

Динамическая диспетчеризация методов

Методика переопределения методов формирует основу для одной из наиболее мощных концепций Java – динамической диспетчеризации методов (ДДМ).

ДДМ – механизм, с помощью которого решение на вызов переопределенной функции принимается во время выполнения, а не во время компиляции. Таким образом Java реализует полиморфизм времени выполнения.

Ссылочная переменная суперкласса может указывать на объект подкласса. У одного суперкласса может быть несколько подклассов, в которых переопределяются методы суперкласса. Когда переопределенный метод вызывается через ссылку суперкласса, Java определяет, какую версию этого метода следует выполнять, основываясь на типе объекта, на который указывает ссылка в момент вызова. Это определение делается во время выполнения.

Комбинируя наследование с переопределением методов, суперкласс может определять общую форму методов, которая будет использоваться всеми его подклассами. Способность существующих кодовых библиотек вызывать методы на экземплярах новых классов без перекомпиляции и при поддержке ясного абстрактного интерфейса - мощный инструмент языка Java (обеспечивает возможность многократного использования кода и устойчивость к ошибкам).

Динамический (во время выполнения) выбор методов производится также при использовании интерфейсов: когда мы обращаемся к методу, определенному в интерфейсе, через ссылочную переменную интерфейса вызывается метод того реализующего класса, на основе которого создан объект, на который указывает ссылочная переменная интерфейса (рассмотрим позже).

2.4.2. Библиотечный интерфейс Comparable

Интерфейсы используются для унификации работы с полиморфными объектами, имеющими разную внутреннюю природу.

Различают класс-интерфейс и класс, реализующий интерфейс.

В классе-интерфейсе определяются только заголовки методов. Класс, включающий (реализующий) интерфейс должен содержать конкретную реализацию всех методов, определенных в интерфейсе.

У одного класса-интерфейса может быть несколько реализующих классов. Унификация (единообразие) работы с объектами, принадлежащими к разным реализующим классам достигается за счет использования интерфейсных ссылок на объекты (будет показано в проекте 3).

Для унификации работы с объектами, при сравнении их на меньше/больше/равно можно использовать библиотечный интерфейс **Comparable <Type>** (параметр **Type** – класс объектов, которые нужно сравнить).

В этом интерфейсе определен один метод:

int compareTo(Type ob);

Метод возвращает целочисленное значение – результат сравнения, который интерпретируется так:

меньше нуля – вызывающий объект меньше объекта-параметра;

больше нуля – вызывающий объект больше объекта-параметра;

равен нулю – вызывающий объект равен объекту-параметру.

Метод сравнивает два объекта одного класса Type, реализующего интерфейс Comparable<Type> (вызывающий объект и объект-параметр должны принадлежать одному классу Type).

Объекты, созданные на основе различных классов Type1 и Type2, реализующих интерфейс Comparable (соответственно, Comparable<Type1> и Comparable<Type2>), являются полиморфными представителями сущности «объект, имеющий возможность сравнения на меньше/больше/равно». Это значит, что для их обработки, связанной со сравнением, может быть использован один и тот же метод, в который указанные объекты передаются через интерфейсную ссылку, например,

public static void sort (Comparable [ ] A){…} (проект 3).

2.4.3. Смешанное наследование

Язык Java непосредственно поддерживает только единичное наследование (у подкласса может быть только один суперкласс). Для моделирования множественного наследования можно использовать интерфейсы. Подкласс может быть наследником одного суперкласса и реализовывать несколько родительских интерфейсов. Иногда это называют смешанным наследованием, поскольку методы интерфейсов, которые реализует родительский класс, переопределяются (по законам наследования) в подклассах. В Проекте3 (Lab_1_3_sem2), описанном в разделе 2.8.3, приведен пример смешанного наследования.

2.5. Порядок выполнения работы

2.5.1. Внимательно изучите структуру и код проекта Проект1(Lab_1_1_sem2), приведенного в разделе 2.8.1.

Создайте собственный (***новый***!) проект с именем Lab_1_1_Фамилия_Группа. По аналогии с примером создайте в этом проекте в соответствии с вариантом задания класс (Класс1), позволяющий создавать объекты, которые можно сравнивать на равенство (использовать переопределение методов equals(), hashCode() и toString() суперкласса Object) и на больше/меньше/равно.

Обратите внимание на то, что реализация метода toString() в примере не использует операцию конкатенации символьных строк (это позволяет избежать «замусоривания» памяти неиспользуемыми объектами).

Можно скопировать Класс1 из предыдущей лабораторной работы и внимательно отредактировать его по аналогии с примером.

Разработайте класс (Класс1Demo), демонстрирующий сравнение и упорядочивание (сортировку) объектов класса Класс1. Действуйте по аналогии с примером.

2.5.2. Внимательно изучите структуру и код проекта Проект2 (Lab_1_2_sem2), приведенного в разделе 2.8.2.

Создайте собственный (***новый!***) проект с именем Lab_1_2_Фамилия_Группа. Скопируйте в данный проект из предыдущего проекта Класс1.

По аналогии с примером разработайте подкласс Класс2 класса Класс1, позволяющий создавать объекты, которые можно сравнивать на равенство (переопределите методы equals(), hashCode() и toString() суперкласса Класс1) и на больше/меньше/равно (переопределите метод compareTo() суперкласса Класс1).

Разработайте класс (Класс2Demo), демонстрирующий сравнение и упорядочивание (сортировку) объектов класса Класс2.

Продемонстрируйте также возможность обращения к объектам класса Класс2 с помощью ссылочной переменной суперкласса (Класс1). Действуйте по аналогии с примером.

2.5.3. Внимательно изучите структуру и код проекта Проект3 (Lab_1_3_sem2), приведенного в разделе 2.8.3.

Собственный (***новый!***) Проект3 создайте следующим образом: сделайте копию предыдущего проекта (Lab_1_2_Фамилия_Группа) и назовите ее Lab_1_3_Фамилия_Группа.

В созданном проекте включите в класс Класс1 библиотечный интерфейс Comparable (тогда метод compareTo(), определенный в классе Класс1, будет считаться реализацией этого интерфейса).

Метод compareTo(), переопределенный в подклассе Класс2, будет считаться реализацией интерфейса Comparable, наследуемого от суперкласса (пример так называемого смешанного наследования).

Разработайте класс (ComparableDemo), демонстрирующий использование интерфейсной ссылки для унификации действий при сортировке любых объектов (в данном случае объектов классов Класс1 и Класс2), реализующих интерфейс Comparable. Действуйте по аналогии с примером.

2.6. Варианты заданий

Данные для разработки классов «Класс1» и «Класс2» (название реализуемой сущности и ее свойства) приведены в таблице 2.1. Поведение сущностей характеризуют следующие методы:

1) конструктор без параметров;

2) конструктор с параметрами, задающими значения всех полей;
3) конструктор с параметром, строящий копию объекта того же класса, переданного как параметр;
4) методы-геттеры;
5) методы-сеттеры;
6) метод toString(), выдающий строку описания объекта;
7) метод equals(), сравнивающий объект с другим объектом того же класса (переданным в метод как параметр) на равенство. *Внимание! В предыдущей лабораторной работе в классе Класс1 использовался метод equals(), перегружающий одноименный метод класса-родителя Object (сигнатуры методов отличались). В данной работе необходимо использовать переопределение метода в подклассе* ***(сформулируйте, чем отличается перегрузка методов от переопределения методов).***
8) Метод hashCode() возвращает значение int (4 байта), которое является числовым представлением объекта (равные объекты должны иметь равные хэш-коды);
9) Метод compareTo(), сравнивающий объект с другим объектом того же класса (переданным в метод как параметр) на меньше/больше/равно;
10) Только для класса Класс1: метод, вычисляющий значение некоторого признака объекта (в примере – метод area()). Содержание метода задано в таблице 2.1.

Название поля, которое нужно добавить в подклассе Класс2 класса Класс1, также задано в таблице 2.1.

Номер варианта задания V необходимо вычислить по формуле

```
int  V = (N<=14) ? variant : N%14;
```

где N – номер студента в списке группы,

variant – номер варианта задания в таблице 2.1.

Таблица 2.1 – Поля (свойства) сущностей, участвующих в проекте

№	Класс1 (сущность 1)	Поля (private)	Метод 10 класса Класс1 возвращает	Поле класса Класс2, расширя-ющее Класс1 (private)
1	Книга	Идентификационный код, название, автор, издательство, год выпуска, количество страниц, тип (1 - учебная, 2 - научная, 3 - художественная), количество экземпляров	Общее количество страниц (число страниц, умноженное на количество экземпляров)	Цена

Продолжение таблицы 2.1

№	Класс1 (сущность 1)	Поля (private)	Метод 10 класса Класс1 возвращает	Поле класса Класс2, расширя-ющее Класс1 (private)
2	Телевизор	Фирма, серия, длина, ширина, высота, вес	Занимаемый объем (произведение длины, ширины, высоты)	Тип экрана (ЭЛТ, LCD, QLED)
3	Семья аквариумных рыбок	Вид, количество особей, рекомендуемый объем воды на 1 рыбку	Общий объем воды (произведение объема на 1 рыбку на число рыбок)	Цена
4	Занятие	Дисциплина, вид занятия (лекция, лабораторное или практическое), число студентов, число методичек	Количество методичек на 1 студента (число студентов, деленное на число методичек)	Число преподава-телей
5	Партия товара	Название товара, код товара, размер партии, тип товара (продукт, моющее средство, ткань), цена единицы товара.	Цена партии (размер партии, умноженный на цену единицы товара)	Рейтинг товара (целое число)
6	Зал	Номер, количество мест, есть мультимедийное оборудование?, есть электронная система голосования?, площадь зала	Площадь, приходящаяся на 1 место (площадь, деленная на число мест).	Количество компьютеров
7	Команда	Название, вид спорта, город, страна, лига, количество штрафных очков, число чемпионатов	Среднее количество штрафных очков за чемпионат (число штрафных очков, деленное на число чемпионатов)	Количество забитых голов
8	Фирма	Название, область деятельности, число сотрудников, минимальная зарплата сотрудника, рейтинг	Минимальный фонд заработной платы (произведение числа сотрудников и минимальной зарплаты)	Город

Продолжение таблицы 2.1

№	Класс1 (сущность 1)	Поля (private)	Метод 10 класса Класс1 возвращает	Поле класса Класс2, расширя-ющее Класс1 (private)
9	Семья	Число взрослых, число детей, есть кошка или собака?, предполагаемая сумма оплаты за квартиру.	Сумма оплаты на одного человека (предполагаемая сумма оплаты, деленная на число взрослых и детей)	Общая площадь квартиры
10	Проездной билет	Город, тип (универсальный, троллейбус, автобус, трамвай), стоимость, льготный?, период действия (число месяцев)	Средняя стоимость проезда за месяц (стоимость, деленная на период действия)	Материал (картон, пластик, ламиниро-ванный)
11	Водитель	Идентификационный номер, фамилия, категория (A, B, C, D, M), возраст, зарплата, надбавка	Общий заработок (зарплата плюс надбавка)	Стаж
12	Турпоездка	Город отправления, город прибытия, вид транспорта (автобус, поезд, самолет), дата прибытия, максимальная стоимость билета.	Максимальная стоимость билетов (туда и обратно): удвоенная максимальная стоимость билета.	Длитель-ность (число дней)
13	Розетка	Код, евро? цвет, мощность, напряжение.	Сила тока (мощность, деленная на напряжение)	Заземление?
14	Растение	Название, тип (травянистое, кустарник, дерево), минимальная температура воздуха, максимальная температура воздуха, требование к увлажнению почвы (интенсивное, среднее, редкое)	Средняя температура воздуха (сумма максимальной и минимальной температуры, деленная на 2)	Редкое?

2.7. Содержание отчета

1) Цель работы.
2) Постановка задачи с указанием данных варианта задания.
3) Структура проекта Проект1.
4) Текст классов.
5) Протокол отладки (описание найденных синтаксических и логических ошибок в программе и способов их устранения).
6) Результаты выполнения программы и их анализ.
7) Структура проекта Проект2.
8) Текст классов.
9) Протокол отладки (описание найденных синтаксических и логических ошибок в программе и способов их устранения).
10) Результаты выполнения программы и их анализ.
11) Структура проекта Проект3.
12) Текст классов.
13) Протокол отладки (описание найденных синтаксических и логических ошибок в программе и способов их устранения).
14) Результаты выполнения программы и их анализ.
15) Выводы.

2.8. Примеры проектов

Описание сущностей, подлежащих программированию, приведено в таблице 2.2.

Таблица 2.2 – Информация для программирования классов

№	Класс1 (сущность 1)	Поля (private)	Метод 10 класса Класс1 возвращает	Поле класса Класс2, расширяющее Класс1 (private)
1	Крышка	Диаметр, материал, цвет, выпуклая?	Площадь крышки	Тип ручки

2.8.1. Проект1 (Lab_1_1_sem2)

Структура проекта приведена на рисунке 2.1.

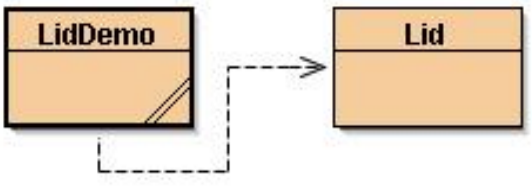

Рисунок 2.1 – Структура проекта Lab_1_1_sem2

В классе Lid (Класс1) описана сущность «Крышка» (круглая, без ручки). Класс LidDemo демонстрирует работу с объектами класса Lid.

Текст программы:

```
public class Lid { //Сущность - Крышка
   //формат записи о крышке:
   private final static String LID_FORMAT_STRING =
      "Крышка: диаметр=%5.2f см, %15s, %15s, %15s";
   //поля - переменные экземпляра (скрыты в классе)
   private double d; //диаметр в сантиметрах
   private String material; //материал
   private String color; //цвет
   private boolean convex ; //выпуклая?
   //методы-конструкторы (используется механизм перегрузки)
   //конструктор без параметров (используется по умолчанию)
   public Lid(){
     d = 0.0;  material = "";
     color = ""; convex = false;
   }
   //конструктор с параметрами, инициализирующими поля объекта
   public Lid(double d, String material, String color,boolean convex){
     this.d = d; this.material = material;
     this.color = color; this.convex = convex;
   }
   //конструктор с параметром-объектом (конструирует клон объекта)
   public Lid(Lid li) {
    d = li.d; material = li.material;
    color = li.color; convex = li.convex;
   }

   //доступ к полям
   //методы-геттеры
   public double getD(){return d;}
   public String getMaterial(){return material;}
   public String getColor(){return color;}
   public boolean getConvex(){return convex;}

   //методы-сеттеры
   public void setD(double d){this.d = d;}
   public void setMaterial(String material)
                              {this.material = material;}
   public void setColor(String color){this.color = color;}
   public void setConvex(boolean convex)
                        {this.convex = convex;}

   //Сравнение объектов на равенство (используется
   //механизм переопределения метода в классе-наследнике)
   //Переопределяется метод equals класса Object
   @Override  //инструкция компилятору: проверять, действительно ли
    // методы переопределены (а не просто перегружены)
   public boolean equals (Object ob){
```

```
    //возвращает true при равенстве текущего объекта
    //и объекта-параметра
        if (ob == null) return false;
        if (ob == this) return true;
        if (getClass() != ob.getClass()) return false;
        Lid li = (Lid) ob;
        return (d == li.d) && material.equals(li.material) &&
              color.equals(li.color) && (convex == li.convex);
    }
    //Переопределяется метод hashCode класса Object
    // Этот метод обязательно должен быть переопределен, если
    // был переопределен метод equals
    public int hashCode () { //возвращает хэш-код объекта
      return 7 * (new Double(d)).hashCode()
            + 11 * material.hashCode()
            + 13 * color.hashCode()
            + 17 * (new Boolean(convex)).hashCode();
    }
    //Переопределяется метод toString класса Object
    public String toString(){ //возвращает строку описания объекта
            return String.format(LID_FORMAT_STRING,
                    d,material,color,convex?"выпуклая":"плоская");
    }
    //Сравнение объектов на больше/меньше производится:
   // сначала по диаметру,
   //если диаметры равны, по материалу,
   //если диаметры и материалы равны, по цвету,
   //если диаметры, материалы и цвета равны, по выпуклости.
    public int compareTo (Lid li){
    /*возвращает отрицательное число, если текущий объект
    меньше объекта-параметра, ноль, если текущий объект равен
    объекту-параметру, положительное число, если текущий объект
    больше объекта-параметра*/
        if  (d < li.d) return -1;
        if ((d == li.d) &&
           (material.compareTo(li.material)<0)) return -1;
        if ((d == li.d) &&
           (material.compareTo(li.material)== 0) &&
           (color.compareTo(li.color)< 0)) return -1;
        if ((d == li.d) &&
           (material.compareTo(li.material)== 0) &&
           (color.compareTo(li.color)== 0) &&
            !convex && li.convex) return -1;
        if ((d == li.d) &&
           (material.compareTo(li.material)== 0) &&
           (color.compareTo(li.color)== 0) &&
            convex == li.convex) return 0;
        else return 1;
    }
    //другие полезные методы
    public double area() {return (Math.PI*d*d/4);} //площадь крышки
} //Lid
```

```
/*Обратите внимание:
1) Для сравнения String-переменных в методе equals() класса Lid
используется не операция отношения (==), а метод equals(),
переопределенный для класса String. Этот способ сравнения гарантирует,
что сравниваться будут именно символьные строки, а не адреса объектов
класса String.
2) В классе String реализован метод compareTo( ), который позволяет
лексикографически (т.е. в соответствии с расположением очередного
символа в кодовой таблице) сравнивать две строки на меньше/равно/больше
*/

public class LidDemo { // Демонстрирует работу с крышками
  public static void bubbleSort (Lid [ ] arr) {
  //сортировка массива объектов класса Lid методом пузырька
        boolean flag;
        for (int m = arr.length-1; m > 0; m--){
           flag = true;
           for (int j = 0; j < m; j++)
              if (arr[j].compareTo(arr[j+1]) > 0) {
                 Lid b = arr[j]; // перестановка – переприсвоение ссылок
                 arr[j] = arr[j+1];
                 arr[j+1] = b;
                 flag = false;
               }
           if (flag) break;
        }
   } // пузырек

   public static void putLidArr (Lid[ ] arr) { //вывод массива объектов класса Lid
      for (int i = 0; i < arr.length; i++)
        System.out.printf ("%s, хэшкод: %d\n", arr[i], arr[i].hashCode());
   }
  public static void main (String args[ ]){
     //создание объектов (крышек)
     /* 1.Создаем крышку lid0 с помощью конструктора без параметров.
     демонстрируем применение сеттеров для установки значений полей.
     Демонстрируем работу геттера getD() и метода area()
      */
     Lid lid0 = new Lid();
     lid0.setD(50); lid0.setMaterial("стеклянная");
     lid0.setColor("прозрачная"); lid0.setConvex(true);
     System.out.printf("Lid0: диаметр = %.2f, площадь равна %.2f\n",
                              lid0.getD(), lid0.area());
     /* 2. Создаем крышки lid1 - lid4 и пустую ссылку lid5.
     Демонстрируем работу методов toString(), equals() и hashCode()
      */
     Lid lid1 = new Lid(30.5,"стеклянная","темная",false);
     Lid lid2 = lid1; //создание псевдонима
     Lid lid3 = new Lid(lid1); //создание клона
```

```
        //Создание крышки с другими значениями полей
        Lid lid4 = new Lid(30.5,"эмалированная","серебристая",true);
        Lid lid5 = null;

        //Вывод информации о созданных крышках
        System.out.printf ("lid1: %s, хэшкод: %d\n", lid1, lid1.hashCode());
        System.out.printf ("lid2: %s, хэшкод: %d\n", lid2, lid2.hashCode());
        System.out.printf ("lid3: %s, хэшкод: %d\n", lid3, lid3.hashCode());
        System.out.printf ("lid4: %s, хэшкод: %d\n", lid4, lid4.hashCode());
        System.out.printf ("lid5: %s\n", lid5);

        //Вывод результатов сравнения на равенство
        System.out.println ("Результаты сравнения на равенство");
        System.out.printf ("lid1==lid2: %s\n", lid1.equals(lid2)); //true
        System.out.printf ("lid1==lid3: %s\n", lid1.equals(lid3)); //true
        System.out.printf ("lid1==lid4: %s\n", lid1.equals(lid4)); //false
        System.out.printf ("lid4==lid5: %s\n", lid4.equals(lid5)); //false

        /* Создаем массив крышек и демонстрируем применение
         метода compareTo() для упорядочивания элементов массива
        (метод compareTo() используется в методе bubbleSort())
         */
        Lid [ ] arr = new Lid [6];
        arr[0] = new Lid(30.5, "металлическая","золотистая",true);
        //отличается от arr[0] выпуклостью:
        arr[1] = new Lid(30.5, "металлическая","золотистая",false);
        //отличается от arr[0] цветом:
        arr[2] = new Lid(30.5, "металлическая","серебристая",true);
        //отличается от arr[0] материалом:
        arr[3] = new Lid(30.5, "керамическая","серебристая",true);
        //отличается от arr[0]  диаметром:
        arr[4] = new Lid(15.5, "металлическая","золотистая",true);
        //не отличается от arr[0]
        arr[5] = new Lid(30.5, "металлическая","золотистая",true);

        //Вывод массива до сортировки:
        System.out.println ("Массив крышек до сортировки");
        putLidArr(arr);

        //Сортировка:
        bubbleSort(arr);

        //Вывод массива после сортировки:
        System.out.println ("Массив крышек после сортировки");
        putLidArr(arr);
    }
} // LidDemo
```

Результаты работы программы приведены на рисунке 2.2. Обратите внимание на то, что равные объекты (крышки) имеют равные хэш-коды.

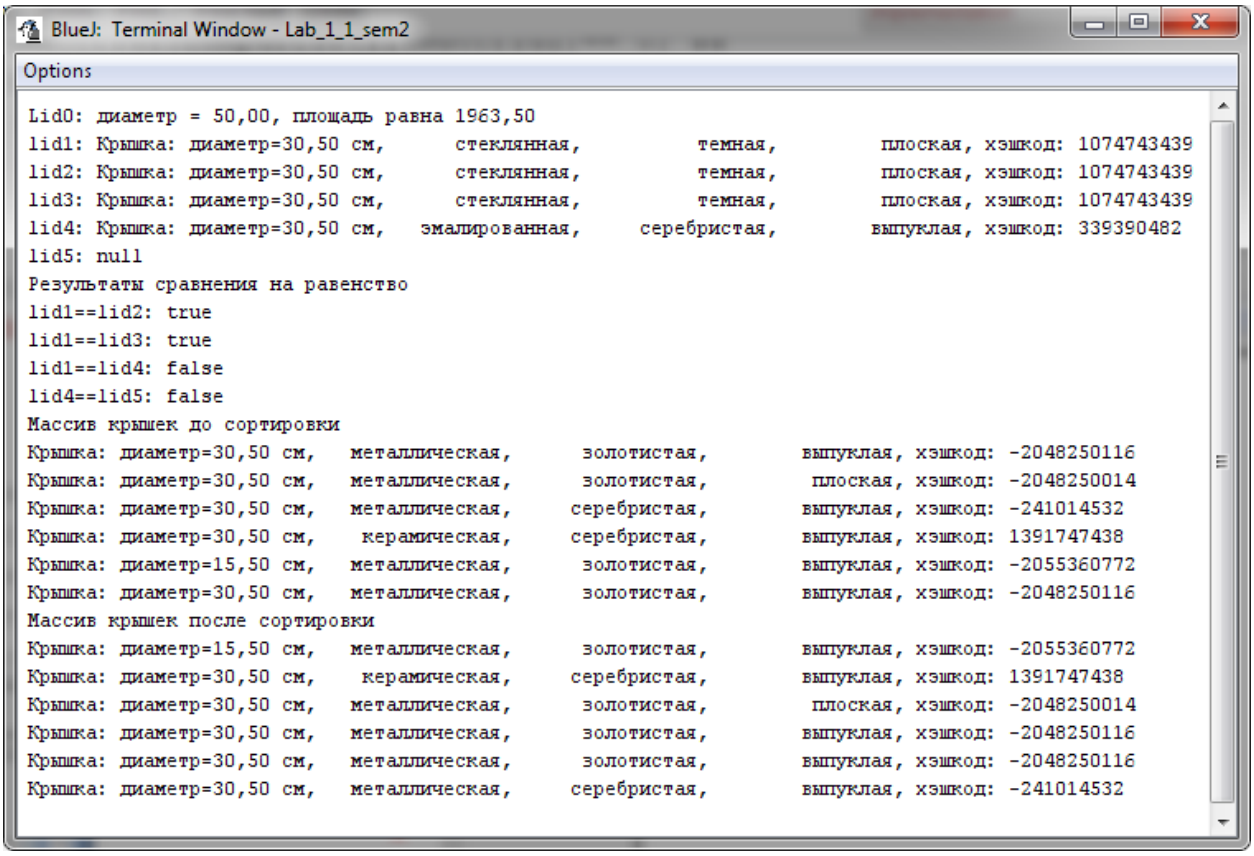

Рисунок 2.2 – Результаты работы программы проекта Lab_1_1_sem2

2.8.2. Проект2 (Lab_1_2_sem2)

Структура проекта приведена на рисунке 2.3.

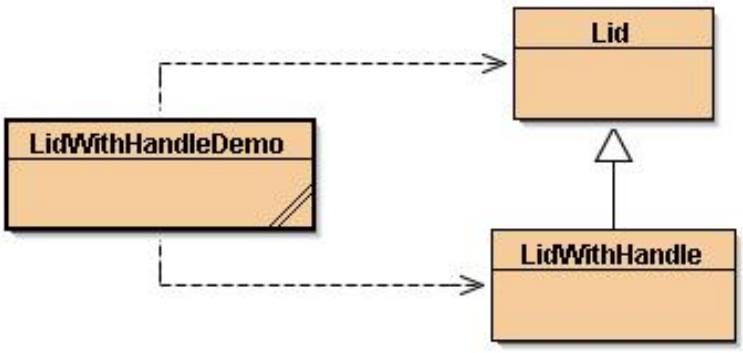

Рисунок 2.3 – Структура проекта Lab_1_2_sem2

Текст программы.

```
public class Lid { //Сущность – Крышка
… //скопировать код класса из предыдущего проекта
} //Lid

public class LidWithHandle extends Lid{ // Сущность - крышка с ручкой,
                                        // наследник класса Крышка
  private final static String LidWithHandle_FORMAT_STRING  = "%s, ручка: %15s";
  private String handle; //добавленное поле - вид ручки
  //конструкторы
  //конструктор бек параметров (используется по умолчанию)
  public LidWithHandle() {super (); handle = "";}
  //конструктор с параметрами, инициализирующими поля объекта
  public LidWithHandle (double d, String material, String color,
                  boolean convex, String handle){
      super (d, material, color, convex); this.handle = handle;
   }
   //конструктор с параметром-объектом (конструирует клон объекта)
   public LidWithHandle (LidWithHandle ob) {
     super (ob); // в конструктор суперкласса передается объект подкласса,
     //но суперкласс распознает в этом объекте
     //только свои собственные члены.
     handle = ob.handle;
   }
   //Геттер и сеттер для добавленного поля
   public String getHandle () {return handle;}
   public void setHandle (String handle) {this.handle=handle;}
   //Переопределяется метод toString предка
   public String toString () { //возвращает строку описания объекта
      return String.format (LidWithHandle_FORMAT_STRING, super.toString(),
                            handle);
   }
   //Переопределяется метод equals предка
   public boolean equals (Object ob) { //сравнение на равенство
     if (!super.equals(ob)) return false; // запускается метод
                          //equals, определенный в суперклассе
     LidWithHandle lwh = (LidWithHandle) ob;
     return handle.equals(lwh.handle);
  }

  //Переопределяется метод hashCode предка
  public int hashCode () {
          return super.hashCode ()
                  + 19 * handle.hashCode();

  }
  //Переопределяется метод compareTo() предка
  public int compareTo (Lid lid) {
     int comp = super.compareTo(lid);
```

```
      if (comp < 0) return -1;
      LidWithHandle  lwh = (LidWithHandle) lid;
      if ((comp == 0) && (handle.compareTo(lwh.handle) < 0)) return -1;
      if ((comp == 0)&& (handle.compareTo(lwh.handle) == 0)) return 0;
      return 1;
    }
} // LidWithHandle


public class LidWithHandleDemo { // Демонстрирует работу с крышками с ручкой
  public static void bubbleSort (Lid[ ] arr) {
  //сортировка массива объектов класса Lid
  // (или его подкласса LidWithHandle) методом пузырька
  //в качестве формального параметра метода определена
  //ссылка суперкласса
        boolean flag;
        for (int m = arr.length-1; m > 0; m--){
           flag = true;
           for (int j = 0; j < m; j++)
              if (arr[j].compareTo(arr[j+1])>0) {
                 Lid b = arr[j];
                 arr[j] = arr[j+1];
                 arr[j+1] = b;
                 flag = false;
               }
           if (flag) break;
        }
  } // пузырек

   public static void putLidArr (Lid [ ] arr) {
      //вывод массива объектов класса Lid (или его подкласса LidWithHandle)
      for (int i=0; i<arr.length; i++)
        System.out.printf ("%s, хэшкод: %d\n", arr[i], arr[i].hashCode());
   }

  public static void main (String args[ ]){
     /* 1.Создаем крышку lwh0 с помощью конструктора без параметров.
     демонстрируем применение сеттеров суперкласса и подкласса для
     установки значений полей.
     Демонстрируем работу геттера getD() и метода area(),
     определенных в суперклассе и
     геттера getHandle (), определенного в подклассе.  */
     LidWithHandle lwh0 = new LidWithHandle();
     lwh0.setD(50); lwh0.setMaterial("стеклянная");
     lwh0.setColor("прозрачная"); lwh0.setConvex(true);
     lwh0.setConvex(true); lwh0.setHandle("литая");
     System.out.printf("Lwd0: диаметр = %.2f, площадь равна %.2f, ручка %s\n",
              lwh0.getD(), lwh0.area(),lwh0.getHandle ());
     /* 2. Создаем крышки lwh1 - lwh4 и пустую ссылку lwh5.
     Демонстрируем работу методов toString(), equals() и hashCode() */
     LidWithHandle lwh1 = new
                  LidWithHandle(30.5,"стеклянная","темная",false,"привинченная");
```

```
LidWithHandle lwh2 = lwh1; //создание псевдонима
LidWithHandle lwh3 = new LidWithHandle(lwh1); //создание клона
//Создание крышки с другими значениями полей
LidWithHandle lwh4 = new
        LidWithHandle(30.5,"эмалированная","серебристая",true,"клепаная");
//Создание null-ссылки
LidWithHandle lwh5 = null;

//Вывод информации о созданных крышках
System.out.printf ("lwh1: %s, хэшкод: %d\n", lwh1, lwh1.hashCode());
System.out.printf ("lwh2: %s, хэшкод: %d\n", lwh2, lwh2.hashCode());
System.out.printf ("lwh3: %s, хэшкод: %d\n", lwh3, lwh3.hashCode());
System.out.printf ("lwh4: %s, хэшкод: %d\n", lwh4, lwh4.hashCode());
System.out.printf ("lwh5: %s\n", lwh5);
//Вывод результатов сравнения на равенство
System.out.println ("Результаты сравнения на равенство");
System.out.printf ("lwh1==lwh2: %s\n", lwh1.equals(lwh2)); //true
System.out.printf ("lwh1==lwh3: %s\n", lwh1.equals(lwh3)); //true
System.out.printf ("lwh1==lwh4: %s\n", lwh1.equals(lwh4)); //false
System.out.printf ("lwh4==lwh5: %s\n", lwh4.equals(lwh5)); //false
//3.Демонстрируем обращение к объекту подкласса с помощью
//ссылочной переменной суперкласса
Lid lid = lwh1;
System.out.println ("Выполнен оператор Lid lid = lwh1;");
System.out.printf ("lid==lwh3: %s\n", lid.equals(lwh3)); //true
//4. Демонстрируем создание, сортировку и вывод массива
// дочерних крышек (с ручкой)
//Создание массива дочерних крышек
LidWithHandle [ ] arrLwh = new LidWithHandle [7];
arrLwh[0] = new LidWithHandle
                (30.5, "металлическая","золотистая",true,"привинченная");
//отличается от arr[0] видом крышки
arrLwh[1] = new LidWithHandle
                 (30.5, "металлическая","золотистая",true,"клепаная");
//отличается от arr[0] выпуклостью:
arrLwh[2] = new LidWithHandle
                 (30.5, "металлическая","золотистая",false,"привинченная");
//отличается от arr[0] цветом:
arrLwh[3] = new LidWithHandle
                (30.5, "металлическая","серебристая",true, "привинченная");
//отличается от arr[0] материалом:
arrLwh[4] = new LidWithHandle
                 (30.5, "керамическая","серебристая",true, "привинченная");
//отличается от arr[0]  диаметром:
arrLwh[5] = new LidWithHandle
                 (15.5, "металлическая","золотистая",true, "привинченная");
//не отличается от arr[0]
arrLwh[6] = new LidWithHandle
                 (30.5, "металлическая","золотистая",true,"привинченная");
//Вывод массива до сортировки:
System.out.println ("Массив крышек до сортировки");
putLidArr(arrLwh);
```

```
        //Сортировка:
        bubbleSort(arrLwh);
        //Вывод массива после сортировки:
        System.out.println ("Массив крышек после сортировки");
        putLidArr(arrLwh);
        //5. Покажем, как те же методы  bubbleSort() и putLidArr() работают
        //с родительскими крышками (класса Lid)
        System.out.print ("\nМассив родительских крышек - без ручки - ");
        System.out.println ("сортируется и выводится теми же методами:");
        //Для этого скопируем сюда часть кода из метода main()
        //предыдущего проекта:
        Lid [ ] arr = new Lid [6];
        arr[0] = new Lid(30.5, "металлическая","золотистая",true);
        //отличается от arr[0] выпуклостью:
        arr[1] = new Lid(30.5, "металлическая","золотистая",false);
        //отличается от arr[0] цветом:
        arr[2] = new Lid(30.5, "металлическая","серебристая",true);
        //отличается от arr[0] материалом:
        arr[3] = new Lid(30.5, "керамическая","серебристая",true);
        //отличается от arr[0]  диаметром:
        arr[4] = new Lid(15.5, "металлическая","золотистая",true);
        //не отличается от arr[0]
        arr[5] = new Lid(30.5, "металлическая","золотистая",true);
        //Вывод массива до сортировки:
        System.out.println ("Массив крышек до сортировки");
        putLidArr(arr);
        //Сортировка:
        bubbleSort(arr);
        //Вывод массива после сортировки:
        System.out.println ("Массив крышек после сортировки");
        putLidArr(arr);
    }
} // LidWithHandleDemo
```

Результаты работы программы приведены на рисунке 2.4. Обратите внимание на то, что равные объекты (крышки с ручкой) имеют равные хэш-коды.

Следует отметить, что в качестве формального параметра метода **bubbleSort()** определена ссылка на объект суперкласса:

public static void bubbleSort (Lid [ ] arr){…}.

Это дало возможность применить метод как для сортировки массива объектов класса Lid (суперкласса), так и для сортировки массива объектов класса LidWithHandle (подкласса). Конкретная версия метода **compareTo()**, используемого для сравнения объектов в методе **bubbleSort(),** выбирается в процессе выполнения программы в зависимости от класса объекта, на который указывает фактический параметр метода **bubbleSort().** Механизм выявления и вызова нужной версии метода в процессе выполнения программы называется динамической диспетчеризацией методов (или поздним связыванием методов). В методе **putLidArr()** в качестве формального параметра также определена ссылка на объект суперкласса (с аналогичной целью). Здесь в процессе выполнения программы будут выбираться нужные (соответствующие

конкретному объекту, ссылка на который передана) версии методов **toString()** и **hashCode()**.

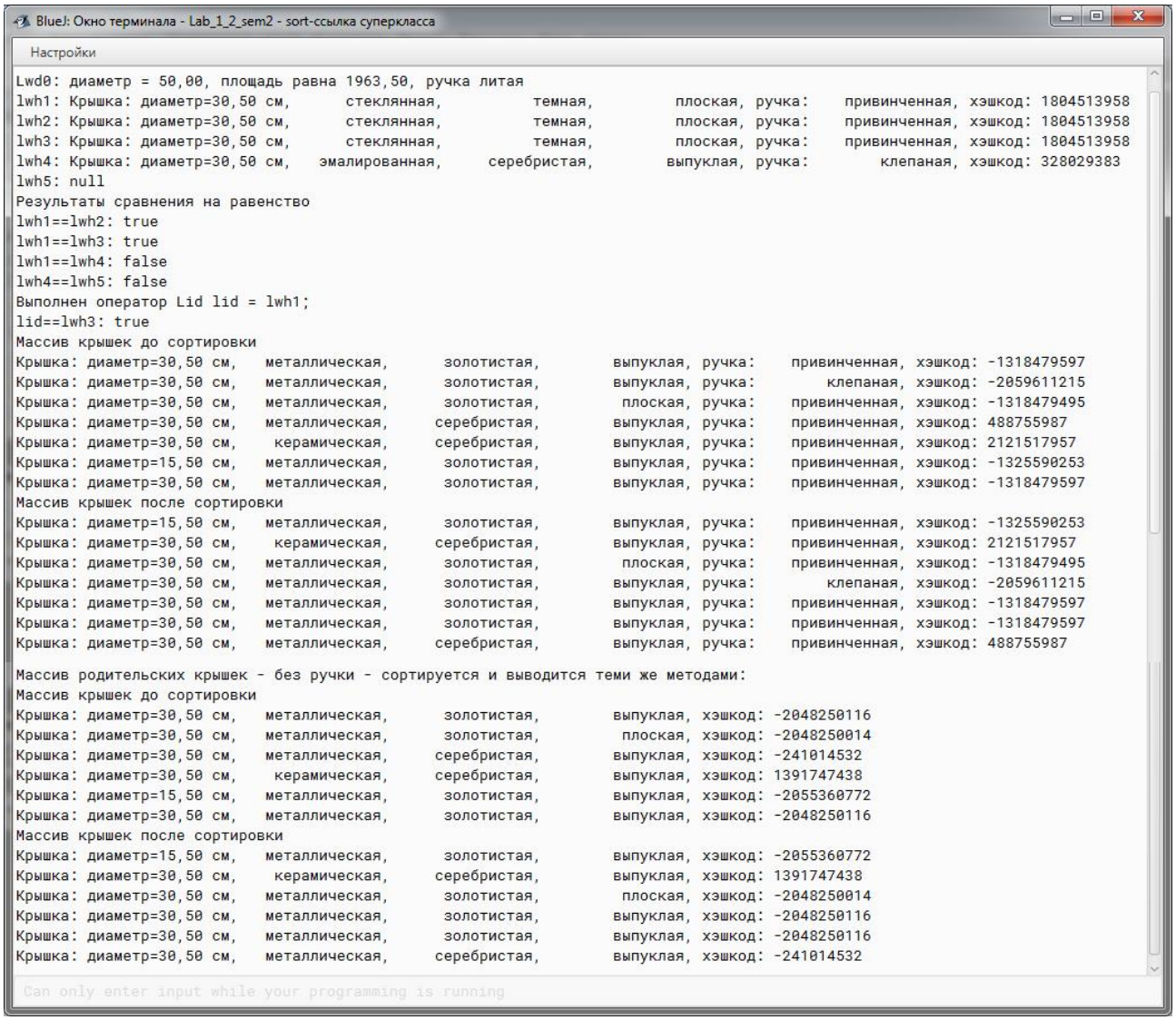

Рисунок 2.4 – Результаты работы программы проекта Lab_1_2_sem2

2.8.3. Проект3 (Lab_1_3_sem2)

Структура проекта приведена на рисунке 2.5.

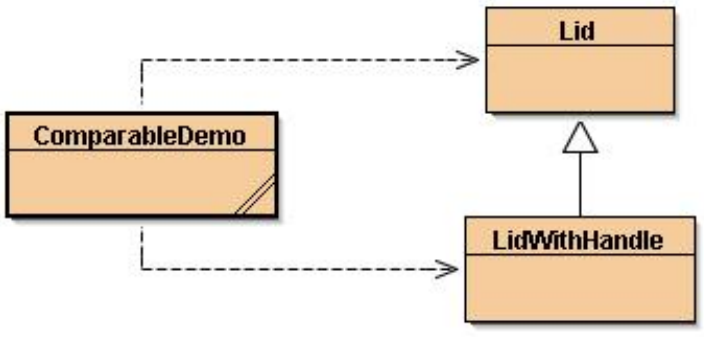

Рисунок 2.5 – Структура проекта Lab_1_3_sem2

Текст программы.

```
public class Lid implements Comparable <Lid> { //Сущность - Крышка
   /* Класс включает (реализует) интерфейс Comparable <Type>,
      значит он должен содержать реализацию метода compareTo(),
      заголовок которого определен в библиотечном
      классе-интерфейсе Comparable:  int  compareTo(Type ob);
    */

   … // текст класса – тот же

   //Сравнение объектов на больше/меньше
   //Теперь это – реализация метода compareTo()
   //интерфейса Comparable <Lid>:
   public int compareTo (Lid li){
      … // текст метода compareTo() – тот же
   }

   … // текст класса – тот же
} // Lid

public class LidWithHandle extends Lid { //Сущность - Крышка с ручкой
  /*Класс LidWithHandle является подклассом класса Lid, реализующего
    интерфейс Comparable <Lid>. Значит, класс LidWithHandle наследует
    указанный интерфейс и должен переопределять (учитывая
   добавленные поля) метод интерфейса compareTo(), заданный в
   суперклассе Lid (имеет место так называемое
   смешанное наследование)*/

   … // текст класса – тот же

   //Смешанное наследование:
   //подкласс наследует интерфейс Comparable от суперкласса,
   //реализует его метод compareTo(), переопределяя одноименный
   //метод суперкласса (по правилам наследования).
  public int compareTo (Lid lid){

      … // текст метода compareTo() – тот же
   }
} // LidWithHandle

public class ComparableDemo { /*Демонстрирует унификацию
(единообразие)  действий с  объектами различных
классов (Lid и LidWithHandle),  реализующих интерфейс  Comparable
 */
  public static void bubbleSort (Comparable[ ] arr) {
 //Сортировка массива объектов Comparable методом пузырька.
 //Метод подходит для сортировки массива, состоящего из любых
 //объектов, включающих интерфейс Comparable, т.к. в нем
 // используются интерфейсные ссылки
       boolean flag;
       for (int m = arr.length-1; m > 0; m--){
```

```
            flag = true;
            for (int j = 0; j < m; j++)
               if (arr[j].compareTo(arr[j+1]) > 0) {
                  Comparable b = arr[j];
                  arr[j] = arr[j+1];
                  arr[j+1] = b;
                  flag = false;
                }
            if (flag) break;
         }
   } // пузырек

   public static void putArr (Comparable [ ] arr) {
       //вывод массива объектов класса Comparable
       for (int i=0; i<arr.length; i++)
          System.out.printf ("%s, хэшкод: %d\n", arr[i], arr[i].hashCode());
   }

   public static void main (String args[ ]){
      //Создание массива крышек типа Lid
      Lid [ ] arr = new Lid [6];
      arr[0] = new Lid(30.5, "металлическая","золотистая",true);
      //отличается от arr[0] выпуклостью:
      arr[1] = new Lid(30.5, "металлическая","золотистая",false);

      //отличается от arr[0] цветом:
      arr[2] = new Lid(30.5, "металлическая","серебристая",true);
      //отличается от arr[0] материалом:
      arr[3] = new Lid(30.5, "керамическая","серебристая",true);
      //отличается от arr[0]  диаметром:
      arr[4] = new Lid(15.5, "металлическая","золотистая",true);
      //не отличается от arr[0]:
      arr[5] = new Lid(30.5, "металлическая","золотистая",true);
      //Вывод массива до сортировки:
      System.out.println ("Массив крышек Lid до сортировки");
      putArr(arr);
      //Сортировка:
      bubbleSort(arr);
      //Вывод массива после сортировки:
      System.out.println ("Массив крышек Lid после сортировки");
      putArr(arr);
      //Создание массива крышек типа LidWithHandle
      LidWithHandle [ ] arr1 = new LidWithHandle [8];
      arr1[0] = new LidWithHandle(30.5, "металлическая", "золотистая", true,
                                          "цельнолитая");
      //отличается от arr1[0] видом крышки:
      arr1[1] = new LidWithHandle(30.5, "металлическая", "золотистая", true,
                                           "привинченная");
      //отличается от arr1[0] видом крышки:
      arr1[2] = new LidWithHandle(30.5, "металлическая", "золотистая", true,
                                          "клепаная");
      //отличается от arr1[0] выпуклостью:
```

```
        arr1[3] = new LidWithHandle(30.5, "металлическая", "золотистая", false,
                                    "привинченная");
        //отличается от arr1[0] цветом:
        arr1[4] = new LidWithHandle(30.5, "металлическая", "серебристая", true,
                                    "привинченная");
        //отличается от arr1[0] материалом:
        arr1[5] = new LidWithHandle(30.5, "керамическая", "серебристая", true,
                                    "привинченная");
        //отличается от arr1[0]  диаметром:
        arr1[6] = new LidWithHandle(15.5, "металлическая", "золотистая", true,
                                    "привинченная");
        //не отличается от arr1[0]:
        arr1[7] = new LidWithHandle(30.5, "металлическая", "золотистая", true,
                                    "цельнолитая");
        //Вывод массива до сортировки:
        System.out.println ("Массив крышек LidWithHandle  до сортировки");
        putArr(arr1);
        //Сортировка:
        bubbleSort(arr1);
        //Вывод массива после сортировки:
        System.out.println ("Массив крышек LidWithHandle после сортировки");
        putArr(arr1);
    }
} // ComparableDemo
```

Результаты работы программы приведены на рисунке 2.6. Обратите внимание на то, что равные объекты (крышки) имеют равные хэш-коды.

В заключение необходимо отметить, что объекты, используемые в программе, могут принадлежать абсолютно разным классам, в том числе, и не связанным отношением наследования. Если указанные классы включают (реализуют) интерфейс Comparable, то для упорядочивания массива объектов каждого класса может применяться один и тот же метод сортировки, в котором в качестве формального параметра определена интерфейсная ссылка.

2.9. Контрольные вопросы

1) В чем состоит принцип наследования и какие преимущества он дает?
2) Как создать подкласс B класса A?
3) Сформулируйте правила доступа из подкласса к членам суперкласса, объявленным как public, private, protected?
4) Какой класс является суперклассом для всех остальных классов?
5) Какие методы этого класса, как правило, переопределяются в классах наследниках?
6) Каким требованиям должен удовлетворять метод equals()?

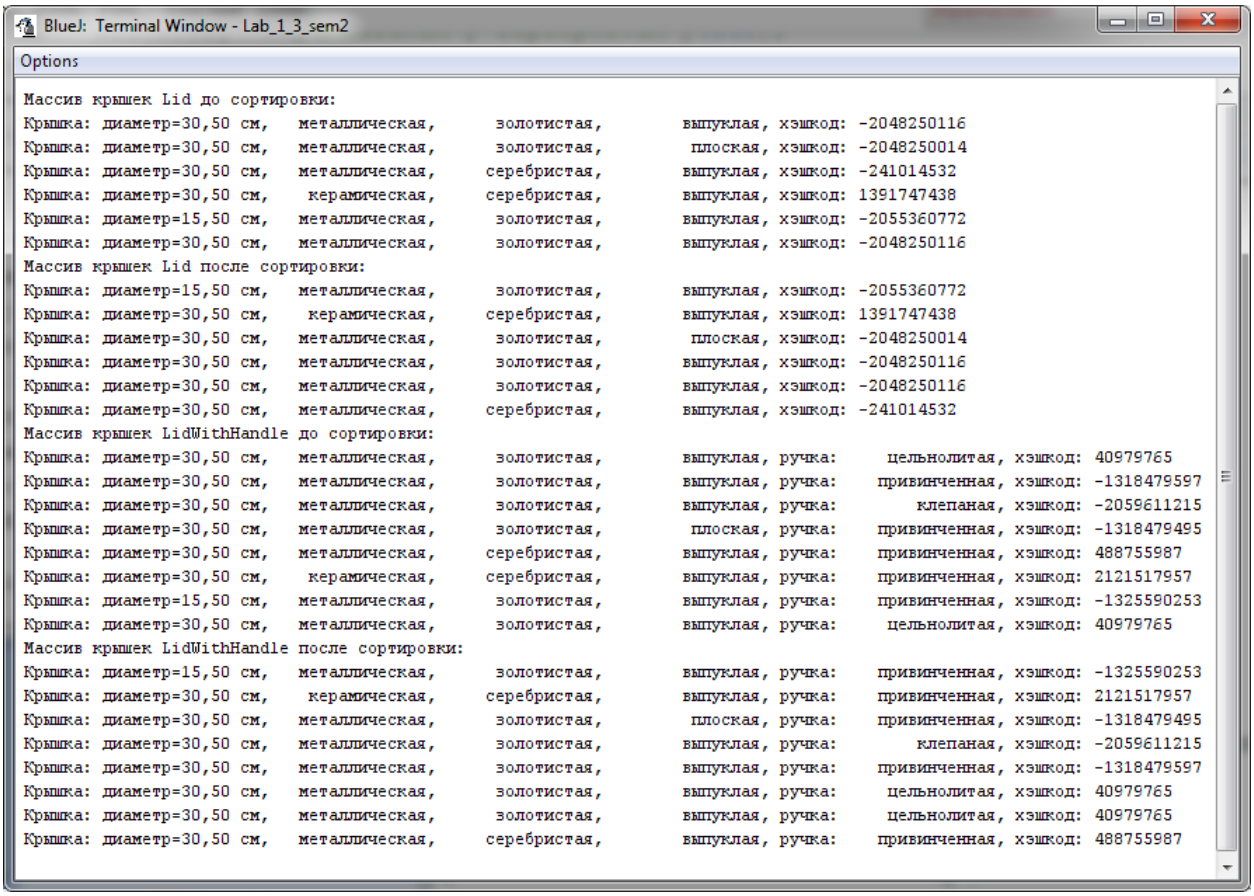

Рисунок 2.6 – Результаты работы программы проекта Lab_1_3_sem2

7) Почему метод equals() класса Класс1 в предыдущей лабораторной работе перегружает одноименный метод класса Object, а в данной работе – переопределяет? Чем перегрузка методов отличается от переопределения?
8) Для чего используется инструкция компилятору @Override?
9) Как в конструкторе подкласса вызвать конструктор суперкласса?
10) Как из подкласса вызвать метод родительского класса, если этот метод переопределен в подклассе?
11) Если в конструкторе подкласса не упомянут конструктор суперкласса, какой конструктор суперкласса используется по умолчанию для инициализации полей суперкласса?
12) Можно ли передать в конструктор суперкласса объект подкласса? Если да, то какие поля этого объекта он распознает?
13) Для чего используется метод compareTo() и какие значения он возвращает?
14) Что возвращает метод hashCode()? Для чего он используется? В каком случае он обязательно должен быть переопределен? Какому правилу должен удовлетворять хэшкод объекта? Какие рекомендации существуют по его вычислению?
15) Что возвращает метод toString()? Где он используется по умолчанию? Какой операции над объектами типа String следует избегать при кодировании этого метода?

16) Можно ли использовать ссылочную переменную суперкласса для работы с объектом подкласса?
17) Назовите преимущества и недостатки подхода, предполагающего обращение к объектам подкласса с помощью ссылочной переменной суперкласса.
17) Что такое динамическая диспетчеризация методов?
18) Как запретить наследование от некоторого класса A (сделать класс финальным)?
19) Что понимают под классом-интерфейсом и под классом, реализующим интерфейс? Для чего используются интерфейсы?
20) Какое преимущество дает использование интерфейсных ссылок (на примере вашего Проекта3)?
21) Какой метод определен в библиотечном интерфейсе Comparable? Для чего используется этот интерфейс (на примере вашего Проекта3)? Как включить интерфейс в класс?
22) Возможно ли множественное наследование в Java?
23) Что понимают под смешанным наследованием?
24) Объясните механизм, позволивший в проекте Проект2 иметь один метод для сортировки и один метод для вывода массива объектов и использовать эти методы для объектов разных классов. Сравните такой вариант с вариантом, подразумевающим наличие перегруженных методов сортировки и вывода для каждого класса объектов, используемых в программе (укажите преимущества и недостатки).
25) Объясните механизм, позволивший в проекте Проект3 иметь один метод для сортировки и один метод для вывода массива объектов и использовать эти методы для объектов разных классов. Сравните такой вариант с вариантом, подразумевающим наличие перегруженных методов сортировки и вывода для каждого класса объектов, используемых в программе. (укажите преимущества и недостатки).
26) И в Проекте2 и в Проекте3 применены подходы, позволяющие иметь один метод для сортировки и один метод для вывода массива объектов и использовать эти методы для объектов разных классов. Почему подход, примененный в Проекте3 является более универсальным, чем подход, примененный в Проекте2?

3. ЛАБОРАТОРНАЯ РАБОТА № 4. МАНИПУЛИРОВАНИЕ ОБЪЕКТАМИ РАЗНЫХ КЛАССОВ В JAVA-ПРОГРАММЕ. ПРОГРАММИРОВАНИЕ ВЗАИМОДЕЙСТВИЯ ПРОСТЫХ ОБЪЕКТОВ

3.1. Цель работы

Целью данной работы является закрепление навыков разработки классов-шаблонов для создания объектов в соответствии с принципом инкапсуляции данных и методов (подпрограмм) для работы с этими данными, а также создания и использования простых объектов в программе, организующей взаимодействие объектов разных классов.

3.2. Постановка задачи

Разработать программу, реализующую взаимодействие двух объектов, заданных вариантом задания. Для создания объектов разработать соответствующие классы. Методы объектов отладить, используя возможности панели объектов BlueJ.

3.3. Внеаудиторная подготовка

Для подготовки к лабораторной работе следует изучить краткие теоретические сведения, приведенные в пункте 3.4 методических указаний. Рекомендуется также ознакомиться с [1] (с.133-153).

3.4. Краткие теоретические сведения

3.4.1. Классы и объекты

ООП строится на четырех основных принципах: **абстракция**, **инкапсуляция, наследование и полиморфизм** [2]**.** В данной лабораторной работе закрепляются навыки использования механизмов инкапсуляции, поддерживаемых языком Java.

Инкапсуляция – это механизм, который связывает код вместе с обрабатываемыми им данными и сохраняет их в безопасности как от внешнего влияния, так и от ошибочного использования.

Основой инкапсуляции в Java является класс. Любая концепция, которую вы хотите реализовать в Java-программе, должна быть инкапсулирована в класс.

Класс – это логическая конструкция, которая определяет структуру и поведение (данные и код) объекта.

Объект – структура, объединяющая некоторое количество данных и способы управления этими данными.

Состояние объекта − совокупность значений элементов данных объекта в данный момент времени.

Методы объекта − код, реализующий способы управления данными объекта (характеризуют поведение объекта) [3, 4].

Можно сказать, что класс определяет новый тип данных. После определения новый тип можно использовать для создания объектов. Таким образом, класс – это шаблон (чертеж), по которому создается объект, а объект – экземпляр класса. Иногда говорят, что класс – это фабрика по производству объектов.

Пусть класс определяет свойства и поведение некоторой сущности, например, сущности «студент». «Студент» имеет свойства: номер зачетки (ключевое, т.е. уникальное для каждого студента), фамилия, имя, отчество, год рождения, группа, увлечение. Эти свойства являются переменными (полями) класса и имеют соответствующие типы. Поведение студента определяется методами класса. Например, в классе может быть определен метод «изменить увлечение».

На основе класса могут быть созданы отдельные экземпляры студентов: Иванов, Петров, и т.д., которые могут менять свои увлечения.

Данные, определяемые в классе называют членами-переменными (member variables), или переменными экземпляра (instance variables), или полями объекта, или свойствами (атрибутами) объекта. Код, оперирующий с этими данными, называют членами-методами (member methods) или просто методами.

В правильно записанных Java-программах методы определяют, как можно использовать члены-переменные.

Доступ к элементам (членам) класса

Переменные-члены класса доступны всем методам-членам класса.

Способ доступа к переменным и методам из других классов определяет спецификатор доступа, который предшествует остальной части объявления элемента (члена) класса.

К переменным и методам класса, имеющим спецификатор public можно получить доступ из любой точки программы, т.е. из других классов и пакетов.

Переменные и методы класса, имеющие спецификатор private доступны только членам класса (т.е. только внутри класса).

Когда никакой спецификатор доступа не используется (по умолчанию) элемент класса считается public (общедоступным) в пределах своего пакета, но к нему нельзя обращаться извне этого пакета (пакеты будут рассмотрены позже).

Спецификатор protected (защищенный) применяется только при использовании наследования (рассмотрим позже). Позволяет элементу быть видимым извне текущего пакета, но только в классах, являющихся прямыми подклассами (потомками) класса, членом которого является этот элемент.

Переменные, определенные в классе называются переменными экземпляра, т.к. каждый экземпляр класса (т.е. каждый объект класса) содержит свою

собственную копию этих переменных. Таким образом, данные одного объекта отделены от данных другого.

Методы определены в классе, но запускаются от имени объекта-экземпляра класса (кроме методов, отмеченных ключевым словом static).

Методы и поля, отмеченные словом static, считаются принадлежностью класса, а не объекта. Такие методы запускаются от имени класса, а имена полей уточняются именем класса, например: double y=Math.sin(x)*Math.PI;

Можно создавать классы, содержащие только переменные. Такой класс будет описывать объекты, аналогичные структурам данных типа «запись» (тип данных «record» в языке Pascal и тип данных struct в языке C/C++).

Можно создавать классы, содержащие только методы (библиотеки методов).

Внимание! В классе, который будет использоваться как шаблон для создания объектов, следует избегать присутствия статических членов. Исключение составляют final-переменные (именованные константы), которые должны разделяться всеми объектами класса (хранятся в статическом контексте класса, а не в объектах).

Класс, который является стартовой точкой программы, должен содержать метод main [5, 6].

Создание объектов. Методы-конструкторы.

Операция new динамически распределяет память для объекта, создаваемого на основе класса, и возвращает адрес соответствующего блока памяти. При создании объекта используется следующая общая форма оператора присваивания: **class_var = new classname();**

Здесь class_var – переменная типа «класс», которая создается;

classname() – один из конструкторов класса, экземпляр которого создается (имя конструктора совпадает с именем класса).

Конструктор – это специальный метод класса, который определяет, какие действия должны выполняться при создании объекта данного класса.

Он должен иметь спецификатор public, т.к. конструктор запускается из другого класса.

Например, в классе TestBox создается объект класса Box.

```
Box box1=new Box() ; // в классе TestBox
```

Имя метода-конструктора совпадает с именем класса.

Конструктор может иметь (или не иметь) параметры.

В классе (для примера – в классе Box) может быть задано несколько конструкторов, отличающихся набором параметров:

```
public class Box{
   private double d, w, h; //поля скрыты в классе
  // методы-конструкторы:
   public Box(){d=0.0; w=0.0; h=0.0;} // без параметров
   public Box(double d1, double w1, double h1 ){
                  d=d1; w=w1; h=h1;} / /с параметрами
...
```

Рекомендуется обязательно задавать в классе конструктор без параметров, даже если, на первый взгляд, кажется, что он не нужен. Дело

в том, что этот конструктор используется как конструктор по умолчанию при реализации принципа наследования.

Сокрытие переменной экземпляра.

Если локальная переменная метода (а формальные параметры являются локальными переменными) имеет такое же имя, как переменная экземпляра, локальная переменная скрывает переменную экземпляра. Т.е. при употреблении данного имени обращение будет происходить только к локальной переменной. В приведенном выше коде эта проблема решается путем использования несовпадающих имен для переменных экземпляра и параметров конструктора.

Второй способ решения данной проблемы – использование ключевого слова this.

Ключевое слово **this** используется, когда у метода возникает необходимость обращаться к объекту, который его вызвал.

this – это ссылка на текущий объект, т.е. объект, метод которого был вызван.

Например, если в вызывающем классе от имени объекта, на который ссылается переменная box1 был запущен метод equals():

box1.equals(box2); , то в ссылочной переменной this будет находиться тот же адрес, что и в ссылочной переменной box1.

Слово this можно использовать везде, где разрешается ссылка на объект текущего класса. Код класса Box может быть изменен следующим образом:

```
public class Box{
   //поля
   private double d, w, h; //скрыты в классе
   //методы-конструкторы
   …
   public Box(double d, double w, double h){ // с параметрами
      this.d=d; this.w=w; this.h=h;}
…
```

Правильный, с точки зрения инкапсуляции, метод создания классов-шаблонов.

Кроме методов-конструкторов в классе могут быть определены так называемые методы-геттеры и методы-сеттеры, через которые осуществляется доступ к private-переменным экземпляра, другие полезные методы, определяющие поведение объекта, а также методы, используемые другими методами класса (созданные с целью улучшения структуры кода).

Используется общее правило: в правильно (с точки зрения инкапсуляции) написанных Java-программах методы определяют, как можно использовать члены-переменные. Такой подход предохраняет код и данные от произвольного доступа из других кодов, определенных вне данного класса. Доступ к коду и данным внутри защитной оболочки (класса) строго контролируется через тщательно спроектированный интерфейс. Так как private-члены класса оказываются доступными другим частям вашей программы только через public-

методы, вы можете быть уверены, что никакие неподходящие действия не выполнятся.

Назначение ссылочных переменных объекта.

Обращение (доступ) к объекту производится с помощью ссылочных переменных.

Пусть в программе выполняется следующая последовательность операторов:

Box b1=new Box(5,3,1);
Box b2=b1;
b1=null;

При выполнении оператора **Box b1=new Box(5,3,1);** осуществляются следующие действия.

1) Создается ссылка на объект типа Box, которой присваивается значение null, т.е. переменная box1 пока не указывает на конкретный объект (эквивалентно **Box b1;**).

2) Операция new распределяет динамически (т.е. во время выполнения программы) память для объекта и возвращает ссылку на нее (адрес блока памяти). Эта ссылка сохраняется в переменной box1 (эквивалентно **b1=new Box(5,3,1);**).

При выполнении оператора **Box b2=b1;** ссылке b2 на объект класса Box будет присвоено значение ссылки b1. Это значит, что ссылочные переменные b1 и b2 будут ссылаться на один и тот же объект, т.е. у объекта будет два псевдонима.

В данном примере копировалось значение ссылки (адреса) объекта. Чтобы получить копию самого объекта, нужно создать новый объект того же класса при помощи операции new, а затем присвоить его полям значения полей первого объекта. При использовании первой версии класса Box (с private-полями) в вызывающий класс (TestBox) придется включить следующие операторы:

```
Box b1=new Box();
b1.setD(b2.getD());
b1.setH(b2.getH());
b1.setW(b2.getW());
```

В классе Box можно определить конструктор, строящий клон объекта, ссылка на который передается в качестве параметра:

```
public class Box{
   //поля
   private double d, w, h; //скрыты в классе
   //методы-конструкторы
   …
   public Box(Box b){ // строит клон объекта b
      d=b.d; w=b.w;.h=w.h;}
…
```

Тогда клон объекта в ввызывающем классе можно создать с помощью этого конструктора:

```
Box b1 = new Box(5,3,1);
Box b2 = new (b1);
```

3.4.2. Работа с панелью объектов BlueJ

Работу с панелью кода BlueJ покажем на примере пректа Lab11_1_Box. Структура проекта Lab11_1_Box изображена на рисунке 3.1.

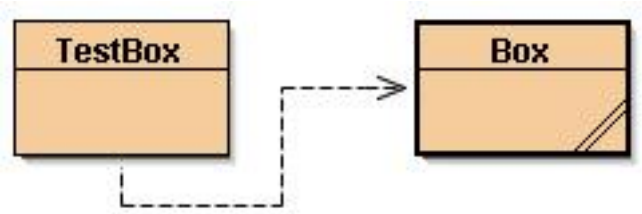

Рисунок 3.1 – Структура проекта Lab11_1_Box

Текст программы.

```
public class Box{ //Сущность – Коробка
   //поля (свойства сущности) – длина, ширина, высота коробки
   private double d, w, h; //скрыты в классе
   //методы-конструкторы
   public Box(){d=0.0; w=0.0; h=0.0;}
   public Box(double d1, double w1, double h1 ){
      d=d1; w=w1; h=h1;}
   //методы-геттеры (доступ к полям)
   public double getD(){return d;}
   public double getW(){return w;}
   public double getH(){return h;}
   //методы-сеттеры (доступ к полям)
   public void setD(double d1){d=d1;}
   public void setW(double w1){w=w1;}
   public void setH(double h1){h=h1;}
   //другие полезные методы
   public double volume(){return d*w*h;} // возвращает объема
   public String toString(){ //возвращает строку описания объекта
     return String.format("Box: d = %.2f, w = %.2f, h = %.2f",d,w,h);
   }
} //Box
//Код вызывающего класса:
public class TestBox{  //проект Lab11_1_Box
  public static void main (String args[ ]){
     //создание объектов
     Box box1=new Box(); //вызов конструктора без параметров
     Box box2=new Box(6.2,4.5,3.9); // вызов конструктора
                                    // с параметрами
     //далее методы, определенные в классе Box, запускаются
     // от имени объектов этого класса box1 и box2
     System.out.printf("Первая коробочка: %s\n", box1.toString());
```

```
        // а здесь метод println использует метод toString объекта,
        // на который указывает ссылка box2,  по умолчанию
        System.out.printf("Вторая коробочка: %s\n", box2);
        box1.setD(5.0);
        box1.setW(5.0);
        box1.setH(5.0);
        System.out.printf(
                    "Первая коробочка после изменения размеров: %s\n",
                     box1.toString());
        double a = box1.volume() + box2.volume();
        System.out.printf("Суммарный объем: %.2f\n", a);

       System.out.printf("Суммарная высота: %.2f\n",
                                  (box1.getH() + box2.getH()));
    }
} //TestBox
```

Щелчок правой клавишей мыши по классу Box открывает контекстное меню (рисунок 3.2). Выбор в меню функции new Box() инициирует создание объекта box1, который является экземпляром класса Box. Он появляется в панели объектов (рисунок 3.3).

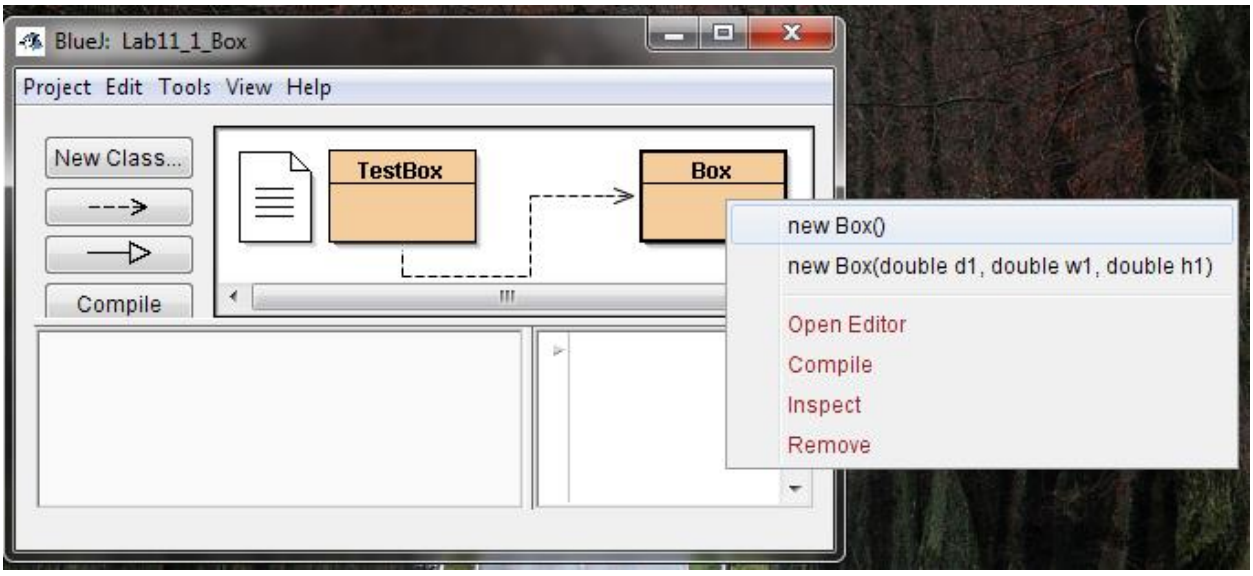

Рисунок 3.2 – Структура проекта и контекстное меню класса Box

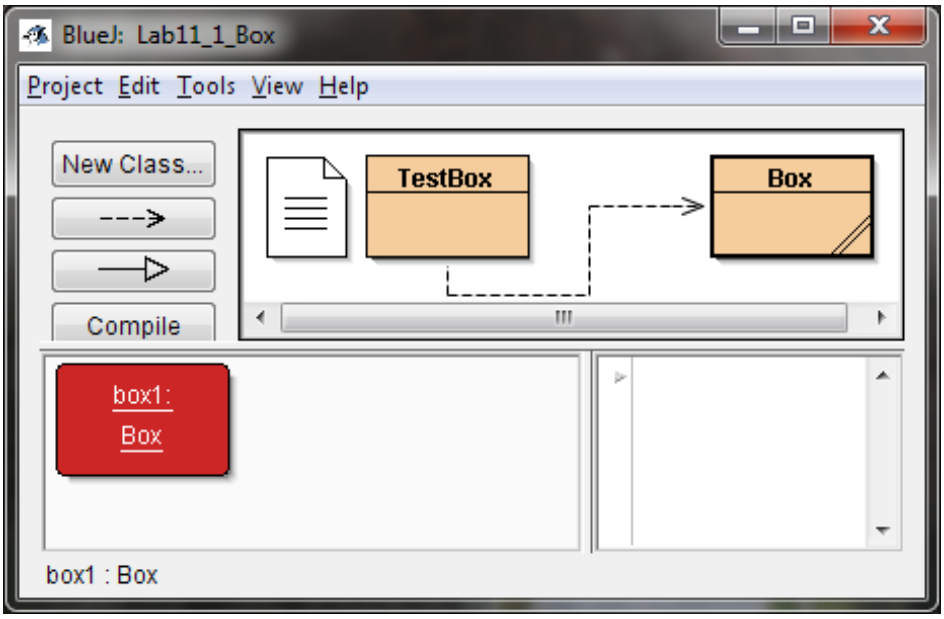

Рисунок 3.3 – Появление объекта box1 в панели объектов

Двойной щелчок левой клавишей мыши на объекте box1 позволяет просмотреть значения полей объекта (переменных экземпляра) – рисунок 3.4.

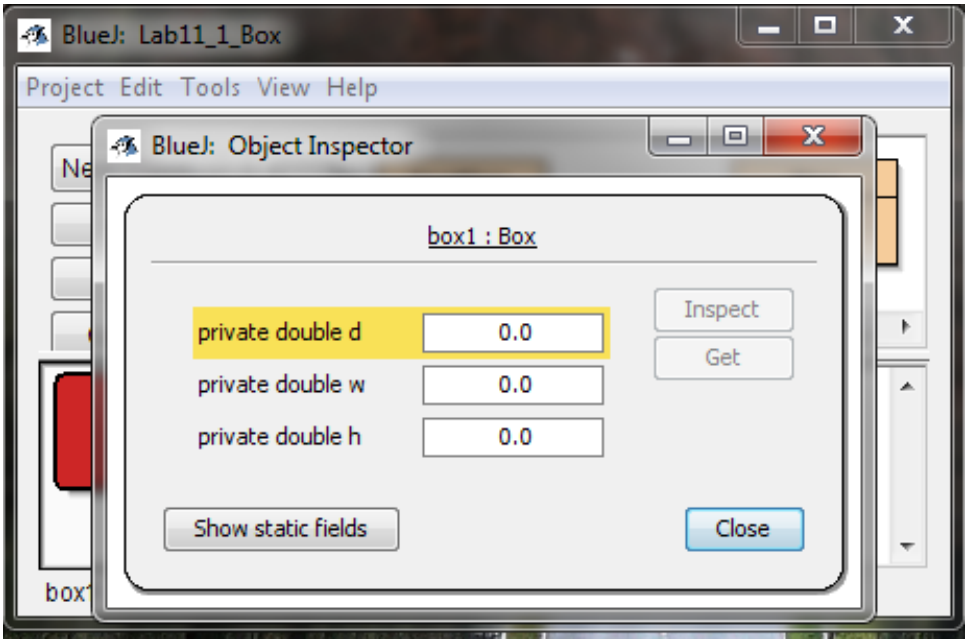

Рисунок 3.4 – Просмотр значений полей объекта box1

Щелчок правой клавишей мыши по объекту Box1 открывает контекстное меню объекта, которое позволяет запустить на выполнение любой из его методов (рисунок 3.5).

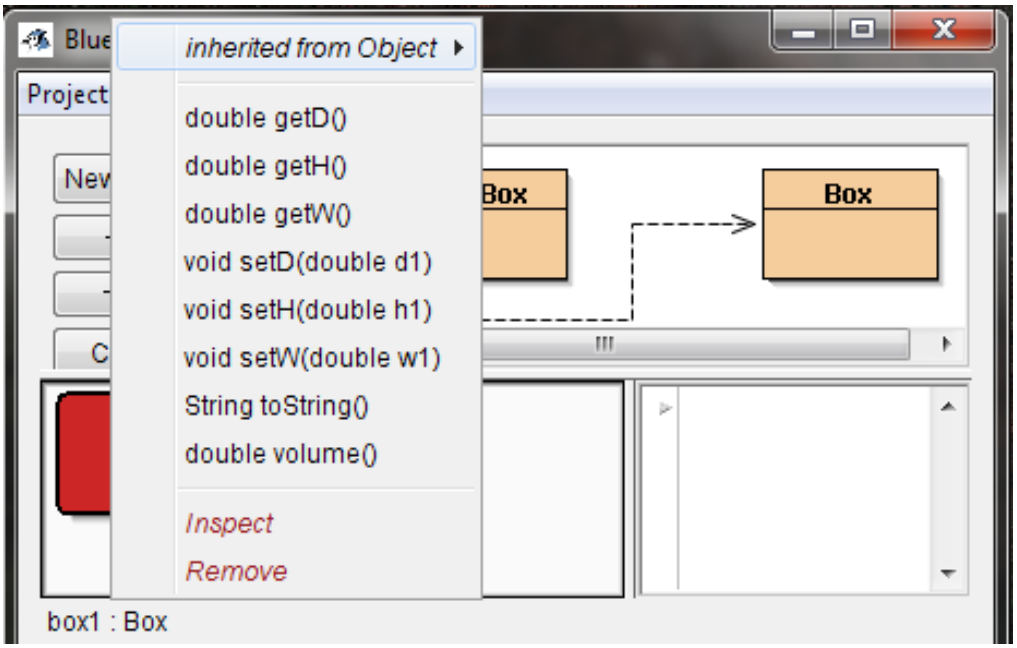

Рисунок 3.5 – Контекстное меню объекта box1

Теперь создадим объект класса Box, используя конструктор с параметрами (рисунки 3.6, 3.7, 3.8, 3.9). В панели объектов появляется экземпляр класса Box с именем Box2.

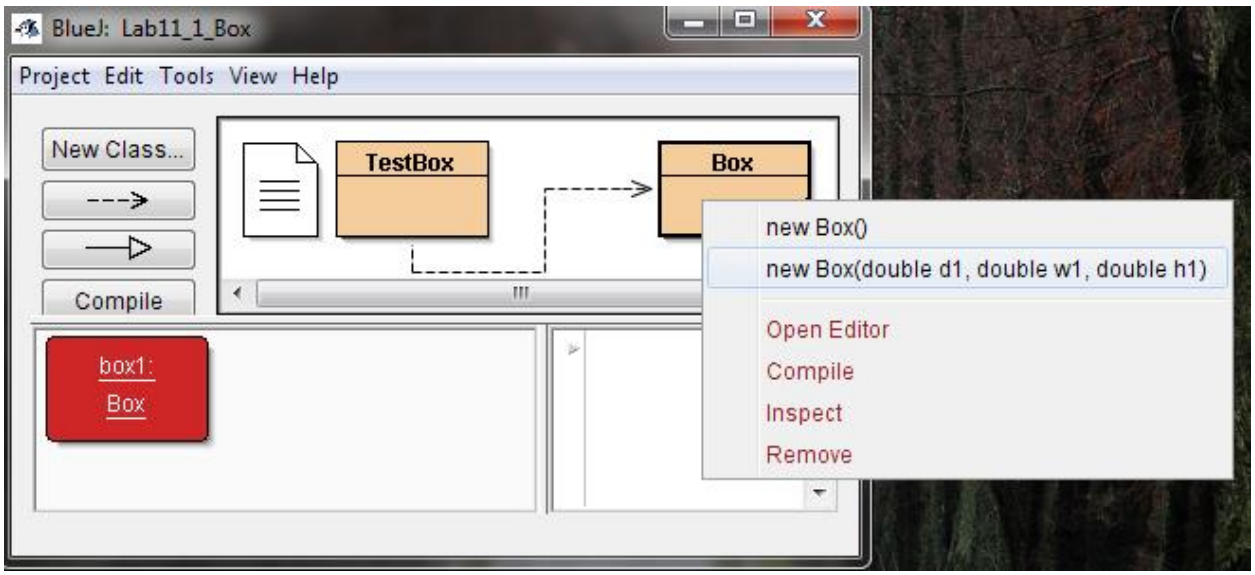

Рисунок 3.6 – Выбор в контекстном меню класса Box функции new, использующей конструктор с параметрами

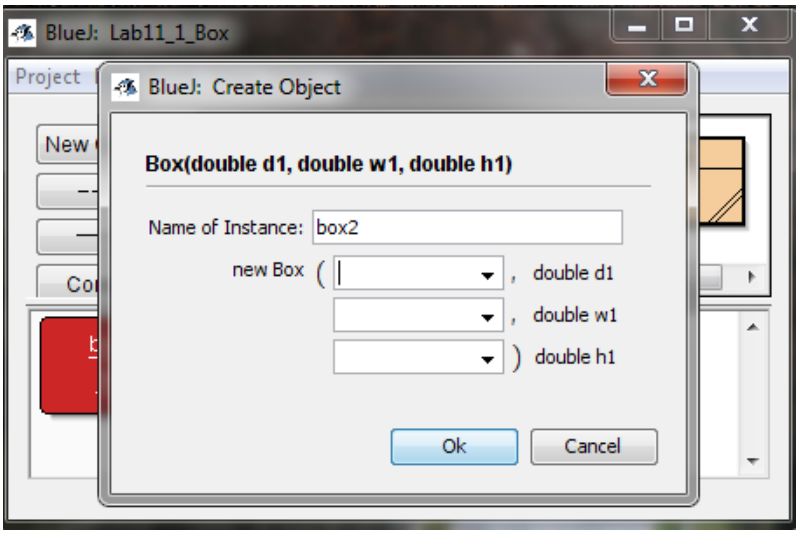

Рисунок 3.7 – Запрос на ввод значений параметров конструктора

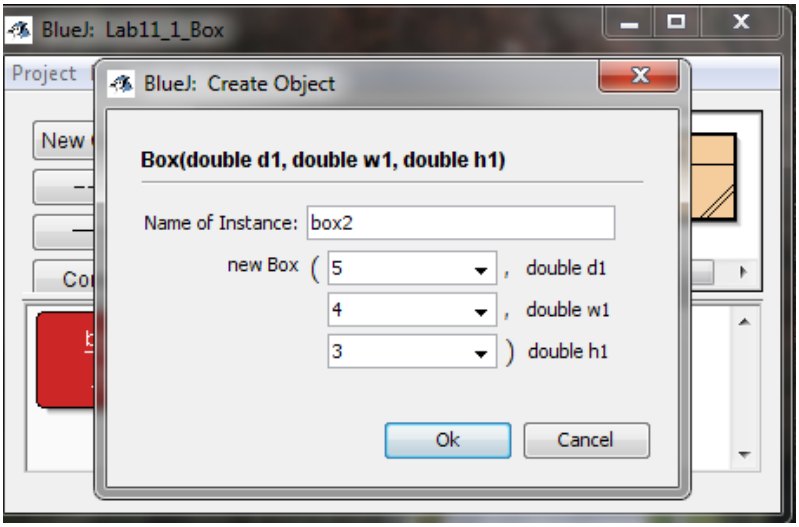

Рисунок 3.8 – Пользователь осуществил ввод

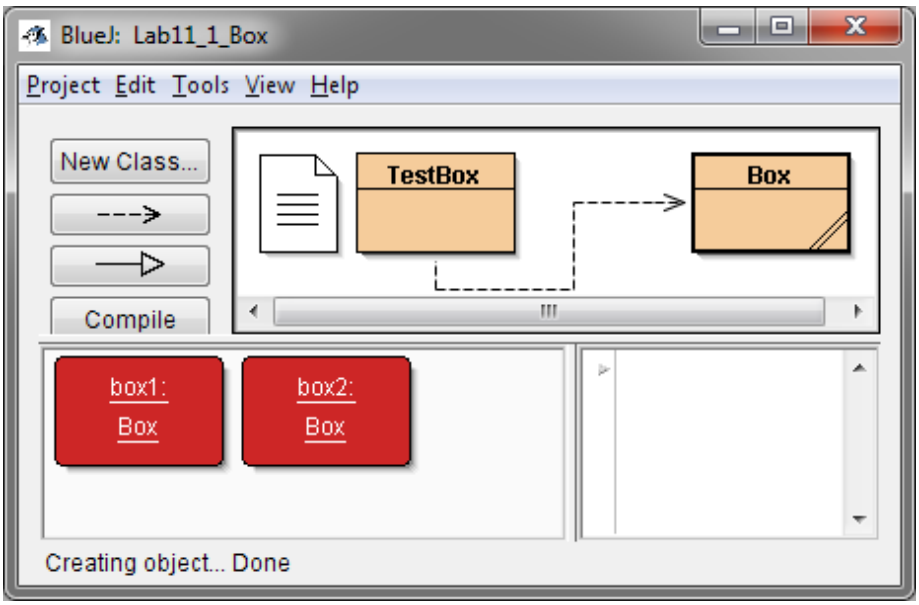

Рисунок 3.9 – Объект Box2 появился в панели объектов

Окно просмотра значений полей объекта box2 изображено на рисунке 3.10.

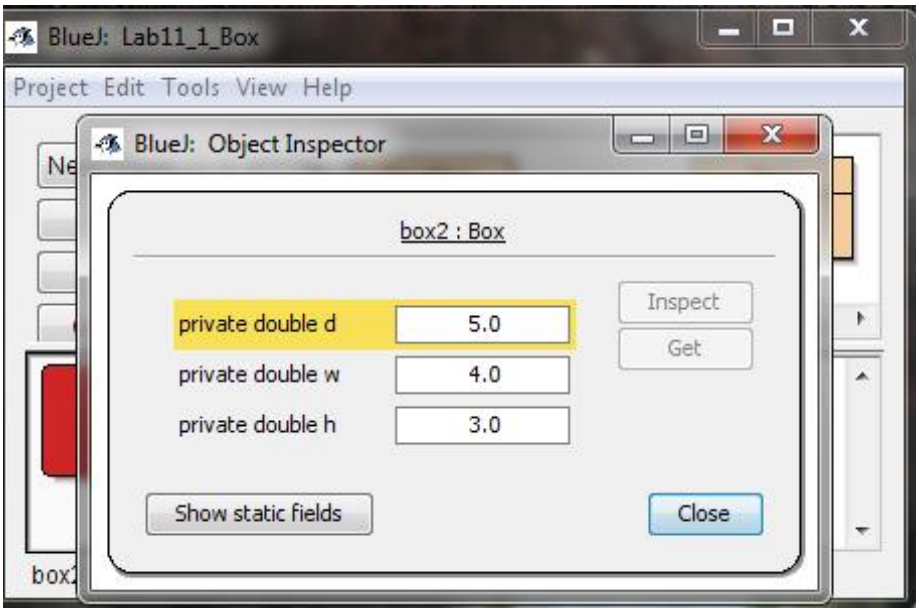

Рисунок 3.10 – Просмотр значений полей объекта box2

Контекстное меню объекта box2 изображено на рисунке 3.11. Рисунки 3.12 и 3.13 иллюстрируют результаты работы методов volume() и toString(), определенных в классе Box, и запущенных от имени объекта box2 (из контекстного меню объекта box2.

Использование панели объектов предоставляет программисту удобную возможность проверить работу методов объекта при различных значениях параметров методов и переменных экземпляра.

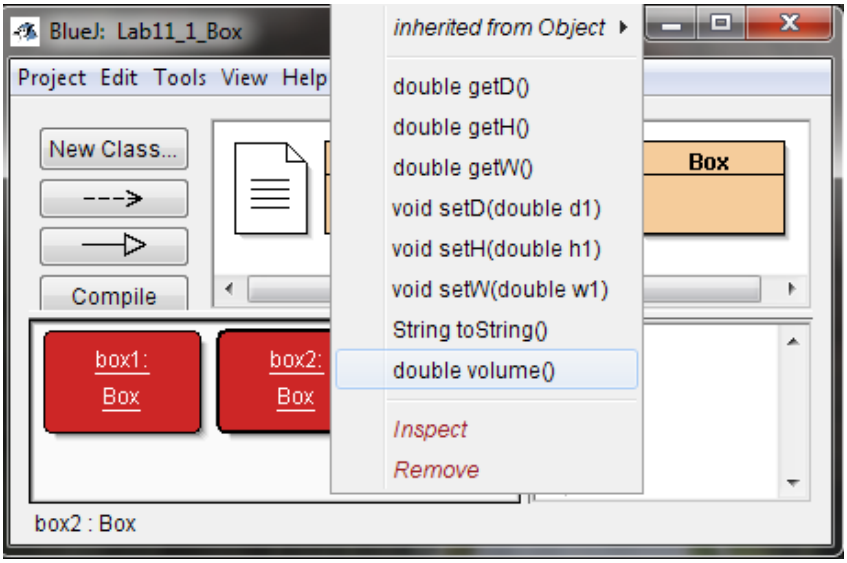

Рисунок 3.11 – Контекстное меню объекта box2

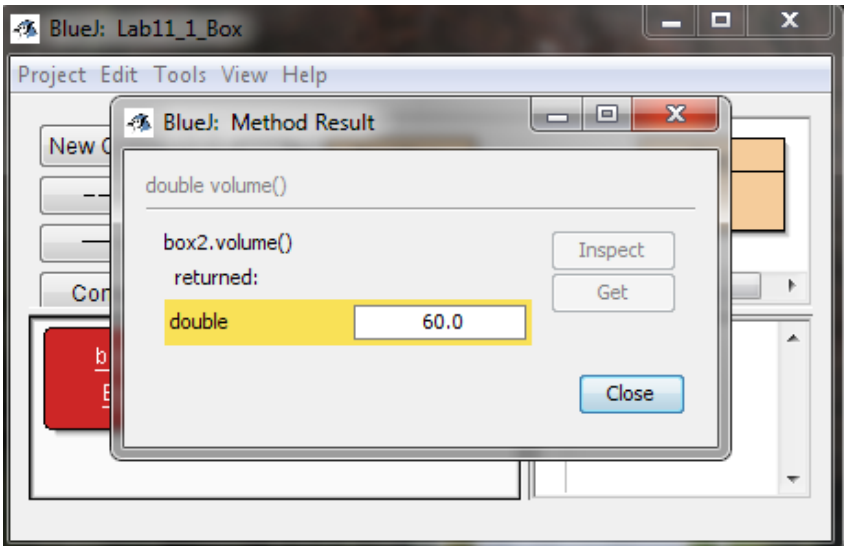

Рисунок 3.12 – Результат работы метода volume() для объекта box2

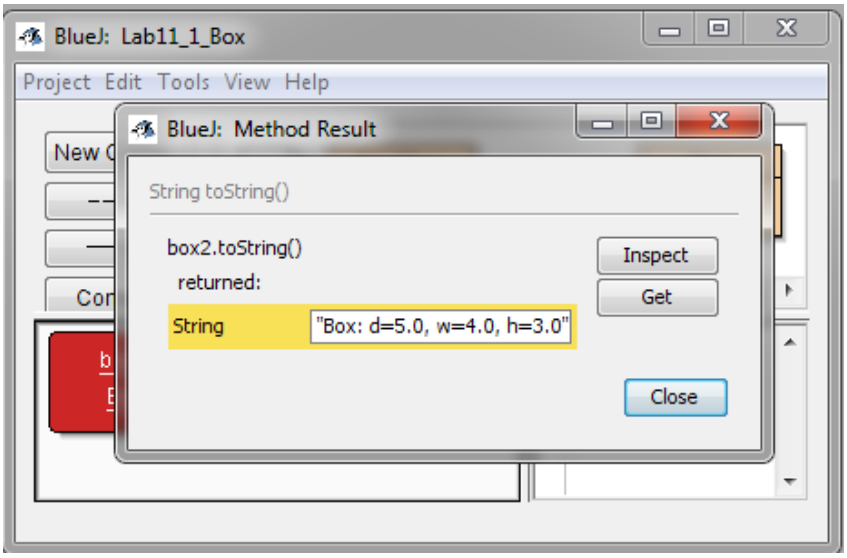

Рисунок 3.13 – Результат работы метода toString() для объекта box2

3.5. Порядок выполнения работы

1) Создайте ***новый*** проект с названием Lab_2_Фамилия_Группа.
2) Скопируйте в созданный проект класс Класс1 (реализует свойства и поведение Сущности1) из проекта Lab_1_3_Фамилия_Группа. Напомним, что данный класс переопределяет методы equals(), hashCode(), toString() суперкласса Object и реализует метод CompareTo() библиотечного интерфейса Comparable.
3) По аналогии с классом Класс1 разработайте Класс2 (реализует свойства и поведение Сущности 2), который тоже переопределяет методы equals(), hashCode(), toString() суперкласса Object и реализует метод CompareTo() библиотечного интерфейса Comparable. Свойства и методы Сущности2 заданы в таблице 3.1.
4) Добавьте в класс Класс1 метод accord(), определяющий соответствие объектов классов Класс1 и Класс2 друг другу.

5) Добавьте в класс Класс2 метод accord(), определяющий соответствие объектов классов Класс2 и Класс1 друг другу.

Внимание! Перед программированием метода accord(), определяющего один из аспектов поведения сущностей 1 и 2 , обязательно составьте блок-схему алгоритма метода и приведите ее в отчете.

6) Разработайте класс ObjDemo, содержащий метод main(). Данный класс должен демонстрировать:
   - создание объектов обоих классов;
   - применение методов equals(), hashCode(), toString() для объектов класса Класс2 (аналогичные действия для объектов класса Класс1 демонстрировались в предыдущей лабораторной работе);
   - создание массива объектов класса Класс1 (Массив1);
   - создание массива объектов класса Класс2 (Массив2);
   - унификацию сортировки элементов массивов Массив1 и Массив2 за счет использования в методе сортировки интерфейсных ссылок (Comparable);
   - взаимодействие объектов, которое выражается в определении соответствия объекта класса Класс1 объекту класса Класс2 и наоборот.

   Действуйте по аналогии с примером, приведенным в разделе 3.8.

7) Добейтесь успешной компиляции программы (устраните все синтаксические ошибки).

8) Создайте (в панели объектов) экземпляры сущностей «Класс1» и «Класс2» с различными значениями полей. Проверьте работу методов, запущенных от имени экземпляров. Особое внимание уделите методу accord(). Этот метод имеет разветвляющуюся структуру и должен быть протестирован по всем ветвям.

9) Запустите метод main() проекта. Проведите комплексную отладку программы, проанализируйте и приведите в отчете результаты выполнения программы. Перед запуском программы в меню окна терминала установите опции «Очищать перед вызовом метода», «Неограниченная буферизация».

Если возникнут трудности с отладкой программы, используйте возможности отладчика BlueJ.

Еще один часто применяемый способ отладки программ (если отладчик в инструментальной системе отсутствует) – вставка в текст программы операторов «отладочной печати», например, System.out.println() для вывода промежуточных значений переменных в окно терминала и обозначения ветвей, по которым реализуется программа. Этот способ отладки неудобен тем, что программисту в ряде случаев придется вместо одного оператора использовать блок операторов (фигурные скобки) и следить за тем, чтобы операторы отладочной печати, а с ними и ставшие ненужными ограничители блоков были удалены.

3.6. Варианты заданий

Номер варианта задания V необходимо вычислить по формуле

int V = (N<=14) ? variant : N%14;

где N – номер студента в списке группы,
variant – номер варианта задания в таблице 2.1.

Структура класса Класс1 для соответствующего варианта задания определена в предыдущей лабораторной работе. Данный класс копируется в проект Lab_2_Фамилия_Группа из проекта Lab_1_3_Фамилия_Группа. После этого в Класс1 добавляется метод accord(), определяющий соответствие объекта класса Класс1 объекту класса Класс2. Сигнатура метода:

public boolean accord (Класс2 obj) .

В таблице 3.1, заданы названия соответствующих сущностей, перечень свойств (полей) Сущности2, условие соответствия сущностей. Свойства, названия которых заканчиваются символом '?' имеют тип boolean. Класс Класс2, определяющий свойства и поведение Сущности 2, программируется по аналогии с классом Класс1, после чего в него добавляется метод accord(), определяющий соответствие объекта класса Класс2 объекту класса Класс1. Сигнатура метода:

public boolean accord (Класс1 obj) .

Таблица 2.1 – Описание сущностей, участвующих в проекте

№	Класс1 (сущ-ность 1)	Класс2 (сущ-ность 2)	Поля (private)	Условие соответствия объектов класса1 и класса 2
1	Книга	Абоне-мент	Номер комнаты, название (научный, учебный, художе-ственный, читальный зал), ответствен-ный методист.	Если количество экземпляров меньше 5, то книга должна храниться в читальном зале, иначе в том абонементе, название которого соответствует типу книги.
2	Телеви-зор	Под-ставка	Длина, ширина, грузо-подъем-ность, материал	Телевизор можно ставить на подставку, если длина и ширина подставки соответственно больше длины и ширины телевизора, грузоподъемность подставки больше или равна весу телевизора.

Продолжение таблицы 2.1

№	Класс1 (сущ-ность 1)	Класс2 (сущ-ность 2)	Поля (private)	Условие соответствия объектов класса1 и класса 2
3	Семья аквари-умных рыбок	Аквари-ум	Объем, форма, материал	Семья рыб может жить в аквариуме, объем которого больше или равен рекомендуемому объему для заданного количества рыбок в семье.
4	Занятие	Аудито-рия	Количество мест, есть доска?, количество компью-теров	Для лекции и практического занятия подходит аудитория с доской, одним компьютером и соответствующим числу студентов количеством мест, для лабораторного занятия подходит аудитория с количеством компьютеров, соответствующим количеству студентов.
5	Партия товара	Магазин	Название магазина, тип магазина (продук-товый, универ-сальный, ткани), мини-мальный размер принима-емой партии товара	Партия товара может быть принята в магазин, если тип товара соответствует типу магазина (в продуктовом магазине могут продаваться только продукты, в магазине тканей только ткани, в универсальном магазине могут продаваться все типы продуктов), размер партии товара не меньше минимально допустимого для магазина.
6	Зал	Конфе-ренция	Название, число участников, использу-ются презента-ции?, предпола-гается электронное голосова-ние?	Зал подходит для проведения конференции, если количество мест больше или равно числу участников конференции, обеспечиваются (при необходимости) требования на использование в докладах презентаций (т.е. наличие мультимедийного оборудования) и проведение электронного голосования.

Продолжение таблицы 2.1

№	Класс1 (сущ-ность 1)	Класс2 (сущ-ность 2)	Поля (private)	Условие соответствия объектов класса1 и класса 2
7	Коман-да	Чемпи-онат	Название, вид спорта, лига, учитыва-ются штрафные очки?, предел по штрафным очкам	Команда может участвовать в чемпионате, если у команды и чемпионата совпадают вид спорта и лига, а также в случае учета чемпионатом штрафных очков, ранее полученных командой, это число не должно превышать установленного для чемпионата предела.
8	Фирма	Проект	Название, отрасль, фонд заработной платы, рейтинг.	Фирма может выполнять проект, если ее область деятельности совпадает с отраслью проекта, рейтинг фирмы больше или равен рейтингу проекта, фонда заработной платы проекта хватает для обеспечения минимальной зарплаты сотрудников фирмы.
9	Семья	Жилье	Тип (дом или квартира), количество комнат, есть балкон?, стоимость аренды	Если в семье более 8 человек, нужен дом, иначе подойдут и дом и квартира. Каждому взрослому нужно по комнате. Двое детей могут жить в одной комнате. Если в семье есть кошка или собака, балкон обязателен. Сумма оплаты, предполагаемая семьей, должна быть больше или равна стоимости аренды жилья.
10	Проезд-ной билет	Транс-портное сред-ство	Город, вид (автобус, троллейбус, трамвай), предостав-ляются льготы?	Проездной билет может быть использован для поездки на транспортном средстве при совпадении их городов, если тип проездного соответствует виду транспортного средства (универсальный проездной соответствует любому виду транспортного средства), выполняется условие по льготам (если проездной льготный, то транспортное средство должно быть с предоставлением льгот).

Продожение таблицы 2.1

№	Класс1 (сущ-ность 1)	Класс2 (сущ-ность 2)	Поля (private)	Условие соответствия объектов класса1 и класса 2
11	Води-тель	Авто-парк	Название, тип (междуго-родние автобусы, грузовые автомобили, городское такси), директор	В автопарке городского такси могут работать водители всех категорий в возрасте от 20 до 60 лет. В автопарке междугородних автобусов могут работать водители категории D, в возрасте от 25 до 55 лет. В автопарке грузовых автомобилей могут работать водители категории С в возрасте от 25 до 55 лет.
12	Тур-поездка	Билет	Город отправления, город прибытия, вид транспорта, код транс-портного средства, дата отправления, время отправления, дата прибытия, время прибытия, стоимость	Для турпоездки можно взять билет, если совпадают города соответственно отправления и прибытия, вид транспорта, дата прибытия, и обещанная агентством максимальная стоимость билета не превышает реальной стоимости билета.
13	Розетка	Вилка	Код, с заземлением?, цвет, мощность	Розетка и вилка подходят друг другу, если они имеют одинаковый цвет и мощность. Для евророзетки нужна вилка с заземлением, для обычной – без заземления.
14	Растение	Регион	Название, минимальная температура за год, максимальная температура за год, количество дождливых дней в году.	Растение подходит для региона, в диапазон допустимых для него температур входит диапазон годовых температур для региона, а количество дождливых дней в году соответствует требованию к увлажнению почвы: интенсивное увлажнение – более 250 дождливых дней в году, среднее – от 150 до 250, редкое – от 50 до 150.

3.7. Содержание отчета

1) Цель работы.
2) Постановка задачи с указанием данных варианта задания.
3) Алгоритм метода accord().
4) Структура проекта.
5) Текст классов.
6) Результаты проверки метода accord() с помощью возможностей панели объектов.
7) Протокол отладки (описание найденных синтаксических и логических ошибок в программе и способов их устранения).
8) Результаты выполнения программы и их анализ.
9) Выводы.

3.8. Пример программы, реализующей взаимодействие двух объектов

Описание сущностей, подлежащих программированию, приведено в таблице 3.2.

Таблица 3.2 – Описание полей сущностей, подлежащих программированию

Класс1 (сущ-ность 1)	Поля (public)	Класс2 (сущ-ность 2)	Поля (private)	Условие соответствия объектов класса1 и класса 2
Крышка	Диаметр, материал, цвет, форма	Кастрюля	Объем, высота, цвет, двойное дно?	Крышка подходит к кастрюле, если она того же цвета и сделана из того же материала, или она стеклянная (стеклянная крышка любого цвета подходит к кастрюле из любого материала любого цвета) и ее площадь превышает площадь дна кастрюли не менее, чем на 0.5 см и не более, чем на 1.5 см

Структура проекта Lab2_sem2 приведена на рисунке 3.14.

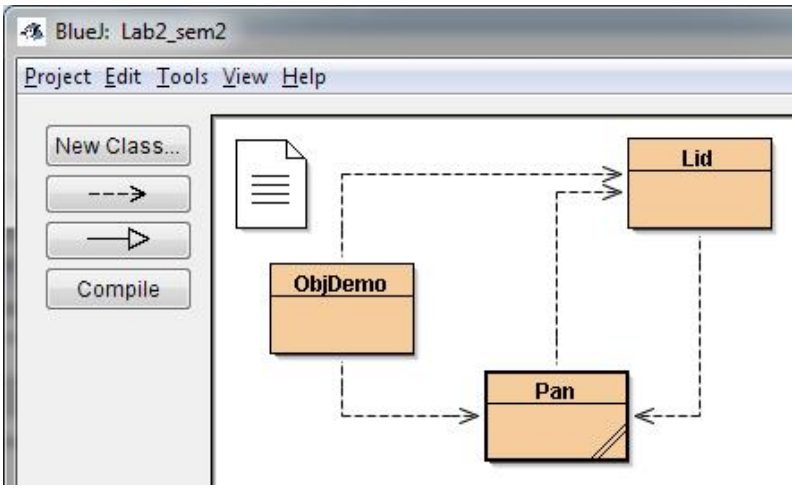

Рисунок 3.14 – Структура проекта, реализующего взаимодействие объектов класса Lid (крышка) и Pan (кастрюля)

Текст программы.

```
public class Lid implements Comparable <Lid> { //Сущность – Крышка

… //текст класса из предыдущей лабораторной работы

   public boolean accord (Pan pan){
   // определяет соответствие крышки кастрюле
   // текущий объект - крышка, объект-параметр - кастрюля
      return (material.equals("стеклянная") || color.equals(pan.getColor())
          && material.equals(pan.getMaterial()))
          && (d-pan.getD()) >= 0.5 && (d-pan.getD()) <= 1.5;
    }

 } //Lid

public class Pan implements Comparable <Pan> {
//Сущность - Кастрюля (Бак)
   /* Класс включает (реализует) интерфейс Comparable <Type>,
      значит он должен содержать реализацию метода compareTo(),
      заголовок которого определен в библиотечном
      классе-интерфейсе Comparable:   int  compareTo(Type ob);
    */
   //формат записи о кастрюле:
   private final static String PAN_FORMAT_STRING =
       "Кастрюля: диаметр=%5.2f см, высота=%5.2f см, %15s, %15s, %18s";
   //поля - переменные экземпляра (скрыты в классе)
   private double d; // диаметр основания
   private double  h; //высота в сантиметрах
   private String material; //материал
   private String color; //цвет
   private boolean doubleBottom ; //двойное дно?
   //методы-конструкторы (используется механизм перегрузки)
   //конструктор без параметров
```

```
public Pan(){
   this.d = 0.0; this.h = 0.0; this.material = "";
   this.color = ""; this.doubleBottom = false;
}
//конструктор с параметрами, инициализирующими поля объекта
public Pan(double d, double h, String material,
          String color, boolean doubleBottom){
   this.d = d; this.h = h;  this.material = material;
   this.color = color;   this.doubleBottom=doubleBottom;
 }
//конструктор с параметром-объектом
//(конструирует клон объекта)
public Pan(Pan p){
  d=p.d; h = p.h; material = p.material;
  color = p.color; doubleBottom = p.doubleBottom;
}
//доступ к полям

//методы-геттеры
public double getD(){return d;}
public double getH(){return h;}
public String getColor(){return color;}
public String getMaterial(){return material;}
public boolean getDoubleBottom(){return doubleBottom;}

//методы-сеттеры
public void setD(double d){this.d = d;}
public void setH(double h){this.h = h;}
public void setColor(String color){this.color = color;}
public void setMaterial(String material)
                                    {this.material = material;}
public void setDoubleBottom(boolean doubleBottom)
                            {this.doubleBottom = doubleBottom;}

//сравнение объектов на равенство (используется
//механизм переопределения метода в классе-наследнике)
//Переопределяется метод equals класса Object
 @Override  //инструкция компилятору: проверять,
  // действительно ли  методы переопределены
 // (а не просто перегружены)
 public boolean equals (Object ob){ //возвращает true при равенстве
                                    //объектов
      if (ob == null) return false;
      if (ob == this) return true;
      if (getClass()!= ob.getClass()) return false;
      Pan p = (Pan) ob;
      return (d == p.d) && (h == p.h)&& material.equals(p.material) &&
            color.equals(p.color) && (doubleBottom == p.doubleBottom);
 }
 //Переопределяется метод hashCode класса Object
 // Этот метод обязательно должен быть переопределен, если
 // был переопределен метод equals
```

```
    public int hashCode () { //возвращает хэш-код объекта
      return 7 * (new Double(d)).hashCode()
        + 11* (new Double(h)).hashCode()
        + 13 * material.hashCode()
        + 17 * color.hashCode()
        + 19 * (new Boolean(doubleBottom)).hashCode();
    }

    //Переопределяется метод toString класса Object
    public String toString(){//возвращает строку описания объекта
      return String.format(PAN_FORMAT_STRING,d, h, material, color,
           doubleBottom?"с двойным дном":"с одинарным дном");
   }

   //Сравнение объектов на больше/меньше
   //Реализация метода compareTo() интерфейса Comparable <Pan>
   public int compareTo (Pan p){
       if  (d < p.d) return -1;
       if ((d == p.d) && (h < p.h)) return -1;
       if ((d == p.d) && (h == p.h) &&
           (material.compareTo(p.material)<0)) return -1;
       if ((d == p.d) && (h == p.h) &&
           (material.compareTo(p.material)== 0) &&
           (color.compareTo(p.color)< 0)) return -1;
       if ((d == p.d) && (h == p.h) &&
           (material.compareTo(p.material)== 0) &&
           (color.compareTo(p.color)== 0) &&
            !doubleBottom && p.doubleBottom) return -1;
        if ((d == p.d) && (h == p.h) &&
           (material.compareTo(p.material)== 0) &&
           (color.compareTo(p.color)== 0) &&
            doubleBottom == p.doubleBottom) return 0;
        else return 1;
    }

   //другие полезные методы
   public boolean accord (Lid li){
    // определяет соответствие крышки кастрюле
    // текущий объект - кастрюля, объект-параметр - крышка
         return (li.getMaterial().equals("стеклянная") || color.equals(li.getColor())
               && material.equals(li.getMaterial()))
               && (li.getD()-d)>=0.5 && (li.getD()-d)<=1.5;
    }
} //Pan

public class ObjDemo { //Работа с объектами разных классов
    public static void bubbleSort (Comparable[ ] arr) {
    //сортировка массива объектов класса Comparable
    //методом пузырька
    //метод bubbleSort () подходит для сортировки массива,
    // состоящего из любых объектов, включающих интерфейс
    //Comparable, т.к. в нем используются интерфейсные ссылки
```

```
        boolean flag;
        for (int m = arr.length-1; m > 0; m--){
            flag=true;
            for (int j = 0; j < m; j++)
                if (arr[j].compareTo(arr[j+1]) > 0) {
                    //перестановка (переприсвоение ссылок)
                    Comparable b = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = b;
                    flag = false;
                 }
            if (flag) break;
        }
    } // пузырек

    //перегружаемые методы вывода массива объектов:
    public static void putArr (Lid [ ] arr) {
        //вывод массива объектов класса Lid
        for (int i = 0; i < arr.length; i++)
            System.out.printf ("%s, хэшкод: %d\n", arr[i], arr[i].hashCode());
    }
    public static void putArr (Pan [ ] arr) {
        //вывод массива объектов класса Pan
        for (int i=0; i < arr.length; i++)
            System.out.printf ("%s, хэшкод: %d\n", arr[i], arr[i].hashCode());
    }
/*Вместо двух перегружаемых методов вывода массива, можно
иметь один метод, в котором в качестве параметра используется
 интерфейсная ссылка (Comparable), как это было сделано
в проекте к предыдущей лабораторной работе. Сравните
 преимущества и недостатки этих двух подходов */

    public static void main (String args[ ]){
        /*1)Создаем кастрюли pan1 - pan4 и пустую ссылку pan5.
          Демонстрируем работу методов toString(), equals() и hashCode()
          для объектов класса Pan */
        System.out.print("1)Демонстрируем работу методов toString(), equals()");
        System.out.println  ("и hashCode()для объектов класса Pan");
        Pan pan1 = new Pan(30, 12, "металлическая", "золотистая", true);
        Pan pan2 = pan1; //создание псевдонима
        Pan pan3 = new Pan(pan1); //создание клона
        //Создание кастрюли с другими значениями полей
        Pan pan4 = new Pan(40, 20, "металлическая", "серебристая", false);
        Pan pan5 = null;
        //Вывод информации о созданных кастрюлях
        System.out.printf ("pan1: %s, хэшкод: %d\n", pan1, pan1.hashCode());
        System.out.printf ("pan2: %s, хэшкод: %d\n", pan2, pan2.hashCode());
        System.out.printf ("pan3: %s, хэшкод: %d\n", pan3, pan3.hashCode());
        System.out.printf ("pan4: %s, хэшкод: %d\n", pan4, pan4.hashCode());
        System.out.printf ("pan5: %s\n", pan5);
        //Вывод результатов сравнения на равенство
        System.out.println ("Результаты сравнения на равенство");
```

```
System.out.printf ("pan1==pan2: %s\n", pan1.equals(pan2));//true
System.out.printf ("pan1==pan3: %s\n", pan1.equals(pan3));//true
System.out.printf ("pan1==pan4: %s\n", pan1.equals(pan4));//false
System.out.printf ("pan4==pan5: %s\n", pan4.equals(pan5));//false

/*2)Демонстрируем унификацию сортировки элементов
   массивов arr1 и arr2 за счет использования в методе
   сортировки bubbleSort() интерфейсных ссылок (Comparable); */
System.out.println();
System.out.print ("2)Демонстрируем сортировку элементов массивов");
System.out.println  (" arr1 и arr2 одним и тем же методом bubbleSort().");
//Создание массива объектов класса Lid
Lid [ ] arr = new Lid [6];
arr[0] = new Lid(30.5, "металлическая","золотистая",true);
//отличается от arr[0] выпуклостью:
arr[1] = new Lid(30.5, "металлическая","золотистая",false);
//отличается от arr[0] цветом:
arr[2] = new Lid(30.5, "металлическая","серебристая",true);
//отличается от arr[0] материалом:
arr[3] = new Lid(30.5, "керамическая","серебристая",true);
//отличается от arr[0]  диаметром:
arr[4] = new Lid(15.5, "металлическая","золотистая",true);
//не отличается от arr[0]
arr[5] = new Lid(30.5, "металлическая","золотистая",true);
//Вывод массива до сортировки:
System.out.println ("Массив крышек Lid до сортировки:");
putArr(arr);
//Сортировка:
bubbleSort(arr);
//Вывод массива после сортировки:
System.out.println ("Массив крышек Lid после сортировки:");
putArr(arr);
//Создание массива объектов класса Pan
Pan [ ] arr1 = new Pan [7];
arr1[0] = new Pan(45.5, 20, "металлическая","серебристая",true);
//отличается от arr1[0] высотой:
arr1[1] = new Pan(45.5, 15, "металлическая","серебристая",true);
//отличается от arr1[0] отсутствием двойного дна:
arr1[2] = new Pan(45.5, 20, "металлическая","серебристая",false);
//отличается от arr1[0] цветом:
arr1[3] = new Pan(45.5, 20, "металлическая","золотистая",true);
//отличается от arr1[0] материалом:
arr1[4] = new Pan(45.5, 20, "керамическая","золотистая",true);
//отличается от arr1[0]  диаметром:
arr1[5] = new Pan(25.5, 20, "металлическая","золотистая",true);
//не отличается от arr1[0]:
arr1[6] = new Pan(45.5, 20, "металлическая","серебристая",true);
//Вывод массива до сортировки:
System.out.println ("Массив кастрюль Pan до сортировки:");
putArr(arr1);
//Сортировка:
bubbleSort(arr1);
```

```
//Вывод массива после сортировки:
System.out.println ("Массив кастрюль Pan после сортировки:");
putArr(arr1);

/* 3)Демонстрируем взаимодействие объектов, которое
  выражается в определении соответствия объекта
  класса Класс1 объекту класса Класс2 и наоборот.*/
System.out.println();
System.out.print ("3)Демонстрируем взаимодействие объектов при ");
System.out.println  ("определении соответствия крышек и кастрюль.");
//создание объектов
//кастрюля:
Pan pan = new Pan(30, 12, "металлическая","серебристая",true);
//крышки:
//первая: стеклянная, размер соответствует
Lid lid1 = new Lid(30.5,"стеклянная","темная",false);
//вторая: материал, цвет, размер соответствуют
Lid lid2 = new Lid(31.5,"металлическая","серебристая", true);
//третья: материал не совпадает
Lid lid3 = new Lid(30.5,"эмалированная","серебристая", true);
//четвертая: цвет не совпадает
Lid lid4 = new Lid(30.5,"металлическая","золотистая", false);
//пятая: крышка велика
Lid lid5 = new Lid(32,"металлическая","серебристая", true);
//шестая: крышка мала
Lid lid6 = new Lid(30,"металлическая","серебристая", true);
//выводим данные об объекте "кастрюля"
System.out.printf("pan: %s\n", pan);
// Выводим данные о объектах "крышка",
System.out.printf("lid1: %s\n", lid1);
System.out.printf("lid2: %s\n", lid2);
System.out.printf("lid3: %s\n", lid3);
System.out.printf("lid4: %s\n", lid4);
System.out.printf("lid5: %s\n", lid5);
System.out.printf("lid6: %s\n", lid6);
// Выводим результаты определения соответствия
System.out.printf("lid1.accord(pan): %s\n", lid1.accord(pan));
System.out.printf("pan.accord(lid1): %s\n", pan.accord(lid1));
System.out.printf("lid2.accord(pan): %s\n", lid2.accord(pan));
System.out.printf("pan.accord(lid2): %s\n", pan.accord(lid2));
System.out.printf("lid3.accord(pan): %s\n", lid3.accord(pan));
System.out.printf("pan.accord(lid3): %s\n", pan.accord(lid3));
System.out.printf("lid4.accord(pan): %s\n", lid4.accord(pan));
System.out.printf("pan.accord(lid4): %s\n", pan.accord(lid4));
System.out.printf("lid5.accord(pan): %s\n", lid5.accord(pan));
System.out.printf("pan.accord(lid5): %s\n", pan.accord(lid5));
System.out.printf("lid6.accord(pan): %s\n", lid6.accord(pan));
System.out.printf("pan.accord(lid6): %s\n", pan.accord(lid6));

//меняем значения полей объекта кастрюля
pan.setD(40); pan.setH(18);
```

```
        pan.setMaterial("эмалированная");
        pan.setColor("белая");
        pan.setDoubleBottom(false);
        //меняем значения полей шестой крышки
        lid6.setD(41); lid6.setMaterial("эмалированная");
        lid6.setColor("белая"); lid6.setConvex(false);
        //проверяем соответствие крышки и кастрюли
        //и выводим соответствующее сообщение
        if (lid6.accord(pan)) {
           System.out.printf("Объект %s\n",lid6);
           System.out.println("соответствует объекту");
           System.out.println(pan);
        }
        else{
           System.out.printf("Объект %s\n",lid6);
           System.out.println("не соответствует объекту");
           System.out.println(pan);
        }
     }
    } //ObjDemo
```

Алгоритм метода **public boolean accord (Pan pan)**, определенного в классе Lid приведен на рисунке 3.15.

Заметим, что в тексте метода вместо достаточно громоздкой совокупности операторов if-else, используется один оператор return, возвращающий значение составного логического выражения:

```
return (material.equals("стеклянная") || color.equals(pan.getColor())
    && material.equals(pan.getMaterial()))
    && (d-pan.getD()) >= 0.5 && (d-pan.getD()) <= 1.5;  .
```

Однако, разработка алгоритма перед написанием кода метода, позволила правильно составить сложное логическое условие и определить необходимое количество тестов (количество различных крышек для кастрюли) для проверки работы метода по всем ветвям.

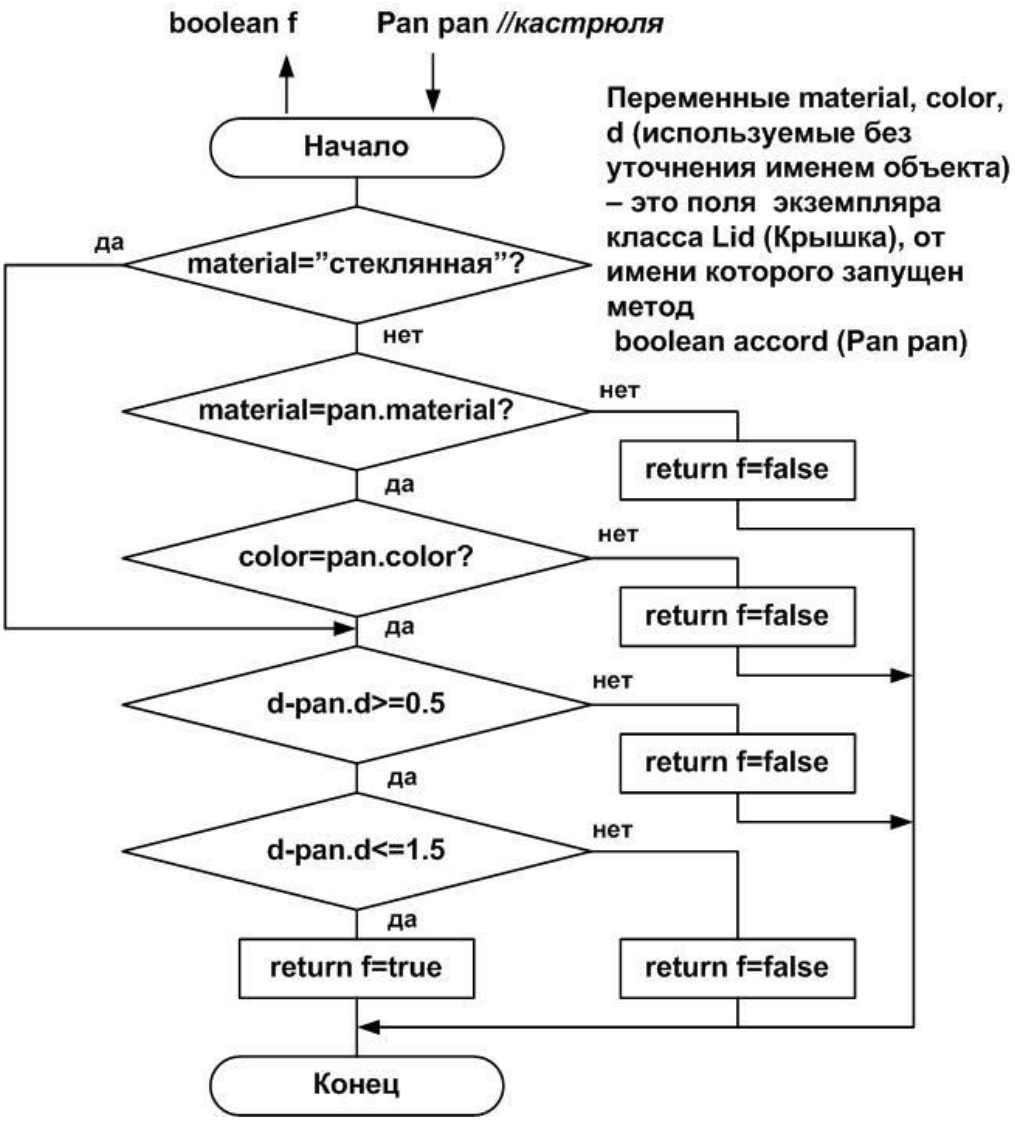

Рисунок 3.15 – Алгоритм метода accord() класса Lid

Перед компиляцией всего проекта классы Lid и Pan, в том числе, метод accord() классов Lid и Pan, были проверены с использованием возможностей панели объектов BlueJ.

Результаты работы программы представлены на рисунках 3.16-3.18.

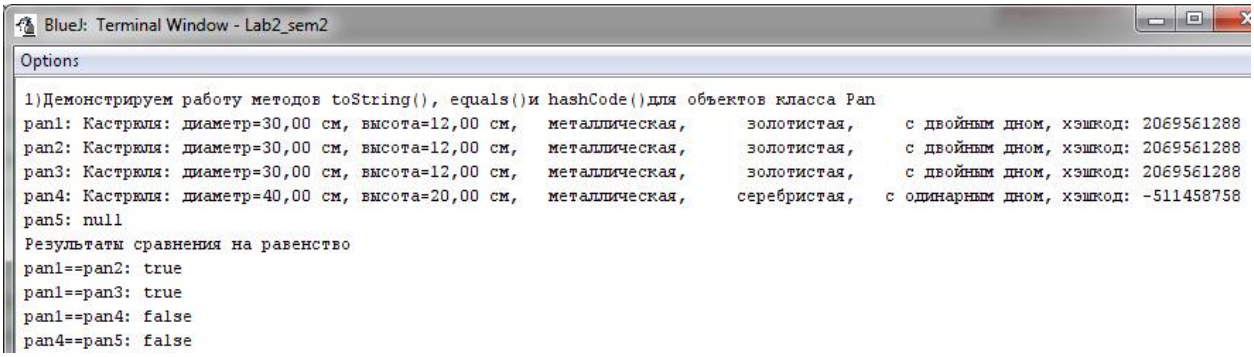

Рисунок 3.16 – Результаты использования методов toString(), equals() и hashCode() для объектов класса Pan

```
2)Демонстрируем сортировку элементов массивов arr1 и arr2 одним и тем же методом bubbleSort().
Массив крышек Lid до сортировки:
Крышка: диаметр=30,50 см,   металлическая,      золотистая,        выпуклая, хэшкод: -2048250116
Крышка: диаметр=30,50 см,   металлическая,      золотистая,         плоская, хэшкод: -2048250014
Крышка: диаметр=30,50 см,   металлическая,     серебристая,        выпуклая, хэшкод: -241014532
Крышка: диаметр=30,50 см,    керамическая,     серебристая,        выпуклая, хэшкод: 1391747438
Крышка: диаметр=15,50 см,   металлическая,      золотистая,        выпуклая, хэшкод: -2055360772
Крышка: диаметр=30,50 см,   металлическая,      золотистая,        выпуклая, хэшкод: -2048250116
Массив крышек Lid после сортировки:
Крышка: диаметр=15,50 см,   металлическая,      золотистая,        выпуклая, хэшкод: -2055360772
Крышка: диаметр=30,50 см,    керамическая,     серебристая,        выпуклая, хэшкод: 1391747438
Крышка: диаметр=30,50 см,   металлическая,      золотистая,         плоская, хэшкод: -2048250014
Крышка: диаметр=30,50 см,   металлическая,      золотистая,        выпуклая, хэшкод: -2048250116
Крышка: диаметр=30,50 см,   металлическая,      золотистая,        выпуклая, хэшкод: -2048250116
Крышка: диаметр=30,50 см,   металлическая,     серебристая,        выпуклая, хэшкод: -241014532
Массив кастрюль Pan до сортировки:
Кастрюля: диаметр=45,50 см, высота=20,00 см,   металлическая,     серебристая,     с двойным дном, хэшкод: -510197304
Кастрюля: диаметр=45,50 см, высота=15,00 см,   металлическая,     серебристая,     с двойным дном, хэшкод: -514522680
Кастрюля: диаметр=45,50 см, высота=20,00 см,   металлическая,     серебристая,   с одинарным дном, хэшкод: -510197190
Кастрюля: диаметр=45,50 см, высота=20,00 см,   металлическая,      золотистая,     с двойным дном, хэшкод: 2082226120
Кастрюля: диаметр=45,50 см, высота=20,00 см,    керамическая,      золотистая,     с двойным дном, хэшкод: -1064016538
Кастрюля: диаметр=25,50 см, высота=20,00 см,   металлическая,      золотистая,     с двойным дном, хэшкод: 2076147656
Кастрюля: диаметр=45,50 см, высота=20,00 см,   металлическая,     серебристая,     с двойным дном, хэшкод: -510197304
Массив кастрюль Pan после сортировки:
Кастрюля: диаметр=25,50 см, высота=20,00 см,   металлическая,      золотистая,     с двойным дном, хэшкод: 2076147656
Кастрюля: диаметр=45,50 см, высота=15,00 см,   металлическая,     серебристая,     с двойным дном, хэшкод: -514522680
Кастрюля: диаметр=45,50 см, высота=20,00 см,    керамическая,      золотистая,     с двойным дном, хэшкод: -1064016538
Кастрюля: диаметр=45,50 см, высота=20,00 см,   металлическая,      золотистая,     с двойным дном, хэшкод: 2082226120
Кастрюля: диаметр=45,50 см, высота=20,00 см,   металлическая,     серебристая,   с одинарным дном, хэшкод: -510197190
Кастрюля: диаметр=45,50 см, высота=20,00 см,   металлическая,     серебристая,     с двойным дном, хэшкод: -510197304
Кастрюля: диаметр=45,50 см, высота=20,00 см,   металлическая,     серебристая,     с двойным дном, хэшкод: -510197304
```

Рисунок 3.17 – Результаты сортировки массива объектов класса Lid и массива объектов класса Pan одним и тем же методом bubbleSort()

```
3)Демонстрируем взаимодействие объектов при определении соответствия крышек и кастрюль.
pan: Кастрюля: диаметр=30,00 см, высота=12,00 см,   металлическая,     серебристая,     с двойным дном
lid1: Крышка: диаметр=30,50 см,      стеклянная,          темная,        плоская
lid2: Крышка: диаметр=31,50 см,   металлическая,     серебристая,       выпуклая
lid3: Крышка: диаметр=30,50 см,   эмалированная,     серебристая,       выпуклая
lid4: Крышка: диаметр=30,50 см,   металлическая,      золотистая,        плоская
lid5: Крышка: диаметр=32,00 см,   металлическая,     серебристая,       выпуклая
lid6: Крышка: диаметр=30,00 см,   металлическая,     серебристая,       выпуклая
lid1.accord(pan): true
pan.accord(lid1): true
lid2.accord(pan): true
pan.accord(lid2): true
lid3.accord(pan): false
pan.accord(lid3): false
lid4.accord(pan): false
pan.accord(lid4): false
lid5.accord(pan): false
pan.accord(lid5): false
lid6.accord(pan): false
pan.accord(lid6): false
Объект Крышка: диаметр=41,00 см,   эмалированная,          белая,        плоская
соответствует объекту
Кастрюля: диаметр=40,00 см, высота=18,00 см,   эмалированная,          белая,   с одинарным дном
```

Рисунок 3.18 – Результаты применения метода accord(), запущенного от имени объектов класса Lid и объектов класса Pan

3.9. Контрольные вопросы

1) Какой основополагающий принцип объектно-ориентированного программирования исследовался в лабораторной работе?
2) Что такое класс?
3) Что такое объект?
4) Что в классе определяет структуру (свойства) некоторой сущности, а что ее природу (поведение)?
5) Что из себя представляют члены класса?
6) Что такое переменная экземпляра и почему она так называется.
7) Назовите синонимы выражения «переменная экземпляра».
8) Как осуществляется доступ к членам класса из методов-членов класса?
10) Как осуществляется управление доступом к членам класса из других классов? Охарактеризуйте различные спецификаторы доступа.
11) Как осуществляется доступ из других классов к переменным экземпляра, отмеченным спецификатором доступа private? Pablic?
12) Для чего нужны методы-геттеры и методы-сеттеры? Какой принцип написания Java-программ считается лучшим с точки зрения безопасности кода, инкапсулированного в классе?
13) Какие статические члены допускаются в классах-шаблонах для создания объектов?
14) Какой метод должен содержать класс, являющийся стартовой точкой программы? Какой спецификатор доступа должен быть у этого метода и почему?
15) Какой операцией создается объект? Приведите пример и опишите последовательность действий, которые при этом производит исполнительная система Java.
16) Что такое конструктор класса и для чего он используется? Может ли в классе быть определено несколько конструкторов? Чем тогда они должны отличаться?
17) Что понимается под перегрузкой методов в Java? Приведите пример из своего проекта. Какое преимущество дает перегрузка?
18) Что такое переопределение методов суперкласса? Какие методы какого класса вы переопределяли? Приведите пример из своего проекта.
19) Для чего используется ключевое слово this? Приведите примеры.
20) Что такое ссылка на объект? Как она определяется в Java-программе? Что означает Null-значение ссылки? Какими способами можно присвоить ссылочной переменной адрес конкретного объекта?
21) Чем отличается копирование объекта от копирования ссылки на объект? Приведите примеры.
22) Что выведет оператор **System.out.println(lid1)** при отсутствии в классе Lid метода toString()?
23) Для чего в классах-шаблонах вашей программы вы использовали интерфейс Comparable? Какой метод этого интерфейса был вами реализован?
24) Назовите преимущества и недостатки подхода, предполагающего обращение к объектам с помощью интерфейсных ссылочных переменных.
25) Опишите возможности, которые предоставляет панель объектов BlueJ.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ноутон П. Java 2 / П. Ноутон, Г. Шилдт. – СПб. : БХВ-Петербург, 2007. – 1072 с. — Текст : непосредственный.

2. Эккель Б. Философия Java / Б. Эккель – СПб : Питер, 2015 – 1168 с. — Текст : непосредственный.

3. Пономарчук, Ю. В. Программирование на языке Java : учебное пособие / Ю. В. Пономарчук, И. В. Кузнецов. — Хабаровск : ДВГУПС, 2021. — 103 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/259451 (дата обращения: 09.11.2022). — Режим доступа: для авториз. пользователей.

4. Хабитуев, Б. В. Программирование на языке Java: практикум : учебное пособие / Б. В. Хабитуев. — Улан-Удэ : БГУ, 2020. — 94 с. — ISBN 978-5-9793-1548-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/171791 (дата обращения: 09.11.2022). — Режим доступа: для авториз. пользователей.

5. Коузен, К. Современный Java: рецепты программирования / К. Коузен. — Москва : ДМК Пресс, 2018. — 275 с. — ISBN 978-5-97060-134-1. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/116121 (дата обращения: 09.11.2022). — Режим доступа: для авториз. пользователей.

6. Гуськова, О. И. Объектно ориентированное программирование в Java : учебное пособие / О. И. Гуськова. - Москва : МПГУ, 2018. - 240 с. - URL: https://znanium.com/catalog/product/1020593 (дата обращения: 10.11.2022). – Режим доступа: по подписке. - ISBN 978-5-4263-0648-6. - Текст : электронный.

ПРОГРАММИРОВАНИЕ. ЯЗЫКИ ВЫСОКОГО УРОВНЯ

Методические указания

Часть 2

Использование принципа наследования – создание суперклассов и подклассов. Взаимодействие объектов разных классов в java-программе

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

В авторской редакции

Изд. № 61/2023. Объем 4 п.л. Усл. печ. л. 3,72. Уч.-изд. л. 3,92.
РИИЦМ ФГАОУ ВО «Севастопольский государственный университет»