МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Программная инженерия»

Отчет по лабораторной работе № 1.

по дисциплине «Информационные технологии и программирование»

по разделу:

«2. Объектно-ориентированное программирование»

Выполнила: студентка группы БПИ2401 Рябова Екатерина

Проверил: Харрасов Камиль Раисович

Москва

2025

Содержание

Цель работы:

Познакомиться с объектно-ориентированным программированием на Java, изучить его принципы, особенности, и способы применения.

Ход работы:

Выполним Листинг 2.1. Реализация инкапсуляции. Класс Account.

Напишем программу из методички, создадим каталог для второй лабораторной работы и сохраним программу под именем Account. Код программы представлен на рисунке 1.

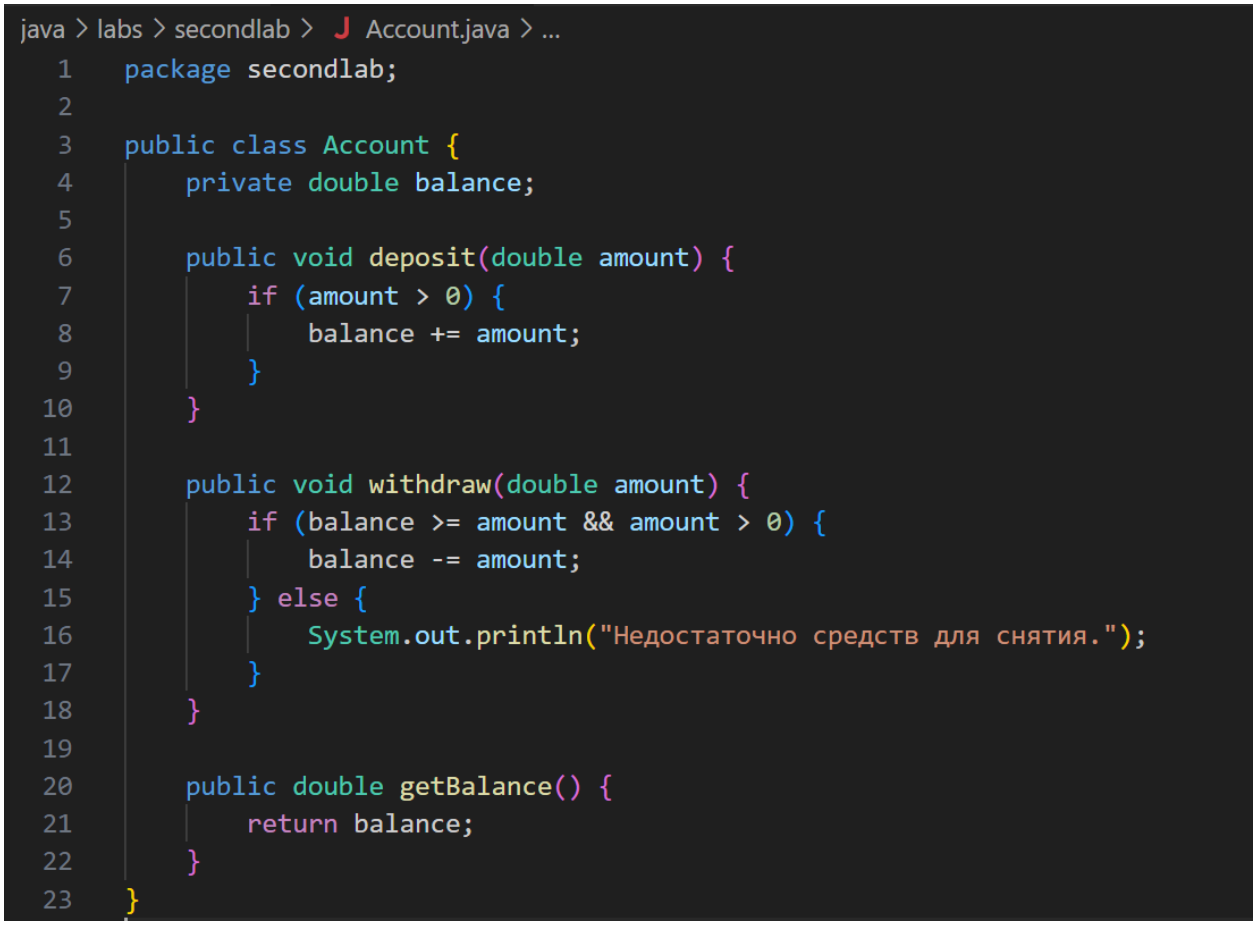

Рисунок 1. Код программы Account.java

Создадим класс AccountMain, в котором протестируем работу класса, создав его экземпляр и пытаясь выполнить различные методы. Проверим так же возможность доступа к «счету» извне. Код файла AccountMain и результаты тестирования представлены на рисунках 2-3.

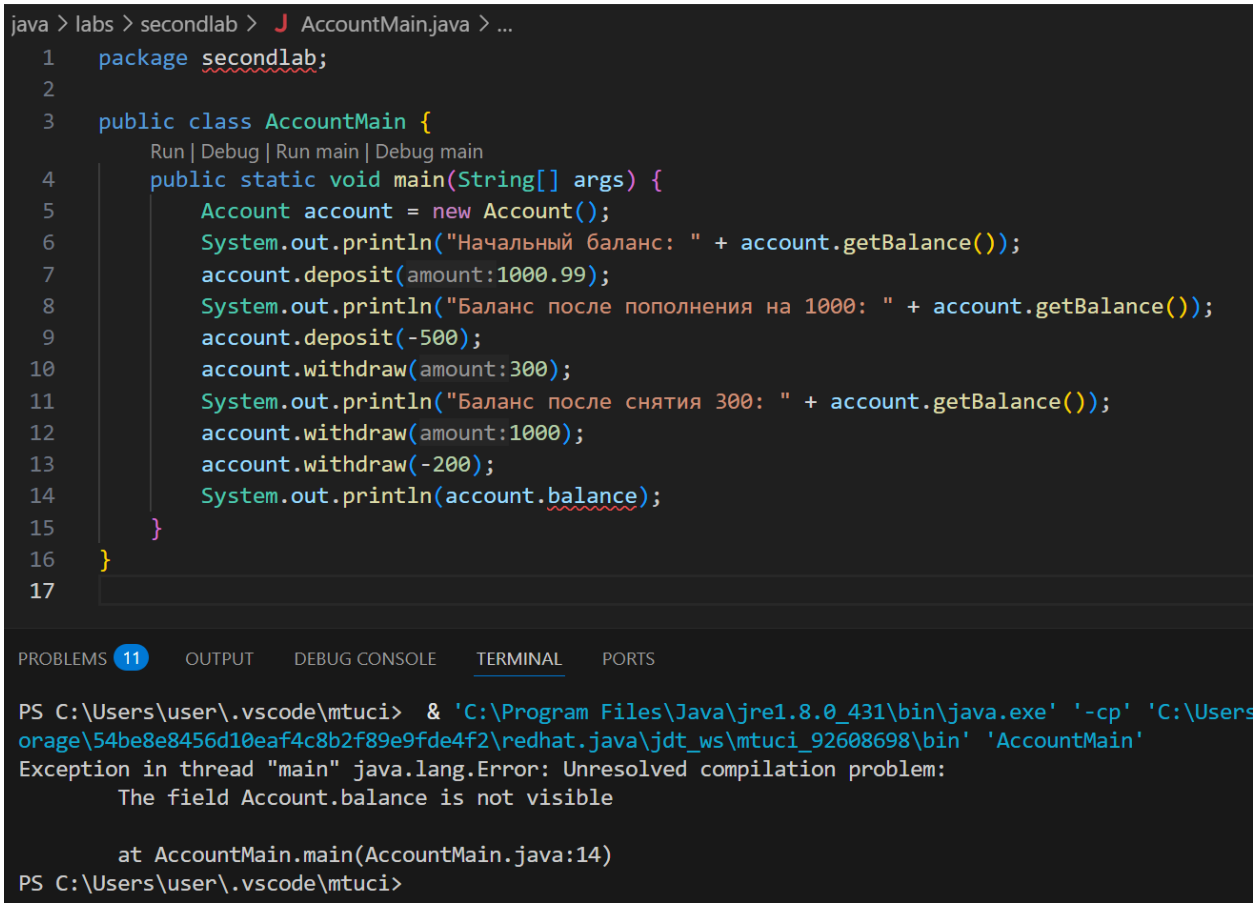

Рисунок 2. Код файла AccountMain и ошибка на этапе компиляции, когда мы пытаемся получить доступ к приватному полю класса.

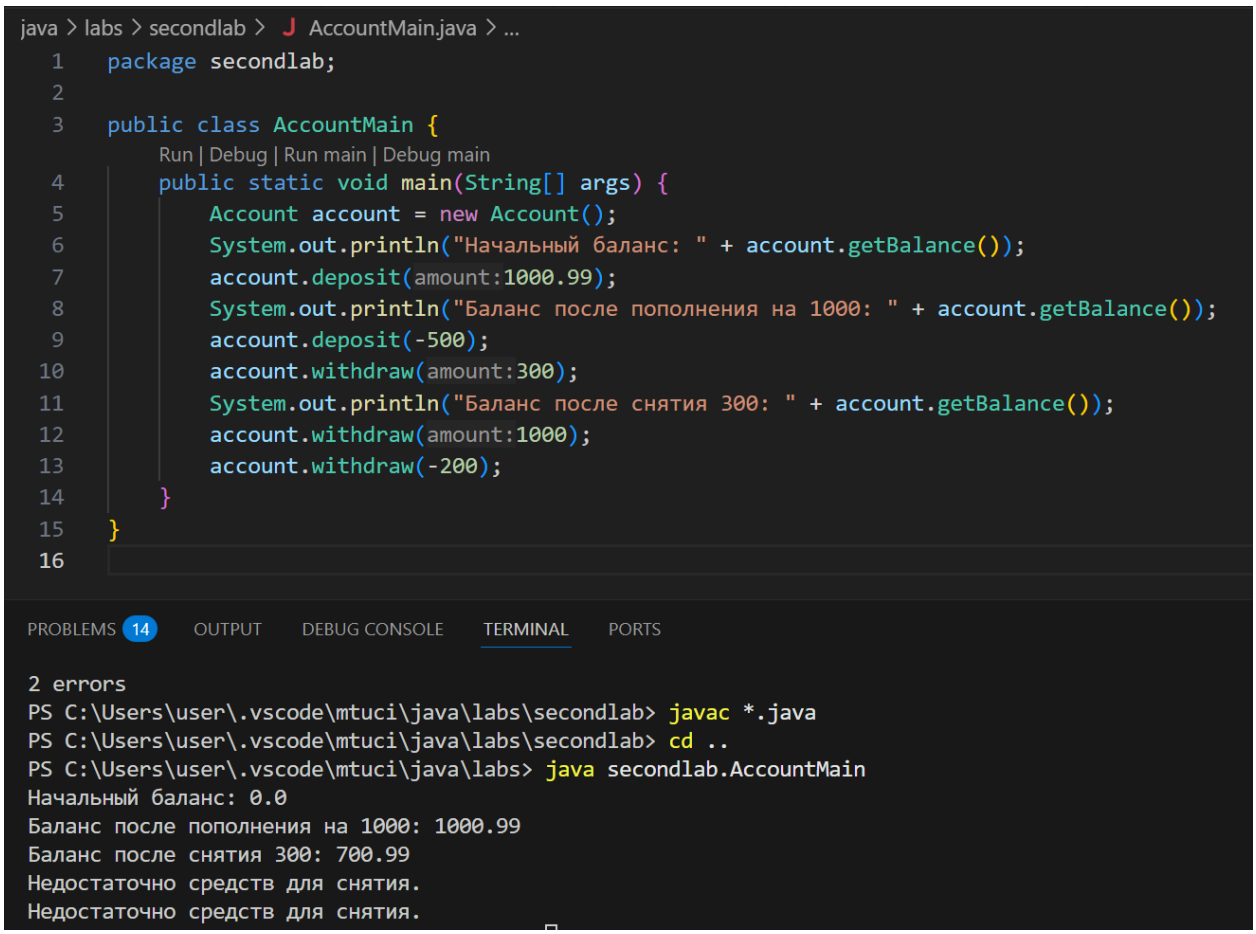

Рисунок 3. Результат корректной работы класса Account.

Выполним Листинг 2.2. Реализация наследования.

Напишем программу из методички. Для этого нам понадобится 2 различных файла – для класса Animal (с наследником Dog) и для класса Main. В файле Main импортировать ничего не нужно, так как Main.java и Animal.java принадлежат одному пакету - animals. При этом конфликтов в папке Animal.java не будет, так как там только один public класс. (Dog же виден в любом файле пакета). Код программы и архитектура представлены на рисунках 4-6.

java > labs > secondlab > animals > J Animal.java > ...

```java
package animals;

public class Animal {
    protected String name;
    public void eat() {
        System.out.println(name + " ест.");
    }
}

class Dog extends Animal {
    public void bark() {
        System.out.println(name + " лает.");
    }
}
```

Рисунок 4. Код файла Animal.java где представлен класс и его наследник.

java > labs > secondlab > animals > J Main.java > ...

```java
package animals;

public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String[ ] args) {
        Dog dog = new Dog();
        dog.name = "Шарик";
        dog.eat(); // Шарик ест.
        dog.bark(); // Шарик лает.
    }
}
```

Рисунок 5. Код программы в файле Main.

Рисунок 6. Архитектура: файлы Main и Animals и их байт-коды.

Заметим на рисунке 7, что несмотря на выделенные vscode ошибки все файлы успешно скомпилировались и корректно выполняются.

```
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab\animals> javac *.java
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab\animals> cd ..
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab> java animals.Main
Шарик ест.
Шарик лает.
```

Рисунок 7. Результат работы программы.

Выполним Листинг 2.4. Реализация интерфейсов.

Напишем программу из методички в файле MyClass. Там напишем два интерфейса, а затем класс, реализующий оба этих контракта. Создадим в том же пакете файл Main, где протестируем нашу программу, создав экземпляр класса MyClass и вызвав методы A и B. Код программы и архитектура представлены на рисунках 8-10.

java > labs > secondlab > interfaces > MyClass.java > ...

```java
package interfaces;

interface InterfaceA {
    void methodA();
}

interface InterfaceB {
    void methodB();
}

public class MyClass implements InterfaceA, InterfaceB {
    // Реализация методов из обоих интерфейсов
    @Override
    public void methodA() {
        System.out.println("Метод A");
    }

    @Override
    public void methodB() {
        System.out.println("Метод B");
    }
}
```

Рисунок 8. Код файле MyClass.java

java > labs > secondlab > interfaces > Main.java > ...

```java
package interfaces;

public class Main {
    Run main | Debug main | Run | Debug
    public static void main(String[ ] args) {
        MyClass exmp = new MyClass();
        exmp.methodA();
        exmp.methodB();
    }
}
```

Рисунок 9. Код в файле Main.java

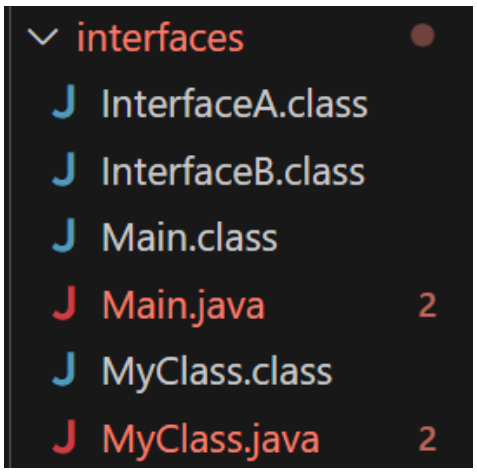

Рисунок 10. Архитектура файлов для листинга 2.4

Можно заметить, что в отдельные байт-коды сгенерировались и интерфейсы. Результат работы представлен на рисунке 11:

```
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab\interfaces> javac *.java
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab\interfaces> cd ..
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab> java interfaces.Main
Метод A
Метод B
```

Рисунок 11. Результат тестирования листинга 2.4

Выполним Листинг 2.5. Реализация перегрузки.

Напишем программу из методички в файле Main. Там напишем оба класса (так как Calculator не public то ошибки не будет, при этом при компиляции у нас сгенерируется два байт-кода по одному на каждый класс.) Код программы и результат работы представлены на рисунках 12-13.

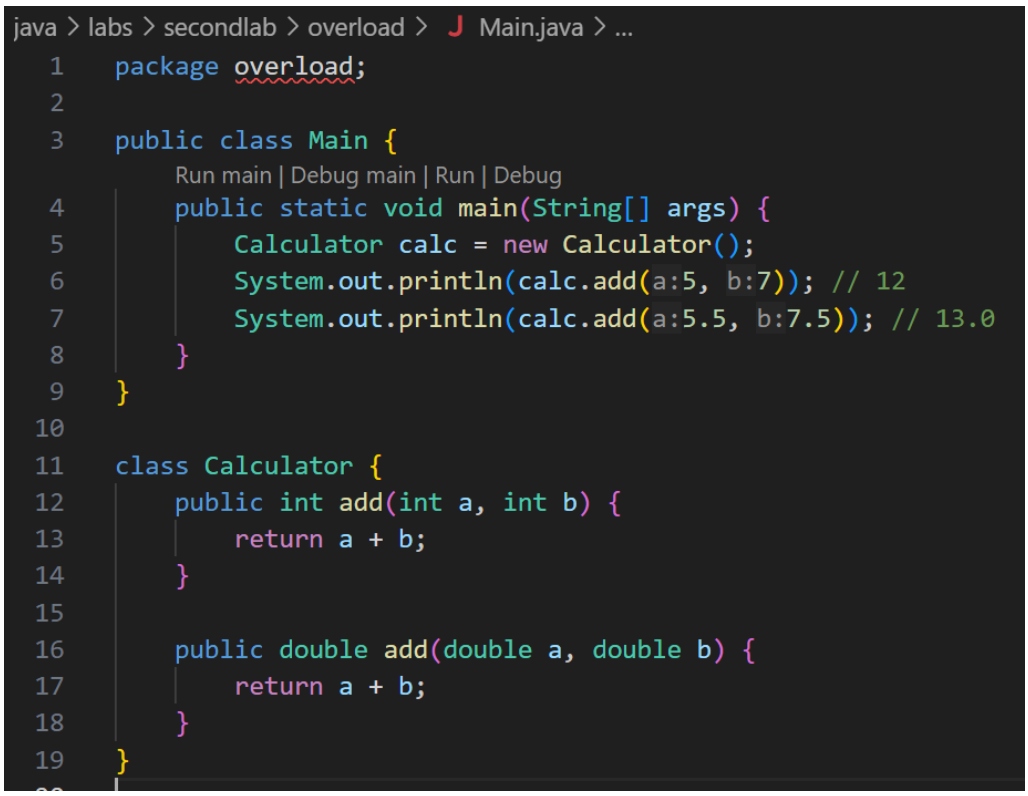

Рисунок 12. Код программы из листинга 2.5

```
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab\overload> javac Main.java
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab\overload> cd ..
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab> java overload.Main
12
13.0
```

Рисунок 13. Результат работы программы.

Выполним Листинг 2.6. Реализация переопределения.

Напишем программу из методички в файле Main. Там напишем оба класса с животными – Animal и его наследник Cat – а также public класс Main. Код программы и результат работы представлены на рисунках 14-15.

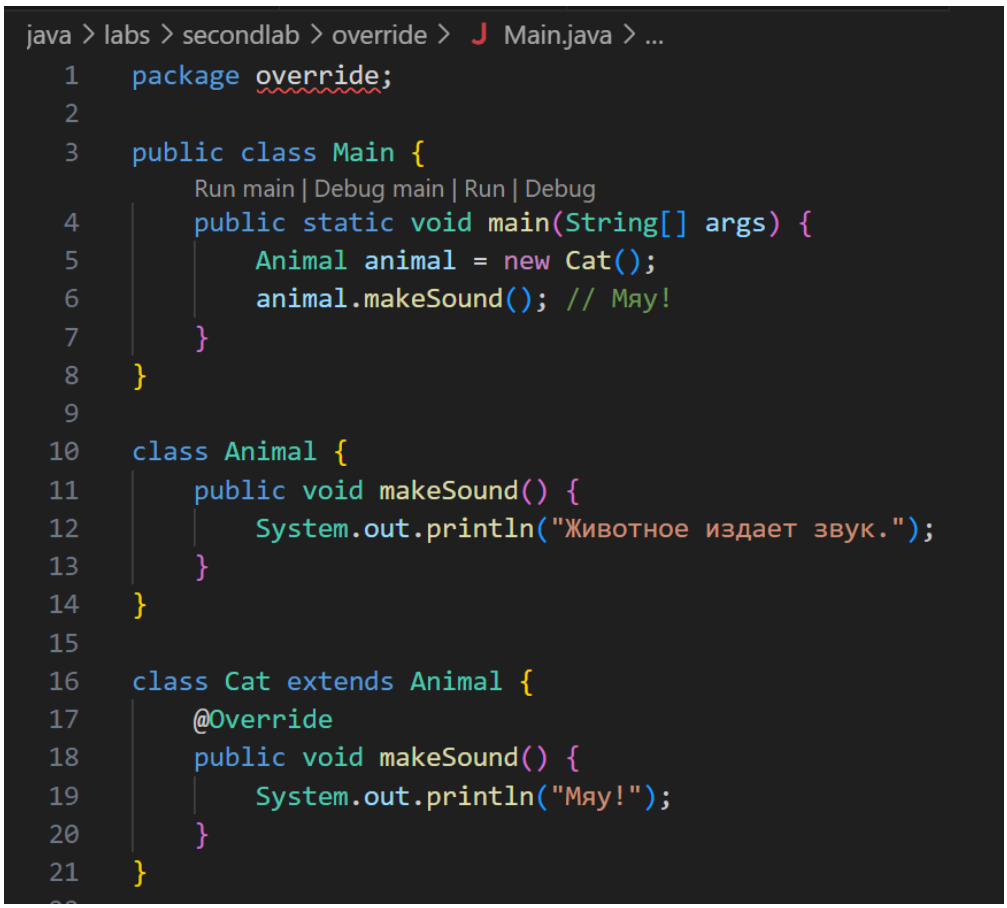

Рисунок 14. Программа листинга 2.6

```
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab> cd override
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab\override> java Main.java
Мяу!
```

Рисунок 15. Результат переопределения метода makeSound().

Выполним Листинг 2.7. Реализация абстракции.

Напишем программу из методички в файле Main. Напишем абстрактный класс Shape с абстрактным методом area(), определенном на нем. От него унаследуем класс Circle (Круга) и Rectangle (Прямоугольника). Для них реализуем метод area(). Код программы и результат работы представлены на рисунках 16-17.

java > labs > secondlab > abstraction > Main.java > Java > Rectangle

```java
package abstraction;

abstract class Shape {
    abstract double area();
}

class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    double area() {
        return Math.PI * radius * radius;
    }
}

class Rectangle extends Shape {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    @Override
    double area() {
        return width * height;
    }
}
```

Рисунок 16. Код программы для вычисления площади фигур ч.1

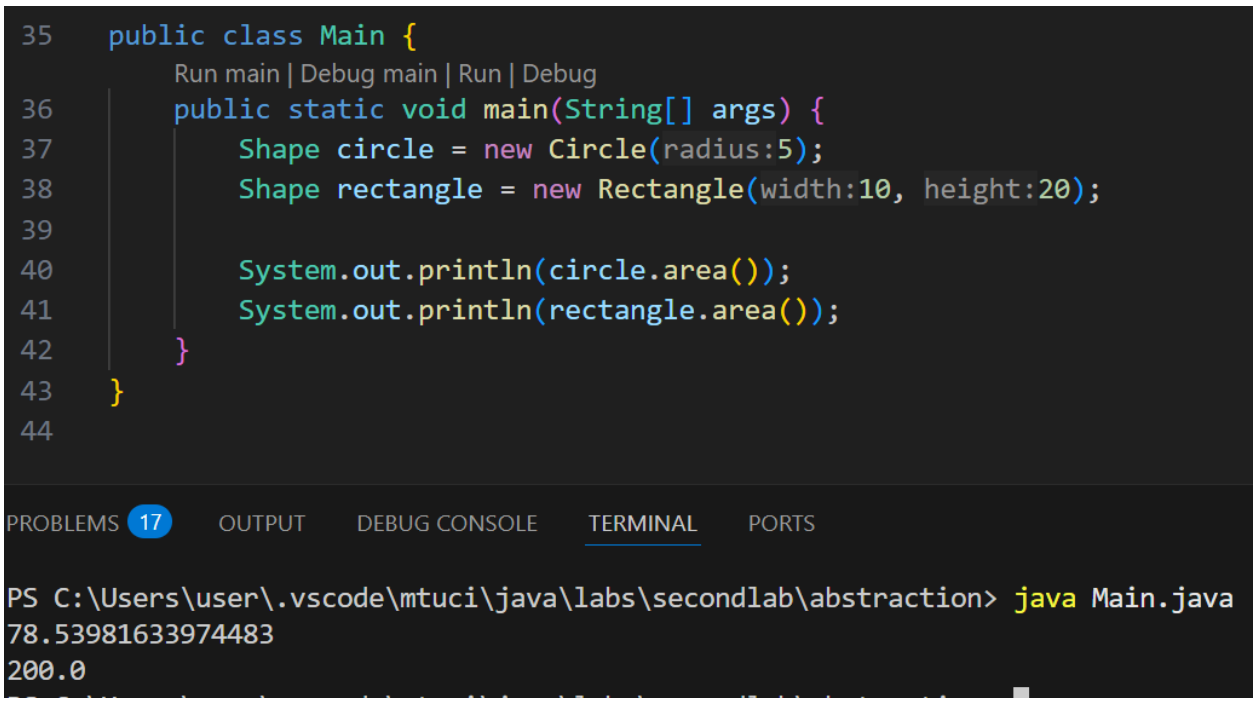

Рисунок 17. . Код программы для вычисления площади фигур ч.2 и результат выполнения этой программы.

Выполним Листинги 2.8 – 2.9.

Напишем программы с реализациями двумерной и трехмерной точек из методички. Создадим package points. Там для каждого из публичных классов Point2d и Point3d создадим по файлу. Создадим файл Main, в котором протестируем работу методов этих классов. Класс Point3d наследуем от Point2d с помощью ключевого слова extends. Напишем абстрактный класс Shape с абстрактным методом area(), определенном на нем. От него унаследуем класс Circle (Круга) и Rectangle (Прямоугольника). Для них реализуем метод area(). Код программы и результат работы представлены на рисунках 18-21.

java > labs > secondlab > points > J Point2d.java > ...

```java
package points;

public class Point2d {
    /** координата X **/
    private double xCoord;
    /** координата Y **/
    private double yCoord;
    /** конструктор с параметрами **/
    public Point2d(double x, double y) {
        xCoord = x;
        yCoord = y;
    }
    /** конструктор по умолчанию **/
    public Point2d() {
        this(x:0, y:0);
    }
    /** получение координаты X **/
    public double getX() {
        return xCoord;
    }
    /** получение координаты Y **/
    public double getY() {
        return yCoord;
    }
    /** установка значения координаты X. **/
    public void setX(double val) {
        xCoord = val;
    }
    /** установка значения координаты Y. **/
    public void setY(double val) {
        yCoord = val;
    }
}
```

Рисунок 18. Файл класса Point2d.

java > labs > secondlab > points > J Point3d.java > ...

```java
package points;

public class Point3d extends Point2d{
    private double zCoord;
    /** конструктор с параметрами **/
    public Point3d(double x, double y, double z) {
        super(x, y);
        zCoord = z;
    }
    /** конструктор по умолчанию **/
    public Point3d() {
        this(x:0, y:0, z:0);
    }
    /** получение координаты Z **/
    public double getZ() {
        return zCoord;
    }
    /** установка значения координаты Z. **/
    public void setZ(double val) {
        zCoord = val;
    }
}
```

Рисунок 19. Файл класса Points3d.

```java
java > labs > secondlab > points > J Main.java > ...
package points;

public class Main {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        Point2d myPoint = new Point2d();
        Point2d myOtherPoint = new Point2d(x:5,y:3);
        Point3d myThirdPoint = new Point3d();
        Point3d myLastPoint = new Point3d(x:4, y:5, z:3);

        myPoint.setX(val:6.2);
        myOtherPoint.setY(val:9);

        System.out.println(myPoint.getX());
        System.out.println(myOtherPoint.getY());

        myThirdPoint.setX(val:12);
        myThirdPoint.setZ(val:13);
        myLastPoint.setY(val:7.9);

        System.out.println(myThirdPoint.getX());
        System.out.println(myThirdPoint.getZ());
        System.out.println(myLastPoint.getY());

    }
}
```

Рисунок 20. Файл класса Main

```
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab> java points.Main
6.2
9.0
12.0
13.0
7.9
```

Рисунок 21. Результат выполнения программы Main.

Задание 1 для выполнения лабораторной работы:

Создайте иерархию классов в соответствии с вариантом. Ваша иерархия должна содержать:

- *абстрактный класс;*

- *два уровня наследуемых классов (классы должны содержать в себе минимум 3 поля и 2 метода, описывающих поведение объекта);*
- *демонстрацию реализации всех принципов ООП;*
- *наличие конструкторов (в том числе по умолчанию);*
- *наличие геттеров и сеттеров;*
- *ввод/вывод информации о создаваемых объектах;*
- *предусмотрите в одном из классов создание счетчика созданных объектов с использованием статической переменной, продемонстрируйте работу.*

Вариант 8: Базовый класс: Мебель. Дочерние классы: Стол, Стул, Кровать.

Так как должно быть два уровня наследуемых классов, то сделаем следующие классы: Кухонный (домашний) стол и Офисный стол – наследуются от класса Стол, Кухонный (домашний) стул и Офисный стул – наследуются от класса Стул, Кровать и Диван-кровать наследуются от класса Кровать.

Создадим пакет furniture, в нем создадим семь файлов:

- Furniture.java с public abstract class Furniture.
- FurnitureWithLegs.java public interface FurnitureWithLegs
- OfficeFurniture.java public interface OfficeFurniture
- Chair.java с public class Chair и двумя обычными классами – OfficeChair и HomeChair.
- Table.java с public class Table и двумя его наследниками – OfficeTable и HomeTable
- Bed.java с public class Bed и наследуемыми от нее BedBed и SofaBed.
- Main.java в котором мы продемонстрируем работу методов.

Public abstract class Furniture

Код в файле Furniture.java представлен в листинге 1.1:

Листинг 1.1

```
package furniture;

public abstract class Furniture {
    protected String name;
    protected String material;
    protected double price;
    private static int count = 0;

    public Furniture() { //конструктор по умолчанию
        count++; //счетчик созданных объектов (учет товаров в мебельном магазине)

    }

    public Furniture(String name, String material, double price) {
        this.name = name;
        this.material = material;
        this.price = price;
        count++; //счетчик созданных объектов (учет товаров в мебельном магазине)
    }

    public abstract void move(); // звук перетаскивания
    public abstract void use();  // как используется

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getMaterial() {
        return material;
    }

    public void setMaterial(String material) {
        this.material = material;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
```

```
        this.price = price;
    }

    public static int getCount() {
        return count;
    }

    public void info() {
        System.out.println("Название: " + name + ", Материал: " + material + ",
Цена: " + price);
    }
}
```

У каждого объекта мебели есть название, цена и материал, из которого он изготовлен. Создадим также еще одно поле – private static count – это счетчик наших объектов, а static он потому, что не является полем одного конкретного объекта, а значит, и внутри методов мы его вызываем без ссылки на объект – this. Создадим два конструктора – по умолчанию и с параметрами. И в том, и в другом, будем прибавлять к счетчику 1. Так при инициализации каждого объекта класса Furniture и любого его наследника счетчик будет увеличиваться на 1. Создадим два абстрактных метода – метод перетаскивания (будет печататься характерный звук при перемещении) и метод использования с краткой инструкцией. Напишем геттеры и сеттеры для всех полей класса, кроме счетчика, для него только метод get. Создадим так же метод, где будет выводиться информация об объекте.

Public interface FurnitureWithLegs

Код в файле FurnitureWithLegs.java представлен в листинге 1.2:

Листинг 1.2

```
package furniture;

public interface FurnitureWithLegs {
    int getNumberOfLegs();
    void setNumberOfLegs(int numberOfLegs);
}
```

У мебели с ножками обязательно должны быть геттеры и сеттеры кол-ва ножек.

Public interface OfficeFurniture

Код в файле OfficeFurniture.java представлен в листинге 1.3:

Листинг 1.3

```
package furniture;

public interface OfficeFurniture {
    int inventory();
}
```

У офисной мебели должен быть реализован метод учета объектов.

Public class Chair extends Furniture

Этот класс реализован в файле Chair.java, код представлен в листинге 1.4:

Листинг 1.4

```
package furniture;

public class Chair extends Furniture implements FurnitureWithLegs{
    protected boolean hasSoftSeat;
    protected int numberOfLegs;
    protected boolean hasArmrests;

    public Chair() {
        super();
    }

    public Chair(String name, String material, double price,
                 boolean hasSoftSeat, int numberOfLegs, boolean hasArmrests) {
        super(name, material, price);
        this.hasSoftSeat = hasSoftSeat;
        this.numberOfLegs = numberOfLegs;
        this.hasArmrests = hasArmrests;
    }

    @Override
    public void move() {
        System.out.println("тшшшшш");
    }

    @Override
    public void use() {
        System.out.println("Это стул - на нем сидят");
    }
```

```
    public boolean hasSoftSeat() {
        return hasSoftSeat;
    }
    public void setHasSoftSeat(boolean hasSoftSeat) {
        this.hasSoftSeat = hasSoftSeat;
    }

    @Override
    public int getNumberOfLegs() {
        return numberOfLegs;
    }

    @Override
    public void setNumberOfLegs(int numberOfLegs) {
        this.numberOfLegs = numberOfLegs;
    }

    public boolean hasArmrests() {
        return hasArmrests;
    }

    public void setHasArmrests(boolean hasArmrests) { this.hasArmrests =
hasArmrests; }

    @Override
    public void info() {
        super.info(); //выводим стандартные параметры
        System.out.println("Мягкое сиденье: " + (hasSoftSeat ? "да" : "нет") + ",
Кол-во ножек: " + numberOfLegs + ", Подлокотники: " + (hasArmrests ? "есть" :
"нет"));
        //используем тернарный оператор
    }
}
```

У стула реализуем интерфейс мебели с ножками. У стула появляются дополнительные параметры – наличие мягкой сидушки, подлокотников и количество ножек. В каждом из двух конструкторов мы вызываем конструктор родительского класса. В случае параметризованного конструктора, так как у нас появились дополнительные параметры, мы будем вынуждены отдельно прописать, что делать с ними. Переопределим методы движения и использования, напишем метод проверки наличий мягкого сидения и подлокотников, геттер кол-ва ножек и сеттеры для всех этих полей. (Для кол-

ва ножек геттер и сеттер мы переопределяем). Переопределим метод вывода информации: для начала вызовем родительский метод, а потом выведем информацию и о дополнительных параметрах.

Class OfficeChair extends Chair

Этот класс реализован там же, где и родительский класс – в файле Chair.java, код представлен в листинге 1.5:

Листинг 1.5

```
class OfficeChair extends Chair implements OfficeFurniture{
    private int height; //высота стула
    private static int invCount = 0;

    public OfficeChair() {
        super();
        this.height = 50; // стандартная высота
        invCount ++;
    }

    public OfficeChair(String name, String material, double price,
                       boolean hasSoftSeat, int numberOfLegs, boolean
hasArmrests,
                       int height) {
        super(name, material, price, hasSoftSeat, numberOfLegs, hasArmrests);
        this.height = height;
        invCount ++;
    }

    @Override
    public void move() {
        System.out.println("вжжжжух");
    }

    @Override
    public void use() {
        System.out.println("Это офисный стул - на нем устраивают гонки по
офису");
    }

    @Override
    public int inventory(){
        return invCount;
    }

    public void changeHeight(int increase) {
```

```
        this.height += increase;
    }

    public int getHeight() {
        return height;
    }

    public void setHeight(int newHeight) {
        this.height = newHeight;
    }

    @Override
    public void info() {
        super.info();
        System.out.println("Текущая высота: " + height + "см");
    }
}
```

У офисных стульев реализуем интерфейс офисной мебели. Таким образом, у офисного стула реализованы оба интерфейса – мебели с ножками и офисной мебели. Предполагается, что офисные стулья на одной ножке с колесиками. Еще у них регулируется высота, которая и станет дополнительным полем, кроме того приватным статическим полем станет и кол-во экземпляров класса. Аналогично предыдущему классу мы вызываем конструктор родительского класса (в данном случае Chair) и инициализируем оставшиеся поля, в том числе прибавляем 1 к счетчику. Переопределяем методы движения и использования, ведь теперь стул на колесиках. Добавляем сеттер и геттер высоты, а также метод ее регулировки – он отличен от геттера тем, что мы изменяем на сколько-то высоту. Аналогично добавим строчку с дополнительным параметром в методе вывода информации. Переопределим метод инвентаризации.

Class HomeChair extends Chair

Этот класс также реализован в файле Chair.java, код представлен в листинге 1.6:

Листинг 1.6

```
class HomeChair extends Chair {
    public HomeChair() {
```

```
        super();
    }

    public HomeChair(String name, String material, double price,
                     boolean hasSoftSeat, int numberOfLegs, boolean hasArmrests)
{
        super(name, material, price, hasSoftSeat, numberOfLegs, hasArmrests);
    }

    @Override
    public void use() {
        System.out.println("Это домашний стул - на него встают, чтобы поменять
лампочки");
    }
}
```

Для домашнего стула, не имеющего дополнительных параметров мы только пропишем конструкторы, в которых используется только super и переопределим метод использования.

Public class Bed extends Furniture

Этот класс реализован в файле Bed.java, код представлен в листинге 1.7:

Листинг 1.7

```
package furniture;

public class Bed extends Furniture{
    protected double width;
    protected double length;
    protected int sleepingPlaces;

    public Bed() {
        super();
    }

    public Bed(String name, String material, double price, double width, double
length, int sleepingPlaces) {
        super(name, material, price);
        this.width = width;
        this.length = length;
        this.sleepingPlaces = sleepingPlaces;
    }

    @Override
    public void move() {
```

```
        System.out.println("скрииииип");
    }

    @Override
    public void use() {
        System.out.println("Это кровать - на ней спят");
    }

    public double getWidth() {
        return width;
    }
    public void setWidth(double width) {
        this.width = width;
    }

    public double getLength() {
        return length;
    }
    public void setLength(double length) {
        this.length = length;
    }

    public int getSleepingPlaces() {
        return sleepingPlaces;
    }
    public void setSleepingPlaces(int sleepingPlaces) {
        this.sleepingPlaces = sleepingPlaces;
    }

    public double area() {
        return width * length;
    }

    @Override
    public void info() {
        super.info();
        System.out.println("Размер: " + width + "*" + length + "м, Спальных мест:
" + sleepingPlaces);
    }
}
```

У кровати есть достаточно важные параметры – длина, ширина и кол-во спальных мест. Переопределим конструкторы, в параметризированном инициализировав эти дополнительные параметры. Переопределим методы движения и использования, добавим геттеры и сеттеры различных параметров

кровати и добавим новый метод, считающий площадь кровати. Переопределим метод вывода информации так, чтобы там указывалась и информация о ширине, длине кровати и кол-ве спальных мест.

Class SofaBed extends Bed

Этот класс реализован в файле Bed.java, где представлен и родительский класс Bed. Код представлен в листинге 1.8:

Листинг 1.8

```
class SofaBed extends Bed {
    private boolean isFoldable; // может ли вообще раскладываться
    private double foldedWidth; // ширина в разложенном состоянии
    private double unfoldedWidth; // ширина в неразложенном состоянии

    public SofaBed() {
        super();
    }

    public SofaBed(String name, String material, double price,
                   double unfoldenWidth, double length, int sleepingPlaces,
boolean isFoldable, double foldedWidth) {
        super(name, material, price, unfoldenWidth, length, sleepingPlaces);
        this.isFoldable = isFoldable;
        this.unfoldedWidth = unfoldenWidth;
        if (! isFoldable){
            this.foldedWidth = unfoldenWidth; //для нераскладывающихся кроватей
        } else{
            this.foldedWidth = foldedWidth;
        }
    }

    @Override
    public void use() {
        System.out.println("Это диван-кровать - на нем смотрят телевизор");
    }

    public void use(String name) {
        System.out.println("Это диван-кровать - на нем смотрят фильм " + name);
    }

    public void fold() {
        width = foldedWidth;
    }

    public void unfold() {
```

```
        width = unfoldedWidth;
    }

    public boolean isFoldable() {
        return isFoldable;
    }

    public double getFoldedWidth() {
        return foldedWidth;
    }

    public double getUnFoldedWidth() {
        return foldedWidth;
    }

    public void setFoldedWidth(double newWidth) {
        this.foldedWidth = newWidth;
    }

     public void setUnfoldedWidth(double newWidth) {
        this.unfoldedWidth = newWidth;
    }

    @Override
    public void info() {
        super.info();
        System.out.println("Раскладной: " + (isFoldable ? "да" : "нет"));
    }
}
```

Предполагается, что диван-кровать может раскладываться. Тогда у нее есть ширина в разложенном и сложенном состоянии, есть и текущая ширина, с помощью которой мы будем считать площадь кровати, например. Конечно, мы могли бы переопределить метод счета площади, чтобы там использовалось поле под каким-нибудь названием currentWidth, но намного проще оставить везде просто width. Поэтому будем работать в этом классе с двумя параметрами дивана – шириной в сложенном и разложенном видах, для которых пропишем сеттеры и геттеры. Добавим геттер для параметра раскладывается-не раскладывается, также, конечно же, добавим методы складывания и раскладывания, где мы без проверки текущего состояния и возможности разложить диван в целом будем присваивать его ширине

значение одной из ширин. (Для не раскладывающихся диванов в конструкторе ширине в разложенном виде присвоим ширину в неразложенном виде, поэтому она просто не будет меняться). Здесь мы перезагрузим метод использования, сделав возможным его использование с параметром. Переопределим метод вывода информации, добавив туда информацию о раскладываемости дивана, вывод который реализуем через тернарный оператор.

Class BedBed extends Bed

Этот класс тоже реализован в файле Bed.java. Код представлен в листинге 1.9:

Листинг 1.9

```
class BedBed extends Bed {
    public BedBed() {
        super();
    }

    public BedBed(String name, String material, double price,
                  double width, double length, int sleepingPlaces) {
        super(name, material, price, width, length, sleepingPlaces);
    }

    @Override
    public void use() {
        System.out.println("Это кровать-кровать - на ней читают книжку");
    }

    public void use(String name) {
        System.out.println("Это кровать-кровать - на ней читают книжку " + name);
    }

}
```

Здесь просто пропишем конструкторы, вызвав в них родительские конструкторы и переопределим метод использования. Добавим его перегрузку: теперь этот метод может принимать параметр – название читаемой книги.

Public class Table extends Furniture

Этот класс реализован в файле Table.java, код представлен в листинге 1.10:

Листинг 1.10

```
package furniture;

public class Table extends Furniture{
    protected String shape;
    protected int numberOfLegs;

    public Table() {
        super();
    }

    public Table(String name, String material, double price, String shape, int
numberOfLegs) {
        super(name, material, price);
        this.shape = shape;
        this.numberOfLegs = numberOfLegs;
    }

    @Override
    public void move() {
        System.out.println("кхххххх");
    }

    @Override
    public void use() {
        System.out.println("Это стол - за ним сидят");
    }

    public void knock() {
        System.out.println("тук-тук");
    }

    public String getShape() {
        return shape;
    }
    public void setShape(String shape) {
        this.shape = shape;
    }

    public int getNumberOfLegs() {
        return numberOfLegs;
    }
    public void setNumberOfLegs(int numberOfLegs) {
```

```
        this.numberOfLegs = numberOfLegs;
    }

    @Override
    public void info() {
        super.info();
        System.out.println("Форма: " + shape + ", Кол-во ножек: " +
numberOfLegs);
    }
}
```

У стола, как и у стула можно посчитать кол-во ножек (но не всегда явно). Также стол характеризуется формой. Пропишем для этих дополнительных полей геттеры и сеттеры. В параметризованном конструкторе добавим инициализацию этих полей. Добавим вывод этих параметров в переопределенный метод вывода информации. Переопределим метод использования и метод звука перетаскивания. Добавим еще один метод – метод постукивания по столу.

Class HomeTable extends Table

Этот класс написан также в файле, содержащим родительский класс - Table.java. Код представлен в листинге 1.11:

Листинг 1.11

```
class HomeTable extends Table implements FurnitureWithLegs{
    private boolean isExtendable;
    private int extendedGuests; // кол-во вмещаемых гостей в разложенном
состоянии
    private int unextendedGuests; // кол-во вмещаемых гостей в неразложенном
состоянии

    public HomeTable() {
        super();
    }

    public HomeTable(String name, String material, double price,
                     String shape, int numberOfLegs, boolean isExtendable, int
extendedGuests, int unextendedGuests) {
        super(name, material, price, shape, numberOfLegs);
        this.isExtendable = isExtendable;
```

```
        this.extendedGuests = extendedGuests;
        this.unextendedGuests = unextendedGuests;
    }

    @Override
    public void use() {
        System.out.println("Это стол - за ним едят");
    }

    public void extend() {
        if (isExtendable) {
            System.out.println("Теперь за столом помещаются " + extendedGuests  +
" человек");
        } else {
            System.out.println("Этот стол не раскладывается, за столом по-
прежнему помещаются" + unextendedGuests + " человек");
        }
    }

    public boolean isExtendable() {
        return isExtendable;
    }
    public int getExtendedGuests() {
        return extendedGuests;
    }
    public int getUnExtendedGuests() {
        return unextendedGuests;
    }
    public void setExtendedGuests(int newExtendedGuests) {
        this.extendedGuests = newExtendedGuests;
    }
    public void setUnExtendedGuests(int newUnExtendedGuests) {
        this.unextendedGuests = newUnExtendedGuests;
    }

    @Override
    public int getNumberOfLegs() {
        return numberOfLegs;
    }
    @Override
    public void setNumberOfLegs(int numberOfLegs) {
        this.numberOfLegs = numberOfLegs;
    }

    @Override
    public void info() {
        super.info();
        System.out.println("Раскладной: " + (isExtendable ? "да" : "нет"));
        if (isExtendable) {
```

```
            System.out.println("В разложенном состоянии вмещает: " +
extendedGuests+ " человек");
        }
    }
}
```

Домашний стол имеет важное отличие от офисного – он умеет раскладываться. Кроме того, у него выраженное кол-во ножек, поэтому реализуем в нем интерфейс мебели с ножками. Кроме поля раскладываемости добавим поля с кол-вом гостей, вмещаемым в его разложенном и сложенном состояниях. Аналогично всему вышеперечисленному напишем конструкторы этого класса. Переопределим метод использования, напишем сеттеры для кол-ва вмещаемых человек и геттеры для всех новых полей. Переопределим метод интерфейса мебели с ножками. Добавим метод разложения с проверкой на раскладываемость и сообщением о кол-ве вмещаемых за стол людей. Добавим переопределения метода вывода информации – теперь кроме всех параметров любого стола и любого объекта мебели будет указываться, раскладывается стол или нет. В случае если раскладывается, будет указываться, сколько он в себя человек может вместить в разложенном виде.

Class OfficeTable extends Table

Этот класс написан тоже в файле Table.java. Код представлен в листинге 1.12:

Листинг 1.12

```
class OfficeTable extends Table implements OfficeFurniture{
    private static int invCount = 0;

    public OfficeTable() {
        super();
        invCount ++;
    }

    public OfficeTable(String name, String material, double price,
                       String shape, int numberOfLegs) {
        super(name, material, price, shape, numberOfLegs);
        invCount ++;
```

```
    }

    @Override
    public int inventory(){
        return invCount;
    }

    @Override
    public void use() {
        System.out.println("Это офисный стол - за ним по кнопочкам клацают");
    }

    @Override
    public void knock() {
        System.out.println("бам-бам");
    }
}
```

Реализуем в этом классе интерфейс офисной мебели. Добавим статическое поля с количеством. Напишем конструкторы этого класса, в них к кол-ву будем прибавлять 1 при инициализации. Остальное мы можем просто пронаследовать их от Table. Переопределим метод с инструкцией и метод с постукиванием, заменив «тук-тук» кулаком по столу на «бам-бам» головой о стол. Переопределим геттер кол-ва экземпляров офисной мебели.

Public class Main

Этот класс содержится в одноименном файле и нужен для того, чтобы продемонстрировать работу с остальными классами. Сам он среди прочих не является ни родителем, ни ребенком. Код представлен в листинге 1.13:

Листинг 1.13

```
package furniture;

public class Main {
    public static void main(String[] args){
        Chair justChair = new Chair();
        HomeChair homeChair = new HomeChair("Ikea kitchen", "дерево", 2000,
false, 4, false);
        OfficeChair officeChair = new OfficeChair("Компьютерное кресло", "кожа",
6000, true, 1, true, 60);
        Table justTable = new Table();
```

```
        HomeTable homeTable = new HomeTable("Ikea kitchen", "дерево", 12000,
"прямоугольный", 4, true, 8, 4);
        OfficeTable officeTable = new OfficeTable("Компьютерный стол", "дерево",
8000, "уголок", 3);
        Bed justBed = new Bed();
        BedBed regularBed = new BedBed("Детская кровать", "дерево", 8000, 1.1,
1.7, 1);
        SofaBed sofaBed = new SofaBed("Тахта", "ткань", 10000, 0.8, 2.0, 2, true,
1.4);


        Furniture[] allFurniture = {justChair, homeChair, officeChair, justTable,
homeTable, officeTable, justBed, regularBed, sofaBed};

        System.out.println("Всего объектов мебели: " + Furniture.getCount());

        for (Furniture furniture : allFurniture) {
            furniture.info();
            furniture.use();
            furniture.move();
        }

        homeTable.knock();
        officeTable.knock();

        System.out.println("Офисных стульев: " + officeChair.inventory());

        System.out.println("Офисных столов: " + officeTable.inventory());


        regularBed.use("'Гарри Поттер'");
        sofaBed.use("'Властелин колец'");


        System.out.println("Текущая ширина дивана: " + sofaBed.getWidth() + "
м");
        sofaBed.fold();
        System.out.println("После раскладывания: " + sofaBed.getWidth() + " м");
        System.out.println("Площадь после раскладывания: " + sofaBed.area() + "
квадратных метров");
        sofaBed.unfold();
        System.out.println("После складывания: " + sofaBed.getWidth() + " м");
        System.out.println("Площадь после складывания: " + sofaBed.area() + "
квадратных метров");

        System.out.println("Текущая высота стула: " + officeChair.getHeight() + "
м");
        officeChair.changeHeight(-10);
        System.out.println("Отрегулированная высота стула: " +
officeChair.getHeight() + " м");
```

```
        System.out.println("Цена стула из Икеи: " + homeChair.getPrice());
        homeChair.setPrice(2200);
        System.out.println("Новая цена после скидки: " + homeChair.getPrice());
    }
}
```

Создадим объекты всевозможных классов (кроме Furniture). Некоторые пустыми, некоторые с параметрами. Выведем общее кол-во экземпляров, используя геттер счетчика. Для автоматизации тестирования создадим массив объектов Furniture, куда поместим все наши экземпляры. Пройдемся по массиву циклом и для каждого предмета мебели вызовем три базовых метода, которые определены везде: метод движения, метод использования и метод вывода информации. Далее протестируем постукивания по столам, инвентаризацию офисной мебели, перегрузку методов использования у кроватей, раскладывание и складывание дивана с выводом текущей ширины и площади, метод регулировки высоты стула. В конце протестируем работу сеттеров для стула. Результаты выполнения Main представлены ниже на рисунках 22 – 23:

```
PS C:\Users\user\.vscode\mtuci\java\labs\secondlab> java furniture.Main
Всего объектов мебели: 9
Название: null, Материал: null, Цена: 0.0
Мягкое сиденье: нет, Кол-во ножек: 0, Подлокотники: нет
Это стул - на нем сидят
тшшшшш
Название: Ikea kitchen, Материал: дерево, Цена: 2000.0
Мягкое сиденье: нет, Кол-во ножек: 4, Подлокотники: нет
Это домашний стул - на него встают, чтобы поменять лампочки
тшшшшш
Название: Компьютерное кресло, Материал: кожа, Цена: 6000.0
Мягкое сиденье: да, Кол-во ножек: 1, Подлокотники: есть
Текущая высота: 60см
Это офисный стул - на нем устраивают гонки по офису
вжжжжух
Название: null, Материал: null, Цена: 0.0
Форма: null, Кол-во ножек: 0
Это стол - за ним сидят
кхххххх
Название: Ikea kitchen, Материал: дерево, Цена: 12000.0
Форма: прямоугольный, Кол-во ножек: 4
Раскладной: да
В разложенном состоянии вмещает: 8 человек
Это стол - за ним едят
кхххххх
Название: Компьютерный стол, Материал: дерево, Цена: 8000.0
Форма: уголок, Кол-во ножек: 3
Это офисный стол - за ним по кнопочкам клацают
кхххххх
Название: null, Материал: null, Цена: 0.0
Размер: 0.0*0.0м, Спальных мест: 0
Это кровать - на ней спят
скрииииип
Название: Детская кровать, Материал: дерево, Цена: 8000.0
Размер: 1.1*1.7м, Спальных мест: 1
Это кровать-кровать - на ней читают книжку
скрииииип
Название: Тахта, Материал: ткань, Цена: 10000.0
Размер: 0.8*2.0м, Спальных мест: 2
Раскладной: да
```

Рисунок 22. Результат работы программы furniture.Main. ч.1

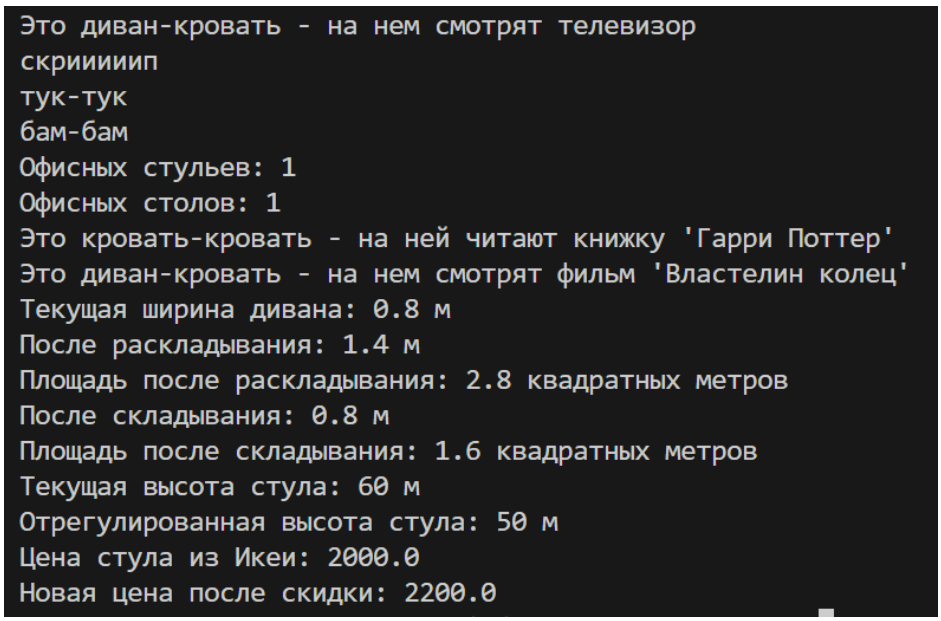

Рисунок 23. Результат работы программы furniture.Main. ч.2

Работа загружена на гитхаб по ссылке: https://github.com/Kateriabova/Riaboava_Kate_BPI2401_IT2.git

Ответы на контрольные вопросы:

1. Что такое абстракция и как она реализуется в языке Java?

Абстракция - это принцип ООП, который позволяет скрыть сложную реализацию и показать только необходимые детали объекта. Реализуется в java с помощью абстрактных классов, методов и интерфейсов.

2. Что такое инкапсуляция и как она реализуется в языке Java?

Инкапсуляция - сокрытие внутренней реализации объекта от внешнего мира и предоставлении доступа к данным только через определенные методы класса. Для реализации применяют различные модификаторы доступа – private, protected, public и геттеры и сеттеры – методы для контролируемого доступа к этим полям.

3. Что такое наследование и как он реализуется в языке Java?

Наследование - это механизм, позволяющий классу-потомку использовать поля и методы класса-родителя. Реализуется с помощью ключевого слова extends.

4. Что такое полиморфизм и как он реализуется в языке Java?

Полиморфизм - это способность объектов с одинаковой спецификацией иметь различную реализацию (полиморфизм как бы позволяет объектам разных типов реагировать по-разному на одни и те же сообщения.) В Java можно это сделать с помощью двух способов: статический (перегрузка методов) и динамический (переопределение методов).

5. Что такое множественное наследование и есть ли оно в Java?

Множественное наследование - возможность класса наследовать от нескольких родителей. В Java нет множественного наследования, но есть множественная реализация интерфейсов вместо этого.

6. Для чего нужно ключевое слово final?

Final запрещает изменение, так: класс нельзя будет наследовать, а метод переопределять, а переменной – менять значение.

7. Какие в Java есть модификаторы доступа?
    - private - доступ только внутри класса.
    - protected - доступ внутри пакета, так же доступ имеют наследники даже вне пакета.
    - public - доступ отовсюду.
    - (по умолчанию) - доступ только внутри пакета.
8. Что такое конструктор? Какие типы конструкторов бывают в Java?

Конструктор - специальный метод для создания объектов. Бывает: по умолчанию, то есть без параметров, параметризованный и копирования (когда он копирует другой объект этого же класса).

9. Для чего нужно ключевое слово this в Java?

This ссылается на текущий объект, это как универсальное имя экземпляра класса, используемое внутри методов и конструкторов. По этой ссылке мы можем обратиться из метода к объекту и изменить/получить значения его полей.

10. Для чего нужно ключевое слово super в Java?

Super ссылается на родительский класс, с помощью него можно вызвать конструктор или метод родителя и получить доступ к полю родителя.

11. Что такое геттеры и сеттеры? Зачем они нужны?

Геттеры (get) - методы для получения значения поля. Сеттеры (set) - методы для установки значения поля. Они нужны для контроля доступа к данным (при том же использовании модификаторов доступа), валидируют данные перед установкой и гарантируют соблюдение инкапсуляции.

12. Что такое переопределение?

Переопределение (override) - изменение реализации метода в классе потомке при соблюдении следующих условий: тот же метод – имя, параметры, возвращаемый тип, аннотация (@Override), переопределяемый метод не private и не final.

13. Что такое перегрузка?

Перегрузка (overload) - создание методов с одинаковым именем, но разными параметрами. Может быть реализована только в пределах одного класса, методы могут иметь разные параметры и может отличаться возвращаемый тип.

Вывод:

В ходе работы я изучила принципы ООП в Java, научилась писать код с использованием классов и их наследования и изучила особенности синтаксиса Java в ООП.