ЛАБОРАТОРНАЯ РАБОТА 4. СТРУКТУРА ПРОГРАММ. НАСЛЕДОВАНИЕ.

Цель лабораторной работы: изучить принципы структурной декомпозиции приложений и наследования классов.

В предыдущих лабораторных работах создавались классы и демонстрировались методы использования возможностей стандартной библиотеки Java SE. Для эффективной разработки и поддержки ООП приложений необходимо принимать дополнительные меры по структурированию кода.

В стандартную библиотеку Java API входят сотни классов. Каждый программист в ходе работы добавляет к ним десятки своих классов. Множество классов растет и становится неуправляемым. Уже давно принято отдельные классы, решающие какую-то одну определенную задачу, объединять в библиотеки классов (объединение по функциональному признаку). Но библиотеки классов, кроме стандартной библиотеки, не являются частью языка.

Разработчики Java включили в язык дополнительную конструкцию – пакеты (packages). Все классы Java распределяются по пакетам. Кроме классов пакеты могут содержать интерфейсы и вложенные подпакеты (subpackages). Образуется древовидная структура пакетов и подпакетов

Эта структура в точности отображается на структуру файловой системы. Все файлы с расширением class (содержащие байт-коды), образующие один пакет, хранятся в одном каталоге файловой системы. Подпакеты образуют подкаталоги этого каталога.

Каждый пакет создает одно пространство имен (namespace). Это означает, что все имена классов, интерфейсов и подпакетов в пакете должны быть уникальны. Имена в разных пакетах могут совпадать, но это будут разные программные единицы. Таким образом, ни один класс, интерфейс или подпакет не может оказаться сразу в двух пакетах. Если надо в одном месте программы использовать два класса с одинаковыми именами из разных

пакетов, то имя класса уточняется именем пакета: .. Такое уточненное имя называется полным именем класса (fully qualified name).
Все эти правила, опять-таки, совпадают с правилами хранения файлов и подкаталогов в каталогах, только в файловых системах для разделения имен каталогов в пути к файлу обычно используется наклонная черта или двоеточие, а не точка.
Еще одно отличие от файловой системы – подпакет не является частью пакета, и классы, находящиеся в нем, не относятся к пакету, а только к подпакету. Поэтому, для того чтобы создать подпакет, не надо предварительно создавать пакет. С другой стороны, включение в программу пакета не означает включение его подпакетов. Пакетами пользуются еще и для того, чтобы добавить к уже имеющимся правам доступа к членам класса private, protected. и public (уже рассматривались в предыдущих лабораторных). В Java существует еще один, «пакетный» уровень доступа. Если член класса не отмечен ни одним из модификаторов private, protected, public, то по умолчанию к нему осуществляется пакетный доступ (default access), т. е. к такому члену может обратиться любой метод любого класса из того же пакета. Пакеты ограничивают и доступ к классу целиком – если класс не помечен модификатором public, то все его члены, даже открытые, public, не будут видны из других пакетов.
Следует обратить внимание на то, что члены с пакетным доступом не видны в подпакетах данного пакета.
Чтобы создать пакет, необходимо в первой строке java-файла с исходным кодом записать строку

```
package имя_пакета;
```

Например:

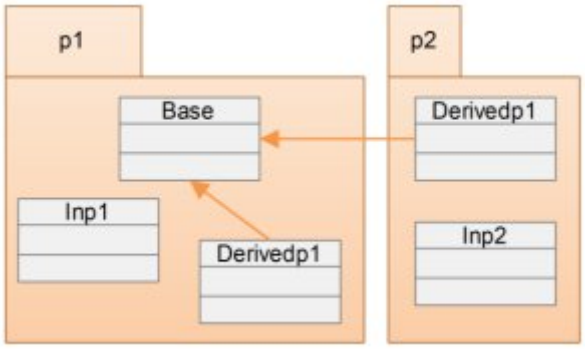
```
package laba_4_ot_Davida_Dmitrievicha_prosto_kapets
```

Имя подпакета уточняется именем пакета. Чтобы создать подпакет с именем, например, subpack, следует в первой строке исходного файла написать:

package имя_пакета. имя_подпакета;
и все классы этого файла и всех файлов с такой же первой строкой попадут в данный подпакет. Можно создать и подпакет подпакета произвольной вложенности
Поскольку строка «package имя_пакета;» только одна и это обязательно первая строка файла, каждый класс попадает только в один пакет или подпакет.
Соглашение «Code Conventions» рекомендует записывать имена пакетов строчными буквами. Тогда они не будут совпадать с именами классов, которые, по соглашению, начинаются с прописной буквы. Кроме того, соглашение советует использовать в качестве имени пакета или подпакета доменное имя своего сайта, записанное в обратном порядке, например:
`com.sun.developer`
Это обеспечит уникальность имени пакета во всем Интернете
Вы можете не указывать пакет при использовании интегрированной среды разработки. Компилятор всегда создает для таких классов безымянный пакет (unnamed package), которому соответствует текущий каталог (current working directory) файловой системы.
Безымянный пакет служит обычно хранилищем небольших пробных или промежуточных классов. Большие проекты лучше хранить в пакетах. Более того, некоторые программные продукты Java вообще не работают с безымянным пакетом. Поэтому в технологии Java рекомендуется все классы помещать в пакеты.
Библиотека классов Java SE 7 API хранится в пакетах java, javax, org. Пакет java содержит только подпакеты applet, awt, beans, dyn, io, lang, math, net, nio, rmi,security, sql, text, util и ни одного класса. Эти пакеты имеют свои подпакеты, например пакет создания графического пользовательского интерфейса и графики java.awt содержит классы, интерфейсы и подпакеты color, datatransfer, dnd, event, font, geom, im, image, print.

Права доступа к членам класса.

В предыдущих лабораторных работах при написании кода спецификаторы доступа указывались как перед определением классов, так и перед полями и методами, принадлежащими классам. Для усвоения принципов распределения классов по пакетам рассмотрим пример:



В файле Base.java описаны три класса: Inp1, Base и класс Derivedp1, расширяющий класс Base. Эти классы размещены в пакете p1.

В файле Inp2.java описаны два класса: Inp2 и класс Derivedp2, расширяющий класс Base. Эти классы находятся в другом пакете p2. Класс Base должен быть помечен при своем описании в пакете p1 модификатором public, иначе из пакета p2 не будет видно ни одного его члена.

В пакете p2, доступ ограничен в большей степени. Из независимого класса можно обратиться только к открытым, public, полям класса другого пакета. Из подкласса можно обратиться еще и к защищенным, protected, полям, но только унаследованным непосредственно, а не через экземпляр суперкласса. Все указанное относится не только к полям, но и к методам. В таблице 1 суммируется информация о доступности полей и методов класса в зависимости от контекста и модификатора доступности.

	Класс	Пакет	Пакет и подклассы	Все классы
private	+			
«package»	+	+		
protected	+	+	*	
public	+	+	+	+

Наследование в Java

Наследование — один из ключевых механизмов ООП, позволяющий создавать новый класс на основе существующего. Новый класс, называемый **производным** (или подклассом), наследует поля и методы родительского класса. Это способствует повторному использованию кода и облегчает его поддержку.

С помощью наследования можно расширить функционал уже имеющихся классов за счет добавления нового функционала или изменения старого.

Например, имеется следующий класс Person, описывающий отдельного человека:

```
class Person {

    String name;
    public String getName(){ return name; }

    public Person(String name){

        this.name=name;
    }

    public void display(){

        System.out.println("Name: " + name);
    }
}
```

И, возможно, впоследствии мы захотим добавить еще один класс, который описывает сотрудника предприятия - класс Employee. Так как этот класс реализует тот же функционал, что и класс Person, поскольку сотрудник - это также и человек, то было бы рационально сделать класс Employee производным (наследником, подклассом) от класса Person, который, в свою очередь, называется базовым классом, родителем или суперклассом:

```
class Employee extends Person{
    public Employee(String name){
        super(name);// если базовый класс определяет конструктор
                    //  то производный класс должен его вызвать
    }
}
```

Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово **extends**, после которого идет имя базового класса. Для класса Employee базовым является Person, и поэтому класс Employee наследует все те же поля и методы, которые есть в классе Person.

Если в базовом классе определены конструкторы, то в конструкторе производного классы необходимо вызвать один из конструкторов базового класса с помощью ключевого слова **super**. Например, класс Person имеет конструктор, который принимает один параметр. Поэтому в классе Employee в конструкторе нужно вызвать конструктор класса Person. То есть вызов super(name) будет представлять вызов конструктора класса Person. При вызове конструктора после слова super в скобках идет перечисление передаваемых аргументов. При этом вызов конструктора базового класса должен идти в самом начале в конструкторе производного класса. Таким образом, установка имени сотрудника делегируется конструктору базового класса.

Причем даже если производный класс никакой другой работы не производит в конструкторе, как в примере выше, все равно необходимо вызвать конструктор базового класса.

Виртуальные методы

По умолчанию, все нестатические методы в Java являются виртуальными — это значит, что их поведение может быть переопределено в подклассе. Виртуальные методы позволяют использовать динамическую диспетчеризацию, которая обеспечивается на этапе выполнения программы.

Переопределение методов (Override)

Метод родительского класса может быть переопределен в подклассе с целью изменить его поведение. Переопределение достигается за счет аннотации @Override перед методом.

```
class Child extends Parent {
    @Override
    public void display() {
        System.out.println("This is a child class.");
    }
}
```

Динамическая диспетчеризация — это механизм, при котором JVM на этапе выполнения решает, какой метод должен быть вызван. Она позволяет вызвать переопределенный метод даже при ссылке на объект родительского класса.

```
Parent obj = new Child();
obj.display(); // Вызывает метод из класса Child.
```

Наследование и возможность переопределения методов открывают нам большие возможности. Прежде всего мы можем передать переменной суперкласса ссылку на объект подкласса:

```
Person sam = new Employee("Sam", "Oracle");
```

Так как Employee наследуется от Person, то объект Employee является в то же время и объектом Person. Грубо говоря, любой работник предприятия одновременно является человеком.

Однако несмотря на то, что переменная представляет объект Person, виртуальная машина видит, что в реальности она указывает на объект Employee. Поэтому при вызове методов у этого объекта будет вызываться та версия метода, которая определена в классе Employee, а не в Person. Например:

```
 public class Program{

    public static void main(String[] args) {

        Person tom = new Person("Tom");
        tom.display();
        Person sam = new Employee("Sam", "Oracle");
        sam.display();
    }
}
class Person {

    String name;

    public String getName() { return name; }

    public Person(String name){

        this.name=name;
    }

    public void display(){

        System.out.printf("Person %s \n", name);
    }
}

class Employee extends Person{

    String company;

    public Employee(String name, String company) {

        super(name);
        this.company = company;
    }
```

```
    @Override
    public void display(){

        System.out.printf("Employee %s works in %s \n",
super.getName(), company);
    }
}
```

Консольный вывод данной программы:

```
Person Tom
Employee Sam works in Oracle
```

При вызове переопределенного метода виртуальная машина динамически находит и вызывает именно ту версию метода, которая определена в подклассе. Данный процесс еще называется **dynamic method lookup** или динамический поиск метода или динамическая диспетчеризация методов.

Запрет наследования

Чтобы запретить наследование класса, используется ключевое слово final. Если класс или метод объявлены как final, то их нельзя наследовать или переопределять соответственно.

```
final class FinalClass {
    // Этот класс не может быть расширен.
}
```

Задания по вариантам.

I. Реализуйте классы с использованием Наследования. У каждого класса должен быть конструктор. Добавьте безопасный доступ к полям класса через сеттеры и геттеры, а также вывод всей информации. Для классов, в котором хранится массив объектов, создайте методы подсчета объекта каждого типа, а также безопасные Add и Pop.

1. Создайте иерархию классов **Animal**, **Dog**, **Cat**, и **Bird**. В классе **Animal** добавьте поля для имени и возраста, метод **makeSound()**. В каждом из подклассов переопределите метод **makeSound()** (для **Dog** — "Гав", для **Cat** — "Мяу", для **Bird** — "Чик-чирик"). Создайте также класс **Zoo**, который будет хранить массив животных и сможет вызвать метод **makeSound()** для каждого животного.

2. Создайте иерархию классов **Person**, **Student**, **Teacher**, и **Principal**. В классе **Person** создайте конструктор с параметрами для имени и возраста. В классе **Student** добавьте поле для номера студенческого билета, в классе **Teacher** — для предмета, а в классе **Principal** — для уровня школы (например, младшая, старшая). Создайте класс **School**, который хранит массив объектов типа **Person** (можно добавлять любого потомка). Реализуйте метод, который будет выводить информацию обо всех людях в школе.

3. Создайте иерархию классов **Shape**, **Circle**, **Square**, и **Triangle**. В классе **Shape** добавьте абстрактный метод **getArea()**, который будет переопределён в каждом подклассе для вычисления площади. Создайте класс **Drawing**, который содержит массив фигур и метод для вычисления общей площади всех фигур.

4. Создайте классы **Vehicle**, **Car**, **Bike**, и **Truck**. В классе **Vehicle** добавьте поле для скорости и метод **move()**, который выводит текущую скорость. В каждом из подклассов создайте метод, который изменяет скорость (например, **accelerate()** для машин и мотоциклов, **loadCargo()** для

грузовиков). Создайте класс **Garage**, который хранит массив транспортных средств и может изменять их скорость.

5. Создайте классы **Parent**, **Child**, и **GrandChild**. В классе **Parent** добавьте метод **display()**, который выводит "Это родитель". В классе **Child** переопределите метод **display()**, чтобы он выводил "Это ребенок". В классе **GrandChild** переопределите метод **display()**, чтобы он выводил "Это внук". Создайте класс **Family**, который хранит массив объектов типа **Parent** и вызывает метод **display()** для каждого элемента массива.

6. Создайте классы **Employee**, **Manager**, **Intern**, и **Director**. В классе **Employee** создайте поле для зарплаты и метод для её отображения. В классе **Manager** добавьте метод **increaseSalary()**, который увеличивает зарплату, в классе **Intern** — метод **study()**, а в классе **Director** — метод **makeDecision()**. Создайте класс **Company**, который хранит сотрудников и может управлять их действиями (например, увеличивать зарплату менеджерам или вызывать действия директора).

7. Создайте классы **Book**, **PrintedBook**, и **EBook**. В классе **Book** добавьте поля для названия и автора. В классе **PrintedBook** добавьте поле для количества страниц, а в классе **EBook** — для размера файла. Создайте класс **Library**, который будет хранить массив книг и выводить информацию о каждой книге.

8. Создайте классы **Appliance**, **WashingMachine**, **Refrigerator**, и **Oven**. В классе **Appliance** добавьте поле для модели и метод **turnOn()**, который выводит "Прибор включён". В каждом из подклассов переопределите метод **turnOn()**, чтобы выводить информацию о конкретном приборе (например, "Стиральная машина включена"). Создайте класс **House**, который хранит массив приборов и может включать их все сразу.

9. Создайте классы **Figure**, **Rectangle**, **Triangle**, и **Circle**. В классе **Figure** добавьте метод для вычисления площади, который будет переопределён в каждом подклассе. В классах **Rectangle** и **Triangle** добавьте дополнительные поля для хранения размеров (сторон, радиуса и т.д.). Создайте класс

Geometry, который хранит несколько фигур и может вычислить общую площадь всех фигур.

10. Создайте классы **Computer**, **Desktop**, **Laptop**, и **Tablet**. В классе **Computer** добавьте поле для мощности процессора и метод **getProcessorPower()**. В каждом подклассе добавьте методы, специфичные для их типов (например, **chargeBattery()** для ноутбуков и планшетов, **setGraphicsCard()** для десктопов). Создайте класс **Office**, который хранит компьютеры и вызывает методы для различных типов компьютеров.

11. Создайте классы **Person**, **Worker**, **Doctor**, и **Engineer**. В классе **Person** добавьте поля для имени и возраста, а также метод для вывода информации. В классе **Worker** добавьте метод **work()**, в классе **Doctor** — метод **diagnose()**, а в классе **Engineer** — метод **design()**. Создайте класс **Hospital**, который хранит объекты разных профессий и вызывает их методы.

12. Создайте классы **BankAccount**, **SavingsAccount**, **CurrentAccount**, и **FixedDepositAccount**. В классе **BankAccount** создайте поля для баланса и методов **deposit()** и **withdraw()**. В каждом подклассе добавьте специфичные для них методы (например, начисление процентов для **SavingsAccount**, ограничение снятий для **FixedDepositAccount**). Создайте класс **Bank**, который управляет счетами клиентов и вызывает соответствующие методы.

13. Создайте классы **Gadget**, **Smartphone**, **Tablet**, и **Smartwatch**. В классе **Gadget** добавьте поля для названия и метода **turnOn()**, который выводит "Гаджет включен". В каждом подклассе добавьте специфичные поля (например, диагональ экрана для планшета, время работы от батареи для часов). Создайте класс **GadgetStore**, который хранит массив гаджетов и позволяет включить каждый из них.

14. Создайте классы **Instrument**, **Guitar**, **Piano**, и **Drums**. В классе **Instrument** добавьте метод **play()**, который выводит "Играю на инструменте". В каждом из подклассов переопределите метод **play()**, чтобы выводить специфичные звуки для каждого инструмента. Создайте класс

Orchestra, который хранит массив инструментов и вызывает метод **play()** для каждого из них.

15. Создайте классы **Furniture**, **Chair**, **Table**, и **Sofa**. В классе **Furniture** добавьте поле для материала и метод **display()**, который выводит информацию о материале. В каждом подклассе добавьте специфичные поля (например, количество ножек для стола, количество мест для дивана). Создайте класс **FurnitureStore**, который хранит несколько предметов мебели и вызывает метод **display()** для каждого из них.

16. Создайте классы **Pet**, **Cat**, **Dog**, и **Fish**. В классе **Pet** добавьте поля для имени и возраста, метод **makeSound()**. В каждом из подклассов переопределите метод **makeSound()** (например, "Мяу" для кошки, "Гав" для собаки). Создайте класс **PetStore**, который хранит массив животных и выводит звуки каждого животного.

II. Реализация класса комплексных чисел.

1. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте методы для сложения двух комплексных чисел.**

2. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте методы для вычитания одного комплексного числа из другого.**

3. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте методы для умножения двух комплексных чисел.**

4. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте методы для деления одного комплексного числа на другое.**

5. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте** $\sqrt{a^2 + b^2}$, **методы для нахождения модуля (модуль числа , где a и b — действительная и мнимая части).**

6. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте метод для нахождения комплексно-сопряжённого числа.**

7. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте методы для возведения комплексного числа в целую степень.**

8. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте методы для преобразования комплексного числа из алгебраической формы в полярную.**

9. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте методы для**

преобразования комплексного числа из полярной формы в алгебраическую.

10. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте метод для сравнения двух комплексных чисел на равенство.**

11. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте метод для вычисления аргумента комплексного числа (аргумент — это угол в полярной форме).**

12. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте метод для нахождения обратного комплексного числа (если число z=a+biz = a + biz=a+bi, обратное.**

$$1/z = \frac{a-bi}{a^2+b^2}$$

13. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте метод для нахождения экспоненты комплексного числа (используйте формулу Эйлера**

$$e^z = e^a(\cos b + i \sin b), \text{ где } z = a + bi).$$

14. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте метод для вычисления натурального логарифма комплексного числа.**

15. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте метод для вычисления синуса комплексного числа (используйте формулу:**

$$\sin(a + bi) = \sin a \cosh b + i \cos a \sinh b).$$

16. **Создайте класс ComplexNumber, содержащий поля для действительной и мнимой частей. Реализуйте метод для вычисления косинуса комплексного числа (используйте формулу:**

$$\cos(a + bi) = \cos a \cosh b - i \sin a \sinh b).$$

III. ВСПОМНИЛ! ГЕОМЕТРИЯ…

Дан класс Point, который моделирует точку в двумерном пространстве. Класс включает в себя следующие конструкторы и публичные методы:

Сигнатура	Описание
public Point()	Создает точку с координатами (0, 0)
public Point(int x, int y)	Создает точку с координатами (x, y)
public void setLocation(int x, int y)	Устанавливает новые координаты точки
public int getX()	Возвращает значение координаты X
public int getY()	Возвращает значение координаты Y
public String toString()	Возвращает строку в виде "(x,y)"
public int distanceFromOrigin()	Возвращает расстояние от начала координат (0, 0) до точки по формуле расстояния Евклида (x1−x2)2+(y1−y2)2(*x*1−*x*2)2+(*y*1−*y*2)2
isInRectangle(top_left, bottom_right)	проверка, находится ли точка внутри прямоугольной области
isInCircle(center, radius)	проверка, находится ли точка внутри окружности с заданным центром и радиусом.

Создайте класс Point3D, который расширяет класс Point через наследование. Он должен вести себя как Point, за исключением того что это должна быть точка в трехмерном пространстве, которая хранит значение координаты Z. Вы должны предоставить те же методы, что и суперкласс, а также реализовать дополнительное поведение

public Point3D()	Создает точку с координатами (0, 0, 0)
public Point3D(int x, int y, int z)	Создает точку с координатами (x, y, z)
public void setLocation(int x, int y, int z)	Устанавливает новые координаты
public int getZ()	Возвращает координату Z

Класс Point3D() должен переопределить требуемые методы, чтобы они работали корректно с учетом третьей координаты. Также класс Point3D должен вести себя иначе в следующих ситуациях:

- при вызове метода setLocation(int x, int y), координата z должна быть выставлена в 0;
- при вызове метода toString(), строка должна выводить три координаты, а не две;
- метод distanceFromOrigin() должны учитывать координату z и возвращать расстояние по формуле (x1−x2)2+(y1−y2)2+(z1−z2)2($x1-x2)2+(y1-y2)2+(z1-z2)2$.
- isInBox(corner1, corner2) — проверка, находится ли точка внутри трехмерного прямоугольного параллелепипеда, задаваемого двумя угловыми точками.
- isInSphere(center, radius) — проверка, находится ли точка внутри сферы с заданным центром и радиусом.

IV. 3D > 2D

1. **Создайте класс Point, который будет содержать координаты X и Y. Реализуйте методы для вычисления расстояния между двумя точками на плоскости. Затем расширьте этот класс до Point3D и добавьте координату Z. Реализуйте метод для вычисления расстояния между двумя точками в трехмерном пространстве.**

2. **Создайте классы Point и Point3D, как в предыдущем задании. Реализуйте методы для перемещения точки в двумерном и трехмерном пространствах, добавив методы moveBy(dx, dy) и moveBy(dx, dy, dz) соответственно.**

3. **Создайте класс Point с координатами X и Y. Реализуйте метод для нахождения угла между двумя точками относительно начала координат (используйте тангенс угла). Затем расширьте этот класс до Point3D, где угол будет вычисляться для трёхмерного пространства.**

4. **Создайте класс Point, содержащий координаты X и Y, и метод для вычисления длины вектора от начала координат до точки. В классе Point3D реализуйте метод для вычисления длины вектора в трехмерном пространстве.**

5. **Создайте классы Point и Point3D. Реализуйте метод для нахождения середины отрезка, заданного двумя точками, как в двумерном, так и в трехмерном пространствах.**

6. **Создайте класс Point и его наследник Point3D. Реализуйте метод для нахождения скалярного произведения двух векторов в двумерном и трехмерном пространствах.**

7. **Создайте классы Point и Point3D. Реализуйте метод для вычисления площади треугольника, заданного тремя точками в двумерном пространстве, и метод для нахождения объема тетраэдра в трехмерном пространстве.**

8. **Создайте классы Point и Point3D. В классе Point реализуйте метод для вращения точки на плоскости вокруг начала координат на заданный**

угол. В классе Point3D добавьте метод для вращения точки вокруг осей X, Y и Z на заданные углы.

9. **Создайте класс Point и его наследник Point3D. В классе Point реализуйте метод для масштабирования координат точки относительно начала координат. В классе Point3D реализуйте метод для масштабирования точки в трехмерном пространстве.**

10. **Создайте классы Point и Point3D. В классе Point реализуйте метод для вычисления расстояния от точки до прямой, заданной уравнением. В классе Point3D реализуйте метод для вычисления расстояния от точки до плоскости, заданной уравнением.**

11. **Создайте классы Point и Point3D. Реализуйте метод для нахождения вектора нормали в двумерном пространстве между двумя точками. В классе Point3D реализуйте метод для вычисления векторного произведения двух векторов в трехмерном пространстве.**

12. **Создайте классы Point и Point3D. Реализуйте метод для зеркального отражения точки относительно прямой (в двумерном пространстве) и относительно плоскости (в трехмерном пространстве).**

13. **Создайте классы Point и Point3D. В классе Point реализуйте метод для проверки, лежат ли три точки на одной прямой. В классе Point3D реализуйте метод для проверки, лежат ли четыре точки на одной плоскости.**

14. **Создайте классы Point и Point3D. Реализуйте метод для нахождения пересечения двух отрезков, заданных двумя точками, в двумерном пространстве. В классе Point3D реализуйте метод для нахождения пересечения двух линий в трехмерном пространстве.**

15. **Создайте классы Point и Point3D. Реализуйте метод для поворота точки относительно заданной оси в двумерном пространстве. В классе Point3D добавьте возможность поворота точки вокруг осей X, Y или Z на заданный угол.**

16. **Создайте классы Point и Point3D. Реализуйте метод для преобразования координат точки из декартовой системы координат в полярную для двумерного пространства и из декартовой системы координат в сферическую для трехмерного пространства.**