Конструктор класса

Что еще можно делать с ним

Содержание

1. Вызов конструктора из конструктора

Если помните, то конструктор это метод, который используется для порождения обьекта класса. И если это метод, то его можно вызвать из другого метода, верно? Ок. Давайте это проверим. Попробуем придумать ситуацию, когда нужно вызывать метод из метода.

В предыдущей лекции мы рассмотрели класс треугольник. Ему мы передавали 3 стороны. И кто-то может сказать, но а что если нам не даны длины сторон, а только например точки на плоскости? Что тогда? Тогда надо высчитать расстояния между этими точками и потом уже отдать в конструктор эти числа. И здесь возникнет вопрос, а где писать этот код? В каком классе, в каком методе? Было бы классно, если бы треугольник получал аргументом 3 точки и сам бы все это внутри рассчитывал.

Ок, давайте начнем с простого. Точка. Это такой класс, у которого 2 числа, так? Икс и игрек.

Давайте начнем с того (запомните, если вам непонятно как решать задачу, начните с простого и двигайтесь дальше), что попробуем написать класс для точки. Ведь если так не делать, то в конструктор треугольника нужно будет передать не 3 аргумента как со сторонами, а все 6.

И чтобы мы так же передавали 3 аргумента нужно упростить и скомпоновать данные. Помните как мы сделали с задачей чатбота? Создали класс который хранил как команду так и результат. Давайте точно так же напишем класс для точки на плоскости и для простоты предположим что у нас сетка целых чисел.

```
public class Point {

    private final int x;
    private final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

}
```

Хорошо, класс для точки у нас уже есть. Что дальше? Дальше нам нужно рассчитать расстояние между 2 точками. И давайте подумаем, где нам написать этот метод.

Этот метод должен найти расстояние между 2 точками. И здесь возникает такой вопрос: написать какой-нибудь статик метод, который получает аргументом 2 точки и выдает расстояние или же есть более правильный способ? Самое главное в программировании это правильный дизайн классов. Т.е. понимание что есть ваш класс и что он может и должен делать. Если я хочу найти расстояние до точки от моей точки, то было бы логично подумать над тем, что я буду использовать данные одной точки и другой. И мы говорили ранее, если наш метод может использовать значения полей класса, то это должен быть нестатик метод. Так? Значит по крайней мере 1 из точек нашего метода будет той, из которой вызывается метод. Т.е. нам не нужен метод который принимает аргументом 2 точки. Достаточно написать метод у точки, который принимает аргументом другую точку. Давайте по крайней мере напишем обьявление этого метода.

```
public class Point {

    private final int x;
    private final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getDistanceTo(Point other) {
    }
}
```

Метод должен быть public чтобы его можно было вызвать из другого места и не статик, потому что мы должны написать логику расчета расстояния используя поля класса. Метод должен вернуть число, а значит не void, но int и принимает аргументом другую точку.

Теперь, как нам найти расстояние между 2 точками? Неудивительно, но по теореме Пифагора! Где стороны треугольника это разница координат. И так как нам нужно будет возвести эту разницу в квадрат, то нам все равно если разница координат будет отрицательной. Пишем!

```
public int getDistanceTo(Point other) {
    return (x - other.x) * (x - other.x) + (y - other.y) * (y - other.y);
}
```

Что у нас здесь – икс минус икс другой точки умноженное на себя плюс игрек нашей точки (можно было бы написать this.y, но так как у нас нет конфликта с именами аргумента и поля, то можно не писать this а сразу y) минус игрек другой точки. Но мы написали неверно, ведь в итоге это отдаст не расстояние между точками, а его квадрат. Хм…. Как же нам вычленить корень квадратный? Написать самому этот метод? Но как? Это слишком сложно. Но это слишком часто используемая штука и скорей всего кто-то уже написал такой метод. Да, так и есть. У нас есть доступ к математическим методам через класс Math. Давайте посмотрим.

```
public int getDistanceTo(Point other) {
    return (int) Math.sqrt((x - other.x) * (x - other.x) + (y - other.y) * (y - other.y));
}
```

Сам метод квадратного корня Math.sqrt отдает double. Но мы упростим нашу жизнь кастанув результат к int. Хорошо. Мы написали метод определения расстояния. Что дальше? А дальше если помните мы хотели, чтобы наш треугольник получал не только 3 стороны, а еще и три точки и сам рассчитывал себе стороны. Как же это сделать? Нужно написать конструктор, который примет 3 точки. Давайте попробуем.

```
public class Triangle {

    private final int sideA;
    private final int sideB;
    private final int sideC;

    public Triangle(Point a, Point b, Point c) {

    }
```

И теперь мы видим, что поля нашего класса выдают ошибку. Наводим курсор а там:

Variable might not been initialized. т.е. переменная может быть непроинициализирована.

Действительно, если мы обьявили их private final что значит они видны изнутри этого класса и должны быть проинициализированы или же сразу или в контрукторе (т.е. тоже сразу), а наш новый конструктор не инциализирует эти стороны. Ок. Значит сейчас нужно их проинициализировать. Давайте тогда сначала найдем расстояния между точками которые пришли в аргумент конструктора.

```
public Triangle(Point a, Point b, Point c) {
    int sideA = a.getDistanceTo(b);
    int sideB = a.getDistanceTo(c);
    int sideC = b.getDistanceTo(c);

}

public Triangle(int sideA, int sideB, int sideC) {
    this.sideA = sideA;
    this.sideB = sideB;
    this.sideC = sideC;
}
```

Находим стороны посредством точек а б и ц. Но чтобы породить обьект класса треугольник нам нужно проинициализировать его поля. Мы можем это сделать напрямую. Так?

Теперь поля класса не выдают ошибку и посмотрите на линии 20 21 22 и линии 14 15 16. Они же одинаковые, не? И вспомним каково истинное предназначение методов – упрощать жизнь и не писать дублирующийся код. Было бы классно как и с методом вызвать

конструктор. Но как его вызвать правильно? Оказывается так же как и в случае с доступом к полям класса через ключевое слово this. Давайте посмотрим.

```
4   public class Triangle {
5
6       private final int sideA;
7       private final int sideB;
8       private final int sideC;
9
10  @   public Triangle(Point a, Point b, Point c) {
11          int sideA = a.getDistanceTo(b);
12          int sideB = a.getDistanceTo(c);
13          int sideC = b.getDistanceTo(c);
14          this.sideA = sideA;
15          this.sideB = sideB;
16          this.sideC = sideC;
17      }
18
19      public Triangle(int sideA, int sideB, int sideC) {
20          this.sideA = sideA;
21          this.sideB = sideB;
22          this.sideC = sideC;
23      }
24
```

```
private final int sideA;
private final int sideB;
private final int sideC;

public Triangle(Point a, Point b, Point c) {
    int sideA = a.getDistanceTo(b);
    int sideB = a.getDistanceTo(c);
    int sideC = b.getDistanceTo(c);
    Triangle(sideA, sideB, sideC);
}
```

Здесь вы можете видеть, что нельзя вызвать конструктор просто как мы его обьявляем в другом классе.

Заменим на слово this и….

```
    private final int sideA;
    private final int sideB;
    private final int sideC;

    public Triangle(Point a, Point b, Point c) {
        int sideA = a.getDistanceTo(b);
        int sideB = a.getDistanceTo(c);
        int sideC = b.getDistanceTo(c);
        this(sideA, sideB, sideC);
    }
```

Все равно ошибка. Да что же там такое? Call to this must be the first statement in constructor body.Т.е. вызов другого конструктора должен быть первым. Значит нам нужно перенести его наверх. Пробуем

```
4   public class Triangle {
5
6       private final int sideA;
7       private final int sideB;
8       private final int sideC;
9
10  @   public Triangle(Point a, Point b, Point c) {
11          this(a.getDistanceTo(b), a.getDistanceTo(c), b.getDistanceTo(c));
12      }
13
14      public Triangle(int sideA, int sideB, int sideC) {
15          this.sideA = sideA;
16          this.sideB = sideB;
17          this.sideC = sideC;
18      }
19
20  @   public static boolean areValidArguments(int sideA, int sideB, int sideC) {
21          return sideA + sideB > sideC &&
22                  sideA + sideC > sideB &&
23                  sideB + sideC > sideA;
24      }
```

Вот теперь у нас все правильно. Но осталось одно но… Валидация. Если помните мы сделали паблик статик методом. А как же быть здесь? Сначала найти стороны и после проверить методом и только потом вызывать конструктор? Опять 25. Лишняя работа. Значит нам опять нужно вернуть валидацию в конструктор.

Вот теперь все в порядке. И нам бы хотелось проверить наш код. Давайте создадим треугольник с координатами для этого, а еще проверим что он прямоугольный.

```java
public class Triangle {

    private final int sideA;
    private final int sideB;
    private final int sideC;

    public Triangle(Point a, Point b, Point c) {
        this(a.getDistanceTo(b), a.getDistanceTo(c), b.getDistanceTo(c));
    }

    public Triangle(int sideA, int sideB, int sideC) {
        if (sideA > 0 && sideB > 0 && sideC > 0 &&
                sideA + sideB > sideC &&
                sideA + sideC > sideB &&
                sideB + sideC > sideA) {
            this.sideA = sideA;
            this.sideB = sideB;
            this.sideC = sideC;
        } else
            throw new IllegalArgumentException("invalid arguments " + sideA + ", " + sideB + ", " + sideC);
    }

    public boolean isRightTriangle() {
        return this.sideA * this.sideA + this.sideB * this.sideB == this.sideC * this.sideC ||
                this.sideA * this.sideA + this.sideC * this.sideC == this.sideB * this.sideB ||
                this.sideC * this.sideC + this.sideB * this.sideB == this.sideA * this.sideA;
    }
}
```

```java
    public static void main(String[] args) {
        Triangle triangle = new Triangle(
                new Point( x: 0,  y: 3),
                new Point( x: 4,  y: 0),
                new Point( x: 0,  y: 0)
        );
        System.out.println(triangle.isRightTriangle());
    }
}
```

```
Main › main()
Run: Main ×
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
true

Process finished with exit code 0
```

Я специально выбрал точки так, чтобы был треугольник со сторонами 3 4 и 5. Вы можете проверить все остальные кейсы.

2. Видимость конструктора и методов

И давайте подумаем вот над чем. Предположим в какой-то момент времени вы решили запретить создание треугольника по 3 сторонам. Что вам делать? Удалять код? Но там же валидация еще написана, тогда придется перемещать код в другой конструктор и слишком много мороки. Мы говорили что конструктор как и метод, а значит у него тоже должны быть области видимости. Но если для метода мы говорили про статик или нестатик, то для конструктора мы будем менять слово public, которое значит – видно отовсюду. Если вы заметили, наши поля были помечены private, что означало что они видны лишь изнутри этого

же класса. Можно логически предположить, что конструктор тоже можно обьявить private. Давайте попробуем.

```
private Triangle(int sideA, int sideB, int sideC) {
    if (sideA > 0 && sideB > 0 && sideC > 0 &&
            sideA + sideB > sideC &&
            sideA + sideC > sideB &&
            sideB + sideC > sideA) {
        this.sideA = sideA;
        this.sideB = sideB;
        this.sideC = sideC;
    } else
        throw new IllegalArgumentException("invalid arguments
}
```

И для проверки вызовем его в мейне.

```
public static void main(String[] args) {
    Triangle triangle = new Triangle( a: 3, b: 4, c: 5);
    System.out.println(triangle.isRightTriangle());
}
```

Как видим, наш конструктор ожидает 3 точки, а не 3 числа. Потому что извне класса конструктор для 3 чисел не виден и недоступен. И давайте напоследок посмотрим на метод проверки прямоугольный треугольник или нет. А что если я не хочу давать его вызывать снаружи, а хочу чтобы он тоже вызывался только внутри класса? Я могу сделать его private?

Конечно же да! Но зачем, спросите меня вы. Давайте напишем метод, который охарактеризует наш треугольник. Типа:

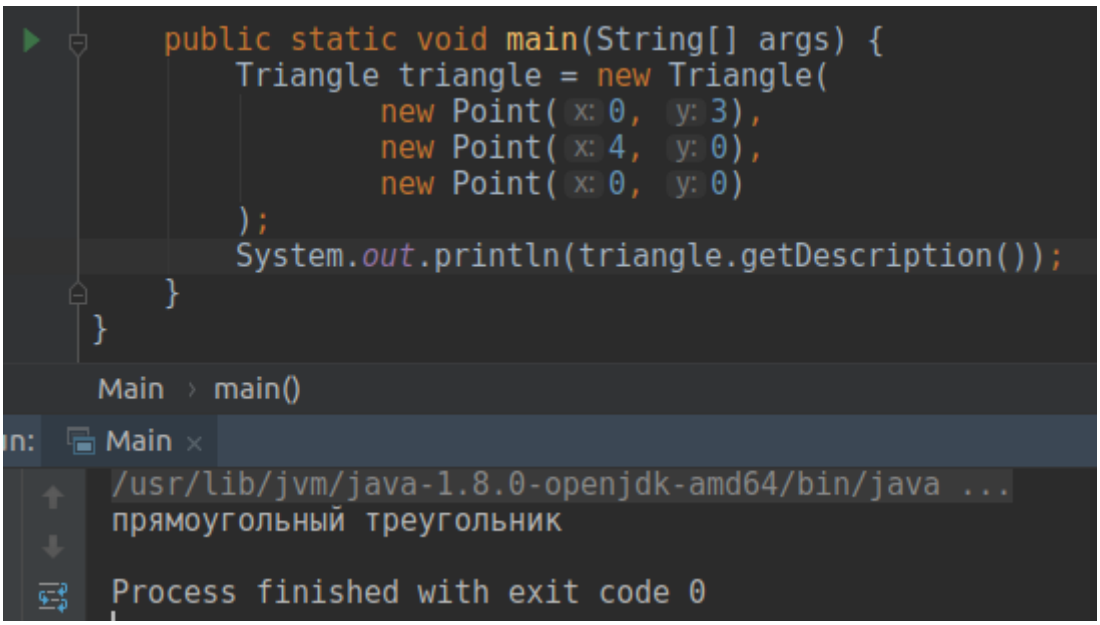
```
public String getDescription() {
    return isRightTriangle()
            ? "прямоугольный треугольник"
            : "обычный треугольник со сторонами " + sideA + " " + sideB + " " + sideC;
}

private boolean isRightTriangle() {
    return this.sideA * this.sideA + this.sideB * this.sideB == this.sideC * this.sideC ||
            this.sideA * this.sideA + this.sideC * this.sideC == this.sideB * this.sideB ||
            this.sideC * this.sideC + this.sideB * this.sideB == this.sideA * this.sideA;
}
```

И вызовем в мейне этот метод.

Давайте подытожим с модификаторами доступа на данный момент
private видно из того класса в котором написано

public – видно везде и доступно отовсюду.



Чтобы закрепить знания напишите класс Прямоугольник у которого тоже будет возможность породить обьект с помощью точек. И еще пусть будет private метод на определение квадрат это или нет и public метод для описания обьекта. Можете добавить еще private метод для расчета площади и использовать его в методе описания обьекта.