Классы и их функции

Исключения и видимость методов

Содержание

1. Ранняя валидация аргументов конструктора. Исключения

В предыдущей лекции мы рассмотрели как писать свои классы. И сегодня мы продолжим эту тему. Помните задачу про треугольник и определение его вида из 11-ой лекции? Сегодня мы напишем класс треугольника, который принимает аргументом 3 стороны.

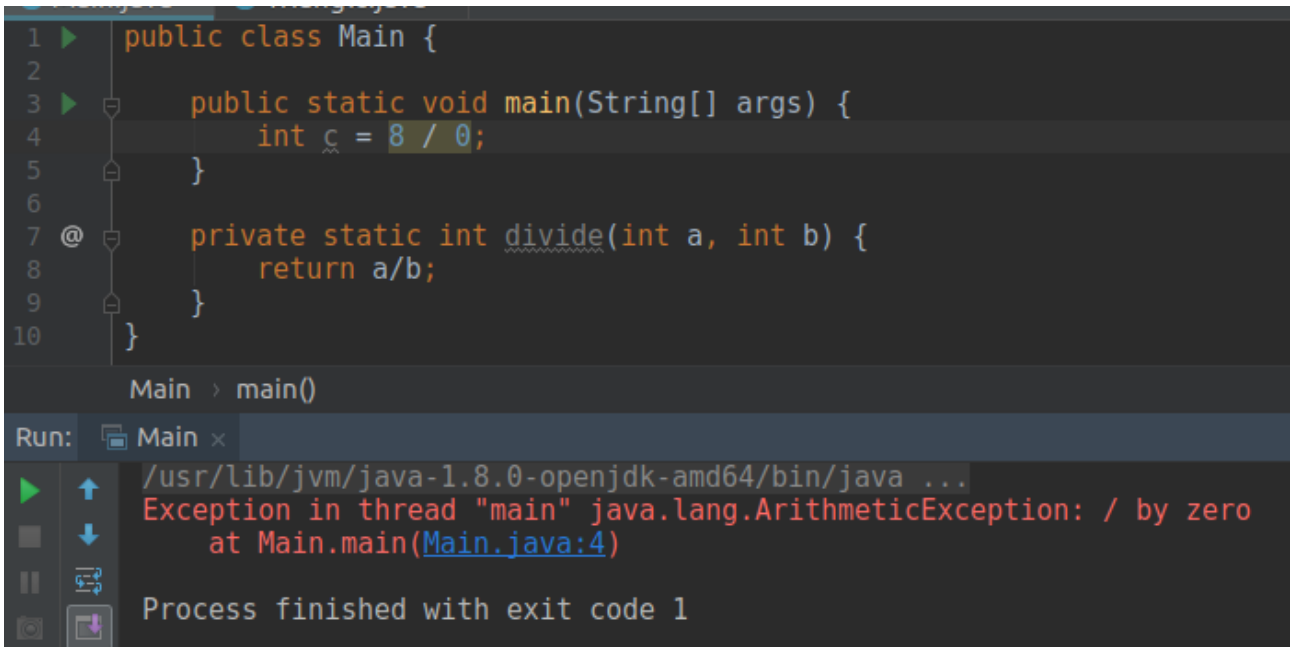
```
*/
public class Triangle {

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

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

Но вам не кажется, что создавать треугольник с любыми тремя числами неправильно? Предположим вы передадите нули или отрицательные числа. А еще по правилу треугольника сумма любых двух сторон должна быть больше чем третья. И где лучше всего сделать эту проверку? Если мы помним, что контруктор это метод создания обьекта класса и он вызывается при инициализации, то это самое правильное место для проверки входных аргументов. Ок, предположим мы написали нашу проверку, но что мы должны сделать в случае неудовлетворения условий?

Для этого давайте на секунду вспомним первые наши задачи, где надо было написать метод деления 2 чисел. Помните мы потом написали условие что делить на ноль нельзя? А что будет если мы просто никаких проверок не напишем. Давайте посмотрим



Во-первых посмотрите на последнее сообщение в консоли – Process finished with exit code 1. Ранее у нас был код 0 что значило без ошибок, штатно. Теперь же исполнение программы завершилось кодом 1. И мы видим сообщение Exception in thread “main” java.lang.ArithmeticException: / by zero и указывается линия кода где произошла ошибка.

Исключения. Они созданы для обработки тех ситуаций, которые не предусмотрены. Т.е. если что-то идет не так как хотелось бы – мы получаем ошибку. В джава есть некий класс ошибок Exception и у него есть множество подвидов. В данном случае мы получили Арифметическое исключение. В последующих лекциях мы остановимся подробнее на исключениях. А в данной лекции вам нужно просто знать, что они есть и они разных видов.

Теперь, мы поняли одну вещь – если нештатная ситуация, то выполнение кода заканчивается выбросом исключения. И все эти нештатные ситуации уже описаны в недрах джава классов.

И здесь возникает логический вопрос. Ок. А что если у меня тоже возможно возникновение нештатной ситуации и я бы хотел прервать выполнение кода? Ведь согласитесь, зачем нам обьект треугольник со сторонами 1, 2 и 3? Да, эти числа больше нуля, но они никак не могут удовлетворить требованию существования треугольника на плоскости. Значит мы должны прервать выполнение такого кода. Было бы классно, если бы у нас была возможность самим бросать исключения. И такое возможно в Java. Давайте посмотрим!

Мы написали наше условие существования треугольника и если оно не удовлетворяется, то мы завершаем выполнение программы исключением. Для этого есть ключевое слово throw после чего идет создание обьекта исключения и мы можем передать поясняющую причину.

Чтобы проверить наш код давайте создадим треугольник в мейн методе мейн класса

(И да, я забыл написать условия чтобы числа были больше нуля, добавьте сами
if (sideA > 0 && sideB > 0 && sideC > 0 && sideA+sideB > sideC ….

```java
public class Triangle {

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

    public Triangle(int sideA, int sideB, int sideC) {
        if (sideA + sideB > sideC &&
                sideA + sideC > sideB &&
                sideB + sideC > sideA) {
            this.sideA = sideA;
            this.sideB = sideB;
            this.sideC = sideC;
        } else {
            throw new IllegalArgumentException(
                    "wrong arguments to create triangle with sides "
                            + sideA + " " + sideB + " " + sideC);
        }
    }
}
```

```java
public class Main {

    public static void main(String[] args) {
        Triangle triangle = new Triangle( sideA: 1,  sideB: 2, sideC: 3);
    }
}
```

```
Main
Run:  Main
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Exception in thread "main" java.lang.IllegalArgumentException: wrong arguments to create triangle with sides 1 2 3
    at Triangle.<init>(Triangle.java:16)
    at Main.main(Main.java:4)

Process finished with exit code 1
```

Мы бы могли поступить иначе, скажем более мягко. Не бросая исключение, а например просто написав условие проверки до создания треугольника. И здесь мы подробно остановимся на методах класса.

2. Функции класса, статические и обычные. Разница

Итак, нам нужно проверить входные аргументы не в конструкторе и при этом не создавая обьект класса. Если вы помните, то наш метод мейн статический, и в нем можно вызывать другие методы которые тоже обьявлены static. В принципе неважно в каком классе обьявлен статик метод, ведь его можно вызвать как если бы он находился в вашем мейн классе. Давайте посмотрим наглядно о чем речь.

Я написал метод валидации аргументов треугольника просто в мейн классе после мейн метода и могу его вызвать перед тем как создать треугольник. Если числа валидные, то можно после проверки спокойно порождать обьект класса Треугольник. И здесь встает вопрос: если метод проверяет валидность данных для создания треугольника и может использоваться только для этого, то почему мы его поместили в мейн классе, который не имеет к нему отношения? т.е. в таком случае нам надо каждый раз писать этот метод и складировать после мейна. Неудобно, согласитесь? Было бы более логично, если бы этот

метод лежал в классе треугольника. Давайте просто выделим его и перенесем в класс Треугольника. Можете вырезать код и вставить в другом классе. Это же просто текст. Ctrl+X, Ctrl+V.

```
    public static void main(String[] args) {
        boolean valid = areValidArguments( sideA: 1,  sideB: 2,  sideC: 3);
        if (valid) {
            Triangle triangle = new Triangle( sideA: 1,  sideB: 2,  sideC: 3);
        }
    }

    private static boolean areValidArguments(int sideA, int sideB, int sideC) {
        return sideA + sideB > sideC &&
                sideA + sideC > sideB &&
                sideB + sideC > sideA;
    }
}
```

```
 */
public class Triangle {

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

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

    private static boolean areValidArguments(int sideA, int sideB, int sideC) {
        return sideA + sideB > sideC &&
                sideA + sideC > sideB &&
                sideB + sideC > sideA;
    }
}
```

Давайте посмотрим что осталось в мейн классе.

```
    public static void main(String[] args) {
        boolean valid = areValidArguments(1, 2, 3);
        if (valid) {
            Triangle triangle = new Triangle( sideA: 1,  sideB: 2,  sideC: 3);
        }
    }

}
```

Видим что среда разработки (в дальнейшем будем говорить просто идеа, у нас же Intellij Idea) выделила красным метод. Давайте поставим курсор на него и посмотрим что не так.

Cannot resolve method ‘areValidArguments(int, int ,int)’.

Не могу найти метод. Давайте просто поставим курсор на нем и нажмем Alt+Enter.

И у нас есть несколько вариантов действий

Create method ‘‘areValidArguments(int, int, int).’ in Main создать метод в мейн классе

Qualify static call ‘Triangle.areValidArguments(int,int,int)’. Вызвать статичексий метод из класса треугольник.

Давайте выберем это действие

```
public static void main(String[] args) {
    boolean valid = Triangle.areValidArguments( sideA: 1,  sideB: 2,  sideC: 3);
    if (valid) {
        Triangle triangle = new Triangle( sideA: 1,  sideB: 2,  sideC: 3);
    }
}
```

Мы видим что метод вызвался у класса треугольника. Мы же обращаемся к методам через точку. Вспомним тот же System.out.println Класс System, в нем поле out и у него метод уже.

И что же не так? Смотрим

‘areValidArguments(int, int,int)’ has private access in class Triangle

У метода приватный доступ. Что бы это значило? По крайней мере то, что его не вызвать из другого класса. Ну по сути это обьяснение и есть суть. Если у вашего метода private видимость, то он доступен только в рамках этого же класса.

Мы знаем что делать. Ставим курсор туда, где подчеркнуто красным и жмем Alt+Enter.

Make ‘Triangle.areValidArguments’ public

Выберем его, а те 2 варианта обсудим позже.

```
*/
public class Triangle {

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

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

    public static boolean areValidArguments(int sideA, int sideB, int sideC) {
        return sideA + sideB > sideC &&
                sideA + sideC > sideB &&
                sideB + sideC > sideA;
    }
}
```

Как видите private сменился на public. И в мейн классе тоже все заработало.

Давайте подытожим на том что у нас есть. В классах может быть статик метод, который можно вызывать без создания обьекта класса в других классах если у него видимость метода public. Хорошо.

Помните мы еще проверяли что наш треугольник прямоугольный, используя в качестве проверки что один из углов 90 градусов. Но сейчас у нас треугольник по 3 сторонам определен. И нам для проверки нужно написать метод по теореме Пифагора.

И здесь давайте остановимся на следующем. Если нам нужно проверить, могут ли 3 числа образовать треугольник то мы пишем статик метод, чтобы не порождать обьект треугольника который не может существовать. А вот если мы уже проверили что такие числа подходят для существования треугольника, то мы можем создать обьект треугольник и уже после этого спокойно сделать проверку того, является он прямоугольным или нет.

Теперь, нам не нужно писать метод который принимает 3 числа. У нас уже есть треугольник с валидными данными по сторонам. Значит мы можем использовать их. Пишем метод! Давайте по-старинке напишем статик.

```
public class Triangle {

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

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

    public static 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;
    }

    public static boolean areValidArguments(int sideA, int sideB, int sideC) {
        return sideA + sideB > sideC &&
                sideA + sideC > sideB &&
                sideB + sideC > sideA;
    }


}
```

И закономерно получаем ошибки. Ведь стороны треугольника доступны лишь после того, как мы их проинициализировали в конструкторе. Т.е. они недоступны из статик методов. Здесь все просто как и всегда. Ставим курсор на this.sideA внутри метода и жмем Alt+Enter и идеа предложит нам сделать метод не статичным. Make isRightTriangle not static. Жмем и вуаля.

Здесь важно еще раз понять, что один метод мы вызываем перед тем как создать треугольник, а второй метод вызываем тогда, когда треугольник точно создан. Давайте глянем в мейн класс и вызовем метод у обьекта.

```java
public class Triangle {

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

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

    public static boolean areValidArguments(int sideA, int sideB, int sideC) {
        return sideA + sideB > sideC &&
                sideA + sideC > sideB &&
                sideB + sideC > sideA;
    }


}
```

```java
public class Main {

    public static void main(String[] args) {
        boolean valid = Triangle.areValidArguments( sideA: 3,  sideB: 4,  sideC: 5);
        if (valid) {
            Triangle triangle = new Triangle( sideA: 3,  sideB: 4,  sideC: 5);
            System.out.println(triangle.isRightTriangle());
        }
    }
}
```

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

Process finished with exit code 0
```

Сравните просто линии кода 4 и 7. Метод валидные аргументы или нет статик и потому вызывется у класса Triangle напрямую, а метод прямоугольный ли треугольник вызывается у обьекта треугольник. Если вы сейчас поймете эту разницу то вам после будет проще.

Еще раз. Если вашему методу нужно задействовать поля класса, то метод без статик. Если же метод работает с данными которые не являются полями класса и не нужен обьект класса, то пишем статик метод. По-хорошему лучше избегать статик методов. Тот же метод валидации аргументов мог бы быть простым методом класса ВалидаторТреугольников.

И как задание попробуйте написать класс Прямоугольника который получает 4 стороны. Добавочный метод является ли он квадратом.