Еще немного логики

Самая лучшая штука в Java – тернарный оператор

Содержание

1. Логическое или

Для начала рассмотрим следующую логику – я бы хотел выпивать только если на календаре пятница и у меня есть деньги.

```
public class Main {

    public static void main(String[] args) {
        print(drink( isFriday: true,  haveMoney: true));
        print(drink( isFriday: true,  haveMoney: false));
        print(drink( isFriday: false,  haveMoney: true));
        print(drink( isFriday: false,  haveMoney: false));
    }

    private static boolean drink(boolean isFriday, boolean haveMoney) {
        return isFriday && haveMoney;
    }

    private static void print(boolean result) {
        System.out.println(result);
    }
}
```

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

Process finished with exit code 0
```

Каковы все возможные исходы?

Сегодня пяница и деньги есть

сегодня пятница, но денег нет

сегодня не пятница, хотя деньги есть

сегодня не пятница и денег тоже нет

Как мы можем понять из этих 4 вариантов я буду пить только если совпадают оба условия.

И в этом особенность логического И &&. Мы будем иметь true тогда и только тогда, когда оба boolean будут true.

И здесь можно задать резонный вопрос: а что если мне достаточно одного условия? Я хочу пить пиво если пятница вне зависимости от того есть у меня много денег или нет, а также я хочу пить пиво если у меня есть деньги и неважно какой день недели. т.е. так

Сегодня пятница – иду пить пиво

Сегодня есть деньги – иду пить пиво

Сегодня пятница и деньги тоже есть – сам Бог велел

На календаре не пятница и ветер в кармане – ну тогда уж не буду пить, ладно, иногда нужно просыхать

И мы можем написать этот код таким образом

```java
public class Main {

    public static void main(String[] args) {
        print(drink( isFriday: true,  haveMoney: true));
        print(drink( isFriday: true,  haveMoney: false));
        print(drink( isFriday: false,  haveMoney: true));
        print(drink( isFriday: false,  haveMoney: false));
    }

    private static boolean drink(boolean isFriday, boolean haveMoney) {
        boolean goToDrink = false;

        if (isFriday) {
            goToDrink = true;
        } else if (haveMoney) {
            goToDrink = true;
        }

        return goToDrink;
    }

    private static void print(boolean result) {
        System.out.println(result);
    }
}
```

Main › drink()

Run: Main

```
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
true
true
true
false

Process finished with exit code 0
```

Но мы знаем, что у нас умная и продвинутая среда разработки, давайте поставим курсор на if и нажмем Alt+Enter. И в выпадающем меню нажмем на Merge sequential if's. T.e. обьединить условия в 1. И мы получим следующее

```
private static boolean drink(boolean isFriday, boolean haveMoney) {
    boolean goToDrink = false;

    if (isFriday || haveMoney) {
        goToDrink = true;
    }

    return goToDrink;
}
```

Если мы пишем сложные условия, то неплохо создать переменную goToDrink и менять ее значение по ходу дела и в конце метода вернуть ее значение. Но как только вы видете что код можно упрстить – то упрощайте. Надеюсь все понимают, что после упрощения (обьединения условий) нам больше не нужна переменная goToDrink. Для этого ставим курсор на линии return goToDrink и нажимаем Alt+Enter и выбираем Move 'return' closer to computation of the value 'goToDrink' и получаем

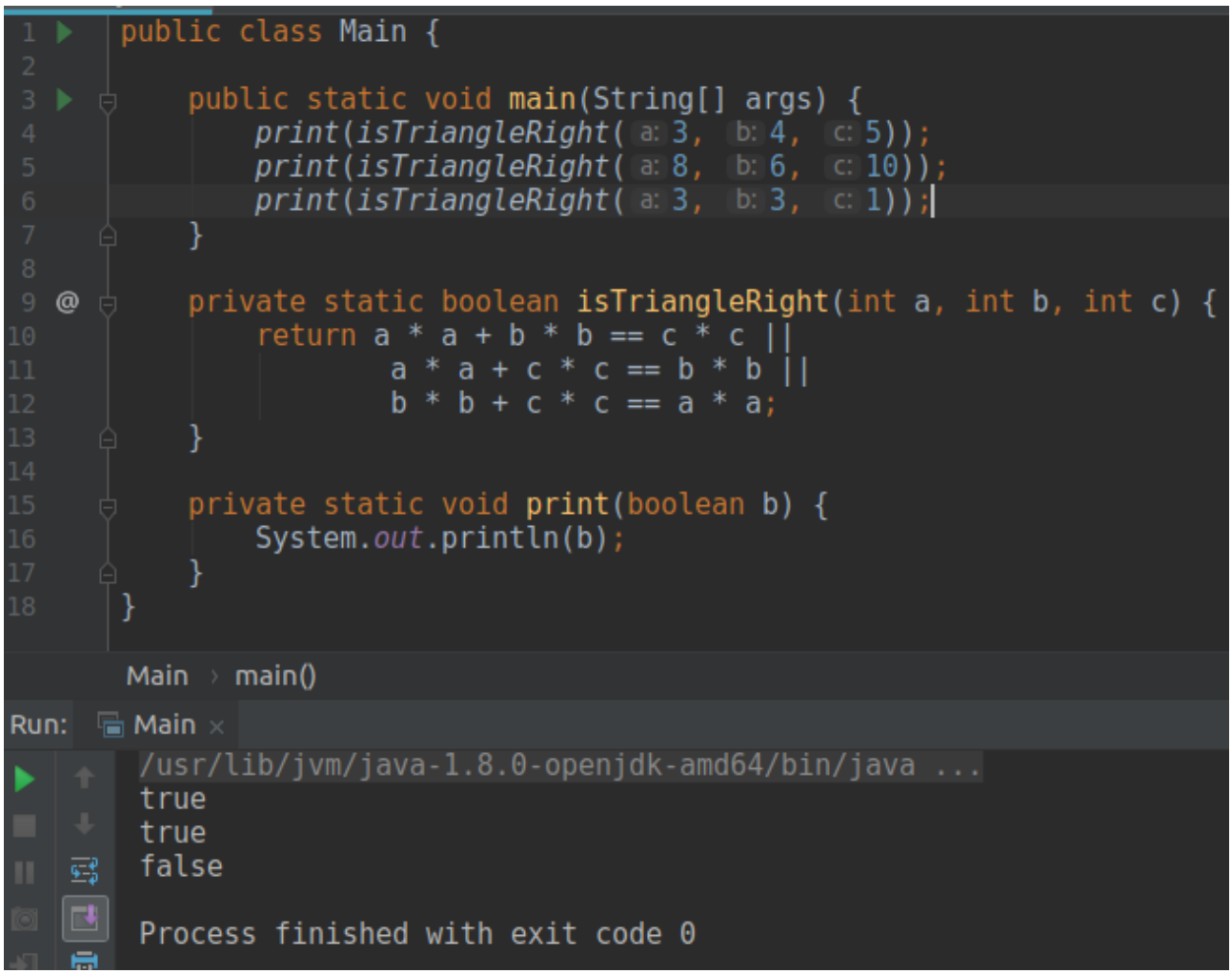
```
10 @    private static boolean drink(boolean isFriday, boolean haveMoney) {
11          if (isFriday || haveMoney) {
12              return true;
13          }
14          return false;
15      }
```

И мы видим как среда разработки выделила if. Несложно догадаться что нужно делать. Упрощаем еще раз и получаем

```
private static boolean drink(boolean isFriday, boolean haveMoney) {
    return isFriday || haveMoney;
}
```

В итоге мы получили выражение с логичеким ИЛИ '||'. Оно работает так, что если одно из значений true, то в итоге все выражение будет true. Конечно же мы (программисты) очень часто пишем условия, да чего там говорить, вся наша жизнь это принятие решений в зависимости от условий. Если будет дождь – взять зонтик. Если будет солнечно и выходные – пойду на пляж. Но что делать когда условия не так просты? Значит нужно комбинировать логические операторы. И для этого мы еще раз вспомним школьную математику. А конкретно теорему Пифагора.

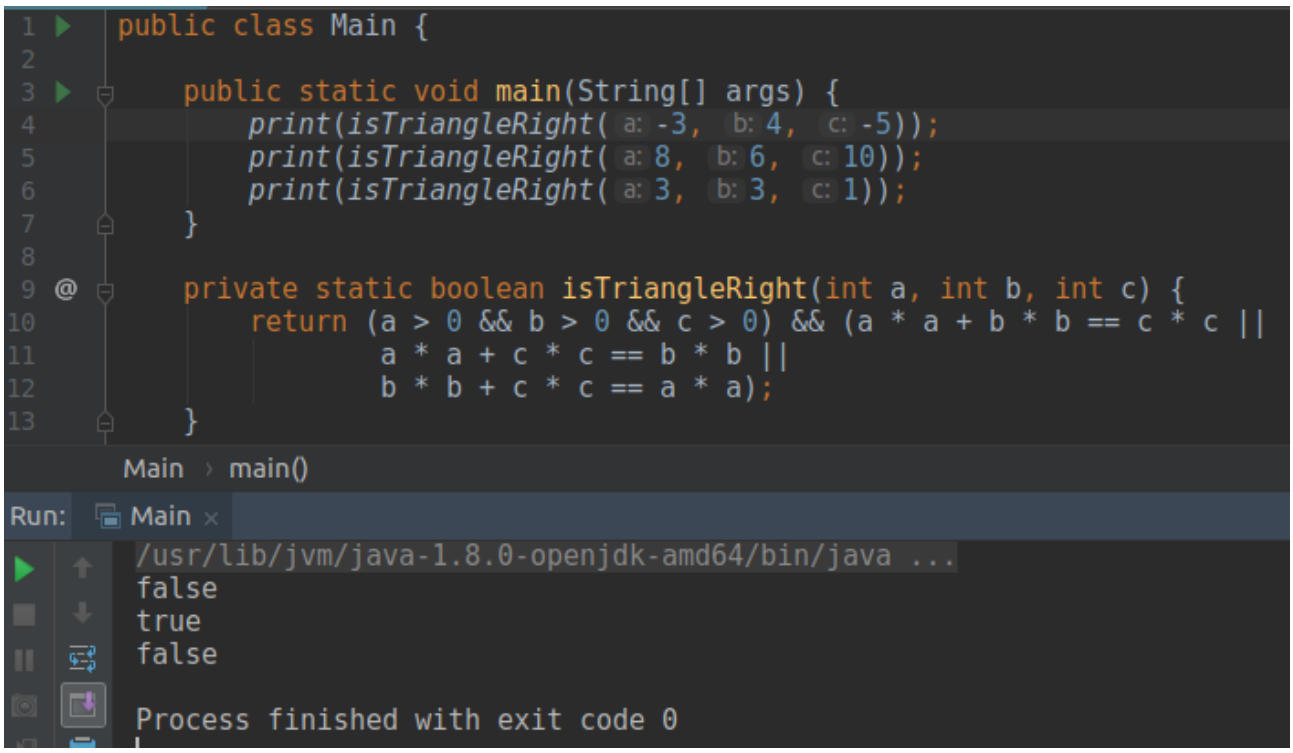
Давайте напишем метод, который проверит, что 3 стороны треугольника образуют прямоугольный треугольник. Можете сейчас сами попробовать написать метод: он принимает 3 числа на вход и отдает boolean – образуют ли эти 3 стороны прямоугольный треугольник. Если кто не помнит, то формула такая $a^2 + b^2 = c^2$. Знаю, у нас нет метода для возведения числа в квадрат, но вы можете его написать сами. Ведь $a^2 = a*a$.



Так как мы не знаем в каком порядке нам выдадут числа на вход, то мы должны проверить все комбинации. Посмотрите на код на линиях 10-12. У нас 3 числа, значит 3 комбинации. Значит они все имеют право на жизнь и надо проверить их все. Здесь конечно было б неплохо проверить что входные аргументы положительные числа. Ведь если мы дадим на вход например -3, -4 и -5, то метод скажет нам true. Но это же не так. Вы можете сейчас сами попробовать добавить проверку, но учитывайте, нужно чтобы одновременно выполнялись 2 условия – все числа положительные и одно из условий с теоремой Пифагора. Здесь нам помогут скобки. Но если вам сложно читать такой код и тем более писать его, никто не заставляет ухищряться, напишите не сразу, а в 2 действия.

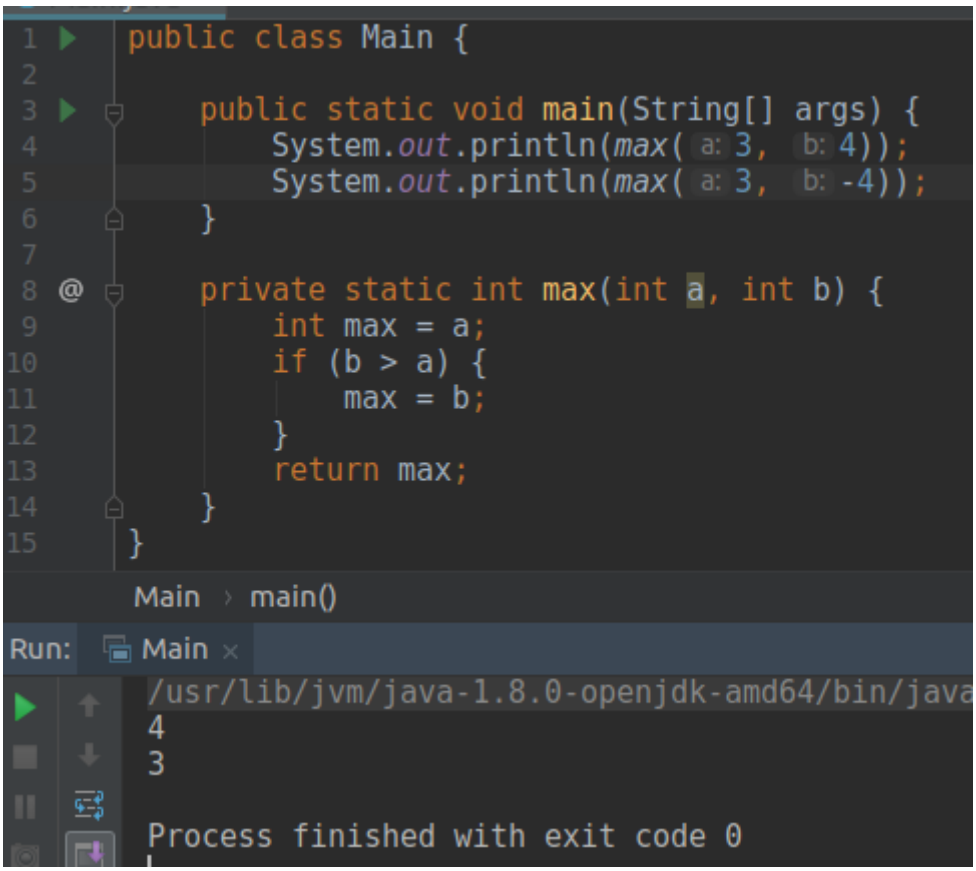
Согласитесь, читать такой код и проще и понятнее. Кстати, старайтесь давать вашим переменным хорошие имена. Из которых сразу понятно что к чему.

```
private static boolean isTriangleRight(int a, int b, int c) {
    boolean numbersPositive = a > 0 && b > 0 && c > 0;
    boolean conditionPythagoras = a * a + b * b == c * c ||
            a * a + c * c == b * b ||
            b * b + c * c == a * a;
    return numbersPositive && conditionPythagoras;
}
```



2. Тернарный оператор

Есть в языке Java такая замечательная штука как тернарный оператор. Ранее мы познакомились с унарным оператором !b, бинарными операторами &&, ||. Теперь настало время для тернарного. Для этого рассмотрим такую простую задачу как найти наибольшее число из двух аргументов. Написать его можно вот так



А теперь я покажу вам как плохо писать код, но это нам поможет узнать что такое тернарный оператор. Вообще запомните – 1 метод 1 return. Это просто best practice.

```
private static int max(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```

Если а больше б, то сразу вернуть а, иначе вернуть б. 5 линий кода и 2 return. Давайте упростим опять же поставив курсор на if и нажав на Alt+Enter. И выберем Replace if else with ?: и получим

```
private static int max(int a, int b) {
    return a > b ? a : b;
}
```

Вот это и есть тернарный оператор. Как его читать? А все так же.

Вернуть а больше б ? Да – а, нет – б.

Запомните, если у вас простой if else то вы всегда можете написать этот код с помощью тернарного оператора (спойлер, в Kotlin он не такой изящный как в Java и можно сказать что его там нет, так что наслаждайтесь пока что).

Ладно, вроде понятно – условие, знак вопроса, значение если условие true, двоеточие, значение для false.

Давайте немного усложним жизнь и напишем метод, который находит максимум из 3 чисел. Вы можете написать его с нуля, а можете использовать метод с 2 аргументами.

```
private static int max(int a, int b, int c) {
    return max(max(a, b), c);
}
```

Сначала найдем максимум первых 2 чисел, после сравним результат с третьим числом.

А можем без использования предыдущего метода? Попробуйте сами.

```
private static int max(int a, int b, int c) {
    int max;
    if (a > b && a > c) {
        max = a;
    } else if (b > a && b > c) {
        max = b;
    } else {
        max = c;
    }
    return max;
}
```

А можем ли мы написать этот же код с помощью тернарного? Конечно же нет, слышу я от вас сразу же. Да, тернарный пригоден для простых случаев, где 1 if else. Но ведь мы можем заменить один результат еще одним тернарным, не так ли? Давайте попробуем.

```
private static int max(int a, int b, int c) {
    return a > b && a > c ? a :
            b > a && b > c ? b : c;
}
```

Здесь каждый сам решает, насколько ему просто читать такой код. Попробуем прочитать его вместе – а больше б и больше чем с, значит а, иначе → б больше а и с одновременно, если да, то б иначе с. Вы опять же можете выделить в переменную второй тернарник. Вот так

```
private static int max(int a, int b, int c) {
    int isBMax = b > a && b > c ? b : c;
    return a > b && a > c ? a : isBMax;
}
```

Сначала мы найдем максимум между б и с, а потом сравним с первым числом.

Для того, чтобы спокойно использовать логические операторы нужно просто получить больше опыта. Поэтому я вам предлагаю написать пару методов

1. Метод получает 3 числа на вход и проверяет – существует ли такой треугольник или нет (он существует если сумма любых 2 сторон больше третьей)

2. Метод получает 4 числа на вход и отображает самое меньшее из них

3. Метод получает на вход параметр булеан типа – сегодня выходной или нет, и сегодня праздник или нет. И если нам надо на работу (в выходные и в праздники не надо) то у нас звонит будильник в 6 утра. Можете усложнить еще так – если просто выходной , то будильник на 11 утра. Но если праздник и выходной совпали, то не надо будильников как и в любой праздничный день.

4. Метод получает на вход 3 числа – значения углов. Исходя из этого проверить его тип – равносторонний, прямоугольный, равнобедренный, тупоугольный, остроугольный.

5. Получаем на вход число – возраст человека. Если он старше 18, то даем доступ к контенту (просто вывести строку в консоль). Но если число неадекватное (сами придумайте проверку возрасту), то выводим – вы бот.

6. Напишите метод, который исходя из входного параметра возраста определит куда ему идти сегодня утром (детсад, школа, колледж, универ, работа, поликлиника (пенсионеры)). Рамки для возраста придумайте сами. Если же входной параметр неадекватный – послать куда подальше.

7. Метод, который не любит Антонов и Денисов. Получаем входной аргумент, если это Антон или Денис, то выводим в консоль – добро пожаловать отсюда. Для всех други имен – добро пожаловать. Можете использовать константу для куска строки.

8. Метод, который считает количество раз когда его вызвали (private static над методом и мейном), Если его вызвали уже пятый раз и более – вывести в консоль – вы превысили лимит. Иначе же – вывести что-то другое.

9. Проверяем состояние воды. На вход получаем одно число и в зависимости от него выводим в консоль агрегатное состояние – лед, вода или пар.

10. Нет ЛГБТ! – написать метод, на вход получаем 2 аргумента (сами поймете какого типа). Если пара гетеросексуальная, то выводим в консоль - совет вам да любовь. Если же гомосексуальная – то выводим в консоль – валите в США!

11. Сравним строки. 2 строки на вход. Отдаем true только если обе строки не пустые и равны друг другу.

12. Проверяем строку на урл. Проверить, что строка начинается с ‘http://’. Метод для проверки найдите сами – угадать можно число логически, просто напишите точку после строки и среда разработки предложит методы. По аналогии с text.isEmpty().

13. Метод получает на вход 5 оценок по 5-бальной шкале. Вывести строку типа – отличник, хорошист, троечник или двоечник исходя из среднего арифметического этих значений. Без округлений.

14. Нет Иванам и Ивановым. Метод на вход получает имя и фамилию по отдельности. Отказать человеку если его зовут Иван или же у него фамилия Иванов. А если это прям Иван Иванов, то вывести строку – КОМБО! Для всех других случаев вывести – Здравствуйте, имя фамилия.

15. Пишем метод модуля. Если входной аргмент числа больше нуля, то вернуть его же, если же меньше нуля, то вернуть это число без знака минус (можно умножить на -1 например). Если это ноль, то вывести в консоль строку – это ноль и вернуть ноль.

16. Прямоугольник или квадрат. На входе 4 числа. Решить, это квадрат или прямоугольник или же простой четырехугольник.

17. Метод анлока урока. На входе номер урока (1 – 100), и булеан – isUserPremium, метод отдает булеан. Если юзер премиум, то можно выбрать любой урок и пройти его. Если же это первый урок, то его может пройти любой юзер без разницы какого статуса. Если номер урока выходит за рамки, то отдаем false, и до этого выдаем в консоль строку – номер невалидный.

18. Метод принимает 2 аргумента, номер игрока и ход строкой. Храним номер игрока в переменной вне метода. Не даем игроку сделать ход если он вызывает метод 2 и более раз подряд. т.е. как в шашках – 1 ход за раз.

19. Предположим у первого игрока преимущество и он может делать 2 хода подряд, тогда как второй игрок может делать лишь 1 ход. Написать метод, который проверит кто его вызывает и выводить в консоль только те ходы которые разрешены тем или иным игроком.

Надеюсь вам хватит этих задач для закрепления знаний по логическим цепочкам и операторам. Вы всегда можете проверить правильность вашего кода вызвав его в мейне.