

Логическая переменная

Когда в дело вступает логика

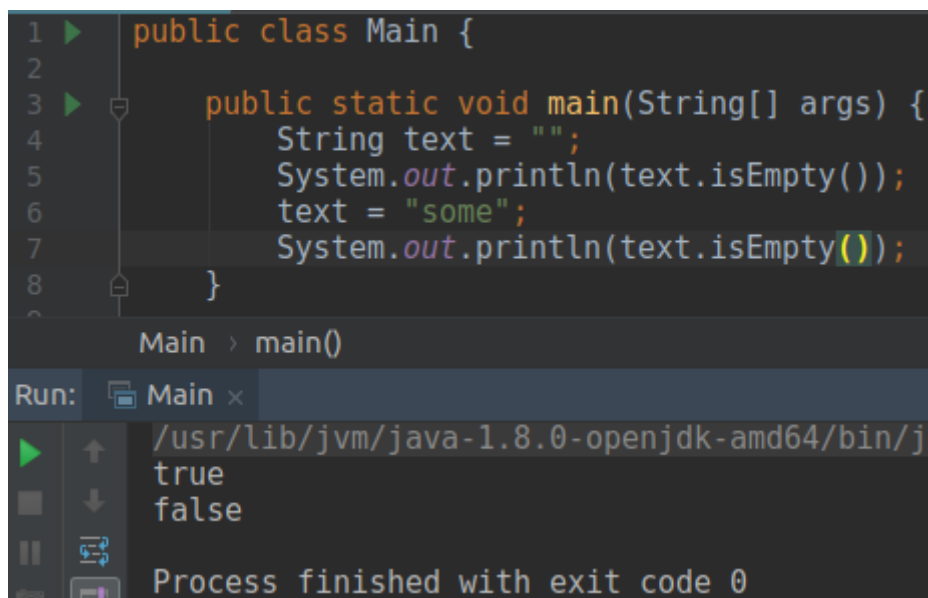
Содержание

1. Как хранить 2 значения
2. Как проверять логику
3. Дебагинг

1. Как хранить 2 значения

В предыдущей лекции мы рассмотрели еще 3 типа переменных. Мы можем хранить строки и числа разного диапазона. Но вернемся к byte: его значения от -128 до 127. А что если мне нужно хранить всего 2 значения? Ну, типа, да/нет, 0 или 1. Одно из двух. Было бы классно иметь какой-то тип для хранения такого рода значений. И оно есть!

Помните мы говорили о том, что у строки есть метод проверки не пустая ли она строка? Давайте посмотрим на следующий код.



```
1 public class Main {
2
3     public static void main(String[] args) {
4         String text = "";
5         System.out.println(text.isEmpty());
6         text = "some";
7         System.out.println(text.isEmpty());
8     }
9 }
```

Main > main()

Run: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/j

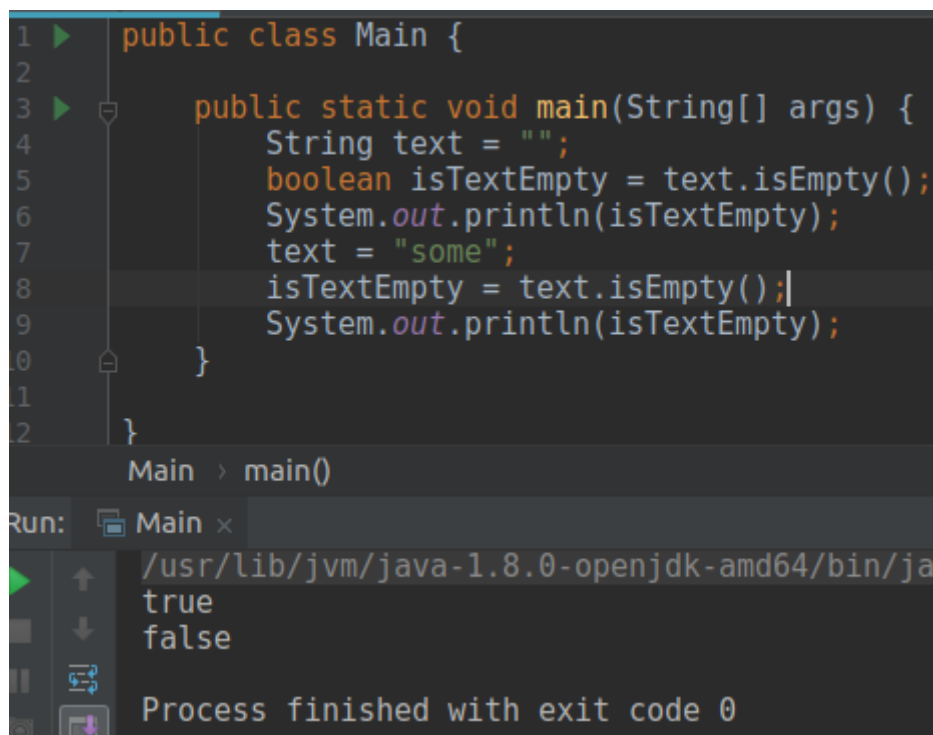
true

false

Process finished with exit code 0

Итак, у нас переменная строки, сначала пустая. Выводим в консоль результат метода проверки пустоты и видим в консоли true, после меняем строку на непустую и опять выводим результат метода и видим false. Что это такое (true/false)? Явно же не строка. Но вы скажете – ну, выглядит как строка. Да, но... Давайте выведем переменную вызова метода isEmpty(). Выделяем text.isEmpty() и нажимаем на Ctrl+Alt+V. Кстати, среда разработки увидела в коде один и тот же код text.isEmpty() и при рефакторинге спросила – вы хотите вынести в переменную только этот код или все, где повторяется этот вызов? Если выбрать все то у вас будет нечто следующее.

И теперь мы можем видеть, что среда разработки сохранила результат вызова метода в переменную типа `boolean`. Так что теперь мы можем уверенно сказать, что это не строка.



```
1 public class Main {
2
3     public static void main(String[] args) {
4         String text = "";
5         boolean isEmpty = text.isEmpty();
6         System.out.println(isEmpty);
7         text = "some";
8         isEmpty = text.isEmpty();
9         System.out.println(isEmpty);
10    }
11
12 }
```

Main > main()

Run: Main x

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

Process finished with exit code 0
```

Дословно мы создали переменную, с названием `isEmpty` является ли пустой строка в которую храним результат вызова метода `text.isEmpty()` пустая ли строка у переменной типа строка. И чтобы обновить значение с `true` на `false` мы должны переопределить так же как и переопределили текст. Линии 7 и 8.

Теперь, предположим, мы хотим проверить какое-нибудь математическое выражение. Для этого нам нужны операторы сравнения ровно так же как и в школьной математике. Давайте посмотрим.

У нас есть число 1 и мы хотим сравнить его с нулем. Вариантов несколько

линия 5, число больше чем 0

линия 6, число меньше чем 0

линия 7, число равно 1 (заметьте, что для присваивания значения мы используем знак `=`, а для сравнения значений `==`. Этот знак работает не только для чисел, а для всех видов переменных)

линия 8, число больше или равно 1

линия 9, число меньше или равно 0

линия 10, число не равно 1. Если для проверки равенства у нас был знак `==` то для проверки не равенства у нас знак `!=`. Его так же можно использовать для других видов переменных.

А знаки `>`, `<`, `>=`, `<=` используем только для чисел.

```
1 public class Main {
2
3     public static void main(String[] args) {
4         int a = 1;
5         print(a > 0);
6         print(a < 0);
7         print(a == 1);
8         print(a >= 1);
9         print(a <= 0);
10        print(a != 1);
11    }
12
13    private static void print(boolean trueOrFalse) {
14        System.out.println(trueOrFalse);
15    }
16 }
```

Main > main()

Run: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

true
false
true
true
false
false

Process finished with exit code 0

И для полноты картины давайте посмотрим на следующий пример. Вот у нас есть метод у строки, который проверяет ее на пустоту. А как мне вывести в консоль обратное? Типа

```
1 public class Main {
2
3     public static void main(String[] args) {
4         String text = "";
5         boolean isEmpty = text.isEmpty();
6         System.out.println("text" + text + " is not empty: " + !isEmpty);
7     }
8 }
```

Main

Run: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...

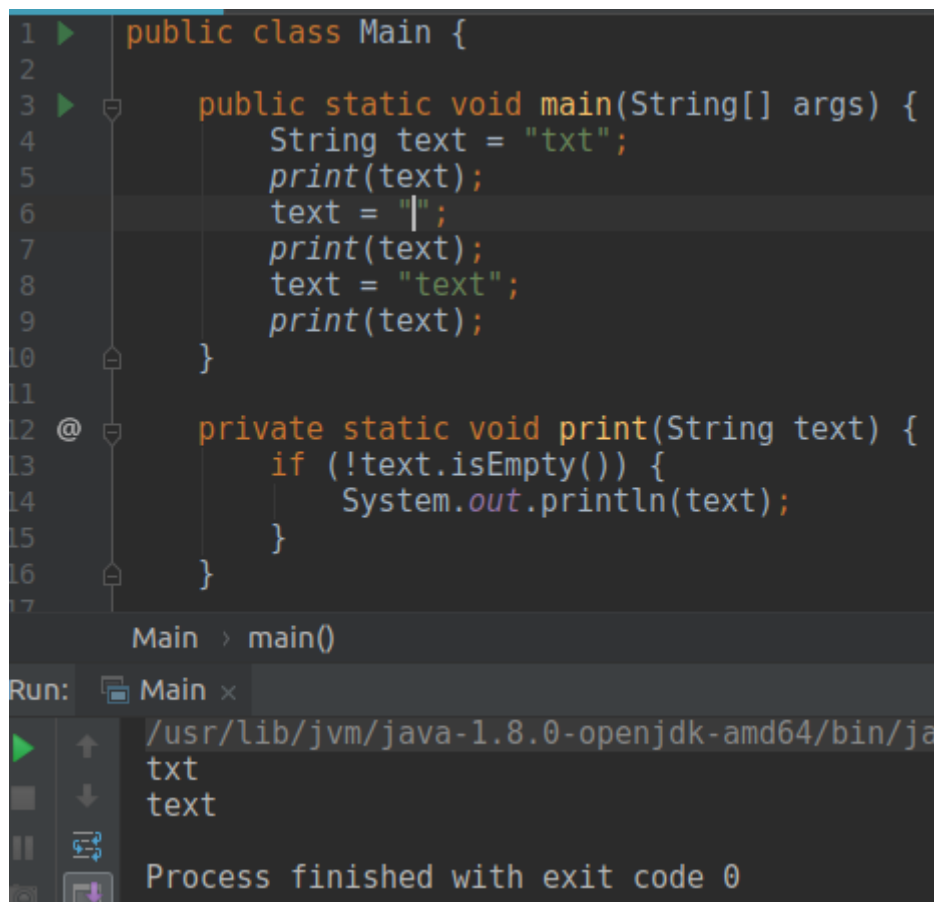
text is not empty: false

Process finished with exit code 0

Так как текст у нас пустой, то переменная `isEmpty` будет `true`. Но мы хотим в консоль вывести признак, что он не пустой, т.е. обратный от пустого. Для этого посмотрим на линию 6. Мы можем получить обратное от значения с помощью знака `!` перед именем переменной.

2. Как проверять логику

Помните когда мы рассматривали метод у строки isEmpty() мы говорили о том, что было бы неплохо перед вызовом System.out.println проверить на пустоту строку и если она пуста то лишний раз не выводить в консоль ничего. Итак, настало время познакомиться с, наверное, самой главной частью синтаксиса языка Java. If



```
1 public class Main {
2
3     public static void main(String[] args) {
4         String text = "txt";
5         print(text);
6         text = "";
7         print(text);
8         text = "text";
9         print(text);
10    }
11
12    private static void print(String text) {
13        if (!text.isEmpty()) {
14            System.out.println(text);
15        }
16    }
17 }
```

Main > main()

Run: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java

txt

text

Process finished with exit code 0

Сначала посмотрим на метод print. На линии 13 мы видим if (!text.isEmpty()). Как видите мы пишем ключевое слово if и после него должны идти в скобках какие-то условия, т.е. нечто, что в итоге даст нам boolean. После открывается фигурная скобка... Но ведь мы использовали их для методов и классов, скажете вы. Да, а еще для if. И сразу скажу вам, что если у вас после проверки (if) идет всего 1 линия кода, то можно фигурные не ставить. Но мы ставим заранее, потому что предполагаем в дальнейшем добавление кода. Тогда будет удобно сразу добавить код между фигурных, а не писать их заново.

Теперь, чтобы вы понимали что этот код работает мы в мейне написали вызов 3 раза. Сначала для непустой строки, потом для пустой и еще раз для непустой. Почему я так сделал?

Подумайте. Ведь если бы код на линии 13 не работал, то мы бы увидели в консоли не 2 текста подряд, а 3 текста, второй был бы пустым, т.е. так

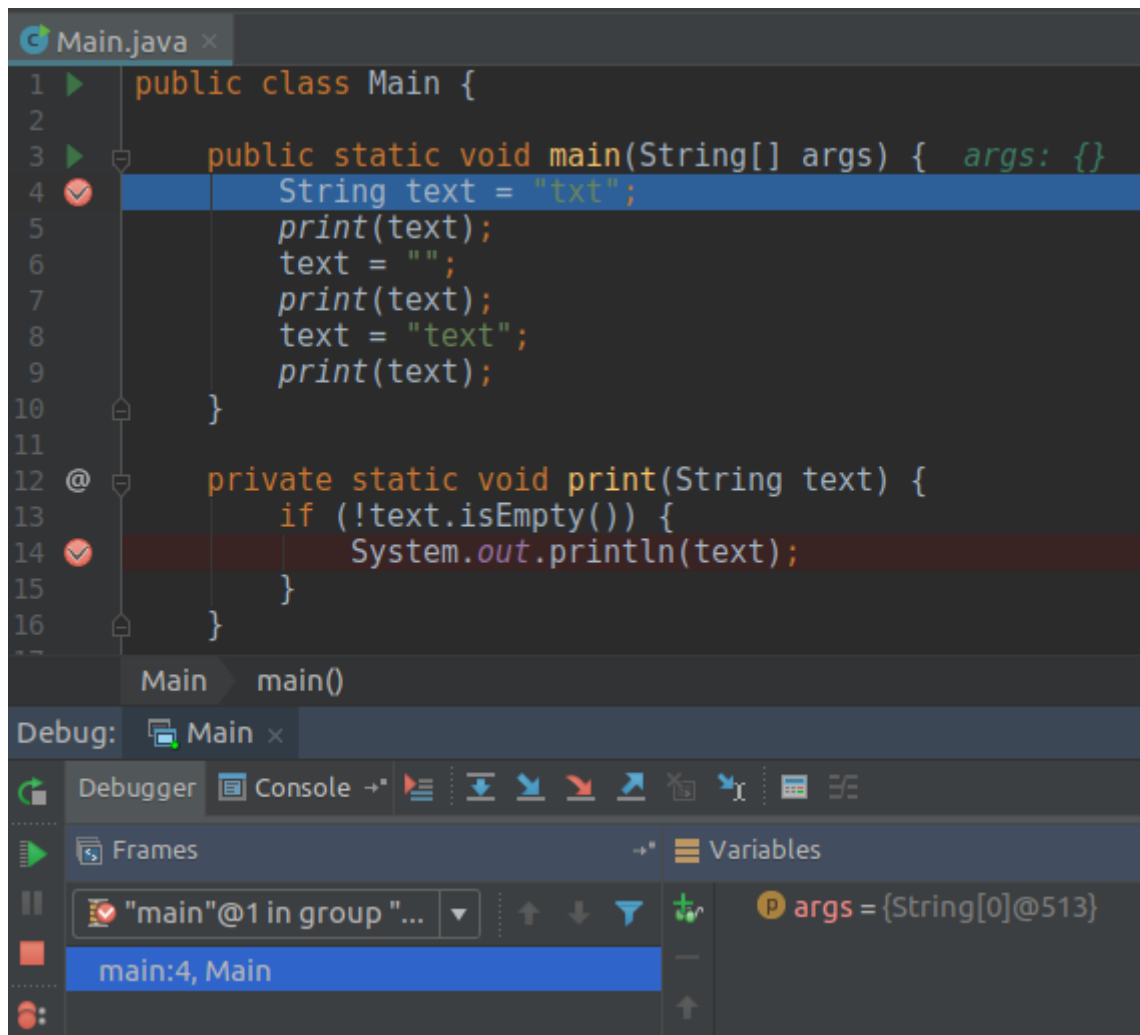
txt

text

Помните мы говорили о том, что компилятор смотрит на код последовательно, сверху вниз? Наверно было бы классно проверить это дело. Видеть вживую по мере продвижения кода где

он остановился, в какой if зашел, а где нет. И это можно сделать, если у вас продвинутая среда разработки. Смотрим.

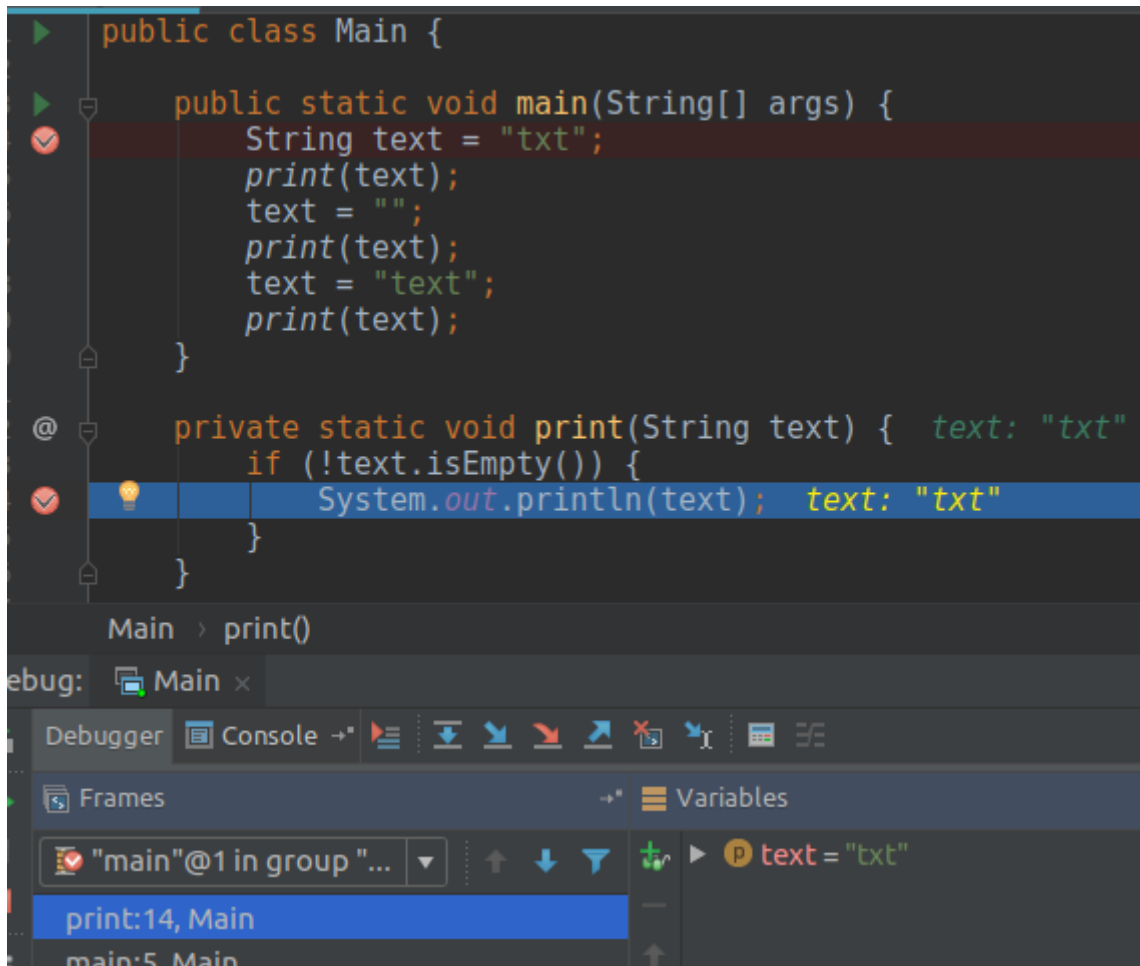
3. Дебагинг



Сначала нам нужно поставить breakpoints точки остановки кода. Для этого просто кликаем на линии кода слева и в итоге получаем красные кружки. После этого нажимаем на Run рядом с мейн методом и выбираем там `Debug.main()`. И вы увидите как наш компилятор остановился на первой строке. Теперь, в дебагере мы можем идти буквально по строчкам дальше. Нажмите на стрелку вниз в окне дебагера внизу или же F8. Если вы продолжите нажимать на F8 то увидите как компилятор идет по строкам и заходит в метод `print` и конкретно на линию 14. Вы можете поставить breakpoint на линии 13 и увидеть как компилятор заходит туда 3 раза. А на линию 14 всего 2 раза.

Продолжая нажимать на F8 мы увидим как компилятор перескакивает с 7 линии на 8 и не заходит на 14. И в конце концов оказывается на линии 10 и завершает работу.

Мы поговорим о дебагере еще не раз, но пока что это тот минимум знаний который необходим и достаточен в большинстве случаев. Просто ставим брейкпойнты и идем по коду, смотрим куда он заходит, а куда нет. Тем самым проверяя наш код в режиме реального времени.



Задание для закрепления.

Помните метод для деления 2 чисел? Школьная математика говорит нам, что на ноль делить нельзя. Давайте тогда улучшим наш код так, чтобы он проверял делитель на ноль и в этом случае ничего не делал. Продебажьте код и посмотрите что компилятор не заходит туда.