

Переменные: как использовать

Первичная и повторная инициализация переменной

Содержание

1. Повторное использование и повторная инициализация переменных
2. Запрет на повторную инициализацию переменной

1. Повторное использование и повторная инициализация переменных

Если у вас возникли трудности с домашним заданием из предыдущей лекции, то вот вам решение

```
1 public class Main {
2
3     public static void main(String[] args) {
4         String first = "Hello World!";
5         String second = "Это моя первая программа на Джава";
6         String total = concat(first, second);
7         print(total);
8     }
9
10    private static String concat(String string1, String string2) {
11        return string1 + "\n" + string2;
12    }
13
14    private static void print(String text) {
15        System.out.println(text);
16    }
17 }
```

Main > main()

Run: Main x

/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
Hello World!
Это моя первая программа на Джава
Process finished with exit code 0

А мы вернемся к тому, на чем закончили предыдущую лекцию.

Как вы помните (а если нет, то откройте лекцию 5 и посмотрите) мы обсуждали переменные. Первое для чего они нужны, чтобы сложный вложенный код читался проще, мы не хотим видеть вызов 2 функций в третьей, которая вызывается четвертой. Нам проще читать слова (переменные) нежели функции. Но это не единственное (далеко не единственное) применение переменных. Точно так же как и с функциями мы можем использовать переменные не 1 раз, а сколько угодно. Давайте посмотрим.

```
1 public class Main {
2
3     public static void main(String[] args) {
4         int sum1 = sum(3, 4);
5         int diff1 = difference(7, 5);
6         final int multiplication1 = multiply(sum1, diff1);
7         print(multiplication1);
8
9         print(multiply(sum1, multiplication1));
10    }
11 }
```

Main > main()

Run: Main x

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

14

98

Process finished with exit code 0

Мы вызвали в мейне 2 раза метод умножения чисел. А почему мы не можем использовать результат перемножения в первом примере второй раз? Посмотрите на линию 9. Мы перемножаем то, что у нас получилось при суммировании 3 и 4 в линии 1 с тем, что у нас получилось при перемножении суммы и разницы на линии 6.

Теперь, перейдем к следующему плюсу использования переменных. Мы поняли что их можно использовать повторно, а что насчет значения переменной? Может мы могли бы использовать одну и ту же переменную но с другим значением? И кто-то спросит зачем так делать вообще? А я покажу зачем. Смотрите.

```
1 public class Main {
2
3     public static void main(String[] args) {
4         int sum1 = sum(3, 4);
5         int diff1 = difference(7, 5);
6         final int multiplication1 = multiply(sum1, diff1);
7         print(multiplication1);
8
9         int sum2 = sum(5, 6);
10        int diff2 = difference(8, 4);
11        final int multiplication2 = multiply(sum2, diff2);
12        print(multiplication2);
13    }
14 }
```

Main > main()

Run: Main x

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

14

44

Process finished with exit code 0

Я просто по аналогии с первым решением сделал еще одно. Создал еще переменные: `sum2`, `diff2`, `multiplication2`. И кто-то скажет – ну а что такого? Сколько нужно столько и создаем переменных, кому какие заботы? Ну... у вас же оперативки не безлимитные, не? Хотя это мало кому что говорит. Ок, на моем ноуте 16 ГБ ОЗУ, и что? Создание еще 3 переменных это как-то в тягость? Ну нет, конечно же. Хоть сто. Но мы же не будем всю жизнь писать программы на 10 линий кода. И если вы будете злоупотреблять созданием новых переменных там, где можно было бы этого избежать, то рано или поздно столкнетесь с ситуацией, когда ваша программа откровенно подтормаживает. Об этом поговорим как-нибудь потом, как в джава происходит работа с памятью. А пока давайте посмотрим на следующий код.



```
1 public class Main {
2
3     public static void main(String[] args) {
4         int sum1 = sum(3, 4);
5         int diff1 = difference(7, 5);
6         final int multiplication1 = multiply(sum1, diff1);
7         print(multiplication1);
8
9         sum1 = sum(5, 6);
10        diff1 = difference(8, 4);
11        final int multiplication2 = multiply(sum1, diff1);
12        print(multiplication2);
13    }
14}
```

Run: Main x

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

14

44

Process finished with exit code 0

Итак, я переиспользовал переменные `sum1` и `diff1`. Что это нам дало? А то, что мы не заняли лишнее место в памяти. А почему же мы не переиспользовали переменную `multiplication1`? Об этом чуть позже. А пока, давайте придумаем хороший пример, где лучше видно зачем нам переиспользовать переменную.

Предположим, что у нас есть какая-то игра, например шахматы. Где шаги делают игроки по очереди. И мы хотим знать кто сделал шаг. Для этого мы создаем игроков (просто зададим им имена и все). И создаем еще одну переменную `currentPlayer`. Мы будем менять ее значение каждый раз когда кто-то сделает шаг. И напишем метод, который выводит в консоль информацию о том, какой игрок сделал шаг. А теперь посмотрите на код на линиях 8, 10, 12. Он одинаковый. Мы вызываем один и тот же метод и передаем ему одну и ту же переменную в качестве аргумента. Это очень упрощает жизнь, не правда ли? Согласен, особо толку от такой программы тоже не особо много, но по крайней мере вы можете увидеть как это работает. И еще одно. Вы заметили код на линиях 7, 9, 11? Раньше мы инициализировали переменную типа строки строкой. А теперь мы сохраняем в переменную `currentPlayer` не строку, а другую переменную? Что? Нет, когда мы пишем `currentPlayer = player1`, то

компилятор джава хранит в переменной `currentPlayer` значение, которое хранится в переменной `player1`, т.е. "John".

```
1 public class Main {
2
3     public static void main(String[] args) {
4         final String player1 = "John";
5         final String player2 = "Mike";
6
7         String currentPlayer = player1;
8         doStep(currentPlayer);
9         currentPlayer = player2;
10        doStep(currentPlayer);
11        currentPlayer = player1;
12        doStep(currentPlayer);
13    }
14
15    private static void doStep(String playerName) {
16        System.out.println(playerName + " is making step");
17    }
18 }
```

Main > main()

Run: Main x

```
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java ...
John is making step
Mike is making step
John is making step

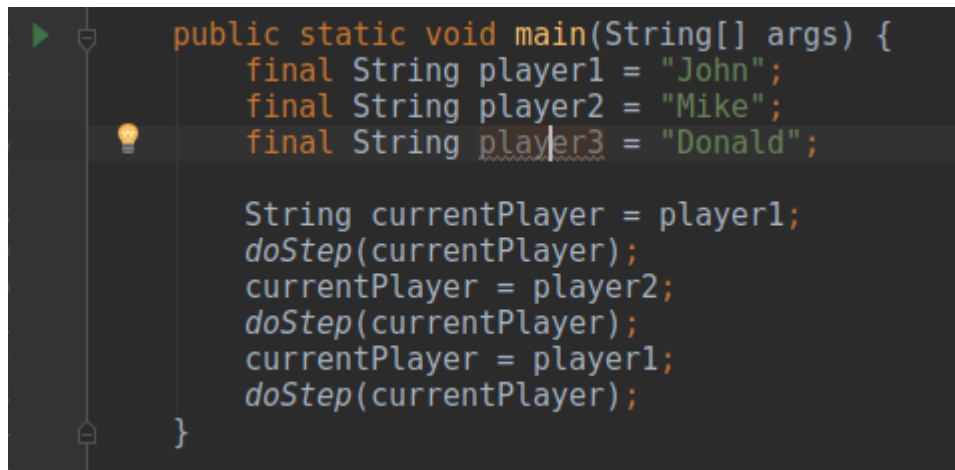
Process finished with exit code 0
```

Помните наш стол с 3 ящиками? Представьте, что у вас в первом ящике первый предмет, а во втором ящике второй предмет. А в третий ящик я не кладу предмет, а кладу скажем так записку. Сначала там пишу: смотри в ящик номер 1, а потом, на следующий день я кладу в ящик номер 3 записку с текстом: смотри в ящик номер 2. Или можете думать не о записках, а о ключах. Я в ящик 3 кладу ключ от ящика номер 1, а в иной день кладу в него ключ от ящика номер 2. И вы (компилятор) видите ключ от какого ящика и открываете его и достаете тот предмет, который там лежит. Надеюсь понятно. Если нет, то ничего страшного, со временем поймете (на других примерах).

Если посмотреть на предыдущий код, то там можно увидеть следующее. У меня есть 2 ящика: `sum1` и `diff1`. Сначала я кладу в эти ящики одни предметы, а потом, когда я их достал и показал на столе, просто кладу в эти ящики следующие предметы. Вместо того, чтобы создавать еще ящики в столе и забивать их предметами, которые мне уже не нужны. Т.е. представьте, что у вас в столе 4 ящика, но вы сначала положили в ящики номер 1 и 2 предметы, потом достали их и показали на столе, а потом у вас появились новые предметы, но вы их не кладете в эти пустые ящики номер 1 и 2, а кладете их в ящики 3 и 4. Зачем? Вы не можете бесконечно создавать ящики. Лучше переиспользовать хранилища.

Ровно так же и работает память в джава. Наши данные (строки, числа) хранятся в памяти до тех пор пока они нужны. Компилятор видит например, что переменная `currentPlayer` используется на линии 12 и будет хранить ее в памяти пока не использует ее. Если бы в

нашей программе были бы еще линии, например 13, 14 и так далее, но там бы мы просто выдавали на экран что-то типа Game Over, то из памяти бы уже убрали эту переменную. Т.е. из ящика с именем currentPlayer достали значение и показали на столе, и не клали бы обратно в этот ящик, потому что этот предмет уже не нужен. Надеюсь понятно. Чтобы это понять, просто посмотрите на этот код.



```
public static void main(String[] args) {  
    final String player1 = "John";  
    final String player2 = "Mike";  
    final String player3 = "Donald";  
  
    String currentPlayer = player1;  
    doStep(currentPlayer);  
    currentPlayer = player2;  
    doStep(currentPlayer);  
    currentPlayer = player1;  
    doStep(currentPlayer);  
}
```

Как видите, мы создали переменную player3, но ее нигде не использовали. Если навести курсор на нее, как на скриншоте, то вы увидите подсказку нашей среды разработки – Variable player3 is never used. Если вы нажмете на лампочку, то там будет первое же предложение Remove variable 'player3'. Т.е. наша умная среда разработки предлагает удалить то, что мы не используем. Это как вы добавили к вашему столу еще один ящик, но никогда бы не использовали его. Логично, не правда ли?

2. Запрет на повторное использование переменной

Настало время вам рассказать про ключевое слово final если оно написано перед объявлением переменной. Давайте попробуем по ходу дела дать нашему игроку номер 1 другое имя, т.е. попробуем в ящик с именем player1 положить другой предмет. (Кстати, если вы меняете содержимое ящиков, то предыдущий предмет нужно положить в мусорное ведро, для особо интересующихся погуглите Java Garbage Collector).

Итак, смотрим. Даже тогда, когда вы еще не запустили код, нажав на Run, то вы можете видеть как среда разработки подчеркивает красной волнистой player1. Да-да, так же как и в Microsoft Word подчеркиваются неправильные слова. Если поставить курсор на эту переменную на линии 10, то можно увидеть подсказку – Cannot assign a value to final variable 'player1'. Что в принципе и высветится если вы нажмете на Run. Т.е. ключевое слово final перед именем переменной означает, что ее нельзя повторно инициализировать. Т.е. менять значение которое находится там. Класть в ящик другой предмет. Считайте что у вас нет прав менять содержимое ящика. Достали предмет, показали на столе (в консоли) и все. Надо класть обратно или выкинуть. Дело в том, что слово final можно писать не только перед именем переменной, но мы пока не будем говорить об этом сейчас, а тогда когда придет время (Забегаая вперед его можно писать перед именем класса и методом).

Вообще мы называем переменные, которые объявлены `final` константами (подробнее в следующей лекции), т.е. постоянными. Согласен, странно немного, ведь суть переменной именно в том, чтобы меняться, не? Как же тогда получается переменная, которую нельзя поменять и она постоянна? Но на это есть простой ответ: так исторически сложилось.

```
1 public class Main {
2
3     public static void main(String[] args) {
4         final String player1 = "John";
5         final String player2 = "Mike";
6
7         String currentPlayer = player1;
8         doStep(currentPlayer);
9         currentPlayer = player2;
10        player1 = "Donald";
11        doStep(currentPlayer);
12        currentPlayer = player1;
13        doStep(currentPlayer);
14    }
15 }
```

Main > main()

Messages: Build ×

- Information: java: Errors occurred while compiling module 'LearnJava'
- Information: javac 1.8.0_282 was used to compile java sources
- Information: 28.03.21 15:29 - Compilation completed with 1 error and 0 warnings in 560 ms

▼ /home/johnny/LearnJava/src/Main.java

Error:(10, 9) java: cannot assign a value to final variable player1

В следующей лекции мы поговорим об уровнях видимости переменных и рассмотрим еще 2 типа. А пока попробуйте разобраться с этим всем. Для этого вот вам задание:

Напишите код, который отображает вечернее обжорство какого-нибудь персонажа.

Вам понадобится переменная для этого персонажа (подумайте сами какого типа), а также придумайте несколько видов еды и напишите функцию, которая бы выводила нечто похожее на – персонаж съел блюдо и он счастлив. Подумайте над тем, чтобы использовать минимальное количество переменных, при условии, что наш обжора любит поесть как минимум 6 разных видов еды за вечер.

Если вы будете претерпевать трудности во время выполнения этого задания, то просто посмотрите эту лекцию еще раз или дождитесь следующей, ее начнем с выполнения этого задания.

Всем удачи!