

ПРОГРАММИРОВАНИЕ НА JAVA: ПЕРЕМЕННЫЕ, ОПЕРАТОРЫ, РАБОТА С ОТЛАДЧИКОМ



ВАСИЛИЙ ДОРОХИН



ВАСИЛИЙ ДОРОХИН

QuadCode, QA Engineer





ПЛАН ЗАНЯТИЯ

1. [Работа в IntelliJ IDEA](#)
2. [Переменные](#)
3. [Операторы](#)
4. [Примитивные типы данных](#)
5. [Отладка](#)
6. [Итоги](#)



INTELLIJ IDEA

IntelliJ IDEA (далее — IDEA) — самая популярная среда разработки в мире Java.

Использование среды разработки позволяет нам удобнее:

1. Создавать программы.
2. Модифицировать их.
3. Тестировать.
4. И т.д.



INTELLIJ IDEA

Навыки эффективного использования позволят вам разрабатывать приложения (программировать на Java, тестировать и т.д.) в разы эффективнее.

Поэтому старайтесь с первых шагов пользоваться не мышью, а именно клавиатурными сокращениями.

Обязательно практикуйтесь в использовании горячих клавиш — не откладывайте на потом.



КЛАВИАТУРНЫЕ СОКРАЩЕНИЯ

Важно: на ноутбуках часто клавиши F1-F8 необходимо нажимать совместно с клавишей Fn — в настройках можно отключить это поведение (иначе вам придётся нажимать на клавишу больше).

Для пользователей Macbook: если у вас Macbook с «урезанной» клавиатурой (нет клавиши Insert), то вам придётся назначить другое клавиатурное сокращение (см. видео).

Мы будем показывать необходимые клавиатурные сокращения по мере прохождения курса.

ОСНОВЫ РАБОТЫ С IDEA

Обратите внимание, что при демонстрации у лектора отображаются те клавиатурные сокращения, которые он использует (включен плагин Presentation Assistant):

Reformat Code via `⌘⌥L` (Ctrl+Alt+L for Win/Linux)

Если вдруг клавиатурные сокращения не отображаются — сообщите об этом лектору!



ПЕРЕМЕННЫЕ

ДАННЫЕ

Посмотрим на таблицу с данными:

	A	B	C
1	150	5	750

Вопрос к аудитории: что вы можете сказать о данных в этой таблице?

ДАННЫЕ

Можно только гадать, что это за данные.

Но всё меняется, если добавить к этим данным некоторые понятные нам "имена":

	A	B	C
1	price	count	total
2	150	5	750

Даже несмотря на то, что они на английском языке, угадывается смысл.

ПЕРЕМЕННЫЕ

То же самое с Java-кодом. Сравним вот этот вариант:

```
public static void main(String[] args) {  
    System.out.println(150 * 5);  
}
```

С ЭТИМ КОДОМ:

```
public static void main(String[] args) {  
    int price = 150;  
    int count = 5;  
    int total = price * count;  
    System.out.println(total);  
}
```

Даже не зная, что такое `int` можно в целом, понять общий смысл: цена умножается на количество, чтобы получить итоговую сумму.

MAGIC CONSTANTS

```
public static void main(String[] args) {  
    System.out.println(150 * 5);  
}
```

Кроме того, вот такие «захардкоженные» (жёстко прописанные в коде) значения часто называют «магическими значениями» (magic values). Потому что через какое-то время все забывают, что они значили и почему имеют такое значение: «тут происходит какая-то магия».

Это считается одним из признаков «плохого кода» в программировании.

ПЕРЕМЕННЫЕ

Переменные — это просто удобные имена для хранения наших данных.

Т.е. мы сами придумали их для того, чтобы нам через неделю, месяц и даже пару лет **было понятно, какие данные и для чего там хранятся.**

Правильно подобранные переменные позволяют сделать программу поддерживаемой — при необходимости изменения будет понятно, что и по какой логике изменяется.

И наоборот, например, вот этот вариант ничем не лучше самого первого:

```
public static void main(String[] args) {  
    int a = 150;  
    int b = 5;  
    int c = a * b;  
    System.out.println(c);  
}
```



ПЕРЕМЕННЫЕ

Q: Но ведь первый вариант программы был короче?

A: В данном случае важна не краткость, а то, что программа становится «понятной» — приложения живут долго, меняются требования, разработчики, команды и проекты.

Возможность безопасно изменять приложения и поддерживать в подавляющем большинстве случаев важнее «краткости».

ПЕРЕМЕННЫЕ

Q: Хорошо, но почему бы их не назвать по-русски, например, цена? Ведь тогда станет ещё понятнее.

A: Над вашим проектом впоследствии могут работать сотрудники из других стран. Если каждый начнёт называть на своём языке, проект превратится в тренировку пользования словарём. Поэтому общепринято все имена давать на английском языке.

Важно: транслит — это не английский язык! Т.е. `cena`, `kolichestvo` и `itogo` — это недопустимые имена*.

Примечание*: компилятору Java всё равно (за рядом исключений), как вы называете свои переменные. Но мы будем отправлять на доработку ДЗ, в которых переменные названы не по правилам.

ПЕРЕМЕННЫЕ

У переменных есть три ключевых свойства:

1. **Имя** — определяет, как обращаться к данным, хранящимся по этому имени. Воспринимайте это как название определённой ячейки в таблице.
2. **Тип данных** — то какие данные можно по этому имени. Нужно для того, чтобы случайно в количество (которое, например, в штуках) не положить неправильных данных: строку много. Компилятор сам будет следить за этим.
3. **Сами данные** — то, какие данные сейчас хранятся. Это значит, что данные можно изменять (так же, как данные в ячейке таблицы).

ОБЪЯВЛЕНИЕ

Объявление — это сообщение компилятору о том, что мы хотим определить в нашем приложении некоторое имя.

Для того, чтобы объявить переменную, нужно указать тип и имя. Объявление завершается символом `;`.

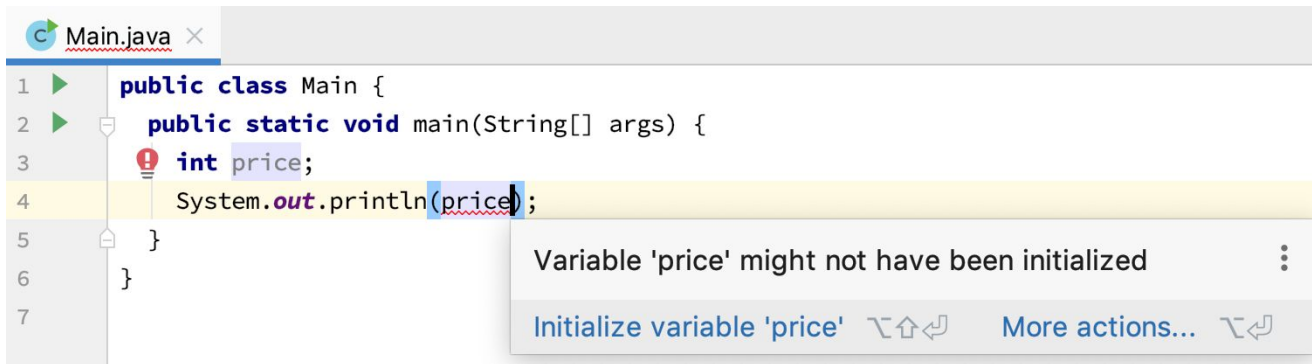
```
public static void main(String[] args) {  
    int price;  
}
```

`int` означает тип для хранения целых чисел (о типах чуть позже).

Переменная, которую мы объявили является неинициализированной (к ней не привязано никакого значения).

ИНИЦИАЛИЗАЦИЯ

Важно: использовать неинициализированную переменную нельзя, т.к. мы к этому имени ещё не «привязали значения»:

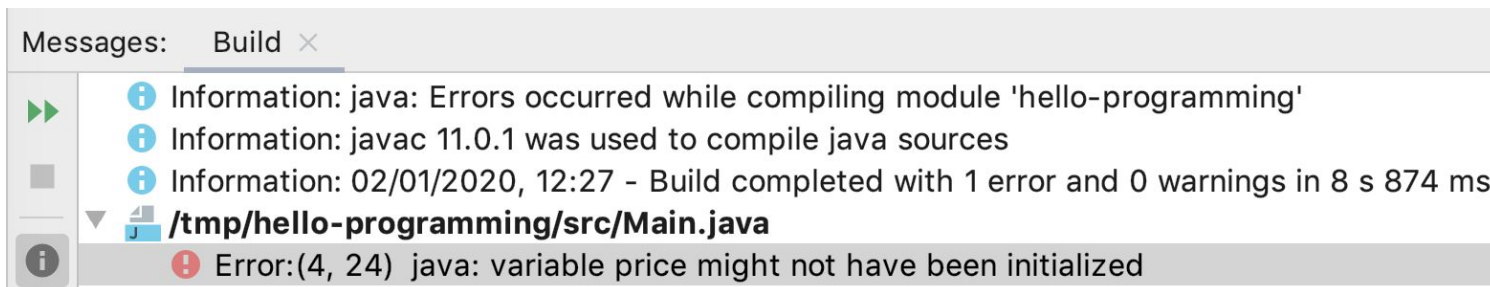


```
1 public class Main {  
2     public static void main(String[] args) {  
3         int price;  
4         System.out.println(price);  
5     }  
6 }  
7
```

Variable 'price' might not have been initialized

[Initialize variable 'price'](#) [More actions...](#)

Об этом же вам сообщит компилятор:



```
Messages: Build ×  
▶▶ Information: java: Errors occurred while compiling module 'hello-programming'  
▶▶ Information: javac 11.0.1 was used to compile java sources  
▶▶ Information: 02/01/2020, 12:27 - Build completed with 1 error and 0 warnings in 8 s 874 ms  
▼ /tmp/hello-programming/src/Main.java  
❗ Error:(4, 24) java: variable price might not have been initialized
```

ИНИЦИАЛИЗАЦИЯ

Единственное, что можно сделать с неинициализированной переменной — инициализировать её (иногда говорят «присвоить значение»):

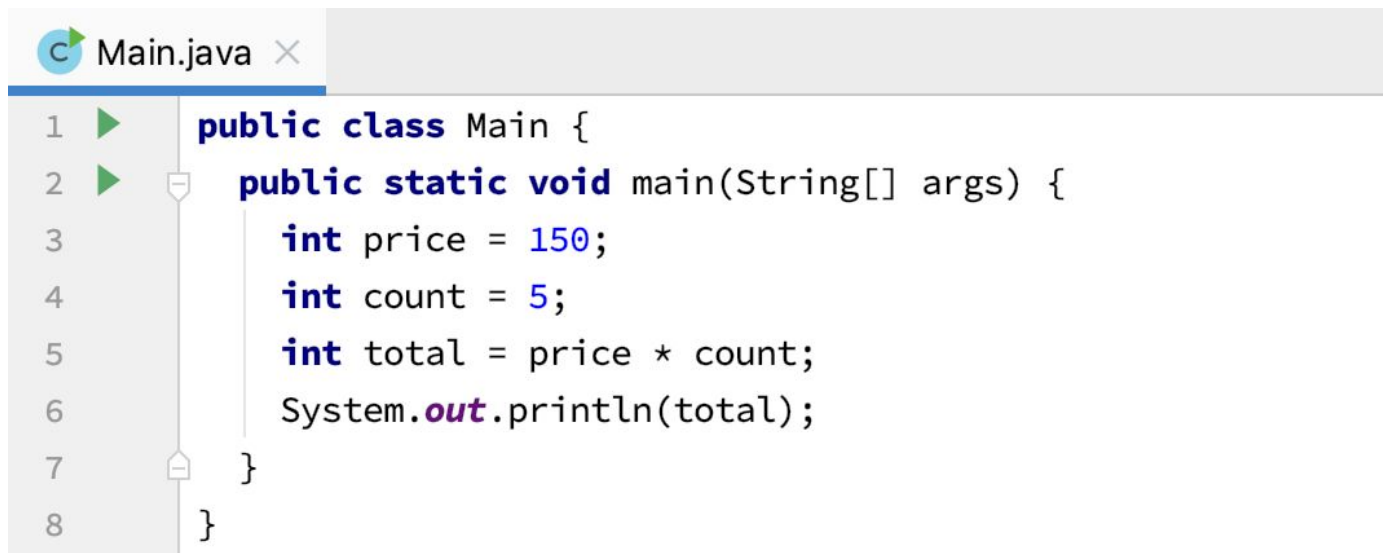
```
public static void main(String[] args) {  
    int price;  
    price = 150;  
}
```

Если мы уже в момент объявления переменной знаем, какое значение хотим ей присвоить, нужно использовать сокращённую форму:

```
public static void main(String[] args) {  
    int price = 150;  
}
```

ИНИЦИАЛИЗАЦИЯ

После того, как мы проинициализировали переменные, мы можем их использовать (выводить в консоль, вычислять на их базе значения других переменных):



```
1  ▶ public class Main {  
2  ▶  ▶ public static void main(String[] args) {  
3      int price = 150;  
4      int count = 5;  
5      int total = price * count;  
6      System.out.println(total);  
7  ▶  }  
8  }
```

ИСПОЛЬЗОВАНИЕ ПЕРЕМЕННЫХ

Обратите внимание на эту строку:

```
int total = price * count;
```

Она работает следующим образом:

1. Вместо имени `price` подставляется то значение, которое сейчас присвоено этому имени: `150`.
2. Вместо имени `count` подставляется то значение, которое сейчас присвоено этому имени: `5`.
3. Вычисляется результат умножения `150 * 5`.
4. В итоге приходим к следующему: `int total = 750`.

Т.е. нужно запомнить, что когда вы встречаете оператор `=`, то сначала «вычисляется» (т.е. фактически подставляются значения) всего, что справа от `=`, после чего вся строка сводится к простому присваиванию.



ОПЕРАТОРЫ

ОПЕРАТОРЫ

Операторы — специальные символы, осуществляющие операции над одним, двумя или тремя операндами и возвращающее результат.

Звучит сложно? На самом деле всё достаточно просто — вспомним обычные арифметические операторы из школы:

- `*` — умножение;
- `/` — деление;
- `%` — остаток от деления;
- `+` — сложение;
- `-` — вычитание.


ОПЕРАТОРЫ

Все эти операторы используются в следующей форме:

```
<type> <result> = <leftOperand> <operator> <rightOperand>;  
// Например:  
int total = price * count;
```

Таким образом:

- `operator` — это `*`;
- `leftOperand` — это `price`;
- `rightOperand` — это `count`.



ПРИМИТИВНЫЕ ТИПЫ ДАННЫХ



ТИПЫ ДАННЫХ

Java — язык со строгой типизацией. Это значит, что создавая переменную, вы должны обязательно указать тип, чтобы в дальнейшем компилятор смог отследить все попытки «положить» в эту переменную «неправильные данные».

Типы в Java делятся на примитивные и ссылочные (будем проходить позже).



ПРИМИТИВНЫЕ ТИПЫ

К примитивным типам относятся:

- `boolean`;
- `byte`, `char`, `int`, `long`;
- `float`, `double`.

ПРИМИТИВНЫЕ ТИПЫ

Q: Почему они называются примитивными?

A: Это просто название для типов, определённых в самом языке в виде ключевых слов.

Самое важное в примитивных типах — они хранят само значение и при использовании присваивания хранимое значение копируется:

```
int width = 10;
int length = width;
// в length тоже 10, но оно "скопировалось"
// т.е. теперь не имеет никакого отношения к тому, что было в width
// с ссылочными типами будет по-другому
```



boolean

`boolean` — самый простой тип данных, может принимать всего два predetermined значения:

- `true` — логическая истина
- `false` — логическая ложь

Используется он тогда, когда необходимо изменить логику программы в зависимости от некоторого условия — и ответ на это условие должен быть однозначный, либо да (`true`), либо нет (`false`).

boolean

Например, при поиске авиабилета вы выбираете: нужен билет в обратную сторону (`true`) или нет (`false`):

ФЕВРАЛЬ 2020						
ПН	ВТ	СР	ЧТ	ПТ	СБ	ВС
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	

Обратный билет не нужен

Скриншот с сайта aviasales.ru.

ЦЕЛОЧИСЛЕННЫЕ ТИПЫ

Хоть все следующие типы и относятся к целочисленным (т.е. могут хранить только целые числа), у ряда из них есть специальное предназначение:

- `byte` — 1 байт, от -128 до 127 (-2^7 до 2^7);
- `char` — 2 байта, от 0 до 65 535 ($2^{16} - 1$);
- `short` — 2 байта, от -32 768 до 32 767 (-2^{15} до 2^{15});
- `int` — 4 байта, от -2 147 483 648 до 2 147 483 647 (-2^{32} до 2^{32});
- `long` — 8 байт, от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 (-2^{64} до 2^{64}).



ЦЕЛОЧИСЛЕННЫЕ ТИПЫ

`byte` чаще всего используется для работы с байтовым представлением (копирование файлов, передача данных по сети).

`short` используется крайне редко (только при необходимости экономии памяти).

`char` для хранения символов (кодировка Unicode).

Поэтому для работы с целыми числами остаются `int` и `long`.

ЦЕЛОЧИСЛЕННЫЕ ТИПЫ

Q: но ведь `int` и `long` занимают больше всего памяти? Не проще ли её сразу экономить?

A: это называется преждевременная оптимизация — попытка "оптимизировать" то, что ещё не требует оптимизации.

Лучше потратить время и усилия на создания работоспособного приложения и только потом заниматься оптимизацией (если это потребуется).

Q: зачем нам знать эти границы?

A: поскольку вы уже прошли курс тестирования, вы должны прекрасно знать, что граничные значения порождают целый класс ошибок.

ВЕЩЕСТВЕННЫЕ ТИПЫ

Вещественные типы предназначены для хранения вещественных чисел (чисел с дробной частью, например, 3.14).

К вещественным относят два типа:

- `float` — 4 байта, от $1.40239846 * 10^{-45}$ до $3.40282347 * 10^{38}$;
- `double` — 8 байт, от $4.9406564584124654 * 10^{-324}$ до $1.7976931348623157 * 10^{308}$.

Проще запомнить, что `double` может хранить в два раза больше значащих цифр, чем `float`.



ХРАНЕНИЕ ЧИСЕЛ

Q: а почему для целых и вещественных разные типы? И что за плавающая точка?

A: это связано с тем, что эти типы по-разному хранятся в памяти.

Так, целые числа хранятся позиционно: 123 — это $1 * 10^2 + 2 * 10^1 + 3 * 10^0$.

В компьютере они хранятся так же, только в виде бит + отдельно хранится знак числа (положительное или отрицательное).

ХРАНЕНИЕ ЧИСЕЛ

Вещественные же хранятся по-другому*: 0.014 — это $14 * 10^{-3}$. При этом отдельно хранится 14 и отдельно степень -3.

Благодаря этому можно хранить как очень большие числа (если степень положительная), так и очень маленькие — близкие к нулю (если степень отрицательная), а десятичная точка «плавает» влево или вправо.

И так же, как и с границами для целых чисел нам это будет нужно для того, чтобы выявлять ошибки, связанные с точностью вычислений.

Примечание*: это очень верхнеуровневое описание. Детальнее вы можете посмотреть в [видео-курсе на Foxford](#).



ОТЛАДКА

ОТЛАДКА

Отладка (Debugging) — это возможность по шагам «пройти» нашу программу так, как это делает Java, для того, чтобы понять, как работает наш код.

Отладчик (Debugger) — инструмент, который позволяет выполнить программу в режиме отладки.

Для того, чтобы запустить программу в режиме отладки, необходимо сделать следующее:

1. Создать конфигурацию запуска (Ctrl + Shift + F10).
2. Поставить точку остановки (Ctrl + F8) — точку, в которой отладчик остановит выполнение программы, чтобы продолжить по шагам.

ТОЧКА ОСТАНОВКИ (BREAKPOINT)

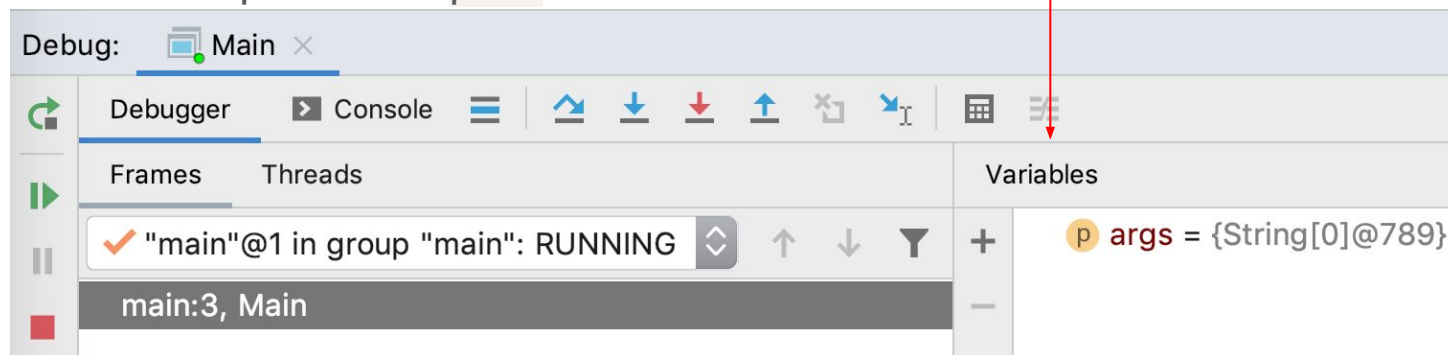
```
1  ▶ public class Main {  
2  ▶   public static void main(String[] args) {  
3  ●   int price = 150;  
4     int count = 5;  
5     int total = price * count;  
6     System.out.println(total);  
7     }  
8  }
```

ОТЛАДКА (DEBUGGING)

После запуска синим будет выделена строка, которая **будет исполнена**

```
1  ▶ public class Main {  
2  ▶  public static void main(String[] args) {  args: {}  
3  ✔  int price = 150;  
4  int count = 5;  
5  int total = price * count;  
6  System.out.println(total);  
7  }  
8  }
```

Обратите внимание: из-за того, что строка ещё не выполнена, в панельке Variables переменной price нет:



ОТЛАДКА (DEBUGGING)

После выполнения этой строки (F8):

```
1  ▶ public class Main {  
2  ▶  public static void main(String[] args) {  args: {}  
3  ✔  int price = 150;  price: 150  
4  int count = 5;  
5  int total = price * count;  
6  System.out.println(total);  
7  }  
8  }
```

В панелике Variables появилась переменная price:





ОТЛАДКА (DEBUGGING)

Демонстрация отладки в IntelliJ IDEA.



ИТОГИ



ИТОГИ

Сегодня мы вплотную занялись программированием и практической работой в IDEA.

Ключевые выводы, которые нужно сделать:

1. Java — строго типизированный язык и у каждого типа есть свои особенности (и узкие места).
2. Отладка (Debugging) позволяет нам “по шагам” пройти программу.
3. Умение работать в IDEA — 50% успеха на начальном этапе.

HOTKEYS

Ключевые клавиатурные сокращения*:

1. `Alt + Insert` в панельке `Project` — генерация файлов.
2. `main + Tab` в редакторе кода — генерация `public static void main....`
3. `Ctrl + Shift + F10` — генерация конфигурации запуска и запуск приложения.
4. `Ctrl + Alt + L` — выравнивание кода.
5. `Ctrl + F8` — установка/снятие точки останова.
6. `Shift + F9` — запуск под отладчиком.
7. `F8` — исполнение следующей “строки” в режиме отладки.

На Mac OS посмотреть клавиатурные сокращения можно через пункт меню `Help -> Keymap Reference`.



ДОМАШНЕЕ ЗАДАНИЕ

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задаем в чате Slack!
- Задачи можно сдавать по частям.
- Зачет по домашней работе проставляется после того, как приняты **все задачи**.



Задавайте вопросы и напишите отзыв о лекции!

ВАСИЛИЙ ДОРОХИН

