Roman Beekeeper

автор тг-канала в t.me/romankh3

09.01.2020 94504 208 + 67

Debug в Intellij IDEA: гайд для новичков

Всем привет, JavaRush сообщество.

Сегодня поговорим о дебаге: что это такое и как дебажить в Intellij IDEA.

Статья рассчитана на людей, у которых есть уже минимальные знания Java Core. Здесь не будет ни фреймворков, ни сложных процессов публикации библиотек. Легкая прогулка. Так что располагайтесь поудобнее — начнем!

Почему Debug тебе необходим

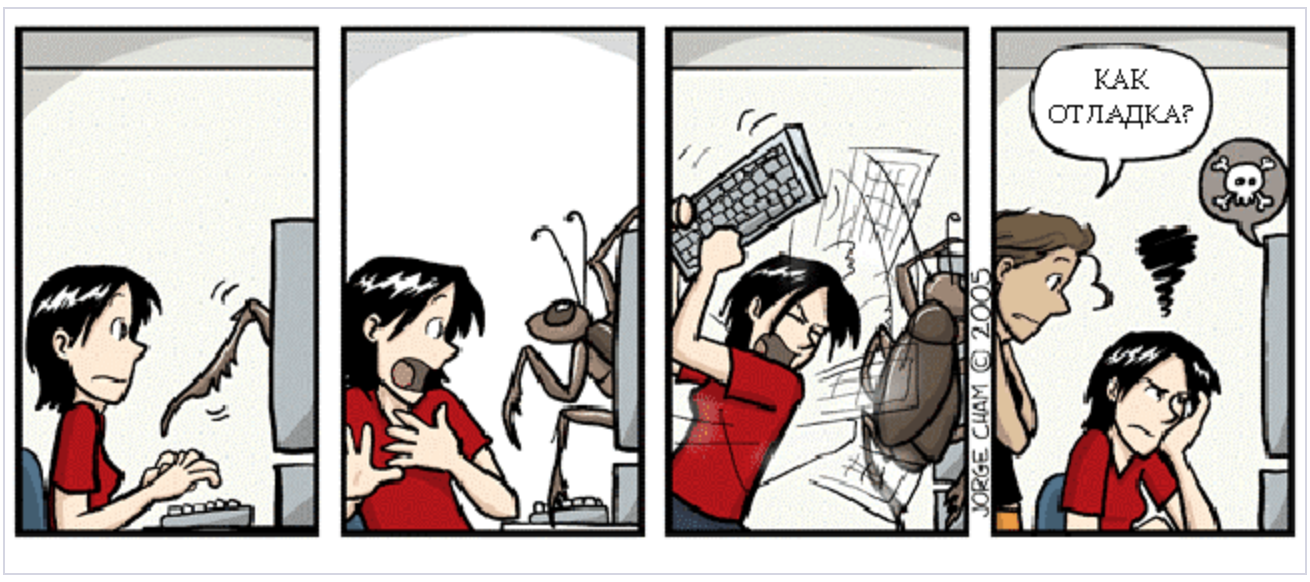
Давайте сразу проясним для себя: кода без багов не бывает… Так устроена жизнь. Поэтому не стоит сразу раскисать и бросать все, если код работает не так, как мы ожидали.

Но что же делать? Конечно, можно наставить `System.out.println` везде, где только можно и потом разгребать вывод в терминале в надежде на то, что получится найти ошибку.

Все-таки можно… и это делают, и делают аккуратно при помощи логирования (можно почитать об этом здесь).

Но если есть возможность запустить на локальной машине код, лучше использовать **Debug**. Сразу хочу заметить, что в этой статье мы будем рассматривать дебаг проекта внутри Intellij IDEA. Если интересно почитать об удаленном дебаге — вот, пожалуйста, статья из нашего ресурса.

Что такое Debug



Debug — это процесс отладки (проверки) кода, когда в процессе его выполнения можно остановиться в обозначенном месте и посмотреть за ходом выполнения. Понять, в каком состоянии находится программа в определенном месте.

Это точно так же, как если бы можно было остановить жизнь и посмотреть на всё со стороны. Круто, правда?

Наша задача состоит в том, чтобы быстро и просто научиться проводить отладку приложений при помощи всеми нами любимой среды разработки Intellij IDEA.

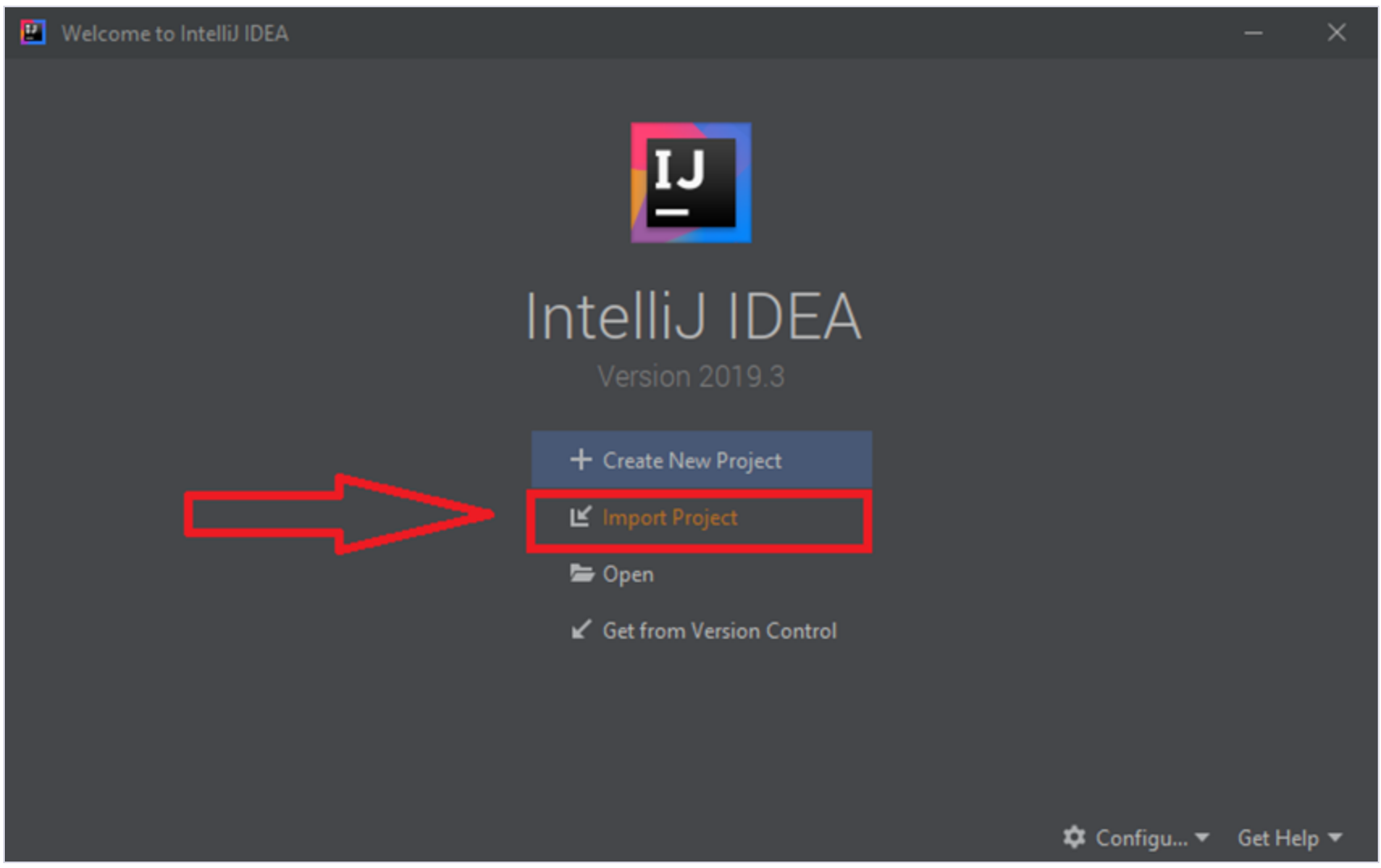
Что нужно для начала отладки

Даю бесплатный совет: пока читаете статью, проделывайте все то, что здесь будет описано, благо есть все для этого.

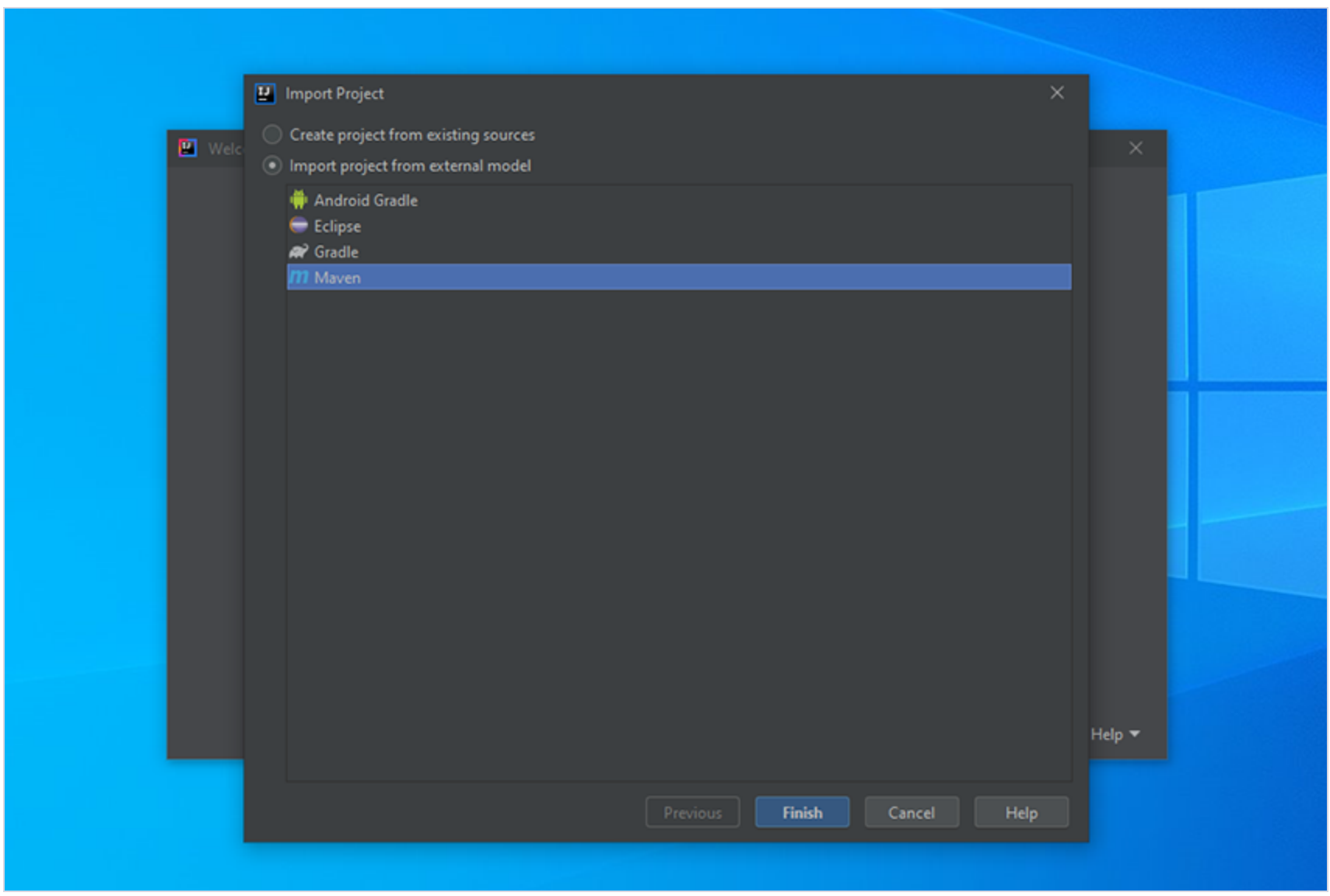
Что нужно:

1. Среда разработки Intellij IDEA версии 2019.3.1 и выше. На случай, если у кого-то её нет, вот ссылка, где можно скачать. Скачивайте Community Edition, так как я буду использовать именно ее.
2. Клонировать проект из GitHub и импортировать его через IDEA.

Открываем IDEA:



Выбираем проект **debug-presentation**, нажимаем **OK** и получаем:



Оставляем **import project from external sources, Maven** и нажимаем **Finish**.

Импортировав проект, можем описать процесс на живом примере.

Немного теории… обещаю :D

Чтобы начать мало-мальски дебажить, нужно понять, что такое **breakPoint** и разобраться в нескольких горячих клавишах, которые нужны для начала.

BreakPoint — это специальный маркер, который отображает место или состояние, на котором нужно остановить приложение.

Поставить breakpoint можно либо нажав левой кнопкой мыши на левую боковую панель, либо кликнув курсором по месту кода и нажав **Ctrl + F8**.

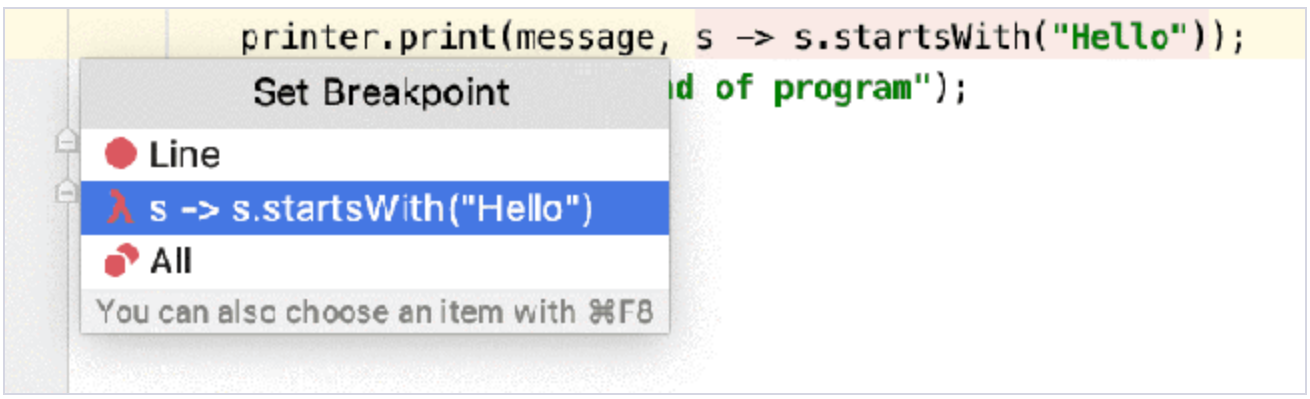
Breakpoint’ы бывают трех видов: метка на строку, метка на переменную и метка на метод.

Выглядит это так:

- На строку:

```
13  class Printer {
14      private boolean ready;
15      void print() {
16          System.out.println("Printing to console");
17      }
```

если в выражении есть лямбда, то IDEA предлагает вам выбор — поставить на всю линию или конкретно в лямбда выражение:



- На метод:

```
13  class Printer {
14      private boolean ready;
15      void print() {
16          System.out.println("Printing to console");
17      }
```

- На класс

```
13  class Printer {
14      private boolean ready;
15      void print() {
16          System.out.println("Printing to console");
17      }
```

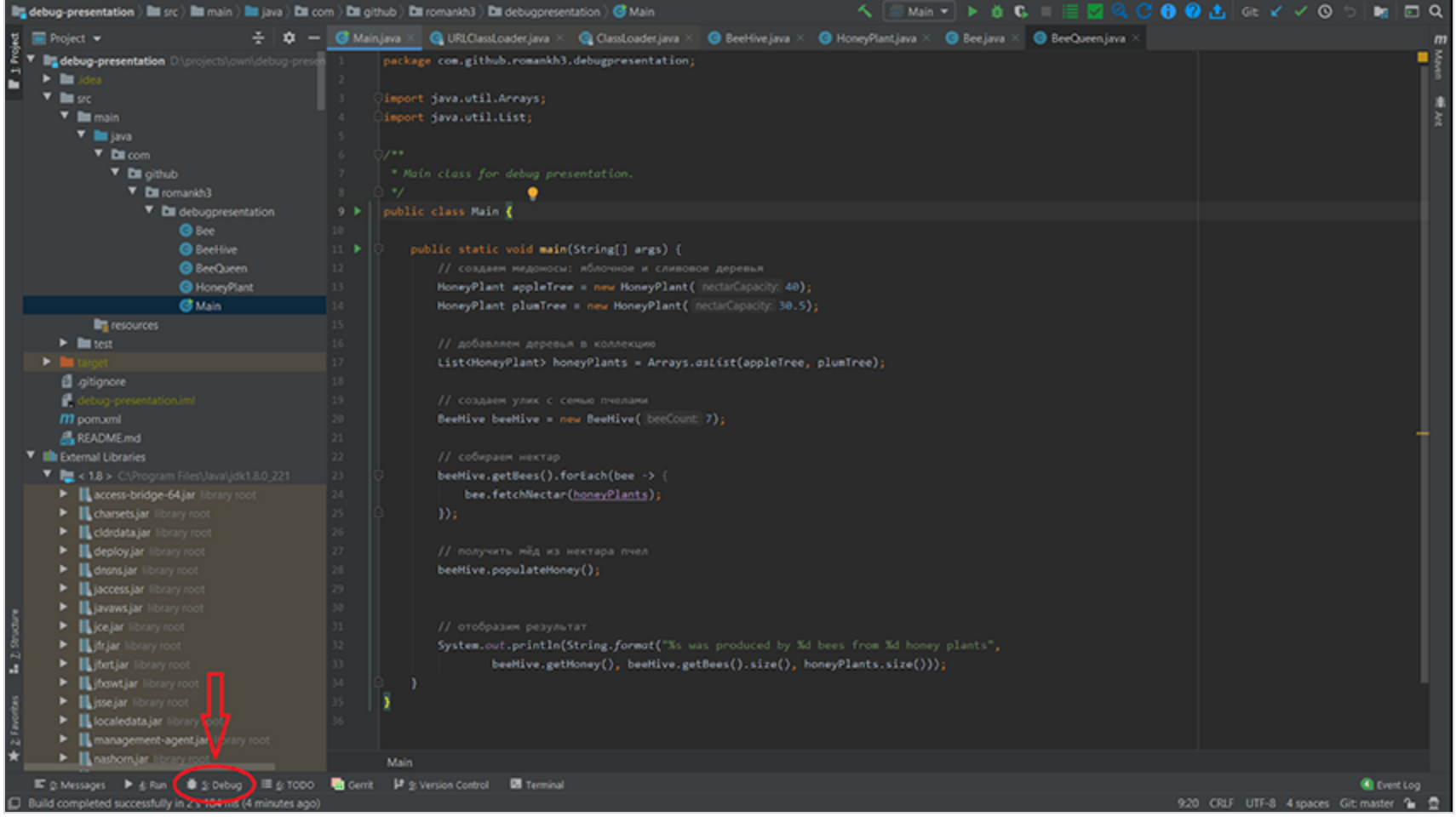
Breakpoint’ы можно удалить, выполнив те же действия, что и при их добавлении.

Бывают ситуации, когда нужно сделать их неактивными (замьютить). Для этого в Debug секции, можно найти значок

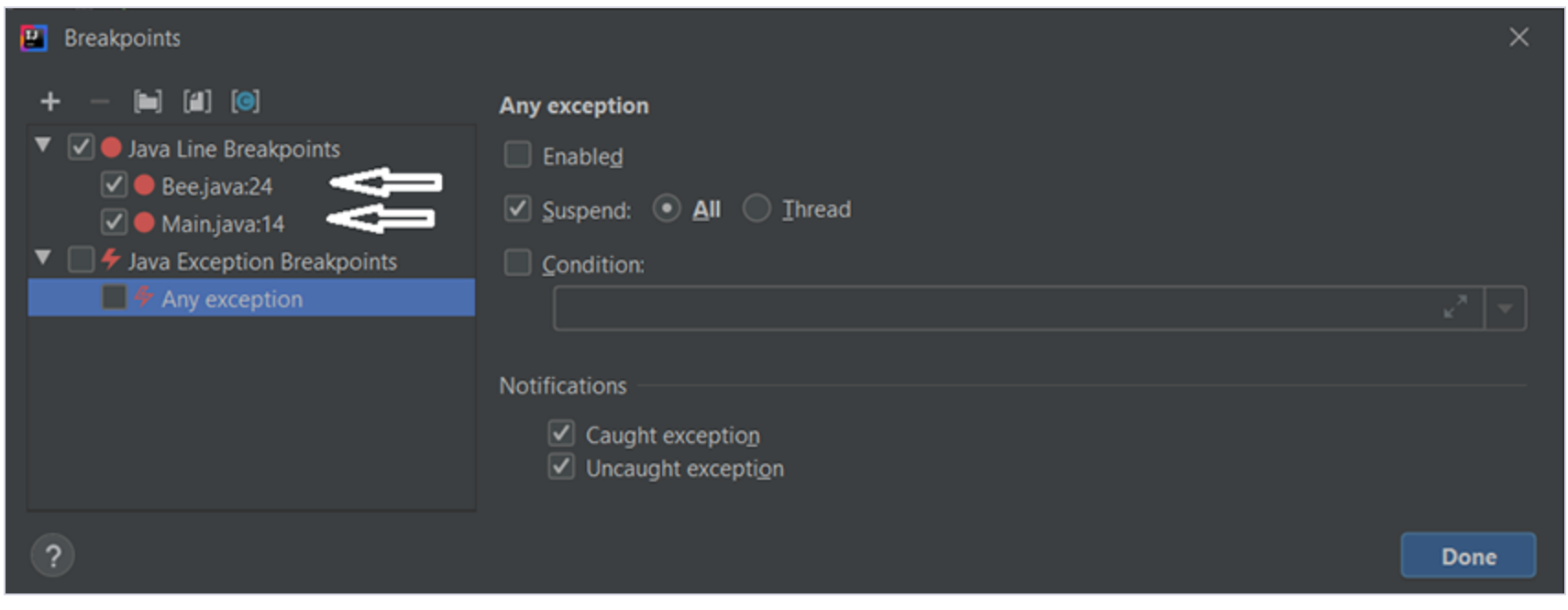
, который сделает все breakpoint’ы неактивными.

Чтобы посмотреть, какие уже выставленные breakpoint’ы, можно или зайти в Debug в левом нижнем углу и найти иконку

, или нажать **Ctrl+Shift+F8**:



Когда зайдем в список breakpoint’ов, увидим:



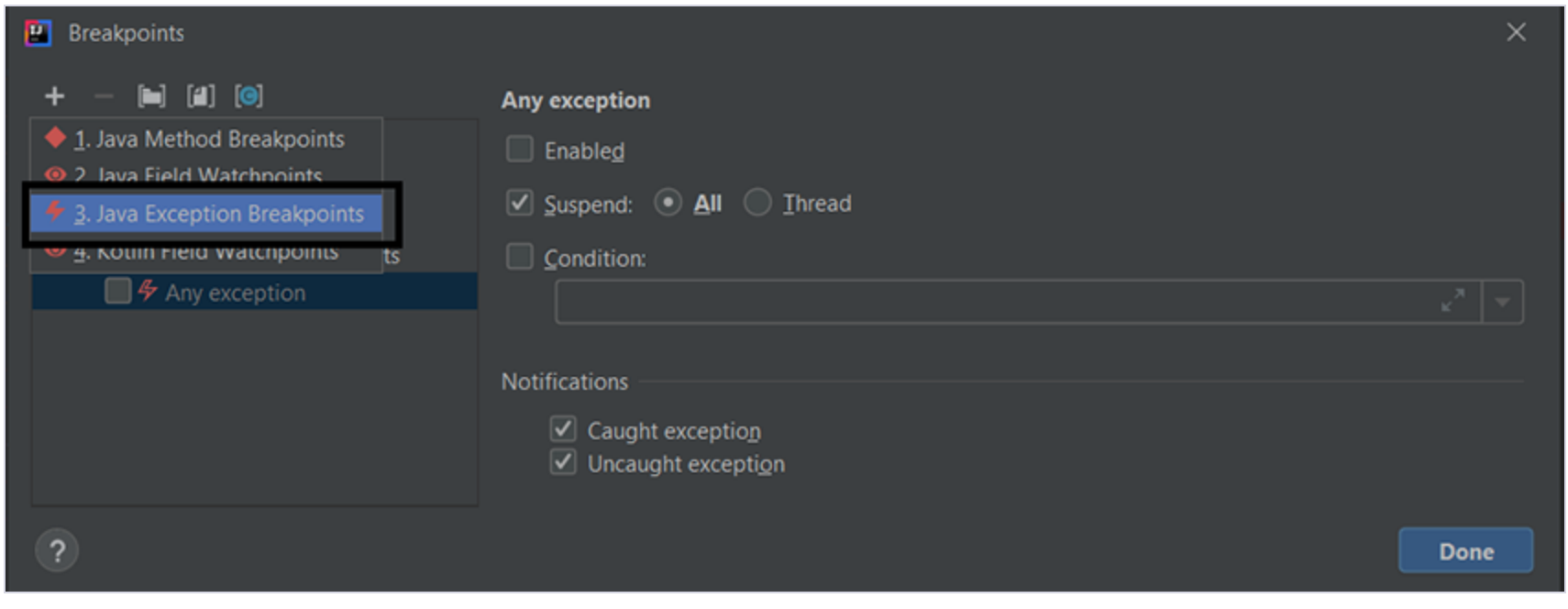
Здесь есть два preakpoint’a:

- Bee.java:24 — в классе Bee на 24-й строке
- Main.java:14 — в классе Main на 14-й строке

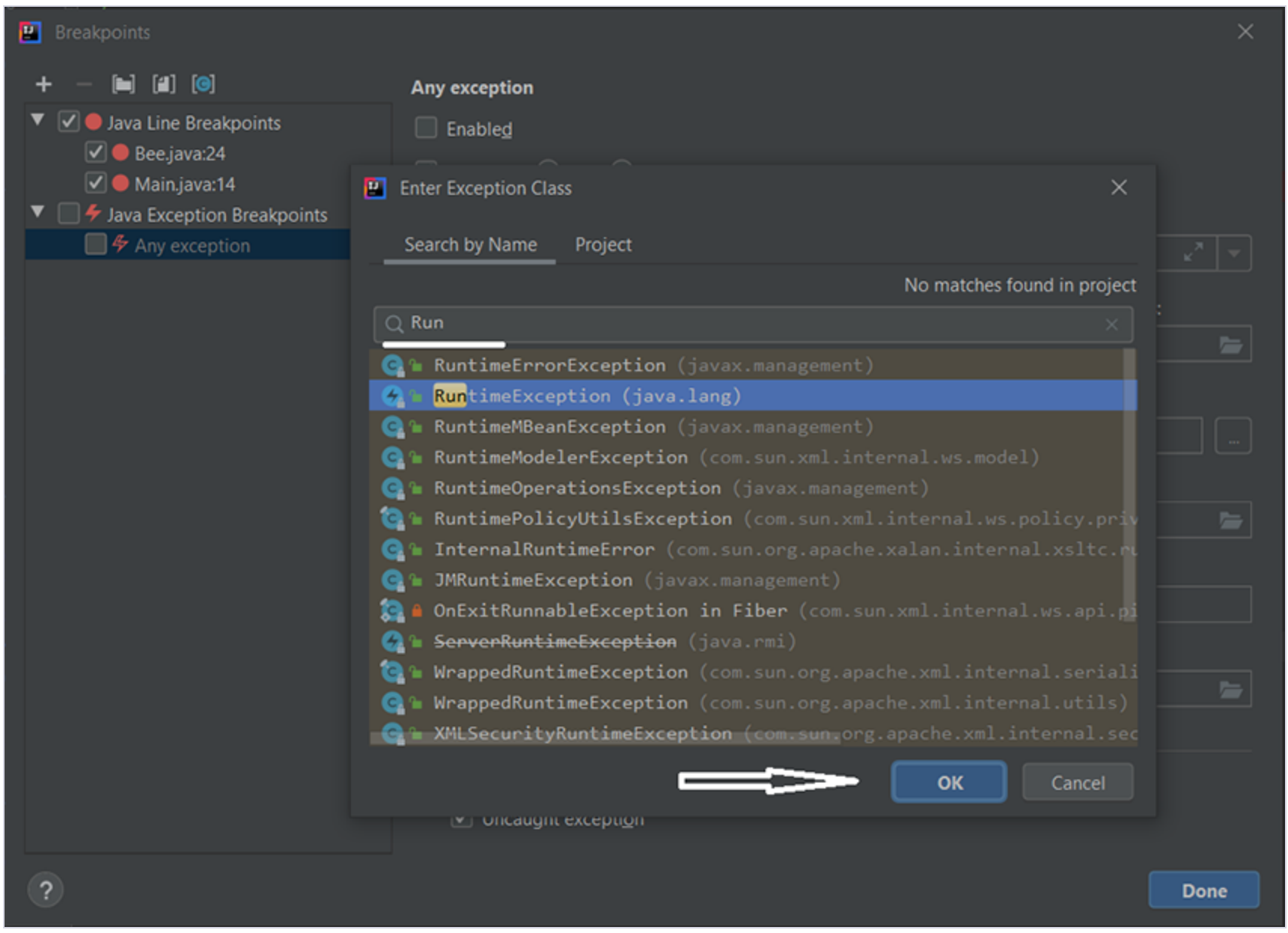
Хочу заметить, что при клонировании проекта к себе не будет этих BreakPoint’ов: их нужно выставить самостоятельно!

Также есть секция **Java Exception Breakpoints**. Очень полезная вещь. При помощи ее можно добавить неявный breakpoint, чтобы программа останавливалась перед выбрасыванием любого исключения или какого-то конкретного.

Добавим для RuntimeException неявный breakpoint. Делается это легко: в верхнем левом углу есть плюсик “+”. Нажимаем на него и выбираем **Java Exceptions Breakpoints**:



В появившемся окне пишем имя исключения, которое нужно добавить, выбираем из предложенного списка и нажимаем **OK**:



На этом ликбез заканчиваем и переходим к практике.

Поехали, будем врываться в дебри дебага

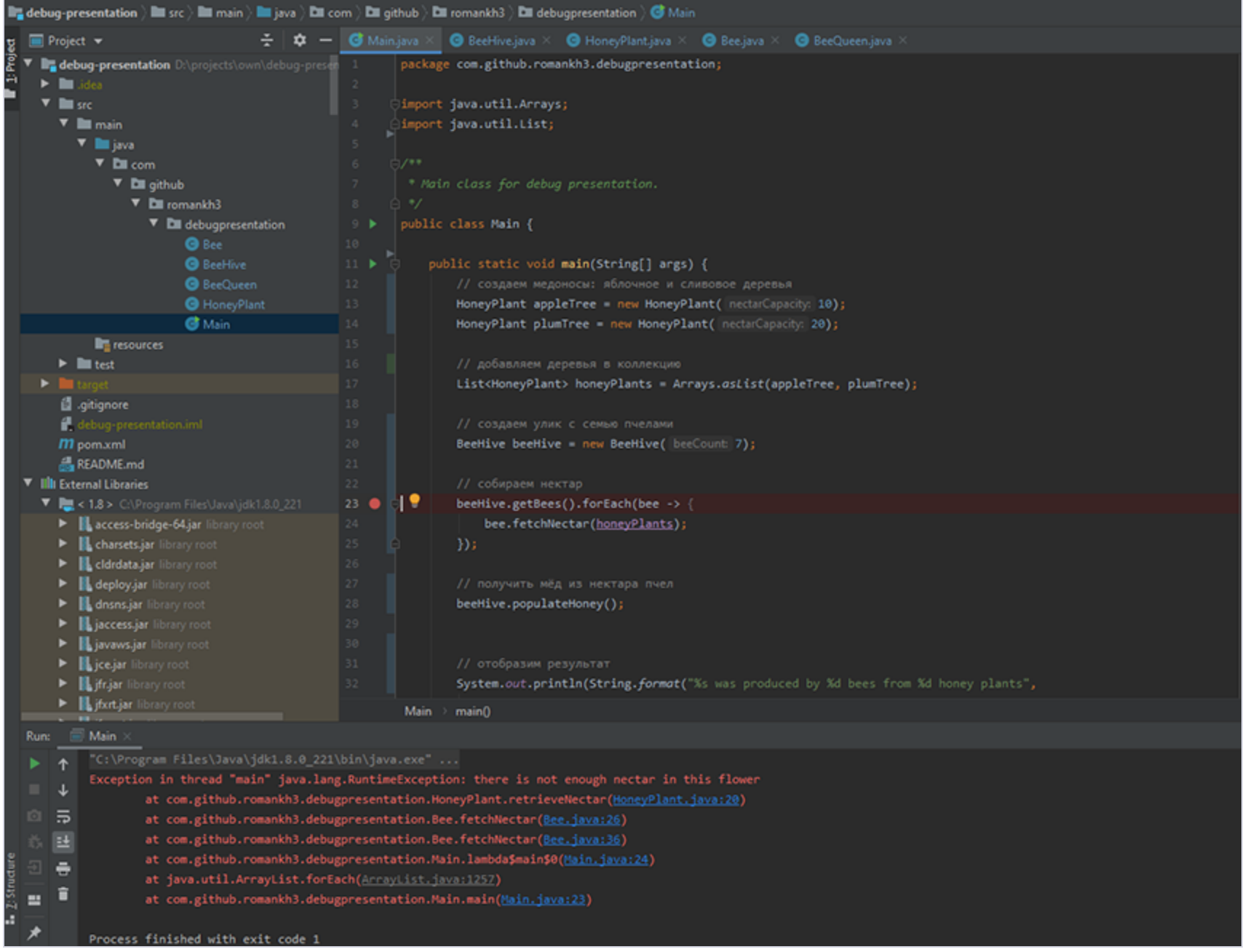
Так как я потомственный пчеловод, для презентации отладки создал проект, который описывает процесс сбора нектара пчелами, переработки нектара в мед и получение меда из улья.

На основе ~~документации~~ README файла, который лежит в корне проекта, читаем: **ожидаемое поведение** — со всех цветков, с которых собирают нектар (как **double** значение), будет собрано количество меда, равное половине собранного нектара.

В проекте есть такие классы:

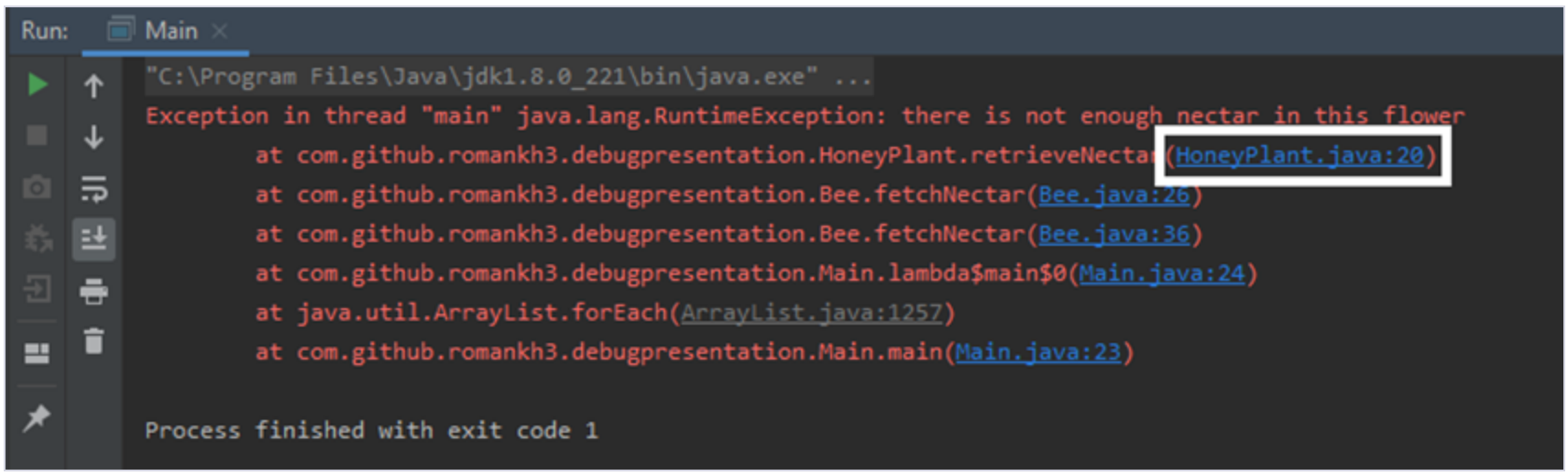
- Bee — обычная рабочая пчела;
- BeeQueen — пчелиная матка;
- BeeHive — улей;
- HoneyPlant — медонос, с которого собирают мед;
- Main — где находится `public static void main()` метод в котором стартует проект.

Если запустить метод `main()`, то окажется, что мало того, что не считается количество меда, так еще и выпадает ошибка…



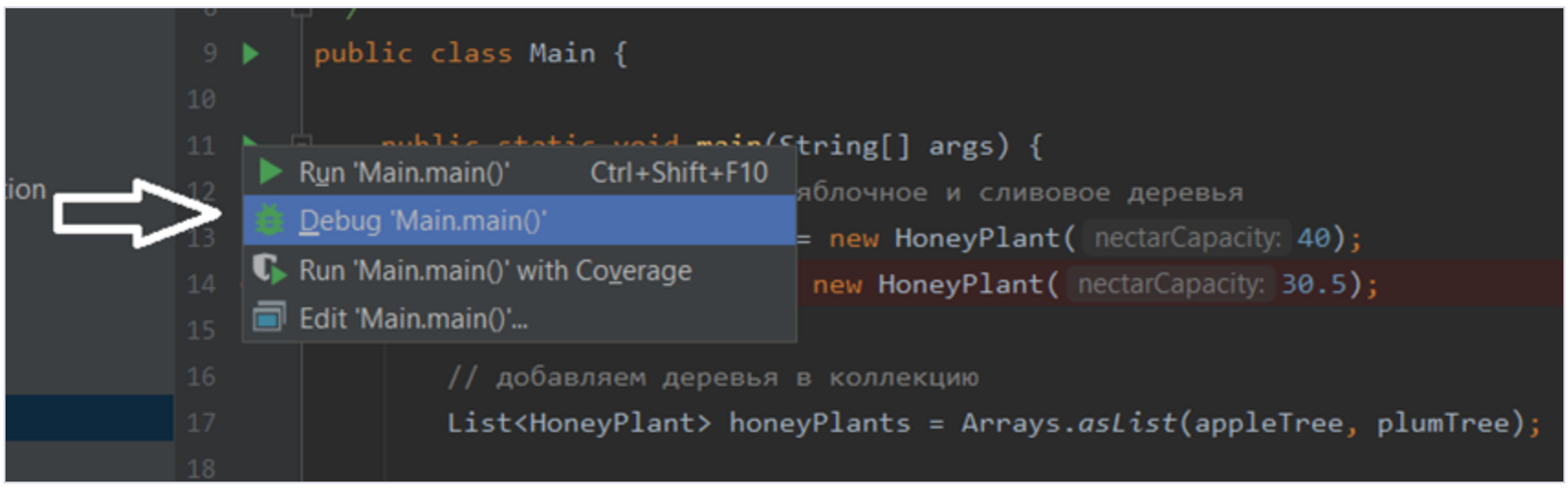
Нужно посмотреть, что же там не так.

Из стек трейса в нижнем правом углу, можем увидеть, что в `HoneyPlant.java:20`, выбрасывается исключение RuntimeException:

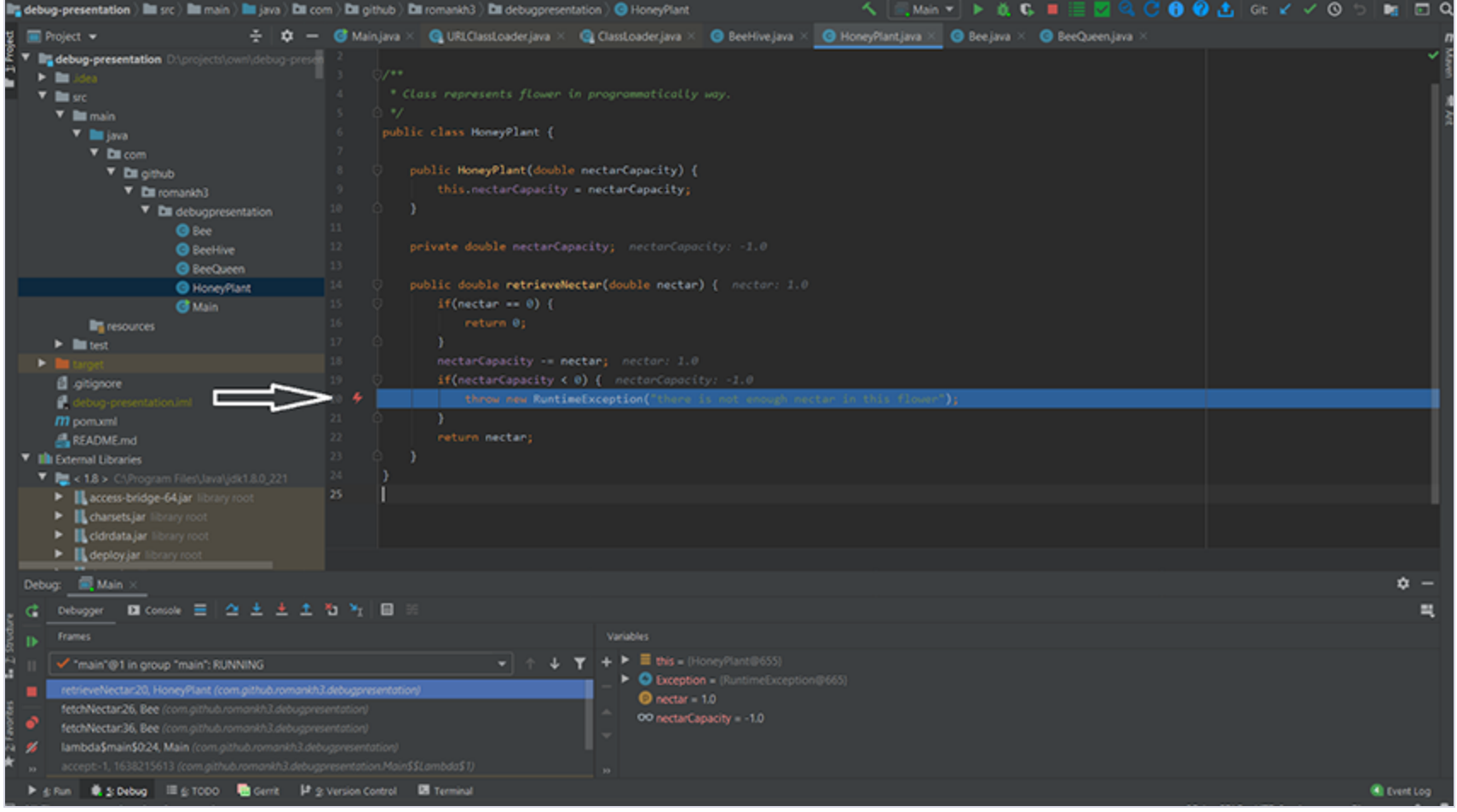


Как раз наш случай: есть RuntimeException, добавим поиск такого исключения, как было описано выше, и запустим `main()` метод в дебаг режиме.

Для этого нажмем на зеленую стрелку-треугольник в Intellij IDEA перед методом `main()`:



и получим остановленную программу в момент перед тем, как сработает исключение с таким значком



Чтоб получить полную информацию, нужно посмотреть в секцию Debug. В ней есть **Variables**, где показаны все переменные, доступные в этой части приложения:

- nectar = 1.0;
- nectarCapacity = -1.0.

Исключение выбрасывается справедливо, так как значение количества нектара, которое есть в медоносе, не может быть отрицательным. Но почему же так происходит? Ведь есть же проверка, что если нектар закончился, то возвращается нулевое значение в строках 15-17:

```
1  if ( nectar == 0 ) {
2          return 0;
3  }
```

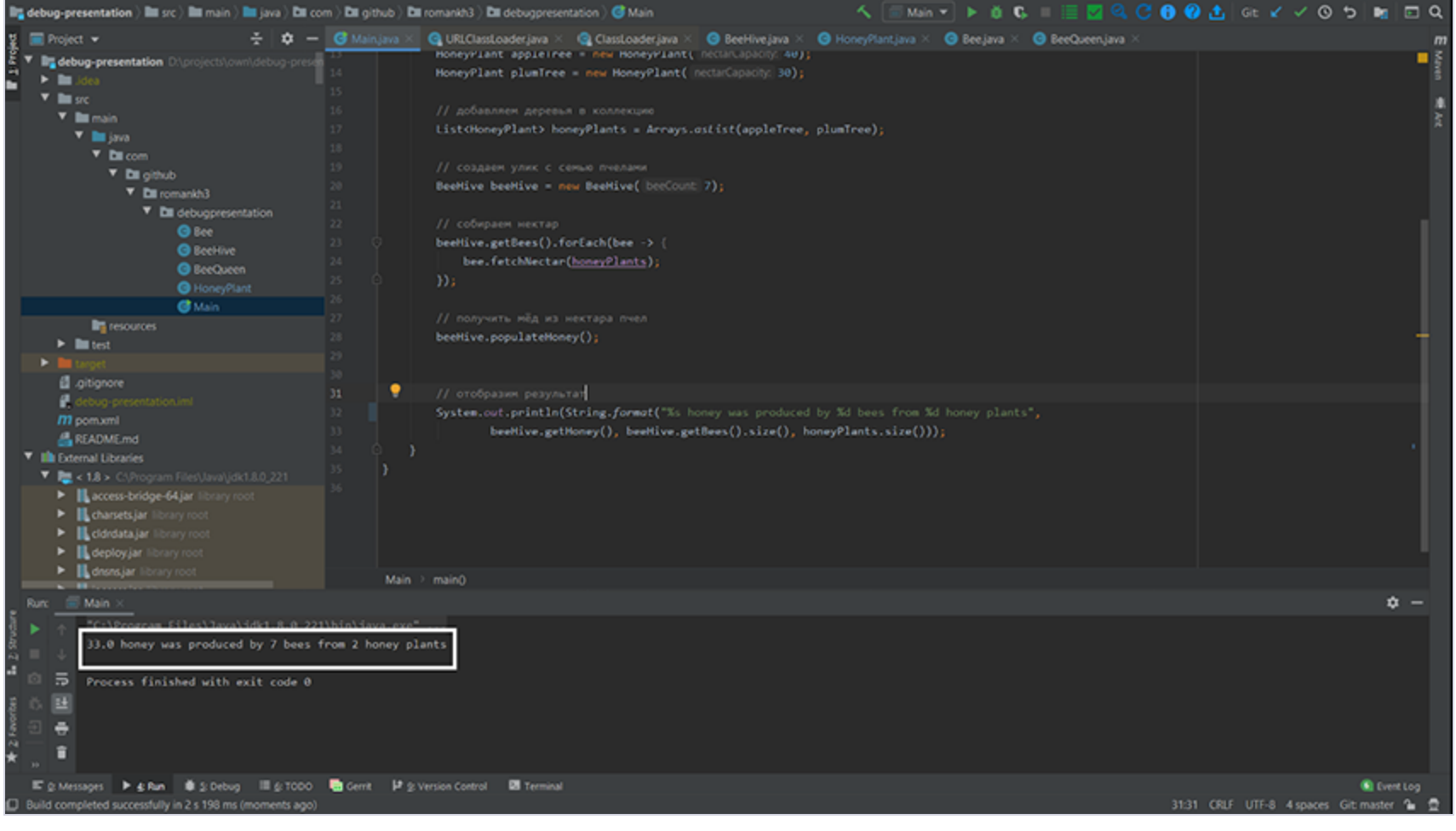
Но загвоздка в том, что проверяет он не ту переменную… и это ошибка в коде. Вместо того, чтобы проверять значение нектара в цветке, который лежит в переменной **nectarCapacity**, программа проверяет значение **nectar**, которое приходит в метод и является тем количеством, которое хотят взять у нектара.

Вот же он, первый баг!

Поэтому ставим правильно и получаем выражение:

```
1  if ( nectarCapacity == 0) {
2          return 0;
3  }
```

Далее, запускаем `main()` метод в обычном режиме `(Run `Main.main()`)` и ошибки больше нет, программа отработала:



Приложение отработало и выдало ответ:

“33.0 honey was produced by 7 bees from 2 honey plants”

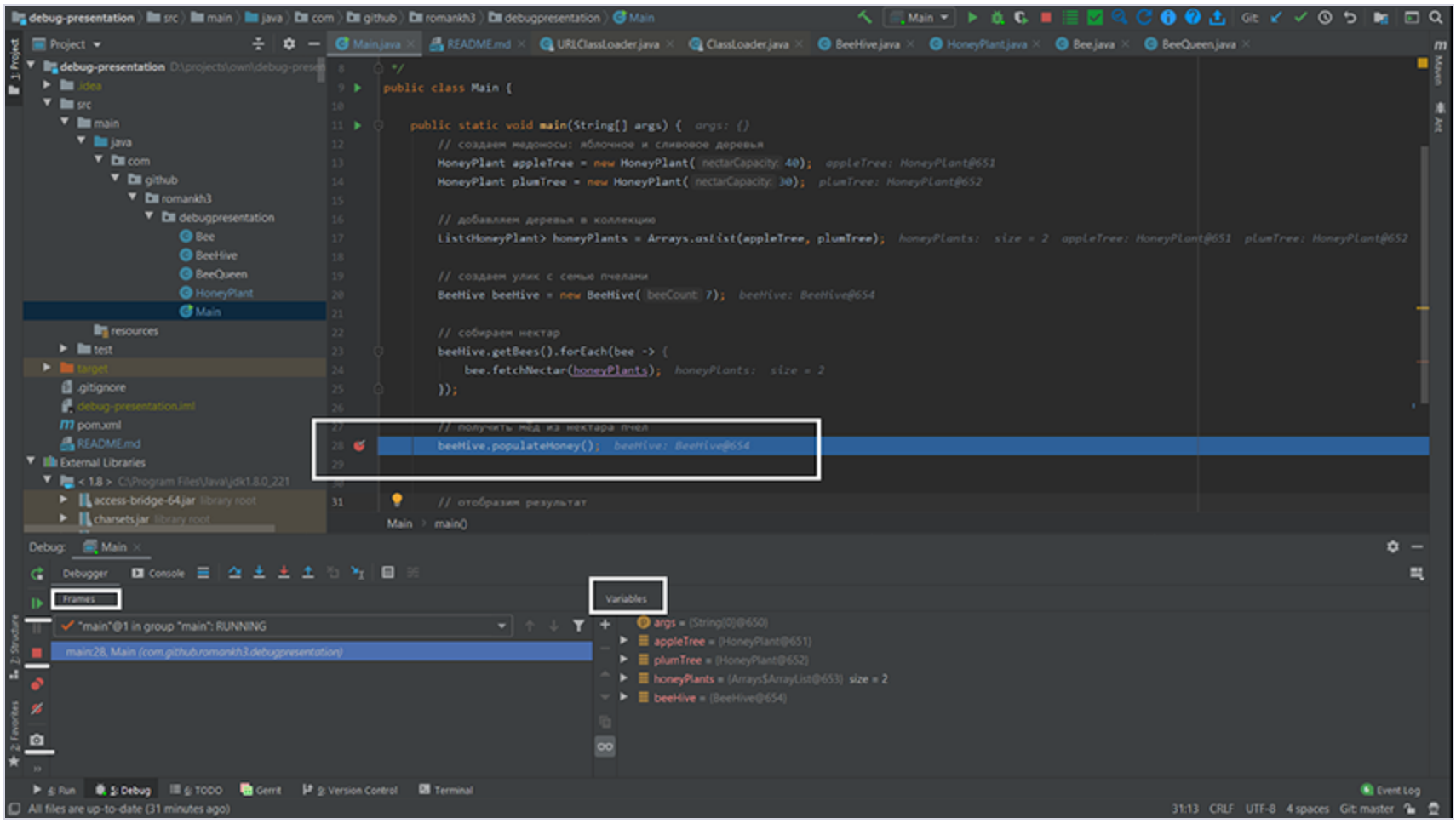
Все бы хорошо, но ответ неправильный… Все потому, что в ~~документации~~ README файле написано, что нектар переходит в мед с пропорцией 2 к 1:

```
## Documentation
Presentation based on honey getting process.

**Note**: 1 honey point = 2 nectar points
```

Из главного метода видно, что есть два медоноса, по 30 и 40 единиц нектара соответственно, поэтому в итоге должно получиться 35 единиц мёда. А пишет, что 33. Куда же делись еще две единицы?... Сейчас узнаем!

Для этого нужно поставим breakpoint в методе `Main.main()` на строке №28, где выполняется `beeHive.populateHoney()` и запускаем `main` метод в режиме Debug:



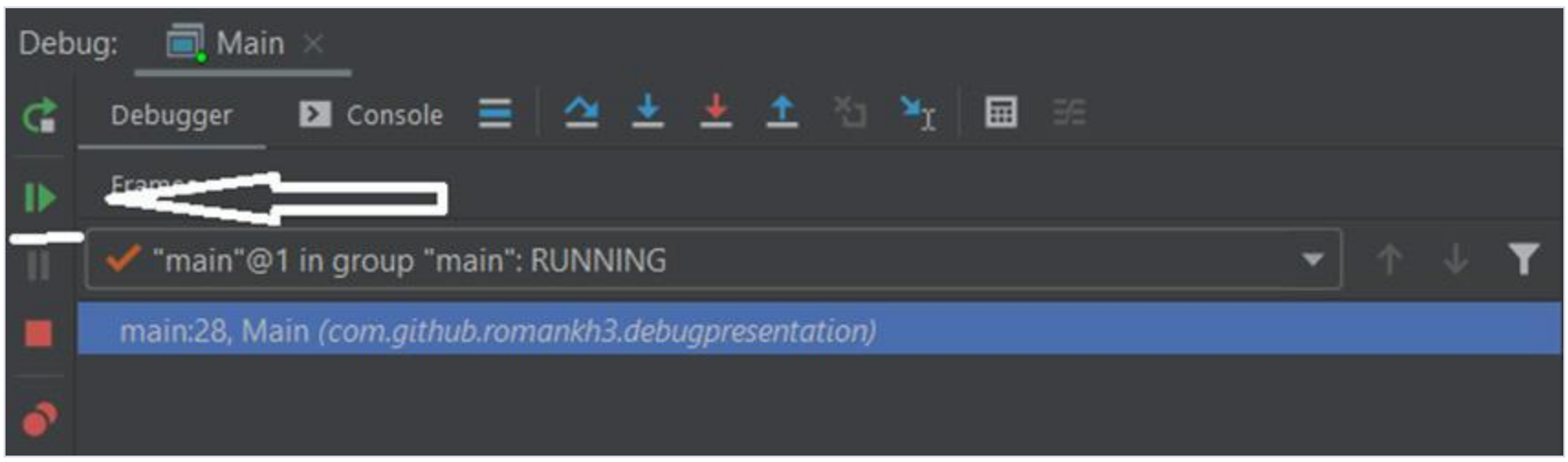
Вот этот момент рассмотрим подробнее. Программа остановилась перед выполнением 28-й строки.

В нижней части видим Debug секцию, в которой описана вся информация по запущенному приложению.

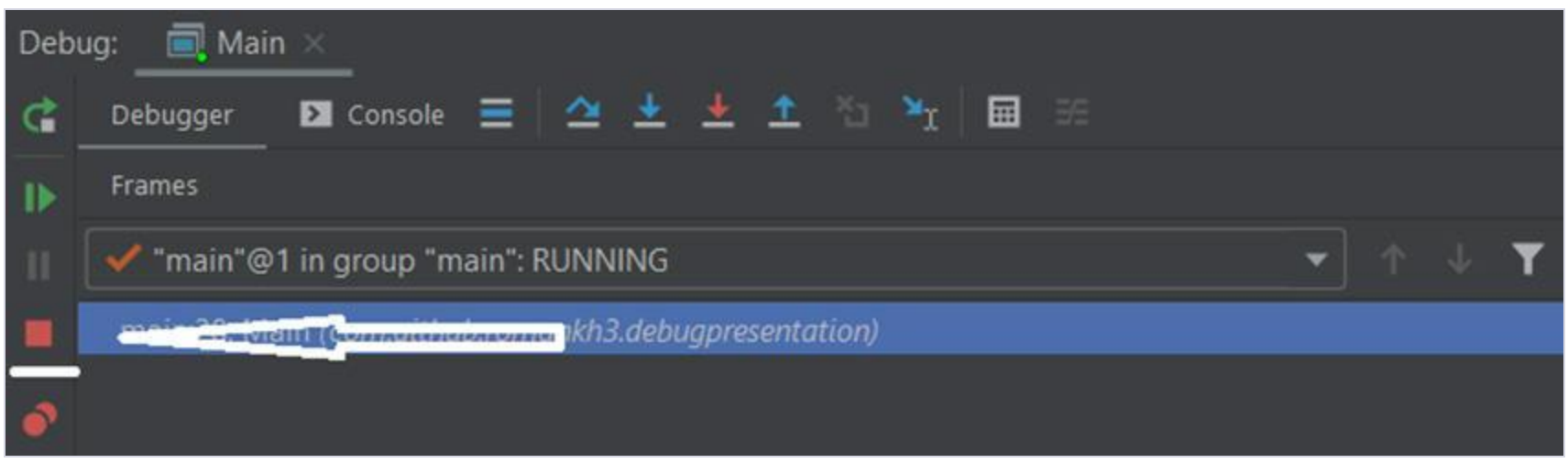
В части Variables, как уже было сказано, есть все переменные и объекты, которые доступны из этой части приложения.

В части Frames показаны шаги, которые проходит приложение, можно посмотреть на предыдущий шаг и получить все локальные данные.

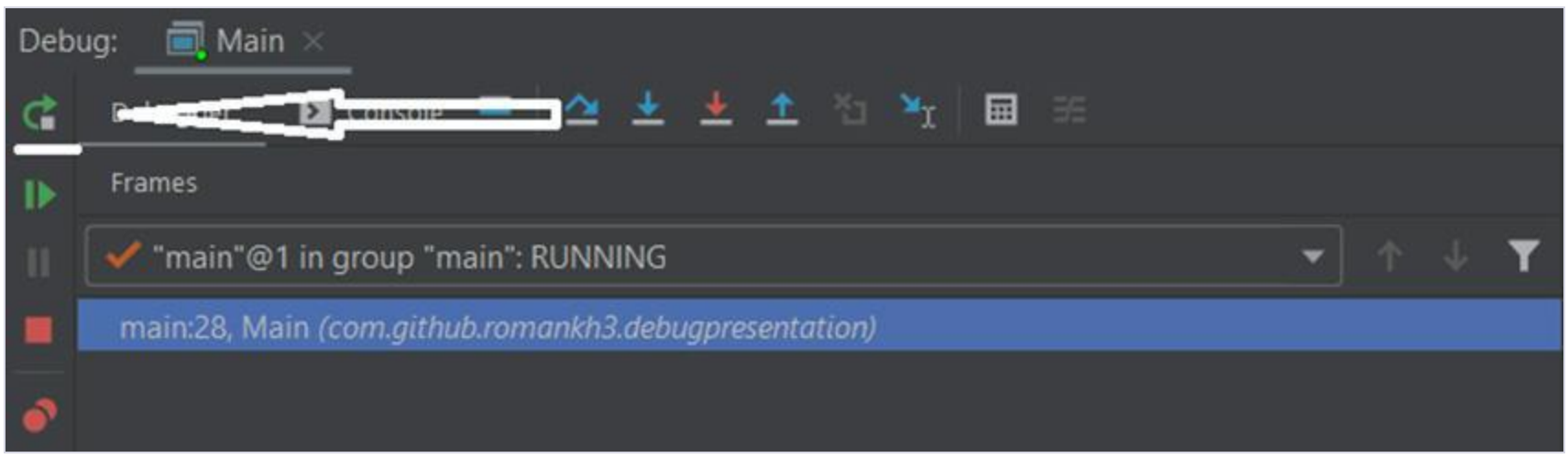
Чтобы программа продолжила работу, можно нажать **F9** или зеленую иконку, как показано ниже:



Чтобы остановить программу, нужно нажать на красный квадрат:



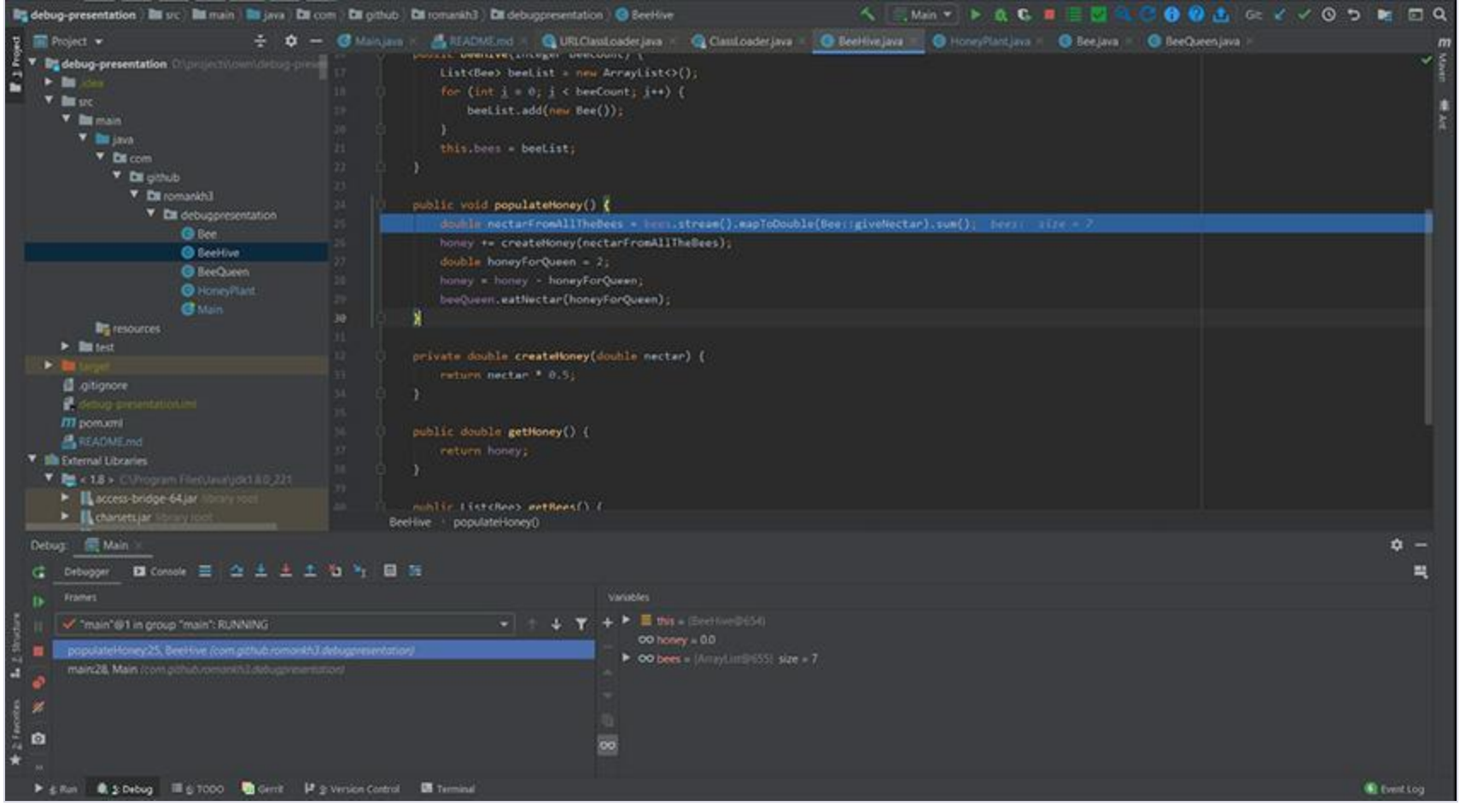
Чтобы перезапустить приложение в режиме дебага, нужно нажать на стрелку:



Далее, чтобы проходить пошагово по работе приложения, можно использовать две клавиши:

- F8 — идти по участку кода и не заходить во внутренние методы;
- F7 — идти по участку кода и заходить во внутренние методы.

Поэтому нам, чтобы зайти в работу метода `beeHive.populateHoney()`, нужно нажать F7, и мы перейдем далее:



Далее, проходим в режиме дебага используя **F8** по этому методу до конца и опишем, что происходит в этом методе:

- 25-я строка — используется Stream API, чтобы собрать мед со всех пчел;
- 26-я строка — мед суммируется уже с существующим;
- 27-я строка — выделяется 2 единицы меда для матки;
- 28-я строка — эти две единицы удаляются из общего количества меда;
- 29-я строка — матка съедает этот мед.

Вот куда делись эти две единицы, ура! После общения с бизнес-аналитиком приходим к выводу, что ~~документация~~ README файл содержит ошибку, и его нужно будет обновить.

Обновим README файл:

```
## Documentation
Presentation based on honey getting process.

**Note**:
*  1 honey point = 2 nectar points
*  2 honey point queen bee eats every time when beehive populates the honey.
```

И все:, все найденные баги починены, можем спокойно продолжать с умным видом пить кофе и читать статейки на ~~хабре~~ JavaRush :)

Подведем итог

За эту статью мы разобрались, что:

- работы без ошибок не бывает и дебаг — это отличный способ их решить;
- что такое breakpoint и какой он бывает;
- как настроить exception breakpoint;
- как проводить навигацию в режиме дебага.

Статья для почитать

- Проект, что используется в статье
- IntelliJ IDEA и Debug: не дайвинг, но снорклинг
- Горькая правда о программировании...
- Официальная документация
- Типы breakpoint'ов. Официальна документация

Смотрите также мои другие статьи:

- Как тестовое задание на собеседование превратилось в open-source библиотеку
- Создание системы мониторинга цен на авиабилеты: пошаговое руководство [Часть 1]
- Гайд по созданию клиента для Skyscanner API и его публикации в jCenter и Maven Central [Часть 1]
- Логирование: что, как, где и чем?

Roman Beekeeper

автор тг-канала в t.me/romankh3

Что то пошло не так, с самого начала. Простите как быть если нет вкладки "Import Project" ?

Ответить 0

Evgen Уровень 6, Речица, Belarus 9 октября, 20:58

Нажимай Get from VCS, во вкладе repository url вставляй ссылку скопированную с гит хаба

Ответить 0

Flexo Уровень 6, Ukraine 19 сентября, 11:48

Пишет java: package lombok does not exist и java: cannot find symbol
symbol: class Data

Ответить 0

Hotric Уровень 9 5 сентября, 05:01

Интересная статья

Ответить 0

Кирилл Уровень 6, Киров, Russian Federation 2 сентября, 14:44

В тексте сказано:
Main.java:14 — в классе Main на 14-й строке
Однако на скриншоте после первого запуска объекта main() breakpoint стоит не на 14 строке а на 23.

Ответить 0

Burati nator Уровень 6, Москва, Russian Federation 31 августа, 18:47

Интересно. Все сделал. Мало что понял. Надеюсь, что все так и задумано. Что план потихоньку заставлять привыкать к интерфейсу, чтобы потом было легче. Иначе я очень-очень тупой =)

Ответить +5

Алексей Уровень 16, Москва, Russian Federation 2 августа, 08:08

Спасибо. Интересная статья. Но почему я не могу сохранить изменения в Redme? Строчку дописываю, а на предпросмотре и в итоговом файле это изменение не появляется?

Ответить 0

Qunjavi qunari в **Java Inquisition** 7 июля, 20:32

Я решил заморить королеву пчелку и все таки выжать эти две капли меда. В следующий раз будут думать прежде чем писать не верную документацию. Так им.

Ответить +6

Roman Beekeeper автор тг-канала в **t.me/romankh3** 9 июля, 20:54

Браво

Ответить +1

Andrey Panov Уровень 6, Israel 26 июня, 21:37

Если у вас не появляется эта молния , то из Breakpoints надо убрать Bee и Main и оставить только RuntimeException .

Ответить +5

G L Уровень 7, Киев, Ukraine 6 июля, 14:29

Интересно, почему так? Ведь в статье указано, что их нужно выставить. Но в итоге получилось, как вы указали

Ответить 0

Max Уровень 18, Poland 14 июля, 13:43

Спасибо, помогло.

Ответить 0

RomaSky QA Automation Engineer 22 июня, 16:07

Для тех у кого так и не всплывает иконка молнии:
1. Перед дебагом жмем ctrl+shift+f8
2. Жмем +
3. в строку копипастим "RuntimeException" (без кавычек)
4. Жмем ОК, далее Done

5. В методе Main жмем на shift+f9 (или debug "main")
6. Готово

Ответить

0

Показать еще комментарии