Debug в IntelliJ IDEA

Написать код - пол дела. Нужно его ещё заставить работать правильно. В этом нам очень сильно помогают IDE и средства отладки.

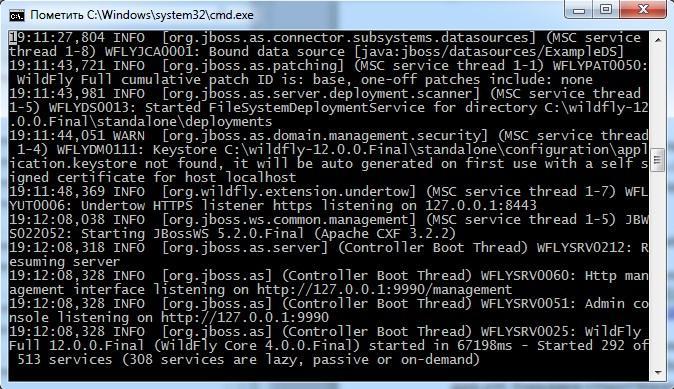
На примере IntelliJ IDEA предлагаю познакомиться с тем, как мы можем узнать, что же происходит с нашим кодом, когда он работает. Debug тема обширная, поэтому данный обзор на глубокое погружение, словно дайвер, не предлагает. Но надеюсь снорклинг точно )

**Вступление**

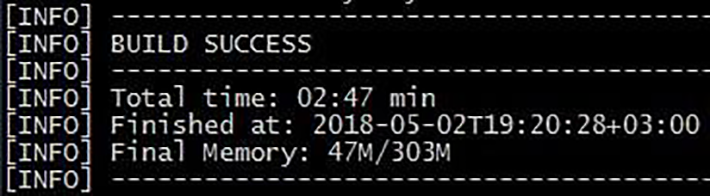
Часть написания кода – это его отладка, дебаг (англ. debug). А если в задачи входит поддержка кода – отладки будет ещё больше. Ну и кроме того, при помощи отладки можно исследовать работу используемых библиотек и фрэймворков так глубоко, как только сможете погрузиться в дебри чужого кода. Для нашего погружения нам потребуется:

* Среда разработки: Бесплатная [IntelliJ Idea Community Edition](https://www.jetbrains.com/idea/download" \t "_blank)
* Установленный [Maven](https://maven.apache.org/" \t "_blank)
* Сервер приложений [WildFly 12.0.0.Final](http://wildfly.org/downloads/" \t "_blank)
* Исходный код официальных примеров: [Quick Start Source Code](https://github.com/wildfly/quickstart/tree/12.0.0.Final" \t "_blank)

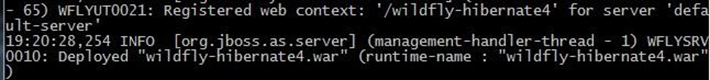
Для начала, распаковываем скачанный архив с Quick Start Source Code. Запускаем IntelliJ Idea и создаём «**New Project from Existing Sources**». Выбираем в подкаталоге **hibernate4** файл **pom.xml**. При импорте указываем «**Import Maven projects automatically**» и завершаем создание проекта оставляя остальные настройки без изменений. Пока импортируется проект разархивируем скачанный сервер приложений WildFly в какой-нибудь каталог. Запускаем сервер при помощи файла bin\standalone.bat (или standalone.sh для \*nix систем). **(!) Важно запуститься с параметром --debug** Ожидаем, когда сервер стартует. Нам напишут, что он started in и укажут время. Выглядеть это будет примерно так:



Далее нужно запустить выбранный нами проект на сервере. Этот процесс описан в небольшой документации, которую можно найти в самом проекте: \hibernate4\README.adoc Как и указано в этой документации, нам нужно в каталоге hibernate4 выполнить команду: mvn clean package wildfly:deploy Ожидаем, когда нам напишут, что сборка выполнена успешно:



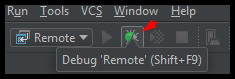
После этого в логе сервера мы можем увидеть, как "задеплоился" новый проект:



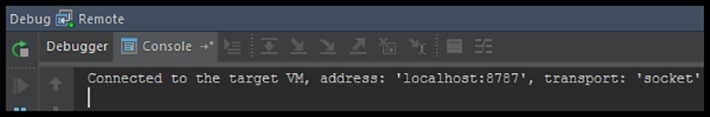
После этого мы переходим на страницу http://localhost:8080/wildfly-hibernate4 и у нас должна отобразиться страница с формой "**Member Registration**". Итак, наша подготовка к экспериментам выполнена и мы можем начинать )) Впереди для наглядности будет много картинок, так что приготовтесь)

**Remote Debug**

Итак, нам нужно настроить Debug режим, чтобы наша IDE управляла исполнение кода на сервере приложений. IntelliJ Idea поставляется в двух вариантах: бесплатный (Community) и платный (Ultimate). Последний можно официально попробовать в виде EAP. В Ultimate версии всё просто - сервер приложений можно запускать сразу из IDE в режиме дебага. А вот в Community версии надо немного действий сделать вручную. Поэтому, рассмотрим случай посложнее, т.е. настройка в Community версии. В Community версии есть некоторые ограничения. В частности, из неё нельзя запускать сервер приложений. Но можно настроить удалённую отладку (Remote Debug), когда где-то отдельно есть запущенный сервер с нужным нам приложением. Воспользуемся описанием настройки отсюда: [Remote debug Wildfly in IntelliJ Idea community edition](https://yongwblog.wordpress.com/2017/03/17/remote-debug-wildfly-in-intellij-idea-community-edition/) (настройка Remote Run Configuration для 8787 порта). После настройки запускаем в Debug режиме нашу новую конфигурацию:

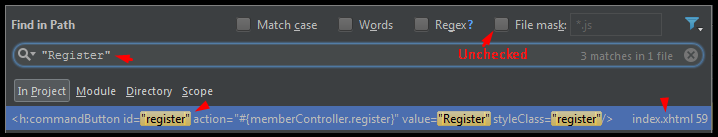


Если всё хорошо, то мы увидим внизу об этом сообщение:

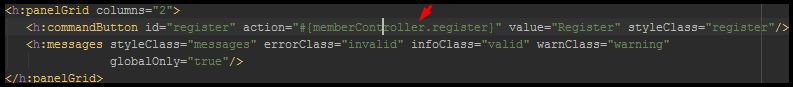


**Процесс Debug'а**

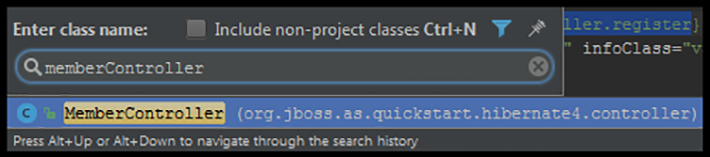
Давайте «отдебажим» сохранение записи. Для этого надо сначала определиться с местом, где мы будем исследовать. Судя по окну, нам нужна кнопка «Register». Давайте найдём её в коде. Итак, нам нужен элемент, у него должен быть текст: "Register". Или она должна как-то относится к этому. Нажимаем Ctrl+Shift+F и ищеём Register в кавычках. Видим, что такая есть на index.xhtml.



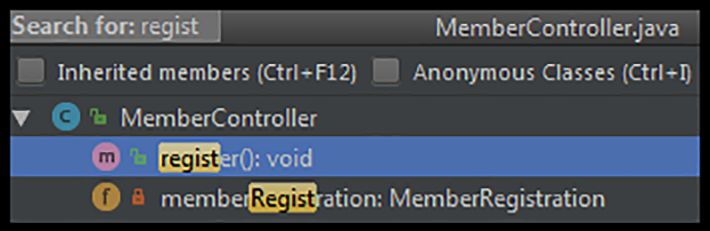
Нажимаем Enter для перехода в найденный источник:



Итак, мы видим, что при регистрации вызывается **memberController.register** Судя по всему, это должен быть некоторый java класс. Нажимаем Ctrl+N, ищем:



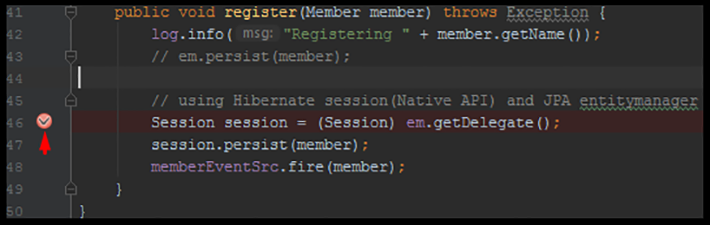
Действительно, такой класс есть . Переходим в него. Видимо, тут должен быть метод register. Нажимаем Ctrl+F12 и ищем метод register



Действительно, нашли. Судя по всему, регистрация происходит тут, в **memberRegistration.register**. Нажимаем Ctrl и кликаем по методу, чтобы «провалиться» в него:



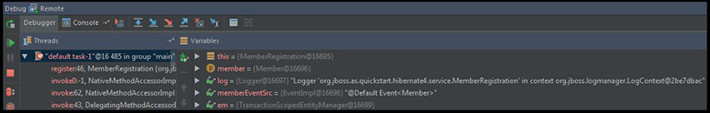
Давайте теперь поставим «точку остановки» или Break Point. Это такой маркер, который говорит, где выполнение кода должно приостановиться. Мы в этот момент получим возможность узнать много интересного. Чтобы его поставить нужно кликнуть в место правее номера строки.



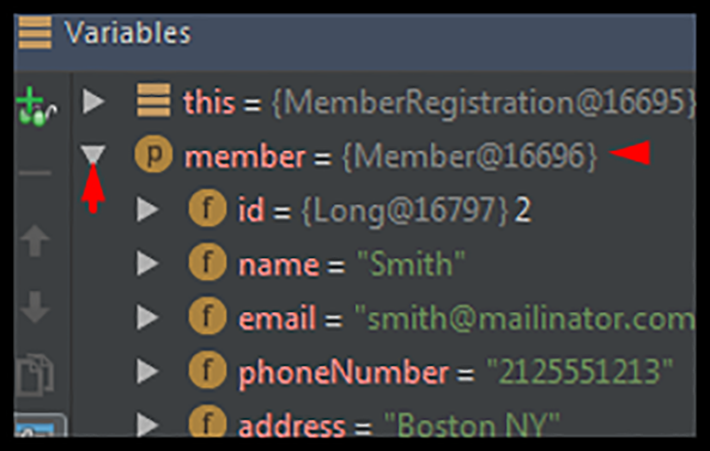
На странице http://localhost:8080/wildfly-hibernate4 заполняем поля и нажимаем кнопку Register. Значок идеи на панели мигнёт:

IntelliJ IDEA и Debug : Не дайвинг, но снорклинг - 13

Перейдя в Idea видно, что на панеле отладки много интересной информации:



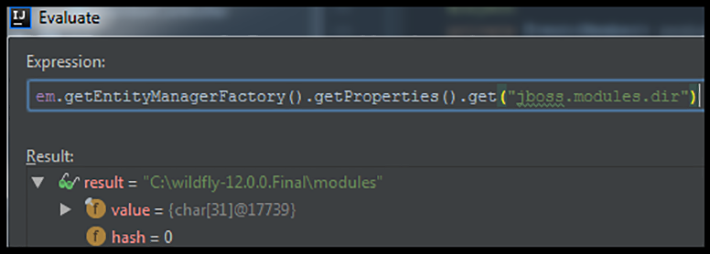
Здесь можно посмотреть значение полей объекта. Например, из чего состоит регистрируемый Member:



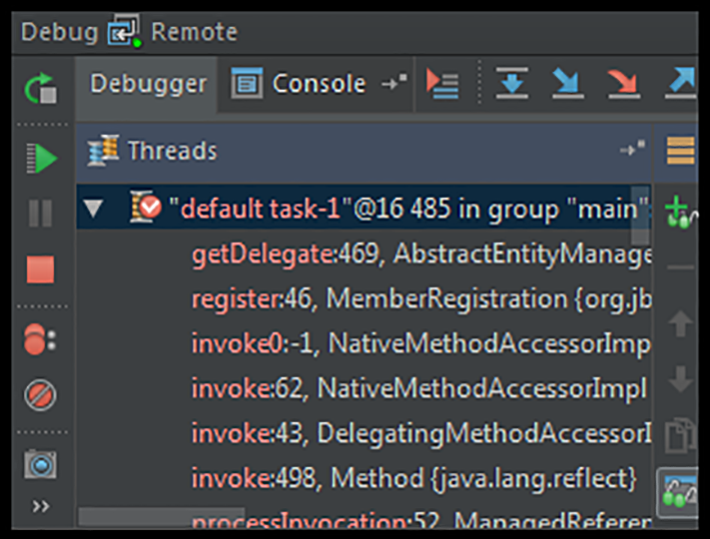
Отлично. Что мы можем ещё сделать? Мы можем открыть контекстное меню и выбрать там Evaluate Expression (или через меню Run -> Evaluate Expression). А ещё лучше на панели управления дебаггером:



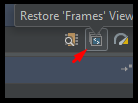
Это супер замечательная возможность в точке останова, имея доступ ко всему, к чему есть доступ к этом месте кода, выполнить любой код, который можно было бы выполнить в этом месте. Например:



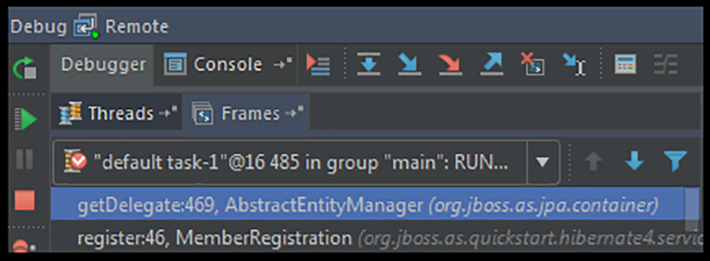
На панели управления дебаггером есть также кнопки управления, которые отвечают за то, куда необходимо переместить управление ходом выполнения программы. Не правда ли, магия?) Нажимая кнопку F8 (Step Out) мы ходим по коду без заходов в методы. Нажав F9 мы прекращаем ходить по строкам кода дебаггером, а отдаём дебаггером управление ходом выполнения программы. Если мы нажмём F7 (Step Into), то мы пойдём по коду с заходом в каждый метод, который встретим на своём пути. Кстати, обратите особое внимание на вот этот информационный блок:



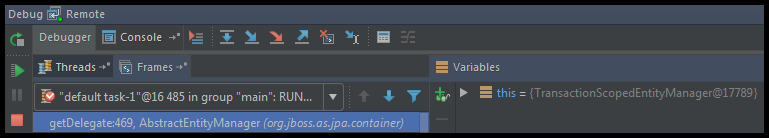
Тут показан поток, в котором мы находимся и методы в стэке текущего потока. Но и это ещё не всё. Для удобства, можно открыть вкладку фрэймов. Для этого она должна быть включена:



Теперь на вкладке фрэймов мы видим информацию о переходе от метода к методу, т.к. начали ходить по коду при помощи Step Into.



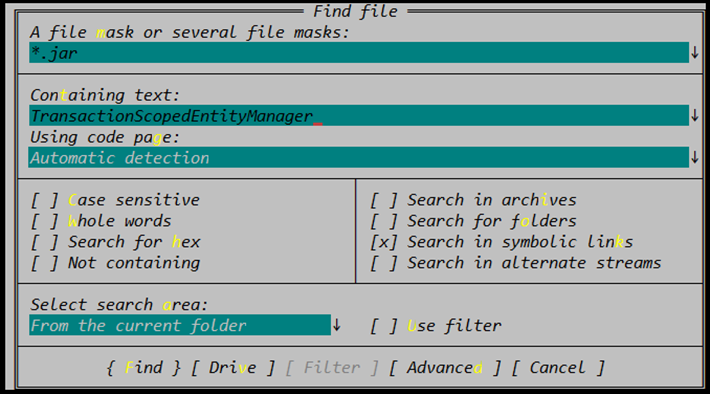
Как мы видим, не всегда нас может перебросить туда, где сейчас выполнение программы. Мы сейчас находимся в «getDelegate:469, AbstractEntityManager (org.jboss.as.jpa.container)». Но на самом деле, мы в реализации. Об этом свидетельствует класс, указанный у this:



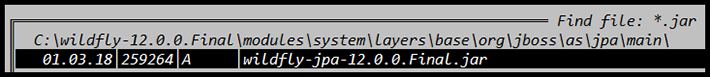
Смотрим на this. Как мы знаем, он указывается на текущий объект. Мы в TransactionScopedEntityManager. Почему же Idea нам не может показать код? Дело в том, что IntelliJ Idea сейчас не знает ни про какой TransactionScopedEntityManager, т.к. он не подключен к нашему проекту (его нет в зависимостях проекта). Когда работает сервер приложений, то внутри него крутится много-много всяких библиотек. А нам про них известно очень мало, т.к. в общем случае нам не нужно копаться во внутренностях, нам нужно чтобы оно просто работало. Но иногда этого требует работа или спортивный интерес. Тогда, надо об этой библиотеке сообщаить Idea, чтобы она знала, где брать код классов.

**Подключение сторонних библиотек для дебага**

Для начала нам самим нужно понять, что же это за библиотека, которую нужно подключить. Первый путь он трудный самый – искать в интернете. Скорость и результат нахождения результата сильно зависят от того, на сколько хорошо вели проект. Например, у WildFly открытый репозиторий. Поэтому, при поиске в гугле «TransactionScopedEntityManager» мы выйдем на <https://github.com/wildfly/wildfly/tree/master/jpa/subsystem> и найдём, что нам нужен wildfly-jpa. Второй способ – правильный. Где лежит сервер, там и искать. В этом могут помочь различные средства. Например, на Windows это может быть [Far Manager](https://www.farmanager.com/download.php?l=ru). Далее пример алгоритма поиска приведён на нём. Установив и запустив его, переключаемся при помощи **Tab** на одну из вкладок, при помощи Alt+F1 для левой вкладки или Alt+F2 для правой и выбираем нужный нам раздел на жёстком диске. Вполне возможно, что в Far Manager после установки открыт каталог самого Far Manager. Чтобы перейти в корень диска следует нажать Ctrl + \. При помощи Alt+F открываем окошко поиска, начинаем набирать название каталоге и нажимаем Enter после того, как каталог найден. Этот поиск хитрый и выделяет те каталоги, которые совпадает с текстом поиска. Если вводятся символы, для которых нет папок, то такие символы ввести нельзя. Переходим таким образом в каталог сервера приложений. Допустим, мы не знаем, где на сервере лежат модули. Возможно, Вы вообще в первый раз в жизни слышите про какой-то там WildFly. Поэтому, нажимаем сразу здесь Alt+F7 для поиска файлов. Итак, логика подсказывает: нам нужен файл с библиотекой. То есть нам нужен jar. Внутри должен быть класс TransactionScopedEntityManager. Т.к. класс = файл, то ищем по «содержит». То есть как-то так:



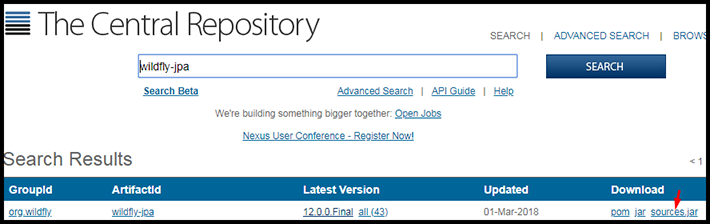
Теперь, ждём результат. Он не заставит себя ждать)



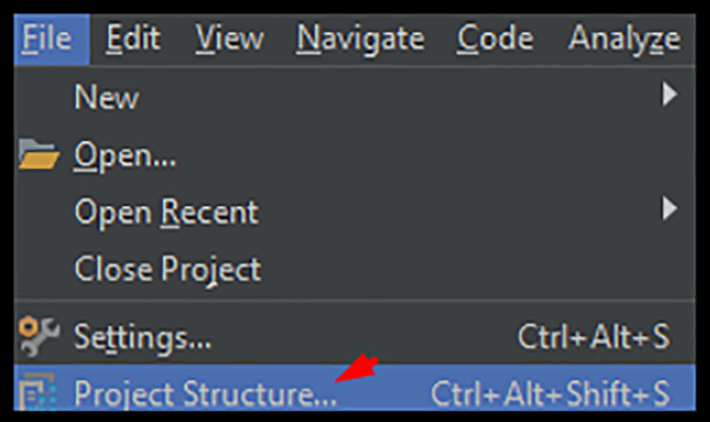
Теперь, нам надо где-то найти исходный код для него. И тут есть 2 варианта:

* Супер полезный [versioneye.com](https://www.versioneye.com/java)
* Центральный [Maven Repository](https://search.maven.org/" \t "_blank)

Воспользуемся, пожалуй, вторым. Найдём там:



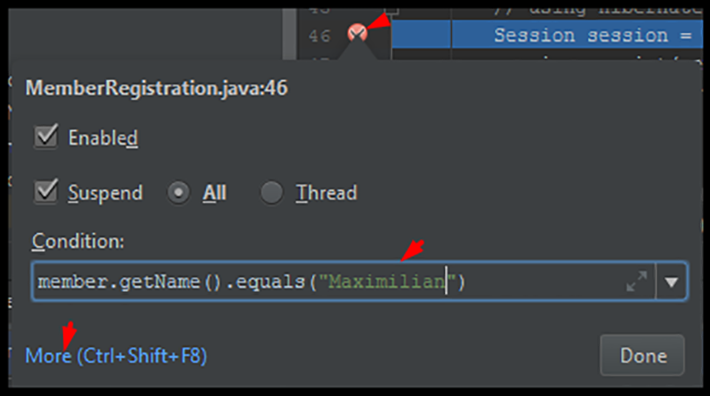
Теперь же переходим к описанию зависимости. На этой страничке есть возможность скачать исходный код. Отлично, теперь код у нас скачан. Осталось подключить библиотеку. Подключается она предельно просто. Нам нужно открыть настройки проекта:



Там выбираем «Libraries» и в раздел «Sources» добавляем полученные исходные коды, а в разделе «Classes» указываем сам jar файл библиотеки из каталога WildFly, который нами был найден при помощи Far Manager’а. После этого мы увидим при переходе по F7 содержимое класса AbstractEntityManager и TransactionScopedEntityManager, а так же станет доступен через поиск по классам через Ctrl+N.

**Break Points с условиями**

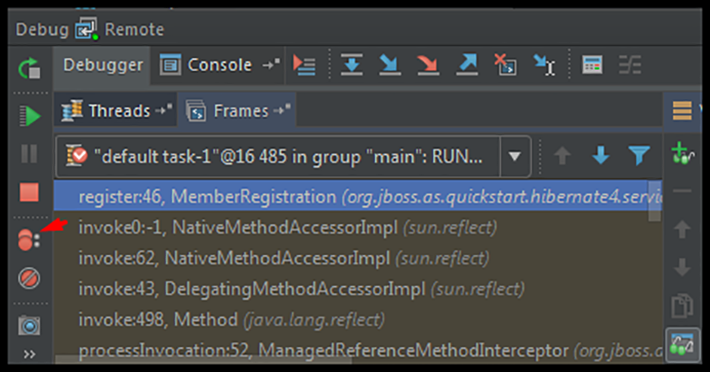
Вернёмся теперь к Break Points. Иногда, мы хотим остановиться не всегда, а только при каком-нибудь условии. Что же делать? И тут нам тоже поможет наша IDE. Поставив Break Point мы можем назначит ей условие. Например, поставим точку и нажмём по ней правой кнопкой мыши:



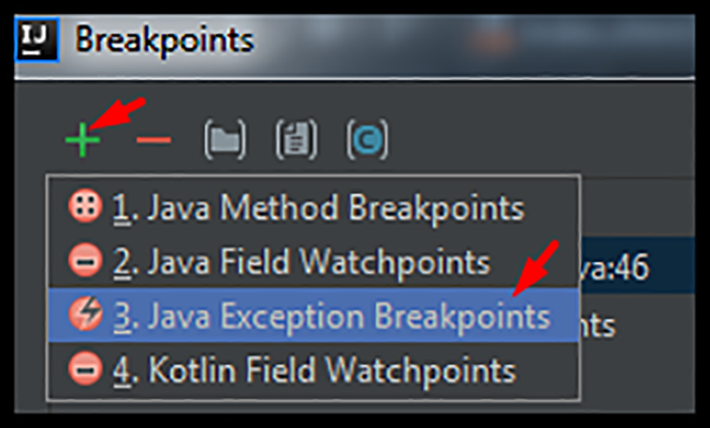
Теперь точка остановки сработает только тогда, когда имя будет Maximilian. По кнопке More Вам будут доступны расширенный набор настроек для Break Point’ов.

**Break Points на исключения**

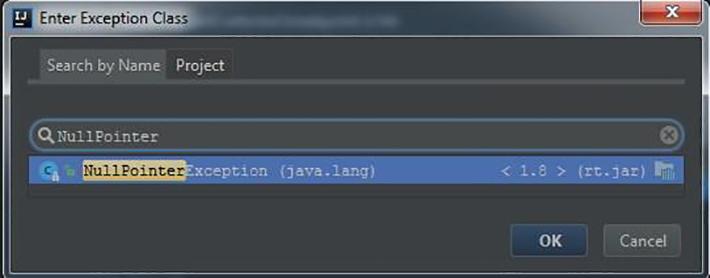
Иногда мы можем получать ошибку и хочется проследить, откуда она идёт. Тогда мы можем добавить точку остановки не на конкретную строку кода, а на место, где будет брошено исключение. Для этого нужно раскрыть список всех точек остановки:



И создать новое правило для выбранного типа исключения:



Например, для NPE:



**HotSwap классов**

Дебаггер — удивительная вещь. Она помимо дебага позволяет изменять классы! Да, ресурсы (такие как xhtml стриницы, например) не изменить так просто. Но вот код java классов можно подменить на лету (это и называется Hot Swap). Для этого достаточно при присоединённом дебаггере изменить класс и выполнить Run -> Reload Changed Classes. Полезный обзор на эту тему: [4 free ways to hot-swap code on the JVM](https://sites.google.com/a/athaydes.com/renato-athaydes/posts/4freewaystohot-swapcodeonthejvm)

**Заключение**

Дебаггер — мощный инструмент, позволяющий разработчику проникнуть в самую глубину выполняемого кода, изучить его во всех деталях. Это позволяет исправлять самые запутанные ошибки. Так же это позволяет лучше понять то, как те или иные библиотеки работают. Даже такой краткий обзор вышел довольно внушительным, но я надеюсь он будет полезен и интересен. Если кого-то смог заинтересовать данный материал, то продолжить погружение можно по ссылкам:

* [Debugging Your First Java Application](https://www.jetbrains.com/help/idea/debugging-your-first-java-application.html)
* [Tutorial: Java Debugging Deep Dive](https://www.jetbrains.com/help/idea/tutorial-java-debugging-deep-dive.html)

