Отладка вашего приложения

Android Studio предоставляет отладчик, который позволяет вам делать следующее и многое другое:

* Выберите устройство для отладки вашего приложения.
* Установите точки останова в вашем коде Java, Kotlin и C / C ++.
* Изучите переменные и оцените выражения во время выполнения.

Эта страница содержит инструкции для основных операций отладчика. Для получения дополнительной документации см. Также [документы по отладке IntelliJ IDEA](https://www.jetbrains.com/help/idea/2019.1/debugging.html) .

Включить отладку

Прежде чем вы сможете начать отладку, вам необходимо подготовиться следующим образом:

* **Включите отладку на вашем устройстве:**

Если вы используете эмулятор, он включен по умолчанию. Но для подключенного устройства необходимо [включить отладку в настройках разработчика устройства](https://developer.android.com/studio/debug/dev-options.html) .

* **Запустите отладочный вариант сборки:**

Вы должны использовать [вариант сборки,](https://developer.android.com/studio/build/build-variants.html) который входит в конфигурацию сборки. Обычно вы можете просто выбрать вариант «отладки» по умолчанию, который включен в каждый проект Android Studio (даже если он не отображается в файле). Но если вы определяете новые типы сборки, которые должны быть отлаживаемыми, вы должны добавить `debuggable true` к типу сборки:[debuggable true](http://google.github.io/android-gradle-dsl/current/com.android.build.gradle.internal.dsl.BuildType.html#com.android.build.gradle.internal.dsl.BuildType:debuggable)build.gradle

андроид {   
    buildTypes {   
        customDebugType {   
            отладке верно ... } } }  
              
          
   

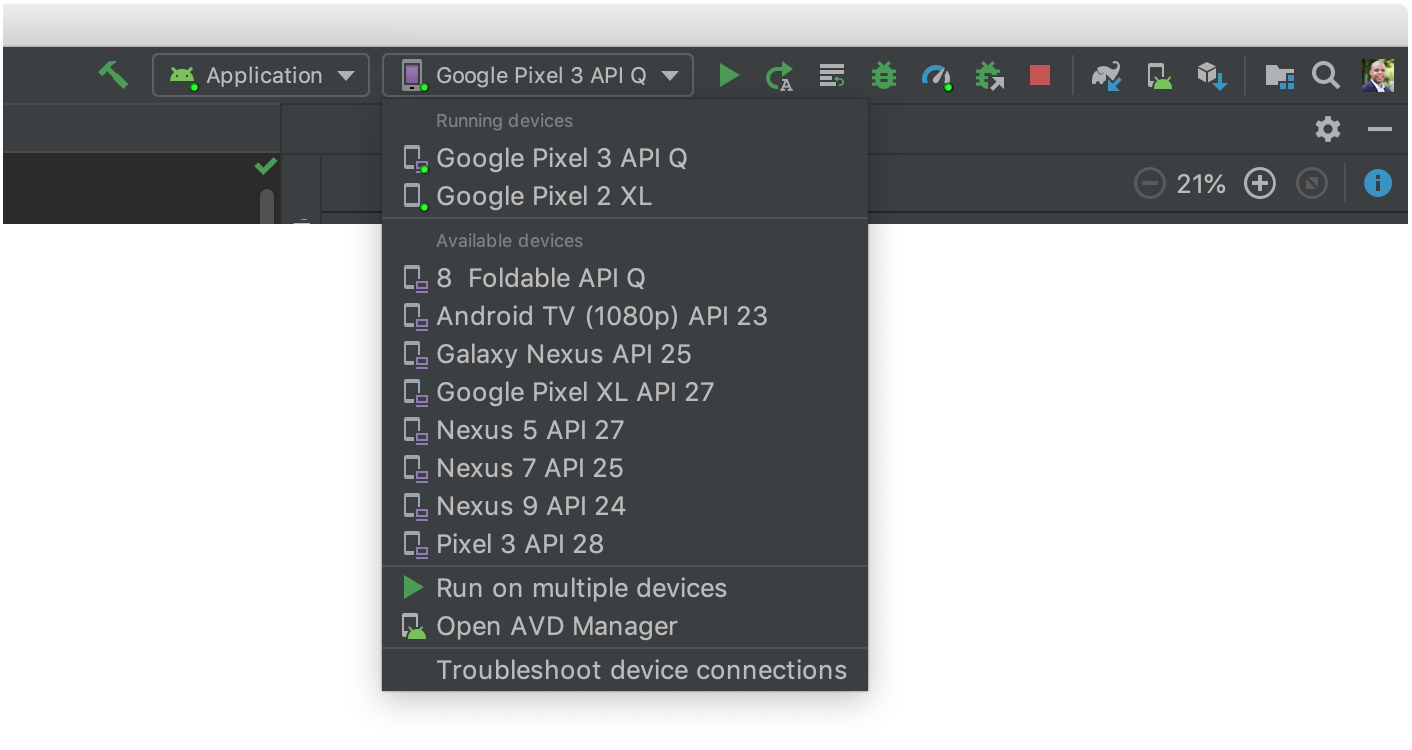
Это свойство также применяется к [модулям с кодом C / C ++](https://developer.android.com/studio/projects/add-native-code.html) . ( jniDebuggableСвойство больше не используется.)

Если ваше приложение зависит от модуля библиотеки, который вы также хотите отлаживать, эта библиотека также должна быть упакована, debuggable trueчтобы она сохраняла свои символы отладки. Чтобы отладочные варианты вашего проекта приложения получили отлаживаемый вариант библиотечного модуля, убедитесь, что вы [публикуете версии своей библиотеки не по умолчанию](https://developer.android.com/studio/projects/android-library.html#publish_multiple_variants) .

Начать отладку

Вы можете начать сеанс отладки следующим образом:

1. Установите некоторые [точки останова](https://developer.android.com/studio/debug#breakPoints) в коде приложения.
2. На панели инструментов выберите устройство для отладки приложения в раскрывающемся меню целевого устройства.

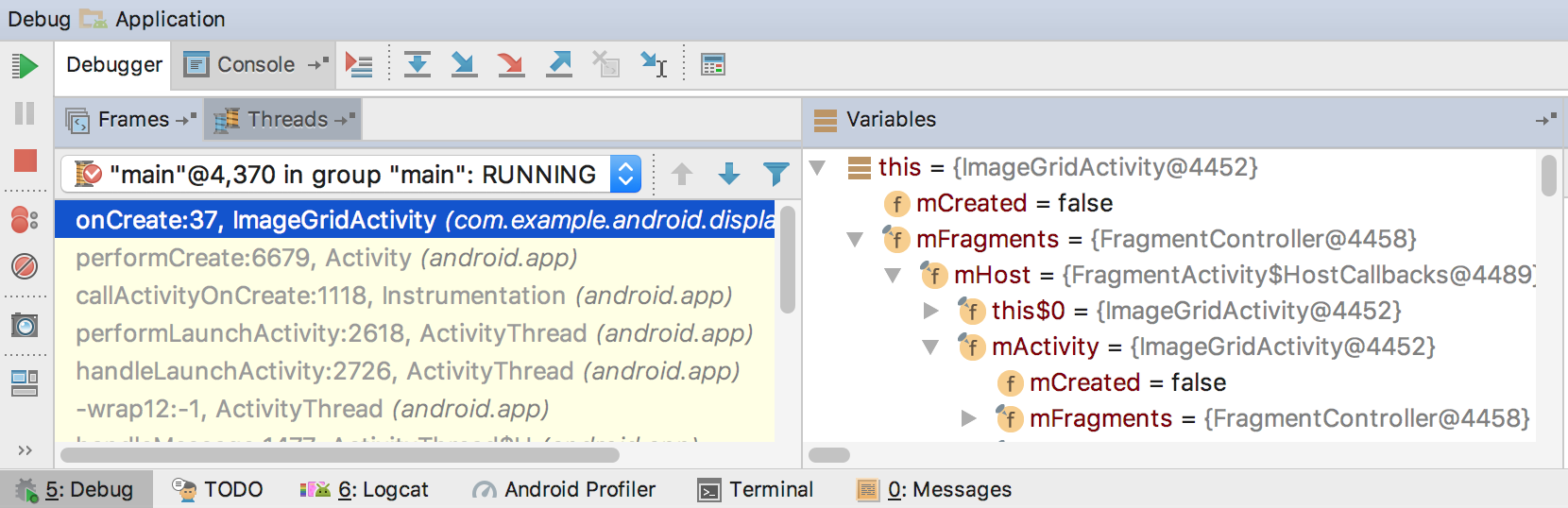


Если у вас нет настроенных устройств, вам нужно либо [подключить устройство через USB,](https://developer.android.com/studio/run/device#connect) либо [создать AVD](https://developer.android.com/studio/run/managing-avds#createavd) для использования [эмулятора Android](https://developer.android.com/studio/run/emulator.html) .

1. На панели инструментов нажмите **Отладка** https://developer.android.com/studio/images/buttons/toolbar-debug.png .

Если вы видите диалоговое окно с вопросом, хотите ли вы «переключиться с Run на Debug», это означает, что ваше приложение уже запущено на устройстве, и оно перезапустится, чтобы начать отладку. Если вы хотите, чтобы тот же экземпляр приложения работал, нажмите « **Отменить отладку»** и вместо этого [присоедините отладчик к запущенному приложению](https://developer.android.com/studio/debug#attach-debugger) .

В противном случае Android Studio создает APK, подписывает его с помощью ключа отладки, устанавливает его на выбранное устройство и запускает его. Если вы [добавляете код C и C ++ в свой проект](https://developer.android.com/studio/projects/add-native-code.html) , Android Studio также запускает [отладчик LLDB](http://lldb.llvm.org/) в окне **Debug** для отладки вашего собственного кода.

1. Если окно « **Отладка»** не открыто, выберите « **Вид»> «Окна инструментов»> «Отладка»** (или нажмите « **Отладка»** https://developer.android.com/studio/images/buttons/toolbar-debug.png на панели инструментов), а затем перейдите на вкладку « **Отладчик** », как показано на рисунке 1. 

**Рисунок 1.** Окно отладчика, показывающее текущий поток и дерево объектов для переменной

Присоедините отладчик к работающему приложению

Если ваше приложение уже запущено на вашем устройстве, вы можете начать отладку без перезапуска приложения следующим образом:

1. Нажмите **Присоединить отладчик к процессу Android** https://developer.android.com/studio/images/buttons/toolbar-attach-debugger.png .
2. В диалоговом окне « **Выбрать процесс** » выберите процесс, к которому вы хотите присоединить отладчик.

Если вы используете эмулятор или рутированное устройство, вы можете установить флажок **Показать все процессы,** чтобы увидеть все процессы.

В раскрывающемся меню « **Отладчик»** вы можете [выбрать другой тип отладки](https://developer.android.com/studio/debug#debug-types) . По умолчанию Android Studio использует тип **автоматической** отладки, чтобы выбрать лучший вариант отладчика, в зависимости от того, содержит ли ваш проект код Java или C / C ++.

1. Нажмите **ОК** .

Откроется окно « **Отладка»** .

**Примечание** . Отладчик и сборщик мусора в Android Studio слабо интегрированы. Виртуальная машина Android гарантирует, что любой объект, о котором знает отладчик, не будет собираться мусором до тех пор, пока отладчик не отключится. Это может привести к накоплению объектов с течением времени, пока подключен отладчик. Например, если отладчик видит работающий поток, связанный с ним **[Thread](https://developer.android.com/reference/java/lang/Thread.html)** объект не собирается сборщиком мусора до тех пор, пока отладчик не отключится, даже если поток завершен.

Изменить тип отладчика

Поскольку для отладки кода Java / Kotlin и кода C / C ++ требуются разные инструменты отладчика, отладчик Android Studio позволяет выбрать тип используемого отладчика. По умолчанию Android Studio решает, какой отладчик использовать, основываясь на том, какие языки он обнаруживает в вашем проекте (с типом **Auto** debugger). Однако вы можете вручную выбрать отладчик в [конфигурации отладки](https://developer.android.com/studio/run/rundebugconfig.html) (нажмите « **Выполнить»> «Редактировать конфигурации»** ) или в диалоговом окне, которое появляется при нажатии « **Выполнить»> «Подключить отладчик к процессу Android»** .

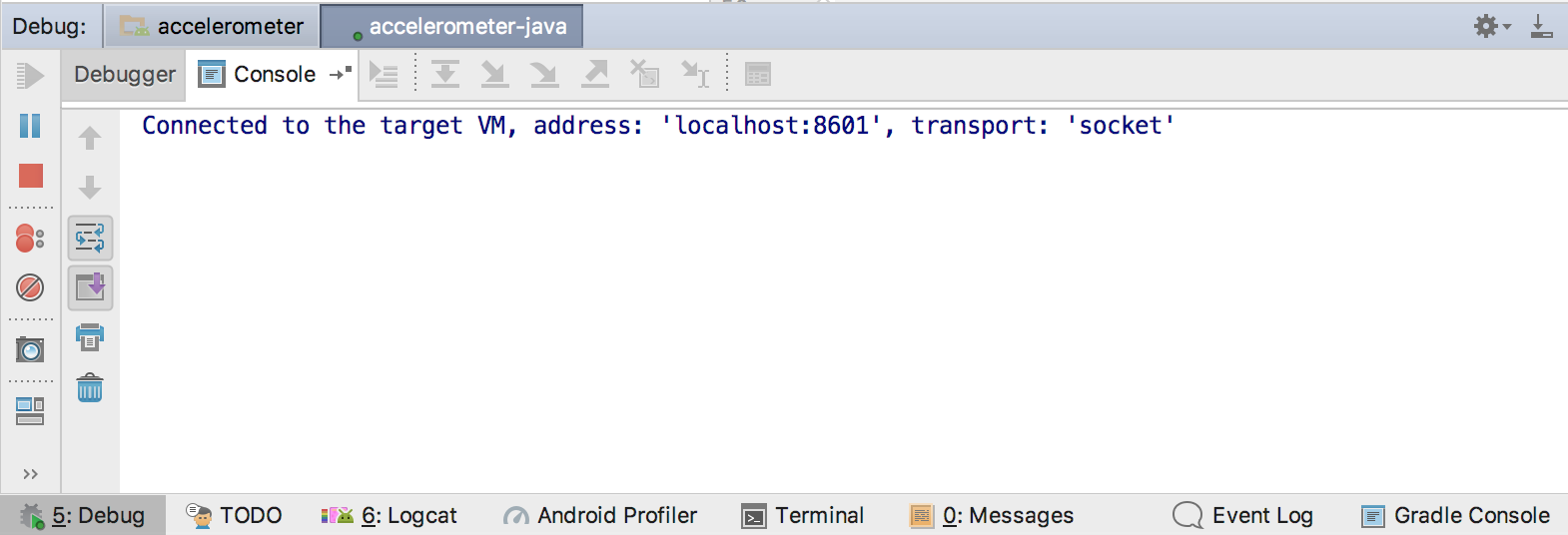
Доступны следующие типы отладки:

* **Авто** : выберите, если вы хотите, чтобы Android Studio автоматически выбирала лучший вариант для кода, который вы отлаживаете. Например, если в вашем проекте есть какой-либо код C или C ++, Android Studio автоматически использует тип **двойной** отладки. В противном случае Android Studio использует тип отладки **Java** .
* **Java** : выберите, если вы хотите отлаживать только код, написанный на Java или Kotlin - отладчик Java игнорирует любые точки останова или наблюдает за тем, что вы установили в своем собственном коде.
* **Собственный** : (Доступно только с кодом C / C ++.) Выберите, если вы хотите использовать только LLDB для отладки вашего кода. При использовании этого типа отладки представление сеанса отладчика Java недоступно. По умолчанию LLDB проверяет только ваш собственный код и игнорирует точки останова в вашем коде Java. Если вы также хотите отлаживать свой Java-код, вам следует переключиться на **автоматический** или **двойной** тип отладки.

**Примечание.** Встроенная отладка не работает в 32-разрядной версии Windows в Android Studio 3.0 и более поздних версиях. Если вы используете 32-разрядную версию Windows и вам необходимо отладить собственный код, вам следует использовать [Android Studio 2.3](https://developer.android.com/studio/archive.html" \l "android-studio-2-3-3) .

* **Dual** : (Доступно только с кодом C / C ++.) Выберите, если вы хотите переключаться между отладкой как Java, так и собственного кода. Android Studio подключает отладчик Java и LLDB к процессу вашего приложения, один для отладчика Java и один для LLDB, так что вы можете проверять точки останова как в Java, так и в собственном коде, без перезапуска приложения или изменения конфигурации отладки.

На рисунке 2 обратите внимание на две вкладки справа от заголовка окна « **Отладка»** . Поскольку в приложении имеется как код Java, так и код C ++, одна вкладка предназначена для отладки собственного кода, а другая - для отладки кода Java, как указано **-java** .



**Рисунок 2.** Вкладка для отладки собственного кода и вкладка для отладки кода Java

**Примечание:** Если вы отладки машинный код, оптимизированный компилятор, вы можете получить следующее предупреждение: **This function was compiled with optimizations enabled. Some debugger features may not be available**. При использовании флагов оптимизации, таких как [**-O**флаги](https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html) , компилятор вносит изменения в ваш скомпилированный код, чтобы он работал более эффективно. Это может привести к тому, что отладчик сообщит о непредвиденной или неправильной информации, поскольку отладчику сложно отобразить оптимизированный скомпилированный код обратно в исходный исходный код. По этой причине вы должны отключить оптимизацию компилятора при отладке собственного кода.

Используйте системный журнал

Системный журнал показывает системные сообщения во время отладки вашего приложения. Эти сообщения содержат информацию из приложений, запущенных на устройстве. Если вы хотите использовать системный журнал для отладки своего приложения, убедитесь, что ваш код записывает сообщения журнала и печатает трассировку стека для исключений, пока ваше приложение находится в стадии разработки.

Напишите сообщения журнала в вашем коде

Чтобы написать сообщения журнала в вашем коде, используйте класс. Сообщения журнала помогают понять поток выполнения, собирая выходные данные отладки системы во время взаимодействия с вашим приложением. Журнальные сообщения могут сказать вам, какая часть вашего приложения вышла из строя. Для получения дополнительной информации о ведении журнала см. [Запись и просмотр журналов](https://developer.android.com/studio/debug/am-logcat.html) . [Log](https://developer.android.com/reference/android/util/Log.html)

В следующем примере показано, как можно добавить сообщения журнала, чтобы определить, доступна ли предыдущая информация о состоянии при запуске вашей деятельности:

[КОТЛИН](https://developer.android.com/studio/debug#kotlin)[ДЖАВА](https://developer.android.com/studio/debug#java)

импорт андроид . Util . Log ... private val TAG : String = MyActivity :: class . Java . simpleName ... class MyActivity : Activity () { ... переопределить fun onCreate ( saveInstanceState : Bundle ?) { ... if ( saveInstanceState ! = null ) { Журнал . d ( TAG  
  
   
  
   
      
      
          
          
            , "onCreate () Восстановление предыдущего состояния" ) / \* восстановление состояния \* / } else { Журнал . d ( TAG , "onCreate () Нет сохраненного состояния доступно" ) / \* инициализировать приложение \* / } } }   
              
          
              
              
          
   

Во время разработки ваш код также может перехватывать исключения и записывать трассировку стека в системный журнал:

[КОТЛИН](https://developer.android.com/studio/debug#kotlin)[ДЖАВА](https://developer.android.com/studio/debug#java)

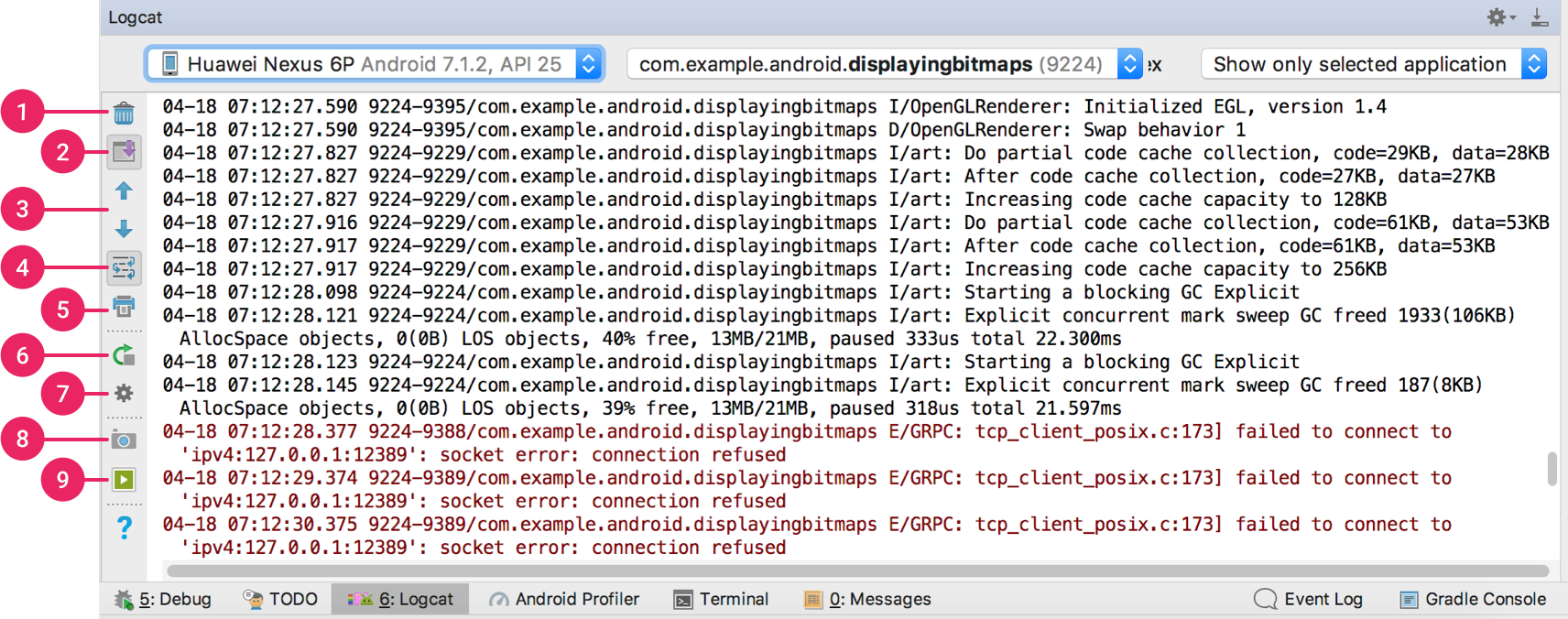
fun someOtherMethod () { try { ... } catch ( e : SomeException ) { Журнал . d ( TAG , "someOtherMethod ()" , e ) } }   
      
          
      
          
   

**Примечание.** Удалите сообщения журнала отладки и вызовы печати трассировки стека из своего кода, когда будете готовы опубликовать свое приложение. Вы можете сделать это, установив **DEBUG** флаг и поместив сообщения журнала отладки в условные операторы.

Просмотр системного журнала

Вы можете просматривать и фильтровать отладочные и другие системные сообщения в окне **Logcat** . Например, вы можете видеть сообщения, когда происходит сборка мусора, или сообщения, которые вы добавляете в свое приложение с помощью [Log](https://developer.android.com/reference/android/util/Log.html)класса.

Чтобы использовать logcat, [запустите отладку](https://developer.android.com/studio/debug#startdebug) и выберите вкладку **Logcat** на нижней панели инструментов, как показано на рисунке 3.



**Рисунок 3.** Окно Logcat с настройками фильтра

Описание logcat и его параметров фильтрации см. В разделе « [Запись и просмотр журналов с помощью Logcat»](https://developer.android.com/studio/debug/am-logcat.html) .

Работа с точками останова

Android Studio поддерживает несколько типов точек останова, которые запускают различные действия по отладке. Наиболее распространенный тип - это точка останова на линии, которая приостанавливает выполнение вашего приложения на указанной строке кода. Во время паузы вы можете исследовать переменные, оценивать выражения, а затем продолжать выполнение строки за строкой, чтобы определить причины ошибок времени выполнения.

Чтобы добавить точку останова строки, выполните следующие действия:

1. Найдите строку кода, где вы хотите приостановить выполнение, затем либо щелкните левый желоб вдоль этой строки кода, либо поместите курсор на строку и нажмите Control + F8 (на Mac: Command + F8).
2. Если ваше приложение уже запущено, вам не нужно обновлять его, чтобы добавить точку останова - просто нажмите « **Присоединить отладчик к процессу Android»** https://developer.android.com/studio/images/buttons/toolbar-attach-debugger.png . В противном случае начните отладку, нажав **Отладка** https://developer.android.com/studio/images/buttons/toolbar-debug.png .

https://developer.android.com/images/tools/as-breakpointline.png

**Рисунок 3.** Красная точка появляется рядом с линией, когда вы устанавливаете точку останова

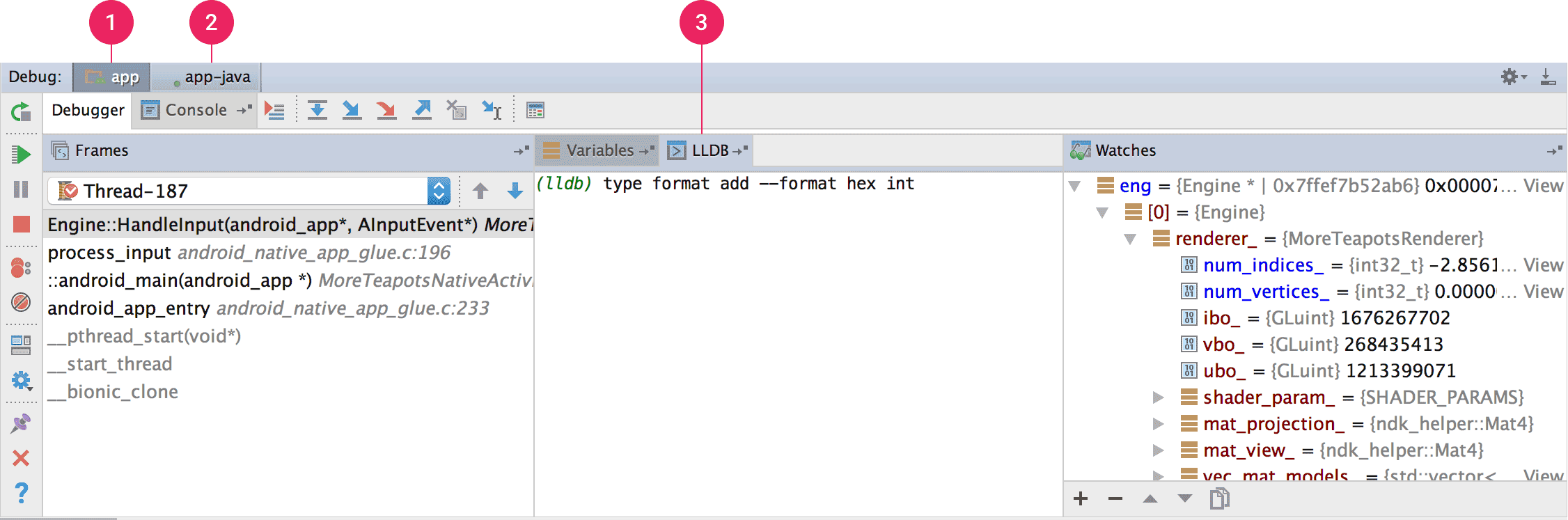
Когда выполнение вашего кода достигает точки останова, Android Studio приостанавливает выполнение вашего приложения. Затем вы можете использовать инструменты на вкладке « **Отладчик** », чтобы определить состояние приложения:

* Чтобы проверить дерево объектов на наличие переменной, разверните ее в представлении « **Переменные»** . Если представление « **Переменные»** не отображается, нажмите « **Восстановить представление переменных»** https://developer.android.com/images/tools/as-varviewbutton.png .
* Чтобы оценить выражение в текущей точке выполнения, нажмите « **Оценить выражение»** https://developer.android.com/studio/images/debug/debugger_button_evaluate_2-0.png .
* Чтобы перейти к следующей строке кода (без ввода метода), нажмите « **Перешагнуть»** https://developer.android.com/studio/images/debug/debugger_button_stepover_2-0.png .
* Чтобы перейти к первой строке внутри вызова метода, нажмите **Step Into** https://developer.android.com/studio/images/debug/debugger_button_stepinto_2-0.png .
* Чтобы перейти к следующей строке за пределами текущего метода, нажмите « **Выйти»** https://developer.android.com/studio/images/debug/debugger_button_stepout_2-0.png .
* Чтобы продолжить нормальное выполнение приложения, нажмите « **Продолжить программу»** https://developer.android.com/studio/images/debug/debugger_button_resume_2-0.png .

Если ваш проект использует какой-либо собственный код, по умолчанию тип **автоматической** отладки присоединяет отладчик Java и LLDB к вашему приложению как два отдельных процесса, поэтому вы можете переключаться между проверкой точек останова Java и C / C ++ без перезапуска приложения или изменения настроек.

**Примечание.** Чтобы Android Studio обнаруживал точки останова в коде C или C ++, необходимо использовать тип отладки, поддерживающий LLDB, например Auto, Native или Dual. Вы можете изменить тип отладки, который использует Android Studio, [отредактировав свою конфигурацию отладки](https://developer.android.com/studio/run/rundebugconfig.html) . Чтобы узнать больше о различных типах отладки, прочитайте раздел об использовании других [типов отладки](https://developer.android.com/studio/debug#debug-types) .

Когда Android Studio развертывает ваше приложение на целевом устройстве, открывается окно «Отладка» с вкладкой или представлением сеанса отладки для каждого процесса отладчика, как показано на рисунке 4.



**Рисунок 4.** Отладка нативного кода с использованием LLDB

1. Android Studio переключается на вкладку ***<your-module>,*** когда отладчик LLDB обнаруживает точку останова в вашем коде C / C ++. В **Фреймы** , **переменные** и **часы** стекла также доступны и работают точно так , как они , если бы вы отладки Java код. Хотя панель « **Потоки»** недоступна в представлении сеанса LLDB, доступ к процессам приложения можно получить с помощью раскрывающегося списка на панели « **Кадры»** . Вы можете узнать больше об этих панелях в разделах о том, как [отлаживать оконные рамы](https://developer.android.com/studio/debug#stackFrames) и [проверять переменные](https://developer.android.com/studio/debug#variablesAndWatches) .

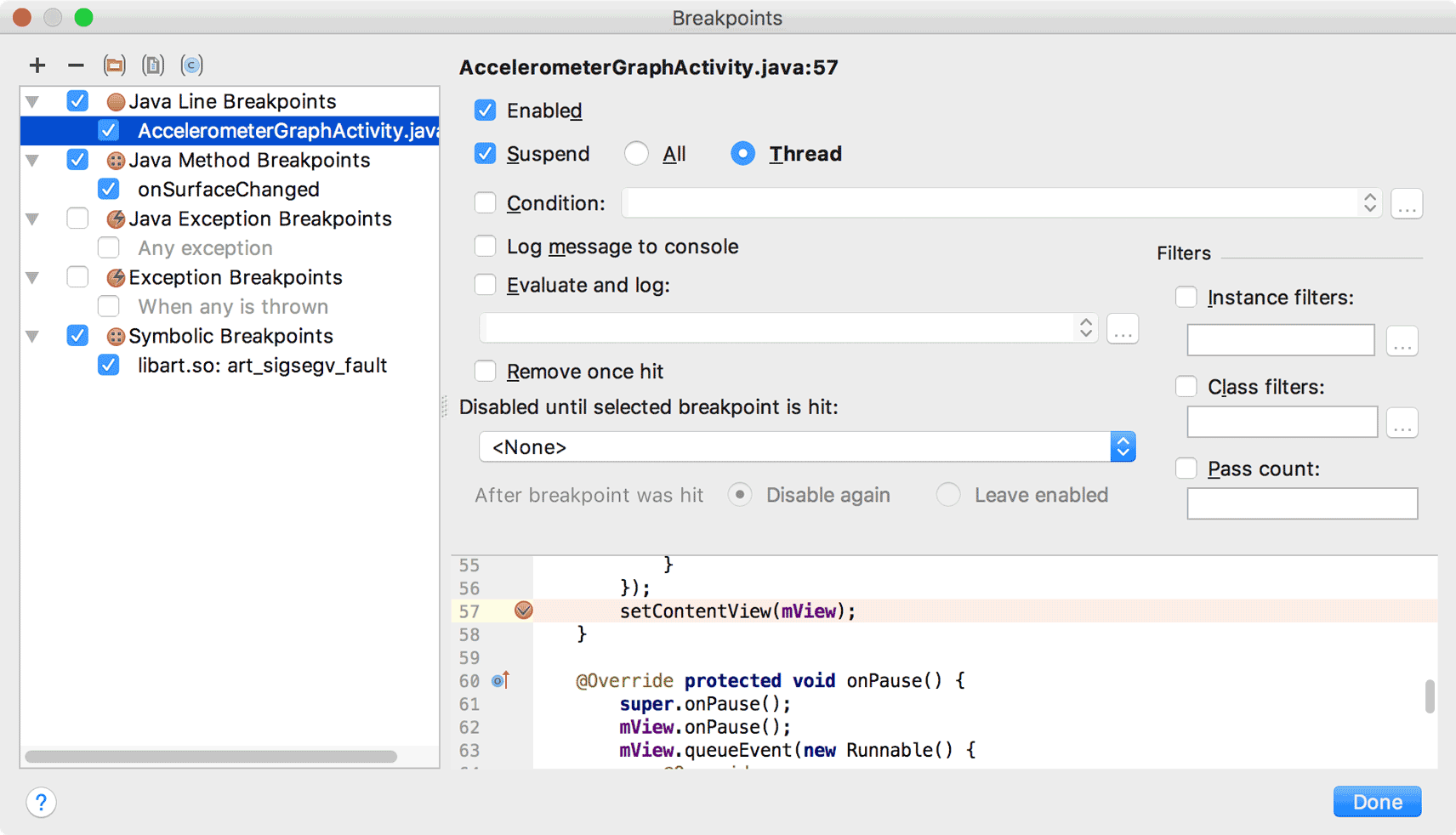
**Примечание.** При проверке точки останова в вашем собственном коде система Android приостанавливает работу виртуальной машины, на которой выполняется байт-код Java вашего приложения. Это означает, что вы не можете взаимодействовать с отладчиком Java или получать какую-либо информацию о состоянии из сеанса отладчика Java во время проверки точки останова в своем собственном коде.

1. Android Studio переключается на вкладку ***<your-module>* -java,** когда отладчик Java обнаруживает **точку** останова в вашем коде Java.
2. При отладке с LLDB вы можете использовать терминал **LLDB** в представлении сеанса LLDB для передачи параметров [командной строки в LLDB](http://lldb.llvm.org/tutorial.html) . Если у вас есть определенные команды, которые вы хотели бы, чтобы LLDB выполнял каждый раз, когда вы начинаете отлаживать свое приложение, либо непосредственно, либо сразу после того, как отладчик подключается к процессу вашего приложения, вы можете [добавить эти команды в свою конфигурацию отладки](https://developer.android.com/studio/run/rundebugconfig.html#debugger-tab) .

Во время отладки кода C / C ++ вы также можете установить специальные типы точек останова, называемых точками *наблюдения* , которые могут приостанавливать процесс вашего приложения, когда ваше приложение взаимодействует с определенным блоком памяти. Чтобы узнать больше, прочитайте раздел о том, как [добавить точки наблюдения](https://developer.android.com/studio/debug#watchpoints) .

Просмотр и настройка точек останова

Чтобы просмотреть все точки останова и настроить параметры точек останова, нажмите « **Просмотреть** точки останова» https://developer.android.com/images/tools/as-viewbreakbutton.pngв левой части окна « **Отладка»** . Появится окно **точек останова** , как показано на рисунке 5.



**Рисунок 5.** Окно Breakpoints перечисляет все текущие точки останова и включает настройки поведения для каждого

Окно **точек останова** позволяет вам включить или отключить каждую **точку** останова из списка слева. Если точка останова отключена, Android Studio не приостанавливает ваше приложение, когда оно достигает этой точки останова. Выберите точку останова из списка, чтобы настроить ее параметры. Вы можете настроить точку останова так, чтобы она сначала была отключена, и чтобы система включала ее после достижения другой точки останова. Вы также можете настроить, следует ли отключать точку останова после ее нажатия. Чтобы установить **точку** останова для любого исключения, выберите **Точки** останова исключения в списке точек останова.

Отладка оконных рам

В окне **отладчика** панель **Frames** позволяет вам проверить кадр стека, который вызвал попадание на текущую точку останова. Это позволяет вам перемещаться и исследовать фрейм стека, а также просматривать список потоков в вашем приложении для Android. Чтобы выбрать поток, используйте выпадающий список выбора потоков и просмотрите его кадр стека. Щелчок по элементам в рамке открывает источник в редакторе. Вы также можете настроить представление потока и экспортировать кадр стека, как описано в руководстве по [оконным фреймам](https://www.jetbrains.com/help/idea/2019.1/debug-tool-window-frames.html) .

Проверьте переменные

В окне **отладчика** панель « **Переменные»** позволяет вам проверять переменные, когда система останавливает ваше приложение на точке останова и вы выбираете кадр на панели « **Кадры»** . Панель « **Переменные»** также позволяет оценивать специальные выражения с использованием статических методов и / или переменных, доступных в выбранном кадре.

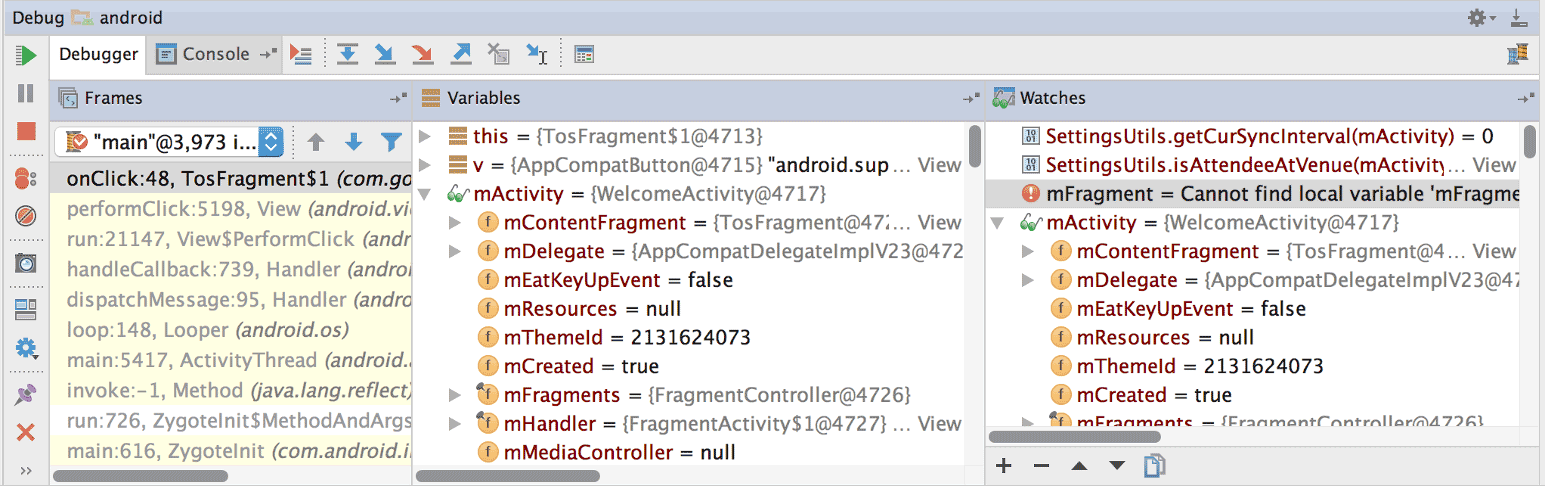
Панель **Watches** обеспечивает аналогичную функциональность, за исключением того, что выражения, добавленные на панель **Watches,** сохраняются между сеансами отладки. Вы должны добавить отслеживание переменных и полей, к которым вы часто обращаетесь или которые предоставляют состояние, которое полезно для текущего сеанса отладки. Эти **переменные** и **часы** стекла выглядят , как показано на рисунке 5.

Чтобы добавить переменную или выражение в список **Watches** , выполните следующие действия:

1. Начните отладку.
2. На панели « **Часы»** нажмите « **Добавить»** https://developer.android.com/studio/images/buttons/ic_plus.png .
3. В появившемся текстовом поле введите имя переменной или выражения, которое вы хотите просмотреть, и нажмите Enter.

Чтобы удалить элемент из списка « **Часы»** , выберите его и нажмите « **Удалить»** https://developer.android.com/studio/images/buttons/ic_minus.png .

Вы можете изменить порядок элементов в списке « **Часы** », выбрав элемент и нажав « **Вверх»** https://developer.android.com/studio/images/buttons/ic_arrow_up.png или « **Вниз»** https://developer.android.com/studio/images/buttons/ic_arrow_down.png .



**Рисунок 6.** Панели Variables и Watches в окне отладчика

Добавить точки наблюдения

Во время отладки кода C / C ++ вы можете установить специальные типы точек останова, называемых точками *наблюдения* , которые могут приостанавливать процесс вашего приложения, когда ваше приложение взаимодействует с определенным блоком памяти. Например, если вы установите два указателя на блок памяти и назначите ему контрольную точку, использование любого указателя для доступа к этому блоку памяти вызовет контрольную точку.

В Android Studio вы можете создать точку наблюдения во время выполнения, выбрав определенную переменную, но LLDB назначает точку наблюдения только блоку памяти, который система выделяет для этой переменной, а не самой переменной. Это отличается от добавления переменной на панель **Watches** , которая позволяет вам наблюдать значение переменной, но не позволяет приостанавливать процесс приложения, когда система считывает или изменяет его значение в памяти.

**Примечание.** Когда процесс приложения выходит из функции и система освобождает свои локальные переменные от памяти, вам необходимо переназначить любые точки наблюдения, созданные вами для этих переменных.

Чтобы установить точку наблюдения, вы должны соответствовать следующим требованиям:

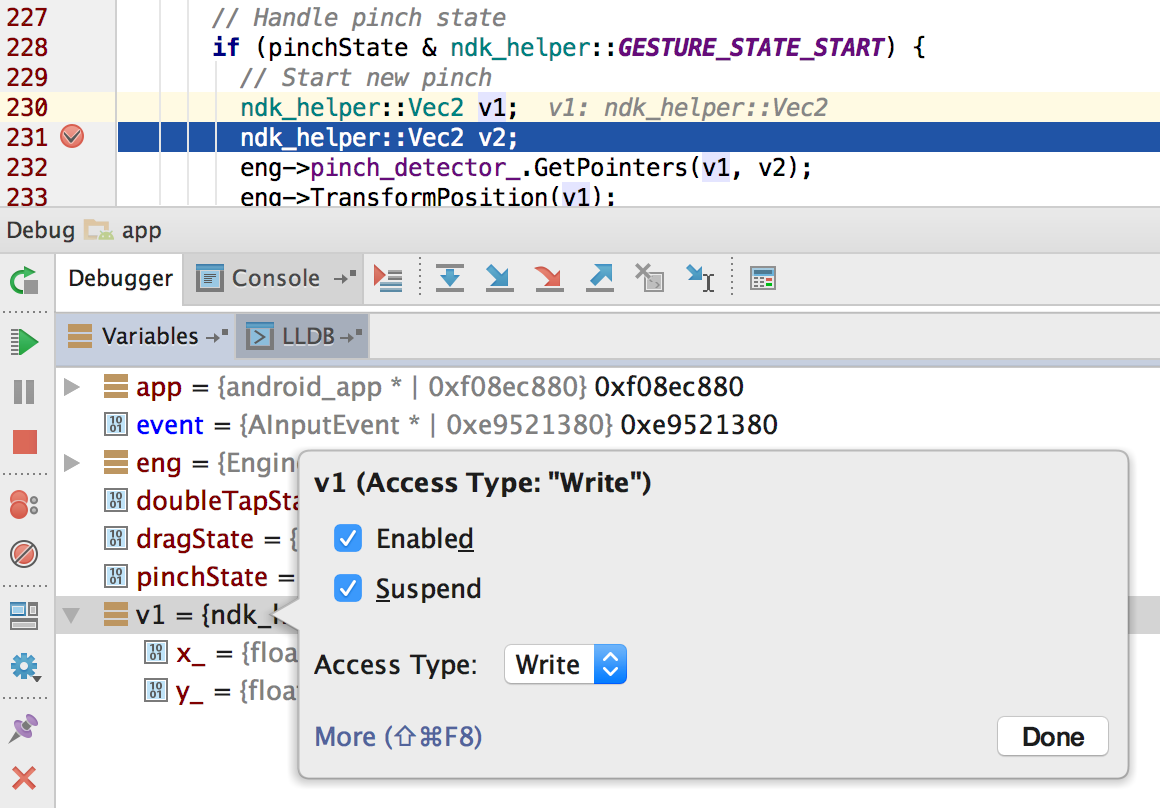
* Ваше целевое физическое устройство или эмулятор использует процессор x86 или x86\_64. Если ваше устройство использует процессор ARM, то вы должны выровнять границу адреса вашей переменной в памяти либо 4 байта для 32-разрядных процессоров, либо 8 байтов для 64-разрядных процессоров. Вы можете выровнять переменную в своем собственном коде, указав замедление переменной, как показано ниже: \_\_attribute\_\_((aligned(***num\_bytes***)))

// Для 64-битного ARM процессора INT my\_counter \_\_attribute\_\_ (( выровненный ( 8 )));

* Вы уже присвоили три или меньше точек наблюдения. Android Studio поддерживает только до четырех точек наблюдения на целевых устройствах x86 или x86\_64. Другие устройства могут поддерживать меньшее количество точек наблюдения.

Если вы соответствуете вышеуказанным требованиям, вы можете добавить точку наблюдения следующим образом:

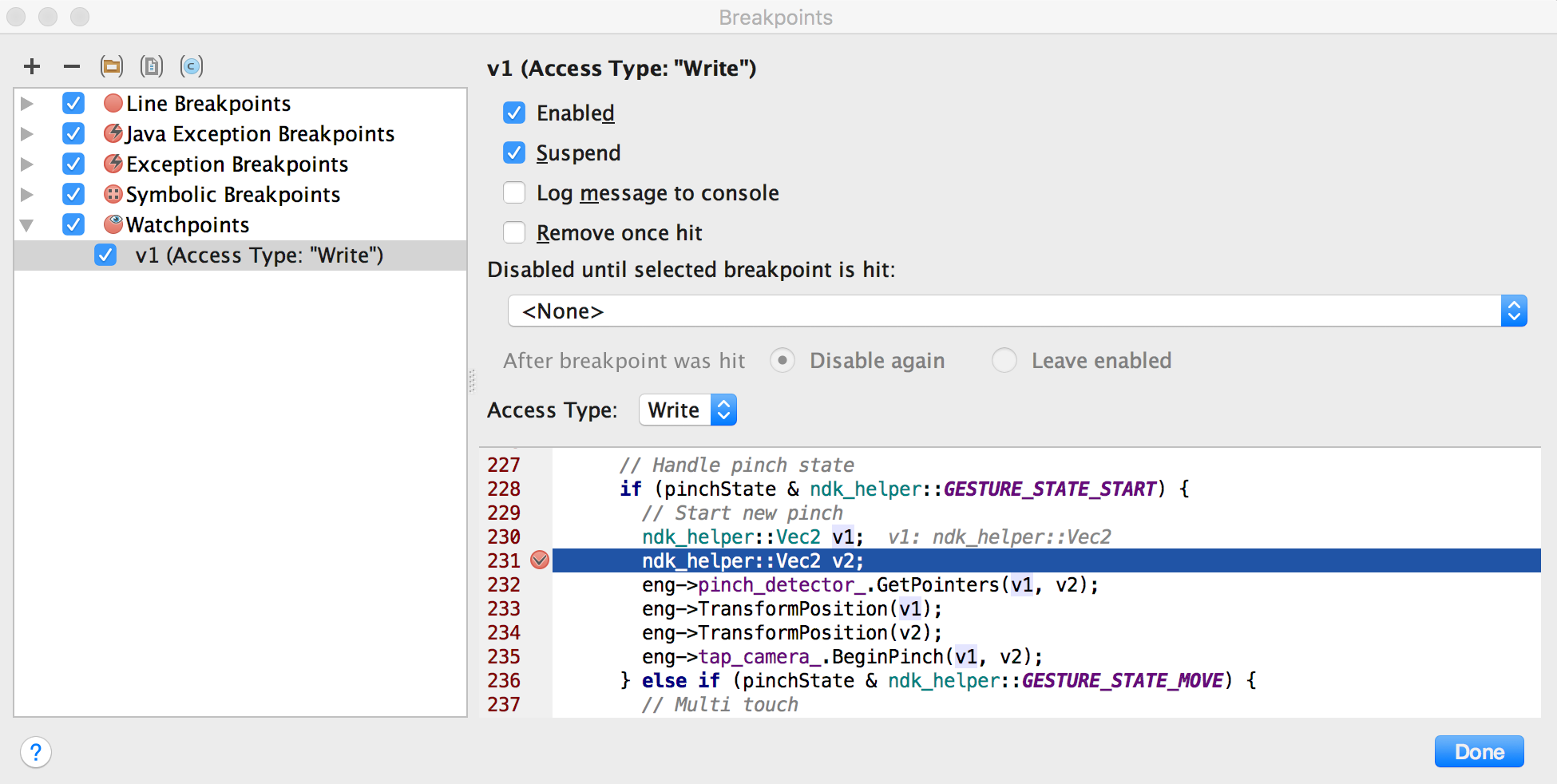
1. Пока ваше приложение приостановлено на точке останова, перейдите к панели « **Переменные»** в представлении сеанса LLDB.
2. Щелкните правой кнопкой мыши переменную, которая занимает блок памяти, которую вы хотите отслеживать, и выберите **Добавить точку наблюдения** . Появится диалоговое окно для настройки вашей точки наблюдения, как показано на рисунке 7.



**Рисунок 7.** Добавление точки наблюдения к переменной в памяти

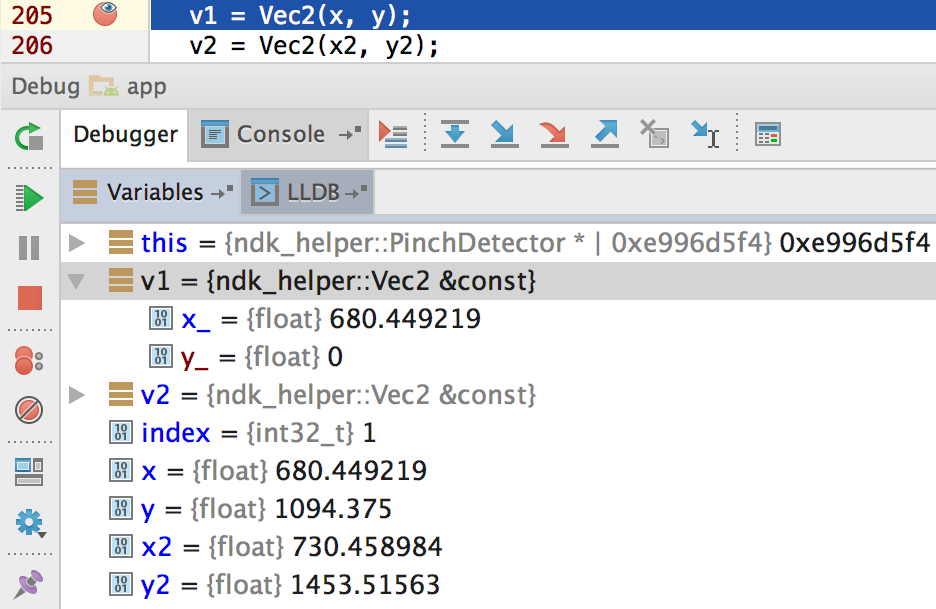
1. Настройте свою точку наблюдения с помощью следующих параметров:
   * **Включено:** Вы можете отменить выбор этого параметра, если хотите сказать, что Android Studio пока игнорирует точку наблюдения. Android Studio все еще сохраняет вашу точку наблюдения, чтобы вы могли получить к ней доступ позже в сеансе отладки.
   * **Приостановить.** По умолчанию система Android приостанавливает процесс приложения, когда он обращается к блоку памяти, назначенному для точки наблюдения. Вы можете отменить выбор этой опции, если вы не хотите, чтобы это поведение - это показывает дополнительные опции, которые вы можете использовать для настройки поведения, когда система взаимодействует с вашей точкой наблюдения: **записать сообщение в консоль** и **удалить [точку наблюдения] при нажатии** .
   * **Тип доступа:** выберите, должно ли ваше приложение запускать вашу точку наблюдения при попытке **чтения** или **записи** в блок памяти, который система выделяет для переменной. Чтобы активировать точку наблюдения при чтении или записи, выберите **Любой** .
2. Нажмите **Готово** .

Чтобы просмотреть все ваши точки наблюдения и настроить параметры точки наблюдения, нажмите « **Просмотреть точки останова»** https://developer.android.com/images/tools/as-viewbreakbutton.png в левой части окна « **Отладка»** . Появится диалоговое окно **Breakpoints** , как показано на рисунке 8.



**Рисунок 8.** Диалоговое окно Breakpoints перечисляет ваши текущие точки наблюдения и включает настройки поведения для каждого

После того, как вы добавите свою точку наблюдения, нажмите « **Возобновить программу»** https://developer.android.com/studio/images/debug/debugger_button_resume_2-0.png в левой части окна « **Отладка»,** чтобы возобновить процесс приложения. По умолчанию, если ваше приложение пытается получить доступ к блоку памяти, для которого вы установили точку наблюдения, система Android приостанавливает процесс вашего приложения, и https://developer.android.com/studio/images/debug/watchpoint-icon_2-2_2x.pngрядом со строкой кода, которую ваше приложение выполнило в последний раз, появляется значок точки наблюдения , как показано на рисунке 9 ,



**Рисунок 9.** Android Studio указывает строку кода, которую ваше приложение выполняет перед тем, как вызвать точку наблюдения

Просмотр и изменение формата отображения значения ресурса

В режиме отладки вы можете просматривать значения ресурсов и выбирать другой формат отображения переменных в вашем Java-коде. При отображенной вкладке « **Переменные** » и выбранном фрейме выполните следующие действия:

1. В списке **переменных** щелкните правой кнопкой мыши в любом месте строки ресурса, чтобы отобразить раскрывающийся список.
2. В раскрывающемся списке выберите « **Просмотреть как»** и выберите нужный формат.

Доступные форматы зависят от типа данных выбранного вами ресурса. Вы можете увидеть один или несколько из следующих параметров:

* + **Класс:** Показать определение класса.
  + **toString:** Показать формат строки.
  + **Object:** Показать определение объекта (экземпляр класса).
  + **Массив:** отображение в формате массива.
  + **Отметка времени:** отображение даты и времени следующим образом: гггг-мм-дд чч: мм: сс.
  + **Авто:** Android Studio выбирает лучший формат в зависимости от типа данных.
  + **Двоичный** код **:** отображение двоичного значения с использованием нулей и единиц.
  + **MeasureSpec:** значение, переданное от родителя выбранному дочернему элементу. См [MeasureSpec.](https://developer.android.com/reference/android/view/View.MeasureSpec.html)
  + **Шестнадцатеричный:** отображать в виде шестнадцатеричного значения.
  + **Примитив:** отображение в виде числового значения с использованием примитивного типа данных.
  + **Целое число:** отображение числового значения типа [Integer](https://developer.android.com/reference/java/lang/Integer.html).

Вы можете создать пользовательский формат (рендер типа данных) следующим образом:

1. Щелкните правой кнопкой мыши значение ресурса.
2. Выберите **Просмотр как** .
3. Выберите **Создать** . Откроется диалоговое окно « Средства **визуализации типов данных Java»**
4. Следуйте инструкциям в средствах [визуализации типов данных Java](https://www.jetbrains.com/help/idea/2019.1/java-type-renderers.html) .