



Debug - режим отладки приложения

В разработке процессы программирования и дебага поочередно сменяют друг друга: сначала мы пишем код, запускаем, в какой-то момент осознаем, что что-то пошло не так, начинаем дебажить, локализуем проблему, делаем поправку, повторяем заново. На практике довольно малый процент случаев, когда все получается с первого раза, практически всегда случаются моменты, которые текущая логика программы не обрабатывает. И эти моменты нужно обработать, чтобы не повредить нежную психику пользователя.

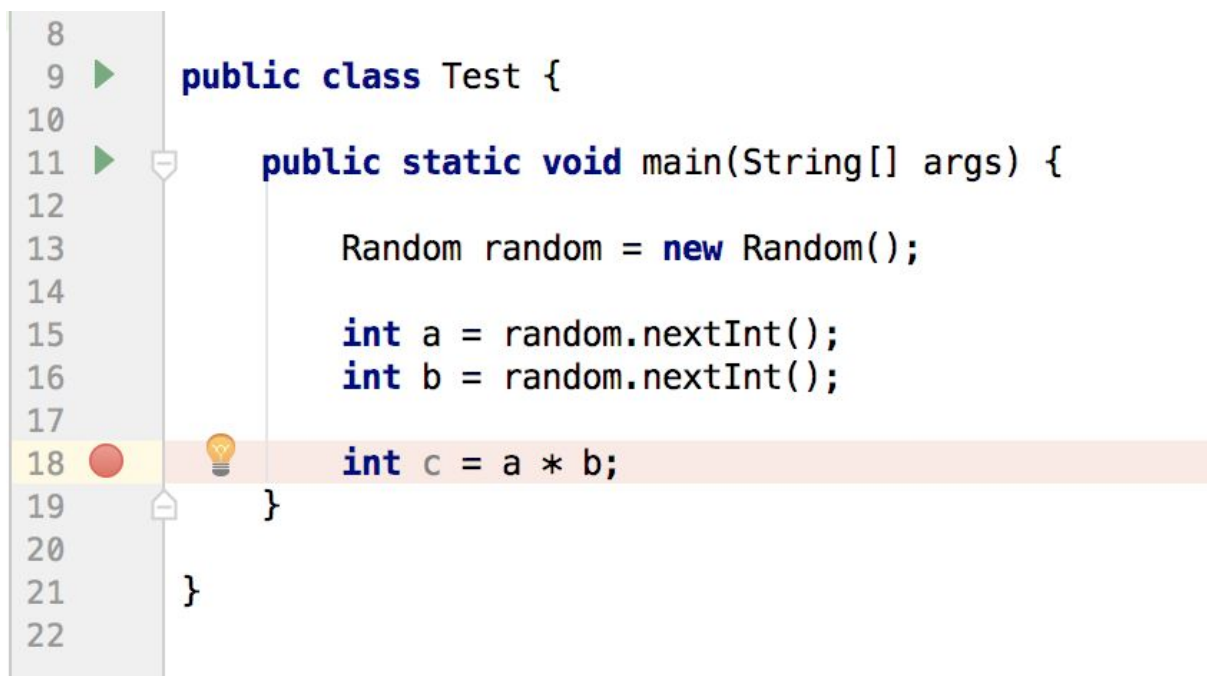
Режим дебага позволяет производить отладку приложения во время его работы. Во время отладки мы получаем возможность отслеживать изменения любой переменной, отлавливать ошибки и следить за ходом выполнения программы.

Запустить дебаг режим можно с помощью кнопки  или по нажатию клавиш Shift + F9. Также режим отладки мы можем запустить в уже запущенном приложении, для

этого можно использовать кнопку  и выбрать нужный нам процесс из диалога. Этот способ можно комбинировать с кодом `Debug.WaitForDebugger()`. Этот код не позволит приложению выполнять работу дальше пока вы не подключите дебаггер.

Чтобы ход выполнения программы был остановлен и режим управления перешел к программисту, необходимо поставить точку останова или breakpoint, в том участке кода, где мы хотим начать изучение программы.

Для того чтобы поставить точку останова нужно кликнуть на необходимую строку левее поля ввода



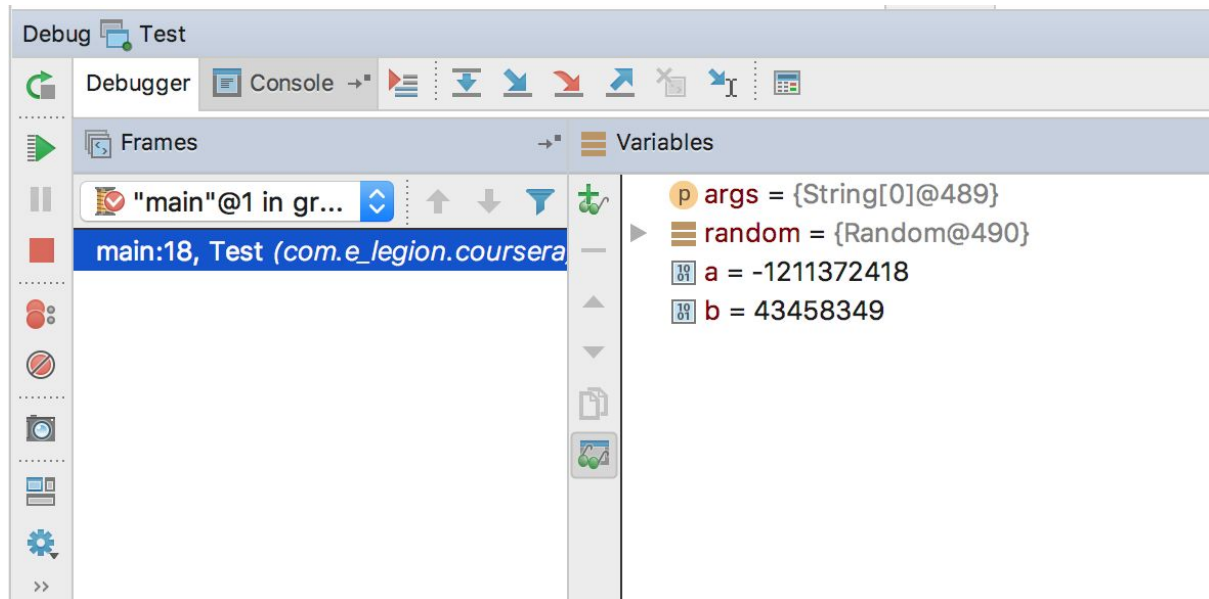
```
8
9  ▶ public class Test {
10
11  ▶ public static void main(String[] args) {
12
13      Random random = new Random();
14
15      int a = random.nextInt();
16      int b = random.nextInt();
17
18  ● int c = a * b;
19  }
20
21 }
22
```

The screenshot shows a code editor with a C# class named 'Test'. The 'main' method is defined. A breakpoint, represented by a red circle, is set on line 18, which contains the code 'int c = a * b;'. The line is highlighted in light orange. The left margin shows line numbers from 8 to 22. There are green play icons on lines 9 and 11, and a lightbulb icon next to the breakpoint on line 18.

Напишем код, как на примере выше. Важный момент - метод `main()`. Он позволит нам запустить этот код отдельно от всего остального приложения. Так как `a` и `b` инициализированы случайным образом, мы понятия не имеем, что в них записано.

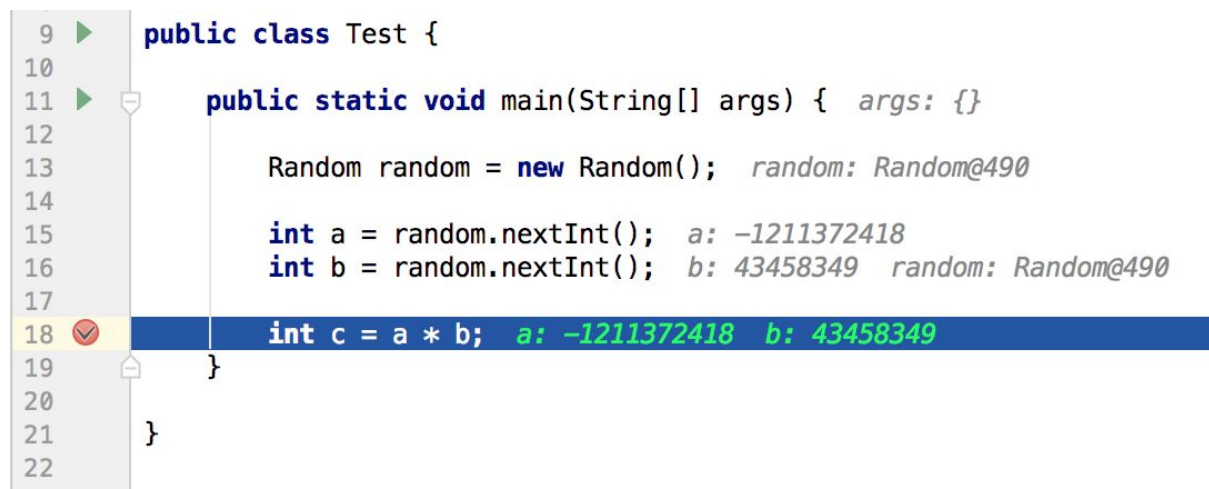
Поставим точку останова на 18 строке и запустим приложение в режиме отладки.

При достижении программы точки останова, открывается вкладка Debug.



Теперь мы знаем конкретные значения.

К слову, текстовый редактор тоже преобразился, в нем также отображаются соответствующие значения.



Разберем подробнее панель управления режимом дебага



Кнопки навигации.

Ход программы выполняется по нажатию на одну из этих кнопок, они позволяют выполнять программу построчно, войти в текущий метод, выйти из него, или выполнить программу до курсора.

Кнопки управления процессом дебага.



перезапустить

продолжить выполнение до следующей точки останова

приостановить - если процесс запущен

остановить

открыть окно со всеми точками останова

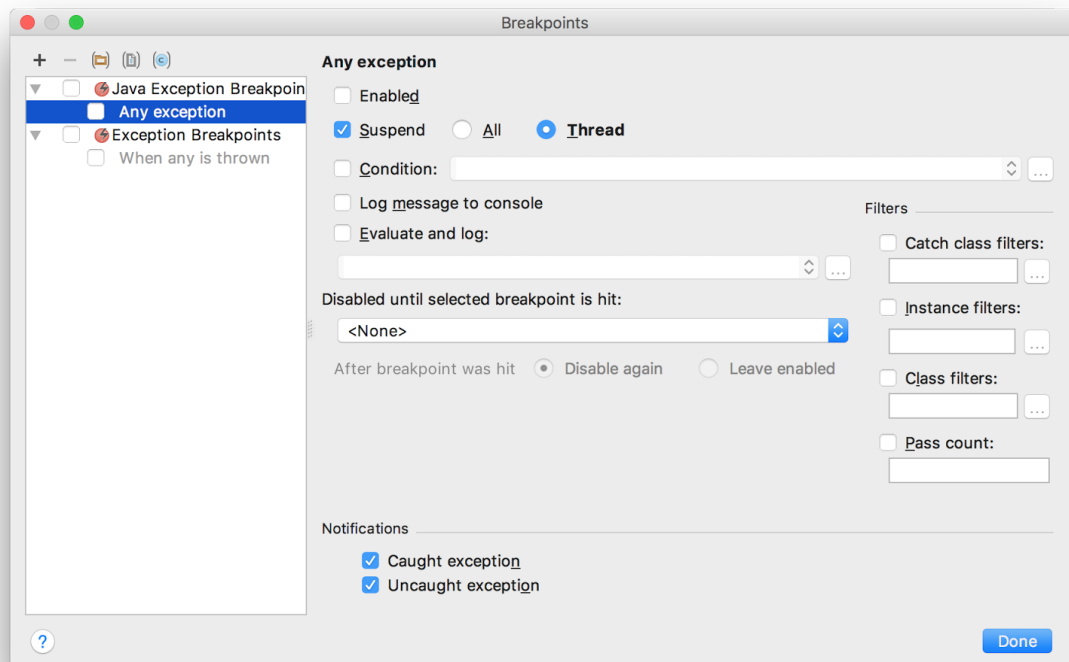
отключить точки останова

показать дампы потоков

отменить изменения в верстке и вернуть к дефолтному состоянию

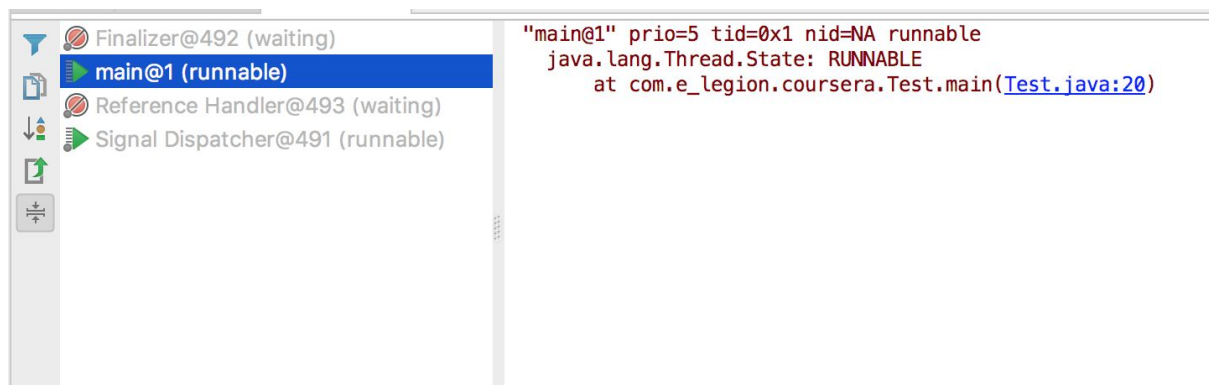
настройки

Окно управления точками останова



Здесь можно изменить настройки точки останова, добавить условия или фильтры.

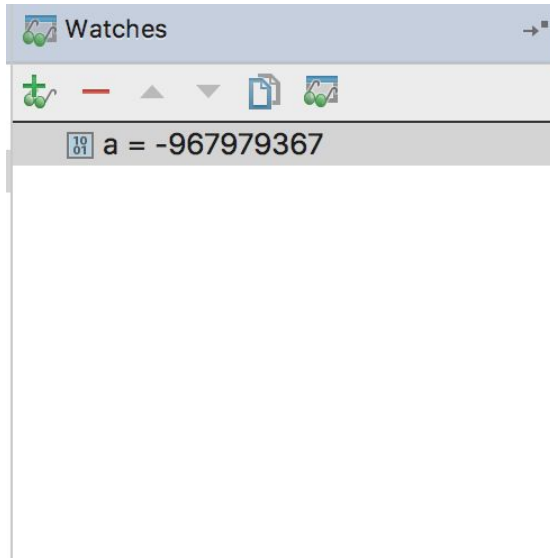
Дамп потоков.



Потоки отсортированы по состояниям: рабочий, приостановленный, ожидающий разблокировки и т.д.

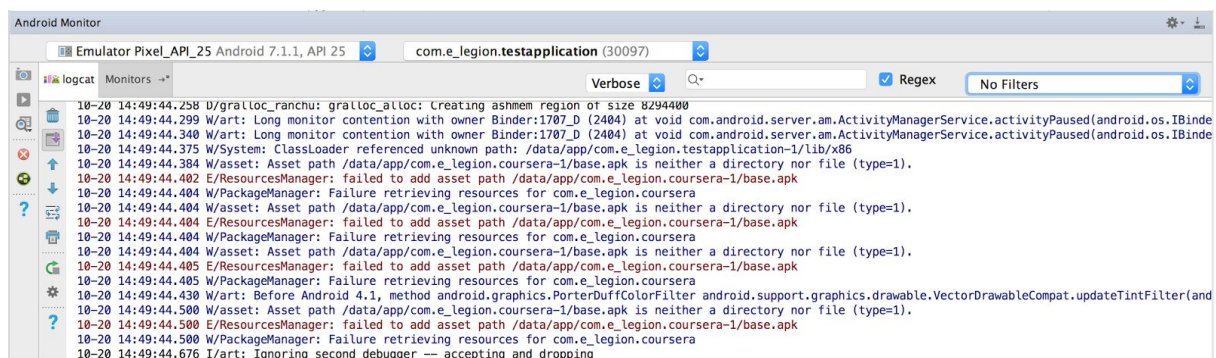
Окно Watches

Щелкнув правой кнопкой на любой переменной, ее можно добавить в Watches или Отслеживаемые переменные. Теперь исполнение программы находится в скоупе этой переменной, ее изменения будут видны в этом окне. Пользуясь этим механизмом, можно хранить в одном месте несколько переменных из разных скоупов и соответственно отслеживать их изменения.



Знакомство с инструментами логгирования

Логи - это сообщения, сохраненные в журнал сообщений приложения, во время его работы. В Android Studio логи отображаются при помощи инструмента logcat. (Находится внутри Android Monitor во Android Studio v.2)



Для того чтобы вывести сообщение в логат, нужно воспользоваться одним из статических методов класса Log.

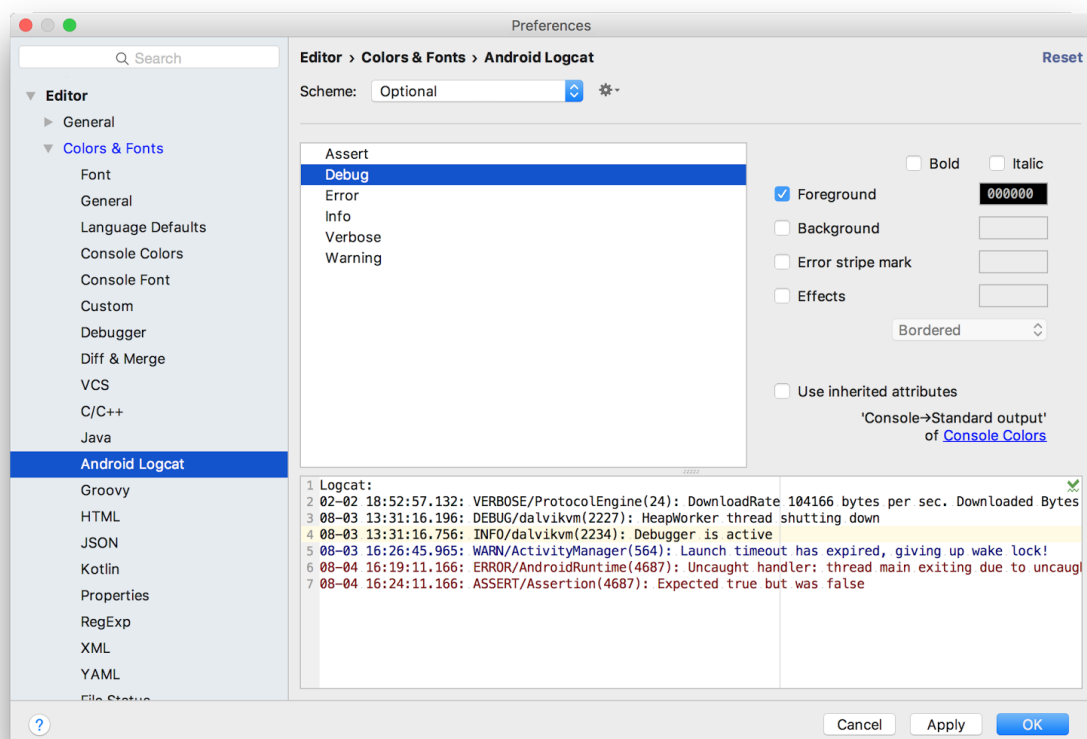
Например, выведем лог в методе onCreate()

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    Log.d(TAG, "onCreate");  
}
```

d - это приоритет сообщения, всего их 6 видов:

- V — Verbose (низший приоритет)
- D — Debug
- I — Info
- W — Warning
- E — Error
- A — Assert

В настройках logcat можно изменить стиль для каждого приоритета



“TAG” - это тег по которому можно найти наше сообщение, также по тегу можно фильтровать выводимые сообщения.

Для удобства, в качестве тега можно использовать название класса:

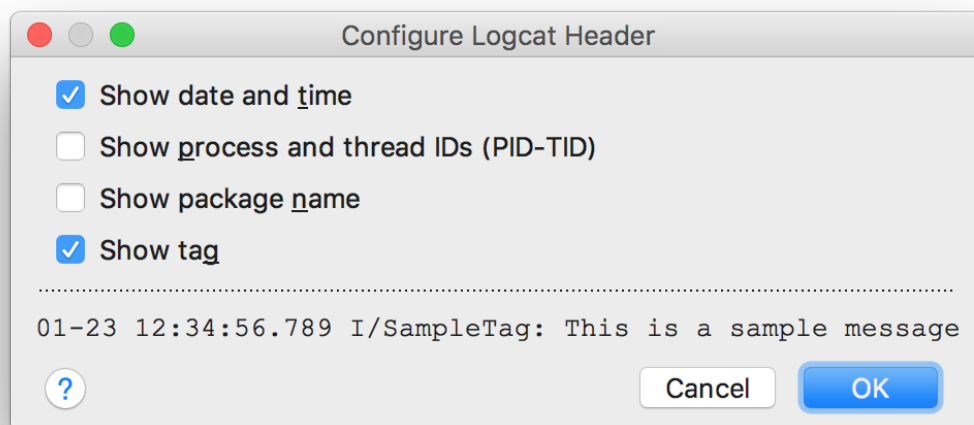
```
private final String TAG = getClass().getSimpleName();
```

Здесь мы фильтруем выводимые сообщения по тегу, также их можно фильтровать по приоритету или просто выводить сообщения содержащие определенное слово.

Нажав на комбинацию ctrl + F в logcat, можно вызвать окно поиска, с помощью которого можно найти нужные вам сообщения.

На панели слева располагаются кнопки управления выводимыми сообщениями

С их помощью можно очистить logcat, выставить курсор на последней строке, пройти по stack trace, разделять выводимые сообщения на строки, открыть настройки, сделать скриншот приложения или сделать видео отображаемого на экране. На экране настроек можно изменить выводимую информацию.



В большинстве случаев логи нужны нам только во время дебага приложения, и мы не хотим отображать их в релизной версии. Для этого, перед выводом лога, нужно проверять BuildConfig флаг.

Выглядит это так:

```
if (BuildConfig.DEBUG) {  
    Log.d(TAG, "onCreate");  
}
```

Для инкапсуляции проверки, можно сделать класс `Logger` со статическими методами вывода логов

```
public class Logger {  
  
    public static void debug(String tag, String message, Object...  
args) {  
        if (BuildConfig.DEBUG) {  
            if (TextUtils.isEmpty(message)) {  
                return;  
            }  
            if (args.length > 0) {  
                message = String.format(message, args);  
            }  
            Log.d(tag, message);  
        }  
    }  
}
```

Теперь мы можем просто написать

```
Logger.debug(TAG, "onCreate");
```

Результат будет виден только в сборке для дебага. Подобным образом можно обернуть все методы класса `Log`.