Модульные тесты

<https://developer.android.com/codelabs/android-training-unit-tests#0>

Введение

Тестирование кода может помочь вам выявлять ошибки на ранних этапах разработки, когда устранение ошибок является наименее затратным. По мере того, как ваше приложение становится больше и сложнее, тестирование повышает надежность вашего кода. С помощью тестов в коде вы можете тестировать небольшие части вашего приложения изолированно, и вы можете тестировать автоматизируемыми и повторяемыми способами.

Android Studio и библиотека поддержки тестирования Android поддерживают несколько различных типов тестов и сред тестирования. В этом практическом занятии вы исследуете встроенные функции тестирования Android Studio и узнаете, как писать и запускать локальные модульные тесты.

*Локальные модульные тесты* - это тесты, которые скомпилированы и выполняются полностью на вашем локальном компьютере с виртуальной машиной Java (JVM). Вы используете локальные модульные тесты для тестирования частей вашего приложения, которым не требуется доступ к платформе Android, устройству или эмулятору на базе Android, например к внутренней логике. Вы также используете локальные модульные тесты для тестирования частей вашего приложения, для которых вы можете создавать поддельные («имитирующие» или заглушки) объекты, которые делают вид, что ведут себя как эквиваленты фреймворка.

Модульные тесты написаны с помощью JUnit, общей среды модульного тестирования для Java.

## Что ты будешь делать

* Запустите начальные тесты в приложении SimpleCalc.
* Добавьте больше тестов в приложение SimpleCalc.
* Запустите модульные тесты, чтобы увидеть результаты.

3. Задача 1. Изучите и запустите CalculatorTest.

Вы пишете и запускаете свои тесты (как модульные, так и инструментальные) внутри Android Studio вместе с кодом для своего приложения. Каждый новый проект Android включает в себя базовые образцы классов для тестирования, которые вы можете расширить или заменить для собственных нужд.

В этой задаче вы вернетесь в приложение SimpleCalc, которое включает базовый класс модульного тестирования.

# Тестируйте приложения на Android   Часть [Android Jetpack](https://developer.android.com/jetpack) .

Тестирование вашего приложения - неотъемлемая часть процесса разработки приложения. Последовательно выполняя тесты для вашего приложения, вы можете проверить правильность, функциональное поведение и удобство использования вашего приложения, прежде чем выпускать его публично.

Тестирование также дает вам следующие преимущества:

* **Быстрая обратная связь** при сбоях.
* **Раннее обнаружение сбоев** в цикле разработки.
* **Более безопасный рефакторинг кода** , позволяющий оптимизировать код, не беспокоясь о регрессиях.
* **Стабильная скорость разработки** , помогающая минимизировать технический долг.

Чтобы узнать больше о типах тестов, которые можно использовать при разработке приложения, см. [Основы тестирования](https://developer.android.com/training/testing/fundamentals) и просмотрите встроенное видео с Google I / O 2019.

Модульная разработка\*\*\*

<https://developer.android.com/codelabs/on-demand-dynamic-delivery#0>

Примеры тестирования:\*\*\*

<https://github.com/android/testing-samples>

**Халтуры и недоделки:**

**Задание 5:**

**Задача 1.** Деление на ноль всегда стоит проверить, потому что это особый случай в арифметике. Как бы вы могли изменить приложение, чтобы более изящно обрабатывать деление на ноль? Чтобы выполнить эту задачу, начните с теста, который покажет, каким должно быть правильное поведение.

Удалите divTwoNumbersZero()метод из CalculatorTestи добавьте новый вызываемый модульный тест, divByZeroThrows()который проверяет div()метод со вторым аргументом, равным нулю, с ожидаемым результатом как IllegalArgumentException.class. Этот тест будет пройден, и в результате он продемонстрирует, что любое деление на ноль приведет к этому исключению.

После того как вы научитесь писать код для [Exception](https://developer.android.com/reference/java/lang/Exception.html" \t "_blank)обработчика, ваше приложение сможет корректно обработать это исключение, например, отобразив [Toast](https://developer.android.com/reference/android/widget/Toast.html" \t "_blank)сообщение пользователю, чтобы изменить операнд 2 с нуля на другое число.

От ДО:

Деление на ноль плавающих чисел дает самое большое плавающее число и это не ошибка.

Деление на ноль целых чисел возбуждает исключение ….. и программа аварийное заканчивается, если  
нет перехвата и обработки этого исключения.

Разобрать как делать

**Проблема 2:** иногда сложно изолировать единицу кода от всех его внешних зависимостей. Вместо того, чтобы упорядочивать свой код сложными способами, чтобы его было легче протестировать, вы можете использовать фиктивный фреймворк для создания фальшивых («фиктивных») объектов, которые притворяются зависимостями. Исследования [Mockito](http://mockito.org/" \t "_blank) рамки, и узнать , как установить его в Android Studio. Напишите тестовый класс для calcButton()метода в SimpleCalc и используйте Mockito для имитации контекста Android, в котором будут выполняться ваши тесты.

[...как в JUnit проверять ожидаемые исключения? • А хотите, я расскажу вам... (barancev.github.io)](http://barancev.github.io/junit-catch-throwable/)

<https://stackoverflow.com/questions/156503/how-do-you-assert-that-a-certain-exception-is-thrown-in-junit-4-tests>

Задание 9:

**9. Домашнее задание**

В этом разделе перечислены возможные домашние задания для студентов, которые работают с этой кодовой лабораторией в рамках курса, проводимого инструктором. Инструктор должен сделать следующее:

При необходимости назначьте домашнее задание.

Сообщите учащимся, как сдавать домашние задания.

Оценивайте домашние задания.

Преподаватели могут использовать эти предложения столько, сколько захотят, и могут свободно назначать любое другое домашнее задание, которое они считают подходящим.

Если вы работаете над этой лабораторией кода самостоятельно, не стесняйтесь использовать эти домашние задания, чтобы проверить свои знания.

Создайте и запустите приложение

Откройте приложение SimpleCalc из практического руководства по использованию отладчика. Вы собираетесь добавить на макет кнопку POW . Кнопка вычисляет первый операнд в степени второго операнда. Например, для операндов 5 и 4 приложение вычисляет 5 в степени 4 или 625.

Прежде чем писать реализацию кнопки питания, подумайте о том, какие тесты вы, возможно, захотите провести, используя этот расчет. Какие необычные значения могут иметь место в этом расчете?

Обновите Calculatorкласс в приложении, чтобы включить pow()метод. Подсказка: обратитесь к документации по java.lang.Mathклассу.

Обновите MainActivityкласс, чтобы подключить POWButton к вычислению.

**Теперь напишите каждый из следующих тестов для вашего pow()метода. Запускайте свой набор тестов каждый раз, когда вы пишете тест, и при необходимости исправляйте исходный расчет в своем приложении: - Оставь это Лене**

Тест с положительными целочисленными операндами.

Тест с отрицательным целым числом в качестве первого операнда.

Тест с отрицательным целым числом в качестве второго операнда.

Тест с 0 в качестве первого операнда и положительным целым числом в качестве второго операнда.

Тест с 0 в качестве второго операнда.

Тест с 0 в качестве первого операнда и -1 в качестве второго операнда. (Подсказка: обратитесь к документации Double.POSITIVE\_INFINITY.)

Тест с -0 в качестве первого операнда и любого отрицательного числа в качестве второго операнда.

**Убедитесь, что в приложении есть следующие функции:**

Он отображает POW,Button который обеспечивает экспоненциальный расчет («степень»).

Реализация MainActivityвключает обработчик кликов для POWButton .

Реализация Calculatorвключает pow()метод, выполняющий расчет.

CalculatorTest()Способ включает в себя отдельные методы испытаний для pow()метода в Calculatorклассе , которые выполняют тесты для отрицательных и 0 операндов, а также для случая 0 и -1 в качестве операндов.