**Практическая работа**

**Работа с Test Driven Development (TDD) в Android с помощью Kotlin**

Теоретические положения

TDD — это методология программного обеспечения, в которой разработчики пишут тесты для фрагментов кода до написания самого кода. Если вы не знакомы с этим модным словечком, пожалуйста, сначала прочитайте [эту](https://www.geeksforgeeks.org/test-driven-development-tdd/) статью. В TDD тест должен быть написан, а затем написан код. Тест может провалиться, код будет обновлен, снова он может не получиться, снова код будет изменен, и процесс будет продолжаться до тех пор, пока код не пройдет тест. Преимуществ реализации данной методики очень много:

* Таким образом, в основном код пишется так, чтобы он соответствовал коду теста. Таким образом, разработчик получает возможность сделать продукт, который соответствует ожиданиям от продукта. Реализация может произойти гораздо быстрее, потому что все заранее спланировано.
* Существует определенный уровень уверенности, поскольку тесты заранее определены, и если код проходит тест, это означает, что код работает так, как ожидалось.
* Если тесты написаны после кодирования, то разработчик может неосознанно написать тесты, которые будут работать в обход кода, делая продукт, отличный от того, что планировалось ранее. С TDD такой проблемы не возникает.

Прочитав преимущества, можно подумать, что эта методология является лучшей, каждая фирма должна взять ее на вооружение, чтобы сделать свой продукт лучшим. Но на практике принятие и внедрение этой методологии может быть довольно сложным делом. Могут быть случаи, когда вы пишете код, который даже не компилируется, потому что тесты пишутся до самого кода, что приведет к ошибке во время компиляции. Написание тестов, которые не компилируются, является серьезным барьером для входа в TDD.

**Эффективное внедрение TDD**

Независимо от сложности реализации TDD, разработчики все равно обходят такой метод. Давайте посмотрим, как можно эффективно реализовать TDD. Для этого мы создадим простое приложение для Android с одним действием на Kotlin.

**Задание**

Давайте создадим *модель StudentDetailViewModel. Модель Student* будет условно показывать некоторую информацию о студенте в конкретном классе. Информация будет включать имя студента, номер списка, дату рождения, адрес и пол.

**Шаг 1: Создайте новый проект в Android Studio**

Чтобы создать новый проект в Android Studio, обратитесь к разделу [Как создать/начать новый проект в Android Studio](https://www.geeksforgeeks.org/android-how-to-create-start-a-new-project-in-android-studio/). Обратите внимание, что в качестве языка программирования выберите **Kotlin**.

**Шаг 2: Добавление зависимостей**

Чтобы использовать TDD, нам нужно добавить необходимые зависимости в файл build.gradle (app). Добавьте следующие зависимости:

*dependencies {  
implementation 'androidx.lifecycle:lifecycle-livedata-core-ktx:2.5.1'  
testImplementation 'junit:junit:4.13.2'  
testImplementation 'org.mockito:mockito-core:3.11.2'  
testImplementation 'org.robolectric:robolectric:4.6.1'  
}*

**Шаг 3: Письменные тесты**

Теперь, когда мы добавили необходимые зависимости, давайте напишем наш первый тестовый случай для *StudentDetailViewModel*. Создайте новый файл Kotlin в каталоге *src/test/java* и назовите его *StudentDetailViewModelTest*. В классе *StudentDetailViewModelTest* давайте напишем тестовый случай, который проверяет, правильно ли создан экземпляр ViewModel.

* Kotlin

|  |
| --- |
| @RunWith(RobolectricTestRunner::class)  class StudentDetailViewModelTest {        private lateinit var viewModel: StudentDetailViewModel        @Before      fun setUp() {          viewModel = StudentDetailViewModel()      }        @Test      fun testViewModelInstantiation() {          assertNotNull(viewModel)      }  } |

В приведенном выше коде мы создали экземпляр *StudentDetailViewModel* в методе *setUp()* и проверили, что экземпляр не равен null в методе *testViewModelInstantiation().*

**Шаг 4: Написание рабочего кода**

Теперь, когда мы написали наш первый тестовый сценарий, давайте напишем производственный код, который пройдет этот тестовый случай. Создайте новый файл Kotlin в каталоге *src/main/java* и назовите его *StudentDetailViewModel*. Добавьте следующий код:

* Kotlin

|  |
| --- |
| class StudentDetailViewModel {        private val \_studentInfo = MutableLiveData<Student>()        val studentInfo: LiveData<Student>          get() = \_studentInfo        fun loadStudentInfo(studentId: String) {          val student = Student(              name =  "Ideal Name",              rollNumber = "20",              dateOfBirth = "2000-01-01",              address ="Ideal Address",              gender = "Male"          )          \_studentInfo.value = student      }    } |

Определение класса **Student** Data:

* Kotlin

|  |
| --- |
| data class Student(      val name: String,      val rollNumber: String,      val dateOfBirth: String,      val address: String,      val gender: String  ) |

**Шаг 5: Написание дополнительных тестов**

Давайте добавим больше тестовых случаев в класс *StudentDetailViewModelTest* для проверки функциональности StudentDetailViewModel:

* Kotlin

|  |
| --- |
| @RunWith(RobolectricTestRunner::class)  class StudentDetailViewModelTest {        private lateinit var viewModel: StudentDetailViewModel        @Before      fun setUp() {          viewModel = StudentDetailViewModel()      }        @Test      fun testViewModelInstantiation() {          assertNotNull(viewModel)      }        @Test      fun testLoadStudent() {          // Set up          val studentDetailViewModel = StudentDetailViewModel()          val expectedName = "Ideal Name"          val expectedRollNo = "20"          val expectedDob = "2000-01-01"          val expectedAddress = "Ideal Address"          val expectedGender = "Male"            val rollNo = "20"              // Exercise          studentDetailViewModel.loadStudentInfo(rollNo)            // Verify          assertEquals(expectedName, studentDetailViewModel.studentInfo.value?.name ?: "")          assertEquals(expectedRollNo, studentDetailViewModel.studentInfo.value?.rollNumber)          assertEquals(expectedDob, studentDetailViewModel.studentInfo.value?.dateOfBirth ?: "")          assertEquals(expectedAddress, studentDetailViewModel.studentInfo.value?.address ?: "")          assertEquals(expectedGender, studentDetailViewModel.studentInfo.value?.gender ?: "")      }  } |

**Шаг 6: Выполнение теста**

Чтобы запустить тесты в Android Studio, вы можете выполнить следующие действия:

1. В представлении Проект перейдите в каталог, содержащий тестовые файлы.
2. Щелкните правой кнопкой мыши файл, который вы хотите запустить, и выберите «Запустить «Тесты в...». из контекстного меню.
3. Средство запуска тестов запустит и выполнит ваши тесты

После завершения тестов вы можете просмотреть результаты на вкладке «**Выполнить**» в нижней части экрана.

**Результат теста**

Если тест пройден, вы увидите сообщение о прохождении теста в консоли **выполнения**:

https://media.geeksforgeeks.org/wp-content/uploads/20230415224859/Screenshot-from-2023-04-15-22-48-21.png

Вы можете добавить больше тестов в *StudentDetailViewModelTest.*