30 Тестирование приложения Cupcake

[1. Введение](https://developer.android.com/codelabs/basic-android-kotlin-compose-test-cupcake?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-test-cupcake#0)

В лаборатории кода [«Навигация между экранами с помощью Compose»](https://developer.android.com/codelabs/basic-android-kotlin-compose-navigation) вы узнали, как добавить навигацию в приложение Compose с помощью компонента Jetpack Navigation Compose.

Приложение Cupcake имеет несколько экранов для навигации и множество действий, которые может выполнять пользователь. Это приложение предоставляет вам прекрасную возможность отточить свои навыки автоматического тестирования! В этой лаборатории кода вы напишете несколько тестов пользовательского интерфейса для приложения Cupcake и узнаете, как добиться максимального покрытия тестами.

Предварительные условия

* Знакомство с языком Kotlin, включая типы функций, лямбда-выражения и функции области видимости.
* Завершение [навигации между экранами с помощью Compose](https://developer.android.com/codelabs/basic-android-kotlin-compose-navigation) codelab.

Что вы узнаете

* Протестируйте компонент Jetpack Navigation с помощью Compose.
* Создайте согласованное состояние пользовательского интерфейса для каждого теста пользовательского интерфейса.
* Создайте вспомогательные функции для тестов.

Что ты построишь

* Тесты пользовательского интерфейса приложения Cupcake

Что вам понадобится

* Последняя версия Android Studio
* Подключение к Интернету для загрузки стартового кода.

## [2. Загрузите стартовый код](https://developer.android.com/codelabs/basic-android-kotlin-compose-test-cupcake?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-test-cupcake#1)

**URL стартового кода:**

[https://github.com/google-developer-training/basic-android-kotlin-compose-training-cupcake](https://github.com/google-developer-training/basic-android-kotlin-compose-training-cupcake/tree/navigation)

**Название филиала со стартовым кодом:**navigation

1. В Android Studio откройте basic-android-kotlin-compose-training-cupcakeпапку.
2. Откройте код приложения Cupcake в Android Studio.

[3. Настройте Cupcake для UI-тестов](https://developer.android.com/codelabs/basic-android-kotlin-compose-test-cupcake?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-test-cupcake#2)

Добавьте androidTestзависимости

Инструмент сборки Gradle позволяет добавлять зависимости для определенных модулей. Эта функциональность предотвращает ненужную компиляцию зависимостей. Вы уже знакомы с implementationконфигурацией включения зависимостей в проект. Вы использовали это ключевое слово для импорта зависимостей в build.gradle.ktsфайл модуля приложения. Использование implementationключевого слова делает эту зависимость доступной для всех исходных наборов в этом модуле; на этом этапе курса вы приобрели опыт работы с наборами исходных текстов main, testи .androidTest

Тесты пользовательского интерфейса содержатся в собственных исходных наборах, называемых androidTest. Зависимости, необходимые только для этого модуля, не нужно компилировать для других модулей, например модуля main, в котором содержится код приложения. При добавлении зависимости, которая используется только тестами пользовательского интерфейса, используйте androidTestImplementationключевое слово, чтобы объявить зависимость в файле модуля приложения build.gradle.kts. Это гарантирует, что зависимости тестов пользовательского интерфейса компилируются только при запуске тестов пользовательского интерфейса.

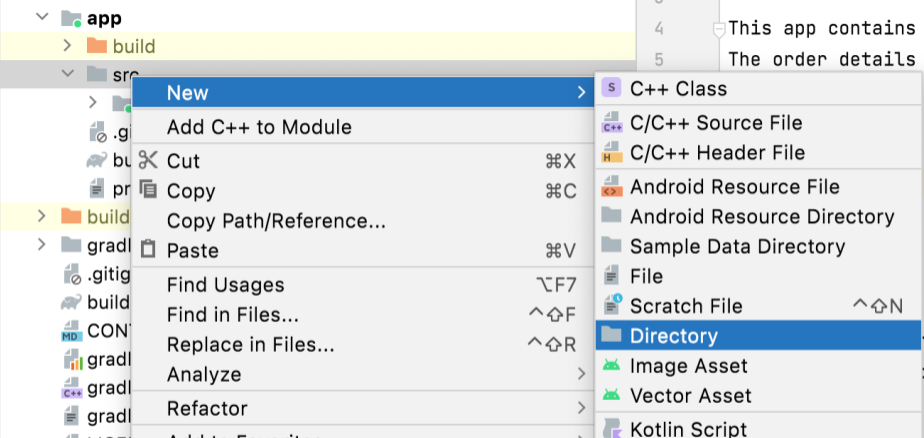
Выполните следующие шаги, чтобы добавить зависимости, необходимые для написания тестов пользовательского интерфейса:

1. Откройте build.gradle.kts(Module :app)файл.
2. Добавьте следующие зависимости в dependenciesраздел файла:

androidTestImplementation(platform("androidx.compose:compose-bom:2023.05.01"))  
androidTestImplementation("androidx.compose.ui:ui-test-junit4")  
androidTestImplementation("androidx.navigation:navigation-testing:2.6.0")  
androidTestImplementation("androidx.test.espresso:espresso-intents:3.5.1")  
androidTestImplementation("androidx.test.ext:junit:1.1.5")

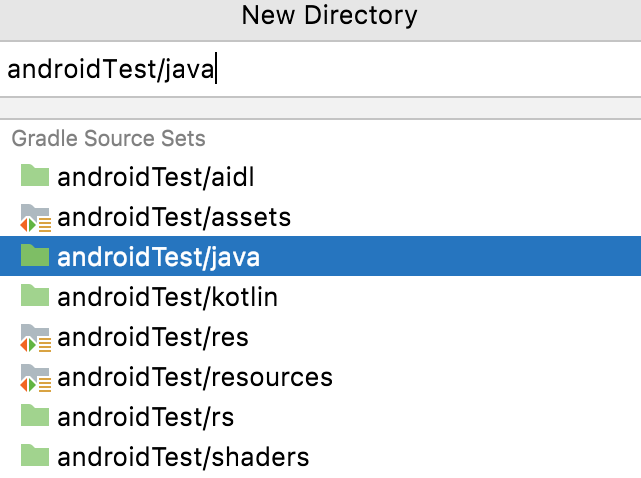
Создайте каталог тестирования пользовательского интерфейса.

1. Щелкните правой кнопкой мыши srcкаталог в представлении проекта и выберите **«Создать»** > **«Каталог»** .



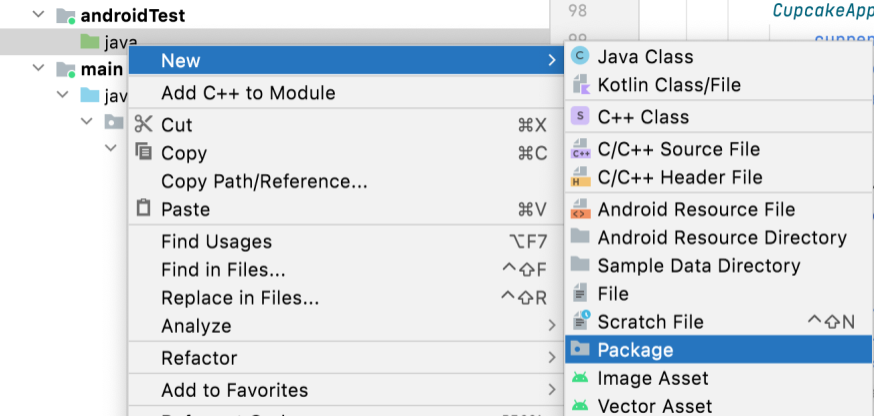
1. Выберите опцию **androidTest/java** .

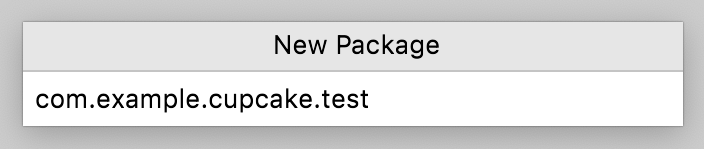
**Примечание.** Возможно, вам придется прокрутить вниз, чтобы найти эту опцию.



Создайте тестовый пакет

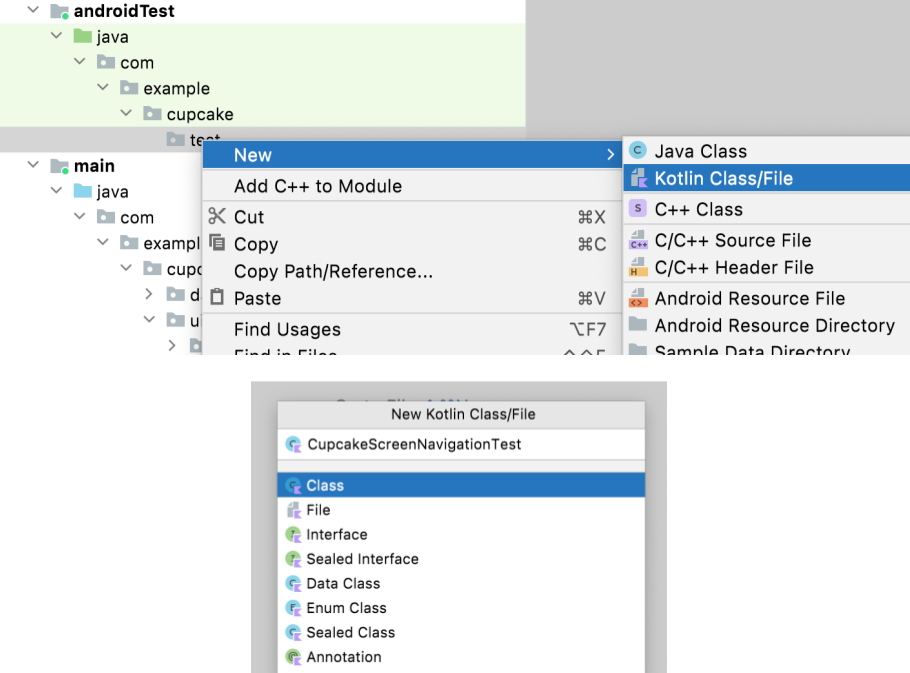
1. Щелкните правой кнопкой мыши androidTest/javaкаталог в окне проекта и выберите **«Создать»** > **«Пакет»** .



1. Назовите пакет **com.example.cupcake.test** .

Создайте класс теста навигации.

В testкаталоге создайте новый класс Kotlin с именем CupcakeScreenNavigationTest.



## [4. Настройте навигационный хост](https://developer.android.com/codelabs/basic-android-kotlin-compose-test-cupcake?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-test-cupcake#3)

В предыдущей лабораторной работе вы узнали, что для тестов пользовательского интерфейса в Compose требуется правило теста Compose. То же самое относится и к тестированию Jetpack Navigation. Однако для тестирования навигации требуется дополнительная настройка с помощью правила тестирования Compose.

При тестировании Compose Navigation у вас не будет доступа к тому же, NavHostControllerчто и в коде приложения. Однако вы можете использовать TestNavHostControllerи настроить правило тестирования с помощью этого навигационного контроллера. В этом разделе вы узнаете, как настроить и повторно использовать правило тестирования для тестов навигации.

1. В CupcakeScreenNavigationTest.ktсоздайте правило тестирования, используя createAndroidComposeRuleи передав его ComponentActivityв качестве параметра типа.

import androidx.activity.ComponentActivity  
import androidx.compose.ui.test.junit4.createAndroidComposeRule  
import org.junit.Rule  
  
@get:Rule  
val composeTestRule = createAndroidComposeRule<ComponentActivity>()

Чтобы убедиться, что ваше приложение переходит в правильное место, вам необходимо сослаться на TestNavHostControllerэкземпляр, чтобы проверить маршрут навигации хоста навигации, когда приложение выполняет действия для навигации.

1. Создайте TestNavHostControllerэкземпляр как lateinitпеременную. В Kotlin lateinitключевое слово используется для объявления свойства, которое можно инициализировать после объявления объекта.

import androidx.navigation.testing.TestNavHostController  
  
private lateinit var navController: TestNavHostController

Затем укажите составной элемент, который вы хотите использовать для тестов пользовательского интерфейса.

1. Создайте метод с именем setupCupcakeNavHost().
2. В setupCupcakeNavHost()методе вызовите setContent()метод созданного вами правила тестирования Compose.
3. Внутри лямбды, переданной методу setContent(), вызовите CupcakeApp()компонуемый объект.

import com.example.cupcake.CupcakeApp  
  
fun setupCupcakeNavHost() {  
    composeTestRule.setContent {  
        CupcakeApp()  
    }  
}

Теперь вам нужно создать TestNavHostContollerобъект в тестовом классе. Вы используете этот объект позже, чтобы определить состояние навигации, поскольку приложение использует контроллер для навигации по различным экранам приложения Cupcake.

1. Настройте хост навигации, используя созданную ранее лямбду. Инициализируйте navControllerсозданную вами переменную, зарегистрируйте навигатор и передайте его TestNavHostControllerкомпонуемому CupcakeAppобъекту.

import androidx.compose.ui.platform.LocalContext  
  
fun setupCupcakeNavHost() {  
    composeTestRule.setContent {  
        navController = TestNavHostController(LocalContext.current).apply {  
            navigatorProvider.addNavigator(ComposeNavigator())  
        }  
        CupcakeApp(navController = navController)  
    }  
}

Каждый тест в CupcakeScreenNavigationTestклассе включает в себя тестирование какого-либо аспекта навигации. Поэтому каждый тест зависит от TestNavHostControllerсозданного вами объекта. Вместо того, чтобы вручную вызывать setupCupcakeNavHost()функцию для каждого теста для настройки контроллера навигации, вы можете сделать это автоматически, используя @Beforeаннотацию, предоставленную библиотекой **junit** . Когда метод помечен @Before, он запускается перед каждым методом, помеченным @Test.

1. Добавьте @Beforeаннотацию к setupCupcakeNavHost()методу.

import org.junit.Before  
  
@Before  
fun setupCupcakeNavHost() {  
    composeTestRule.setContent {  
        navController = TestNavHostController(LocalContext.current).apply {  
            navigatorProvider.addNavigator(ComposeNavigator())  
        }  
        CupcakeApp(navController = navController)  
    }  
}

## [5. Напишите навигационные тесты](https://developer.android.com/codelabs/basic-android-kotlin-compose-test-cupcake?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-test-cupcake#4)

## Подтвердите начальный пункт назначения

Напомним, что при создании приложения Cupcake вы создали enumкласс с именем CupcakeScreen, который содержал константы, определяющие навигацию по приложению.

**CupcakeScreen.kt**

/\*\*  
\* enum values that represent the screens in the app  
\*/  
enum class CupcakeScreen(@StringRes val title: Int) {  
   Start(title = R.string.app\_name),  
   Flavor(title = R.string.choose\_flavor),  
   Pickup(title = R.string.choose\_pickup\_date),  
   Summary(title = R.string.order\_summary)  
}

Все приложения, имеющие пользовательский интерфейс, имеют тот или иной домашний экран. Для Cupcake таким экраном является **экран начала заказа** . Контроллер навигации в CupcakeAppсоставном элементе использует Startэлемент перечисления CupcakeScreen, чтобы определить, когда переходить к этому экрану. Если при запуске приложения маршрут назначения еще не существует, для маршрута назначения узла навигации установлено значение CupcakeScreen.Start.name.

Сначала вам нужно написать тест, чтобы убедиться, что **экран заказа на запуск** является текущим маршрутом назначения при запуске приложения.

1. Создайте функцию с именем cupcakeNavHost\_verifyStartDestination()и аннотируйте ее @Test.

import org.junit.Test  
  
@Test  
fun cupcakeNavHost\_verifyStartDestination() {  
}

Теперь вы должны подтвердить, что первоначальный маршрут назначения навигационного контроллера — это **экран стартового заказа** .

1. Убедитесь, что ожидаемое имя маршрута (в данном случае CupcakeScreen.Start.name) равно маршруту назначения текущей записи обратного стека навигационного контроллера.

import org.junit.Assert.assertEquals  
...  
  
@Test  
fun cupcakeNavHost\_verifyStartDestination() {  
    assertEquals(CupcakeScreen.Start.name, navController.currentBackStackEntry?.destination?.route)  
}

**Примечание.** Созданный вами файл AndroidComposeTestRuleавтоматически запускает приложение, отображая CupcakeAppкомпонуемый объект перед выполнением любого @Testметода. Таким образом, вам не нужно выполнять какие-либо дополнительные действия в методах тестирования для запуска приложения.

## Создание вспомогательных методов

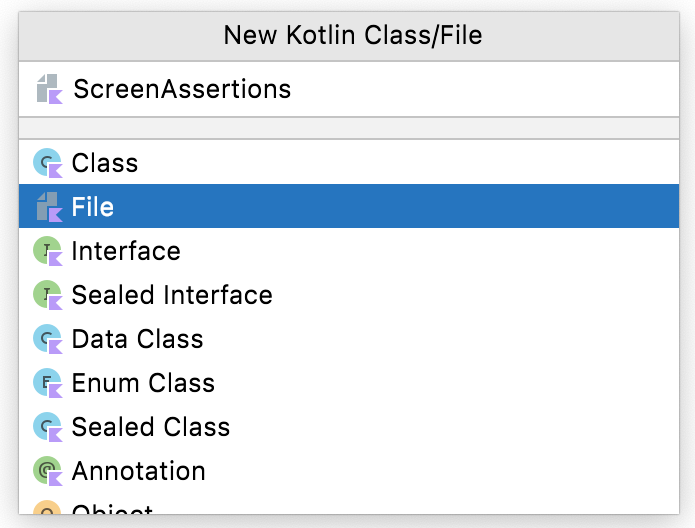
Тесты пользовательского интерфейса часто требуют повторения шагов, чтобы привести пользовательский интерфейс в состояние, в котором можно протестировать конкретную часть пользовательского интерфейса. Пользовательский интерфейс также может требовать сложных утверждений, требующих нескольких строк кода. Утверждение, которое вы написали в предыдущем разделе, требует большого количества кода, и вы используете это утверждение много раз при тестировании навигации в приложении Cupcake. В таких ситуациях написание вспомогательных методов в ваших тестах избавит вас от написания дублированного кода.

Для каждого написанного вами навигационного теста вы используете nameсвойство элементов CupcakeScreenперечисления, чтобы проверить правильность текущего маршрута назначения навигационного контроллера. Вы пишете вспомогательную функцию, которую можете вызывать всякий раз, когда захотите сделать такое утверждение.

Выполните следующие шаги, чтобы создать эту вспомогательную функцию:

1. Создайте пустой файл Kotlin в testкаталоге с именем ScreenAssertions.

**Примечание.** Обязательно создайте **файл** , а не **класс** . Этот файл будет использоваться для создания функции расширения, для которой потребуется пустой файл Kotlin.



1. Добавьте функцию расширения к NavControllerвызываемому классу assertCurrentRouteName()и передайте строку для ожидаемого имени маршрута в сигнатуре метода.

fun NavController.assertCurrentRouteName(expectedRouteName: String) {  
  
}

1. В этой функции утверждается, что expectedRouteNameравен маршруту назначения текущей записи обратного стека навигационного контроллера.

import org.junit.Assert.assertEquals  
...  
  
fun NavController.assertCurrentRouteName(expectedRouteName: String) {  
    assertEquals(expectedRouteName, currentBackStackEntry?.destination?.route)  
}

1. Откройте файл CupcakeScreenNavigationTest и измените cupcakeNavHost\_verifyStartDestination()функцию, чтобы она использовала новую функцию расширения вместо длинного утверждения.

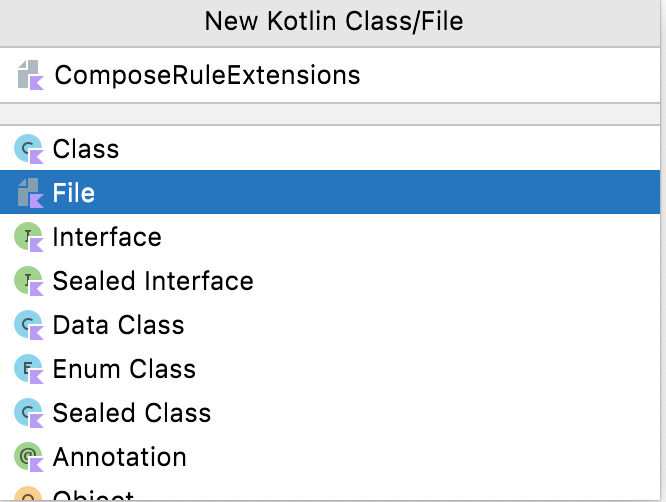
@Test  
fun cupcakeNavHost\_verifyStartDestination() {  
    navController.assertCurrentRouteName(CupcakeScreen.Start.name)  
}

Ряд тестов также требует взаимодействия с компонентами пользовательского интерфейса. В этой кодовой лаборатории эти компоненты часто находятся с помощью строки ресурса. Вы можете получить доступ к составному объекту по его строке ресурса с помощью Context.getString()метода, о котором вы можете прочитать [здесь](https://developer.android.com/reference/android/content/Context#getString(int)) . При написании UI-теста в Compose реализация этого метода выглядит следующим образом:

composeTestRule.onNodeWithText(composeTestRule.activity.getString(R.string.my\_string)

Это подробная инструкция, и ее можно упростить, добавив функцию расширения.

1. Создайте в пакете новый файл com.example.cupcake.testс именем **ComposeRuleExtensions.kt** . Убедитесь, что это обычный файл Kotlin.



1. Добавьте следующий код в этот файл.

import androidx.activity.ComponentActivity  
import androidx.annotation.StringRes  
import androidx.compose.ui.test.SemanticsNodeInteraction  
import androidx.compose.ui.test.junit4.AndroidComposeTestRule  
import androidx.compose.ui.test.onNodeWithText  
import androidx.test.ext.junit.rules.ActivityScenarioRule  
  
fun <A : ComponentActivity> AndroidComposeTestRule<ActivityScenarioRule<A>, A>.onNodeWithStringId(  
    @StringRes id: Int  
): SemanticsNodeInteraction = onNodeWithText(activity.getString(id))

Эта функция расширения позволяет сократить объем кода, который вы пишете при поиске компонента пользовательского интерфейса по его строковому ресурсу. Вместо того, чтобы писать это:

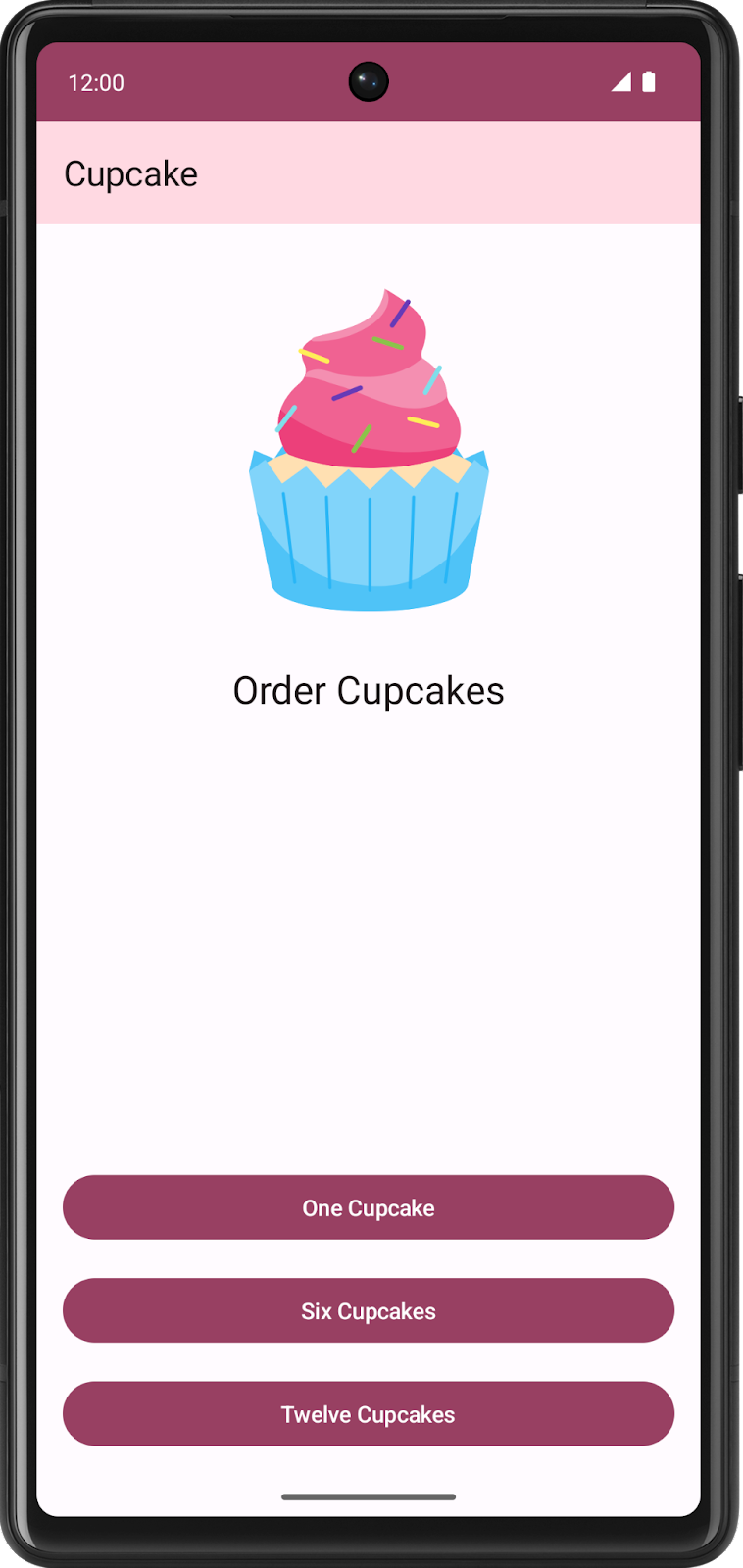
composeTestRule.onNodeWithText(composeTestRule.activity.getString(R.string.my\_string)

Теперь вы можете использовать следующую инструкцию:

composeTestRule.onNodeWithStringId(R.string.my\_string)

## Убедитесь, что на начальном экране нет кнопки «Вверх».

В оригинальном дизайне приложения Cupcake нет кнопки «Вверх» на панели инструментов начального экрана.



На экране «Пуск» отсутствует кнопка, поскольку с этого экрана некуда переходить вверх, поскольку это начальный экран. Выполните следующие действия, чтобы создать функцию, которая подтверждает, что на начальном экране нет кнопки «Вверх»:

1. Создайте метод с именем cupcakeNavHost\_verifyBackNavigationNotShownOnStartOrderScreen()и аннотируйте его @Test.

@Test  
fun cupcakeNavHost\_verifyBackNavigationNotShownOnStartOrderScreen() {  
}

В Cupcake для кнопки «Вверх» в качестве описания содержимого задана строка из R.string.back\_buttonресурса.

1. Создайте переменную в тестовой функции со значением ресурса R.string.back\_button.

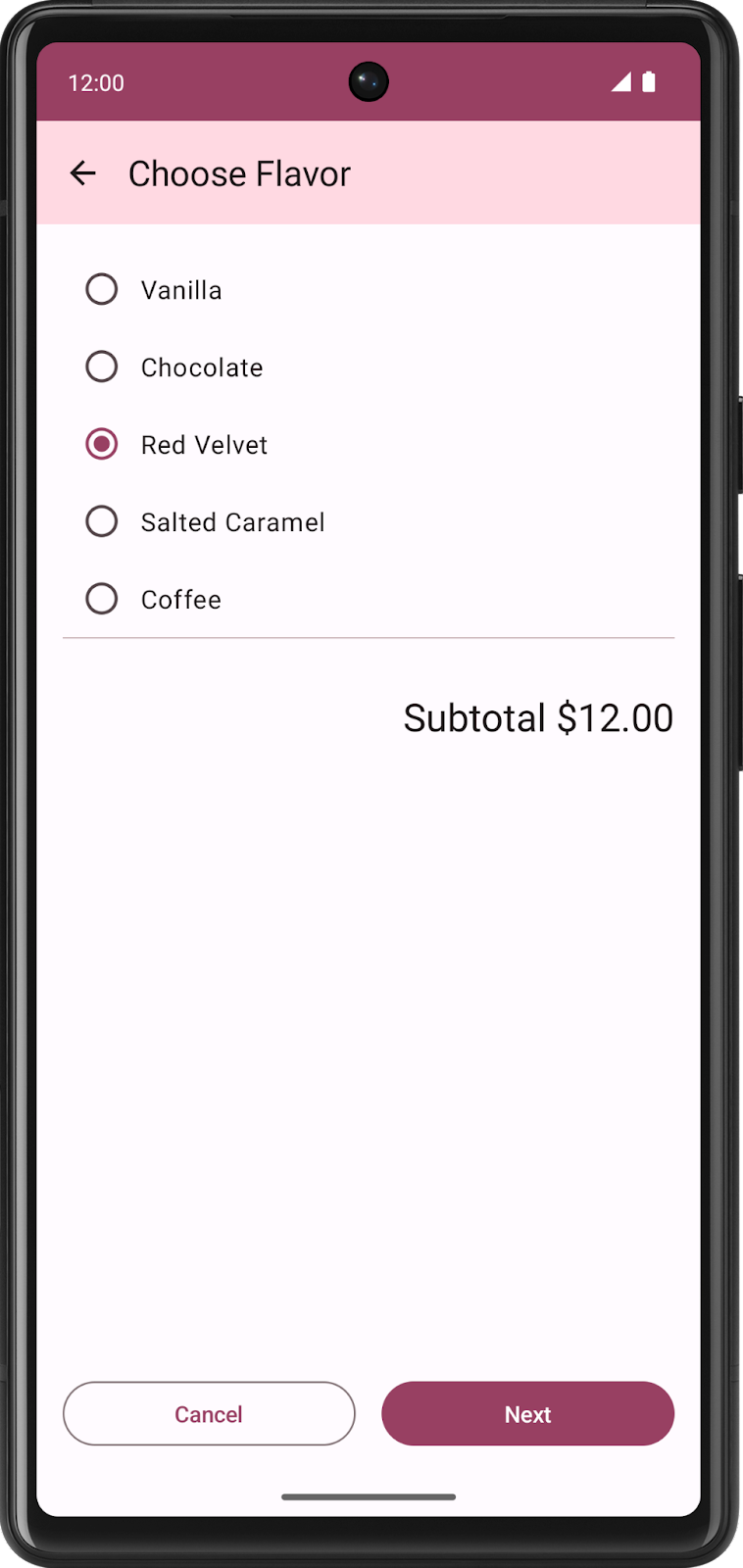
@Test  
fun cupcakeNavHost\_verifyBackNavigationNotShownOnStartOrderScreen() {  
    val backText = composeTestRule.activity.getString(R.string.back\_button)  
}

1. Утвердите, что узел с таким описанием содержимого не существует на экране.

@Test  
fun cupcakeNavHost\_verifyBackNavigationNotShownOnStartOrderScreen() {  
    val backText = composeTestRule.activity.getString(R.string.back\_button)  
    composeTestRule.onNodeWithContentDescription(backText).assertDoesNotExist()  
}

## Проверьте переход на экран Flavor.

Нажатие на одну из кнопок на начальном экране запускает метод, который инструктирует навигационный контроллер перейти на экран Flavor.



В этом тесте вы пишете команду для нажатия кнопки, чтобы активировать эту навигацию, и проверяете, что маршрут назначения — это экран Flavor.

1. Создайте функцию с именем cupcakeNavHost\_clickOneCupcake\_navigatesToSelectFlavorScreen()и аннотируйте ее @Test.

@Test  
fun cupcakeNavHost\_clickOneCupcake\_navigatesToSelectFlavorScreen(){  
}

**Примечание.** Имена методов тестирования отличаются от имен функций, которые вы используете в коде приложения. Хорошее соглашение об именах для методов тестирования следующее: thingUnderTest\_TriggerOfTest\_ResultOfTest. Используя в качестве примера функцию, которую вы только что создали, становится ясно, что вы тестируете хост навигации по кексам, нажимая кнопку **One Cupcake** , и ожидаемым результатом является переход к экрану Flavor.

1. Найдите кнопку **One Cupcake** по ее идентификатору строкового ресурса и выполните по ней действие щелчка.

import com.example.cupcake.R  
...  
  
@Test  
fun cupcakeNavHost\_clickOneCupcake\_navigatesToSelectFlavorScreen() {  
    composeTestRule.onNodeWithStringId(R.string.one\_cupcake)  
        .performClick()  
}

1. Убедитесь, что текущее имя маршрута — это экранное имя Flavor.

@Test  
fun cupcakeNavHost\_clickOneCupcake\_navigatesToSelectFlavorScreen() {  
    composeTestRule.onNodeWithStringId(R.string.one\_cupcake)  
        .performClick()  
    navController.assertCurrentRouteName(CupcakeScreen.Flavor.name)  
}

## Напишите больше вспомогательных методов

Приложение Cupcake имеет в основном линейную навигацию. Если не нажать кнопку **«Отмена»** , вы можете перемещаться по приложению только в одном направлении. Поэтому, тестируя более глубокие экраны приложения, вы можете обнаружить, что повторяете код для перехода к областям, которые хотите протестировать. Эта ситуация заслуживает использования большего количества вспомогательных методов, так что вам придется написать этот код только один раз.

Теперь, когда вы протестировали навигацию по экрану Flavor, создайте метод, который осуществляет переход на экран Flavor, чтобы вам не приходилось повторять этот код для будущих тестов.

**Примечание.** Имейте в виду, что это не методы тестирования. Они ничего не тестируют и должны выполняться только при явном вызове. Поэтому их не следует помечать @Test.

1. Создайте метод с именем navigateToFlavorScreen().

private fun navigateToFlavorScreen() {  
}

1. Напишите команду, чтобы найти кнопку **One Cupcake** и выполнить по ней действие щелчка, как вы это делали в предыдущем разделе.

private fun navigateToFlavorScreen() {  
    composeTestRule.onNodeWithStringId(R.string.one\_cupcake)  
        .performClick()  
}

Помните, что кнопка **«Далее»** на экране «Аромат» не будет доступна для нажатия, пока не будет выбран аромат. Этот метод предназначен только для подготовки пользовательского интерфейса к навигации. После вызова этого метода пользовательский интерфейс должен находиться в состоянии, в котором кнопка **«Далее»** доступна для нажатия.

1. Найдите в пользовательском интерфейсе узел со R.string.chocolateстрокой и щелкните его, чтобы выбрать.

private fun navigateToFlavorScreen() {  
    composeTestRule.onNodeWithStringId(R.string.one\_cupcake)  
        .performClick()  
    composeTestRule.onNodeWithStringId(R.string.chocolate)  
        .performClick()  
}

Посмотрите, сможете ли вы написать вспомогательные методы, которые будут переходить на экран «Выбор» и экран «Сводка». Прежде чем искать решение, попробуйте выполнить это упражнение самостоятельно.

**Примечание.** Для перехода к экрану «Сводка» сначала необходимо выбрать дату на экране «Самовывоз». Вам необходимо создать дату для выбора в пользовательском интерфейсе.

Для этого используйте следующий код:

private fun getFormattedDate(): String {  
    val calendar = Calendar.getInstance()  
    calendar.add(java.util.Calendar.DATE, 1)  
    val formatter = SimpleDateFormat("E MMM d", Locale.getDefault())  
    return formatter.format(calendar.time)  
}

private fun navigateToPickupScreen() {  
    navigateToFlavorScreen()  
    composeTestRule.onNodeWithStringId(R.string.next)  
        .performClick()  
}  
  
private fun navigateToSummaryScreen() {  
    navigateToPickupScreen()  
    composeTestRule.onNodeWithText(getFormattedDate())  
        .performClick()  
    composeTestRule.onNodeWithStringId(R.string.next)  
        .performClick()  
}

Когда вы тестируете экраны за пределами начального экрана, вам необходимо запланировать тестирование функциональности кнопки «Вверх», чтобы убедиться, что она направляет переход к предыдущему экрану. Рассмотрите возможность создания вспомогательной функции для поиска и нажатия кнопки «Вверх».

private fun performNavigateUp() {  
    val backText = composeTestRule.activity.getString(R.string.back\_button)  
    composeTestRule.onNodeWithContentDescription(backText).performClick()  
}

## Максимальное покрытие тестированием

Набор тестов приложения должен тестировать как можно большую часть функциональности приложения. В идеальном мире набор тестов пользовательского интерфейса должен охватывать 100% функциональности пользовательского интерфейса. На практике такого объема тестирования трудно достичь, поскольку существует множество внешних по отношению к вашему приложению факторов, которые могут повлиять на пользовательский интерфейс, например, устройства с уникальными размерами экрана, различные версии операционной системы Android и сторонние приложения, которые могут повлиять на другие приложения на телефоне.

Один из способов максимизировать покрытие тестами — писать тесты вместе с функциями по мере их добавления. При этом вы не зайдете слишком далеко вперед в освоении новых функций и не будете возвращаться назад, чтобы запомнить все возможные сценарии. На данный момент Cupcake — довольно маленькое приложение, и вы уже протестировали значительную часть его навигации! Однако есть и другие состояния навигации, которые нужно протестировать.

Посмотрите, сможете ли вы написать тесты для проверки следующих состояний навигации. Попробуйте реализовать их самостоятельно, прежде чем искать решение.

* Переход к начальному экрану путем нажатия кнопки «Вверх» на экране Flavor.
* Переход к начальному экрану путем нажатия кнопки «Отмена» на экране Flavor.
* Переход к экрану «Самовывоз»
* Переход к экрану «Аромат» путем нажатия кнопки «Вверх» на экране «Самовывоз».
* Переход к начальному экрану путем нажатия кнопки «Отмена» на экране «Самовывоз».
* Переход к экрану сводки
* Переход к начальному экрану путем нажатия кнопки «Отмена» на экране «Сводка».

@Test  
fun cupcakeNavHost\_clickNextOnFlavorScreen\_navigatesToPickupScreen() {  
    navigateToFlavorScreen()  
    composeTestRule.onNodeWithStringId(R.string.next)  
        .performClick()  
    navController.assertCurrentRouteName(CupcakeScreen.Pickup.name)  
}  
  
@Test  
fun cupcakeNavHost\_clickBackOnFlavorScreen\_navigatesToStartOrderScreen() {  
    navigateToFlavorScreen()  
    performNavigateUp()  
    navController.assertCurrentRouteName(CupcakeScreen.Start.name)  
}  
  
@Test  
fun cupcakeNavHost\_clickCancelOnFlavorScreen\_navigatesToStartOrderScreen() {  
    navigateToFlavorScreen()  
    composeTestRule.onNodeWithStringId(R.string.cancel)  
        .performClick()  
    navController.assertCurrentRouteName(CupcakeScreen.Start.name)  
}  
  
@Test  
fun cupcakeNavHost\_clickNextOnPickupScreen\_navigatesToSummaryScreen() {  
    navigateToPickupScreen()  
    composeTestRule.onNodeWithText(getFormattedDate())  
        .performClick()  
    composeTestRule.onNodeWithStringId(R.string.next)  
        .performClick()  
    navController.assertCurrentRouteName(CupcakeScreen.Summary.name)  
}  
  
@Test  
fun cupcakeNavHost\_clickBackOnPickupScreen\_navigatesToFlavorScreen() {  
    navigateToPickupScreen()  
    performNavigateUp()  
    navController.assertCurrentRouteName(CupcakeScreen.Flavor.name)  
}  
  
@Test  
fun cupcakeNavHost\_clickCancelOnPickupScreen\_navigatesToStartOrderScreen() {  
    navigateToPickupScreen()  
    composeTestRule.onNodeWithStringId(R.string.cancel)  
        .performClick()  
    navController.assertCurrentRouteName(CupcakeScreen.Start.name)  
}  
  
@Test  
fun cupcakeNavHost\_clickCancelOnSummaryScreen\_navigatesToStartOrderScreen() {  
    navigateToSummaryScreen()  
    composeTestRule.onNodeWithStringId(R.string.cancel)  
        .performClick()  
    navController.assertCurrentRouteName(CupcakeScreen.Start.name)  
}

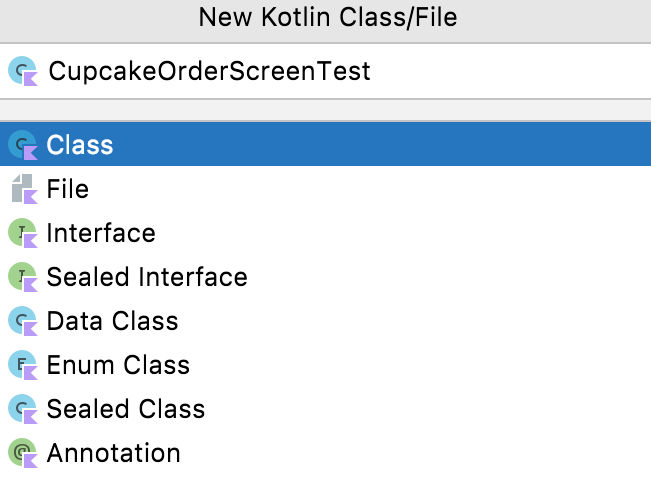
## [6. Напишите тесты для экрана «Заказ»](https://developer.android.com/codelabs/basic-android-kotlin-compose-test-cupcake?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-test-cupcake#5)

Навигация — это лишь один аспект функциональности приложения Cupcake. Пользователь также взаимодействует с каждым из экранов приложения. Вам необходимо проверить, что отображается на этих экранах и что действия, предпринятые на этих экранах, дают правильные результаты. SelectOptionScreen **—** важная часть приложения.

В этом разделе вы напишете тест, проверяющий правильность настройки содержимого на этом экране.

## Проверьте содержимое экрана «Выбор вкуса»

1. Создайте новый класс внутри app/src/androidTestкаталога с именем CupcakeOrderScreenTest, где содержатся другие тестовые файлы.



1. В этом классе создайте файл AndroidComposeTestRule.

@get:Rule  
val composeTestRule = createAndroidComposeRule<ComponentActivity>()

1. Создайте функцию с именем selectOptionScreen\_verifyContent()и аннотируйте ее @Test.

@Test  
fun selectOptionScreen\_verifyContent() {  
  
}

В этой функции вы в конечном итоге устанавливаете для содержимого правила Compose значение SelectOptionScreen. Это гарантирует, что SelectOptionScreenсоставной элемент запускается напрямую, и навигация не требуется. Однако для этого экрана требуется два параметра: список вариантов вкуса и промежуточный итог.

1. Создайте список вариантов вкуса и промежуточный итог для передачи на экран.

@Test  
fun selectOptionScreen\_verifyContent() {  
    // Given list of options  
    val flavors = listOf("Vanilla", "Chocolate", "Hazelnut", "Cookie", "Mango")  
    // And subtotal  
    val subtotal = "$100"  
}

1. Задайте составное содержимое, SelectOptionScreenиспользуя только что созданные значения.

Обратите внимание, что этот подход аналогичен запуску компонуемого объекта из файла MainActivity. Единственная разница в том, что метод MainActivityвызывает CupcakeAppкомпонуемый объект, а здесь вы вызываете SelectOptionScreenкомпонуемый объект. Возможность изменить компонуемый объект, из которого вы запускаете, setContent()позволяет запускать определенные компонуемые объекты вместо того, чтобы тест явно проходил через приложение, чтобы добраться до области, которую вы хотите протестировать. Такой подход помогает предотвратить сбой теста в областях кода, не связанных с текущим тестом.

@Test  
fun selectOptionScreen\_verifyContent() {  
    // Given list of options  
    val flavors = listOf("Vanilla", "Chocolate", "Hazelnut", "Cookie", "Mango")  
    // And subtotal  
    val subtotal = "$100"  
  
    // When SelectOptionScreen is loaded  
    composeTestRule.setContent {  
        SelectOptionScreen(subtotal = subtotal, options = flavors)  
    }  
}

На этом этапе теста приложение запускает SelectOptionScreenсоставной элемент, и вы можете взаимодействовать с ним с помощью тестовых инструкций.

1. Выполните итерацию по flavorsсписку и убедитесь, что каждый строковый элемент списка отображается на экране.
2. Используйте этот onNodeWithText()метод, чтобы найти текст на экране, и используйте этот assertIsDisplayed()метод, чтобы убедиться, что текст отображается в приложении.

@Test  
fun selectOptionScreen\_verifyContent() {  
    // Given list of options  
    val flavors = listOf("Vanilla", "Chocolate", "Hazelnut", "Cookie", "Mango")  
    // And subtotal  
    val subtotal = "$100"  
  
    // When SelectOptionScreen is loaded  
    composeTestRule.setContent {  
        SelectOptionScreen(subtotal = subtotal, options = flavors)  
    }  
  
    // Then all the options are displayed on the screen.  
    flavors.forEach { flavor ->  
        composeTestRule.onNodeWithText(flavor).assertIsDisplayed()  
    }  
}

1. Используя тот же метод, чтобы убедиться, что приложение отображает текст, убедитесь, что приложение отображает на экране правильную строку промежуточного итога. Найдите на экране R.string.subtotal\_priceидентификатор ресурса и правильное промежуточное значение, а затем убедитесь, что приложение отображает это значение.

import com.example.cupcake.R  
...  
  
@Test  
fun selectOptionScreen\_verifyContent() {  
    // Given list of options  
    val flavors = listOf("Vanilla", "Chocolate", "Hazelnut", "Cookie", "Mango")  
    // And subtotal  
    val subtotal = "$100"  
  
    // When SelectOptionScreen is loaded  
    composeTestRule.setContent {  
        SelectOptionScreen(subtotal = subtotal, options = flavors)  
    }  
  
    // Then all the options are displayed on the screen.  
    flavors.forEach { flavor ->  
        composeTestRule.onNodeWithText(flavor).assertIsDisplayed()  
    }  
  
    // And then the subtotal is displayed correctly.  
    composeTestRule.onNodeWithText(  
        composeTestRule.activity.getString(  
            R.string.subtotal\_price,  
            subtotal  
        )  
    ).assertIsDisplayed()  
}

Напомним, что кнопка **«Далее»** неактивна, пока не выбран элемент. Этот тест проверяет только содержимое экрана, поэтому последнее, что нужно проверить, — это то, что кнопка **«Далее»** отключена.

1. Найдите кнопку **«Далее»** , используя тот же подход, чтобы найти узел по идентификатору строкового ресурса. Однако вместо проверки того, что приложение отображает узел, используйте метод assertIsNotEnabled().

@Test  
fun selectOptionScreen\_verifyContent() {  
    // Given list of options  
    val flavors = listOf("Vanilla", "Chocolate", "Hazelnut", "Cookie", "Mango")  
    // And subtotal  
    val subtotal = "$100"  
  
    // When SelectOptionScreen is loaded  
    composeTestRule.setContent {  
        SelectOptionScreen(subtotal = subtotal, options = flavors)  
    }  
  
    // Then all the options are displayed on the screen.  
    flavors.forEach { flavor ->  
        composeTestRule.onNodeWithText(flavor).assertIsDisplayed()  
    }  
  
    // And then the subtotal is displayed correctly.  
    composeTestRule.onNodeWithText(  
        composeTestRule.activity.getString(  
            R.string.subtotal\_price,  
            subtotal  
        )  
    ).assertIsDisplayed()  
  
    // And then the next button is disabled  
    composeTestRule.onNodeWithStringId(R.string.next).assertIsNotEnabled()  
}

## Максимальное покрытие тестированием

Тест содержимого экрана «Выбор вкуса» проверяет только один аспект одного экрана. Существует ряд дополнительных тестов, которые вы можете написать, чтобы увеличить покрытие кода. Попробуйте написать следующие тесты самостоятельно, прежде чем загружать код решения.

* Проверьте содержимое начального экрана.
* Проверьте содержимое экрана сводки.
* Убедитесь, что кнопка **«Далее»** активна, когда опция выбрана на экране «Выбор вкуса».

При написании тестов помните о любых вспомогательных функциях, которые могут сократить объем кода, который вы пишете!

## [8. Резюме](https://developer.android.com/codelabs/basic-android-kotlin-compose-test-cupcake?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-4-pathway-2%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-test-cupcake#7)

Поздравляем! Вы узнали, как протестировать компонент Jetpack Navigation. Вы также освоили некоторые фундаментальные навыки написания тестов пользовательского интерфейса, такие как написание вспомогательных методов многократного использования, способы их использования setContent()для написания кратких тестов, настройку тестов с помощью @Beforeаннотаций и способы обеспечения максимального покрытия тестами. Продолжая создавать приложения для Android, не забывайте писать тесты вместе с кодом вашей функции!