

SAMSUNG



# Kotlin

## Базовый Курс

3.12. Тестирование пользовательского интерфейса

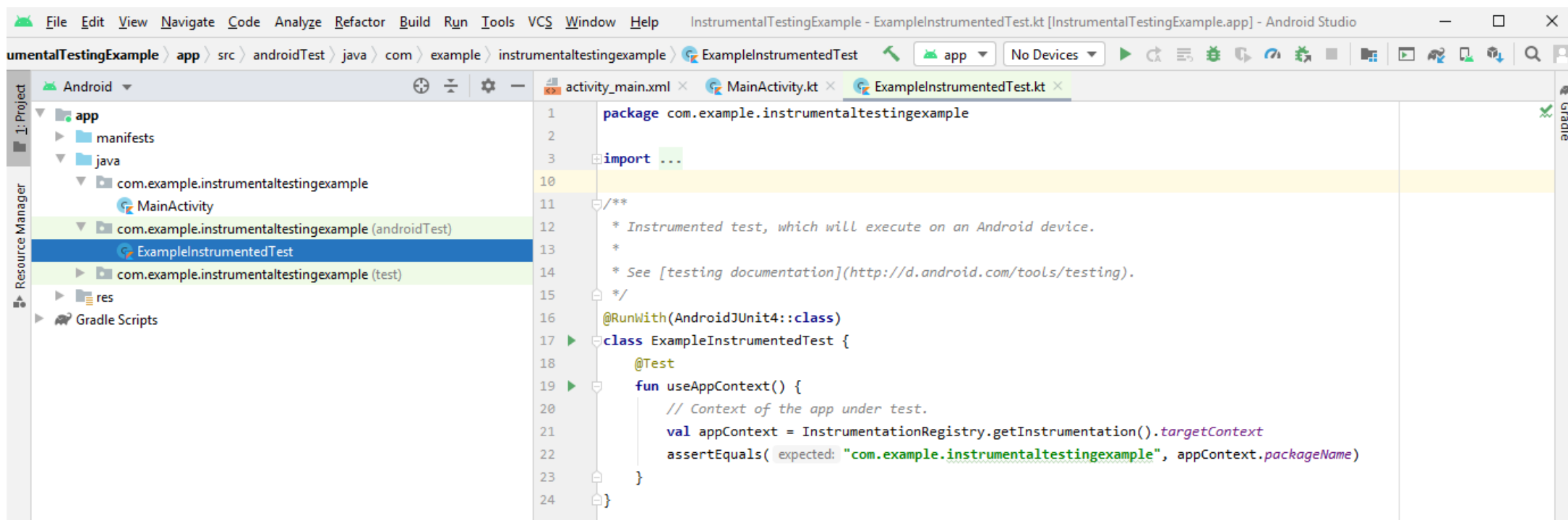


# Инструментальные тесты

SAMSUNG



**Инструментальные тесты** – это тесты, работающие на реальных устройствах или эмуляторах которые благодаря этому могут использовать все возможности системы Android.



```
1 package com.example.instrumentaltestingexample
2
3 import ...
4
5
6
7
8
9
10
11 /**
12  * Instrumented test, which will execute on an Android device.
13  *
14  * See [testing documentation](http://d.android.com/tools/testing).
15  */
16 @RunWith(AndroidJUnit4::class)
17 class ExampleInstrumentedTest {
18     @Test
19     fun useAppContext() {
20         // Context of the app under test.
21         val appContext = InstrumentationRegistry.getInstrumentation().targetContext
22         assertEquals( expected: "com.example.instrumentaltestingexample", appContext.packageName)
23     }
24 }
```



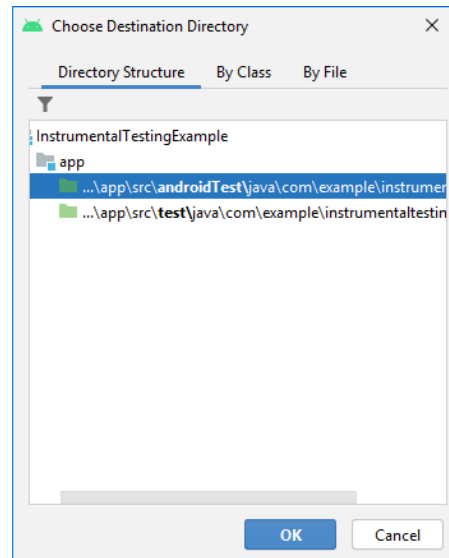
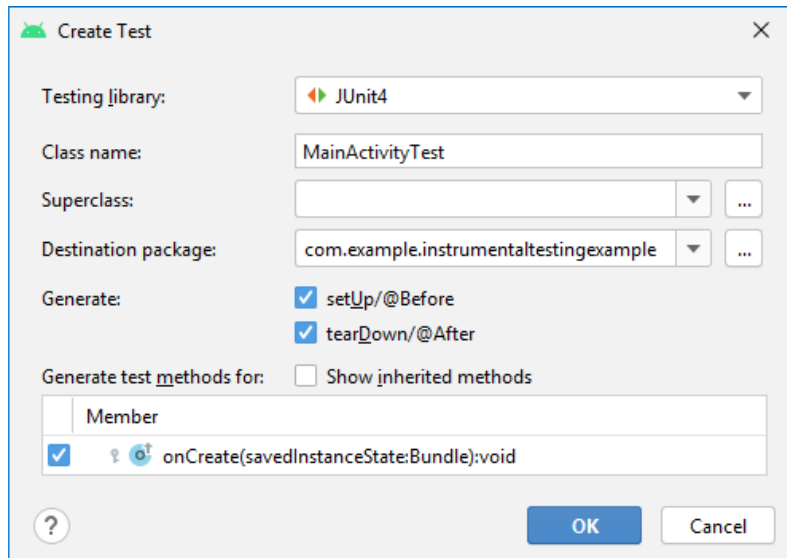


# Инструментальные тесты

SAMSUNG



1. Откройте файл, содержащий код, который вы хотите протестировать;
2. Выберите класс или метод, который вы хотите протестировать, затем нажмите Ctrl+Shift+T;
3. В появившемся меню нажмите кнопку "Create New Test";
4. Появится диалоговое окно, в котором нужно выбрать необходимые параметры;
5. Далее нужно выбрать директорию для создания файла с тестами.



```
class MainActivityTest {  
    @Before fun setUp() {}  
    @After fun tearDown() {}  
    @Test fun onCreate() {}  
}
```





# Espresso



SAMSUNG



```
onView(ViewMatcher)  
    .perform(ViewAction)  
    .check(ViewAssertion);
```

```
onData(ObjectMatcher)  
    .DataOptions  
    .perform(ViewAction)  
    .check(ViewAssertion);
```



## View Matchers

### USER PROPERTIES

```
withId(...)
withText(...)
withTag(...)
withTagValue(...)
hasContentDescription(...)
withContentDescription(...)
withHint(...)
withSpinnerText(...)
hasLinks()
hasEllipsizedText()
hasMultiLineText()
```

### UI PROPERTIES

```
isDisplayed()
isCompletelyDisplayed()
isEnabled()
hasFocus()
isClickable()
isChecked()
isNotChecked()
withEffectiveVisibility(...)
isSelected()
```

### OBJECT MATCHER

```
allOf(Matchers)
anyOf(Matchers)
is(...)
not(...)
endsWith(String)
startsWith(String)
instanceOf(Class)
```

### HIERARCHY

```
withParent(Matcher)
withChild(Matcher)
hasDescendant(Matcher)
isDescendantOfA(Matcher)
hasSibling(Matcher)
isRoot()
```

### INPUT

```
supportsInputMethods(...)
hasIMEAction(...)
```

### CLASS

```
isAssignableFrom(...)
withClassName(...)
```

### ROOT MATCHERS

```
isFocusable()
isTouchable()
isDialog()
withDecorView()
isPlatformPopup()
```

### SEE ALSO

```
Preference matchers
Cursor matchers
Layout matchers
```

## Data Options

```
inAdapterView(Matcher)
atPosition(Integer)
onChildView(Matcher)
```

## View Actions

### CLICK/PRESS

```
click()
doubleClick()
longClick()
pressBack()
pressIMEActionButton()
pressKey(Int/EspressoKey)
pressMenuKey()
closeSoftKeyboard()
openLink()
```

### GESTURES

```
scrollTo()
swipeLeft()
swipeRight()
swipeUp()
swipeDown()
```

### TEXT

```
clearText()
typeText(String)
typeTextIntoFocusedView(String)
replaceText(String)
```

## View Assertions

```
matches(Matcher)
doesNotExist()
selectedDescendantsMatch(...)
```

### LAYOUT ASSERTIONS

```
noEllipsizedText(Matcher)
noMultiLineButtons()
noOverlaps(Matcher)
```

### POSITION ASSERTIONS

```
isLeftOf(Matcher)
isRightOf(Matcher)
isLeftAlignedWith(Matcher)
isRightAlignedWith(Matcher)
isAbove(Matcher)
isBelow(Matcher)
isBottomAlignedWith(Matcher)
isTopAlignedWith(Matcher)
```

```
intended(IntentMatcher);
```

```
intending(IntentMatcher)  
    .respondWith(ActivityResult);
```

## Intent Matchers

### INTENT

```
hasAction(...)
hasCategories(...)
hasData(...)
hasComponent(...)
hasExtras(...)
hasExtras(Matcher)
hasExtraWithKey(...)
hasType(...)
hasPackage()
toPackage(String)
hasFlags(...)
isInternal()
```

### URI

```
hasHost(...)
hasParamWithName(...)
hasPath(...)
hasParamWithValue(...)
hasScheme(...)
hasSchemeSpecificPart(...)
```

### BUNDLE

```
hasEntry(...)
hasKey(...)
hasValue(...)
```

### COMPONENT NAME

```
hasClassName(...)
hasPackageName(...)
hasShortClassName(...)
hasMyPackageName()
```

v2.1.0, 4/21/2015

- Espresso
- UIAutomator
- Robolectric



# Введение в Espresso для Android

SAMSUNG



```
dependencies {  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.0'  
}
```

Таким образом при инструментальном тестировании возникают следующие задачи:

- поиск элементов пользовательского интерфейса в приложении;
- проверка действий над элементами UI;
- проверка результатов действий над элементами пользовательского интерфейса.

Для решения этих задач в Espresso есть три типа методов :

- ViewMatchers — позволяют найти объект в текущей иерархии представлений
- ViewAssertions — позволяют проверить состояние объекта и подтвердить, что состояние соответствует критериям
- ViewActions — эти методы позволяют выполнять различные действия с объектами.





# Основы работы с Espresso в Kotlin

SAMSUNG



```
// withId(R.id.my_view) объект ViewMatcher  
// click() объект ViewAction  
// matches(isDisplayed()) объект ViewAssertion  
onView(withId(R.id.my_view))  
    .perform(click())  
    .check(matches(isDisplayed()))
```

**onView(withId(R.id.my\_view))**

**onView(allOf(withId(R.id.my\_view), withText("Hello!")))**

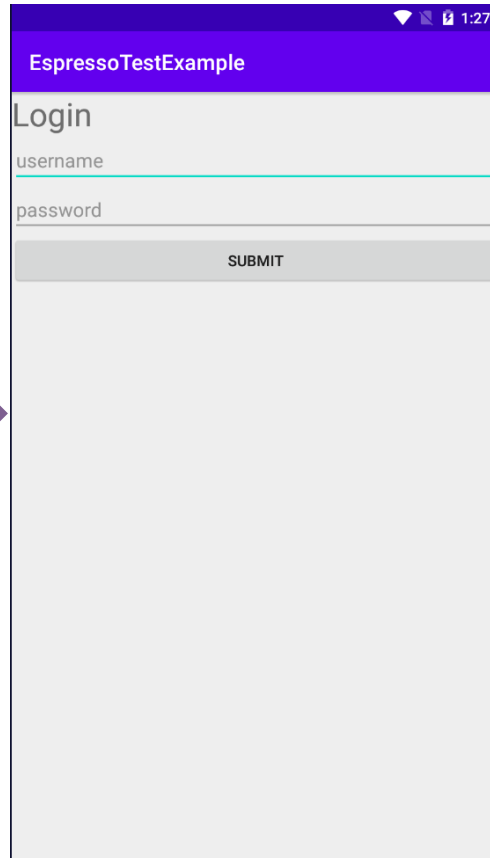
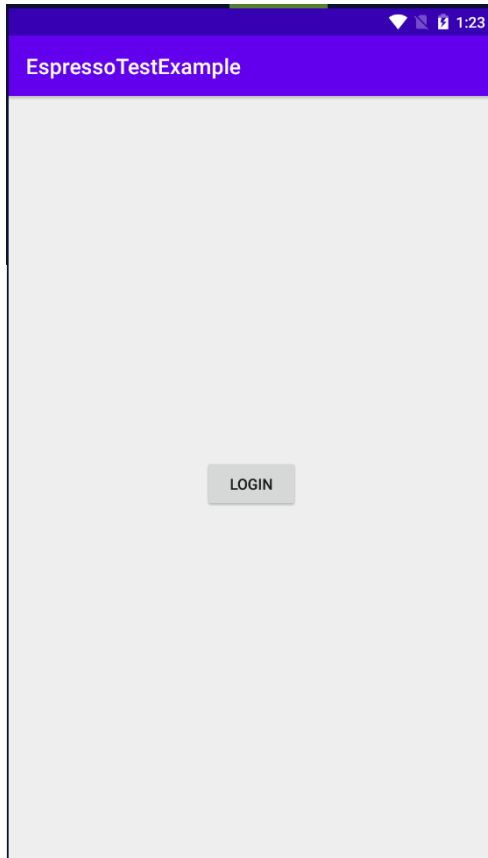
- `ViewActions.click()` - возвращает действие нажатия кнопки;
- `ViewActions.typeText()` - возвращает действие ввода текста;
- `ViewActions.pressKey()` - возвращает действие нажатия на клавишу;
- `ViewActions.clearText()` - возвращает действие, очищающее текст в представлении





# Пример

SAMSUNG



```
@RunWith(AndroidJUnit4::class)
class MainActivityTest{

    @Rule
    @JvmField
    var activityRule = ActivityTestRule<MainActivity>(
        MainActivity::class.java
    )

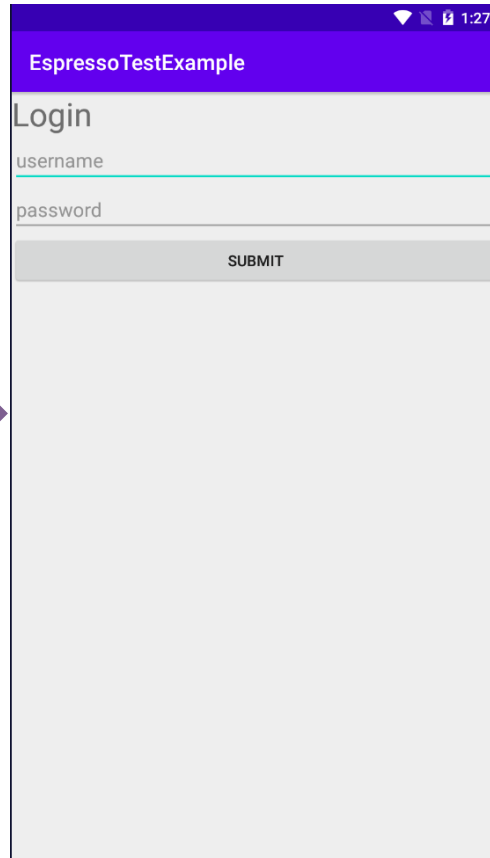
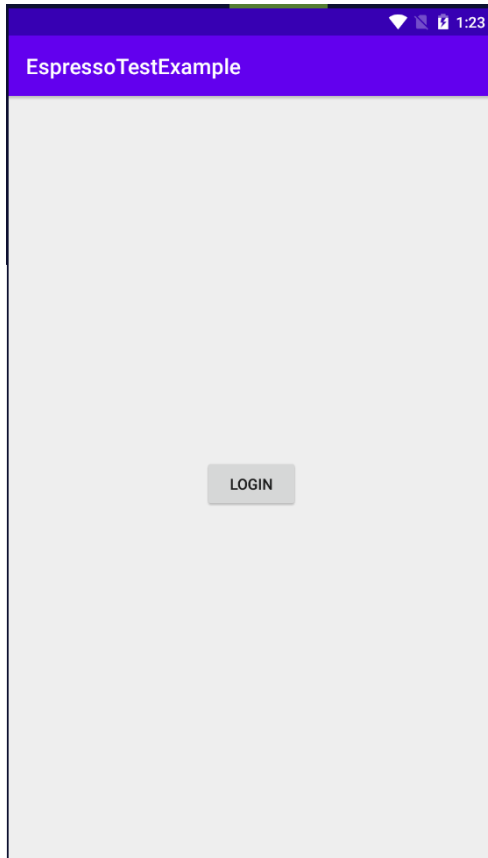
    @Test
    @Throws(Exception::class)
    fun clickLoginButton() {
        onView(withId(R.id.btn_login))
        onView(withId(R.id.btn_login)).check(matches(isDisplayed()))
        onView(withId(R.id.btn_login)).perform(click())
    }
}
```





# Пример

SAMSUNG



```
class LoginActivityTest{  
    @Rule  
    @JvmField  
    var activityRule = ActivityTestRule<LoginActivity>(  
        LoginActivity::class.java  
    )  
  
    private val username = "emityakov"  
    private val password = "password"  
  
    @Test  
    fun clickLoginButton() {  
        onView(withId(R.id.et_username)).perform(ViewActions.typeText(username))  
        onView(withId(R.id.et_password)).perform(ViewActions.typeText(password))  
  
        onView(withId(R.id.btn_submit)).perform(ViewActions.click())  
  
        Espresso.onView(withId(R.id.tv_login))  
            .check(matches(withText("Success")))  
    }  
}
```







**SAMSUNG**



Спасибо

