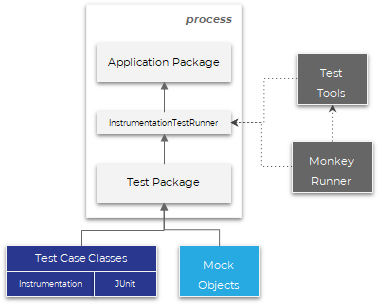
**Lab 30: Junit Testing using Kotlin**

# **Introduction**

The SDK tools include tools for setting up and running test applications, and the Android framework comes with an integrated testing framework that makes it easier to test every component of your application. The SDK tools assist you in setting up and running your tests inside an emulator or the device you are aiming for, whether you are working in Eclipse with ADT or working from the command line.

The build and test tools for Android presum that test projects are set up in accordance with a common hierarchy of tests, test case classes, test packages, and test projects.

* You may test any component of your application using the integrated testing framework provided by the Android framework.
* The SDK tools contain equipment for installing and running test programs.
* The SDK tools assist you in setting up and running your tests inside an emulator or the device you are aiming for, whether you are working in Eclipse with ADT or working from the command line.



**Let’s get Started**

In this experiment we will develop an Android App to demonstrate the perform Junit Testing on Android App. We will take an Android App to perform JUnit Testing.

**Step 1: Create a new Project**

To create a new project in Android Studio. Note that select **Kotlin**as the programming language.

**Step 2: Add dependency to the build.gradle file and click “sync now”**

*testImplementation “com.google.truth:truth:1.0.1”*

*androidTestImplementation “com.google.truth:truth:1.0.1”*

**Step 3: Working with the RegistrationUtil.kt file**

Create a new Kotlin file **RegistrationUtil**and choose its type as an object. Since this is a singleton we do not need to create an object of it while using it in other classes. It has a function called **validRegistrationInput**which requires three arguments **username**, **password**, and **confirm password.**

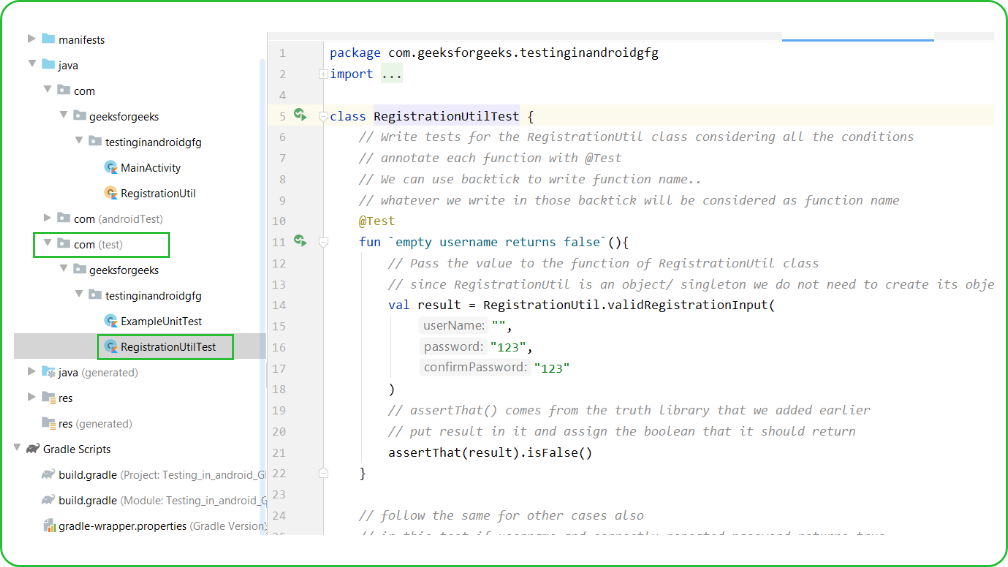
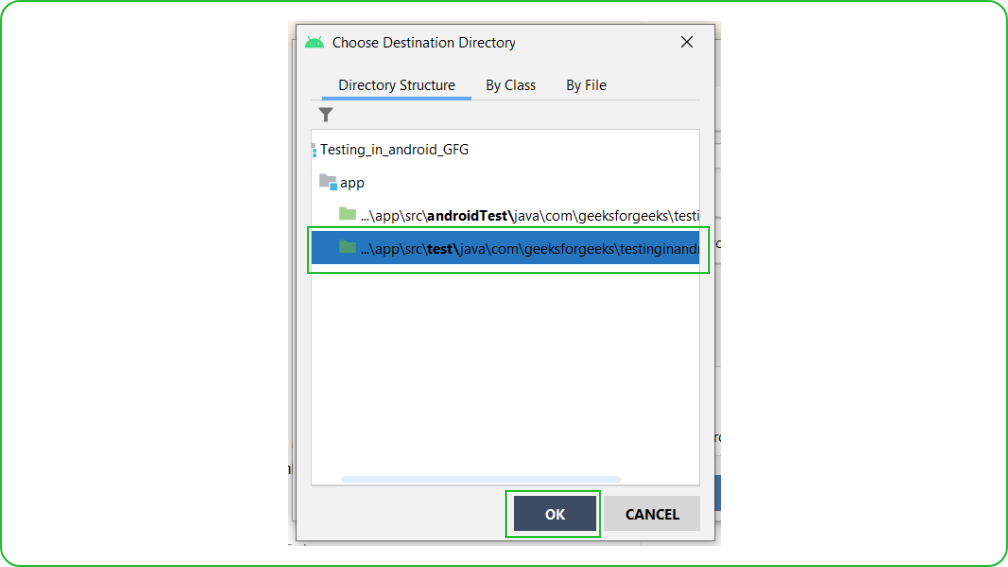
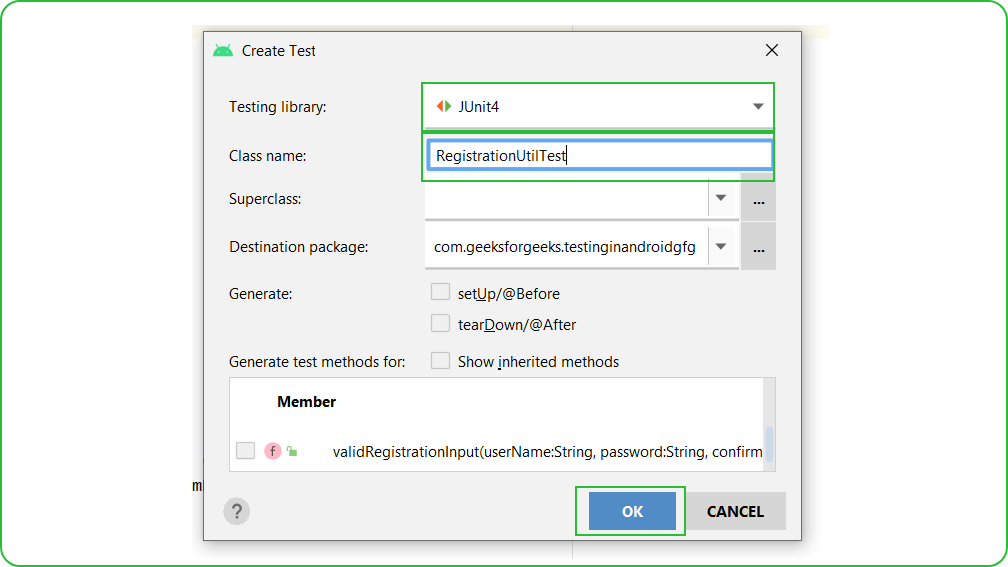
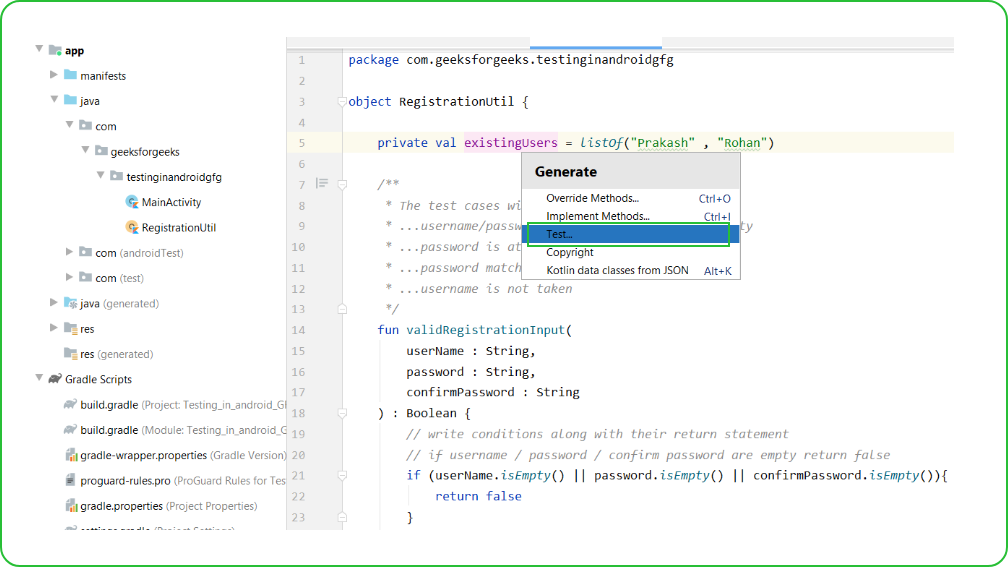
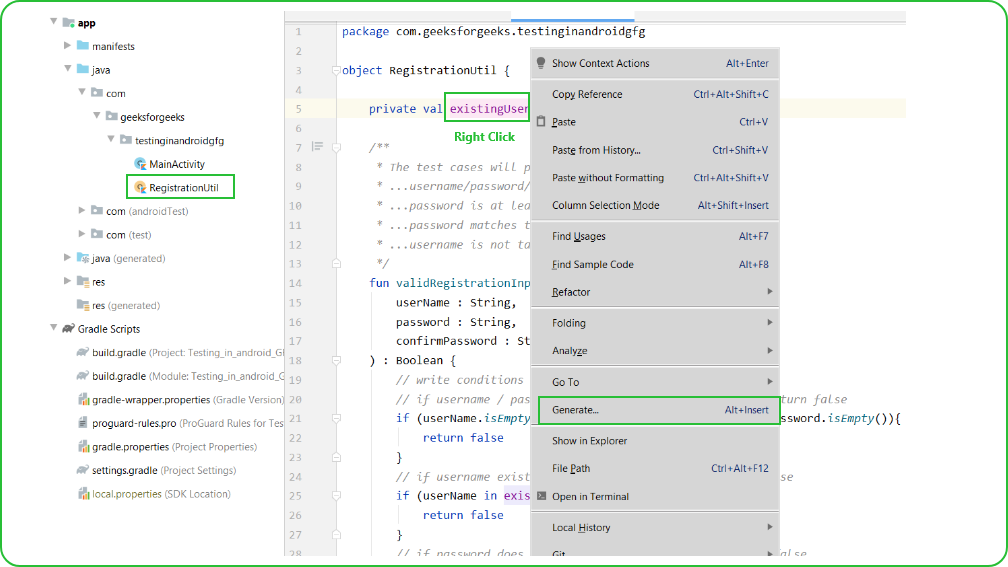
We will be testing this function with different sets of inputs with the following test cases.

* Username, password, confirm password should not be empty.
* Password must contain at least two digits.
* Password matches the confirmed password.
* Username must not be taken.

|  |
| --- |
| object RegistrationUtil {    **private** val existingUsers = listOf("Rahul" , "Rohan")        fun validRegistrationInput(          userName : String,          password : String,          confirmPassword : String      ) : Boolean {          // write conditions along with their return statement          // if username / password / confirm password are empty return false  **if** (userName.isEmpty() || password.isEmpty() || confirmPassword.isEmpty()){  **return** **false**          }          // if username exists in the existingUser list return false  **if** (userName in existingUsers){  **return** **false**          }          // if password does not matches confirm password return false  **if** (password != confirmPassword){  **return** **false**          }          // if digit count of the password is less than 2 return false  **if** (password.count { it.isDigit() } < 2){  **return** **false**          }  **return** **true**      }  } |

**Step 4: Create a test class**

In order to create a test class of **RegistrationUtil**right-click**on**RegistrationUtil then click generate and then select the test. A dialog will open, from the dialog choose Testing library as JUnit4 and keep the class name as default that is **RegistrationUtilTest**, and click ok. After that, another dialog will open to choose the destination directory, choose the one which has **..app\src\test\.**because our test class does notrequireany context from the application. Below is the screenshot to guide you create the test class.



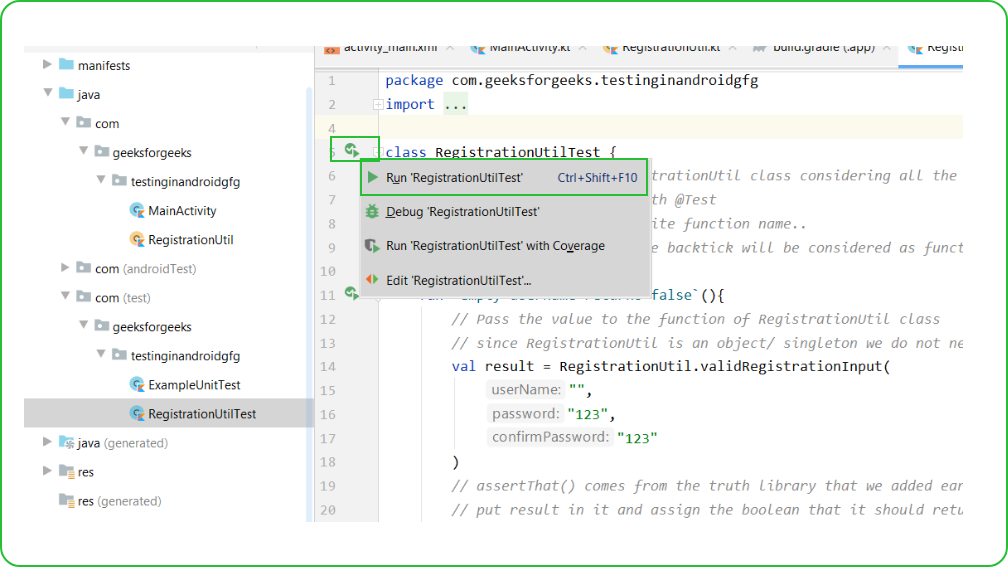
**Step 5: Working with RegistrationUtilTest.kt file**

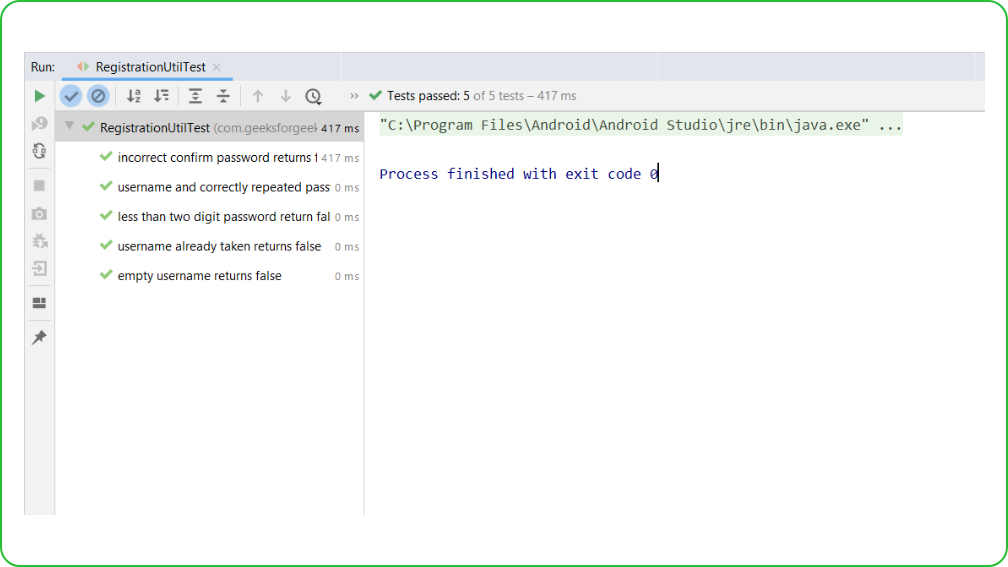
Go to**RegistrationUtilTest.kt** file and write the following code.  Comments are added inside the code to understand the code in more detail.

|  |
| --- |
| **import** com.google.common.truth.Truth.assertThat  **import** org.junit.Test    **class** RegistrationUtilTest {      fun `empty username returns **false**`(){          // Pass the value to the function of RegistrationUtil class          // since RegistrationUtil is an object/ singleton we do not need to create its object          val result = RegistrationUtil.validRegistrationInput(              "",              "123",              "123"          )          // assertThat() comes from the truth library that we added earlier          // put result in it and assign the boolean that it should return          assertThat(result).isFalse()      }        // follow the same for other cases also      // in this test if username and correctly repeated password returns true      @Test      fun `username and correctly repeated password returns **true**`() {          val result = RegistrationUtil.validRegistrationInput(              "Rahul",              "123",              "123"          )          assertThat(result).isTrue()      }        // in this test userName already taken returns false      @Test      fun `username already taken returns **false**`() {          val result = RegistrationUtil.validRegistrationInput(              "Rohan",              "123",              "123"          )          assertThat(result).isFalse()      }       // if confirm password does nt matches the password return false      @Test      fun `incorrect confirm password returns **false**`() {          val result = RegistrationUtil.validRegistrationInput(              "Rahul",              "123",              "1234"          )          assertThat(result).isFalse()      }       // in this test if password has less than two digits than return false      @Test      fun `less than two digit password **return** **false**`() {          val result = RegistrationUtil.validRegistrationInput(              "Rahul",              "abcd1",              "abcd1"          )          assertThat(result).isFalse()      }  } |

**Step 6: Run the Test cases**

To run the test case click on the little run icon neartheclass name and then select Run **RegistrationUtilTest**. If all the test cases pass you will get a green tick in the Run console. In our case, all tests have passed.





**Voila!!** We have successfully completed this lab.