Junit 5

1) Testing **multiple assertions** (grouped assertions) with **assertAll()**

2) **Defining timeouts in your tests.** If you want to ensure that a test fails, if it isn't done in a certain amount of time you can use the **assertTimeout()** method. This assert fails the method if the timeout is exceeded.

3) How to **disable tests.** The **@Disabled** or **@Disabled("Why disabled")** annotation marks a test to be disabled. This is useful when the underlying code has been changed and the test case has not yet been adapted of if the test demonstrates an incorrect behavior in the code which has not yet been fixed. It is best practice to provide the optional description, why the test is disabled.

4) Функциональность динамических тестов JUnit 5 может быть достигнута с помощью параметризованных тестов . Кроме того, параметризованные тесты следуют стандартному жизненному циклу тестов JUnit, и для них выполняются методы @beforeach и @afterEach. В то время как жизненный цикл динамических тестов совершенно другой, и у них нет доступа к методам @beforeach и @afterEach.

5) **Parameterized Tests**

```
public static int[][] data() {
    return new int[][] { { 1 , 2, 2 }, { 5, 3, 15 }, { 121, 4, 484 } };
}

@ParameterizedTest
@MethodSource(value =  "data")
void testWithStringParameter(int[] data) {
    MyClass tester = new MyClass();
    int m1 = data[0];
    int m2 = data[1];
    int expected = data[2];
    assertEquals(expected, tester.multiply(m1, m2));
}
```

6) Data sources

Annotation	Description
`@ValueSource(ints = { 1, 2, 3 })`	Lets you define an array of test values. Permissible types are `String`, `int`, `long`, or `double`.
`@EnumSource(value = Months.class, names = {"JANUARY", "FEBRUARY"})`	Lets you pass Enum constants as test class. With the optional attribute `names` you can choose which constants should be used. Otherwise all attributes are used.
`@MethodSource(names = "genTestData")` COPY JAVA	The result of the named method is passed as argument to the test.
`@CsvSource({ "foo, 1", "'baz, qux', 3" })` `void testMethod(String first, int second) {`	Expects strings to be parsed as Csv. The delimiter is `','`.
`@ArgumentsSource(MyArgumentsProvider.class)`	Specifies a class that provides the test data. The referenced class has to implement the `ArgumentsProvider` interface.

7) Creating mock objects with the Mockito API

- Using the `@ExtendWith(MockitoExtension.class)` extension for JUnit 5 in combination with the **`@Mock`** annotation on fields
- Using the static **`mock()`** method.
- Using the **`@Mock`** annotation.

To mock classes we just use such statesment : **`when(Something).thenReturn(true);`**

You also can use methods like **`anyString`** or **`anyInt`** to define that dependent on the input type a certain value should be returned.

Тогда это выглядит так : **`when(Something(anyString)).thenReturn(true);`**

8) Wrapping Java objects with Spy

@Spy or the spy() method can be used to wrap a real object. Every call, unless specified otherwise, is delegated to the object.

Mock () - подменяет только тот метод который мы определили а остальные будут возвращать void

Spy() - подменяет только тот метод который мы определили но остальные методы будут возвращать то что они возвращают при обычном поведении екземпляра класса

9) Verify the calls on the mock objects

Mockito keeps track of all the method calls and their parameters to the mock object. You can use the verify() method on the mock object to verify that the specified conditions are met.

```
// call method testing on the mock with parameter 12
database.setUniqueId(12);
database.getUniqueId();
database.getUniqueId();


// now check if method testing was called with the parameter 12
verify(database).setUniqueId(ArgumentMatchers.eq(12));

// was the method called twice?
verify(database, times(2)).getUniqueId();

// other alternatives for verifiying the number of method calls for a method
verify(database, never()).isAvailable();
verify(database, never()).setUniqueId(13);
verify(database, atLeastOnce()).setUniqueId(12);
verify(database, atLeast(2)).getUniqueId();

// more options are
// times(numberOfTimes)
// atMost(numberOfTimes)
// This let's you check that no other methods where called on this object.
// You call it after you have verified the expected method calls.
verifyNoMoreInteractions(database);
```

In case you do not care about the value, use the anyX, e.g., **anyInt**, **anyString()**, or **any(YourClass.class)** methods.

10) ArgumentCaptor

ArgumentCaptor<CLASS> - позволяет нам проверить что какой либо метод был вызван с определённым аргументом когда либо ранее пример :

```
@Captor
private ArgumentCaptor<List<String>> captor;

@Test
public final void shouldContainCertainListItem(@Mock List<String> mockedList) {
    var asList = Arrays.asList("someElement_test", "someElement");
    mockedList.addAll(asList);

    verify(mockedList).addAll(captor.capture());
    List<String> capturedArgument = captor.getValue();
    assertThat(capturedArgument, hasItem("someElement"));
}
```

11) doAnswer - позволяет добовлять дополнительную логику при вызове определённого метода к примету провести какие-то вычесления либо добавить в лист значение как в примере :

```
// with doAnswer():
doAnswer(returnsFirstArg()).when(list).add(anyString());
// with thenAnswer():
when(list.add(anyString())).thenAnswer(returnsFirstArg());
// with then() alias:
when(list.add(anyString())).then(returnsFirstArg());
```

Instumentation Tests

1) Activity testing

Необходимо создавать ActivityTextRule из которого с помощью функции getActivity() можно будет получить активность которая открылась и получить из неё вьювы и тестировать их

```
@Rule
public ActivityTestRule<MainActivity> rule  = new  ActivityTestRule<>(MainActivity.class);

 @Test
 public void ensureListViewIsPresent() throws Exception {
     MainActivity activity = rule.getActivity();
     View viewById = activity.findViewById(R.id.listview);
     assertThat(viewById,notNullValue());
     assertThat(viewById, instanceOf(ListView.class));
     ListView listView = (ListView) viewById;
     ListAdapter adapter = listView.getAdapter();
     assertThat(adapter, instanceOf(ArrayAdapter.class));
     assertThat(adapter.getCount(), greaterThan(5));

 }
```

Что бы при открытии активности добаить какието значения в бандл интента или особым образом пределить интентр который будет её запускать при обьявлении Rule необходимо будет за оверрайдить интент :

```
   @Rule
  public ActivityTestRule<SecondActivity> rule  = new  ActivityTestRule<SecondActivity>
(SecondActivity.class)
   {
       @Override
       protected Intent getActivityIntent() {
           InstrumentationRegistry.getTargetContext();
           Intent intent = new Intent(Intent.ACTION_MAIN);
           intent.putExtra("MYKEY", "Hello");
           return intent;
       }
   };
```

2) Service testing

Для такого тестирования необходимо использывать ServiceTestRule

```
@Rule
public final ServiceTestRule mServiceRule = new ServiceTestRule();

// test for a service which is started with startService
@Test
public void testWithStartedService() {
    mServiceRule.
    startService(new Intent(InstrumentationRegistry.getTargetContext(),
            MyService.class));
    // test code
}

@Test
// test for a service which is started with bindService
public void testWithBoundService() {
    IBinder binder = mServiceRule.
            bindService(new Intent(InstrumentationRegistry.getTargetContext(),
                    MyService.class));
    MyService service = ((MyService.LocalBinder) binder).getService();
    assertTrue("True wasn't returned", service.doSomethingToReturnTrue());
}
```

Preformancies

Map Operation Performance Comparison

Map	Size	Average (operations/10 seconds)	%STD	%DIF (with Hashtable)
Hashtable	1000	34034	6.2%	0%
HashMap	1000	27818	0.06%	-22%
LinkedHashMap	1000	25514	0.03%	-33%
IdentityHashMap	1000	11650	0.04%	-192%
ConcurrentHashMap	1000	12420	2.9%	-174%
HashSet	1000	26888	0.04%	-26%

App Startup библеотека

Эта библиотека позволяеет нам более еффективно скомпелировать какие-то компоненты приложения (сделав это заранее при скачивании приложенияе а не делая это при каждом запуске) к примеру так можно создать WorckManager либо Logger

```
class WorkManagerInitializer : Initializer<WorkManager> {
    override fun create(context: Context): WorkManager {
        val configuration = Configuration.Builder().build()
        WorkManager.initialize(context, configuration)
        return WorkManager.getInstance(context)
    }
    override fun dependencies(): List<Class<out Initializer<*>>> {
        // No dependencies on other libraries.
        return emptyList()
    }
}
```

Спящий режим приложения

Если ваше приложение предназначено для Android 11 (уровень API 30) или выше и пользователь не взаимодействует с вашим приложением в течение нескольких месяцев, система переводит ваше приложение в состояние гибернации. Система оптимизирует пространство для хранения, а не производительность, и система защищает пользовательские данные. Это поведение системы похоже на то, что происходит, когда пользователь вручную принудительно останавливает ваше приложение из системных настроек.