## Testcases for Functional Requirements

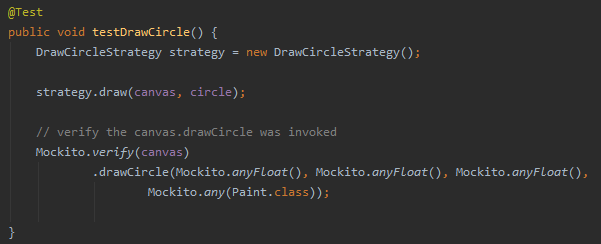
We established a testing strategy of multiple testing techniques such as peer code reviews, End-to-End-Testing, UI tests such as Espresso tests and Unit tests. The latter were constructed using the JUnit and Mockito frameworks and established for all the main classes in the model (DrawStrategies, export functionality, Graphical Elements), the ViewModel (CanvasViewModel) and the View (CanvasView and MainActivity). For the concrete test cases, see the section below.

### Tests for Classes in Model

For the classes in the model, we first created test classes for the DrawStrategies. In the respective DrawStrategy test classes, mocks are first created with the @Mock annotation, i.e. dummy instances. With these mocks, the functionality of the method can now be tested. Mockito.verify checks whether the respective draw method is called. With the Mockito any methods, random values are assigned to the parameters of the draw method.

*Code snippet:*

*(implementation\Sketch\_App\app\src\test\java\at\ac\univie\se2ws2020team0310\sketch\_app\model\draw\DrawCircleStrategyTest.java):*

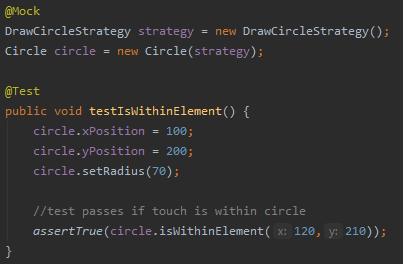


The tests for the export classes throw an IOException if an unsupported FileFormat is used, e.g.: pdf.

For the graphical element classes, we tested the functionality of the isWithinElement method, i.e. the method that checks whether a selected point on the screen is located within a graphical element. To do this, we first created dummy instances using @Mock annotation. The mocks get coordination and all other parameters that are needed for the calculation in the isWithinElement method. Then the test method uses assertTrue() to check whether the isWithinElement method is true for a specific point(x,y).

*Code snippet:*

*(implementation\Sketch\_App\app\src\test\java\at\ac\univie\se2ws2020team0310\sketch\_app\model\graphicalElements\CircleTest.java):*



In the test class GraphicalElementFactoryTest, the create methods of the respective element types are checked. For the example Triangle. First attributes are defined, such as colour, size, strokeWidth, which must be passed to create a GraphicalElement. In the test method for Triangle, a Triangle is first created with the attributes defined above. Then assertTrue is used to check whether the created element has the correct type, i.e. Triangle, and whether the drawStrategy used is the DrawTriangleStrategy. The assertEquals method is used to check whether the attributes defined above match the attributes that can be retrieved using getter methods.

In the ElementCollectionTest with the iterators and collections, various mock objects are first created, which can be added to the array list or otherwise retrieved for the respective methods. The testGet() method uses Mockito verify() to check whether the get method is called. In testAdd, the same method is used to test whether the add method is called when an element is added. In testIndexOf, assertEquals is used to check whether the expected index and the actual index output by the indexOf method match. In testContains it is checked whether contains() returns false for an element that is not in the array list.

### Tests for Classes in ViewModel

### Tests for Classes in View