Junit Guide

* **Setting field variable that doesn’t have setter method**
  + org.springframework.test.util.ReflectionTestUtils
  + ReflectionTestUtils.setField(classObject, “fieldName”, objectToSet);
* **Mocking abstract class**
  + AbstractClass object = Mockito.mock(AbstractClass.class, Mockito.CALLS\_REAL\_METHODS);
* **Mocking testing class’s method(Not private methods)**
  + PowerMockito.method(ClassName.class, "methodName", Argument1Type.class, Argument2Type.class);
* **Spying**
  + Mockito.spy(object);
  + Creates a copy of original object.
* **Mocking new Instance**
  + Use PowerMockito.whenNew(Class or Constructor)
  + Put @PrepareForTest(class of new Instance.class)
* **Cannot be mocked**
  + **private method** of the testing class
  + **new** **instance** of objects in the testing class(Testing class can be put in PrepareForTest, but then the code coverage becomes zero as the classes put in PrepareForTest are ignored by the code coverage software)
* **Create temporary file, for testing with File.class**
  + org.junit.rules.TemporaryFolder folder = new TemporaryFolder()
  + folder.newFolder()
  + File tempFile = folder.newFile(“anyName.txt”)
  + File created, file path = tempFile.getAbsolutePath()
* **Mocking Anonymous Inner Class and Covering its methods**
  + Instead of using *.thenReturn()*, use *.thenAnswer(arg)*
  + In arg provide anonymous inner class of *Answer()* interface.
  + The Overridden method *answer(InvocationOnMock invocation)* will be called when the *when()* condition comes true.
  + Inside *answer(InvocationOnMock invocation)* method, call *invocation.getArguments()* to get arguments of the method called the mock.
  + Get the desidered argument and type cast it to the anonymous inner class.
  + Now call the anonymous inner class method on the instance got from the argument.
  + Mock the resultSet and can also define *when()* conditions on it.
  + The original method to be mocked should always be called on a SPY object, as it calls the real method.
  + Calling the method on the mocked object would always give null value as the real method would never be called hence the anonymous inner class will also not be called.
* **Mock private static final field of a class**
  + Ref - *https://stackoverflow.com/questions/23162520/powermock-mock-out-private-static-final-variable-a-concrete-example*

static void setFinalStatic(Field field, Object newValue) throws Exception {

field.setAccessible(true);

// remove final modifier from field

Field modifiersField = Field.class.getDeclaredField("modifiers");

modifiersField.setAccessible(true);

modifiersField.setInt(field, field.getModifiers() & ~Modifier.FINAL);

field.set(null, newValue);

}

* **Mock static Private methods**

public class Util {

public static String method(){

return anotherMethod();

}

private static String anotherMethod() {

throw new RuntimeException(); // logic was replaced with exception.

}

}

@PrepareForTest(Util.class)

@RunWith(PowerMockRunner.class)

public class UtilTest {

@Test

public void testMethod() throws Exception {

PowerMockito.spy(Util.class);

PowerMockito.doReturn("abc").when(Util.class, "anotherMethod");

String retrieved = Util.method();

Assert.assertNotNull(retrieved);

Assert.assertEquals(retrieved, "abc");

}

}

**Actual method**

public final class GenerateResponse{

private static Map<String, String> getErrorDetails(JSONObject jsonObject) {

return null;

}

public static String method1() {

Map<String, String> map = getErrorDetails(new JSONObject());

return map.get("abc");

}

}

**Test method**

@RunWith(PowerMockRunner.class)

@PrepareForTest(GenerateResponse.class)

public class GenerateResponseTest {

@Test

public void testFrameQtcErrorResponse() throws Exception {

Map<String, String> errorDtls = new HashMap<String, String>();

errorDtls.put("abc", "alphabets");

PowerMockito.mockStatic(GenerateResponse.class, Mockito.CALLS\_REAL\_METHODS);

PowerMockito.doReturn(errorDtls).when(GenerateResponse.class,

"getErrorDetails", Matchers.any(JSONObject.class));

String response = GenerateResponse.method1();

System.out.println("response =" + response);

}

}