Working with Mockito Framework

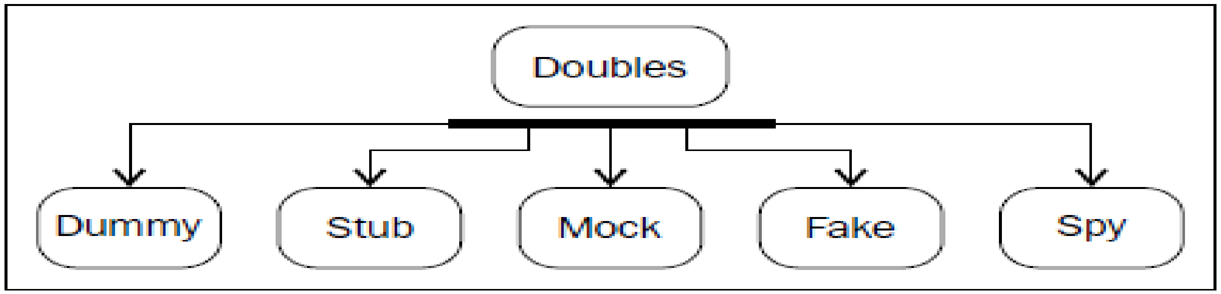
April 23, 2019

Objective

- ❑ Introduction to test doubles
- ❑ Ability to understand Mock and Working with Mockito Framework
- ❑ Ability to Understand Qualities of unit testing
- ❑ Ability to Mock objects
- ❑ Ability to Stub methods with Mockito Framework
- ❑ Verifying the method invocation with Mockito Framework
- ❑ Ability to use argument matcher and working with the ArgumentMatcher Class
- ❑ Ability to handle and throwing exceptions
- ❑ Ability to Stub consecutive calls and Stubbing with Answer
- ❑ Ability to understand and Implement Spying objects with Mockito
- ❑ Ability to capturing arguments with ArgumentCaptor
- ❑ Ability to verify the invocation order of methods and Resetting mock objects
- ❑ Ability to work with Mockito annotations
- ❑ Ability to implement Behavior-driven development with Mockito

Test Doubles

- Test doubles act as stunt doubles
- Test doubles are categorized into five types. The following diagram shows these types:



Test Doubles cont..

Dummy

- ❑ An empty object passed in an invocation (usually only to satisfy a compiler when a method ar- gument is required)

Fake

- ❑ An object having a functional implementation, but usually in a simplified form, just to satisfy the test (e.g., an in-memory database)

Stub

- ❑ An object with hardcoded behavior suitable for a given test (or a group of tests)

Mock

- ❑ An object with the ability to a) have a programmed expected behavior, and b) verify the interactions occurring in its lifetime (this object is usually created with the help of mocking framework)

Spy

- ❑ A mock created as a proxy to an existing real object; some methods can be stubbed, while the un- stubbed ones are forwarded to the covered object

Mockito ?

A test may take time to execute due to the following reasons:

- ❑ Sometimes a test acquires a connection from the database that fetches/updates data
- ❑ It connects to the Internet and downloads files
- ❑ It interacts with an SMTP server to send e-mails
- ❑ It performs I/O operations

Mockito is a mocking framework helpful in creating mocks and spies in a simple and intuitive way, while at the same time providing great control of the whole process.

Understand Mockito

- ❑ Mockito is an open source mock unit testing framework for Java.
- ❑ Mockito allows mock object creation, verification, and stubbing.

http://code.google.com/p/mockito/

Why Mockito?

- ❑ If a test suite runs for an hour, the purpose of quick feedback is compromised.
- ❑ Unit tests should act as a safety net and provide quick feedback; this is the main principle of TDD.

Understand Mockito cont..

A test may take time to execute due to the following reasons:

- ❑ Sometimes a test acquires a connection from the database that fetches/updates data
- ❑ It connects to the Internet and downloads files
- ❑ It interacts with an SMTP server to send e-mails
- ❑ It performs I/O operations

Now the question is do we really need to acquire a database connection or download files to unit test code?

- ❑ To unit test these parts, the external dependencies need to be mocked out.
- ❑ Mockito plays a key role in mocking out external dependencies

Benefits of Mocking

Unit test reliability:

- ❑ You mock test unfriendly objects so that your test becomes reliable.
- ❑ hey don't fail for any unavailable external object as you mock the external object.

Unit tests can be automated:

- ❑ Mockito makes unit test configuration simple as the tests can mock external dependencies, such as a web service call or database access.

Extremely fast test execution:

- ❑ Unit tests access mock objects, so delay in external service call or slow I/O operations can be isolated.

Qualities of unit testing

Unit tests should adhere to the following principles:

- **Order independent and isolated :** The ATest.java test class should not be dependent on the output of the BTest.java test class
- **Trouble-free setup and run:** Unit tests should not require a DB connection or an Internet connection or a clean-up temp directory.
- **Effortless execution:** Unit tests should run fine on all computers, not just on a specific computer.
- **Formula 1 execution:** A test should not take more than a second to finish the execution.

Mockito provides APIs to mock out the external dependencies and achieve the qualities mentioned here.

Working with Mockito

Implementation Mocking : Case Study Detail

“We will implement the mock objects with a stock quote example. In the real world, people invest money on the stock market—they buy and sell stocks. A stock symbol is an abbreviation used to uniquely identify shares of a particular stock on a particular market, such as stocks of Facebook are registered on NASDAQ as FB and stocks of Apple as AAPL. “

http://www.wikijava.org/wiki/Downloading_stock_market_quotes_from_Yahao!_finance

Mocking objects : Creating Object Mocks

1. A mock can be created with the help of a static mock() method

```
    MarketWatcher marketWatcher =Mockito.mock(MarketWatcher.class);
            Portfolio portfolio = Mockito.mock(Portfolio.class);
```

2. Java's static import feature and static import the mock method of the **org.mockito.Mockito** class

```
    MarketWatcher marketWatcher = mock(MarketWatcher.class);
    Portfolio portfolio = mock(Portfolio.class);
```

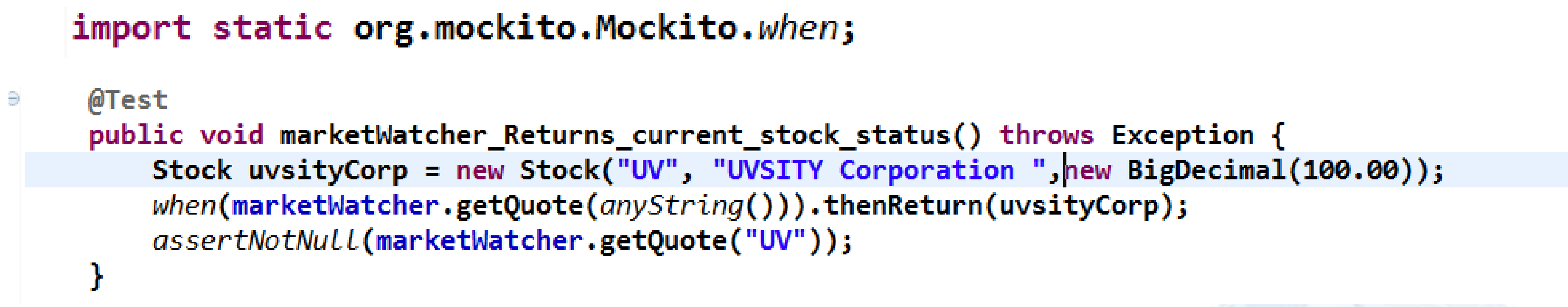
3. @Mock annotation

```
@Mock
MarketWatcher marketWatcher;
@Mock
Portfolio portfolio;
```

To work with the @Mock annotation, you are required to call **MockitoAnnotations.initMocks(this)** before using the mocks, or use **MockitoJUnitRunner** as a JUnit runner

Mockito : Stubbing methods

- The Mockito framework supports stubbing and allows us to return a given value when a specific method is called.
- This can be done using **Mockito.when() along with thenReturn ().**



```
import static org.mockito.Mockito.when;

@Test
public void marketWatcher_Returns_current_stock_status() throws Exception {
    Stock uvsityCorp = new Stock("UV", "UVSITY Corporation ",new BigDecimal(100.00));
    when(marketWatcher.getQuote(anyString())).thenReturn(uvsityCorp);
    assertNotNull(marketWatcher.getQuote("UV"));
}
```

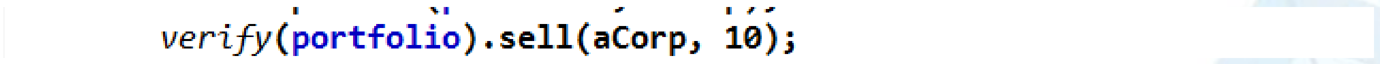
- The when() method represents the trigger, that is, when to stub.
- The following methods are used to represent what to do when the trigger is triggered:
- thenReturn(x): This returns the x value.
- thenThrow(x): This throws an x exception.
- thenAnswer(Answer answer): Unlike returning a hardcoded value, a dynamic user-defined logic is executed.
- thenCallRealMethod(): This method calls the real method on the mock object.

Verifying the method invocation

- To verify a redundant method invocation, or to verify whether a stubbed method was not called but was important from the test perspective.
- We should manually verify the invocation; for this, we need to use the static **verify().**

Why do we use verify?

- The verify method verifies the invocation of mock objects.
- Mockito does not automatically verify all stubbed calls .
- The void methods don't return values, so you cannot assert the returned values.
- Hence, verify is very handy to test the void methods.



```
verify(portfolio).sell(aCorp, 10);
```

Verifying the method invocation : Times

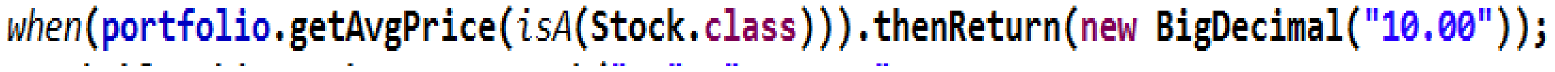
- ❑ The verify() method has an overloaded version that takes Times as an argument. If 0 is passed to Times, it infers that the method will not be invoked in the testing path.

The following methods are used in conjunction with verify:

- **times(int wantedNumberOfInvocations):** This method is invoked exactly *n* times .
- **never():** This method signifies that the stubbed method is never called.
- **atLeastOnce():** This method is invoked at least once, and it works fine if it is invoked multiple times.
- **atLeast(int minNumberOfInvocations):** This method is called at least *n* times, and it works fine if the method is invoked more
- **atMost(int maxNumberOfInvocations**): This method is called at the most n times
- **only():** The only method called on a mock fails if any other method is called on the mock object
- **timeout(int millis):** This method is interacted in a specified time range.

Using argument matcher

- ❑ ArgumentMatcher is a Hamcrest matcher with a predefined describeTo() method.
- ❑ The Matchers.argThat(Matcher) method is used in conjunction with the verify method to verify whether a method is invoked with a specific argument value.
- ❑ Mockito returns expected values when a method is stubbed. If the method takes arguments, the argument must match during the execution;



```
when(portfolio.getAvgPrice(isA(Stock.class))).thenReturn(new BigDecimal("10.00"));
```

- ❑ Mockito provides built-in matchers such as anyInt(), anyDouble(), anyString(), anyList(), and anyCollection().
- ❑ The isA argument checks whether the passed object is an instance of the class type passed in the isA argument

For More Detail about built in Matchers:
http://docs.mockito.googlecode.com/hg/latest/org/mockito/Matchers.html

Wildcard matchers are used to verify the indirect inputs to the mocked dependencies .

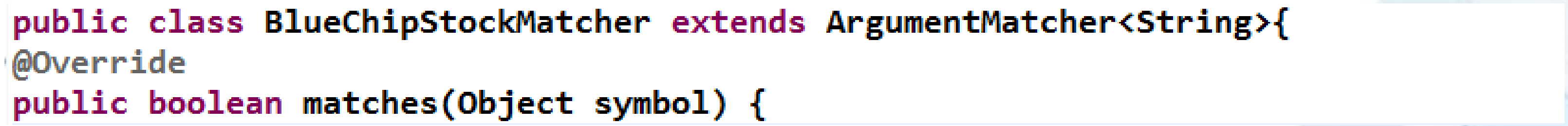
Using argument matcher : ArgumentMatcher class

The ArgumentMatcher class allows the creation of customized argument matchers.

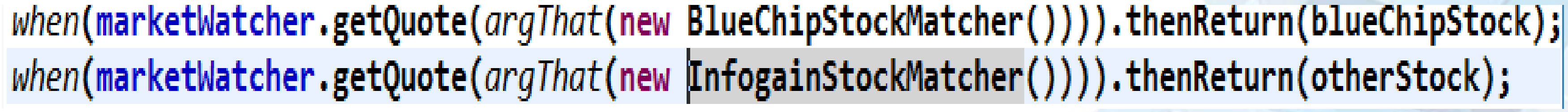
We will create a mock for the MarketWatcher.getQuote method that takes a String object.

We wish to make this method conditional.

If a blue chip stock symbol is passed to the method, then the method will return $1000.00; otherwise, it will return $5.00.



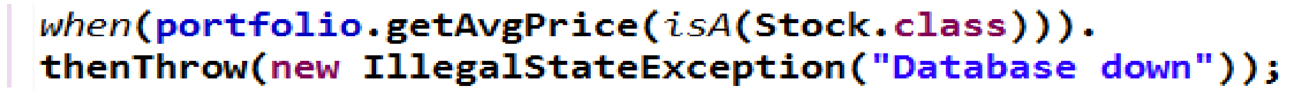
```
public class BlueChipStockMatcher extends ArgumentMatcher<String>{
@Override
public boolean matches(Object symbol) {
```



```
when(marketWatcher.getQuote(argThat(new BlueChipStockMatcher()))).thenReturn(blueChipStock);
when(marketWatcher.getQuote(argThat(new InfogainStockMatcher()))).thenReturn(otherStock);
```

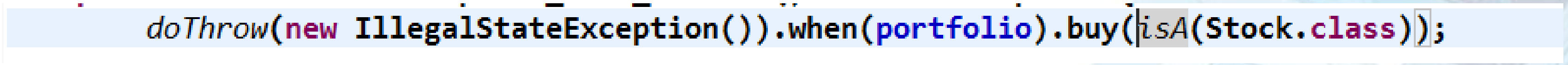
Throwing exceptions

- ❑ Unit tests are not meant for only happy path testing.
- ❑ We should test our code for the failure conditions too.
- ❑ Mockito provides an API to raise an error during testing.
- ❑ Mockito provides a method called **thenThrow(Throwable**); this method throws an exception when the stubbed method is invoked.



```java
when(portfolio.getAvgPrice(isA(Stock.class))).
thenThrow(new IllegalStateException("Database down"));
```

The following is the syntax to throw an exception from a method that returns void:
doThrow(exception).when(mock).voidmethod(arguments);

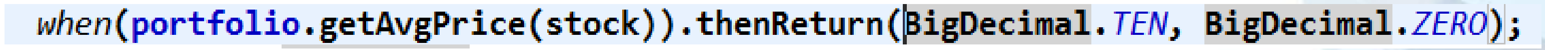


```java
        doThrow(new IllegalStateException()).when(portfolio).buy(isA(Stock.class));
```

Stubbing consecutive calls

Stubbing a method for consecutive calls is required in the following situations:

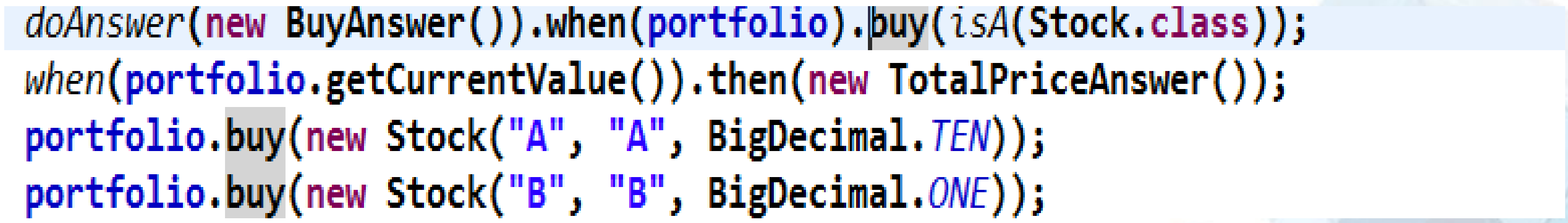
- ❑ Calling a stubbed method in a loop when you need different results for different calls
- ❑ When you need one invocation to throw an exception and other invocations to return a value
- ❑ We need to test a condition where the first call will return a value, the next call should not find any value, and then again it should return a value.
- ❑ The varargs version of thenReturn(objects...) takes comma-separated return values
  .



```
when(portfolio.getAvgPrice(stock)).thenReturn(BigDecimal.TEN, BigDecimal.ZERO);
```

Stubbing with an Answer

- ❑ Stubbed methods return a hardcoded value but cannot return an on the fly result.
- ❑ The Mockito framework provides the callbacks to compute the on the fly results.
- ❑ Mockito allows stubbing with the generic Answer interface.
- ❑ This is a callback; when a stubbed method on a mock object is invoked, the answer(InvocationOnMock invocation) method of the Answer object is called.
- ❑ This Answer object's answer() method returns the actual object.



```
doAnswer(new BuyAnswer()).when(portfolio).buy(isA(Stock.class));
when(portfolio.getCurrentValue()).then(new TotalPriceAnswer());
portfolio.buy(new Stock("A", "A", BigDecimal.TEN));
portfolio.buy(new Stock("B", "B", BigDecimal.ONE));
```

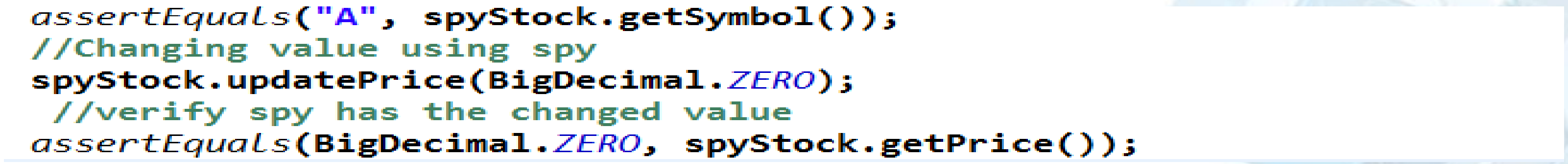
Spying objects

- ❑ A Mockito spy object allows us to use real objects instead of mocks by replacing some of the methods with the stubbed ones.
- ❑ This behavior allows us to test the legacy code; one cannot mock a class that needs to be tested.
- ❑ Legacy code comes with methods that cannot be tested, but other methods use them; so, these methods need to be stubbed to work with the other methods.
- ❑ A spy object can stub the nontestable methods so that other methods can be tested easily.

Spy is also known as partial mock; one example of the use of spy in the real world is dealing with legacy code.

Declaration of spy is done using the following code:

```
SomeClass realObject = new RealImplemenation();
SomeClass spyObject = spy(realObject);
```



```
assertEquals("A", spyStock.getSymbol());
//Changing value using spy
spyStock.updatePrice(BigDecimal.ZERO);
 //verify spy has the changed value
assertEquals(BigDecimal.ZERO, spyStock.getPrice());
```

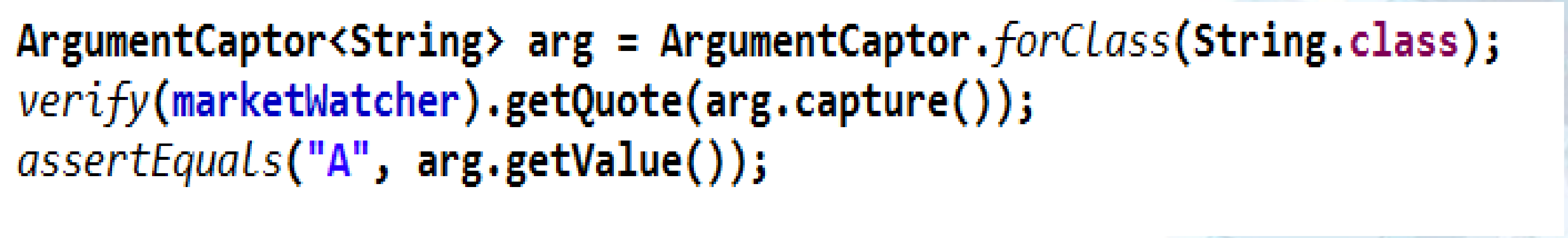
Stubbing void methods

- In the Throwing exceptions , we learned that doThrow is used for throwing exceptions for the void methods.
- The Stubbing with an Answer we learned how to use doAnswer for the void methods.
- Now we will explore the other void methods: doNothing, doReturn, doThrow, and doCallRealMethod.
- The doNothing() API does nothing.
- By default, all the void methods do nothing.
- However, if you need consecutive calls on a void method, the first call is to throw an error, the next call is to do nothing, and then the next call to perform some logic using doAnswer() and Syntax is:

```
doThrow(new RuntimeException()).
doNothing(). doAnswer(someAnswer). when(mock).someVoidMethod();
//this call throws exception
mock.someVoidMethod();
// this call does nothing
mock.someVoidMethod();
```

Capturing arguments with ArgumentCaptor

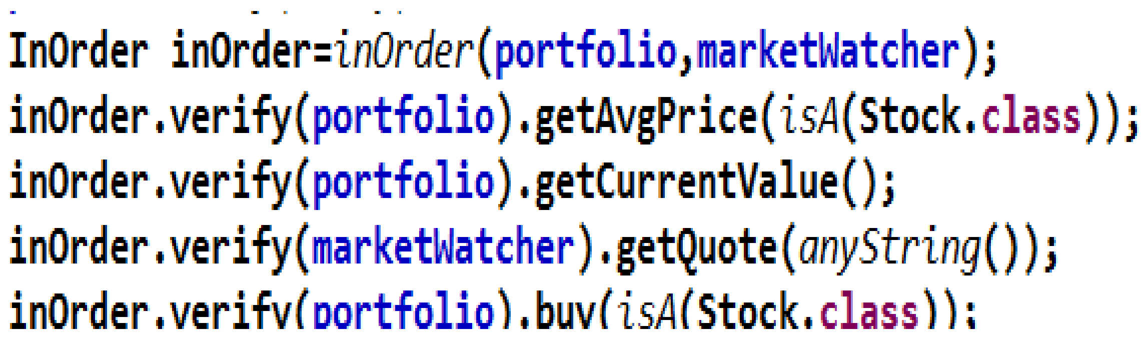
- ArgumentCaptor is used to verify the arguments passed to a stubbed method.
- Sometimes, we compute a value, then create another object using the computed value, and then call a mock object using that new object .
- ArgumentCaptor provides an API to access objects that are instantiated within the method under the test.
- Mockito verifies argument values in natural Java style by using an equals() method.
- This is also the recommended way of matching arguments because it makes tests clean and simple.
- In some situations though, it is necessary to assert on certain arguments after the actual verification



```
ArgumentCaptor<String> arg = ArgumentCaptor.forClass(String.class);
verify(marketWatcher).getQuote(arg.capture());
assertEquals("A", arg.getValue());
```

Verifying the invocation order

- ❑ Mockito facilitates verifying if interactions with a mock were performed in a given order using the InOrder API.
- ❑ It allows us to create InOrder of mocks and verify the call order of all calls of all mocks.



```
InOrder inOrder=inOrder(portfolio,marketWatcher);
inOrder.verify(portfolio).getAvgPrice(isA(Stock.class));
inOrder.verify(portfolio).getCurrentValue();
inOrder.verify(marketWatcher).getQuote(anyString());
inOrder.verify(portfolio).buy(isA(Stock.class));
```

Changing the default settings

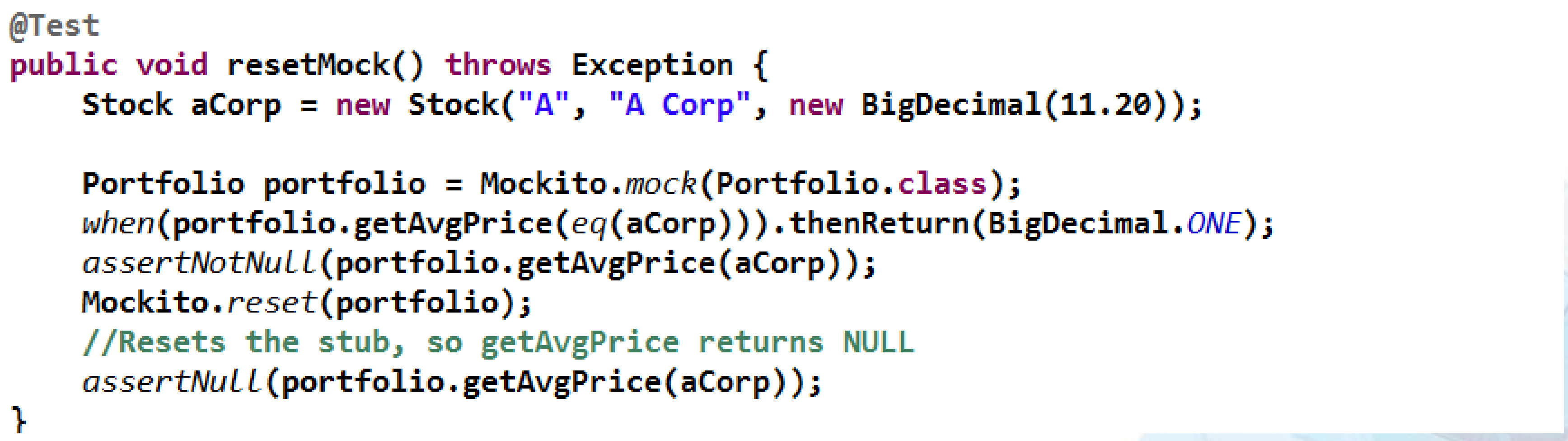
- Nonstubbed methods of a mock object return default values such as null for an object and false for a Boolean. However, Mockito allows us to change the default settings.

The following are the allowed settings:

- **RETURNS_DEFAULTS**: This is the default setting. It returns null for object, false for Boolean, and so on.
- **RETURNS_SMART_NULLS**: This returns spy of a given type.
- **RETURNS_MOCKS:** This returns mocks for objects and the default value for primitives.
- **RETURNS_DEEP_STUBS:** This returns a deep stub.
- **CALLS_REAL_METHODS:** This calls a real method.

Resetting mock objects

- ❑ A static method reset(T…) enables resetting mock objects. The reset method should be handled with special care; if you need to reset a mock, you will most likely need another test.
- ❑ A reset method clears the stubs.



```
@Test
public void resetMock() throws Exception {
    Stock aCorp = new Stock("A", "A Corp", new BigDecimal(11.20));

    Portfolio portfolio = Mockito.mock(Portfolio.class);
    when(portfolio.getAvgPrice(eq(aCorp))).thenReturn(BigDecimal.ONE);
    assertNotNull(portfolio.getAvgPrice(aCorp));
    Mockito.reset(portfolio);
    //Resets the stub, so getAvgPrice returns NULL
    assertNull(portfolio.getAvgPrice(aCorp));
}
```

Exploring Mockito annotations

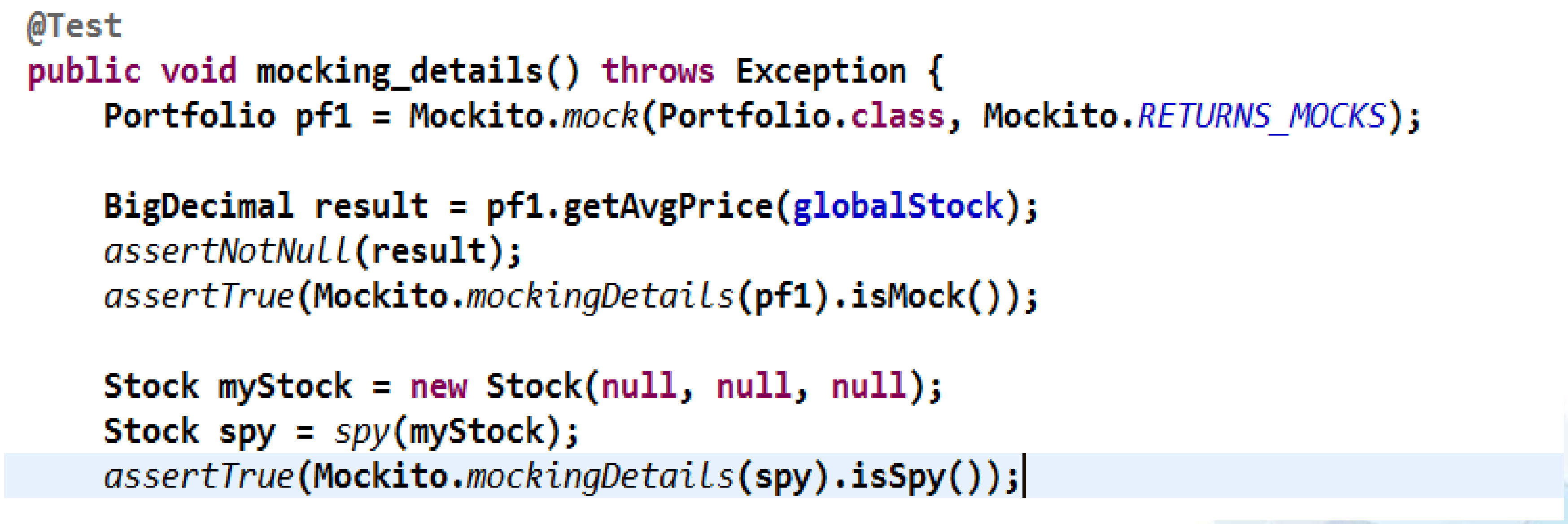
Mockito supports the @Mock annotation for mocking.

Mockito supports the following three useful annotations:

- **@Captor**: This simplifies the creation of ArgumentCaptor, which is useful when the argument to capture is a super generic class, such as List<Map<String,Set<String>>.
- **@Spy**: This creates a spy of a given object. Use it instead of spy (object).
- **@InjectMocks:** This injects mock or spy fields into the tested object automatically using a constructor injection, setter injection, or field injection.

Determining mocking details

- Mockito.mockingDetails identifies whether a particular object is a mock or a spy.



```
@Test
public void mocking_details() throws Exception {
    Portfolio pf1 = Mockito.mock(Portfolio.class, Mockito.RETURNS_MOCKS);

    BigDecimal result = pf1.getAvgPrice(globalStock);
    assertNotNull(result);
    assertTrue(Mockito.mockingDetails(pf1).isMock());

    Stock myStock = new Stock(null, null, null);
    Stock spy = spy(myStock);
    assertTrue(Mockito.mockingDetails(spy).isSpy());
```

Behavior-driven development with Mockito

- ❑ BDD is a software engineering process based on TDD.
- ❑ BDD combines the best practices of TDD, domain-driven development (DDD), and object-oriented programming (OOP).
- ❑ The business stakeholders talk about business interests, and the development team talks about technical challenges.
- ❑ BDD provides a universal language that allows useful communication and feedback between the stakeholders .
- ❑ **Dan North developed BDD, created the JBehave framework for BDD**, and proposed the following best practices:
- ❑ Unit test names should start with the word should and should be written in the order of the business value
- ❑ Acceptance tests (AT) should be written in a user story manner, such as "As a (role) I want (feature) so that (benefit)"
- ❑ Acceptance criteria should be written in terms of scenarios and implemented as "Given (initial context), when (event occurs), then (ensure some outcomes)"

BDD: Cont..

- ❑ Let's write a user story for our stock broker simulation:

  Story: A stock is sold

  In order to maximize profit

  As a Stock broker

  I want to sell a stock w**hen** the price goes up by 10 percent

- ❑ The following is a scenario example:

  Scenario: 10 percent increase in stock price should sell the stock in the market

  Given a customer previously bought FB stocks at $10.00 per share

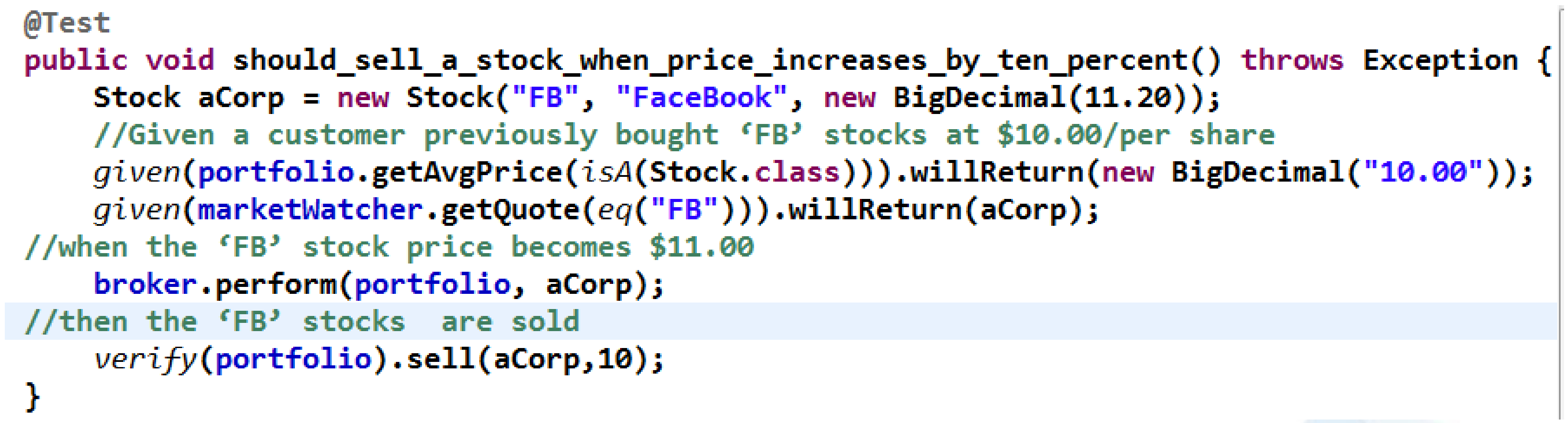
  And he currently has 10 shares left in his portfolio

  When the FB stock price becomes $11.00

  Then I should sell all the FB stocks and the portfolio should have zero FB stocks

Mockito supports the BDD style of writing tests using the **given-when-then** syntax.

BDD: Cont..



```
@Test
public void should_sell_a_stock_when_price_increases_by_ten_percent() throws Exception {
    Stock aCorp = new Stock("FB", "FaceBook", new BigDecimal(11.20));
    //Given a customer previously bought 'FB' stocks at $10.00/per share
    given(portfolio.getAvgPrice(isA(Stock.class))).willReturn(new BigDecimal("10.00"));
    given(marketWatcher.getQuote(eq("FB"))).willReturn(aCorp);
//when the 'FB' stock price becomes $11.00
    broker.perform(portfolio, aCorp);
//then the 'FB' stocks  are sold
    verify(portfolio).sell(aCorp,10);
}
```

For BDD Detail : http://jbehave.org/

infogain

Thank You

April 23, 2019