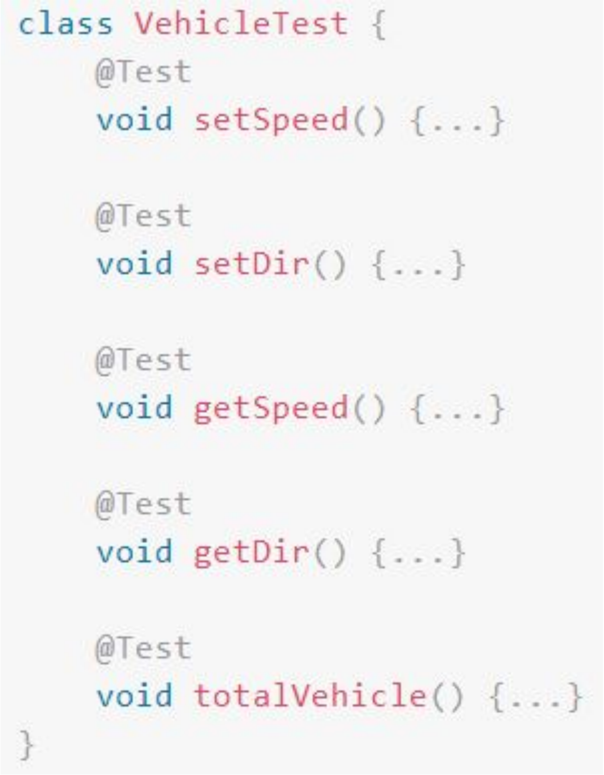
Lab 2: Advanced Unit Testing

Software Testing 2021

Week 3

Remember Lab 1 ?

Class Vehicle

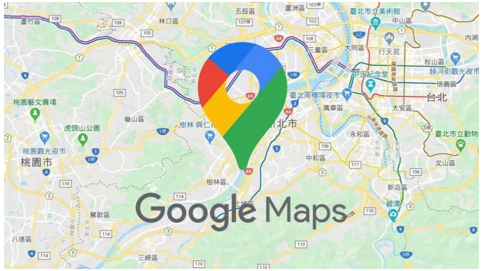


```
class VehicleTest {
    @Test
    void setSpeed() {...}

    @Test
    void setDir() {...}

    @Test
    void getSpeed() {...}

    @Test
    void getDir() {...}

    @Test
    void totalVehicle() {...}
}
```

An Intelligent Vehicle, How To Test It?

Spotify®

Sign in with Google



NETFLIX

Problem

Usually, the classification to be tested will have some external dependencies, may cause:

- Testing may be slow due to dependencies.

  eg. Network, database, files, external objects, etc.

- The result of the misjudgment test is whether the SUT itself is wrong or the dependent object is wrong

- Wait for the development of dependent objects to be completed before testing the object under test

- Unable to test.

  eg. the development environment is different from the formal environment

Solution: Test Double

Dummy:

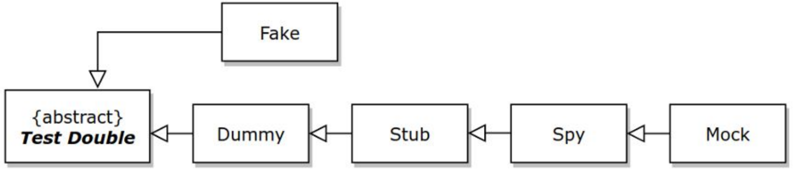
- It is used as a placeholder when an argument needs to be filled in.

Stub:

- It provides fake data to the SUT (System Under Test).

Spy:

- It records information about how the class is being used.



Mock:

- It defines an expectation of how it will be used. It will cause a failure if the expectation isn't met.

Fake:

- It is an actual implementation of the contract but is unsuitable for production.

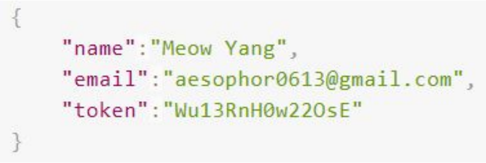
Reference: https://jesusvalerareales.medium.com/testing-with-test-doubles-7c3abb9eb3f2

Example

Real World

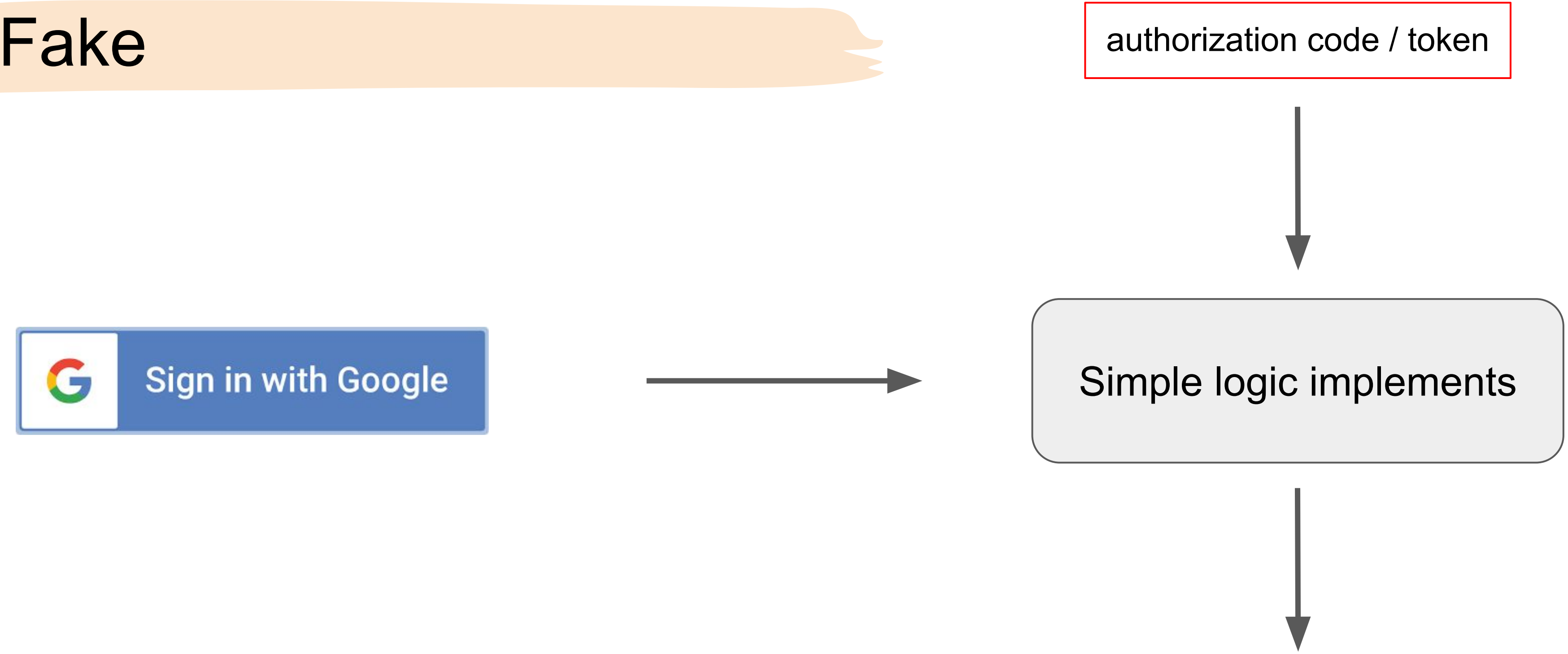
authorization code / token

Sign in with Google



```
{
    "name":"Meow Yang",
    "email":"aesophor0613@gmail.com",
    "token":"Wu13RnH0w22OsE"
}
```

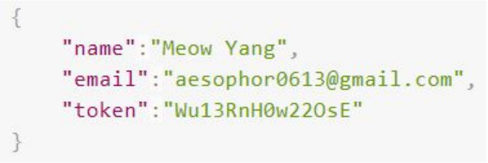
Fake



authorization code / token

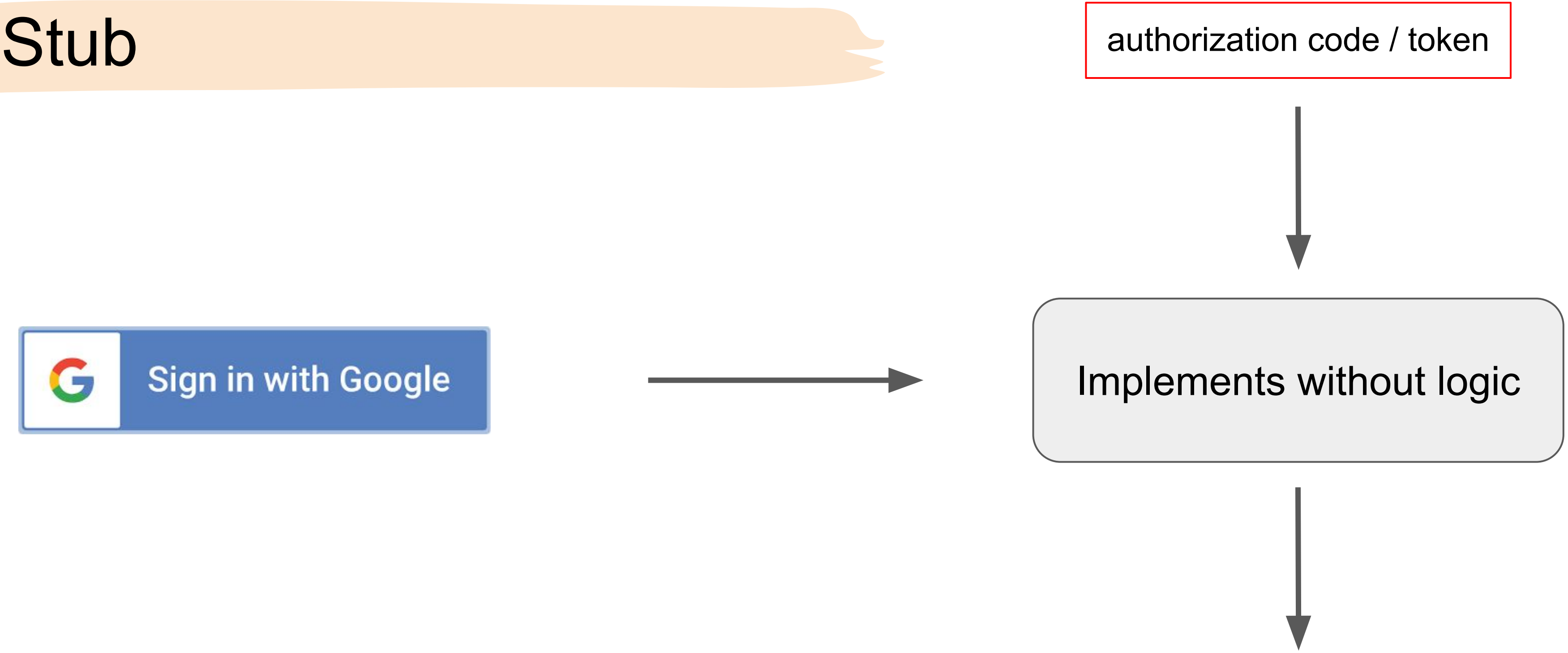
Sign in with Google

Simple logic implements



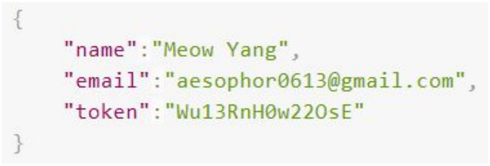
```
{
    "name":"Meow Yang",
    "email":"aesophor0613@gmail.com",
    "token":"Wu13RnH0w22OsE"
}
```

Stub



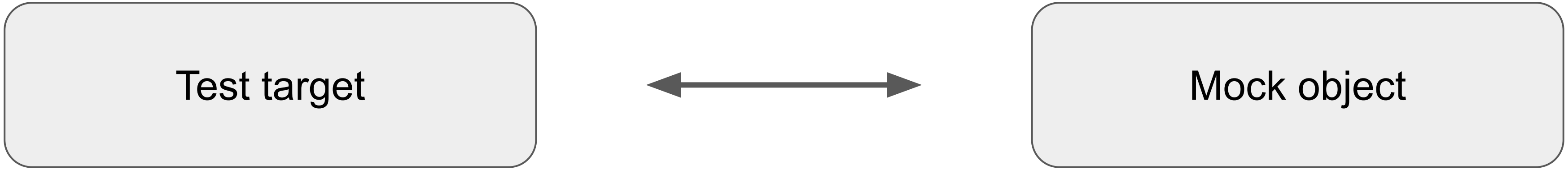
authorization code / token

Implements without logic



```
{
    "name":"Meow Yang",
    "email":"aesophor0613@gmail.com",
    "token":"Wu13RnH0w22OsE"
}
```

Mock

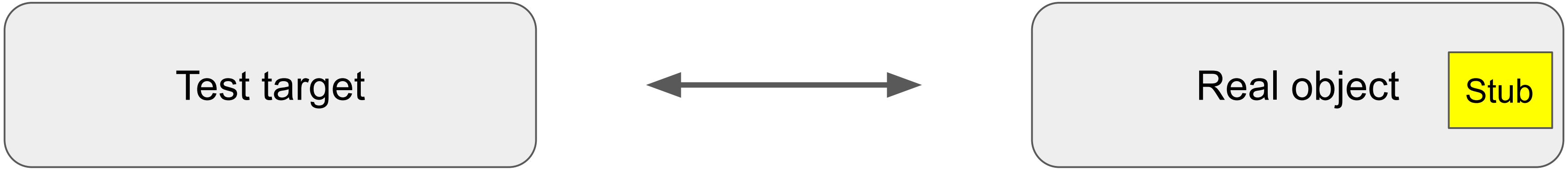


Test target

Mock object

Only care the interactive between target and Mock object

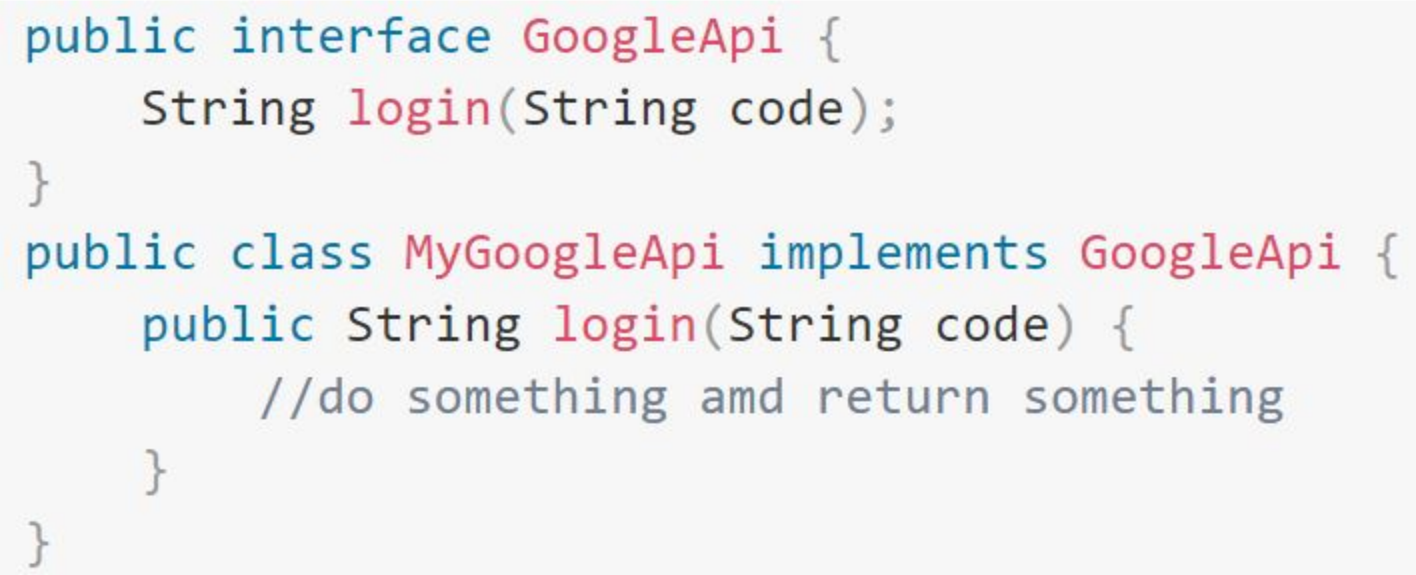
Spy



Can check the interactive between target and Mock object

Fake

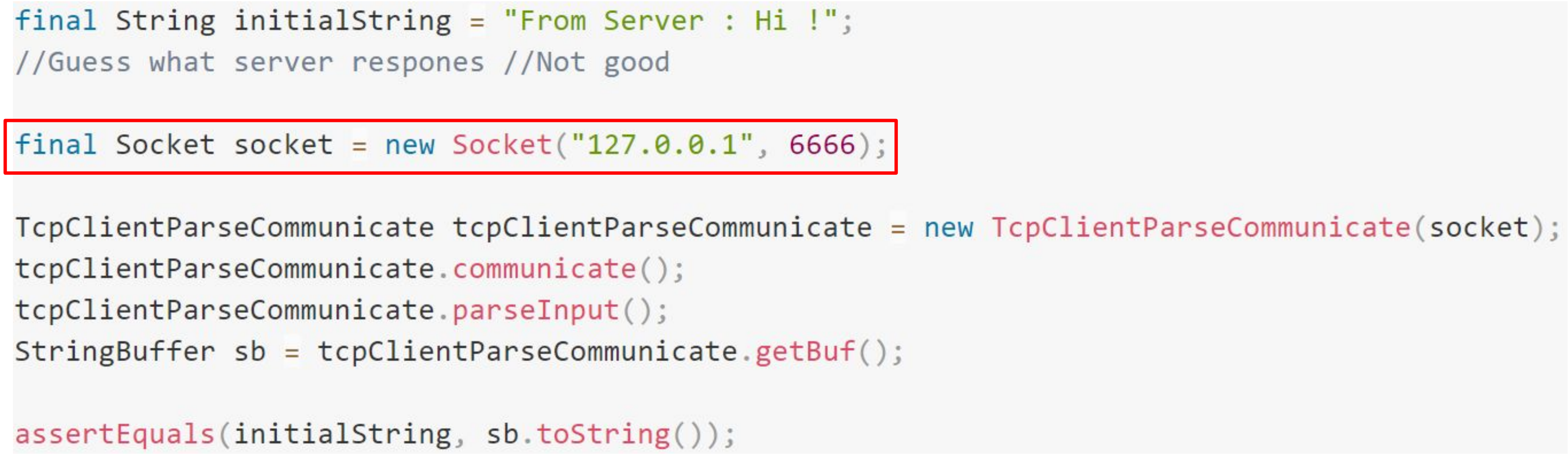
Fake



```
public interface GoogleApi {
    String login(String code);
}
public class MyGoogleApi implements GoogleApi {
    public String login(String code) {
        //do something amd return something
    }
}
```

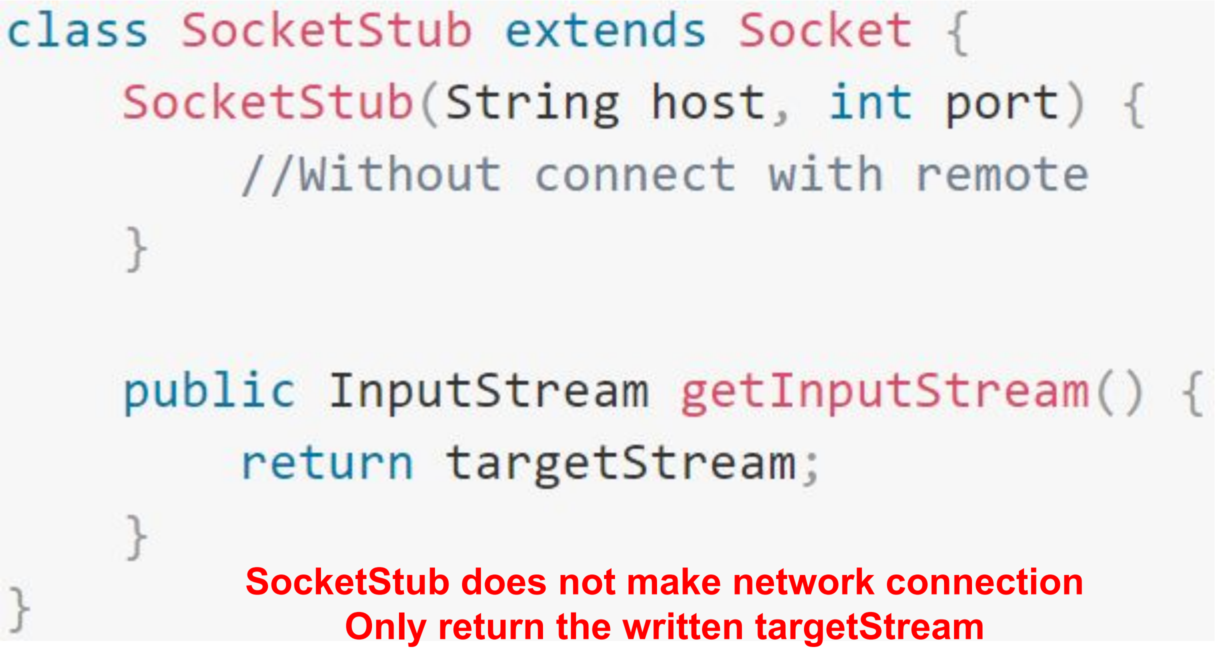
Stub

In Case Require Network Connection



```
final String initialString = "From Server : Hi !";
//Guess what server respones //Not good

final Socket socket = new Socket("127.0.0.1", 6666);

TcpClientParseCommunicate tcpClientParseCommunicate = new TcpClientParseCommunicate(socket);
tcpClientParseCommunicate.communicate();
tcpClientParseCommunicate.parseInput();
StringBuffer sb = tcpClientParseCommunicate.getBuf();

assertEquals(initialString, sb.toString());
```

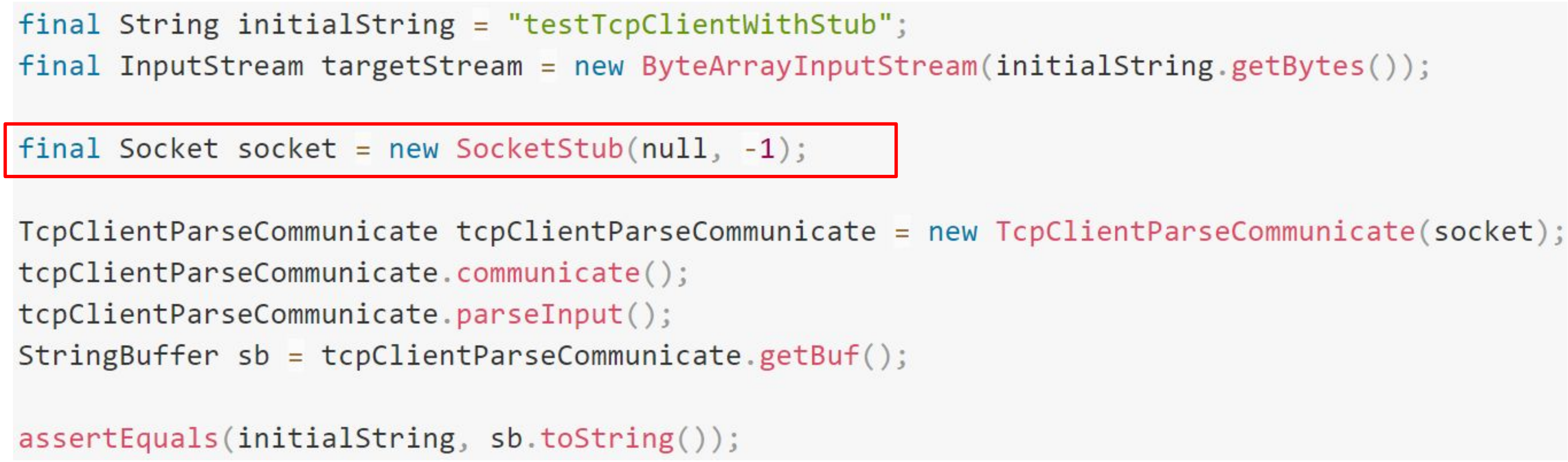
Stub Test



```java
class SocketStub extends Socket {
    SocketStub(String host, int port) {
        //Without connect with remote
    }

    public InputStream getInputStream() {
        return targetStream;
    }
}
```

SocketStub does not make network connection
Only return the written targetStream

Stub Test - Cont.



```
final String initialString = "testTcpClientWithStub";
final InputStream targetStream = new ByteArrayInputStream(initialString.getBytes());

final Socket socket = new SocketStub(null, -1);

TcpClientParseCommunicate tcpClientParseCommunicate = new TcpClientParseCommunicate(socket);
tcpClientParseCommunicate.communicate();
tcpClientParseCommunicate.parseInput();
StringBuffer sb = tcpClientParseCommunicate.getBuf();

assertEquals(initialString, sb.toString());
```

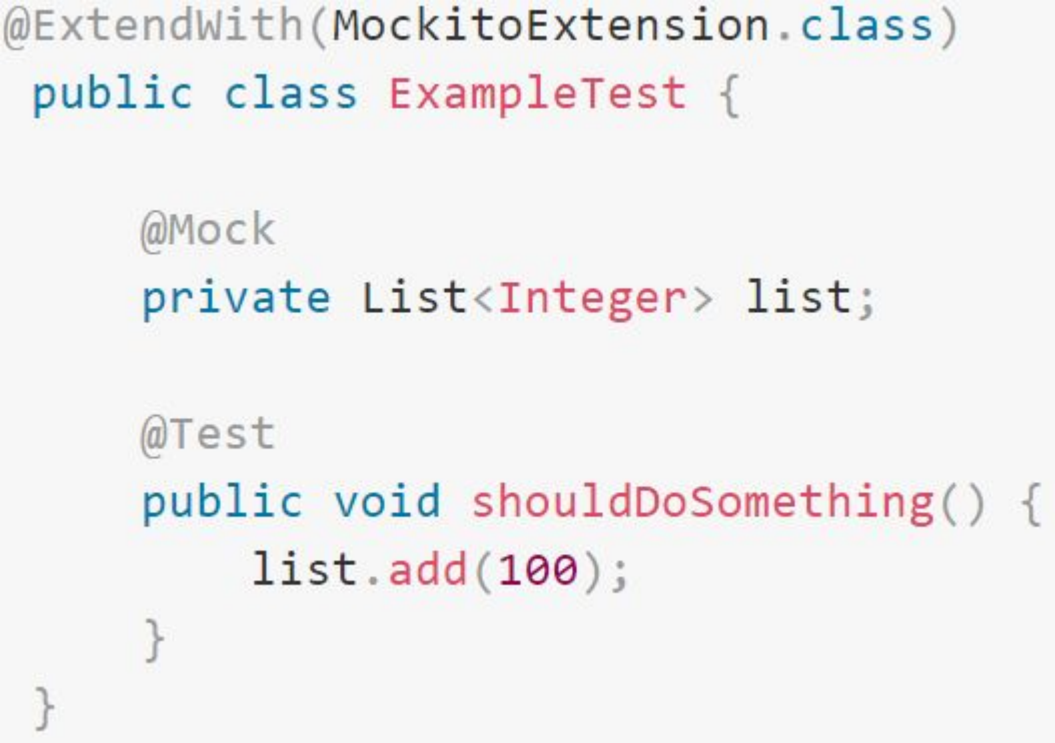
Mockito

It is a widely used testing framework, especially it can easily handle dependency injection scenarios, and it is relatively helpful to write Unit Test with it.

Can more easily handle and construct a variety of Test Double to conduct Unit Test.

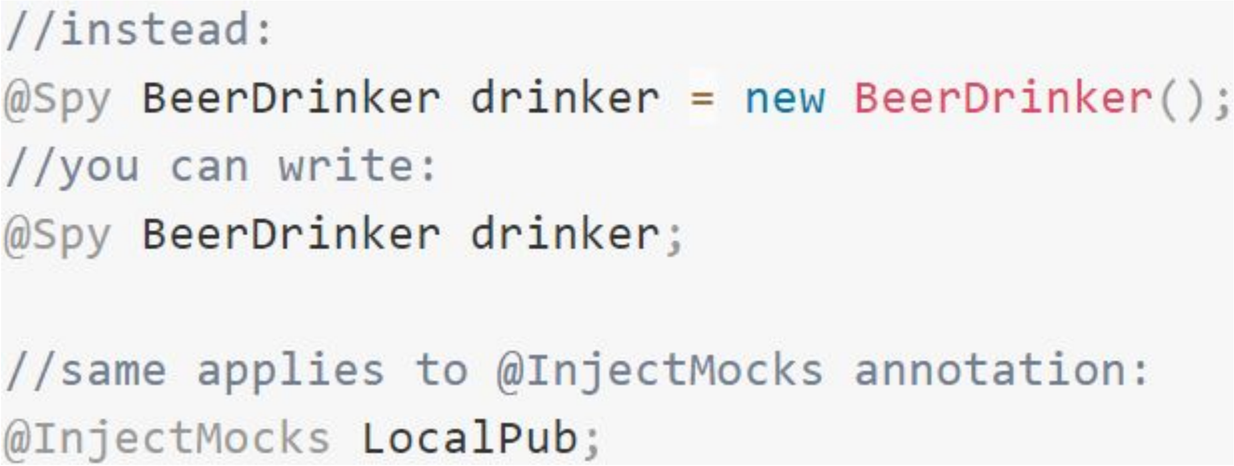
https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html

Basic Structure



```java
@ExtendWith(MockitoExtension.class)
 public class ExampleTest {

     @Mock
     private List<Integer> list;

     @Test
     public void shouldDoSomething() {
         list.add(100);
     }
 }
```

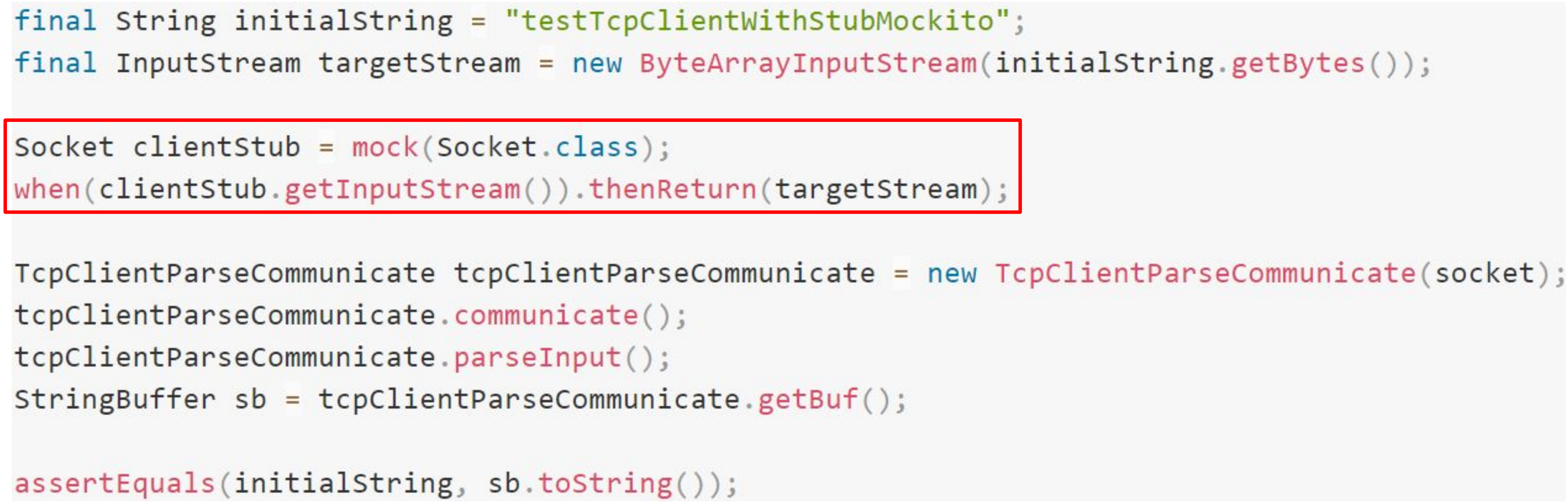
Constructor Injection



```
//instead:
@Spy BeerDrinker drinker = new BeerDrinker();
//you can write:
@Spy BeerDrinker drinker;

//same applies to @InjectMocks annotation:
@InjectMocks LocalPub;
```

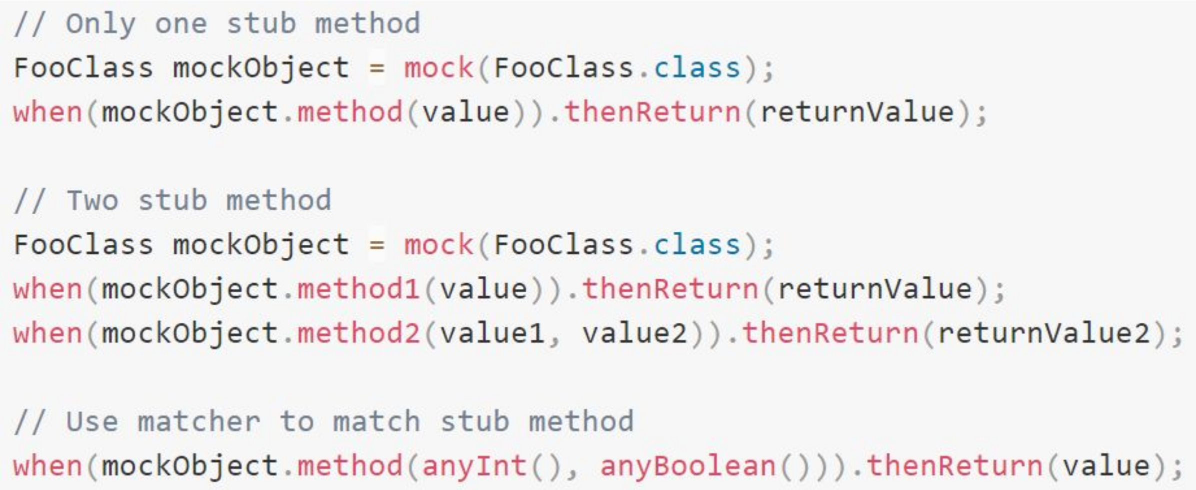
mockito will try to initialize the **@InjectMocks** variables in **@Spy**, either by constructing method, set method or variable injection.

Stub Test With Mockito



```
final String initialString = "testTcpClientWithStubMockito";
final InputStream targetStream = new ByteArrayInputStream(initialString.getBytes());

Socket clientStub = mock(Socket.class);
when(clientStub.getInputStream()).thenReturn(targetStream);

TcpClientParseCommunicate tcpClientParseCommunicate = new TcpClientParseCommunicate(socket);
tcpClientParseCommunicate.communicate();
tcpClientParseCommunicate.parseInput();
StringBuffer sb = tcpClientParseCommunicate.getBuf();

assertEquals(initialString, sb.toString());
```

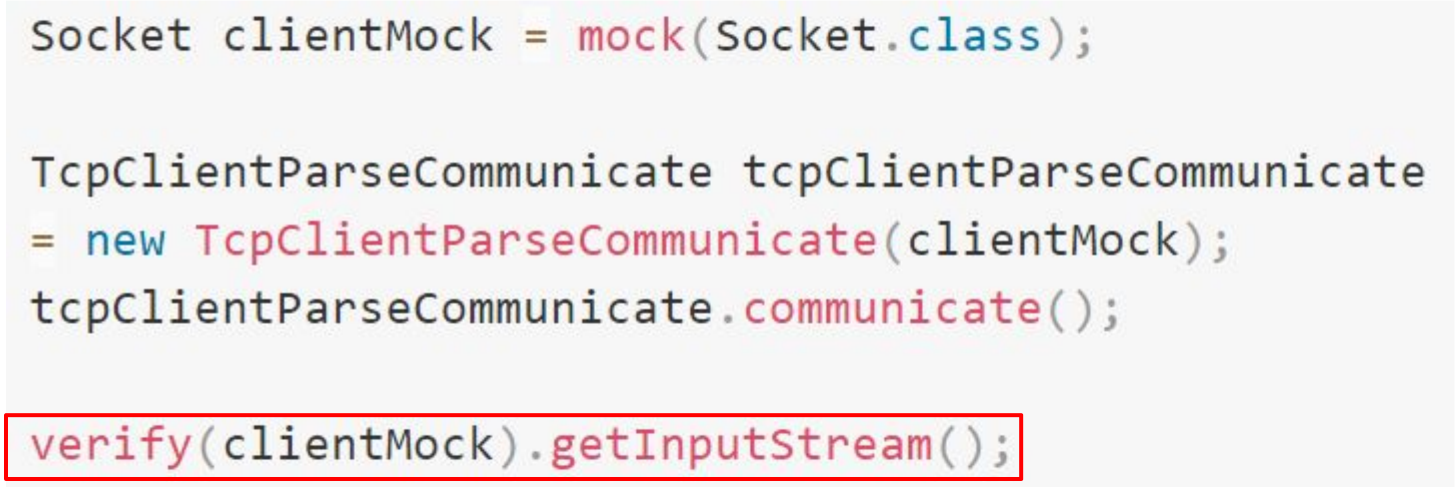
Cheat Sheet



```
// Only one stub method
FooClass mockObject = mock(FooClass.class);
when(mockObject.method(value)).thenReturn(returnValue);

// Two stub method
FooClass mockObject = mock(FooClass.class);
when(mockObject.method1(value)).thenReturn(returnValue);
when(mockObject.method2(value1, value2)).thenReturn(returnValue2);

// Use matcher to match stub method
when(mockObject.method(anyInt(), anyBoolean())).thenReturn(value);
```

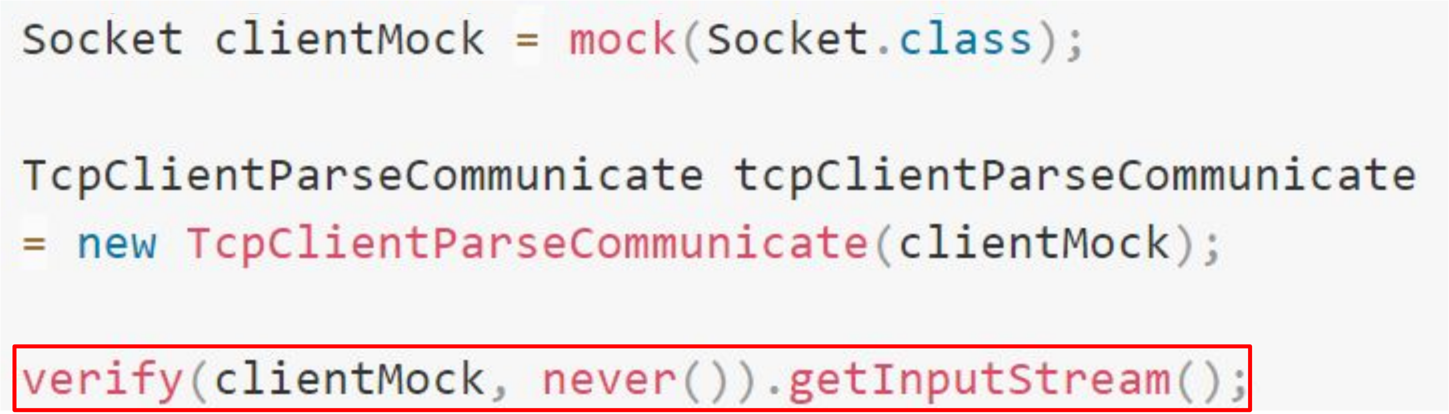
Mock

Mock Test With Mockito



```
Socket clientMock = mock(Socket.class);

TcpClientParseCommunicate tcpClientParseCommunicate
= new TcpClientParseCommunicate(clientMock);
tcpClientParseCommunicate.communicate();

verify(clientMock).getInputStream();
```

Mock Test With Mockito - Cont.



```
Socket clientMock = mock(Socket.class);

TcpClientParseCommunicate tcpClientParseCommunicate
= new TcpClientParseCommunicate(clientMock);

verify(clientMock, never()).getInputStream();
```

Cheat Sheet

- Frequency

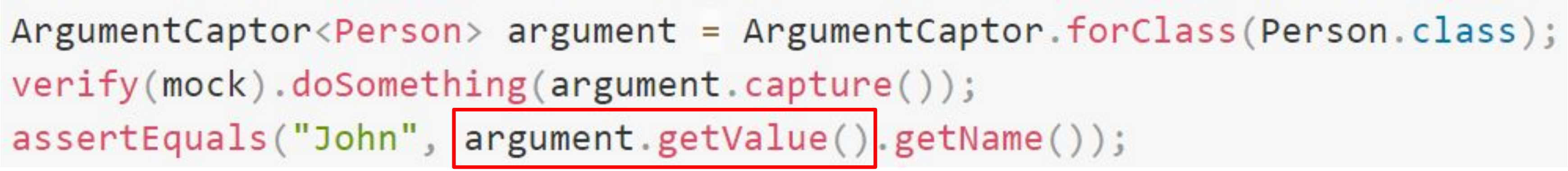
  ```
  verify(mockObject).method();
  verify(mockObject, times(666)).method();
  verify(mockObject, never()).method();
  ```

- Argument Type

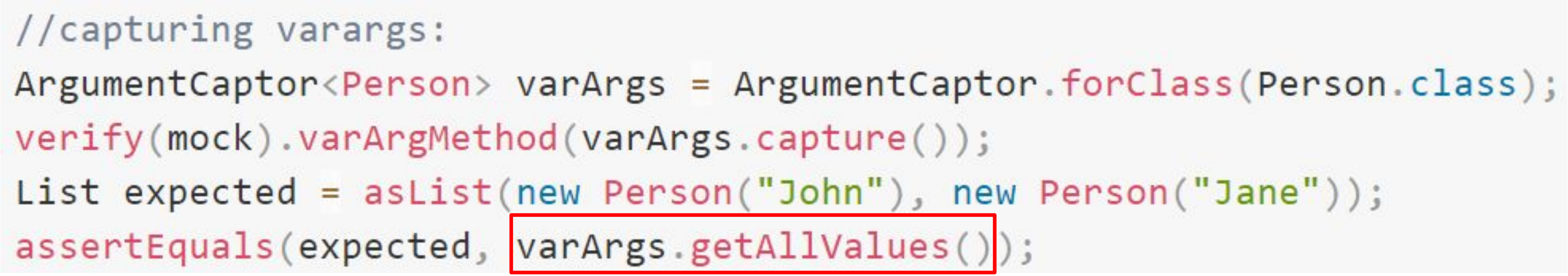
  ```
  verify(mockObject).method( “robert”);
  verify(mockObject).method( anyString() );
  verify(mockObject).method( 2021, 3, 11);
  verify(mockObject).method( anyInt(), anyInt(), anyInt() );
  ```

Cheat Sheet - Cont.

- Capturing Arguments



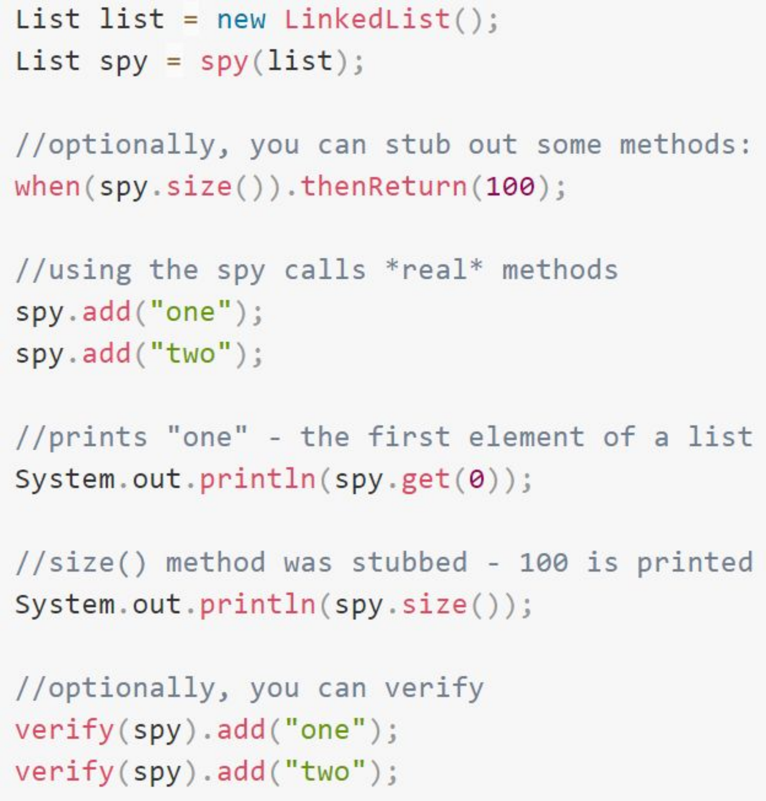
```
ArgumentCaptor<Person> argument = ArgumentCaptor.forClass(Person.class);
verify(mock).doSomething(argument.capture());
assertEquals("John", argument.getValue().getName());
```



```
//capturing varargs:
ArgumentCaptor<Person> varArgs = ArgumentCaptor.forClass(Person.class);
verify(mock).varArgMethod(varArgs.capture());
List expected = asList(new Person("John"), new Person("Jane"));
assertEquals(expected, varArgs.getAllValues());
```

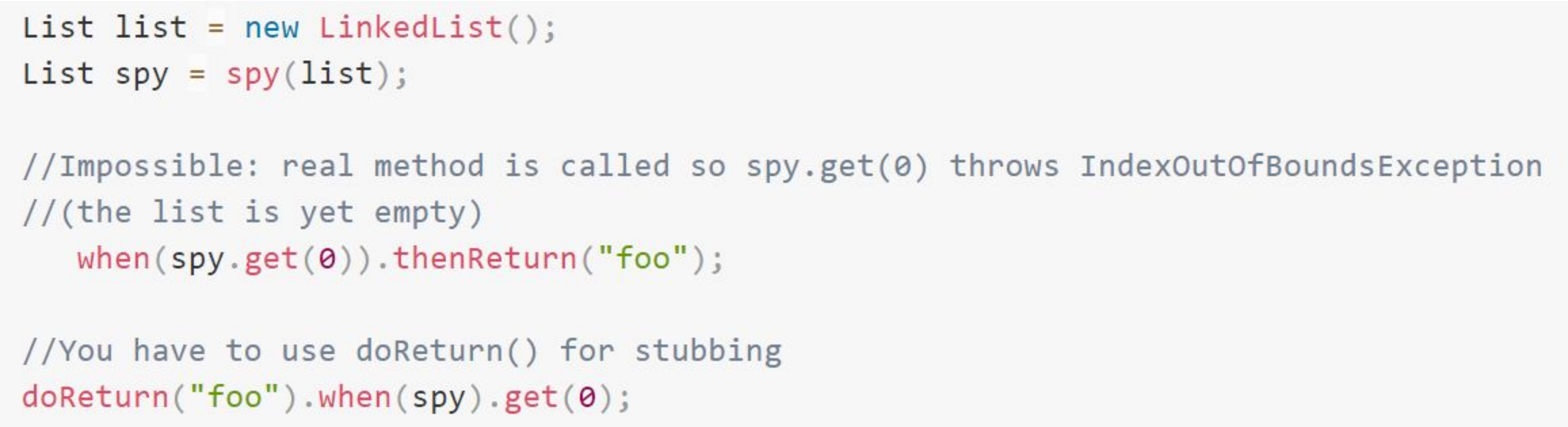
Spy

Example



```
List list = new LinkedList();
List spy = spy(list);

//optionally, you can stub out some methods:
when(spy.size()).thenReturn(100);

//using the spy calls *real* methods
spy.add("one");
spy.add("two");

//prints "one" - the first element of a list
System.out.println(spy.get(0));

//size() method was stubbed - 100 is printed
System.out.println(spy.size());

//optionally, you can verify
verify(spy).add("one");
verify(spy).add("two");
```

Example - Cont.



```
List list = new LinkedList();
List spy = spy(list);

//Impossible: real method is called so spy.get(0) throws IndexOutOfBoundsException
//(the list is yet empty)
   when(spy.get(0)).thenReturn("foo");

//You have to use doReturn() for stubbing
doReturn("foo").when(spy).get(0);
```

Cheat Sheet

You can use doThrow(), doAnswer(), doNothing(), doReturn() and doCallRealMethod() in place of the corresponding call with when(), for any method. It is necessary when you :

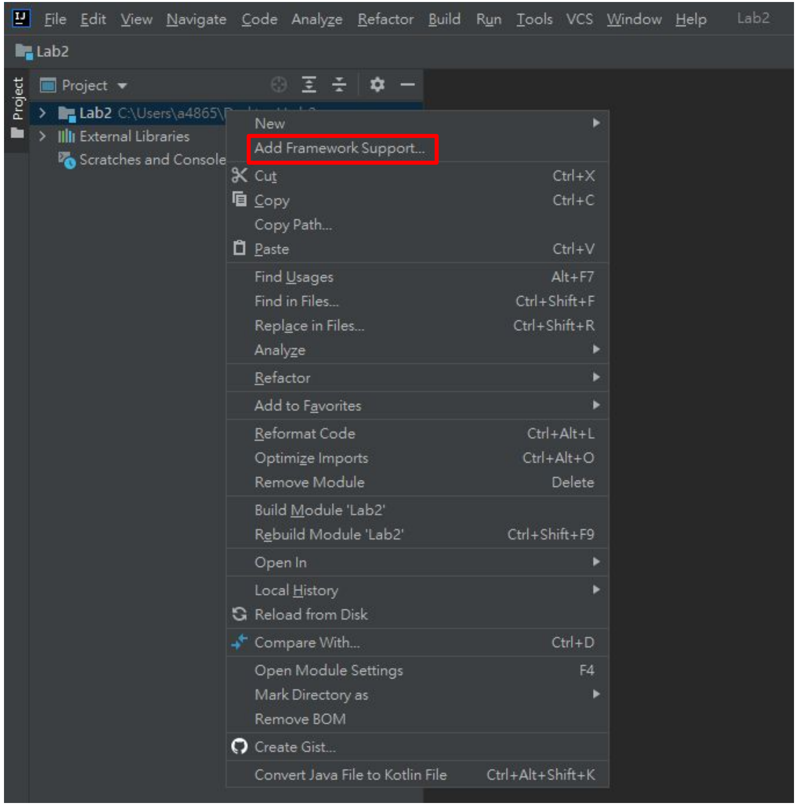
- stub void methods
- stub methods on spy objects
- stub the same method more than once, to change the behaviour of a mock in the middle of a test.

Lab

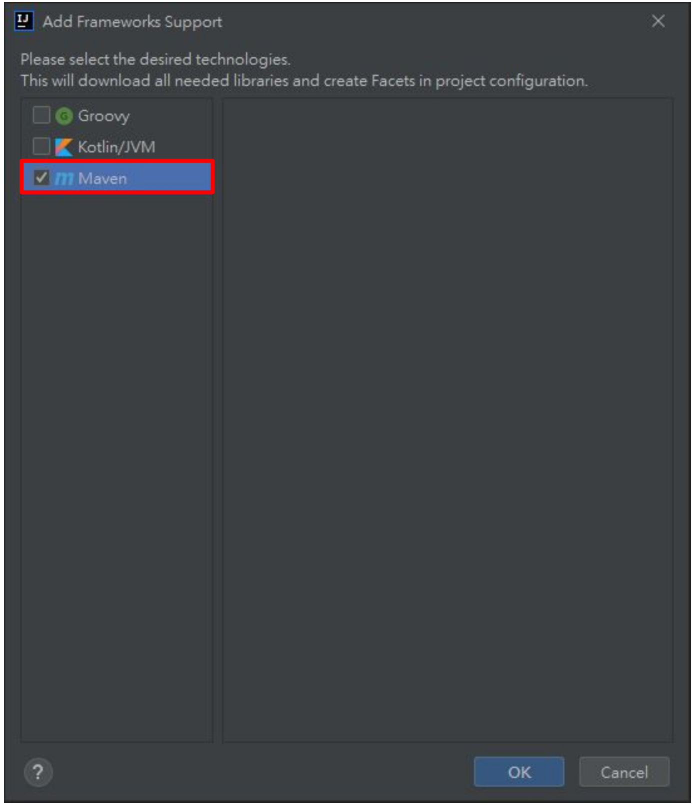
Lab 2

1. Download **SoftwareTesting2021.java** from Github.
    a. https://github.com/iasthc/NYCU-Software-Testing-2021
2. Write tests for SoftwareTesting2021 class which satisfy the following case:
    a. If a **fever** student **enter the class** on **Wednesday** , verify that hospital doesn’t do any treatment.
    b. If a **fever** student **enter the class** on **Thursday**, assert the output correct.
    c. Assume 3 students go to hospital. Verify **patientLog** in **hospital** will record patient’s **studentid** with **spy** method. **Don’t stub getLog function**.
    d. Use **stub** method to test **getScore** function to avoid connection to outer database.
    e. Implement **paypalService** interface as a **fake** object to test donate function.
3. Name your test function test_a to test_e which belong to each case.
4. Upload SoftwareTesting2021Test.java to E3

Import mockito - Method 1 (Maven)



Import mockito - Method 1

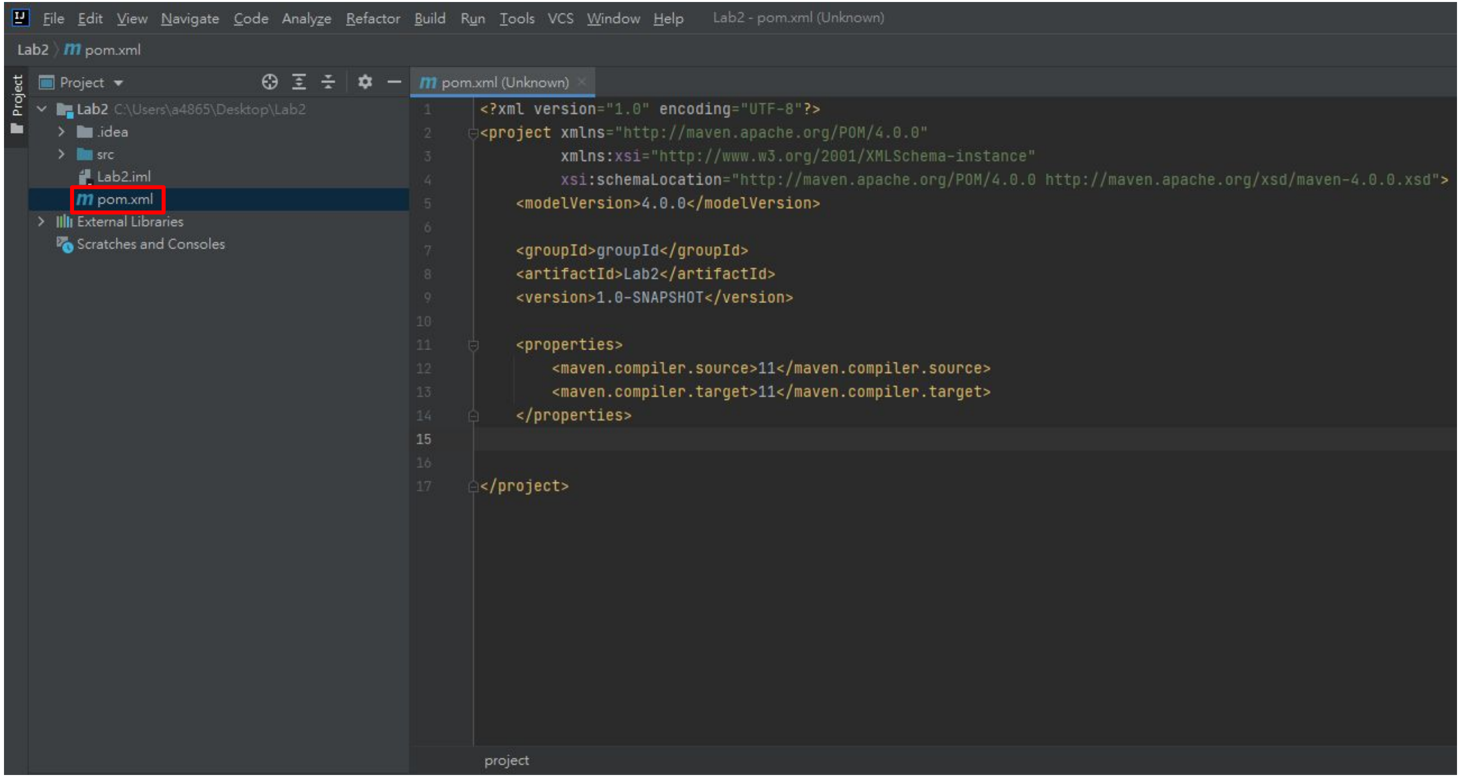


Add Frameworks Support

Please select the desired technologies.
This will download all needed libraries and create Facets in project configuration.

- □ Groovy
- □ Kotlin/JVM
- ☑ Maven

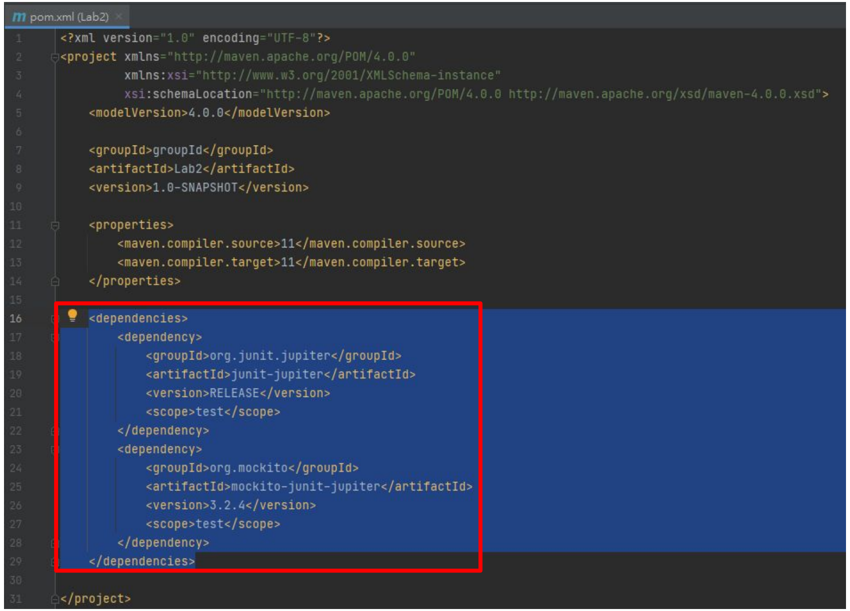
Import mockito - Method 1



```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>groupId</groupId>
    <artifactId>Lab2</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>


</project>
```

Import mockito - Method 1

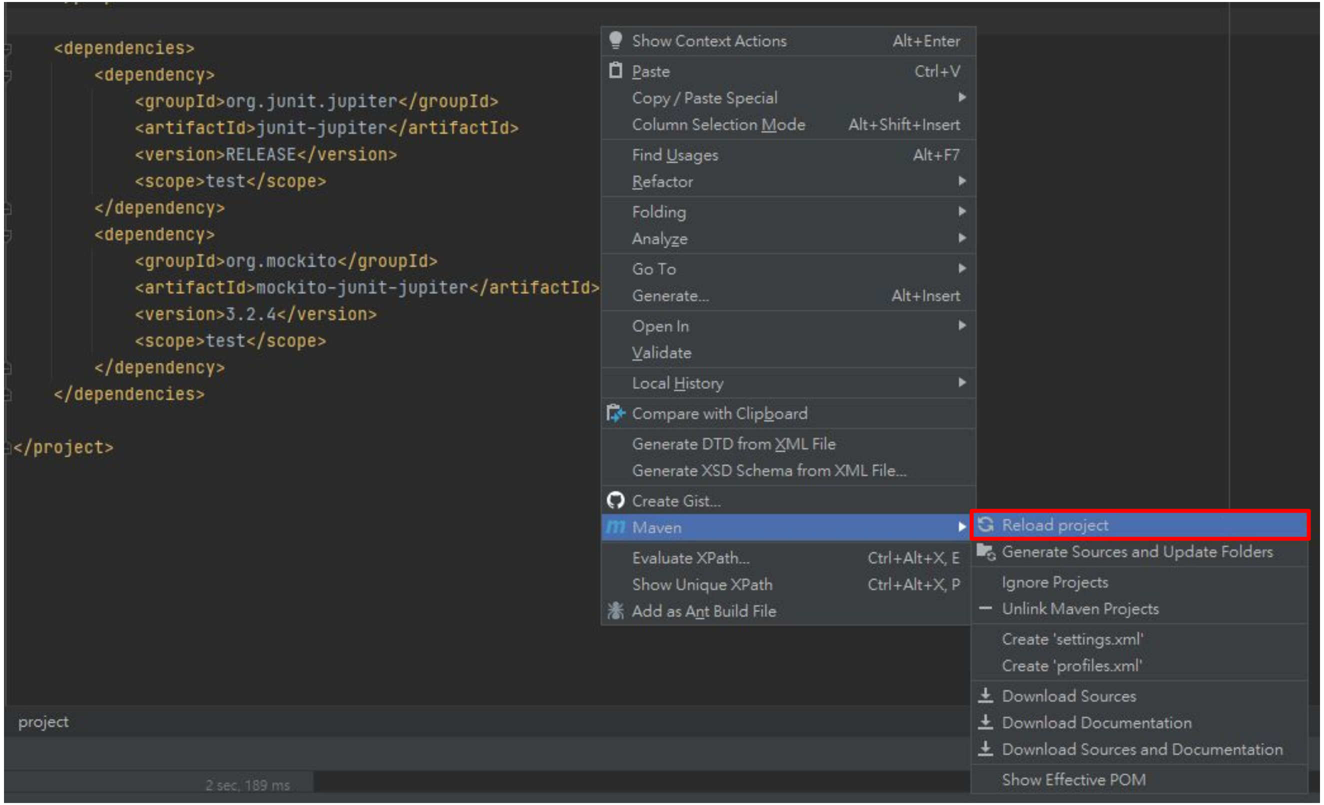
```
<dependencies>
      <dependency>
         <groupId>org.junit.jupiter</groupId>
         <artifactId>junit-jupiter</artifactId>
         <version>RELEASE</version>
         <scope>test</scope>
      </dependency>
      <dependency>
         <groupId>org.mockito</groupId>
         <artifactId>mockito-junit-jupiter</artifactId>
         <version>3.2.4</version>
         <scope>test</scope>
      </dependency>
   </dependencies>
```



pom.xml (Lab2)

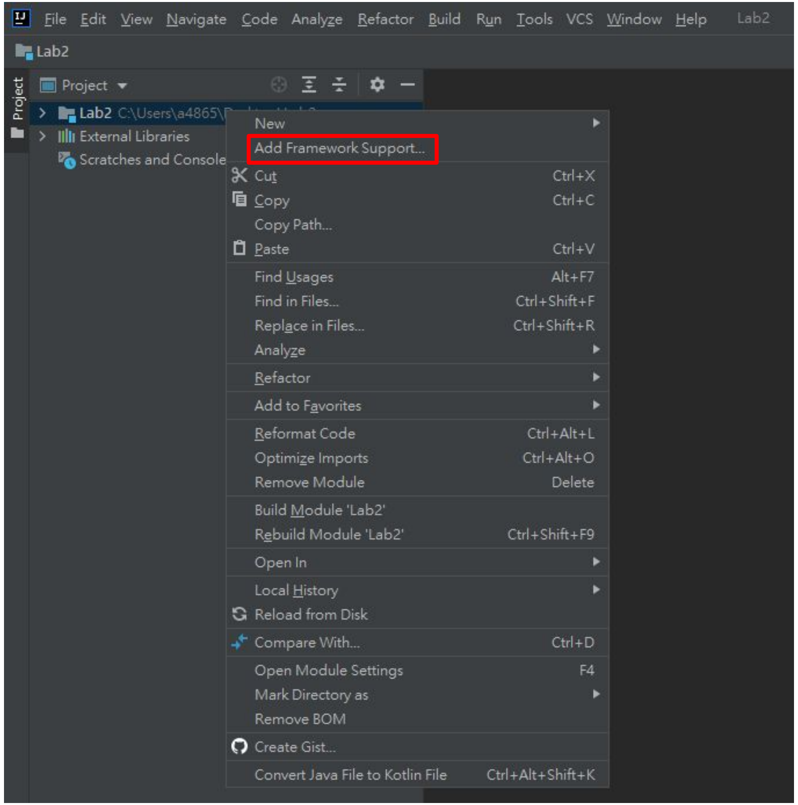
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>groupId</groupId>
    <artifactId>Lab2</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>RELEASE</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-junit-jupiter</artifactId>
            <version>3.2.4</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>
```

Import mockito - Method 1



```
    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>RELEASE</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-junit-jupiter</artifactId>
            <version>3.2.4</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>
```

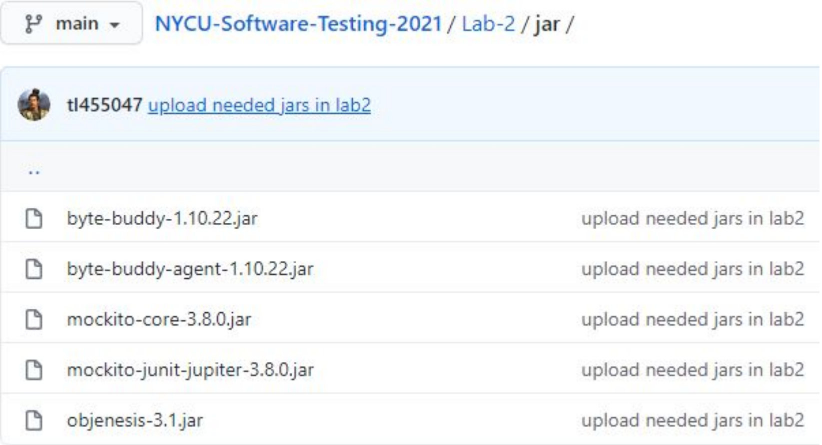
Import mockito - Method 1



Import mockito - Method 2 (JAR)

Steps for adding external jars in IntelliJ IDEA:

1. Click `File` from the toolbar
2. Project Structure (CTRL + SHIFT + ALT + S on Windows/Linux, ⌘ + ; on Mac OS X)
3. Select Modules at the left panel
4. Dependencies tab
5. '+' → JARs or directories



main ▾ NYCU-Software-Testing-2021 / Lab-2 / jar /

tl455047 upload needed jars in lab2

..

byte-buddy-1.10.22.jar	upload needed jars in lab2
byte-buddy-agent-1.10.22.jar	upload needed jars in lab2
mockito-core-3.8.0.jar	upload needed jars in lab2
mockito-junit-jupiter-3.8.0.jar	upload needed jars in lab2
objenesis-3.1.jar	upload needed jars in lab2

Reference

Reference

https://www.cwiki.us/pages/viewpage.action?pageId=47843410