Mockito :- is an open-source mocking framework, for testing java applications. Mockito is used to mock interfaces so that dummy functionality/ stub can be added to a mock interface that can be used in unit testing. It helps us to write beautiful tests with a clean and simple API.

Test Doubles:- Somebody who looks like you, but isn’t you. Looks exactly same but behave differently.

You’re testing your code and also any other external resources(connecting to database) , it takes time.

int myfunction(…)

…

Socket.connect(google.com)

mock

Socket.connect(google.com)

int myfunction(…)

…

Socket.connect(google.com)

Mocks :- replaces external interface. They are not used for checking function behaviors or return values from a function call. It is used to check the Mock function is called or not?

How many times the mock function is called? When it was called ? What parameters are called?

Features

1. Generated class
2. Doesn’t do anything
3. Behaviour verfication

Stubs

Generates Predefined outputs

Handwritten class

State verification

Return success, failure, exception reports

Fakes :- Working on the implementation

1. Quickstart New Maven project
2. Create simple class (Calculator)

**public** **class** Calculator {

**public** **int** add(**int** i, **int** j){

**return** i+j;

}

}

1. Test case

**public** **class** TestCalculator {

Calculator c=**null**;

@Before

**public** **void** setup()

{

c=**new** Calculator();

}

@Test

**public** **void** testAdd(){

*assertEquals*(5,c.add(2, 3));

}

}

1. Fake service and stub

Class Calculator{

**public** Calculator(CalculatorService service){

**this**.service=service;

}…. }

**public** **class** TestCalculator {

Calculator c=**null**;

CalculatorService service=**new** CalculatorService() {

**public** **int** add(**int** i, **int** j) {

// **TODO** Auto-generated method stub

**return** 0;

}

};

@Before

**public** **void** setup()

{

c=**new** Calculator(service);

}

@Test

**public** **void** testAdd(){

*assertEquals*(10,c.perform(2, 3));

}

}

Mock the service :- calling the mock service with the help of fake service.

<!-- https://mvnrepository.com/artifact/org.mockito/mockito-core -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>2.8.9</version>

<scope>test</scope>

</dependency>

Calculator c=null;

//remove the stub – fake object

CalculatorService service=Mockito.mock(CalculatorService.**class**);

@Test

**public** **void** testPerform(){

//testing the mock service

*when*(service.add(2, 3)).thenReturn(5);

*assertEquals*(10,c.perform(2, 3));

// service being called or not.

*verify*(service).add(2, 3);

}

// if you have multiple services for # of classes

@Mock

CalculatorService service; // NullPointerException

@Rule public MockitoRule rule=MockitoJUnit.rules();

Code Quality

The concept that includes the importance of code quality and coding standards.

Static Java source code Analyzer tools

PMD rulesets

Checkstyle :- is a development tool to help us to write java code that adheres to coding standards.

SONAR :- is an open source continuous code quality integration tool for identify bugs and the code quality in the code base.

FindBugs

a. Bad practice

1. confusing method names.

2. comaparison strings ==

b. Dodgy code

1. useless control flow

2. repeated null check

c. Malicious code

1. final members

d. Performance

1. concatenating strings using +

e. Security

1. Hardcoded constant with password

2. a prepared statement

Implementation of PMD :-

1. Google (pmd plugin for maven) copy the url link
   1. https://dl.bintray.com/pmd/pmd-eclipse-plugin/updates/
2. In eclipse – help – install new software…
   1. Click on add
   2. Give repository name and paste the copied url link

------ carelessness

String str1;

String str2;

//scanner object to scan two strings

if(str1==str2)

sop (“valid”);

else

sop(“invalid”);

equals

compareTo

What is Static Code Analyzer

1. Test the quality of our code through scanning our source code.
2. Find out the potential errors before running the application under test.

Why do we need SCA?

Enhance the code review quality.

The quality of your code is increased immediately, because the process of SCA is exactly the same as doing code review by Humans, the only difference is review by human is replaced by System.

Cost saving :- review the code quality at earlier stage rather later stage.

What is FindBugs :- find error , error in the logics…

It increases in checking or reviewing logical error.

Ensuring code quality and identifying issues without running the code that can save the huge amount of time.

FindBugs is an open source project for a static analysis of the Java (bytecode) to identify potential bugs.

The pom.xml also incudes findbugs and reporting plugins and properties.

In Maven build lifecycle – Stages

1. validate :- checks for necessary information for build
2. compile :- compiles the source code
3. test-compile :- compiles the test source code
4. test :- run unit tests
5. package :- packages compiled source code into jar / war format

3 built-in file cycles.

1. default : - follows all the stages while deployment
2. clean :- to clean the project and removes all files generated by the previous build.
3. site :- to create the project’s site documentation.

Build :- validate, compile, test, package, verify, install, deploy

1. download sonar
2. ../bin:- batch file :- Sonar server to start
   1. localhost:9000

Goals :- clean install sonar:sonar

Findbugs dependency

<dependency>

<groupId>org.codehaus.mojo</groupId>

<artifactId>findbugs-maven-plugin</artifactId>

<version>3.0.6-SNAPSHOT</version>

<type>maven-plugin</type>

</dependency>