Mockito Guide

# About

Unit testing involves testing functionality in isolation to reduce the effect of other classes or the system. Mockito is used in conjunction with JUnit to create test double objects or *mock* objects. Which proves to be very useful in *Top-Down programming* as it allows users to verify the behavior of the S*ystem Under Test* without fully implementing all the lower level dependencies.

Here, we will discuss how to import and use Mockito.

The remaining document is organized in the following manner:

1. First, a brief overview of some concepts and terms.
2. Next, an example of Mockito used in Spring.
3. Finally, an example of Mockito used in Android.

# Concepts/Terms

## Why/When to use Mockito

Mockito is meant to test classes which have lower-level dependencies without actually needing the dependencies. It allows you to *mock* the dependencies at runtime and define their behavior as needed. This ideology fits in with Top-Down programming because it ensures the higher-level code functions as intended. If your higher-level class works then you will not have to redesign the entire project if you find out later that your design has a major flaw.

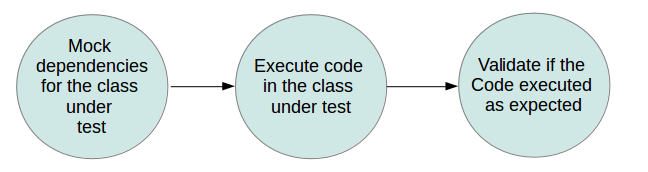
## System Under Test (SUT)

System Under Test refers to the code which is being tested by the unit tests you have written. However, most likely your code will only be testing individual classes so we can call it Class Under Test (CUT).

## Mocking

*Mocking* is the process of creating *mock* objects to abstract away the implementation details of a certain class. This allows classes to be tested without their dependencies.

## Typical Workflow



# Import Mockito into Android/Spring

## Android

// Add this dependency to your app level build.gradle

androidTestCompile "org.mockito:mockito-android:3.3.0"

## Spring Boot

<-- Add this dependency to your pom.xml -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>3.3.0</version>

</dependency>

# Mockito Examples

After you have imported Mockito, you will use the various methods and annotations provided by the framework to create mock objects. If you have been working with Spring Boot, you probably already have some experience working with annotations. The following are the key method and annotation:

* The static method “mock()”
* The annotation “@Mock”

## Install GitLab-Runner on your server.

Source - <https://docs.gitlab.com/runner/install/linux-manually.html>

Step 1: Simply download one of the binaries for your system:

sudo wget -O /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64

Step 2: Give it permissions to execute:

sudo chmod +x /usr/local/bin/gitlab-runner

Step 3: Create a GitLab CI user:

sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash

Step 4: Provide Sudo access to “GitLab Runner”

sudo usermod -aG wheel gitlab-runner

Step 5: We want to auto deploy jar to server using user name – “GitLab Runner”, we have given sudo permissions to “GitLab Runner” but we want to bypass giving password, so that our scripts are not interrupted. For that -

sudo visudo

Uncomment the line

**#%Wheel ALL= (ALL) NOPASSWD: ALL**

by removing the # in front by pressing x (lower case)

:wq to save and exit. (that is colon w q)

Important Note: DO NOT CORRUPT THIS FILE, IF YOU MAKE ANY UNWANTED CHANGES, SIMPLY EXIT BY ’:q!’ (exit without saving)

Step 6: Install and run as service:

sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-runner

sudo gitlab-runner start

Sometimes the gitlab runners are not started. You can try doing

sudo gitlab-runner run

## Install Docker on your server.

Source: https://www.tecmint.com/install-docker-and-learn-containers-in-centos-rhel-7-6/

Step 1: Install docker

yum install docker

yum install epel-release

yum install docker-io

Step 2: After, Docker package has been installed, start the daemon, check its status and enable it system wide using the below commands

systemctl start docker

systemctl status docker

systemctl enable docker

Step 3: Enable Docker service

service docker start

service docker status

chkconfig docker on

Step 4: Verify if Docker is on - “You see a hello from docker message.”

docker run hello-world

## Install JDK

Note that server has only JRE installed and not JDK, you try that by running javac option.

Source: https://www.digitalocean.com/community/tutorials/how-to-install-java-on-centos-and-fedora

To install JDK:

sudo yum install java-1.8.0-openjdk-devel

Verify JDK is installed by running javac option again.

## Install maven

Source: https://www.tecmint.com/install-apache-maven-on-centos-7/

**Step1:** cd /usr/local/src

**Step2:** sudo wget http://www-us.apache.org/dist/maven/maven-3/3.5.4/binaries/apache-maven-3.5.4-bin.tar.gz

**Step3:** sudo tar -xf apache-maven-3.5.4-bin.tar.gz

**Step4:** sudo mv apache-maven-3.5.4/ apache-maven/

**Step5:** cd /etc/profile.d/

**Step6:** sudo vim maven.sh

Type i to enter insert mode and type in:

# Apache Maven Environment Variables

# MAVEN\_HOME for Maven 1 - M2\_HOME for Maven 2

export M2\_HOME=/usr/local/src/apache-maven

export PATH=${M2\_HOME}/bin:${PATH}

**Step7:** sudo chmod +x maven.sh

**Step8:** source /etc/profile.d/maven.sh

**Step9:** mvn --version

Maven should be running now.

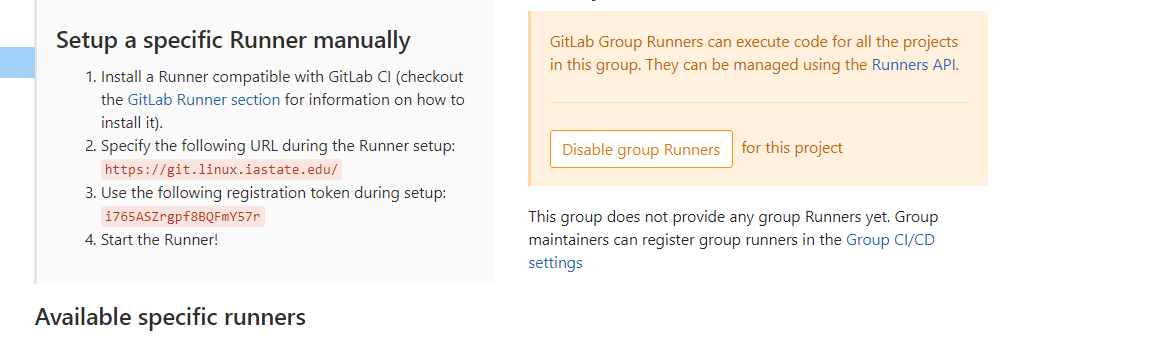
## Register GitLab Runner

We would need at least 2 GitLab runners – Frontend + Backend.

Step1: Verify if you have any Runners registered already.

Navigate to settings-> CI/CD in your GITLAB repo. Expand runner option.

You would notice either you have no specific runner, or they are not enabled.



Step2: To Create a specific runner for Frontend – Android.

* Type the command

sudo gitlab-runner register by logging into your server.

* You will be prompted to add a gitlab-ci coordinator url , enter-

[*https://git.linux.iastate.edu/*](https://git.linux.iastate.edu/)

* Next you will be prompted to enter a token, you can find the token value in your git repo by navigating to *settings-> CI/CD, expand runner, under specific runner, you will find the registration token,* copy and paste the value in your console.
* Next enter any description
* Next enter tags, *tags are important,* as they will be used later, name them appropriately, so that it’s easier for you to identify, just in case you end up creating a lot of runners – example – android\_tag.
* Finally, you will be asked to select executor, *enter docker.*
* Default Docker image enter - *alpine:latest*
* Your runner is up and running and you can verify this by again navigating to *settings->CI/CD,* expanding runner, you should see an active runner created (green for active), with the provided tags.

Step3: Create a specific runner for Backend – Spring

* Repeat the same steps as above, name your tags appropriately and for *executor enter shell* and **NOT Docker.**

# Acknowledgements

Preethi Pandian did the vast majority of work on this.

**If some parts of this document are not clear, or you feel that there could be more examples – please let us know on Piazza.**