

ECSE 429 Software Validation (Fall  
2018)

***Hands-on Tutorials***

McGill University

# Table of Contents

1. Git Recap	1
1.1. Git and GitHub	2
1.1.1. Installing Git	2
1.1.2. Logging in to GitHub	2
1.1.3. Creating a repository on GitHub	2
1.1.4. Cloning to a local repository	2
1.1.5. Git basics	3
1.1.6. Browsing commit history on GitHub	7
1.1.7. Linux commands cheat sheet:	8
1.2. Travis CI	9
2. Code Reviews	11
2.1. GitHub Code Reviews	12
2.2. Gerrit Code Reviews	14
2.2.1. Making a Change with Gerrit	16
2.2.2. Review and Manage a Change with Gerrit	17
3. Unit Testing Frameworks	20
4. Static Analysis	21
4.1. Prerequisites	22
4.2. SonarQube Static Code Analysis	23
4.2.1. Offline SonarQube Installation	23
4.2.2. SonarQube Integrated With Travis	30
4.3. SonarCloud integration for Project deliverable	36
4.4. Infer Static Analyzer	38
5. Test Generation and Code Coverage	39
5.1. EvoSuite Installation and Initial Project Setup	40
5.2. Generating Tests using EvoSuite	41
5.3. Adding the Generated Tests to the Build	42
5.4. Measuring Code Coverage with Android Studio	43
5.5. Cleaning Up the Project	44
6. Mutation Testing	45
6.1. Mutation Testing Using PIT	46
6.2. Using Gradle with PIT testing	51
6.3. Configuring PIT testing	56
6.4. Using PIT in Eclipse	57
7. Integration Testing	59
7.1. Testing Using Mocks	60
7.2. Local UI Tests	62
7.3. Android Instrumentation Tests	64

7.3.1. Project dependencies and setup .....	64
7.3.2. Creating an instrumentation test .....	65
8. Android User Interface Tests .....	69
8.1. Running Android Emulator on Travis-CI .....	73
9. Acceptance Testing .....	73
10. Model-based Testing .....	84
10.1. GraphWalker .....	85
10.2. Setting up GraphWalker .....	85
10.3. Creating a test model .....	85
10.4. Test generation .....	87

Developed by Daniel Varro, Mathieu Boucher, and Marton Bur.

- [HTML version](#)
- [PDF version](#)

Sections of the tutorial will continuously be published at this web page.

# 1. Git Recap

# 1.1. Git and GitHub

## 1.1.1. Installing Git

Install the Git version control system (VCS) from <https://git-scm.com/downloads>.

## 1.1.2. Logging in to GitHub

1. Create an account on <https://github.com/>.
2. Verify your e-mail, then log in using your credentials.

## 1.1.3. Creating a repository on GitHub

1. Visit <https://github.com/> then click on *New repository* (green button on the right).
2. Set your user as the owner of the repository.
3. Give a name for the repository (e.g., *ecse429-tutorial-1*), leave it *public*, then check *Initialize this repository with a README*. Click on *Create repository* afterwards. At this point the remote repository is ready to use.

### Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 **ecse321testuser** ▾

Repository name

**ecse321-tutorial-1** ✓

Great repository names are short and memorable. Need inspiration? How about **furry-octo-journey**.

Description (optional)



**Public**

Anyone can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

**Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾

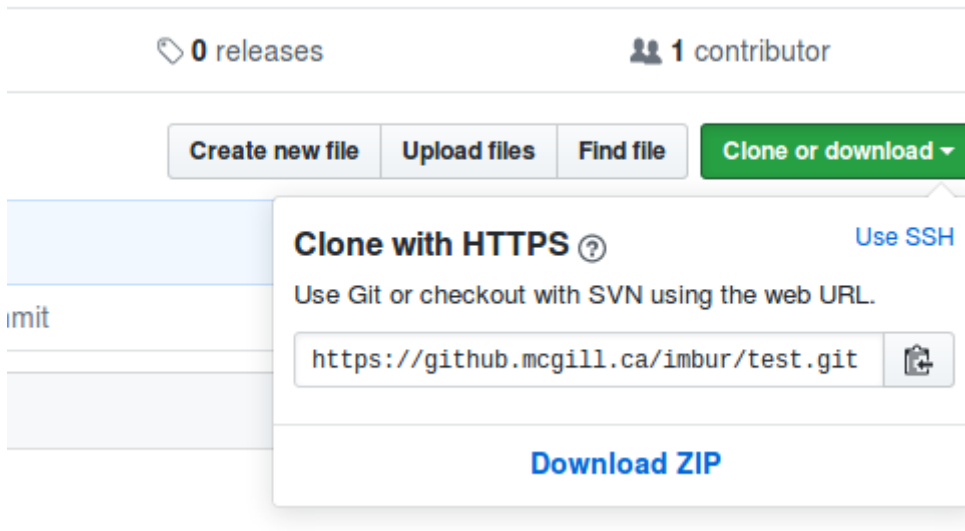


**Create repository**

## 1.1.4. Cloning to a local repository

1. Open up a terminal (Git bash on Windows).

2. Navigate to the designated target directory (it is typical to use the `git` folder within the home directory for storing Git repositories, e.g., `cd /home/username/git`).
3. Using a Git client, clone this repository to your local Git repository. First, get the repository URL (use HTTPS for now).



Then, issue `git clone https://url/of/the/repository.git`

You should get an output similar to this:

```

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university
$ git clone git@github.com:mcgill-ecse321/class-notes.git
Cloning into 'class-notes'...
remote: Counting objects: 290, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 290 (delta 0), reused 0 (delta 0)
Receiving objects: 96% (279/290), 5.68 MiB | 314 KiB/s
Receiving objects: 100% (290/290), 5.91 MiB | 313 KiB/s, done.
Resolving deltas: 100% (59/59), done.
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university
$

```

4. Verify the contents of the *working copy* of the repository by `ls -la ./repo-name`. The `.git` folder holds version information and history for the repository.

### 1.1.5. Git basics

1. Open up a terminal and configure username and email address. These are needed to identify the author of the different changes.

```

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git config --global user.name "shabbir-hussain"

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git config --global user.email shabbir.hussain@outlook.com

```

Glossary — Part 1:

- **Git** is your version control software
  - **GitHub** hosts your repositories
  - A **repository** is a collection of files and their history
  - A **commit** is a saved state of the repository
2. Enter the working directory, then check the history by issuing `git log`. Example output:

```
commit 2a0735092cea1b7f7c850a48b86e8847bf979236
Author: Shabbir Hussain <mohd.husn001@gmail.com>
Date: Thu Aug 28 15:33:09 2014 -0400

    almost finished seat checking

commit 90bfbac1c8134a87d16caf89c9ff66104f8b7fb7
Author: Shabbir Hussain <mohd.husn001@gmail.com>
Date: Thu Aug 28 14:30:07 2014 -0400

    fixed wishlist null ptr exception

commit ca4a6921005e89dace34226560921c9770a82574
Author: Shabbir Hussain <mohd.husn001@gmail.com>
Date: Thu Aug 28 11:03:19 2014 -0400

    grade checker hotfix
```

3. Adding and committing a file: use the `git add` and `git commit` commands.

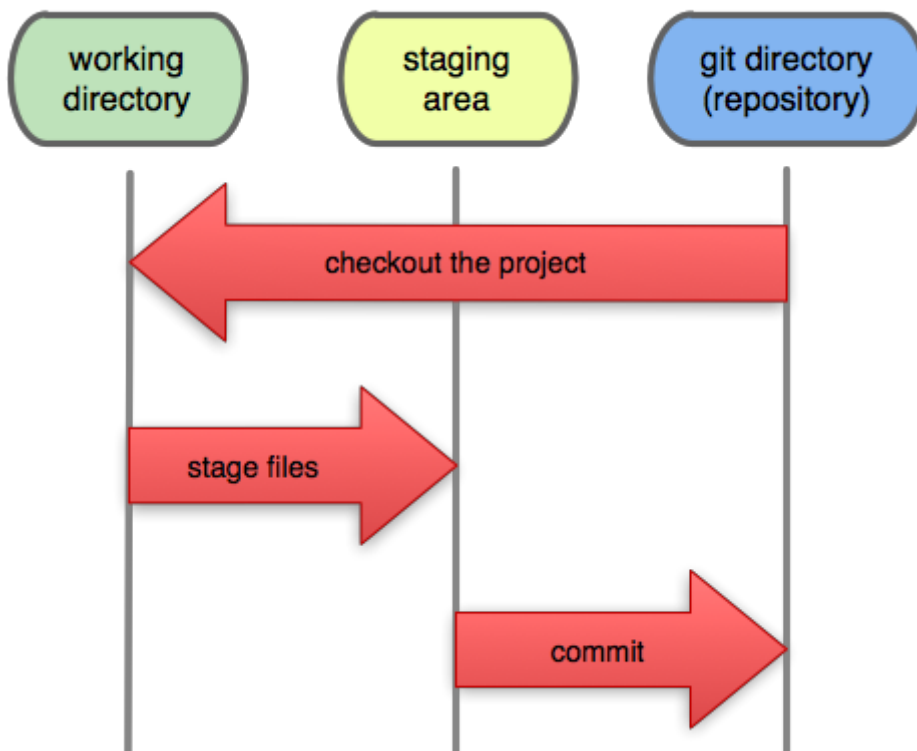
```
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ touch helloworld.java

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git add helloworld.java

Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git commit -m 'added hello world file to the project'
[master (root-commit) f4a1ddc] added hello world file to the project
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 helloworld.java
```

The effect of these commands are explained on the figure below:

## Local Operations



## Glossary — Part 2:

- **Working Directory:** files being worked on right now
- **Staging area:** files ready to be committed
- **Repository:** A collection of commits

4. Checking current status is done with `git status`.

```
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   helloworld.java
#
no changes added to commit (use "git add" and/or "git commit -a")
```

5. Staging and unstaging files: use `git reset` to remove files from the staging area.

```
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git add .
```

```
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   helloworld.class
#       modified:  helloworld.java
#
```

```
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git reset helloworld.class
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:  helloworld.java
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       helloworld.class
```

**Important:** only staged files will be committed.



```
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git diff helloworld.java
diff --git a/helloworld.java b/helloworld.java
index 28fe9d9..de3a7d2 100644
--- a/helloworld.java
+++ b/helloworld.java
@@ -1,6 +1,6 @@
 public class helloworld{

     public static void main(String[] args){
-         System.out.println("Hello World");
+         System.out.println("Hello World")
     }
 }
```

6. To display detailed changes in unstaged files use `git diff`, while use `git diff --staged` to show changes within files staged for commit.

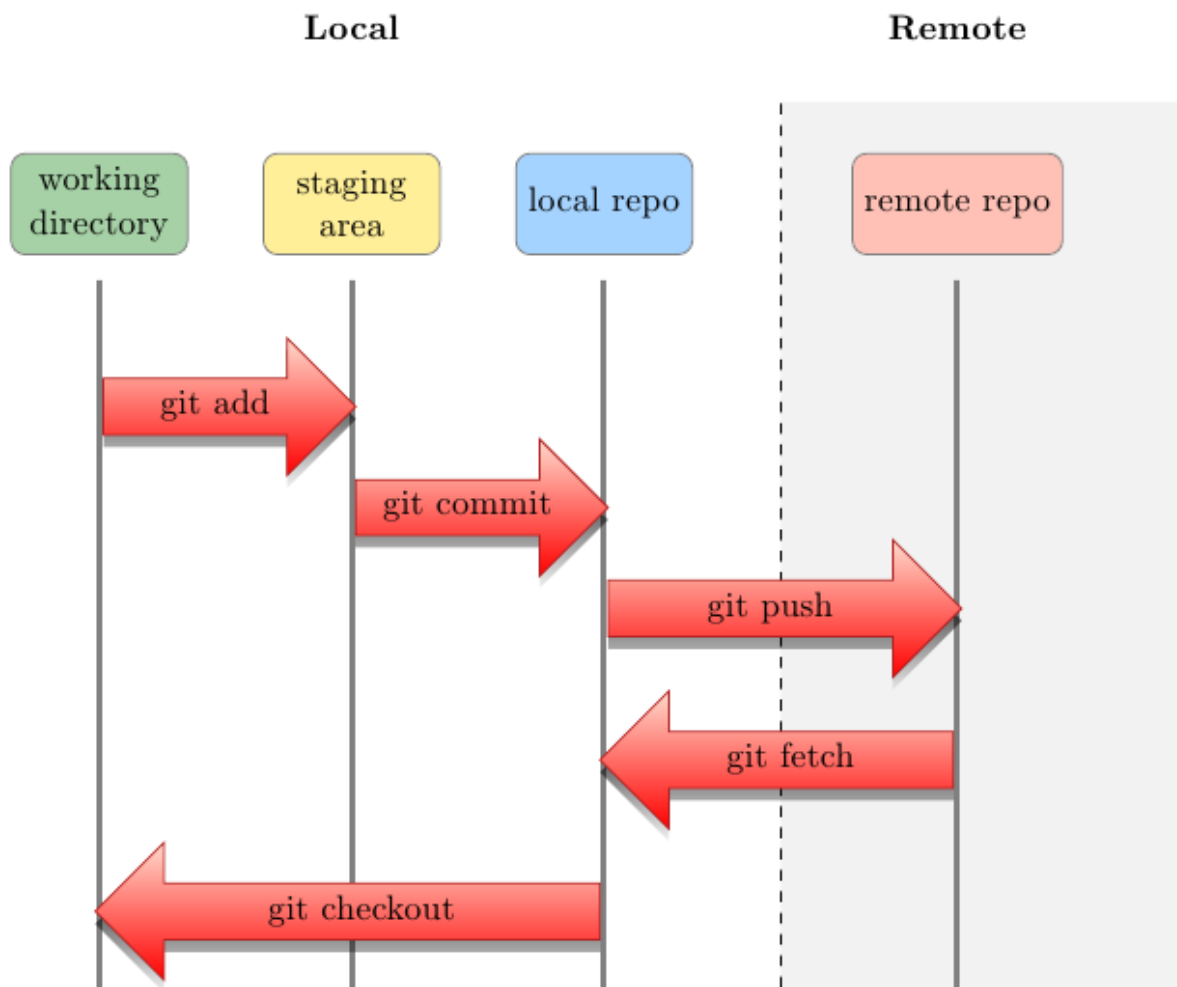
```
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git diff helloworld.java
diff --git a/helloworld.java b/helloworld.java
index 28fe9d9..de3a7d2 100644
--- a/helloworld.java
+++ b/helloworld.java
@@ -1,6 +1,6 @@
 public class helloworld{

     public static void main(String[] args){
-         System.out.println("Hello World");
+         System.out.println("Hello World")
     }
 }
```

7. Reverting to a previous version is done using `git checkout`.

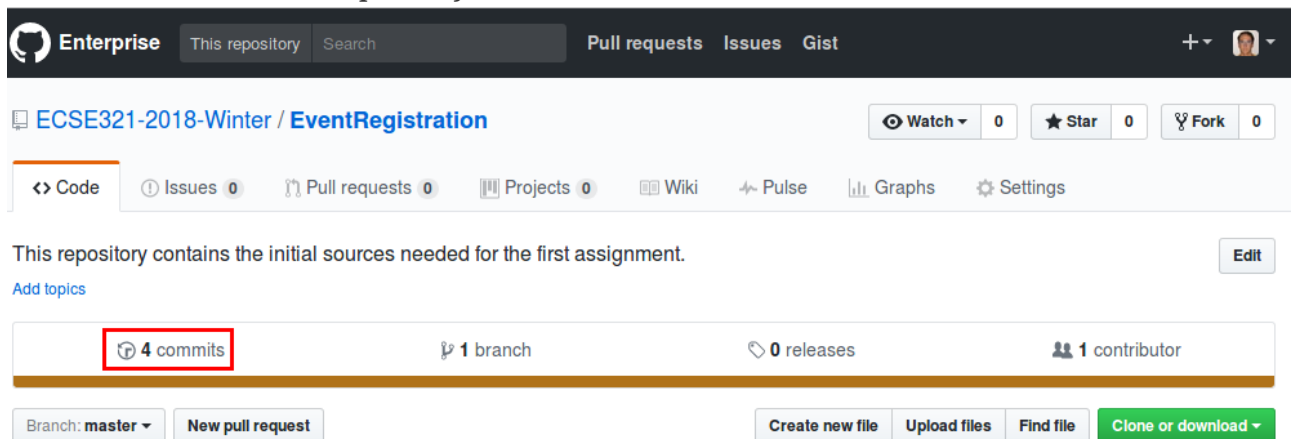
```
Shabbir@SHABBIR-LAPTOP ~/Documents/code/university/myfirstrepo (master)
$ git checkout helloworld.java
```

8. The commands `git pull` (or the `git fetch` + `git rebase` combination) and `git push` are used to synchronize local and remote repositories.



### 1.1.6. Browsing commit history on GitHub

1. You can browse pushed commits in the remote repository online using GitHub. You can select the *commits* menu for a repository.



To get a link for a specific commit, click on the button with the first few characters of the hash of the commit.

The screenshot shows the GitHub interface for the repository 'ECSE321-2018-Winter / EventRegistration'. At the top, there are navigation links for 'Pull requests', 'Issues', and 'Gist'. Below the repository name, there are buttons for 'Watch', 'Star', and 'Fork'. The main content area shows a list of commits on the 'master' branch. The commit 'Adding initial Java Spring project seed' is highlighted with a red box around its hash '58c992b'.

### 1.1.7. Linux commands cheat sheet:

- **cd**: change/navigate directory
- **ls**: list contents of a directory
- **ls -la**: list all contents of a directory in long listing format
- **touch**: create a file
- **cp**: copy a file
- **mv**: move a file
- **rm**: remove a file
- **mkdir**: create a directory
- **cp -r**: copy a directory recursively with its contents
- **rmdir**: remove a directory
- **rm -rf**: force to recursively delete a directory (or file) and all its contents
- **cat**: concatenate and print contents of files
- **nano**: an easy-to-use text editor

The source for most of the images in the Git documentation: <https://github.com/shabbir-hussain/ecse321tutorials/blob/master/01-githubTutorial1.pptx>

## 1.2. Travis CI

1. Go to <https://travis-ci.com/>, click on Sign up with GitHub.
2. Click on the green authorize button at the bottom of the page.
3. Activate Travis-CI on your GitHub account
4. Select the repositories you want to build with Travis (make sure to include your repository that you created for this tutorial). You can modify this setting anytime later as well.
5. In your working copy of your repository, create a small Maven project with Spring Boot.
  - Make sure you have Maven 3 installed (`mvn --version`).
  - Create the `src/main/java/Main.java` file with the content

```
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.SpringApplication;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@Configuration
@EnableAutoConfiguration
@RestController
public class Main {
    public static void main(String[] args) {
        SpringApplication.run(Main.class, args);
    }

    @RequestMapping("/")
    public String greeting(){
        return "Hello world!";
    }
}
```

- Create a `pom.xml` in the root of the working copy.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://maven.apache.org/POM/4.0.0"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>ca.mcgill.ecse429</groupId>
    <artifactId>tutorial1</artifactId>
    <version>1.0</version>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
```

```

    <version>2.0.4.RELEASE</version>
    <relativePath/>
  </parent>

  <name>tutorial1</name>

  <properties>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <executions>
          <execution>
            <goals>
              <goal>build-info</goal>
            </goals>
            <configuration>
              <additionalProperties>
                <java.source>${maven.compiler.source}</java.source>
                <java.target>${maven.compiler.target}</java.target>
              </additionalProperties>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>

```

- Add a `.gitignore` to ignore generated resources by Git:

```

*.class
target/

```

- Make sure your application is compiling by running `mvn clean install`

## 6. Create a file called `.travis.yml`:

```
language: java
script:
- mvn clean install
```

7. Commit and push your work. If everything is set up correctly, the build should trigger and Travis should run your build using Maven.

## 2. Code Reviews

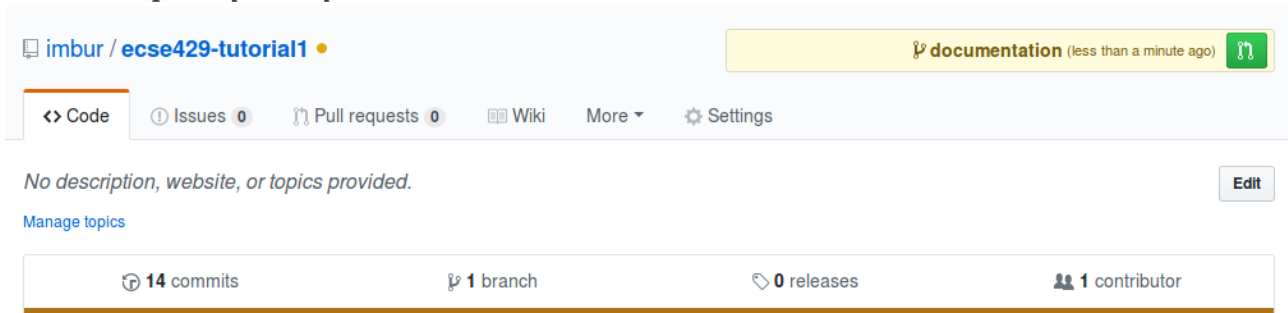
## 2.1. GitHub Code Reviews

1. Navigate to the folder for the working copy of your *ecse429-tutorial1* from the previous tutorial.
2. Create & checkout a new branch named called `documentation`. (`git branch documentation && git checkout documentation`)
3. Edit `README.md`.

```
# ecse429-tutorial1
```

```
A default java project generated by Maven.
```

1. Commit your changes and push this `documentation` branch. (`git add . && git commit -m "Adding brief project description" && git push`)
2. Open the main page for the repository on Github. Click on the green button in the upper right corner to open a *pull request*.



3. Study the possible settings, then create the pull request.

# Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: master + compare: documentation ✓ Able to merge. These branches can be automatically merged.

**Adding brief project description**

Write Preview

Leave a comment

Create pull request

**Reviewers** ⚙️

No reviews

---

**Assignees** ⚙️

No one—assign yourself

---

**Labels** ⚙️

None yet

---

**Projects** ⚙️

None yet

---

**Milestone** ⚙️

No milestone

1 commit 1 file changed 0 commit comments 1 contributor

Commits on Sep 15, 2018

imbur Adding brief project description Verified ✓ 93742ee

4. On the next page, click on *Files changed*. Place a comment in the updated file and start a review.

imbur / ecse429-tutorial1 ✓

Code Issues 0 Pull requests 1 Wiki More Settings

## Adding brief project description #2 Edit

Open imbur wants to merge 1 commit into master from documentation

Conversation 0 Commits 1 Checks 2 Files changed 1

all commits Jump to... +4 -1

Diff settings No Whitespace Review changes

5 README.md Show comments <> Copy path View

... -1 +1,4

1 - # ecse429-tutorial1

1 + # ecse429-tutorial1

2 +

3 + A default java project generated by Maven.

Write Preview

Great executive summary!

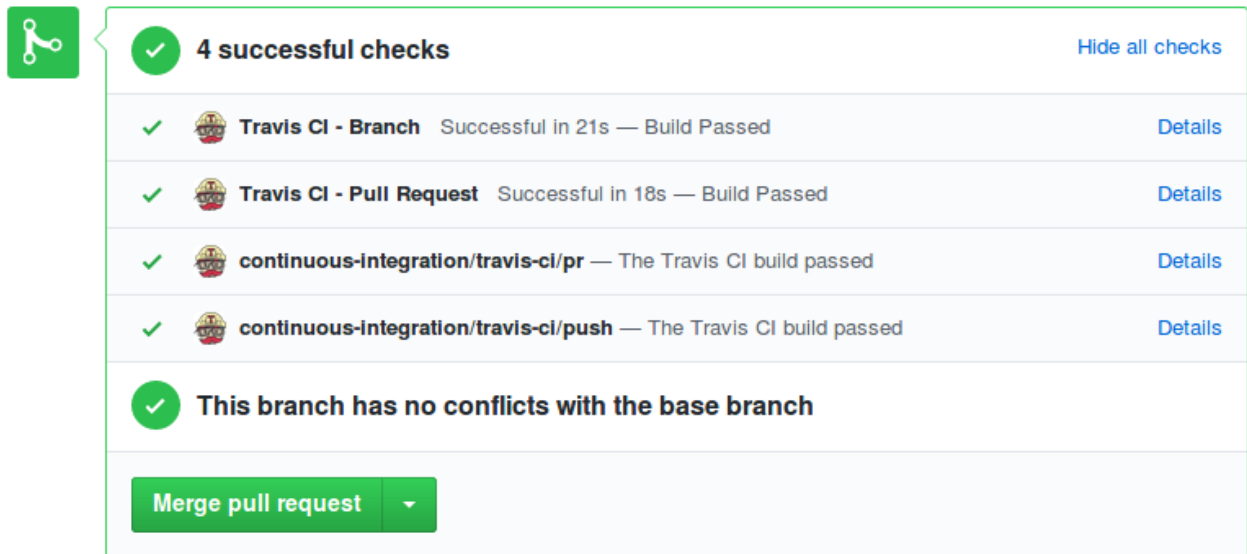
Start a review

4 +



- Normally, the owner and the reviewer are two different participants, and GitHub offers more options when completing a review: *approve changes* or *request changes*. This time simply click on *Finish review* and *Comment*.
- Merge the pull request. Delete the branch afterwards.

Add more commits by pushing to the **documentation** branch on **imbur/ecse429-tutorial1**.



The screenshot shows a GitHub pull request interface. At the top, there is a green checkmark icon and the text "4 successful checks" with a "Hide all checks" link. Below this, there are four check items, each with a green checkmark, a Travis CI logo, and a "Details" link:

- Travis CI - Branch: Successful in 21s — Build Passed
- Travis CI - Pull Request: Successful in 18s — Build Passed
- continuous-integration/travis-ci/pr — The Travis CI build passed
- continuous-integration/travis-ci/push — The Travis CI build passed

Below the checks, there is a green checkmark icon and the text "This branch has no conflicts with the base branch". At the bottom, there is a green button labeled "Merge pull request" with a dropdown arrow.

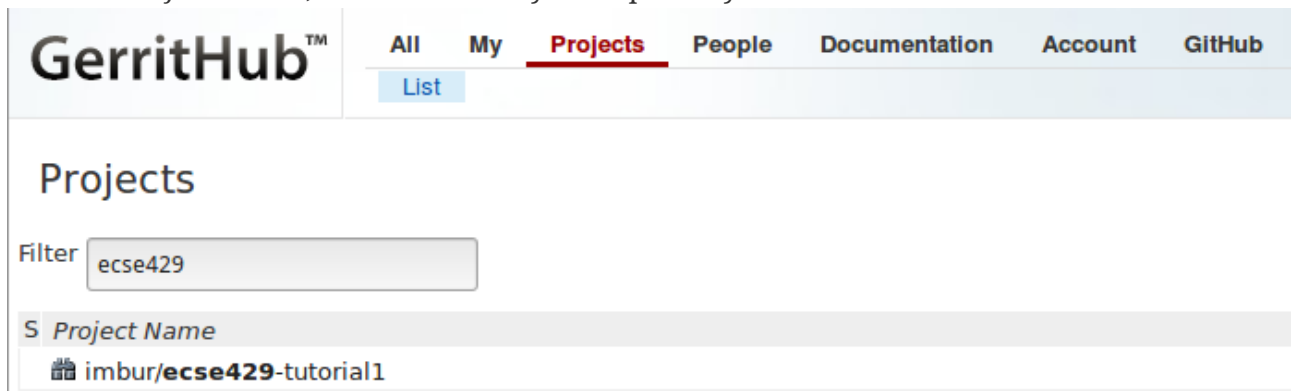
## 2.2. Gerrit Code Reviews

- Go to <http://gerrithub.io/>, click on *First time Sign In*.
- Authenticate with GitHub, then select the repositories you want to replicate with Gerrit. It is best to select the the public [\[your GitHub user\]/ecse429-tutorial1](#) repository created previously to follow the steps of this tutorial.

### NOTE

You can select repositories to replicate anytime by navigating to *gerrithub.io > Open Gerrit Code Review > GitHub (in the top menubar) > Repositories*

- Create a GerritHub password for HTTP access. Go to *your profile (upper right corner) > Settings > HTTP Password*, then click on *Generate password*. Make sure to save it somewhere, otherwise you need to generate it again next time.
- Click on *Projects > List*, then search for your repository.



The screenshot shows the GerritHub interface. At the top, there is the GerritHub logo and a navigation bar with tabs: "All", "My", "Projects" (highlighted), "People", "Documentation", "Account", and "GitHub". Below the navigation bar, there is a "List" button. The main content area is titled "Projects" and has a "Filter" input field containing "ecse429". Below the filter, there is a table with a header "Project Name" and one row containing the repository name "imbur/ecse429-tutorial1".

- Select your project named after your repository, then set the settings below and click on *Save Changes*.

## Project imbur/ecse429-tutorial1

Clone Clone with commit-msg hook anonymous http http ssh  
 git clone ssh://imbur@review.gerrithub.io:29418/imbur/ecse429-tutor...

### Description

Example project on demonstrating how to use Gerrit CR

### Project Options

State: Active

Submit Type: Merge if Necessary

Allow content merges: INHERIT (true)

Create a new change for every commit not in the target branch: INHERIT (false)

Require Change-Id in commit message: TRUE

Reject implicit merges when changes are pushed for review: INHERIT (false)

Set all new changes private by default: INHERIT (false)

Set all new changes work-in-progress by default: INHERIT (false)

Enable adding unregistered users as reviewers and CCs on changes: INHERIT (false)

Match authored date with committer date upon submit: INHERIT (false)

Maximum Git object size limit:

Contributor Agreements

Require Signed-off-by in commit message: TRUE

Save Changes

### Project Commands

Commands: Delete Project Create Change Edit Config

6. Copy the command from the `git clone` command shown above the *Description* text box after changing to *Clone with commit-msg hook* and *http*.

#### NOTE

Make sure you are issuing the `git clone` command in a folder that *is not part of any git repository working copy* in your file system. (In other words, you should do it in your `~/git` folder or the like.)

Clone Clone with commit-msg hook anonymous http http ssh  
 git clone https://imbur@review.gerrithub.io/a/imbur/ecse429-tutoria...

#### NOTE

Optionally, if you have SSH key installed to GitHub, you can import it to GerritHub and use that for authentication.

7. Paste the command to the terminal. Enter your generated password when prompted.

### 2.2.1. Making a Change with Gerrit

1. Navigate to the repository. Edit the `.travis.yml` file:

```
language: java
install: true
script:
  - cd tutorial1
  - mvn clean install
```

2. Add, commit and push the changes. (One-liner: `git add . && git commit -m "Making Travis skip the install step" && git push`)
3. Observe the created commit message with `git log`. Observe the changes synced to GitHub, too.
4. Create a new change to the code, remove the recently added `install: true` line from `.travis.yml` and create a new commit. Do not push it yet!
5. Issue `git push origin HEAD:refs/for/master`. The outputs should show that a new Gerrit change is created.

```
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1.01 KiB | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1)
remote: Processing changes: new: 1, done
remote:
remote: New Changes:
remote:  https://review.gerrithub.io/#/c/imbur/ecse429-tutorial1/+425671 Removing
no-op install step from travis conf
remote:
To https://imbur@review.gerrithub.io/a/imbur/ecse429-tutorial1
* [new branch]      HEAD -> refs/for/master
```

6. Navigate to your GerritHub dashboard. Observe the changes there.

## My Reviews

*Subject*

**Work in progress**  
(None)

**Outgoing reviews**  
★ Removing no-op install step from travis conf

### NOTE

GitHub does not sync Gerrit changes, since they are pushed to a gerrit-specific branch (`refs/for/[branchname]`).

## 2.2.2. Review and Manage a Change with Gerrit

1. Click on the change and explore the options on the web interface.

The screenshot shows the Gerrit web interface for a change. At the top, there are navigation tabs: "All", "My" (selected), "Projects", "People", "Documentation", "Account", and "GitHub". Below these are sub-tabs: "Changes", "Draft Comments", "Edits", "Watched Changes", "Starred Changes", and "Groups".

The main content area shows a change titled "Change 425671 - Needs Verified Label" with the description "Removing no-op install step from travis conf". The change ID is "Iee350d637f86282c46c55936313026277e5a4179".

On the right side, there is a metadata panel with the following information:
 

- Owner: Marton Bur
- Assignee: (empty)
- Reviewers: (empty)
- Project: imbur/ecse429-tutorial1
- Branch: master
- Topic: (empty)
- Strategy: Merge if Necessary
- Uploaded: 46 minutes ago
- Hashtags: (empty)

 Below this panel are several action buttons: "Cherry Pick", "Move Change", "Rebase", "Abandon", "Follow-Up", "Mark Private", "Mark Reviewed", and "WIP".

Below the metadata panel, there is a section for "Code-Review" which is "Verified".

At the bottom, there is a "Files" section showing a diff against the "Base" branch. The diff includes a "Commit Message" and a ".travis.yml" file. The ".travis.yml" file has a red bar indicating a change, with a "Comments Size" of 1. Below the diff is a "History" section showing "Marton Bur" uploaded patch set 1 at 4:25 PM.

2. Comment on the `.travis.yml` file within the change suggesting that JDK8 should be used with the project.

GerritHub™ All **My** Projects People Documentation Account GitHub  
 Changes Draft Comments Edits Watched Changes Starred Changes

imbur/ecse429-tutorial1 / .travis.yml Patch Set Base 1 (GitHub)

1 language: java  
 2 install: true

**Draft**

Removing this no-op line is a good idea, however, as a prerequisite for the build, jdk 8 should be added as requirement.

Save Discard

3 script:  
 4 - cd tutorial1  
 5 - mvn clean install  
 6

- The comment will remain a draft up to the point until you submit your review. Click on the *Reply...* button on the overview page for the change.

GerritHub™ All **My** Projects People Documentation Account GitHub  
 Changes Draft Comments Edits Watched Changes Starred Changes Groups

Change 425671 - Needs Verified Label  
 Removing no-op install step from travis conf  
 Change-Id: Iee350d637f86282c46c55936313026277e5a4179

Reply...

-2 -1 0 +1 +2  
 Code-Review     Do not submit  
 Verified    No score

.travis.yml  
**Base, Line 2:** Removing this no-op line is a good idea, however, as a prerequisite for the build, jdk 8 should be added as requirement.

Post Cancel

Author Marton Bur <marton.bur@gmail.com> Sep 15, 2018 4:24 PM  
 Committer Marton Bur <marton.bur@gmail.com> Sep 15, 2018 4:24 PM  
 Commit f1671d72067dfc823babcdf7bed2d869b4ceaa88 (GitHub)  
 Parent(s) a4fd55564d8a451e5437fd95890e67ca9fb3c47a (GitHub)  
 Change-Id Iee350d637f86282c46c55936313026277e5a4179

**Files** Open All Diff against: Base Edit

File Path	Comments Size
Commit Message	
.travis.yml	drafts: 1 1 +0, -1

**History** Expand All Hide tagged comments

Marton Bur Uploaded patch set 1. 4:25 PM

- Implement the modifications.

```
language: java
jdk:
  - oraclejdk8
script:
  - cd tutorial1
  - mvn clean install
```

- Append these modifications to the change

```

> git add .
> git commit --amend
> git push origin HEAD:refs/for/master

```

## CAUTION

It is very important to add that in git amending a pushed commit is a history-rewriting operation, that is in most cases to be avoided. However, Gerrit follows these changes by amending a previous commit that holds the previous version of the change.

## 6. Observe the new patch set online.

The screenshot shows the GerritHub interface for a change set. The change is titled "Change 425671 - Needs Verified Label" and is described as "Removing no-op install step from travis conf". The author is Marton Bur, and the commit message is "Removing no-op install step from travis conf". The change is currently in the "Needs Verified Label" state. The interface shows the change details, including the author, committer, commit hash, and parent(s). It also displays the "Files" section with a table of file paths and their comment sizes. The "History" section shows the sequence of patch sets uploaded by Marton Bur.

File Path	Comments Size
Commit Message	
.travis.yml	3
	+2, -1

History	Time
Marton Bur Uploaded patch set 1.	4:25 PM
Marton Bur Patch Set 1: Code-Review-1 (1 comment)	5:23 PM
Marton Bur Uploaded patch set 2.	5:29 PM

## 7. Approve and verify the change.

The screenshot shows the Gerrit code review interface. The "Code-Review" section has a radio button selected for "Looks good to me, approved". The "Verified" section has a radio button selected for "Verified". The interface also shows a "Post" button and a "Cancel" button. Below the buttons, there are options to "MARK PRIVATE", "MARK REVIEWED", and "WIP".

## 8. Submit the changes to the master branch.

Change **425671** - Ready to Submit Reply...

Removing no-op install step from travis conf  
 Change-Id: Iee350d637f86282c46c55936313026277e5a4179

Owner Marton Bur

Assignee

Reviewers Marton Bur x

Project [imbur/ecse429-tutorial1](#)

Branch [master](#)

Topic

Strategy Merge if Necessary

Updated 6 seconds ago

Hashtags

Cherry Pick

Move Change

Rebase

Abandon

Follow-Up

Mark Private

Mark Reviewed

WIP

Submit

Submit patch set 2 into master

---

Author [Marton Bur <marton.bur@gmail.com>](#)

Committer [Marton Bur <marton.bur@gmail.com>](#)

Commit [51dc62e9822eb038e1cfa76f88ab9034c65b8e44](#)

Parent(s) [a4fd55564d8a451e5437fd95890e67ca9fb3c47a](#)

Change-Id [Iee350d637f86282c46c55936313026277e5a4179](#)

Sep 15, 2018 4:24 PM

Sep 15, 2018 5:29 PM

[\(GitHub\)](#)

[\(GitHub\)](#)

Code-Review +2 Marton Bur x

Verified +1 Marton Bur x

9. Soon the commit becomes visible in GitHub as well. Make sure to pull afterwards to update your local branches.

### 3. Unit Testing Frameworks

This material is included in two public tutorials:

1. AssertJ: <http://www.vogella.com/tutorials/AssertJ/article.html> (Sections 1-6)
2. JUnit5: <http://www.vogella.com/tutorials/JUnit/article.html#junit5> (Sections 10-11)

## 4. Static Analysis



## 4.1. Prerequisites

This tutorial assumes that you have `maven` version 3.x.x installed on your computer. To create a new, default, maven project, simply type the command `mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DgroupId=com.mcgill.ca.tutorial3 -DartifactId=tutorial3` in the folder of your choice.

Next we will update the JUnit dependencies. Locate your `pom.xml` file that was created after the above command. The default generated `pom.xml` should look something like:

```
<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/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mcgill.ca</groupId>
  <artifactId>tutorial3</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>tutorial-3</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Change the version number for JUnit from 3.8.1 to 4.12 and remove the `<scope>test</scope>` tag.

Change the `AppTest.java` class definition to a single empty test:

```
package com.mcgill.ca.tutorial3;

import org.junit.Test;

public class AppTest
{
    @Test
    public void emptyTest() {
    }
}
```

Finally, to make sure everything is working correctly, run the command `mvn clean install` and check that the build succeeded.

No git repository is created for this project. If you already have a repository containing this project and wish to use that one, ignore the steps having the *(Optional)* tag

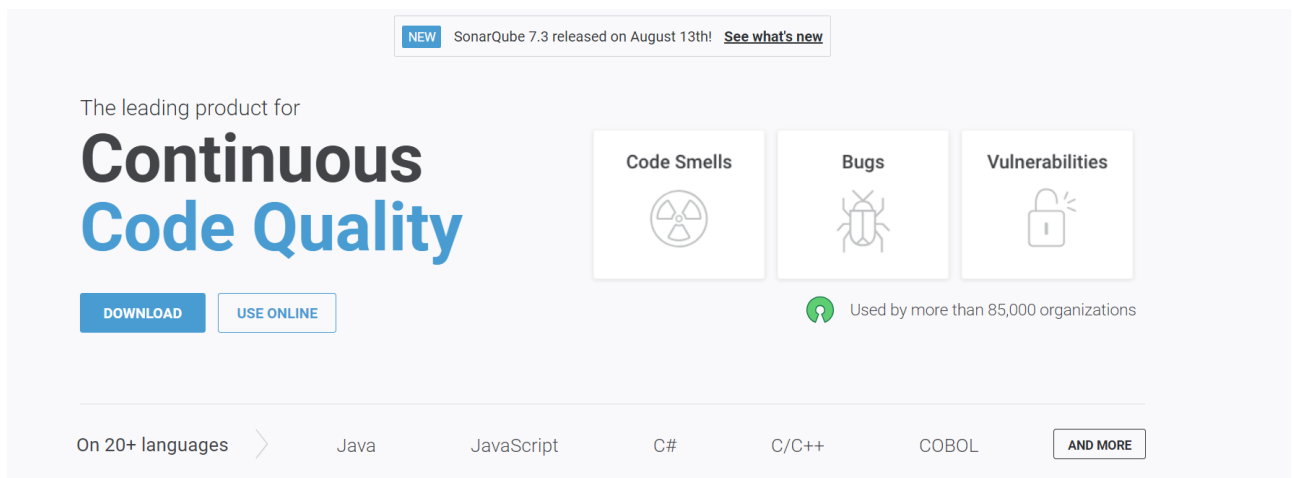
This default project will be used in the next sections.

## 4.2. SonarQube Static Code Analysis

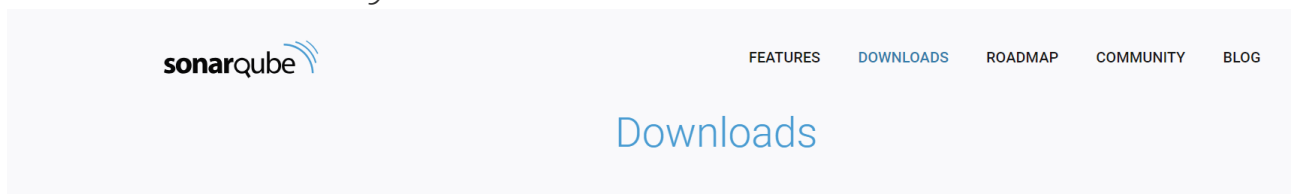
This section looks at how to use SonarQube, both integrated with Travis and as a standalone setup.

### 4.2.1. Offline SonarQube Installation

1. Navigate to the folder of the working copy of your repository used during the previous tutorial. If you no longer keep a working copy, clone it again.
2. Go to the SonarQube main page (<https://www.sonarqube.org/>) and click on the blue *Download* button.



3. Click on the blue *Community Edition 7.3* button



4. Extract the sonarqube-7.3.zip to the location of your choice.  
For this tutorial, we unzipped it in C:\
5. Start the SonarQube Server.  
Assuming you are in the root folder of the sonarqube directory, that is `C:\sonarqube-7.3` this example, you run either:  
`.\bin\[OS]\StartSonar.bat` for windows or `./bin/[OS]/sonar.sh console` for linux or mac

environments.

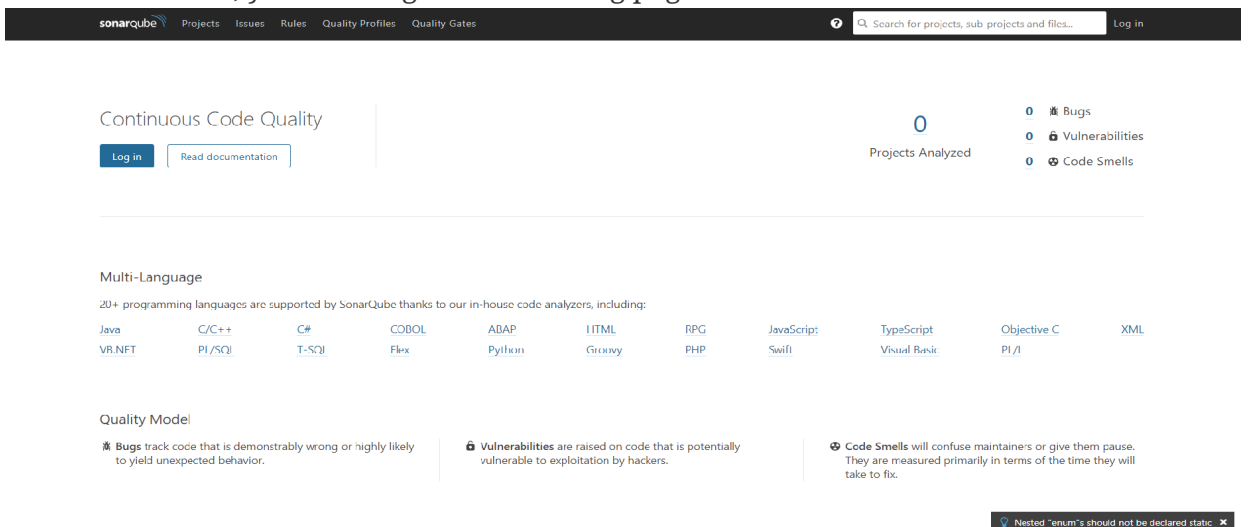
The OSs that are included are:

- windows-x86-32
- windows-x86-64
- macosx-universal-64
- linux-x86-32
- linux-x86-64

In this example, we have a Windows machine, so we use `.\bin\windows-x86-64\StartSonar.bat` as a command. The server should boot and when it is finally ready, you should get an output similar to this

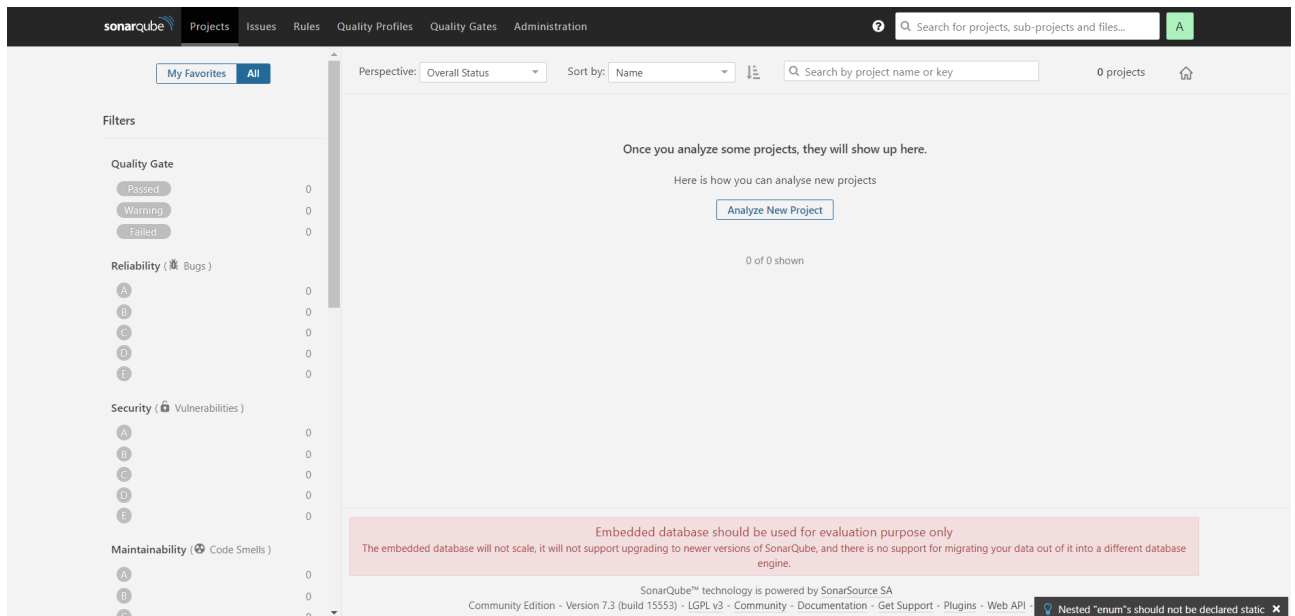
```
ca SonarQube
jvm 1 | 2018.09.20 23:40:35 INFO app[][o.s.a.AppFileSystem] Cleaning or creating temp directory C:\sonarqube-7.3\temp
jvm 1 | 2018.09.20 23:40:35 INFO app[][o.s.a.es.EsSettings] Elasticsearch listening on /127.0.0.1:9001
jvm 1 | 2018.09.20 23:40:35 INFO app[][o.s.a.p.ProcessLauncherImpl] Launch process[[key='es', ipcIndex=1, logFileNa
ePrefix=es]] from [C:\sonarqube-7.3\elasticsearch]: C:\Program Files\Java\jdk1.8.0_161\jre\bin\java -XX:+UseConcMarkSwe
pGC -XX:CMSInitiatingOccupancyFraction=75 -XX:+UseCMSInitiatingOccupancyOnly -XX:+AlwaysPreTouch -server -Xss1m -Djava.
wt.headless=true -Dfile.encoding=UTF-8 -Djna.nosys=true -Djdk.io.permissionsUseCanonicalPath=true -Dio.netty.noUnsafe=t
ue -Dio.netty.noKeySetOptimization=true -Dio.netty.recycler.maxCapacityPerThread=0 -Dlog4j.shutdownHookEnabled=false -D
log4j2.disable.jmx=true -Dlog4j.skipJansi=true -Xms512m -Xmx512m -XX:+HeapDumpOnOutOfMemoryError -Delasticsearch -Des.pa
h.home=C:\sonarqube-7.3\elasticsearch -cp lib/* org.elasticsearch.bootstrap.Elasticsearch -Epath.conf=C:\sonarqube-7.3\
temp\conf\es
jvm 1 | 2018.09.20 23:40:35 INFO app[][o.s.a.SchedulerImpl] Waiting for Elasticsearch to be up and running
jvm 1 | 2018.09.20 23:40:35 INFO app[][o.e.p.PluginsService] no modules loaded
jvm 1 | 2018.09.20 23:40:35 INFO app[][o.e.p.PluginsService] loaded plugin [org.elasticsearch.transport.Netty4Plugin]
jvm 1 | 2018.09.20 23:40:43 INFO app[][o.s.a.SchedulerImpl] Process[es] is up
jvm 1 | 2018.09.20 23:40:43 INFO app[][o.s.a.p.ProcessLauncherImpl] Launch process[[key='web', ipcIndex=2, logFileNa
mePrefix=web]] from [C:\sonarqube-7.3]: C:\Program Files\Java\jdk1.8.0_161\jre\bin\java -Djava.awt.headless=true -Dfile
encoding=UTF-8 -Djava.io.tmpdir=C:\sonarqube-7.3\temp -Xmx512m -Xms128m -XX:+HeapDumpOnOutOfMemoryError -cp ./lib/commo
n/*;C:\sonarqube-7.3\lib\jdbc\h2\h2-1.3.176.jar org.sonar.server.app.WebServer C:\sonarqube-7.3\temp\sq-process211827281
179159996properties
jvm 1 | 2018.09.20 23:40:54 INFO app[][o.s.a.SchedulerImpl] Process[web] is up
jvm 1 | 2018.09.20 23:40:54 INFO app[][o.s.a.p.ProcessLauncherImpl] Launch process[[key='ce', ipcIndex=3, logFileNa
ePrefix=ce]] from [C:\sonarqube-7.3]: C:\Program Files\Java\jdk1.8.0_161\jre\bin\java -Djava.awt.headless=true -Dfile.e
coding=UTF-8 -Djava.io.tmpdir=C:\sonarqube-7.3\temp -Xmx512m -Xms128m -XX:+HeapDumpOnOutOfMemoryError -cp ./lib/common/
;C:\sonarqube-7.3\lib\jdbc\h2\h2-1.3.176.jar org.sonar.ce.app.CeServer C:\sonarqube-7.3\temp\sq-process4243881039700642
59properties
jvm 1 | 2018.09.20 23:40:58 INFO app[][o.s.a.SchedulerImpl] Process[ce] is up
jvm 1 | 2018.09.20 23:40:58 INFO app[][o.s.a.SchedulerImpl] SonarQube is up
```

SonarQube created a default webserver at localhost:9000. If you type the latter in your Internet browser, you should get the following page



## 6. Log in the server

The default account created for your page is admin:admin



## 7. Create Basic Code Smells in the Project

Locate the file `App.java` in your maven project and change the class definition to the following:

```
package ca.mcgill.ecse429;

public class App
{
    private int x;

    public static void main( String[] args )
    {
        String s = null;
        System.out.println(s.length());
    }
}
```

## 8 Edit the `pom.xml`

Add the Sonar Maven plugin and its configuration to the project (e.g., after the `</dependencies>` closing tag)

```

...
<build>
  <plugins>
    <plugin>
      <groupId>org.sonarsource.scanner.maven</groupId>
      <artifactId>sonar-maven-plugin</artifactId>
      <version>3.5.0.1254</version>
    </plugin>
  </plugins>
</build>

<profiles>
  <profile>
    <id>sonar</id>
    <activation>
      <activeByDefault>>true</activeByDefault>
    </activation>
    <properties>
      <!-- Optional URL to server. Default value is http://localhost:9000 -->
      <sonar.host.url>
        http://localhost:9000
      </sonar.host.url>
    </properties>
  </profile>
</profiles>
...

```

+

**NOTE** For gradle 2.1 and above, add the following snippet to the build script:

```

plugins {
  id "org.sonarqube" version "2.6.2"
}

```

+ Then, you can run analysis using `gradle sonarqube`

## 9 Build the Project

In another terminal, run the command `mvn clean install sonar:sonar` at the root of the project folder

You should get an output similar to this:

```

[INFO] Sensor XML Sensor [xml]
[INFO] Sensor XML Sensor [xml] (done) | time=77ms
[INFO] Sensor Zero Coverage Sensor
[INFO] Sensor Zero Coverage Sensor (done) | time=11ms
[INFO] Sensor Java CPD Block Indexer
[INFO] Sensor Java CPD Block Indexer (done) | time=13ms
[INFO] SCM provider for this project is: git
[INFO] 3 files to be analyzed
[INFO] 0/3 files analyzed
[WARNING] Missing blame information for the following files:
[WARNING] * pom.xml
[WARNING] * src/main/java/mcgill/ecse429/tutorial2/App.java
[WARNING] * src/test/java/mcgill/ecse429/tutorial2/AppTest.java
[WARNING] This may lead to missing/broken features in SonarQube
[INFO] 1 file had no CPD blocks
[INFO] Calculating CPD for 0 files
[INFO] CPD calculation finished
[INFO] Analysis report generated in 70ms, dir size=34 KB
[INFO] Analysis reports compressed in 16ms, zip size=12 KB
[INFO] Analysis report uploaded in 21ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard?id=mcgill.ecse429%3Atutorial2
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://localhost:9000/api/ce/task?id=AWX6UG4FKQha6VudJpJT
[INFO] Task total time: 4.301 s
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.815 s
[INFO] Finished at: 2018-09-21T00:08:19-04:00

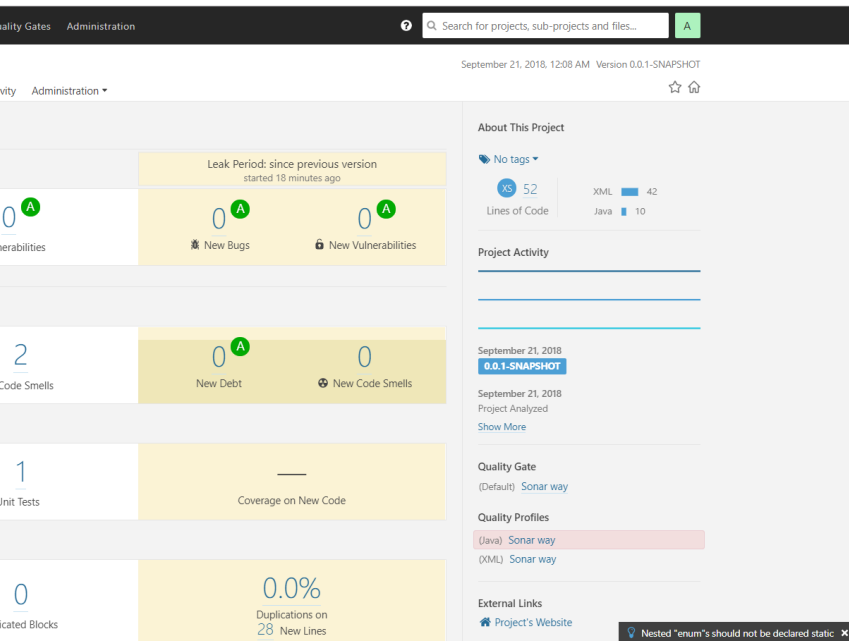
```

If we visit the link in the terminal output (it follows the format of [http://localhost:9000/dashboard?id=<INSERT\\_PROJECT\\_NAME>](http://localhost:9000/dashboard?id=<INSERT_PROJECT_NAME>)), we are brought to a page similar to this:

The screenshot shows the SonarQube dashboard for a project named 'tutorial2'. The top navigation bar includes 'sonarqube', 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', and 'Administration'. A search bar is present on the right. The main content area is divided into several sections:

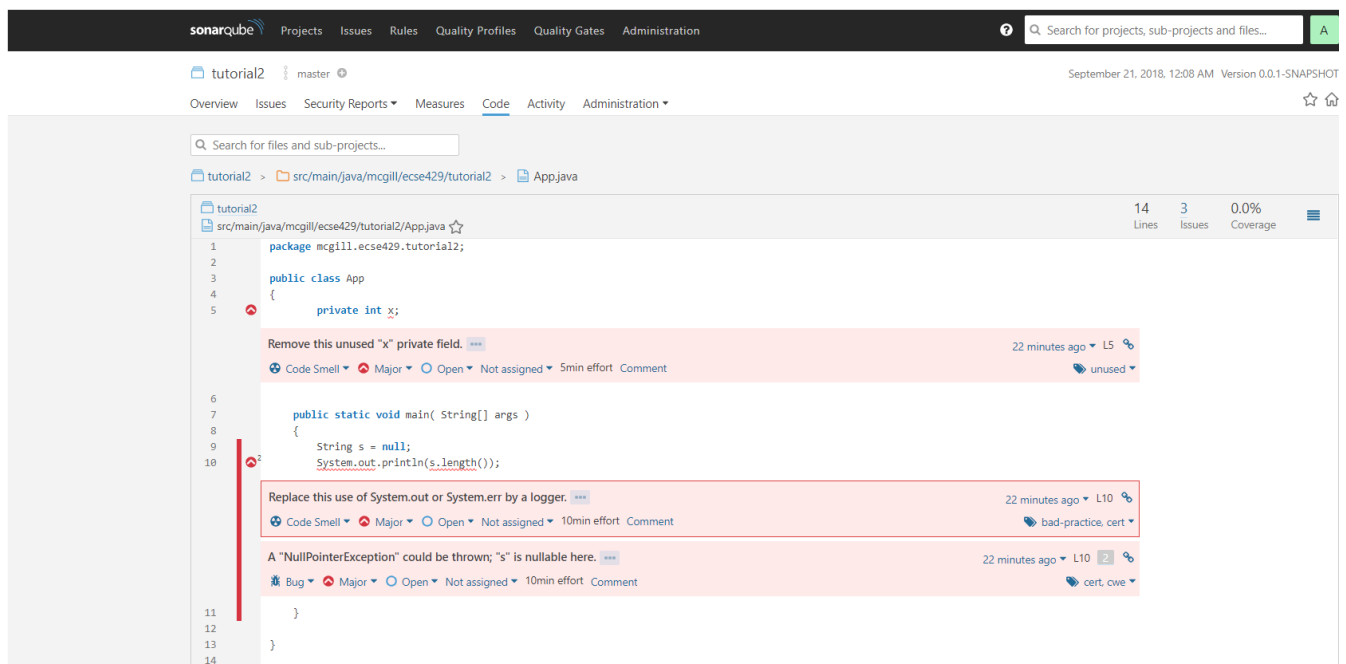
- Quality Gate:** Shows a 'Passed' status with a green checkmark.
- Bugs:** Displays 1 bug (yellow circle with 'C') and 0 vulnerabilities (green circle with 'A'). A 'Leak Period' of 18 minutes is noted.
- Code Smells:** Shows 15min of debt (green circle with 'A') and 2 code smells (yellow circle with 'A').
- Coverage:** Shows 0.0% coverage (red circle) and 1 unit test (yellow circle).
- Duplications:** Shows 0.0% duplications (green circle) and 0 duplicated blocks (yellow circle).

On the right side, there is a sidebar with 'About This Project' (52 lines of code, 42 XML, 10 Java), 'Project Activity' (last analyzed on September 21, 2018), 'Quality Gate' (Default: Sonar way), 'Quality Profiles' (Java: Sonar way, XML: Sonar way), and 'External Links' (Project's Website). A notification at the bottom right states: 'Nested "enum"s should not be declared static.'



## 10 Investigate the Error Causes:

Click on *Code* in the toolbar above. Then select the package containing the `App.java` class from the list and click on `App.java` and click on the red icons on the left of your code.



## Adding Code Coverage

1. Edit the `pom.xml` and add a new profile somewhere between the `<profiles>` tags

```

<profile>
<id>sonar-coverage</id>
<activation>
<activeByDefault>true</activeByDefault>
</activation>
<build>
<pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.7.8</version>
    </plugin>
  </plugins>
</pluginManagement>
<plugins>
  <plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <configuration>
      <append>true</append>
    </configuration>
    <executions>
      <execution>
<id>agent-for-ut</id>
<goals>
  <goal>prepare-agent</goal>
</goals>
      </execution>
      <execution>
<id>jacoco-site</id>
<phase>verify</phase>
<goals>
  <goal>report</goal>
</goals>
      </execution>
    </executions>
  </plugin>
</plugins>
</build>
</profile>

```

## 2. Write a Dummy Test

In `AppTest.java`, substitute `emptyTest` with the following:



```
@Test
public void mainTest() {
    App.main(null);
}
```

3 Run the command `mvn clean install sonar:sonar` once again

4 Use the outputted URL to inspect the changes in SonarQube

5 Click on the 75% of the Code Coverage Section

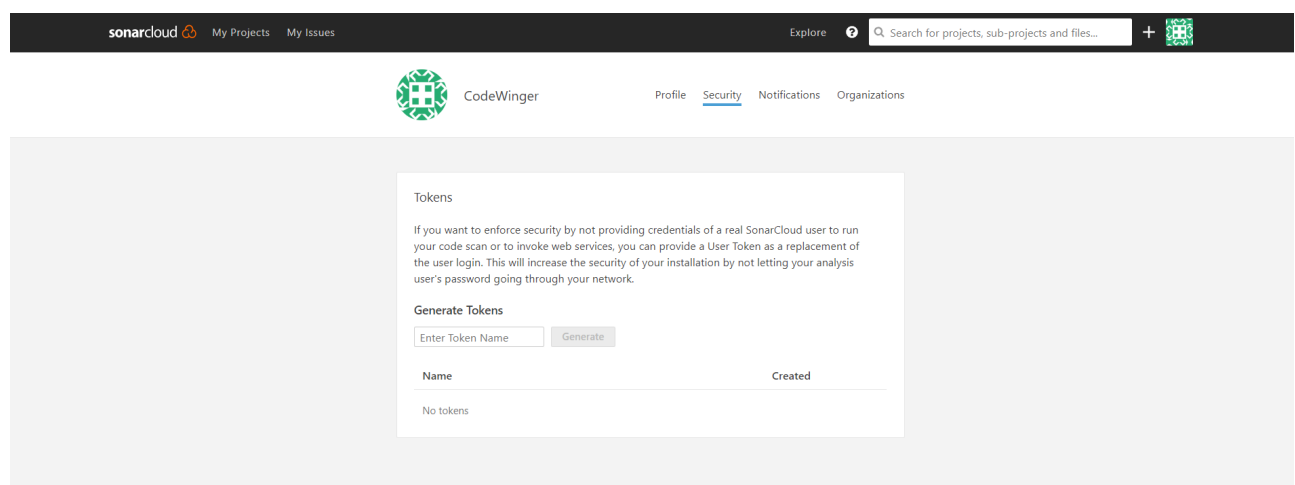
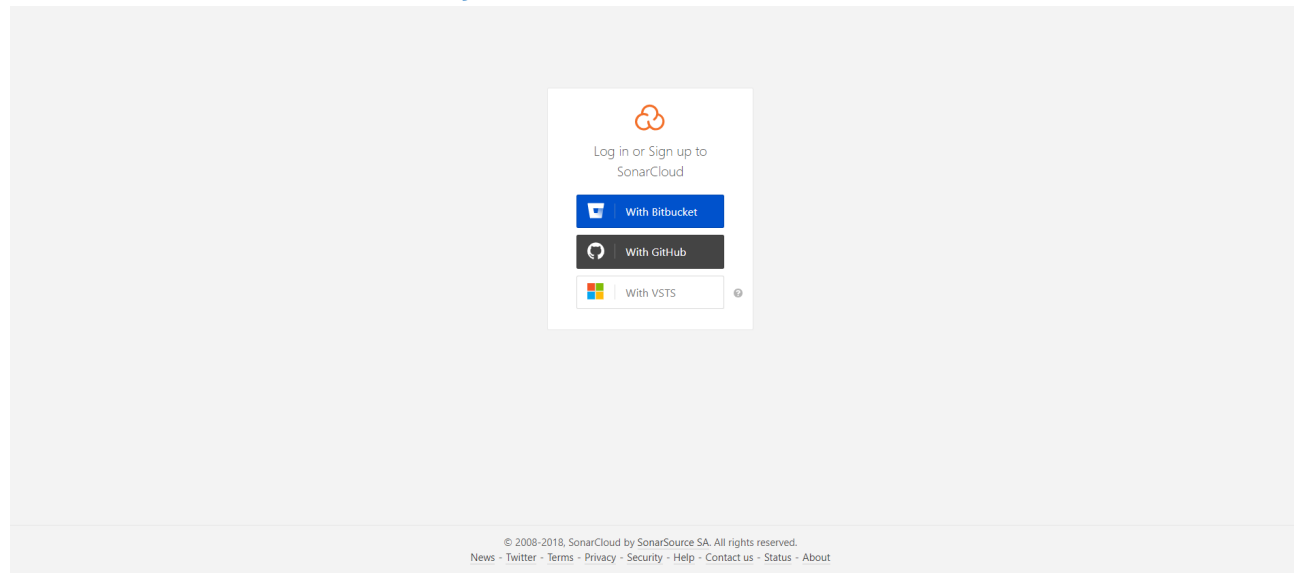
6 Click on the App.java file in the list

7 Look to the left of the red error icons. We now see which lines were tested by the unit tests.

## 4.2.2. SonarQube Integrated With Travis

1. Register your Github Account in this link (<https://sonarcloud.io/sessions/new?>)

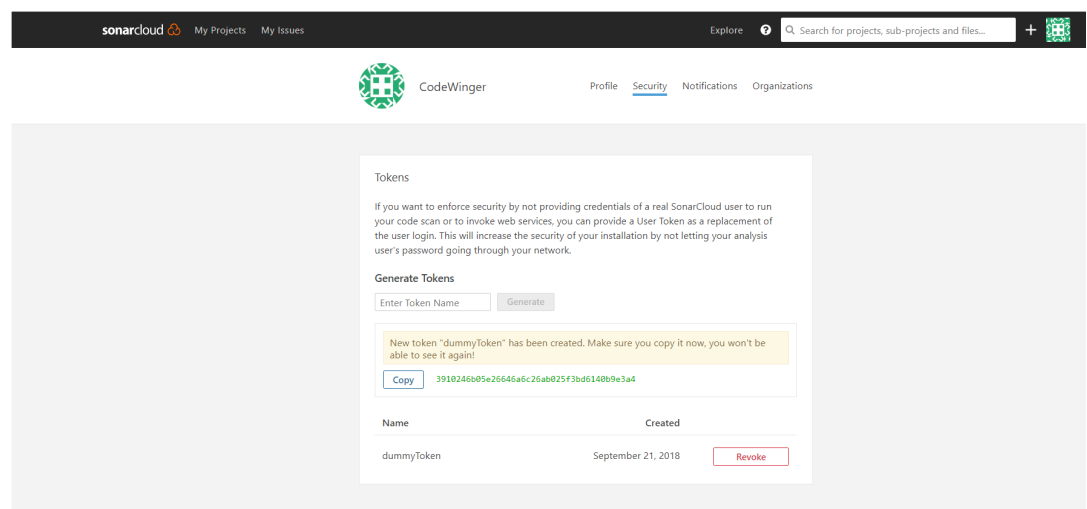
return\_to=%2Faccount%2Fsecurity)



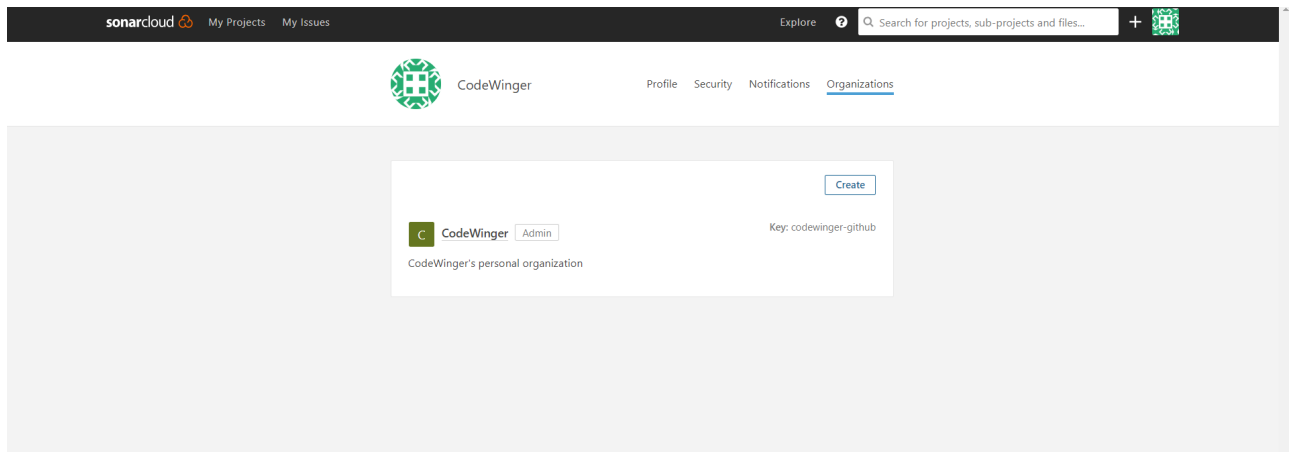
2. Add a security token of your choice and store in a file of your choice

It is important to copy it somewhere; you will not be able to see it again afterwards

**NOTE**



3. Click on your picture in the top right-hand side of the toolbar, next to the search bar, then on the button *My Organizations*



4. Jot down the key given by SonarCloud.io under the *Create* button
5. Add a `.travis.yml` file to the project's directory

```
language: java
sudo: false
install: true

addons:
  sonarcloud:
    organization: "[YOUR ORGANIZATION KEY]"
    token:
      secure: "[YOUR GENERATED TOKEN]"

jdk:
  - oraclejdk8

script:
  - mvn clean org.jacoco:jacoco-maven-plugin:prepare-agent package sonar:sonar

cache:
  directories:
    - '$HOME/.m2/repository'
    - '$HOME/.sonar/cache'
```

For this example, the `addons` section would look like

```
addons:
  sonarcloud:
    organization: "codewinger-github"
    token:
      secure: "3910246b05e26646a6c26ab025f3bd6140b9e3a4"
```

- 6 Create a new local repository with the command `git init` (*Optional*)
- 7 Create a new upstream repository on GitHub (*Optional*)

After creating the repository, from the command line enter the commands:

```
git remote add origin https://github.com/[YOUR GITHUB ID]/[YOUR REPO NAME].git
```

In my case, the above would look like:

```
git remote add origin https://github.com/CodeWinger/tutorial3.git
```

#### 8 Add a `.gitignore` file (*Optional*)

```
# Compiled class file
*.class

# Log file
*.log

# BlueJ files
*.ctxt

# Mobile Tools for Java (J2ME)
.mtj.tmp/

# Package Files #
*.jar
*.war
*.nar
*.ear
*.zip
*.tar.gz
*.rar

# virtual machine crash logs, see
http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*
```

9 Push your work from the `master` branch to GitHub with the command `git add . && git commit -m "SonarCloud Integration" & git push origin master:master --set-upstream`

We need to first analyze the `main` branch before we can analyze new branches along with their differences with the `master` branch Your output should look something like this:

```

1448 [INFO] Sensor XML Sensor [xml] (done) | time=133ms
1449 [INFO] Sensor Zero Coverage Sensor
1450 [INFO] Sensor Zero Coverage Sensor (done) | time=13ms
1451 [INFO] Sensor JavaSecuritySensor [security]
1452 [INFO] UCFGs: 2, excluded: 2, source entrypoints: 0
1453 [INFO] No UCFGs have been included for analysis.
1454 [INFO] Sensor JavaSecuritySensor [security] (done) | time=80ms
1455 [INFO] Sensor CSharpSecuritySensor [security]
1456 [INFO] UCFGs: 0, excluded: 0, source entrypoints: 0
1457 [INFO] No UCFGs have been included for analysis.
1458 [INFO] Sensor CSharpSecuritySensor [security] (done) | time=0ms
1459 [INFO] Sensor Java CPD Block Indexer
1460 [INFO] Sensor Java CPD Block Indexer (done) | time=16ms
1461 [INFO] SCM provider for this project is: git
1462 [INFO] 3 files to be analyzed
1463 [INFO] 3/3 files analyzed
1464 [INFO] 1 file had no CPD blocks
1465 [INFO] calculating CPD for 0 files
1466 [INFO] CPD calculation finished
1467 [INFO] Analysis report generated in 150ms, dir size=63 KB
1468 [INFO] Analysis reports compressed in 14ms, zip size=23 KB
1469 [INFO] Analysis report uploaded in 1122ms
1470 [INFO] ANALYSIS SUCCESSFUL, you can browse https://sonarcloud.io/dashboard?id=mcgill.ecse429%3Atutorial2
1471 [INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
1472 [INFO] More about the report processing at https://sonarcloud.io/api/ce/task?id=AWX9QxmlHR8j8dJrX_Iy
1473 [INFO] Task total time: 13.329 s
1474 [INFO] -----
1475 [INFO] BUILD SUCCESS
1476 [INFO] -----
1477 [INFO] Total time: 54.065 s
1478 [INFO] Finished at: 2018-09-21T17:52:37Z
1479 [INFO] Final Memory: 74M/914M
1480 [INFO] -----
1481
1482

```

10 Copy the outputted URL (<https://sonarcloud.io/dashboard?id=mcgill.ecse429%3Atutorial2>, in this case) and paste it in your Internet browser.

The screenshot shows the SonarCloud dashboard for a project named 'tutorial2'. The main dashboard area displays several key metrics:

- Quality Gate:** Passed (indicated by a green circle).
- Bugs:** 1 (indicated by a yellow circle with 'C').
- Vulnerabilities:** 0 (indicated by a green circle with 'A').
- Code Smells:** 3 (indicated by a green circle with 'A').
- Debt:** 25min (indicated by a green circle with 'A').
- Coverage:** 57.1% (indicated by a green circle).
- Unit Tests:** 1 (indicated by a green circle).
- Duplications:** 0.0% (indicated by a green circle).
- Duplicated Blocks:** 0 (indicated by a green circle).

The right sidebar contains the following sections:

- About This Project:** Public, No tags, 34 Lines of Code, XML 20, Java 14.
- Project Activity:** September 21, 2018, 0.0.1-SNAPSHOT.
- Quality Gate:** (Default) Sonar way.
- Quality Profiles:** (Java) Sonar way, (XML) Sonar way.
- External Links:** Project's Website.
- Project Key:** mcgill.ecse429:tutorial2 (Copy).
- Organization Key:** codewinger-github (Copy).
- Get project badges:** (button).

11 From the `master` branch, create and checkout a new branch `git branch sonarqube-travis-integration` && `git checkout sonarqube-travis-integration`

12 Modify `App.java`'s class definition:

```

public class App
{
    public static void main( String[] args )
    {
        if (args == null || args.length == 0) {
            System.out.println("no input");
            return;
        }

        for (String argument : args) {
            System.out.println(argument);
        }
    }
}

```

13 Modify `AppTest.java`'s class definition:

```

import org.junit.Test;

public class AppTest
{
    @Test
    public void testMainMethodForNull() {
        App.main(null);
    }

    @Test
    public void testMainMethodForEmptyArray() {
        App.main(new String[0]);
    }

    @Test
    public void testMainMethodForNonEmptyArray() {
        String[] args = {"Hello", "World"};
        App.main(args);
    }
}

```

14 Push your work on the branch and check the output in SonarCloud.io (`git add . && git commit -m "SonarCloud Integration" & git push -u origin sonarqube-travis-integration`)

The screenshot shows the SonarCloud interface for a project named 'tutorial2'. The left sidebar contains a 'Filters' panel with the following details:

- Type:** Bug (0), Vulnerability (0), Code Smell (2), Security Hotspot (0)
- Severity:** Blocker (0), Critical (0), Major (2), Minor (0), Info (0)
- Resolution:** (collapsed)
- Status:** (collapsed)
- Creation Date:** (collapsed)
- Language:** (collapsed)
- Rule:** (collapsed)
- Standard:** (collapsed)
- Tag:** (collapsed)
- Module:** (collapsed)
- Directory:** (collapsed)
- File:** (collapsed)
- Assignee:** (collapsed)
- Author:** (collapsed)

The main content area shows two issues for the file `src/.../java/mcgill/ecse429/tutorial2/App.java`:

- Issue 1:** Replace this use of System.out or System.err by a logger. (Code Smell, Major, 10min effort, 15 minutes ago, L12)
- Issue 2:** Replace this use of System.out or System.err by a logger. (Code Smell, Major, 10min effort, 15 minutes ago, L17)

At the bottom of the page, there is a footer with copyright information: © 2008-2018, SonarCloud by SonarSource SA. All rights reserved. Links for News, Twitter, Terms, Privacy, Security, Help, Contact us, Status, and About are provided.

## 4.3. SonarCloud integration for Project deliverable

1. One team member is registered to SonarCloud organization McGill ECSE429 Fall2018. Follow the steps with that team member's SonarCloud account.
2. Create a SonarCloud project *under the organization* (and not under your user). Name it **team-NO-blokish**, where **NO** is the placeholder for the two digit team number. Assign a unique project key to it afterwards as shown below.

The screenshot shows the 'Create project(s)' form in SonarCloud. The 'Create manually' tab is active. The form contains the following fields:

- Organization:** McGill ECSE 429 Fall 2018 (with a link to 'I want to create another organization')
- Project name:** team-00-blokish
- Project key:** ecse429-team-00-blokish

A 'Create' button is located at the bottom left of the form.

3. Generate an access token for the project.

## Analyze your project

We initialized your project on SonarCloud, now it's up to you to launch analyses!

Below are the commands to use to do an analysis.  
If you are using Travis CI, the SonarCloud Travis Add-on makes it easier to run these commands with your CI process.  
Simply follow the [guide to integrating with Travis CI](#).

### 1 Provide a token

Generate a token

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point of time in your user account.

### 2 Run analysis on your project

#### About This Project

No tags

#### Quality Gate

(Default) [Sonar way](#)

#### Project Key

#### Organization Key

4. On the next page, you will be shown two snippets:

- one is to be added to the `build.gradle` file
- the other one is the command you can use out-of-the box to build and analyze on SonarCloud

### 2 Run analysis on your project

What is your project's main language?

You are developing primarily in Java: what is your build technology?

Execute the Scanner for Gradle from your computer

Running an analysis with Gradle is straightforward. You just need to declare the `org.sonarqube` plugin in your `build.gradle` file:

```
plugins {
  id "org.sonarqube" version "2.6"
}
```

and run the following command:

```
./gradlew sonarqube \
-Dsonar.projectKey=ecse429-team-00-blokish \
-Dsonar.organization=ecse429-fall2018 \
-Dsonar.host.url=https://sonarcloud.io \
-Dsonar.login=
```

Please visit the [official documentation of the Scanner for Gradle](#) for more details.

Once the analysis is completed, this page will automatically refresh and you will be able to browse the analysis results.



## 4.4. Infer Static Analyzer

1. Create and checkout a new branch from master (`git branch infer && git checkout infer`)
2. Edit the `.travis.yml` file:

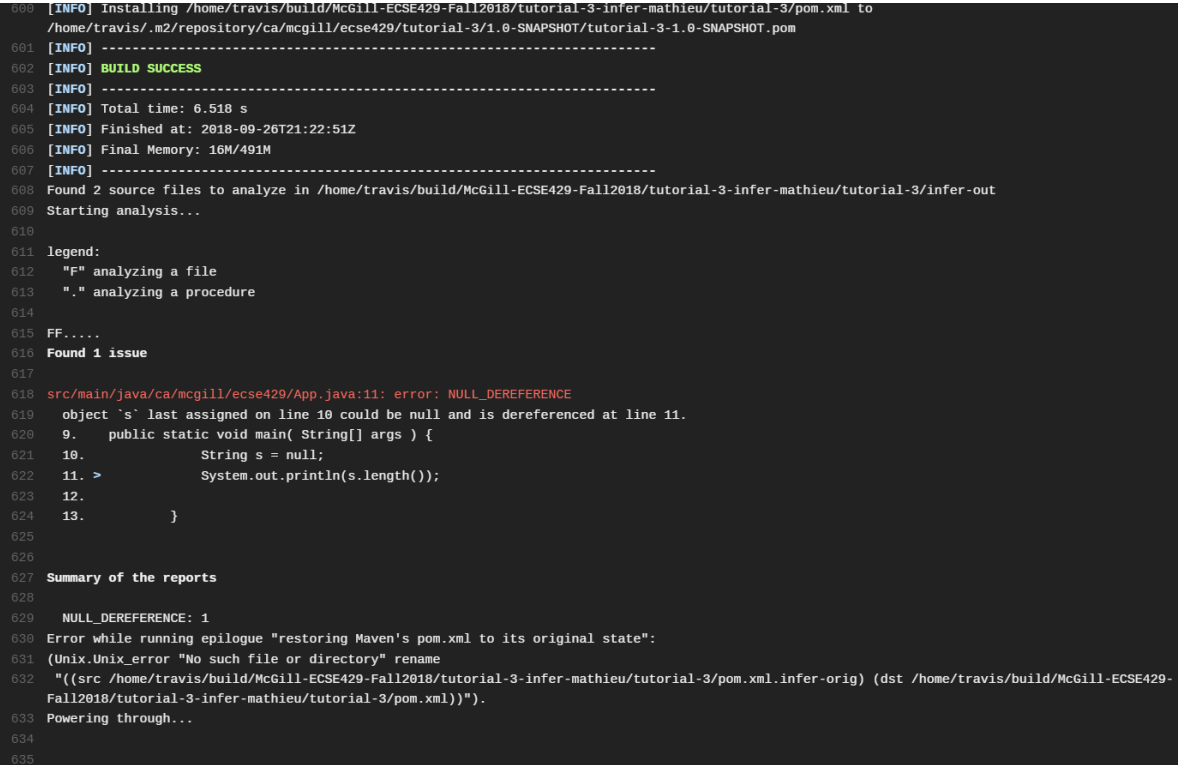
```
dist: trusty
sudo: required
language: java
jdk: oraclejdk7

script:
  - #TODO: get infer archive and extract it
  - #TODO: add infer executable to the path
  - #TODO: invoke infer with your project
```

3. Introduce a null warning In the default file `App.java`, change the main method to the following:

```
public static void main( String[] args ) {
    String s = null;
    System.out.println(s.length());
}
```

4. Commit and push the 2 modified files (`commit add . && git commit -m "Added Infer Static Analyzer" && git push -u origin infer`)
5. Go to <https://travis-ci.com/> to see your build processes. It should look something similar to this:



```
600 [INFO] Installing /home/travis/build/McGill-ECSE429-Fall2018/tutorial-3-infer-mathieu/tutorial-3/pom.xml to
/home/travis/.m2/repository/ca/mcgill/ecse429/tutorial-3/1.0-SNAPSHOT/tutorial-3-1.0-SNAPSHOT.pom
601 [INFO] -----
602 [INFO] BUILD SUCCESS
603 [INFO] -----
604 [INFO] Total time: 6.518 s
605 [INFO] Finished at: 2018-09-26T21:22:51Z
606 [INFO] Final Memory: 16M/491M
607 [INFO] -----
608 Found 2 source files to analyze in /home/travis/build/McGill-ECSE429-Fall2018/tutorial-3-infer-mathieu/tutorial-3/infer-out
609 Starting analysis...
610
611 legend:
612 "F" analyzing a file
613 "." analyzing a procedure
614
615 FF....
616 Found 1 issue
617
618 src/main/java/ca/mcgill/ecse429/App.java:11: error: NULL_DEREFERENCE
619 object `s` last assigned on line 10 could be null and is dereferenced at line 11.
620 9.   public static void main( String[] args ) {
621 10.       String s = null;
622 11. >       System.out.println(s.length());
623 12.
624 13.   }
625
626
627 Summary of the reports
628
629 NULL_DEREFERENCE: 1
630 Error while running epilogue "restoring Maven's pom.xml to its original state":
631 (Unix.Unix_error "No such file or directory" rename
632 "((src /home/travis/build/McGill-ECSE429-Fall2018/tutorial-3-infer-mathieu/tutorial-3/pom.xml.infer-orig) (dst /home/travis/build/McGill-ECSE429-
Fall2018/tutorial-3-infer-mathieu/tutorial-3/pom.xml))").
633 Powering through...
634
635 Top ^
```

# 5. Test Generation and Code Coverage

# 5.1. EvoSuite Installation and Initial Project Setup

**NOTE** For this session, we are reusing a slightly modified version of the example project [available here](#).

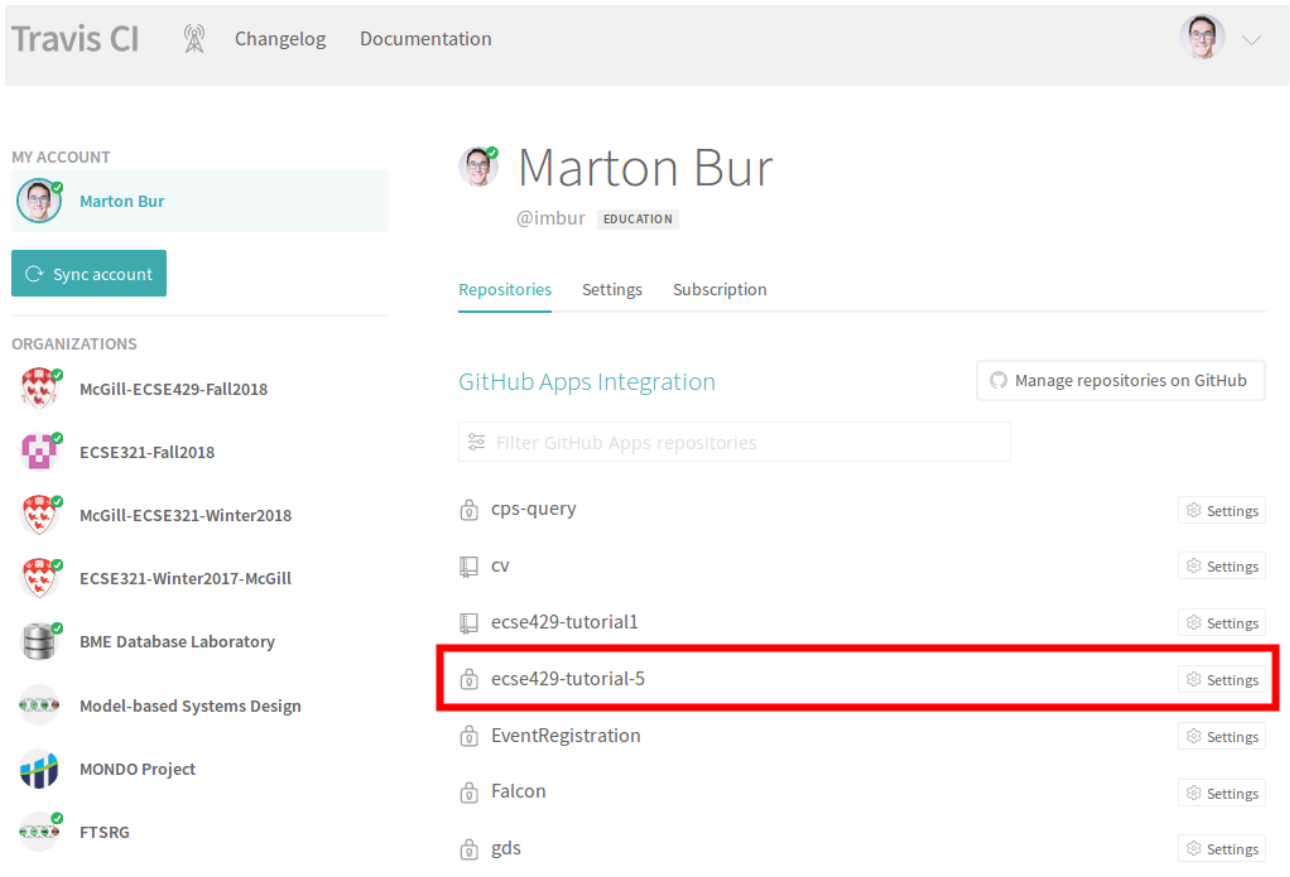
1. Download EvoSuite Commandline tool from <http://www.evosuite.org/downloads/>

**IMPORTANT** EvoSuite is not compatible yet with Java 9 and above.

2. Fork the [ecse429-tutorial-5](#) repository **under your GitHub user**.

**IMPORTANT** After creating your fork, it is important to apply the steps below to your own fork (and not on the organization's repository).

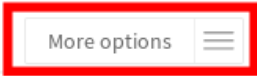
3. Go to <https://travis-ci.com/account/repositories> and make sure you see the fork on Travis. If not, synchronize with GitHub.



4. Click on *Settings* with the cogs icon next to the repository's name. Under *More options* > *Settings* Set the value of the `SONAR_TOKEN` environment variable (the value is the token obtained from sonarcloud.io earlier) for **your fork's** build.

imbur / ecse429-tutorial-5  build unknown

Current Branches Build History Pull Requests Settings



### General

- Build pushed branches 
- Limit concurrent jobs 
- Build pushed pull requests 

### Auto Cancellation

Auto Cancellation allows you to only run builds for the latest commits in the queue. This setting can be applied to builds for Branch builds and Pull Request builds separately. Builds will only be canceled if they are waiting to run, allowing for any running jobs to finish.

- Auto cancel branch builds
- Auto cancel pull request builds

### Environment Variables

Notice that the values are not escaped when your builds are executed. Special characters (for bash) should be escaped accordingly.



- Clone your fork of the repository to your computer.
- Replace **both** `project.key` and `project.name` properties in `.travis.yml` with `ecse429-tutorial-5-YOUR_GITHUB_USERNAME`. Commit and push your changes afterwards and see your project automatically built on Travis-CI and analyzed on SonarCloud.

## 5.2. Generating Tests using EvoSuite

### NOTE

In the following steps, we assume that the `EVOSUITE` environment variable points to the `evosuite.jar` executable, and the program can be started by issuing `java -jar $EVOSUITE`. For Windows: in the default Windows command line tool this would be `java -jar %EVOSUITE%`, while also make sure that the firewall settings allow EvoSuite to run.

- Navigate to the folder where your working copy of your tutorial 5 repository is located.
- Explore the project sources.
- Build the project locally using `./gradlew build` to generate classfiles.
- You can run the app with `./gradlew run`.

5. Go to the folder `simple/` and generate tests for the class `SimpleStaticSut`

```
java -jar $EVOSUITE -class
com.github.cseppento.gradle.evosuite.testprojects.simple.SimpleStaticSut -projectCP
build/classes/java/main/ -Duse_separate_classloader=false
```

6. Explore the switches for EvoSuite with `java -jar $EVOSUITE -help!` Also observe the output of `java -jar $EVOSUITE -listParameters!`
7. Go to (or stay in) the folder `simple/` and generate tests for the class `SimpleObjectSut` with Branch coverage and also print the all test goals as well as the missed goals.

```
java -jar $EVOSUITE -class
com.github.cseppento.gradle.evosuite.testprojects.simple.SimpleObjectSut -projectCP
build/classes/java/main/ -Duse_separate_classloader=false -criterion BRANCH
-Dprint_goals=true -Dprint_missed_goals
```

8. Try other settings according to [EvoSuite documentation](#) to generate tests for the `SimpleObjectSut` class!

## 5.3. Adding the Generated Tests to the Build

1. Tests are generated under `simple/evosuite-tests`. We need to add them to the build.
2. First, add EvoSuite runtime to project dependencies to `simple/simple-deps.gradle`:  
`implementation 'org.evosuite:evosuite-standalone-runtime:1.0.6'`
3. Modify the test sources list in `simple/simple-app.gradle` by adding the following lines to the end of the file:

```
sourceSets {
    test {
        java {
            srcDirs = ['./src/test/java' , './evosuite-tests']
        }
    }
}
```

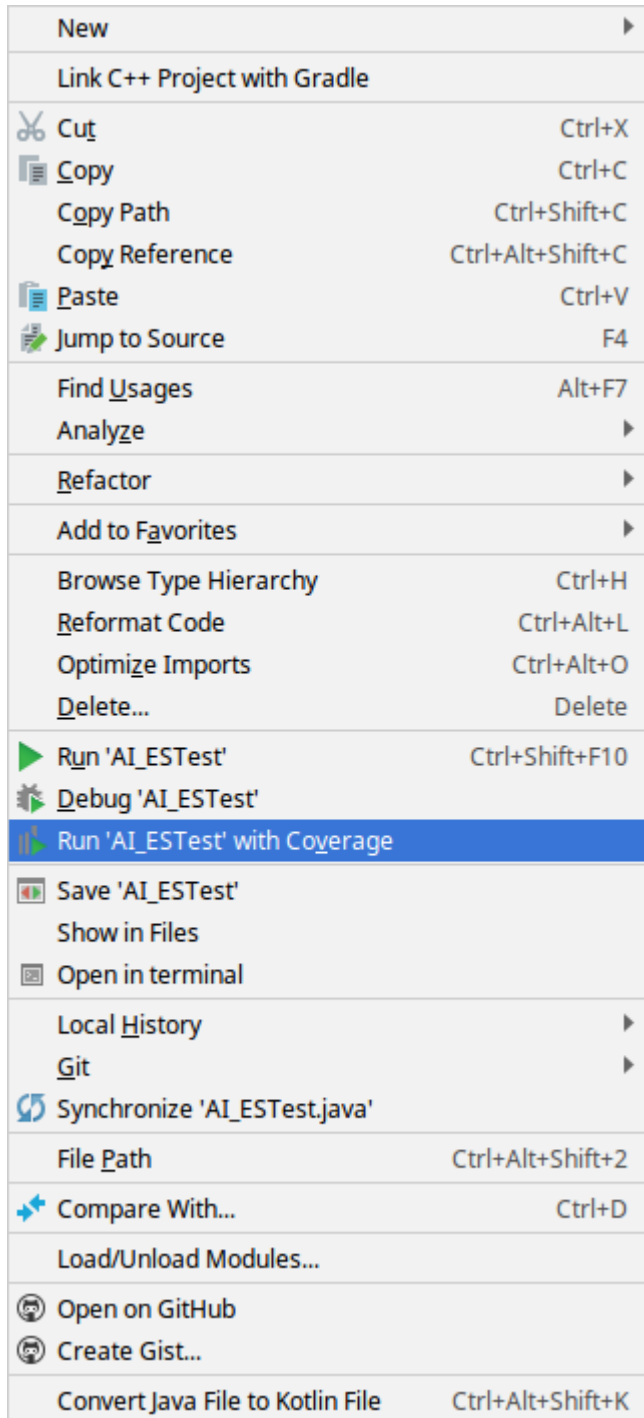
4. Commit your tests and make sure they are executed by Gradle.

### NOTE

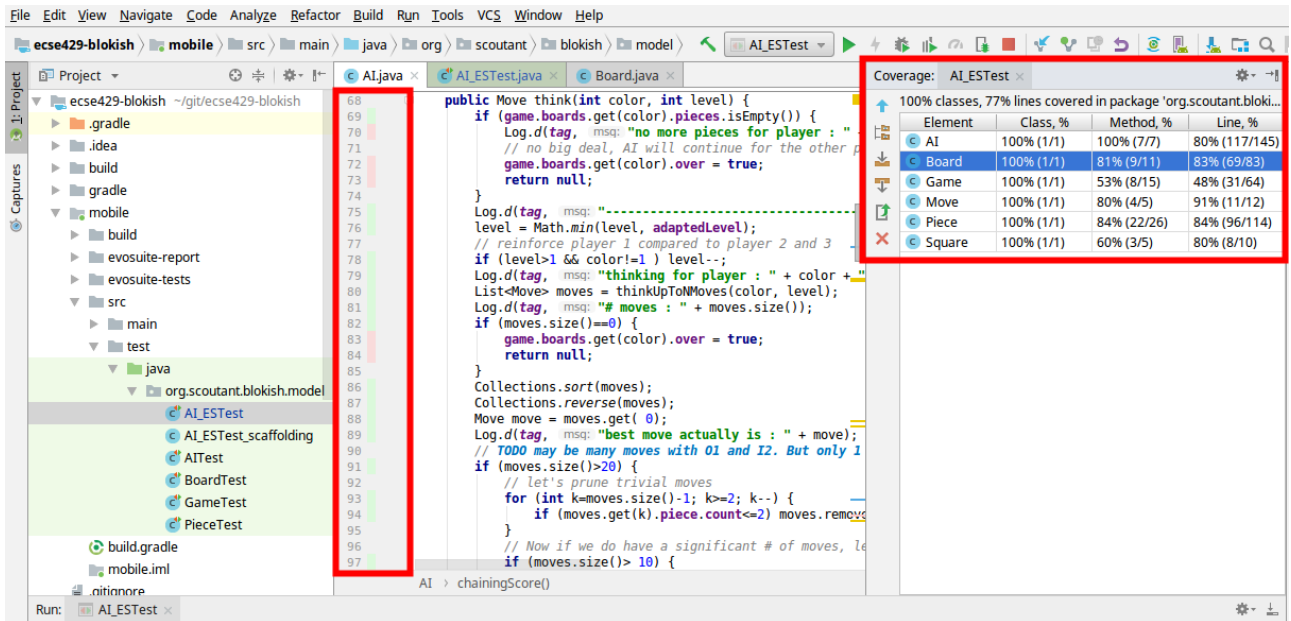
You might encounter build failure if you have `use_separate_classloader=false` set in EvoSuite test cases. If this is the case, switch it back to `use_separate_classloader=true`.

## 5.4. Measuring Code Coverage with Android Studio

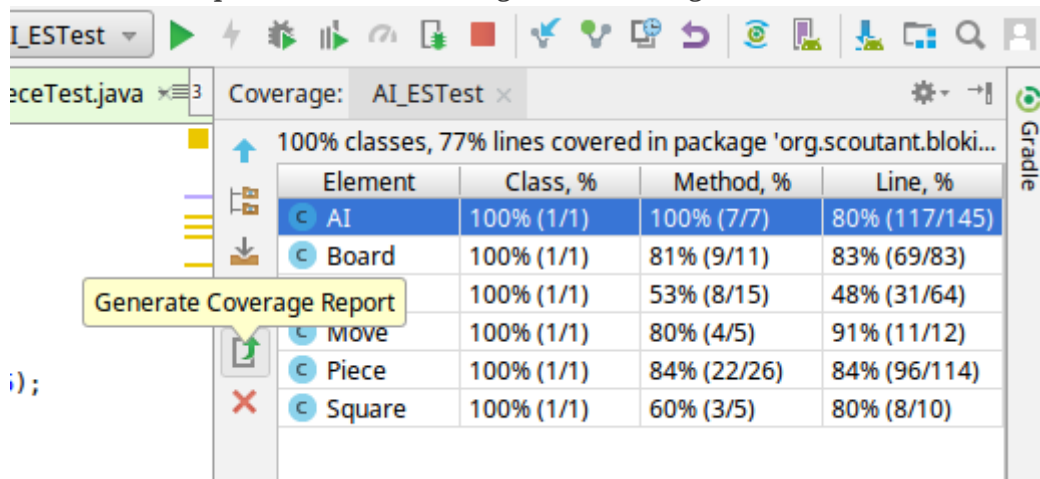
1. Select a single test class or multiple tests classes, then open the right click menu to run them with coverage



2. Android Studio will show the coverage report after running the test. It also highlights which lines were executed by at least one of the test cases



3. You can also export and view coverage results using the button shown below



## 5.5. Cleaning Up the Project

**IMPORTANT:** At the end of the tutorial, turn off the build on Travis-CI and delete the project in SonarCloud as shown below, so that this demo project is not taking up the allowed line count on SonarCloud

Travis:

### General

Build pushed branches ?

Build pushed pull requests ?

Limit cor

### Auto Cancellation

Auto Cancellation allows you to only run builds for the latest commits in the queue. This setting can be applied to build waiting to run, allowing for any running jobs to finish.

SonarCloud:

The screenshot shows the SonarCloud interface for the project 'ecse429-tutorial-05-imbur'. The top navigation bar includes 'sonarcloud', 'My Projects', and 'My Issues'. Below this, the project name 'McGill ECSE 429 Fall 2018 / ecse429-tutorial-05-imbur' is displayed with a 'master' branch indicator. The main navigation menu includes 'Overview', 'Issues', 'Security Reports', 'Measures', 'Code', 'Activity', and 'Administration'. The 'Administration' menu is open, showing options like 'General Settings', 'Branches & Pull Requests', 'Quality Profiles', 'Quality Gate', 'Custom Measures', 'Links', 'Permissions', 'Background Tasks', 'Update Key', 'Webhooks', and 'Deletion'. The 'Deletion' option is highlighted with a red box. The main content area shows 'Quality Gate' with a 'Passed' status, 'Bugs' and 'Vulnerabilities' counts (both 0), and 'Code Smells' count (0).

## 6. Mutation Testing



# 6.1. Mutation Testing Using PIT

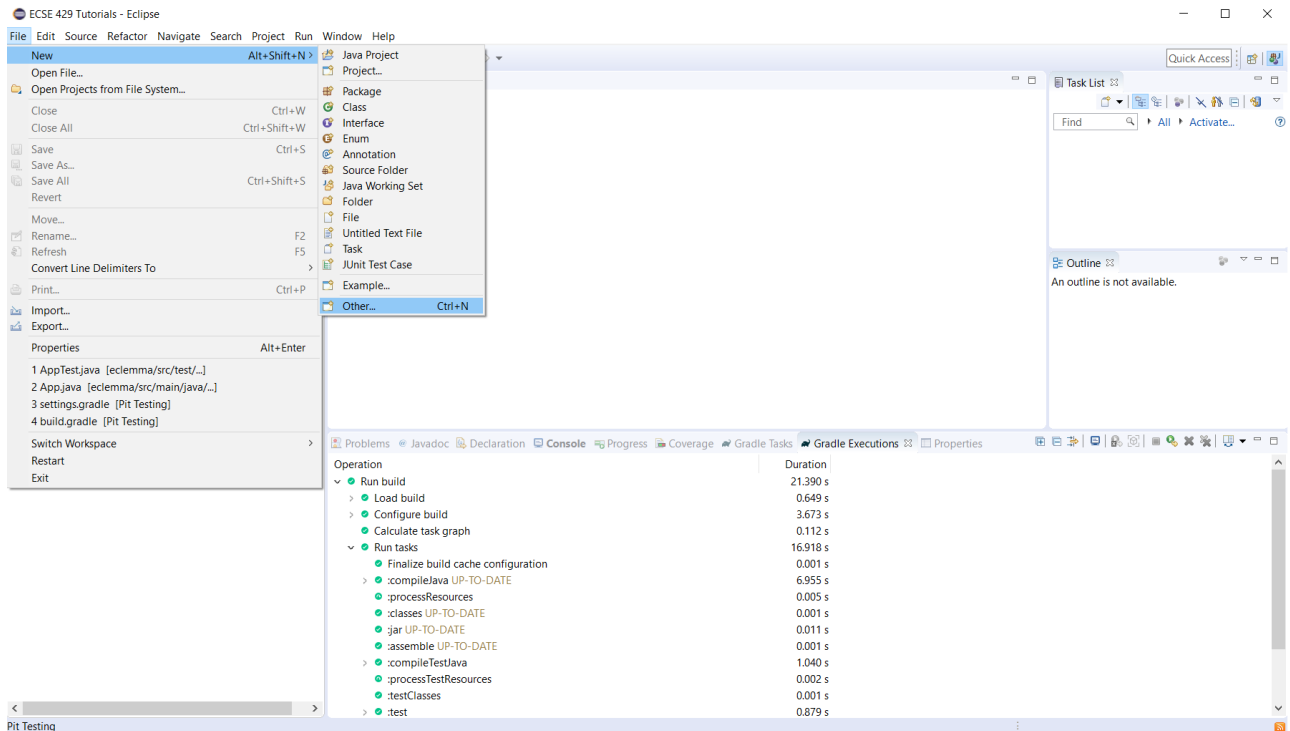
This tutorial covers the basics of using PIT (link:<http://pitest.org/>), both as part of a build using Gradle and using an Eclipse plugin.

## NOTE

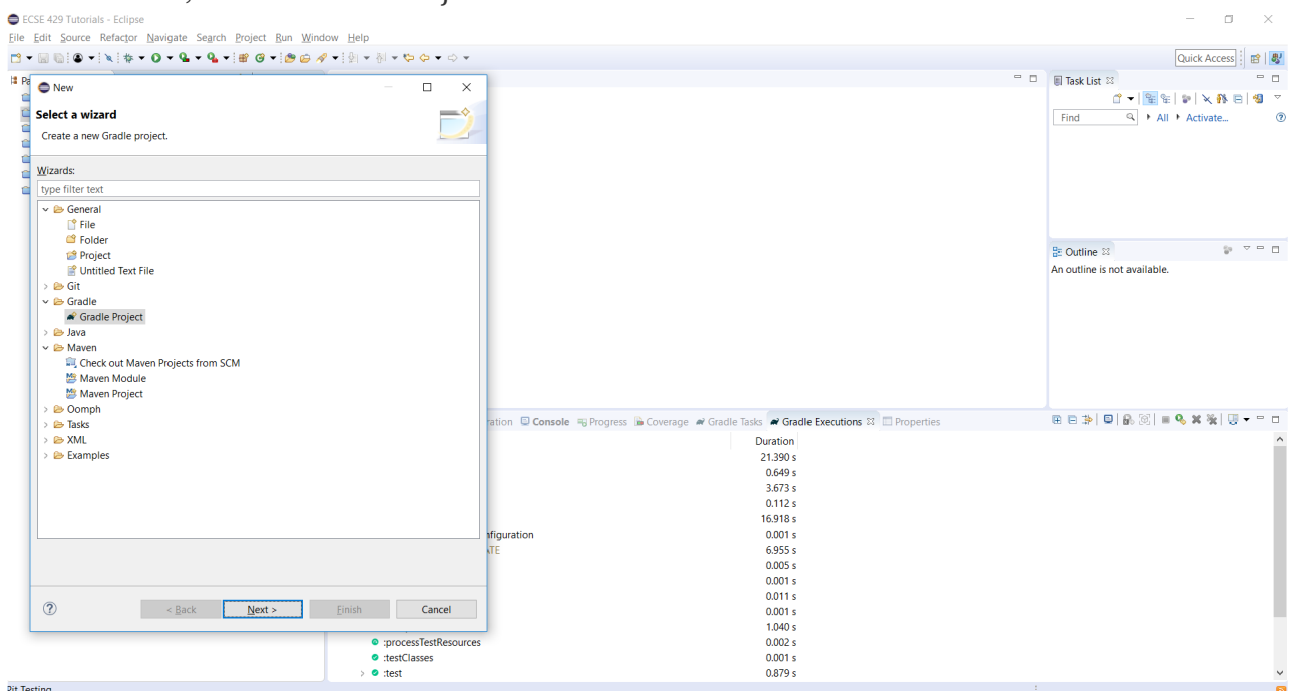
For this tutorial, we assume that you the Gradle Eclipse plugin installed on your computer.

We will create a Gradle project from scratch and be testing the method from the fourth question of your first assignment `returnAverage(int[], int, int, int)`.

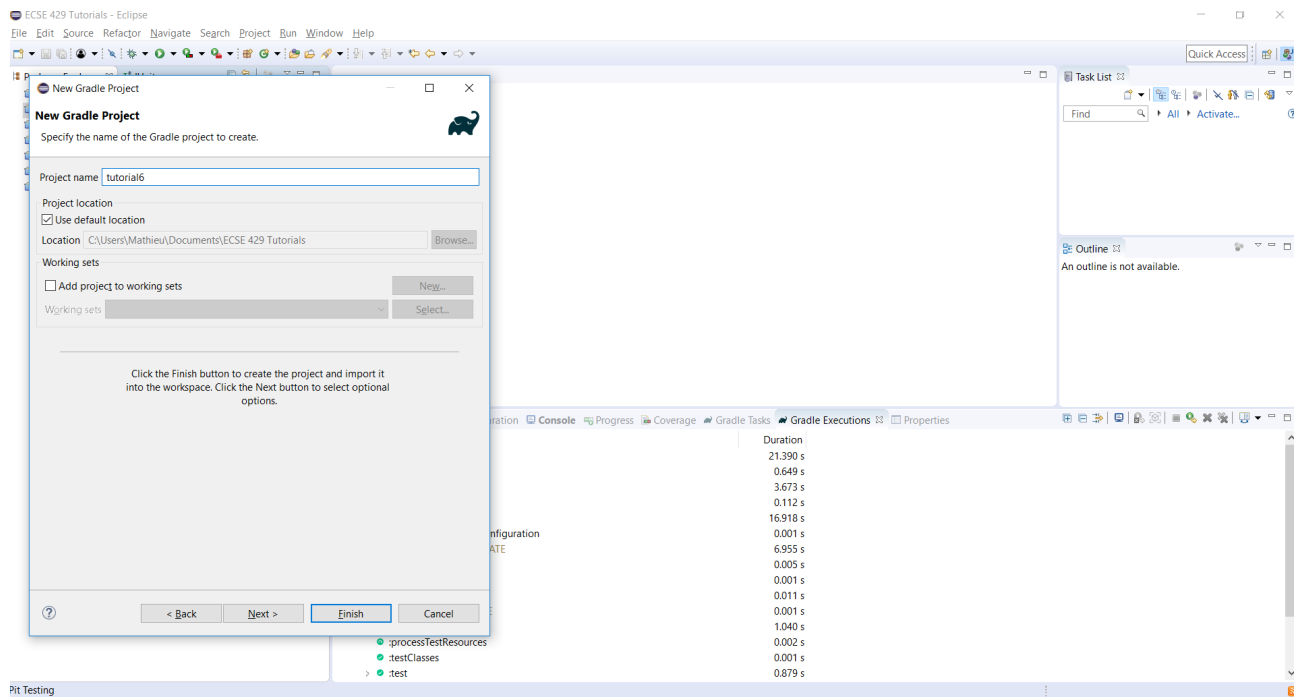
### 1. Create a new Gradle project in Eclipse by clicking on *File > New > Other*



### 2. Under *Gradle*, choose Gradle Project

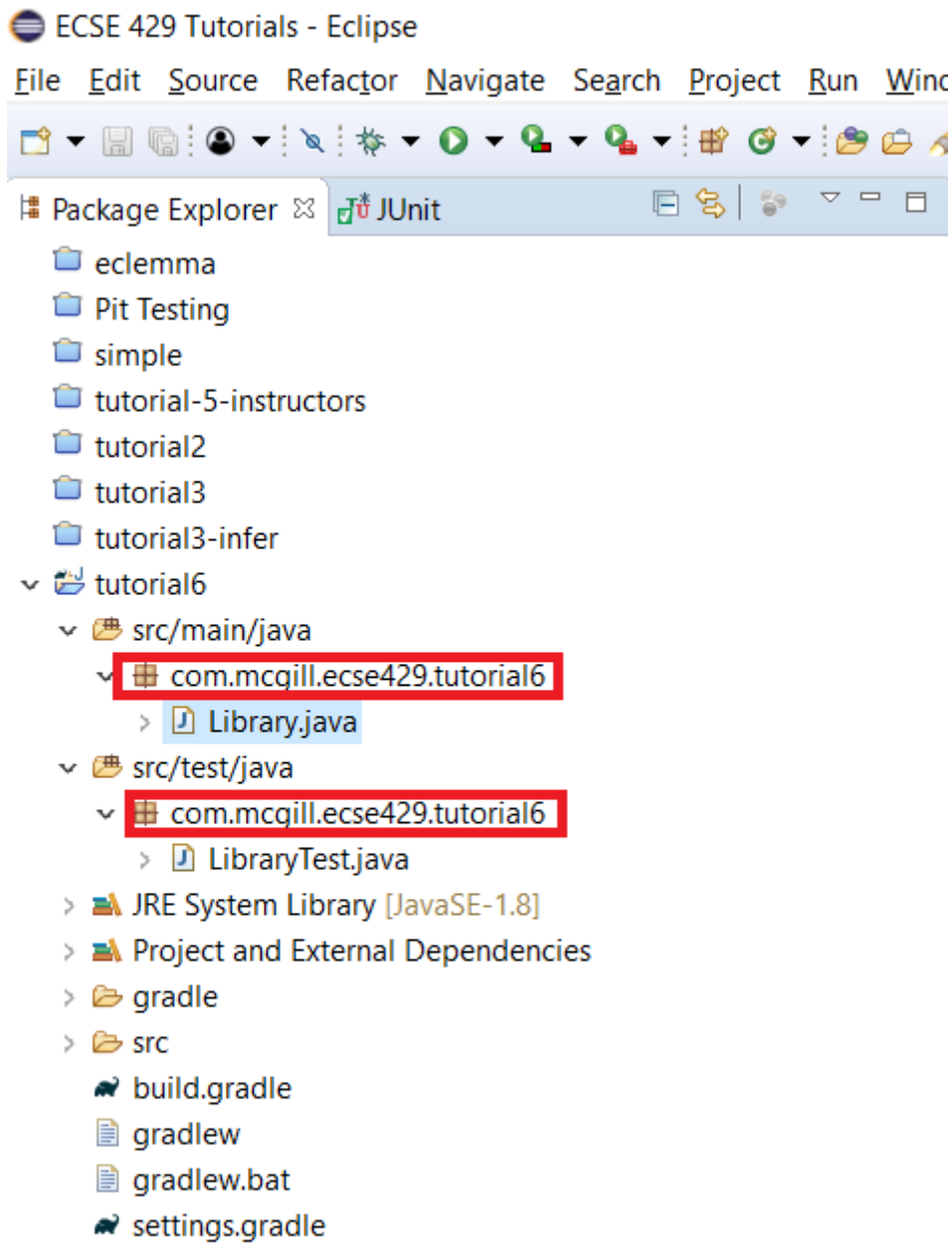


### 3. Click on *Next*, *Next*, then name your project *tutorial6*, click on *Finish*



**NOTE** | The project may take some time to be created.

4. Create a new package instead of the default ones for both the source and test folders (e.g `com.mcgill.ecse429.tutorial6`) and move the default generated classes (`Library` and `LibraryTest`) to this package.



5. Change the code in the `Library` class

```

package com.mcgill.ecse429.tutorial6;

public class Library {

    public static double returnAverage(int value[], int arraySize, int MIN, int
MAX) {
        int index, ti, tv, sum;
        double average;
        index = 0;
        ti = 0;
        tv = 0;
        sum = 0;
        while (ti < arraySize && value[index] != -999) {
            ti++;
            if (value[index] >= MIN && value[index] <= MAX) {
                tv++;
                sum += value[index];
            }
            index++;
        }
        if (tv > 0)
            average = (double) sum / tv;
        else
            average = (double) -999;
        return average;
    }
}

```

6. Change the code in the `LibraryTest` class

```

package com.mcgill.ecse429.tutorial6;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

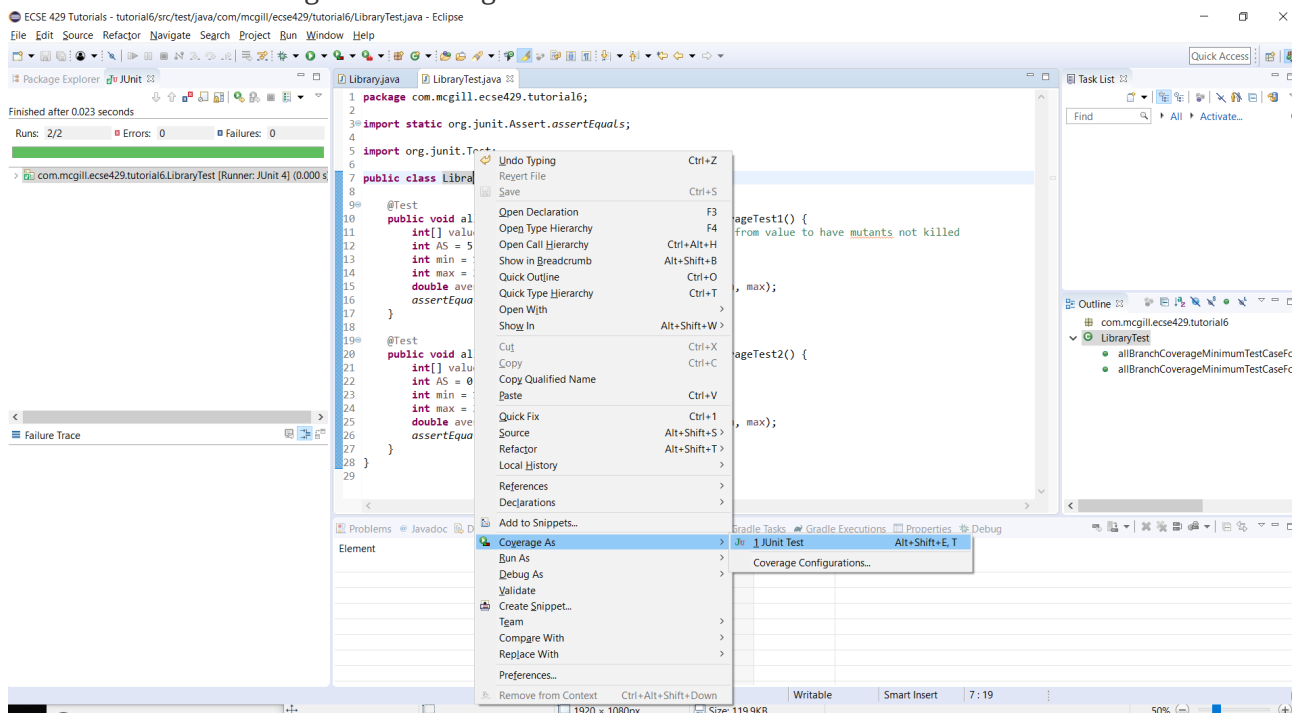
public class LibraryTest {

    @Test
    public void allBranchCoverageMinimumTestCaseForReturnAverageTest1() {
        int[] value = {5, 25, 15, -999};
        int AS = 4;
        int min = 10;
        int max = 20;
        double average = Library.returnAverage(value, AS, min, max);
        assertEquals(15, average, 0.1);
    }

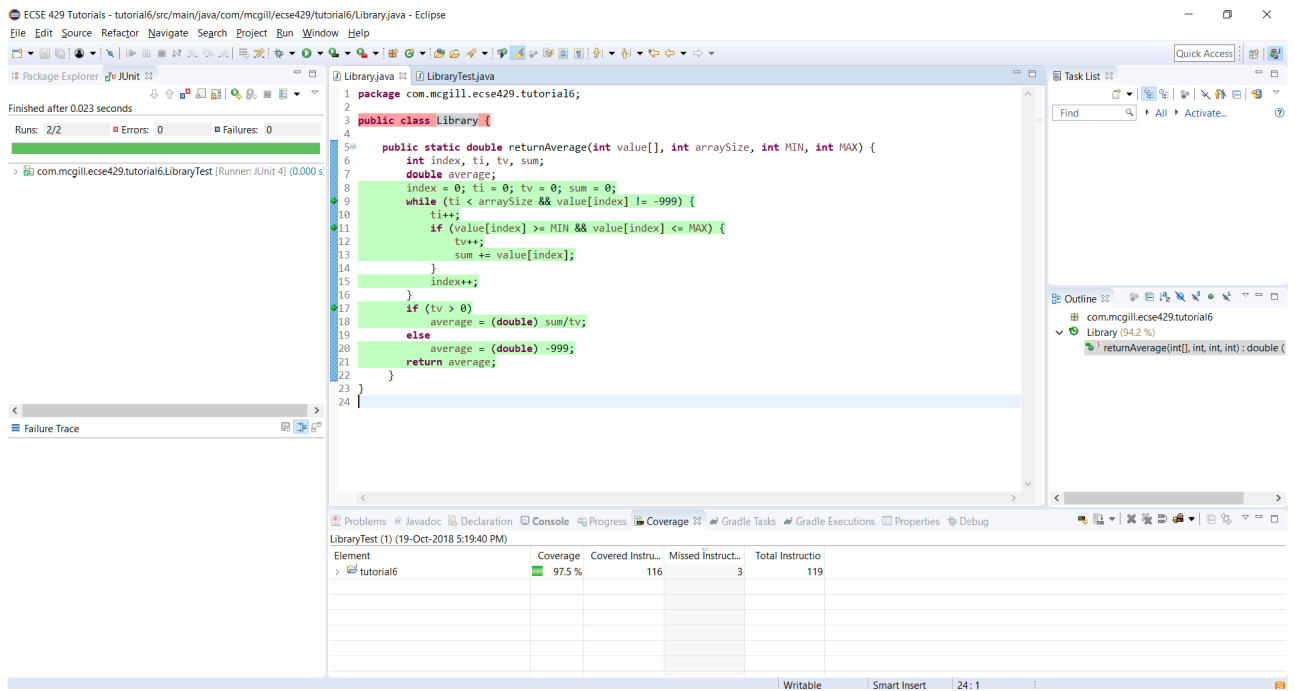
    @Test
    public void allBranchCoverageMinimumTestCaseForReturnAverageTest2() {
        int[] value = {};
        int AS = 0;
        int min = 10;
        int max = 20;
        double average = Library.returnAverage(value, AS, min, max);
        assertEquals(-999.0, average, 0.1);
    }
}

```

## 7. Run the Test in coverage mode using Eclipse.



## 8. Verify that we have 100% branch coverage.



## 6.2. Using Gradle with PIT testing

1. Modify the `build.gradle` file to include the pitest plugin

```
apply plugin: 'java'
apply plugin: 'java-library'

sourceCompatibility = JavaVersion.VERSION_1_8

buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'info.solidsoft.gradle.pitest:gradle-pitest-plugin:1.3.0'
    }
}

repositories {
    jcenter()
}

dependencies {
    testCompile "junit:junit:4.12"

    compile "args4j:args4j:2.0.21"
    compile "org.codehaus.groovy:groovy-all:2.3.7"
}

apply plugin: 'info.solidsoft.pitest'

pitest {
    targetClasses = ["com.mcgill.ecse429.tutorial6*"]
    timestampedReports = true
}
```

2. Navigate to the project folder and run the command `gradle build pitest` (you can also use the Gradle wrapper script)

```
MINGW64:/c:/Users/Mathieu/Documents/ECSE 429 Tutorials/tutorial6
> scan classpath : < 1 second
> coverage and dependency analysis : < 1 second
> build mutation tests : < 1 second
> run mutation analysis : < 1 second

-----
> Total : 1 seconds
-----

- Statistics
-----
>> Generated 15 mutations Killed 13 (87%)
>> Ran 17 tests (1.13 tests per mutation)
-----

- Mutators
-----

> org.pitest.mutationtest.engine.gregor.mutators.ConditionalBoundaryMutator
>> Generated 4 Killed 3 (75%)
> KILLED 3 SURVIVED 1 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
-----

> org.pitest.mutationtest.engine.gregor.mutators.IncrementsMutator
>> Generated 3 Killed 2 (67%)
> KILLED 2 SURVIVED 1 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
-----

> org.pitest.mutationtest.engine.gregor.mutators.ReturnValsMutator
>> Generated 1 Killed 1 (100%)
> KILLED 1 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
-----

> org.pitest.mutationtest.engine.gregor.mutators.MathMutator
>> Generated 2 Killed 2 (100%)
> KILLED 2 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
-----

> org.pitest.mutationtest.engine.gregor.mutators.NegateConditionalsMutator
>> Generated 5 Killed 5 (100%)
> KILLED 5 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
-----

BUILD SUCCESSFUL in 10s
5 actionable tasks: 5 executed
```

3. In your projects root folder, open the file `./build/reports/pitests/{TIMESTAMP}/index.html`, then navigate to the report for the `Library` class



## Library.java

```
1 package com.mcgill.ecse429.tutorial6;
2
3 public class Library {
4
5     public static double returnAverage(int value[], int arraySize, int MIN, int MAX) {
6         int index, ti, tv, sum;
7         double average;
8         index = 0; ti = 0; tv = 0; sum = 0;
9         while (ti < arraySize && value[index] != -999) {
10             ti++;
11             if (value[index] >= MIN && value[index] <= MAX) {
12                 tv++;
13                 sum += value[index];
14             }
15             index++;
16         }
17         if (tv > 0)
18             average = (double) sum/tv;
19         else
20             average = (double) -999;
21         return average;
22     }
23 }
```

### Mutations

```
9 1. changed conditional boundary → SURVIVED
2. negated conditional → KILLED
3. negated conditional → KILLED
10 1. Changed increment from 1 to -1 → SURVIVED
1. changed conditional boundary → SURVIVED
2. changed conditional boundary → SURVIVED
11 3. negated conditional → KILLED
4. negated conditional → KILLED
12 1. Changed increment from 1 to -1 → KILLED
13 1. Replaced integer addition with subtraction → KILLED
15 1. Changed increment from 1 to -1 → KILLED
17 1. changed conditional boundary → KILLED
2. negated conditional → KILLED
18 1. Replaced double division with multiplication → SURVIVED
21 1. replaced return of double value with -(x + 1) for com/mcgill/ecse429/tutorial6/Library::returnAverage → KILLED
```

### Active mutators

- INCREMENTS\_MUTATOR
- VOID\_METHOD\_CALL\_MUTATOR
- RETURN\_VALS\_MUTATOR
- MATH\_MUTATOR
- NEGATE\_CONDITIONALS\_MUTATOR
- INVERT\_NEGS\_MUTATOR
- CONDITIONALS\_BOUNDARY\_MUTATOR

### Tests examined

- com.mcgill.ecse429.tutorial6.LibraryTest.allBranchCoverageMinimumTestCaseForReturnAverageTest1(com.mcgill.ecse429.tutorial6.LibraryTest) (7 ms)

### NOTE

You may have a different output as the one shown. This is normal as the types and number of mutants may vary each run.

4. Look at the reports and get familiar with it.

We can see the code coverage, the location where the mutations took place, the mutations themselves, the types of mutators and the test suite used to assess mutation score.

5. From the output above, we update the test cases

The test cases are not killing all mutants due to not checking `value[index] == MIN` and `value[index] == MAX` conditions in the boundary. A similar problem occurs for `ti == arraySize` in the while loop. `LibraryTest.java` now contains:

```
package com.mcgill.ecse429.tutorial6;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class LibraryTest {

    @Test
    public void allBranchCoverageMinimumTestCaseForReturnAverageTest1() {
        int[] value = {5, 25, 10, 20, -999};
        int AS = 5;
        int min = 10;
        int max = 20;
        double average = Library.returnAverage(value, AS, min, max);
        assertEquals(15, average, 0.1);
    }

    @Test
    public void allBranchCoverageMinimumTestCaseForReturnAverageTest2() {
        int[] value = {};
        int AS = 0;
        int min = 10;
        int max = 20;
        double average = Library.returnAverage(value, AS, min, max);
        assertEquals(-999.0, average, 0.1);
    }

}
```

6. Rerun pit mutation with `gradle build pitest` and reopen the outputted report `index.html`

## Library.java

```
1 package com.mcgill.ecse429.tutorial6;
2
3 public class Library {
4
5     public static double returnAverage(int value[], int arraySize, int MIN, int MAX) {
6         int index, ti, tv, sum;
7         double average;
8         index = 0; ti = 0; tv = 0; sum = 0;
9         while (ti < arraySize && value[index] != -999) {
10            ti++;
11            if (value[index] >= MIN && value[index] <= MAX) {
12                tv++;
13                sum += value[index];
14            }
15            index++;
16        }
17        if (tv > 0)
18            average = (double) sum/tv;
19        else
20            average = (double) -999;
21        return average;
22    }
23 }
```

### Mutations

```
9 1. changed conditional boundary → KILLED
2. negated conditional → KILLED
3. negated conditional → KILLED
10 1. Changed increment from 1 to -1 → SURVIVED
1. changed conditional boundary → KILLED
11 2. changed conditional boundary → KILLED
3. negated conditional → KILLED
4. negated conditional → KILLED
12 1. Changed increment from 1 to -1 → KILLED
13 1. Replaced integer addition with subtraction → KILLED
15 1. Changed increment from 1 to -1 → KILLED
17 1. changed conditional boundary → KILLED
2. negated conditional → KILLED
18 1. Replaced double division with multiplication → KILLED
21 1. replaced return of double value with -(x + 1) for com/mcgill/ecse429/tutorial6/Library::returnAverage → KILLED
```

### Active mutators

- INCREMENTS\_MUTATOR
- VOID\_METHOD\_CALL\_MUTATOR
- RETURN\_VALS\_MUTATOR
- MATH\_MUTATOR
- NEGATE\_CONDITIONALS\_MUTATOR
- INVERT\_NEGS\_MUTATOR
- CONDITIONALS\_BOUNDARY\_MUTATOR

### Tests examined

- com.mcgill.ecse429.tutorial6.LibraryTest.allBranchCoverageMinimumTestCaseForReturnAverageTest1(com.mcgill.ecse429.tutorial6.LibraryTest) (7 ms)
- com.mcgill.ecse429.tutorial6.LibraryTest.mutationTestCaseForReturnAverageTest1(com.mcgill.ecse429.tutorial6.LibraryTest) (0 ms)
- com.mcgill.ecse429.tutorial6.LibraryTest.allBranchCoverageMinimumTestCaseForReturnAverageTest2(com.mcgill.ecse429.tutorial6.LibraryTest) (1 ms)

7. After the second run, we see that the line `ti++;` inside the while loop is useless and a code smell!

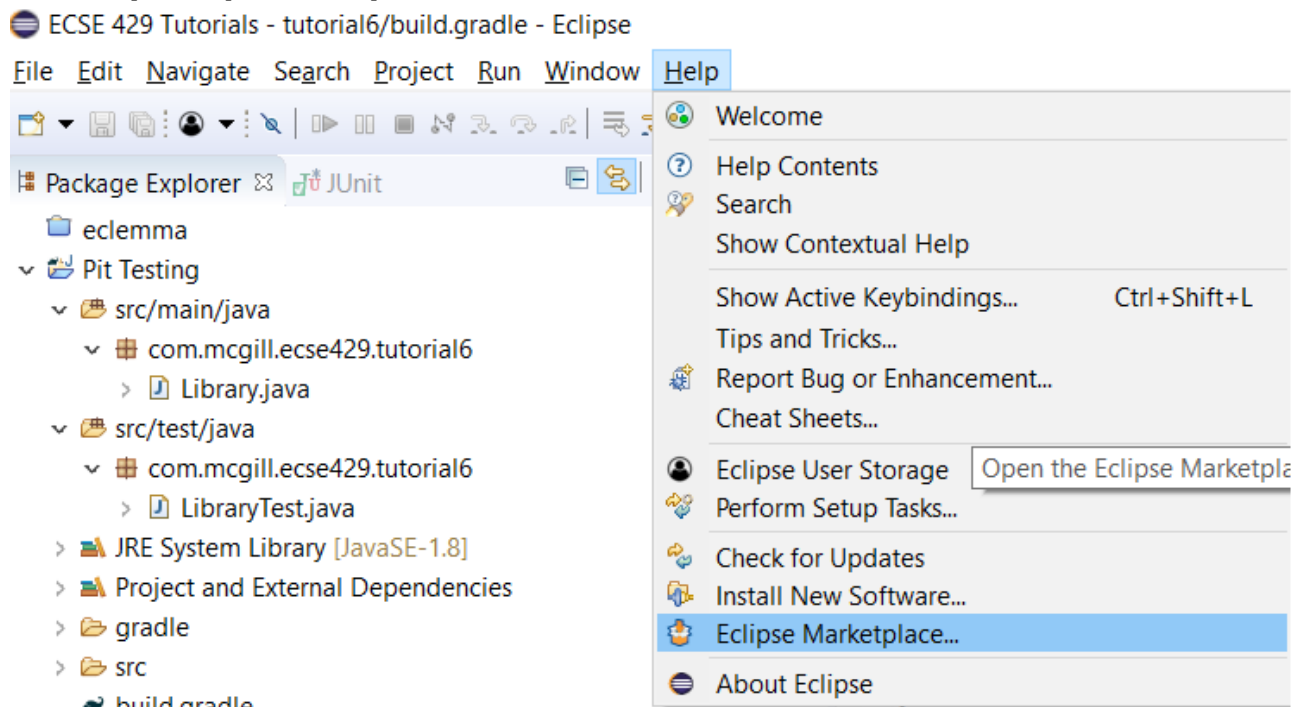
## 6.3. Configuring PIT testing

1. To understand the different mutants, you can go [here](#).
2. You can customize the plugin in the `build.gradle` file using all the parameters for the command line.  
Reference: <http://pitest.org/quickstart/commandline/>
3. For example, we can specify the mutators we want in `build.gradle`:

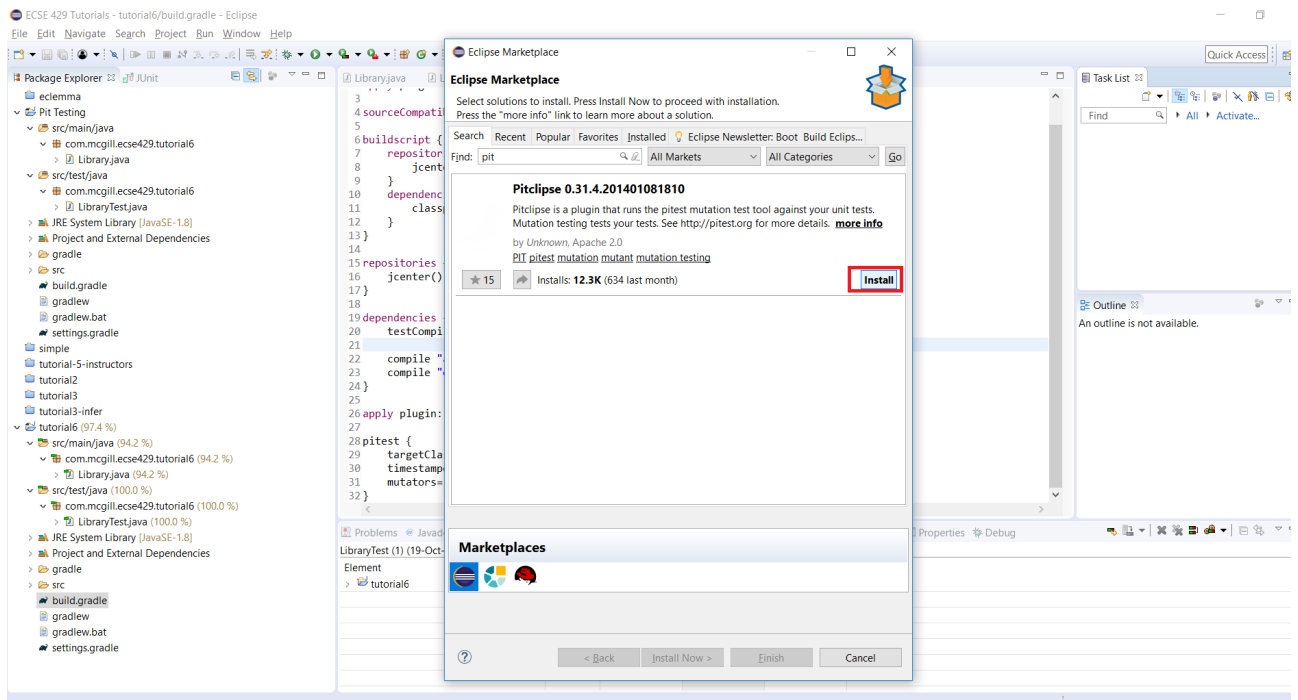
```
...
pitest {
    targetClasses = ["com.mcgill.ecse429.tutorial6*"]
    timestampedReports = true
    mutators=['NEGATE_CONDITIONALS', 'CONDITIONALS_BOUNDARY']
}
...
```

## 6.4. Using PIT in Eclipse

1. Go to *Help > Eclipse Marketplace*



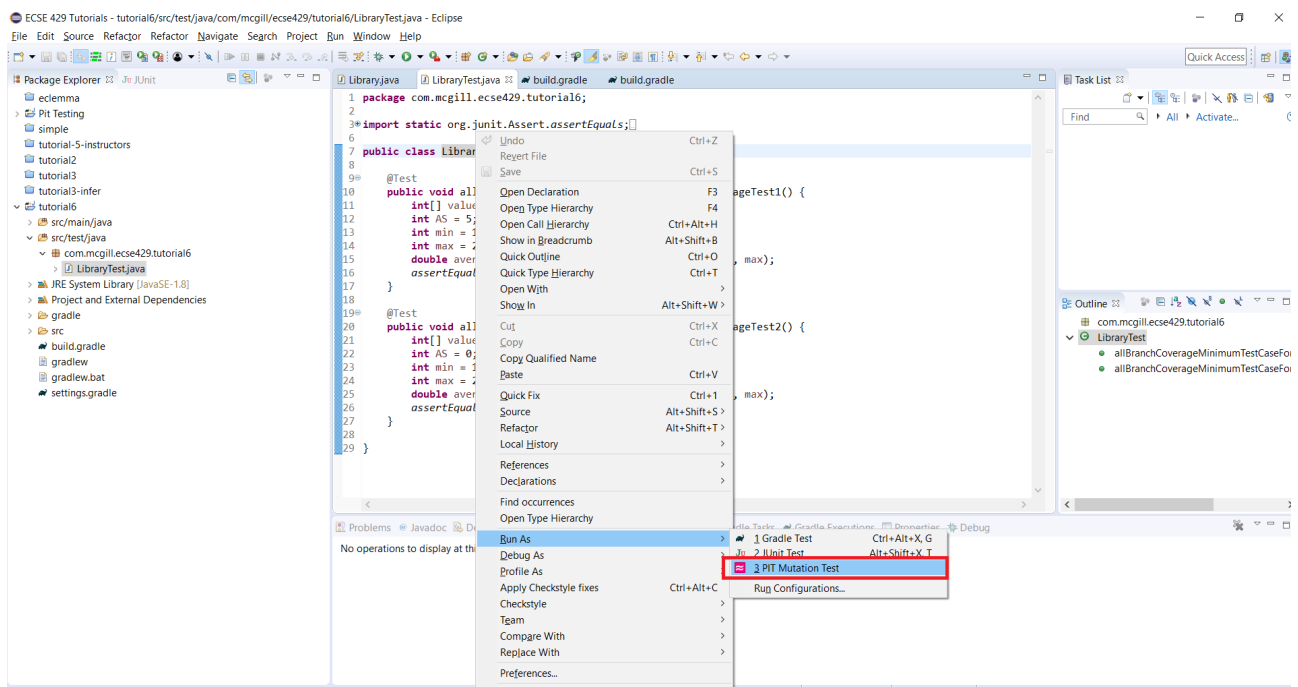
2. Type *pit* in the search box and find *Pitclipse*



3. Restart your Eclipse after the installation is successful

4. You can now execute the tests in `LibraryTest` class by selecting *Pit Mutation Test* from the available run configurations.

**NOTE** If you get a `NullPointerException` for the first run, try re-running it.



5. Check the output in the console for where the report is generated.

```
ECSE 429 Tutorials - tutorial6/src/test/java/com/mcgill/ecse429/tutorial6/LibraryTest.java - Eclipse
File Edit Source Refactor Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console Progress Coverage Gradle Tasks Gradle Executions Properties Debug PIT Mutations PIT Summary
<terminated> LibraryTest (2) [PIT Mutation Test] C:\Program Files\Java\jdk1.8.0_161\bin\javaw.exe (Oct 19, 2018, 7:44:59 PM)
-----
- Statistics
-----
>> Generated 15 mutations Killed 14 (93%)
>> Ran 16 tests (1.07 tests per mutation)
-----
- Mutators
-----
> org.pitest.mutationtest.engine.gregor.mutators.ConditionalBoundaryMutator
>> Generated 4 Killed 4 (100%)
> KILLED 4 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
-----
> org.pitest.mutationtest.engine.gregor.mutators.IncrementsMutator
>> Generated 3 Killed 2 (67%)
> KILLED 2 SURVIVED 1 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
-----
> org.pitest.mutationtest.engine.gregor.mutators.ReturnValsMutator
>> Generated 1 Killed 1 (100%)
> KILLED 1 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
-----
> org.pitest.mutationtest.engine.gregor.mutators.MathMutator
>> Generated 2 Killed 2 (100%)
> KILLED 2 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
-----
> org.pitest.mutationtest.engine.gregor.mutators.NegateConditionalsMutator
>> Generated 5 Killed 5 (100%)
> KILLED 5 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 0
-----
Sending results: PitResults [htmlResultFile=C:\Users\Mathieu\Documents\ECSE 429 Tutorials\.metadata\.plugins\org.pitest.pitclipse.core\html_results\201810191945\index.html] projects=[tutorial6]
Closing server
Closed
```

**NOTE** The output of the report should be identical to the one we generated with the Gradle plugin

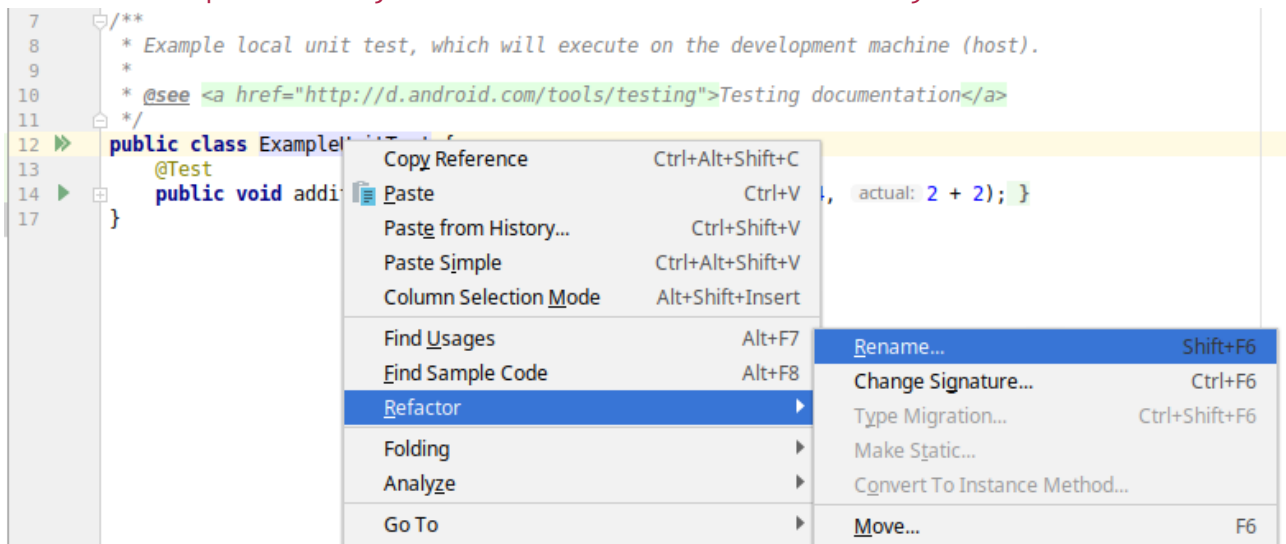
## 7. Integration Testing

## 7.1. Testing Using Mocks

1. Clone (or fork and clone the fork of) the initial project from [this link](#) and import it to Android Studio.
2. Open the project with Android Studio and verify that it builds fine from the IDE (or by running the corresponding Gradle wrapper).
3. Explore the (not too complex) application sources. Start the app and try its features as well.
4. Add the Mockito gradle dependency in `app/build.gradle` for compiling test sources: `testImplementation 'org.mockito:mockito-core:2.23.0'`
5. In the same `build.gradle` file, add to the `android` task

```
testOptions {
    unitTests.returnDefaultValues = true
}
```

6. Locate the `ExampleUnitTest.java` and refactor the class' name to `IntegrationTest`.



7. Add the following code to the class:

```

package ca.mcgill.ecse321.eventregistration;

import com.loopj.android.http.AsyncHttpClient;
import com.loopj.android.http.BlackholeHttpResponseHandler;
import com.loopj.android.http.RequestParams;

import org.junit.Before;
import org.junit.Test;
import org.mockito.Mock;
import org.mockito.Mockito;

public class IntegrationTest {

    @Mock
    AsyncHttpClient httpClient;

    HttpUtils httpUtils;

    @Before
    public void setUp(){
        httpUtils = new HttpUtils(httpClient);
    }

    @Test
    public void testHttpPostCalled() {
        String url = "participants/";
        RequestParams params = new RequestParams();
        BlackholeHttpResponseHandler handler = new BlackholeHttpResponseHandler();

        // Call the SUT
        httpUtils.post(url, params, handler);

        // Verify interactions
        Mockito.verify(httpClient).post(HttpUtils.getBaseUrl() + url, params,
handler);
    }
}

```

8. Run the test as JUnit test (failure is expected). Try understanding the error message!
9. Add the following annotation to the test class.



```
// ... other imports
import org.junit.runner.RunWith;
import org.mockito.junit.MockitoJUnitRunner;

@RunWith(MockitoJUnitRunner.class)
public class IntegrationTest {
    // ...
}
```

## 7.2. Local UI Tests

1. Add the following to the module's `build.gradle` file within the `android` task

```
android {
    testOptions {
        unitTests {
            includeAndroidResources = true
        }
    }
    // ... other settings
}
```

2. In this same Gradle file, add/verify dependencies

```
testImplementation 'junit:junit:4.12'
testImplementation 'org.mockito:mockito-core:2.23.0'
testImplementation 'org.robolectric:robolectric:4.0'
```

3. Create a new test class `LocalUITest` within the `src/test/java` directory in package `ca.mcgill.ecse321.eventregistration`

```

package ca.mcgill.ecse321.eventregistration;

import com.loopj.android.http.AsyncHttpClient;
import com.loopj.android.http.RequestParams;
import com.loopj.android.http.ResponseHandlerInterface;

import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.MockitoJUnit;
import org.mockito.junit.MockitoRule;
import org.robolectric.Robolectric;
import org.robolectric.RobolectricTestRunner;

@RunWith(RobolectricTestRunner.class)
public class LocalUITest {

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

    @Mock
    public AsyncHttpClient httpClient;

    @Test
    public void aTest() {
        MainActivity mainActivity = Robolectric.setupActivity(MainActivity.class);
        mainActivity.setHttpUtils(new HttpUtils(httpClient));
        mainActivity.addParticipant(null);

        Mockito.verify(httpClient).post(Mockito.anyString(), Mockito.<RequestParams>any(), Mockito.<ResponseHandlerInterface>any());
    }
}

```

#### 4. Run it as a local test.

## 7.3. Android Instrumentation Tests

With Android, we have previously created **Local tests** that run on your local machine only. In this tutorial we show how you can create **Instrumented tests**, unit tests that run on an Android device or emulator. See more about the difference between local and instrumentation tests at: <https://developer.android.com/training/testing/unit-testing/>.

### 7.3.1. Project dependencies and setup

This section takes configurations from the [Android documentation](#). You can follow the steps below by reusing the example project from [the first step of the previous section](#).

1. Add the used libraries to `<module-name>/src/main./AndroidManifest.xml` **within** the `<application>...</application>` tags

```
<uses-library android:name="android.test.base"
    android:required="false" />
<uses-library android:name="android.test.runner"
    android:required="false" />
<uses-library android:name="android.test.mock"
    android:required="false" />
```

The next few steps will update the module's `build.gradle` file (`<module-name>/build.gradle` file).

1. Specify the used libraries within the `android` task as well.

```
android {
    // ... other settings
    useLibrary 'android.test.runner'
    useLibrary 'android.test.base'
    useLibrary 'android.test.mock'
    // ... other settings
}
```

2. Add the following instrumentation testing dependencies.

```
dependencies {
    // ... other dependencies
    // Core library
    androidTestImplementation 'androidx.test:core:1.0.0'
    // AndroidJUnitRunner and JUnit Rules
    androidTestImplementation 'androidx.test:runner:1.1.0'
    androidTestImplementation 'androidx.test:rules:1.1.0'
    // Assertions
    androidTestImplementation 'androidx.test.ext:junit:1.0.0'
    androidTestImplementation 'androidx.test.ext:truth:1.0.0'
    androidTestImplementation 'com.google.truth:truth:0.42'
    // Espresso dependencies
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-contrib:3.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-intents:3.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-accessibility:3.1.0'
    androidTestImplementation 'androidx.test.espresso:espresso-web:3.1.0'
    // Add these for applications having significant library dependencies
    androidTestImplementation 'com.google.dexmaker:dexmaker:1.2'
    // Android-specific Mockito
    androidTestImplementation 'com.google.dexmaker:dexmaker-mockito:1.2:'
}
}
```

#### NOTE

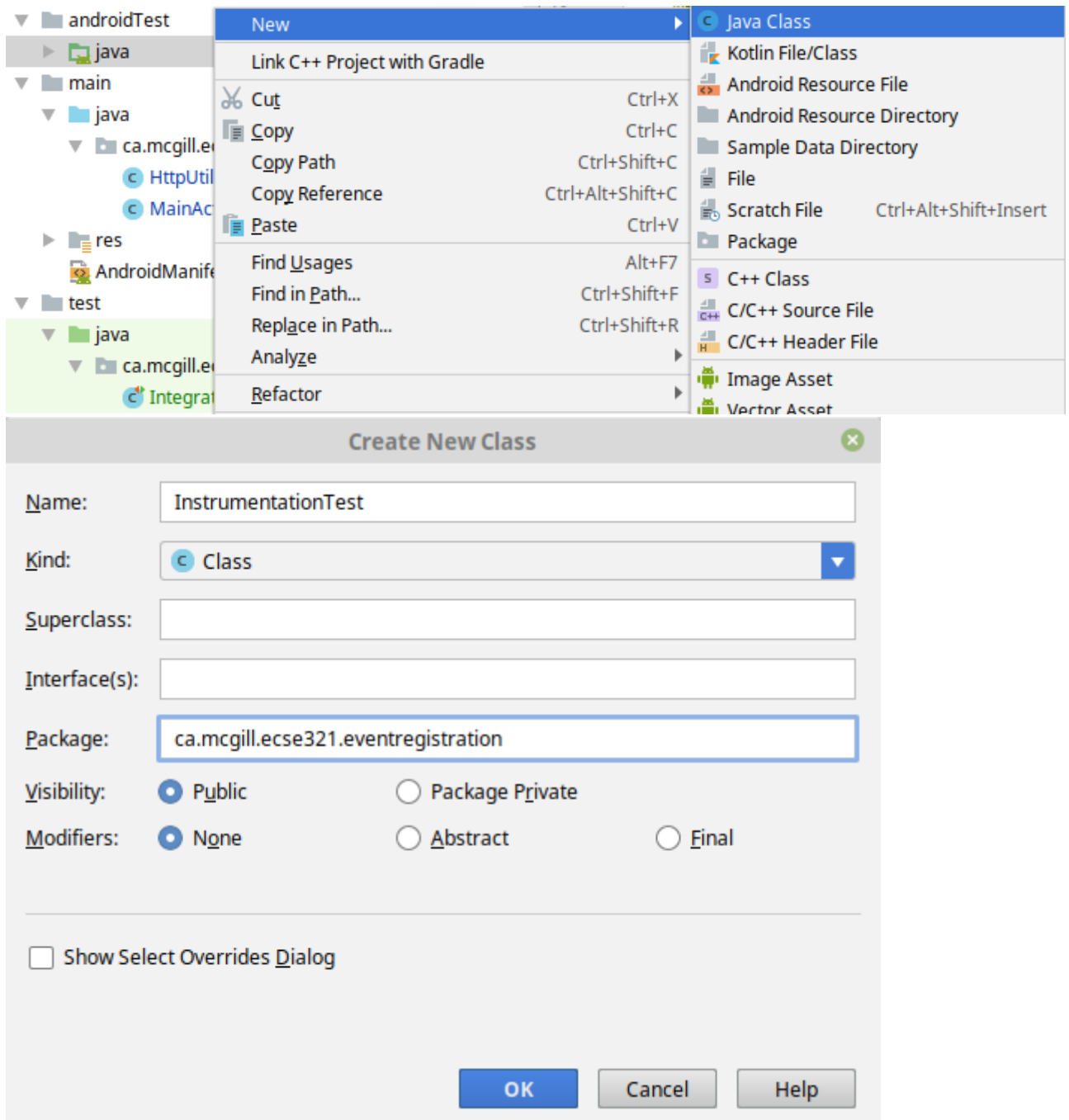
For instrumentation tests only Mockito 1.9.x can be used currently (via the dexmaker dependency).

3. Update the `android` task as shown below. You may need to download the new SDK version.

```
android {
    compileSdkVersion 28
    // ...
    defaultConfig {
        minSdkVersion 18
        targetSdkVersion 28
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    // ...
}
```

### 7.3.2. Creating an instrumentation test

1. Create the `<module-name>/src/androidTest/java` folder structure. This is the default location where the framework will look for instrumentation tests.
2. Create a new test class `InstrumentationTest` under the same package name as the package name of the main activity.



3. Add the imports and annotations as follows

```

package ca.mcgill.ecse321.eventregistration;

import android.widget.Button;
import android.widget.ListView;

import com.loopj.android.http.AsyncHttpClient;
import com.loopj.android.http.JsonHttpResponseHandler;
import com.loopj.android.http.RequestParams;
import com.loopj.android.http.ResponseHandlerInterface;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.invocation.InvocationOnMock;
import org.mockito.runners.MockitoJUnitRunner;
import org.mockito.stubbing.Answer;

import androidx.test.filters.SmallTest;
import androidx.test.rule.ActivityTestRule;

import static com.google.common.truth.Truth.assertThat;

@RunWith(MockitoJUnitRunner.class)
@SmallTest
public class InstrumentationTest {

}

```

**NOTE**

If you are not planning on using Mockito with instrumentation tests, you can replace the `@RunWith(MockitoJUnitRunner.class)` annotation with `@RunWith(AndroidJUnit4.class)`.

4. Within this class, create a private class that passes the JSON answer for a HTTP request via the handler (JSON Handler) extracted from the method call

```

private class ParticipantsAnswer implements Answer<JSONArray> {
    @Override
    public Object answer(InvocationOnMock invocation) throws JSONException {
        Object[] arguments = invocation.getArguments();
        JsonHttpResponseHandler h = (JsonHttpResponseHandler) arguments[2];
        h.onSuccess(200,null,new JSONArray()
            .put(new JSONObject().put("name", "P1"))
            .put(new JSONObject().put("name", "P2"))
            .put(new JSONObject().put("name", "P3")));
        return null;
    }
}

```

##### 5. Add fields to the `InstrumentationTest` class

```

@Mock
AsyncHttpClient httpClient;

@Rule
public ActivityTestRule<MainActivity> activityTestRule = new
ActivityTestRule<>(MainActivity.class);

```

##### 6. Add the implementation to the test case

```

@Test
public void aTest() throws Throwable {
    final MainActivity mainActivity = activityTestRule.getActivity();
    mainActivity.setHttpUtils(new HttpUtils(httpClient));

    Mockito.when(httpClient.get(Mockito.anyString(),Mockito.<RequestParams>anyObject(),
Mockito.<ResponseHandlerInterface>anyObject())).thenReturn(new
ParticipantsAnswer());

    final ListView participantList =
mainActivity.findViewById(R.id.participant_list);
    final int numberOfParticipants = participantList.getAdapter().getCount();

    assertThat(numberOfParticipants).isEqualTo(0);

    Button button = (Button) mainActivity.findViewById(R.id.button);
    button.performClick();

    int numberOfParticipants2 = participantList.getAdapter().getCount();
    assertThat(numberOfParticipants2).isEqualTo(3);
}

```

7. After right clicking on the class you can run the instrumentation test. Select the virtual device to run the tests on.
8. The tests failed. Try understanding the error message!
9. Fix the tests by including the last part of the test in `runOnUiThread()`

```
activityTestRule.runOnUiThread(new Runnable() {
    @Override
    public void run() {
        Button button = (Button) mainActivity.findViewById(R.id.button);
        button.performClick();
        int numberOfParticipants2 = participantList.getAdapter().getCount();
        assertThat(numberOfParticipants2).isEqualTo(3);
    }
});
```

10. Finally, exploit the *Fluent API* of Mockito and clean up your code. Import all static methods with `static import`!

## 8. Android User Interface Tests

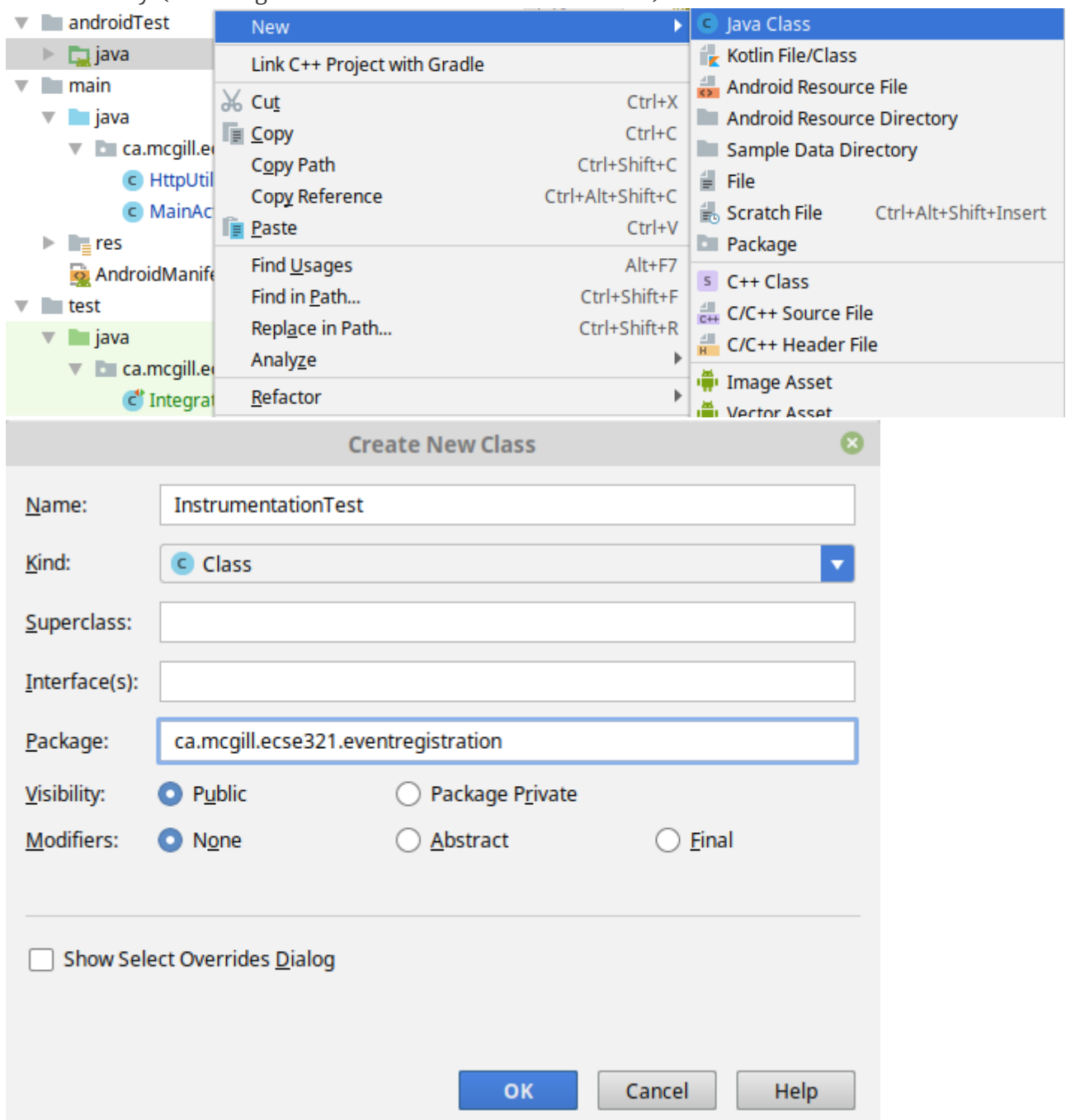


This tutorial shows how you can create UI tests using instrumentation testing with the [Robotium framework](#).

1. Clone the initial Android project from [here](#) and import it to Android Studio.
2. Add the following dependency to the `app` module

```
dependencies {  
    // ... other dependencies  
    implementation 'com.jayway.android.robotium:robotium-solo:5.2.1'  
}
```

3. Create the `<module-name>/src/androidTest/java` folder structure.
4. Create a new test class `RobotiumTest` under the same package name as the package name of the main activity. (In the figure the class name does not match.)



## 5. Add the initial implementation of the Robotium test class

```
package ca.mcgill.ecse321.eventregistration;

import android.test.ActivityInstrumentationTestCase2;
import android.widget.EditText;

import com.robotium.solo.Solo;

public class RobotiumTest extends ActivityInstrumentationTestCase2<MainActivity> {
    private Solo solo;

    private static final String LAUNCHER_ACTIVITY_FULL_CLASSNAME =
"ca.mcgill.ecse321.eventregistration.MainActivity";

    private static Class<?> launcherActivityClass;

    static {
        try {
            launcherActivityClass =
Class.forName(LAUNCHER_ACTIVITY_FULL_CLASSNAME);
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    @SuppressWarnings("unchecked")
    public RobotiumTest() throws ClassNotFoundException {
        super((Class<MainActivity>) launcherActivityClass);
    }

    private String participantName;
    private String testParticipantName = "TestParticipant";

    @Override
    public void setUp() throws Exception {
        super.setUp();
        solo = new Solo(getInstrumentation());

        int uniqueId = 1586; // Ensure uniqueness of name
        participantName = "Participant" + uniqueId;
        getActivity();
    }

    @Override
    public void tearDown() throws Exception {
        solo.finishOpenedActivities();
        super.tearDown();
    }
}
```

6. Add a test case to see if registration succeeds

```
public void testRegisterSucceeds() throws InterruptedException {
    solo.waitForActivity("MainActivity", 2000);

    EditText editText = solo.getEditText("Who?");
    solo.enterText(editText, participantName);
    solo.clickOnText(solo.getString(R.string.newparticipant_button));

    boolean errorTextFound = solo.waitForText("exception", 1, 5000);
    assertFalse(errorTextFound);
}
```

7. Create a participant named *TestParticipant* by issuing a POST request to <https://eventregistration-backend-123.herokuapp.com/participants/TestParticipant>

```
# Add a participant called TestParticipant to the database
curl -X POST https://eventregistration-backend-
123.herokuapp.com/participants/TestParticipant

# Make sure participant was added by issuing a get request
curl GET https://eventregistration-backend-123.herokuapp.com/participants/
```

8. Add a test case to see if registration fails if the participant is already in the database

```
public void testRegisterFails() throws InterruptedException {
    solo.waitForActivity("MainActivity", 2000);

    EditText editText = solo.getEditText("Who?");
    solo.enterText(editText, testParticipantName);
    solo.clickOnText(solo.getString(R.string.newparticipant_button));

    boolean errorTextFound = solo.waitForText("exception", 1, 5000);
    assertTrue(errorTextFound);
}
```

9. Add a test case to see if the list of participant contains the existing participant "TestParticipant"

```
public void testListRefreshes() {
    solo.clickOnText(solo.getString(R.string.refresh_participant_list));

    boolean textFound = solo.waitForText(testParticipantName, 1, 5000, true);

    assertTrue(textFound);
}
```

10. Start the Robotium Instrumentation Test on selected device

### **8.1. Running Android Emulator on Travis-CI**

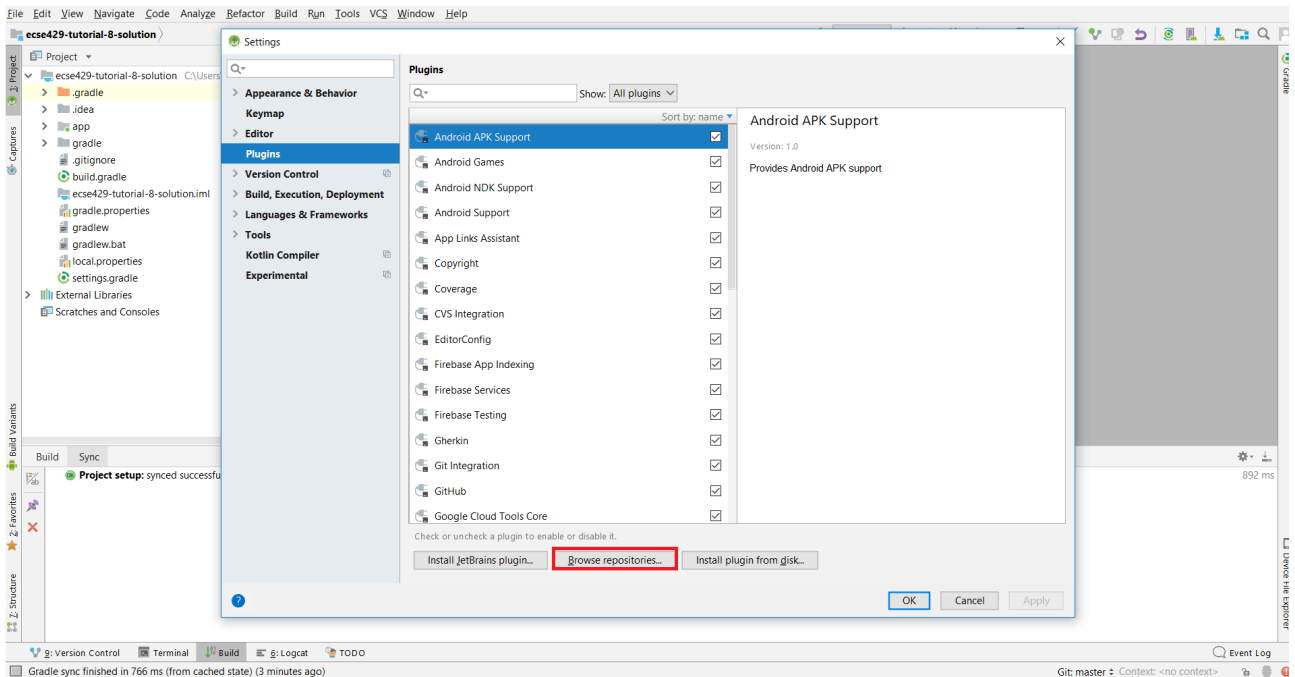
Instrumentation tests require a runtime Android. This means Travis needs to run one instance during the build to successfully execute tests. You can use the following example configuration for getting started on how to run an emulator on Travis-CI: <https://gist.github.com/harmittaa/7d3c51041ffd0e54cda9807e95593309>

## **9. Acceptance Testing**

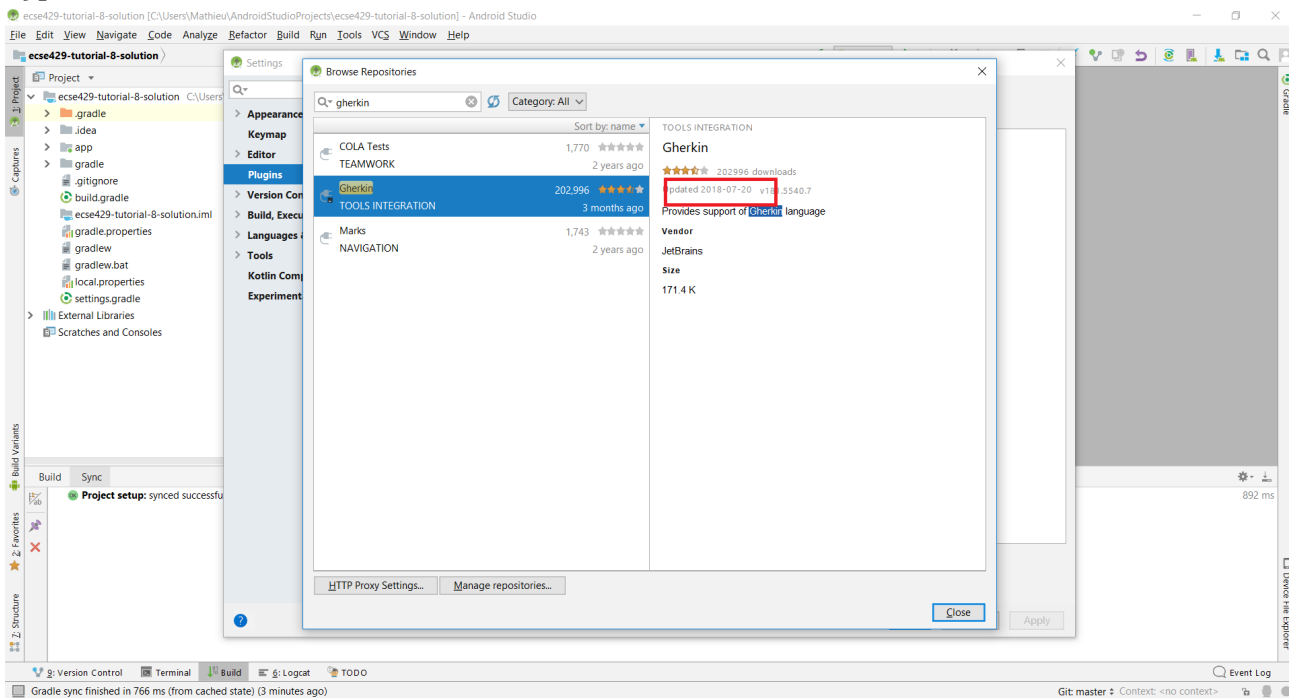
This tutorial shows how you can write acceptance test using the [Cucumber tool](#) in a Behaviour-Driven Development approach.

1. Clone the initial Android project from [here](#) and import it to Android Studio
2. Install the [Gherkin](#) plugin for Android Studio

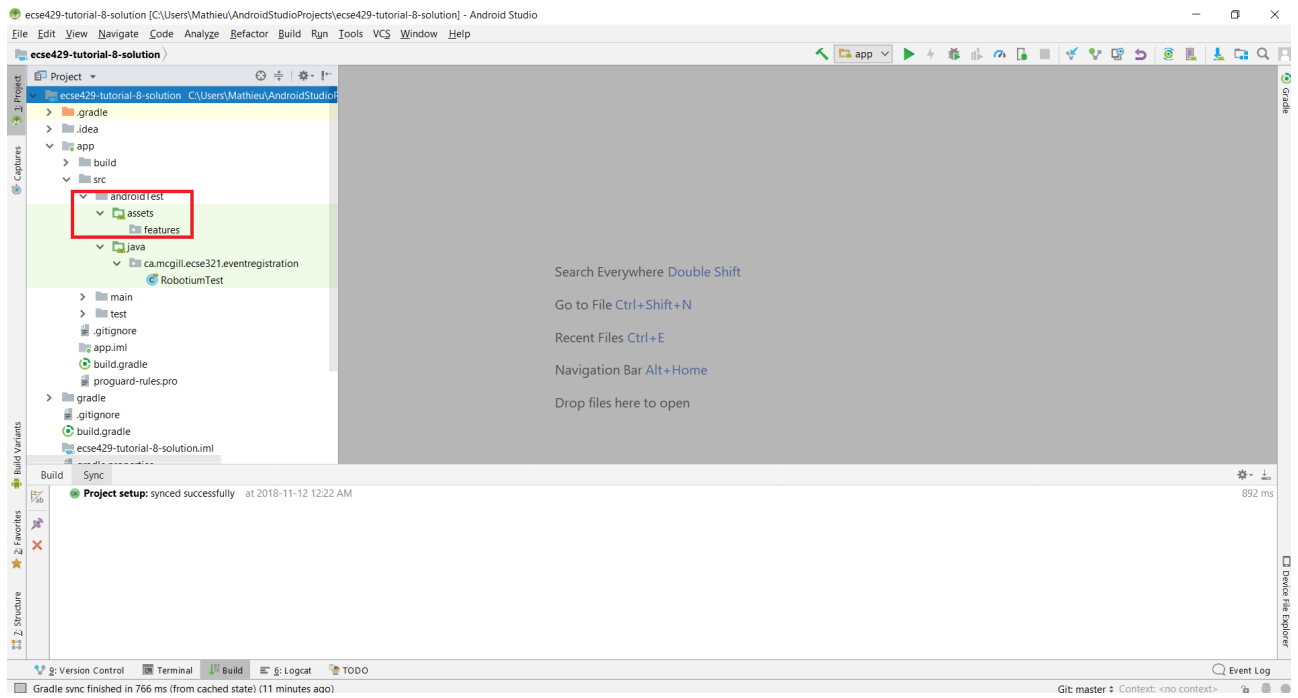
Click on *File > Settings > Plugins > Browse Repositories*



3. Type *Gherkin* in search box, install it and restart Android Studio.



4. Under `androidTest` folder, create a new folder structure `androidTest/assets/features`



5. Add a new file called `addParticipant.feature` in the features folder

Feature: Add a participant

Add a test participant to the main activity

Scenario Outline: Add new participant to the main activity

Given I have a MainActivity

When I input username <username> in text field 'Who?'

And I press 'Add Participant' button

When I click on the 'Refresh Participant List' button

Then I should see <see> in the list

Examples:

username	see
some_random_guy_1	some_random_guy_1
some_random_guy_2	some_random_guy_2
some_random_guy_3	some_random_guy_3

6. Add the following lines to the app's `build.gradle` file within the `android` tags

```

android {
    // ...
    defaultConfig {
        // ...
        testInstrumentationRunner
"ca.mcgill.ecse321.eventregistration.test.Instrumentation"
        // ...
    }
    // ...

    sourceSets {
        androidTest {
            assets {
                assets.srcDirs = ['src/androidTest/assets']
            }
            java {
                java.srcDirs = ['src/androidTest/java']
            }
        }
    }
    // ...

    compileOptions {
        // ...
        sourceCompatibility = '1.8'
        targetCompatibility = '1.8'
        // ...
    }
}
}

```

7. Add the following lines in **dependencies** section

```

dependencies {
    // ...

    //Runner
    androidTestImplementation( 'com.android.support.test:runner:0.4.1' ){
        exclude module: 'junit'
    }
    androidTestImplementation 'io.cucumber:cucumber-junit:3.0.2'
    androidTestImplementation group: 'io.cucumber', name: 'cucumber-android',
version: '3.0.2'
    androidTestImplementation 'io.cucumber:cucumber-picocontainer:3.0.2'
    androidTestImplementation group: 'io.cucumber', name: 'cucumber-jvm', version:
'3.0.2', ext: 'pom'
    androidTestImplementation 'io.cucumber:cucumber-core:3.0.2'
    androidTestImplementation 'io.cucumber:cucumber-jvm-deps:1.0.6'
    // ...
}

```

**NOTE** | For convenience, below is the full specification of the `app.gradle` file

```

apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "ca.mcgill.ecse321.eventregistration"
        minSdkVersion 18
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
"ca.mcgill.ecse321.eventregistration.test.Instrumentation"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
    useLibrary 'android.test.runner'
    useLibrary 'android.test.base'
    useLibrary 'android.test.mock'

    sourceSets {
        androidTest {
            assets {
                assets.srcDirs = ['src/androidTest/assets']
            }
        }
    }
}

```



```

        java {
            java.srcDirs = ['src/androidTest/java']
        }
    }
}
compileOptions {
    sourceCompatibility = '1.8'
    targetCompatibility = '1.8'
}
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    implementation 'com.android.support:design:28.0.0'
    testImplementation 'junit:junit:4.12'

    implementation 'com.loopj.android:android-async-http:1.4.9'

    implementation 'com.jayway.android.robotium:robotium-solo:5.2.1'

    //Runner
    androidTestImplementation( 'com.android.support.test:runner:0.4.1' ){
        exclude module: 'junit'
    }

    androidTestImplementation 'io.cucumber:cucumber-junit:3.0.2'
    androidTestImplementation group: 'io.cucumber', name: 'cucumber-android',
version: '3.0.2'
    androidTestImplementation 'io.cucumber:cucumber-picocontainer:3.0.2'
    androidTestImplementation group: 'io.cucumber', name: 'cucumber-jvm', version:
'3.0.2', ext: 'pom'
    androidTestImplementation 'io.cucumber:cucumber-core:3.0.2'
    androidTestImplementation 'io.cucumber:cucumber-jvm-deps:1.0.6'
}
}

```

8. Sync after making the changes to the `app.gradle` file
9. Delete the current implementation of `RobotiumTest` in the `java` folder
10. Refactor or create a new package named `ca.mcgill.ecse321.eventregistration.test` under `androidTest/java`
11. Create a new class named `Instrumentation` in the `ca.mcgill.ecse321.eventregistration.test` package with the following code

```

package ca.mcgill.ecse321.eventregistration.test;

import android.os.Bundle;

import cucumber.api.android.CucumberInstrumentationCore;
import android.support.test.runner.AndroidJUnitRunner;

public class Instrumentation extends AndroidJUnitRunner {
    private final CucumberInstrumentationCore instrumentationCore = new
CucumberInstrumentationCore(this);

    @Override
    public void onCreate(Bundle arguments) {
        super.onCreate(arguments);
        instrumentationCore.create(arguments);
    }

    @Override
    public void onStart() {
        waitForIdleSync();
        instrumentationCore.start();
    }
}

```

12. Create a new class named `CucumberRunner` in the `ca.mcgill.ecse321.eventregistration.test` package with the following code

```

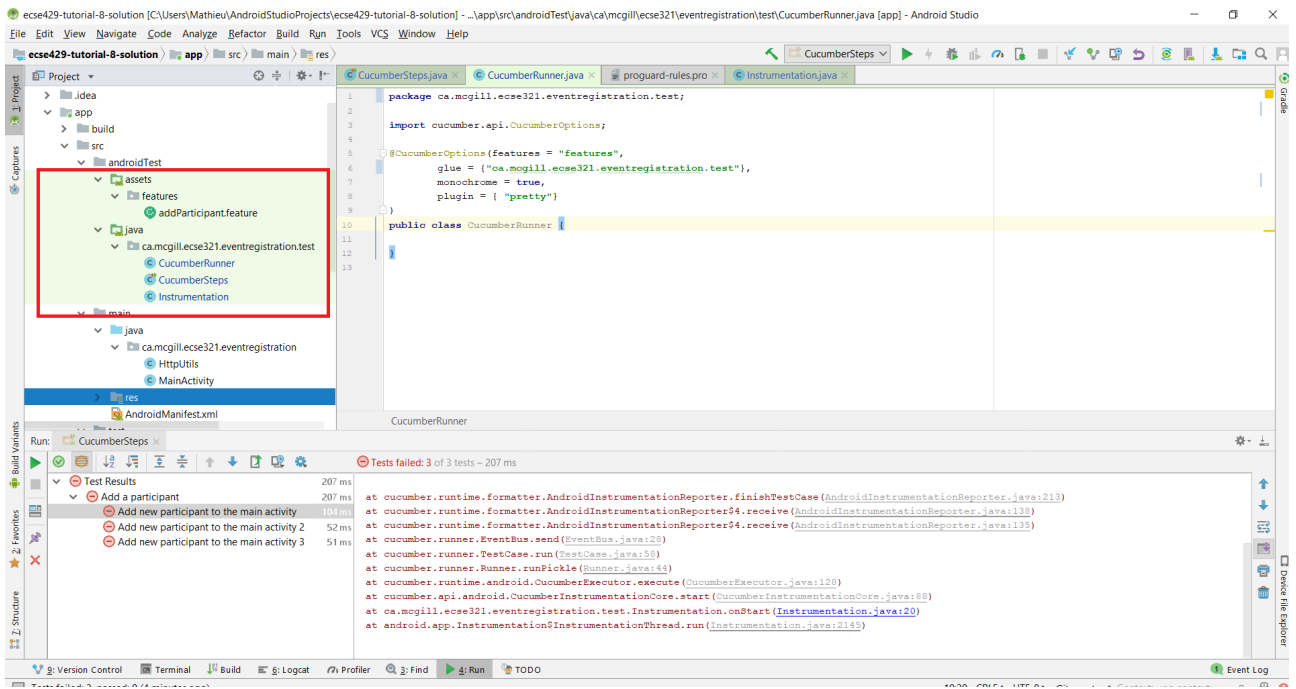
package ca.mcgill.ecse321.eventregistration.test;

import cucumber.api.CucumberOptions;

@CucumberOptions(features = "features",
    glue = {"ca.mcgill.ecse321.eventregistration.test"},
    monochrome = true,
    plugin = { "pretty"}
)
public class CucumberRunner {
}

```

13. Create a new class `CucumberSteps` again in the `ca.mcgill.ecse321.eventregistration.test`. Your file structure should look like



## 14. Add initializing code to `CucumberSteps`

```

package ca.mcgill.ecse321.eventregistration.test;

import android.test.ActivityInstrumentationTestCase2;

import com.robotium.solo.Solo;

import ca.mcgill.ecse321.eventregistration.MainActivity;

public class CucumberSteps extends ActivityInstrumentationTestCase2<MainActivity> {
    private Solo solo;

    private static final String LAUNCHER_ACTIVITY_FULL_CLASSNAME =
"ca.mcgill.ecse321.eventregistration.MainActivity";
    private static Class<?> launcherActivityClass;

    static {
        try {
            launcherActivityClass =
Class.forName(LAUNCHER_ACTIVITY_FULL_CLASSNAME);
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

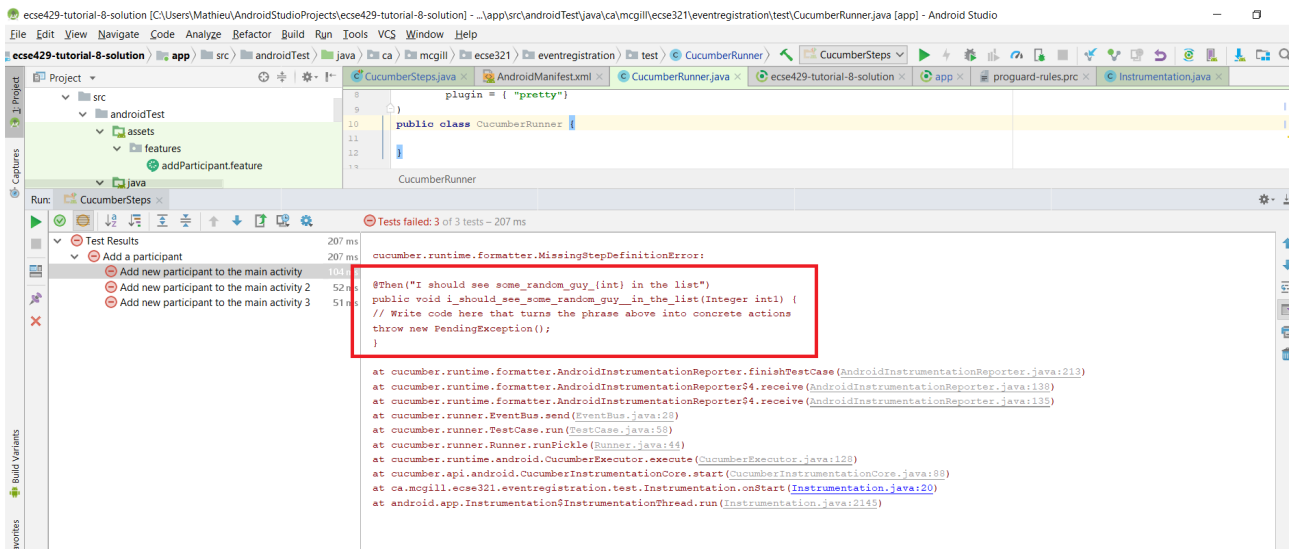
    public CucumberSteps() throws ClassNotFoundException {
        super((Class<MainActivity>) launcherActivityClass);
    }

    @Override
    public void setUp() throws Exception {
        super.setUp();
    }

    @Override
    public void tearDown() throws Exception {
        solo.finishOpenedActivities();
        getActivity().finish();
        super.tearDown();
    }
}

```

15. Run the tests under `CucumberSteps`. Take a look at the logs for the test failures



16. Add the following code to `CucumberSteps` to finish writing the tests

```

// ===== Test Implementation
=====
@Given("I have a MainActivity")
public void i_have_a_MainActivity() throws Exception {
    solo = new Solo(getInstrumentation());
    getActivity();
}

@When("I input username some_random_guy_{int} in text field {string}")
public void i_input_username_some_random_guy__in_text_field(Integer int1, String
string) {
    solo.waitForActivity("MainActivity", 2000);

    String username = "some_random_guy" + int1;

    EditText editText = solo.getEditText(string);
    solo.enterText(editText, username);
}

@When("I press {string} button")
public void i_press_button(String buttonName) {
    solo.waitForActivity("MainActivity", 2000);

    //click on button
    solo.clickOnText(buttonName);

    //make sure no error has been made
    boolean errorTextFound = solo.waitForText("exception", 1, 5000);
    assertFalse(errorTextFound);
}

@When("I click on the {string} button")
public void i_click_on_the_button(String buttonName) {
    solo.waitForActivity("MainActivity", 2000);

    //click on button
    solo.clickOnText(buttonName);
}

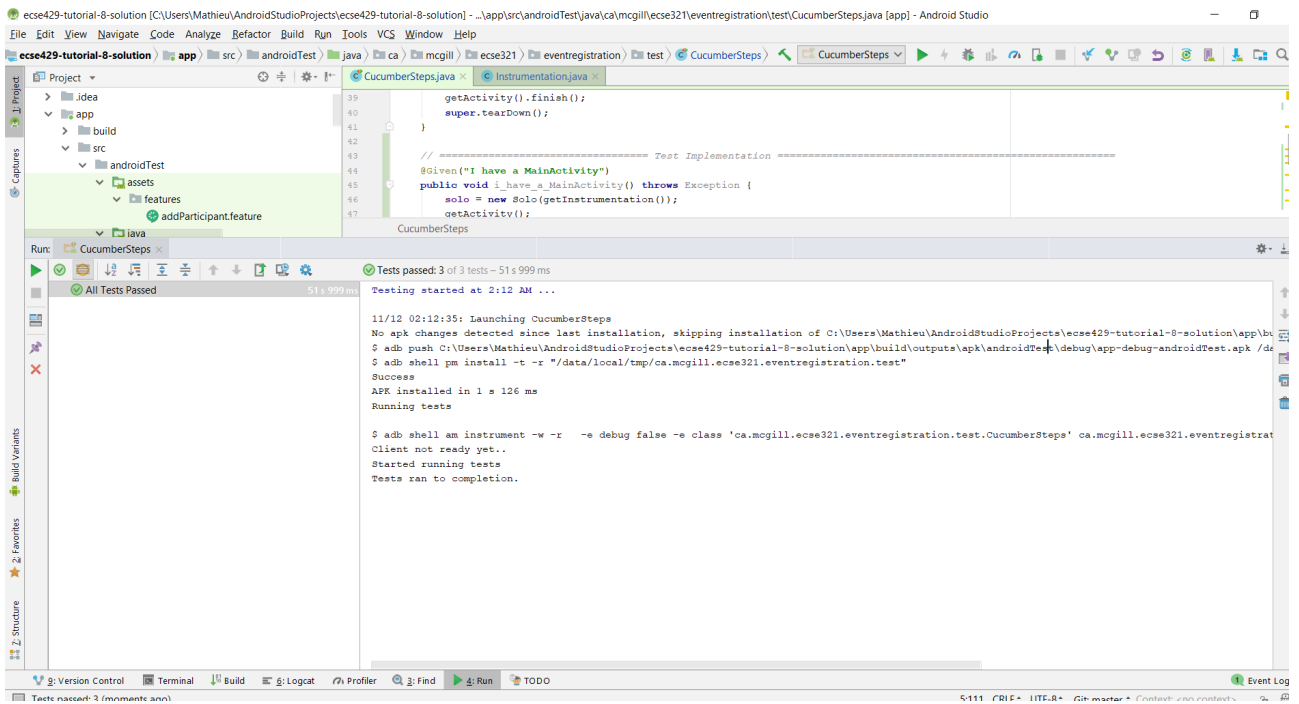
@Then("I should see some_random_guy_{int} in the list")
public void i_should_see_some_random_guy__in_the_list(Integer int1) {
    solo.waitForActivity("MainActivity", 2000);

    String expectedAddedParticipant = "some_random_guy" + int1;

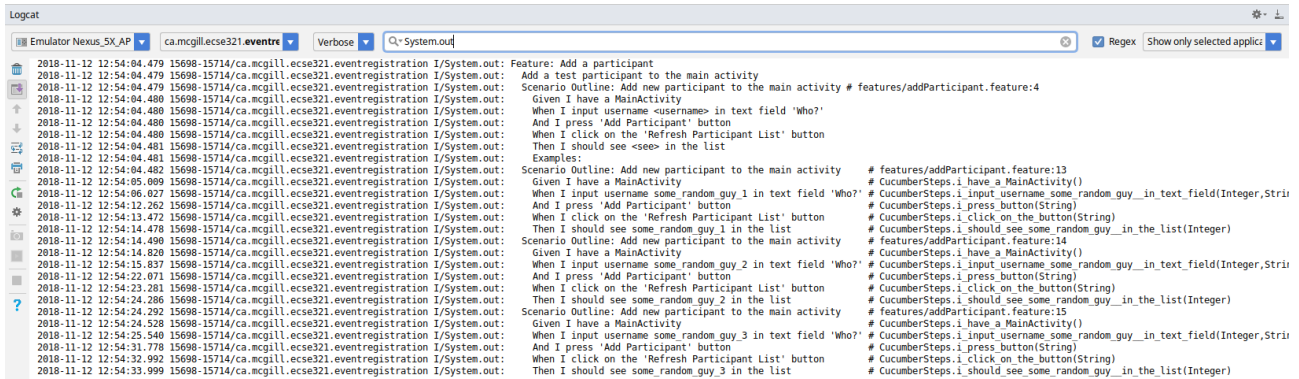
    boolean textFound = solo.waitForText(expectedAddedParticipant, 1, 5000, true);
    assertTrue(textFound);
}

```

17. Finally, rerun the tests under `CucumberSteps`



18. Under the *Logcat* tab you can filter the view to see the output of Cucumber



## 10. Model-based Testing

## 10.1. GraphWalker

For reference, the main documentation for GraphWalker is accessible at <http://graphwalker.github.io/>.

## 10.2. Setting up GraphWalker

1. Download the Latest stable standalone CLI (3.4.2) jar file from [here](#).
2. We assume that the `GRAPHWALKER` environment variable points to the jarfile (e.g., it points to `/home/user/graphwalker-cli-3.4.2.jar`). See its help by running

```
java -jar $GRAPHWALKER --help
```

## 10.3. Creating a test model

We are going to create a test model using the online UMPLE tool.

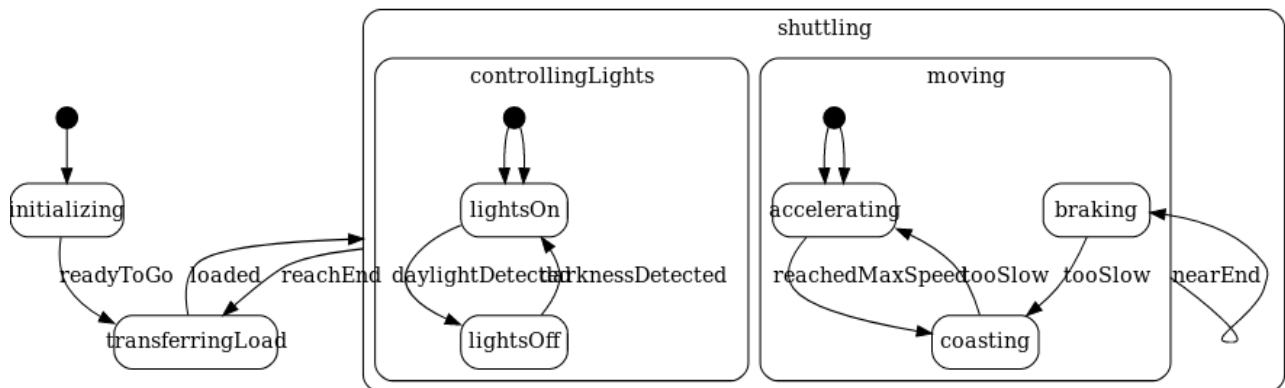
1. Go to <https://cruise.eecs.uottawa.ca/umpleonline/>
2. Insert the following example state machine code describing the possible states of a shuttler and insert it into the text editor (example taken from [here](#))



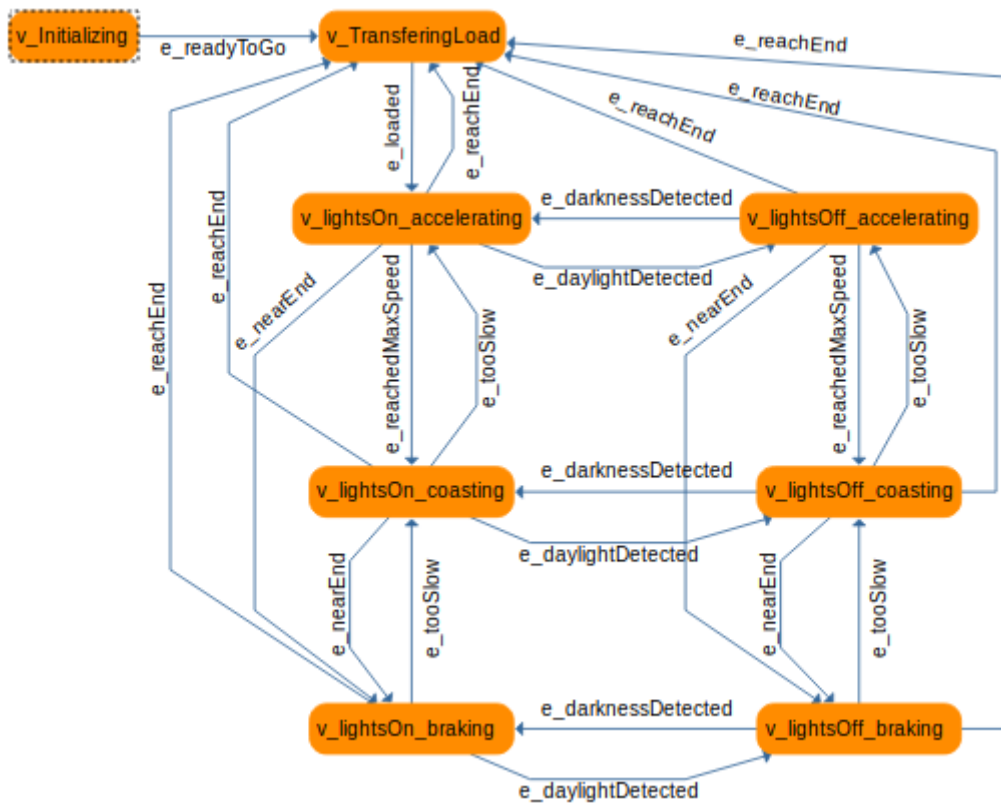
```

class TrackShuttler {
  sm {
    initializing {
      readyToGo -> transferringLoad;
    }
    transferringLoad {
      loaded -> shuttling;
    }
    shuttling {
      reachEnd -> transferringLoad;
      moving {
        nearEnd -> braking;
        accelerating {
          reachedMaxSpeed -> coasting;
        }
        coasting {
          tooSlow -> accelerating;
        }
        braking {
          tooSlow -> coasting;
        }
      }
    }
  }
  ||
  controllingLights {
    lightsOn {
      daylightDetected -> lightsOff;
    }
    lightsOff {
      darknessDetected -> lightsOn;
    }
  }
}

```



3. Take a look what code is generated by the tool based on the statechart!
4. Using the [yEd tool](#), create the flattened version of the state machine



**NOTE** | You can download the resulting graphml [from here](#)

## 10.4. Test generation

1. Use the GraphWalker tool to generate a test sequence based on your criteria. For example, to generate *full vertex coverage* by applying a *random walk* strategy, use

```
java -jar $GRAPHWALKER offline --model shuttler.graphml
"random(vertex_coverage(100))" --start-element v_Initializing
```

Output:

```
{"currentElementName":"v_Initializing"}
{"currentElementName":"e_readyToGo"}
{"currentElementName":"v_TransferringLoad"}
{"currentElementName":"e_loaded"}
{"currentElementName":"v_lightsOn_accelerating"}
{"currentElementName":"e_daylightDetected"}
{"currentElementName":"v_lightsOff_accelerating"}
{"currentElementName":"e_reachEnd"}
{"currentElementName":"v_TransferringLoad"}
{"currentElementName":"e_loaded"}
{"currentElementName":"v_lightsOn_accelerating"}
{"currentElementName":"e_nearEnd"}
{"currentElementName":"v_lightsOn_braking"}
{"currentElementName":"e_daylightDetected"}
{"currentElementName":"v_lightsOff_braking"}
{"currentElementName":"e_tooSlow"}
{"currentElementName":"v_lightsOff_coasting"}
{"currentElementName":"e_tooSlow"}
{"currentElementName":"v_lightsOff_accelerating"}
{"currentElementName":"e_reachEnd"}
{"currentElementName":"v_TransferringLoad"}
{"currentElementName":"e_loaded"}
{"currentElementName":"v_lightsOn_accelerating"}
{"currentElementName":"e_daylightDetected"}
{"currentElementName":"v_lightsOff_accelerating"}
{"currentElementName":"e_darknessDetected"}
{"currentElementName":"v_lightsOn_accelerating"}
{"currentElementName":"e_reachedMaxSpeed"}
{"currentElementName":"v_lightsOn_coasting"}
```

**TIP** Use the [jq tool](#) to prettify the output of the test generator. Pipe the results:

```
GRAPHWALKER_COMMAND | jq -r .currentElementName
```

2. Experiment with other setting, such as creating a test case from one node to another node with A\* traversal! See more [here](#).
3. [Other useful MBT resources](#)