Continuous Integration INRIA

Java Exercises

Vincent Rouvreau - https://sed.saclay.inria.fr [*]

February 28, 2017

Contents

Java Exercises

1 Preamble

To go through this exercise, you will need to install :

1. Git (sudo apt-get install git — sudo yum install git)
2. A JDK (sudo apt-get install openjdk-7-jdk — sudo yum install java-1.7.0-openjdk)
3. Maven (sudo apt-get install maven — sudo yum install apache-maven)

2 Unit testing

2.1 Local setup

Clone the git repository from INRIA forge and create your own branch

First, you will create your personal `git` repository on the INRIA forge as a branch of the main repository for the `TPCISedSaclay` project:

- Go to https://gforge.inria.fr/projects/tpcisedsaclay
- Click on the **CODE SOURCE** or **SCM** tab
- Click on **Request a personal repository**

[*]This practical tutorial has been originally written by Maurice Brémond, Gaëtan Harter and David Parsons from SED INRIA Rhône-Alpes. I would like to thank them for their help and support.

- Back to the **SCM** tab, look for the command to access your personal repository

WARNING : do not use the anonymous access (containing anon-scm)

Then on a terminal, clone the content of your personal `git` repository. The correct command should look like this:

```
git clone \
  git+ssh://<yourforgelogin>@scm.gforge.inria.fr/gitroot/
  tpcisedsaclay/users/<yourforgelogin>.git
cd <yourforgelogin>/cxx
```

Project file tree

You should have retrieved the following file tree:

```
<yourforgelogin>/java
|_ pom.xml
|_ src
   |_ main
   |  |_ java
   |     |_ fr
   |        |_ inria
   |           |_ sed
   |              |_ Sphere.java
   |_ tst
      |_ java
         |_ fr
            |_ inria
               |_ sed
                  |_ TestSphere.java
```

The project is made up of :

- A Maven **Project Object Model** file named `pom.xml`
- A file `Sphere.java` implementing the class `Sphere`
- A file `SphereTest.java` implementing its test class `SphereTest`.

Check that you can build the project

```
$ pwd
yourpath/<yourforgelogin>/java
$ mvn package
```

You should see these lines (among others) :

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

Testing your code locally

Run the (single) test

```
$ mvn test
```

You should see something like

```
-------------------------------------------------------
T E S T S
-------------------------------------------------------
Running fr.inria.sed.SphereTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.069 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

2.2 Jenkins setup

Log into the project's Jenkins instance

1. Connect to the INRIA Continuous Integration Web portal : https://ci.inria.fr/.
2. Log in and click **Dashboard** in the top menu
3. As you have been added to project **TPCISedSaclay**, click on the **Jenkins** button. You may be prompted to log into `Jenkins`, use the same login/passwd as for the `ci.inria.fr` portal.

Running your first test with Jenkins:

- From our Jenkins dashboard page, click **New Item** in the menu on the left
- Provide a name (`<yourforgelogin>` for instance) for this new item (avoid spaces since it is likely to lead to errors) and select **Maven project**.

`Git` configuration :

- In the new item's configuration page (which you will be redirected to after clicking **OK**), choose **Git** as your **Source Code Manager**
- Copy the anonymous URL to your personal repository into the **Repository URL** field:

```
https://scm.gforge.inria.fr/anonscm/git/tpcisedsaclay/users/<yourforgelogin>.git
```

An important step for the continuous integration setup is the build trigger. A simple option is to choose to build periodically : this is suitable for some nightly or weekly tests that may be time consuming and are not meant to be launched after each commit, but this should be avoided for short periods.
In our case, we want the results to be displayed as soon as possible, so we choose to launch the build after a `post-commit` hook [1]:

- Click on **Poll SCM**
- In the **Schedule** field, cut and paste:

```
# Leave empty. We don't poll periodically, but need
# polling enabled to let HTTP trigger work
```

Click on **Save** button.

create the post-commit hook on the INRIA forge server

For this, you can copy the following file:

```
$ ssh <yourforgelogin>@scm.gforge.inria.fr
$ cp /gitroot/tpcisedsaclay/users/vrouvrea.git/hooks/post-receive \
    /gitroot/tpcisedsaclay/users/<yourforgelogin>.git/hooks/
```

[1] It is explained in ci.inria.fr FAQ documentation

And then modify the post-commit hook `post-receive` with your personal repository:

```
#!/bin/sh
wget -q -O - --auth-no-challenge --no-check-certificate \
  http://ci.inria.fr/tpcisedsaclay/git/notifyCommit?url=
  https://scm.gforge.inria.fr/anonscm/git/tpcisedsaclay/users/<yourforgelogin>.git
```

2.3 Jenkins build configuration

In the **Build** section :

- Change the root `POM` path to `java/pom.xml`
- Type `test` in **Goals and options** section.

Save and run

- Click **Save** at the bottom of the page
- Click **Build Now** in the menu on the left.

Check the output

In the **Build History** on the left :

- Click on the last build (hopefully #1)
- Select **Console Output**
- You can also check the **Test Result**.

2.4 Exercise 1

Have a look at the code of the method `Sphere::computeVolume()`.

To do so, edit `java/src/main/java/fr/inria/sed/Sphere.java`.

The code should be like that :

```
public double computeVolume1() {
return 4 * Math.PI * Math.pow(radius_, 3) / 3;
}
```

We might want the value : `4 * Math.PI / 3` to be computed once and for all.

- Extract this value into a class data member

- Run the test again. It should fail. Here is the output you should get :

```
-------------------------------------------------------
T E S T S
-------------------------------------------------------
Running fr.inria.sed.SphereTest
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.076 sec <<< FAILURE

Results :

Failed tests:
testComputeVolume(fr.inria.sed.SphereTest):
expected:<14.137166941154069> but was:<14.137166941154067>
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
```

- Check that tests fail on `Jenkins` instance.

  Hint : Have you commited and pushed your changes ?

2.5 Exercise 2

Why does the test in `SphereTest.java` fail ?[2]
Our response to this issue strongly depends on what kind of application we are working on :

- In some cases, we want to be aware that something has changed, even when the change is the tiniest. In that case, the test we already have is just what we want.
- In other cases, we need not worry about such a small difference and hence do not want to be bothered by tests complaining.
- Find a way to modify the appropriate test in `SphereTest.java` so small rounding differences do not cause errors.
- Check that tests pass on CI.

3 Code Coverage

At this stage, all our tests pass. But what does that mean regarding our application ?

[2]This is because of `floating point arithmetics rounding errors`

Not much you may say, but can you quantify it and will you be able to tell on a real project ?
This is when **test coverage** becomes handy.

3.1 Run test coverage

Run the following `maven` command :

```
$ pwd
yourpath/<yourforgelogin>/java
$ mvn cobertura:cobertura
```

`Cobertura` should have produced test coverage results in the following directory :
`java/target/site/cobertura`
Use your favorite Web browser (`firefox, chrome, iceweasel, ...`) to visualize the generated results of the test coverage :

```
$ firefox target/site/cobertura/index.html &
```

3.2 Exercise 3

- Populate your tests to achieve 100

3.3 Integrate Cobertura to your reporting local Website

Add the following to the file `pom.xml`

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>cobertura-maven-plugin</artifactId>
      <version>2.7</version>
    </plugin>
  </plugins>
</reporting>
```

The report generation is now included in the build life cycle (**site** phase).

- Generate the report :

```
$ pwd
yourpath/<yourforgelogin>/java
$ mvn site
```

- Use your favorite Web browser (`firefox, chrome, iceweasel, ...`) to visualize the generated report :

```
$ firefox target/site/project-reports.html &
```

3.4 Exercise 3bis - Coverage Report on Jenkins

Adding build feedback

One of the good things with `Jenkins` is its ability to nicely present your test results.
The tools we used to run tests and code quality checks have been selected based on the availability of the corresponding `Jenkins` plugins.

Add coverage report

- Go back to your item's configuration page (use the menu on the left)
- In the **Build** section, add to the **Goals and options** field :

  `cobertura:cobertura -Dcobertura.report.format=xml`
- Click on **Add post-build action** and select **Publish Cobertura Coverage Report** from the drop-down menu
- The Cobertura xml report pattern is :

  `**/target/site/cobertura/coverage.xml (it is the example provided below the field)`
- Save and Run the test
- Check the output : in the `Build History` on the left, click on the last build, you should have a new entry named **Coverage Report**, have a look at it

3.5 Add a new class to our project

Let's add a new class `Alphabet` and its test class `AlphabetTest` to our project :

```
$ git merge origin/alpha-java
```

You should have the following file tree :

```
<yourforgelogin>/java
|_ pom.xml
|_ src
   |_ main
   |  |_ java
   |     |_ fr
   |        |_ inria
   |           |_ sed
   |              |_ Alphabet.java
   |              |_ Sphere.java
   |_ tst
      |_ java
         |_ fr
            |_ inria
               |_ sed
                  |_ AlphabetTest.java
                  |_ SphereTest.java
```

Exercise 4

- Generate and visualize the corresponding coverage report
- Make what changes are necessary to achieve 100% test coverage
  Hint : you may not need any additional tests ;)
- Check that you get the same results on `Jenkins`.

3.6 Stylecheck

Run a style checker

Let's try and run a style checker :

```
$ pwd
yourpath/<yourforgelogin>/java
$ mvn checkstyle:checkstyle
```

Then use your favorite Web browser (`firefox, chrome, iceweasel, ...`) to visualize the generated checkstyle report :

```
$ firefox target/site/checkstyle.html
```

We get plenty of errors with the default style `sun_checks.xml`. Indeed, our coding style is closer to `Google` style than it is to `Sun` style.

Run Google style checker

Try running with `Google` checks (provided by the plugin)

```
$ pwd
yourpath/<yourforgelogin>/java
$ mvn checkstyle:checkstyle -Dcheckstyle.config.location=google_checks.xml
```

NOTE : this can also be achieved by adding the following lines to your `pom.xml` :

```
<properties>
  <checkstyle.config.location>google_checks.xml</checkstyle.config.location>
</properties>
```

It is getting better but we might want to make a few changes to this default behavior.

Exercise 5

Make your own custom style checker

- Retrieve a local copy of `Google` check file (name it `custom_checks.xml`)

```
$ wget -O custom_checks.xml http://tinyurl.com/z9pfdlx
```

- Tell `Maven` to use this configuration file by changing the value of variable `checkstyle.config.location`:

```
$ pwd
yourpath/<yourforgelogin>/java
$ mvn checkstyle:checkstyle -Dcheckstyle.config.location=custom_checks.xml
```

- Modify the file `custom_checks.xml` so that it accepts single character parameter names.

- You may also consider removing trailing underscores from data members or amending `checkstyle.xml`

Integrate checkstyle to your reporting local Website

Add the following to the file `pom.xml`

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      <version>2.16</version>
      <configuration>
        <configLocation>custom_checks.xml</configLocation>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

The check style report is now included in the build life cycle (**site** phase).

- Generate the report :

```
$ pwd
yourpath/<yourforgelogin>/java
$ mvn site
```

- Use your favorite Web browser (`firefox, chrome, iceweasel, ...`) to visualize the generated report :

```
$ firefox target/site/project-reports.html &
```

Exercise 5bis - Checkstyle Report on Jenkins

- Go back to your item's **configuration** page.
- In the **Build** section,
  - Add to the `Goals and options` field :
    `checkstyle:checkstyle -Dcheckstyle.config.location=custom_checks.xml`
  - Or change all the `Goals and options` field to : `site`

- In the **Build Settings** section, check the **Publish Checkstyle analysis results** box.
- Save and run the test.
- Check the output : in the **Build History** on the left, click on the last build, you should have a new entry named **Checkstyle Warnings**. Have a look at it.