Assignment 5

Continuous Integration

Due Date: Feb. 13, 2020

Yasaman Amannejad, 2020

Table of Contents

1 SUMMARY

1.1 SOFTWARE UNDER TEST/DEVELOPMENT

The system to be tested for this assignment is JFreeChart. Use the "JFreeChart v1.3.zip" file from Blackboard. This version contains the source code of the Range class + some bugs!

1.2 GROUP WORK

This assignment should be completed in groups that you formed in the first assignment.

1.3 ASSIGNMENT PROCESS

Section 2 and 3 provide an overview of working with git and creating a workflow in GitHub. If you are familiar with these concepts, you can skip these sections.

In this assignment, you will practice setting up a continuous integration workflow. You need to first create a **private** Git repository. You will add your JFreechart source code and the latest versions of the tests you have built for the **Range class** to your Git repository. In this assignment, we will only focus on the Range class. You will add a workflow to your repository to automatically run your tests when the code in the repository gets updated (e.g., on push).

Since, you are using a defective version of the Range class, when running your tests your workflow will initially fail. Your task is to fix the code of the Range class and make your workflow pass successfully! Fixing the methods are simple, the focus is on setting the workflow correctly, and working with Git.

It is recommended that all team members work together and create the workflow and then each team member fixes some of the methods of the Range class and push the changes to the shared repository. This way all members will have the chance to practice working with Git and contribute to the project All team members' contributions must be visible in the commit history.

Note that in creating your workflow, you must find and pass the required libraries/dependencies to the JFreechart, and also add the required Junit dependencies to your project.

Note: If you are using Junit standalone console, it expects the name of the tests to end with the keyword Test, "e.g., RangeTest". You may need to refactor the name of your test classes before adding them to your repository.

1.4 DELIVERABLES

- When your repository is complete, you can add me (username: comp3505) as a collaborator to your repository. This way I can review your code and the structure of your repository. comp3505 is the username of the github account that I will use.
- **No report** or other form of submissions are required for this assignment.

Marking scheme is provided for you in Section 5.

2 FAMILIARIZATION

2.1 GIT

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Git was initially created by Linus Torvalds in 2005 for development of the Linux kernel. Git has gained a vast amount of interest in the last few years and is now among the top 10 required skills for IT professionals[1].

- ♣ You can download and install Git on your machine from https://git-scm.com/downloads
- ♣ If you are new to Git, please read and follow these links to practice working with Git:
  1. https://guides.github.com/activities/hello-world/
  2. https://guides.github.com/introduction/git-handbook/
- ♣ If you need a quick review of Git commands, you can use Git cheat sheet: https://education.github.com/git-cheat-sheet-education.pdf

[1] https://www.techrepublic.com/article/the-most-in-demand-technologies-for-it-professionals/

2.2 CONTINUOUS INTEGRATION

Continuous integration (CI) is often mentioned when people talk about modern development process. CI is a practice that focuses on integrating code more frequently and preparing for an easier release of the code. Developers practicing continuous integration merge their changes back to the repository as often as possible. The developer's changes are validated by creating a build and running automated tests against the build. By doing so, you avoid the integration hell that usually happens when people wait for the release day to merge their changes into the release branch. Continuous integration puts a great emphasis on automated tests to check that the application is not broken whenever new commits are integrated into the repository.

There exist many tools that support CI. Recently, GitHub has provided GitHub Actions that can easily integrate CI into your development repositories. Here is more information that you can read to learn about GitHub Actions:

- **Automating your workflow with GitHub Actions**: https://help.github.com/en/actions/automating-your-workflow-with-github-actions
- **Workflow syntax for GitHub Actions**: https://help.github.com/en/actions/automating-your-workflow-with-github-actions/workflow-syntax-for-github-actions

3 CREATE YOUR FIRST CI PIPELINE

If you are familiar with working with CI in Git, you can skip this section. Otherwise, pleased follow the steps in detail.

In this section, you will create your first CI workflow. Since you will need some Linux command line knowledge for creating your workflow, it is recommended that you follow the following steps on a cloud instance.

1. Create an Ubuntu 18.04 instance in your Google cloud account.
2. ssh to your instance.
3. Use the following commands to prepare your instance
   - Update your instance
     ```
     sudo apt update
     ```
   - Install Java (to compile your java code)
     ```
     sudo apt install default-jdk
     java -version
     ```
   - Install Git (to work with Git)
     ```
     sudo apt install git
     ```

4. Create a Github account for yourself. Login to your account and fork this repository (There is a fork option in GitHub UI). It will create a copy of my repository in your account.
   https://github.com/Yasaman-A/Assign5-preparation

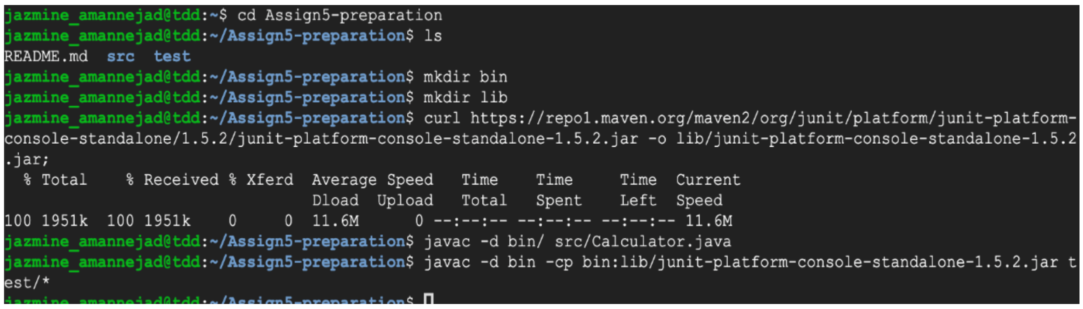
5. Now, clone your own repository to your cloud instance. In your cloud instance, type:
   ```
   git clone THE URL OF YOUR REPOSITORY
   ```

6. Enter to your Git folder
   ```
   cd Assign5-preparation
   ```
7. Check the files in your current directory. There should be a src and test folders.
   ```
   ls
   ```

8. **Running your tests from the command line**: Now, you have a copy of the repository in your cloud instance, let's first make sure that we know how to run the tests from the command line. To run the tests from the command line, follow these instructions:
   - Create a bin folder
     ```
     mkdir bin
     ```
   - Create a lib folder
     ```
     mkdir lib
     ```
   - Download `junit-platform-console-standalone-1.5.2` jar file and store it in your lib folder. Other versions should also work the same.

```
curl https://repo1.maven.org/maven2/org/junit/platform/junit-
platform-console-standalone/1.5.2/junit-platform-console-
standalone-1.5.2.jar -o lib/junit-platform-console-standalone-
1.5.2.jar
```

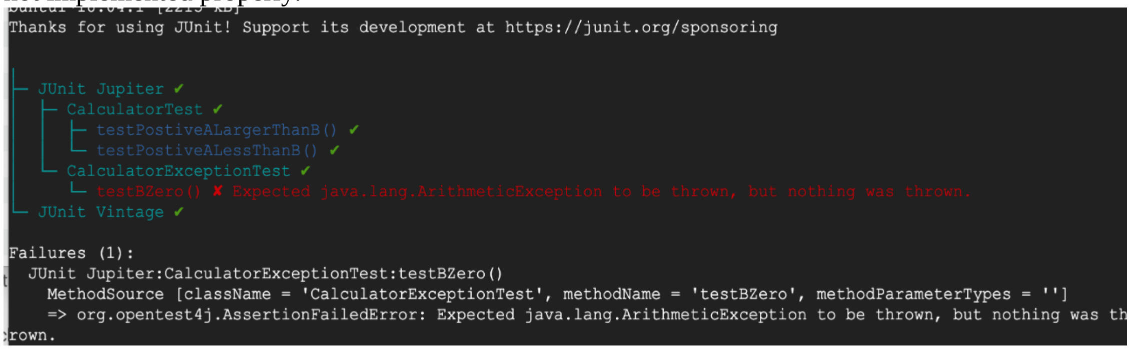
- Compile your source files
  ```
  javac -d bin/ src/Calculator.java
  ```
- Compile your test files
  ```
  javac -d bin -cp bin:lib/junit-platform-console-standalone-
  1.5.2.jar test/*
  ```
- Run your test files
  ```
  java -jar lib/junit-platform-console-standalone-1.5.2.jar --
  class-path bin --scan-class-path
  ```



```
jazmine_amannejad@tdd:~$ cd Assign5-preparation
jazmine_amannejad@tdd:~/Assign5-preparation$ ls
README.md  src  test
jazmine_amannejad@tdd:~/Assign5-preparation$ mkdir bin
jazmine_amannejad@tdd:~/Assign5-preparation$ mkdir lib
jazmine_amannejad@tdd:~/Assign5-preparation$ curl https://repo1.maven.org/maven2/org/junit/platform/junit-platform-
console-standalone/1.5.2/junit-platform-console-standalone-1.5.2.jar -o lib/junit-platform-console-standalone-1.5.2
.jar;
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1951k  100 1951k    0     0  11.6M      0 --:--:-- --:--:-- --:--:-- 11.6M
jazmine_amannejad@tdd:~/Assign5-preparation$ javac -d bin/ src/Calculator.java
jazmine_amannejad@tdd:~/Assign5-preparation$ javac -d bin -cp bin:lib/junit-platform-console-standalone-1.5.2.jar t
est/*
```

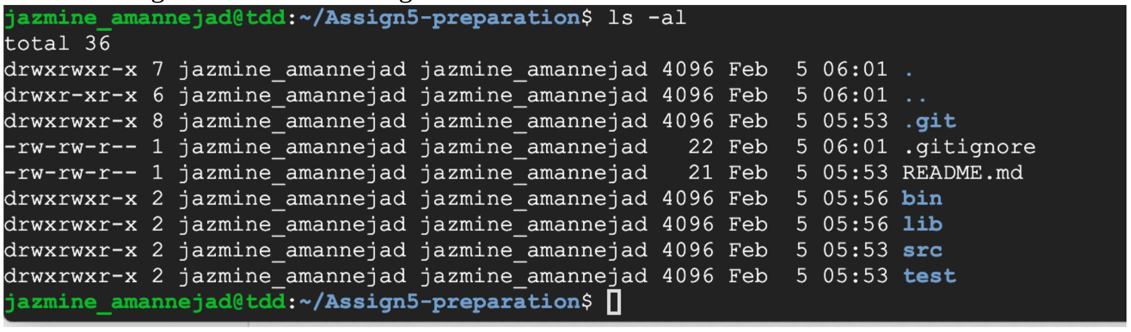
<span style="color:red">NOTE</span>: Please note that `junit-platform-console-standalone` expects that the name of your test files end with the keyword Test. You may need to update some of your test names.

9. Results of your test should look like this. One of the tests is failing, because the code under test is not implemented properly.



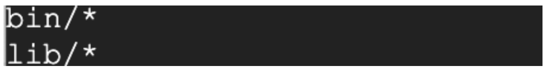
```
Thanks for using JUnit! Support its development at https://junit.org/sponsoring


├─ JUnit Jupiter ✔
│  ├─ CalculatorTest ✔
│  │  ├─ testPostiveALargerThanB() ✔
│  │  └─ testPostiveALessThanB() ✔
│  └─ CalculatorExceptionTest ✔
│     └─ testBZero() ✘ Expected java.lang.ArithmeticException to be thrown, but nothing was thrown.
└─ JUnit Vintage ✔

Failures (1):
  JUnit Jupiter:CalculatorExceptionTest:testBZero()
    MethodSource [className = 'CalculatorExceptionTest', methodName = 'testBZero', methodParameterTypes = '']
    => org.opentest4j.AssertionFailedError: Expected java.lang.ArithmeticException to be thrown, but nothing was th
rown.
```

10. Now, to practice working with Git add/commit/push commands, let's modify the .gitignore file and push it to the repository. The .gitignore file is already created and exists in the main directory of your repository (If was initially created in the repository that you forked). You can see the list of all files including the hidden files using `ls -al` command.



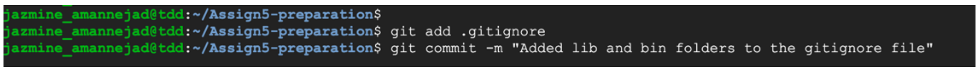
```
jazmine_amannejad@tdd:~/Assign5-preparation$ ls -al
total 36
drwxrwxr-x 7 jazmine_amannejad jazmine_amannejad 4096 Feb  5 06:01 .
drwxr-xr-x 6 jazmine_amannejad jazmine_amannejad 4096 Feb  5 06:01 ..
drwxrwxr-x 8 jazmine_amannejad jazmine_amannejad 4096 Feb  5 05:53 .git
-rw-rw-r-- 1 jazmine_amannejad jazmine_amannejad   22 Feb  5 06:01 .gitignore
-rw-rw-r-- 1 jazmine_amannejad jazmine_amannejad   21 Feb  5 05:53 README.md
drwxrwxr-x 2 jazmine_amannejad jazmine_amannejad 4096 Feb  5 05:56 bin
drwxrwxr-x 2 jazmine_amannejad jazmine_amannejad 4096 Feb  5 05:56 lib
drwxrwxr-x 2 jazmine_amannejad jazmine_amannejad 4096 Feb  5 05:53 src
drwxrwxr-x 2 jazmine_amannejad jazmine_amannejad 4096 Feb  5 05:53 test
jazmine_amannejad@tdd:~/Assign5-preparation$
```

11. Open the file using vim or nano and modify the content. You need to add the lib and bin folders to your .gitignore file, to prevent these folders from being pushed to your repository.
    ```
    nano .gitignore
    ```



```
bin/*
lib/*
```

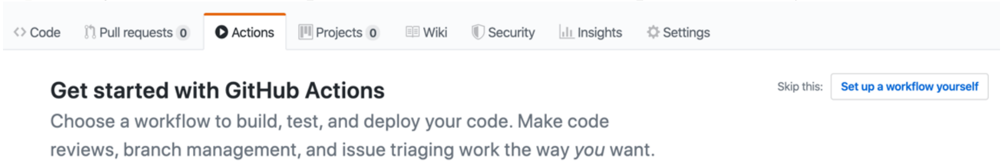
12. Use git add, commit to update your local repository.



```
jazmine_amannejad@tdd:~/Assign5-preparation$
jazmine_amannejad@tdd:~/Assign5-preparation$ git add .gitignore
jazmine_amannejad@tdd:~/Assign5-preparation$ git commit -m "Added lib and bin folders to the gitignore file"
```

13. Use push command to update your remote repository.
    `git push origin master`
    - You may need to provide your username/email and password. Follow the instructions.

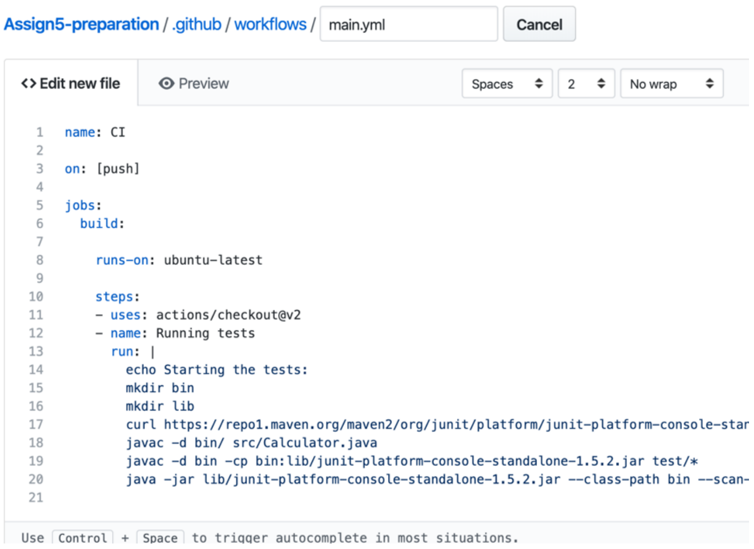
14. Now, let's create a workflow in our repository that runs our tests anytime we push changes to the repository. Go to Github, open Actions tab. Select "set up a workflow yourself".



Get started with GitHub Actions

Choose a workflow to build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way *you* want.

15. Modify the content, and add the commands that you used when running from command line to your .yml file.

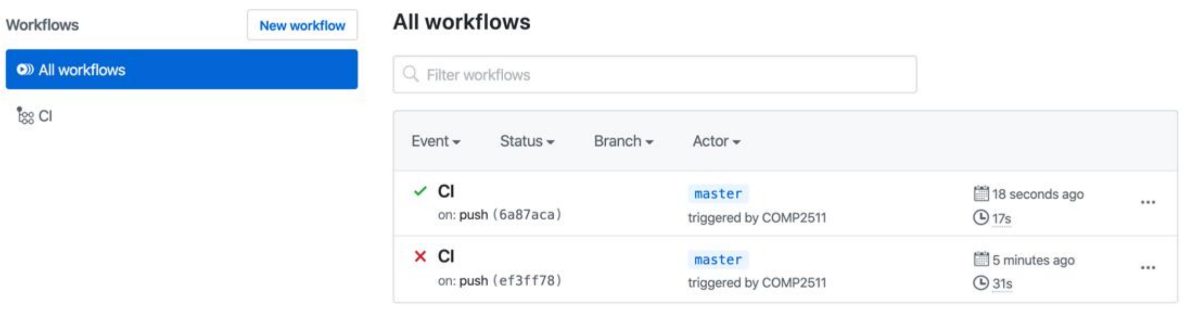


```
name: CI

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Running tests
      run: |
        echo Starting the tests:
        mkdir bin
        mkdir lib
        curl https://repo1.maven.org/maven2/org/junit/platform/junit-platform-console-star
        javac -d bin/ src/Calculator.java
        javac -d bin -cp bin:lib/junit-platform-console-standalone-1.5.2.jar test/*
        java -jar lib/junit-platform-console-standalone-1.5.2.jar --class-path bin --scan-

```

Use Control + Space to trigger autocomplete in most situations.

16. Commit your changes from the UI.
17. Now, your workflow is ready. If you make any push to your repository, your workflow will be triggered. Lets' test it!

18. Remember that the code under test was not implemented properly, and therefore one of the tests was failing. Open your Calculator.java file from your cloud instance, modify the code, and push your changes to the repository. Your workflow will run and if your implementation is correct, it will pass, otherwise, it will fail.



✓ CI on: push (6a87aca)	master triggered by COMP2511	18 seconds ago 17s	...
✗ CI on: push (ef3ff78)	master triggered by COMP2511	5 minutes ago 31s	...

4 INSTRUCTIONS

4.1 CREATE A REMOTE REPOSITORY

1. One member of the team should create a **private** Git repository (on GitHub), and then add the rest of the team members as collaborators to that projects.
2. As a group, do the following steps:
   a. Clone the git repository to your computer.
   b. Download the files for assignment 5 from Blackboard and add them to your Git repository (add, commit, push).
   c. Add the latest versions of the tests for the Range class to the repository.
   d. Create a “`.gitignore`” file to exclude the file/folders that should not be added to the Git repository.
   e. Push your changes to the shared repository on GitHub.
3. Now, all members together, should create the workflow in the GitHub repository, and complete the content of the `yml` file.
   a. If you want to first check your commands, you can clone your git repository on your cloud instance and try to run the tests from the command line. When all your commands work, you can add them to your workflow.

   b. Compile your source files
   When compiling your JFreechart, you need to compile multiple java files. You can use `$(find src/ -name "*.java")` command to find and list all java files in the src folder to compile.

   When compiling your JFreechart, you need add the required compile time dependencies to the class path. You can use the -cp option to add the dependencies to your class path. The general command format will be like this (you may need to change the file names and paths based on your repository):

   ```
   javac -cp lib/jcommon.jar:lib/servlet.jar:lib/xml.jar:lib/ -d bin/
   $(find src/ -name "*.java")
   ```

   The required jar files are given in the assignment folder, and here are the link that you can find and add them to your workflow[2]:
   i. jcommon
   ii. servlet
   iii. XML api

   c. Compile your test files
   Now, you need to compile your test classes with Junit (This step will be similar to what you did in Section 3). Junit standalone console expects the name of the tests to end with the keyword Test, “e.g., RangeTest”. You may need to refactor the name of some of your test methods before adding them to your repository.
   i. If you are using Junit 4, please refer to the Appendix.

   d. Run your tests
   Running tests will be similar to what you did in Section 3.

   e. Run your workflow
   You can add your commands to your workflow. When your workflow is ready (runs your tests), it will fail, because the Range class some defects and you need to fix them. You should be able to see the failing tests.

4. Team members should clone the repository and start fixing the Range class. Each member of the team should fix some of the methods in the Range class and will push their changes to the repository. Your team will continue this process until your workflow passes successfully.
   a. When pushing your changes, if you notice a merge conflict, you need to first pull the updated/new files from the repository and resolve the conflicts (Learn more).

[2] https://mvnrepository.com/artifact/org.jfree/jfreechart/1.0.17

5. When your workflow passes successfully, your project is ready. From this point on, if any change happens in your repository, tests will run automatically, and if your workflow fails that means that the changes are not compatible with the expected behaviour.

Expand your knowledge

- In many real projects, adding updated/new files to the repository happens with `pull-request`. Read about `pull requests` to understand the difference between `pull, push,` and `pull-requests`.
  https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests

- In many real projects, most of the developments happen in branches other than the `master` branch. Read about the advantages of using branches in your development process. Learn how to create branches and start using them!
  https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-branches

5 SUBMISSION

5.1 DELIVERABLES

- When your repository is complete, you can add me (**username: comp3505**) as a collaborator to your Github repository. This way I can review your code and also the structure of your repository. comp3505 is the username of the github account that I will use for this course.
- **No report** or other form of submissions are required for this assignment.

5.2 GRADING SCHEMA

Marking Scheme	
Correctness of the implemented methods (Range class is updated correctly)	10%
Correct setup and use of the Git repository - gitignore is added. - additional files are not stored in the repository - proper messages are used for commits - team members contributions are all visible in the commit history.	50%
Correct setup of the Git workflow - the .yml file is defined correctly, and the content is clean	40%

6 Appendix

This section is for Junit 4 users only.

1. Create a test suite which calls all your tests
2. Create a Java class in your test folder and name it TestRunner.
3. Add the following code to the body of your TestRunner. It will run your tests and will print out the result for you.

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
	public static void main(String[] args) {
		Result result = JUnitCore.runClasses(AllTests.class);
		System.out.println("All tests were successful? " + result.wasSuccessful());
		if (result.getFailures() != null) {
			for (Failure failure : result.getFailures()) {
				System.out.println(failure.toString());
			}
			System.exit(1);
		}
	}
}
```

4. **Compile your code:** You need to use the following dependencies to compile your source and test code
   - jcommon
   - servlet
   - XML api
   - Junit 4

```
javac -cp lib/jcommon.jar:lib/servlet.jar:lib/xml.jar:lib/junit.jar -d bin/
$(find . -name "*.java")
```

(you may need to change the file names and paths in this command based on your folder structure)

5. **Run your code:** Then use the following library as your runtime dependency:
   - hamcrest

```
java -cp bin:lib/junit.jar:lib/hamcrest.jar TestRunner
```