# JUNIT – Java Unit Testing

Tom Harron, Aquinas (FRC 4011) Programming Mentor, [rundad48@gmail.com](mailto:rundad48@gmail.com)

Tom’s github repository: <https://github.com/TomHarron1/RobotLearningDay>

**Goal: Test Robot code in VSCode, without being connected to Robot.**

## Outline

1. Required Installations – setting up VSCode
2. What is JUNIT?
   1. Run tests on Circle class (no robot-dependent code)
3. Refactoring code
   1. BadRobot.java
   2. GoodRobot.java
4. Writing JUNIT tests – GoodRobotTest.java
5. JUNIT – test code with robot-dependent code. We need Mockito.

## Required Installations – setting up VSCode

1. FRC installers install a special VSCode
   1. FRC installation of VSCode will not disrupt other installs of VSCode on your system.
   2. Installer will create desktop icons – always use these to work with robot code!
   3. FRC website: <https://wpilib.screenstepslive.com/s/currentCS/m/java>
      1. Installing C++ and Java Development Tools for FRC
      2. Installing the FRC Update Suite (All Languages)
   4. Software is installed to **c:\Users\Public\frc2019**
2. Use VSCode to create template project – I selected TimedRobot
   1. The entire project directory is created for you.
   2. Type CTRL-SHIFT-P (“command palette”), enter WPILib: Create a new project <Enter>
      1. Project type: template
      2. Language: java
      3. Project base: Timed Robot
      4. Folder: c:\<your path>\TrainingDay\RobotLearningDay
         1. This assumes you pulled some from github (next session)
      5. Project name: TimedRobot
      6. Team number: 4011 (for Aquinas)
      7. Click “Generate Project” button
3. JUNIT requirements
   1. Using “Extensions” button in VSCode, be sure these are installed:
      1. Java Test Runner
      2. Language Support for Java by Red Hat
      3. Debugger for Java
      4. Read more about JUNIT here: <https://code.visualstudio.com/docs/java/java-testing>
   2. Copy C:\Users\Public\frc2019\maven\junit\junit\4.12 to C:\tom\TrainingDay\RobotLearningDay\TimedRobot\build\libs
   3. Create folders C:\tom\TrainingDay\RobotLearningDay\TimedRobot\src\test\frc\robot
      1. You need to create test, then frc, then robot
      2. These folders match structure under \src\main\java\frc\robot
4. Edit gradle.build file – be sure you have these things (**in bold**)

// Maven central needed for JUnit

**repositories {**

**mavenCentral()**

**}**

// Defining my dependencies. In this case, WPILib (+ friends), and vendor libraries.

// Also defines JUnit 4.

dependencies {

compile wpi.deps.wpilib()

compile wpi.deps.vendor.java()

nativeZip wpi.deps.vendor.jni(wpi.platforms.roborio)

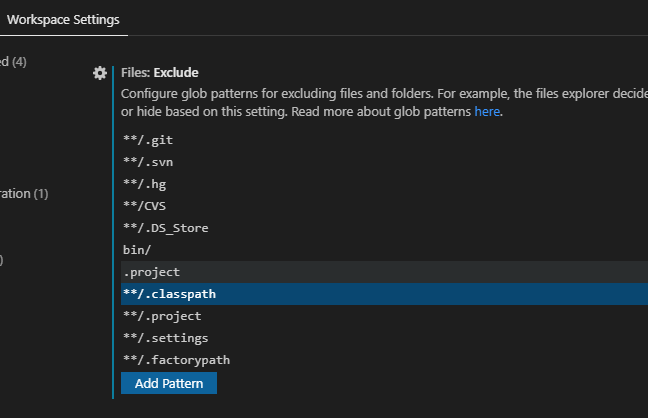
nativeDesktopZip wpi.deps.vendor.jni(wpi.platforms.desktop)

**testCompile 'junit:junit:4.12'**

**testCompile 'org.mockito:mockito-core:3.2.0'**

}

1. Edit .classpath file (which by default is not seen in VSCode!)
   1. **To make this file visible in your project**
      1. CTRL-SHIFT-P (bring up Command Palette), enter Open Settings (UI)
      2. Search for: files:exclude
      3. Click on the ‘Workspace Settings’
      4. Delete the entry of: \*\*/.classpath



* 1. Add the bold section – this will enable JUNIT testing

<?xml version="1.0" encoding="UTF-8"?>

<classpath>

<classpathentry kind="src" output="bin/main" path="src/main/java">

<attributes>

<attribute name="gradle\_scope" value="main"/>

<attribute name="gradle\_used\_by\_scope" value="main,test"/>

</attributes>

</classpathentry>

**<classpathentry kind="src" output="/bin/test" path="src/test">**

**<attributes>**

**<attribute name="gradle\_scope" value="test"/>**

**<attribute name="gradle\_used\_by\_scope" value="test"/>**

**<attribute name="test" value="true"/>**

**</attributes>**

**</classpathentry>**

<classpathentry kind="con" path="org.eclipse.jdt.launching.JRE\_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/JavaSE-11/"/>

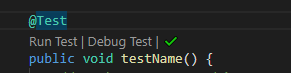
<classpathentry kind="con" path="org.eclipse.buildship.core.gradleclasspathcontainer"/>

<classpathentry kind="output" path="bin/default"/>

</classpath>

If the yellow/bold section is present, you will correctly see the “Run Test | Debug Test” options.

These options are referred to as “**Code Lens**” – clickable links added to your code to do stuff!



If the yellow/bold section is NOT present, these options will not appear, which will prohibit you from running your tests.



1. **WARNING!** If you edit the build.gradle file (like add another dependent library), it will overwrite .classpath file and delete the bold/yellow section. This will cause JUNIT to not work (the CodeLens links disappear) – so you need to edit .classpath and put the information back. Then build again!
2. Define Maven, Gradle, Mockito?
   1. Maven? Web-based repository of LOTS of libraries, and build tool (like Gradle). Maven uses a pom.xml file to list dependencies (source folder, folder containing .jar files, et.c)
   2. Gradle? Build utilities – to build your running robot code! Scans your projects to figure out what has changed, and which pieces need to be rebuilt. For example, if you edit Robot.java, Gradle detects and will recompile and create a new Robot.class file (compiled java code)
   3. Mockito? A software package that “fakes” the presence of a class, used with JUNIT. For example, let’s say you test a class that open a database, but you don’t have the database present. Mockito will let you fake out that class - “mock it” with a placeholder so you can run the rest of your code.
   4. What is JUNIT? See next section 😊

## What is JUNIT?

1. JUNIT is a plugin for VSCode (use Extension to install)
   1. Main website: <https://junit.org/junit5/>
   2. JUNIT stands for Java Unit Testing
   3. JUNIT enables you to write tests to test your methods.
   4. A “unit” is a chunk of code you want to test, that is – **a method (aka function)**
   5. Assuming you have a class named “Robot”, you create a “test class” call TestRobot
      1. Each test starts with annotation of @Test

class **Robot** {

public int calculateAngle() {

return(50);

}

} // end of classRobot

class **TestRobot** {

Robot myRobot = new Robot();

**@Test**

Public void testAngle() {

assertEquals(50, myRobot.calcualateAngle());

}

} // end of classTestRobot

We’ll look at Circle.java (implementation) and CircleTest.java (junit tests)

* Add another method to Circle class
* Add some tests to CircleTest.java

## Refactoring code

**Refactor code** – restructure existing computer code, WITHOUT changing the external behavior!

**\*\* The goal is to improve the CODE ITSELF, without changing what the program does at all**

**This must be done in tiny steps.    Change code, test to ensure no changes, REPEAT!**

### Advantages

1. Improve code **readability**
2. Reduce code **complexity**
3. Improve source code **maintainability** (easier to make changes, without fear of breaking things!)
4. Improve **extensibility** (the ability to easily add more code, and extend functionality) without breaking things!)

### 

### Checklist of things to refactor

1. Add more and better comments
2. Remove “magic numbers” from code by using CONSTANTS
   1. Change:    if (xbox.getButton() == 3)      to

                                                    if (xbox.getButton() == LEFT\_SHOULDER\_BUTTON)

1. Break code into small methods (strive for methods about one screen long) – makes code simpler
   1. Each method should do **something simple**, that is very well defined.
   2. Don’t keep adding code to methods (which makes them more complicated).  Instead, add more methods!
2. Break code into CLASSES. Remove code form Robot.java that is related into a class – perhaps LimeLight.java, Motor.java, etc.
3. Improve complicated logical expressions
4. Avoid having deeply nested structures

          if (x < 6) {

             if (x ==7) {

                 while (sensorReading < 50) {

                     if (level == 7) {

**Instead have:**

          if (x < 6) {

             if (x == 7) {

                 processLightLevel(sensorReading)

             }

          }

          public static processLightLevel(int sensorReading) {

             while (sensorReading < 50) {

if (level == 7) {

                     …

         }

### Refactoring session

BEFORE: BadRobot.java

AFTER: GoodRobot.java

## Writing JUNIT Tests- TestGoodRobot.java

JUNIT is a framework that allow you to easily create tests (independent from your project code)

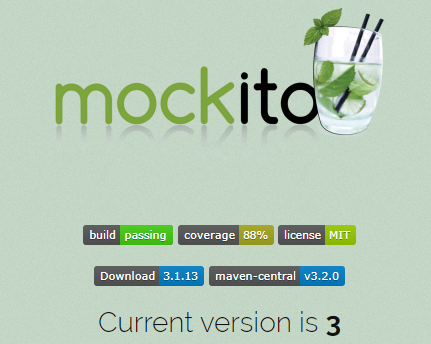
Each test starts with the annotation:    @Test

Tips on writing methods:

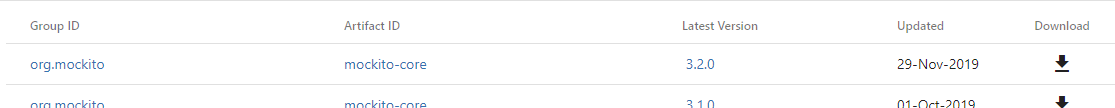
1. JUNIT assumes you have methods to test!!
   1. A method should be simple (fewer lines of code is best)
   2. A method should perform a single task (keep it simple)
   3. It is best for methods to return a value – so you can JUNIT test them. **Avoid “void” methods**
2. Method names should start with a verb. Examples: turnWheel(), getTurnValue(), calculateAngle(), etc.

## JUNIT – test code with robot-dependent code – we need Mockito

1. Download Mockito jar file from here: <https://site.mockito.org/>
   1. Click on maven-central button



* 1. Click on download button for version 3.2.0



* 1. Copy the file (mockito-core-3.2.0.jar) from your Downloads area to C:\tom\TrainingDay\RobotLearningDay\TimedRobot\build\libs (that’s where junit file is)

1. Here’s a good YouTube video about this (20 minutes): <https://www.youtube.com/watch?v=vmRFiF9hd2E>
   1. The Installation steps in this document help make this work (the video is missing some steps)
2. Here’s a nice website on Mockito to learn more:
   1. <https://www.vogella.com/tutorials/Mockito/article.html>
3. Let’s look at Motor.class and MotorTest.class

Motor.java will be used by the Robot when it is operating.

MotorTest.java contains JUNIT tests – we’ll only run this on the PC (development)

**Note:** This technique will allow you test Motor.java, even though the robot it not running. You can test everything (like check how fast the motor is moving, or set the motor speed, etc.) – but you can “fake it” – so that you can run and test other code that does NOT require the robot to be running (like some calculations perhaps, etc.)