

# IntelliJ Setup & Introduction to Travis CI

## IntelliJ & Travis CI lab

- This lab will present one of the most popular IDEs for web development called **IntelliJ**. In the third part of the lab, you will have the opportunity to learn about **Travis CI** which is a hosted, distributed continuous integration service that is being used to build and test programming projects hosted on GitHub.
- **Why learn about Travis CI?** When working in industry, many companies rely on automation to repeatedly verify developer changes, integrating those changes into the code baseline, building and deploying the entire software package, and performing unit and functional testing. Getting familiar with continuous integration will prepare you for your future as a professional software engineer. Travis is highly customizable in a sense that you can choose which notifications to receive and when to test the code. This lab will walk through a setup and use of Travis CI for a small project provided below in a hands-on way to see how it

notifies you when there are problems and when all is well.

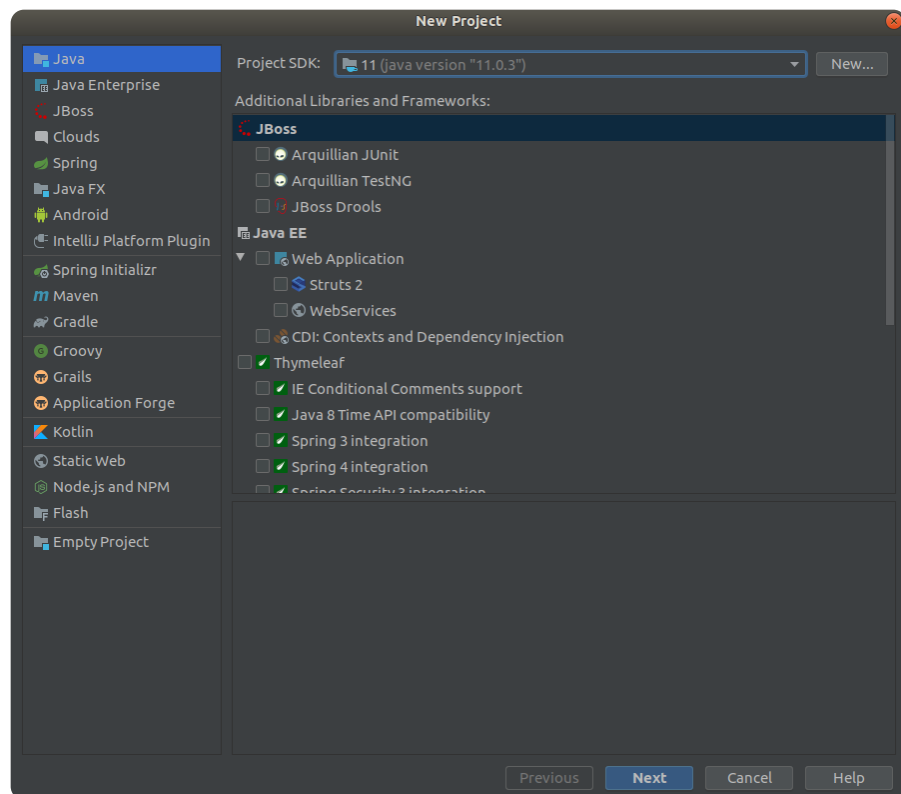
---

## Part 0: Pre-Lab:

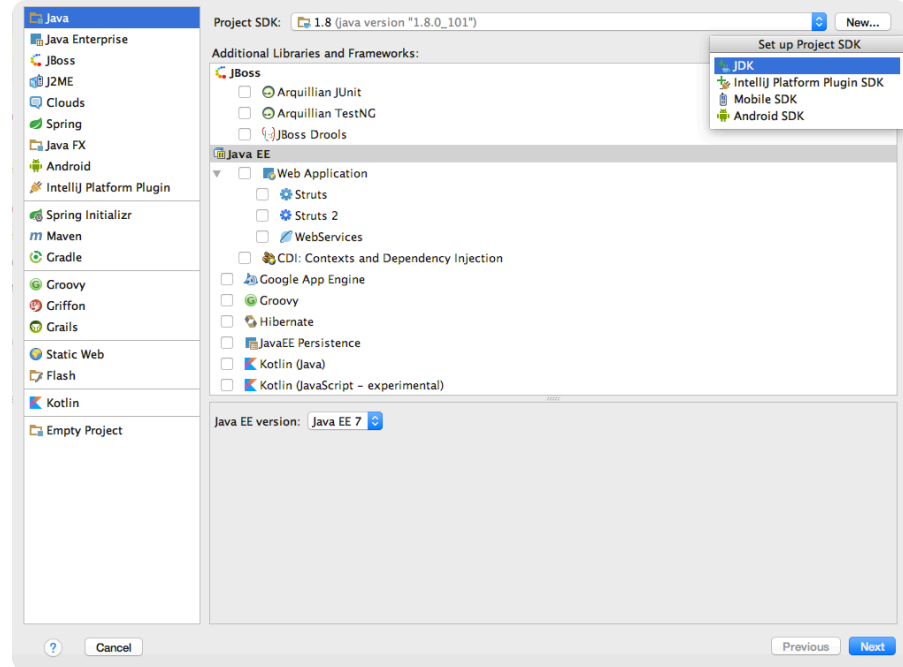
- Make sure you have completed the [Pre-Lab](#) section before you go to Part 1 of this lab.

## Part 1: Introduction To IntelliJ Use

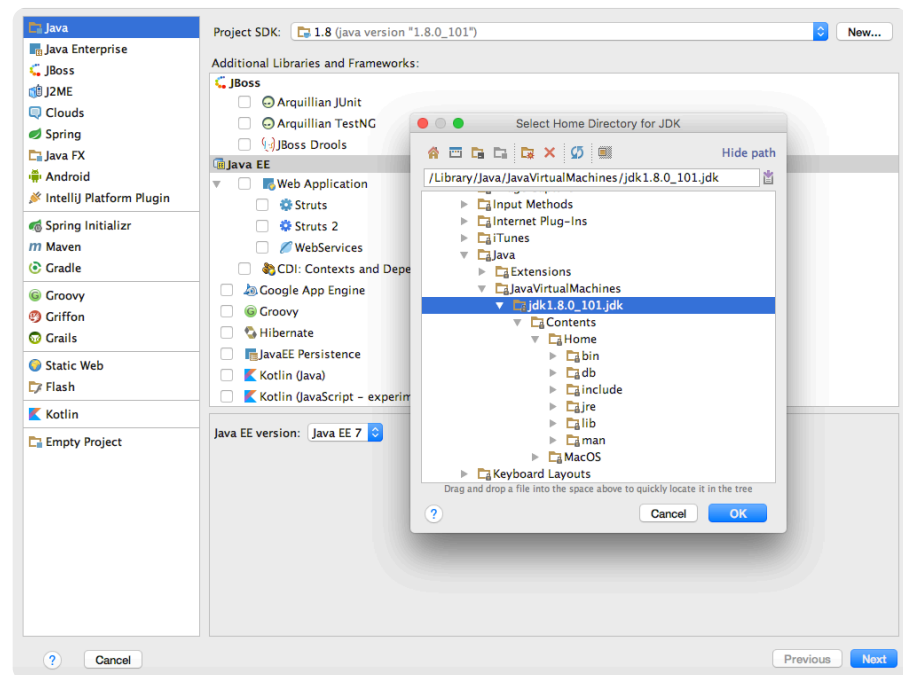
- Create a Project:
  1. Go to **File->New Project**.  
Your java SDK should be selected by default on the dropdown at the top, labelled **Project SDK**, then click **Next**.



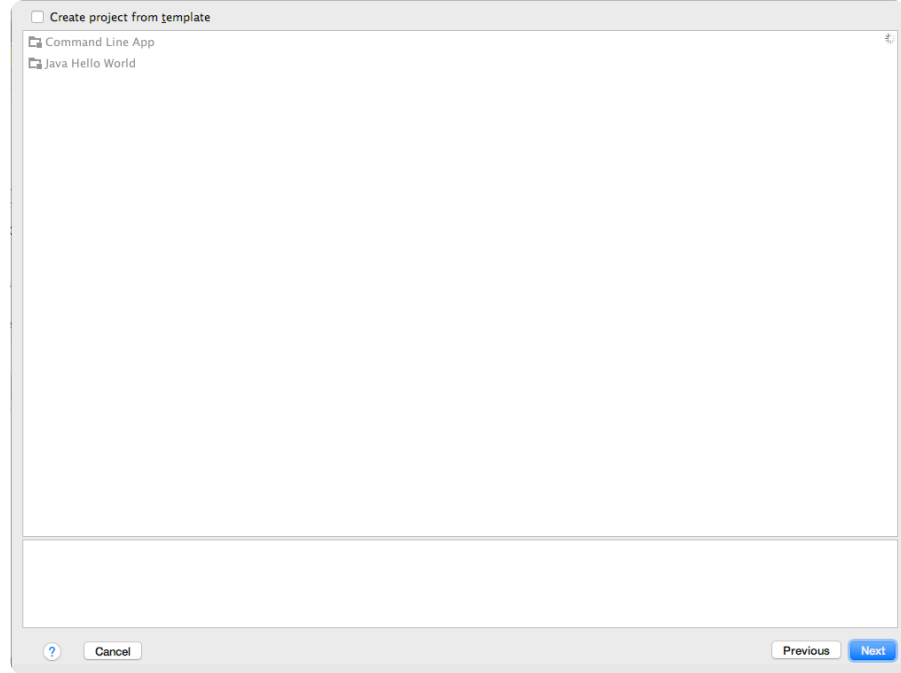
2. If Java is not selected by default, click **New** next to **Project SDK**.



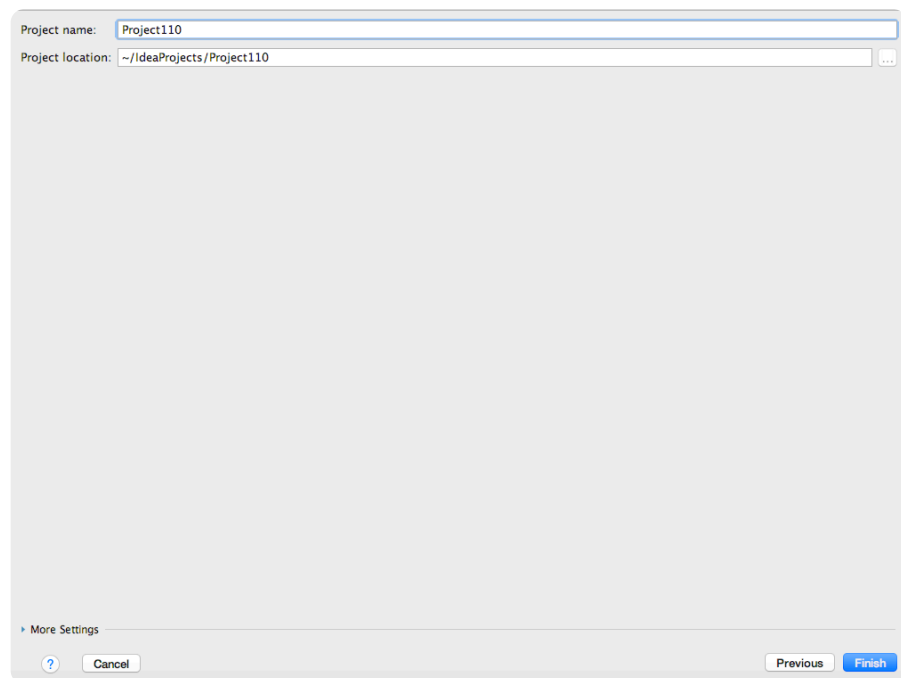
Select **JDK**, then in the new box choose the **JDK** directory (Where JDK is installed).



3. Click **Next** through the following screen displaying templates.



4. Choose a project name and do not change the Project Location field and click on **Finish** button.



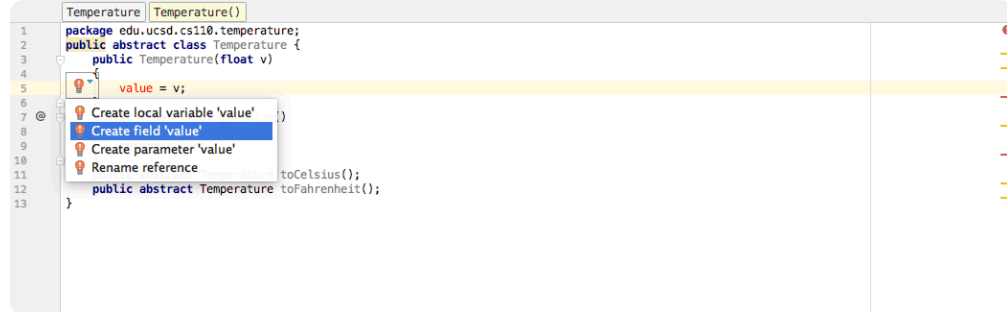
5. Right click on the **src** folder located under the CSE110 Project folder (If the tab with the project files is not open press `alt+1` , or select the dropdown near the top left with the down arrow, and then open the project folder to find the src folder) and select `New->Package` . This

will bring up a naming prompt, name your new package `edu.ucsd.cs110.temperature`.

- Creating Java Classes:
  1. Create a new class by right clicking the new package `edu.ucsd.cs110.temperature`.
  2. Select the `New->Java Class`. Name the new class `Temperature`.
  3. Fill in the body with the code below.

```
package edu.ucsd.cs110.temperature;
public abstract class Temperature {
    public Temperature(float v)
    {
        value = v;
    }
    public final float getValue()
    {
        return value;
    }
    public abstract Temperature toCelsius();
    public abstract Temperature toFahrenheit(
);
}
```

1. Notice that there are quite a few errors displayed, the variable `value` is underlined in red, as it is probably undeclared. To see if it is declared, right click on the variable **value** and select `Go To->Declaration`. Since it is not declared, it prompts you that it cannot find declaration to go to. You will also see a small red lightbulb appear on that line. Clicking the **lightbulb** reveals the text box, select the **Create field 'value'** option to create the field, then select **float** as its type.



2. Now you should look at the class declaration `public abstract class Temperature`. The `abstract` keyword here is a simple implementation of a concept called Inheritance.

**Background:** *Inheritance is an Object Oriented Design theme in which you create one superclass (Temperature in this case) and you create multiple subclasses that inherit from, or extend the functionality of the superclass. When you declare Temperature to be abstract, what you are saying is that you want to use this class as a framework for other classes, and as a result you do not ever want to instantiate a Temperature object by calling `new Temperature(value)`. This concept is not to be confused with declaration, which looks like this: `Temperature temp;`. Declaration simply means that you want a variable to be of type Temperature, and that when you instantiate it you should use a class that inherits from, or extends the Temperature class.*

Lets look into inheritance some more by adding subclasses for Temperature.

Create two new classes, Fahrenheit and Celsius by selecting package **cs110.temperature**->**New**->**Java Class**. Name the class and copy the code below into each of the respective classes.

- Fahrenheit:

```
package edu.ucsd.cs110.temperature;
```

```

public class Fahrenheit extends Temperature
{
    public Fahrenheit(float t)
    {
        super(t);
    }
    public String toString()
    {
        // TODO: Complete this method
        return "";
    }
}

```

- Celsius:

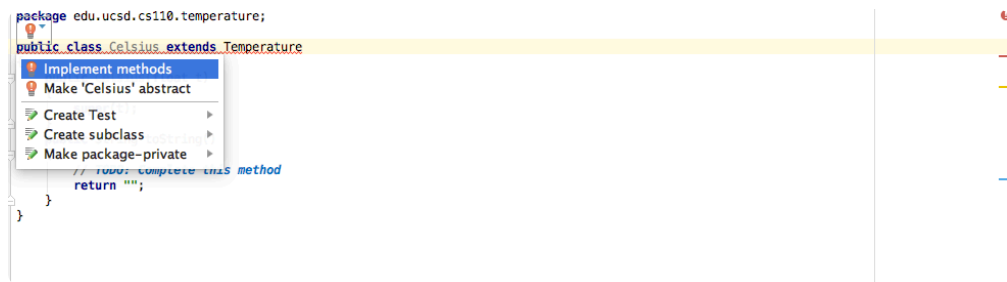
```

package edu.ucsd.cs110.temperature;
public class Celsius extends Temperature
{
    public Celsius(float t)
    {
        super(t);
    }
    public String toString()
    {
        // TODO: Complete this method
        return "";
    }
}

```

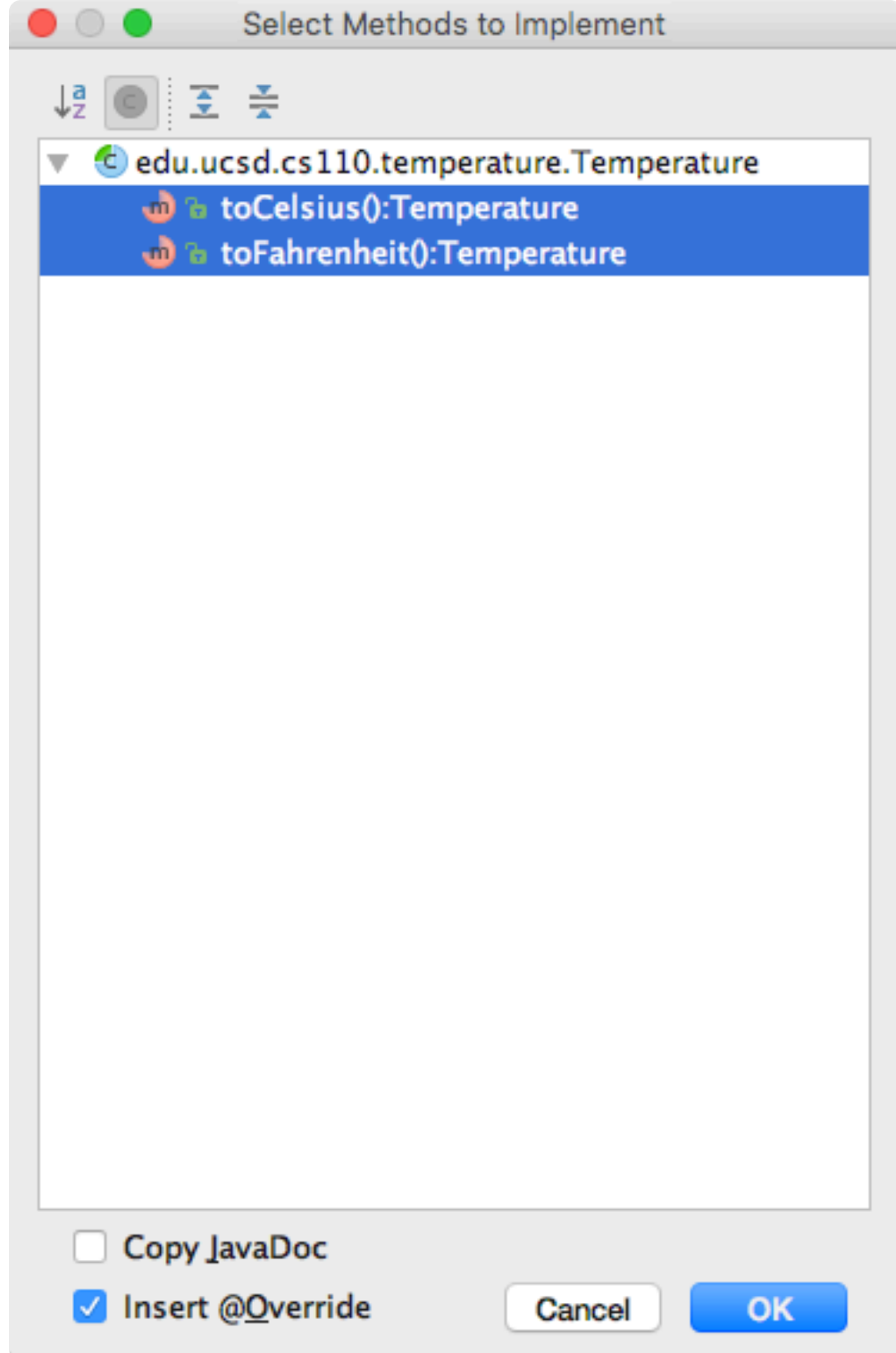
- You should notice errors in both of these classes. If you hover over the name of each class, the error message should tell you that the class needs to extend the abstract methods in Temperature. If you go to the Temperature class you can see the two abstract methods, `toCelsius()` and `toFahrenheit()`. There are two things you should notice about these methods, they both have the

keyword **abstract** in their declaration, and they both do not have a function body. This is because you cannot call these functions directly, you must create an implementation in a subclass that will inherit the abstract. Click on the line containing the error in your newly created files and a red light bulb appears. Click the **lightbulb** and select **Implement Methods**.



- A pop up Box will appear, keep the default values.





- Apply the above steps for both classes.
- Save your work by pressing CTRL+S.
- Create a new Java class to convert between temperatures. Select package **cs110.temperature**->**New->Java Class**. Name the class `TemperatureConverter`.

Add the following main method inside the class body:

```
package edu.ucsd.cs110.temperature;

public class TemperatureConverter {

    public static void main(String args[]) throws IOException
    {
        String input = null;
        Temperature inputTemp = null, outputTemp = null;

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        while(true)
        {
            System.out.println("\nAvailable units: C, F");

            System.out.print("Enter temperature to convert (i.e. 36.8 C, 451 F): ");
            if((input = reader.readLine()) == null) System.exit(0);
            String[] temp_in = input.split(" ");
            float temp_val = Float.parseFloat(temp_in[0]);

            switch(temp_in[1].toLowerCase().charAt(0))
            {
                case 'c':
                    inputTemp = new Temperature(temp_val);
                    break;
                case 'f':
                    inputTemp = new Temperature(temp_val);
                    break;
                default:
```

```

        System.out.println("Invalid entry
        !!\n\n");
        continue;
    }

    System.out.print("Enter the unit to conver
    rt T0: ");
    if((input = reader.readLine()) == null) S
    ystem.exit(0);

    switch(input.toLowerCase().charAt(0))
    {
        case 'c':
            outputTemp = inputTemp.toCelsius(
        );
            break;
        case 'f':
            outputTemp = inputTemp.toFahrenhe
        it();
            break;
        default:
            System.out.println("Invalid entry
            !!\n\n");
            continue;
    }

    System.out.println("\n The converted temp
    erature is " + outputTemp.toString() + "\n\n");
    }
}
}

```

- IntelliJ will likely complain that it cannot resolve `BufferedReader`, `InputStreamReader`, and `IOException`. To resolve these, click over the underlined word and an option will appear prompting you to press ALT+Enter to import the classes (If ALT+Enter isn't working, CTRL/CMD + Click on the error, then click on the lightbulb, and

select import class in the dropdown). If you can't resolve it in this way just add the java.io library to your header definition.

```
public class TemperatureConverter {  
    public static void main(String args[]) throws IOException  
    {  
        String input = null;  
        Temperature inputTemp = null, outputTemp = null;  
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));  
  
        while(true)  
        {  
            System.out.println("\nAvailable units: C, F");  
  
            System.out.print("Enter temperature to convert (i.e. 36.8 C, 451 F): ");  
            if((input = reader.readLine()) == null) System.exit(0);  
            String[] temp_in = input.split(" ");  
            float temp_val = Float.parseFloat(temp_in[0]);  
  
            switch(temp_in[1].toLowerCase().charAt(0))  
            {  
                case 'c':  
                    inputTemp = new Temperature(temp_val);  
                    break;  
                case 'f':  
                    inputTemp = new Temperature(temp_val);  
                    break;  
                default:  
                    System.out.println("Invalid entry!!\n\n");  
                    continue;  
            }  
  
            System.out.print("Enter the unit to convert TO: ");  
            if((input = reader.readLine()) == null) System.exit(0);  
  
            switch(input.toLowerCase().charAt(0))  
            {  
                case 'c':  
                    outputTemp = inputTemp.toCelsius();  
                    break;  
                case 'f':  
                    outputTemp = inputTemp.toFahrenheit();  
                    break;  
                default:  
                    System.out.println("Invalid entry!!\n\n");  
                    continue;  
            }  
  
            System.out.println("\n The converted temperature is " + outputTemp.toString() + "\n\n");  
        }  
    }  
}
```

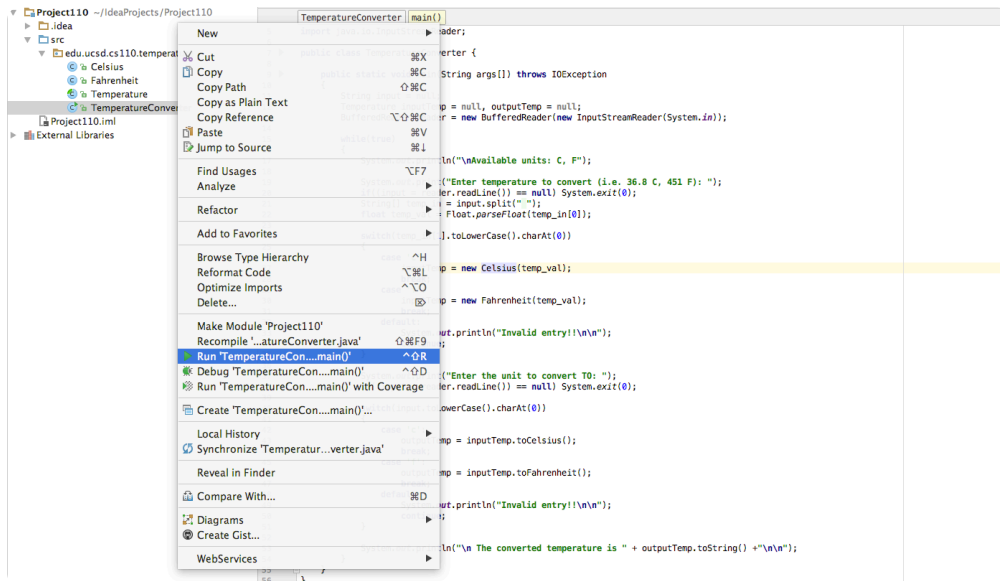
- One more error should remain. Remember how you have to instantiate subclasses in place of super classes? Let's try that by changing Temperature in our main method to either Celsius or Fahrenheit. The switch statement code should give you clues as to which subclass to use in each case.

---

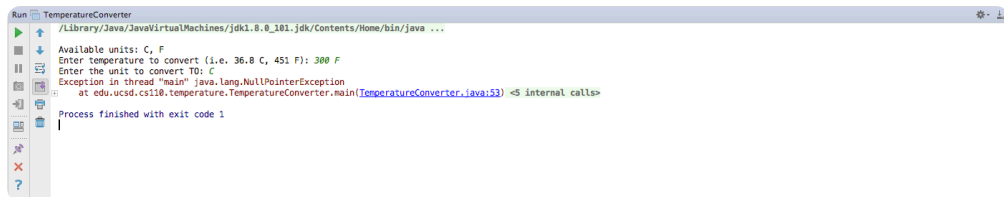
## Running your Java Project

- At this point all errors should be resolved. If any remain please resolve them before continuing further.
- Now right click on a blank space in the canvas and

select **Run temperatureCon..main()** or press **CTRL+SHIFT+F10**.



- Your code should prompt you to enter in a temperature and a C or F.



After you insert your input the console will return an error! It should say that there is a **NullPointerException**. This means that a method is called on an object that was not yet created. In order to fix this error, let's start debugging the code to see where the issue is at.

## In Progress Check

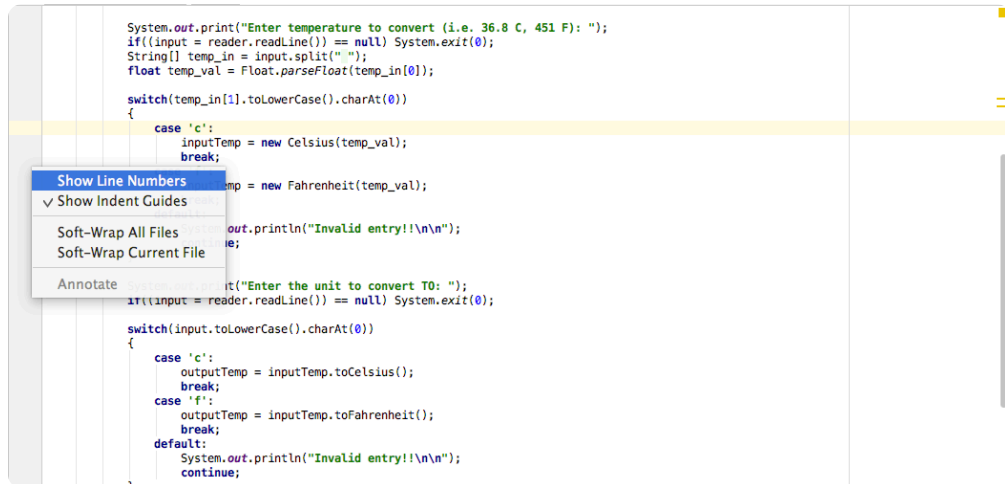
- When you reach the error message shown in the picture above, submit a Google Form as mentioned

on piazza. A member of the staff will verify that you have successfully installed IntelliJ IDEA and begun the coding process.

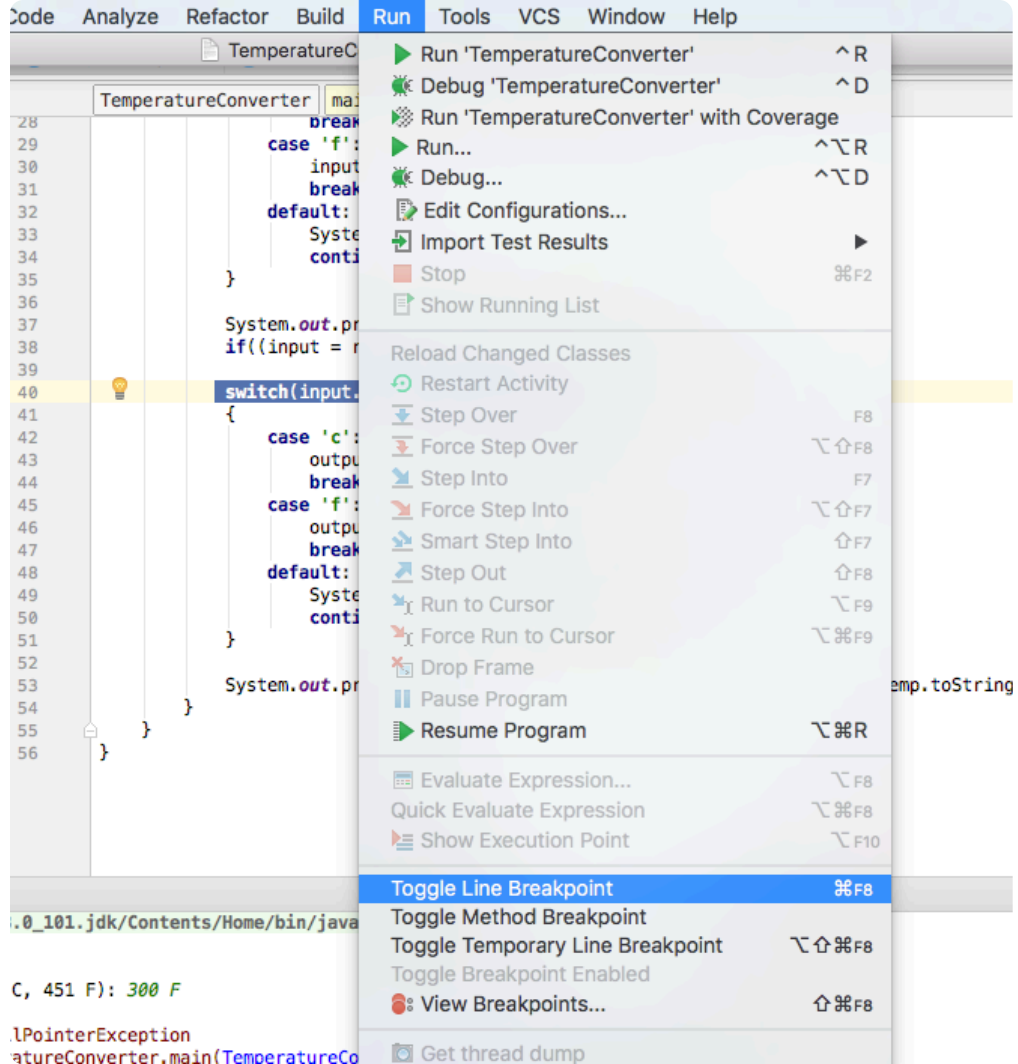
---

## Debugging a Java Project

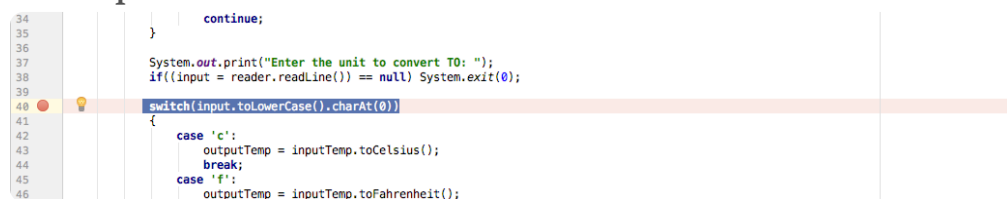
- At the moment, there is a `NullPointerException` in your code. To debug this, go roughly to line 40 in `TemperatureConverter.java` (The second switch statement). You can toggle line numbers in IntelliJ by right clicking on the left margin of the code block and selecting **Show line numbers**.



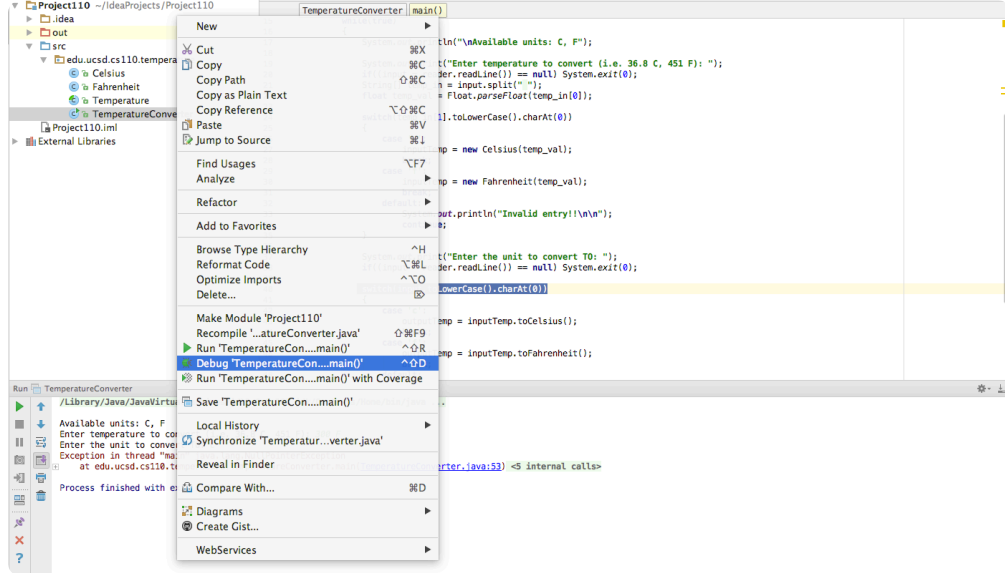
- Now, select this line with your cursor and set a break point by selecting **RUN ->Toggle Line Breakpoint** or pressing CTRL+F. It is also possible to add the breaking point by just left clicking on the left margin.



- A red dot should appear next to the line with a breakpoint set.



- Right click on the canvas and select **Debug temperaturCon...main()**.



- The code will stop at the selected breakpoint and all the variable values can be seen in the bottom right window. Please ensure that you actually enter in inputs at this step.
- You will see a series of tools at the top of the bottom left window which look like arrows pointing up and down. Step over F8, Step Into F7, and so forth. Take some time to play with these options with different breakpoints to familiarize yourself with their functions.
- Run the program with the above breakpoint, but this time press Step Over until you reach the line `outputTemp = inputTemp.toCelsius();` or `outputTemp = inputTemp.toFahrenheit();`. When you reach one of these lines press Step Into F7.
  - You will see that the outputTemp is currently set to null.
  - Make sure to follow the steps above for the other conversion (i.e. if you started with Celsius to Fahrenheit, try Fahrenheit to Celsius).



```
package edu.ucsd.cs110.temperature;

public class Celsius extends Temperature
{
    public Celsius(float t)
    {
        super(t);
    }

    @Override
    public Temperature toCelsius() {
        return null;
    }

    @Override
    public Temperature toFahrenheit() {
        return null;
    }

    public String toString()
    {
        // TODO: Complete this method
        return "";
    }
}
```

- Now that you have found the error using IntelliJ's debugger, time to fix it! But before you fix it, let's explore a more robust way to detect errors in your code using IntelliJ.

---

## Part 2: JUNIT Testing

- JUnit is a tool built into IntelliJ that is very efficient for testing. With JUnit, you can practice what is called *Test Driven Development* (or TDD). TDD is a coding practice where you write the tests for code before you write the code itself. When you write the new test, you want to make sure that it fails at first, then as you write the code you pass more and more tests, until you eventually pass them all. This helps a lot of programmers determine how the body of an algorithm should look before they even write it! TDD also helps to make sure that the new code you write does not break your previously written code, because all of those methods will have tests associated with them too. To help get this Temperature program working properly, we are going to provide you with some JUnit tests, which you will use, and then add to

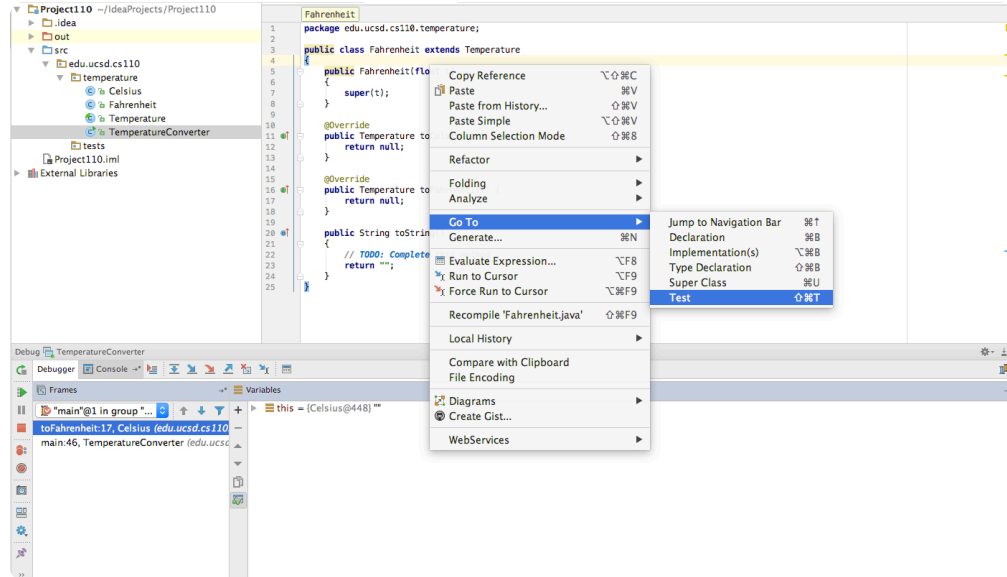
when we expand the functionality of the program.

- To begin, right click the **src** file in the left window.

Select **New -> Package**. Name the package

`edu.ucsd.cs110.tests`

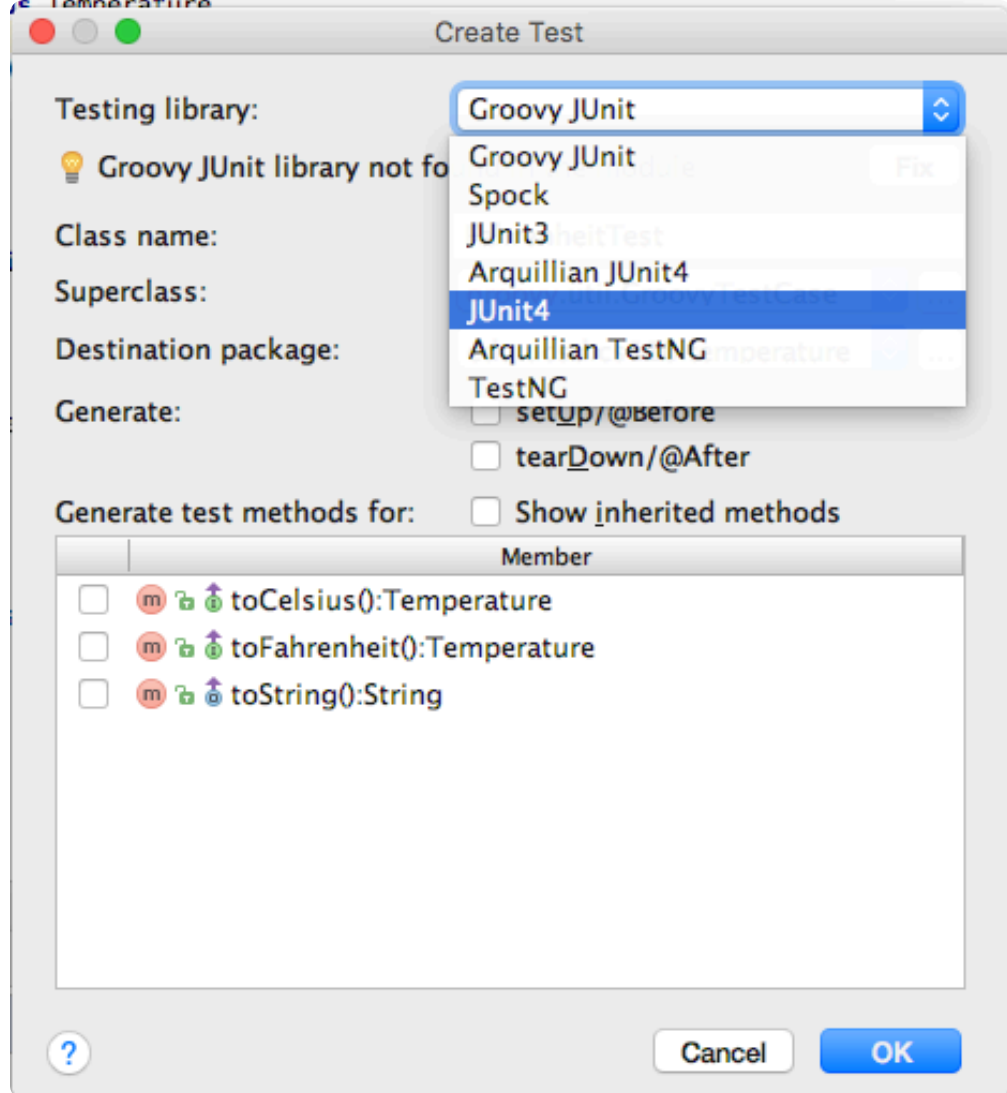
- Next navigate to the Fahrenheit class and select the line containing the `public class Fahrenheit extends Temperature {` declaration. **Right Click -> Go To -> Test -> Create new Test..**



- Once clicked select the **JUnit 4** option. If a pop up warning appears click **Fix** to the right of it to install JUnit 4. Set the Destination package to

`edu.ucsd.cs110.tests` by clicking on the 3 dots.

Once this is done click **OK** to create the Test.



Create Test

Testing library: JUnit4

⚡ JUnit4 library not found in the module Fix

Class name: FahrenheitTest

Superclass:

Destination package: edu.ucsd.cs110.tests

Generate:

- ☐ setUp/@Before
- ☐ tearDown/@After

Generate test methods for: ☐ Show inherited methods

	Member
<input type="checkbox"/>	toCelsius():Temperature
<input type="checkbox"/>	toFahrenheit():Temperature
<input type="checkbox"/>	toString():String

? Cancel OK

Choose Destination Package - edu.ucsd.cs110.tests

Hide path

edu.ucsd.cs110.tests

- ▼ <default>
  - ▼ edu
    - ▼ ucsd
      - ▼ cs110
        - temperature
        - tests

Cancel OK

After this, a class called FahrenheitTest will be created inside of `edu.ucsd.cs110.tests`. The newly created FahrenheitTest class might still complain about JUnit4 not being in the path, but this is an easy fix. Hover over the red line and, like before, a red lightbulb will appear. Click on the lightbulb and select the option that adds JUnit4 to the path of this project from the IntelliJ distribution. (If you cannot figure it out just ask a tutor)

- Now, copy paste the following code into the FahrenheitTest File:

```
package edu.ucsd.cs110.tests;
import edu.ucsd.cs110.temperature.Fahrenheit;
import edu.ucsd.cs110.temperature.Temperature;
import junit.framework.TestCase;

public class FahrenheitTest extends TestCase{
    private float delta = 0.001f;

    public void testFahrenheit(){
        float value = 12.34f;
        Fahrenheit temp = new Fahrenheit(value);

        assertEquals(value, temp.getValue(), delta);
    }

    public void testFahrenheitToString(){
        float value = 12.34f;

        Fahrenheit temp = new Fahrenheit(value);
        String string = temp.toString();

        String beginning = "" + value;
        String ending = " F";

        // Verify the suffix of the formatted str
```

```

ing
    assertTrue(string.startsWith(beginning));

    // Verify the prefix of the formatted string
ing
    assertTrue(string.endsWith(ending));

    // Verify the middle of the formatted string
ing
    int endIndex = string.indexOf(ending);

    // (Hint: what is the length of the middle of the string?)
    assertTrue(string.substring(0, endIndex).equals(beginning));
}

public void testFahrenheitToFahrenheit()
{
    Fahrenheit temp = new Fahrenheit(32);

    Temperature convert = temp.toFahrenheit();
;
    assertEquals(32, convert.getValue(), delta);
}

public void testFahrenheitToCelsius(){
    Fahrenheit temp = new Fahrenheit(32);

    Temperature convert = temp.toCelsius();
    assertEquals(0, convert.getValue(), delta);
);

    temp = new Fahrenheit(212);
    convert = temp.toCelsius();

    assertEquals(100, convert.getValue(), delta);
ta);
}
}

```

- Repeat the entire process for the Celsius class, and copy paste the following code below:

```
package edu.ucsd.cs110.tests;
import edu.ucsd.cs110.temperature.Celsius;
import edu.ucsd.cs110.temperature.Temperature;
import junit.framework.TestCase;

public class CelsiusTest extends TestCase{
    private float delta = 0.001f;

    public void testCelsius(){
        float value = 12.34f;
        Celsius temp = new Celsius(value);

        assertEquals(value, temp.getValue(), delta);
    }

    public void testCelsiusToString(){
        float value = 12.34f;

        Celsius temp = new Celsius(value);
        String string = temp.toString();

        String beginning = "" + value;
        String ending = " C";

        // Verify the suffix of the formatted string
        assertTrue(string.startsWith(beginning));

        // Verify the prefix of the formatted string
        assertTrue(string.endsWith(ending));

        // Verify the middle of the formatted string
        int endIndex = string.indexOf(ending);

        // (Hint: what is the length of the middle
```

```

e of the string?)
        assertTrue(string.substring(0, endIndex).
equals(beginning));
    }

    public void testCelsiusToCelsius()
    {
        Celsius temp = new Celsius(0);
        Temperature convert = temp.toCelsius();
        assertEquals(0, convert.getValue(), delta
);
    }

    public void testCelsiusToFahrenheit(){
        Celsius temp = new Celsius(0);

        Temperature convert = temp.toFahrenheit()
;
        assertEquals(32, convert.getValue(), delt
a);

        temp = new Celsius(100);
        convert = temp.toFahrenheit();

        assertEquals(212, convert.getValue(), del
ta);
    }
}

```

- Let's examine the code to get a better understanding of how it works. Open up your "CelsiusTest.java" file. There are a few things you should take a look at:
  - First, note the class inheritance: extends TestCase. TestCase is used to specify a test fixture which holds actual tests. If the word TestCase is in red, then junit may be missing from your project. Click on the word and press



ALT+Enter to resolve this.

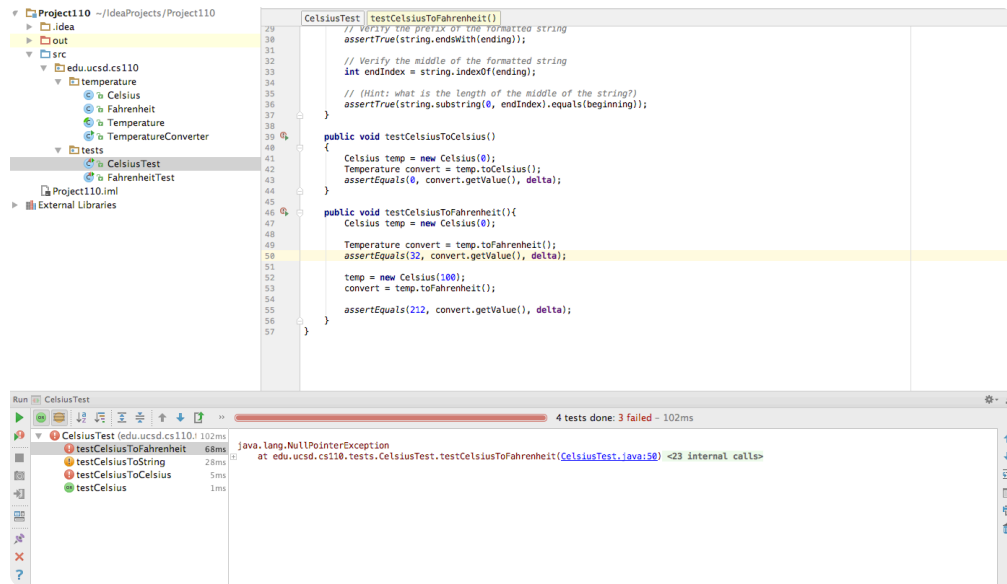
- Next, observe that each method begins with test. Prefixing a method with test indicates to the test fixture that the method should be considered a test case.
- Finally, each test case has one or more assertions. These method calls are used to verify expected results for a variety of types. For example assertTrue takes a boolean argument and verifies that it is true. assertEquals takes two parameters and verifies that one.equals(two) returns true.

**IMPORTANT:** Note the example

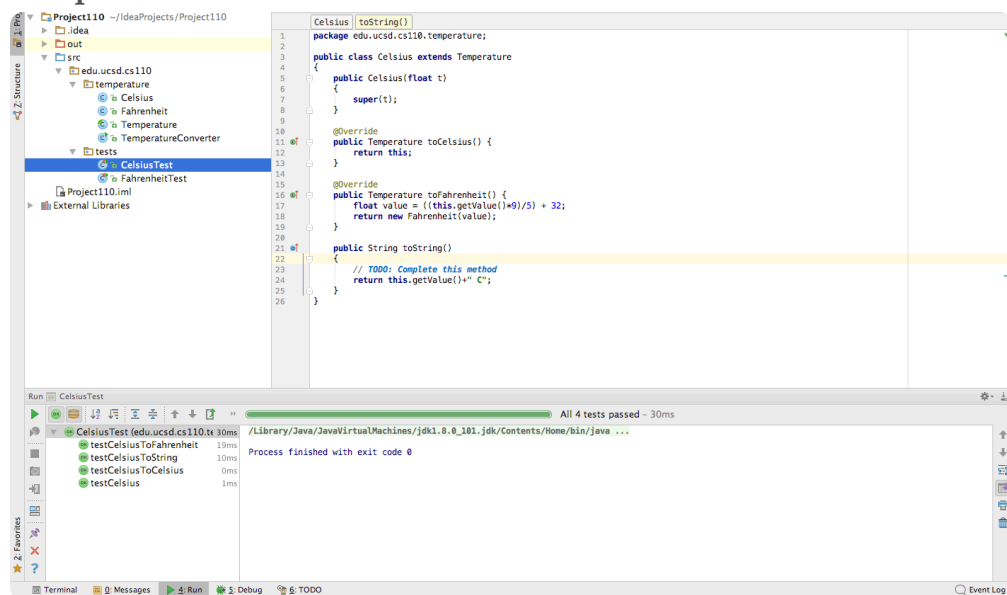
`testCelsiusToFahrenheit()` , which calls a three-argument assertEquals method. For floating point comparisons, you should always use this form of `assertEquals` , which takes a third parameter. (Our temperatures are floating point comparisons). Because precise equality between floating point numbers is not guaranteed, this form of assertEquals relaxes the definition of “equality” by allowing the two values being compared to differ by a small amount. This amount is defined as the variable delta in the above example, which indicates that you want the values to be equal to at least three decimal places.

- Now run CelsiusTest.java (Right click on the **CelsiusTest** class in the left window and click **Run**) and direct your attention to the lower left window. You should see 1 test passed and 3 tests failed. Double click on the first failed test

testCelsiusToFahrenheit. This will take you to the failed test case. It also displays a message as to why the test failed.



- Now you know there is something wrong with the convert object. Navigate to the method called that resulted in the failed assertion. This method should be in Celsius.java. Navigate to the Fahrenheit and Celsius classes and implement the methods there such that all the tests pass. Be sure to look at the tests being ran in order to understand the expected output!



- Note: Make sure to implement all the functions (

toCelsius(), toFahrenheit(), toString() ) for both Celsius and Fahrenheit classes.

Hint:  $F = ((C * 9) / 5) + 32$ ,  $C = ((F - 32) * 5) / 9$

- To run all the tests at once for convenience, right click on the **tests** folder and click **Run Tests in edu.ucsd.cs110.tests**
  - After you successfully complete the code and it passes all the test cases you should move to the third part of the lab: Travis CI.
- 

## Part 3: Travis CI

- About Travis CI

Travis CI is not a testing application itself, rather it serves as an application which helps developers automate whatever build process (which may include testing) that they would normally run when propagating a change from their local workstation to the released production-level codebase. This process is called continuous integration (CI). Examples of build managers for Java include Gradle (which Android Studio uses), Maven, and Ant. Ant is the simplest and the build manager you will use in this lab.

- Signing up for Travis CI

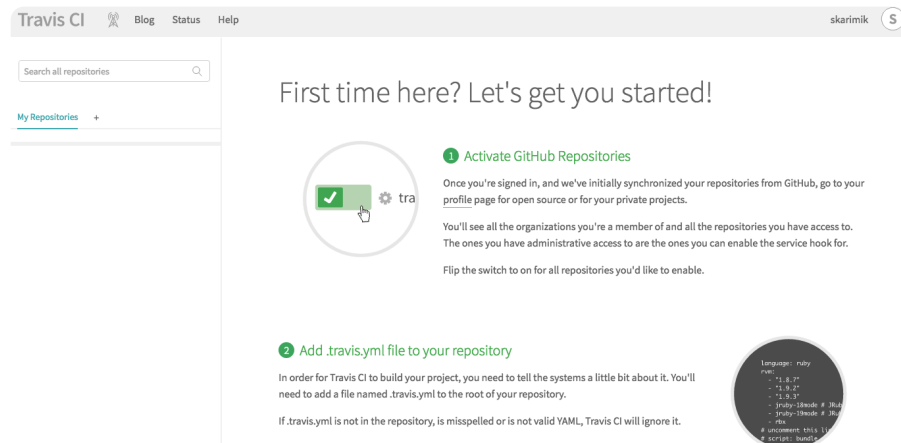
- First, you need to have a GitHub repository for this project, if you do not remember how to do that, refer back to the [git tutorial](#) where you learn to create a new GitHub repository **without** a README file. Please ensure that you initialize the git repo and set the upstream.
- Before you continue, make sure your project files are pushed onto your repository. In order to push your files to the GitHub repository, navigate to your working directory and then type the following commands in the Terminal:

```
git remote add origin https://github.com/USERNAME/REPOSITORY.git
git add .
git commit -m "message"
git push origin master
```

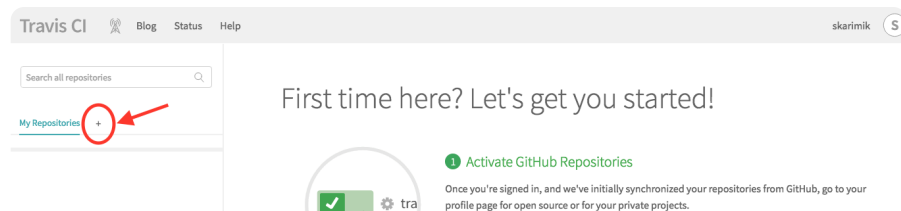
- **Note:** A common mistake is initializing the git repo in the wrong directory. Make sure you're initializing your git repo in the top level of your IntelliJ project. Whatever directory is listed at the top is the directory you should git init in, not the directory above nor any of the directories below.
- Visit [Travis web](#) and click **Sign in with GitHub** to link your GitHub account to Travis CI, this will make it so that you are able to utilize continuous integration on a GitHub repository.

# Test and Deploy with Confidence

Easily sync your GitHub projects with Travis CI and you'll be testing your code in minutes!








- **Note:** If you ever want to stop Travis from having access to your GitHub account, simply visit: [Revoker](#) and click on **Revoke**.
- Now you need to add your newly created repository to Travis CI, do this by clicking the + (plus symbol) on the left part of the main screen.



After you have a GitHub repository set, find your newly create repository and click the button next to its name to activate this GitHub repository for Travis. This will turn the button green and appear like below.

We're only showing your public repositories. You can find your private projects on [travis-ci.com](https://travis-ci.com).

## Legacy Services Integration

Filter repositories		
 2048-AI	<input type="checkbox"/>	<a href="#">Settings</a>
 AI-Projects	<input type="checkbox"/>	<a href="#">Settings</a>
 cse15L_Lab10	<input type="checkbox"/>	<a href="#">Settings</a>
 Lab5	<input type="checkbox"/>	<a href="#">Settings</a>
 Lab5VettingCSE110	<input checked="" type="checkbox"/>	<a href="#">Settings</a>

- Placing the necessary files in the project's folder
  - Use the terminal and change your current directory to be the directory of the CSE110 project. Then, download the necessary files from the public repository to this directory. This step requires you to clone a public repository provided for you in GitHub. To do so, please **follow the steps below very carefully, paying attention to flags:**

```
cd "To the root directory of the project"
git clone https://github.com/garygillespie/Travis.git
cp -r Travis/* .
cp Travis/. * .
rm -rf ./Travis
```

If warnings appear saying something similar to

```
cp: Travis/.git is a directory (not
```

`copied)` don't worry, this is intended.

**You just obtained several files, detailed below.**

**Verify these files exist in your directory.**

- **lib:**

These are the library files that Travis will need in order to build the code and test it, such as the JUnit class files.

- **.classpath:**

The classpath file contains src and target entries that correspond with folders in the project.

- **.travis.yml:**

This file is a basic Travis configuration file that tells Travis CI how our application should run during the build process and can be configured to use a specific version of the JDK, or even to use multiple JDK's to test your application for compatibility on many systems. **(This file always needs to be in the root directory of your GitHub repository for Travis to function correctly).**

- **README.md:**

This is the file that will contain the result of the process from Travis either build failed or

succeeded.

- **build.xml:**

This is an xml file which is part of a simple build system called Ant. This file is essentially the build's instructions or directions which Travis CI eventually runs in the cloud. This file specifies which testing framework to use for tests, which dependencies to load, and what the general flow of execution will be to complete the core build process.

- **IMPORTANT:** Before we push these files to our Repository, let's configure the `.travis.yml` to test using only the version of the JDK we used for the project. Edit the file by removing the extra JDK versions in the "jdk:" target. Also some TravisVM's don't have Ant installed, so we'll need to add some lines to ensure it is. Your file should look something like this:

```
language: java
jdk: oraclejdk11
before_install:
- sudo apt-get install -y a
nt
- sudo apt-get install -y a
nt-optional
```

- We now have the Travis config files and your project



source code in our directory. Let's push it to the GitHub repository you set up earlier.

- Using the username and repository name from before, type `git remote add origin https://github.com/USERNAME/REPOSITORY.git` and hit enter to add the remote you set up earlier to your local repo.
- Run `git remote -v` to verify that the remote has been set correctly.
  - If the remote isn't set as expected, use the following command to set the remote URL. `git remote set-url origin https://github.com/USERNAME/REPOSITORY.git`
- Enter the following command `git push -u origin master` to upload your files into your remote repo.

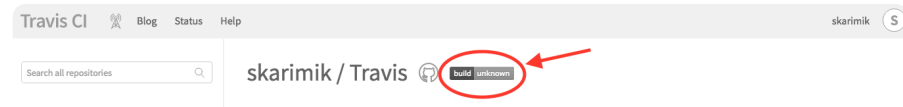
- **Embedding Status Image:**

- **a.** Now you are going to embed a status image (or Build Badge) within the README.md for your GitHub Repository which will show the status of your most recent Travis Build on your GitHub repository. The status image automatically updates whenever you load your GitHub repository main page or README.md (In reality, there can be up to a 2 minute delay but it is typically much faster). This makes detecting any push which causes the build to break much simpler to detect.
- **b.** For example, a Travis CI Build Badge which

has passed its most recent build will look like the image below.

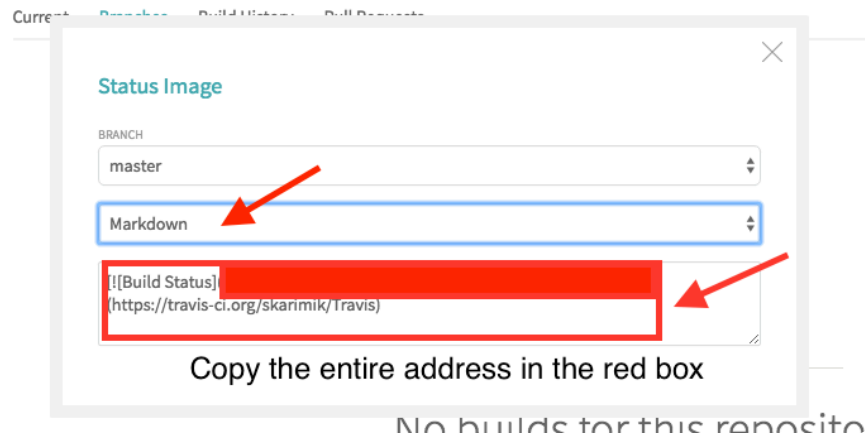
**build** passing

- o **c.** To get the status badge for your Travis CI Repository, navigate to [Travis CI](#) and click on the Build Badge which is to the right of your title header (Highlighted in the picture below with red).



- o **d.** Select the master branch and Markdown in the dialog box. Then, copy the code given in the Result box.

skarimik / Travis  **build** unknown



**Important Note:** Make sure to copy EVERYTHING inside the Result box to your README.md file. The image above contains an address that is colored over because it contains a private token.

- o **e.** Now, using Terminal inside the directory of your project run the command below to open the README.md and paste the code that you

copied from last step in this file.

**Note:** Make sure that you copy paste correctly.

```
vim README.md
```

Then, save and quit.

- Pushing the changes to GitHub:

In order to push the changes to the GitHub repository, type the following commands in the Terminal:

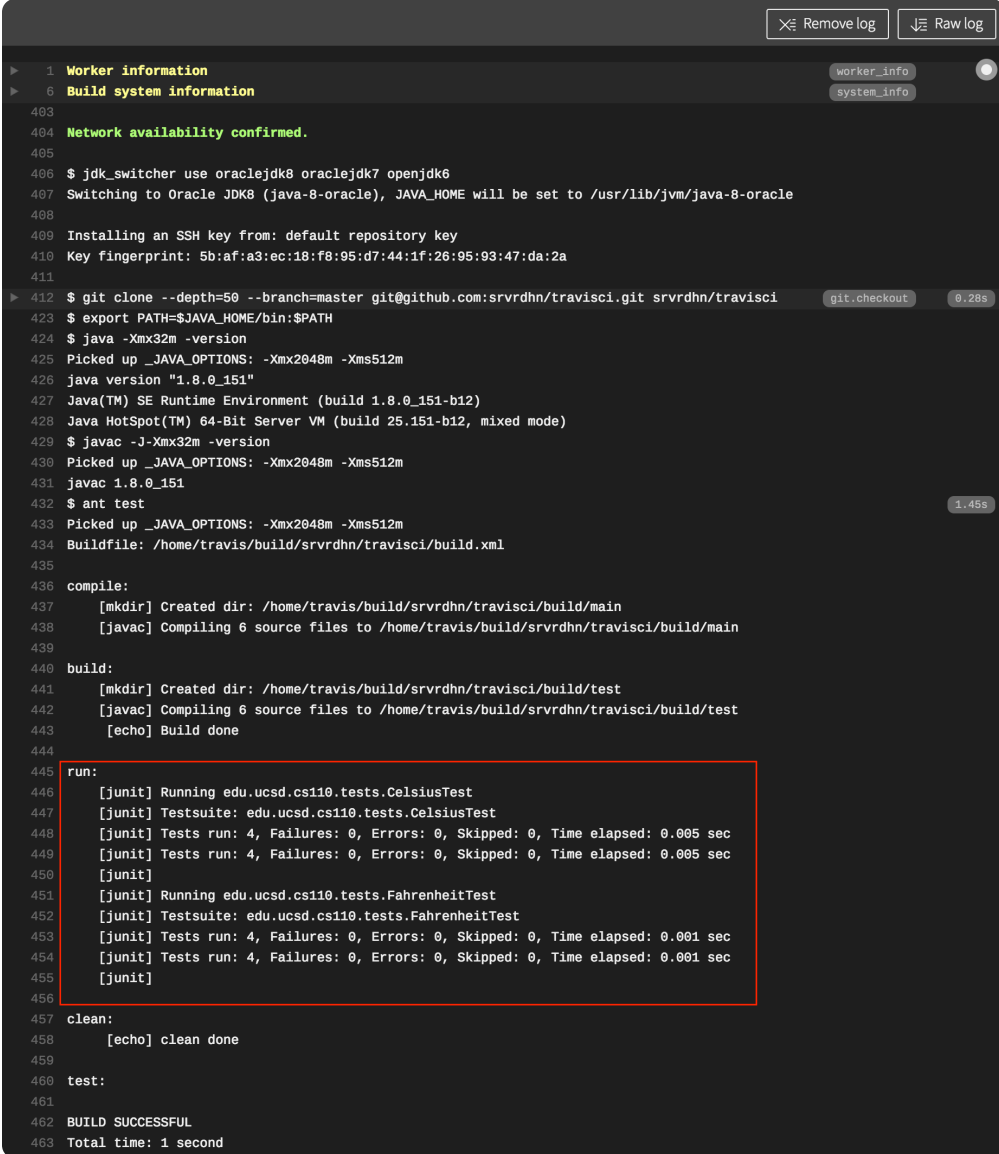
```
git add .  
git commit -m "message"  
git push origin master
```

- Travis should run and pass:

Given that the project is correct and debugged, Travis should not fail and it should produce a passing badge which will be placed inside README.md. Note that the process may take a while for the build to start and complete.

- If Travis is taking a while to rebuild your latest commit, you can force it to do so by going to your GitHub repo and doing the following: Settings -> Integrations & services -> click on Travis CI -> Test service (in the top right corner)

**IMPORTANT:** Ensure that TravisCI is actually running your tests by checking that the Job Log looks similar to the one shown under.



The screenshot displays a TravisCI job log for a build. At the top, there are buttons for 'Remove log' and 'Raw log'. The log is organized into sections: 'Worker information', 'Build system information', and 'Network availability confirmed'. The build process includes cloning the repository, setting up the Java environment (JDK 8), and installing an SSH key. The main build steps are: cloning the repository, setting the PATH, running 'java -Xmx32m -version', running 'javac -J-Xmx32m -version', and running 'ant test'. The test run section is highlighted with a red box, showing the execution of 'edu.ucsd.cs110.tests.CelsiusTest' and 'edu.ucsd.cs110.tests.FahrenheitTest', both of which passed. The build concludes with a 'clean' step and a 'test' step, resulting in a 'BUILD SUCCESSFUL' status. The total time for the build is 1 second.

```

1 Worker information
6 Build system information
403
404 Network availability confirmed.
405
406 $ jdk_switcher use oraclejdk8 oraclejdk7 openjdk6
407 Switching to Oracle JDK8 (java-8-oracle), JAVA_HOME will be set to /usr/lib/jvm/java-8-oracle
408
409 Installing an SSH key from: default repository key
410 Key fingerprint: 5b:af:a3:ec:18:f8:95:d7:44:1f:26:95:93:47:da:2a
411
412 $ git clone --depth=50 --branch=master git@github.com:svrdrhn/travisci.git svrdrhn/travisci
413 $ export PATH=$JAVA_HOME/bin:$PATH
414 $ java -Xmx32m -version
415 Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
416 java version "1.8.0_151"
417 Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
418 Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
419 $ javac -J-Xmx32m -version
420 Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
421 javac 1.8.0_151
422 $ ant test
423 Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
424 Buildfile: /home/travis/build/svrdrhn/travisci/build.xml
425
426 compile:
427 [mkdir] Created dir: /home/travis/build/svrdrhn/travisci/build/main
428 [javac] Compiling 6 source files to /home/travis/build/svrdrhn/travisci/build/main
429
430 build:
431 [mkdir] Created dir: /home/travis/build/svrdrhn/travisci/build/test
432 [javac] Compiling 6 source files to /home/travis/build/svrdrhn/travisci/build/test
433 [echo] Build done
434
435 run:
436 [junit] Running edu.ucsd.cs110.tests.CelsiusTest
437 [junit] Testsuite: edu.ucsd.cs110.tests.CelsiusTest
438 [junit] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.005 sec
439 [junit] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.005 sec
440 [junit]
441 [junit] Running edu.ucsd.cs110.tests.FahrenheitTest
442 [junit] Testsuite: edu.ucsd.cs110.tests.FahrenheitTest
443 [junit] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 sec
444 [junit] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 sec
445 [junit]
446
447 clean:
448 [echo] clean done
449
450 test:
451
452 BUILD SUCCESSFUL
453 Total time: 1 second

```

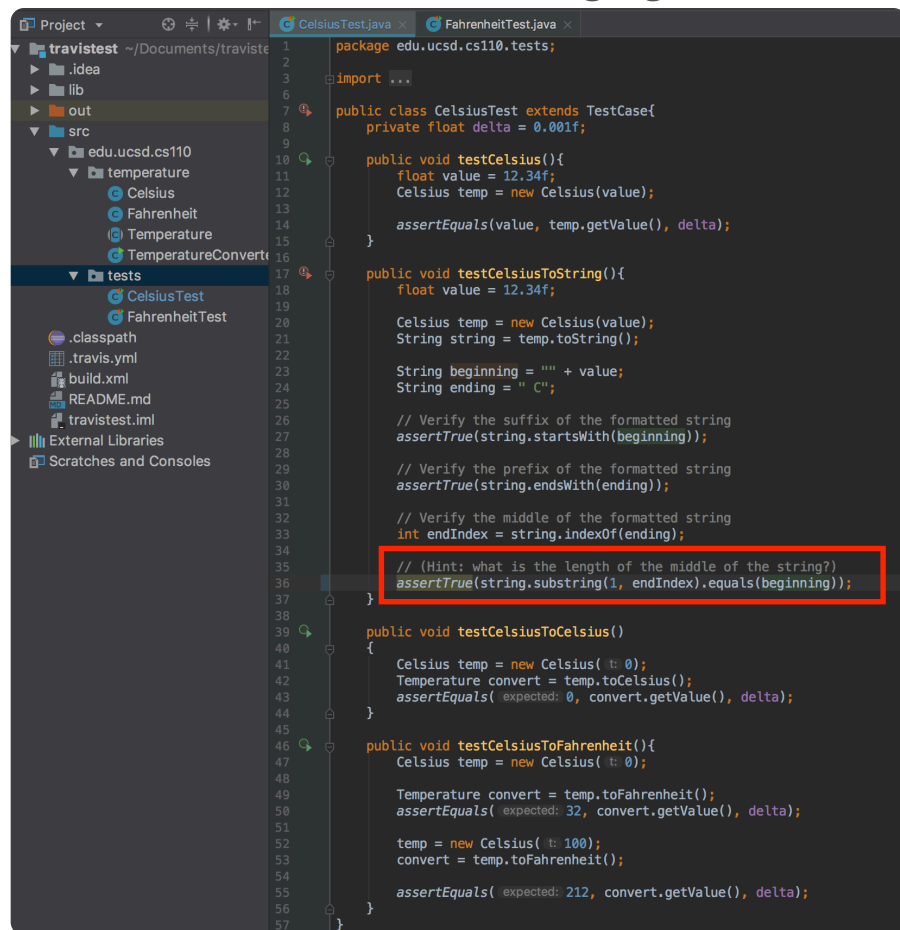
## • Failing tests in TravisCI

We have seen how TravisCI looks and works when all of our JUnit tests pass, however, as developers we know that this is not always the case.

A key feature of Travis is instant feedback whenever someone pushes a git commit causes tests to fail. In

this section, we will modify some of our tests to fail and cause TravisCI to complain.

- **a.** Lets go back to the CelsiusTest.Java class and modify the testCelsiusToString() method. We will change the index of the substring() method within the assertTrue() method from 0 to 1. Since this will return a string which we don't expect, the assertTrue() should fail. Be sure to save the file after changing it.



```
1 package edu.ucsd.cs110.tests;
2
3 import ...
4
5 public class CelsiusTest extends TestCase{
6     private float delta = 0.001f;
7
8     public void testCelsius(){
9         float value = 12.34f;
10        Celsius temp = new Celsius(value);
11
12        assertEquals(value, temp.getValue(), delta);
13    }
14
15    public void testCelsiusToString(){
16        float value = 12.34f;
17
18        Celsius temp = new Celsius(value);
19        String string = temp.toString();
20
21        String beginning = "" + value;
22        String ending = " C";
23
24        // Verify the suffix of the formatted string
25        assertTrue(string.startsWith(beginning));
26
27        // Verify the prefix of the formatted string
28        assertTrue(string.endsWith(ending));
29
30        // Verify the middle of the formatted string
31        int endIndex = string.indexOf(ending);
32
33        // (Hint: what is the length of the middle of the string?)
34        assertTrue(string.substring(1, endIndex).equals(beginning));
35    }
36
37    public void testCelsiusToCelsius()
38    {
39        Celsius temp = new Celsius( 0);
40        Temperature convert = temp.toCelsius();
41        assertEquals( expected: 0, convert.getValue(), delta);
42    }
43
44    public void testCelsiusToFahrenheit(){
45        Celsius temp = new Celsius( 0);
46
47        Temperature convert = temp.toFahrenheit();
48        assertEquals( expected: 32, convert.getValue(), delta);
49
50        temp = new Celsius( 100);
51        convert = temp.toFahrenheit();
52
53        assertEquals( expected: 212, convert.getValue(), delta);
54    }
55
56 }
57
```


- **b.** Follow the steps you performed recently to push the changes to your Github. Once these changes are committed, we will wait for TravisCI to notice the updated code in the repository and run the tests. Once TravisCI finishss running your unit tests, you will be presented with a log alerting you that your

tests have failed.

```
1 Worker information
6 Build system information

483
484 Network availability confirmed.
485
486 $ jdk_switcher use oraclejdk8 oraclejdk7 openjdk8
487 Switching to Oracle JDK8 (java-8-oracle), JAVA_HOME will be set to /usr/lib/jvm/java-8-oracle
488
489 Installing an SSH key from: default repository key
490 Key fingerprint: 5b:af:a3:ec:18:f8:95:d7:44:1f:26:95:93:47:da:2a
491
492 $ git clone --depth=50 --branch=master git@github.com:svrdrhn/travisci.git svrdrhn/travisci
493 $ export PATH=$JAVA_HOME/bin:$PATH
494 $ java -Xmx32m -version
495 Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
496 java version "1.8.0_151"
497 Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
498 Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
499 $ javac -J-Xmx32m -version
500 Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
501 javac 1.8.0_151
502 $ ant test
503 Picked up _JAVA_OPTIONS: -Xmx2048m -Xms512m
504 Buildfile: /home/travis/build/svrdrhn/travisci/build.xml
505
506 compile:
507 [mkdir] Created dir: /home/travis/build/svrdrhn/travisci/build/main
508 [javac] Compiling 6 source files to /home/travis/build/svrdrhn/travisci/build/main
509
510 build:
511 [mkdir] Created dir: /home/travis/build/svrdrhn/travisci/build/test
512 [javac] Compiling 6 source files to /home/travis/build/svrdrhn/travisci/build/test
513 [echo] Build done
514
515 run:
516 [junit] Running edu.ucsd.cs110.tests.CelsiusTest
517 [junit] Testsuite: edu.ucsd.cs110.tests.CelsiusTest
518 [junit] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.007 sec
519 [junit] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.007 sec
520 [junit]
521 [junit] Testcase: testCelsiusToString(edu.ucsd.cs110.tests.CelsiusTest): FAILED
522 [junit] null
523 [junit] junit.framework.AssertionFailedError
524 [junit] at edu.ucsd.cs110.tests.CelsiusTest.testCelsiusToString(Unknown Source)
525 [junit]
526 [junit]
527
528 BUILD FAILED
529 /home/travis/build/svrdrhn/travisci/build.xml:43: Test edu.ucsd.cs110.tests.CelsiusTest failed
530
531 Total time: 1 second
532
533
```

- c. Now that we have experience with both passing and failing tests, let's revert our changes to have the tests pass again. Undo the steps taken in part a, and be sure to commit and push your changes to your repository. This will cause TravisCI to run the tests again, but pass this time. Your Build History in TravisCI should look similar to this.

svrdrhn / travisci  build passing

Current	Branches	Build History	Pull Requests	More options
✓ master	fixed the break	✓ #3 passed	19 sec	
👤 Srivardhan Annam		👤 e857dae	about 4 hours ago	
✗ master	Make celsius test fail	✗ #2 failed	40 sec	
👤 Srivardhan Annam		👤 c994c28	about 4 hours ago	
✓ master	changed readme	✓ #1 passed	21 sec	
👤 Srivardhan Annam		👤 8ff3193	about 4 hours ago	

## • Completion Checkoff:

When your README.md file contains the passing badge, submit a Google Form as mentioned on piazza. Show the tutor your Travis CI Build History tab with your passing, then failing, then passing build.

Please take a moment to give your feedback regarding this lab at the this [link](#). We appreciate your feedback and we will use your feedback to improve the future labs.