LAB 0

Project Structure

Download `lab0-start.zip` from D2L. Unzip it to a proper location. Take a look at the structure of the project.

Folder	Content
`src/main/java/`	Where your Java source code would be
`src/test/java/`	Where your JUnit tests would be
`config/checkstyle/checkstyle.xml`	The file that specifies the code format, used by Gradle
`build.gradle`	Some basic configuration
`instructor-tests.gradle`	Allows you to run my tests on your code
`build/reports/checkstyle/main.html`	Style check report (after coding is done)
`build/reports/tests/test/`	Test reports of your tests (after coding is done)
`build/reports/tests/instTest`	Test reports of my tests (after coding is done)

Import the Project into Eclipse

- run `gradle eclipse` inside `lab0/`
- Launch Eclipse
- Unselect `Project -> Build Automatically` since we will build via Gradle
- Use `File -> Open Projects from File System ...` and find where the `lab0` folder is

Now you can see the project skeleton in Eclipse.

Formatting Settings in Eclipse

Go to `Eclipse -> Preferences -> Java -> Code Style -> Formatter` and `New` a new style based on `Java Conventions`. Click `Edit` and choose `Indentation` tab. Choose `Spaces only` for `Tab policy`. Set both `Indentation size` and `Tab size` to `2`. Save the new style as `SWE200`.

Go to `Eclipse -> Preferences -> General -> Text Editors`. Set `displayed tab width` to `2`. Check `Insert spaces for tabs`.

TIP: You can format the whole project by right-clicking the project in the explorer window and `Source -> Format`. It also helps to organize the imports by `Organize Imports`.

Running the Project

You'll see what project you are to write in the next section. Here assume you've finished writing code and tests.

- `gradle test` runs your tests on your project
- `gradle instTest` runs my tests on your project
- `gradle checkstyleMain` checks the code formatting of your source code
- `gradle checkstyleTest` checks the formatting of your tests

Each of those will generate reports. You'll need some of them for your submission. You do not need to submit check style report for your tests. You do not need to submit test report of your own tests.

A Simple Project

You will write a Java class that stores a temperature value and can return the stored temperature in Kelvin, Fahrenheit or Celsius unit. The package name we will use is `labZero`. This will cause a checkstyle error on purpose. The name of the class is `Temperature`. It has one and only public constructor `Temperature()` which takes no parameter.

It has three *getter* methods and three *setter* methods. They are:

- `double getKelvin()`
- `double getFahrenheit()`
- `double getCelsius()`
- `void setByKelvin(double k)`
- `void setByFahrenheit(double f)`
- `void setByCelsius(double c)`

I hope the names are self-explanatory.

Note that temperature cannot get below `0 K` (assuming `0 k` is valid). Whenever the Temperature instance is set to a value below that, a `TemperatureException` is thrown. If any getter method is used on a temperature object before it is set to a value, a `TemperatureException` is also thrown. For example, getting the value right after the constructor. Though it is more reasonable to make `TemperatureException` a `checked exception`, we will make it an `unchecked exception` for convenience this time. The implementation of `TemperatureException` is given as following:

```
public class TemperatureException extends RuntimeException {
}
```

Here are a few use cases that help clarify the requirements:

```
Temperature t1 = new Temperature();
double c = t1.getCelsius(); // should throw TemperatureException
```

```
Temperature t1 = new Temperature();
t1.setByKelvin(-1); // should throw TemperatureException
```

```
Temperature t1 = new Temperature();
t1.setByCelsius(-275); // should throw TemperatureException
```

```
Temperature t1 = new Temperature();
t1.setByKelvin(273.15);
double c = t1.getCelsius(); // c should be 0.0
```

```
Temperature t1 = new Temperature();
t1.setByFahrenheit(-40.0);
double c = t1.getCelsius(); // c should be -40.0
```

The Test Plan

The order of your tests in `TemperatureTest.java` must be the same as the order of the following plan (see grading part for the penalty). In the context of `JUnit`, the term *test* may refer to either a test method or a test class. We will use *test*

case for a single *test method* and *test class* for *test class*.

- 1 to 2 test cases for getting uninitialized temperature values
- 1 to 2 test cases for setting invalid temperature values
- 1 to 2 test cases for each of the following cases
  - set by Celsius and get by Celsius
  - set by Fahrenheit and get by Fahrenheit
  - set by Kelvin and get by Kelvin
- 1 to 2 test cases for each of the following cases
  - set by Celsius and get by Fahrenheit
  - set by Celsius and get by Kelvin
  - ...
- at least 3 test cases to show that
  - if a `Temperature t` is set by Kelvin value `k` and `f` is `t`'s Fahrenheit value, then setting `t` to `f` and getting the Kelvin value `k'` must equal to `k`.
- at least 3 test cases to show that
  - if a `Temperature t1` is set by Celsius value `c` and `t2` is set by `t1`'s Fahrenheit value, then `t2`'s Celsius value must equal to `c`

Testing for Exception and Floating-point Numbers

```
@Test
public void testUninitialized() {
  try {
    Temperature t = new Temperature();
    t.getCelsius();
    fail();
  } catch (TemperatureException e) {
    //This is expected
  }
}
```

```
private final static double DELTA = 0.01;

@Test
public void testFtoC() {
  Temperature t = new Temperature();
  t.setByFahrenheit(32);
  assertEquals(0.0, t.getCelsius(), DELTA);
}
```

Submitting Labs

PDF on GradeScope

Please make a single PDF file that follows the order below:

- *Instructor Sheet*
- *Test Requirement Sheet*
- *Self Check Sheets*
- Test report (Instructor's tests), including:
  - Top level (Classes not packages) report (index.html printed) Note: If any test fails, still use the "Classes" page, not the "Failed tests" page.

    - Leaf level reports (the pages that show the individual tests) Note: If any test fails, still use the "Tests" page, not the "Failed tests" page.
- Checkstyle report (main.html printed)
- Java source files for tests (or git-diff; will be explained in lab2)
- Java source files for application (or git-diff)
- Other materials (if needed by the specific lab)

Digital

You will use `gitlab.engr.ship.edu` for source code submission starting from Lab 4.

Grading

I'll grade your Lab 0 but it is NOT part of your final grade.

Late Policy

- 20% off within 24 hours after the deadline
- labs not accepted after 24 hours pass the deadline
- Submission time is usually set at 10a on the due date for both sections

Rubrics

All the upcoming labs will have the following structure.

- Self check sheet (5%)
- Functionality (Instructor's tests) (50%)
    - The first failing test will cost you 10%.
    - The remaining failing tests will cost you 5% *each*.
- Checkstyle (10%)
    - 2% per violation until the 10% runs out.
- Test implementation (30%)
    - Checking your pdf submission of the tests.
    - 3% for missing a test requirement
    - Detailed test requirement will be provided every lab. You must created tests that comply with the requirement.
    - The test cases (in each test class) must follow the order in the requirement description. (5%)
- Miscellaneous (5%)
    - e.g. pdf stack not following the expected order
    - items will be listed in the grading sheet

Important Notes

Any attempt to tamper the `build.gradle` file, the instructor's tests and/or the reports in your submission is considered serious academic dishonesty. In most cases, I will just look at the pdf. However, I may request your source code if I think your code and the test results may not match.