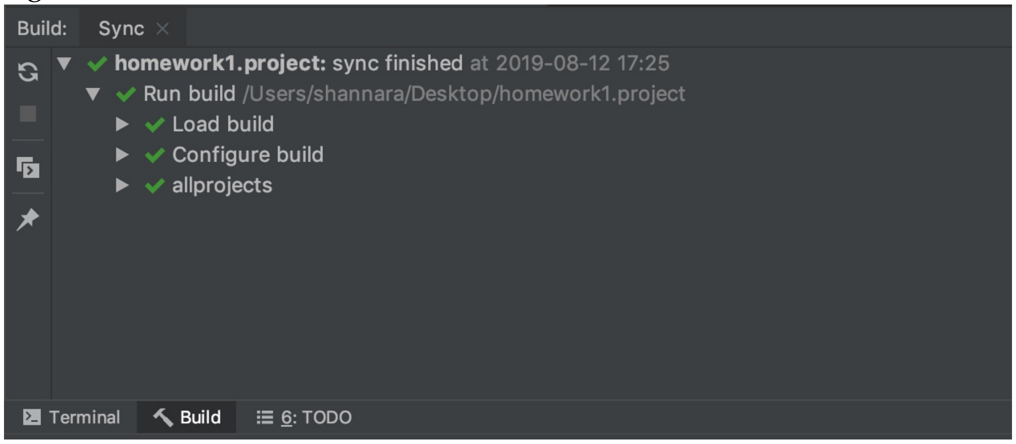
CS-420-1: Object-Oriented Design
Fall 2019
Northeastern Illinois University
Homework #1: Due Thursday, 09/05/19 at 5:00 p.m.
Interfaces

Opening/importing/running the required test code in IntelliJ

1. You **must** have Java 11 installed on your computer (do this first)!
2. You **must** have IntelliJ Community edition 1.4 (not 2.1 or higher) installed on your computer.
3. Unzip the provided project files into a directory of your choosing.
4. Open IntelliJ. Choose ”Import project” and select the unzipped project file folder (the one that contains all of the files).
5. Choose ”Import project from external model”, select Gradle and click Next.
6. The Gradle project name/location should be filled in. Leave it as is. The checkbox ”Create separate module per source set” should be checked and the radio button for ”Use default Gradle wrapper (recommended)” should also be selected. Make sure that the Gradle JVM is using Java 11. If not, you will either need to select that version or navigate to the JDK and find it (see possible locations below):
   Mac users: /Library/Java/JavaVirtualMachines/jdk-11.0.4.jdk/Contents/Home
   Windows users (probably): C:\Program Files\Java\jdk-11\bin”
7. Click Finish.
8. If this worked, you should see the following message in the Terminal at the bottom of IntelliJ. You may have to click ”Import Gradle changes” using a little pop-up found at the bottom right-hand corner of IntelliJ.



```
homework1.project: sync finished at 2019-08-12 17:25
  Run build /Users/shannara/Desktop/homework1.project
    Load build
    Configure build
    allprojects
```

9. You will need to enable Gradle as the TestRunner. Go to IntelliJ Idea (next to File - on the left) and choose Preferences. Then go to Build, Execution, and Deployment. Click on Gradle. Look for Delegate settings (beneath Gradle JVM). Choose Build and run using Gradle (not IntelliJ Idea) and choose Run tests using Gradle (not IntelliJ Idea). Click ok.
10. To view your project directory structure, go to View → Tool Windows → Project. You will see your project directory structure on the left-hand side of the screen.

11. All your Java code/files must be created inside the `src` → `main` → `java` folder.
12. All the test code is found inside the `src` → `test` → `java`. **Do not modify any of the code in the `test` → `java` folder unless you are uncommenting tests!**

Problem 1

- Create an interface named `Calculate` inside the `src` → `main` → `java` folder that has the following (right-click on the `java` folder and choose `New` → `Java Class`):
    1. An abstract method named `multiply` that takes two doubles as parameters and does not return anything.
    2. An abstract method named `add` that takes two doubles as parameters and does not return anything.
    3. An abstract method named `withinTolerance` that takes a `Calculate` type as the first parameter and a `double` as the second parameter (representing the tolerance). The method should return a `boolean`.
    4. An abstract version of the Object `toString` method.
    5. A default implementation of a method named `divide` that takes two doubles (numerator and denominator, with the numerator first) as parameters and returns a double. The method should throw an `ArithmeticException` with the message `"Divide by zero error"` if the result of dividing the numerator by the denominator is Infinite or NaN (hint: You may find a method from the Java API `Double` class helpful).
- Check that the default implementation was created correctly! Navigate to the `src` → `test` → `java` folder, open the `CalculateTest.java` file and uncomment the tests in that file. Then click on the `CalculateTest.java` file and choose `Run CalculateTest`. If this builds successfully, then the `divide` method in the interface was created correctly (Note: abstract methods cannot be unit-tested and I will review these manually). If the tests fail, you should scroll up in the terminal to see which tests failed and fix your interface to make the tests pass (look at the stack traces).
- You can run one individual test at a time by selecting the whole test (including the @Test annotation), right-clicking, and choosing Run <ClassName-TestName>.
- Create a concrete class named `CalculationResult` in the `src` → `main` → `java` folder that has/does the following:
    1. Implements the `Calculate` interface.
    2. A private instance variable named `result`.
    3. A properly named getter for the `result` instance variable.
    4. Implement the `multiply` method to add together the parameters and set the instance variable with the resulting value.
    5. Implement the `add` method to add together the parameters and set the instance variable with the resulting value.

6. Implement the `withinTolerance` method such that the method returns true if the `result` instance variables of object calling the method (`this`) and `other` are within the specified parameter tolerance of each other.
7. Implement the `toString` method to return the value stored in the instance variable in the following String format: `"Result: value"`.

- Go to the `src` → `test` → `java` folder, open the `CalculationResultTest.java` file, uncomment the tests and then run the tests in that file (similar to above). Fix your code so that all the tests pass.
- Go to the `src` → `test` → `java` folder, right click on the `java` folder and choose `"Run tests in homework1.project"`. All the tests in both the test classes must pass for your code to be correct.

Problem 2

You have been provided with a simple interface named `Vending`. You are going to redesign this interface to use a skeletal implementation. In order to do this, answer the questions below and place your name and the answers in the `questions.txt` file (found in `src` → `main` → `files`), properly enumerated for clarity.

1. Which method(s) in `Vending` will be your primitives in the interface and will have default methods?
2. Which method(s) in `Vending` cannot be primitives and must be implemented in the skeletal implementation class (i.e. the abstract class)? Why?

- In `src` → `main` → `java`, create an abstract class that is named correctly for a skeletal implementation.
- Modify the `Vending` interface so that the primitive methods have default implementations and create the other methods in the abstract class. Note that the method(s) that are not primitives should still be abstract method(s) in the interface, but implemented in the abstract class. See method descriptions below.
- The methods (regardless of where they are implemented) should do the following:
  1. `start`: Should return the String `"Starting process"`
  2. `stop`: Should return the String `"Stopping process"`
  3. `toString`: Should return the String `"Vending processes"`
- Once you have completed the changes to the `Vending` interface and the abstract class, go to the `VendingTest` class in the `src` → `test` → `java` folder and uncomment the tests. In order to have all the tests pass, the methods must be created in the correct locations, be spelled correctly (including the abstract class name), and follow the method descriptions.

Submitting your homework to D2L

- All the tests for all the problems in the test directory must be passing!
- In IntelliJ, under `File`, choose "`Export to Zip File`".
  Use the default name of `homework1.project.zip`.
- Submit this file to D2L.