Assignment Due Dates

1. Lab 0 Due: Wednesday, October 22, 2025 @ 11:59 pm EDT
   Lab 0 Late: Friday, October 24, 2025 @ 11:59 pm EDT (No Penalty)
2. HW 0 Due: Tuesday, October 28, 2025 @ 11:59 pm EDT
   HW 0 Late: Wednesday, October 29, 2025 @ 11:59 pm EDT (No Penalty)

You will be doing this lab and homework *individually*.

Recommended Background Knowledge

- ✓ Designing functions that consume inputs as parameters and return outputs using expressions
- ✓ Familiar with local variables and constants.
- ✓ Familiar with the arithmetic operations
- ✓ Familiar with Boolean logic, numeric comparisons, and if-expressions/statements
- ✓ Familiar with `String`s and string operations like concatenation
- ✓ Familiar with documentation (purposes) and unit testing (formerly `check-expects`).

Assignment Goals

- ✓ To become familiar with writing imperative code in Java
- ✓ To make sure you can write simple methods in Java
- ✓ To learn how to document methods with a Javadoc
- ✓ To become familiar with Java's standard library of arithmetic and string operations
- ✓ To be able to combine simple functions into more complicated methods (helper functions)
- ✓ To learn how to write and run tests for Java code (JUnit4)

The Assignment

Preliminaries

a. Using IntelliJ is recommended.
   - Using another IDE (Eclipse, VSCode) is up to you, and you will be responsible for figuring out how to use it correctly.

b. Use of JDK version 17 (Sept 2021) is recommended.
   - Versions newer than 17 may not work on the autograder.

c. You are encouraged to make a new project **for each** lab + homework assignment.

d. **You should write test cases in the lab individually** (but may collaborate on ideas with peers)

Overview

HW 0 asks you to define two classes for two different calculators in a stateless, procedural style (no fields).

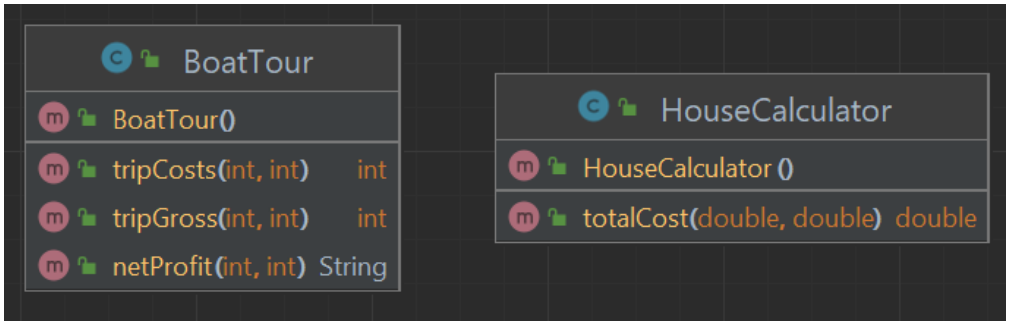
`BoatTour.java` defines the following methods for computing the costs, gross profit, and net profit for running fishing boat tours for some number of adults and kids.

- `tripCosts()` consumes an integer number of adults and an integer number of kids on the trip, and produces the amount the trip will cost us, as an integer, given that: adult supplies cost `$7` per adult, kid supplies cost `$5` per kid, and fuel for the boat costs `$1500` per trip.
- `tripGross()` consumes an integer number of adults and an integer number of kids on the trip, and produces the amount the trip will cost us, as an integer, given that: adult tickets are `$60` and kid tickets are `$45`.
- `netProfit()` consumes an integer number of adults and an integer number of kids on the trip, and produces a `String` of the net profit of the trip in the following formats: `"trip profit: $1234"` if the trip earned money or broke even, and `"trip profit: -$1234"` if the trip lost money. `netProfit()` must call `tripCosts()` and `tripGross()` to do the calculation.

`HouseCalculator.java` must define the following methods (in addition to any helper methods you create):

- `totalCost()` consumes a purchase price and a down payment amount (`double`s) and produces a dollar amount (a `double`). **You may assume the down payment is less than or equal to the purchase price.**
  The total cost is computed as the sum of the purchase price, simple interest, and bank fees. Simple interest is calculated as 8% * the principal * 30 (years). The principal is calculated as the difference between the purchase price and the down payment. The bank fees are $2500.
- You must define at least one **`protected`** helper method (function) and test the helper method as part of the homework. `totalCost()` must call this helper.

As UML



Lab 0

Summary

Write JUnit4-style tests for the HW 0 methods in a file called **`Examples.java`** and submit to Gradescope's autograder.

The purpose of this lab is to test your tests **before** starting the homework.

Prompt

You will do the following: the first two steps of the HtDP methodology from CS1101, adapted for writing object-oriented code in Java:

- **Step 1: Write class and method stubs** for **`BoatTour.java`** and **`HouseCalculator.java`** using correct class, function, and parameter names. Spelling, Capitalization (CapitalCase, camelCase, etc.) matter.
- **Step 2: Write Example unit tests** in a file called **`Examples.java`** using JUnit4 to test each method from HW 0.
  - Collaborate on test cases with your lab leader and peers
  - Come up with as many edge cases as possible

Using this **`Examples.java`** template, write test methods that test each of the `public` methods from the HW overview and UML. Fill in the `???`s to rename and specify example input/output for each test method. Add similar test methods as needed until all **`BoatTour`** and **`HouseCalculator`** methods are sufficiently tested.

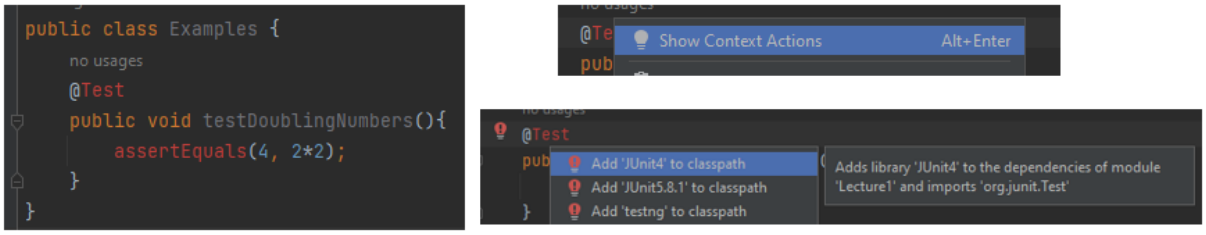
```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class Examples {

  @Test
  public void testBTCosts???() {
    BoatTour bt = new BoatTour();
    assertEquals(???, bt.tripCosts(???, ???));
  }

  @Test
  public void testTotalCost???() {
    HouseCalculator hc = new HouseCalculator();
    assertEquals(???, hc.totalCost(???, ???), 0.01);
  }

}
```

Do **not** submit tests for your **`protected`** **helper methods** to Lab 0, as the instructor's solution may not have those methods (and so the tests will not compile). You should test your helper methods as you complete HW 0 **after** Lab 0 is complete.

To make a new class file

- Right-click the `/src/` folder in your project view in IntelliJ
- Hover over **New** and click **Java Class**

- Name the class appropriately (Make sure it is CapitalCase)
- Make sure it has `public class ClassName { … }`
  - Where `ClassName` should match the file (and the prompt) exactly
- To stub out methods for **`BoatTour`** and **`HouseCalculator`**
  - Write their type signatures
  - Give them a body that returns a default placeholder value (like `0`, `""`, or `null` as appropriate)
  - You will replace this default body with actual code when you complete the homework.
- When making **`Examples.java`**, follow the prompts to import JUnit4 as a library by right-clicking the red syntax that needs to be imported, like `@Test`.



When finished with test cases, **`Examples.java`** should successfully **compile** locally (have no syntax or type errors), but the run tests will **not** pass on your stubs. This is to be expected, as you have not completed the homework yet, since the stubs don't *do* anything other than make sure the code is well-formed/spelled/typed correctly.

To turn in

- **Submit** only your **`Examples.java`** unit tests to Lab 0's Gradescope autograder
- Gradescope will run your unit tests on the correct instructor's solution
- Gradescope will **only** give you points if both:
  - **All** your tests pass on a correct solution (your tests are correct)
  - Your tests cover a significant % of the instructor's solution code

Notes

Lab helps you know that you understand the homework and that your tests are correct **before** implementing the rest of the homework.

While working on the homework, you are expected to run your tests from the lab to check if your code is correct. When attending office hours or asking questions on Piazza, you are expected to have example test cases that represent your question.

Common Problems in Lab Submissions

- **"I didn't get full points."**
  - **Example output from the previous term:**

1) Students' Examples.java should have 1 unit test for expenses(), one for revenue(), and one for profit() (6/10)

```
Test Failed: Student tests covered 66.666667% of the reference solution.
Coverage Breakdown:
expenses(): 1 out of 1 covered
profit(): 0 out of 1 covered
revenue(): 1 out of 1 covered

    at edu.wpi.lab0.Autograder.stu_test_reference_coverage:45 (Autograder.java)
...
Time: 0.002

OK (3 tests)
```

  - **Reason:** You didn't hit every conditional case possible in the instructor's solution.
  - **Fix:** Try more elaborate or different test cases.
- **"I got a 0, and the Autograder Output printed a bunch of stuff."**
  - **Example output from the previous term:**

Autograder Output

```
src/edu/wpi/lab0/correct/Examples.java:8: error: cannot find symbol
    assertEquals(486, new BoatTour().expenses(2, 2), 0.01);
    ^
  symbol:   method assertEquals(int,double,double)
  location: class Examples
src/edu/wpi/lab0/correct/Examples.java:13: error: cannot find symbol
    assertEquals(105, new BoatTour().revenue(1, 1), 0.01);
    ^
  symbol:   method assertEquals(int,double,double)
  location: class Examples
src/edu/wpi/lab0/correct/Examples.java:19: error: cannot find symbol
    assertEquals(-410, bt.revenue(1, 0) - bt.expenses(1, 0), 0.01);
    ^
  symbol:   method assertEquals(int,double,double)
  location: class Examples
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
3 errors
```

1) Students' Examples.java should have 1 unit test for expenses(), one for revenue(), and one for profit() (0/10)

```
Test Failed:     java.lang.ClassNotFoundException: edu.wpi.lab0.correct.Examples
    at jdk.internal.loader.BuiltinClassLoader.loadClass:641 (BuiltinClassLoader.java)
    at jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass:188 (ClassLoaders.java)
    at java.lang.ClassLoader.loadClass:525 (ClassLoader.java)
    at edu.wpi.lab0.Autograder.stu_test_reference_coverage:39 (Autograder.java)
```

  - **Reason:** Your code didn't compile.
  - **Fix:** Check that you have the imports correctly for the JUnit libraries.
  - **Fix:** Check that you named each function and class correctly.
  - **Fix:** Check that the parameter and return types are correct.
  - **Note:** `"Cannot find symbol"` in the output means something isn't imported or is misnamed. And it tells you what it is. In the example above, it doesn't know.
  - **Note:** Please read the entire error message and autograder output and break it down.

HW 0

1. **Implement** the method stubs in `BoatTour.java` to perform the expected computations.

   Rubric:
   [15 points] Autograder tests (only some are visible when submitting)
   [10 points] Documentation and parameter names
   - The class `BoatTour` is documented with a Javadoc style comment
   - All three methods are documented with Javadoc style comments
   - Parameter names are meaningful and in camelCase
   - `netProfit()` calls both `tripCosts()` and `tripGross()`

   [5 points] Magic numbers are stored in variables with good names in camelCase

2. **Implement** the method stubs in `HouseCalculator.java` to perform the expected computations.

   Rubric:
   [15 points] Autograder tests (only some are visible when submitting)
   [5 points] Protected helper method
   - Define a protected helper method for a subcomputation (like interest)
   - The protected helper method is tested in `Examples.java` with a unit test
   - `totalCost()` calls this helper method to do part of the calculation

   [10 points] Documentation and naming
   - The class `HouseCalculator` is documented with a Javadoc style comment
   - All methods are documented with Javadoc style comments (including helpers)
   - Parameter names are meaningful and in camelCase

To turn in

1. `BoatTour.java`
2. `HouseCalculator.java`
3. `Examples.java`

Note

You may not be shown all the autograder test cases until after the assignment is graded. Test your code locally using the lab's tests until you are confident in its correctness.

You should put your name in an `@author` tag in the class Javadoc comments.

Academic Integrity

Do not upload any part of this assignment prompt to an LLM like ChatGPT, and do not use CoPilot or IntelliJ's AI assistants for writing code for this course.

If you reference any resources other than those provided by the course that may make your code look suspicious to graders, you should cite their use in a comment at the beginning of your `Examples.java` file.

Reference

https://www.diveintheater.com/trip.html