Lab 1: Introduction to Java
12:00 PM, Jan 24, 2018

Contents

1 Welcome to CS 18 Labs! 1
2 Java Types 2
  2.1 Java Syntax 3
3 Getting Started 3
  3.1 Command Line 4
4 Ash's Awesome Aria 5
5 CS 18 Testing 5
  5.1 Testing Library 6
6 The Console 8
7 Working from Home 8

Objectives

By the end of this lab, you will know:

- basic Java syntax
- Java types

By the end of this lab, you will be able to:

- use Eclipse to write and compile Java programs
- use an editor to write and the command line to compile Java programs
- write tests for your Java programs

1 Welcome to CS 18 Labs!

Welcome to another exciting semester of Computer Science: An Integrated Introduction. CS 18 labs work in pretty much the same way that CS 17 labs did.
Here are some of the cool things you will do in labs this semester:

- Implement important data structures such as linked lists and graphs.
- Explore more sorting algorithms and other algorithmic techniques, like dynamic programming.
- Read and write from files and streams, and communicate with a server.
- Learn important design and testing principles, which will serve you well in all of your future CS endeavors (and we hope there will be many!).

**Our Expectations:** When working through CS 18 labs, you will often need to call over a lab TA to check over your progress and OK you to continue. At these checkpoints beginning with the next lab (lab 02), you must have code that has good style, header comments for each method you write, and full testing. If you call over a TA when your code is not finished according to these requirements, they will inform you what is missing, and you'll need to make the necessary changes before calling the TA back and getting the OK.

**“Just for Fun” Tasks:** At the end of many labs, you will see sections or tasks labeled as “just for fun.” If you reach those tasks before the end of lab, you must work on them. However, if you finish all the required tasks in time, you will still receive full lab credit without completing the just for fun tasks.

**Note:** For today’s lab only, you and your partner should each have your own computer, so you can both complete the setup you’ll need to work on assignments for the semester! However, other than that setup task, you should still do the lab with a partner.

**Task:** The first thing both you and your partner should do today is read the [CS 18 collaboration policy](#) and sign our form to agree to it. *If you do not fill out the collaboration policy form, you will not receive credit for this lab.*

## 2 Java Types

**Task:** For each Java expression below, write down the type of its value (*not* its value). If the expression could have more than one type, pick a valid one. If there is a type clash, explain why on a piece of paper or in a text file.

**Note:** For the problems containing functions, the types of the arguments and the return type have been removed and replaced with `??`. You may assume that whatever `??` signifies would not have caused a type clash.

1. 3
2. 3.0
3. 3 + 3.0
4. 3/2
5. `true`
6. "32"

7. 23 > 34

8. ((23 > 34) || ('A' == 'A')) && !(true && false)

9. The conditional:

```java
if (x == y) {
    return 2;
} else {
    return true;
}
```

10. The function:

```java
??? sumUpTo(??? n) {
    if (n == 0) {
        return 0;
    } else {
        return n + sumUpTo(n - 1);
    }
}
```

**Hint:** The last one isn’t just a number! Think about type signatures.

## 2.1 Java Syntax

**Task:** Correct the numerous syntax and type errors in the following Java expressions and function definitions. Your corrected code should do the same thing as the original was intended to do, while changing as little of the original code as possible.

```java
(+ 2 3)
sumUpTo 2

// Returns whether root is the square root of square.
boolean isSquareRoot(int root, int square) {
    return (root * root == square);
}

// Returns the product of two numbers.
double (double a, double b) {
    return a * b;
}

// Returns whether or not the input is a power of two.
int isAPowerOfTwo(int a) {
    if (a == 1) {
        true;
    } else if !isEven(a) {
```
"false";
} else {
    isAPowerOfTwo?(a / 2);
}

| You’ve reached a checkpoint! Please call over a lab TA to review your work. |

3 Getting Started

Now that you’ve seen the basics (don’t worry if you haven’t memorized Java syntax yet!), you are ready to get started developing Java code. You will be using two primary tools to develop Java code over the course of this semester: an IDE (Integrated Development Environment) called Eclipse and the command line accompanied by a text editor.

Task: Follow the [Eclipse Setup Guide](#) to set up Eclipse on the machine you are using for this lab. Be sure to import the tester library as well as following the basic setup steps.

The Eclipse Setup Guide is found on the course website, and includes directions for setting up Eclipse on your personal machine as well, should you need them.

You’ll notice that the `cs0180_setup` script you ran created a directory structure inside `~/.course/cs0180/workspace`. It is beneficial to understand how these directories are organized and interact.

First, inside the workspace folder, you’ll find the subdirectories `javaproject` and `scalaproject`. The former is for your Java programs, and the latter is for your Scala work.

Inside each of these subdirectories are three further subdirectories: `bin`, `src`, and `sol`. The first, `bin`, will be used to store byte code (or binaries) generated by the compiler. The second, `src`, will be used to store source code we provide to you. The third, `sol`, will be used to store solution files that you write.

Inside each of these subdirectories are further subdirectories corresponding to each assignment (e.g. `lab01`, `hw01`, `sparkzilla`, etc). Inside `src/assignment` and `sol/assignment`, there is again a subdirectory called `src` or `sol`, respectively.

In contrast, inside the `bin` assignment subdirectories are two further subdirectories: one for `src` binaries, and the other for `sol` binaries.
In effect: the files you create and write should be created in `workspace/javaproject/sol/assignment/sol`, until we switch to Scala, when your files will similarly be created in `workspace/scalaproject/sol/assignment/sol` (these are the directories you’ll be working with most!)

This structure may sound overly complicated, but it integrates well with Eclipse. Not to mention, this system separates the code you write from the code we give you, which will save you a few headaches in the long run. Don’t worry– you’ll be used to it in no time!

### 3.1 Command Line

As you may know from CS 17, an easy way to get started developing code is to use the command line and an accompanying text editor. We will not be using this method in this particular lab, but have written a Java-specific guide on this, located on our course website’s main page. This method will come in handy for particular projects in the course, but for the most part, we find Eclipse developing to be more convenient.

### 4 Ash’s Awesome Aria

Ash needs to put together a playlist to take on the Elite Four. He needs you to help him organize his music into classes following the special MBTA guidelines below. It is your job to give Ash the groove he needs to win!

**Task:** Construct a class to represent a song. Each song should have a name, an artist, and a length. Further, you should include a method called `getName`, which takes no arguments and returns the (String) name of a song; a method called `getLength`, which takes no arguments and returns the (double) length of the song; and a method called `isLongerThan`, taking a double and returning a boolean indicating if the song is longer than the given number.

To get started, in Eclipse, be sure that you have the Java perspective enabled (the J in the top right). Then, navigate to `javaproject > sol` and right-click `lab01`. Select New > Class, and name it `Song`, then click Finish.

**Hint:** Try approaching this one piece at a time: you’ll need to declare all the fields each Song should have, define a constructor for the Song class, and write the `getName` and `isLongerThan` methods.

Songs usually are part of a particular album. In this next part, you’ll add a new class (perhaps named `Album`) to represent an album, and add a field to your `Song` class to represent the album a
particular song belongs to.

An album should at minimum have a title, author, and genre. You should also write a method, `inGenre`, in the Album class that takes a (String) genre name and returns a boolean indicating if the album is in that genre.

**Task:** Create a new file for the Album class and modify your Song class.

**Hint:** To compare two `String`s, you can’t use `==`! You must use `.equals`. For example, to test equality between `s1` and `s2`, you could do `s1.equals(s2)`.

---

5 CS 18 Testing

By now, you’re surely pining for the design recipe you know and love. Never fear, for the design recipe is here to stay! Strictly speaking, the design recipe specifies that you should write your tests before you write any method. The only reason we did things backwards in this lab is because these exercises are meant to teach you the necessary syntax (not logic) to enable you to write methods in Java. Do not try this at home!

*In all your future programming endeavors, write your tests before you write any code!*

Further: **Do not fake your testing output.** Similarly, **do not omit tests on which your code fails.** Better for you to come clean about failures in your program than for us to detect them ourselves. You will lose points twice if you omit mention of known failures: once for missing tests, and again for missing functionality. If you indicate failures, you will lose points only for missing functionality.

5.1 Testing Library

In CS 18, you’ll be writing tests for all your assignments in tester classes, using a testing library called the Prima Tester library. This is the library you imported when setting up Eclipse earlier in this lab. If you have not done this yet, you *must* at this time in order to complete the lab.

Note that even after adding the tester JAR to the build path, you must still include it in each individual tester file, by including the following line at the top:

```java
import tester.Tester;
```

**Naming:** Recall that in Java, a file must have the exact same name as the class contained in that file (e.g. the Song class lives in the `Song.java` file). Similarly, all the tests that you write for Song must be in a class called SongTest, saved in a file named `SongTest.java`, in the same directory as `Song.java`.

In CS 18 (as in all your programming endeavors) you should test all of the methods you write. Do so using our tester library. Your tester class should include a method that contains all the tests corresponding to a single method you wrote (e.g. `testGetName` contains all tests for the method `getName`). Each of these methods must take as input a single argument `t` of type `Tester`. In addition, each test method’s names must begin with the word `test`. This is very important, as any method
whose name does not begin with test will not be run by the tester! Finally, your tester class will need a main method.

For example, suppose we have a class Arithmetic, containing methods add and divide:

```java
public class Arithmetic {
    public static int add(int num1, int num2) {
        return num1 + num2;
    }

    public static double divide(double num1, double num2) {
        return num1 / num2;
    }
}
```

The tester class for Arithmetic would look like this:

```java
import tester.Tester;

public class ArithmeticTest {
    public static void testAdd(Tester t) {
        // Tests for add go here
    }

    public static void testDivide(Tester t) {
        // Tests for divide go here
    }

    public static void main(String[] args) {
        // Main method
    }
}
```

Once you have created this structure, you’re ready to write some tests. Here’s where we take advantage of the power of the testing library. It provides two methods for us: `checkExpect` and `checkInexact`. These methods take as input an actual result—produced by a call to the method being tested—and an expected result. Then, the output of running the tester reflects whether the actual and expected values match. Specifically:

- `checkExpect` can work as usual, taking as input an actual and expected result, and passing if the two are equal
- `checkExpect` can also take in a single argument of type boolean. In this version, the test passes if the input evaluates to true, and fails otherwise.
- `checkInexact` takes a third argument, a double, representing the maximum acceptable error, or difference, between the two values. This method is analogous to check-within in Racket, and within in Pyret.

Using these methods, we can flesh out the example ArithmeticTest class:
import tester.Tester;

public class ArithmeticTest {

    public static void testAdd(Tester t) {
        t.checkExpect(Arithmetic.add(0, 0), 0);
        t.checkExpect(Arithmetic.add(1, 2), 3);
        t.checkExpect(Arithmetic.add(0, -16), -16);
    }

    public static void testDivide(Tester t) {
        t.checkInexact(Arithmetic.divide(0.0, 1.0), 0.0, 0.0001);
        t.checkInexact(Arithmetic.divide(17.0, 2.0), 8.5, 0.0001);
        t.checkInexact(Arithmetic.divide(-15.0, 3.0), -5.0, 0.0001);
    }

    public static void main(String[] args) {
        Tester.run(new ArithmeticTest());
    }
}

In addition to adding calls to checkExpect and checkInexact, we have also written a main method. You should copy this sample main method to your files, changing the name of the class from ArithmeticTest to reflect the name of the class that you are testing.

Task: Create a new class, called SongTest, and use it to test all of your Song and Album methods, building off the Arithmetic example above. In addition, use checkInexact to test the length field of your Song class.

Task: Run your test class by selecting the drop-down arrow next to the green play button at the top of the screen, and choosing Run As > 1 Java Application. You should see the corresponding message print to the console! Be sure all tests pass.

Note: The first time you run a particular file, you must do so manually, as described above. After this, you can simply press the green play button. This button does not run the file you have open, but rather, the last file you ran. Keep this in mind as you work with larger projects!

You’ve reached a checkpoint! Please call over a lab TA to review your work.

6 The Console

The tasks you have just completed show you what we expect in CS 18 in terms of testing. However, there are quicker methods to see results. While this is not encouraged, it can be useful in certain situations. For this reason, we'll introduce the println.

Task: Create a new class called FakeREPL, with a main method. In the main method, invoke a call to println, like this:

System.out.println("Almost done with the first lab!")

Eclipse Tip: Typing “sysout”, then ctrl + space, and then enter, will type System.out.println for you!
What this does is print a String to the console. Go ahead and invoke this method a few times in the main method!

**Task:** Run your program. Don’t forget to use the drop-down by the green button! Simply pressing the green button will run whatever you last ran, not the current file.

If it doesn’t automatically pop up, look for the Console tab at the bottom of your screen, or call over a TA to help you find it. You should see your messages printed out! Using `println` is one quick way to peek into your program, although you should always opt to use the tester instead. When you hand in any assignment in CS 18, extraneous `println`s should always be deleted.

### 7 Working from Home

If you would like to work on a personal computer, you can set up a home Eclipse environment exactly as you did your CS environment earlier in the lab. This guide can be found on the course website. In addition, if you would like to work using a text editor and compiling from the command line, a guide to do this (which applies both to the department machines and to a personal computer!) can also be found on the course website.

Once a lab TA signs off on your work, you’ve finished the lab! Congratulations! Before you leave, make sure both partners have access to the code you’ve just written.

**Note:** Before you leave, be sure both partners filled out the form agreeing to the collaboration policy, and have written your logins on the board.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS18 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/cs018/feedback](http://cs.brown.edu/courses/cs018/feedback)