Introduction to Java Programming

CSE/IT 213

NMT Department of Computer Science and Engineering

"Actually I made up the term 'object-oriented', and I can tell you I did not have C++ in mind. "

— Alan Kay

"Java, I don't care about. What a horrible language."

— Linus Torvalds (inventor of Linux)

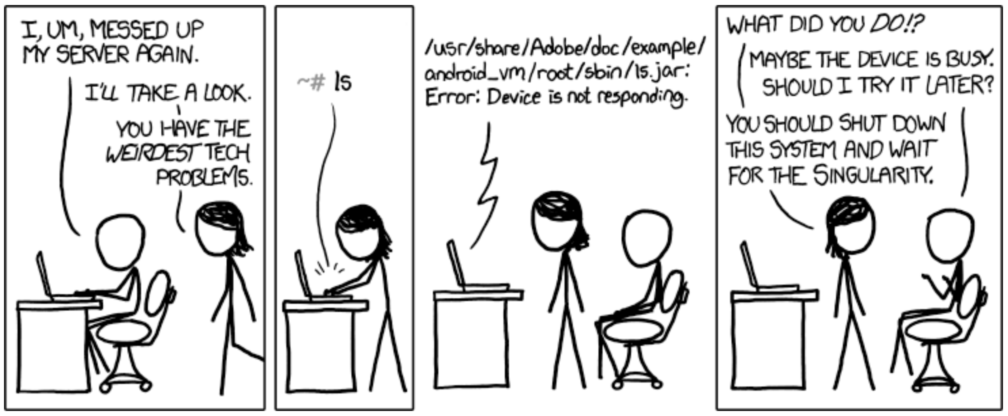

I, UM, MESSED UP MY SERVER AGAIN.
I'LL TAKE A LOOK.
YOU HAVE THE WEIRDEST TECH PROBLEMS.
~# ls
/usr/share/Adobe/doc/example/android_vm/root/sbin/ls.jar: Error: Device is not responding.
WHAT DID YOU DO!?
MAYBE THE DEVICE IS BUSY. SHOULD I TRY IT LATER?
YOU SHOULD SHUT DOWN THIS SYSTEM AND WAIT FOR THE SINGULARITY.

Figure 1: `https://xkcd.com/1084/`

Introduction

The main goal of this homework is for you to set up your Java programming environment, and to introduce you to features of Java that are already familiar from C.

Development Environment

Unlike in C, Java programs aren't compiled to native machine code. Instead, the program is compiled to a special byte-code which can only be run on top of the Java Virtual Machine (JVM). Therefore there are two separate packages needed for compiling and running code Java programs:

- The Java Runtime Environment (JRE) provides the JVM, which only lets you *run* a compiled Java program
- The Java Development Kit (JDK) is what you need to *compile* your Java programs into JVM byte code

For this class, you should make sure you have a recent version of JDK 8 installed (the JRE is included in the JDK). If you don't already have it, you can download the package for free from Oracle's website [1]. If you are running Linux, then you should use your system's package manager instead of downloading packages from their website. On Debian or Ubuntu you can install Java by running this command:

```
$ sudo apt-get install openjdk-8-jre openjdk-8-jdk
```

IntelliJ

IntelliJ is an Integrated Development Environment (IDE) that allows you to write, compile, debug, and package your programs using a single tool. Using an IDE can make it much easier to write large programs in Java. Other popular examples of IDEs include **Eclipse** [2] and **NetBeans** [3]. No particular IDE or text editor setup is required for this class, as long as you can build and submit your code in the required format.

This section will walk you through the setup for IntelliJ IDEA. Windows and Mac OS users can download and install the community edition from the developer's website [4]. If you are using a Debian based Linux distro, you can add the following line to the end of `/etc/apt/sources.list`:

```
deb http://ppa.launchpad.net/mmk2410/intellij-idea-community/ubuntu trusty main
```

[1] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[2] https://eclipse.org/downloads/
[3] https://netbeans.org/downloads/
[4] https://www.jetbrains.com/idea/download/

Then you can install IntelliJ by running the following commands:

```
$ sudo apt-get update
$ sudo apt-get install intellij-idea-community
```

If you are using another distribution that doesn't have a pre-built package, you can find instructions on their website to run the software manually [5].

The first time you start up IntelliJ, it may walk you through a quick dialog menu to customize the IDE. When everything is set up properly, you should see the following dialog box:

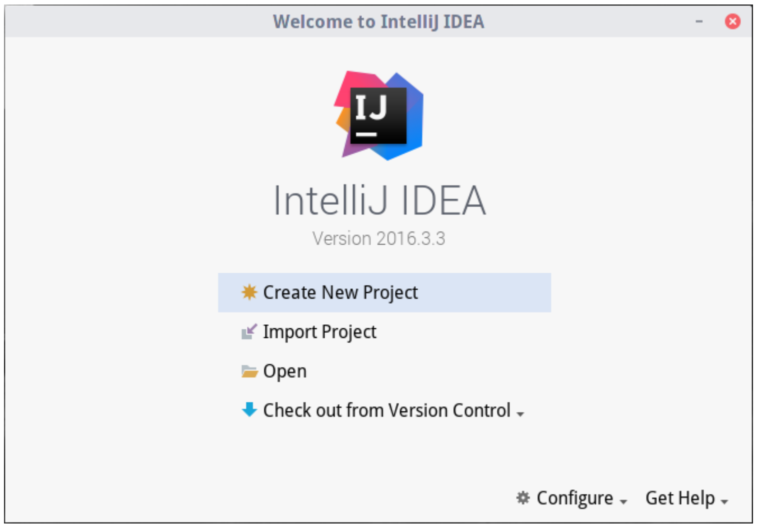


Figure 2: IntelliJ start up dialog

You can click "Create New Project" to get started on this assignment. On Linux you may need to manually set the Project SDK by selecting `New -> JDK` and providing the path to your JVM installation. That path will most likely be `/usr/lib/jvm/default`, but your mileage may vary. When that is set up, you can create the project "Homework 0" at whatever location you would like. You should see an empty project with the following layout:

[5] `https://www.jetbrains.com/help/idea/2016.3/installing-and-launching.html`

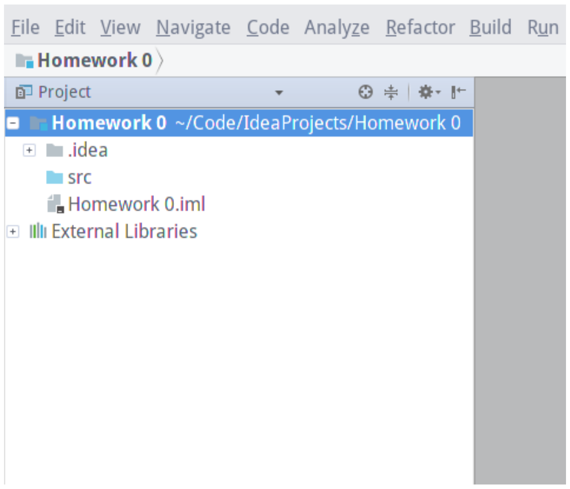

File Edit View Navigate Code Analyze Refactor Build Run
Homework 0
Project
Homework 0 ~/Code/IdeaProjects/Homework 0
.idea
src
Homework 0.iml
External Libraries

Figure 3: IntelliJ project window

The brand new project will contain nothing but an empty `src/` directory, where you will write your code. Each homework assignment will be a new Java project, and every project will consist of multiple *packages* – one for each individual problem.

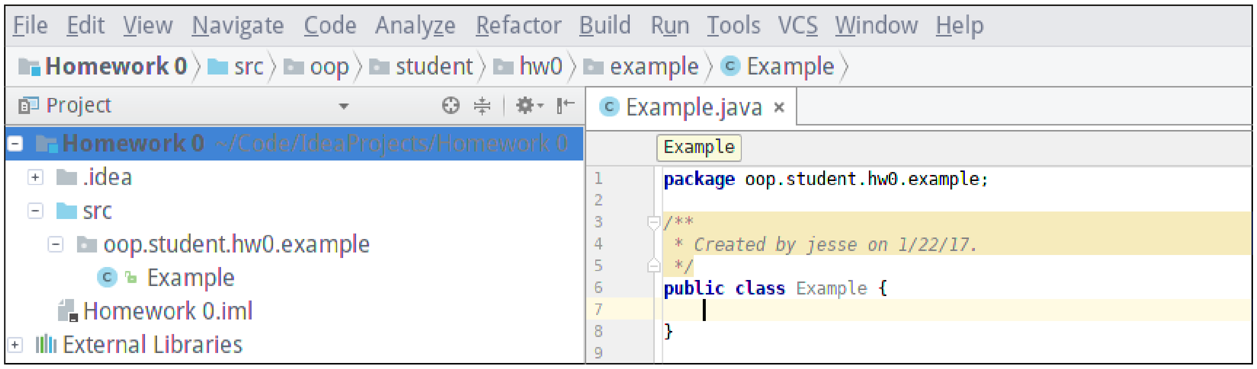
Packages and Classes

Packages are a way to group together programs that are all similar or rely on one another. All of the code you submit in this class should be contained within a package. For this class you will use the following naming scheme for Java packages:

```
oop.<first initial><last name>.hw<N>.<problem>
```

Where `<N>` is the homework assignment, (e.g., this is `hw0`), `<problem>` is the *spelled out* problem number. For example, Java Lovelace would name the package for the first problem on this assignment `oop.jlovelace.hw0.one` . We will follow this convention throughout the semester, as it makes grading easier. Package names are always lower case.

To create a package in IntelliJ, right click on the `src/` directory and select `New -> Package` . You can now add Java source code to the project by right clicking on the package and selecting `New -> Class` and giving a name for the new program. For example, if you created a new class called `Example` , then IntelliJ should show you code in the following layout:


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Homework 0 src oop student hw0 example Example
Project
Example.java
Homework 0 ~/Code/IdeaProjects/Homework 0
.idea
src
oop.student.hw0.example
Example
Homework 0.iml
External Libraries

```
package oop.student.hw0.example;

/**
 * Created by jesse on 1/22/17.
 */
public class Example {
    
}
```

Figure 4: A new class within an example project

When it comes time to test out your programs, you can click the green "Run" button in the upper right toolbar. Output should be displayed in a console at the bottom of the window.

Style Guide

Every Java file has the following organizational layout:

```
package oop.example.hw.problem;

import code.from.other.Libraries;

/**
 * Documentation...
 */
class ClassName {
    // code...
}
```

That is: the package name and import statements, followed by the class documentation and the class definition. Each file in Java is dedicated to defining a single *class*, which must have the same name as the file itself.

Each class, and every function or method defined within it, must be commented using the Javadoc style [6]. The basic format for class comments looks like this:

[6] http://www.oracle.com/technetwork/articles/java/index-137868.html

```
/**
 * A brief description of what the class does, 80 characters or less.
 *
 * <p> If necessary, a longer description of the class can go below the summary,
 * as long as the sections are separated by an empty line. Javadoc recognizes
 * <em>HTML tags</em>, if you want to include formatting. Separate paragraphs
 * with the "<p>" tag.
 *
 * @author Your Name
 * @version HW <N>, #<problem>
 * @bugs Use this tag to indicate any bugs or known issues with your submission.
 * If there are no bugs in your code, write "None."
 */
```

When you write your source code, it should be formatted according to the Google programming style guide for Java [7] [8]. The only exception is for block indentation – each new block should be indented by 4 spaces, (rather than 2, which is what Google suggests).

Java Basics

Java, like all object oriented languages, is centered around the concept of a `class`. A `class` in Java is similar to a `struct` in C; it stores a collection of variables in a single data record. The difference is that in Java classes are also where you define the functions that operate on that data. In object oriented programming, functions inside a class are called *methods*, and the variable members are called *attributes*. You define a method using about the same syntax you would use to define a function in C:

```
ReturnType methodName(Type arg, Type2 arg2) { ... return result; }
```

You'll find that much of the basic syntax of Java is already familiar to you, like `while` loops, `if ... else ...` statements, and `for` loops. Knowing C is really helpful for getting started solving simple problems in Java.

[7] If you are curious about the "right" way to format something, you can read the style guide here: `https://google.github.io/styleguide/javaguide.html`

[8] IntelliJ also has the ability to auto format your code (from the menu, select: `Code -> Reformat Code`). You can the configure formatter to use Google's settings by downloading `intellij-java-google-style.xml` from `https://github.com/google/styleguide/`. Under `File -> Settings` select `Editor -> Code Style -> Java` . You can import and edit the XML settings file by selecting `Manage -> Import` .

```java
public class Example {
    public static void code(int n) {
        // if/else statement
        if (n < 0) {
            System.out.println("n is negative");
        } else if (n == 0) {
            System.out.println("n is zero");
        }

        // while loop
        while (n <= 0) {
            n = n + 10;
        }

        // for loop
        for (int i = 1; i <= n; i++) {
            System.out.println(i + " is positive");
        }
    }
}
```

Figure 5: Familiar language features in Java

Main

Every Java program starts by calling the `main` method for the class you are running. Not every class needs to have a main method – there should only be one `main` per package. Other classes will be used as *libraries* within the main class. In most programs the declaration of the `main` method will look exactly the same:

```java
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello World!");
4     }
5 }
```

Figure 6: Hello World in Java, with `main` declared on line 2

The `public static` modifiers are mandatory for the main method of any program. The reasons why will be made more clear as you continue in this class. The return type `void` is another holdover from C; it tells us that the method doesn't return anything. The input argument is an array of strings, `(String[] args)`. This array is the Java equivalent of `(int argc, char *argv[])`; it stores any command line arguments passed to the program. Note that we can use one argument instead of two because Java arrays can keep track of their lengths automatically.

Input and Output

Printing to the console is usually done using `System.out.println`, which is like Java's version of `printf`[9]. Instead of using format strings, the `println` method can implicitly convert any variable to a string before printing it. It also remembers to add the new-line character (`"\n"`) for you. In the example program in Figure 5, (on line 17), you can see that Java even allows you to add an `int` to a `String` using the + operator, and print the resulting `String`.

In Java, `System.out` is an object containing a reference to the special file, `stdout`. As you may be able to guess, you can also read `stdin` using another object called `System.in`. The easiest way to parse input from the console is to create a `Scanner` to read the input stream for you. The example below shows you how scanners work:

```java
// You need to import Scanner before you can use it
import java.util.Scanner;
...
Scanner input = new Scanner(System.in);
String text = input.nextLine();
// This reads an integer from the console and converts it for you, but it will
// throw an exception if the input does not start with an integer
int num = input.nextInt();
```

Figure 7: Input Scanner example

Reading from `stdin` isn't a special case – you can use a `Scanner` to parse input from any file. In Java you can open a file using a `FileReader` if you want to read from the file, or a `FileWriter` if you want to write to the file. The following code shows some examples of how you can read and write to files in Java:

[9] You can also use `System.out.printf` if you want to use format strings similar to C's `printf`

```java
public void simpleIO() throws IOException {
    // the file reader will throw an exception if the file can't be opened
    FileReader file = new FileReader("/path/to/cool_file.txt");
    Scanner input = new Scanner(file);

    // Check if the we can read a line from the file
    if (input.hasNextLine()) {
        String line = input.nextLine();
        System.out.println(line);
    }

    // You can open a file for writing, and use a PrintWriter to write to it
    FileWriter hello = new FileWriter("/path/to/hello.txt");
    PrintWriter output = new PrintWriter(hello);

    // Print writers behave similar to the System.out.print* methods
    output.println("Hello World!");

    // Closes the Scanner/PrintWriter and the underlying files
    input.close();
    output.close();
}
```

Figure 8: Examples for file input and output

Problems

All of your solutions on this assignment should be written within the main method of each class. Although this would generally be considered bad style, each of these problems is simple enough to be solved quickly using only a few dozen lines of code.

Be sure to follow the guidelines for package naming, coding style, and documentation for your classes. You can find more relevant background information in Chapter 3 of *Core Java*.

Problem 1: Fizz Razz Buzz

This is a variation of the children's counting game, Fizzbuzz. The problem is to count from to one hundred, but to replace certain numbers with words like Fizz, Razz, and Buzz. Write a program called `Fizz.java` that loops over each number from 1 to 100, and does the following:

- If the number is a multiple of 3, print "Fizz"
- If the number is a multiple of 4, print "Razz"
- If the number is a multiple of 5, print "Buzz"
- If the number is a multiple of 3 and 4, print "FizzRazz"
- If the number is a multiple of 4 and 5, print "RazzBuzz"
- If the number is a multiple of 3 and 5, print "FizzBuzz"
- If the number is a multiple of 3, 4 and 5, print "FizzRazzBuzz"
- Otherwise just print the number

Example output:

```
1
2
Fizz
Razz
Buzz
Fizz
7
Razz
Fizz
Buzz
11
FizzRazz
13
14
FizzBuzz
Razz
...
```

Hint: You can use `System.out.print` to print a `String` to the console *without* adding a newline character to the end.

Problem 2: Rock Paper Scissors

This is another problem inspired by children's games. Both players choose one of "Rock", "Paper", and "Scissors" to play against each other. Rock beats Scissors, Scissors beats Paper, and Paper beats Rock. Write a Java program called `RPS.java` that plays Rock Paper Scissors against the user in a `while` loop.

Prompt the user for their move, and use a `Scanner` to read their input from the console. Your handling of the input should be totally case insensitive, so that `"rock"`, `"Rock"`, and `"r0cK"` will all count as the same play. If the user enters an invalid string, print `"Error!"` and prompt the user to correct their move. Then use `System.out.println` to print the computer's move in response. After each round print out whether the winner was the player or the computer, then start over. The computer's strategy is simple: play `"rock"`, then `"paper"`, then `"scissors"`, and repeat. Keep playing until the computer wins two rounds in a row.

Example output:

```
Your move? [rock/paper/scissors]: paper
> Rock!
Player wins!

Your move? [rock/paper/scissors]: lizard
Error! Please enter "rock" "paper" or "scissors": scissors
> Paper!
Player wins!

Your move? [rock/paper/scissors]: scissors
> Scissors!
Draw!

Your move? [rock/paper/scissors]: Scissors
> Rock!
Computer wins!

Your move? [rock/paper/scissors]: r0Ck
> Paper!
Computer wins!

Good game!
```

Hint: To check if two strings are equal, you need to use the `.equals` method, rather than the == operator. So you should use `move.equals("paper")` instead of `move == "paper"`. Java 8 also allows you to `switch` on strings.

Problem 3: Guessing Game

Write a program called Guess.java, that implements the guessing game, "I am thinking of a number between x and y."

Use a Scanner to prompt the user for the lower and upper bounds of the number they're thinking of, and correct the error if $y < x$ (i.e. swap the numbers). Use a binary search algorithm to iteratively guess what number they are thinking of. On each guess, prompt the user to see whether your guess is less than, greater than, or equal to their number. When the user responds that the guess was correct, print the number of guesses the program made and then exit.

Example output:

```
Think of a number!
Lower bound: 1
Upper bound: 100

Is it 50? [lt/eq/gt]: lt
Is it 25? [lt/eq/gt]: gt
Is it 37? [lt/eq/gt]: eq

It took 3 guesses to find the number 37!
```

Problem 4: Alice in Wonderland

On Canvas there is a file called alice.txt, which contains the text of Lewis Carroll's *Alice's Adventures in Wonderland*. Download the file and import it into the root of your Homework 0 project directory.

Write the program Alice.java to process the text of this file. Use a Scanner to read lines from alice.txt, and write code to modify the file. Translate all of the odd numbered lines to all lower case, and all of the even numbered lines to all upper case, skipping any empty lines. That is, if the line before a blank line is lower case then the next line after the blank should be upper case, regardless of the offset line number. Leave the blank lines in your output file so that the formatting of the text remains the same.

Write your output to a new file called alice_in_mixed_case.txt. Use a PrintWriter object to write your translated output to the file.

Example output:

```
chapter i. down the rabbit-hole

ALICE WAS BEGINNING TO GET VERY TIRED OF SITTING BY HER SISTER ON THE
bank, and of having nothing to do: once or twice she had peeped into the
BOOK HER SISTER WAS READING, BUT IT HAD NO PICTURES OR CONVERSATIONS IN
it, and what is the use of a book, thought alice without pictures or
CONVERSATIONS?

so she was considering in her own mind (as well as she could, for the
HOT DAY MADE HER FEEL VERY SLEEPY AND STUPID), WHETHER THE PLEASURE
of making a daisy-chain would be worth the trouble of getting up and
PICKING THE DAISIES, WHEN SUDDENLY A WHITE RABBIT WITH PINK EYES RAN
close by her.

...
```

Hint: You can use `while (input.hasNextLine()) { ... }` to loop over every line in a file. A blank line contains only whitespace characters, which means `blankLine.trim()` will compare equal to an empty string.

Submission

Create a TAR archive containing all of the packages and source files for your project. You can do this from the Linux command line by navigating to your `IdeaProjects/` directory and running:

```
$ tar -czvf cse213_<firstname>_<lastname>_hw0.tar.gz "Homework 0/"
```

Windows users can acheive the same thing using the tool 7-zip [10]. Make sure you use the naming scheme above, and that the most recent versions of all your source files are included in the archive. Upload your submission to Canvas before the due date.

[10] `http://www.7-zip.org/download.html`