Debugging Lab

Due: June 14th - 16th (hand in by your next section)

1 Overview

Welcome to the debugging lab! Bugs are the worst. They hinder our progress on projects and sometimes make our programs behave in ways we don't understand. Our goal for this lab is to demystify some common bugs, introduce some tools to help understand them better, and put those tools into practice by debugging some code. We will guide you through debugging one program, and then have you practice what you've learned by debugging another program on your own.

2 Installing and Handing in

1. Install four files: `Athlete.java`, `Competition.java`, `Ranking.java`, `Vaulter.java` in your `debugIntro` directory from GitHub. Click **here** to get the files.

2. To "hand in", have a TA check that you're done in your section or later at TA hours. Remember that the due date is one week after you start the lab in section - complete it and hand it in by your next section. This means each person's due date depends on their section time.

3 Common Errors and Exceptions

3.1 Java Errors and Exception

You've probably run into a lot of these errors in CS15, but we wanted to give you a refresher on these bugs and where to look first when you encounter them.
The first thing to keep in mind is that there are two main classes of errors in a Java program: compiler errors and runtime errors.

Compiler errors occur when the Java compiler detects something unusual when trying to compile your code, before your program is even run. This is often a syntax error, an uninitialized variable, or a similar problem that simply doesn't follow the Java rulebook.

Runtime errors occur after you have successfully compiled your code, but encounter a problem when you try running through your program. Runtime errors are often harder to detect and resolve because they often depend on the value of a given variable at runtime.

1. **NullPointerException** - This usually occurs when you try to use something that hasn't been initialized or instantiated. Here are some possible causes:

- Calling a method on an uninitialized variable - You should check that every variable was properly initialized before being used, (especially instance variables since they can be easy to miss). For example, calling a method on `team` will throw a NullPointerException if you have not initialized it with a value:

  ```
  Team team;
  team.getAthlete();
  ```

  Fix: make sure to initialize `team` with a value.

- Trying to access a method/field of a null object - This can also happen when accessing null objects in arrays. For example, if `teams[0]` is null, calling a method on it will result in a NullPointerException too.

  ```
  Team[] teams = new Team[3];
  teams[0].getScore();
  ```

2. **ArrayIndexOutOfBoundsException** - This happens when you try to index into a position of an array that does not exist. The following code, for example,

   ```
   Country[] countries = new Country[5];
   countries[5] = new Country("USA"); // error here!
   ```

   will throw a `java.lang.ArrayIndexOutOfBoundsException:5` error. This is because Java (and Python) arrays are indexed by 0, this means the largest possible index for the `countries` array is 4, not 5!

   Additionally, here are some more causes of ArrayIndexOutOfBoundsExceptions:

   - Off by one errors - By far the most common reason for this exception is miscounting by one. For example, you might have an array of size 10, and accidentally have a for loop that iterates from 0 up to and including 10. This would result in an exception on the last pass of your loop.
   - Calculation errors - In a similar vein to before, you might accidentally calculate an index incorrectly, causing it to be negative or too large. This is especially common while working with problems where you have to access certain array entries, like if you had to get all the diagonal entries of a 2D array.
   - Rounding errors - This is especially relevant when dealing with the base cases of methods. Often, you might be asked to find the midpoint of an array, which involves doing some division. Since you are working with `int`s, Java will round up/down, potentially in a way that causes this exception.

3.2 Python Errors and Exception

In this lab, you will be debugging programs written in Java, but you'll also be writing Python programs in this class! The errors thrown are similar to the ones in Java, just with slightly different names. Here are some common errors you might encounter (but feel free to look them up as you encounter more):

1. **SyntaxError** - Similar to Java syntax errors, this occurs when the code doesn't obey Python syntax. For example, the following will throw a syntax error in `Python` because you can't assign a string to an integer:

   ```
   >>> 'foo' = 1
   SyntaxError:  invalid syntax
   ```

   Note: In Python, whitespace (spaces, tabs, indents) is used to denote scope (whereas in Java, this is done with brackets). A common syntax error is misplacing or incorrectly using whitespace! This means that you should be extra careful about indents and spaces when you're coding in Python.

2. **NameError** - This happens when a variable is not found in the local or global scope (make sure all variables you use have been defined!). This is equivalent to a `cannot find symbol` error in Java.

   ```
   >>> x
   NameError:  name 'x' is not defined
   ```

3. **AttributeError** - This is thrown when referencing an invalid field or method (e.g. if the method for the object doesn't exists, or when the object itself is null).

   ```
   >>> team = None
   >>> team.getAthlete()
   AttributeError:  'NoneType' object has no attribute 'getAthlete'
   ```

   Look back at the **NullPointerException** section above, does it look familiar?

4. **IndexError** - Similar to an **ArrayIndexOutOfBoundsException** in Java, this happens when you index into an invalid index of a list.

   ```
   >>> list = [1,2,3]
   >>> list[3]
   IndexError:  list index out of range
   ```

5. **TypeError** - Unlike Java programs, where the compiler will tell you when you have mismatched types, Python programs will only tell you at run-time if you have a type error. This usually happens when a function is applied to an object of an incorrect

type. For example, here's what happens when you try to add a string to an int:

```
>> "two" + 2
TypeError:  must be str, not int
```

3.3 General Bugs

1. **Infinite loops** - You compile the program with no errors, and it seems to run... but then it never terminates and now everything is stuck!! This could be a nasty case of an infinite loop.

   Infinite loops happen when the condition of a loop is never false, and so the code continues to execute forever. The most common way for this to happen is with a while loop:

   ```
   int count = 0;
   while(count < 5) {
    System.out.println(count);
   }
   ```

   Since the condition of the while loop is always `true`, the while loop never breaks, and number 0 is printed forever.

   Fix: Make sure the while loop's condition is met at some point! One possible way to fix this here is by adding `count++;` at the end of the while loop so it will eventually reach 5!

   Note: Infinite loops can also happen with `for` loops! We'll leave it as an exercise for you to figure out why this code causes an infinite loop:

   ```
   for(int i = 0; i < 5; i--) {
    System.out.println(i);
   }
   ```

2. **Logical bugs** - Your code compiles and runs without errors, but it doesn't seem to be doing what you intended it to do. Oftentimes, these bugs might be more difficult to solve because there is no immediate error message indicating what might've gone wrong. While there is no one size fits all solution to every problem, here are some strategies to debug them:

   - **Print lines** - Good 'ol printlines is often your first line of defense. When a method isn't behaving the way you want it to, try adding printlines and check if something is not logically adding up.

Note: Sometimes, a mere `System.out.println("hello")` might not always be the most helpful when debugging. When you are printing the value of a variable, it is generally a good idea to include a descriptive text with it so that when you look at your console, the printlines have context. For example:

Don't print: `System.out.println(name);`

Do print: `System.out.println("Team name:  " + name);`

Now let's look at how you can use printlines to solve some common errors.

(a) **NullPointerException** - For example, let's say you run into a null pointer and are not sure where in your program that is happening. You could insert:

```
System.out.println(variable == null);
```

throughout your code to find the point of error.

(b) **Errors with Arrays** - Say that you have an array that keeps going out of bounds. You could use printlines to track the variables within each loop to find where the error is happening. If you wanted to check the contents of an array, simply using

```
System.out.println(_array);
```

may end up printing something like this: `[I@6d06d69c`.

Instead, you might need to loop over the array and print out the contents at each specific index.

- **Commenting out Code** - This can be helpful to narrow down which line of code is causing the bug. Try commenting out various parts of your code to see if the behavior of your program changes.
- **Use the Debugger** - Eclipse also has it's own built-in debugger! The Eclipse Debugger is designed to give you tools that will make debugging your programs much easier. Rather than writing 20 print lines and running the entire program, the Eclipse debugger allows you to accurately inspect different portions of your code while it's running at your discretion. Here's a guide on how to use it!

4 Using IntelliJ

In order to set up IntelliJ to complete the lab and all the following projects, do the following after getting the stencil files:

- Open IntelliJ and select `File->New->Project`
  - If it isn't already selected, select Java 8 as the Project SDK. Then click "Next".
  - Uncheck "Create project from template" if it is checked. Then click "Next".
  - Enter "debugIntro" for the project name.
  - Do not use the default location. Instead, hit the "Browse" button, navigate to your debugIntro folder, and click OK.
  - Click "Finish."
  - If it isn't already made for you, create a source folder: right-click the project name and use `New->Directory` to create a new folder in your new project named "src". Then right-click "src" and use `Mark Directory As->Sources Root` to make it a source folder.
  - Now we'll create two new packages. Use `File->New->Package` to create a new package in your new source folder named "ranking" and move the `Ranking.java` and `Vaulter.java` files into this package.
  - Repeat the last step. Use `File->New->Package` to create a new package in your new source folder named "olympics" and move the `Athlete.java` and `Competition.java` files into this package. Ignore any errors.
  - You should now have two packages under your "src" folder: `ranking` and `olympics`.
- To run the programs: Right-click on `Ranking.java` under the `ranking` folder and select Run. Now you can run your program by pressing the green "play" button at the top of your screen and selecting "Java application" if prompted. Press the red "stop" button to terminate the program.

5 Guided Debugging

The TAs have written these two Java programs, but they seem to be a little bug-ridden. Let's try debugging them! After setting up IntelliJ, navigate into the `ranking` package.

We recommend making two new folders in your `debugIntro` directory and moving the stencil files as follows: `Ranking.java` and `Vaulter.java` into `ranking` and `Athlete.java` and `Competition.java` into `olympics`. In order to run this on the terminal, `cd` into your new `ranking` directory and type:

- `javac *.java` to compile
- `cd ..` to go up a directory
- `java ranking.Ranking` to run

5.1 Ranking

The HTAs are competing in the pole-vaulting event at the Olympics but the judges lost the final ranking and only have the raw results. Jumping to the rescue, the CS16 TAs write a program to help.

Open up `ranking` as a project in your favorite text editor (we recommend IntelliJ). This folder contains a buggy version of a program modeling a pole-vaulting competition between Vaulters. Take a few minutes to familiarize yourself with the two files, `Vaulter.java` and `Ranking.java`. The purpose of `Ranking.java` is to rank the competitors by their highest height jumped and add a new Vaulter to the existing ranking in order. Unfortunately, it doesn't seem to be working.

1. Run `Ranking.java` as a Java Application. You should see that the following lines are printed:
   ```
   Running competition...
   Adding vaulter to rankings...
   ```

   The code is running, but it doesn't terminate and no error messages are printed - if the program is short then this often indicates an **infinite loop**. Look in the above section to see how we can deal with infinite loops!

   Strategy - Print Statements: Although this is not always the case, infinite loops can often happen when the code never breaks out of a `while` loop. Add the following line inside of the `while` loop: `System.out.println(curVaulterIndex)`. Now, run the program. What is being printed? Is `curVaulterIndex` changing the way you expected it to for each iteration of the loop? If not, why might this lead to an infinite loop?

2. Now, you should run into the following exception:
   `java.lang.ArrayIndexOutOfBoundsException: 4`
   This means that an error occurs when trying to index into an array; the code is attempting to access an index which does not exist within the length of the array. See the section above for some possible causes of this type of error.
   Although your line number may be different (if you have added or deleted lines of code), note that the line `Ranking.java:70` indicates that the exception occurred at line `70` of the file `Ranking.java`.

   Therefore, look at line `70` of your file (or whichever line is specified by your stack trace) and see whether you can determine *why* your code is accessing an index not within the array.

Note: You will find that for some problems, it's necessary to try multiple different debugging strategies.

Strategy 1 - Print Statements: Try adding the following line inside of the for loop that adds the remaining players into the ranking: `System.out.println("newRankings at i: " + newRankings[i])`. Now when you run the program, notice that the same `ArrayIndexOutOfBoundsException` error is thrown, this time after several print statements have been outputted.

Note: Sometimes a well-meaning print statement is not as helpful as we think it is going to be. In cases like this, what else can we do?

Strategy 2 - Assess Logic: When you go to the line producing the error, you may first want to see where each of the arrays into which you are indexing is initialized. Your error means that indexing into an array is being done incorrectly, so it's valuable to think about how the arrays in question are constructed and manipulated up to the point producing the error. How is `newRankings` initialized? What is its length? Is this length dependent on the length of `oldRanking` in any way?

3. Immediately, you should notice that all of the information printed is `null`. First, determine where the null value is being printed and then attempt to determine where the null value is originating from. **Hint:** You may need to look at the file `Vaulter.java`.

Strategy - Use Program Output: Since the Vaulter Rankings are being printed in `main` in the `Ranking.java` file, you should already be able to tell that the names generated are *null*. This is particularly convenient for debugging the methods called inside of the print statement because their results are immediately printed out!

4. There were a few ways to solve the previous bugs. If you were able to get the results shown in step 5, skip this step. If not, read on. Now, you should see that `Rank 2: Amy` is written twice. Since we are trying to add a new Vaulter `Doug` and the code was unable to do so, this indicates that either the new Vaulter was never added to the ranking or it was added at some point and overwritten. See if you can determine when this may have occurred.

Strategy - Print Statements You'll want to find the state of the rankings within multiple points of the `addVaulterToRankings` method. That is, you'll want to print out the rankings at multiple spots as the rankings are calculated to try figure out where things are going wrong. Copy the for loop and print statement in main (after the multi-line comment starting with ("Print Results. Expect:") and try pasting it in another spot inside of the `addVaulterToRankings` method.

Stuck? Take a look at the for loop that adds the remaining players into the ranking. Pay close attention to the indices used in the loop!

5. At this stage, the list you print out should look like this:
```
Rank 1:  Amy
Rank 2:  Doug
Rank 3:  Lisa
Rank 4:  Lucy
Rank 5:  Lionel
```

6. Lastly, test out your code with the other vaulter `rando` as well as with `Doug`.

5.2 Olympics

The previous sections of this lab introduced you to some common bugs and strategies to address them. Now that you have these tools, it's time to put them to the test!

This next program models a competition between a group of Athletes and prints out the names of all Athletes who have qualified (i.e. their race time is under the qualifying time). To try this out: go into the `olympics` package and run `Competition.java` as a Java Application. This program again features some common Java errors, but now it's your turn to fix them.

If you are running this on your terminal, `cd` into your new `olympics` directory and type:

- `javac *.java` to compile
- `cd ..` to go up a directory
- `java olympics.Competition` to run

Once you have addressed the errors and the program is bug-free, you should see the following output:

```
Amy
Lionel
```

Now both the `ranking` and `olympics` packages should be error-free. Refer to Section 2 for instructions on how to hand in when you're done!