CS 16 Java coding conventions

Coding conventions make your code easier to read and debug. Thus, for both our and your benefit, we would like you to use the conventions outlined in this document for your Java code[1].

Startup

1. Header comment at the top of each class

```
/**
* This class is my example class. It mimics a real class, and explains
* how a well-coded and styled Java class should look. It would have a
* brief description of what the class does, and any relevant changes
* from the typical structure.
*
*/
```

Spacing

1. Code should be indented in a logical manner. Everything within a class should be indented at least once, and everything within a method should be indented again. Beyond that, sections within for/while loops should be indented once further, as well as sections in large if/else statements. Java code should be indented as if it were Python and won't work without proper indenting. End brackets should line up with the corresponding start bracket.[2].

```
public class MyExampleClass extends Example {
        private int[] _myInts;

        public MyExampleClass(){
                _myInts = new int[5];
                for (i = 0; i < 5; i++) {
                        _myInts[i] = i*i + i;
                }
        }

        public boolean isOneInList(){
                for (i = 0; i < 5; i++){
```

[1]Actually, we insist on it. If you don't follow the conventions, you'll lose credit on the program.

[2]Note: This example does not include comments (and would thus have points deducted). See below section on Comments for more details.

```
                        if (_myInts[i] == 1){
                                return true;
                        }
                }
        }

}
```

Naming

1. Class names start with a capital letter: `Stack`, `Queue`, `Hashtable` or `HashTable` (the "CapWords" convention)

2. Functions/methods are written in "camelCase" (first word all lowercase, every other first letters capitalized): `findExampleFunction`

3. Instance variables (that can be accessed throughout the class) should begin with an underscore; local variables should not. `_myList` (instance) vs. `myList` (local)

4. Exceptions are classes, so also use CapWords; they end with the word Error ("`BadDataError`")

5. Constants: all caps with underscores: `MAX_OVERFLOW`

Comments!

1. Write a header comment for every class, function, and method. Use block comments.

2. The header comment should be written with a description of the class/function/method. For functions and methods, it should also explain what the inputs and outputs are, an example (if relevant), and explanations for any possible exceptions that can be thrown.

```
/**
* isOneInList(int[] list): list -> Boolean
*
* Purpose: This method checks whether the inputed array contains the number one.
*
* Example:  [0,0,0,4,5] -> False
*           [0,1,0,4,5] -> True
*
* Throws: InvalidInputException: If the list passed in is null
*/
public boolean isOneInList(int[] list){
        \\ code elided
}
```

3. Long chunks of code should be commented throughout with line comments to explain functionality and design choices.

```
...
...
\\ This for() loop checks through the whole list and adds 1 to each item
for (i = 0; i < myList.size(); i++){
        myList[i] += 1;
}
...
...
```

Runtime Expectations

If we specify a runtime, you must meet that runtime. If we do NOT specify a runtime, you should figure out the most efficient way possible to do the problem. In this case, "most efficient" means big-O. However, you should also consider the constant factor–within reason. This means that you don't need to stress about tweaking your code for the tiniest changes to make it a bit more efficient, especially if it will make your code less clear or concise, but you also shouldn't do obviously inefficient things when there is a much more efficient way that is just as easy to implement as is the less efficient way.

For example, if we ask you to write a function that takes as input an integer i and outputs one plus that integer, your function should return "i + 1" as opposed to something unnecessarily more complex (and with a bigger constant factor), such as `i + sqrt(0 + 7 - 7 + 1^2)`. Both of those would return the correct answer, and both would be O(1), but clearly the second has an unnecessarily large constant factor. Note that we will not necessarily tell you what the best big-O runtime is. Sometimes, it's part of the problem to figure that out yourself.