Exercise 1: What’s in a name? (Java Identifiers)

Being able to give things specific names makes it far easier to communicate. As Java programmers, we need to decide what names to use for different kinds of things in our Java programs, including the following: classes, methods, constants and variables.

**Part (a):** Some names (like *Picasso*) are allowed in Java, and some (like *fra-tz!!*) are not. Furthermore, there are conventions about how upper and lower-case letters should be used in Java names. Can you say what the rules are for valid Java names, and what the upper/lower-case conventions are for each kind of Java name (classes, methods, constants and variables)?

**Part (b):** In the Java code below, the programmer has used animals for most of the names (a very bad idea!), and has also failed to follow the Java upper/lower-case conventions. For each animal name, decide whether it is the name of a **class**, a **method**, a **constant** or a **variable**. Then say whether the name follows the upper/lower-case naming conventions, and if not, how it should be changed to follow those conventions. Finally, replace the animal name with a name that follows Java’s upper-lower-case conventions and makes it easier to understand what the program does.

```java
import java.util.Scanner;

class elephant {
    public static void main(String[] args) {
        double octopus = 3.14159;
        Scanner Whoopingcrane = new Scanner(System.in);
        int ORANGUTAN;
        System.out.print("Please enter the radius of a circle: ");
        ORANGUTAN = Whoopingcrane.nextInt();
        bluewhale(2*octopus*ORANGUTAN);
    }

    private void bluewhale(double Meercat) {
        System.out.print("The circumference of the circle is ");
        System.out.println(Meercat);
    }
}
```
Part (c): Now let’s get more practice with Java names by playing a game. Divide into
groups of two or three. Each group will get cards that say valid, valid but poor, or
invalid. Your Team Leader will write a name and your group should decide which cate-
gory it belongs in and why. Put the card that corresponds to your answer face down on
the table in front of you. When all groups have placed their cards, reveal your answers
and determine the correct answer (keep score if you wish).

Exercise 2: Twisted Words (Algorithms)

Remember that computers are fast but dumb. They will do exactly what they’re told,
even if it’s the wrong thing. They can only follow simple, step-by-step instructions. A
list of such instructions is called an algorithm. For this exercise, we’ll combine the idea
of algorithms and the idea of writing simple Java code, like in the picasso exercise from
last week.

This time, instead of an Artist called picasso, we’ll assume that we have something
called word, which represents an English word. The word has operations that rearrange
the letters in the word, an operation that tells you how many letters are in the word, and
an operation that prints the word. Most of the operations have one or two integer ar-
guments that specify positions in the word. Because Java (and many other modern pro-
gramming languages) starts counting from zero, we’ll do that for the positions, too. For
example, if word represents CANDY, we’ll say that the letter C is in position zero, the let-
ter A is in position one, and so on.

Below are descriptions of the operations.

moveToFront( int pos )
Move the letter in position pos to the beginning of the word (i.e., to position zero).

moveToEnd( int pos )
Move the letter in position pos to the end of the word.

swap( int pos1, int pos2 )
Swap the letter in position pos1 with the letter in position pos2.

length()
Tell how many letters are in the word.

print( )
Print the word.
**Part (a):** Below are some examples. In each case, the letter or letters that will be moved are shown in bold in the first column. Work with a partner to write the results in the third column, then compare your answers with another pair.

<table>
<thead>
<tr>
<th>Original word</th>
<th>Java code</th>
<th>word after the code executes</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>word.moveToFront(2)</td>
<td>S T</td>
</tr>
<tr>
<td>STOP</td>
<td>word.moveToEnd(0)</td>
<td>T O P</td>
</tr>
<tr>
<td>TOPS</td>
<td>word.swap(0,2)</td>
<td>P O S</td>
</tr>
</tbody>
</table>

**Part (b):** Remember that an algorithm is a clear, step-by-step description of how to perform a task. Below is an algorithm that does something to a word.

Step 1: Let $j$ be 0 and let $k$ be one less than the number of letters in the word.

Step 2: while ($j < k$) repeat the following two sub-steps.

Sub-step (i): word.swap($j$, $k$)

Sub-step (ii): Change $j$ to $j + 1$ and change $k$ to $k - 1$.

Step 3: word.print()

- What is the result of executing this algorithm if word is initially PUPILS?
- What if word were initially HELLO?
- Tracing an algorithm on a specific example can help us figure out what the algorithm does in general. What does the algorithm above do in general?

**Part (c):** Now assume we only have the moveToFront method. Write an algorithm that has the same effect on the word as the algorithm given above.

If you finish early, try it again assuming that you only have the moveToEnd method.

**Part (d):** Now turn your algorithm into actual Java code, and add it to the TestWord class on one of the laptops. Use the code that is already there to help you. Swap with someone else, and test each other’s code.
Exercise 3: Petals Around the Rose (Logical Thinking)

To work on your logical thinking skills, we’ll play a game called Petals Around the Rose (and that name is significant). Some say that the smarter you are, the harder it will be to figure out how the game works because you will look for solutions that are way too complicated.

To play, your Team Leader will roll 5 dice and will tell you the answer for that roll. You can keep whatever notes you wish. After 5 to 10 rolls of the dice, stop and think about what you’ve seen so far. Make a list of questions that you’d like to ask. For example, you might want to know whether the order of the dice is important. You won’t actually be allowed to ask your questions, but you will be allowed to place the dice on the table and ask what the answer is for that roll. So for each of your questions, also write down one roll of the dice that will help you to answer them.

Once everyone has their questions, go around the table with each person asking one question until all questions have been used up. Keep track of the answers, then think about what you’ve learned. Do you have the solution? If not, come up with new questions.

If you think you have the solution, ask your Team Leader (or another student who knows the solution) to roll the dice. You give the answers and the other person will tell you whether you’re right or wrong. If you get the right answer 6 times in a row, we’ll assume that you’ve got the solution!

Continue like this until you’ve all found the solution or you run out of time.
**Identifier Exercise**

1. Provide teams the answer cards and decide if they will keep score (see scoring below).

2. Have each team submit one identifier that they think is tricky.

3. Write the following identifiers, one at a time, on the board. Mix in the ones submitted by the teams.

   - `xyz` (poor)
   - `This Is Fine` (invalid)
   - `WillThisWork?` (invalid)
   - `public` (invalid)
   - `three#s` (invalid)
   - `Public` (poor)
   - `big_bear` (probably poor)
   - `fire-fly` (invalid)
   - `MAX_VAL` (valid − constant)
   - `PB&J` (invalid)
   - `abiggreenpickle` (poor)
   - `my$money` (poor -- no $)
   - `letterCount` (valid − method/var)
   - `3people` (invalid)
   - `mendota22` (poor)
   - `everything’s$_1` (invalid)

**Suggested scoring:**

1 point to each team for each correct answer (excluding their own submission)

1 point bonus for each valid but poor identifier, to the first team that gives a correct reason why it is poor

1 point bonus for each invalid identifier, to the first team that gives a correct reason why it is invalid

-4 points if a team gets its own submission wrong

2 points to the team that submits an identifier, for every other team that gets it wrong (i.e., 2 points for every team that gets fooled)