Exercise 1: Java variables and types

Remember that Java has the following rules:

• Rule 1: Variables must be declared and initialized before they are used.

• Rule 2: Operators (like +,-, and so on) must be applied only to variables and constants of the correct type.

• Rule 3: In general, the types of the left and right-hand sides of an assignment must match (an exception is that an int value can be assigned to a double variable, but not vice-versa).

• Rule 4: A method call must have the correct number of arguments, and each argument must have the correct type (again, an int can be used where a double is expected, but not vice-versa).

• Rule 5: The condition of an if statement or a while loop must be a boolean expression (must evaluate to true or false).

Last week we had a Recipe class that provided the following methods:

```java
int numEggs( int N )
int numCupsFlour( int N )
```

Look at the code below (with line numbers included for reference). Assume that variables fudge and lasagna are Recipe objects that have been correctly declared and initialized.

```java
1. String numEggsForFudge;
2. int numEggsForLasagna;
3. int num1 = 25.0;
4. int num2;
5. 
6. numEggsForFudge = fudge.numEggs("one");
7. numEggsForLasagna = lasagna.numEggs();
8. totalNumEggs = numEggsForFudge + numEggsForLasagna;
9. 
10. if (fudge.numEggs(2)) {
11.     while (num2 > fudge.numEggs(1)) {
12.         num2 = num - lasagna.numEggs(2);
13.         num = true + lasagna.numEggs(1);
14.     }
15. }
```

Find all of the violations of Rules 1–5 that the Java compiler would report for this code. Then compare your answers with someone else. If you finish early, fix the code (there may be more than one way to do this).
Exercise 2: If statements

Take the following quiz from the person next to you:

What animal are you?

DO YOU LIKE TO RUN?

You are a horse!

DO YOU LIKE CHEESE?

You are a fish!

You are a cat!

You are a mouse!

DO YOU LIKE THE OUTDOORS?

Part (a): Working in a group, draw a similar flowchart for a “What animal (or food, or plant, or whatever) are you?” quiz. Use exactly four diamonds. Then write a Java program for your flowchart on a laptop and set it up so that another group can run your program without seeing your code. Assume that the user will enter 1 for yes, and 0 for no. To read input, use the Java Scanner class. Here are two useful methods of the Scanner class:

int nextInt() // Finds and returns the next token of the input as an int
String next() // Finds and returns the next complete token from the input.

If you finish early, think about what you would have to change in the code if the user types in Yes or No instead of 1 or 0.

Part (b): Now everyone pass your laptop to the group on your left. Run the program you got over and over, drawing the corresponding flowchart. Compare your flowchart with the one the designing group wrote. Are they the same?
Exercise 3: The String class

Java’s String class has many useful methods including the following:

int length()
    Returns the number of characters in this string.

String substring(int beginIndex, int pastEnd)
    Returns a new string that is the substring of this string that starts at position beginIndex and ends at position pastEnd - 1 (the position of the first character is 0). Error if beginIndex is negative, or pastEnd is greater than the length of this string.

int indexOf(int ch)
    Returns the position within this string of the first occurrence of ch. If no such character occurs in this string, then -1 is returned.

int compareTo(String anotherString)
    Returns 0 if this string is the same as anotherString; returns a negative integer if this string comes before anotherString in lexicographic order (dictionary order); returns a positive integer if this string comes after anotherString in lexicographic order.

For example, if String s represents “abc”, then the call

    s.compareTo( "ax" )

would return a negative integer (because “abc” comes before “ax” in lexicographic order).

boolean equals(Object anObject)
    Returns true if anObject is a String that represents the same sequence of characters as this string; otherwise returns false.
Part (a): To get some practice using the String methods, play the following game (in pairs).

- Each person thinks of a short word, 3 to 5 letters long. Whoever guesses their partner’s word first wins!
- Each person takes two cards from a pile shared by all groups. If you ever have more than one of the same card, you may trade one of them in if you wish.
- Alternate turns. When it’s your turn, pick one card, then play one of your three cards. If it has a blank (e.g., `s.indexOf(__)`), you get to fill in the blank with whatever you want. Your opponent tells you what value their word, `s`, would return if the method call on your card were made. For example, if your opponent is thinking of the word “hat” and you play `s.substring(1, 2)`, your opponent must say “a”; if you play `s.substring(2, 4)`, your opponent must say “error” (because “hat” has only 3 letters).
- You win if you play an `s.equals` card and your opponent says “true”, or if you play an `s.compareTo` card and your opponent says 0 (i.e., you guessed their word).

To avoid mistakes, write down your word, and number the letters starting from zero.

Part (b): A palindrome is a string that is the same forwards and backwards. To get ready to work on palindromes, watch a palindrome video, and read a palindrome poem!

Assume that you have a String variable called `word`. Write an algorithm that uses the String methods on the previous page plus the `charAt` method (all listed below) to determine whether the sequence of characters in `word` is a palindrome. Assume that all punctuation has been removed and all letters are lower case.

```java
char charAt(int index)
    // Returns the character at the specified index. (Remember that the index of the first character is 0.)

int length()

String substring(int beginIndex, int pastEnd)

int indexOf(int ch)

int compareTo(String anotherString)
```

-4-
Exercise 4: Logical Thinking

Assume that you have 8 coins, and you know that 7 are OK but one is bad. You know that the bad coin has a different weight than the good coins, but you don’t know whether it’s heavier or lighter.

Figure out how, using only a balance scale, you can find out which is the bad coin using just 3 weighings. Hint: Find a way to determine that half of the coins are OK with just 1 weighing.

Now do the same thing assuming that you have 9 coins, one of which is bad. (Still use just 3 weighings to find the bad coin.)

And now for a real challenge, do the same thing assuming that you have 13 coins.
Cards for the String class exercise

<table>
<thead>
<tr>
<th>s.length()</th>
<th>s.length()</th>
<th>s.length()</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.length()</td>
<td>s.length()</td>
<td>s.substring(0, 1)</td>
</tr>
<tr>
<td>s.substring(0, 1)</td>
<td>s.substring(0, 1)</td>
<td>s.substring(0, 1)</td>
</tr>
<tr>
<td>s.substring(0, 1)</td>
<td>s.substring(0, 2)</td>
<td>s.substring(0, 2)</td>
</tr>
<tr>
<td>s.substring(0, 2)</td>
<td>s.substring(0, 2)</td>
<td>s.substring(1, 2)</td>
</tr>
<tr>
<td>s.substring(1, 2)</td>
<td>s.substring(1, 2)</td>
<td>s.substring(1, 2)</td>
</tr>
<tr>
<td>s.substring(2, 3)</td>
<td>s.substring(2, 3)</td>
<td>s.substring(2, 3)</td>
</tr>
<tr>
<td>s.substring(2, 3)</td>
<td>s.substring(2, 4)</td>
<td>s.substring(2, 4)</td>
</tr>
<tr>
<td>s.substring(2, 4)</td>
<td>s.substring(2, 4)</td>
<td>s.substring(3, 4)</td>
</tr>
</tbody>
</table>
s.substring(3, 4)  s.substring(3, 4)  s.substring(3, 4)

s.substring(3, 5)  s.substring(3, 5)  s.substring(3, 5)

s.substring(3, 5)  s.indexOf(‘a’)  s.indexOf(‘a’)

s.indexOf(‘a’)  s.indexOf(‘a’)  s.indexOf(‘a’)

s.indexOf(‘e’)  s.indexOf(‘e’)  s.indexOf(‘e’)

s.indexOf(‘e’)  s.indexOf(‘i’)  s.indexOf(‘i’)

s.indexOf(‘i’)  s.indexOf(‘i’)  s.indexOf(‘o’)

s.indexOf(‘o’)  s.indexOf(‘o’)  s.indexOf(‘o’)

- 7 -
<table>
<thead>
<tr>
<th>Cards for the String class exercise</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.indexOf(‘u’)</td>
</tr>
<tr>
<td>s.indexOf(‘u’)</td>
</tr>
<tr>
<td>s.indexOf(‘u’)</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.indexOf(___ )</td>
</tr>
<tr>
<td>s.compareTo( ___ )</td>
</tr>
<tr>
<td>s.compareTo( ___ )</td>
</tr>
<tr>
<td>s.compareTo( ___ )</td>
</tr>
</tbody>
</table>
Cards for the String class exercise

s.compareTo( ___ )   s.compareTo( ___ )   s.compareTo( ___ )

s.compareTo( ___ )   s.compareTo( ___ )   s.compareTo( ___ )

s.compareTo( ___ )   s.compareTo( ___ )   s.compareTo( ___ )

s.compareTo( ___ )   s.compareTo( ___ )   s.compareTo( ___ )

s.compareTo( ___ )   s.compareTo( ___ )   s.compareTo( ___ )

s.compareTo( ___ )   s.compareTo( ___ )   s.compareTo( ___ )

s.equals( ___ )     s.equals( ___ )     s.equals( ___ )

s.equals( ___ )     s.equals( ___ )     s.equals( ___ )

s.equals( ___ )     s.equals( ___ )     s.equals( ___ )

s.equals( ___ )     s.equals( ___ )     s.equals( ___ )
Cards for the String class exercise

s.length( ) s.length( ) s.length( )

s.length( ) s.length( ) s.length( )

s.length( ) s.length( ) s.length( )

s.length( ) s.length( ) s.length( )