

# WES-CS GROUP MEETING #4

## 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 literals 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 `long` or `double` variable, but not vice-versa).
- Rule 4: A method call must have the correct number of arguments, the types of the arguments must match the types of the corresponding parameters, and the value returned must match the method's return type (again, an `int` can be used where a `double` or `long` is expected, but not vice-versa).

Work with a partner to find all of the violations of these rules that the Java compiler would report in the following code. Then compare your answers with the other pairs.

```
public double doDivision( double denom ) {
    String returnVal;
    double num1 = 4.4;
    num2 = 5.5;
    returnVal = (num1 + 2)/denom;
    return returnVal;
}
```

```
public void printVal( int oneVal ) {
    double returnVal, dblVal;
    int int1, int2;
    int2 = doDivision(2, 3) + int1;
    dblVal = doDivision( "hello" ) - true;
    System.out.println(returnVal);
    return returnVal;
}
```

## Exercise 2: 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 opponent'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).

### Part (b).

A *palindrome* is a string that is the same forwards and backwards. To see some examples, watch the video at

<http://www.youtube.com/watch?v=Nej4xJe4Tdg>

Assume that you have a `String` variable called `word`. Write an algorithm 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.

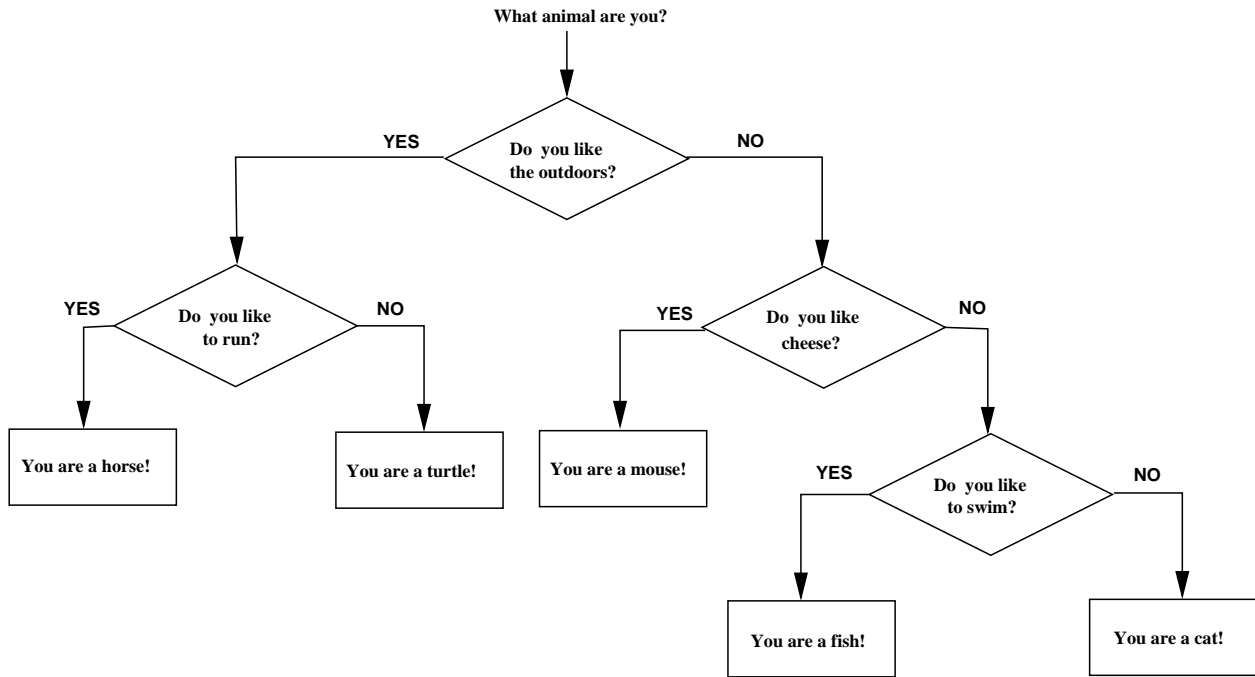
You might find it helpful to use the following `String` method (in addition to the ones on the previous page):

```
char charAt(int index)
```

Returns the character at the specified index. (Remember that the index of the first character is 0.)

### Exercise 3: If statements

Take the following quiz from the person next to you:



**Part (a).**

Working in groups, arrange one set of cards in the right order so you create Java code that implements the flowchart on the previous page. 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:

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

Hint: Start by finding the code just for the lower-right part of the flowchart (the part that asks the question “*Do you like to swim?*”).

Once you have that, add the code for “*Do you like cheese?*”

Then finish the code.

**Part (b).**

What would you have to change in the code if the user types in `Yes` or `No` instead of 1 or 0?

**Part (c).**

Now divide into three groups. Each group draws a similar flowchart for a “What animal (or food, or plant, or whatever) are you?” quiz. Don’t use more than 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. 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 4: Logical Thinking

A person visiting an island with a nasty dictator was thrown in jail for no reason, and told that he had been sentenced to death. However, he would be able to choose the method of execution in the following unusual way: He was to make one statement of fact (e.g. "I am a bird" or "I am wearing a brown, polyester shirt"). If the statement is true, he will be beheaded. If it is false, he will be hanged. At this point, it is clear that the dictator is not a logician, because he has given the prisoner a way out!

**Problem:** What could the prisoner say in order to avoid execution?

The prisoner manages to solve the question above (did you?), and he languishes in jail for a while. Then one day he manages to escape, and he flees down the road, trying to reach the sea. He gets to a fork in the road, and there is a native of the island standing there. Now there is something strange about this island's natives: each person either always tells the truth, or always lies. The prisoner doesn't know whether the native is a liar or a truth-teller.

**Problem:** What single yes-no question could he ask so that he would know which fork in the road leads to the sea, no matter whether the native is a liar or a truth-teller?

### Blue or Pink Cards

```
Scanner in = new Scanner(System.in);

int input;

System.out.println("What animal are you?");

System.out.println("Do you like the outdoors?");

input = in.nextInt();

if ( input == 1) {

System.out.println("Do you like to run?");

input = in.nextInt();

if (input == 1) System.out.println("You are a horse!");

else System.out.println("You are a turtle!");

} else {
```

### Blue or Pink Cards

```
System.out.println("Do you like cheese?");

input = in.nextInt();

if (input == 1) System.out.println("You are a mouse!");

else {

System.out.println("Do you like to swim?");

input = in.nextInt();

if (input == 1 ) System.out.println("You are a fish!");

else System.out.println("You are a cat!");

}

}
```

s.length()

s.length()

s.length()

s.length()

s.length()

s.substring(0, 1)

s.substring(0, 1)

s.substring(0, 1)

s.substring(0, 1)

s.substring(0, 1)

s.substring(0, 2)

s.substring(0, 2)

s.substring(0, 2)

s.substring(0, 2)

s.substring(1, 2)

s.substring(1, 2)

s.substring(1, 2)

s.substring(1, 2)

s.substring(2, 3)

s.substring(2, 3)

s.substring(2, 3)

s.substring(2, 3)

s.substring(2, 4)

s.substring(2, 4)

s.substring(2, 4)

s.substring(2, 4)

s.substring(3, 4)

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')

s.indexOf('u')

s.indexOf('u')

s.indexOf('u')

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

s.indexOf( \_\_\_ )

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.length( )

s.length( )

s.length( )

s.length( )

s.length( )

s.length( )

s.length( )

s.length( )

s.length( )

s.length( )

s.length( )

s.length( )