Practice with Java syntax using jshell

This document gives you a set of exercises in writing Java snippets using the tool *jshell*. Jshell is similar to the Python interpreter or the Matlab command window. It's a great way to experiment with basic Java syntax. However, as your projects get bigger, you'll want a more structured and reproducible way of testing them, which is why we use the JUnit framework built into Eclipse for most of this course.

The exercises will help you learn the syntax and semantics of:

method declarations
method calls
return statement
printing
basic operations, including the remainder operation %
if- and if-else statements
conditional expression
for-loop and while-loop.

How to learn about jshell

To find out about the basics of jshell, type jshell into the JavaHyperText (JHT) filter field and read the pdf file about the basics of jshell. You will be typing multi-line methods into jshell. It is often easier to type them in an text editor, make sure they are correct, and then copy-paste them into jshell.

Exercises

Now that you have done that, complete the following tasks by typing code into jshell (you will remember the syntax better if you type, rather than copy-paste). We recommend saving a copy of your working code, along with notes about any mistakes you made along the way, to a document that you can reference later.

1. Type the following function into jshell:

```
/** = the product of x with itself. */
int square(int x) {
   return x*x;
}
```

Try calling your function by typing square(3) at the prompt (which should return 3*3 = 9).

Note: In the method above,
(1) `int` is the return type —the type of the value to be returned.
(2) `square` is the name of the method.
(3) `int x` is a declaration of parameter `x`. If there are more parameters, separate their declarations with a comma.
(4) `{ ... }` is a *block*, called the body of the method. It contains a *return statement*.

2. Read the JHT entry on return .

3. Write a function named "product" that returns the product of two integers. Confirm that it works by typing product(3, 4) at the prompt and checking its answer. Try a few examples of your own.

4. Write a procedure named printRemainder that prints the remainder when its first integer parameter is divided by its second. A procedure uses void instead of a return type. The body can be a single statement System.out.println(...); where you replace the "..." by the expression that computes the remainder using operator %. If you are unsure how to do this, look at JHT entry %

5. We're going to ask you to write six different methods to compute the maximum of two values in order to introduce you to the if-statement, the if-else statement, the conditional expression (ternary operator), the return statement in a procedure, the block, and the local variable. To start, read the pdf file on JHT entry if-statement .

Practice with Java syntax using jshell

5A. Write a function max1 that returns the larger of its two integer parameters. Use an if-else statement to determine whether the first parameter is larger than the second. The then-part and else-part should be return statements.

5B. Write a function max2 that returns the larger of its two integer parameters. The body should consist of (1) an if-statement (not an if-else statement) whose then part is a return statement and (2) a return statement. This manner of not using an else-statement in the case that the then-part returns can often simplify code.

5C. Write a function max3 that returns the larger of its two integer parameters. The body should be a single return statement whose expression is a conditional expression. If need be, look at the JHT entry ? . Yes, just type a query (question mark) into the Filter Field.

5D. Write this procedure max4. , which prints the larger of its two integer parameters:

```
void max4(int x, int y) {
   if (x >= y) {
      System.out.println("max is " + x);
      return;
   }
   System.out.println("max is " + y);
}
```

Put these calls into jshell to see whether the procedure works: max4(5, 6) max4(7, -2).

Here are two important points about this procedure.

(1) The then-part consists of a *block* {...} . A block is needed because the then-part has two statements in it. Whenever the then-part or else-part requires several statements, *aggregate* them using a block.

(2) This example shows how to use a return statement in a procedure. Its execution terminates execution of the method body; nothing more happens after that return statement is executed.

5E. Write this procedure max5 in jshell, which returns the larger of its two integer parameters. Then call it twice with calls like max5(5, 3) and max5(2, 8).

```
int max5(int x, int y) {
   int z= y;
   if (x >= y) z= x;
   return z;
}
```

The first statement of the method declares (and assigns to) a *local variable*, z. At this point, read the first page of the pdf file linked to in the entry local variable of JavaHyperText.

5F. Write procedure max6, below. You will see that it doesn't compile. The issue has to do with the *scope* of a local variable, which you read about in the pdf file mentioned in 5E.

```
int max6(int x, int y) {
   if (x >= y) { int z= x; }
   else {int z= y; }
   return z;
}
```

To fix this, you have to declare z before the if-statement, assigning some value to it. Then, use assignments to z and not declarations of z in the then-part and else-part. Go ahead, rewrite this function into jshell and write some calls on it to make sure it works correctly.

6. Read about while-loops in the pdf file in JHT entry while-loop. . Don't lok at point 4; we'll cover that later in the course. Implement the following function using a while-loop. You have to implement the "TODO" comment. We suggest that you first write all the code somewhere and then copy-paste it into jshell. After that, write several calls on the method to make sure that it is correct.

Here are some possible calls: powerCeiling(4) is 4, powerCeiling(5) is 8, powerCeiling(9) is 16.

```
/** = the first power of 2 that is >= threshold, where threshold is >= 1 */
int powerCeiling(int threshold) {
   int pow= 1;  // = 2^0
   // TODO: Multiply pow by 2 until it satisfies the requirement of the function's return value

   return pow;
}
```

7. Read about the for-loop in JHT entry for-loop . Read only the first 6 points in the pdf file. Then complete the following method, put it into jshell, and test it. Note: If n is bigger than m, the sum is 0. Do not write a special case for the case n > m; your loop should handle it.

```
/** = sum of n..m --of these values: n, n+1, n+2, ..., m */
int sum(int n, int m) {


}
```

8. Write a method sumE(n, m) to compute the sum of the even values in n..m —i.e. the even values in the set n, n+1, n+2, ..., m. Here are points to consider.

(1) The first value to be added to the sum is either n or n+1, depending on whether n is even or not. say, Since, the control variable, say k, should be that first value, its calculation is not just a simple assignment like k= n. Therefore, it may make sense to declare and initialize k before the loop, and the loop <initialization> will be empty.

(2) The loop increment should add 2 to the control variable