

NAME _____

DATE _____

PERIOD _____

UNIT 4 · TOPIC 4.16 · WORKSHEET

Calls All the Way Down

Trace the calls, stack them up, and catch the recursion that never comes back.

About 30 minutes

9 questions

Cumulative through 4.16

TOPIC

4.16

**DIRECTIONS**

Work in pencil, and write exactly what prints. Every method shown is a public static method in the same class as main.

PART A · 3 QUESTIONS

A

Warm up your brain

1

YOU'RE THE JVM

You're the JVM. What does `System.out.println(hop(11));` print, and how many calls to `hop` are made in all?

```
public static int hop(int n) {  
    if (n <= 2) {  
        return n;  
    }  
    return hop(n - 3) + 1;  
}
```

A prints 5, 4 calls**B** prints 2, 4 calls**C** prints 5, 3 calls**D** prints 3, 3 calls

2 PENCIL CHECK

main calls `tally(4096)`. Fill in the call stack: one row per call, the first call at the top. Then fill in what each call returns. (Remember that `int` division drops the remainder.)

```
public static int tally(int n) {
    if (n < 10) {
        return 1;
    }
    return 1 + tally(n / 10);
}
```

Call	Value of n	Base case?	Returns
tally(4096)	4096	no	

3 FACE-OFF

Tonight: a for loop and a recursive method argue about who repeats better. Fill in the blanks.

LOOP When I repeat, I go back to my condition. My loop control variable keeps track of how far I've gotten.

RECURSION I repeat by _____. My _____ keep track of how far I've gotten, and every call gets its own copy.

LOOP I stop when my condition becomes _____. How do *you* stop?

RECURSION I stop when I reach a _____, a path that finishes without calling again.

LOOP If my condition never turns false, I run forever.

RECURSION If I never reach a base case, the waiting calls pile up until the program crashes with a _____.

LOOP Admit it. There are jobs only one of us can do.

RECURSION Nope. Any recursive solution can be rewritten as an _____ one, and any loop can be rewritten with recursion.

PART B · 4 PUZZLES

B Puzzles

4 CODE TILES

Hopper digs down for treasure, then climbs back up. Fill the `else` branch of `dig` with three tiles so that `dig(3)` prints the target output. Two tiles are decoys.

```
public static void dig(int n) {
    if (n == 0) {
        System.out.println("treasure!");
    } else {
        // your tiles go here
    }
}
```

<code>System.out.println("up " + n);</code>	<code>dig(n + 1);</code>
<code>System.out.println("dig " + n);</code>	<code>System.out.println("up " + (n - 1));</code>
<code>dig(n - 1);</code>	

Target output, lines 1 to 4	Lines 5 to 7
<pre>dig 3 dig 2 dig 1 treasure!</pre>	<pre>up 1 up 2 up 3</pre>

THE ELSE BRANCH



FAIR QUESTION

Why do the lines after a recursive call come out backwards?

Because a call that makes a recursive call has to *wait* for it. Picture a stack of plates: each new call goes on top, and only the top one can finish. So the last call to start is the first to finish, and whatever comes after the recursive call runs newest-first.

5 SNIPPET POOL

A lily pond's frog count triples every week. `grow(start, weeks)` should return the count after that many weeks, so `grow(2, 3)` returns 54. Fill each hole from the pool.

```
public static int grow(int start, int weeks) {
    if (weeks == (1)) {
        return start;
    }
    return (2) grow(start, (3));
}
```

THE POOL

Each snippet can be used only once, and not every snippet is needed.

1	3 +	weeks - 1	0
start *	weeks + 1	3 *	

6 ONE-TOKEN FIX

`doubles(n)` should return 2 to the power `n`, so `System.out.println(doubles(5));` should print 32. It prints 0 instead. Change exactly one token to fix it.

```
1 public static int doubles(int n) {
2     if (n == 0) {
3         return 0;
4     }
5     return 2 * doubles(n - 1);
6 }
```

LINE, OLD TOKEN, NEW TOKEN



WATCH OUT

Having a base case isn't enough

The recursion has to *reach* it. Check the argument in each recursive call: it must move toward a base case, and it must not be able to jump over it. `if (n == 10)` with a recursive call on `n + 4` works from 2 (2, 6, 10) but not from 3 (3, 7, 11, ...).

7 YOU'RE THE COMPILER

You are the compiler, and then the JVM. Each method is in its own program, and `main` prints a call to it with an argument of 5, such as `System.out.println(a(5));`. Does it compile? If it does, does it print a value or crash?

Method	Compiles?	Prints, or what goes wrong
1 <pre>public static int a(int n) { return n + a(n - 1); }</pre>		
2 <pre>public static int b(int n) { if (n == 0) { return 0; } b(n - 1); }</pre>		
3 <pre>public static int c(int n) { if (n <= 0) { return 0; } return 2 + c(n - 2); }</pre>		
4 <pre>public static int d(int n) { if (n == 0) { return 0; } return 2 + d(n - 2); }</pre>		

PART C · 2 QUESTIONS

C Twins and a lie

8 MATCH-UP

Every recursive method has a loop twin that does the same job. Each method below is called with 4. Match it to the loop that prints exactly the same thing. One loop has no twin.

Recursive method	Loop twin
1 public static void r1(int n) { if (n > 0) { System.out.print(n + " "); r1(n - 1); } }	
2 public static void r2(int n) { if (n > 0) { r2(n - 1); System.out.print(n + " "); } }	
3 public static void r3(int n) { if (n > 0) { System.out.print(n + " "); r3(n - 2); } }	

Loop	Code	Loop	Code
P	for (int i = 1; i <= n; i++) { System.out.print(i + " "); }	Q	for (int i = n; i > 0; i -= 2) { System.out.print(i + " "); }
R	for (int i = 0; i < n; i++) { System.out.print(i + " "); }	S	for (int i = n; i > 0; i--) { System.out.print(i + " "); }

9 TWO TRUTHS AND A LIE

main runs `System.out.println(pile(4));`. Two of these claims are true and one is a lie. Circle the lie, then rewrite it so it's true.

```
public static int pile(int n) {  
    if (n == 1) {  
        return 2;  
    }  
    int below = pile(n - 1);  
    return below + n;  
}
```

- **1** The program prints 11.
- **2** `pile(1)` is the first call to finish, and `pile(4)` is the last.
- **3** All the calls share one variable `below`, so when `pile(4)` reaches its last line, `below` holds 2, the value the base case returned.

THE LIE, REWRITTEN