

NAME

DATE

PERIOD

UNIT 1 · TOPIC 1.8 · WORKSHEET

Notes for People

Find the code hiding in the comments, read a Javadoc, and catch the precondition that's being misunderstood.

About 30 minutes

9 questions

Cumulative through 1.8

TOPIC

1.8



DIRECTIONS

Work in pencil. For code, write exactly what prints, one line per `println`. Watch exactly where each comment starts and stops. Calls like `"hopper".indexOf("pp")` follow their documentation; full rules for calls come later.



THINK FIRST

A program prints 7. You add three comments to it and change nothing else. Could it print something different now? Guess in the margin; question 1 settles it.

PART A · 4 QUESTIONS

A

Warm up your brain

1 YOU'RE THE JVM

You are the Java Virtual Machine. You run only what the compiler kept. What prints?

```
1 int pts = 6;
2 // pts += 4;
3 pts *= 3; /* pts -= 2;
4 pts += 1; */ pts -= 5;
5 System.out.println(pts);
```

A 25

B 13

C 12

D 18



FAIR QUESTION

Do comments make a program slower or bigger?

No. The compiler throws every comment away before it translates the program, so the translated program the computer runs contains none of them. A program with a hundred comments runs exactly like the same program with none.

2 YOU'RE THE COMPILER

You are the compiler. For each snippet, decide whether it compiles. If it does, write what it prints. If it doesn't, say why in a few words.

Snippet	Compiles?	Output, or why not
1 <code>int a = 5; // a = 10;</code> <code>System.out.println(a);</code>		
2 <code>/* int b = 5; */</code> <code>System.out.println(b);</code>		
3 <code>int c = 5; /* add one</code> <code>c++;</code> <code>System.out.println(c);</code>		
4 <code>/** the count */ int d = 7;</code> <code>System.out.println(d);</code>		
5 <code>int e = 4;</code> <code>e *= 2; // double it */</code> <code>System.out.println(e);</code>		
6 <code>int f = 3; */</code> <code>System.out.println(f);</code>		



WATCH OUT

The compiler can't tell when a comment is wrong. In songs `+= 3; // add the 2` new songs, the note says 2, but the code adds 3, and the code is what runs. When a comment and the code disagree, the code wins, so update the comment every time you change the code.

3 FACE-OFF

Tonight: a Precondition and a Postcondition argue about who matters more. Fill in the blanks.

PRECONDITION	I'm what must be true _____ a method or code segment runs. I'm the fine print in the deal.
POSTCONDITION	I'm what's guaranteed _____ it runs, as long as you kept your side of the deal.
PRECONDITION	Break me and anything can happen: a _____ answer, or a crash.
POSTCONDITION	I describe the outcome: the value that's _____, or the current values of an object's _____.
PRECONDITION	And the compiler never checks either of us, because we're written in _____.
POSTCONDITION	True. We're written for _____, not for the computer.

4 PENCIL CHECK

Here is a simplified Javadoc comment from the `String` class's source code, sitting above the header of its `indexOf` method. Use only the comment to fill in the table. (Read past `public` for now; a later topic explains it.)

```
/**
 * Returns the index where str first appears in this string,
 * counting from 0. Returns -1 if str does not appear.
 *
 * @param str the text to look for
 * @return the index where str first appears, or -1
 */
public int indexOf(String str)
```

Question	Answer
What must you give the method?	
Which tag describes what the method gives back?	
What type of value does it give back?	
<code>"hopper".indexOf("pp")</code>	
<code>"hopper".indexOf("z")</code>	

PART B · 3 PUZZLES

B Puzzles

5 CODE TILES

A stopwatch program turns seconds into whole minutes and leftover seconds. It already imports Scanner, and the top of main says what it needs and what it promises. Someone knocked the rest of main off the fridge. Put the tiles back in order so that, when the user types 200, it prints the output below. One tile is a decoy and isn't used.

```
// Precondition: the user types whole seconds, not negative.
// Postcondition: prints whole minutes, then leftover seconds (0 to 59).
```

<code>seconds %= 60;</code>	<code>Scanner console = new Scanner(System.in);</code>
<code>System.out.println(seconds);</code>	<code>// seconds %= 60;</code>
<code>int seconds = console.nextInt();</code>	<code>System.out.println(seconds / 60);</code>

Target output

3

20

YOUR PROGRAM

6 SNIPPET POOL

Fill each numbered hole with a comment marker from the pool so that the code prints 17. Each snippet is used at most once, and some aren't needed.

```
1 int coins = 20;
2 (1) coins += 10;
3 coins -= 4; (2) coins *= 2; (3)
4 coins += 1;
5 System.out.println(coins);
```

THE POOL

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

/	//	/	/
\\	**		

7 WHO AM I?

Some documentation ideas are playing a party game. Each gives a clue about itself. Name the idea.

Clue	I am...
I start with two slashes, and the end of the line is the end of me.	
I open with one star after the slash, and only my */ can stop me.	
I open with two stars, and a tool turns me into API documentation.	
Inside a Javadoc comment, I describe a value the method needs to be given.	
I must be true just before a method runs, and the method won't check me.	
I'm the tool that comes with Java and builds documentation pages from certain comments.	

PART C · 2 QUESTIONS

C

Catch the mistake

8

TWO TRUTHS AND A LIE

Two claims are true and one is a lie. Circle the lie, then rewrite it so it's true.

```
// Precondition: people is greater than 0.  
// Postcondition: each = slices per person, extra = slices left over.  
int slices = 14;  
int people = 4;  
int each = slices / people;  
int extra = slices % people;
```

- **1** After the code runs, each is 3 and extra is 2.
- **2** The postcondition is a promise about what is true after the code runs.
- **3** If people were 0, the code would not compile, because of the precondition.

THE LIE, REWRITTEN

9

ONE-TOKEN FIX

The ticket booth's code prints 30, but every line of math is supposed to run, so it should print 32. Change exactly one token to fix it.

```
1 int tickets = 50; // tickets printed  
2 tickets -= 8;    /* sold before lunch */  
3 tickets -= 12;   /* sold after lunch */  
4 tickets += 2;    /* two returned */  
5 System.out.println(tickets);
```

LINE, OLD TOKEN, NEW TOKEN