



AP CSA • UNIT 1 • TOPIC 1.8

Notes for People

Documentation with comments



What is this program for?

PAUSE AND TRY

You can say what it prints. Can you say what it is for?

```
int t = 45;  
t *= 3;  
System.out.println(t);
```

OUTPUT

135

What is 45?

Why times 3?

Would you know a month from now?

By the end of class you can...

1 Explain who comments are for and why they never change what a program does.

2 Write and read `//` and `/* */` comments, and predict output when code has them.

3 Explain how Javadoc comments, `/** */`, become API documentation, and read `@param` and `@return`.

4 State a precondition and explain who is responsible for making it true.

5 State a postcondition and tell it apart from a precondition.

A large, white, rounded square containing the number '1' in a bold, black, sans-serif font.

PART 1

Comments are notes for people

The compiler skips every comment



A **comment** is text written for people. The compiler **ignores** it, so it is never checked, translated, or run.

SAME PROGRAM, WITH COMMENTS

```
// Minutes of practice a day, for a 3-day weekend camp.  
int t = 45;    // minutes per day  
t *= 3;       // three days of camp  
System.out.println(t);    // total minutes
```

OUTPUT

135

Same output as before. Only the human reader got something new.

Who reads comments? You, a month from now, and every other programmer who opens the file.

Commenting out a line

```
1  int laps = 5;
2  // System.out.println("Warming up");
3  laps += 2;
4  System.out.println(laps);
```

OUTPUT

7

Line	What happens	laps
1	laps is set to 5	5
2	Comment: skipped. "Warming up" never prints.	5
3	Add 2	7
4	Print laps	7

A large, white, rounded square containing the number '2' in a bold, teal, sans-serif font.

2

PART 2

Single-line and block comments

// runs to the end of the line

END-OF-LINE COMMENTS

```
double price = 2.5;    // dollars per ticket
int count = 4;         // tickets bought
double cost = price * count;    // 4 times 2.5
System.out.println(cost);
```

OUTPUT

10.0

Before //

Ordinary code. It is translated and it runs.

// itself

Marks where the comment starts.

After //

Ignored, all the way to the end of the line.

KEY IDEA

`/* */` runs until `*/`

A BLOCK COMMENT

```
/* Counts the books on two shelves.  
   Shelf two was added in June. */  
int books = 40;  
books += 25;  
System.out.println(books);
```

OUTPUT

65

Kind	Starts with	Ends at	Good for
Single-line	//	The end of that line	A short note, or a note after code
Block	/*	The next */, even mid-line	Notes several lines long

A comment that eats your code

CODE AFTER //

```
// books on the shelf  
int books = 40;  
System.out.println(books);
```

NO CLOSING */

```
/* Prints how many books  
   are on the shelf.  
public static void main(...) {
```

COMPILER SAYS

```
cannot find symbol  
symbol:   variable books
```

COMPILER SAYS

```
error: unclosed comment  
error: reached end of file  
       while parsing
```



A // comment is always the **last** thing on its line. A /* comment keeps going until */, so type the */ the moment you type /*.

Which lines run?

PAUSE AND TRY

Write the exact output. Mark each line: runs, or comment?

```
1  int score = 8;  
2  /* score += 10;  
3  score *= 2; */ score -= 3;  
4  // score = 0;  
5  System.out.println(score);
```

YOUR OUTPUT

A large, white, rounded square containing a bold, teal number '3'.

3

PART 3

Javadoc comments

/ */ becomes the API documentation**

INSIDE STRING (SIMPLIFIED)

```
/**
 * A String holds a sequence of characters.
 */
public class String {
    /**
     * Returns the number of characters.
     * @return the number of characters
     */
    public int length() {
        // ...counts the characters...
    }
}
```

Above the class

Becomes the description at the top of the page.

Above the method

Its first sentence describes the method.

@return tag

Describes the value the method gives back.

// inside

A note for maintainers. Not on the page.

Reading @param and @return

FROM THE MATH CLASS (SIMPLIFIED)

```
/**
 * Returns a raised to the power b.
 *
 * @param a    the base
 * @param b    the exponent
 * @return     a raised to the power b
 */
public static double pow(double a, double b)
```

```
System.out.println(Math.pow(5.0, 2.0));
```

Tag	Tells you
@param a	The first value: the base
@param b	The second value: the exponent
@return	What comes back: a to the power b

OUTPUT
25.0

Which one reaches the documentation?

PAUSE AND TRY

A programmer wants this description on the API page. Which works?

A `// Returns the total price in cents.`

B `/* Returns the total price in cents. */`

C `/** Returns the total price in cents. */`

A large, white, rounded square containing the number '4' in a bold, teal, sans-serif font.

PART 4

Preconditions and postconditions

Your side of the deal, and the code's

Precondition

MUST BE TRUE BEFORE

Making it true is the job of whoever uses the code. The code does not check it.

Example

x is not negative.

Postcondition

GUARANTEED AFTER

Describes the result: the value returned, or the values of an object's attributes.

Example

rounded holds x rounded to the nearest whole number.

When the precondition is broken

PRECONDITION MET

```
// Precondition: x is not negative.  
double x = 6.8;  
int rounded = (int) (x + 0.5);  
System.out.println(rounded);
```

OUTPUT

7

PRECONDITION BROKEN

```
// Precondition: x is not negative.  
double x = -6.8;  
int rounded = (int) (x + 0.5);  
System.out.println(rounded);
```

OUTPUT

-6



The nearest whole number to -6.8 is -7. No error, no warning: just a **wrong answer**.

"If the input were bad, the code would tell me"



There is no expectation that code checks its own precondition. Meeting it is **your** job.

```
// Precondition: count is greater than 0.  
int total = 90;  
int count = 0;  
System.out.println(total / count);
```

WHEN IT RUNS

```
Exception in thread "main"  
java.lang.ArithmeticException:  
/ by zero
```

It compiles: the compiler never reads the precondition. It crashes when it runs.

Check the promise

PAUSE AND TRY

Answer all three with a partner.

```
// Precondition: cents is not negative.  
// Postcondition: dollars holds the whole  
//    dollars, and left holds the cents  
//    left over (0 to 99).  
int cents = 437;  
int dollars = cents / 100;  
int left = cents % 100;
```

1. What are dollars and left? Does the postcondition hold?
2. What must be true before the code runs?
3. If cents were -437, is the code broken?

Words to know

Comment

Text for people. The compiler ignores it.

// and /* */

Single-line comment, to the end of the line. Block comment, to the next */.

Javadoc /** */

Above a class or method; the javadoc tool builds API documentation from it.

@param / @return

Javadoc tags: a value the method needs / the value it gives back.

Precondition

Must be true just before the code runs. The code does not check it.

Postcondition

Always true after the code runs: what is returned, or attribute values.

Before and after



Explain to your partner the difference between a precondition and a postcondition. Who is responsible for each?

SENTENCE STARTER

A precondition must be true _____, and _____ makes it true. A postcondition is guaranteed _____.

Next: the Notes for People puzzle worksheet.