

2.8 for Loops

Self-Study Guide · About 45 minutes · With answer key

By the end you will be able to

- Name the three parts of a for loop header and say what each one does.
- Trace a for loop step by step, showing when the setup, the test, the body, and the update run.
- Change where a for loop starts, where it stops, and how it steps, and count how many times its body runs.
- Rewrite a for loop as an equivalent while loop, and a counting while loop as a for loop.

What to remember from earlier

- A while loop checks its condition before every pass, including the first. ([Topic 2.7: while Loops, 2.7.2 Check, Run, Repeat](#))
- If the condition is false at the start, the loop body never runs. ([Topic 2.7: while Loops, 2.7.3 When the Body Never Runs](#))
- A loop whose condition never becomes false is an infinite loop. ([Topic 2.7: while Loops, 2.7.4 Infinite Loops](#))
- A loop that runs one time too many or too few has an off-by-one error. ([Topic 2.7: while Loops, 2.7.5 Off-by-One Errors](#))
- `count++` adds 1, `count--` subtracts 1, and `count += 2` adds 2. ([Topic 1.6: Compound Assignment Operators](#))

2.8.1 The Three Parts of a for Loop Header

Many loops count. A program prints seat numbers 1 through 30, runs a game for 10 rounds, or counts down from 3 to liftoff. In Topic 2.7 you wrote counting loops with `while`. Here is one that prints 1 through 5:

```
1 int count = 1;
2 while (count <= 5)
3 {
4     System.out.println(count);
5     count++;
6 }
```

```
1
2
3
4
5
```

Look at what it takes to make this loop count. Three separate pieces control it, and they sit in three different places:

- line 1 gives the counter its starting value, before the loop;

- line 2 holds the test that decides whether another pass happens;
- line 5 changes the counter, at the very bottom of the body.

All three must be right, or the loop goes wrong. If line 5 is forgotten, `count` stays 1 forever and the loop never ends, the infinite loop from Topic 2.7. In a longer body, line 5 can be many lines away from line 2, so it is easy to lose track of how the loop moves.

Because counting loops are so common, Java has a loop that gathers all three pieces onto one line. Here is the same loop written that way:

```
1 for (int count = 1; count <= 5; count++)
2 {
3     System.out.println(count);
4 }
```

It prints exactly the same five lines. The setup, the test, and the change now sit together on line 1, so you can read how the loop counts at a glance: start at 1, keep going while `count <= 5`, and add 1 each time.

This is a **for loop**: a loop statement whose header holds the setup, the test, and the update of the loop in one place. Like a while loop, it is a kind of **iterative statement**, a statement that repeats a block of code. The first line, from `for` to the closing parenthesis, is the **loop header**. The statements between the braces are the **loop body**, the code that repeats, just as in a while loop. Inside the parentheses, the header has exactly three parts, separated by two semicolons:

```
1 for (int count = 1; count <= 5; count++)
```

Part	In this header	What it does
Initialization	<code>int count = 1</code>	Creates the counter and gives it a starting value.
Boolean expression	<code>count <= 5</code>	The test. The body runs only while this is <code>true</code> .
Update	<code>count++</code>	Changes the counter after each pass, so the test can become <code>false</code> .

The **initialization** is the first part of the header; it sets up the variable that controls the loop. The **Boolean expression** (often called the loop's condition) is the second part, and it works exactly like the condition of a while loop. The **update** is the third part; it changes the variable, usually by counting up or down.

The variable created in the initialization is the **loop control variable**. Its value controls how many times the loop runs, because the test checks it and the update changes it. In this loop the loop control variable is `count`.

A few details of the syntax matter:

- The two separators are **semicolons**, the same symbol that ends a statement. They are not commas.
- The update has no semicolon after it, even though `count++` needs one as a statement on its own line. The closing parenthesis ends the header, and the body follows directly, with no semicolon in between (2.8.2 shows what goes wrong if you add one).
- The body goes in braces, exactly like a while loop's body.

KEY IDEA

A for loop header has three parts separated by semicolons: the initialization (set up the loop control variable), the Boolean expression (keep going while it is true), and the update (change the variable after each pass).

WORKED EXAMPLE Messages for each lap, then one final message

This loop uses a different variable name and puts text around the counter. Watch which line repeats and which runs only once.

```
1 for (int lap = 1; lap <= 3; lap++)
2 {
3     System.out.println("Lap " + lap + " done");
4 }
5 System.out.println("Race over");
```

The loop control variable is `lap`. It starts at 1, the test is `lap <= 3`, and the update is `lap++`.

Pass	lap	lap <= 3	Printed
1	1	true	Lap 1 done
2	2	true	Lap 2 done
3	3	true	Lap 3 done
end	4	false	

Line 3 is inside the body, so it runs once per pass, and each time `"Lap " + lap + " done"` joins the current value of `lap` into the text. Line 5 is after the closing brace, so it is not part of the loop. It runs once, after the test finally comes out `false`:

```
Lap 1 done
Lap 2 done
Lap 3 done
Race over
```

WATCH OUT

Commas instead of semicolons. The three parts look like a list, so students often separate them with commas:

```
1 for (int count = 1, count <= 5, count++)
2 {
3     System.out.println(count);
4 }
```

This does not compile. Java reports:

```
error: ';' expected
```

Java reads the header from left to right. After `int count = 1` it expects a semicolon to end the initialization; a comma there means something else to Java, so the rest of the header no longer makes sense. (Exact error wording can vary between Java versions, and you may see more than one error for the same line.) The fix is to use semicolons between the parts: `for (int count = 1; count <= 5; count++)`.

PAUSE AND TRY

Question 1 Multiple choice

Look at this loop header:

```
1 for (int floor = 10; floor > 0; floor -= 3)
```

Which part is the update?

- A. `int floor = 10`
- B. `floor > 0`
- C. `floor -= 3`
- D. `floor`

Question 2 Tracing

What does this code print? Make a table with the value of `stars`, the test, and what prints on each pass.

```
1 for (int stars = 1; stars <= 4; stars++)
2 {
3     System.out.println(stars * 10);
4 }
```

2.8.2 The Order a for Loop Runs In

The header lists its three parts left to right, but Java does **not** run them left to right once and then move on. If you misread the order, you will get the wrong output when you trace. So it is worth learning the exact sequence.

Think of a coach running laps with a team. Before practice starts, the coach sets the lap counter to 1, once. Then, before every lap, the coach checks: "Have we done enough?" If not, the team runs a full lap. Only after the lap is finished does the coach add one to the counter. Then the coach checks again. The counter is never changed in the middle of a lap.

A for loop works the same way:

1. **Initialization**, once. Java runs the first part of the header a single time, before anything else. It never runs again during this loop.
2. **Test**. Java evaluates the Boolean expression right away. If it is `false`, the loop is over: Java skips the body and continues with the statement after the loop.
3. **Body**. If the test was `true`, Java runs the entire body, top to bottom.
4. **Update**. Only after the whole body has finished, Java runs the third part of the header.

5. **Back to step 2.** Java tests again with the updated value, and the cycle repeats until the test is `false`.

Written as a sequence for a loop that runs its body twice:

```
initialization → test (true) → body → update
                → test (true) → body → update
                → test (false) → statement after the loop
```

Notice two things. The update happens at the end of each pass, never before the body. And the test always runs one more time than the body: the final test is the one that comes out `false` and stops the loop. Each run of the body is called an **iteration**, so "the loop iterates three times" means the body runs three times.

KEY IDEA

Initialization runs once. Then: test, body, update, test, body, update, ... until the test is false. The update always comes after the whole body and before the next test.

WORKED EXAMPLE Every step of a three-day loop

Trace each step in order. The "Step" column names which part of the loop Java is running.

```
1 for (int day = 1; day <= 3; day++)
2 {
3     System.out.println("Day " + day);
4 }
5 System.out.println("Done");
```

Step	day	What happens	Printed
Initialization	1	day is created and set to 1.	
Test	1	1 <= 3 is true; run the body.	
Body (line 3)	1	Print the current day.	Day 1
Update	2	day++ makes day equal 2.	
Test	2	2 <= 3 is true; run the body.	
Body	2		Day 2
Update	3	day++	
Test	3	3 <= 3 is true; run the body.	
Body	3		Day 3
Update	4	day++	
Test	4	4 <= 3 is false; the loop ends.	
After loop (line 5)		Print Done.	Done

```
Day 1
Day 2
Day 3
Done
```

Count the steps: the initialization ran once, the body ran 3 times, the update ran 3 times, and the test ran 4 times. The loop control variable reached 4, a value that was never printed, because the update made it 4 and then the test stopped the loop before the body could use it.

Now change one thing: make the test `false` from the very start. In Topic 2.7 you saw that a while loop whose condition is false at the start never runs its body. A for loop behaves the same way, because the test comes before the first pass.

WORKED EXAMPLE A guest list with no guests

```
1 int guests = 0;
2 for (int guest = 1; guest <= guests; guest++)
3 {
4     System.out.println("Welcome, guest " + guest);
5 }
6 System.out.println("Guest list finished");
```

Step	guest	guests	What happens
Line 1		0	guests is set to 0.
Initialization	1	0	guest is created and set to 1.
Test	1	0	1 <= 0 is false; skip the loop.
Line 6			Print the final message.

Guest list finished

The initialization still runs, once, because it comes before the test. But the body never runs, and neither does the update, since the update only ever runs after a pass of the body. A for loop can run its body zero times.

WATCH OUT

A semicolon at the end of the header. Most statements end with a semicolon, so students sometimes put one after the closing parenthesis out of habit:

```
1 for (int round = 1; round <= 3; round++);
2 {
3     System.out.println("Ding!");
4 }
```

You might expect `Ding!` three times, but this prints it only once:

Ding!

Here is why. A semicolon by itself is a complete statement that does nothing. Java takes that empty statement as the loop body. So the loop counts `round` from 1 to 4 doing nothing each pass, ends, and only then does Java reach the braces on lines 2–4. Those braces are now an ordinary block after the loop, so `Ding!` prints once. The code compiles without any warning, which makes this mistake hard to spot.

The fix is to delete the semicolon so the braces become the body again:

```
1 for (int round = 1; round <= 3; round++)
2 {
3     System.out.println("Ding!");
4 }
```

This prints Ding! three times.

PAUSE AND TRY

Question 3 Multiple choice

This loop prints 2, 4, and 6, one per line. How many times does Java evaluate the Boolean expression `page <= 6`?

```
1 for (int page = 2; page <= 6; page += 2)
2 {
3     System.out.println(page);
4 }
```

- A. 2
- B. 3
- C. 4
- D. 6

Question 4 Tracing

What does this code print? Pay attention to when `step` changes.

```
1 for (int step = 1; step <= 3; step++)
2 {
3     System.out.println("Start " + step);
4     System.out.println("End " + step);
5 }
```

2.8.3 Changing the Start, Stop, and Step

Not every loop counts 1, 2, 3 up to some number. A countdown goes backward. A program that prints even numbers jumps by 2. Because each part of the header is an ordinary piece of Java, you can change any of the three to make the loop count the way you need:

- the **initialization** decides where the counting starts;
- the **Boolean expression** decides where it stops;
- the **update** decides how big each step is and in which direction.

The order of execution from 2.8.2 does not change. Only the values do.

WORKED EXAMPLE Counting down

To count down, start high, keep going while the variable is still big enough, and subtract each time.

```
1 for (int countdown = 3; countdown >= 1; countdown--)
2 {
3     System.out.println(countdown);
4 }
5 System.out.println("Liftoff!");
```

Pass	countdown	countdown >= 1	Printed
1	3	true	3
2	2	true	2
3	1	true	1
end	0	false	

```
3
2
1
Liftoff!
```

All three parts changed together. The start is 3, the test uses `>=` because the variable is moving down toward 1, and the update is `countdown--`. If you kept a counting-up test such as `countdown <= 3`, it would stay true as the variable kept shrinking (2, 1, 0, -1, ...), so the loop would not stop at 1.

Now vary only the step: count up, but by 2 instead of 1. Any update from Topic 1.6 works, such as `+= 2`.

WORKED EXAMPLE Stepping by two

```
1 for (int num = 2; num <= 10; num += 2)
2 {
3     System.out.println(num);
4 }
```

Pass	num	num <= 10	Printed
1	2	true	2
2	4	true	4
3	6	true	6
4	8	true	8
5	10	true	10
end	12	false	

The loop runs 5 times, not 10, because each update jumps 2. The loop ends when `num` is 12, a value it reaches by skipping past 11.

Counting how many times a loop runs

To predict output, you often need to know how many times the body runs. Programmers frequently name a plain counting variable `i`, and many loops start at `0`. Compare these three loops, which differ only in their start value and in `<` versus `<=`:

```

1 for (int i = 0; i < 5; i++)
2 {
3     System.out.print(i + " ");
4 }
5 System.out.println();
6 for (int i = 0; i <= 5; i++)
7 {
8     System.out.print(i + " ");
9 }
10 System.out.println();
11 for (int i = 1; i <= 5; i++)
12 {
13     System.out.print(i + " ");
14 }
15 System.out.println();

```

```

0 1 2 3 4
0 1 2 3 4 5
1 2 3 4 5

```

(Each `System.out.println()` with nothing in the parentheses just ends the line of output.)

Header	Values of <code>i</code>	Iterations
<code>for (int i = 0; i < 5; i++)</code>	0, 1, 2, 3, 4	5
<code>for (int i = 0; i <= 5; i++)</code>	0 through 5	6
<code>for (int i = 1; i <= 5; i++)</code>	1 through 5	5

When a loop adds 1 each time, you can count without listing every value:

- With `<`, the stop value itself is left out, so the loop runs **stop – start** times. `0` up to but not including `5` is $5 - 0 = 5$ times.
- With `<=`, the stop value is included, so it runs **stop – start + 1** times. `0` through `5` is $5 - 0 + 1 = 6$ times.

For a loop that steps by more than 1, or counts down, list the values in a short table as in the examples above. That always works.

KEY IDEA

The initialization sets where counting starts, the test sets where it stops, and the update sets the size and direction of each step. With a step of 1, `<` leaves out the stop value and `<=` includes it.

WATCH OUT

The off by one error. Topic 2.7 described a loop that runs one time too many or one time too few. With for loops, the usual cause is choosing `<` when you meant `<=`, or the reverse. This loop is supposed to print pages 1 through 5:

```
1 for (int page = 1; page < 5; page++)
2 {
3     System.out.println("Printing page " + page);
4 }
```

```
Printing page 1
Printing page 2
Printing page 3
Printing page 4
```

Page 5 is missing. When `page` becomes 5, the test `5 < 5` is `false`, so the loop ends before printing it. The code compiles and runs, so only checking the output reveals the bug. Students make this mistake because "up to 5" sounds like `< 5`, but `<` stops **before** 5.

Fix it by including the stop value with `<=`:

```
1 for (int page = 1; page <= 5; page++)
2 {
3     System.out.println("Printing page " + page);
4 }
```

Now the test `5 <= 5` is `true`, the last page prints, and the loop stops when `page` is 6. A quick check for any loop: work out the first value and the last value the body will see, and make sure both are the ones you want.

A different kind of mistake makes the test impossible to turn `false`. In this loop the variable starts at 1 and goes up by 2, so it is always odd. It can never equal 10, so `odd != 10` is always `true`, and the loop is an infinite loop:

```
1 for (int odd = 1; odd != 10; odd += 2)
2 {
3     System.out.println(odd);
4 }
```

Before you run a loop, check that its update moves the loop control variable toward a value that makes the test `false`.

PAUSE AND TRY**Question 5** Multiple choice

How many lines does this code print?

```
1 for (int seat = 0; seat <= 10; seat++)
2 {
3     System.out.println("Seat " + seat);
4 }
```

- A. 9
- B. 10
- C. 11
- D. 12

Question 6 Tracing

What does this code print? Show the value of `coins` and the test for each pass.

```
1 for (int coins = 1; coins < 40; coins *= 3)
2 {
3     System.out.println(coins);
4 }
```

2.8.4 Rewriting Between for and while

You now have two loops. Are they really different tools? Not in what they can do: any for loop can be rewritten as a while loop that does exactly the same thing, and any while loop can be rewritten as a for loop. Knowing how to switch between them helps you read code written either way, and it explains what the for loop is doing behind the scenes.

Go back to the order from 2.8.2: initialization once, then test, body, update, repeated. A while loop can follow that same order if you put each piece in the right place. Here is the general shape of a for loop:

```
for (initialization; test; update)
{
    body
}
```

And here is the while loop that does the same thing:

```
initialization;
while (test)
{
    body
    update;
}
```

- The **initialization** moves to a statement just before the while loop, so it still runs once.
- The **test** becomes the while loop's condition. Both loops check it before every pass.
- The **update** moves to the **last line** of the while body, so it still runs after everything else in the body and before the next test. It now needs its own semicolon, because it is a statement.

KEY IDEA

To turn a for loop into a while loop, put the initialization before the loop, use the test as the while condition, and make the update the last statement in the body. Reverse those moves to go from while to for.

WORKED EXAMPLE The countdown, rewritten as a while loop

Here is the countdown from 2.8.3 again, followed by its while version:

```
1 for (int countdown = 3; countdown >= 1; countdown--)
2 {
3     System.out.println(countdown);
4 }
5 System.out.println("Liftoff!");
```

```
1 int countdown = 3;
2 while (countdown >= 1)
3 {
4     System.out.println(countdown);
5     countdown--;
6 }
7 System.out.println("Liftoff!");
```

Piece	In the for loop	In the while loop
Initialization	int countdown = 3	Line 1, before the loop
Test	countdown >= 1	Line 2, the while condition
Body	println(countdown)	Line 4
Update	countdown--	Line 5, last in the body

Both print the same output:

```
3
2
1
Liftoff!
```

The rewrite works in the other direction too. A while loop with a clear starting value, a test, and an update as its last statement can become a for loop. This one halves a number each pass:

WORKED EXAMPLE A halving while loop, rewritten as a for loop

```
1 int size = 64;
2 while (size > 1)
3 {
4     System.out.println(size);
5     size /= 2;
6 }
```

Line 1 is the initialization, `size > 1` is the test, and `size /= 2` on line 5 is the update, the last statement in the body. Move them into a header:

```
1 for (int size = 64; size > 1; size /= 2)
2 {
3     System.out.println(size);
4 }
```

Both print:

```
64
32
16
8
4
2
```

After 2 is printed, the update makes `size` equal 1 (integer division, `2 / 2`), and `1 > 1` is false.

So which should you use? Either one works, but a for loop is the natural choice when the loop is controlled by a counter that starts, stops, and steps in a way you know ahead of time, because the whole plan sits in the header. A while loop reads more naturally when there is no counter, for example "keep going until the balance drops below zero."

WATCH OUT

Putting the update in the wrong place, or leaving it out. When writing the while version, students sometimes put the update at the top of the body:

```
1 int count = 1;
2 while (count <= 5)
3 {
4     count++;
5     System.out.println(count);
6 }
```

```
2
3
4
5
6
```

It was meant to print 1 through 5, like the for loop in 2.8.1. Because `count++` now runs before the print, every printed value is one higher. 1 is never printed, and 6 is printed on the last pass because the test checked `5 <= 5` before the update made it 6. The fix is to move `count++`; to be the last line of the body, which gives back the code from the start of 2.8.1. A for loop cannot make this mistake, because Java always runs its update after the whole body. Leaving the update out completely is worse. Without a line that changes `count`, it stays 1, the test `1 <= 5` is true forever, and the program prints 1 again and again without stopping, an infinite loop:

```
1 int count = 1;
2 while (count <= 5)
3 {
4     System.out.println(count);
5 }
```

WATCH OUT

Using a for loop's control variable after the loop. A variable declared in a for loop header belongs to that loop. When the loop ends, the variable is gone, so this does not compile:

```
1 for (int lap = 1; lap <= 3; lap++)
2 {
3     System.out.println("Lap " + lap);
4 }
5 System.out.println("Laps run: " + lap);
```

```
error: cannot find symbol
symbol:   variable lap
```

The last line asks for `lap`, but the variable was created inside the header and stopped existing when the loop ended. (Exact error wording can vary between Java versions.) This is one real difference between the two kinds of loops: in the while version, the variable is declared before the loop, so it still exists afterward and holds the value that made the test `false`:

```
1 int lap = 1;
2 while (lap <= 3)
3 {
4     System.out.println("Lap " + lap);
5     lap++;
6 }
7 System.out.println("Final value of lap: " + lap);
```

```
Lap 1
Lap 2
Lap 3
Final value of lap: 4
```

If you need the counter's value after the loop, declare it before the loop, as the while version does.

PAUSE AND TRY

Question 7 Multiple choice

Which for loop prints exactly the same output as this while loop?

```
1 int lives = 3;
2 while (lives > 0)
3 {
4     System.out.println("Lives: " + lives);
5     lives--;
6 }
```

- A. for (int lives = 3; lives > 0; lives--)
- B. for (lives > 0; int lives = 3; lives--)
- C. for (int lives = 3; lives >= 0; lives--)
- D. for (int lives = 0; lives > 3; lives++)

Each choice has the same body: `System.out.println("Lives: " + lives);`

Question 8 Debugging

A student rewrote `for (int ticket = 1; ticket <= 4; ticket++)` as a while loop. The original printed Ticket 1 through Ticket 4, but the rewrite prints Ticket 1 over and over and never stops. Find the error and correct it.

```
1 int ticket = 1;
2 while (ticket <= 4)
3 {
4     System.out.println("Ticket " + ticket);
5 }
```

Putting it together

This program follows a runner through four laps of a 400-meter track. It uses a for loop with an if statement (Topic 2.3) inside its body and keeps a running total. Before reading the trace, predict all five lines of output.

WORKED EXAMPLE Track practice

TrackPractice.java

```

1 public class TrackPractice
2 {
3     public static void main(String[] args)
4     {
5         int totalMeters = 0;
6         for (int lap = 1; lap <= 4; lap++)
7         {
8             totalMeters += 400;
9             if (lap % 2 == 0)
10            {
11                System.out.println("Lap " + lap + ": water break");
12            }
13            else
14            {
15                System.out.println("Lap " + lap + ": keep going");
16            }
17        }
18        System.out.println("Total meters: " + totalMeters);
19    }
20 }

```

Line 5 sets `totalMeters` to 0 before the loop. Line 6 is the header: `lap` starts at 1, the test is `lap <= 4`, and the update is `lap++`. On every pass, line 8 adds 400 to the total, and the if statement on line 9 checks whether `lap` is even (`lap % 2` is the remainder after dividing by 2, so it is 0 for even numbers).

lap (test <code>lap <= 4</code>)	<code>totalMeters</code> after line 8	<code>lap % 2 == 0</code>	Printed
1 (true)	400	false	Lap 1: keep going
2 (true)	800	true	Lap 2: water break
3 (true)	1200	false	Lap 3: keep going
4 (true)	1600	true	Lap 4: water break
5 (false): loop ends	1600		

Then line 18, after the loop, prints the total once:

```

Lap 1: keep going
Lap 2: water break
Lap 3: keep going
Lap 4: water break
Total meters: 1600

```

Three details are worth a second look. The whole body, both line 8 and the if statement, runs before `lap++`, so the message on each pass always uses the same `lap` value that was tested. `totalMeters` is declared before the loop, so it keeps its value from pass to pass and still exists for line 18. And the test ran 5 times while the body ran 4 times; on the fifth test, `lap` was 5, and the loop ended.

Vocabulary

Term	Definition	Example
iterative statement	A statement that repeats a block of code; while and for loops are both iterative statements	<code>while (n > 0)</code>
for loop	A loop whose header holds the initialization, the Boolean expression, and the update	<code>for (int i = 0; i < 3; i++)</code>
loop header	The first line of a for loop, from <code>for</code> to the closing parenthesis	<code>for (int i = 0; i < 3; i++)</code>
loop body	The statements in braces that repeat on each pass	<code>{ System.out.println(i); }</code>
initialization	The first part of a for loop header; runs once, before the first test	<code>int i = 0</code>
Boolean expression	The second part of a for loop header; the body runs only while it is true	<code>i < 3</code>
update	The third part of a for loop header; runs after each full pass of the body	<code>i++</code>
loop control variable	The variable set up in the initialization, checked by the test, and changed by the update	<code>i</code>
iteration	One run of the loop body	The loop above has 3 iterations

Chapter summary

- A for loop puts the three things that control a counting loop in one header, separated by semicolons: the initialization, the Boolean expression, and the update. The variable the initialization sets up is the loop control variable.
- The initialization runs only once. Then the loop repeats: test, whole body, update. The update always comes after the entire body and before the next test, and the test runs one more time than the body. If the test is false at the start, the body runs zero times.
- Changing the start value, the test, or the update changes where a loop starts, where it stops, and how it steps. With a step of 1, `<` leaves out the stop value and `<=` includes it; mixing them up causes off by one errors.
- Every for loop can be rewritten as a while loop by putting the initialization before the loop, using the test as the condition, and making the update the last statement in the body, and a counting while loop can be rewritten as a for loop the same way in reverse.
- A variable declared in a for loop header exists only inside that loop. A stray semicolon after the header makes the loop body empty.

Review questions

Question 9 Multiple choice

How many times does this code print Hi?

```
1 for (int k = 3; k <= 12; k += 3)
2 {
3     System.out.println("Hi");
4 }
```

- A. 3
- B. 4
- C. 9
- D. 10

Question 10 Tracing

This loop counts down by 4 and keeps a running total. Write the exact output, and show `n`, the test, and `total` for each pass.

```
1 int total = 0;
2 for (int n = 10; n > 0; n -= 4)
3 {
4     total += n;
5     System.out.println(n + " " + total);
6 }
```

Question 11 Multiple choice

Which while loop prints exactly the same output as this for loop?

```
1 for (int x = 5; x < 20; x += 5)
2 {
3     System.out.println(x);
4 }
```

Loop 1

```
1 int x = 5;
2 while (x < 20)
3 {
4     x += 5;
5     System.out.println(x);
6 }
```

Loop 2

```
1 int x = 5;
2 while (x < 20)
3 {
4     System.out.println(x);
5     x += 5;
6 }
```

Loop 3

```
1 int x = 0;
2 while (x < 20)
3 {
4     System.out.println(x);
5     x += 5;
6 }
```

Loop 4

```
1 int x = 5;
2 while (x > 20)
3 {
4     System.out.println(x);
5     x += 5;
6 }
```

- A. Loop 1
- B. Loop 2
- C. Loop 3
- D. Loop 4

Question 12 Debugging

This loop should print the even numbers from 10 down to 2: 10, 8, 6, 4, 2. It prints only 10, 8, 6, and 4. Find the error and correct the line.

```
1 for (int num = 10; num > 2; num -= 2)
2 {
3     System.out.println(num);
4 }
```

Question 13 Short answer

A student rewrites the first loop below as a for loop by moving `count++` into the header, which gives the second loop. The first prints 1, 2, 3; the second prints 0, 1, 2. Explain why the outputs differ.

```
1 int count = 0;
2 while (count < 3)
3 {
4     count++;
5     System.out.println(count);
6 }
```

```
1 for (int count = 0; count < 3; count++)
2 {
3     System.out.println(count);
4 }
```

Reflection

Check each statement. If the answer is "not yet," reread the linked subsection.

- ☐ I can name the initialization, Boolean expression, and update in a for loop header. Reread: 2.8.1
The Three Parts of a for Loop Header
- ☐ I can trace a for loop in order: initialization once, then test, body, update. Reread: 2.8.2 The Order a
for Loop Runs In
- ☐ I can change where a for loop starts, stops, and steps, and count its iterations. Reread: 2.8.3
Changing the Start, Stop, and Step
- ☐ I can spot and fix an off by one error caused by `<` versus `<=`. Reread: 2.8.3 Changing the Start, Stop,
and Step
- ☐ I can rewrite a for loop as a while loop, and a counting while loop as a for loop. Reread: 2.8.4
Rewriting Between for and while

Check Your Reasoning

Answers and explanations for every question, numbered to match.

Question 1

C. The update is the third part of the header, after the second semicolon. `floor -= 3` changes the loop control variable after each pass, here by subtracting 3.

- **A** is wrong: `int floor = 10` is the initialization, the first part. It creates the variable and gives it its starting value.
- **B** is wrong: `floor > 0` is the Boolean expression, the second part. It is the test, not the change.
- **D** is wrong: `floor` is the loop control variable itself, not a part of the header. The update is the statement that changes it.

Reread 2.8.1 *The Three Parts of a for Loop Header* if you chose differently.

Question 2

```
10
20
30
40
```

Pass	stars	stars <= 4	Printed
1	1	true	10
2	2	true	20
3	3	true	30
4	4	true	40
end	5	false	

The body prints `stars * 10`, so it uses the current value of the loop control variable each pass. The loop control variable itself still goes up by 1; only the printed value is ten times larger. If you printed 50 too, you ran the body when `stars` was 5, but the test `5 <= 4` is `false`, so the loop ends first.

Reread 2.8.1 *The Three Parts of a for Loop Header* if your answer differed.

Question 3

C. The test runs before every pass and one last time to stop the loop. It is `true` for `page` values 2, 4, and 6 (three passes), and then the update makes `page` equal 8, and the test `8 <= 6` is `false`. That is $3 + 1 = 4$ tests.

- **A** is wrong: it misses both the first test (the one before the first pass) and the final one.
- **B** is wrong: 3 is the number of times the body runs, not the test. The final, `false` test also counts.
- **D** is wrong: 6 is the last value printed. The loop steps by 2, so it does not test once for every number from 1 to 6.

Reread 2.8.2 *The Order a for Loop Runs In* if you chose differently.

Question 4

```

Start 1
End 1
Start 2
End 2
Start 3
End 3

```

Pass	step during the whole body	Printed
1	1	Start 1, End 1
2	2	Start 2, End 2
3	3	Start 3, End 3
end	4 (test is false)	

The update `step++` runs only after both lines of the body have finished. So `step` holds the same value on line 3 and line 4 of every pass. If you wrote `End 2` after `Start 1`, you ran the update in the middle of the body; a for loop never does that.

Reread 2.8.2 *The Order a for Loop Runs In* if your answer differed.

Question 5

C. The loop adds 1 each time and uses `<=`, so it runs $\text{stop} - \text{start} + 1 = 10 - 0 + 1 = 11$ times, printing `Seat 0` through `Seat 10`. Starting at 0 adds a pass that is easy to forget.

- **A** is wrong: 9 would leave out both `Seat 0` and `Seat 10`.
- **B** is wrong: 10 is $\text{stop} - \text{start}$, the count for `<`. With `<=`, the value 10 is included, which adds one more pass.
- **D** is wrong: the loop stops when `seat` becomes 11, and the body never runs for that value.

Reread 2.8.3 *Changing the Start, Stop, and Step* if you chose differently.

Question 6

```

1
3
9
27

```

Pass	coins	coins < 40	Printed
1	1	true	1
2	3	true	3
3	9	true	9
4	27	true	27
end	81	false	

The update `coins *= 3` triples the variable after each pass, so the values grow quickly. After 27 is printed, the update makes `coins` equal 81, and `81 < 40` is `false`. Do not use the $\text{stop} - \text{start}$ formula here; it only works when the step is 1. If you got 1 4 7 ..., you added 3 instead of multiplying.

Reread 2.8.3 *Changing the Start, Stop, and Step* if your answer differed.

Question 7

A. The while loop's initialization is `int lives = 3`, its test is `lives > 0`, and its update, the last line of the body, is `lives--`. Placing them in that order in the header gives choice A, which prints `Lives: 3`, `Lives: 2`, and `Lives: 1`, just like the while loop.

- **B** is wrong: the parts are in the wrong order. The initialization must come first, so this does not compile (Java reports `not a statement` for the first part).
- **C** is wrong: changing `>` to `>=` adds a pass. It also prints `Lives: 0`, an off by one error.
- **D** is wrong: it starts at 0, and the test `0 > 3` is `false` right away, so it prints nothing. Swapping the start and stop values does not reverse the loop.

Reread 2.8.4 *Rewriting Between for and while* if you chose differently.

Question 8

The update is missing. The for loop's third part, `ticket++`, was dropped during the rewrite. With nothing changing `ticket`, it stays 1, the test `1 <= 4` is always `true`, and the loop is an infinite loop. It compiles, because a loop without an update is legal Java; only running it shows the problem.

Corrected code, with the update as the last line of the body:

```
1 int ticket = 1;
2 while (ticket <= 4)
3 {
4     System.out.println("Ticket " + ticket);
5     ticket++;
6 }
```

This prints `Ticket 1`, `Ticket 2`, `Ticket 3`, and `Ticket 4`, and then the test `5 <= 4` is `false`.

Reread 2.8.4 *Rewriting Between for and while* if you missed it.

Question 9

B. The loop control variable takes the values 3, 6, 9, and 12; each one passes the test `k <= 12`, so the body runs 4 times. Then the update makes `k` equal 15, and `15 <= 12` is `false`.

- **A** is wrong: it leaves out 12. With `<=`, the stop value is included when the variable lands on it exactly.
- **C** is wrong: 9 is $12 - 3$, the stop – start formula, which only works when the step is 1. This loop steps by 3.
- **D** is wrong: 10 is $12 - 3 + 1$, the `<=` formula for a step of 1. Again, the step here is 3, so list the values instead.

Reread 2.8.3 *Changing the Start, Stop, and Step* if you chose differently.

Question 10

```
10 10
6 16
2 18
```

n	n > 0	total after line 4	Printed
---	-------	--------------------	---------

10	true	0 + 10 = 10	10 10
6	true	10 + 6 = 16	6 16
2	true	16 + 2 = 18	2 18
-2	false	18	

On each pass, line 4 adds the current `n` to `total` first, and then line 5 prints both. The update `n -= 4` runs only after the print. After the third pass, `n` becomes `-2`; the test `-2 > 0` is `false`, so the loop ends without adding `-2`. The loop never lands on `0`, but it still stops, because the test is `n > 0` rather than `n != 0`.

Reread 2.8.2 *The Order a for Loop Runs In* and 2.8.3 *Changing the Start, Stop, and Step* if your output differed.

Question 11

B. The for loop prints 5, 10, and 15. Loop 2 places each piece where it belongs: the initialization `int x = 5`; before the loop, the test `x < 20` as the condition, and the update `x += 5`; as the last statement of the body. It prints the same three lines.

- **A** is wrong: Loop 1 runs the update before the print, so it prints 10, 15, and 20. A for loop's update always runs after the whole body.
- **C** is wrong: Loop 3 starts at 0 instead of 5, so it prints 0, 5, 10, and 15, one extra line.
- **D** is wrong: Loop 4 flips the test to `x > 20`. Since `5 > 20` is `false` at the start, it prints nothing.

Reread 2.8.4 *Rewriting Between for and while* if you chose differently.

Question 12

The test on line 1 is off by one pass. When `num` reaches 2, the test `2 > 2` is `false`, so the loop ends before printing 2. The test must still be `true` when `num` is 2.

Corrected code:

```
1 for (int num = 10; num >= 2; num -= 2)
2 {
3     System.out.println(num);
4 }
```

Now `2 >= 2` is `true`, so 2 prints; then the update makes `num` equal 0, and `0 >= 2` is `false`. (`num > 0` would also work, since the next value after 2 is 0.) The original compiled and ran, so only checking the last value the body sees reveals the bug.

Reread the Watch Out in 2.8.3 *Changing the Start, Stop, and Step* if you missed it.

Question 13

Model answer: In the while loop, `count++` comes before the print, so each pass adds 1 first and then prints the new value: 1, 2, 3. A for loop always runs its update after the whole body, so moving `count++` into the header moved it to after the print. Now each pass prints first and adds 1 afterward, giving 0, 1, 2. The rewrite rule only produces an equivalent loop when the update is the last statement of the while body, and here it was the first.

A complete answer must:

- say that the while loop updates before printing,

- say that a for loop's update runs after the entire body, so the for loop prints before updating, and
- connect the difference to where the update sat in the while body.

Reread *2.8.2 The Order a for Loop Runs In* and *2.8.4 Rewriting Between for and while* if your answer differed.