

2.8 for Loops

Lesson Plan

Pacing: 3 class periods of 45 minutes

Learning objectives: 2.8.A

Essential knowledge: 2.8.A.1, 2.8.A.2, 2.8.A.3

Materials

- Projector and the Start, Stop, Step slide deck
- Whiteboard and markers
- Mini whiteboards or scrap paper (trace tables)
- IDE on every computer (Days 2 and 3)
- Printed Start, Stop, Step puzzle worksheet (answer key for you) and exit ticket

Agenda

Minutes	Activity
5	Day 1 warm-up — three lines run this loop (slide 2)
12	Day 1 direct instruction — the three parts of the header (slides 3–7)
10	Day 1 direct instruction — the order a for loop runs in (slides 9–10)
13	Day 1 guided practice — count the passes and the tests (slide 11) and worksheet question 2
5	Day 1 closure — say the order out loud
5	Day 2 warm-up — how many passes?
5	Day 2 direct instruction — a semicolon after the header (slide 12)
11	Day 2 direct instruction — start, stop, step (slides 14–15)
6	Day 2 direct instruction — the off by one error (slide 16)
12	Day 2 guided practice — worksheet questions 1 and 4
4	Day 2 independent practice — start worksheet question 3
2	Day 2 closure — first and last value
5	Day 3 warm-up — same loop, one line
12	Day 3 direct instruction — for loop to while loop and back (slides 18, 6)
6	Day 3 direct instruction — the loop control variable's scope (slide 19)
10	Day 3 guided practice — which while loop matches? (slide 20) and worksheet question 6
7	Day 3 independent practice — worksheet questions 5 and 8
5	Day 3 exit ticket

Overview

Day 1 is the `for` loop's header and the order Java runs it in (2.8.A.1, 2.8.A.2): the initialization runs once, then the loop repeats the Boolean expression (the test), the body, and the update until the test is false. The test runs once more than the body. Day 2 is controlling the count: changing where a loop starts, stops, and steps, counting iterations, the off by one error, and the stray semicolon after the header. Day 3 connects `for` to `while` (2.8.A.3): any `for` loop can be rewritten as a `while` loop and a counting `while` loop can be rewritten as a `for` loop, and the loop control variable declared in the header exists only inside the loop.

Students traced `while` loops in Topic 2.7, including the check before every iteration, zero iterations, infinite loops, and off-by-one errors. Stay inside the topic boundary: nested loops are Topic 2.11, string algorithms with loops are Topic 2.10, and the standard algorithms (totals, counting with an `if`, digits, minimum and maximum) are Topic 2.9. Loop bodies today only print or add to a simple total, and the loop control variable is not changed inside the body, as the CED asks. The CED does not ask for the enhanced `for` loop; do not teach it.

TEACHER NOTE

The slides are the Start, Stop, Step deck (22 slides, with speaker notes); slide numbers below refer to it. Day 1 uses slides 1–11, Day 2 uses slides 12–16, and Day 3 uses slides 17–22. The Start, Stop, Step puzzle worksheet (10 questions, about 35 minutes) is split across the three days: question 2 in class on Day 1, questions 1 and 4 in class on Day 2, questions 3 and 7 as Day 2 homework, question 6 in class on Day 3, and questions 5, 8, 9, and 10 after Day 3.

Files whose names end in `.snippet.java` hold only the statements that go inside `main`; paste them into the `main` of a scratch class to run them. All files are in the topic's `examples/` folder.

- **Day 1 demonstrations:** `SeatCount`, `CountWhile`, `CountFor`, `DayTrace`, `CommaHeader.snippet.java` (compile error), `LapMessages`, `StartEnd`, `MiddleUpdateFor`, `MiddleUpdateWhile`.
- **Day 2 demonstrations:** `SemicolonHeader`, `SemicolonHeaderFixed`, `CountCompare`, `StepThree`, `HalvingFor`, `CountDownTotal`, `EvenCountdownBug`, `EvenCountdownFixed`, `PagesOffByOne`, `PagesFixed`.
- **Day 3 demonstrations:** `HalvingWhile`, `LiftoffWhile`, `LoopVariableAfter.snippet.java` (compile error), `LoopVariableAfterFixed`, `FivesFor`, `FivesChoiceA` to `FivesChoiceD`, `LivesFor`, `LivesWhile`.
- **Printed only (exit ticket):** `ForExitTrace` and `ForExitRewrite`.

`EvenCountdownBug` never ends. Run it only after students have predicted that, and stop it with the IDE's stop button.

Day 1 warm-up: three lines run this loop

Show slide 2, the `while` loop from Topic 2.7 that prints `Seat 1` to `Seat 4`. Students point to the three lines that control the loop (the declaration before it, the condition, and the update inside it) and write which of the three is easiest to forget and what happens if you forget it. Expect "the update, and then it never ends." Today's loop gathers all three into one line.

Day 1 direct instruction: the three parts of the header

1. **Goals** (slide 3). Read the five goals; they are spread over the three days.
2. **Same loop, one line** (2.8.A.1, slide 6). Put the `while` version and the `for` version of the seat loop side by side. A `for` loop is an iterative statement whose header has three parts separated by two semicolons: the **initialization**, the **Boolean expression**, and the **update**. Both loops print the same output (`SeatCount`, `CountWhile`, and `CountFor` show the same idea).
3. **The three parts** (slide 5). Initialization (`int seat = 1`) creates the **loop control variable** and gives it a starting value. The Boolean expression (`seat <= 4`) is the test; the body runs only while it is true. The update (`seat++`) changes the variable after each pass, so the test can become false.
4. **Semicolons, not commas** (slide 7). Show `CommaHeader.snippet.java`: it does not compile and the error is `';' expected`. Fix it by replacing the commas with semicolons.

MISCONCEPTION Commas in the header

Students separate the three parts with commas, as in a method call. The parts are separated by semicolons. Have students read the header aloud as "start, test, update" and touch a semicolon between each.

MISCONCEPTION The header is the whole loop

Students think the braces after the header are optional decoration. The body is what repeats; the header only controls how many times. Point at the body and say its statements are what runs on every pass.

Day 1 direct instruction: the order a for loop runs in

1. **The five steps** (2.8.A.2, slide 9). The initialization runs once, before anything else. Then Java tests the Boolean expression: if it is false, the loop is over. If it is true, the whole body runs top to bottom, and only then does the update run. Then Java goes back to the test with the new value.
2. **A three-day loop** (slide 10). Walk through the table row by row: `day 1, 2, 3` are true and each prints a line; `day 4` is false and nothing runs. Count together: 1 initialization, 4 tests, 3 bodies, 3 updates. Run `DayTrace` to confirm.
3. **Zero iterations**. If the test is false the first time, the body never runs. The test always runs one more time than the body, because the last test is the false one.
4. **Update after the body**. Compare `MiddleUpdateFor` (prints 0, 1, 2) with `MiddleUpdateWhile`, where the update is the first body line (prints 1, 2, 3). In a `for` loop the update is written first in the header but runs after the body.

MISCONCEPTION The update runs before the body

Students read the header left to right and run the update early because it is on the line above the body. The update is written in the header but runs after every complete pass. This is worksheet question 2's "Think first" box.

MISCONCEPTION The initialization runs on every pass

Students reset the variable each time. The initialization runs exactly once. Ask: "If it ran every time, would the loop ever end?"

Day 1 guided practice: count the passes and the tests

Students work slide 11: `for (int p = 5; p <= 20; p += 5) printing p`. They write the times the body runs (4), the times the test runs (5), and the value of `p` that stops it (25). Then they do worksheet question 2 (you're the JVM) in pairs: `n` goes 2, 5, 8 and 11 stops it, so the output is 2 5 8 15, choice A. For each wrong choice ask which step of the order it skips.

TEACHER NOTE

Question 2's wrong choices: **B** (2 5 7) prints 7, which is not a value `n` takes (2, 5, 8). **C** (5 8 11 24) runs the update before the first pass. **D** (2 5 8 11 26) runs the body once too often, as if `11 <= 8` were true. Circulate and look for students who skip the table on a "simple" loop.

Day 1 closure

Partners say the order out loud with the slide 22 starter: "First the initialization runs once. Then the loop repeats test, body, update until the test is false."

Day 2 warm-up: how many passes?

Write `for (int i = 0; i < 4; i++)` on the board and ask how many times the body runs and which values `i` takes. Students answer on whiteboards: 4 times, 0 to 3. Then change `<` to `<=` and repeat: 5 times, 0 to 4.

Day 2 direct instruction: a semicolon after the header

Show slide 12 and ask students to predict the output of three beeps. It prints one `Beep!`: the loop ran 3 times with an empty body (the lone `;`), then the braces ran once as an ordinary block. Run `SemicolonHeader` (Ding! once) and `SemicolonHeaderFixed`. The program compiles either way, so the only warning is the output. When a loop prints once, check the end of the header line.

MISCONCEPTION A semicolon after the for header

Students end the header with `;` as they would end a statement. The `;` becomes the body, and the braces are no longer part of the loop. Teach the habit of reading the whole header line for a stray `;` whenever output appears once instead of many times.

Day 2 direct instruction: start, stop, step

1. **Counting iterations** (slide 14). Fill in the slide's table. Initialization decides where counting starts, the Boolean expression decides where it stops, and the update decides the size and direction of the step. `i < 4` from 0 gives 4 iterations; `i <= 4` from 0 gives 5; `i <= 4` from 1 gives 4. Run `CountCompare`.
2. **Different starts and steps** (slide 15). Run `StepThree` (a step of 3 runs 4 times), `HalvingFor`, and the slide examples: `row` counting down by 4 prints 9, 5, 1; `n *= 3` prints 3, 9, 27. Counting down needs `>` or `>=` and an update that goes down.
3. **Check the first and last value** the body sees, and that the update moves toward making the test false. Show `EvenCountdownBug` only after students predict that a countdown with `c++` never ends, then `EvenCountdownFixed`.

MISCONCEPTION Counting down with the wrong update or test

Students write `c > 0` and `c++`. The loop never ends because every update moves away from the stop. Ask: "Which way does the update push the variable, and which way does the test need it to go?"

Day 2 direct instruction: the off by one error

Slide 16. The goal is pages 1 through 5; `page < 5` stops at 4 and `page <= 5` includes 5. Run `PagesOffByOne` and `PagesFixed`. Students list the starting value, the last value the body should see, and which operator allows it. The same check applies to loops that start at 0: `i < 6` runs 6 times, `i <= 6` runs 7.

MISCONCEPTION Using `<` when the last value must be included

Students leave out the stop value. When a task says "through 5", the test must let 5 pass. Have students test their header on the last value the output needs and on the value just after it.

Day 2 guided practice

Students complete worksheet question 1 (header table) in pairs, then question 4 (snippet pool) with the whole class checking: 12, `t > 0`, `t -= 4` prints 12 8 4 go!. The pairs must write the iterations and last value for every row, including the loop that never runs

(`for (int i = 10; i < 5; i++)` gives none). Key rows: `i = 1; i <= 10; i += 3` runs 4 times and ends on 10; `i = 2; i < 40; i *= 2` runs 5 times and ends on 32.

Day 2 independent practice

Students start worksheet question 3 (code tiles: running score) and finish it with question 7 (face-off) as homework. The decoy tile uses `round < 3`, which would print only two rounds.

Day 2 closure

Students write the first and last value `i` takes in `for (int i = 3; i <= 12; i += 3)` (3 and 12) and how many times the body runs (4).

Day 3 warm-up: same loop, one line

Show `FivesFor` without running it. Students write what it prints on whiteboards, then compare with `FivesChoiceA` to `FivesChoiceD` when you show them. Do not explain the difference yet; the point is that two loops can look different and print the same thing.

Day 3 direct instruction: for loop to while loop and back

1. **Rewriting a `for` as a `while`** (2.8.A.3, slide 18). The initialization becomes a statement just before the loop, the Boolean expression becomes the `while` condition, and the update becomes the last statement of the body. `HalvingFor` and `HalvingWhile` print the same output.
2. **Rewriting a counting `while` as a `for`** (slide 6, `LiftoffWhile`). Move the declaration into the header, the condition becomes the test, and the final `seconds--` becomes the update.
3. **Which one to use.** When the counter starts, stops, and steps in a way you know ahead of time, `for` puts the plan on one line. A `while` loop with no counter ("keep going until the user quits") says what it means. Neither is more powerful.
4. **Update placement.** A `while` version with the update at the top of the body does not match the `for` loop; the `for` update runs after the body (`LivesFor` and `LivesWhile`).

MISCONCEPTION The update goes first in the while version

Students put the update at the top because it is first in the `for` header. The update must be the last statement in the body, or the output shifts by one step. Compare the outputs line by line.

MISCONCEPTION A while loop can do things a for loop cannot

Students think the two are different in kind. Any `while` loop can be rewritten as a `for` loop and the reverse. Choosing between them is about readability.

Day 3 direct instruction: the loop control variable's scope

Slide 19. A variable declared in the header exists only inside the loop. Show

`LoopVariableAfter.snippet.java`: printing `lap` after the loop gives `cannot find symbol`. Two fixes: declare `lap` before the loop (and drop `int` in the header), as in `LoopVariableAfterFixed` and the `while` version on the slide, which prints `Lap 1`, `Lap 2`, `3`. Connect it to the rewrite: a `while` loop's variable is declared before the loop and so it survives.

MISCONCEPTION Using the loop variable after the loop

Students expect the final value of the loop variable to be available after the loop. When the variable is declared in the header, it is gone once the loop ends. Declare it before the loop if the value is needed later.

Day 3 guided practice: which while loop matches?

Slide 20: `for (int k = 10; k > 0; k -= 3)` printing `k`. Students decide between Version A and Version B; the answer is B (it prints `10 7 4 1`; version A updates first and prints `7 4 1 -2`). Then they do worksheet question 6 (match-up) in pairs. Matches: loop 1 prints R (`4 6 8 10`), loop 2 prints P (`2 4 6`), loop 3 prints S (`2 4 6 8`) and is the one that matches the `for` loop, loop 4 prints Q (`0 2 4 6 8`).

Day 3 independent practice

Students do worksheet questions 5 (you're the compiler) and 8 (one-token fix: change `c++` to `c--` so the loop counts down) alone. Questions 9 and 10 are homework. In question 5, snippet 1 has commas in the header and does not compile; snippet 2 prints `10 5 2 1`; snippet 3 does not compile because `k` is out of scope; snippet 4 compiles and prints `hop` once because of the stray semicolon.

Day 3 exit ticket

Give the exit ticket for the last 5 minutes. Collect it as students leave.