

1.6 Compound Assignment Operators

Lesson Plan

Pacing: 1 class period of 45 minutes

Learning objectives: 1.6.A

Essential knowledge: 1.6.A.1, 1.6.A.2

Materials

- Projector
- Whiteboard and markers
- Scrap paper or mini whiteboards for predictions
- IDE on every computer
- Update in Place slide deck
- Printed Update in Place puzzle worksheet and exit ticket

Agenda

Minutes	Activity
5	Warm-up — update the balance the long way (slide 2)
9	Direct instruction — +=, the += slip, and the five compound operators (slides 3–10)
7	Direct instruction — the type still decides, and the right side first (slides 11–15)
6	Direct instruction — ++ and --, as statements on their own (slides 16–18)
11	Guided practice — trace, run, and change the food drive program
7	Closure — say it shorter (slides 19–20), then the exit ticket

Overview

One period on a single learning objective (1.6.A): write assignment statements with compound assignment operators and work out the value stored. Students learn +=, -=, *=, /=, and %= (1.6.A.1), and the post-increment ++ and post-decrement -- operators (1.6.A.2). The CED's suggested skill is 3.A, determining the result of statements by their execution order, so most of the period is tracing: every update starts from the value the variable holds **at that moment**.

Students already have everything this topic builds on: `int` and `double` variables (Topic 1.2), the five arithmetic operators, integer division, `%`, and precedence (Topic 1.3), assignment that evaluates the right side first and the rule that a variable needs a value before it is used (Topic 1.4), and widening and the `(int)` cast (Topic 1.5). Stay inside the topic boundary:

- A label and a value still take a `print` followed by a `println`; joining text and numbers in one statement is Topic 1.15.
- No comments in code (Topic 1.8) and no `Math` methods (Topics 1.10 and 1.11).

- `++` and `--` appear only **after** the variable, as a statement on their own line. The CED excludes the prefix form (`++x`) and using `++` or `--` inside another expression. If a student asks, say both exist in Java but follow extra rules that are not part of the course or the exam, and move on.

TEACHER NOTE

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.

- **Direct instruction:** `ScorePlusEquals`, `CartTotal`, `PlusEqualsTypo`, `FiveOperators`, `GemChain`, `BalanceChain`, `HiddenCast`, `HiddenCastLong`, `HiddenCastFixed`, `RightSideFirst`, `RightSideMore`, `ExpansionWrong`, `LapCounter`, `TemperatureUp`, `NotAStatement`, and `LapsUninitialized`.
- **Guided practice:** `FoodDrive`, and `FoodDriveRecount` as the model for the change in step 2 (do not show it until students have predicted).
- **Printed only (exit ticket):** `ExitCoins`. The puzzle worksheet and its key are separate Word files; their code is printed on the page.

The Update in Place deck runs in order through the period; its speaker notes hold the answers to every Pause and Try slide. Before class, open a scratch class in the IDE next to the deck so each snippet is one paste away. Compiler messages are quoted from a recent JDK; the wording can vary slightly between Java versions.

Warm-up: update the balance

Show slide 2 (Update the balance): an account starts with \$50, gets a \$20 deposit, then a \$15 purchase. Students write the two updates the Topic 1.4 way and the final balance. Expected: `balance = balance + 20;`, `balance = balance - 15;`, and it displays 55. Ask: "Which value of `balance` did the second update use, 50 or 70?" It used 70, because the first update replaced 50 (Topic 1.4: the right side is evaluated first, then the result replaces the old value).

Then point at the variable name: it appears twice in each update. "Programs update a variable from its own value all the time. Today you learn the short form."

TEACHER NOTE

Keep the warm-up to five minutes. If many students wrote 35, they used the starting value; that is the most important error to fix today, so name it now and come back to it in every trace.

Direct instruction: `+=`, the five operators, and the `=+` slip

Read the goals on slide 3, then start Part 1 (slide 4).

1. **`+=` means "add to"** (1.6.A.1, slide 5). `score += 10;` does exactly what `score = score + 10;` does: Java reads the current value of `score`, adds 10, and stores the result back in `score`. A **compound assignment operator** does arithmetic between the variable's current value and the

value on the right, then assigns the result to the variable on the left. "Compound" means it does two jobs in one symbol. Run `ScorePlusEquals: 50`, then `75`, because line 4 starts from `50`, not `40`.

2. **Only the left side changes** (slide 6, a running total that starts at 0). Run `CartTotal: total += price`; twice and `total += 1.25`; give `8.25`. `price` is only read, so it stays `3.5`. The left side must be a variable, because the result has to be stored somewhere.
3. **The `+=` slip** (slide 7). Run `PlusEqualsTypo: score += 10`; displays `10`, not `50`. Java reads it as `score = +10`; , a plain assignment of positive ten. It compiles, so only the output shows the bug. Memory hook: the arithmetic happens first, so the arithmetic symbol comes first.
4. **All five operators** (1.6.A.1, slides 8–9). Each row of slide 9 starts with `n` at `20`: `-- 6` gives `14`, `*= 3` gives `60`, `/= 6` gives `3`, `%= 6` gives `2`. Run `FiveOperators` to see all five with `6` on the right: `26`, `14`, `120`, `3`, `2`. Ask: "Why is `/= 3` and not `3.33`?" Both values are `int`, so Topic 1.3's integer division applies.
5. **Predict a chain** (slide 10). Students trace `temp` before you reveal the table: `50`, `42`, `84`, `16`, `2`. Then run `GemChain`, the same story with gems (spend `8`, double, trade five-for-one, keep the remainder after groups of five): `42`, `84`, `16`, `1`. It displays `Gems: 1`.

MISCONCEPTION `+=` is the same as `+=`

The two look almost identical, and both compile. `x += 5`; stores positive five and throws the old value away; `x -= 5`; stores negative five. Students will not get an error message, so teach them to check the order of the symbols whenever an update gives a suspiciously small or negative value. The puzzle worksheet's Watch Out box (after question 4) repeats it.

MISCONCEPTION Every update starts from the starting value

In a chain of updates, students apply each operator to the value from the declaration instead of the value the line before stored. In `GemChain` (and slide 10) they write `50 / 5` on line 4 instead of `84 / 5`. Insist on a trace table with one row per line for every chain today, and ask "what does the variable hold right now?" before each row.

Direct instruction: the right side first, and the type still decides

1. **The type rules still apply** (slide 11). Run `BalanceChain`: the same kinds of updates on a `double` keep the decimal part, so `135.0 / 4` gives `33.75`. Compare line 4 of `GemChain`, where both values were `int` and the decimal part was dropped. The operator is the same; the types decide the result. Then give pairs slide 12 (Trace k): `12`, `2`, `8`, `2`. Line 3 is where most mistakes happen: `12 / 5` is `2`, not `2.4`.
2. **The hidden cast**. Run `HiddenCast: int price = 15; price *= 1.5`; displays `22`. `15 * 1.5` is the `double` `22.5`, and a compound operator quietly converts its result to the variable's type, as an `(int)` cast would (Topic 1.5), so the decimal part is truncated. Then run `HiddenCastLong`: the long form, `price = price * 1.5`; , does not compile (possible lossy conversion from `double` to `int`). This is the one place where `x *= value` and `x = x * value` behave differently. The fix (`HiddenCastFixed`) is to use a `double` when a value can have a decimal part.

3. **Predict: RightSideFirst** (Part 3, slides 13–14). `stock` is 10; what does `stock *= 2 + 3`; store? Many students apply Topic 1.3 precedence and say 23. Reveal 50. Java evaluates the **entire** right side first, then combines it with the variable: `stock = stock * (2 + 3)`. Run `RightSideMore` to show the same rule with `-=` and `/=` (20 and 10); slide 14 shows it with `stock -= sold + lost`; (24) and `pay *= 3 - 1`; (14). Then give pairs slide 15 (Right side first): 2, 3, 10.
4. **Expanding without parentheses.** Run `ExpansionWrong`: `seatsA = seatsA - 4 + 6`; gives 32, while `seatsB = seatsB - (4 + 6)`; gives 20. When you rewrite a compound operator the long way, put parentheses around the old right side.

MISCONCEPTION Precedence splits the right side

Students who just learned that `*` happens before `+` read `stock *= 2 + 3`; as `stock * 2 + 3`. A compound operator is not part of the arithmetic on its right; it waits for that arithmetic to finish. Have students draw parentheses around the right side of every compound operator before they evaluate it, then do the inside first.

MISCONCEPTION An int update keeps the decimal part

Students expect `price *= 1.5`; on an `int` to store 22.5, or expect the line not to compile because the long form would not. It compiles and truncates. Two questions catch it: "What type is the variable on the left?" and "Can that type hold the result?" If the answer to the second is no, the decimal part is dropped, not rounded.

Direct instruction: ++ and --

Start Part 4 (slide 16).

1. **Adding or subtracting one** (1.6.A.2, slide 17). The **post-increment operator** `++` adds 1 to the value stored in a numeric variable, and the **post-decrement operator** `--` subtracts 1; the new value is assigned to the variable. `laps++`; `laps += 1`; and `laps = laps + 1`; all do the same thing. Run `LapCounter`: `Laps`: 3 and `Lives`: 2. Starting a counter at 0 and adding one each time something happens is a pattern students will use constantly.
2. **Any numeric variable.** Run `TemperatureUp`: 71.5 goes up one to 72.5, then down two to 70.5. `++` changes the value by exactly one; it does not move to the next whole number.
3. **One form only.** In this course `++` and `--` go after the variable, in a statement by themselves. The same goes for every compound operator.
4. **It must store something** (slide 18: `count + 1` displays 6 but `count` is still 5). Run `NotAStatement`: `laps + 1`; gives not a statement. It computes a value and puts it nowhere. Then run `LapsUninitialized`: `int laps`; `laps++`; gives variable `laps` might not have been initialized, because `++` starts by reading the current value, and there is none (Topic 1.4).

MISCONCEPTION `laps + 1;` adds one to laps

Students write the arithmetic without an assignment and expect the variable to change. Only `=`, the compound operators, `++`, and `--` store a value. Ask: "Where does the answer go?" If there is no variable to the left of an assignment operator and no `++` or `--`, the answer goes nowhere, and Java will not compile the line.

MISCONCEPTION A counter can start without a value

Students declare `int count;` and start with `count++;`, thinking a new `int` holds 0. Every compound operator, including `++`, reads the variable first, so the variable must already have a value. Make "give the counter a starting value when you declare it" part of the counter pattern: `int count = 0;`.

Guided practice: the food drive program

Show `FoodDrive`. A class collects 36 cans on the first day and 18 on the second, removes 4 dented cans, and then a sponsor donates one can for every can collected. The cans are packed in boxes of 12.

1. **Predict on paper.** Pairs build a trace table with a column for `cans` and one for `boxes`, then write the exact output. Run it to check: Full boxes: 8 and Cans left over: 4. Trace: 36, 54, 50, 100; boxes copies 100, then $100 / 12$ is 8; $100 \% 12$ is 4.
2. **Change and predict.** Change line 3 to `cans -= 4 + 2;` (two more cans were dented). Pairs predict before running. Expected: `cans` becomes 96 after line 4, so Full boxes: 8 and Cans left over: 0 (`FoodDriveRecount`). Students who get 104 cans (and 8 left over) subtracted 4 and then added 2.
3. **Rewrite the long way.** Pairs rewrite lines 2–4 and line 6 as plain assignment statements: `cans = cans + 18;`, `cans = cans - (4 + 2);`, `cans = cans * 2;`, `boxes = boxes / 12;`. Check that every rewrite of line 3 has the parentheses.

Debrief: ask why line 5 had to come before line 7. `boxes` must copy the total before `%=` replaces it with the leftover count; after line 7, `cans` no longer holds 96 or 100.

TEACHER NOTE

If computers are not available, run the program on the projector after each prediction. Pairs that finish early can add a counter: declare `int trips = 0;` and add `trips++;` once for each of the two days cans were collected, then display it.

Independent practice

The Update in Place puzzle worksheet (ten questions, about 30 minutes) is homework. Part A (questions 1–3) traces chains of compound operators and the right-side rule; Part B (questions 4–6) orders code tiles, picks operators from a pool, and compares `/=` and `%=` on `int` and `double`

values; Part C (questions 7–10) catches compiler errors (no starting value, `laps + 1;`, a literal on the left), fixes one operator, reviews overflow from Topic 1.5 with `++`, and contrasts `=` with `+=`.

Closure and exit ticket

Flash slide 19 (Words to know), then show slide 20 (Say it shorter): pairs rewrite the four long-form updates (`miles += 12;`, `cash -= price;`, `eggs %= 12;`, `count++;`). Ask the class to finish one more sentence aloud: "`x -= a + b;` means ..." (`x = x - (a + b);`). Hand out the puzzle worksheet, then give the exit ticket (about five minutes). Question 1 checks tracing a chain of compound operators with an expression on the right (1.6.A.1), and question 2 writing `+=` and `++` and recognizing the `+=` slip (1.6.A.1, 1.6.A.2). Re-teach the weaker one at the start of Topic 1.7.