

1.3 Expressions and Output

Lesson Plan

Pacing: 3 class periods of 45 minutes

Learning objectives: 1.3.A, 1.3.B, 1.3.C

Essential knowledge: 1.3.A.1, 1.3.B.1, 1.3.B.2, 1.3.B.3, 1.3.C.1, 1.3.C.2, 1.3.C.3, 1.3.C.4, 1.3.C.5, 1.3.C.6

Materials

- Projector and the Cursor and Calculator slide deck
- Whiteboard and markers
- Mini whiteboards or scrap paper (Day 2)
- IDE on every computer (Days 1 and 3)
- Printed Cursor and Calculator puzzle worksheet (answer key for you) and exit ticket

Agenda

Minutes	Activity
5	Day 1 warm-up — predict the lunch menu
12	Day 1 direct instruction — <code>println</code> , <code>print</code> , and the cursor (slides 3–7)
13	Day 1 direct instruction — literals, string literals, escape sequences (slides 8–11)
12	Day 1 guided practice — match the poster output in the IDE
3	Day 1 closure — where does the next output land?
5	Day 2 warm-up — calculator vs. Java (slide 2)
10	Day 2 direct instruction — arithmetic expressions and the type of the result (slides 12–13)
10	Day 2 direct instruction — integer division (slide 15)
8	Day 2 direct instruction — the remainder operator (slide 14)
9	Day 2 guided practice — whiteboard rounds, value and type (slide 16)
3	Day 2 closure — back to the eggs (slide 22)
5	Day 3 warm-up — 14 or 20?
12	Day 3 direct instruction — operator precedence and parentheses (slides 17–18, 20)
8	Day 3 direct instruction — dividing an int by zero (slide 19)
13	Day 3 guided practice — predict, then break <code>PizzaParty</code> on purpose
7	Day 3 closure — exit ticket

Overview

Day 1 is output: what `System.out.println` and `System.out.print` do to the cursor (1.3.A.1), what a literal and a string literal are (1.3.B.1, 1.3.B.2), and the three escape sequences `\`, `\\`, and `\n`

(1.3.B.3). Day 2 is arithmetic: expressions built from values, variables, and the five operators, and the rule that decides whether a result is an `int` or a `double` (1.3.C.1, 1.3.C.2), then integer division (1.3.C.3) and the remainder operator (1.3.C.4). Day 3 is compound expressions: operator precedence, left-to-right order, and parentheses (1.3.C.5), and why dividing an `int` by the `int` zero compiles but throws an `ArithmeticException` (1.3.C.6).

Students have seen `System.out.println` display text (Topic 1.1) and have declared `int`, `double`, and `boolean` variables (Topic 1.2). Stay inside the topic boundary. Every program only displays values; students do not join text and numbers in one statement (that is Topic 1.15), so a label and a value always take a `print` followed by a `println`. Variables are declared with literal starting values, as in Topic 1.2; say that the full rules for `=` are Topic 1.4. Do not convert between `int` and `double` (Topic 1.5), use `+=` (Topic 1.6), or declare `String` variables. Two exclusions from the CED: `%` is used only with a first operand that is not negative and a positive second operand, and dividing by zero is shown only with two `int` values.

TEACHER NOTE

The slides are the Cursor and Calculator deck (22 slides, with speaker notes); slide numbers below refer to it. Day 1 uses slides 1 and 3–11 (output and literals), Day 2 uses slides 2 and 12–16 and closes with slide 22, and Day 3 uses slides 17–21. The Cursor and Calculator puzzle worksheet (10 questions, about 30 minutes) is split across the three nights by topic: questions 2, 6, and 8 after Day 1; questions 1, 3, and 7 after Day 2; questions 4, 5, 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:** `MenuBoard.java`, `ReadySetGo`, `LabelThenValue`, `PrintRunsTogether`, `PrintRunsTogetherFixed`, `LiteralKinds`, `DoubleLiterals`, `MissingQuotes.java`, `QuoteInside.java`, `EscapeQuotes`, `NewlineEscape`, `PathIllegal.java`, `PathNewline`, `PathFixed`, `PrintlnTypo.java`. `FairPoster.java` is the model solution for the guided practice; do not show it until students have tried.
- **Day 2 demonstrations:** `TicketMath`, `OperatorTable`, `ResultTypes`, `TicketVariables`, `QuotedExpression`, `CookieShare`, `AverageTwo`, `HalfCup`, `HalfCupFixed`, `CookieRemainder`, `RemainderCases`, `PercentMistake`.
- **Day 3 demonstrations:** `PrecedenceSteps`, `AverageParens`, `PerPerson`, `PerPersonFixed`, `MixedOrder`, `TeamSplit.java`, `ZeroDivisor`, and `PizzaParty.java` (create it in the IDE before class; students also type it).
- **Printed only (exit ticket):** `QuickCheck` and `NoGuests`.

Day 1 warm-up: predict the lunch menu

Show `MenuBoard.java` in the IDE without running it. Students write the exact output, one console line per line of paper. Most will write three lines; ask what line 7, `System.out.println()`; with nothing inside the parentheses, does. Take two or three answers ("nothing", "an error", "a blank line"), then run it: Today's lunch:, Veggie tacos, an empty line, Enjoy!.

TEACHER NOTE

Do not explain `public class` or `main` again. Remind students that the lines inside `main` are the steps and run top to bottom (Topic 1.1).

Day 1 direct instruction: `println`, `print`, and the cursor

1. **Goals** (slide 3). Goals 1 and 2 are today's; goals 3–5 are Days 2 and 3.
2. **The console and the cursor** (1.3.A.1). Output goes to the **console**, the IDE's output window. A marker called the **cursor** shows where the next character will be written. Every display starts at the cursor.
3. **`println` does two jobs, in order** (slide 5): display the value inside the parentheses, then move the cursor to the start of the next line. Slide 5 is the warm-up program with a cursor table: line 7 had nothing to display, but still moved the cursor, so an empty line appeared.
4. **Predict: ReadySetGo.** Two `print` statements, then two `println` statements. Students predict, then run it: `Ready set go!` on one line and `Race over.` on the next. `print` displays its value and leaves the cursor right after it. The only spaces in the output are the ones inside the quotes.
5. **Labels and values** (slide 6). Show `LabelThenValue: print` for the label, `println` for the value, so each pair ends its own line. `"Level:"` has no space, so 3 lands against the colon. Joining text and a number in one statement waits for Topic 1.15.
6. **Check: where is the cursor?** (slide 7). Students write the output alone, then compare with a partner: `XY`, `Z`, `W`. The empty `println()` does **not** make an empty line here, because the cursor was in the middle of a line; it just ends `Z`.

MISCONCEPTION `print` starts its own line

Students write `print` for every statement and expect one line each (`PrintRunsTogether` displays `Line oneLine two`). Give them one sentence: `print` never moves to a new line, so the next output starts exactly where it stopped. Run `PrintRunsTogetherFixed` to show the fix.

MISCONCEPTION An empty `println()` always makes an empty line

It makes an empty line only when the cursor is already at the start of a line. In the middle of a line, it just ends that line. Have students say where the cursor is before every `println()` they trace.

MISCONCEPTION `println` with a capital `I`

In many fonts a lowercase `l` and a capital `I` look alike. Java is case-sensitive, so `printIn` gives `cannot find symbol` and nothing runs (`PrintlnTypo.java`). Spell it out loud once: p-r-i-n-t-l-n, "print line."

Day 1 direct instruction: literals, string literals, escape sequences

1. **Predict: LiteralKinds** (slide 9 afterward). Five `println` statements: `42`, `3.75`, `true`, `"42"`, and `"Room 101, 2nd floor!"`. Students predict, then run it. Lines 1 and 4 look the same on the console.
2. **Literals** (1.3.B.1, 1.3.B.2). A **literal** is the code for a fixed value. The way it is written tells Java its type: no decimal point is an `int`, a decimal point is a `double`, `true` and `false` with no quotes are `boolean`, and characters in double quotes are a **string literal**. The quotes mark where the text starts and stops and are not displayed. `"42"` is text, the character 4 followed by the character 2; Java calculates with `42` but only displays `"42"`.
3. **How a double literal is displayed** (slide 9). Run `DoubleLiterals`: `2.50` displays as `2.5`, but `7.0` keeps its `.0`. A `double` always shows at least one digit after the decimal point. Inside a string literal every character, including every space, is kept; `" "` is the empty string.
4. **Predict: QuoteInside.java**. `"She said 'hi' to me."` Ask whether it compiles. Run it: errors such as `' )' or ', ' expected` (the wording varies between Java versions). The second quote ended the string literal.
5. **Escape sequences** (1.3.B.3, slide 10). Inside a string literal, a backslash `\` starts an **escape sequence**: two characters that stand for one special character. The course uses three: `\` (a double quote), `\\` (one backslash), and `\n` (a new line). It is the **backslash**, not the forward slash `/`. Run `EscapeQuotes`, then `NewLineEscape`: one `println` can display two lines, and a `print` whose text ends in `\n` still leaves the cursor on a new line.

MISCONCEPTION Forgetting the quotes around text

Without quotes, Java reads a word as the name of a variable. `MissingQuotes.java` gives `cannot find symbol` because there is no variable named `Welcome`. Ask: "Is this a fixed piece of text? Then it needs quotes."

MISCONCEPTION A single backslash in a file path

Students type `C:\data` as it appears on screen. `\d` is not an escape sequence Java knows, so `PathIllegal.java` reports `illegal escape character`. Worse, `C:\new` compiles: `\n` becomes a new line and the output is `Saved in C:` and `ew` on two lines, a logic error (`PathNewLine`). The fix for both is `\\` (`PathFixed`). Slide 11 shows the `C:\new` case side by side with its fix.

MISCONCEPTION The quotes are part of the text

Students expect `System.out.println("Hi");` to display `"Hi"` with quotes. The quotes are markers only. To display a quote, the text must contain `\"`.

Day 1 guided practice: match the poster

Write the target output on the board or project it from this plan. Students create `FairPoster.java` in the IDE and write a program that displays it exactly, using **exactly four statements**, at least one `print`, and at least one `\n`:

```
Spring "Code" Fair
Files go in C:\fair\projects
Room 101

Bring a friend!
```

Students run their program and compare it with the target character by character. Debrief by asking two or three students how they made the empty line and how they displayed the quotes and the backslashes. There is more than one correct program; `FairPoster.java` is one.

TEACHER NOTE

The four-statement limit forces at least one line break to come from `\n`, because five console lines need at least one statement that ends two lines. Common fixes you will make while circulating: a forward slash `/n`, a single backslash in the path, and an empty line that disappears because a `print` left the cursor mid-line. If computers are not available, do this on paper and run two student versions on the projector.

Day 1 independent practice

Hand out the puzzle worksheet. Questions 2, 6, and 8 (be the JVM with `print` and `println`, the escape-sequence match-up, and a one-token fix to a line break) are homework. Students keep the worksheet for Days 2 and 3.

Day 1 closure

Ask: "After `System.out.print("A");` runs, where is the cursor? After `System.out.println("A");`?" Students answer on a sticky note or in their notes: right after the `A`; at the start of the next line.

Day 2 warm-up: calculator vs. Java

Show slide 2. Students write a guess for each question: what Java says for `7 / 2` when a calculator says 3.5, and how many full cartons and leftover eggs 50 eggs make in cartons of 12. Take a few guesses without confirming them; the closure (slide 22) comes back to both.

Day 2 direct instruction: arithmetic expressions and the type of the result

1. **Vocabulary** (1.3.C.1). An **expression** is code that Java evaluates to produce a single value. An **arithmetic expression** is built from numeric values (literals and variables) and **arithmetic operators**. The values an operator works on are its **operands** (the word from Topic 1.2).
2. **The five operators** (1.3.C.2, slide 13). Show the operator table and run `OperatorTable: +, -, *` (there is no `x` key), `/`, and `%`. Division and remainder get their own segments today.

3. **Calculate or display?** Run `TicketMath`: `3 * 12` displays `36`, and `"3 * 12"` displays the characters `3 * 12`. The quotes turn anything into text that is displayed, not calculated. Slide 13 shows both lines.
4. **Predict: ResultTypes** (slide 13). `6 * 2`, `6 * 2.0`, `1.5 + 2.5`, `10 - 0.25`. Run it: `12`, `12.0`, `4.0`, `9.75`. The rule: two `int` operands give an `int`; at least one `double` operand gives a `double`. Java decides from the types, not from whether the answer happens to be whole, so `1.5 + 2.5` is `4.0`.
5. **Variables as operands.** Run `TicketVariables`. Java uses the value the variable holds when the expression is evaluated. Evaluating an expression does not change the variable: after the three lines, `tickets` still holds `3`.

MISCONCEPTION Putting the expression inside quotes

Students write `System.out.println("laps * 400");` and expect `1600`. Run `QuotedExpression`: anything in quotes is a string literal, displayed as typed. If you want the value, leave the quotes off.

MISCONCEPTION A whole-number answer is an int

Students say `4 * 2.5` is the `int` `10` because the answer is whole. Point back to `1.5 + 2.5` displaying `4.0`. The operand types decide the result type before any arithmetic happens.

Day 2 direct instruction: integer division

1. **Seventeen cookies, five friends.** Cookies cannot be broken: each friend gets `3`. Predict and run `CookieShare`: `3`, `3`, `0`, `3.4`, `3.4`.
2. **The rule** (1.3.C.3). When both operands are `int`, the result is only the integer part of the quotient. `19 / 5` is `3.8` in math and `3` in Java: no rounding. When the first `int` is smaller than the second, the result is `0`. When at least one operand is a `double`, the result is the full quotient as a `double`, and it does not matter which side the `double` is on.
3. **Same numbers, different types** (slide 15). `AverageTwo`: `total / 2` gives `87`, `total / 2.0` gives `87.5`. `HalfCup`: `1 / 2` gives `0`, a logic error that compiles and runs. The fix in this topic is to write one operand as a `double` literal, `1.0 / 2` (`HalfCupFixed`); other ways of converting types are Topic 1.5.

MISCONCEPTION Integer division rounds

Students expect `19 / 5` to be `4`. Have them write the full quotient first (`3.8`), then cross out everything after the decimal point. Integer division keeps the whole-number part; it never rounds up.

MISCONCEPTION 1 / 2 is one half

Students write `1 / 2` for a half and get `0`, with no error message to warn them. Teach the habit: whenever you see `/`, check the two operand types before working out the value.

Day 2 direct instruction: the remainder operator

1. **What is left over** (1.3.C.4). Run `CookieRemainder`: $17 / 5$ is 3 and $17 \% 5$ is 2. $a \% b$ is what is left after taking as many whole groups of b out of a as possible. Write the check on the board: $17 = 5 \times 3 + 2$. $/$ counts the whole groups; $\%$ counts what is left.
2. **Hours and minutes** (slide 14). A 135-minute movie: $135 / 60$ is 2, $135 \% 60$ is 15, and $135 / 60.0$ is 2.25. Check it: $135 = 60 \times 2 + 15$.
3. **Patterns**. Run `RemainderCases` and fill in the table together: $20 \% 5$ is 0 (divides evenly), $3 \% 5$ is 3 (a smaller first number is all left over), $1234 \% 10$ is 4 (the last digit), $125 / 60$ and $125 \% 60$ are hours and minutes, and $\% 2$ is 0 for even numbers and 1 for odd ones.
4. **Scope**. In this course the first operand is never negative and the second is always positive.

MISCONCEPTION % means percent

Students read $50 \% 10$ as "50 percent of 10" and expect 5. Run `PercentMistake`: it displays 0, because 10 goes into 50 five times with nothing left. Java has no percent operator; 50 percent of 10 is $0.5 * 10$.

MISCONCEPTION A smaller first number gives a remainder of 0

Students confuse $8 / 12$ (0 whole groups) with $8 \% 12$ (all 8 left over). Ask: "How many whole twelves fit in 8? None. So how much is left?"

Day 2 guided practice: whiteboard rounds

Show one expression at a time. Students write the value **and** the type on a mini whiteboard, then hold it up on your signal. Discuss any split before moving on. Start with the six expressions on slide 16 ($47 / 10$, $47 \% 10$, $47 / 10.0$, $6 \% 8$, $6 / 8$, $10 \% 5$), then continue with the table below as time allows.

Expression	Value and type
$9 / 2$	4, int
$9 / 2.0$	4.5, double
$9 \% 2$	1, int
$2 / 9$	0, int
$2 \% 9$	2, int
$4.5 * 2$	9.0, double
$100 \% 10$	0, int
" $9 / 2$ "	not evaluated; displays as text

Finish with a word problem: 70 students ride buses that hold 30 each. Which expression gives the number of full buses, and which gives the students left over? ($70 / 30$ is 2; $70 \% 30$ is 10.)

TEACHER NOTE

If you have no mini whiteboards, students can hold up scrap paper or write in their notes and show a partner. Keep each round to about a minute; the point is quick feedback on the type rule and on `/` versus `%`.

Day 2 independent practice

Puzzle worksheet questions 1, 3, and 7 (values of expressions, sticker-sharing code tiles, and the divide-versus-remainder face-off) are homework. The last row of question 1, `2 + 6 / 2`, previews Day 3's precedence rules; tell students to try it and check it on Day 3.

Day 2 closure

Return to the warm-up with slide 22. Students answer it in Java (`50 / 12` is 4 full cartons, `50 % 12` is 2 eggs left over), then finish the sentence starter: "`7 / 2` is 3 because both operands are `int`, so Java keeps only the integer part." Look for "the types of the two operands," not "whether the answer is whole."

Day 3 warm-up: 14 or 20?

Write `2 + 3 * 4` on the board. Students vote: 14 or 20? Ask one student from each side to explain. A program cannot be ambiguous, so Java needs a fixed rule. Run the first line of `PrecedenceSteps` to settle it: 14. Then check the last row of worksheet question 1 (`2 + 6 / 2` is 5).

Day 3 direct instruction: operator precedence and parentheses

1. **Compound expressions and precedence** (1.3.C.5, slide 18). An expression with more than one operator is a **compound expression**. **Operator precedence** decides which operators are grouped with their operands first: parentheses first, then `*`, `/`, and `%` (one level), then `+` and `-` (a lower level). Operators with the same precedence go **left to right**.
2. **When it happens**. The compiler groups the values with operators when the program is compiled, as if adding invisible parentheses: `2 + 3 * 4` means `2 + (3 * 4)`. The calculation happens when the program runs.
3. **Predict: PrecedenceSteps**. Work the five lines step by step on the board: 14, 20, 10, 12, 6. Slide 18 shows the first four with their first and second steps. Stress `20 - 6 - 4` is 10, not 18; in `24 / 4 * 2`, `*` does not go before `/`; they share a level, so the left one goes first.
4. **Parentheses change the grouping**. Run `AverageParens`: `quiz1 + quiz2 / 2` is 130, higher than either quiz. `(quiz1 + quiz2) / 2` is 87, and `/ 2.0` keeps the `.5`: 87.5. Show `PerPerson` (31) and `PerPersonFixed` (9) as a second example.

5. **The type is decided one operation at a time.** Predict and run `MixedOrder: 1 / 2 * 4.0` is `0.0`, because `1 / 2` was already integer division by the time Java reached `4.0`. `1.0 / 2 * 4` and `4.0 * 1 / 2` are both `2.0`.
6. **Precedence check** (slide 20). Students find each value alone and say which operation happens first: `10 - 2 * 3` is 4, `(10 - 2) * 3` is 24, `18 / 3 / 2` is 3, `7 + 8 % 3` is 9.

MISCONCEPTION Always left to right

Students who learned to read left to right evaluate `2 + 3 * 4` as 20. Have them circle the operator that goes first and rewrite the expression with the invisible parentheses before they calculate.

MISCONCEPTION Multiplication before division, addition before subtraction

Students who learned a memory aid for the order of operations do `*` before `/` and `+` before `-`. In Java (and in math), `*`, `/`, and `%` share one level and `+` and `-` share another; ties go left to right. `24 / 4 * 2` is 12, not 3.

MISCONCEPTION Forgetting parentheses when a sum should be divided

`quiz1 + quiz2 / 2` divides only `quiz2`. Teach the sanity check: an average can never be bigger than the largest value, and a share of a bill can never be bigger than the bill. If it is, look for a missing pair of parentheses.

MISCONCEPTION A double later in the expression fixes the type

Students expect `1 / 2 * 4.0` to be `2.0` because there is a `double` in it. Each operation's type depends only on its own two operands, and `1 / 2` happens first. Have students label the type of every intermediate result.

Day 3 direct instruction: dividing an int by zero

1. **Predict:** `TeamSplit.java` (1.3.C.6; slide 19 shows the same pattern with 30 slices and 0 guests). `snacks` holds 24 and `teams` holds 0. Ask: "Does it compile? What is displayed?" Run it: `Splitting snacks...`, then `java.lang.ArithmeticException: / by zero` naming line 8. `Done!` never appears.
2. **Why it compiles.** `snacks / teams` follows every rule of Java. The compiler checks how the code is written, not what the values will be; the divisor is only known to be zero when the line runs. This is the run-time error students saw in Topic 1.1, now explained: no `int` is the answer to $24 \div 0$, so Java throws an `ArithmeticException` instead of making up a number.
3. **Only the divisor matters.** Run `ZeroDivisor: 0 / 24` is 0 (fine), and `24 / (3 - 3)` throws, because the parentheses produce a zero divisor. `Finished` never appears.
4. **Scope.** The course covers dividing by zero only when both values are `int`. Dividing by zero when a `double` is involved behaves differently and is not tested.

MISCONCEPTION It compiled, so the division is safe

The compiler never works out whether a divisor is zero. Give students the question to ask at every `/`: "Could the value after the slash be zero when this line runs?" If it is, expect an `ArithmeticException` at that line, with earlier output already displayed and nothing after it.

MISCONCEPTION Any zero in a division causes the exception

Students expect `0 / 24` to throw. Zero is only a problem as the second operand. `24` goes into `0` zero times, so the answer is `0`.

Day 3 guided practice: predict, then break PizzaParty

Students type `PizzaParty.java` (or open the copy you made) and write their predicted output **before** running it. Run it and check: `*** "Pizza Party" Planner ***`, `Total slices: 24`, `Slices each: 4`, `Left over: 4`, `Total cost: $37.5`, `Cost each: $7.5`, an empty line, `Thanks for planning!`.

Then, one at a time, students make each change, predict, run, record, and undo:

1. On line 14, add parentheses: `pizzas * (slicesPerPizza / friends)`.
2. On line 7, change `5` to `0`.
3. On line 21, change `"\nThanks for planning!"` to `"\\nThanks for planning!"`.
4. On line 18, change `println` to `print`.

For each, students record what was displayed and which rule explains it. Debrief: (1) `Slices each: 3`, because `8 / 5` is now done first as integer division (1), then `3 * 1`. (2) The first two lines appear, then `Slices each:` and an `ArithmeticException` at line 14; nothing after it runs. (3) The last line becomes `\nThanks for planning!` with no empty line before it, because `\\` is a backslash and the `n` is an ordinary letter. (4) `Total cost: $37.5Cost each: $7.5` on one line, because `print` leaves the cursor after `37.5`.

TEACHER NOTE

In change 2, the exception message may appear on the same console line as `Slices each:`, because `print` left the cursor there. That is expected; point out that the label came from line 13, which ran before line 14 failed. If computers are not available, demonstrate the four changes yourself after students write predictions.

Day 3 independent practice

Puzzle worksheet questions 4, 5, 9, and 10 (be the compiler, the quiz-average snippet pool, two truths and a lie about `1 / 2 * 4.0`, and who am I) are homework. Question 9 settles the Think First prediction from question 2.

Day 3 closure and exit ticket

Show slide 21 (vocabulary) while students settle. Give the exit ticket (about five minutes). Question 1 checks output, escape sequences, and string literals (1.3.A, 1.3.B), question 2 precedence and the result type (1.3.C.2, 1.3.C.5), and question 3 dividing by zero (1.3.C.6). Re-teach the weakest one at the start of Topic 1.4.