

1.4 Assignment Statements and Input

Lesson Plan

Pacing: 2 class periods of 45 minutes

Learning objectives: 1.4.A, 1.4.B

Essential knowledge: 1.4.A.1, 1.4.A.2, 1.4.A.3, 1.4.B.1

Materials

- Projector and the Store It and Scan It slide deck
- Whiteboard and markers
- Scrap paper or mini whiteboards for trace tables
- IDE on every computer (Day 2)
- Printed Store It and Scan It puzzle worksheet (answer key for you) and exit ticket

Agenda

Minutes	Activity
5	Day 1 warm-up — is <code>points = points + 5</code> true?
8	Day 1 direct instruction — the assignment operator, initialize then change (slides 3–5)
10	Day 1 direct instruction — the right side first, a copy and not a formula (slides 6–8)
12	Day 1 direct instruction — the type of an expression, assign before use (slides 10–11)
7	Day 1 guided practice — the swap challenge
3	Day 1 closure — what does <code>=</code> really do? (slide 20)
5	Day 2 warm-up — a nickname that arrives later
8	Day 2 direct instruction — reference variables and <code>null</code> (slides 12–13)
8	Day 2 direct instruction — forms of input and setting up a <code>Scanner</code> (slides 14–16)
9	Day 2 direct instruction — <code>nextInt</code> , <code>nextDouble</code> , <code>nextLine</code> , in order (slides 17–18)
10	Day 2 guided practice — write <code>CarWash</code> in the IDE, then break it
5	Day 2 closure — exit ticket

Overview

Day 1 is assignment (1.4.A). Students use `=` to initialize a variable and to change it, learn that Java evaluates the whole right side to one value before storing it (1.4.A.2), find the type of that value and check that the variable can hold it (1.4.A.1, 1.4.A.3), and see that a variable must be assigned before it is used. Day 2 starts with reference variables and `null` (the rest of 1.4.A.1), then moves to input (1.4.B.1): the forms input can take, and reading text with a `Scanner` using `nextInt`, `nextDouble`, and `nextLine`.

Students have declared `int`, `double`, and `boolean` variables (Topic 1.2), displayed values with `print` and `println`, and evaluated arithmetic expressions, including integer division and `%` (Topic 1.3). 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.
- Every stored value has the same type as its variable. Do not store an `int` value in a `double` variable or convert between types; that is Topic 1.5. Do not use `+=` or `++` (Topic 1.6).
- `String` appears only as a type for text, assigned a string literal or `null`. Do not call `String` methods.
- `new Scanner(...)` and `input.nextInt()` are used as patterns. Say once that creating objects is Topic 1.13 and calls like `input.nextInt()` are Topic 1.14. Use only `nextInt`, `nextDouble`, and `nextLine`.
- Two CED exclusions: assignment inside an expression (such as `a = b = 4;`) is not tested, so do not show it, and no specific form of user input is tested, so `Scanner` is a classroom tool, not exam content.

TEACHER NOTE

The slides are the Store It and Scan It deck (20 slides, with speaker notes); slide numbers below refer to it. Day 1 uses slides 1–11 and closes with slide 20; Day 2 uses slides 12–19. The deck names its `Scanner` variable `console` and the topic's examples name it `input`; tell students the name is the programmer's choice. The Store It and Scan It puzzle worksheet (10 questions, about 30 minutes) is split across the two nights: questions 1, 2, 4, 5, 7, and 10 are about assignment and go home after Day 1; questions 3, 6, 8, and 9 use `null` or a `Scanner` and go home after Day 2.

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:** `PointsUpdate`, `TicketCount`, `PlayerLives`, `ReversedAssignment`, `CopyValue`, `TotalNotUpdated`, `TotalUpdated`, `QuizAverage`, `TypeFromOperations`, `LossyAverage`, `BooleanIntoInt`, `StepGoal`, `Uninitialized`, `ScoreTrace`, and `SwapFixed` (the swap challenge's solution; do not show it until pairs have tried).
- **Day 2 demonstrations:** `NicknameLater`, `NullInt`, `QuotedNull`, `UnassignedString`, `MottoNull`, `AgeFromText.java`, `NoImport.java`, `SnackOrder.java`, `OrderReaderLines.java`, `WrongInput.java`, `WrongInputFixed.java`, and `CarWash.java` (the model solution for the guided practice; do not show it until students have tried).
- **Printed only (exit ticket):** `ExitCopy` and `ArcadeTokens.java`.

Every program in this topic that reads input gets it from a string, as in `new Scanner("16")`, so it has one exact output to trace. To read the keyboard instead, replace the string with `System.in`. Before Day 2, check that a keyboard program runs in your IDE and that students can type into its console.

Day 1 warm-up: `is points = points + 5 true?`

Write `points = points + 5` on the board. Ask: "In math, is this ever true?" (No number equals itself plus 5.) Then show `PointsUpdate` and ask students to write what Java displays. Take a few answers, run it (15), and leave the question open: "Why is that legal in Java? By the end of today you will be able to explain it."

TEACHER NOTE

Slide 2 (a scoreboard where Home gets its score plus 2) is an unplugged version of the same warm-up. Use it instead of, or just before, `PointsUpdate` if you want students to talk before they see code.

Day 1 direct instruction: the assignment operator

1. **Goals** (slide 3). Goals 1–3 are today's; goals 4 and 5 (`null` and the `Scanner`) are Day 2's.
2. **Assignment** (1.4.A.2, slide 5). An **assignment statement** stores a value in a variable. It uses the **assignment operator** `=`: the variable on the left, the value on the right, a semicolon at the end. Read `tickets = 3`; as "tickets gets 3," not "tickets equals 3." In math `=` states a fact; in Java it is an action that happens once, when the line runs.
3. **Initialize, then change** (1.4.A.1, 1.4.A.2, slide 5). The first assignment **initializes** the variable. Every later assignment changes its value: the new value replaces the old one, which is gone. Slide 5 shows `tickets` going from 8 to 5. Students then predict `TicketCount`, and trace it with you (3, then 8). The same `println` displays different values because each one displays what the variable holds at that moment.
4. **The type goes only in the declaration.** Show `PlayerLives: int lives = 3`; declares and initializes in one statement; `lives = 2`; has no type, because `lives` already exists.

MISCONCEPTION Writing the assignment backward

Because `=` looks like the math equals sign, students write `10 = score`; . Run `ReversedAssignment`: unexpected type (required: variable, found: value). The left side must name a storage place; `10` is a value with nowhere to store anything. Have students read every assignment aloud as "the left gets the right."

Day 1 direct instruction: the right side first

1. **Two steps, always** (1.4.A.2, 1.4.A.3, slide 6). An **expression** is code that produces one value. Java (1) evaluates the whole right side using the values variables hold right now, then (2) stores that one value on the left. Walk through slide 6 (`steps = steps + 450`; look up 300, evaluate 750, store 750), then go back to the warm-up and do the same on the board: evaluate `points + 5` with `points` still 10, get 15, then store 15. That is why a variable can appear on both sides.
2. **A copy, not a link** (slide 7). Students predict slide 7 (`doubled` stays 8 after `base` changes), then predict `CopyValue` and `TotalNotUpdated`. Run both. `tuesday = monday`; copies the value 12; changing `monday` later leaves `tuesday` alone. `total = price * quantity`; stores 12 once; changing `quantity` on the next line does not recompute it.

3. **The fix is to run the assignment again.** Show `TotalUpdated`: assigning `total` again after the change displays 20.
4. **Trace it** (slide 8). Students fill in the trace table alone on scrap paper, then compare with a partner: after the four lines `x` is 10 and `y` is 2, so the output is 10, then 2.

MISCONCEPTION An assignment updates itself like a spreadsheet formula

Students who use spreadsheets expect `total` to change when `quantity` changes. Java stores a number, not a formula. Nothing is illegal, so there is no error message: this is a logic error (Topic 1.1). Ask, for every variable in a trace: "Which line last stored a value in it? Has that line run again since?"

MISCONCEPTION Copying links two variables

After `int tuesday = monday`; students expect the two to change together. Draw two separate boxes: line 2 put a copy of 12 in the second box, and the boxes are not connected. `CopyValue` displays 20 and 12.

Day 1 direct instruction: the type of an expression, assign before use

1. **Every expression has a type** (1.4.A.3, slide 10). Slide 10 lists five expressions with their values and types; cover the last two columns and have students fill them in. Then students predict the type and value of each right side in `QuizAverage`: `85 + 90 + 78` is the `int` 253, `total / 3.0` is the `double` 84.33333333333333, and `total / 3` is the `int` 84. Java applies the Topic 1.3 rules one operation at a time; the last operation's result decides the type. `TypeFromOperations` (`10 / 4 * 2.0`) gives the `double` 4.0, even though 4 is whole.
2. **Compatible types** (1.4.A.1, slide 10). The value must come from a type the variable accepts: `int` values for `int` variables, `double` values for `double` variables, `true` or `false` for `boolean` variables. Show `LossyAverage`: `int average = total / 3.0`; does not compile (possible lossy conversion from `double` to `int`), because an `int` has no place to keep the decimal part. `BooleanIntoInt` shows the same error family for `int count = true`; . Topic 1.5 shows a few cases where Java converts a value for you; today, match the types.
3. **Assign before you use** (1.4.A.1, slide 11). Slide 11 shows the error and its fix with `lives`. Then show `StepGoal` and `Uninitialized` side by side. Both declare a variable with no value. `StepGoal` assigns `steps` before line 3 uses it and displays 5000. `Uninitialized` uses `bonus` in `50 + bonus` before any value is stored, and the compiler reports variable `bonus` might not have been initialized. The error points at the line that **uses** the variable; look upward from there for the missing assignment.

MISCONCEPTION A whole-number value is an `int`

Students think `3.0 * 4` can go in an `int` because 12.0 has nothing after the decimal point. The compiler checks the type, not the particular value: any expression with a `double` operand in its last operation is a `double`. Have students label the type of every intermediate result.

MISCONCEPTION A declaration gives the variable a default value

Students expect `int bonus;` to hold 0. It holds nothing usable, and Java refuses to compile code that uses it. Ask why that is better than a silent 0: a made-up 0 would give a believable wrong answer, a logic error, while a compile error forces the programmer to choose the starting value.

Day 1 guided practice: the swap challenge

1. **Predict.** Pairs trace `ScoreTrace` on scrap paper with columns for `first` and `second`. The student who wrote it wanted to swap the values. Output: 10, then 9. The swap failed: `first = second;` replaced 5 with 9, and the 5 was gone before `second = first;` ran.
2. **Fix it.** Pairs write a swap that works, using one extra variable. Call on one pair to write theirs on the board, then trace it with the class. Show `SwapFixed: int saved = first;` keeps the 5 before it is replaced, and the output is 9, then 5.

TEACHER NOTE

If pairs are stuck after two minutes, give the hint: "Before you replace a value, where could you keep a copy of it?" Pairs that finish early can trace `TotalNotUpdated` again and change one line so it displays 20.

Day 1 independent practice

Hand out the puzzle worksheet. Questions 1, 2, 4, 5, 7, and 10 (be the JVM, a trace table, math's `=` versus Java's `=`, the average-jump snippet pool, a one-name swap fix, and a one-sentence wrap-up) are homework, about 18 minutes. Question 1 ends with a Think First prediction that question 2 settles. Students keep the worksheet for Day 2.

Day 1 closure

Show slide 20. Partners explain, step by step, what Java does when it runs `total = total + price;`, using the sentence starter: "First Java evaluates the right side using the values the variables hold now, then it stores the result in `total`." If time allows, students also finish this sentence on a sticky note: "`total = price * quantity;` stores ... , and it is worked out when ..." Look for "one value (a number, not a formula)" and "the moment that line runs, using the values the variables hold then."

Day 2 warm-up: a nickname that arrives later

Show `NicknameLater` in the IDE (not slide 12, which shows the answer) and ask students to write the output. Most expect an error on line 2 or an empty line. Run it: `null`, then `Ace`. Ask: "Line 1 stored something. What was it?"

Day 2 direct instruction: reference variables and null

1. **Reference variables** (1.4.A.1). From Topic 1.2, any type that is not primitive is a reference type. `String`, the type for text, is one. A reference variable does not hold a number or `true/false`; it is connected to an object, such as a piece of text. Objects get their full treatment in Topics 1.12 and 1.13; today, two facts: a reference variable can be assigned an object (a string literal is one), or it can be assigned `null`.
2. `null` (1.4.A.1, slide 12). The literal `null` means the variable is not connected to any object. It is a fixed value written in the code, like `3` or `true`, so it has no quotes. Assigning it counts as initializing, so `NicknameLater` line 2 compiles, and displaying the variable shows the word `null`. Line 3 replaces `null` with `"Ace"`, an ordinary assignment.
3. **Only reference types.** Point to the right side of slide 12 (`double price = null;` does not compile) and run `NullInt: incompatible types: <null> cannot be converted to int`. A primitive variable always holds an actual value, so "no object" makes no sense for it.
4. **Check: compiles or not?** (slide 13). Students decide alone for each of the four snippets, then compare with a partner: (1) compiles and displays `3.0`; (2) compiles and displays `2`, because `pages` is assigned before it is used; (3) does not compile, because `10 / 4.0` is a `double`; (4) does not compile, because `boolean` is primitive and cannot hold `null`.

MISCONCEPTION `null` and "null" are the same

`QuotedNull` and `NicknameLater` both display `null` at first. With quotes it is a string literal, four letters of real text; without quotes it means no object. Ask: "Is there a piece of text here, or is there no object?" Write `null` without quotes when you mean "no object."

MISCONCEPTION `null` means the variable was never assigned

Students treat "holds `null`" and "never assigned" as the same thing. Run `UnassignedString` (variable `motto` might not have been initialized) next to `MottoNull` (displays `null`). `null` is a real value that the program chose to store; a variable that was never assigned has no value and cannot be used.

Day 2 direct instruction: forms of input and setting up a Scanner

1. **Input** (1.4.B.1, slide 15). **Input** is data a program receives from outside itself. Ask students for examples, then sort them into the four forms on the left of slide 15: tactile (touches, presses, movements), audio (sound), visual (images or video), and text (characters). The CED does not test any specific form of input; the course uses text.
2. **The Scanner class** (1.4.B.1, slide 15). A `Scanner` is one way to get text input from the keyboard. Walk through the first two rows of the table on slide 15: `import java.util.Scanner;` at the top

of the file, before the class; `Scanner input` declares a reference variable; `new Scanner(System.in)` creates a `Scanner` that reads the keyboard (`System.in`) and is assigned to `input`. Say once: creating objects with `new` is explained in Topic 1.13; use the line as a pattern for now.

3. **Reading from a string.** Show `AgeFromText.java`. `new Scanner("16")` reads the text `16` exactly as if a user had typed it, so the output is always the same: `Age: 16`.
`int age = input.nextInt();` is an ordinary assignment: the right side produces one `int`, stored in `age`. Calls like `input.nextInt()` are explained in Topic 1.14; read it as "ask `input` for the next whole number." Then replace `"16"` with `System.in`, run it, and type an age. Slide 16 (`DogYears`) is the same pattern as a complete keyboard program with a prompt.

MISCONCEPTION `Scanner` needs no import, like `System`

`System` is always available, so students skip the import. Run `NoImport.java`:
 cannot find symbol (symbol: class `Scanner`), reported once for each place the name appears on line 5. The fix is `import java.util.Scanner;` as the first line of the file. Topic 1.7 explains why some classes need an import.

Day 2 direct instruction: `nextInt`, `nextDouble`, `nextLine`

1. **Three reading calls** (1.4.B.1, the last three rows of slide 15). `nextInt()` reads the next piece as a whole number and produces an `int`; `nextDouble()` reads the next piece as a number that may have a decimal point and produces a `double`; `nextLine()` reads everything to the end of the current line, spaces included, and produces a `String` (without the line break). Picture a marker that shows where the `Scanner` reads next; every read moves it forward. Pieces are separated by spaces or line breaks, and `\n` in the string is a line break (Topic 1.3).
2. **Predict: `SnackOrder.java`.** The input is `Ana Ortiz` on one line and `4 1.25` on the next. Students write the value each line stores, then the output. Trace the marker on the board: `customer` gets `"Ana Ortiz"` (the space is included because `nextLine` stops only at the line break), `quantity` gets `4`, `price` gets `1.25`. Output: `Ana Ortiz`, then `5.0` (an `int` times a `double` is a `double`).
3. **Order matters, spacing does not.** Run `OrderReaderLines.java`: the numbers are spread over extra lines and spaces (`3`, a blank line, then `2.5`), and the reads still work. The output is `Ada Lovelace`, then `Cost: 7.5`. `nextInt()` and `nextDouble()` skip spaces and line breaks to find their piece. The order of the calls must match the order of the input.

MISCONCEPTION `nextInt` rounds a decimal, or the compiler catches bad input

Run `WrongInput.java`: it compiles, displays `Reading the quantity...`, then stops with an `InputMismatchException` when `nextInt()` meets `three`. The compiler checks the code, not the input, so a mismatch shows up only while running. `nextInt()` also refuses `4.5` rather than dropping the decimal part; slide 17 shows the same crash when the second `nextInt()` meets `2.5`. Run `WrongInputFixed.java` with matching input (`3 2.5`) to show the fix. When a real user types the input, a prompt such as `How many?` tells them what to type. Close with slide 18 (same code, three inputs): `6 3` and `3 6` both display `18`, and `6 three` throws an `InputMismatchException` at the second read.

MISCONCEPTION `nextLine` stops at a space

Students expect `nextLine()` to read only `Ana`. The number-reading calls stop at a space; `nextLine()` stops only at the end of the line.

Day 2 guided practice: write `CarWash`, then break it

Students create `CarWash.java` in the IDE and write a program that reads a team name (one line), then the number of cars washed (a whole number) and the price per car (a decimal), and displays the team name and the money raised. They read the keyboard with `new Scanner(System.in)` and add a prompt with `print` before each read (for example `System.out.print("Team name? ");`). They run it and type `Robotics Club`, press Enter, then type `23 8.5`. Expected: `Robotics Club`, then `195.5` (23×8.5).

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

1. If you stored the money raised in a variable, declare it as an `int` instead of a `double`.
2. Run it and type `twenty` for the number of cars.
3. Delete the `import` line.
4. Run it and type `23` and `8.5` on separate lines.

Debrief: (1) does not compile: possible lossy conversion from `double` to `int`, because the number of cars times the price per car is a `double`. (2) compiles and starts, then stops with an `InputMismatchException` at the `nextInt()` line. (3) cannot find symbol for `Scanner`. (4) the same output, because the number-reading calls skip line breaks.

TEACHER NOTE

`CarWash.java` in the examples folder is one correct solution. It reads from the string `"Robotics Club\n23 8.5"` instead of the keyboard so its output is fixed, and it has no prompts; a student version with `System.in` and prompts shows the prompts before the same two output lines. If computers are not available, write the program together on the projector and have students predict each change before you run it.

Day 2 independent practice

Puzzle worksheet questions 3, 6, 8, and 9 (snack-stand code tiles with a `Scanner`, be the compiler, two truths and a lie about `null` and `"null"`, and who am I) are homework, about 12 minutes.

Day 2 closure and exit ticket

Show slide 19 (vocabulary) while students settle. Give the exit ticket (about five minutes). Question 1 checks assignment and evaluation order (1.4.A.2), question 2 compatible types, `null`, and assign before use (1.4.A.1, 1.4.A.3), and question 3 reading input with a `Scanner` (1.4.B.1). Re-teach the weakest one at the start of Topic 1.5.