

2.2 Boolean Expressions

Lesson Plan

Pacing: 2 class periods of 45 minutes

Learning objectives: 2.2.A

Essential knowledge: 2.2.A.1, 2.2.A.2, 2.2.A.3

Materials

- Projector and the True or False? slide deck
- Whiteboard and markers
- Mini whiteboards or scrap paper
- IDE on every computer
- Printed True or False? puzzle worksheet (answer key for you) and exit ticket

Agenda

Minutes	Activity
5	Day 1 warm-up — yes, no, or something else? (slides 2–3)
12	Day 1 direct instruction — == and != (slides 4–6)
13	Day 1 direct instruction — <, >, <=, >= and the boundary (slides 7–9)
10	Day 1 guided practice — overdue books and boundary rounds (slide 10)
5	Day 1 closure — which operator?
5	Day 2 warm-up — is it true?
8	Day 2 direct instruction — arithmetic before comparison (slides 11–13)
8	Day 2 direct instruction — storing and printing booleans (slides 14–16)
11	Day 2 direct instruction — comparing Strings (slides 17–19)
8	Day 2 guided practice — same password? (slide 20)
5	Day 2 closure and exit ticket (slides 21–22)

Overview

Day 1 is relational operators on numbers: == and !=, then <, >, <=, and >=, with attention to values exactly on the boundary (2.2.A.1, 2.2.A.2). Day 2 is evaluating comparisons that contain arithmetic, storing a `boolean` result in a variable and printing it, and comparing `String` objects with `equals` and `compareTo` instead of == (2.2.A.2, 2.2.A.3).

Students know the `boolean` type (Topic 1.2), arithmetic and precedence (Topic 1.3), assignment (Topic 1.4), `double` round-off (Topic 1.5), references and `new` (Topic 1.13), and `equals` and `compareTo` (Topic 1.15). Stay inside the topic boundary. Conditions here are only evaluated and printed; the `if` statement is Topic 2.3. Do not use `&&`, `||`, or `!` (Topic 2.5): when students try to write a range such as `60 < t < 80`, tell them to write each comparison on its own for now.

TEACHER NOTE

The slides are the True or False? deck (22 slides, with speaker notes); slide numbers below refer to it. Day 1 uses slides 1–10 and Day 2 uses slides 11–22. The puzzle worksheet (10 questions, about 35 minutes) is split: questions 1, 2, and 7 after Day 1; questions 3, 4, 5, 6, 8, 9, and 10 after Day 2.

Files ending in `.snippet.java` hold only statements that go inside `main`; paste them into a scratch class to run them. Examples are in the topic's `examples/` folder.

- **Day 1 demonstrations:** `EqualCheck`, `LivesCheck`, `PinWrong`, `AssignNotCompare`, `PinFixed`, `PassingScore`, `SpeedTrace`, `PriceAndTemps`, `ArrowWrong`, `PointsCheck`, `MixedEqual`.
- **Day 2 demonstrations:** `GroupCheck`, `EvenOdd`, `MinutesTrace`, `RoundOffCompare`, `TimesThree`, `RoundOffFixed`, `StoreAnswer`, `LabelWrong`, `LabelFixed`, `SameText`, `OrderCheck`, `NameOrder`, `CaseOrder`, `StringLess`, `ChainWrong`, `ResultWrong`, `TeamWrong`, `TeamFixed`; `GymEntry.java` and `TicketCheck.java` are full programs for extra practice.
- **Exit ticket:** reuses `VotesTrace`, `TwoCopies`, and `PassingScore`. Keep `VotesTrace` and `TwoCopies` out of class so those two questions are new to students; `PassingScore` is shown on Day 1.

Day 1 warm-up: yes, no, or something else?

Show slide 2. Students decide which of the four game questions can only be answered yes or no. Questions 1 and 3 can; "How many lives are left?" and "What is the player's name?" have many possible answers. A program can use a yes-or-no question to decide what to do next. Show the goals on slide 3 and name today's vocabulary: a **relational operator** compares two values, and a **Boolean expression** evaluates to `true` or `false`.

Day 1 direct instruction: `==` and `!=`

1. **A comparison answers true or false** (2.2.A.1, slide 5). `a == b` is "a is equal to b"; `a != b` is "a is not equal to b". Run `EqualCheck`: `true`, `false`, then after `guess = 4`, `false`, `true`. The two operators always give opposite answers for the same values.
2. **Predict: `LivesCheck`**. `lives == maxLives` is `false`, `lives != maxLives` is `true`; after `lives = 5` the first is `true`. The last lines store the result of `!=` in a variable.
3. **One = stores, two == ask** (slide 6). Show `PinWrong`: `boolean match = entered == pin`; does not compile (incompatible types: `int` cannot be converted to `boolean`) because `entered == pin` stores and gives back an `int`. Show `PinFixed` (`entered == pin`, which prints `false`). `AssignNotCompare` is a second example.
4. **Mixed types and booleans**. `MixedEqual`: `3 == 3.0` is `true` because the `int` is widened for the comparison; `==` and `!=` also work on two `boolean` values.

MISCONCEPTION One equals sign asks if two things are equal

In math, `=` means "is equal to". In Java it means "store". Have students read `=` aloud as "gets" and `==` as "is equal to" every time.

MISCONCEPTION `==` and `!=` can give the same answer

For any two values exactly one of them is `true`. If a student writes both as `true`, ask them to substitute the values from the variables.

Day 1 direct instruction: the size operators and the boundary

1. **Four size operators** (2.2.A.1, slide 8). `<` smaller, `>` larger, `<=` smaller or the same, `>=` larger or the same. Run `PassingScore` with `score 70`: `false`, `true`, `false`, `true`. Only the "or equal" operators are true when the values are equal, so the boundary value 70 is the one to test.
2. **From words to operators** (slide 9). "More than", "over", "above" are `>`; "at least" is `>=`; "less than", "under", "below" are `<`; "at most", "no more than" are `<=`. Ask students whether each phrase counts the boundary.
3. **Negatives and double against int**. Run `PriceAndTemps`: `19.99 < 20` is `true`, `-5 > -12` is `true` (`-5` is farther right), and `-5 <= -12` is `false`.
4. **Write it in the right order**. Show `ArrowWrong`: `=>` does not compile. Java has `>=` and `<=` only, with no space between the characters.

MISCONCEPTION Typing `=>` or `=<` or `>=`

People say "equal or greater" but Java wants the angle bracket first: `>=`, `<=`. Show `ArrowWrong`; the compiler reports `illegal start of expression`.

MISCONCEPTION At least and more than mean the same thing

"At least 14" includes 14 (`>= 14`); "more than 14" does not (`> 14`). Use slide 10's library fine: a book 14 days late is not "more than 14 days late".

MISCONCEPTION Bigger absolute value means bigger number

Students say `-12 > -5` is `true` because 12 is bigger than 5. Draw a number line: `-5` is farther right, so `-5 > -12`.

Day 1 guided practice: overdue books and boundary rounds

1. **Slide 10**. With `daysLate = 14`, students write the value of `daysLate > 14` (`false`), `daysLate >= 14` (`true`), `daysLate != 14` (`false`), and `-3 > -10` (`true`). Which expression is the right test for a fine on books "more than 14 days late"? `daysLate > 14`, because 14 days exactly is not more than 14.

2. **Whiteboard rounds.** Show one expression at a time; students write `true` or `false` on a mini whiteboard and hold it up. Use `PointsCheck` (`points = 50: true, true, true, true`) first, then these with `int a = 9; int b = 4;`

Expression	Value
<code>a < b</code>	<code>false</code>
<code>a >= 9</code>	<code>true</code>
<code>b != 4</code>	<code>false</code>
<code>a - b <= 5</code>	<code>true</code>
<code>-a < -b</code>	<code>true</code>

TEACHER NOTE

If you have no mini whiteboards, students can write answers on scrap paper and show a partner. Keep each round to about a minute. Have students say the boundary value out loud before answering.

Day 1 independent practice

Puzzle worksheet questions 1, 2, and 7 (be the JVM with a stored comparison, evaluate seven Boolean expressions, and a one-token fix to a boundary) are homework. Question 1 previews Day 2's idea that a `boolean` variable keeps its value; question 2's last row, `"cat".compareTo("dog") < 0`, previews Day 2's String work.

Day 1 closure

Ask: "A coupon works when the total is at most \$25. Which operator, `<` or `<=`?" Students answer on a sticky note: `<=`, because exactly \$25 must work.

Day 2 warm-up: is it true?

Write `17 % 4 == 0` on the board. Students write `true` or `false` and say which operation Java does first. Take two answers, then say today's idea: the arithmetic happens before the comparison.

Day 2 direct instruction: arithmetic before comparison

1. **Arithmetic first, then compare** (2.2.A.2, slide 12). Run `GroupCheck`: `14 % 4 == 0` is `false` (remainder 2), `14 / 4 >= 3` is `true` (integer division gives 3), and `12.5 + 8.25` is 20.75, so `total * 2 > 40` is `true`. The whole expression is one `boolean`.
2. **Even and odd.** `EvenOdd: n % 2 == 0` is `true` exactly when `n` is even. Add `MinutesTrace` for one more round of `/`, `%`, and `-` inside comparisons.

3. **== on double results** (slide 13). Run `RoundOffCompare`: `0.1 + 0.2` prints `0.30000000000000004`, so `sum == 0.3` is false. `TimesThree` shows the same for `0.1 * 3 == 0.3`. `RoundOffFixed`: `Math.abs(sum - 0.3) < 0.000001` is true, and `cents == 30` with `int` values is exact.

MISCONCEPTION == is a good way to compare two double results

`double` values can carry a tiny round-off error, so two numbers that should match in math may not match in Java. Compare very close values with `Math.abs(x - y) < tolerance`, or keep exact amounts, such as cents, in an `int`.

MISCONCEPTION Writing a range the way math does

`60 < temperature < 80` does not compile (`ChainWrong`): Java works out `60 < temperature`, gets a `boolean`, and cannot compare it with `80`. Tell students each comparison stands alone for now.

Day 2 direct instruction: storing and printing booleans

1. **A variable holds the answer, not the question** (slide 15). `boolean low = battery < 20;` evaluates the comparison once, when it is assigned. Run `StoreAnswer`: `isTeen` is false for age 12, stays false after `age = 14`, and only becomes true when assigned again.
2. **Labels need parentheses** (slide 16). `LabelWrong` (`"Even: " + count % 2 == 0`) does not compile because `+` happens before `==` and a `String` cannot be compared to an `int`. `LabelFixed` prints `Even: true` with `"Even: " + (count % 2 == 0)`.
3. **The type must match**. `ResultWrong` (`String passed = score >= 70;`) gives incompatible types: `boolean` cannot be converted to `String`; `ResultFixed` stores it in a `boolean`.

MISCONCEPTION A boolean variable updates when its values change

Students think `low` becomes true once `battery` drops below 20. The comparison was evaluated once, on the line where `low` was assigned. Run `StoreAnswer` and make students trace the variable line by line.

MISCONCEPTION No parentheses around a comparison in a label

`+` happens before any relational operator, so `"Low: " + battery < 20` does not compile. Wrap the comparison: `"Low: " + (battery < 20)`.

Day 2 direct instruction: comparing Strings

1. **== compares references** (2.2.A.3, slide 18). Run `SameText`: two new `String("Tiger42")` objects. `equals` is true, `==` is false, `!=` is true. `==` asks whether both variables refer to the very same object; `equals` compares the characters. Draw two boxes on the board with the same text.

2. **Alphabetical order: compareTo** (slide 19). Run `OrderCheck` and `NameOrder`. `x.compareTo(y)` is negative when `x` comes first, zero for the same text, and positive when `x` comes after; compare the result with `0`. `x < y` does not compile (`StringLess`) because the size operators work only on numbers.
3. **Case matters.** `CaseOrder`: uppercase letters come before lowercase ones, so `"Zoo".compareTo("apple") < 0` is `true`, and `"Zoo".equals("zoo")` is `false`.
4. **A real bug.** `TeamWrong` reads two words that are both `blue` with `next()`; `==` prints `false`. `TeamFixed` uses `equals` and prints `true`. The code compiled, so only the wrong output shows the bug.

MISCONCEPTION `==` is fine for comparing Strings

It sometimes appears to work, which makes it worse. `==` compares references and not characters, so two Strings with the same text can still give `false`. Always use `equals` for text.

MISCONCEPTION Using `<` directly on Strings

`"apple" < "banana"` does not compile (`StringLess`). Use `compareTo` and compare its result with `0`.

Day 2 guided practice: same password?

Slide 20. Students write the exact output of the `saved`, `typed`, `match` code on their own before comparing with a partner: `true`, `false`, `false`. `match` was evaluated before `typed` was reassigned, so it stays `true`; `saved == typed` compares references; `saved.equals(typed)` compares `"tiger"` with `"Tiger"`, and case matters. Ask: "Which line uses the wrong tool?" (the `==` line).

Then, with `int guess = 8; int secret = 8;` on the board, students write three Boolean expressions that evaluate to `true`, three that evaluate to `false`, and one that compares two `String` objects the right way.

Day 2 independent practice

Puzzle worksheet questions 3, 4, 5, 6, 8, 9, and 10 are homework: rebuild a program from code tiles (question 3), fill the snippet pool (question 4), the `==` versus `equals` face-off (question 5), two truths and a lie about round-off (question 6), be the compiler (question 8), and the reflection (question 10).

Day 2 closure and exit ticket

Show slide 22; partners finish the sentence starter. Give the exit ticket (about five minutes). Question 1 checks storing and printing a comparison (2.2.A.2), question 2 checks `==` versus `equals` on Strings (2.2.A.3), and question 3 checks reading a size comparison on its boundary

(2.2.A.1). Re-teach the weakest one at the start of Topic 2.3. Slide 21 (vocabulary) is a reference for students.