

1.11 Math Class

Lesson Plan

Pacing: 2 class periods of 45 minutes

Learning objectives: 1.11.A

Essential knowledge: 1.11.A.1, 1.11.A.2, 1.11.A.3

Materials

- Projector
- Whiteboard and markers
- Scrap paper or mini whiteboards for predictions
- IDE on every computer
- Math Class Puzzles slide deck
- Printed Math Class Puzzles worksheet and exit ticket

Agenda

Minutes	Activity
5	Day 1 warm-up — which tool for which job? (slides 2–3)
7	Day 1 direct instruction — <code>java.lang</code> and a class of only class methods (slides 4–5)
8	Day 1 direct instruction — absolute value and its two versions (slide 6)
10	Day 1 direct instruction — powers, square roots, and nested calls (slides 7–9)
8	Day 1 guided practice — write the expression
5	Day 1 independent practice — puzzle worksheet questions 1–2
2	Day 1 closure — value and type of a Math call
5	Day 2 warm-up — predict three nested calls
7	Day 2 direct instruction — <code>Math.random</code> , endpoints, and stand-ins (slides 10–11)
10	Day 2 direct instruction — random int values in a range (slides 12–15)
8	Day 2 direct instruction — random double values in a range (slide 16)
8	Day 2 guided practice — Math in a program, then write the range expression (slides 17–19)
2	Day 2 closure — count, multiply, cast, add (slides 20–21)
5	Day 2 exit ticket

Overview

Two periods on one learning objective: write expressions that call the `Math` class's methods and determine the value they produce (1.11.A). Day 1 covers why `Math` needs no import line and is called through its class name (1.11.A.1, 1.11.A.2), then `abs`, `pow`, and `sqrt`, including nested calls (1.11.A.2). Day 2 covers `Math.random()` and how arithmetic and casting turn its result into a random `double` or `int` in a stated range, with each endpoint inclusive or exclusive (1.11.A.2, 1.11.A.3).

The CED's suggested skills are 2.C (write code that uses procedural abstractions) and 4.A (describe the behavior of code). Both days ask students to write expressions and to describe exactly which values an expression can produce.

Stay inside the topic boundary:

- The five `Math` headers in the CED are the whole inventory: `abs(int)`, `abs(double)`, `pow`, `sqrt`, and `random`. Do not show other `Math` methods.
- Students do not write `if` statements or loops (Unit 2). Random programs cannot check their own results, so students check them with **stand-ins**: a value the call could return, such as `0.0` or `0.999`, substituted in place of the call. "Stand-in" is the study guide's name for the technique, not College Board vocabulary.
- A label and a number still take a `print` followed by a `println`; joining text and numbers is Topic 1.15.
- Headers in the `Math` documentation start with `public static`. `static` was taught in Topic 1.10; `public` is explained in a later topic and can be read past.

TEACHER NOTE

Every program below is in the topic's `examples/` folder. 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.

- **Day 1, the `Math` class:** `SquareTiles`, `MissingClassName`, and `LowercaseMath`.
- **Day 1, absolute value:** `GuessDistance`, `AbsTypes`, `AbsDiscarded`, and `AbsStored`.
- **Day 1, powers and square roots:** `PowOrder`, `PowToInt`, `PowFixed`, `Ladder`, `LadderWrong`, and `SqrtRound`.
- **Day 1 guided practice:** `MathPractice` (the answers to "write the expression").
- **Day 2 warm-up:** `WarmUpNested`.
- **Day 2, `Math.random`:** `RandomSame` and `RandomArgument`.
- **Day 2, random double values:** `TempRange`, `SignedRange`, and `PrecedenceWrong`.
- **Day 2, random int values:** `DieRoll`, `RangeCheck`, and `CastTooEarly`.
- **Day 2 guided practice:** `RangePractice` (the answers, checked with stand-ins).
- **Printed only (exit ticket):** `ExitMath` and `ExitRangeCheck`. The puzzle worksheet and its key are separate Word files; their code is printed on the page.

The Math Class Puzzles deck covers both days: slides 1–9 on Day 1 and slides 10–21 on Day 2. Its speaker notes hold the answers to every Pause and Try slide. Compiler messages are quoted from a recent JDK; the wording can vary slightly between Java versions. Before class, open `SquareTiles` and a scratch class in the IDE next to the deck.

Day 1 warm-up: which tool for which job?

Show slide 2. Give pairs two minutes with the Java Quick Reference to pick one `Math` method for each of four jobs: how far apart 4 and 19 are (`Math.abs(4 - 19)`), the side of a square with area 64 (`Math.sqrt(64)`), 2 to the 10th power (`Math.pow(2, 10)`), and a coin flip (`Math.random`). Ask: "How would you find the side of the square using only `+`, `-`, `*`, `/`, and `%`?" Pairs can check guesses (`8 * 8`) but cannot compute the answer from 64; Java already has that algorithm written. Do not work out the coin flip yet; it is Day 2. Read the goals on slide 3.

TEACHER NOTE

Keep this to five minutes. The point is to feel the gap the `Math` class fills, not to invent a square-root algorithm. If a student suggests trying numbers until one works, agree that it is a real algorithm, and point out that it is a whole program for one small question.

Day 1 direct instruction: `java.lang` and a class of only class methods

Start Part 1 (slide 4).

1. **No import line** (1.11.A.1, slide 5). In Topic 1.4, `Scanner` needed `import java.util.Scanner;`. `Math` needs nothing. It is in the package `java.lang`, and every class in `java.lang` is **available by default**: visible to every program without an import. `System` is in `java.lang` too, which is why `System.out.println` has never needed an import.
2. **Only class methods** (1.11.A.2). Every `Math` method has `static` in its header, so every one is called as `Math.methodName(arguments)`. Nothing has to be set up first.
3. **The inventory** (slide 5). Show the five headers from the CED, which are also the `Math` methods on the Java Quick Reference students have during the AP exam: `int abs(int x)`, `double abs(double x)`, `double pow(double base, double exponent)`, `double sqrt(double x)`, and `double random()`. Today covers the first four; Day 2 covers `random`.
4. **Run `SquareTiles`**. Point out the three things: no import line, the call starts with `Math.`, and the returned `12.0` is stored in `side` and then used in `4 * side`, which prints `48.0`.
5. **Predict: `MissingClassName`**. Ask, "Does `sqrt(roomArea)` compile, since `Math` is available by default?" It does not: `cannot find symbol, symbol: method sqrt(double)`. Then show `LowercaseMath: math.sqrt(...)` gives `cannot find symbol, symbol: variable math`, because Java is case-sensitive (Topic 1.1).

MISCONCEPTION Available by default means the class name can be left out

Students hear "available by default" and drop `Math.` from the call. "Available" means the class is visible without an import; the call still needs the class name, because Topic 1.10's rule still holds: the class name is optional only inside the class that defines the method. Run `MissingClassName` and read location: `class MissingClassName` aloud: Java looked in the student's class and found no `sqrt`.

Day 1 direct instruction: absolute value and its two versions

1. **The idea**. A guessing game's target is 50. Guesses of 42 and 61 are off by `-8` and `11`. For "how close," the sign does not matter: the distances are 8 and 11. The **absolute value** is a number's distance from zero, never negative.
2. **Two headers, one name** (1.11.A.2, slide 6). `abs` is overloaded (Topic 1.9): `int abs(int x)` and `double abs(double x)`. Java picks the version whose parameter type matches the argument, so an `int` in gives an `int` out and a `double` in gives a `double` out. Students predict the slide's

three lines before you reveal 15, 0.75, 11; in the third, the argument `9 - 20` is worked out first, so the call is really `abs(-11)`.

3. **Run GuessDistance.** The argument `guess - target` is evaluated first; `abs` receives `-8` and returns `8`, then receives `11` and returns `11`. Ask: "Would `Math.abs(target - guess)` change the answers?" No; the distance is the same either way.
4. **Predict: AbsTypes.** Students write four lines of output: `4`, `4.0`, `3.5`, `6`. The first two differ only in the type of the argument.
5. **Predict: AbsDiscarded.** Most students say `15`. It prints `-15`. `abs` received a copy of `change` (call by value, Topic 1.9) and returned `15`, and nothing stored the return value. `AbsStored` writes `change = Math.abs(change);` and prints `15`.

MISCONCEPTION `Math.abs(x)` makes `x` positive

The call reads like a command, so students expect the variable to change. It compiles with no warning, which makes the mistake hard to find. Every `Math` method returns a new value and leaves its arguments alone. Ask of every call: "Where does the returned value go?" If the answer is "nowhere," the call had no effect the program can see.

Day 1 direct instruction: powers, square roots, and nested calls

1. **pow** (1.11.A.2, slide 7). `double pow(double base, double exponent)` returns `base` raised to `exponent`. The **base** is the number multiplied; the **exponent** is how many times. Run `PowOrder`: `Math.pow(2, 3)` is `8.0` and `Math.pow(3, 2)` is `9.0`. Arguments match parameters by position, so the first is always the base. Java has no power operator; `2 ^ 3` does not compute `8`.
2. **Always a double** (slides 7–8). Slide 7's `Math.pow(10, 3)` prints `1000.0` and `Math.sqrt(144)` prints `12.0`, even though both answers are whole. Slide 8's `int big = Math.pow(10, 3);` does not compile; a cast fixes it. Show `PowToInt` for the same error in the IDE:
`int area = Math.pow(side, 2);` does not compile,
 possible lossy conversion from `double` to `int`, even though `49` is whole. The compiler goes by the return type in the header, not the value. `PowFixed` keeps a `double` (`49.0`) or casts on purpose (`49`).
3. **sqrt.** `double sqrt(double x)` returns the **nonnegative** square root: `Math.sqrt(20.25)` is `4.5`, not `-4.5`. Treat a nonnegative argument as a precondition (Topic 1.8): a negative number has no real square root, and `sqrt` does not check for you. What Java returns for a negative argument is outside the course.
4. **Nested calls: inside out.** Show `Ladder`: a ladder's foot is 6 feet from a wall and its top is 8 feet up, so its length is the square root of $(6^2 + 8^2)$. Build the steps on the board:
`Math.pow(base, 2)` is `36.0`, `Math.pow(height, 2)` is `64.0`, the sum is `100.0`, `Math.sqrt(100.0)` is `10.0`. An argument must have a value before the method that receives it can run.
5. **Predict: LadderWrong.** It takes the square root of each piece and prints `14.0`. $\sqrt{36} + \sqrt{64} = 6 + 8 = 14$, but $\sqrt{(36 + 64)} = 10$. It is a logic error (Topic 1.1); the compiler accepts every call.
6. **Casting a result.** Slide 7's last line, `(int) Math.pow(10, 3)`, prints `1000`. `SqrtRound` prints `6.6332495807108`, `6`, and `7` for the square root of `44`. The second line truncates with `(int)`; the third uses the rounding pattern `(int) (x + 0.5)` from Topic 1.5. A `Math` result is an ordinary `double`.

7. **Pause and try** (slide 9). Students write exactly what each line prints: `19.0 (3 + 16.0)`, `9.0 (Math.sqrt(81))`, and `7` (the cast truncates about 7.07). Ask who wrote 19 or 9, and why.

MISCONCEPTION A whole-number answer can go in an int

Students store `Math.pow(7, 2)` or `Math.sqrt(49)` in an `int` because the answer "is" 49 or 7. The type of the result comes from the header, and `pow` and `sqrt` always return a `double`. Have students point to the return type in the header before they choose the variable's type.

MISCONCEPTION The square root of a sum is the sum of the square roots

Students split `Math.sqrt(a + b)` into `Math.sqrt(a) + Math.sqrt(b)`. Give them a test they can check by hand: legs 6 and 8 must give 10. LadderWrong gives 14, which is just 6 + 8: the ground distance plus the wall height, as if the ladder were bent at a right angle. One `Math.sqrt( ... )` goes around the whole sum.

Day 1 guided practice: write the expression

Write the four tasks below on the board. Pairs write one statement for each task on mini whiteboards, then hold them up:

Task	Statement	Value
Print the distance between the scores 73 and 88.	<code>System.out.println(Math.abs(73 - 88));</code>	15
Store 1.5 to the power 2 in <code>area</code> .	<code>double area = Math.pow(1.5, 2);</code>	2.25
Store the side of a square whose area is 6.25 in <code>side</code> .	<code>double side = Math.sqrt(6.25);</code>	2.5
Store the diagonal of a 5-by-12 rectangle in <code>diagonal</code> .	<code>double diagonal = Math.sqrt(Math.pow(5, 2) + Math.pow(12, 2));</code>	13.0

Run `MathPractice` to check the values. Expect `int` variables for tasks 2 to 4 (they do not compile; send students to the header), `Math.abs(88 - 73)` for task 1 (also correct), and `Math.pow(2, 5)` for task 4 (reversed arguments: that is 2^5).

TEACHER NOTE

For task 4, have one pair write the inside-out steps on the board: `25.0`, `144.0`, `169.0`, `13.0`. A diagonal splits the rectangle into two right triangles, the same shape as the ladder.

Day 1 independent practice

Hand out the Math Class Puzzles worksheet. Students start Part A, questions 1–2 (the value and type of `pow`, `abs`, and `sqrt` calls); circulate and check question 1: students who choose 15 assumed `pow` returns an `int`. Homework is questions 1–3, 7, and 8: finish Part A, the `PowerUp` code

tiles, which statements compile, and the `abs` versus `pow` face-off. Questions 4–6, 9, and 10 use `Math.random`, so they wait for Day 2.

Day 1 closure

Point at `Math.abs(-4)`, `Math.abs(-4.0)`, `Math.pow(2, 2)`, and `Math.sqrt(16)` on the board. For each, students hold up the value and the type: 4 (int), 4.0 (double), 4.0 (double), 4.0 (double). Only `abs` can return an int.

Day 2 warm-up: predict three nested calls

Show `WarmUpNested` and have students write the exact output: 8.0, 27.0, 7. Line 1: `3 - 67` is -64, `abs` returns 64, `sqrt` returns 8.0. Line 2: `Math.sqrt(9)` is 3.0, and `Math.pow(3.0, 3)` is 27.0. Line 3: the cast applies to the value `sqrt` returned, about 7.07, and truncates it to 7.

Day 2 direct instruction: Math.random, endpoints, and stand-ins

Start Part 2 (slide 10).

1. **The header** (1.11.A.2). `double random()` has an empty parameter list, so the call is `Math.random()` with no arguments. Each call returns a new, unpredictable `double`.
2. **The range** (slide 11). The smallest possible result is 0.0, and 0.0 can occur; every result is less than 1.0, and 1.0 cannot occur. Each end of a range is an **endpoint**; it is **inclusive** when the value can occur and **exclusive** when it cannot (1.11.A.3). `Math.random()` is 0.0 inclusive to 1.0 exclusive.
3. **Stand-ins**. You cannot predict the call's value, so trace with values it *could* return. 0.0 gives the low end; a value just under 1.0, such as 0.999, shows the high end.
4. **Run RandomSame**. It always prints 0.0: `r` holds one value, and a value minus itself is 0.0. `Math.random() - Math.random()` makes two calls with two different values, so it almost never gives 0.0. When a program needs the same random value twice, store it once.
5. **Predict: RandomArgument**. `Math.random(6)` does not compile: required: no arguments, found: int. There is no version of `random` that takes a range; the range comes from arithmetic after the call.

MISCONCEPTION `Math.random(6)` gives a number up to 6

Students who have used random functions elsewhere expect to pass the range. The header has an empty parameter list, and Topic 1.9 says arguments must match it in number and type. Show the compiler message, then promise the arithmetic that does the job.

MISCONCEPTION `Math.random()` can return 1.0

"Between 0 and 1" sounds like both ends are included. Point to slide 11's number line (filled dot at 0.0, open dot at 1.0), say the range in words, "0.0 inclusive, 1.0 exclusive," and ask for a

value that is possible (0.0, 0.5, 0.999) and one that is not (1.0, -0.1). The exclusive 1.0 is the reason a die expression can never produce 7.

Day 2 direct instruction: random int values in a range

1. **Stretch, cast, slide** (1.11.A.3, slide 12). Build `(int) (Math.random() * 20) + 1` one operation at a time and track the smallest and largest value: `Math.random() * 20` gives 0.0 up to just under 20.0; `(int) (...)` truncates it to one of 0 through 19 (Topic 1.5); `+ 1` gives 1 through 20. A die works the same way: `(int) (Math.random() * 6) + 1` gives 1 to 6.
2. **Run DieRoll**. Stand-ins 0.0, 0.25, 0.5, and 0.875 give `r * 6` values of 0.0, 1.5, 3.0, and 5.25 and rolls of 1, 2, 4, and 6.
3. **Endpoints after the cast**. The `double` range had an exclusive 6.0; the `int` results 0 through 5 have both endpoints inclusive. Describe an `int` range by listing which whole numbers are possible.
4. **The count** (slide 13). For `low` to `high`, both inclusive, the multiplier is the **count** of whole numbers, `high - low + 1`: `(int) (Math.random() * (high - low + 1)) + low`. Run `RangeCheck` for 10 to 20: with the stand-in 0.999, `* 11` gives 20, and the version with `* 10` gives 19. The `* 10` version can never produce 20. Slide 13 does the same for an age from 50 through 60: count 11, so `(int) (Math.random() * 11) + 50`.
5. **Exclusive ends for int ranges**. "From 10 up to but not including 20" means 10 through 19: count 10. Decide which whole numbers are allowed, then count them.
6. **Predict: the cast too early** (slide 14). `(int) Math.random() * 20 + 1` is 1 every time. The cast applies only to `Math.random()`, which always truncates to 0. `CastTooEarly` shows the same thing with a die.
7. **Pause and try** (slide 15). For each of three expressions, students give the type and the smallest and largest value. A, `(int) (Math.random() * 8) + 3`, is an `int` from 3 through 10. B, with no cast, is a `double` from 3.0 inclusive to 11.0 exclusive. C, `(int) (Math.random() * 8 + 3)`, is also an `int` from 3 through 10: adding before the cast shifts the range first. B leads into the next section.

MISCONCEPTION The multiplier is the high end, or high minus low

Students write `* 12` for 5 to 12, or `* 7`. Have them count the allowed values on their fingers for a small case (1 to 6 is six values) and then check the high end with the stand-in 0.999. `RangeCheck` shows the `* 10` version stopping one short.

MISCONCEPTION The cast applies to the whole expression

`(int) Math.random() * 6 + 1` looks like "cast the random roll." A cast applies only to the value right after it, so it truncates `Math.random()` to 0 and the result is always 1. The program compiles and runs, which makes it a logic error. Parentheses around the product fix it: `(int) (Math.random() * 6) + 1`.

Day 2 direct instruction: random double values in a range

1. **Leave out the cast** (1.11.A.3, slide 16). For a random `double`, multiply by the width of the range and add the low end, with no cast. The slide's wait time, `Math.random() * 2.5 + 1.0`, runs from 1.0 inclusive up to 3.5 exclusive, the same kinds of endpoints as `Math.random()` itself.
2. **Stretch, then slide.** A weather simulation needs a temperature from 60.0 up to 75.0. Multiply to set the **width**: `Math.random() * 15` gives 0.0 up to 15.0. Add to set the starting point: `+ 60` gives 60.0 up to 75.0.
3. **Run TempRange.** Stand-ins 0.0, 0.5, and 0.75 give 60.0, 67.5, and 71.25. Values just under 1.0 give values just under 75.0. The endpoints keep their kind: 60.0 inclusive, 75.0 exclusive.
4. **The pattern.** `Math.random() * width + low`, where the width is high minus low.
5. **A range below zero.** A wind speed from -1.0 up to 1.0 has width 2 and low end -1: `Math.random() * 2 - 1`. `SignedRange` gives -1.0, -0.5, and 0.5 for stand-ins 0.0, 0.25, and 0.75.
6. **Predict: PrecedenceWrong.** With the stand-in 0.5, `r + 60 * 15` prints 900.5, because `*` happens before `+` (Topic 1.3). `r * 15 + 60` prints 67.5.

MISCONCEPTION Writing the slide before the stretch

Students write `Math.random() + 60 * 15`, reading it left to right as "random, then 60 to 75." Precedence computes `60 * 15` first, giving 900.0 up to 901.0. Teach the one-line check: substitute the stand-in 0.0, and the result must be the low end of the range.

Day 2 guided practice: Math in a program, then write the range expression

Start Part 3 (slide 17). Trace slide 18's `Square` program with the input 2.5: `Math.pow(side, 2)` is 6.25, and `Math.sqrt(area)` is 2.5, back to the side. Point out the import for `Scanner (java.util)` and none for `Math (java.lang)`.

Then show slide 19: a random card value from 2 through 14, both included. Pairs write the expression, `(int) (Math.random() * 13) + 2`, and check it with the stand-ins 0.0 and 0.999 (2 and 14). Then write the three tasks below on the board. Pairs write an expression for each task, then check both ends with stand-ins:

Task	Expression	Possible values
A wait time from 0.0 up to but not including 10.0 seconds	<code>Math.random() * 10</code>	0.0 inclusive to 10.0 exclusive
A speed from 5.0 up to but not including 8.0	<code>Math.random() * 3 + 5</code>	5.0 inclusive to 8.0 exclusive
A secret number from 1 to 100, both inclusive	<code>(int) (Math.random() * 100) + 1</code>	1, 2, ..., 100

Run `RangePractice` to check them: with the stand-in 0.0 it prints 0.0, 5.0, and 1 (the low ends); with 0.5 the two `double` expressions give 5.0 and 6.5; with 0.999 the secret number gives 100

(the high end). Its other `int` line checks a step from `-3` to `3`, which is puzzle worksheet question 4; save it for going over the homework.

TEACHER NOTE

The card task is where most errors appear: `* 12` (forgetting the `+ 1` in the count) or `+ 1` (the wrong low end). Ask the pair whether 14 can ever appear, and what the stand-in `0.0` gives: the result must be 2.

Day 2 independent practice

Puzzle worksheet questions 4–6, 9, and 10 are homework: a snippet pool for a random `int` from `-3` through `3`, which values `Math.random()` and a cast can produce, a one-token fix for a die that never rolls 6, a Who Am I? on the five `Math` methods, and a reflection on why `(int) (Math.random() * 4)` can never be 4.

Day 2 closure

Point to a card on slide 20 and have a student give an example. Then show slide 21: pairs explain how to build a random `int` from 10 through 30 with the sentence starter (count 21 values, multiply by 21, cast the whole product with `(int)`, add 10). Contrast it with a `double`: multiply by the **width** and add the **low end**, with no cast.

Day 2 exit ticket

Give the exit ticket (about five minutes). Question 1 checks the value and type of `abs`, `pow`, and `sqrt` calls, including a nested call (1.11.A.2); question 2 checks writing a random `int` expression and its endpoints (1.11.A.3). Re-teach the weaker one at the start of Topic 1.12.