

1.14 Calling Instance Methods

Lesson Plan

Pacing: 3 class periods of 45 minutes

Learning objectives: 1.14.A

Essential knowledge: 1.14.A.1, 1.14.A.2

Materials

- Projector and the Ask the Object slide deck
- Whiteboard and markers
- IDE on every computer
- Printed Ask the Object puzzle worksheet (answer key for you) and exit ticket

Agenda

Minutes	Activity
5	Day 1 warm-up — how many characters?
10	Day 1 direct instruction — instance methods and class methods (slides 1, 3, 6–7)
10	Day 1 direct instruction — the dot operator (slides 2, 4–5)
7	Day 1 guided practice — write the call
9	Day 1 independent practice — puzzle worksheet questions 2, 5, and 7
4	Day 1 closure — which calls compile?
5	Day 2 warm-up — find the @
10	Day 2 direct instruction — arguments and return values (slides 8–10)
6	Day 2 guided practice — fix the calls
12	Day 2 direct instruction — calling methods on a Scanner (slides 9, 11)
9	Day 2 independent practice — puzzle worksheet questions 1 and 4
3	Day 2 closure — what decides the result?
5	Day 3 warm-up — empty, "null", or null?
10	Day 3 direct instruction — calling a method on null (slides 12–14)
6	Day 3 guided practice — print it, or crash? (slide 15) and find the null
8	Day 3 guided practice — putting it together (slides 16–18)
8	Day 3 independent practice — puzzle worksheet questions 3, 6, and 8–10
3	Day 3 closure — the question before every dot (slides 19–20)
5	Day 3 exit ticket

Overview

Three periods on one learning objective, 1.14.A: develop code to call instance methods and determine the result of these calls. Day 1 is about what an instance method is and how to call one: instance methods are called on objects of the class, using the dot operator with the object's name (1.14.A.1). Day 2 applies the same calling rules to arguments, return values, and a second class: `indexOf` needs an argument, a non-void result must be stored or used, a void call stands alone, and a `Scanner` changes as it reads, so repeated calls return different values (1.14.A.1). Day 3 is about the one call that cannot work: a method call on a `null` reference compiles but throws a `NullPointerException` (1.14.A.2). Day 3 ends by putting the calls together in one program, then the rest of the puzzle worksheet and the exit ticket.

The CED's suggested skills are 2.C (write program code involving procedural abstractions) and 3.C (determine the result or output of such code). Every day has prediction before running, and on Day 3 students read, fix, and run a complete program that calls instance methods on a `Scanner` and on a `String`.

Stay inside the topic boundary:

- `String` and `Scanner` are the running examples. The only `String` methods are `int length()` (met in Topic 1.7) and `int indexOf(String str)`, which is new here. Explain each when it first appears; the full `String` method list waits for Topic 1.15.
- `indexOf` needs positions, counted from 0. Teach just enough to use it: the first character is at position 0, and `-1` means "not found." Topic 1.15 covers positions fully.
- The `Scanner` methods are `nextInt()`, `nextDouble()`, and `nextLine()` from Topic 1.4, plus `next()` and the void method `close()`, which are new here. Examples build a `Scanner` from a `String` (the `Scanner(String)` constructor from Topic 1.13) so their output is predictable.
- The deck's `Scoreboard` and the worksheet's `Counter` are made-up classes, given only as constructor and method headers, so students can call void and non-void methods on objects that keep their own data. Writing classes is Unit 3.
- Joining text with `+` is Topic 1.15, so a label and a value still take a `print` followed by a `println`. A string literal such as `"Boston"` gives a `String` object without `new`; that shortcut is also covered in Topic 1.15.
- Method headers for `main` start with `public static`. Students can read past them; they are explained in a later topic.

TEACHER NOTE

The slides are the Ask the Object deck (20 slides, with speaker notes); slide numbers below refer to it. Day 1 uses slides 1–7, Day 2 uses slides 8–11, and Day 3 uses slides 12–20. The Ask the Object puzzle worksheet (10 questions, about 30 minutes) runs across all three days: questions 2, 5, and 7 need only Day 1 material, questions 1 and 4 need Day 2, and questions 3, 6, and 8–10 need Day 3 (question 3 includes a call on `null`).

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, instance and class methods:** `CityLengths`, `ClassVsInstance`, `StaticLength`, and `LengthAsArgument`.
- **Day 1, the dot operator:** `SharedReference` and `MissingParentheses`; `GuidedCalls` holds the guided-practice answers and prints them.

- **Day 2, arguments and return values:** `EmailParts`, `EmailSearches`, `VoidInExpression`, `WrongArgument` with `WrongArgumentFixed`, and `IgnoredResult` with `UsedResult`.
- **Day 2, Scanner:** `ScannerTokens`, `ScoreReader`, `TwoScanners`, `SharedScanner`, and `WrongToken` with `WrongTokenFixed`.
- **Day 3, null:** `NicknameNull`, `EmptyNickname`, `CoachNull`, and `MiddleName` with `MiddleNameFixed`; `OrderSlip` opens putting it together.
- **Printed only (exit ticket):** `ExitCities` and `ExitKiwis`.

Compiler messages are quoted from a recent JDK; the wording can vary slightly between Java versions. The first line of a `NullPointerException` ends with the name of the variable that was `null` or, depending on how the program was compiled, a placeholder such as `<local1>`; tell students to read the line number, not that ending.

Day 1 warm-up: how many characters?

Write `Math.abs(-6)` on the board and ask: "Where does `abs` get the number it works on?" (From the argument, `-6`.) Then ask: "How many characters does a city name have?" Let students call out answers until someone says "Which city?" "`Boston`" has 6 and "`Chicago`" has 7. The question has no answer until you know which string you mean.

Draw out the point: `abs` gets everything from its argument, but "how many characters?" needs a particular `String` object. Today's first idea is a method that is asked of one specific object.

TEACHER NOTE

Students used `length()` in Topic 1.7 and again in Topic 1.12 without the full calling rules. Today those rules are the subject; acknowledge that they have written these calls before and are now learning exactly what each piece does.

Day 1 direct instruction: instance methods and class methods

1. **Instance methods** (1.14.A.1, slides 1 and 3). Read the four goals on slide 3; the first is today's. Some methods are questions you ask a specific object, such as "how long are you?" A method that is called on an object (an instance of a class) and works with that object's own data is an **instance method**. Compare a **class method** from Topic 1.10, such as `Math.abs`, which belongs to the class as a whole; anything it needs arrives as arguments.
2. **Reading the header.** A class method's header includes `static`: `static int abs(int x)`. An instance method's header does not: `int length()`. The empty parameter list of `length()` is not what makes it an instance method; `static double random()` also has empty parentheses and is a class method. Look for `static`.
3. **Predict: CityLengths** (slide 6 afterward). The same call on two objects: "`Boston`" gives 6, and "`Red Sox`" gives 7 because the space counts. Same method, different object, different answer. Slide 6 shows the same with "`frog`" and "`hopper`"; its last line uses `indexOf`, which is Day 2, so say only that the object before the dot decides whose data is used.

4. **Side by side: ClassVsInstance.** Both lines print 6, for different reasons: `Math.abs(-6)` gets its data from its argument; `city.length()` gets it from the object.
5. **Predict: StaticLength** (slide 7 afterward). `String.length()` does not compile: non-static method `length()` cannot be referenced from a static context. "Non-static" means "instance method"; the message asks, "the length of which string?" The class is a blueprint and has no characters. Then show `LengthAsArgument: length(city)` does not compile either (cannot find symbol, method `length(String)`). The object goes in front of the dot, not inside the parentheses.

MISCONCEPTION Calling an instance method through the class name

Because `Math.abs(-6)` uses the class name, students write `String.length()`. Ask the compiler's question aloud: "The length of which string?" The class `String` is a blueprint, not a string with characters. The fix is to put a variable that refers to an object in front of the dot: `city.length()`.

MISCONCEPTION Passing the object as an argument

Students who learned class methods first write `length(city)`, handing the object over the way `-6` is handed to `abs`. An instance method is called **on** the object, so the object goes before the dot. `LengthAsArgument` shows the compiler looking for a method named `length` that takes a `String`, and finding none.

Day 1 direct instruction: the dot operator

1. **The form** (1.14.A.1, slides 4–5). To call an instance method, write the object's variable name, a dot, the method name, and parentheses with any arguments: `objectVariable.methodName(arguments)`. The **dot operator** connects the method to the thing it belongs to: a class for a class method, an object for an instance method.
2. **Take one statement apart.** `int letters = city.length();` `city` says which object, `.` says "use this method of that object," `length` names the method, `()` makes it a call, and `int letters` keeps the result.
3. **What happens when it runs.** `city` holds a reference, like an address (Topic 1.12). Java follows the reference to the object and runs `length()` on that object. As with any call (Topic 1.9), the program pauses, runs the method, and continues with the returned value in place of the call.
4. **You have done this since Topic 1.1** (slide 2). Ask what comes before each dot in the slide's three lines: `console` and `name` are variables that refer to objects, and `System.out` is an object whose job is printing, and `println` is one of its instance methods: `System.out.println("Hi")` calls `println` on that object.
5. **Predict: SharedReference.** Line 2 copies the reference, so `player` and `captain` lead to the same "Jordan"; lines 3 and 4 both print 6. After line 5, `captain` refers to "Li". Output: 6, 6, 6, 2. The method runs on whatever object the variable refers to **at the moment of the call**.
6. **Predict: MissingParentheses.** `song.length` does not compile: cannot find symbol, with symbol: variable `length`. Without parentheses Java looks for a variable named `length`, and `String` has none.

MISCONCEPTION Leaving off the parentheses

Students drop the parentheses when there are no arguments to put in them. The parentheses are what make a call; without them, Java reads `length` as a variable. Have students say "dot, name, parentheses" every time they write a call this week.

MISCONCEPTION Copying a variable copies the object

After `String captain = player;`, students picture two strings and expect a later change to `captain` to leave a copy behind, or expect `player` to change when `captain` is reassigned. Only one object exists; = copied the reference. Reassigning `captain` points it at a different object and leaves `player` alone. Trace with arrows from each variable to its object.

Day 1 guided practice: write the call

Write the two declarations and four tasks on the board. Pairs write one statement for each task, using `String school = "Lincoln High";` and `String motto = "Learn";`:

1. Store the number of characters in `school` in an `int` variable `size`.
2. Print the number of characters in `motto`.
3. Declare `copy` so it refers to the same object as `motto`, then print `copy`'s length.
4. A classmate wrote `int total = String.length() + motto.length;` to add the two lengths. Fix it.

Answers: `int size = school.length();` (12: the space counts), `System.out.println(motto.length());` (5), `String copy = motto;` then `System.out.println(copy.length());` (5, the same object), and `int total = school.length() + motto.length();` (17). Run `GuidedCalls` to confirm: it prints 12, 5, 5, 17.

TEACHER NOTE

For task 4, ask which two mistakes the classmate made before giving the fix: the class name in front of the first dot, and no parentheses on the second call. Both appeared in today's compile-error demos.

Day 1 independent practice

Hand out the puzzle worksheet. Students work questions 2, 5, and 7, which use the made-up `Counter` class from the box on the first page: a be-the-JVM trace with two `Counter` objects, code tiles with a `Counter.click()` decoy, and two truths and a lie about two variables that share one `Counter`. Whatever is not finished is homework. Check question 5: students who use the `Counter.click()` tile are calling an instance method through the class name. Check question 7: students who accept claim 3 think each variable gets its own object.

TEACHER NOTE

Questions 2, 5, and 7 need only Day 1 material; the `Counter` box gives every header they need. Students keep the worksheet: questions 1 and 4 come on Day 2, and questions 3, 6, and 8–10 on Day 3.

Day 1 closure: which calls compile?

Write four statements on the board, each using `String city = "Boston";`: `int a = city.length();`, `int b = String.length();`, `int c = city.length;`, and `int d = length(city);`. Students show a thumb up or down for each. Only the first compiles; ask a different student to give the reason for each of the other three.

Day 2 warm-up: find the @

Write `sam@school.org` on the board. Ask: "How many characters come after the @? How did you work it out?" Most students count from the @ to the end. Ask what a program would need to know to do the same: where the @ is, and how long the whole string is. Those are today's two method calls.

Day 2 direct instruction: arguments and return values

1. **A method that needs an argument** (slide 8). `int indexOf(String str)` searches the string it is called on for `str` and returns the **position** where it first appears. Positions count from **0**. If the text does not appear, the result is **-1**. Topic 1.15 covers positions fully.
2. **Same rules as Topic 1.9** (slide 9). Arguments must match the parameter list in number, order, and type. A non-void method returns a value that you store in a variable or use in an expression. The only new part is the object in front of the dot.
3. **Predict: EmailParts**. Number `sam@school.org` from 0 on the board. `email.indexOf("@")` is 3; `email.length()` is 14; `14 - 3 - 1` is 10. Each call is replaced by its returned value, and the arithmetic proceeds as usual.
4. **Change one thing: EmailSearches**. Same object, different arguments: `"s"` gives 0 (the first match wins), `"school"` gives 4, `"o"` gives 7, and `!"` gives -1. The result depends on both the object and the argument.
5. **A void instance method**. `println`'s header is `void println(String x)`. It returns nothing, so `VoidInExpression (int result = System.out.println("Hello");)` does not compile: incompatible types: void cannot be converted to int. A void call is a statement by itself. Slide 10's made-up `Scoreboard` class shows both kinds side by side: `addPoints` is void and stands alone, `getScore` returns a value to print, and each object keeps its own score (3, 7, then 5).

MISCONCEPTION Counting positions from 1

Students number characters the way people do, starting at 1, and get 4 for the @ in `sam@school.org`. Have them write the position numbers under the characters, starting with 0, every time until it is automatic. Point out that the last position is one less than the length.

Day 2 guided practice: fix the calls

Show three short programs from the guide in the IDE, one at a time. For each, pairs decide: does it compile, what is wrong, and what is the fix?

Program	Compiles?	What is wrong and the fix
WrongArgument	No	<code>length</code> has no parameters, so <code>city.length("Denver")</code> passes one argument too many. Fix: <code>city.length()</code> ; it prints 6.
VoidInExpression	No	<code>println</code> is void, so there is nothing to store in <code>result</code> . Fix: call it as a statement by itself.
IgnoredResult	Yes	It runs and prints only <code>Password checked</code> . The 7 that <code>length()</code> returns is thrown away. Fix: print it, as <code>UsedResult</code> does.

The compiler message for `WrongArgument` is
 method `length` in class `String` cannot be applied to given types, followed by
 required: no arguments and found: `String`.

MISCONCEPTION Calling a method shows its result

Students write `password.length()`; on its own and expect the length to appear. Calling a method does not print anything unless the method's job is printing. A returned value that is not stored or used is gone. `IgnoredResult` compiles, which is what makes this mistake hard to see; compare it with `UsedResult`.

Day 2 direct instruction: calling methods on a Scanner

1. **Scanner calls are instance calls** (slide 9). Since Topic 1.4 students have written `input.nextInt()`. That is an instance method called on a `Scanner` object with the dot operator. New today: a **token** is one chunk of input separated from the next by spaces; `String next()` reads the next token as a `String`; and `void close()` says you are finished with the `Scanner` and returns nothing.
2. **Objects can change.** A `Scanner` remembers how far it has read. Each `nextInt()` or `next()` returns the next token and moves that object's place forward. `city.length()` gives the same answer every time, because measuring a string changes nothing about it. Show `ScannerTokens: 12` and `7` from two identical `nextInt()` calls, `pears` from `next()`, and `close()` as a statement on its own. It prints `19` and `pears`.
3. **Predict: ScoreReader.** `a` is 8, `b` is 3, so line 10 prints 5; line 11 continues where `scores` stopped and prints 9, not 8.

4. **Predict: TwoScanners.** Two objects, each with its own place: 5, 20, 10, 40. Reading from evening does not move morning.
5. **Predict: SharedScanner.** Scanner `b = a;` copies a reference, so there is one Scanner. Line 9 reads 1 (and throws it away, which is allowed); line 10 reads through `b` from the same place and prints 2. Then slide 11 (Two names, one scoreboard): one new, so one Scoreboard, and it prints 4.
6. **Predict: WrongToken.** It prints Reading the count... and then stops with `java.util.InputMismatchException`. It compiles, because `reader.nextInt()` is a correct call; the problem is that the next token is pears. `WrongTokenFixed` reads the tokens in order and prints 12.

MISCONCEPTION `nextInt` finds the first number anywhere

Students think `nextInt()` skips ahead to the next number. It reads the **next** token, whatever it is, and fails if that token is not a whole number. Have students mark the Scanner's place with a caret (▸) in the text and move it one token per call.

MISCONCEPTION A second variable starts a second Scanner

In `SharedScanner`, students expect `b` to start reading from the beginning. Only `new` creates an object; `Scanner b = a;` copies a reference, so both variables lead to the same object and the same place in its text.

Day 2 independent practice

Students work puzzle worksheet questions 1 and 4: the values of `length()` and `indexOf` calls on "lilypad", and a snippet pool that reads a word with `next()` and calls `length()` and `indexOf` on it. Students who finish go back to any of questions 2, 5, and 7 they did not finish. Whatever is not finished is homework. Check question 1: students who give 1 for `word.indexOf("l")` are counting positions from 1.

Day 2 closure: what decides the result?

Ask: "When an instance method runs, what can change its result?" Collect three answers: which object is in front of the dot, the arguments, and, for objects like a Scanner, what earlier calls already did to that object.

Day 3 warm-up: empty, "null", or null?

Show four declarations of `label`: `""`, `"null"`, `null`, and `" "`. Students write what `label.length()` gives for each. Expected: 0, 4, a crash, and 1. Leave the third one unexplained; it is today's topic.

Day 3 direct instruction: calling a method on null

1. **Recall `null`** (Topics 1.4 and 1.13): a reference variable holding `null` refers to no object.
2. **The rule** (1.14.A.2, slides 12–13). The dot operator works by following the reference to an object. With `null` there is nothing to follow, so Java throws a **`NullPointerException`**, a run-time error (Topic 1.1).
3. **Why it compiles.** The compiler checks types, not values: `nickname` is a `String`, and `String` has `length()`. Whether the variable holds an object when the line runs is something only running the program reveals. Slide 14 sorts four calls into compile-time and run-time failures.
4. **Predict: `NicknameNull`.** It prints 3 and `Nickname removed.`, then stops at line 9. Line 10 never runs. Read the message as in Topic 1.1: exception name, `Cannot invoke "String.length()",` and the line number, `NicknameNull.java:9`.
5. **Empty is not `null`.** `EmptyNickname` prints 0: `" "` is a real object with no characters. `null` means there is no object at all.

MISCONCEPTION It compiled, so the variable must have an object

Students assume any call the compiler accepts will run. The compiler cannot tell what a variable will hold when a line runs. Teach the habit for tracing: before every dot, ask "does this variable refer to an object right now?" The fix for a `NullPointerException` is to make the variable refer to a real object before the call.

MISCONCEPTION `null` and the empty string are the same

Both seem to mean "nothing." `" "` is an object with zero characters, so `length()` returns 0; `null` is the absence of an object, so the call throws. And `"null"` in quotes is an ordinary four-letter string. The warm-up's four declarations make all three cases visible.

Day 3 guided practice: print it, or crash? and find the null

Start with slide 15. Pairs decide what each snippet prints or whether it crashes. Expected: A prints `null` (`println` can print a `null` reference; no method is called on it), B crashes, C prints 0, and D crashes (`addPoints` is called on `null`).

Then show `CoachNull`. Pairs trace what each variable holds after each line and write the complete output. Expected: 3, then a `NullPointerException` at line 9; line 10 never runs. Line 8 copies the reference in `coach` into `backup`, and that reference is `null`. Copying `null` does not create an object.

Then show `MiddleName` and ask for a fix. Any assignment that gives `middle` a real `String` works; `MiddleNameFixed` uses `"Elena"` and prints 5, 5, Done.

Day 3 guided practice: putting it together

Run `OrderSlip` once in the IDE as a model: it reads a number with `nextInt()` and a word with `next()` from a `Scanner`, closes it, and then calls `length()` and `indexOf("g")` on the word. Output: Quantity: 3, Item: bagels, Letters in item: 6, Position of g: 2. Slide 17 shows the same

pattern from the keyboard: `console.next()` asks the `Scanner`, and `word.length()` and `word.indexOf("e")` ask the `String`. With `breeze` typed, it prints 6 and 2.

Then slide 18 (Two bugs, two lines). Pairs find both mistakes before anyone types: line 2 needs parentheses, `console.next()`, and line 3 needs the object before the dot, `word.length()`. Students type the fixed program in the IDE and run it with a word of their choice.

Day 3 independent practice

Students work puzzle worksheet questions 3, 6, and 8–10: a be-the-compiler table whose snippets include a call on `null`, a face-off between the compiler and the JVM about `w.length()` when `w` is `null`, a one-token fix for a `NullPointerException`, a who-am-I review, and a two-sentence reflection on what the object before the dot does. Whatever is not finished is homework. Check question 3, snippet 5: students who write "does not compile" think the compiler can see that the variable holds `null`.

Day 3 closure: the question before every dot

Leave slide 19 (Words to know) up. Ask the class to finish the sentence together: "Before every dot, I ask ..." (Does this variable refer to an object right now?) Then ask one student to name the two things in front of a dot that work (a class name for a class method, an object variable for an instance method) and the one value that compiles but fails (`null`). If time allows, slide 20's sentence starter is a quick pair-share.

Day 3 exit ticket

Give the exit ticket (about five minutes). Question 1 checks determining the result of instance calls through shared and reassigned references, and the `NullPointerException` (1.14.A.1, 1.14.A.2); question 2 checks writing instance calls on a `Scanner` and a `String` (1.14.A.1). Re-teach the weaker one at the start of Topic 1.15.