

1.10 Calling Class Methods

Self-Study Guide · About 45 minutes · With answer key

By the end you will be able to

- Explain what a class method is and find the keyword that marks one in a method header.
- Call a class method using the class name, a dot, and the method name.
- Work out the value a class method call produces, including calls nested inside other calls.
- Explain when the class name can be left out of a call, and fix calls that leave it out where it is needed.

What to remember from earlier

- A class groups related data and behaviors and can be used to make objects. ([Topic 1.7: Application Program Interface \(API\) and Libraries](#), [1.7.5 Classes Are Types That Make Objects](#))
- A method is a named block of code that runs only when it is called. ([Topic 1.9: Method Signatures](#), [1.9.1 Methods Are Named Blocks of Code](#))
- Arguments must match the parameters in number, order, and type. ([Topic 1.9: Method Signatures](#), [1.9.4 Parameters and Arguments](#))
- A non-void method's return value must be stored or used in an expression. ([Topic 1.9: Method Signatures](#), [1.9.7 Return Values](#))
- When both numbers are int, / drops the decimal part, so 9 / 4 is 2. ([Topic 1.3: Expressions and Output](#), [1.3.6 Integer Division](#))
- Casting with (int) drops the decimal part of a double. ([Topic 1.5: Casting and Range of Variables](#), [1.5.2 Casting a double to an int](#))

1.10.1 Class Methods Belong to the Class

Suppose you need the square root of 49 in a program. Back in Topic 1.7 you read the documentation for Java's `Math` class, and in Topic 1.9 you read its method headers, so you know a `sqrt` method exists. But knowing a method exists is not enough. You need to know how to actually reach it from your code, and that depends on what kind of method it is.

Recall from Topic 1.7 that a class is like a blueprint: it can be used to make objects, also called **instances** of the class. Some methods do their job for one particular object. Asking for the length of a piece of text, for example, only makes sense once you say *which* text; Java has to know which `String` object you mean. You will learn to call that kind of method in Topics 1.12 to 1.14.

Finding a square root is different. `sqrt` needs nothing except the number you hand it as an argument. There is no particular object involved; the job belongs to the `Math` class as a whole. A method like this is a **class method**: a method that belongs to the class itself rather than to any one instance of it. You can use a class method without making any objects at all.

An everyday comparison: your school's main office can tell anyone the date of the next holiday. You ask "the office," not a particular student, because the answer belongs to the school as a whole. Asking a student for *their* locker number is different: the answer depends on which student

you ask. Class methods are like the office. (The comparison stops working at one point: a Java class is not a person who answers questions; it is a set of code that runs when called.)

How can you tell a class method from other methods? Look at its header. A class method has the keyword `static` in its header, placed before the return type and the method name. `static` is the word that says "this method belongs to the class."

KEY IDEA

A class method belongs to the class, not to any one object made from it. Its header contains the keyword `static` before the method name.

WORKED EXAMPLE Reading class method headers

Here are three headers you can find in the `Math` documentation. Each is the first line of a method you will use in this chapter:

```
1 public static double sqrt(double a)
2 public static int abs(int a)
3 public static double pow(double a, double b)
```

Take the first header apart, word by word:

Part	What it tells you
<code>public</code>	This word is explained in a later topic; read past it for now.
<code>static</code>	This is a class method: it belongs to the <code>Math</code> class, not to an object.
<code>double</code>	The return type: the call sends back a <code>double</code> .
<code>sqrt</code>	The method name.
<code>(double a)</code>	The parameter list: one <code>double</code> argument goes in.

All three headers contain `static`, so all three are class methods. That is true of every method in `Math`.

Now vary the class. Here is a header from the `String` documentation, which you saw in Topic 1.9:

```
1 public int length()
```

There is no `static`. That means `length` is **not** a class method: it belongs to each `String` object, which makes sense because every piece of text has its own length. You do not need to know how to call it yet; the point is that the presence or absence of `static` is how you tell the two kinds apart.

You have actually been writing a class method since your very first program. Look at the header every program starts with: `public static void main(String[] args)`. It contains `static`, so `main` is a class method. That is how Java can start your program by running `main` without first making any object.

WATCH OUT

"**static describes the value that comes back.**" Because `static` sits right next to the return type, students sometimes read `static double` as a special kind of `double`, or think `static` means "the result never changes." Neither is true. `Math.sqrt(49.0)` and `Math.sqrt(16.0)` return different values. The return type is only the word just before the method name (`double`). `static` says who the method belongs to: the class.

PAUSE AND TRY**Question 1** Multiple choice

Which header belongs to a class method?

- A. `public int length()`
- B. `public static double abs(double a)`
- C. `public String toUpperCase()`
- D. `public boolean isEmpty()`

Question 2 Short answer

The header `public static double sqrt(double a)` contains `static`. Explain what that tells you about the method and why it makes sense for a square root method to be designed this way.

1.10.2 Calling with the Class Name and a Dot

You know `sqrt` belongs to the `Math` class. But `Math` is not the only class in Java; many classes could have a method named `sqrt`, and you could write one yourself. When you call a class method from your program, how does Java know which class's method you mean?

You tell it. The usual way to call a class method is to write the class name, then a dot, then the method name and its arguments in parentheses:

```
1 ClassName.methodName(arguments)
```

The dot is called the **dot operator**. Read it as "go into": `Math.sqrt(49.0)` means "go into the `Math` class and run its `sqrt` method with the argument `49.0`." The class name tells Java where to look; the parts after the dot follow the rules you learned in Topic 1.9, so the arguments must match the parameter list in number, order, and type.

KEY IDEA

Call a class method by writing the class name, a dot, the method name, and the arguments in parentheses, as in `Math.sqrt(49.0)`.

WORKED EXAMPLE Calling two Math methods

This code uses two class methods from `Math`:

- `Math.sqrt` returns the square root of its argument as a `double`.

- `Math.abs` returns the absolute value of its argument: the number without its minus sign, so `-12.5` becomes `12.5` and `8` stays `8`.

```
1 double side = Math.sqrt(49.0);
2 System.out.println(side);
3 double total = Math.abs(-12.5) + 2.5;
4 System.out.println(total);
```

Go line by line:

Line	What Java does	Result
1	Goes into <code>Math</code> , runs <code>sqrt</code> with <code>49.0</code> . The call returns <code>7.0</code> , which is stored in <code>side</code> .	side is 7.0
2	Prints <code>side</code> .	7.0
3	Runs <code>Math.abs(-12.5)</code> , which returns <code>12.5</code> . The call is part of an expression, so Java computes <code>12.5 + 2.5</code> .	total is 15.0
4	Prints <code>total</code> .	15.0

Output:

```
7.0
15.0
```

Both methods are non-void, so their return values had to be used: line 1 stores it in a variable, and line 3 uses it inside an arithmetic expression.

Now change the argument types. In this version, the arguments are `int` values instead of `double` values:

```
1 int morningTemp = 12;
2 int afternoonTemp = 3;
3 int change = Math.abs(afternoonTemp - morningTemp);
4 System.out.println(change);
5 System.out.println(Math.sqrt(16));
```

```
9
4.0
```

Two things changed, for different reasons:

- `afternoonTemp - morningTemp` is `3 - 12`, which is `-9`. The argument is an `int`, so Java uses the version of `abs` whose header is `public static int abs(int a)`. Recall from Topic 1.9 that a method is **overloaded** when several methods share a name but have different signatures. `abs` is overloaded: there is one version for `int` and one for `double`, and Java picks the one whose parameter type matches the argument. This one returns an `int`, so `9` prints with no decimal point.
- `sqrt` has only one version, and its parameter is a `double`. Java can store an `int` in a `double` parameter, so the argument `16` is accepted and becomes `16.0`. The return type is still `double`, so the result prints as `4.0`, not `4`.

The lesson: the type of value that comes back is the return type in the header of the method that runs. For `sqrt` that is always `double`, even when the argument looks like an `int`.

WATCH OUT

Passing an argument the method cannot accept. The class name and dot get you to the right method, but the arguments still have to fit the parameter list. A student who wants the square root of 25 might put the number in quotes:

```
1 double root = Math.sqrt("25");
2 System.out.println(root);
```

The compiler stops with error: `incompatible types: String cannot be converted to double`. "25" in quotes is text, not a number, and `sqrt` takes a `double`. Remove the quotes: `Math.sqrt(25)` or `Math.sqrt(25.0)`.

PAUSE AND TRY

Question 3 Tracing

Write the exact output of this code. Remember to pay attention to the return type of each method.

```
1 int loss = -7;
2 System.out.println(Math.abs(loss));
3 System.out.println(Math.abs(-7.0));
4 System.out.println(Math.sqrt(64));
```

Question 4 Multiple choice

`Math.pow` has the header `public static double pow(double a, double b)`. It returns `a` raised to the power `b`, so `pow(5, 2)` means 5^2 . Which statement compiles and stores 5^2 in a variable?

- A. `double square = Math.pow(5, 2);`
- B. `int square = Math.pow(5, 2);`
- C. `double square = pow.Math(5, 2);`
- D. `double square = Math(pow, 5, 2);`

1.10.3 Working Out What a Call Produces

Real programs rarely call one method on a line by itself. A formula might need a square root of a sum of powers, which means one call sits inside another. To predict what such code prints, you need a reliable way to work out the value of each call.

Here is the rule, built from what you know about expressions (Topic 1.3) and about method calls (Topic 1.9):

1. **Arguments first.** Before a method runs, Java works out the value of each argument completely. If an argument is an expression, it is evaluated with the usual rules, including integer division.
2. **The method runs.** The parameter is given a copy of the argument's value, and the method's code runs.
3. **The call is replaced by its return value.** Once the method returns, the whole call `Math.sqrt(...)` acts like the value it sent back, and the rest of the expression continues with that value.

When calls are nested, apply the rule from the inside out: the inner call is an argument of the outer call, so it is finished first.

KEY IDEA

To find the value of a call, evaluate its arguments first, then replace the whole call with the value the method returns. Nested calls are worked out from the inside out.

WORKED EXAMPLE A square root of a sum of powers

A right triangle has legs 3 and 4. The long side is the square root of $(3^2 + 4^2)$. In Java:

```
1 double legA = 3.0;
2 double legB = 4.0;
3 double longSide = Math.sqrt(Math.pow(legA, 2) + Math.pow(legB, 2));
4 System.out.println(longSide);
```

Line 3 has three calls. Work it out one step at a time; each row replaces one piece with its value:

Step	Expression so far	What happened
1	<code>Math.sqrt(Math.pow(3.0, 2) + Math.pow(4.0, 2))</code>	Variables replaced by their values
2	<code>Math.sqrt(9.0 + Math.pow(4.0, 2))</code>	<code>Math.pow(3.0, 2)</code> returned 9.0
3	<code>Math.sqrt(9.0 + 16.0)</code>	<code>Math.pow(4.0, 2)</code> returned 16.0
4	<code>Math.sqrt(25.0)</code>	The argument of <code>sqrt</code> is now one value
5	<code>5.0</code>	<code>sqrt</code> returned 5.0

`longSide` holds 5.0, and line 4 prints:

```
5.0
```

`sqrt` could not start until its whole argument, `Math.pow(...) + Math.pow(...)`, was a single value. That is why the inner calls ran first.

Now change only the type of the numbers inside an argument. The two lines below differ by one character:

```
1 System.out.println(Math.sqrt(9 / 4));
2 System.out.println(Math.sqrt(9.0 / 4));
```

```
1.4142135623730951
1.5
```

In the first line, Java evaluates the argument `9 / 4` before calling `sqrt`. Both numbers are `int`, so this is integer division and the result is `2`, not `2.25`. `sqrt` never sees `2.25`; it receives `2` (as `2.0`) and returns the square root of `2`. In the second line, `9.0 / 4` is `2.25`, and the square root of `2.25` is `1.5`. The method behaved the same both times; the arguments were different because they were evaluated first.

WATCH OUT

Calling a non-void method and ignoring its result. A student wants to remove the minus sign from a balance:

```
1 double balance = -4.5;
2 Math.abs(balance);
3 System.out.println(balance);
```

This compiles and prints `-4.5`. Line 2 does call `abs`, and the call does return `4.5`, but nothing stores or uses that value, so it is thrown away. The method received a copy of `balance` (call by value, from Topic 1.9), so `balance` itself is untouched. Students make this mistake because "`Math.abs(balance)`" reads like a command, "make balance positive." It is really a question, "what is the absolute value of balance?", and you must keep the answer:

```
1 double balance = -4.5;
2 balance = Math.abs(balance);
3 System.out.println(balance);
```

This version prints `4.5`.

PAUSE AND TRY

Question 5 Tracing

Write the value stored in `result` and the exact output. Show each step of your work.

```
1 double result = Math.sqrt(Math.abs(-36)) + Math.pow(2, 3);
2 System.out.println(result);
```

Question 6 Debugging

A student wants the whole-number part of the square root of 30 and writes the code below. It does not compile. Explain why, and write a corrected first line.

```
1 int wholeRoot = Math.sqrt(30.0);
2 System.out.println(wholeRoot);
```

1.10.4 Calls Inside the Defining Class

So far, every class method you have called lived in `Math`, while your code lived in a different class. But you can write class methods in your own classes too. In Topic 1.9 you saw methods with a header and a body; if that header includes `static`, the method is a class method of your class. When your own class calls its own class method, do you still need to write the class name?

You may, but you do not have to. The class a method is written in is its **defining class**. When the call is written inside the defining class, Java already knows which class to look in first: the one it is in. So the class name and dot are optional. When the call comes from a different class, as every call to `Math` from your programs does, the class name is required, because Java will not search other classes for you.

KEY IDEA

Inside a method's defining class, a class method can be called with or without the class name. From any other class, the class name and dot are required.

WORKED EXAMPLE Two ways to call your own class method

`ShiftPlanner` defines a class method, `toMinutes`, and then calls it twice from `main`, which is in the same class:

`ShiftPlanner.java`

```
1 public class ShiftPlanner
2 {
3     public static int toMinutes(int hours)
4     {
5         return hours * 60;
6     }
7
8     public static void main(String[] args)
9     {
10        int shortShift = toMinutes(4);
11        int longShift = ShiftPlanner.toMinutes(9);
12        System.out.println(shortShift);
13        System.out.println(longShift);
14    }
15 }
```

- Lines 3 to 6 define `toMinutes`. Its header has `static`, so it is a class method of `ShiftPlanner`. It returns its parameter times 60.
- Line 10 calls it **without** the class name: `toMinutes(4)`. This is allowed because the call is inside `ShiftPlanner`, the defining class. Flow of control jumps to line 5, which returns `4 * 60`, that is 240, and 240 is stored in `shortShift`.
- Line 11 calls it **with** the class name: `ShiftPlanner.toMinutes(9)`. This also works; it is simply more than Java needs here. The call returns 540.

```
240
540
```

Both forms run exactly the same method.

Now vary the kind of method. `printDivider` below is a **void** class method, so it is called as a statement on its own rather than inside an expression:

ScoreCard.java

```
1 public class ScoreCard
2 {
3     public static void printDivider()
4     {
5         System.out.println("-----");
6     }
7
8     public static void main(String[] args)
9     {
10        System.out.println("Round 1: 18");
11        printDivider();
12        System.out.println("Round 2: 24");
13        ScoreCard.printDivider();
14        System.out.println("Done");
15    }
16 }
```

```
Round 1: 18
-----
Round 2: 24
-----
Done
```

Line 11 (`printDivider();`) and line 13 (`ScoreCard.printDivider();`) both pause `main`, run line 5, and return to the next line of `main`. The class-name rule is the same for void and non-void class methods.

WATCH OUT

Leaving out the class name outside the defining class. After seeing `toMinutes(4)` work, students often try the same shortcut with `Math`:

NoClassName.java

```
1 public class NoClassName
2 {
3     public static void main(String[] args)
4     {
5         double root = sqrt(81.0);
6         System.out.println(root);
7     }
8 }
```

```
NoClassName.java:5: error: cannot find symbol
    double root = sqrt(81.0);
                  ^
symbol:   method sqrt(double)
location: class NoClassName
```

(Only the main lines of the message are shown.) The shortcut works only inside the defining class. `sqrt` is defined in `Math`, but this call is inside `NoClassName`, so Java looks in

NoClassName, finds no method called `sqrt`, and reports "cannot find symbol." The message even says where it looked: `location: class NoClassName`. Fix it by naming the class: `Math.sqrt(81.0)`.

WATCH OUT

Forgetting static on your own method. If the header leaves out `static`, the method is no longer a class method:

MissingStatic.java

```
1 public class MissingStatic
2 {
3     public int toMinutes(int hours)
4     {
5         return hours * 60;
6     }
7
8     public static void main(String[] args)
9     {
10        int shift = toMinutes(4);
11        System.out.println(shift);
12    }
13 }
```

The compiler reports

non-static method `toMinutes(int)` cannot be referenced from a static context. `main` is a class method, so no object exists when it runs, and a method without `static` belongs to an object (Topics 1.12 to 1.14 explain those). Add `static` to the header on line 3, and the call on line 10 works.

PAUSE AND TRY

Question 7 Multiple choice

A class named `Converter` defines `public static int toInches(int feet)`. Inside `Converter`'s `main` method, which of these calls compile?

- I. `int height = toInches(5);`
- II. `int height = Converter.toInches(5);`
- III. `int height = converter.toInches(5);`

- A. I only
- B. II only
- C. I and II only
- D. I, II, and III

Question 8 Short answer

The same `Converter` class from Question 7 is used by a second class, `Measure`. In `Measure`'s `main` method, a student writes `int height = toInches(5);` and it does not compile. Explain why the call works in `Converter` but not in `Measure`, and give the corrected call.

Putting it together

This program finds straight-line distances on a city map, where each place has an across position and an up position. It defines its own class method, `distance`, which calls two `Math` class methods inside it.

MapDistance.java

```

1  public class MapDistance
2  {
3      public static double distance(double x1, double y1, double x2, double y2)
4      {
5          double across = x2 - x1;
6          double up = y2 - y1;
7          return Math.sqrt(Math.pow(across, 2) + Math.pow(up, 2));
8      }
9
10     public static void main(String[] args)
11     {
12         double toLibrary = distance(0.0, 0.0, 3.0, 4.0);
13         double toPark = MapDistance.distance(2.0, 1.0, 8.0, 9.0);
14         System.out.print("Library: ");
15         System.out.println(toLibrary);
16         System.out.print("Park: ");
17         System.out.println(toPark);
18         System.out.print("Difference: ");
19         System.out.println(Math.abs(toLibrary - toPark));
20     }
21 }
```

Look at the calls first, before tracing:

Line	Call	Class name used?	Why
7	<code>Math.sqrt(...)</code> , <code>Math.pow(...)</code>	Yes, required	<code>sqrt</code> and <code>pow</code> are defined in <code>Math</code>
12	<code>distance(...)</code>	No	<code>distance</code> is defined in <code>MapDistance</code> , this class
13	<code>MapDistance.distance(...)</code>	Yes, optional	Same method; the class name is allowed
19	<code>Math.abs(...)</code>	Yes, required	<code>abs</code> is defined in <code>Math</code>

Now trace. Line 12 pauses `main` and runs `distance` with `x1 = 0.0`, `y1 = 0.0`, `x2 = 3.0`, `y2 = 4.0`:

Line in <code>distance</code>	Work	Value
5	<code>across = 3.0 - 0.0</code>	3.0
6	<code>up = 4.0 - 0.0</code>	4.0
7	<code>Math.sqrt(Math.pow(3.0, 2) + Math.pow(4.0, 2))</code> → <code>Math.sqrt(9.0 + 16.0)</code> → <code>Math.sqrt(25.0)</code>	returns 5.0

Control returns to line 12, and `toLibrary` holds 5.0. Line 13 runs the same method with 2.0, 1.0, 8.0, 9.0: `across` is 6.0, `up` is 8.0, and `Math.sqrt(36.0 + 64.0)` is `Math.sqrt(100.0)`, so `toPark` holds 10.0.

Lines 14 to 17 print each label with `print` (which stays on the same line) and then the distance with `println`. On line 19, the argument `toLibrary - toPark` is evaluated first: `5.0 - 10.0` is `-5.0`. Then `Math.abs(-5.0)` returns `5.0`, which `println` displays after the label from line 18:

```
Library: 5.0
Park: 10.0
Difference: 5.0
```

Every idea from the chapter appears: `static` marks `distance` as a class method, `Math` methods are called through the class name, arguments are evaluated before each call and each call is replaced by its return value, and the class name is optional only for the method defined in this class.

Vocabulary

Term	Definition	Example
instance	An object made from a class	One particular <code>String</code>
class method	A method that belongs to the class itself, not to any one instance; needs no object to call	<code>Math.sqrt</code>
static	The keyword in a method header, before the method name, that marks a class method	<code>public static int abs(int a)</code>
dot operator	The <code>.</code> between a class name and a method name; it says which class the method is in	<code>Math.abs(-3)</code>
defining class	The class in which a method is written	<code>Math</code> for <code>sqrt</code>

Chapter summary

- A **class method** belongs to its class, not to any instance of the class, so it can be used without making an object. Its header contains `static` before the method name; `main` is one.
- The usual way to call a class method is the class name, the dot operator, the method name, and the arguments: `Math.sqrt(49.0)`. The arguments must fit the parameter list, and the return type in the header decides the type of the result.
- To find the value of a call, evaluate its arguments first, then replace the call with its return value. Nested calls are worked out from the inside out. A non-void call whose result is not stored or used has no effect on your variables.
- Inside a method's defining class, the class name is optional in the call. From any other class it is required, and leaving it out causes a "cannot find symbol" error.

Review questions

Question 9 Multiple choice

Which statement about class methods is true?

- A. A class method can only be called from inside the class that defines it.

- B. A class method is called through the class name and needs no object, and its header contains `static`.
- C. The keyword `static` means a class method always returns the same value.
- D. A class method must be called with the class name, even inside its defining class.

Question 10 Tracing

Write the exact output of this program.

Workout.java

```
1 public class Workout
2 {
3     public static void printLaps(int laps)
4     {
5         System.out.print("Laps: ");
6         System.out.println(laps);
7     }
8
9     public static void main(String[] args)
10    {
11        System.out.println("Start");
12        printLaps(Math.abs(-6));
13        Workout.printLaps(10 - 4 * 2);
14        System.out.print("Pace: ");
15        System.out.println(Math.sqrt(2.25));
16    }
17 }
```

Question 11 Debugging

This program should print the area of a square with side 3.0, but it has **two** compile errors. Find both, explain each, and write the corrected lines.

AreaHelper.java

```
1 public class AreaHelper
2 {
3     public double squareArea(double side)
4     {
5         return pow(side, 2);
6     }
7
8     public static void main(String[] args)
9     {
10        System.out.println(squareArea(3.0));
11    }
12 }
```

Question 12 Multiple choice

What does this code print on its **first** line?

```
1 System.out.println(Math.sqrt(5 / 2 * 8) + Math.abs(3 - 10));
2 System.out.println(Math.sqrt(5.0 / 2 * 8) + Math.abs(3 - 10));
```

- A. 11.0

- B. 11
- C. 11.47213595499958
- D. -3.0

Question 13 Tracing

Here a class method calls another class method of the same class, not just `main`. Write the exact output.

PaintJob.java

```
1 public class PaintJob
2 {
3     public static double wallArea(double width, double height)
4     {
5         return width * height;
6     }
7
8     public static double gallonsNeeded(double width, double height)
9     {
10        return wallArea(width, height) / 400.0;
11    }
12
13    public static void main(String[] args)
14    {
15        System.out.println(PaintJob.wallArea(20.0, 10.0));
16        System.out.println(gallonsNeeded(20.0, 10.0));
17        System.out.println(Math.abs(wallArea(5.0, 8.0) - 100.0));
18    }
19 }
```

Reflection

Check each statement. If the answer is "not yet," reread the linked subsection.

- ☐ I can explain what makes a method a class method and find static in its header. Reread: 1.10.1
Class Methods Belong to the Class
- ☐ I can call a class method using the class name and the dot operator. Reread: 1.10.2 Calling with the
Class Name and a Dot
- ☐ I can work out the value of nested class method calls, evaluating arguments first. Reread: 1.10.3
Working Out What a Call Produces
- ☐ I can explain when the class name may be left out of a call and when it is required. Reread: 1.10.4
Calls Inside the Defining Class

Check Your Reasoning

Answers and explanations for every question, numbered to match.

Question 1

B. A class method has the keyword `static` in its header before the method name. Only B contains `static`.

- **A** is wrong: it has no `static`, so it belongs to each object (each `String` has its own `length`), not to the class.
- **C** is wrong: no `static`. Having a class name such as `String` as the return type does not make it a class method.
- **D** is wrong: no `static`. The return type `boolean` tells you what comes back, not who the method belongs to.

Reread 1.10.1 *Class Methods Belong to the Class* if you chose differently.

Question 2

Model answer: `static` means `sqrt` is a class method: it belongs to the `Math` class itself, not to any object made from the class. That makes sense because a square root depends only on the number passed in as the argument. There is no particular object whose information the method needs, so there is no reason to require one.

A complete answer must:

- say that `static` marks a class method that belongs to the class, not to an instance, and
- explain that everything `sqrt` needs arrives through its argument.

Reread 1.10.1 *Class Methods Belong to the Class* if your answer differed.

Question 3

```
7
7.0
8.0
```

- `Math.abs(loss)`: the argument is an `int`, so the `int` version of `abs` runs and returns the `int` `7`.
- `Math.abs(-7.0)`: the argument is a `double`, so the `double` version runs and returns `7.0`.
- `Math.sqrt(64)`: `sqrt` always returns a `double`, so the result is `8.0` even though the argument was an `int`.

Reread 1.10.2 *Calling with the Class Name and a Dot* if your output differed.

Question 4

A. The call is written class name, dot, method name, arguments: `Math.pow(5, 2)`. The method returns a `double` (`25.0`), and it is stored in a `double` variable.

- **B** is wrong: the return type is `double`, and Java will not store a `double` in an `int` variable without a cast. The compiler reports
`incompatible types: possible lossy conversion from double to int`.
- **C** is wrong: the class name and method name are reversed. The class name comes first, before the dot.

- **D** is wrong: the method name is not an argument. It goes after the dot, and only the values the method needs go inside the parentheses.

Choice B written out as code, which does not compile:

```
1 int square = Math.pow(5, 2);
2 System.out.println(square);
```

Reread 1.10.2 *Calling with the Class Name and a Dot* if you chose differently.

Question 5

Work from the inside out, one call at a time:

Step	Expression so far	What happened
1	<code>Math.sqrt(Math.abs(-36)) + Math.pow(2, 3)</code>	Start
2	<code>Math.sqrt(36) + Math.pow(2, 3)</code>	<code>Math.abs(-36)</code> returned the <code>int</code> 36
3	<code>6.0 + Math.pow(2, 3)</code>	<code>Math.sqrt(36)</code> returned the <code>double</code> 6.0
4	<code>6.0 + 8.0</code>	<code>Math.pow(2, 3)</code> returned 8.0 (2 ³ as a double)
5	<code>14.0</code>	Addition

result holds 14.0, and the output is:

```
14.0
```

If you wrote 14, remember that `sqrt` and `pow` both return `double`, so the sum is a `double`. Reread 1.10.3 *Working Out What a Call Produces* if your steps differed.

Question 6

Why: `Math.sqrt` returns a `double` (here about 5.477). Java will not quietly store a `double` in an `int` variable, because the decimal part would be lost. The compiler reports incompatible types: possible lossy conversion from `double` to `int`.

Corrected code: cast the return value to `int`, as you learned in Topic 1.5:

```
1 int wholeRoot = (int) Math.sqrt(30.0);
2 System.out.println(wholeRoot);
```

The call is evaluated first and returns about 5.477; the cast then drops the decimal part, so `wholeRoot` holds 5 and the program prints 5.

Reread 1.10.3 *Working Out What a Call Produces* if you missed the error.

Question 7

C. The call is inside `Converter`, the defining class, so the class name is optional: `I` (without it) and `II` (with it) both compile and call the same method.

- **A** is wrong: adding the class name is never an error; it is just optional here.
- **B** is wrong: inside the defining class, the class name may be left out, so `I` also compiles.
- **D** is wrong: Java is case-sensitive (Topic 1.1). `converter` with a lowercase `c` is not the class name, so Java reports "cannot find symbol."

Reread *1.10.4 Calls Inside the Defining Class* if you chose differently.

Question 8

Model answer: `toInches` is defined in `Converter`. Leaving out the class name only works when the call is inside the defining class. In `Measure`, Java looks for `toInches` in `Measure`, does not find it, and reports "cannot find symbol." The corrected call names the class:

```
int height = Converter.toInches(5);
```

A complete answer must:

- say that the class name is optional only inside the defining class (`Converter`), and
- give the call with the class name and dot: `Converter.toInches(5)`.

Reread *1.10.4 Calls Inside the Defining Class* if your answer differed.

Question 9

B. Class methods belong to the class, so the usual call is `ClassName.method(...)` with no object, and their headers include `static`.

- **A** is wrong: your programs call `Math.sqrt` from their own classes all the time. From other classes, you just need the class name.
- **C** is wrong: `static` says who the method belongs to, not anything about its result. `Math.sqrt(4.0)` and `Math.sqrt(9.0)` return different values.
- **D** is wrong: inside the defining class the class name is optional, as `toMinutes(4)` in `ShiftPlanner` showed.

Reread *1.10.1 Class Methods Belong to the Class* and *1.10.4 Calls Inside the Defining Class* if you chose differently.

Question 10

```
Start
Laps: 6
Laps: 2
Pace: 1.5
```

Line	What happens	Output
11	Prints the text.	Start
12	Argument first: <code>Math.abs(-6)</code> returns 6. Then <code>printLaps(6)</code> runs lines 5 and 6 and returns.	Laps: 6
13	Argument first: <code>10 - 4 * 2</code> is <code>10 - 8</code> , which is 2. <code>Workout.printLaps(2)</code> runs lines 5 and 6.	Laps: 2
14-15	<code>print</code> shows the label; <code>Math.sqrt(2.25)</code> returns 1.5, which <code>println</code> shows on the same line.	Pace: 1.5

Line 12 calls `printLaps` without the class name and line 13 with it; both are fine because the calls are inside `Workout`. The argument of `Math.abs` is an `int`, so it returns the `int` 6, which prints as 6, not 6.0.

Reread *1.10.3 Working Out What a Call Produces* and *1.10.4 Calls Inside the Defining Class* if your output differed.

Question 11

Error 1 (line 3): the header has no `static`, so `squareArea` is not a class method. `main` is a class method and has no object, so line 10's call fails with

non-static method `squareArea(double)` cannot be referenced from a static context. Fix: `public static double squareArea(double side)`.

Error 2 (line 5): `pow` is defined in `Math`, not in `AreaHelper`, so leaving out the class name only makes Java look in `AreaHelper`. The compiler reports `cannot find symbol` for method `pow(double,int)`. Fix: `return Math.pow(side, 2);`.

Line 10's call, `squareArea(3.0)` without the class name, is fine once the method is static, because it is inside the defining class. Here is the corrected program (renamed so it can be saved alongside the original):

`AreaHelperFixed.java`

```
1 public class AreaHelperFixed
2 {
3     public static double squareArea(double side)
4     {
5         return Math.pow(side, 2);
6     }
7
8     public static void main(String[] args)
9     {
10        System.out.println(squareArea(3.0));
11    }
12 }
```

It prints `9.0`.

Reread [1.10.4 Calls Inside the Defining Class](#) if you missed either error.

Question 12

A. Arguments are evaluated before each call. `5 / 2 * 8` uses integer division: `5 / 2` is 2, then `2 * 8` is 16. `Math.sqrt(16)` returns `4.0`. `3 - 10` is `-7`, and `Math.abs(-7)` returns 7. Then `4.0 + 7` is `11.0`.

- **B** is wrong: `sqrt` returns a `double`, so the sum is a `double` and prints with `.0`.
- **C** is wrong: that is the **second** line's output. It comes from doing `5 / 2` as `2.5`, but with two `int` values Java drops the decimal part before `sqrt` is ever called.
- **D** is wrong: it skips `abs`. `Math.abs(-7)` returns 7, not `-7`.

Reread [1.10.3 Working Out What a Call Produces](#) if you chose differently.

Question 13

```
200.0
0.5
60.0
```

Line	What happens	Output
15	<code>PaintJob.wallArea(20.0, 10.0)</code> returns <code>20.0 * 10.0</code> .	200.0

16	<code>gallonsNeeded(20.0, 10.0)</code> runs line 10. Inside it, <code>wallArea(20.0, 10.0)</code> returns <code>200.0</code> , then <code>200.0 / 400.0</code> is <code>0.5</code> , which goes back to <code>main</code> .	<code>0.5</code>
17	Argument first: <code>wallArea(5.0, 8.0)</code> returns <code>40.0</code> ; <code>40.0 - 100.0</code> is <code>-60.0</code> . <code>Math.abs(-60.0)</code> returns <code>60.0</code> .	<code>60.0</code>

Line 10 calls `wallArea` without the class name from inside `gallonsNeeded`. That works because the shortcut depends on which class the call is in, not on which method: every method of `PaintJob` can call `wallArea` either way. `Math.abs` still needs `Math.` because `abs` is not defined in `PaintJob`.

Reread *1.10.4 Calls Inside the Defining Class* and *1.10.3 Working Out What a Call Produces* if your output differed.