

Calling on the Class

Calling class methods



What comes before the dot?

PAUSE AND TRY

Look at what comes right before the dot in each call. What is it?

```
Scanner console = new Scanner(System.in);  
int age = console.nextInt();  
  
double root = Math.sqrt(25.0);
```

Before the first dot is console, a variable made on the line above it. Before the second dot is Math. Where was Math ever made?

By the end of class you can...

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

A large white square with rounded corners, tilted slightly to the right, containing a large black number '1' in the center.

1

PART 1

Methods that belong to the class

static marks a class method



A **class method** belongs to the class, not to any one object made from it. Its header has the keyword **static** before the return type.

CLASS METHODS (STATIC)

```
public static int abs(int a)
public static double abs(double a)
```

Called through the class: no object needed.

INSTANCE METHODS (NO STATIC)

```
public String next()
public int nextInt()
```

Called on an object, such as a Scanner.

Reading a header, word by word

```
public static int abs(int a)
```

Part	What it tells you
<code>public</code>	Explained in a later unit. Read past it for now.
<code>static</code>	A class method: it belongs to the Math class, not to an object.
<code>int</code>	The return type: the call sends back an int.
<code>abs</code>	The method name.
<code>(int a)</code>	The parameter list: one int argument goes in.

The test for any header: **static** before the return type means a class method. No **static** means an instance method.



static does not mean "never changes"



static says who the method belongs to: the class. It says nothing about the value that comes back.

SAME METHOD, DIFFERENT RESULTS

```
System.out.println(Math.abs(-3));  
System.out.println(Math.abs(-40));
```

OUTPUT

```
3  
40
```

In `public static int abs(int a)`, the return type is **int**, the one word just before the method name. **static** is not part of the return type.

A large, bold, teal number '2' is centered within a white rounded square that has a slight 3D effect and a shadow.

PART 2

Class name, dot, method

ClassName.methodName(arguments)



Write the **class name**, a **dot**, the **method name**, and the **arguments** in parentheses. No object needed.

TWO MATH CALLS

```
double side = Math.sqrt(81.0);  
System.out.println(side);  
double gap = Math.abs(-6.5);  
System.out.println(gap);
```

OUTPUT

```
9.0  
6.5
```

Math.sqrt(x) returns the square root of x. **Math.abs(x)** returns x without its minus sign.

The header decides the type that comes back

PREDICT, THEN CHECK

```
int drop = 4 - 11;  
System.out.println(Math.abs(drop));  
System.out.println(Math.abs(-2.5));  
System.out.println(Math.sqrt(36));
```

OUTPUT

7
2.5
6.0

Call	Version that runs	Returns
Math.abs(drop)	int abs(int a)	int 7
Math.abs(-2.5)	double abs(double a)	double 2.5
Math.sqrt(36)	double sqrt(double a)	double 6.0

Which one compiles?

PAUSE AND TRY

Math.pow(a, b) gives a to the power b. Which line stores 2 to the 5th?

A `double p = Math.pow(2, 5);`

B `int p = Math.pow(2, 5);`

C `double p = pow.Math(2, 5);`

D `double p = Math(pow, 2, 5);`

A large, bold, teal-colored number '3' centered within a white rounded square with a slight drop shadow.

3

PART 3

What a call turns into

Arguments first, then the call becomes its value

```
double d = Math.sqrt(Math.pow(6, 2) + Math.pow(8, 2));
```

Expression so far	What happened
<code>Math.sqrt(Math.pow(6, 2) + Math.pow(8, 2))</code>	Start
<code>Math.sqrt(36.0 + Math.pow(8, 2))</code>	<code>Math.pow(6, 2)</code> returned 36.0
<code>Math.sqrt(36.0 + 64.0)</code>	<code>Math.pow(8, 2)</code> returned 64.0
<code>Math.sqrt(100.0)</code>	The argument is now one value
10.0	<code>Math.sqrt</code> returned 10.0

Arguments first, then the method runs, then **the call is replaced by its return value**. Nested calls go inside out.

The argument is worked out with the usual rules

ONE CHARACTER APART

```
System.out.println(Math.sqrt(25 / 4));  
System.out.println(Math.sqrt(25.0 / 4));
```

OUTPUT

2.449489742783178
2.5

Call	Argument becomes	sqrt returns
Math.sqrt(25 / 4)	6 (int division drops .25)	about 2.449
Math.sqrt(25.0 / 4)	6.25	2.5

A method only sees the **value** of its argument, never the expression you typed.

Calling it is not the same as keeping it



A non-void call whose result is not stored or used **changes nothing**. The value comes back and is thrown away.

SPOT THE LINE THAT DOES NOTHING

```
double temp = -8.0;  
Math.abs(temp);  
System.out.println(temp);  
temp = Math.abs(temp);  
System.out.println(temp);
```

OUTPUT

```
-8.0  
8.0
```

Store the result, or use it in an expression.

Trace it from the inside out

PAUSE AND TRY

What does this print? Write one step per line.

```
double r = Math.abs(5 - 9) + Math.sqrt(Math.pow(3, 2));  
System.out.println(r);
```

Step 1: _____ Step 2: _____ Step 3: _____ Output: _____

A large, bold, teal-colored number '4' is centered within a white square with rounded corners. The square is slightly tilted to the right.

4

PART 4

Inside the defining class

Inside its own class, the name is optional

TICKETS.JAVA

```
public class Tickets {  
    public static int total(int adults, int kids) {  
        return adults * 12 + kids * 8;  
    }  
  
    public static void main(String[] args) {  
        int family = total(2, 3);  
        int pair = Tickets.total(1, 1);  
        System.out.println(family);  
        System.out.println(pair);  
    }  
}
```

OUTPUT

48
20

The **defining class** is the class a method is written in. Inside it, `total(2, 3)` and `Tickets.total(2, 3)` run the same method.

Outside the defining class, the name is required

GARDEN.JAVA

```
public class Garden {  
    public static void main(String[] args) {  
        double side = sqrt(64.0);  
        System.out.println(side);  
    }  
}
```



Without a class name, Java looks only in the class the call is written in. `sqrt` is defined in **Math**.

COMPILER SAYS

```
Garden.java:3: error: cannot find symbol  
    symbol:   method sqrt(double)  
    location: class Garden
```

Fix: `Math.sqrt(64.0)`

Which calls compile in Race?

PAUSE AND TRY

Timer has this class method. Each call below is written in Race.

```
public static int toSeconds(int minutes)
```

- I. `int a = toSeconds(3);`
- II. `int a = Timer.toSeconds(3);`
- III. `int a = timer.toSeconds(3);`

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

Words to know

Class method

A method that belongs to the class, not to one object. Needs no object to call.

static

The keyword in a header, before the return type, that marks a class method.

Dot operator

The . between the class name and the method name, as in `Math.abs(-3)`.

Defining class

The class a method is written in. Inside it, the class name is optional.

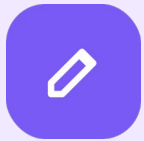
Argument

A value passed in a call. Worked out before the method runs.

Return value

The value a non-void method sends back. The call is replaced by it.

When do you need the class name?



Explain to your partner why `Math.sqrt(49.0)` needs `Math.` in your program, but `total(2, 3)` did not need `Tickets.` inside `Tickets`.

SENTENCE STARTER

Inside the _____ class, the class name is optional, but from any other class Java looks only _____, so _____.

Next: the Calling on the Class puzzle worksheet.