

Read the Header

Method signatures



One name, many kinds of values

PAUSE AND TRY

All five lines call println. How can one method accept all of these?

```
System.out.println(5);  
System.out.println(2.5);  
System.out.println("hi");  
System.out.println();  
System.out.println("end");
```

An int, a double, a String, and nothing at all.

One method can't have a parameter that is all of those at once. So what is going on?

By the end of class you can...

- 1 Explain what a method is and why you can use one without seeing its code.
- 2 Read a method header: return type, name, parameters, and signature.
- 3 Check that a call's arguments match in number, order, and type.
- 4 Trace a call into a method and back, and use a return value correctly.
- 5 Explain call by value and predict which overloaded method runs.

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 are named blocks

A method runs only when it is called

```
public class Reminder {  
    public static void main(String[] args) {  
        System.out.println("Homeroom at 8:00");  
        printReminder();  
        System.out.println("Lunch at 11:30");  
        printReminder();  
    }  
  
    public static void printReminder() {  
        System.out.println("Bring your book");  
    }  
}
```

OUTPUT

```
Homeroom at 8:00  
Bring your book  
Lunch at 11:30  
Bring your book
```

A **method** is a named block of code. A **block** is code inside braces. Writing a method does not run it; a **call** does.

The parentheses are the call



A call is the method's name **followed by parentheses**, even when nothing goes inside them.

DOES NOT COMPILE

```
printReminder;
```

COMPILER SAYS

```
error: not a statement
```

A CALL

```
printReminder();
```

OUTPUT

```
Bring your book
```

Use it without seeing inside

FROM THE MATH DOCUMENTATION

```
/**  
 * Returns a double value greater than or  
 * equal to 0.0 and less than 1.0.  
 */  
public static double random()
```

```
double r = Math.random();
```

`r` holds some double from 0.0 up to, but not including, 1.0. You never see how it was picked.



Procedural abstraction lets you use a method by knowing **what** it does, even if you don't know **how** it was written. The header and the documentation are all you need.

A large white square with rounded corners, tilted slightly to the right, containing the number '2' in a teal font.

2

PART 2

Reading a header

From header to signature

```
public static void printTotal(double price, int qty)
```

RETURN TYPE NAME PARAMETER LIST

Header	Signature
public static void printTotal(double price, int qty)	printTotal(double, int)
public String substring(int beginIndex, int endIndex)	substring(int, int)
public static double random()	random()

Signature = the name + the ordered list of parameter types. The return type and parameter names are left out.

Parameters get the arguments, in order

```
public static void main(String[] args) {  
    printTotal(4.5, 3);  
}  
  
public static void printTotal(double price, int qty) {  
    System.out.println(price * qty);  
}
```

OUTPUT

13.5

Position	Argument	Parameter
1st	4.5	double price
2nd	3	int qty

Parameter: a variable in the header. **Argument:** the value in the call. They match by position.

Right values, wrong order



Arguments must match the signature in **number, order, and type**. Check the header, left to right.

DOES NOT COMPILE

```
printTotal(3, 4.5);
```

COMPILER SAYS

```
incompatible types: possible  
lossy conversion from double to int
```

Argument	Goes to	OK?
3	double price	yes: 3 widens to 3.0
4.5	int qty	no: would lose .5

Which calls compile?

PAUSE AND TRY

Use the header below. Which calls compile? For those that do, what prints?

```
public static void printTotal(double price, int qty)
```

A `printTotal(2.0, 5);`

C `printTotal(2, 5);`

B `printTotal(5, 2.0);`

D `printTotal(2.0);`

A large, white, rounded square containing the number '3' in a bold, teal, sans-serif font.

3

PART 3

Calls, returns, and copies

Jump in, run, come back

```
public static void main(String[] args) {  
    System.out.println("measuring");  
    int total = toInches(3) + 4;  
    System.out.println(total);  
}  
  
public static int toInches(int feet) {  
    System.out.println("converting");  
    return feet * 12;  
}
```

OUTPUT

```
measuring  
converting  
40
```



A call **interrupts** the normal order: the method's statements run first. A **return** statement sends a value back and ends the method. Control returns to the point right after the call, and the value takes the call's place.

void or non-void?

VOID METHOD

```
void printTotal(double price, int qty)
```

```
printTotal(4.5, 3);
```

Nothing comes back. The call stands alone as a statement.

NON-VOID METHOD

```
int toInches(int feet)
```

```
int inches = toInches(2);  
System.out.println(toInches(2) + 1);
```

An int comes back. Store it, or use it in an expression.



double t = printTotal(4.5, 3); does not compile: void cannot be converted to double. **toInches(5);** compiles, but its answer is thrown away.

The method gets a copy

```
public static void main(String[] args) {  
    int count = 5;  
    addOne(count);  
    System.out.println(count);  
}  
  
public static void addOne(int n) {  
    n = n + 1;  
    System.out.println(n);  
}
```

OUTPUT

6
5

Step	count	n
call starts	5	5 (copy)
n = n + 1	5	6
back in main	5	gone

Call by value: each parameter starts as a copy of its argument. Changing the copy never changes the caller's variable.

A large, white, rounded square containing a bold, dark teal number '4'.

4

PART 4

Overloaded methods

Same name, different signatures

```
public static int area(int side) {  
    return side * side;  
}  
  
public static int area(int width, int height) {  
    return width * height;  
}
```

Methods are **overloaded** when they share a name but have **different signatures**. The arguments decide which one runs.

```
System.out.println(area(4));  
System.out.println(area(4, 5));
```

OUTPUT

```
16  
20
```

Adding **double area(int length)** fails: its signature is area(int) again. "method area(int) is already defined."

Which show runs?

PAUSE AND TRY

Which version does each call run? Write "none" if it does not compile.

THREE VERSIONS

```
public static void show(int n)
public static void show(double d)
public static void show(int n, double d)
```

A `show(3);`**B** `show(3.0);`**C** `show(3, 4);`**D** `show(3.0, 4);`

Words to know

Method

A named block of code that runs only when it is called.

Parameter

A variable in the header that receives a value.

Argument

The value passed in when the method is called.

Signature

The name plus the ordered list of parameter types.

void / return

void gives nothing back; return sends a value back.

Overloaded

Same name, different signatures.

Read this header out loud



Tell your partner everything this header says:

```
public static String initials(String first, String last)
```

SENTENCE STARTER

It returns a _____, it needs _____, and its signature is _____.

Next: the Read the Header puzzle worksheet.