

And, Or, Not

Compound Boolean expressions



Who gets into the skate park?

PAUSE AND TRY

Rule: you need a helmet and you must be signed in. Who gets in?

Skater	Helmet?	Signed in?	Gets in?
Ana	yes	yes	
Ben	yes	no	
Cy	no	yes	
Dee	no	no	

Then: what if the rule said helmet **or** signed in? Who gets in now?

By the end of class you can...

- 1 Use ! (not) to flip a boolean value or a comparison.
- 2 Join conditions with && (and) and || (or), and find the value of the result.
- 3 Apply the order !, then &&, then ||, and use parentheses to change it.
- 4 Explain when Java skips the right side, and use a guard to prevent a crash.
- 5 Spot conditions that don't compile or don't mean what you intended.

A white rounded square with a thin black border, containing the number '1' in a bold, black, sans-serif font.

1

PART 1

NOT: flip it

! turns true into false, and false into true



The **NOT** operator `!` goes in front of one boolean value. `!a` is true when `a` is false, and false otherwise.



To flip a comparison, wrap the whole comparison in parentheses:
`!(volume < 20)`

CODE

```
boolean muted = true;  
System.out.println(!muted);  
  
int volume = 35;  
System.out.println(!(volume < 20));
```

OUTPUT

```
false  
true
```

WATCH OUT

! grabs only the value right next to it



! runs before any comparison, so `!volume < 20` means `(!volume) < 20`. But `volume` is an `int`, and only a `boolean` can be flipped.

DOES NOT COMPILE

```
int volume = 35;  
System.out.println(!volume < 20);
```

FIXED

```
int volume = 35;  
System.out.println(!(volume < 20));
```

COMPILER SAYS

```
error: bad operand type int  
for unary operator '!'
```

"Unary" means the operator takes **one** operand. The parentheses turn the comparison into a `boolean` first.

2

PART 2

AND and OR

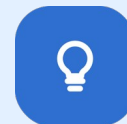
Both, or at least one

a	b	a && b	a b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Either way, the whole expression is one boolean value.



AND a && b is true only when **both** sides are true.



OR a || b is true when **at least one** side is true, including both.

A range with &&, a choice with ||

```
int minutes = 45;
if (minutes >= 30 && minutes <= 60) {
    System.out.println("Just right");
} else {
    System.out.println("Too short or too long");
}
```

OUTPUT

Just right

```
String day = "Sun";
if (day.equals("Sat") || day.equals("Sun")) {
    System.out.println("Sleep in");
} else {
    System.out.println("Alarm at 7");
}
```

OUTPUT

Sleep in

Each side must be a whole boolean



Both sides of `&&` and `||` must be complete boolean expressions. Java does not fill in the missing piece.

DOES NOT COMPILE

```
boolean ok = 30 <= minutes <= 60;
```

error: bad operand types
for binary operator '<='

FIXED

```
boolean ok = 30 <= minutes && minutes <= 60;
```

DOES NOT COMPILE

```
boolean weekend = day.equals("Sat") || "Sun";
```

error: bad operand types
for binary operator '||'

FIXED

```
boolean weekend = day.equals("Sat") || day.equals("Sun");
```

When English "and" means ||



Ask: must **one** person pass **both** tests (&&)? Or is passing **either** test enough (||)?

"Students and teachers get 10% off."

SOUNDS RIGHT, MEANS THE WRONG THING

`isStudent && isTeacher`

True only for someone who is both. A student alone gets no discount.

WHAT THE STORE MEANT

`isStudent || isTeacher`

True for any student and any teacher. Either one is enough.

True or false?

PAUSE AND TRY

Use the values on the left. Is each expression true or false?

```
boolean hasKey = true;  
boolean locked = true;  
int code = 1234;
```

Expression	Value
hasKey && !locked	
!hasKey locked	
code == 1234 && hasKey	
!(code == 1234) !hasKey	

A white rounded square containing the number '3' in a bold, dark teal font.

3

PART 3

Who goes first?

The order: !, then &&, then ||

1 Parentheses whatever is inside goes first

2 ! (not) flips the single value next to it

3 Comparisons < <= > >= == !=

4 && (and) before any ||

5 || (or) last

READ IT LIKE JAVA

```
a || b && c
```

```
// Java reads it as  
a || (b && c)
```



Like $2 + 3 * 4$ in math: the "times" (&&) happens before the "plus" (||). Parentheses change the order.

Say it with parentheses

"Members or orders over \$50 get free shipping, this week only."

```
boolean member = true;  
double total = 20.0;  
boolean thisWeek = false;  
System.out.println(member || total > 50 && thisWeek);  
System.out.println((member || total > 50) && thisWeek);
```

OUTPUT

```
true  
false
```



Line 1 gives a member free shipping in the wrong week, because && runs first. If you need the || to happen first, you must add the parentheses.

What prints?

PAUSE AND TRY

Write the three lines this prints. Show which operator runs first on each line.

```
int x = 4;  
System.out.println(x > 0 || x > 10 && x < 2);  
System.out.println(!(x > 3) && x < 10);  
System.out.println(!(x > 3 && x < 10));
```

YOUR 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

Short-circuit evaluation

Java skips a side that can't change the answer



Java evaluates the **left** side first. `false && ...` is always false, and `true || ...` is always true, so Java skips the right side. That is **short-circuit evaluation**.



A **guard** is a safe check on the left that keeps a risky right side from running when it would crash.

```
String word = "";  
if (word.length() > 0  
    && word.substring(0, 1).equals("h")) {  
    System.out.println("Starts with h");  
} else {  
    System.out.println("No h at the start");  
}
```

OUTPUT

No h at the start

The guard has to come first



Short-circuiting only skips the **right** side. If the risky part is on the left, it runs before the guard is ever checked.

GUARD ON THE WRONG SIDE

```
String word = "";
if (word.substring(0, 1).equals("h")
    && word.length() > 0) {
    System.out.println("Starts with h");
}
```

WHEN IT RUNS

```
Exception in thread "main"
java.lang.
StringIndexOutOfBoundsException:
Range [0, 1) out of bounds
for length 0
```

It compiles. The crash happens while the program runs, so the compiler can't warn you. Put the guard on the left.

Crash or no crash?

PAUSE AND TRY

For each line, write what prints, or "crash".

```
int boxes = 0;
int cookies = 24;
/* A */ System.out.println(boxes == 0 || cookies / boxes > 5);
/* B */ System.out.println(boxes != 0 && cookies / boxes > 5);
/* C */ System.out.println(cookies / boxes > 5 || boxes == 0);
```

Words to know

Logical operator

!, &&, or ||: works on boolean values and gives a boolean.

Compound Boolean expression

Smaller boolean expressions joined by logical operators.

NOT, AND, OR

! flips a value.
&& needs both.
|| needs one or both.

Precedence

The order operators run: !, then &&, then ||.

Short-circuit evaluation

Skipping the right side when the left side decides the answer.

Guard

A safe check on the left that protects a risky right side.

Both, or either?



Write one rule from your school day that needs && and one that needs ||. Then tell your partner how you chose.

SENTENCE STARTER

I use && when _____, and || when _____.

Next: the And, Or, Not puzzle worksheet.