Chapter 8: Loops

Computer Fundamentals and Introduction to Programming

for · while · do-while · Nested loops · Examples

Sercan KÜLCÜ · Giresun University

© Sercan Külcü. All rights reserved. · Interactive HTML version of these slides: https://sercankulcu.github.io/files/computer/slides/Chapter_08_Loops.html

INTRODUCTION

Learning Outcomes

Loops

Choose between for, while and do-while.

Patterns

Compute with counters and accumulators.

Nesting

Produce row-column patterns with nested loops.

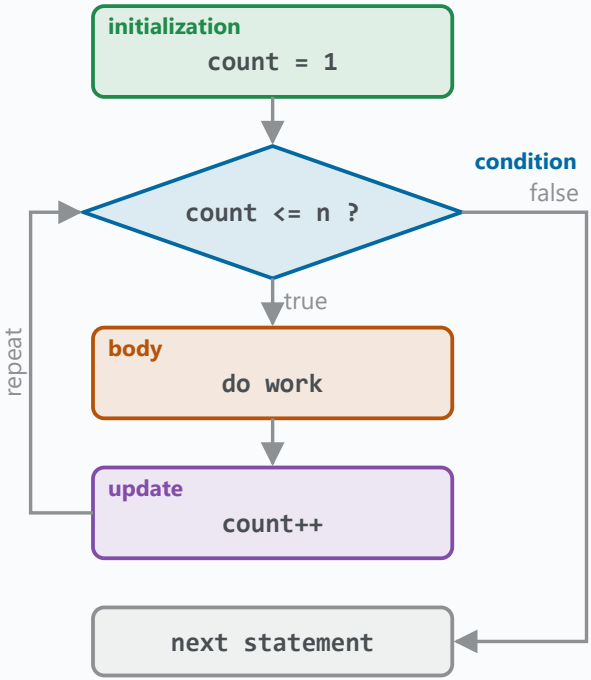
Tracing

Verify a loop step by step at its boundary values.

Parts of a Loop

- **Initialization**: state set up once, before the loop.
- **Condition**: tested before each pass (after it in do-while); false means exit.
- **Body**: the work done on each pass.
- **Update**: moves the condition toward false; forget it and the loop never ends.

These four colors mean the same thing throughout the chapter.

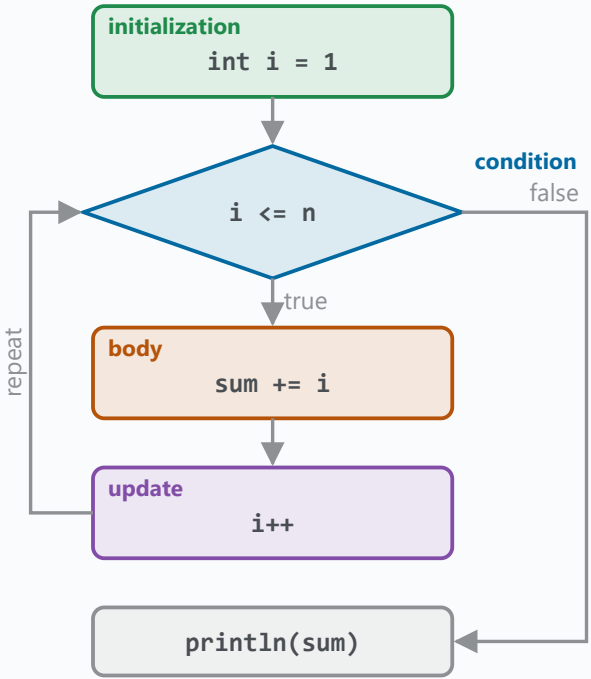


The for Loop

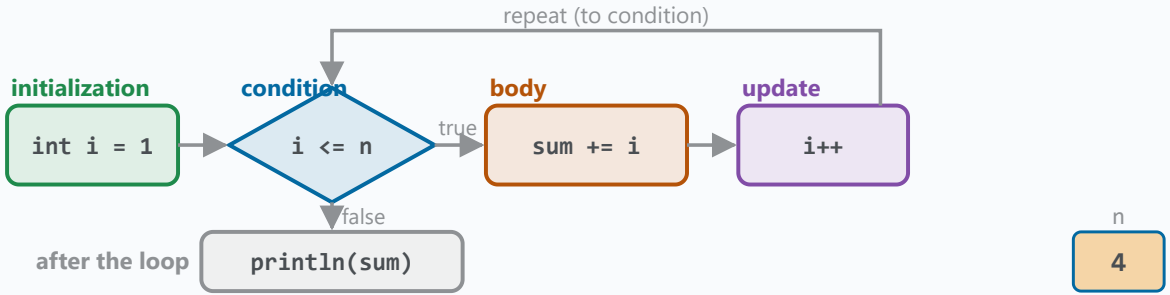
```
int n = 4;
int sum = 0;
for (int i = 1; i <= n; i++) {
    sum += i;
}
System.out.println(sum);
```

initialization condition body update

- Use for when the number of passes is known; all four parts fit on one line.
- Order: initialization → condition → body → update → condition …
- i exists only inside the loop.



Code Trace: 1 + 2 + 3 + 4 with `for`



n = 4: we will add 1 through 4.

Step 1 / 17

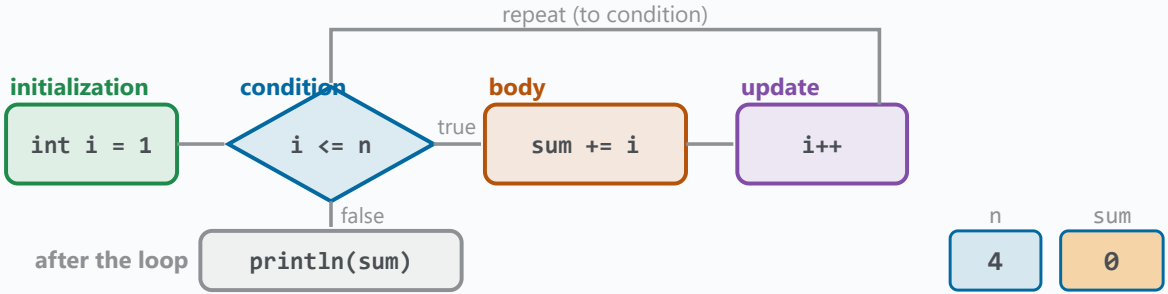
```
▶ int n = 4;
2 int sum = 0;
3 for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

n = 4

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



The accumulator starts at **sum = 0** (nothing added yet).

Step 2 / 17

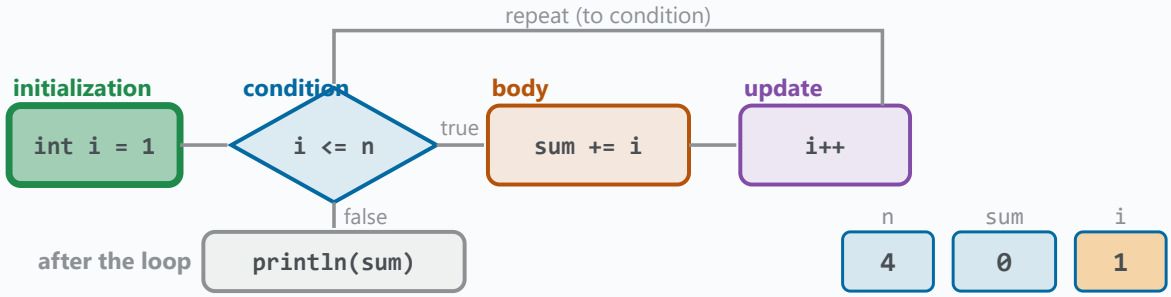
```
1 int n = 4;
▶ int sum = 0;
3 for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 0`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Initialization: `int i = 1` runs only once.

Step 3 / 17

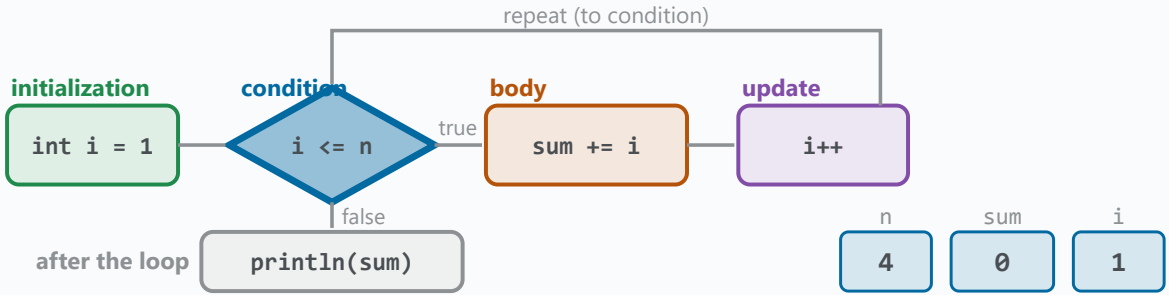
```
1 int n = 4;
2 int sum = 0;
▶ for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 0` `i = 1`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Condition: `i <= n` → 1 <= 4 → **true** → enter the body.

Step 4 / 17

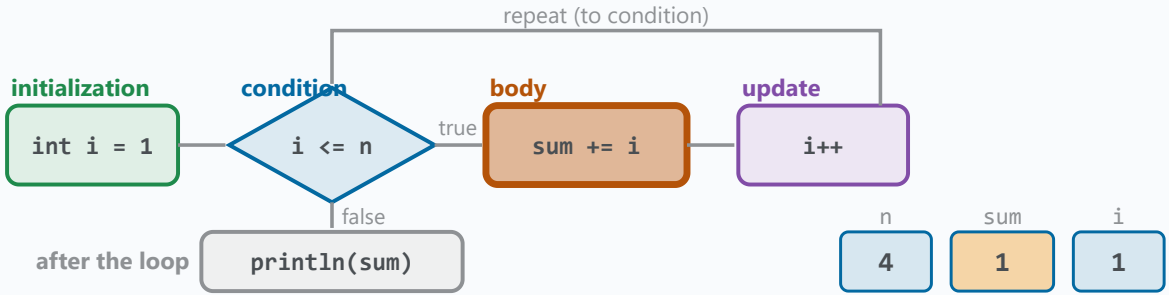
```
1 int n = 4;
2 int sum = 0;
▶ for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 0` `i = 1`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Body: sum = 0 + 1 = **1**.

Step 5 / 17

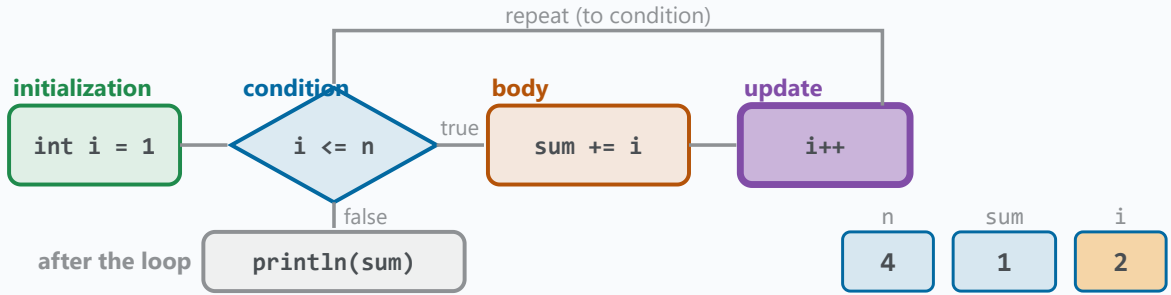
```
1 int n = 4;
2 int sum = 0;
3 for (int i = 1; i <= n; i++) {
▶     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 1` `i = 1`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Update: `i++` → i = 2. Then back to the condition.

Step 6 / 17

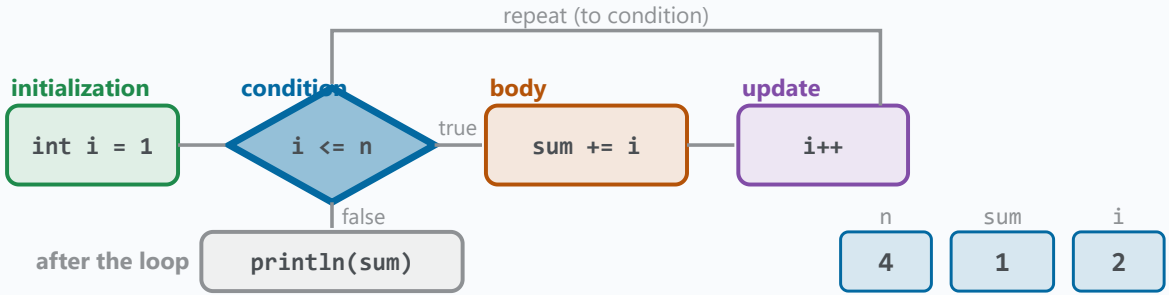
```
1 int n = 4;
2 int sum = 0;
▶ for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 1` `i = 2`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Condition: `i <= n` → 2 <= 4 → **true** → enter the body.

Step 7 / 17

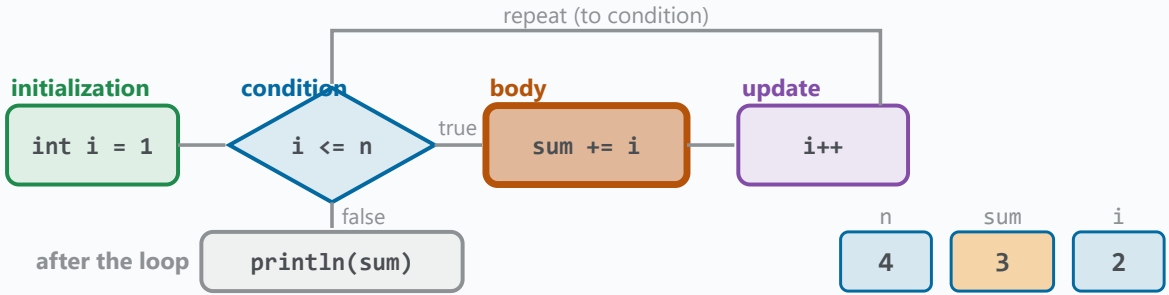
```
1 int n = 4;
2 int sum = 0;
▶ for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 1` `i = 2`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Body: sum = 1 + 2 = **3**.

Step 8 / 17

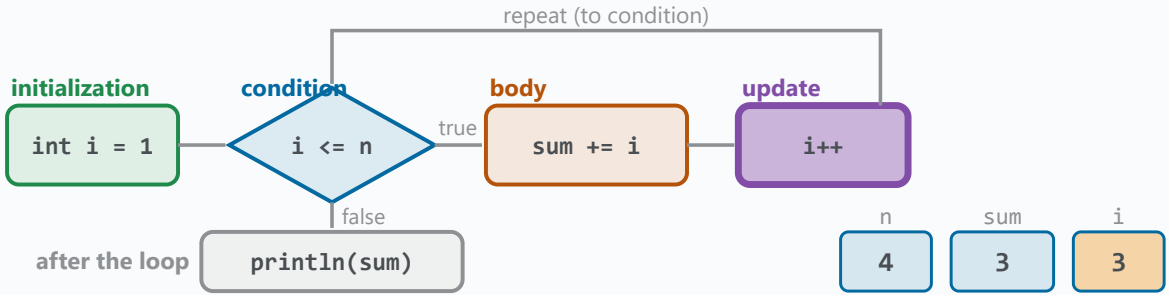
```
1 int n = 4;
2 int sum = 0;
3 for (int i = 1; i <= n; i++) {
▶     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 3` `i = 2`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Update: `i++` → i = 3. Then back to the condition.

Step 9 / 17

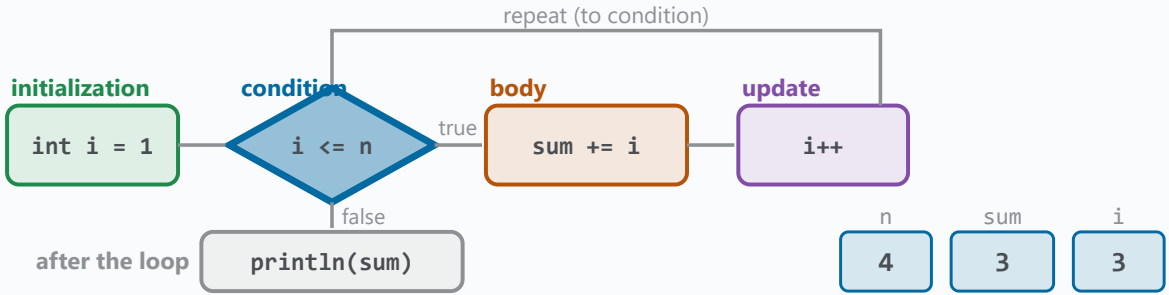
```
1 int n = 4;
2 int sum = 0;
▶ for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 3` `i = 3`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Condition: `i <= n` → 3 <= 4 → **true** → enter the body.

Step 10 / 17

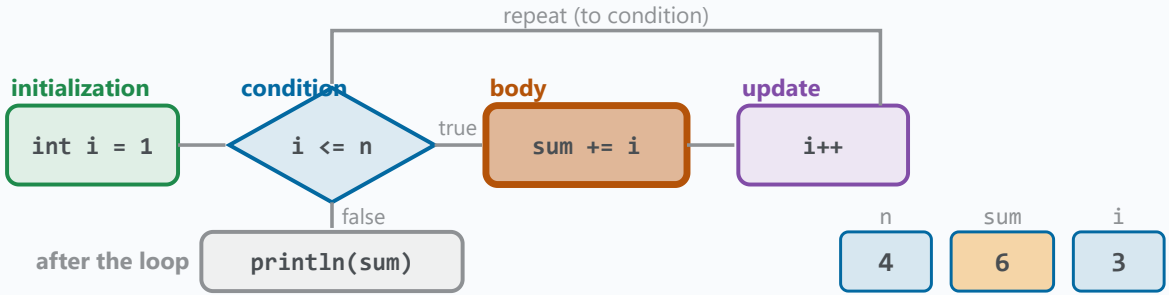
```
1 int n = 4;
2 int sum = 0;
▶ for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 3` `i = 3`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Body: sum = 3 + 3 = **6**.

Step 11 / 17

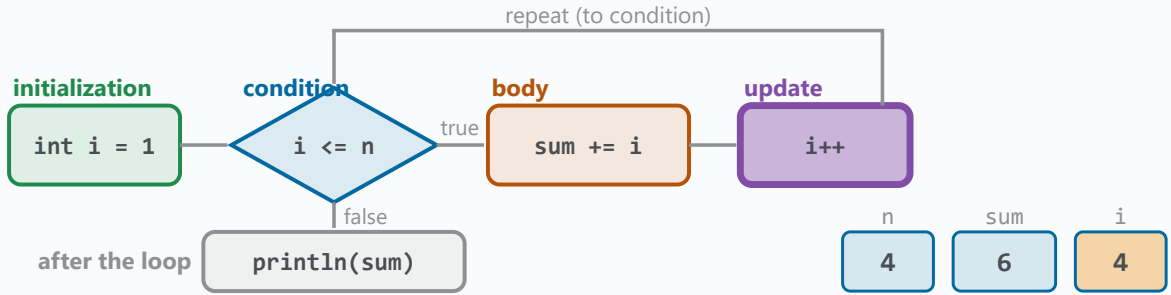
```
1 int n = 4;
2 int sum = 0;
3 for (int i = 1; i <= n; i++) {
▶     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 6` `i = 3`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Update: `i++` → i = 4. Then back to the condition.

Step 12 / 17

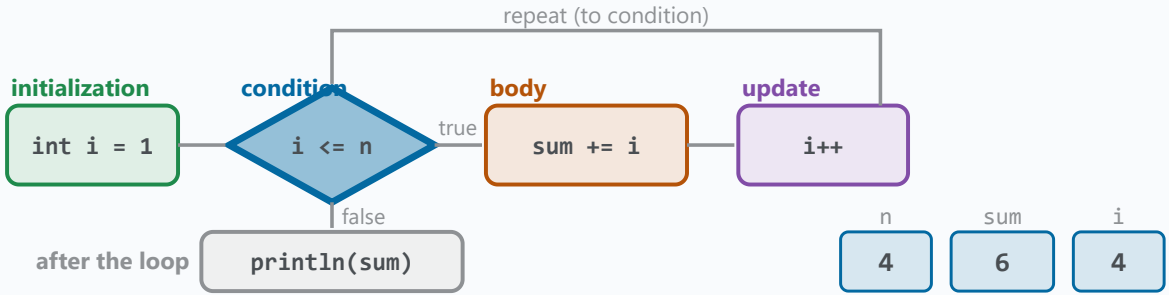
```
1 int n = 4;
2 int sum = 0;
▶ for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 6` `i = 4`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Condition: `i <= n` → 4 <= 4 → **true** → enter the body.

Step 13 / 17

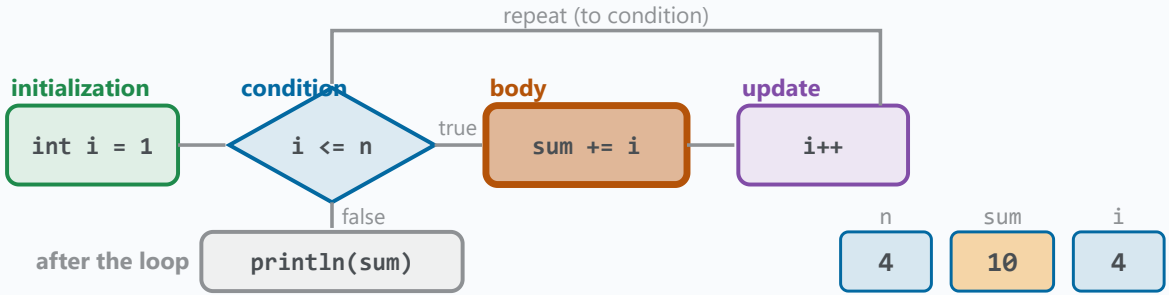
```
1 int n = 4;
2 int sum = 0;
▶ for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 6` `i = 4`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Body: sum = 6 + 4 = **10**.

Step 14 / 17

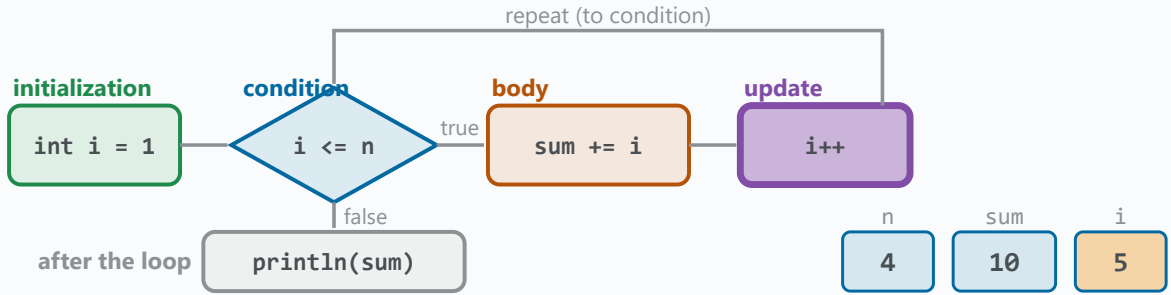
```
1 int n = 4;
2 int sum = 0;
3 for (int i = 1; i <= n; i++) {
▶     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 10` `i = 4`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Update: `i++` → i = 5. Then back to the condition.

Step 15 / 17

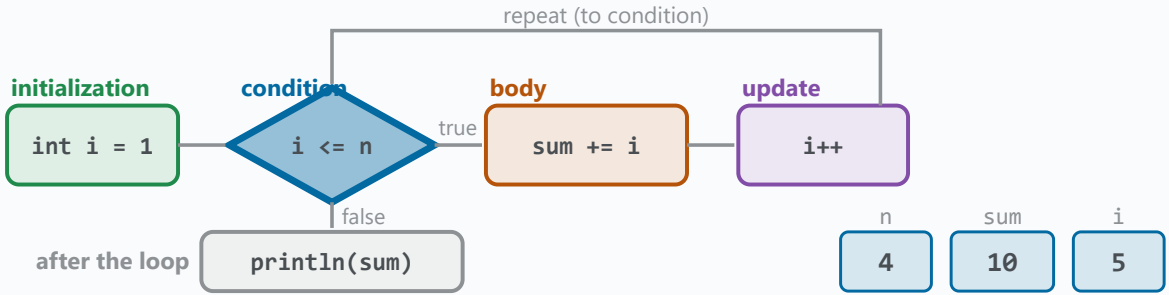
```
1 int n = 4;
2 int sum = 0;
▶ for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

n = 4 | sum = 10 | i = 5

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



Condition: `i <= n` → 5 <= 4 → **false** → the loop ends.

Step 16 / 17

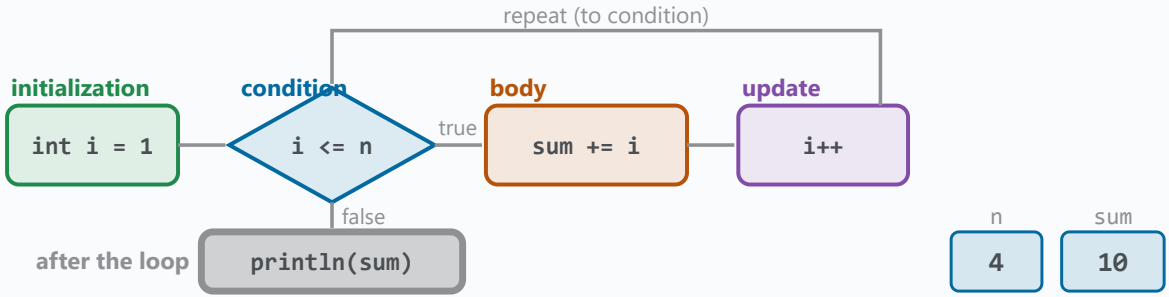
```
1 int n = 4;
2 int sum = 0;
▶ for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
6 System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 10` `i = 5`

OUTPUT

Code Trace: 1 + 2 + 3 + 4 with `for`



After the loop, `println` prints **10**. i is gone: its scope was the loop. Body ran 4 times, condition 5 times.

Step 17 / 17

```
1 int n = 4;
2 int sum = 0;
3 for (int i = 1; i <= n; i++) {
4     sum += i;
5 }
▶ System.out.println(sum);
```

VARIABLES

`n = 4` `sum = 10`

OUTPUT

```
10
```

Flexible Forms of the for Loop

Header	Runs
`for (int i = 0, j = 9; i < j; i++, j--)`	i and j together: (0,9) … (4,5), 5 passes
`for (int i = 0; i <= 10; i += 2)`	0 2 4 6 8 10
`for (int i = 5; i >= 1; i--)`	countdown: 5 4 3 2 1

```
int k = 0;
for (;;) {            // all parts empty: always true
    k++;
    if (k == 3) break;  // exit is required
}
System.out.println(k);   // 3
```

- Initialization, condition and update are each optional.
- Initialization can declare several variables of one type; the update can hold several comma-separated expressions.

LOOPS

`++i` vs. `i++`

```
for (int i = 0; i < 3; i++) System.out.print(i + " ");   // 0 1 2
for (int i = 0; i < 3; ++i) System.out.print(i + " ");   // 0 1 2
```

In the update part of a for, the expression's value is unused; both just add 1 to i, so **there is no difference**.

Start	Expression	result	after
`a = 5`	`b = a++`	b = 5	a = 6
`a = 5`	`b = ++a`	b = 6	a = 6
`idx = 0`	`d[idx++]`	10	idx = 1
`idx = 0`	`d[++idx]`	20	idx = 1

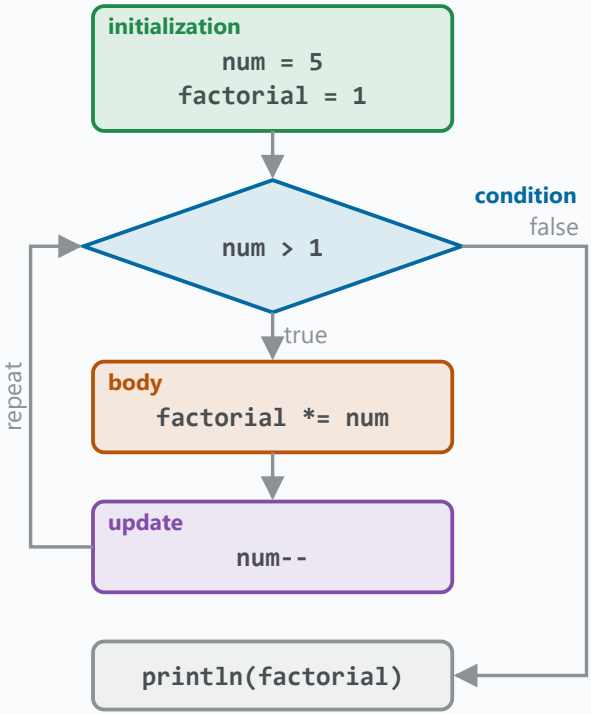
- `i++` yields the old value, then increments; `++i` increments, then yields the new value.
- In the table `d = {10, 20, 30}`; the difference shows only when the value is used. Write `i++` for counters; it reads better.

The while Loop

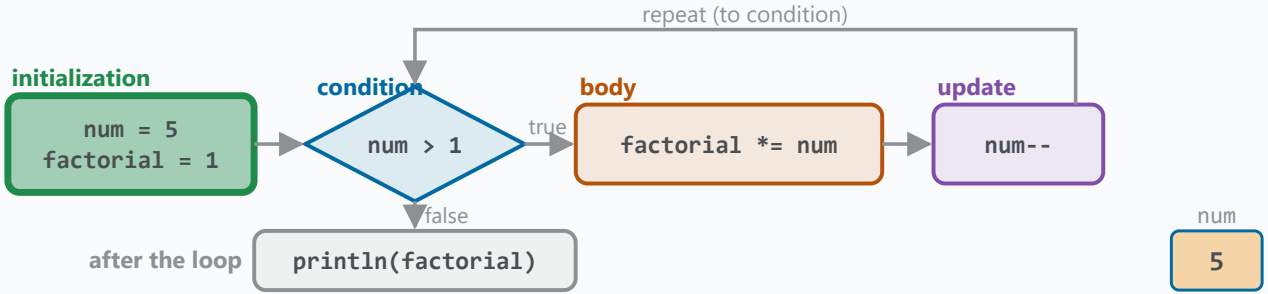
```
int num = 5;
int factorial = 1;
while (num > 1) {
    factorial *= num;
    num--;
}
System.out.println(factorial);
```

initialization condition body update

- The condition is checked **before** the body.
- If it is false at the start, the body never runs.
- Initialization goes before the loop, the update inside the body.



Code Trace: 5! with `while`



Initialization: num = 5 (we will compute 5!).

Step 1 / 16

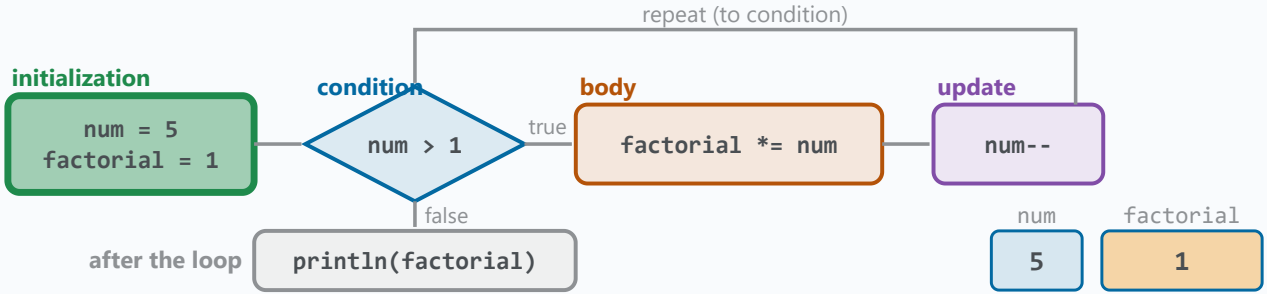
```
▶ int num = 5;
2 int factorial = 1;
3 while (num > 1) {
4     factorial *= num;
5     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 5`

OUTPUT

Code Trace: 5! with `while`



A product accumulator starts at **1** (with 0 the result would always be 0).

Step 2 / 16

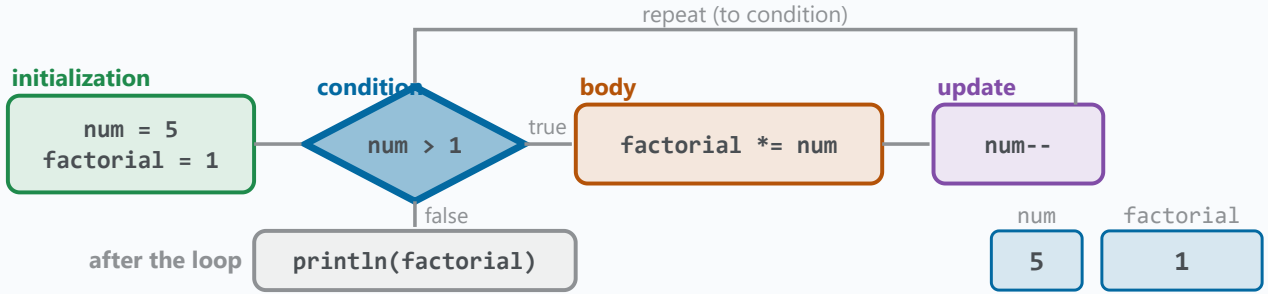
```
1 int num = 5;
▶ int factorial = 1;
3 while (num > 1) {
4     factorial *= num;
5     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 5` `factorial = 1`

OUTPUT

Code Trace: 5! with `while`



Condition: 5 > 1 → **true** → run the body.

Step 3 / 16

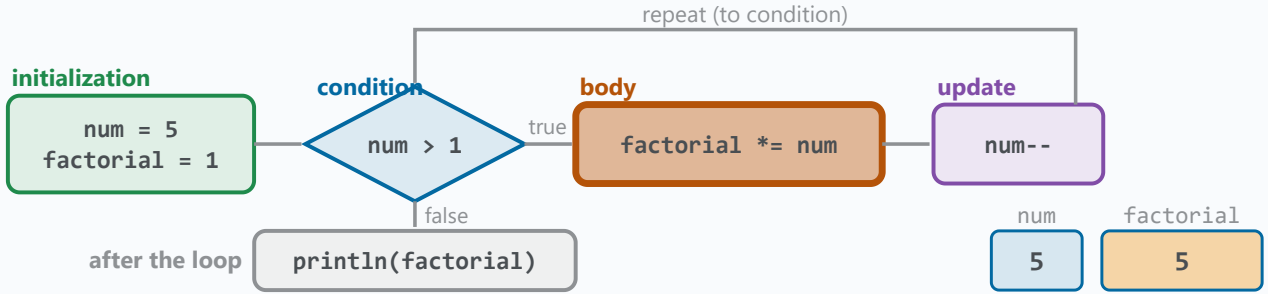
```
1 int num = 5;
2 int factorial = 1;
▶ while (num > 1) {
4     factorial *= num;
5     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 5` `factorial = 1`

OUTPUT

Code Trace: 5! with `while`



factorial = 1 × 5 = **5**.

Step 4 / 16

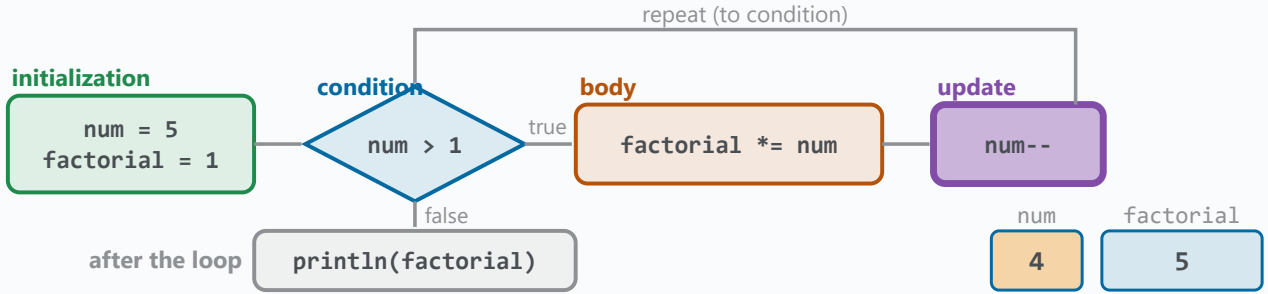
```
1 int num = 5;
2 int factorial = 1;
3 while (num > 1) {
▶     factorial *= num;
5     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 5` `factorial = 5`

OUTPUT

Code Trace: 5! with `while`



Update: num-- → **4**. The condition moves toward false.

Step 5 / 16

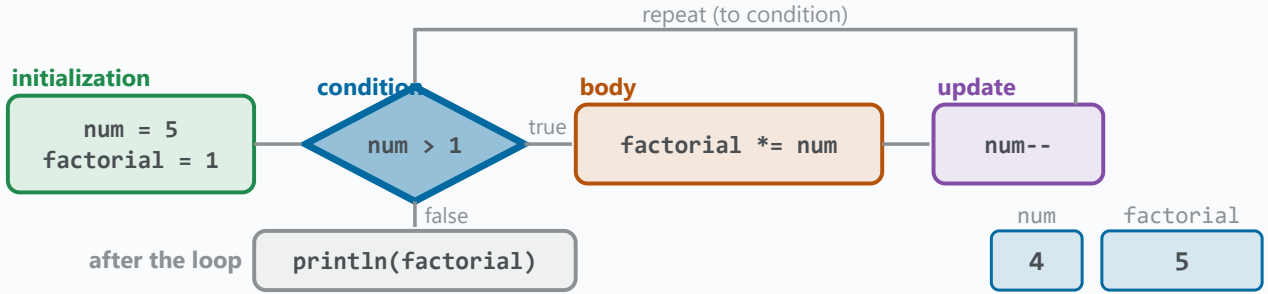
```
1 int num = 5;
2 int factorial = 1;
3 while (num > 1) {
4     factorial *= num;
▶     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 4` `factorial = 5`

OUTPUT

Code Trace: 5! with `while`



Condition: 4 > 1 → **true** → run the body.

Step 6 / 16

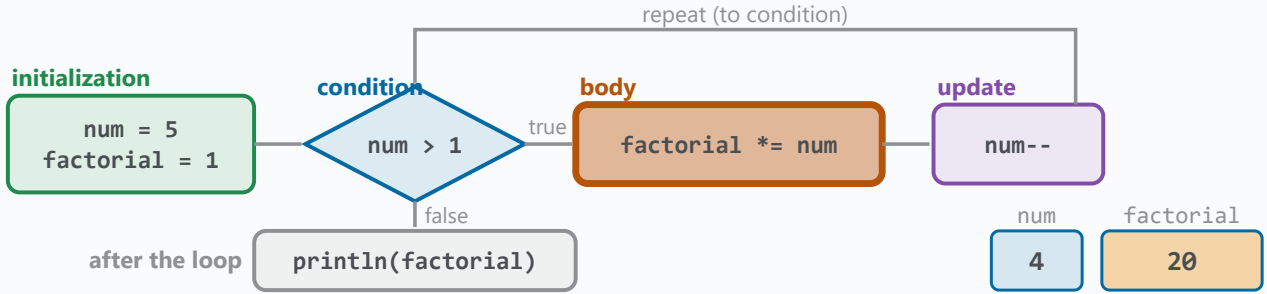
```
1 int num = 5;
2 int factorial = 1;
▶ while (num > 1) {
4     factorial *= num;
5     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 4` `factorial = 5`

OUTPUT

Code Trace: 5! with `while`



factorial = 5 × 4 = **20**.

Step 7 / 16

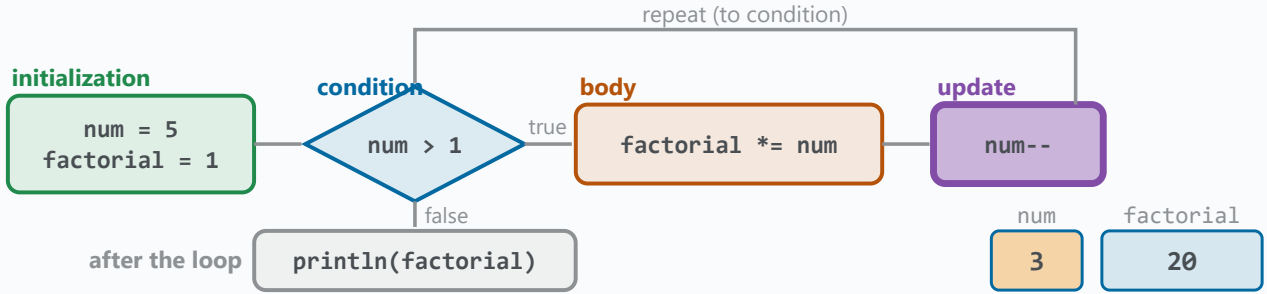
```
1 int num = 5;
2 int factorial = 1;
3 while (num > 1) {
▶     factorial *= num;
5     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 4` `factorial = 20`

OUTPUT

Code Trace: 5! with `while`



Update: num-- → **3**. The condition moves toward false.

Step 8 / 16

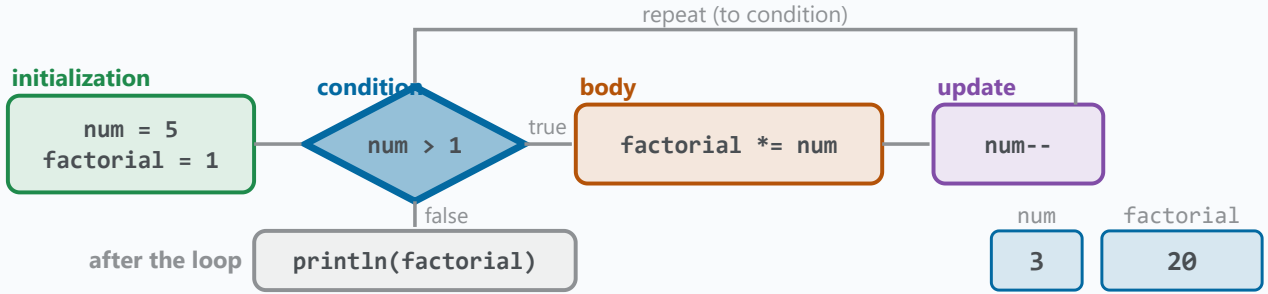
```
1 int num = 5;
2 int factorial = 1;
3 while (num > 1) {
4     factorial *= num;
▶     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 3` `factorial = 20`

OUTPUT

Code Trace: 5! with `while`



Condition: 3 > 1 → **true** → run the body.

Step 9 / 16

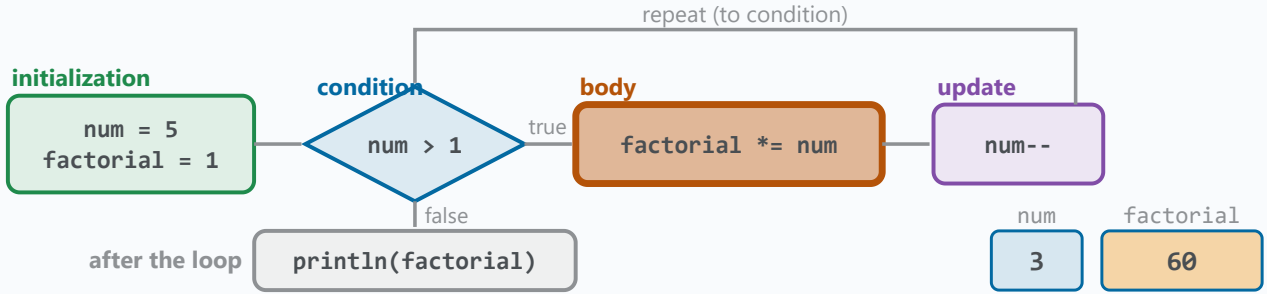
```
1 int num = 5;
2 int factorial = 1;
▶ while (num > 1) {
4     factorial *= num;
5     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 3` `factorial = 20`

OUTPUT

Code Trace: 5! with `while`



factorial = 20 × 3 = **60**.

Step 10 / 16

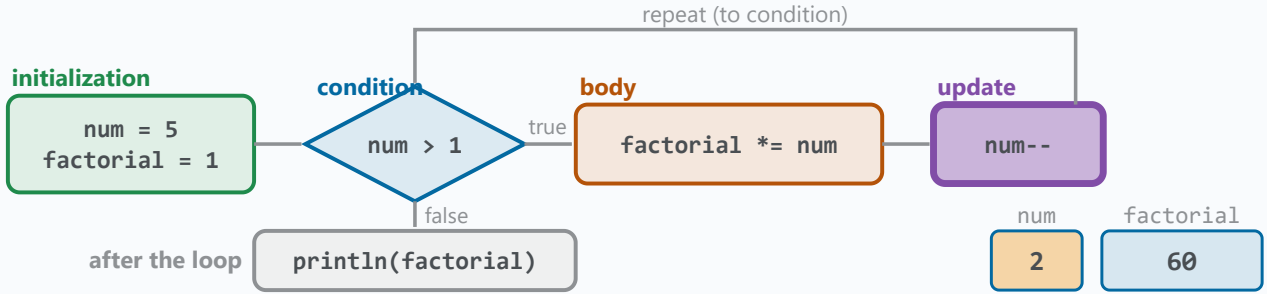
```
1 int num = 5;
2 int factorial = 1;
3 while (num > 1) {
▶     factorial *= num;
5     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 3` `factorial = 60`

OUTPUT

Code Trace: 5! with `while`



Update: num-- → **2**. The condition moves toward false.

Step 11 / 16

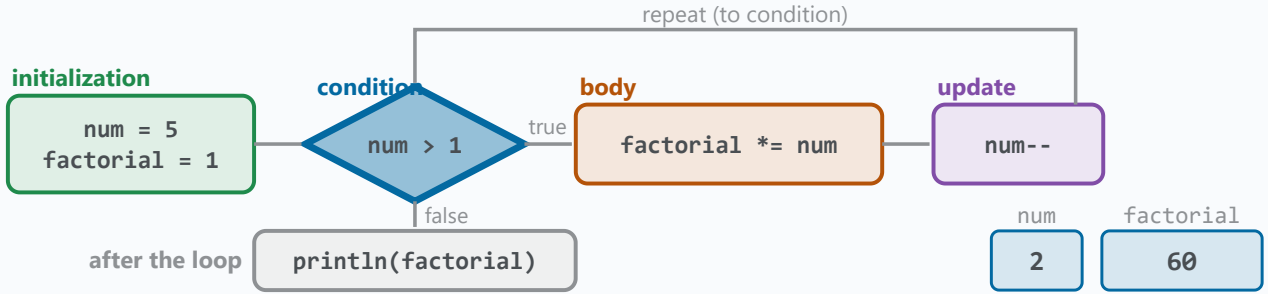
```
1 int num = 5;
2 int factorial = 1;
3 while (num > 1) {
4     factorial *= num;
▶     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 2` `factorial = 60`

OUTPUT

Code Trace: 5! with `while`



Condition: 2 > 1 → **true** → run the body.

Step 12 / 16

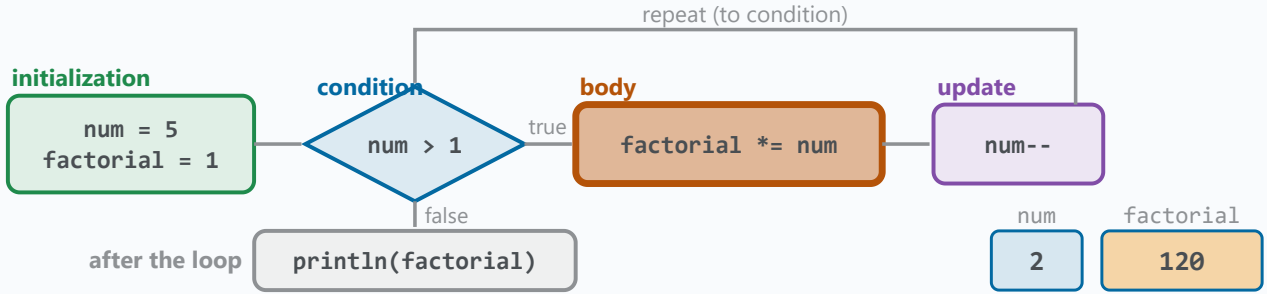
```
1 int num = 5;
2 int factorial = 1;
▶ while (num > 1) {
4     factorial *= num;
5     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 2` `factorial = 60`

OUTPUT

Code Trace: 5! with `while`



factorial = 60 × 2 = **120**.

Step 13 / 16

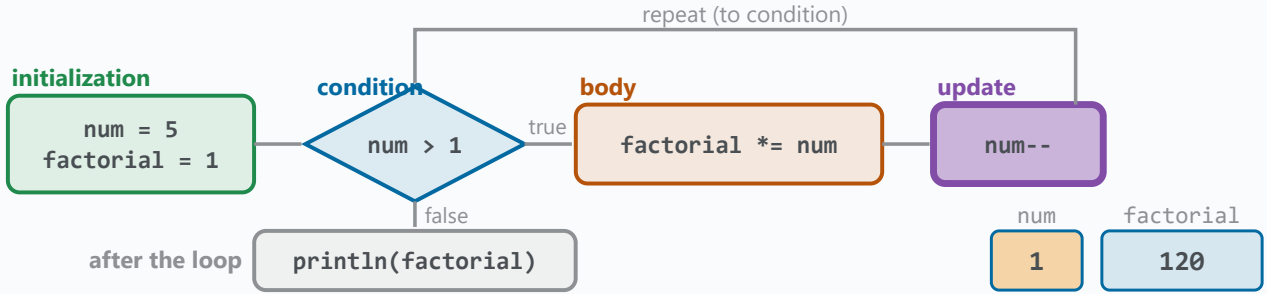
```
1 int num = 5;
2 int factorial = 1;
3 while (num > 1) {
▶     factorial *= num;
5     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 2` `factorial = 120`

OUTPUT

Code Trace: 5! with `while`



Update: num-- → **1**. The condition moves toward false.

Step 14 / 16

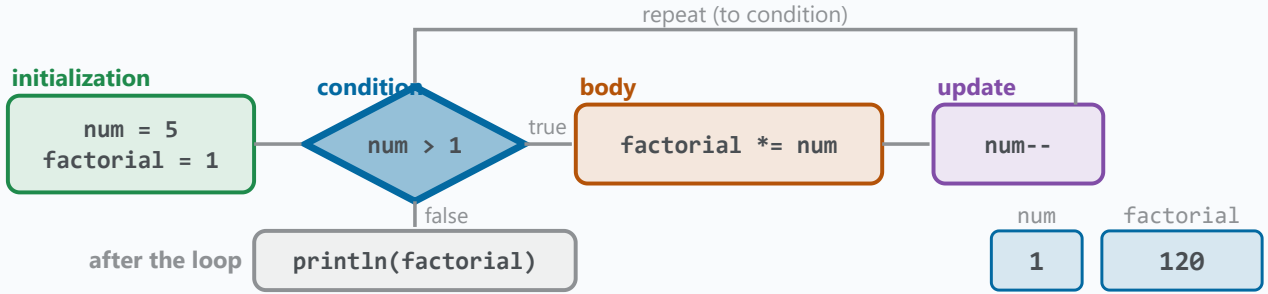
```
1 int num = 5;
2 int factorial = 1;
3 while (num > 1) {
4     factorial *= num;
▶     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 1` `factorial = 120`

OUTPUT

Code Trace: 5! with `while`



Condition: 1 > 1 → **false** → exit the loop.

Step 15 / 16

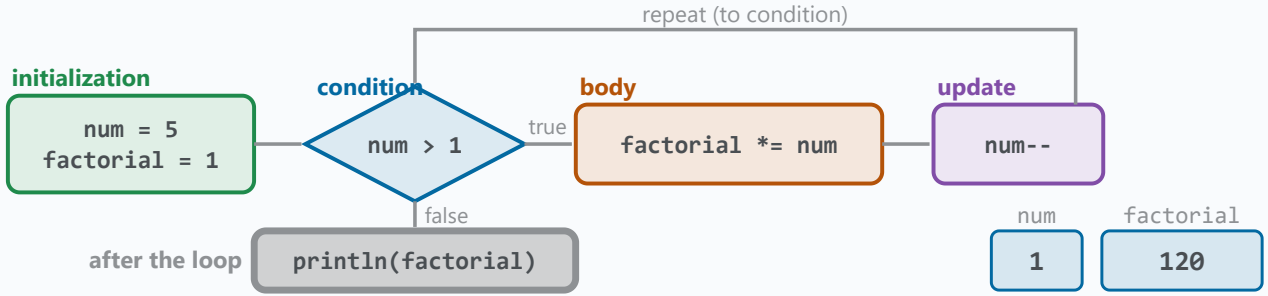
```
1 int num = 5;
2 int factorial = 1;
▶ while (num > 1) {
4     factorial *= num;
5     num--;
6 }
7 System.out.println(factorial);
```

VARIABLES

`num = 1` `factorial = 120`

OUTPUT

Code Trace: 5! with `while`



120 is printed: 5 × 4 × 3 × 2 = 120. The body ran 4 times; no need to multiply by 1.

Step 16 / 16

```
1 int num = 5;
2 int factorial = 1;
3 while (num > 1) {
4     factorial *= num;
5     num--;
6 }
▶ System.out.println(factorial);
```

VARIABLES

`num = 1` `factorial = 120`

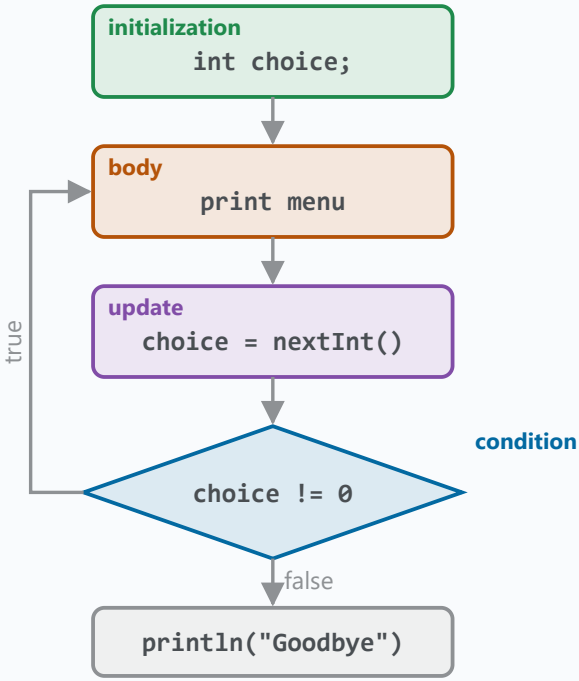
OUTPUT

```
120
```

The do-while Loop

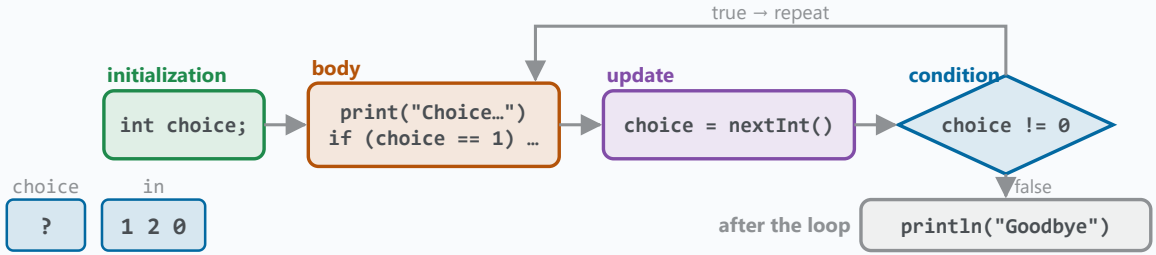
```
Scanner in = new Scanner(System.in);
int choice;
do {
   System.out.print("Choice (1/0): ");
   choice = in.nextInt();
   if (choice == 1)
      System.out.println("Hello!");
} while (choice != 0);
System.out.println("Goodbye");
```

initialization condition body update



- The condition is checked **after** the body → the body runs at least once.
- The natural choice for menus and input validation.
- Don't forget the semicolon after `while (…);`.

Code Trace: a `do-while` menu (input 1, 2, 0)



A Scanner reads from the keyboard (`import java.util.Scanner;` at the top). The user will type **1, 2, 0**.

Step 1 / 15

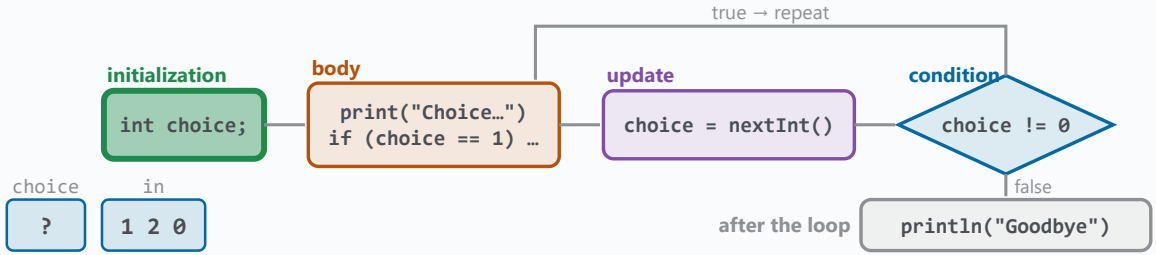
```
▶ Scanner in = new Scanner(System.in);
2 int choice;
3 do {
4   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
8 } while (choice != 0);
9 System.out.println("Goodbye");
```

VARIABLES

`in = Scanner`

OUTPUT

Code Trace: a `do-while` menu (input 1, 2, 0)



choice is declared without a value; that's fine, since the condition is first tested after the body.

Step 2 / 15

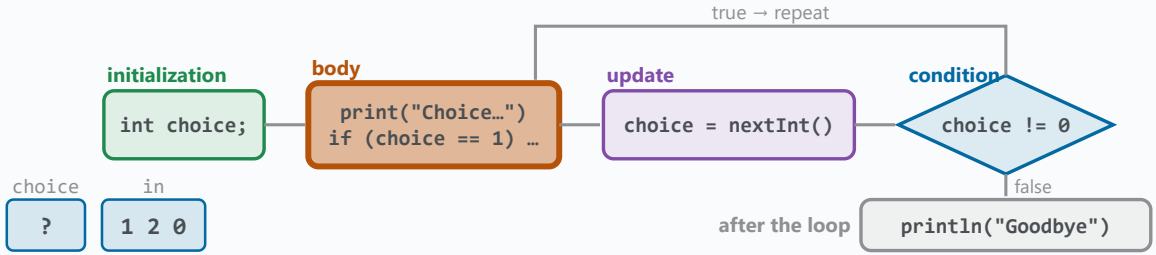
```
1 Scanner in = new Scanner(System.in);
▶ int choice;
3 do {
4   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
8 } while (choice != 0);
9 System.out.println("Goodbye");
```

VARIABLES

`in = Scanner` `choice = ?`

OUTPUT

Code Trace: a `do-while` menu (input 1, 2, 0)



Pass 1: the body runs **without checking the condition**; the menu is printed.

Step 3 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
▶   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
8 } while (choice != 0);
9 System.out.println("Goodbye");
```

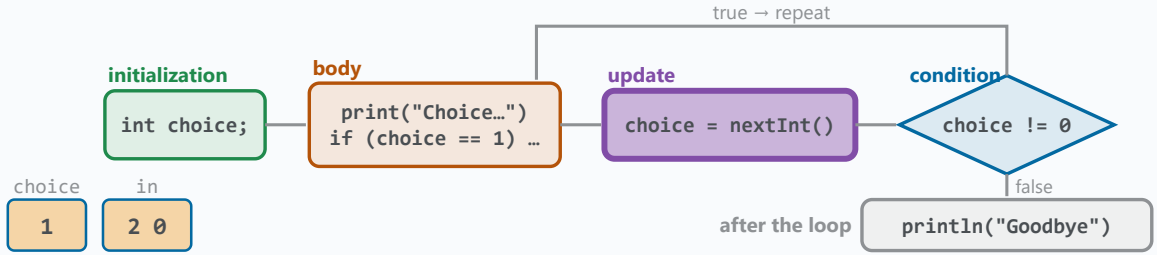
VARIABLES

`in = Scanner` `choice = ?`

OUTPUT

```
Choice (1/0):
```

Code Trace: a `do-while` menu (input 1, 2, 0)



The user types **1** → choice = 1.

Step 4 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
4   System.out.print("Choice (1/0): ");
▶   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
8 } while (choice != 0);
9 System.out.println("Goodbye");
```

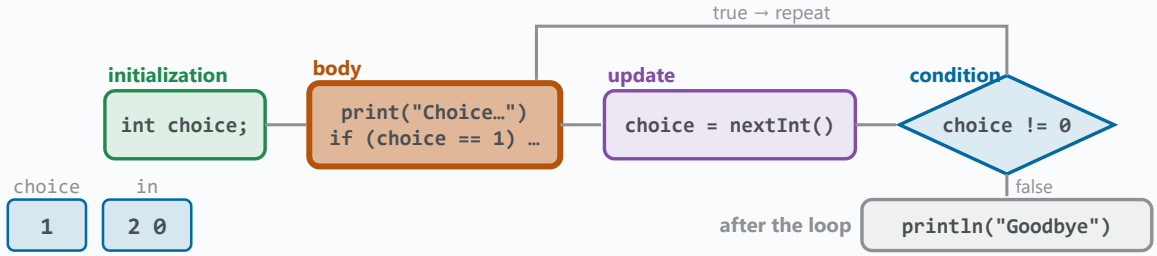
VARIABLES

`in = Scanner` `choice = 1`

OUTPUT

```
Choice (1/0): 1
```

Code Trace: a `do-while` menu (input 1, 2, 0)



choice == 1 → true: «Hello!» is printed.

Step 5 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
4   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
▶   if (choice == 1)
▶     System.out.println("Hello!");
8 } while (choice != 0);
9 System.out.println("Goodbye");
```

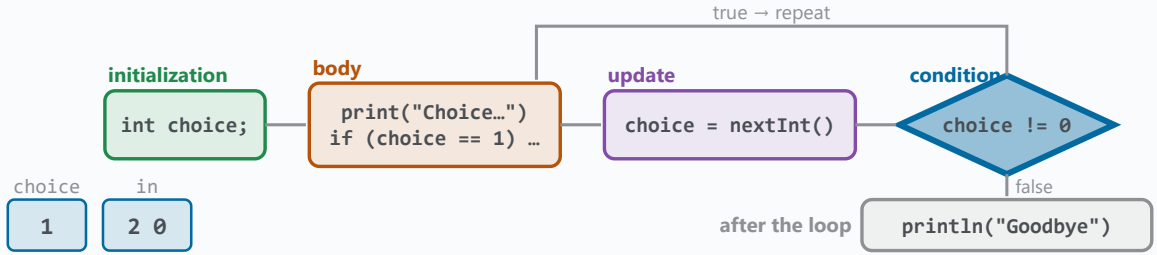
VARIABLES

`in = Scanner` `choice = 1`

OUTPUT

```
Choice (1/0): 1
Hello!
```

Code Trace: a `do-while` menu (input 1, 2, 0)



Condition: 1 != 0 → **true** → back to the top.

Step 6 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
4   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
▶ } while (choice != 0);
9 System.out.println("Goodbye");
```

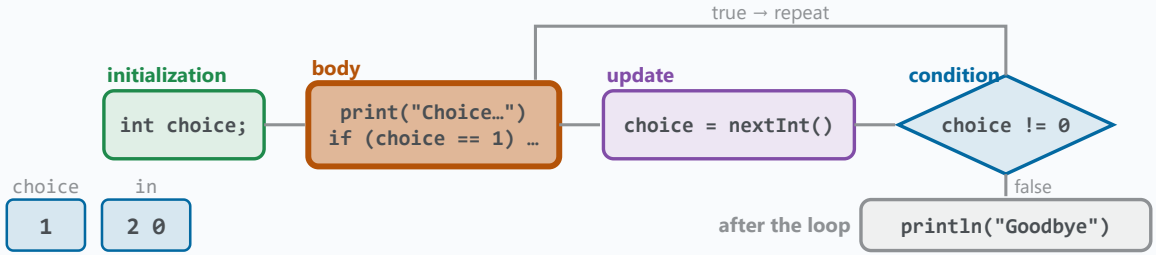
VARIABLES

`in = Scanner` `choice = 1`

OUTPUT

```
Choice (1/0): 1
Hello!
```

Code Trace: a `do-while` menu (input 1, 2, 0)



Pass 2: the body runs **without checking the condition**; the menu is printed.

Step 7 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
▶   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
8 } while (choice != 0);
9 System.out.println("Goodbye");
```

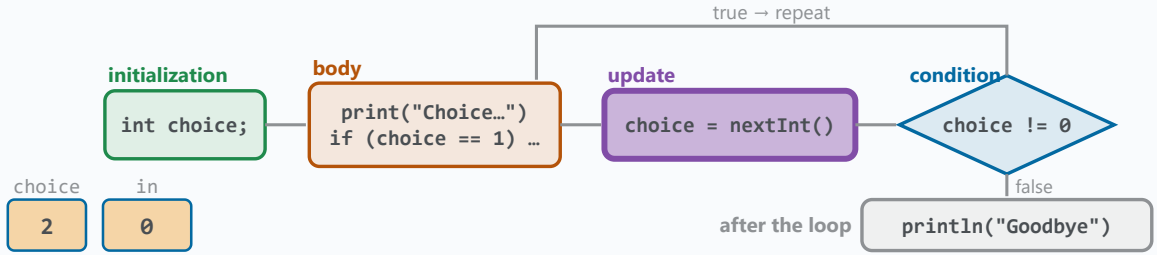
VARIABLES

`in = Scanner` `choice = 1`

OUTPUT

```
Choice (1/0): 1
Hello!
Choice (1/0):
```

Code Trace: a `do-while` menu (input 1, 2, 0)



The user types **2** → choice = 2.

Step 8 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
4   System.out.print("Choice (1/0): ");
▶   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
8 } while (choice != 0);
9 System.out.println("Goodbye");
```

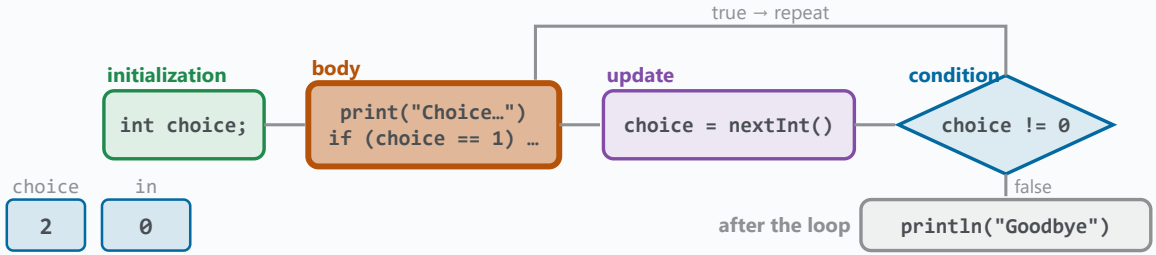
VARIABLES

`in = Scanner` `choice = 2`

OUTPUT

```
Choice (1/0): 1
Hello!
Choice (1/0): 2
```

Code Trace: a `do-while` menu (input 1, 2, 0)



choice == 1 → false: nothing happens.

Step 9 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
4   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
▶  if (choice == 1)
7     System.out.println("Hello!");
8 } while (choice != 0);
9 System.out.println("Goodbye");
```

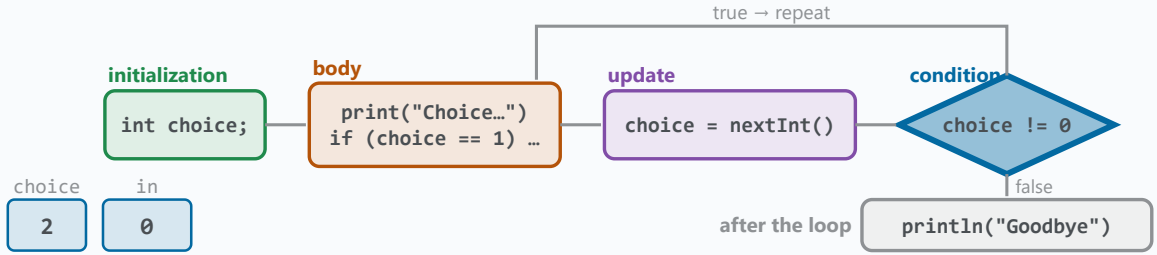
VARIABLES

`in = Scanner` `choice = 2`

OUTPUT

```
Choice (1/0): 1
Hello!
Choice (1/0): 2
```

Code Trace: a `do-while` menu (input 1, 2, 0)



Condition: 2 != 0 → **true** → back to the top.

Step 10 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
4   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
▶ } while (choice != 0);
9 System.out.println("Goodbye");
```

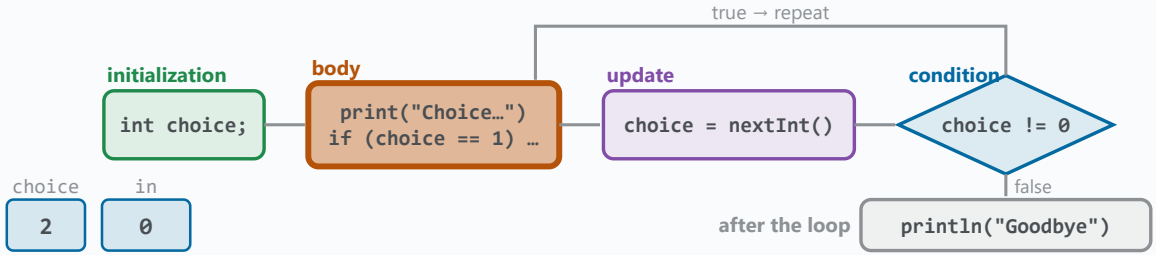
VARIABLES

`in = Scanner` `choice = 2`

OUTPUT

```
Choice (1/0): 1
Hello!
Choice (1/0): 2
```

Code Trace: a `do-while` menu (input 1, 2, 0)



Pass 3: the body runs **without checking the condition**; the menu is printed.

Step 11 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
▶   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
8 } while (choice != 0);
9 System.out.println("Goodbye");
```

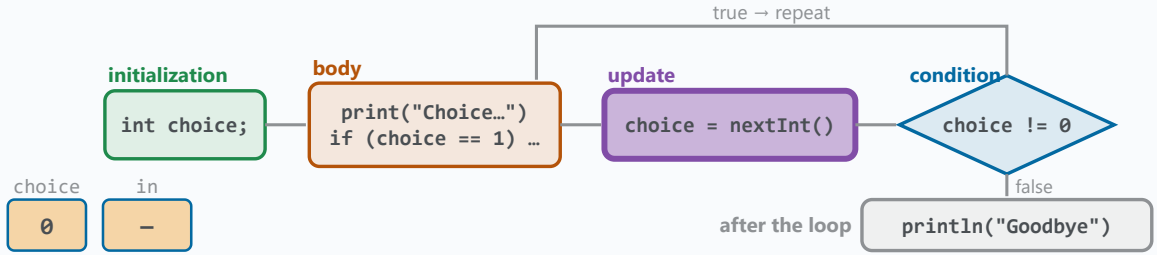
VARIABLES

`in = Scanner` `choice = 2`

OUTPUT

```
Choice (1/0): 1
Hello!
Choice (1/0): 2
Choice (1/0):
```

Code Trace: a `do-while` menu (input 1, 2, 0)



The user types **0** → choice = 0.

Step 12 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
4   System.out.print("Choice (1/0): ");
▶   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
8 } while (choice != 0);
9 System.out.println("Goodbye");
```

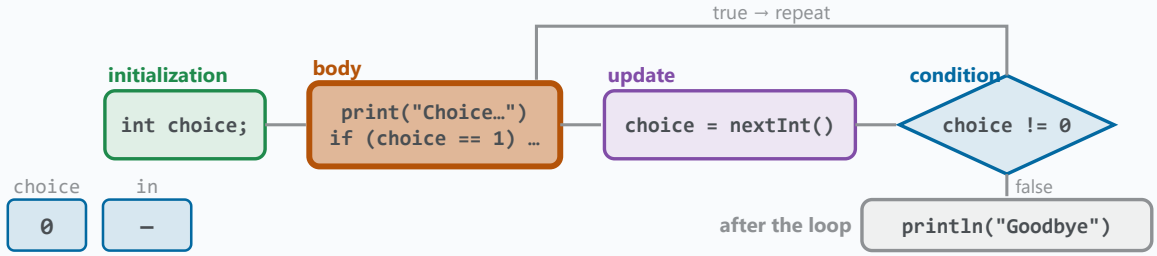
VARIABLES

`in = Scanner` `choice = 0`

OUTPUT

```
Choice (1/0): 1
Hello!
Choice (1/0): 2
Choice (1/0): 0
```

Code Trace: a `do-while` menu (input 1, 2, 0)



choice == 1 → false: nothing happens.

Step 13 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
4   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
▶  if (choice == 1)
7     System.out.println("Hello!");
8 } while (choice != 0);
9 System.out.println("Goodbye");
```

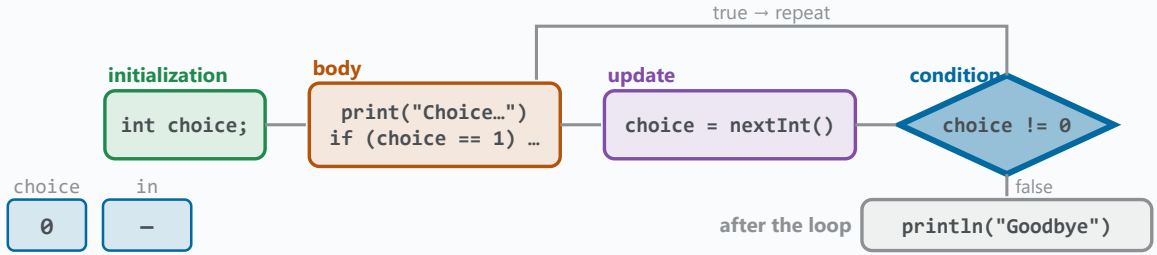
VARIABLES

`in = Scanner` `choice = 0`

OUTPUT

```
Choice (1/0): 1
Hello!
Choice (1/0): 2
Choice (1/0): 0
```

Code Trace: a `do-while` menu (input 1, 2, 0)



Condition: 0 != 0 → **false** → the loop ends.

Step 14 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
4   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
▶ } while (choice != 0);
9 System.out.println("Goodbye");
```

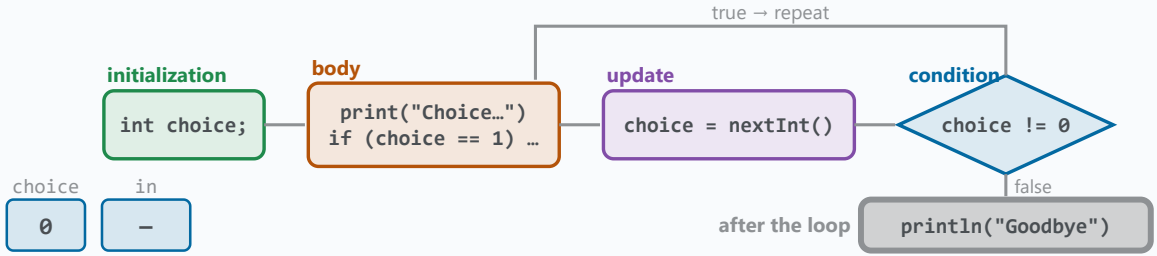
VARIABLES

`in = Scanner` `choice = 0`

OUTPUT

```
Choice (1/0): 1
Hello!
Choice (1/0): 2
Choice (1/0): 0
```

Code Trace: a `do-while` menu (input 1, 2, 0)



The body ran 3 times and the condition was tested 3 times (a while would test it once more than the body runs).

Step 15 / 15

```
1 Scanner in = new Scanner(System.in);
2 int choice;
3 do {
4   System.out.print("Choice (1/0): ");
5   choice = in.nextInt();
6   if (choice == 1)
7     System.out.println("Hello!");
8 } while (choice != 0);
▶ System.out.println("Goodbye");
```

VARIABLES

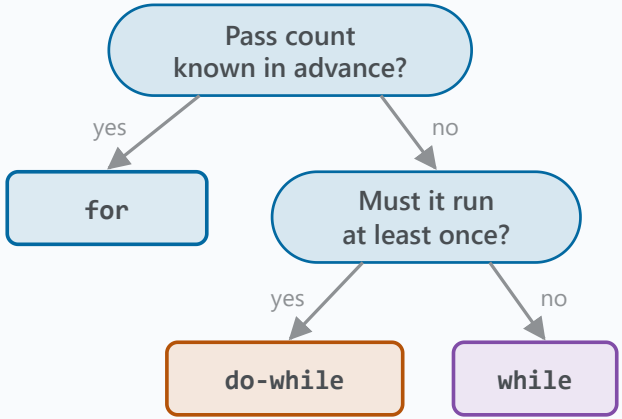
`in = Scanner` `choice = 0`

OUTPUT

```
Choice (1/0): 1
Hello!
Choice (1/0): 2
Choice (1/0): 0
Goodbye
```

Choosing a Loop

Situation	Choice
Count from 1 to n, repeat n times	for
Process digits until the number is 0	while
Show a menu, ask until the input is valid	do-while
Add until the user types “0”	while / do-while

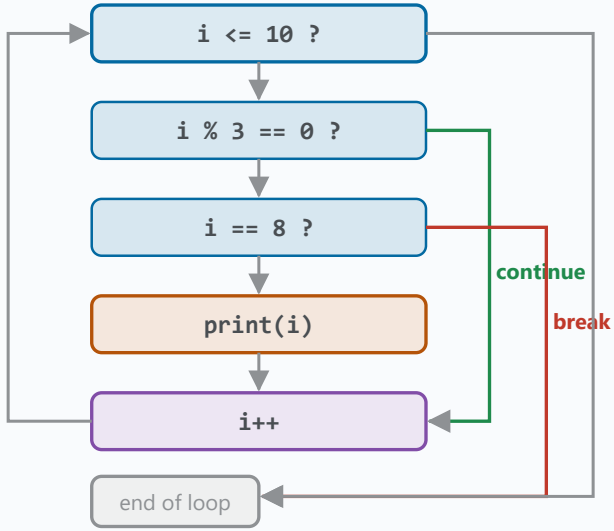


Each can be rewritten as the others; choose for readability.

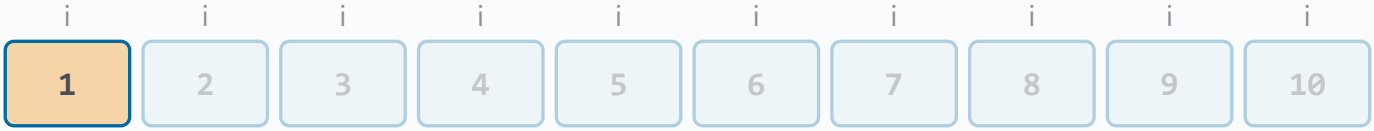
break and continue

```
for (int i = 1; i <= 10; i++) {
    if (i % 3 == 0) continue;
    if (i == 8) break;
    System.out.print(i + " ");
}
System.out.println("| done");
```

- **continue**: skips the rest of the pass and goes to the **update** (i++).
- **break**: ends the nearest enclosing loop at once.
- Overuse makes the flow hard to follow.



Code Trace: `break` and `continue`



i = 1: 1 % 3 = 1 → false; 1 == 8 → false. Both checks pass.

Step 1 / 14

```
1 for (int i = 1; i <= 10; i++) {
▶     if (i % 3 == 0) continue;
▶     if (i == 8) break;
4     System.out.print(i + " ");
5 }
6 System.out.println("| done");
```

VARIABLES

`i = 1`

OUTPUT

Code Trace: `break` and `continue`



1 is printed.

Step 2 / 14

```
1 for (int i = 1; i <= 10; i++) {
2     if (i % 3 == 0) continue;
3     if (i == 8) break;
▶     System.out.print(i + " ");
5 }
6 System.out.println("| done");
```

VARIABLES

`i = 1`

OUTPUT

```
1
```

Code Trace: `break` and `continue`



i = 2: 2 % 3 = 2 → false; 2 == 8 → false. Both checks pass.

Step 3 / 14

```
1 for (int i = 1; i <= 10; i++) {
▶     if (i % 3 == 0) continue;
▶     if (i == 8) break;
4     System.out.print(i + " ");
5 }
6 System.out.println("| done");
```

VARIABLES

`i = 2`

OUTPUT

```
1
```

Code Trace: `break` and `continue`



2 is printed.

Step 4 / 14

```
1 for (int i = 1; i <= 10; i++) {
2     if (i % 3 == 0) continue;
3     if (i == 8) break;
▶     System.out.print(i + " ");
5 }
6 System.out.println("| done");
```

VARIABLES

`i = 2`

OUTPUT

```
1 2
```

Code Trace: `break` and `continue`



i = 3: 3 % 3 == 0 → **true** → **continue**: print is skipped, straight to i++.

Step 5 / 14

```
1 for (int i = 1; i <= 10; i++) {
▶       if (i % 3 == 0) continue;
3       if (i == 8) break;
4       System.out.print(i + " ");
5   }
6   System.out.println("| done");
```

VARIABLES

`i = 3`

OUTPUT

```
1 2
```

Code Trace: `break` and `continue`



i = 4: 4 % 3 = 1 → false; 4 == 8 → false. Both checks pass.

Step 6 / 14

```
1 for (int i = 1; i <= 10; i++) {
▶     if (i % 3 == 0) continue;
▶     if (i == 8) break;
4     System.out.print(i + " ");
5 }
6 System.out.println("| done");
```

VARIABLES

`i = 4`

OUTPUT

```
1 2
```

Code Trace: `break` and `continue`



4 is printed.

Step 7 / 14

```
1 for (int i = 1; i <= 10; i++) {
2     if (i % 3 == 0) continue;
3     if (i == 8) break;
▶     System.out.print(i + " ");
5 }
6 System.out.println("| done");
```

VARIABLES

`i = 4`

OUTPUT

```
1 2 4
```

Code Trace: `break` and `continue`



i = 5: 5 % 3 = 2 → false; 5 == 8 → false. Both checks pass.

Step 8 / 14

```
1  for (int i = 1; i <= 10; i++) {
▶      if (i % 3 == 0) continue;
▶      if (i == 8) break;
4      System.out.print(i + " ");
5  }
6  System.out.println("| done");
```

VARIABLES

`i = 5`

OUTPUT

```
1 2 4
```

Code Trace: `break` and `continue`



5 is printed.

Step 9 / 14

```
1 for (int i = 1; i <= 10; i++) {
2     if (i % 3 == 0) continue;
3     if (i == 8) break;
▶     System.out.print(i + " ");
5 }
6 System.out.println("| done");
```

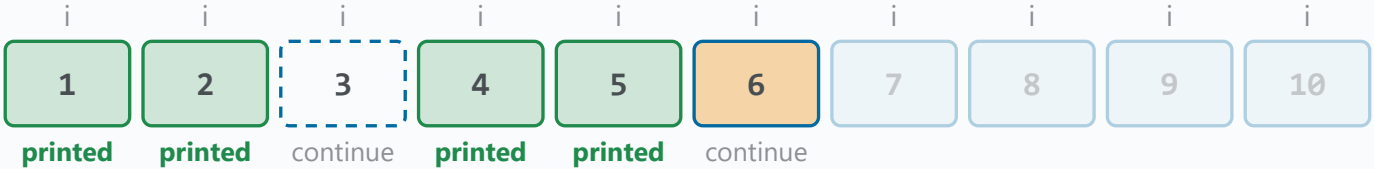
VARIABLES

`i = 5`

OUTPUT

```
1 2 4 5
```

Code Trace: `break` and `continue`



i = 6: 6 % 3 == 0 → **true** → **continue**: print is skipped, straight to i++.

Step 10 / 14

```
1 for (int i = 1; i <= 10; i++) {
▶       if (i % 3 == 0) continue;
3       if (i == 8) break;
4       System.out.print(i + " ");
5   }
6   System.out.println("| done");
```

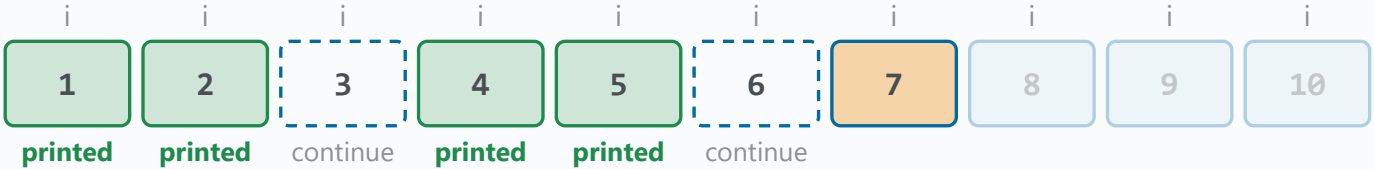
VARIABLES

`i = 6`

OUTPUT

```
1 2 4 5
```

Code Trace: `break` and `continue`



i = 7: 7 % 3 = 1 → false; 7 == 8 → false. Both checks pass.

Step 11 / 14

```
1 for (int i = 1; i <= 10; i++) {
▶     if (i % 3 == 0) continue;
▶     if (i == 8) break;
4     System.out.print(i + " ");
5 }
6 System.out.println("| done");
```

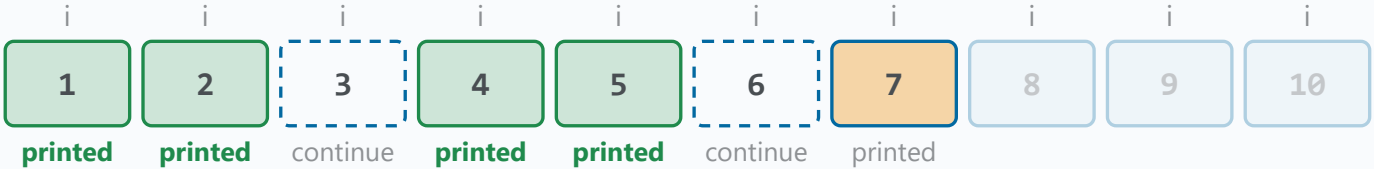
VARIABLES

`i = 7`

OUTPUT

```
1 2 4 5
```

Code Trace: `break` and `continue`



7 is printed.

Step 12 / 14

```
1 for (int i = 1; i <= 10; i++) {
2     if (i % 3 == 0) continue;
3     if (i == 8) break;
▶     System.out.print(i + " ");
5 }
6 System.out.println("| done");
```

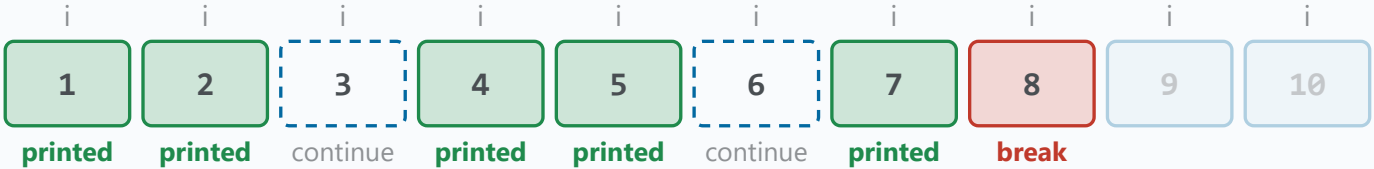
VARIABLES

`i = 7`

OUTPUT

```
1 2 4 5 7
```

Code Trace: `break` and `continue`



i = 8: 8 % 3 = 2 → false; 8 == 8 → **true** → **break**: the loop ends here; 9 and 10 are never reached.


```
1 for (int i = 1; i <= 10; i++) {
▶     if (i % 3 == 0) continue;
▶     if (i == 8) break;
4     System.out.print(i + " ");
5 }
6 System.out.println("| done");
```

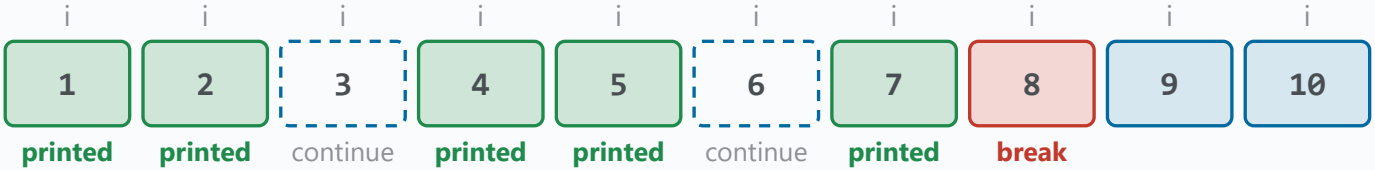
VARIABLES

`i = 8`

OUTPUT

```
1 2 4 5 7
```

Code Trace: `break` and `continue`



The loop ended with break; execution continues after the loop.

Step 14 / 14

```
1 for (int i = 1; i <= 10; i++) {
2     if (i % 3 == 0) continue;
3     if (i == 8) break;
4     System.out.print(i + " ");
5 }
▶ System.out.println("| done");
```

VARIABLES

`i = (gone)`

OUTPUT

```
1 2 4 5 7 | done
```

Labeled `break outer`: leave both loops at once

i = 1, j = 1: 1·1 = 1 ≠ 4, `1 1` is printed.

Step 1 / 6

```
1 outer:
2 for (int i = 1; i <= 3; i++) {
3     for (int j = 1; j <= 3; j++) {
4         if (i * j == 4) break outer;
▶         System.out.println(i + " " + j);
6     }
7 }
8 System.out.println("done");
```

VARIABLES

`i = 1` `j = 1`

OUTPUT

```
1 1
```

Labeled `break outer`: leave both loops at once

j = 2: 1·2 = 2 ≠ 4 → `1 2` is printed.

Step 2 / 6

```
1 outer:
2 for (int i = 1; i <= 3; i++) {
3     for (int j = 1; j <= 3; j++) {
4         if (i * j == 4) break outer;
▶         System.out.println(i + " " + j);
6     }
7 }
8 System.out.println("done");
```

VARIABLES

`i = 1` `j = 2`

OUTPUT

```
1 1
1 2
```

Labeled `break outer`: leave both loops at once

j = 3: 1·3 = 3 ≠ 4 → `1 3` is printed; the inner loop ends.

Step 3 / 6

```
1 outer:
2 for (int i = 1; i <= 3; i++) {
3     for (int j = 1; j <= 3; j++) {
4         if (i * j == 4) break outer;
▶         System.out.println(i + " " + j);
6     }
7 }
8 System.out.println("done");
```

VARIABLES

`i = 1` `j = 3`

OUTPUT

```
1 1
1 2
1 3
```

Labeled `break outer`: leave both loops at once

i = 2; the inner loop restarts: j = 1, 2·1 = 2 ≠ 4 → `2 1` is printed.

Step 4 / 6

```
1 outer:
2 for (int i = 1; i <= 3; i++) {
3     for (int j = 1; j <= 3; j++) {
4         if (i * j == 4) break outer;
▶         System.out.println(i + " " + j);
6     }
7 }
8 System.out.println("done");
```

VARIABLES

`i = 2` `j = 1`

OUTPUT

```
1 1
1 2
1 3
2 1
```

Labeled `break outer`: leave both loops at once

i = 2, j = 2: 2·2 = 4 → `break outer`. Without printing, control leaves **both loops**; i = 3 is never tried.

Step 5 / 6

```
1 outer:
2 for (int i = 1; i <= 3; i++) {
3     for (int j = 1; j <= 3; j++) {
▶         if (i * j == 4) break outer;
5         System.out.println(i + " " + j);
6     }
7 }
8 System.out.println("done");
```

VARIABLES

`i = 2` `j = 2`

OUTPUT

```
1 1
1 2
1 3
2 1
```

Labeled `break outer`: leave both loops at once

A labeled break jumps to the line after the outer loop: `done` is printed. Output: 4 number lines + done.

Step 6 / 6

```
1 outer:
2 for (int i = 1; i <= 3; i++) {
3     for (int j = 1; j <= 3; j++) {
4         if (i * j == 4) break outer;
5         System.out.println(i + " " + j);
6     }
7 }
▶ System.out.println("done");
```

VARIABLES

`i = (gone)` `j = (gone)`

OUTPUT

```
1 1
1 2
1 3
2 1
done
```

Comparison: a plain `break` ends only the inner loop

i = 1: 1·j ≠ 4 for j = 1, 2, 3; `1 1`, `1 2`, `1 3` are printed.

Step 1 / 6

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
3         if (i * j == 4) break;
▶         System.out.println(i + " " + j);
5     }
6 }
7 System.out.println("done");
```

VARIABLES

`i = 1` `j = 3`

OUTPUT

```
1 1
1 2
1 3
```

Comparison: a plain `break` ends only the inner loop

i = 2, j = 1: 2·1 = 2 ≠ 4 → `2 1` is printed.

Step 2 / 6

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
3         if (i * j == 4) break;
▶         System.out.println(i + " " + j);
5     }
6 }
7 System.out.println("done");
```

VARIABLES

`i = 2` `j = 1`

OUTPUT

```
1 1
1 2
1 3
2 1
```

Comparison: a plain `break` ends only the inner loop

i = 2, j = 2: 2·2 = 4 → a plain `break` ends only the **inner** loop; the outer loop goes on.

Step 3 / 6

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
▶         if (i * j == 4) break;
4         System.out.println(i + " " + j);
5     }
6 }
7 System.out.println("done");
```

VARIABLES

`i = 2` `j = 2`

OUTPUT

```
1 1
1 2
1 3
2 1
```

Comparison: a plain `break` ends only the inner loop

i++ → 3; the inner loop restarts: j = 1 → `3 1` is printed.

Step 4 / 6

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
3         if (i * j == 4) break;
▶         System.out.println(i + " " + j);
5     }
6 }
7 System.out.println("done");
```

VARIABLES

`i = 3` `j = 1`

OUTPUT

```
1 1
1 2
1 3
2 1
3 1
```

Comparison: a plain `break` ends only the inner loop

j = 2 and j = 3: 3·2 = 6, 3·3 = 9 ≠ 4 → `3 2` and `3 3` are printed.

Step 5 / 6

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
3         if (i * j == 4) break;
▶         System.out.println(i + " " + j);
5     }
6 }
7 System.out.println("done");
```

VARIABLES

`i = 3` `j = 3`

OUTPUT

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

Comparison: a plain `break` ends only the inner loop

The outer loop ends too and `done` is printed. A plain break gives 7 number lines, a labeled break 4.

Step 6 / 6

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
3         if (i * j == 4) break;
4         System.out.println(i + " " + j);
5     }
6 }
▶ System.out.println("done");
```

VARIABLES

`i = (gone)` `j = (gone)`

OUTPUT

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
done
```

Infinite Loops




```
for (int i = 0; i < 5; i--) → i = 0, -1, -2, … always true

while (i < 5) { … } // no i++ → i stays 0
```

- If the condition never becomes false, the loop never ends: the program seems “frozen”.
- Note: with `i--` the `int` eventually wraps around and the loop stops after about 2.1 billion passes; in practice it is infinite.
- The counter may move the wrong way or never be updated at all.
- Deliberate `while (true)` loops still need a `break` condition inside. If a program hangs, stop it with Ctrl+C.

LOOPS · STEP THROUGH THE CODE

The double Counter Trap: `x += 0.1`

Start: **n = 0**. The goal: step by 0.1 and loop 10 times.

Step 1 / 9

```
▶ int n = 0;
2 for (double x = 0; x < 1.0; x += 0.1) {
3     n++;
4 }
5 System.out.println(n);
```

VARIABLES

`n = 0`

OUTPUT

With `x != 1.0` the loop would never end, since x never equals 1.0 exactly. Use an `int` counter, not a real number.

The double Counter Trap: `x += 0.1`

`x = 0.0`, condition `0.0 < 1.0` → **true**.

Step 2 / 9

```
1 int n = 0;
▶ for (double x = 0; x < 1.0; x += 0.1) {
3     n++;
4 }
5 System.out.println(n);
```

VARIABLES

`n = 0` `x = 0.0`

OUTPUT

With `x != 1.0` the loop would never end, since x never equals 1.0 exactly. Use an `int` counter, not a real number.

The double Counter Trap: `x += 0.1`

Body: **n = 1**.

Step 3 / 9

```
1 int n = 0;
2 for (double x = 0; x < 1.0; x += 0.1) {
      n++;
4 }
5 System.out.println(n);
```

VARIABLES

n = 1 | x = 0.0

OUTPUT

With `x != 1.0` the loop would never end, since x never equals 1.0 exactly. Use an `int` counter, not a real number.

The double Counter Trap: `x += 0.1`

Update: x = 0.1. 0.1 has no exact binary representation; small errors accumulate.

Step 4 / 9

```
1 int n = 0;
▶ for (double x = 0; x < 1.0; x += 0.1) {
3     n++;
4 }
5 System.out.println(n);
```

VARIABLES

`n = 1` `x = 0.1`

OUTPUT

With `x != 1.0` the loop would never end, since x never equals 1.0 exactly. Use an `int` counter, not a real number.

The double Counter Trap: `x += 0.1`

End of pass 10 (8 passes skipped here): **n = 10**, x = 0.8999999999999999.

Step 5 / 9

```
1 int n = 0;
2 for (double x = 0; x < 1.0; x += 0.1) {
      n++;
4 }
5 System.out.println(n);
```

VARIABLES

`n = 10` `x = 0.8999999999999999`

OUTPUT

With `x != 1.0` the loop would never end, since x never equals 1.0 exactly. Use an `int` counter, not a real number.

The double Counter Trap: `x += 0.1`

Update: x = 0.9999999999999999, **not** exactly 1.0. The condition is true: one extra pass.

Step 6 / 9

```
1 int n = 0;
▶ for (double x = 0; x < 1.0; x += 0.1) {
3     n++;
4 }
5 System.out.println(n);
```

VARIABLES

n = 10 | x = 0.9999999999999999

OUTPUT

With `x != 1.0` the loop would never end, since x never equals 1.0 exactly. Use an `int` counter, not a real number.

The double Counter Trap: `x += 0.1`

Pass 11: **n = 11**.

Step 7 / 9

```
1 int n = 0;
2 for (double x = 0; x < 1.0; x += 0.1) {
▶     n++;
4 }
5 System.out.println(n);
```

VARIABLES

n = 11 | x = 0.9999999999999999

OUTPUT

With `x != 1.0` the loop would never end, since x never equals 1.0 exactly. Use an `int` counter, not a real number.

The double Counter Trap: `x += 0.1`

x = 1.0999999999999999, the condition is false → the loop ends.

Step 8 / 9

```
1 int n = 0;
▶ for (double x = 0; x < 1.0; x += 0.1) {
3     n++;
4 }
5 System.out.println(n);
```

VARIABLES

n = 11 | x = 1.0999999999999999

OUTPUT

With `x != 1.0` the loop would never end, since x never equals 1.0 exactly. Use an `int` counter, not a real number.

The double Counter Trap: `x += 0.1`

11 is printed; 10 was expected.

Step 9 / 9

```
1 int n = 0;
2 for (double x = 0; x < 1.0; x += 0.1) {
3     n++;
4 }
▶ System.out.println(n);
```

VARIABLES

`n = 11` `x = 1.0999999999999999`

OUTPUT

```
11
```

With `x != 1.0` the loop would never end, since x never equals 1.0 exactly. Use an `int` counter, not a real number.

VERIFICATION

Loop Invariants

- A property that holds at the **start** of every pass.
- In the sum loop: `sum = 1 + 2 + … + (i − 1)`.
- When the loop ends (i = n + 1), the invariant tells us the result is correct.

start of pass: i = 1



No number processed yet: sum = 0 (the empty sum). The invariant holds from the start.

Step 1 / 5

Loop Invariants

- A property that holds at the **start** of every pass.
- In the sum loop: `sum = 1 + 2 + … + (i − 1)`.
- When the loop ends (i = n + 1), the invariant tells us the result is correct.

start of pass: i = 2



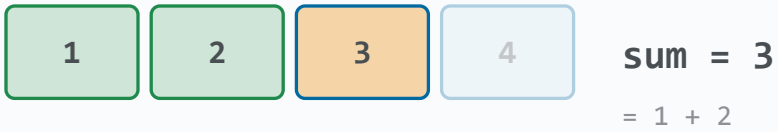
At the start of pass i = 2, sum holds **1 … 1** added up: 1.

VERIFICATION

Loop Invariants

- A property that holds at the **start** of every pass.
- In the sum loop: `sum = 1 + 2 + … + (i − 1)`.
- When the loop ends (i = n + 1), the invariant tells us the result is correct.

start of pass: i = 3

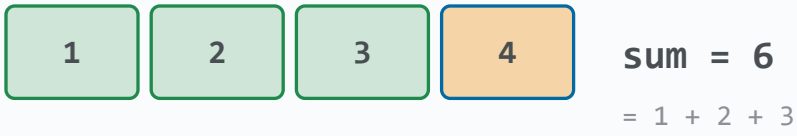


At the start of pass i = 3, sum holds **1 … 2** added up: 3.

Loop Invariants

- A property that holds at the **start** of every pass.
- In the sum loop: `sum = 1 + 2 + … + (i − 1)`.
- When the loop ends (i = n + 1), the invariant tells us the result is correct.

start of pass: i = 4



At the start of pass i = 4, sum holds **1 … 3** added up: 6.

Loop Invariants

- A property that holds at the **start** of every pass.
- In the sum loop: `sum = 1 + 2 + … + (i − 1)`.
- When the loop ends (i = n + 1), the invariant tells us the result is correct.

start of pass: i = 5

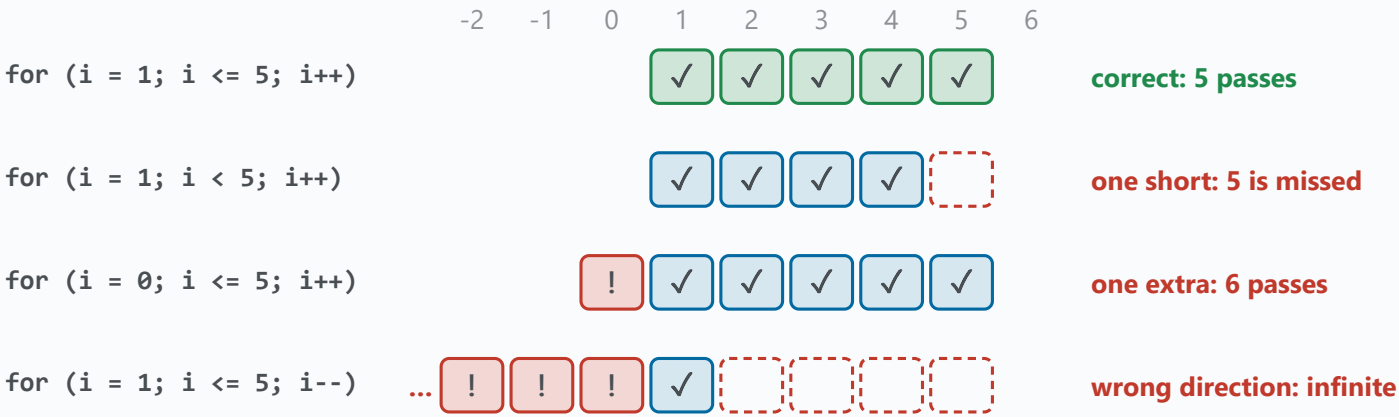


`sum = 10`

`= 1 + 2 + 3 + 4`

i = 5 → condition false. The invariant still holds: sum = 1 + 2 + 3 + 4 = 10 → the result is correct.

Off-by-One Errors (count 1 to 5)



- `<` vs. `<=`, or starting at 0 vs. 1, means one pass too few or too many (“off-by-one”).
- Test: check the first pass, the last pass and the pass count by hand.

VERIFICATION · STEP THROUGH THE CODE

Checking the Bounds: `i < 5` → 5 passes

Start: **i = 0**, count = 0. Condition `0 < 5` → true.

Step 1 / 7

```
▶ int count = 0;
▶ for (int i = 0; i < 5; i++) {
3     System.out.print(i + " ");
4     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 0` `count = 0`

OUTPUT

VERIFICATION · STEP THROUGH THE CODE

Checking the Bounds: `i < 5` → 5 passes

Pass 1: prints i = 0, count becomes 1.

Step 2 / 7

```
1 int count = 0;
2 for (int i = 0; i < 5; i++) {
▶     System.out.print(i + " ");
▶     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 0` `count = 1`

OUTPUT

```
0
```

VERIFICATION · STEP THROUGH THE CODE

Checking the Bounds: `i < 5` → 5 passes

Pass 2: prints i = 1, count becomes 2.

Step 3 / 7

```
1 int count = 0;
2 for (int i = 0; i < 5; i++) {
▶     System.out.print(i + " ");
▶     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 1` `count = 2`

OUTPUT

```
0 1
```

VERIFICATION · STEP THROUGH THE CODE

Checking the Bounds: `i < 5` → 5 passes

Pass 3: prints i = 2, count becomes 3.

Step 4 / 7

```
1 int count = 0;
2 for (int i = 0; i < 5; i++) {
▶     System.out.print(i + " ");
▶     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 2` `count = 3`

OUTPUT

```
0 1 2
```

VERIFICATION · STEP THROUGH THE CODE

Checking the Bounds: `i < 5` → 5 passes

Pass 4: prints i = 3, count becomes 4.

Step 5 / 7

```
1 int count = 0;
2 for (int i = 0; i < 5; i++) {
▶     System.out.print(i + " ");
▶     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 3` `count = 4`

OUTPUT

```
0 1 2 3
```

VERIFICATION · STEP THROUGH THE CODE

Checking the Bounds: `i < 5` → 5 passes

Pass 5: prints i = 4, count becomes 5.

Step 6 / 7

```
1 int count = 0;
2 for (int i = 0; i < 5; i++) {
▶     System.out.print(i + " ");
▶     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 4` `count = 5`

OUTPUT

```
0 1 2 3 4
```

VERIFICATION · STEP THROUGH THE CODE

Checking the Bounds: `i < 5` → 5 passes

i++ → 5; `5 < 5` → false, the loop ends. Exactly **5 passes**: i = 0…4.

Step 7 / 7

```
1 int count = 0;
2 for (int i = 0; i < 5; i++) {
3     System.out.print(i + " ");
4     count++;
5 }
▶ System.out.println("| rounds: " + count);
```

VARIABLES

`i = (gone)` `count = 5`

OUTPUT

```
0 1 2 3 4 | rounds: 5
```

VERIFICATION · STEP THROUGH THE CODE

Off-by-One: `i <= 5` → 6 passes

Start: **i = 0**, count = 0. Condition `0 <= 5` → true.

Step 1 / 8

```
▶ int count = 0;
▶ for (int i = 0; i <= 5; i++) {
3     System.out.print(i + " ");
4     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 0` `count = 0`

OUTPUT

VERIFICATION · STEP THROUGH THE CODE

Off-by-One: `i <= 5` → 6 passes

Pass 1: prints i = 0, count becomes 1.

Step 2 / 8

```
1 int count = 0;
2 for (int i = 0; i <= 5; i++) {
▶     System.out.print(i + " ");
▶     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 0` `count = 1`

OUTPUT

```
0
```

VERIFICATION · STEP THROUGH THE CODE

Off-by-One: `i <= 5` → 6 passes

Pass 2: prints i = 1, count becomes 2.

Step 3 / 8

```
1 int count = 0;
2 for (int i = 0; i <= 5; i++) {
▶     System.out.print(i + " ");
▶     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 1` `count = 2`

OUTPUT

```
0 1
```

VERIFICATION · STEP THROUGH THE CODE

Off-by-One: `i <= 5` → 6 passes

Pass 3: prints i = 2, count becomes 3.

Step 4 / 8

```
1 int count = 0;
2 for (int i = 0; i <= 5; i++) {
▶     System.out.print(i + " ");
▶     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 2` `count = 3`

OUTPUT

```
0 1 2
```

VERIFICATION · STEP THROUGH THE CODE

Off-by-One: `i <= 5` → 6 passes

Pass 4: prints i = 3, count becomes 4.

Step 5 / 8

```
1 int count = 0;
2 for (int i = 0; i <= 5; i++) {
▶     System.out.print(i + " ");
▶     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 3` `count = 4`

OUTPUT

```
0 1 2 3
```

VERIFICATION · STEP THROUGH THE CODE

Off-by-One: `i <= 5` → 6 passes

Pass 5: prints i = 4, count becomes 5.

Step 6 / 8

```
1 int count = 0;
2 for (int i = 0; i <= 5; i++) {
▶     System.out.print(i + " ");
▶     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 4` `count = 5`

OUTPUT

```
0 1 2 3 4
```

VERIFICATION · STEP THROUGH THE CODE

Off-by-One: `i <= 5` → 6 passes

Pass 6: prints i = 5, count becomes 6.

Step 7 / 8

```
1 int count = 0;
2 for (int i = 0; i <= 5; i++) {
▶     System.out.print(i + " ");
▶     count++;
5 }
6 System.out.println("| rounds: " + count);
```

VARIABLES

`i = 5` `count = 6`

OUTPUT

```
0 1 2 3 4 5
```

VERIFICATION · STEP THROUGH THE CODE

Off-by-One: `i <= 5` → 6 passes

i++ → 6; `6 <= 5` → false. The `<=` caused one extra pass: count = **6**, not the expected 5.

Step 8 / 8

```
1 int count = 0;
2 for (int i = 0; i <= 5; i++) {
3     System.out.print(i + " ");
4     count++;
5 }
▶ System.out.println("| rounds: " + count);
```

VARIABLES

`i = (gone)` `count = 6`

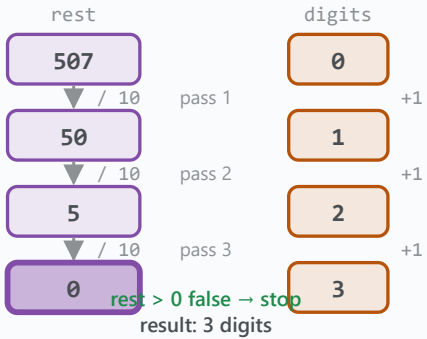
OUTPUT

```
0 1 2 3 4 5 | rounds: 6
```

EXAMPLE

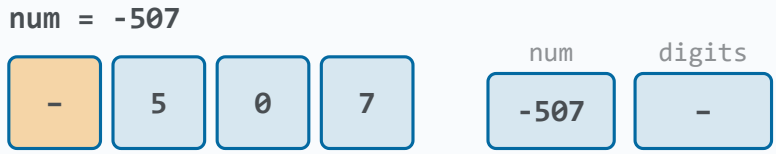
Counting Digits

```
int num = -507;
long rest = num;
if (rest < 0) rest = -rest;
int digits = 0;
do {
    digits++;
    rest /= 10;
} while (rest > 0);
System.out.println(digits);
```



- Each `/ 10` drops the last digit; we count the divisions.
- Thanks to **do-while**, 0 also counts as 1 digit; no special case needed.
- The long copy avoids overflow when negating an extreme value such as `-2147483648`.

Code Trace: counting digits (num = −507)



Let's try a negative number: **num = −507**.

Step 1 / 14

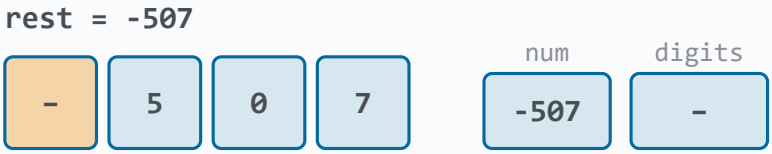
```
▶ int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
6     digits++;
7     rest /= 10;
8 } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

num = −507

OUTPUT

Code Trace: counting digits (num = −507)



The int is copied into a long (the negation of −2147483648 does not fit in an int, but fits in a long).

Step 2 / 14

```
1 int num = -507;
▶ long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
6     digits++;
7     rest /= 10;
8 } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

num = −507 rest = −507

OUTPUT

Code Trace: counting digits (num = −507)



−507 < 0 → true → **rest = 507**. The sign does not change the digit count.

Step 3 / 14

```
1 int num = -507;
2 long rest = num;
▶ if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
6     digits++;
7     rest /= 10;
8 } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

num = −507 rest = 507

OUTPUT

Code Trace: counting digits (num = −507)



Counter **digits = 0**.

Step 4 / 14

```
1 int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
▶ int digits = 0;
5 do {
6     digits++;
7     rest /= 10;
8 } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

num = −507 | rest = 507 | digits = 0

OUTPUT

Code Trace: counting digits (num = −507)



Body: one digit counted → digits = **1**.

Step 5 / 14

```
1 int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
▶     digits++;
7     rest /= 10;
8 } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

num = −507 | rest = 507 | digits = 1

OUTPUT

Code Trace: counting digits (num = −507)



rest = 507 / 10 = **50** (integer division drops the last digit).

Step 6 / 14

```
1 int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
6     digits++;
▶     rest /= 10;
8 } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

num = −507 | rest = 50 | digits = 1

OUTPUT

Code Trace: counting digits (num = −507)



Condition: 50 > 0 → **true** → repeat.

Step 7 / 14

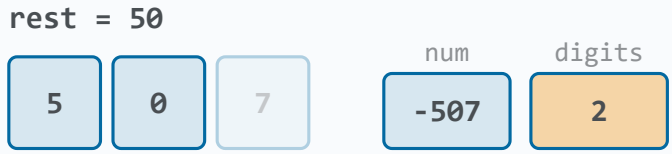
```
1 int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
6     digits++;
7     rest /= 10;
▶ } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

`num = −507` `rest = 50` `digits = 1`

OUTPUT

Code Trace: counting digits (num = −507)



Body: one digit counted → digits = **2**.

Step 8 / 14

```
1 int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
▶     digits++;
7     rest /= 10;
8 } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

num = −507 | rest = 50 | digits = 2

OUTPUT

Code Trace: counting digits (num = −507)



rest = 50 / 10 = **5** (integer division drops the last digit).

Step 9 / 14

```
1 int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
6     digits++;
▶     rest /= 10;
8 } while (rest > 0);
9 System.out.println(digits);
```

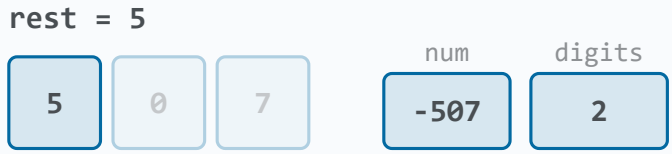
VARIABLES

num = −507 | rest = 5 | digits = 2

OUTPUT

EXAMPLES · STEP THROUGH THE CODE

Code Trace: counting digits (num = −507)



Condition: 5 > 0 → **true** → repeat.

Step 10 / 14

```
1 int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
6     digits++;
7     rest /= 10;
▶ } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

`num = −507` `rest = 5` `digits = 2`

OUTPUT

Code Trace: counting digits (num = −507)



Body: one digit counted → digits = **3**.

Step 11 / 14

```
1 int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
▶     digits++;
7     rest /= 10;
8 } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

`num = −507` `rest = 5` `digits = 3`

OUTPUT

Code Trace: counting digits (num = −507)



rest = 5 / 10 = **0** (integer division drops the last digit).

Step 12 / 14

```
1 int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
6     digits++;
▶     rest /= 10;
8 } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

`num = −507` `rest = 0` `digits = 3`

OUTPUT

Code Trace: counting digits (num = −507)

rest = 0

5	0	7

num	digits
-507	3

Condition: 0 > 0 → **false** → done.

Step 13 / 14

```
1 int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
6     digits++;
7     rest /= 10;
▶ } while (rest > 0);
9 System.out.println(digits);
```

VARIABLES

num = **−507** rest = **0** digits = **3**

OUTPUT

EXAMPLES · STEP THROUGH THE CODE

Code Trace: counting digits (num = −507)



3 is printed. For num = 0, do-while would still run the body once → 1 digit (correct).

Step 14 / 14

```
1 int num = -507;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int digits = 0;
5 do {
6     digits++;
7     rest /= 10;
8 } while (rest > 0);
▶ System.out.println(digits);
```

VARIABLES

num = **−507** rest = **0** digits = **3**

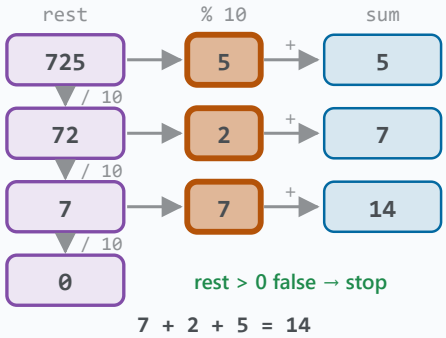
OUTPUT

```
3
```

EXAMPLE

Sum of Digits

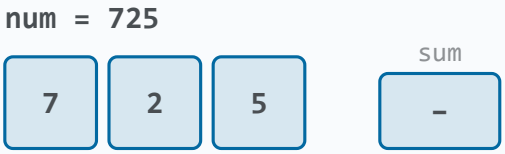
```
int num = 725;
long rest = num;
if (rest < 0) rest = -rest;
int sum = 0;
while (rest > 0) {
    sum += (int) (rest % 10);
    rest /= 10;
}
System.out.println(sum);
```



- `% 10` gives the last digit.
- `/ 10` removes the processed last digit.
- A while is enough here: for 0 the body never runs and sum stays 0 (correct).

EXAMPLES · STEP THROUGH THE CODE

Code Trace: sum of digits (num = 725)



A number with digits 7, 2 and 5: **725**.

Step 1 / 15

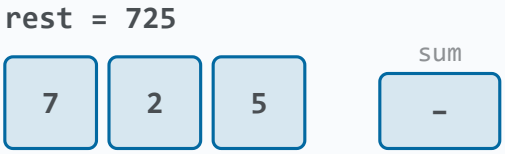
```
▶ int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
5 while (rest > 0) {
6     sum += (int) (rest % 10);
7     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = 725

OUTPUT

Code Trace: sum of digits (num = 725)



A copy to work on: rest = 725.

Step 2 / 15

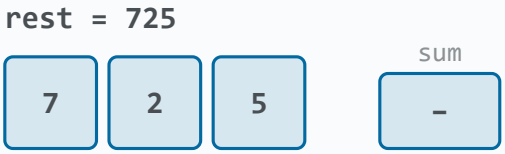
```
1 int num = 725;
▶ long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
5 while (rest > 0) {
6     sum += (int) (rest % 10);
7     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = 725 rest = 725

OUTPUT

Code Trace: sum of digits (num = 725)



725 < 0 → false: the sign stays.

Step 3 / 15

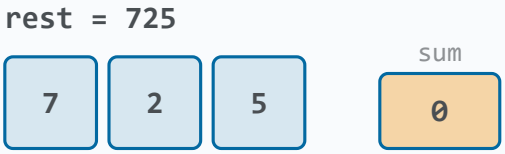
```
1 int num = 725;
2 long rest = num;
▶ if (rest < 0) rest = -rest;
4 int sum = 0;
5 while (rest > 0) {
6     sum += (int) (rest % 10);
7     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = 725 rest = 725

OUTPUT

Code Trace: sum of digits (num = 725)



Accumulator **sum = 0**.

Step 4 / 15

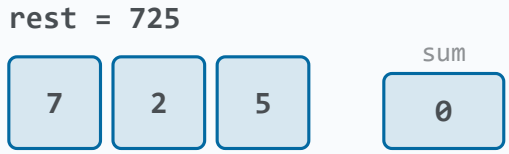
```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
▶ int sum = 0;
5 while (rest > 0) {
6     sum += (int) (rest % 10);
7     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = 725 | rest = 725 | sum = 0

OUTPUT

Code Trace: sum of digits (num = 725)



Condition: 725 > 0 → **true**.

Step 5 / 15

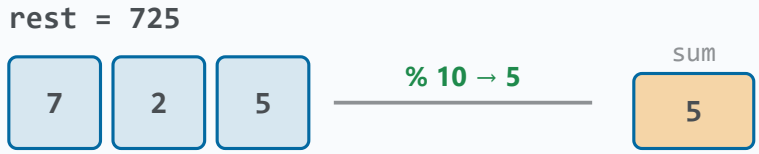
```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
▶ while (rest > 0) {
6     sum += (int) (rest % 10);
7     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = **725** rest = **725** sum = **0**

OUTPUT

Code Trace: sum of digits (num = 725)



Last digit 725 % 10 = **5**; sum = 0 + 5 = **5**.

Step 6 / 15

```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
5 while (rest > 0) {
▶     sum += (int) (rest % 10);
7     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = **725** | rest = **725** | sum = **5**

OUTPUT

Code Trace: sum of digits (num = 725)



rest = 725 / 10 = **72**: the processed digit is dropped.

Step 7 / 15

```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
5 while (rest > 0) {
6     sum += (int) (rest % 10);
▶     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = **725** | rest = **72** | sum = **5**

OUTPUT

Code Trace: sum of digits (num = 725)



Condition: 72 > 0 → **true**.

Step 8 / 15

```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
▶ while (rest > 0) {
6     sum += (int) (rest % 10);
7     rest /= 10;
8 }
9 System.out.println(sum);
```

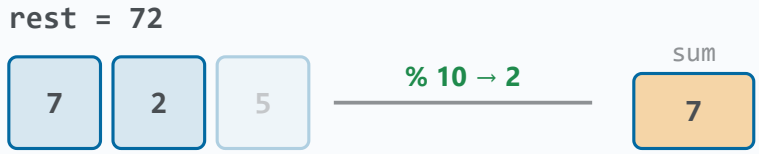
VARIABLES

num = **725** rest = **72** sum = **5**

OUTPUT

EXAMPLES · STEP THROUGH THE CODE

Code Trace: sum of digits (num = 725)



Last digit 72 % 10 = **2**; sum = 5 + 2 = **7**.

Step 9 / 15

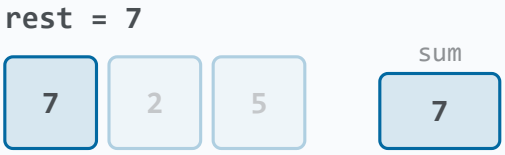
```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
5 while (rest > 0) {
▶     sum += (int) (rest % 10);
7     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = **725** rest = **72** sum = **7**

OUTPUT

Code Trace: sum of digits (num = 725)



rest = 72 / 10 = **7**: the processed digit is dropped.

Step 10 / 15

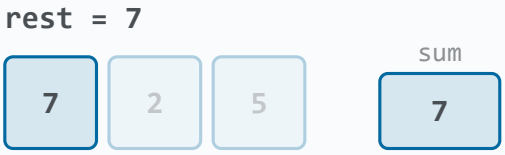
```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
5 while (rest > 0) {
6     sum += (int) (rest % 10);
▶     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = **725** | rest = **7** | sum = **7**

OUTPUT

Code Trace: sum of digits (num = 725)



Condition: 7 > 0 → **true**.

Step 11 / 15

```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
▶ while (rest > 0) {
6     sum += (int) (rest % 10);
7     rest /= 10;
8 }
9 System.out.println(sum);
```

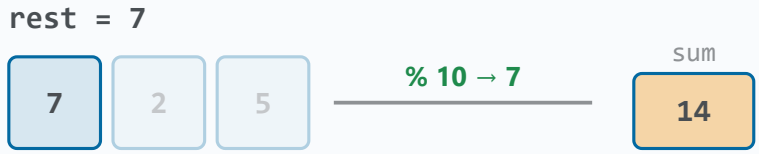
VARIABLES

`num = 725` `rest = 7` `sum = 7`

OUTPUT

EXAMPLES · STEP THROUGH THE CODE

Code Trace: sum of digits (num = 725)



Last digit 7 % 10 = **7**; sum = 7 + 7 = **14**.

Step 12 / 15

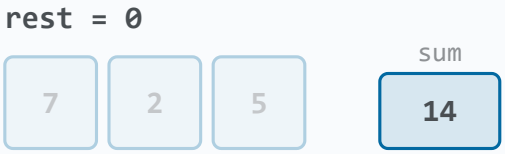
```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
5 while (rest > 0) {
▶     sum += (int) (rest % 10);
7     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = **725** rest = **7** sum = **14**

OUTPUT

Code Trace: sum of digits (num = 725)



rest = 7 / 10 = **0**: the processed digit is dropped.

Step 13 / 15

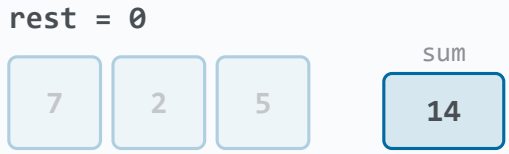
```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
5 while (rest > 0) {
6     sum += (int) (rest % 10);
▶     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = 725 | rest = 0 | sum = 14

OUTPUT

Code Trace: sum of digits (num = 725)



Condition: 0 > 0 → **false**.

Step 14 / 15

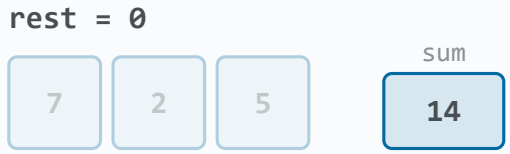
```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
▶ while (rest > 0) {
6     sum += (int) (rest % 10);
7     rest /= 10;
8 }
9 System.out.println(sum);
```

VARIABLES

num = **725** rest = **0** sum = **14**

OUTPUT

Code Trace: sum of digits (num = 725)



14 is printed: 7 + 2 + 5 = 14. `num` is unchanged; only its copy (`rest`) was used up.

Step 15 / 15

```
1 int num = 725;
2 long rest = num;
3 if (rest < 0) rest = -rest;
4 int sum = 0;
5 while (rest > 0) {
6     sum += (int) (rest % 10);
7     rest /= 10;
8 }
▶ System.out.println(sum);
```

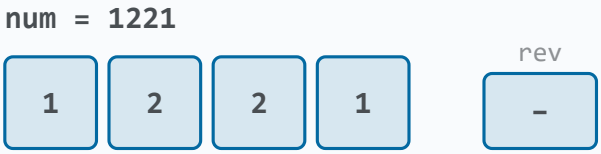
VARIABLES

`num = 725` `rest = 0` `sum = 14`

OUTPUT

```
14
```

Reversing a Number and Palindromes (1221)



A number that reads the same backward is a **palindrome**. Let's try 1221.

Step 1 / 17

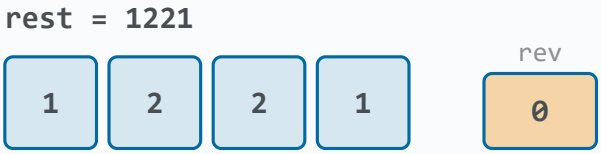
```
▶ int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

num = 1221

OUTPUT

Reversing a Number and Palindromes (1221)



We work on a copy; the reversed number is built up from 0.

Step 2 / 17

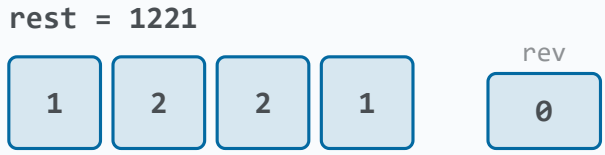
```
1 int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

num = 1221 rest = 1221 rev = 0

OUTPUT

Reversing a Number and Palindromes (1221)



Condition: 1221 > 0 → **true**.

Step 3 / 17

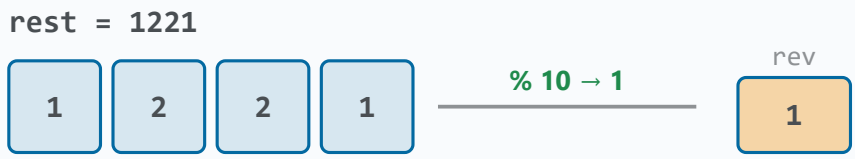
```
1 int num = 1221;
2 int rest = num, rev = 0;
▶ while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

`num = 1221` `rest = 1221` `rev = 0`

OUTPUT

Reversing a Number and Palindromes (1221)



digit = 1221 % 10 = 1; rev = 0 × 10 + 1 = **1** (earlier digits shift one place left).

Step 4 / 17

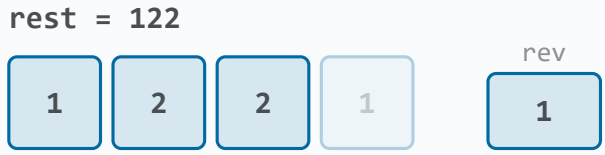
```
1 int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
▶     int digit = rest % 10;
▶     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

num = **1221** rest = **1221** rev = **1** digit = **1**

OUTPUT

Reversing a Number and Palindromes (1221)



rest = 1221 / 10 = **122**.

Step 5 / 17

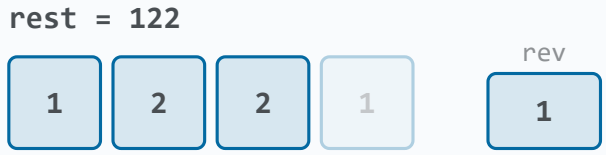
```
1 int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
▶     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

`num = 1221` `rest = 122` `rev = 1` `digit = 1`

OUTPUT

Reversing a Number and Palindromes (1221)



Condition: 122 > 0 → **true**.


```
1 int num = 1221;
2 int rest = num, rev = 0;
▶ while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

`num = 1221` `rest = 122` `rev = 1`

OUTPUT

Reversing a Number and Palindromes (1221)



digit = 122 % 10 = 2; rev = 1 × 10 + 2 = **12** (earlier digits shift one place left).

Step 7 / 17

```
1 int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
▶     int digit = rest % 10;
▶     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

num = **1221** rest = **122** rev = **12** digit = **2**

OUTPUT

Reversing a Number and Palindromes (1221)



rest = 122 / 10 = **12**.

Step 8 / 17

```
1 int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
▶     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

num = **1221** | rest = **12** | rev = **12** | digit = **2**

OUTPUT

Reversing a Number and Palindromes (1221)



Condition: 12 > 0 → **true**.

Step 9 / 17

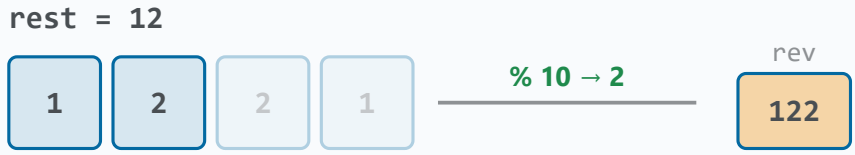
```
1 int num = 1221;
2 int rest = num, rev = 0;
▶ while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

`num = 1221` `rest = 12` `rev = 12`

OUTPUT

Reversing a Number and Palindromes (1221)



digit = 12 % 10 = 2; rev = 12 × 10 + 2 = **122** (earlier digits shift one place left).

Step 10 / 17

```
1 int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
▶     int digit = rest % 10;
▶     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

num = 1221 | rest = 12 | rev = 122 | digit = 2

OUTPUT

Reversing a Number and Palindromes (1221)



rest = 12 / 10 = **1**.

Step 11 / 17

```
1 int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
▶     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

num = **1221** | rest = **1** | rev = **122** | digit = **2**

OUTPUT

Reversing a Number and Palindromes (1221)



Condition: 1 > 0 → **true**.

Step 12 / 17

```
1 int num = 1221;
2 int rest = num, rev = 0;
▶ while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

`num = 1221` `rest = 1` `rev = 122`

OUTPUT

Reversing a Number and Palindromes (1221)



digit = 1 % 10 = 1; rev = 122 × 10 + 1 = **1221** (earlier digits shift one place left).

Step 13 / 17

```
1 int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
▶     int digit = rest % 10;
▶     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

num = 1221 | rest = 1 | rev = 1221 | digit = 1

OUTPUT

Reversing a Number and Palindromes (1221)



rest = 1 / 10 = **0**.

Step 14 / 17

```
1 int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
▶     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

num = **1221** | rest = **0** | rev = **1221** | digit = **1**

OUTPUT

Reversing a Number and Palindromes (1221)



Condition: 0 > 0 → **false**.

Step 15 / 17

```
1 int num = 1221;
2 int rest = num, rev = 0;
▶ while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

`num = 1221` `rest = 0` `rev = 1221`

OUTPUT

Reversing a Number and Palindromes (1221)



The reversed number **1221** is printed.


```
1 int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
6     rest /= 10;
7 }
▶ System.out.println(rev);
9 System.out.println(rev == num);  // palindrome?
```

VARIABLES

num = **1221** rest = **0** rev = **1221**

OUTPUT

```
1221
```

Reversing a Number and Palindromes (1221)



1221 == 1221 → **true**: 1221 is a palindrome. (For 123 the reverse is 321 → false.)

Step 17 / 17

```
1 int num = 1221;
2 int rest = num, rev = 0;
3 while (rest > 0) {
4     int digit = rest % 10;
5     rev = rev * 10 + digit;
6     rest /= 10;
7 }
8 System.out.println(rev);
▶ System.out.println(rev == num);  // palindrome?
```

VARIABLES

`num = 1221` `rest = 0` `rev = 1221`

OUTPUT

```
1221
true
```

EXAMPLES · STEP THROUGH THE CODE

Armstrong Numbers: 153 = $1^3 + 5^3 + 3^3$



Armstrong number: equals the sum of its digits each raised to the number of digits. 153 has three digits → cubes.

Step 1 / 13

```
▶ int num = 153;
2 int rest = num, sum = 0;
3 while (rest > 0) {
4     int r = rest % 10;
5     sum += r * r * r;   // 3 digits: cube
6     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

num = 153

OUTPUT

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



rest = 153, sum = 0.

Step 2 / 13

```
1 int num = 153;
▶ int rest = num, sum = 0;
3 while (rest > 0) {
4     int r = rest % 10;
5     sum += r * r * r;   // 3 digits: cube
6     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

num = 153 | rest = 153 | sum = 0

OUTPUT

EXAMPLES · STEP THROUGH THE CODE

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



Condition: 153 > 0 → **true**.

Step 3 / 13

```
1 int num = 153;
2 int rest = num, sum = 0;
▶ while (rest > 0) {
4     int r = rest % 10;
5     sum += r * r * r;   // 3 digits: cube
6     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

num = **153** rest = **153** sum = **0**

OUTPUT

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



r = 3; sum = 0 + 3×3×3 = **27**.

Step 4 / 13

```
1 int num = 153;
2 int rest = num, sum = 0;
3 while (rest > 0) {
▶     int r = rest % 10;
▶     sum += r * r * r;   // 3 digits: cube
6     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

num = 153 | rest = 153 | sum = 27 | r = 3

OUTPUT

EXAMPLES · STEP THROUGH THE CODE

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



rest /= 10 → 15.

Step 5 / 13

```
1 int num = 153;
2 int rest = num, sum = 0;
3 while (rest > 0) {
4     int r = rest % 10;
5     sum += r * r * r;   // 3 digits: cube
▶     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

num = 153 | rest = 15 | sum = 27 | r = 3

OUTPUT

EXAMPLES · STEP THROUGH THE CODE

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



Condition: 15 > 0 → **true**.

Step 6 / 13

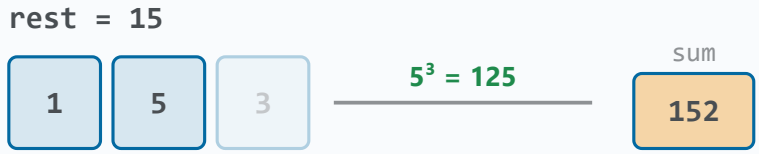
```
1 int num = 153;
2 int rest = num, sum = 0;
▶ while (rest > 0) {
4     int r = rest % 10;
5     sum += r * r * r;   // 3 digits: cube
6     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

num = **153** rest = **15** sum = **27**

OUTPUT

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



r = 5; sum = 27 + 5×5×5 = **152**.

Step 7 / 13

```
1 int num = 153;
2 int rest = num, sum = 0;
3 while (rest > 0) {
▶     int r = rest % 10;
▶     sum += r * r * r;   // 3 digits: cube
6     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

num = 153 | rest = 15 | sum = 152 | r = 5

OUTPUT

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



rest /= 10 → 1.

Step 8 / 13

```
1 int num = 153;
2 int rest = num, sum = 0;
3 while (rest > 0) {
4     int r = rest % 10;
5     sum += r * r * r;   // 3 digits: cube
▶     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

num = 153 | rest = 1 | sum = 152 | r = 5

OUTPUT

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



Condition: 1 > 0 → **true**.

Step 9 / 13

```
1 int num = 153;
2 int rest = num, sum = 0;
▶ while (rest > 0) {
4     int r = rest % 10;
5     sum += r * r * r;   // 3 digits: cube
6     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

`num = 153` `rest = 1` `sum = 152`

OUTPUT

EXAMPLES · STEP THROUGH THE CODE

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



r = 1; sum = 152 + 1×1×1 = **153**.

Step 10 / 13

```
1 int num = 153;
2 int rest = num, sum = 0;
3 while (rest > 0) {
▶     int r = rest % 10;
▶     sum += r * r * r;   // 3 digits: cube
6     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

num = 153 | rest = 1 | sum = 153 | r = 1

OUTPUT

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



rest /= 10 → 0.

Step 11 / 13

```
1 int num = 153;
2 int rest = num, sum = 0;
3 while (rest > 0) {
4     int r = rest % 10;
5     sum += r * r * r;   // 3 digits: cube
▶     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

num = 153 | rest = 0 | sum = 153 | r = 1

OUTPUT

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



Condition: 0 > 0 → **false**.


```
1 int num = 153;
2 int rest = num, sum = 0;
▶ while (rest > 0) {
4     int r = rest % 10;
5     sum += r * r * r;   // 3 digits: cube
6     rest /= 10;
7 }
8 System.out.println(sum == num);
```

VARIABLES

num = **153** rest = **0** sum = **153**

OUTPUT

EXAMPLES · STEP THROUGH THE CODE

Armstrong Numbers: 153 = 1^3 + 5^3 + 3^3



1 + 125 + 27 = 153 → **true**. The other 3-digit Armstrong numbers: 370, 371, 407.

Step 13 / 13

```
1 int num = 153;
2 int rest = num, sum = 0;
3 while (rest > 0) {
4     int r = rest % 10;
5     sum += r * r * r;   // 3 digits: cube
6     rest /= 10;
7 }
▶ System.out.println(sum == num);
```

VARIABLES

num = **153** rest = **0** sum = **153**

OUTPUT

```
true
```

Perfect Numbers: 28

`sum = 0`

1	2	3	4	5	6	7	8	9	10	11	12	13	14

Perfect number: equals the sum of its divisors other than itself (6 = 1 + 2 + 3). Let's try 28.

Step 1 / 18

```
▶ int n = 28;
2 int sum = 0;
3 for (int i = 1; i <= n / 2; i++) {
4     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

`n = 28`

OUTPUT

Perfect Numbers: 28

sum = 0

1	2	3	4	5	6	7	8	9	10	11	12	13	14

The divisor sum starts at 0. Apart from n itself, the largest divisor is at most n / 2 = 14.

Step 2 / 18

```
1 int n = 28;
▶ int sum = 0;
3 for (int i = 1; i <= n / 2; i++) {
4     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

n = 28 sum = 0

OUTPUT

Perfect Numbers: 28

`sum = 1`



i = 1: 28 % 1 = 0 → divisor, sum = **1**.

Step 3 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

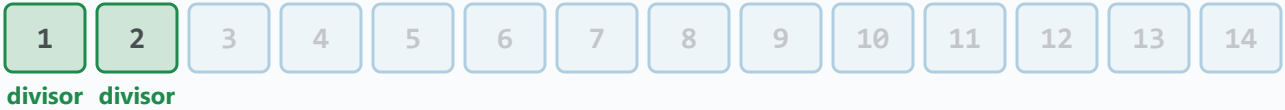
VARIABLES

`n = 28` `sum = 1` `i = 1`

OUTPUT

Perfect Numbers: 28

`sum = 1 + 2 = 3`



i = 2: 28 % 2 = 0 → divisor, sum = **3**.

Step 4 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

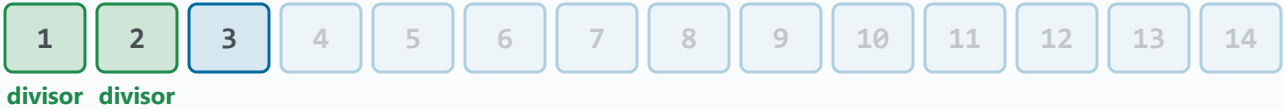
VARIABLES

`n = 28` `sum = 3` `i = 2`

OUTPUT

Perfect Numbers: 28

`sum = 1 + 2 = 3`



i = 3: 28 % 3 = 1 → not a divisor.

Step 5 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

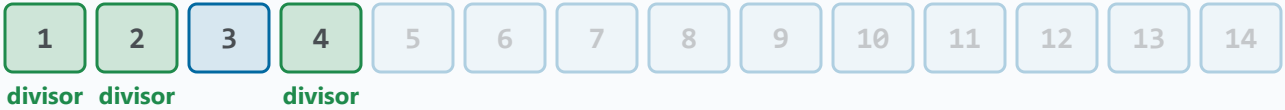
VARIABLES

`n = 28` `sum = 3` `i = 3`

OUTPUT

Perfect Numbers: 28

`sum = 1 + 2 + 4 = 7`



i = 4: 28 % 4 = 0 → divisor, sum = **7**.

Step 6 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

`n = 28` `sum = 7` `i = 4`

OUTPUT

Perfect Numbers: 28



i = 5: 28 % 5 = 3 → not a divisor.

Step 7 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

n = 28 | sum = 7 | i = 5

OUTPUT

Perfect Numbers: 28



i = 6: 28 % 6 = 4 → not a divisor.

Step 8 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

n = 28 | sum = 7 | i = 6

OUTPUT

Perfect Numbers: 28

`sum = 1 + 2 + 4 + 7 = 14`



i = 7: 28 % 7 = 0 → divisor, sum = **14**.

Step 9 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

`n = 28` `sum = 14` `i = 7`

OUTPUT

Perfect Numbers: 28

`sum = 1 + 2 + 4 + 7 = 14`



i = 8: 28 % 8 = 4 → not a divisor.

Step 10 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

`n = 28` `sum = 14` `i = 8`

OUTPUT

Perfect Numbers: 28



i = 9: 28 % 9 = 1 → not a divisor.

Step 11 / 18

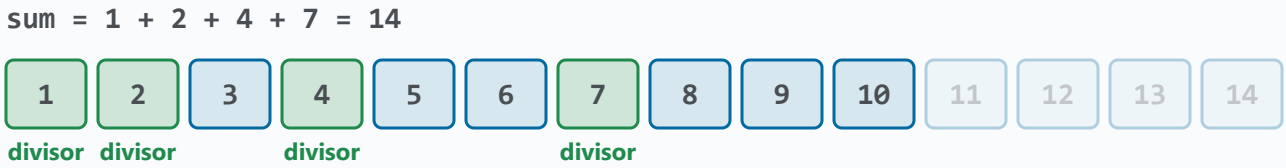
```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

n = 28 | sum = 14 | i = 9

OUTPUT

Perfect Numbers: 28



i = 10: 28 % 10 = 8 → not a divisor.

Step 12 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

n = 28 | sum = 14 | i = 10

OUTPUT

Perfect Numbers: 28



i = 11: 28 % 11 = 6 → not a divisor.

Step 13 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

n = 28 | sum = 14 | i = 11

OUTPUT

Perfect Numbers: 28



i = 12: 28 % 12 = 4 → not a divisor.

Step 14 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

n = 28 | sum = 14 | i = 12

OUTPUT

Perfect Numbers: 28



i = 13: 28 % 13 = 2 → not a divisor.

Step 15 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

n = 28 | sum = 14 | i = 13

OUTPUT

Perfect Numbers: 28



i = 14: 28 % 14 = 0 → divisor, sum = **28**.

Step 16 / 18

```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
▶     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

n = 28 | sum = 28 | i = 14

OUTPUT

Perfect Numbers: 28



i = 15 <= 14 → false: the loop ends.

Step 17 / 18

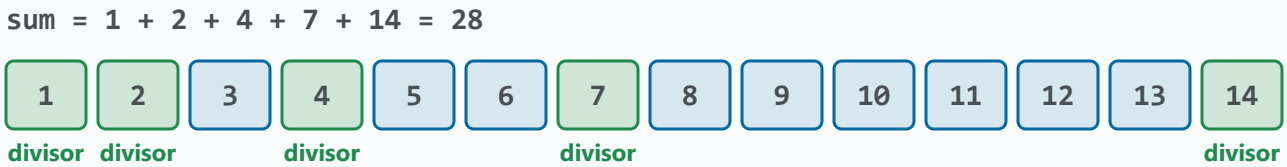
```
1 int n = 28;
2 int sum = 0;
▶ for (int i = 1; i <= n / 2; i++) {
4     if (n % i == 0) sum += i;
5 }
6 System.out.println(sum == n);
```

VARIABLES

n = 28 sum = 28

OUTPUT

Perfect Numbers: 28



1 + 2 + 4 + 7 + 14 = 28 → **true**. The next perfect numbers: 496, 8128.

Step 18 / 18

```
1 int n = 28;
2 int sum = 0;
3 for (int i = 1; i <= n / 2; i++) {
4     if (n % i == 0) sum += i;
5 }
▶ System.out.println(sum == n);
```

VARIABLES

`n = 28` `sum = 28`

OUTPUT

```
true
```

EXAMPLES · STEP THROUGH THE CODE

Decimal to Binary: 13 → 1101

division

remainder (bit)

read the bits bottom to top

binary	place
–	–

Converting 13 to binary: divide by 2 repeatedly and collect the remainders.

Step 1 / 16

```
▶ int n = 13;
2 int bin = 0, place = 1;
3 while (n > 0) {
4     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

n = 13

OUTPUT

Decimal to Binary: 13 → 1101

division

remainder (bit)

read the bits bottom to top

binary	place
0	1

We build the binary result from decimal digits (1s and 0s); place = 1 is the ones place.

Step 2 / 16

```
1 int n = 13;
▶ int bin = 0, place = 1;
3 while (n > 0) {
4     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

n = 13 | bin = 0 | place = 1

OUTPUT

Decimal to Binary: 13 → 1101

division

remainder (bit)

read the bits bottom to top

binary	place
0	1

Condition: 13 > 0 → **true**.

Step 3 / 16

```
1 int n = 13;
2 int bin = 0, place = 1;
▶ while (n > 0) {
4     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

`n = 13` `bin = 0` `place = 1`

OUTPUT

Decimal to Binary: 13 → 1101

division

13 / 2 = 6

remainder (bit)

1 ← bit

read the bits bottom to top

binary	place
1	1

13 % 2 = **1** → bin += 1 × 1 → **1**.

Step 4 / 16

```
1 int n = 13;
2 int bin = 0, place = 1;
3 while (n > 0) {
▶     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

`n = 13` `bin = 1` `place = 1`

OUTPUT

Decimal to Binary: 13 → 1101

division	remainder (bit)	read the bits bottom to top
`13 / 2 = 6`	`1`	binary `1` · place `10`

Next bit is one place left: place = 10; n = n / 2 = **6**.

Step 5 / 16

```
1  int n = 13;
2  int bin = 0, place = 1;
3  while (n > 0) {
4      bin += (n % 2) * place;
       place *= 10;
       n /= 2;
7  }
8  System.out.println(bin);    // 1101
```

VARIABLES

`n = 6` `bin = 1` `place = 10`

OUTPUT

Decimal to Binary: 13 → 1101

division

13 / 2 = 6

remainder (bit)

1

read the bits bottom to top

binary	place
1	10

Condition: 6 > 0 → **true**.

Step 6 / 16

```
1 int n = 13;
2 int bin = 0, place = 1;
▶ while (n > 0) {
4     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

`n = 6` `bin = 1` `place = 10`

OUTPUT

Decimal to Binary: 13 → 1101

division

13 / 2 = 6

6 / 2 = 3

remainder (bit)

1

0 ← bit

read the bits bottom to top

binary	place
1	10

6 % 2 = **0** → bin += 0 × 10 → **1**.

Step 7 / 16

```
1 int n = 13;
2 int bin = 0, place = 1;
3 while (n > 0) {
▶     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

`n = 6` `bin = 1` `place = 10`

OUTPUT

Decimal to Binary: 13 → 1101

division	remainder (bit)
13 / 2 = 6	1
6 / 2 = 3	0

read the bits bottom to top

binary	place
1	100

Next bit is one place left: place = 100; n = n / 2 = **3**.

Step 8 / 16

```
1 int n = 13;
2 int bin = 0, place = 1;
3 while (n > 0) {
4     bin += (n % 2) * place;
      place *= 10;
      n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

n = 3 | bin = 1 | place = 100

OUTPUT

Decimal to Binary: 13 → 1101

division	remainder (bit)
13 / 2 = 6	1
6 / 2 = 3	0

read the bits bottom to top

binary	place
1	100

Condition: 3 > 0 → **true**.

Step 9 / 16

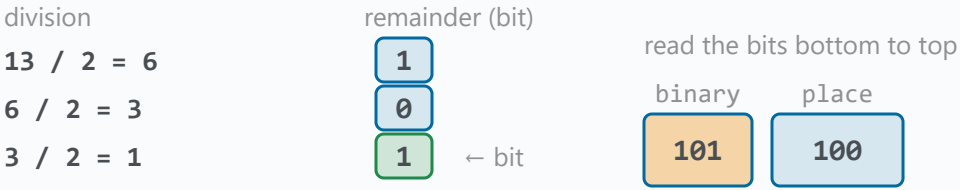
```
1 int n = 13;
2 int bin = 0, place = 1;
▶ while (n > 0) {
4     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

`n = 3` `bin = 1` `place = 100`

OUTPUT

Decimal to Binary: 13 → 1101



3 % 2 = **1** → bin += 1 × 100 → **101**.

Step 10 / 16

```
1 int n = 13;
2 int bin = 0, place = 1;
3 while (n > 0) {
▶     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

`n = 3` `bin = 101` `place = 100`

OUTPUT

Decimal to Binary: 13 → 1101

division	remainder (bit)
13 / 2 = 6	1
6 / 2 = 3	0
3 / 2 = 1	1

read the bits bottom to top

binary	place
101	1000

Next bit is one place left: place = 1000; n = n / 2 = **1**.

Step 11 / 16

```
1 int n = 13;
2 int bin = 0, place = 1;
3 while (n > 0) {
4     bin += (n % 2) * place;
      place *= 10;
      n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

`n = 1` `bin = 101` `place = 1000`

OUTPUT

Decimal to Binary: 13 → 1101

division	remainder (bit)
13 / 2 = 6	1
6 / 2 = 3	0
3 / 2 = 1	1

read the bits bottom to top

binary	place
101	1000

Condition: 1 > 0 → **true**.

Step 12 / 16

```
1 int n = 13;
2 int bin = 0, place = 1;
▶ while (n > 0) {
4     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

`n = 1` `bin = 101` `place = 1000`

OUTPUT

Decimal to Binary: 13 → 1101

division	remainder (bit)
13 / 2 = 6	1
6 / 2 = 3	0
3 / 2 = 1	1
1 / 2 = 0	1 ← bit

read the bits bottom to top

binary	place
1101	1000

1 % 2 = **1** → bin += 1 × 1000 → **1101**.

Step 13 / 16

```
1 int n = 13;
2 int bin = 0, place = 1;
3 while (n > 0) {
▶     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

`n = 1` `bin = 1101` `place = 1000`

OUTPUT

Decimal to Binary: 13 → 1101

division	remainder (bit)
13 / 2 = 6	1
6 / 2 = 3	0
3 / 2 = 1	1
1 / 2 = 0	1

read the bits bottom to top

binary	place
1101	10000

Next bit is one place left: place = 10000; n = n / 2 = **0**.

Step 14 / 16

```
1 int n = 13;
2 int bin = 0, place = 1;
3 while (n > 0) {
4     bin += (n % 2) * place;
      place *= 10;
      n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

n = 0 | bin = 1101 | place = 10000

OUTPUT

Decimal to Binary: 13 → 1101

division	remainder (bit)
13 / 2 = 6	1
6 / 2 = 3	0
3 / 2 = 1	1
1 / 2 = 0	1

read the bits bottom to top

binary	place
1101	10000

Condition: 0 > 0 → **false**.

Step 15 / 16

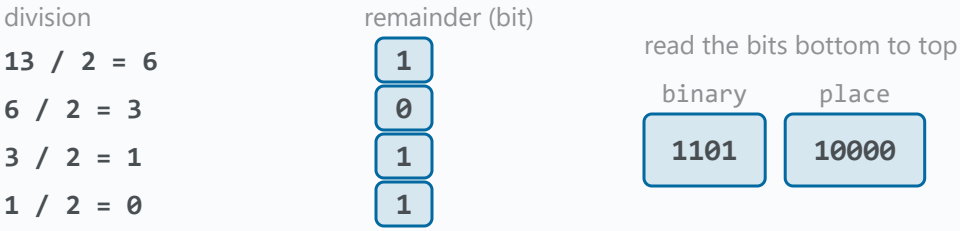
```
1 int n = 13;
2 int bin = 0, place = 1;
▶ while (n > 0) {
4     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
8 System.out.println(bin);    // 1101
```

VARIABLES

`n = 0` `bin = 1101` `place = 10000`

OUTPUT

Decimal to Binary: 13 → 1101



1101 is printed: 1·8 + 1·4 + 0·2 + 1·1 = 13. (For large numbers int overflows; build the result as a String instead.)

Step 16 / 16

```
1 int n = 13;
2 int bin = 0, place = 1;
3 while (n > 0) {
4     bin += (n % 2) * place;
5     place *= 10;
6     n /= 2;
7 }
▶ System.out.println(bin);    // 1101
```

VARIABLES

`n = 0` `bin = 1101` `place = 10000`

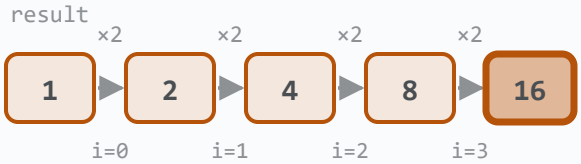
OUTPUT

```
1101
```

EXAMPLE

Exponentiation

```
int base = 2, exp = 4;
int result = 1;
for (int i = 0; i < exp; i++) {
    result *= base;
}
System.out.println(result);
```

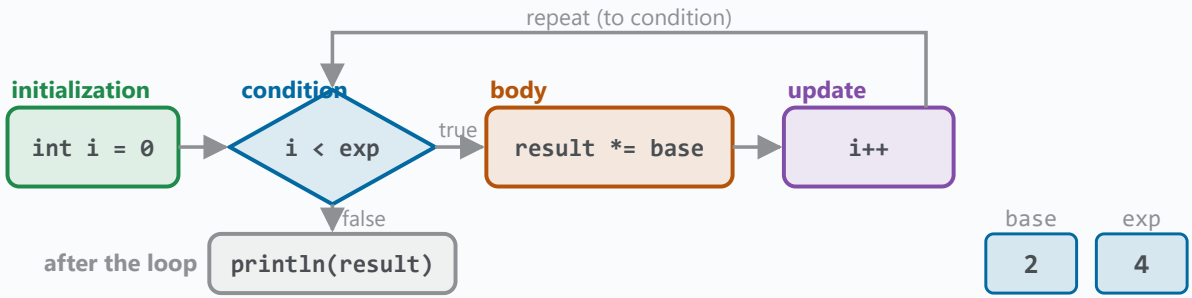


i = 4: 4 < 4 false → stop

2 × 2 × 2 × 2 = 16

- The loop runs exactly **exp** times; each pass multiplies by the base once.
- If exp = 0 the loop never runs and result stays 1 ($2^0 = 1$).
- Negative exponents are not handled; large results overflow int.

Code Trace: 2^4



We will compute 2^4: base = 2, exp = 4.

Step 1 / 17

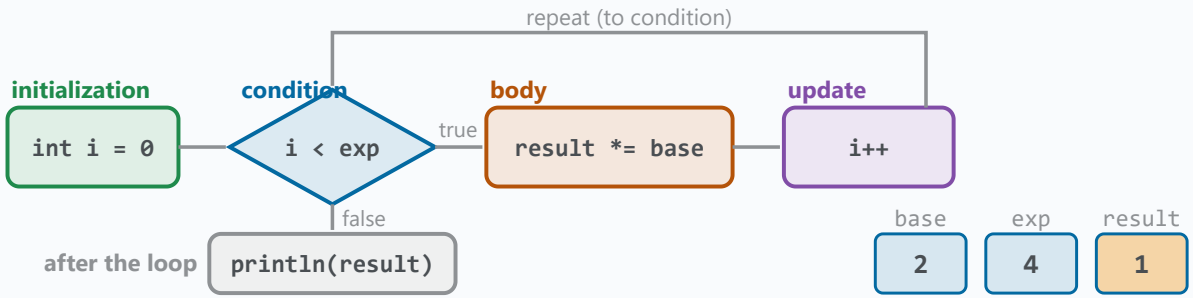
```
▶ int base = 2, exp = 4;
2 int result = 1;
3 for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 exp = 4

OUTPUT

Code Trace: 2^4



1 is the identity for multiplication: result = 1 (= 2^0).

Step 2 / 17

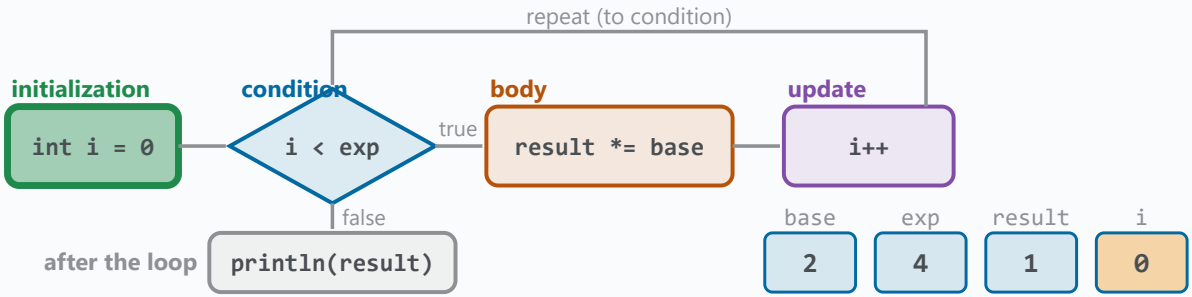
```
1 int base = 2, exp = 4;
▶ int result = 1;
3 for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 1

OUTPUT

Code Trace: 2^4



Initialization: i = 0. The loop will run exp times (0, 1, 2, 3).

Step 3 / 17

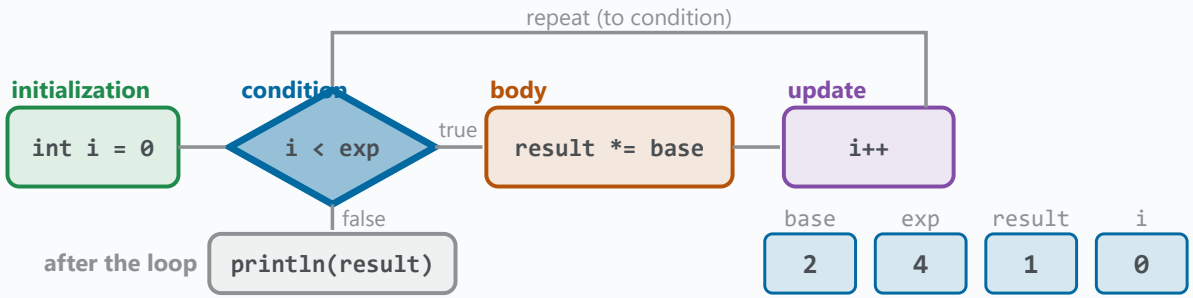
```
1 int base = 2, exp = 4;
2 int result = 1;
▶ for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 1 | i = 0

OUTPUT

Code Trace: 2^4



Condition: 0 < 4 → **true**.

Step 4 / 17

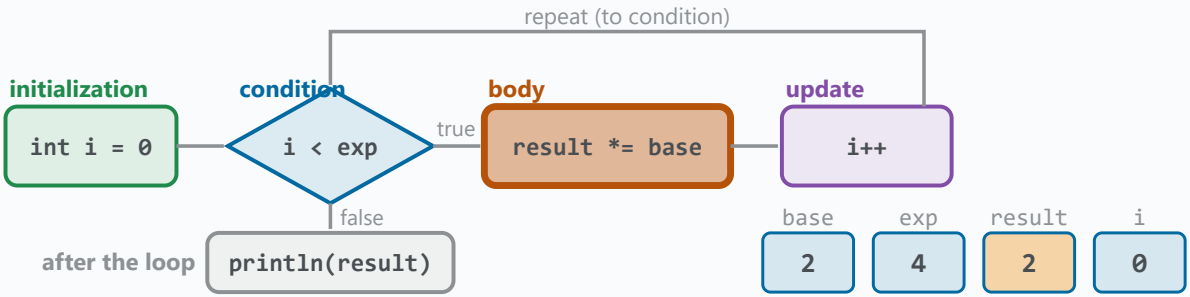
```
1 int base = 2, exp = 4;
2 int result = 1;
▶ for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 1 | i = 0

OUTPUT

Code Trace: 2^4



result = 1 × 2 = **2** (= 2^1).

Step 5 / 17

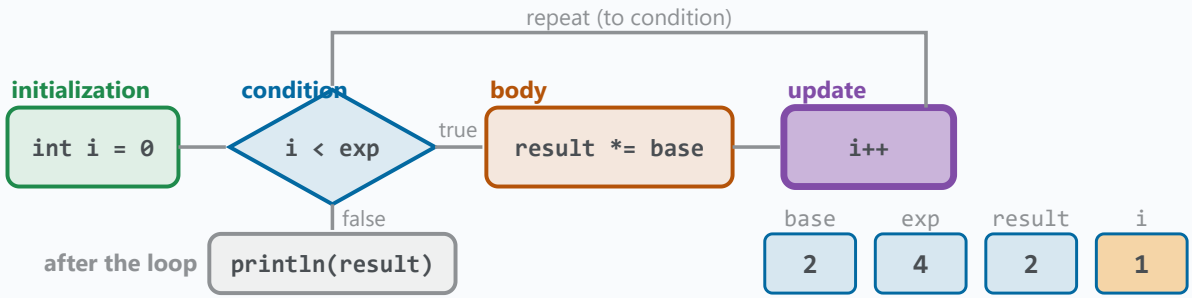
```
1 int base = 2, exp = 4;
2 int result = 1;
3 for (int i = 0; i < exp; i++) {
▶     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 2 | i = 0

OUTPUT

Code Trace: 2^4



Update: i = 1.

Step 6 / 17

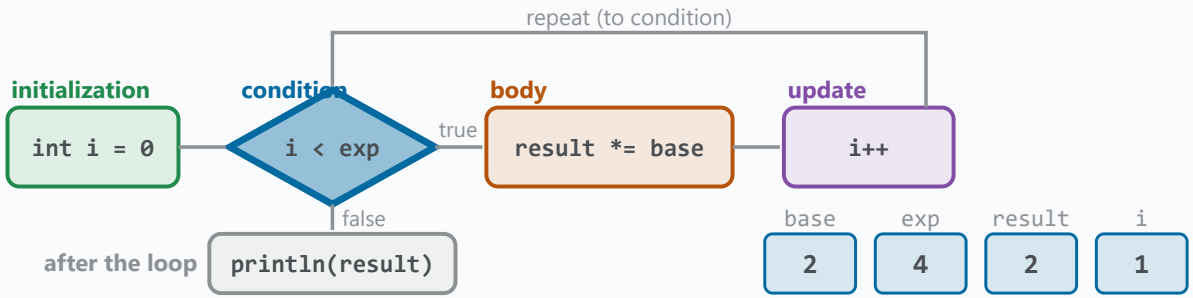
```
1 int base = 2, exp = 4;
2 int result = 1;
▶ for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 2 | i = 1

OUTPUT

Code Trace: 2^4



Condition: 1 < 4 → **true**.

Step 7 / 17

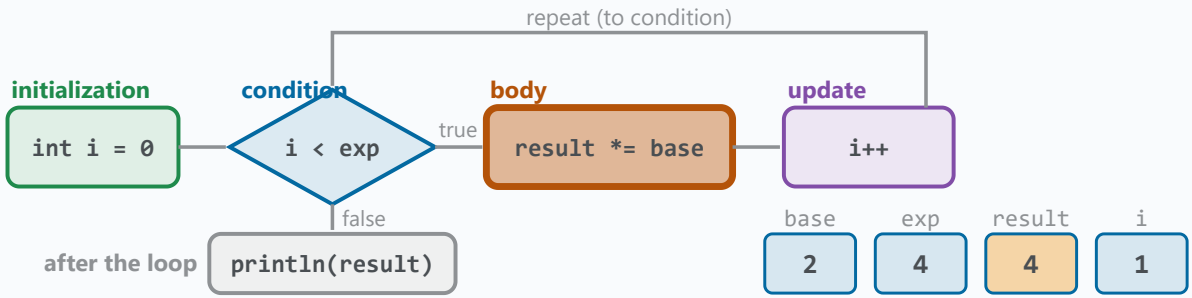
```
1 int base = 2, exp = 4;
2 int result = 1;
▶ for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

`base = 2` `exp = 4` `result = 2` `i = 1`

OUTPUT

Code Trace: 2^4



result = 2 × 2 = **4** (= 2^2).

Step 8 / 17

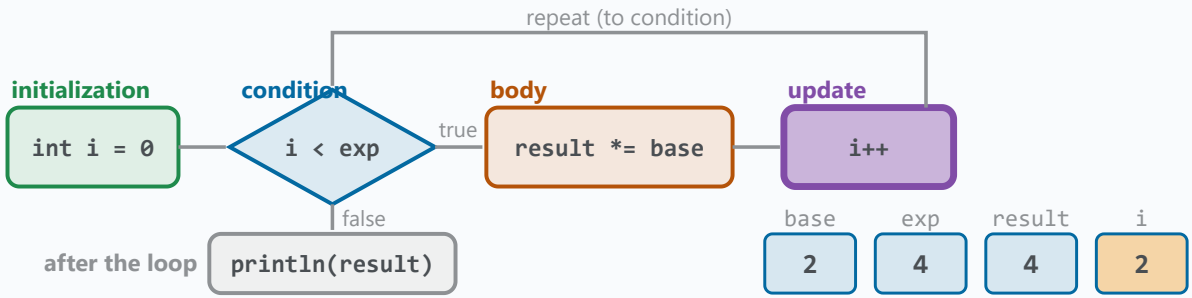
```
1 int base = 2, exp = 4;
2 int result = 1;
3 for (int i = 0; i < exp; i++) {
▶     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 4 | i = 1

OUTPUT

Code Trace: 2^4



Update: i = 2.

Step 9 / 17

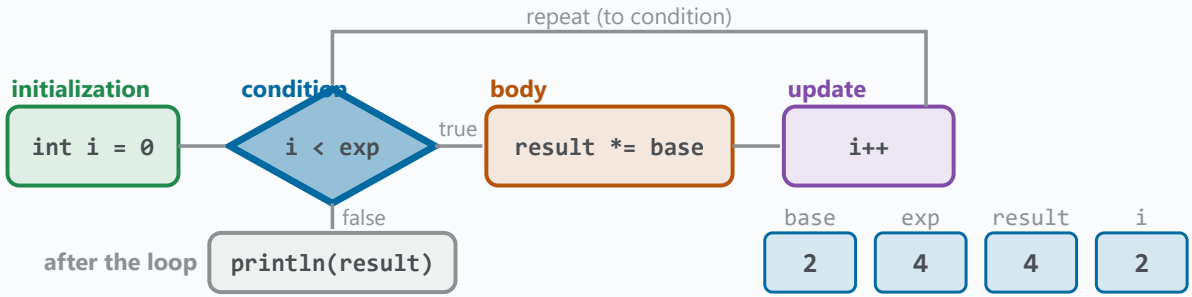
```
1 int base = 2, exp = 4;
2 int result = 1;
▶ for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 4 | i = 2

OUTPUT

Code Trace: 2^4



Condition: 2 < 4 → **true**.

Step 10 / 17

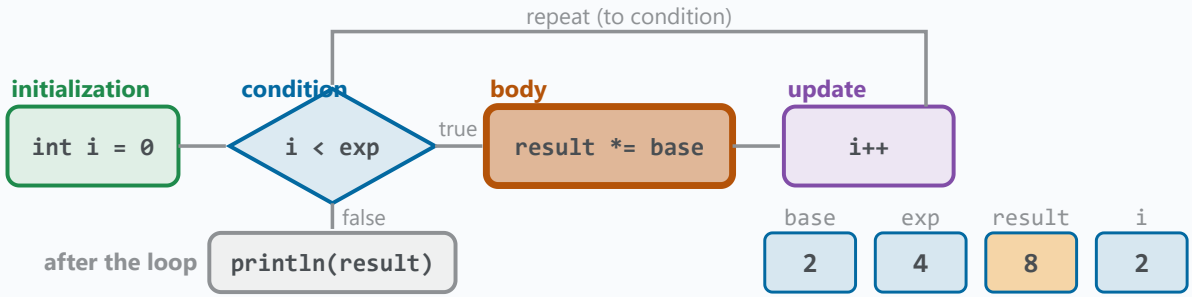
```
1 int base = 2, exp = 4;
2 int result = 1;
▶ for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 4 | i = 2

OUTPUT

Code Trace: 2^4



result = 4 × 2 = **8** (= 2^3).

Step 11 / 17

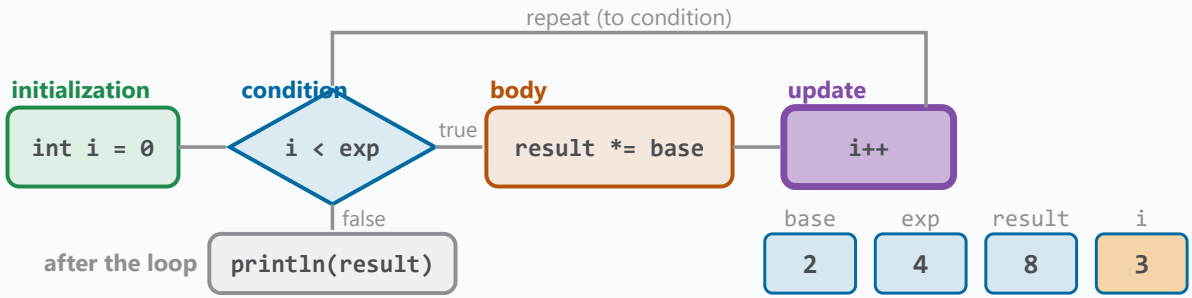
```
1 int base = 2, exp = 4;
2 int result = 1;
3 for (int i = 0; i < exp; i++) {
▶     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 8 | i = 2

OUTPUT

Code Trace: 2^4



Update: i = 3.

Step 12 / 17

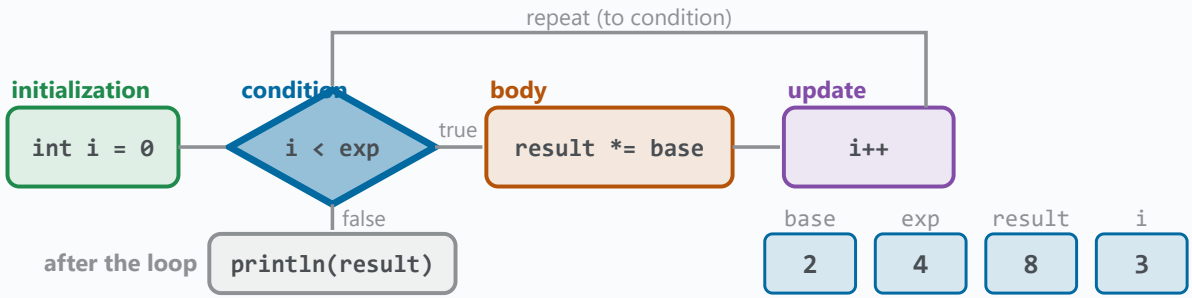
```
1 int base = 2, exp = 4;
2 int result = 1;
▶ for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 8 | i = 3

OUTPUT

Code Trace: 2^4



Condition: 3 < 4 → **true**.

Step 13 / 17

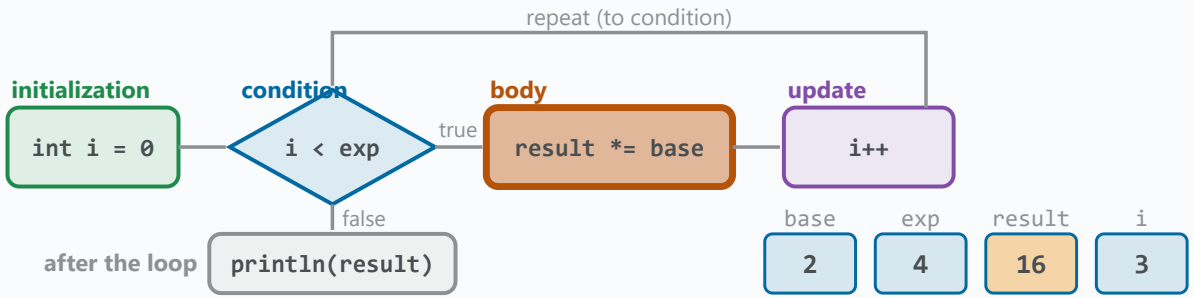
```
1 int base = 2, exp = 4;
2 int result = 1;
▶ for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 8 | i = 3

OUTPUT

Code Trace: 2^4



result = 8 × 2 = **16** (= 2^4).

Step 14 / 17

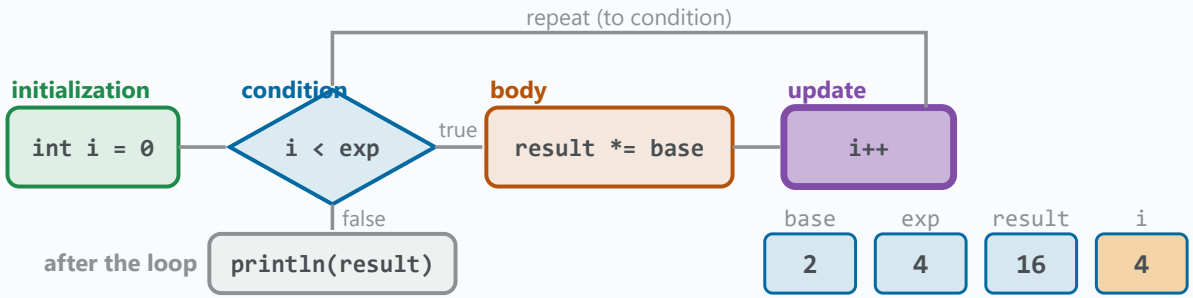
```
1 int base = 2, exp = 4;
2 int result = 1;
3 for (int i = 0; i < exp; i++) {
▶     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 16 | i = 3

OUTPUT

Code Trace: 2^4



Update: i = 4.

Step 15 / 17

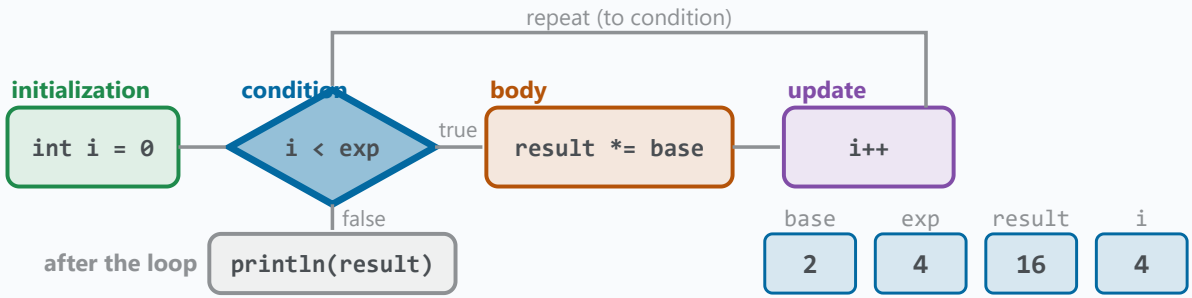
```
1 int base = 2, exp = 4;
2 int result = 1;
▶ for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 16 | i = 4

OUTPUT

Code Trace: 2^4



Condition: 4 < 4 → **false**.

Step 16 / 17

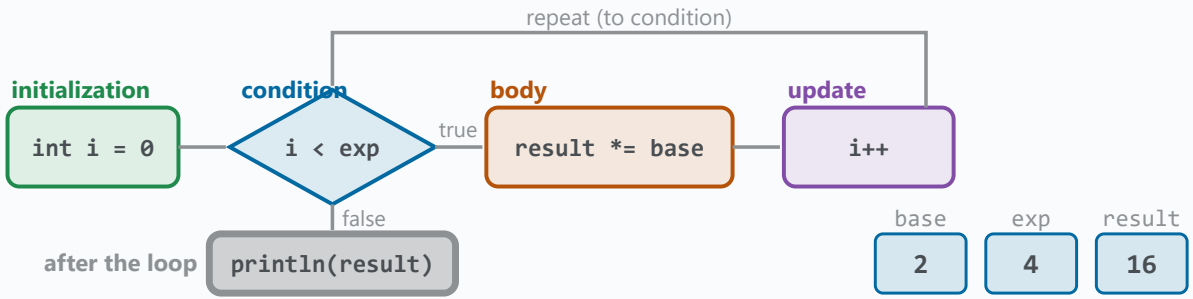
```
1 int base = 2, exp = 4;
2 int result = 1;
▶ for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
6 System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 16 | i = 4

OUTPUT

Code Trace: 2^4



16 is printed. Invariant: at the start of each pass, result = 2^i.

Step 17 / 17

```
1 int base = 2, exp = 4;
2 int result = 1;
3 for (int i = 0; i < exp; i++) {
4     result *= base;
5 }
▶ System.out.println(result);
```

VARIABLES

base = 2 | exp = 4 | result = 16

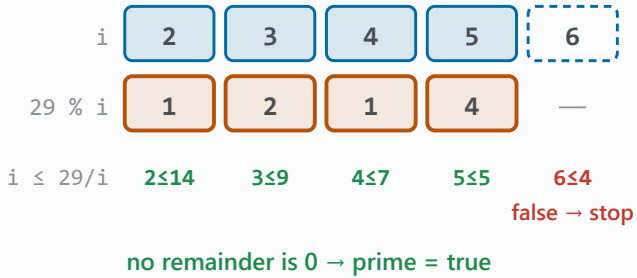
OUTPUT

```
16
```

EXAMPLE

Prime Numbers

```
int n = 29;
boolean prime = n >= 2;
for (int i = 2; prime && i <= n / i; i++)
{
    if (n % i == 0) prime = false;
}
System.out.println(prime);
```



- 0 and 1 are not prime: `prime = n >= 2`.
- Divisors need only be checked up to √n: `i <= n / i` ≡ i × i ≤ n (no overflow risk).
- Thanks to `prime &&`, the loop stops once a divisor is found.

EXAMPLES · STEP THROUGH THE CODE

Code Trace: is 29 prime?



Is **n = 29** prime?

Step 1 / 17

```
▶ int n = 29;
2 boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = 29

OUTPUT

Code Trace: is 29 prime?



29 >= 2 → **true**. Assume prime until proven otherwise.

Step 2 / 17

```
1 int n = 29;
▶ boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 29` `prime = true`

OUTPUT

Code Trace: is 29 prime?



Start: the first divisor candidate is i = 2.

Step 3 / 17

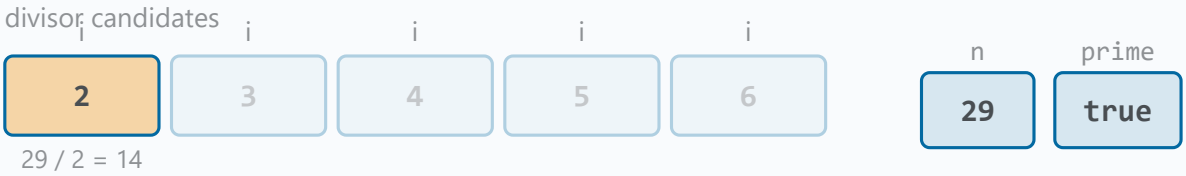
```
1 int n = 29;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = 29 | prime = true | i = 2

OUTPUT

Code Trace: is 29 prime?



Condition: true && 2 <= 29/2 = 14 → **true**.

Step 4 / 17

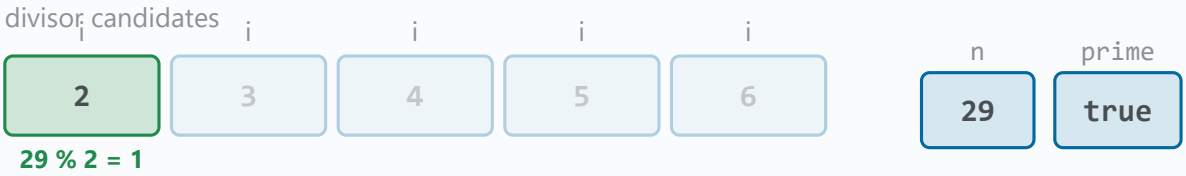
```
1 int n = 29;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 29` `prime = true` `i = 2`

OUTPUT

Code Trace: is 29 prime?



29 % 2 = 1 → not a divisor.

Step 5 / 17

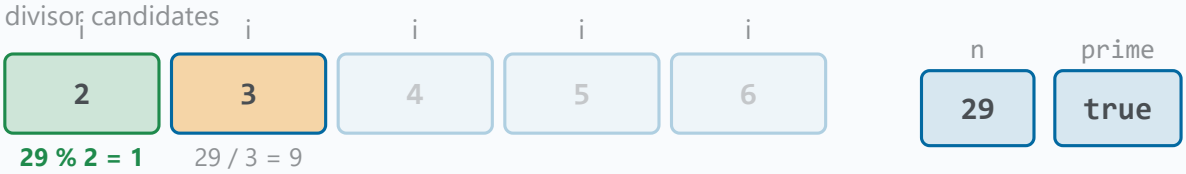
```
1 int n = 29;
2 boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
▶     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 29` `prime = true` `i = 2`

OUTPUT

Code Trace: is 29 prime?



Update: i = 3.

Step 6 / 17

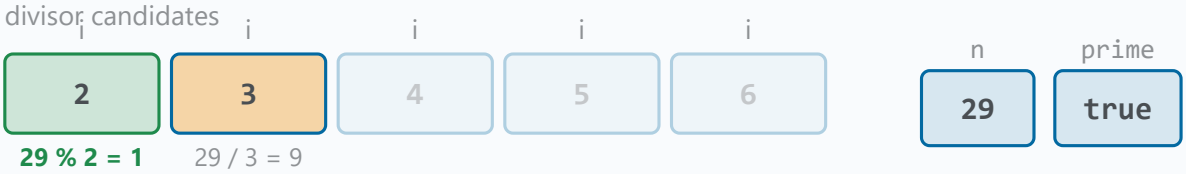
```
1 int n = 29;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = 29 | prime = true | i = 3

OUTPUT

Code Trace: is 29 prime?



Condition: true && 3 <= 29/3 = 9 → **true**.

Step 7 / 17

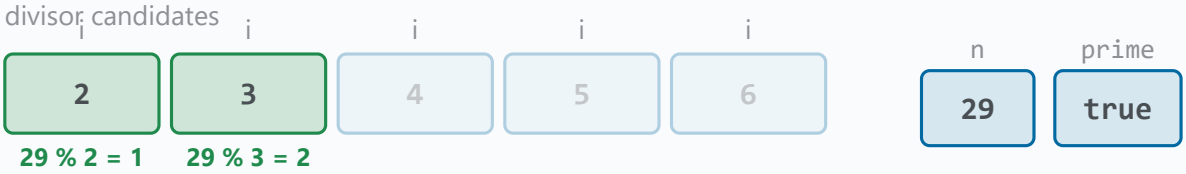
```
1 int n = 29;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 29` `prime = true` `i = 3`

OUTPUT

Code Trace: is 29 prime?



29 % 3 = 2 → not a divisor.

Step 8 / 17

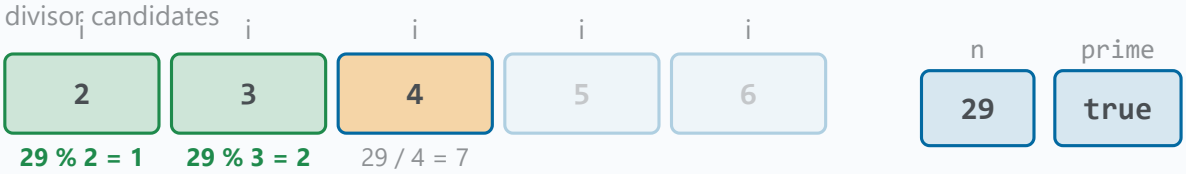
```
1 int n = 29;
2 boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
▶     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = 29 | prime = true | i = 3

OUTPUT

Code Trace: is 29 prime?



Update: i = 4.

Step 9 / 17

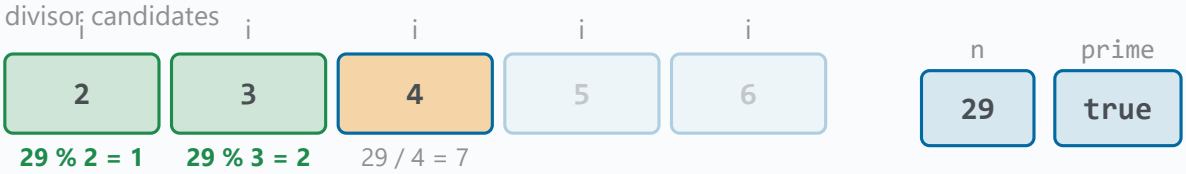
```
1 int n = 29;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = 29 | prime = true | i = 4

OUTPUT

Code Trace: is 29 prime?



Condition: true && 4 <= 29/4 = 7 → **true**.

Step 10 / 17

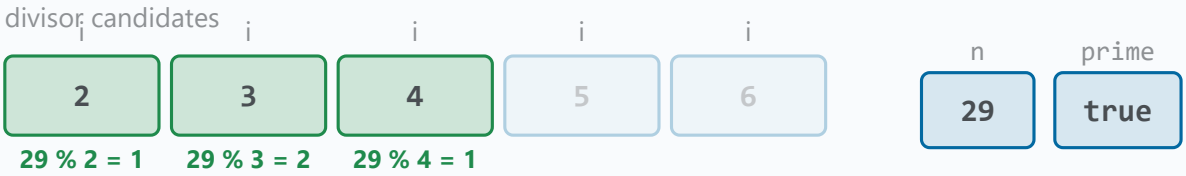
```
1 int n = 29;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 29` `prime = true` `i = 4`

OUTPUT

Code Trace: is 29 prime?



29 % 4 = 1 → not a divisor.

Step 11 / 17

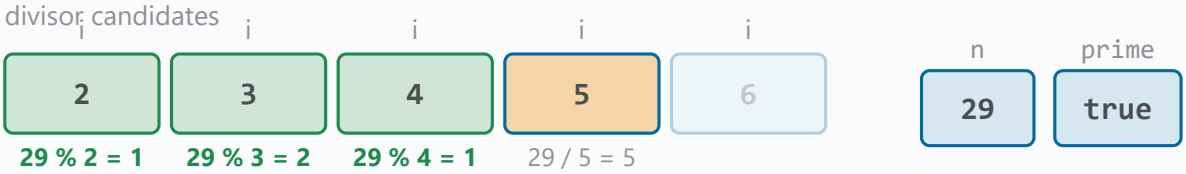
```
1 int n = 29;
2 boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
▶     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = 29 | prime = true | i = 4

OUTPUT

Code Trace: is 29 prime?



Update: i = 5.

Step 12 / 17

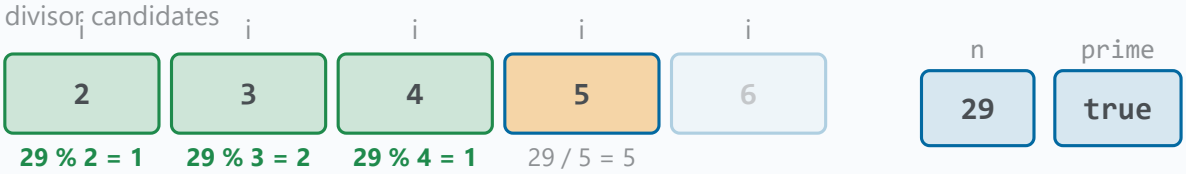
```
1 int n = 29;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = 29 | prime = true | i = 5

OUTPUT

Code Trace: is 29 prime?



Condition: true && 5 <= 29/5 = 5 → **true**.

Step 13 / 17

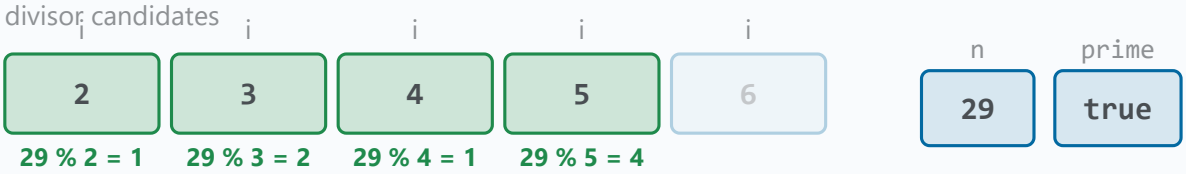
```
1 int n = 29;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 29` `prime = true` `i = 5`

OUTPUT

Code Trace: is 29 prime?



29 % 5 = 4 → not a divisor.

Step 14 / 17

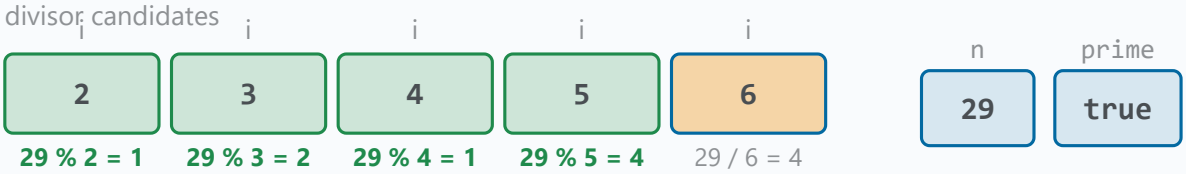
```
1 int n = 29;
2 boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
▶     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 29` `prime = true` `i = 5`

OUTPUT

Code Trace: is 29 prime?



Update: i = 6.

Step 15 / 17

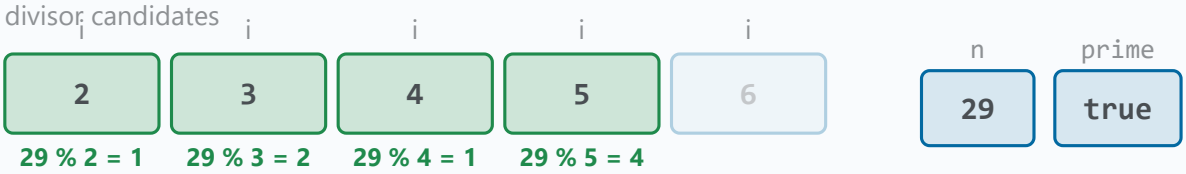
```
1 int n = 29;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 29` `prime = true` `i = 6`

OUTPUT

Code Trace: is 29 prime?



Condition: true && 6 <= 29/6 = 4 → **false**: we are past √29, so no larger divisor needs checking.

Step 16 / 17

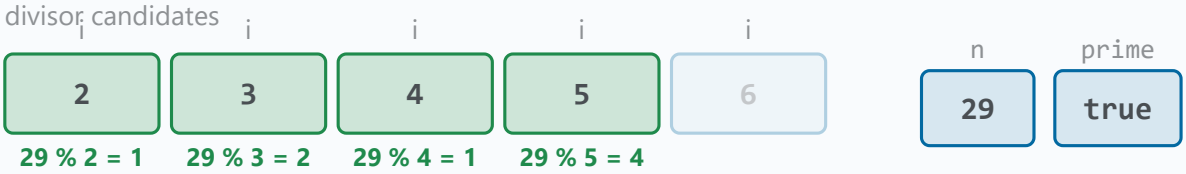
```
1 int n = 29;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 29` `prime = true` `i = 6`

OUTPUT

Code Trace: is 29 prime?



true is printed. 4 candidates were tried.

Step 17 / 17

```
1 int n = 29;
2 boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
▶ System.out.println(prime);
```

VARIABLES

n = **29** prime = **true**

OUTPUT

```
true
```

Code Trace (cont.): is 21 prime?



Is **n = 21** prime?

Step 1 / 11

```
▶ int n = 21;
2 boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = 21

OUTPUT

Code Trace (cont.): is 21 prime?



21 >= 2 → **true**. Assume prime until proven otherwise.

Step 2 / 11

```
1 int n = 21;
▶ boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 21` `prime = true`

OUTPUT

Code Trace (cont.): is 21 prime?



Start: the first divisor candidate is i = 2.

Step 3 / 11

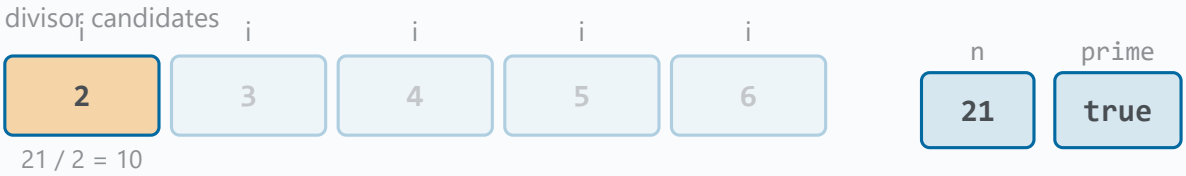
```
1 int n = 21;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = 21 | prime = true | i = 2

OUTPUT

Code Trace (cont.): is 21 prime?



Condition: true && 2 <= 21/2 = 10 → **true**.

Step 4 / 11

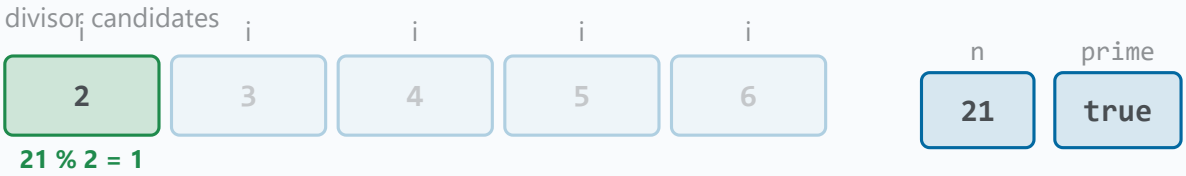
```
1 int n = 21;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 21` `prime = true` `i = 2`

OUTPUT

Code Trace (cont.): is 21 prime?



21 % 2 = 1 → not a divisor.

Step 5 / 11

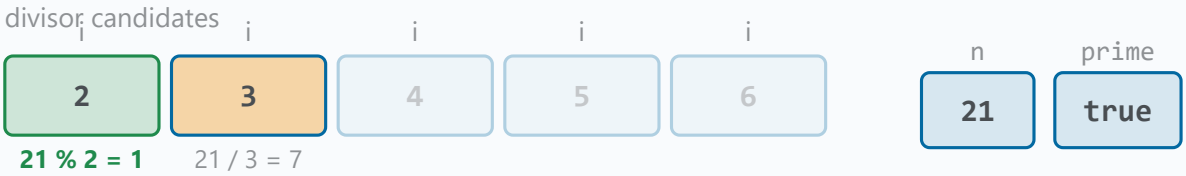
```
1 int n = 21;
2 boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
▶     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 21` `prime = true` `i = 2`

OUTPUT

Code Trace (cont.): is 21 prime?



Update: i = 3.

Step 6 / 11

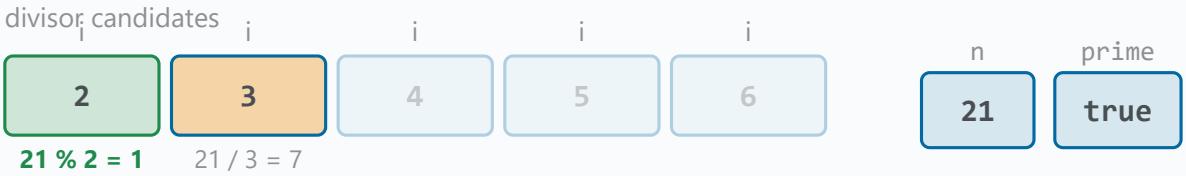
```
1 int n = 21;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = 21 | prime = true | i = 3

OUTPUT

Code Trace (cont.): is 21 prime?



Condition: true && 3 <= 21/3 = 7 → **true**.

Step 7 / 11

```
1 int n = 21;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = **21** | prime = **true** | i = **3**

OUTPUT

Code Trace (cont.): is 21 prime?



21 % 3 = 0 → **divisor found**: prime = false.

Step 8 / 11

```
1 int n = 21;
2 boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
▶     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 21` `prime = false` `i = 3`

OUTPUT

Code Trace (cont.): is 21 prime?



Update: i = 4.

Step 9 / 11

```
1 int n = 21;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

n = 21 | prime = false | i = 4

OUTPUT

Code Trace (cont.): is 21 prime?



Condition: prime = false → && short-circuits, `i <= n / i` is never evaluated → the loop ends.

Step 10 / 11

```
1 int n = 21;
2 boolean prime = n >= 2;
▶ for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
6 System.out.println(prime);
```

VARIABLES

`n = 21` `prime = false` `i = 4`

OUTPUT

Code Trace (cont.): is 21 prime?



false is printed. The loop stopped as soon as a divisor was found.

Step 11 / 11

```
1 int n = 21;
2 boolean prime = n >= 2;
3 for (int i = 2; prime && i <= n / i; i++) {
4     if (n % i == 0) prime = false;
5 }
▶ System.out.println(prime);
```

VARIABLES

`n = 21` `prime = false`

OUTPUT

```
false
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

For each k from 2 to 20, a primality test (inner loop) runs.

Step 1 / 21

```
▶ int n = 20;
2 for (int k = 2; k <= n; k++) {
3     boolean prime = true;
4     for (int i = 2; i <= k / i; i++) {
5         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20

OUTPUT

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 2: inner loop never runs (2 > k / 2) → prime = true → printed.

Step 2 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
      boolean prime = true;
      for (int i = 2; i <= k / i; i++) {
5         if (k % i == 0) { prime = false; break; }
6     }
      if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 2 | prime = true

OUTPUT

```
2
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 3: inner loop never runs (2 > k / 2) → prime = true → printed.


```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
▶     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
5         if (k % i == 0) { prime = false; break; }
6     }
▶     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 3 | prime = true

OUTPUT

```
2 3
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 4: 4 % 2 = 0 → prime = false, **break** (the inner loop ends early).


```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
3     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
▶         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 4 | prime = false | i = 2

OUTPUT

```
2 3
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 5: no divisor in 2 … 2 → prime = true → printed.

Step 5 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
▶     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
5         if (k % i == 0) { prime = false; break; }
6     }
▶     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 5 | prime = true

OUTPUT

```
2 3 5
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 6: 6 % 2 = 0 → prime = false, **break** (the inner loop ends early).


```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
3     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
▶         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 6 | prime = false | i = 2

OUTPUT

```
2 3 5
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 7: no divisor in 2 … 2 → prime = true → printed.

Step 7 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
▶     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
5         if (k % i == 0) { prime = false; break; }
6     }
▶     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 7 | prime = true

OUTPUT

```
2 3 5 7
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 8: 8 % 2 = 0 → prime = false, **break** (the inner loop ends early).

Step 8 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
3     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
▶         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 8 | prime = false | i = 2

OUTPUT

```
2 3 5 7
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 9: 9 % 3 = 0 → prime = false, **break** (the inner loop ends early).


```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
3     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
▶         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 9 | prime = false | i = 3

OUTPUT

```
2 3 5 7
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 10: 10 % 2 = 0 → prime = false, **break** (the inner loop ends early).

Step 10 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
3     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
▶         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 10 | prime = false | i = 2

OUTPUT

```
2 3 5 7
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 11: no divisor in 2 … 3 → prime = true → printed.

Step 11 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
      boolean prime = true;
      for (int i = 2; i <= k / i; i++) {
5         if (k % i == 0) { prime = false; break; }
6     }
      if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 11 | prime = true

OUTPUT

```
2 3 5 7 11
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 12: 12 % 2 = 0 → prime = false, **break** (the inner loop ends early).

Step 12 / 21

```
1  int n = 20;
2  for (int k = 2; k <= n; k++) {
3      boolean prime = true;
▶      for (int i = 2; i <= k / i; i++) {
▶          if (k % i == 0) { prime = false; break; }
6      }
7      if (prime) System.out.print(k + " ");
8  }
```

VARIABLES

n = 20 | k = 12 | prime = false | i = 2

OUTPUT

```
2 3 5 7 11
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 13: no divisor in 2 … 3 → prime = true → printed.

Step 13 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
▶     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
5         if (k % i == 0) { prime = false; break; }
6     }
▶     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 13 | prime = true

OUTPUT

```
2 3 5 7 11 13
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 14: 14 % 2 = 0 → prime = false, **break** (the inner loop ends early).

Step 14 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
3     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
▶         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 14 | prime = false | i = 2

OUTPUT

```
2 3 5 7 11 13
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 15: 15 % 3 = 0 → prime = false, **break** (the inner loop ends early).

Step 15 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
3     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
▶         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 15 | prime = false | i = 3

OUTPUT

```
2 3 5 7 11 13
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 16: 16 % 2 = 0 → prime = false, **break** (the inner loop ends early).

Step 16 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
3     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
▶         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 16 | prime = false | i = 2

OUTPUT

```
2 3 5 7 11 13
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 17: no divisor in 2 … 4 → prime = true → printed.

Step 17 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
▶     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
5         if (k % i == 0) { prime = false; break; }
6     }
▶     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

`n = 20` `k = 17` `prime = true`

OUTPUT

```
2 3 5 7 11 13 17
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 18: 18 % 2 = 0 → prime = false, **break** (the inner loop ends early).

Step 18 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
3     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
▶         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 18 | prime = false | i = 2

OUTPUT

```
2 3 5 7 11 13 17
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 19: no divisor in 2 … 4 → prime = true → printed.

Step 19 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
▶     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
5         if (k % i == 0) { prime = false; break; }
6     }
▶     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 19 | prime = true

OUTPUT

```
2 3 5 7 11 13 17 19
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 20: 20 % 2 = 0 → prime = false, **break** (the inner loop ends early).

Step 20 / 21

```
1 int n = 20;
2 for (int k = 2; k <= n; k++) {
3     boolean prime = true;
▶     for (int i = 2; i <= k / i; i++) {
▶         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = 20 | k = 20 | prime = false | i = 2

OUTPUT

```
2 3 5 7 11 13 17 19
```

Listing the Primes from 1 to n (n = 20)

green = prime, faded = composite, dashed = not checked yet



k = 21 > 20: done. There are **8** primes in 2..20. (The sieve of Eratosthenes is faster.)

Step 21 / 21

```
1 int n = 20;
▶ for (int k = 2; k <= n; k++) {
3     boolean prime = true;
4     for (int i = 2; i <= k / i; i++) {
5         if (k % i == 0) { prime = false; break; }
6     }
7     if (prime) System.out.print(k + " ");
8 }
```

VARIABLES

n = **20**

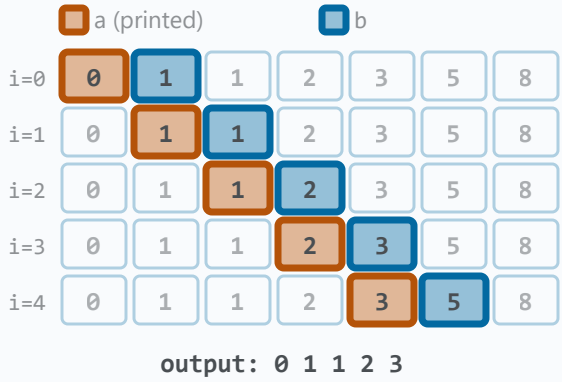
OUTPUT

```
2 3 5 7 11 13 17 19
```

EXAMPLE

The Fibonacci Sequence

```
int terms = 5;
long a = 0, b = 1;
for (int i = 0; i < terms; i++) {
    System.out.print(a + " ");
    long next = a + b;
    a = b;
    b = next;
}
```



output: 0 1 1 2 3

- Each term is the sum of the previous two: 0 1 1 2 3 5 8 ...
- The update order keeps both previous values; writing `b = next` first would lose the old b.
- The sequence grows fast; even long overflows after term 93.

EXAMPLES · STEP THROUGH THE CODE

Code Trace: the first 5 Fibonacci numbers



We will print the first **5** Fibonacci numbers.

Step 1 / 18

```
▶ int terms = 5;
2 long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
5     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

`terms = 5`

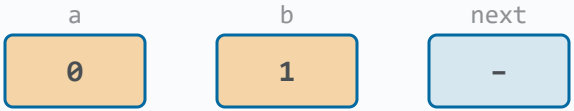
OUTPUT

EXAMPLES · STEP THROUGH THE CODE

Code Trace: the first 5 Fibonacci numbers

printed sequence





The first two terms: a = 0, b = 1.

Step 2 / 18

```
1 int terms = 5;
▶ long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
5     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

terms = 5 | a = 0 | b = 1

OUTPUT

Code Trace: the first 5 Fibonacci numbers



i = 0 < 5 → true. Print the next term, **a = 0**.

Step 3 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
▶ for (int i = 0; i < terms; i++) {
▶     System.out.print(a + " ");
5     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

terms = 5 | a = 0 | b = 1 | i = 0

OUTPUT

```
0
```

Code Trace: the first 5 Fibonacci numbers



next = 0 + 1 = **1**.

Step 4 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
▶     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

terms = 5 | a = 0 | b = 1 | i = 0 | next = 1

OUTPUT

```
0
```

Code Trace: the first 5 Fibonacci numbers



Shift: a = 1, b = 1. Order matters: a = b first, then b = next. Then i++.

Step 5 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
5     long next = a + b;
▶     a = b;
▶     b = next;
8 }
```

VARIABLES

terms = 5 | a = 1 | b = 1 | i = 0 | next = 1

OUTPUT

```
0
```

Code Trace: the first 5 Fibonacci numbers



i = 1 < 5 → true. Print the next term, **a = 1**.

Step 6 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
▶ for (int i = 0; i < terms; i++) {
▶     System.out.print(a + " ");
5     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

terms = **5** | a = **1** | b = **1** | i = **1**

OUTPUT

```
0 1
```

Code Trace: the first 5 Fibonacci numbers



next = 1 + 1 = **2**.

Step 7 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
▶     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

`terms = 5` `a = 1` `b = 1` `i = 1` `next = 2`

OUTPUT

```
0 1
```

Code Trace: the first 5 Fibonacci numbers



Shift: a = 1, b = 2. Order matters: a = b first, then b = next. Then i++.

Step 8 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
5     long next = a + b;
▶     a = b;
▶     b = next;
8 }
```

VARIABLES

terms = 5 | a = 1 | b = 2 | i = 1 | next = 2

OUTPUT

```
0 1
```

Code Trace: the first 5 Fibonacci numbers



i = 2 < 5 → true. Print the next term, **a = 1**.

Step 9 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
▶ for (int i = 0; i < terms; i++) {
▶     System.out.print(a + " ");
5     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

`terms = 5` `a = 1` `b = 2` `i = 2`

OUTPUT

```
0 1 1
```

Code Trace: the first 5 Fibonacci numbers



next = 1 + 2 = **3**.

Step 10 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
▶     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

terms = 5 | a = 1 | b = 2 | i = 2 | next = 3

OUTPUT

```
0 1 1
```

Code Trace: the first 5 Fibonacci numbers



Shift: a = 2, b = 3. Order matters: a = b first, then b = next. Then i++.

Step 11 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
5     long next = a + b;
▶     a = b;
▶     b = next;
8 }
```

VARIABLES

terms = 5 | a = 2 | b = 3 | i = 2 | next = 3

OUTPUT

```
0 1 1
```

Code Trace: the first 5 Fibonacci numbers



i = 3 < 5 → true. Print the next term, **a = 2**.

Step 12 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
▶ for (int i = 0; i < terms; i++) {
▶     System.out.print(a + " ");
5     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

terms = 5 | a = 2 | b = 3 | i = 3

OUTPUT

```
0 1 1 2
```

Code Trace: the first 5 Fibonacci numbers



next = 2 + 3 = **5**.

Step 13 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
▶     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

terms = 5 | a = 2 | b = 3 | i = 3 | next = 5

OUTPUT

```
0 1 1 2
```

Code Trace: the first 5 Fibonacci numbers



Shift: a = 3, b = 5. Order matters: a = b first, then b = next. Then i++.

Step 14 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
5     long next = a + b;
▶     a = b;
▶     b = next;
8 }
```

VARIABLES

terms = 5 | a = 3 | b = 5 | i = 3 | next = 5

OUTPUT

```
0 1 1 2
```

Code Trace: the first 5 Fibonacci numbers

printed sequence





i = 4 < 5 → true. Print the next term, **a = 3**.

Step 15 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
▶ for (int i = 0; i < terms; i++) {
▶     System.out.print(a + " ");
5     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

terms = 5 | a = 3 | b = 5 | i = 4

OUTPUT

```
0 1 1 2 3
```

Code Trace: the first 5 Fibonacci numbers



next = 3 + 5 = **8**.

Step 16 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
▶     long next = a + b;
6     a = b;
7     b = next;
8 }
```

VARIABLES

terms = 5 | a = 3 | b = 5 | i = 4 | next = 8

OUTPUT

```
0 1 1 2 3
```

Code Trace: the first 5 Fibonacci numbers



Shift: a = 5, b = 8. Order matters: a = b first, then b = next. Then i++.

Step 17 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
3 for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
5     long next = a + b;
▶     a = b;
▶     b = next;
8 }
```

VARIABLES

terms = 5 | a = 5 | b = 8 | i = 4 | next = 8

OUTPUT

```
0 1 1 2 3
```

Code Trace: the first 5 Fibonacci numbers



Condition: 5 < 5 → **false** → the loop ends. Output: 0 1 1 2 3.

Step 18 / 18

```
1 int terms = 5;
2 long a = 0, b = 1;
▶ for (int i = 0; i < terms; i++) {
4     System.out.print(a + " ");
5     long next = a + b;
6     a = b;
7     b = next;
8 }
```

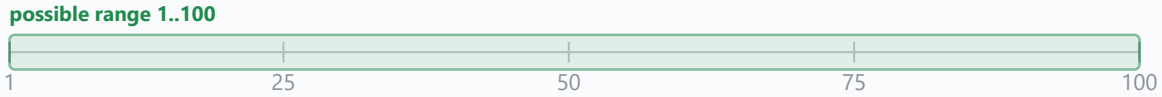
VARIABLES

`terms = 5` `a = 5` `b = 8` `i = 5`

OUTPUT

```
0 1 1 2 3
```

Number Guessing Game (while + Random + Scanner)



`Random` generates random numbers (`import java.util.Random;` and `java.util.Scanner` at the top).

Step 1 / 15

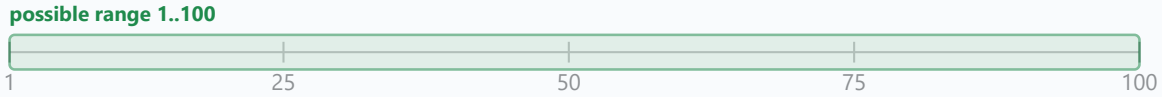
```
▶ Random rnd = new Random();
2 Scanner in = new Scanner(System.in);
3 int target = rnd.nextInt(100) + 1;  // 1..100
4 int guess = 0, tries = 0;
5 while (guess != target) {
6   guess = in.nextInt();
7   tries++;
8   if (guess < target) System.out.println("Higher");
9   if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

VARIABLES

`rnd = Random`

OUTPUT

Number Guessing Game (while + Random + Scanner)



A Scanner to read from the keyboard.

Step 2 / 15

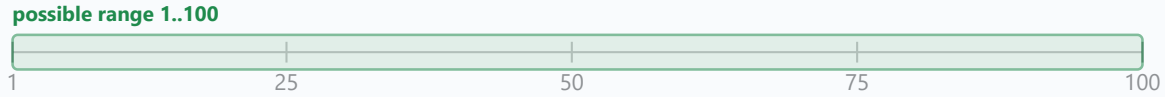
```
1  Random rnd = new Random();
▶ Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
5  while (guess != target) {
6    guess = in.nextInt();
7    tries++;
8    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

VARIABLES

`rnd = Random` `in = Scanner`

OUTPUT

Number Guessing Game (while + Random + Scanner)



target = 37 (hidden)

`nextInt(100)` gives 0..99; +1 makes it 1..100. This run picked **target = 37** (it differs every run).

Step 3 / 15

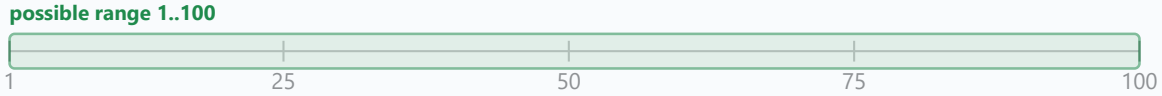
```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
▶ int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
5  while (guess != target) {
6    guess = in.nextInt();
7    tries++;
8    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

VARIABLES

`target = 37`

OUTPUT

Number Guessing Game (while + Random + Scanner)



target = 37 (hidden)

guess = 0 can never be the target → the loop is sure to start.

Step 4 / 15

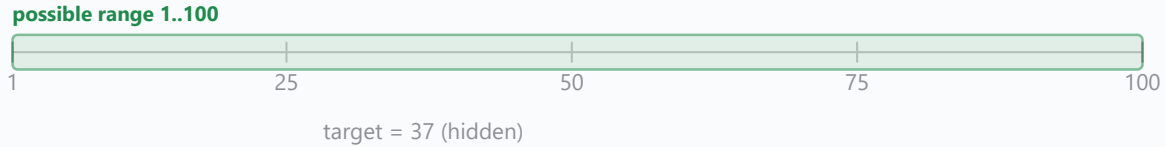
```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
▶ 4  int guess = 0, tries = 0;
5  while (guess != target) {
6    guess = in.nextInt();
7    tries++;
8    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

VARIABLES

`target = 37` `guess = 0` `tries = 0`

OUTPUT

Number Guessing Game (while + Random + Scanner)



Condition: 0 != 37 → **true**.

Step 5 / 15

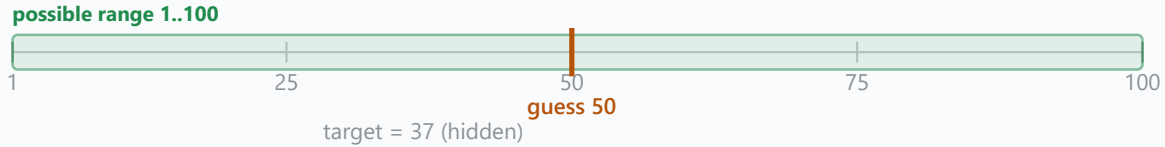
```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
▶  while (guess != target) {
6    guess = in.nextInt();
7    tries++;
8    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

VARIABLES

`target = 37` `guess = 0` `tries = 0`

OUTPUT

Number Guessing Game (while + Random + Scanner)



The user types **50**; tries = 1.

Step 6 / 15

```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
5  while (guess != target) {
▶    guess = in.nextInt();
▶    tries++;
8    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

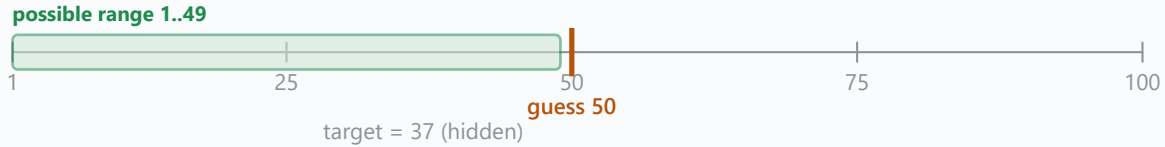
VARIABLES

`target = 37` `guess = 50` `tries = 1`

OUTPUT

```
50
```

Number Guessing Game (while + Random + Scanner)



50 > 37 → «Lower». Possible range 1..49. Guessing the middle each time halves the range.

Step 7 / 15

```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
5  while (guess != target) {
6    guess = in.nextInt();
7    tries++;
8    if (guess < target) System.out.println("Higher");
▶    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

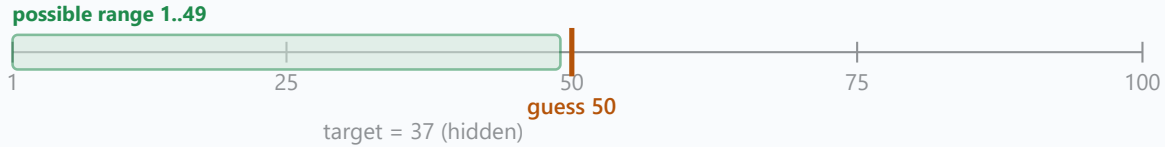
VARIABLES

`target = 37` `guess = 50` `tries = 1`

OUTPUT

```
50
Lower
```

Number Guessing Game (while + Random + Scanner)



Condition: 50 != 37 → **true**.

Step 8 / 15

```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
▶  while (guess != target) {
6    guess = in.nextInt();
7    tries++;
8    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

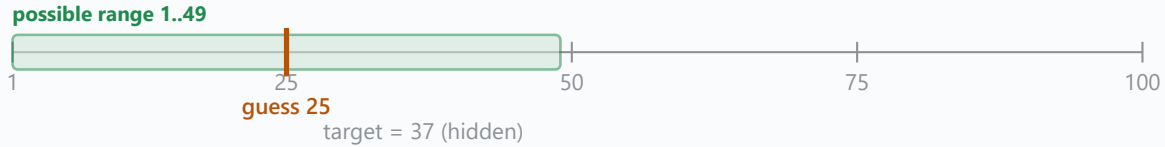
VARIABLES

`target = 37` `guess = 50` `tries = 1`

OUTPUT

```
50
Lower
```

Number Guessing Game (while + Random + Scanner)



The user types **25**; tries = 2.

Step 9 / 15

```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
5  while (guess != target) {
▶    guess = in.nextInt();
▶    tries++;
8    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

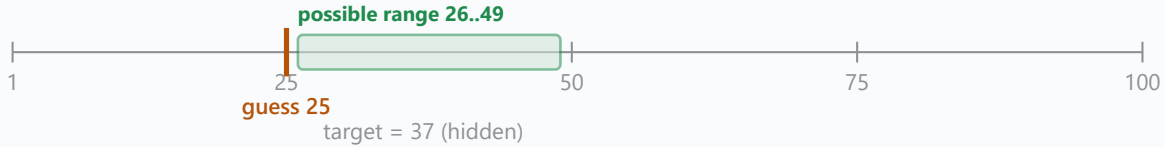
VARIABLES

`target = 37` `guess = 25` `tries = 2`

OUTPUT

```
50
Lower
25
```

Number Guessing Game (while + Random + Scanner)



25 < 37 → «Higher». Possible range 26..49.

Step 10 / 15

```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
5  while (guess != target) {
6    guess = in.nextInt();
7    tries++;
▶    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

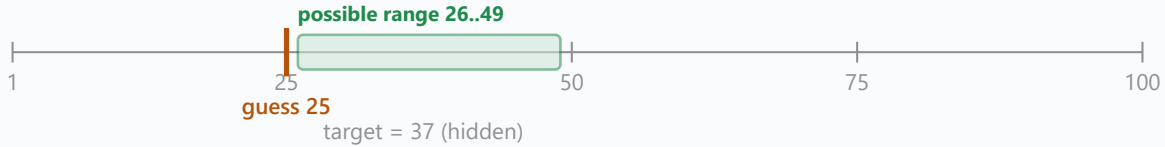
VARIABLES

`target = 37` `guess = 25` `tries = 2`

OUTPUT

```
50
Lower
25
Higher
```

Number Guessing Game (while + Random + Scanner)



Condition: 25 != 37 → **true**.

Step 11 / 15

```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
▶  while (guess != target) {
6    guess = in.nextInt();
7    tries++;
8    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

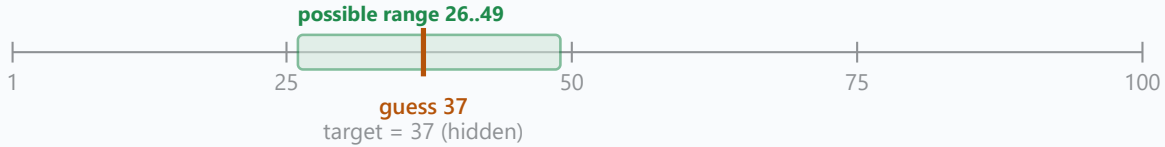
VARIABLES

`target = 37` `guess = 25` `tries = 2`

OUTPUT

```
50
Lower
25
Higher
```

Number Guessing Game (while + Random + Scanner)



The user types **37**; tries = 3.

Step 12 / 15

```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
5  while (guess != target) {
▶    guess = in.nextInt();
▶    tries++;
8    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

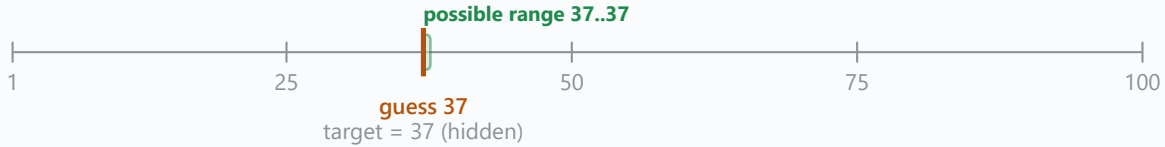
VARIABLES

`target = 37` `guess = 37` `tries = 3`

OUTPUT

```
50
Lower
25
Higher
37
```

Number Guessing Game (while + Random + Scanner)



37 is neither smaller nor larger: both ifs are false, no message.

Step 13 / 15

```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
5  while (guess != target) {
6    guess = in.nextInt();
7    tries++;
▶    if (guess < target) System.out.println("Higher");
▶    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

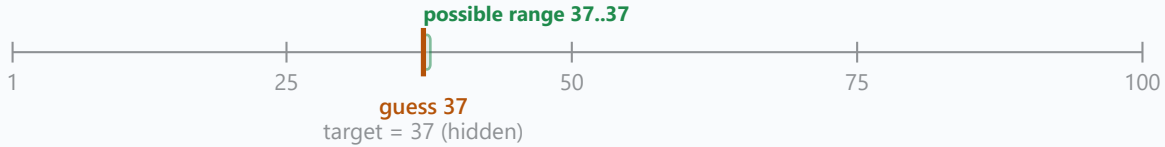
VARIABLES

`target = 37` `guess = 37` `tries = 3`

OUTPUT

```
50
Lower
25
Higher
37
```

Number Guessing Game (while + Random + Scanner)



Condition: 37 != 37 → **false** → the loop ends.

Step 14 / 15

```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
▶  while (guess != target) {
6    guess = in.nextInt();
7    tries++;
8    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
11 System.out.println("Got it in " + tries + " tries");
```

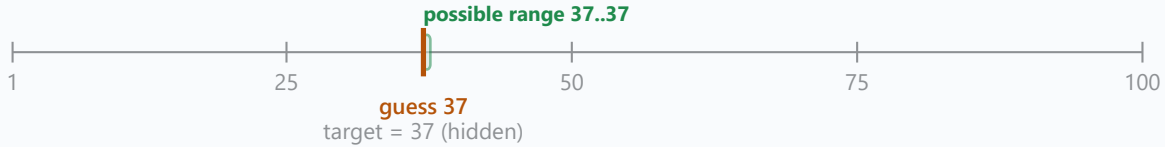
VARIABLES

`target = 37` `guess = 37` `tries = 3`

OUTPUT

```
50
Lower
25
Higher
37
```

Number Guessing Game (while + Random + Scanner)



The number of passes is unknown in advance → **while** is the right choice.

Step 15 / 15

```
1  Random rnd = new Random();
2  Scanner in = new Scanner(System.in);
3  int target = rnd.nextInt(100) + 1;  // 1..100
4  int guess = 0, tries = 0;
5  while (guess != target) {
6    guess = in.nextInt();
7    tries++;
8    if (guess < target) System.out.println("Higher");
9    if (guess > target) System.out.println("Lower");
10 }
▶  System.out.println("Got it in " + tries + " tries");
```

VARIABLES

`target = 37` `guess = 37` `tries = 3`

OUTPUT

```
50
Lower
25
Higher
37
Got it in 3 tries
```

EXAMPLES · STEP THROUGH THE CODE

Sentinel Value: `while (true)` + `break`, input 5, 3, 0

Input queue (numbers to be typed): **5, 3, 0**. The condition is `true`, so the loop never ends by itself; `break` provides the exit.

Step 1 / 7

```
▶ Scanner sc = new Scanner(System.in);
▶ int sum = 0;
3 while (true) {
4     int num = sc.nextInt();
5     if (num == 0) break;
6     sum += num;
7 }
8 System.out.println("Sum: " + sum);
```

VARIABLES

`sum = 0` `input = [5, 3, 0]`

OUTPUT

Sentinel Value: `while (true)` + `break`, input 5, 3, 0

`nextInt()` reads **5** from the queue. 5 == 0 → false, no break.

Step 2 / 7

```
1 Scanner sc = new Scanner(System.in);
2 int sum = 0;
3 while (true) {
▶     int num = sc.nextInt();
▶     if (num == 0) break;
6     sum += num;
7 }
8 System.out.println("Sum: " + sum);
```

VARIABLES

`sum = 0` `num = 5`

`input = [3, 0]`

OUTPUT

Sentinel Value: `while (true)` + `break`, input 5, 3, 0

sum = 0 + 5 = 5. Back to the top (true).

Step 3 / 7

```
1 Scanner sc = new Scanner(System.in);
2 int sum = 0;
3 while (true) {
4     int num = sc.nextInt();
5     if (num == 0) break;
▶     sum += num;
7 }
8 System.out.println("Sum: " + sum);
```

VARIABLES

`sum = 5` `num = 5`

`input = [3, 0]`

OUTPUT

Sentinel Value: `while (true)` + `break`, input 5, 3, 0

`nextInt()` → **3**. 3 == 0 → false.

Step 4 / 7

```
1 Scanner sc = new Scanner(System.in);
2 int sum = 0;
3 while (true) {
▶     int num = sc.nextInt();
▶     if (num == 0) break;
6     sum += num;
7 }
8 System.out.println("Sum: " + sum);
```

VARIABLES

`sum = 5` `num = 3` `input = [0]`

OUTPUT

Sentinel Value: `while (true)` + `break`, input 5, 3, 0

sum = 5 + 3 = 8.

Step 5 / 7

```
1 Scanner sc = new Scanner(System.in);
2 int sum = 0;
3 while (true) {
4     int num = sc.nextInt();
5     if (num == 0) break;
▶     sum += num;
7 }
8 System.out.println("Sum: " + sum);
```

VARIABLES

`sum = 8` `num = 3` `input = [0]`

OUTPUT

Sentinel Value: `while (true)` + `break`, input 5, 3, 0

`nextInt()` → **0** (the sentinel). 0 == 0 → true → `break`: exit the loop; 0 is not added.

Step 6 / 7

```
1 Scanner sc = new Scanner(System.in);
2 int sum = 0;
3 while (true) {
▶     int num = sc.nextInt();
▶     if (num == 0) break;
6     sum += num;
7 }
8 System.out.println("Sum: " + sum);
```

VARIABLES

`sum = 8` `num = 0` `input = []`

OUTPUT

EXAMPLES · STEP THROUGH THE CODE

Sentinel Value: `while (true)` + `break`, input 5, 3, 0

`Sum: 8` is printed. A sentinel marks the end of the data and is not part of it.

Step 7 / 7

```
1 Scanner sc = new Scanner(System.in);
2 int sum = 0;
3 while (true) {
4     int num = sc.nextInt();
5     if (num == 0) break;
6     sum += num;
7 }
▶ System.out.println("Sum: " + sum);
```

VARIABLES

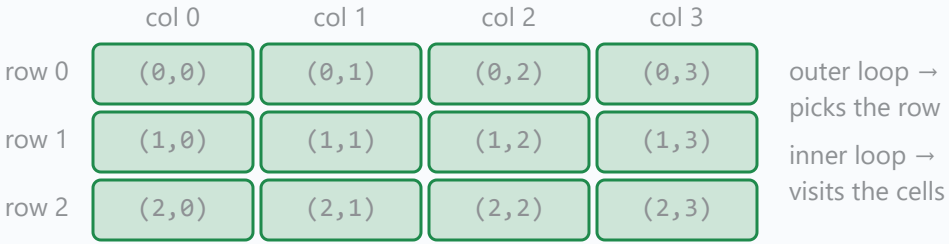
`sum = 8`

OUTPUT

```
Sum: 8
```

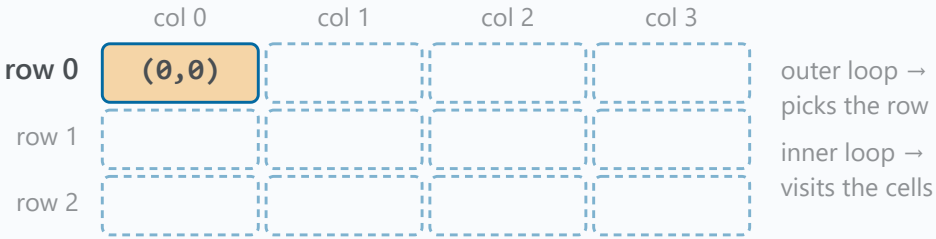
Nested Loops

```
for (int row = 0; row < 3; row++) {
  for (int col = 0; col < 4; col++) {
    System.out.printf("(%d,%d) ", row, col);
  }
  System.out.println();
}
```



- The inner loop runs to completion on **every** pass of the outer loop.
- The body runs rows × cols = 3 × 4 = 12 times.

Code Trace: a 3 × 4 grid



Outer loop **row = 0**; the inner loop starts at col = 0 → (0,0).

Step 1 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

VARIABLES

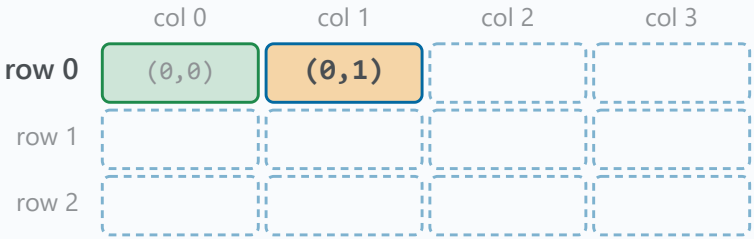
`row = 0` `col = 0`

`printf count = 1`

OUTPUT

```
(0,0)
```

Code Trace: a 3 × 4 grid



col++ → 1 < 4 → true → **(0,1)** is printed.

Step 2 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

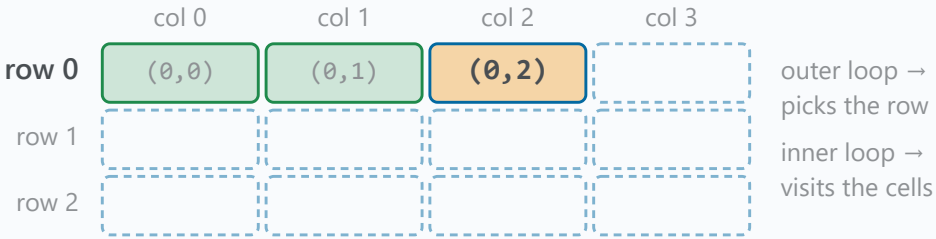
VARIABLES

`row = 0` `col = 1` `printf count = 2`

OUTPUT

```
(0,0) (0,1)
```

Code Trace: a 3 × 4 grid



col++ → 2 < 4 → true → **(0,2)** is printed.

Step 3 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

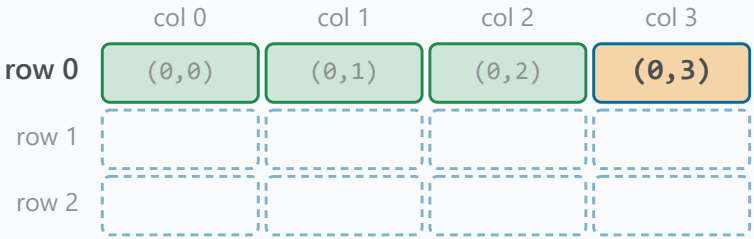
VARIABLES

`row = 0` `col = 2` `printf count = 3`

OUTPUT

```
(0,0) (0,1) (0,2)
```

Code Trace: a 3 × 4 grid



col++ → 3 < 4 → true → **(0,3)** is printed.

Step 4 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

VARIABLES

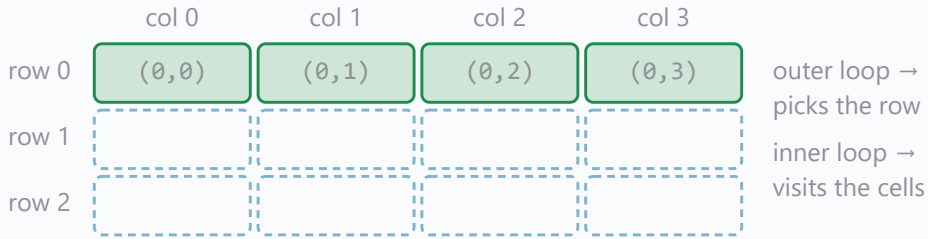
`row = 0` `col = 3`

`printf count = 4`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
```

Code Trace: a 3 × 4 grid



col = 4 < 4 → false: the inner loop is done. `println` ends the line.

Step 5 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
3     System.out.printf("(%d,%d) ", row, col);
4   }
▶   System.out.println();
6 }
```

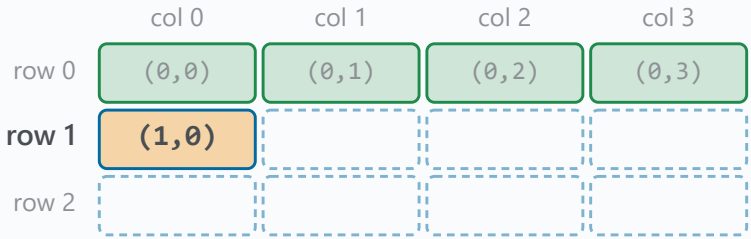
VARIABLES

`row = 0` `printf count = 4`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
```

Code Trace: a 3 × 4 grid



row++ → **row = 1**; the inner loop **restarts** at col = 0.

Step 6 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

VARIABLES

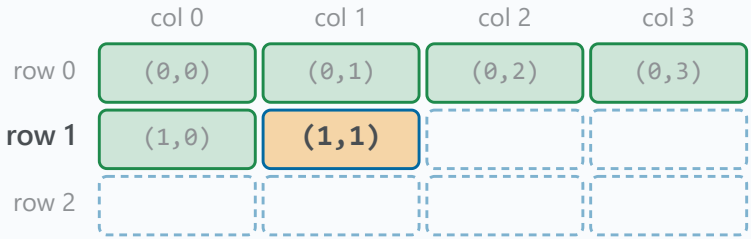
`row = 1` `col = 0`

`printf count = 5`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
(1,0)
```

Code Trace: a 3 × 4 grid



col++ → 1 < 4 → true → **(1,1)** is printed.

Step 7 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

VARIABLES

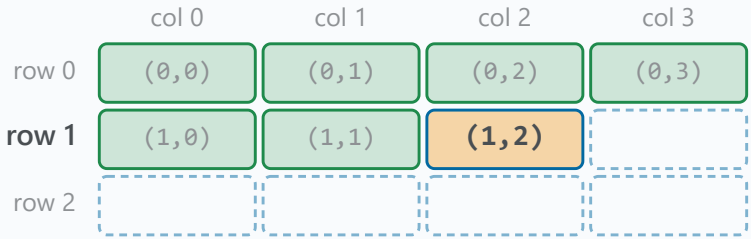
`row = 1` `col = 1`

`printf count = 6`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1)
```

Code Trace: a 3 × 4 grid



col++ → 2 < 4 → true → **(1,2)** is printed.

Step 8 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

VARIABLES

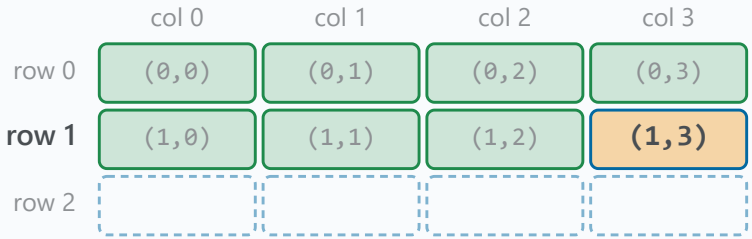
`row = 1` `col = 2`

`printf count = 7`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2)
```

Code Trace: a 3 × 4 grid



col++ → 3 < 4 → true → **(1,3)** is printed.

Step 9 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

VARIABLES

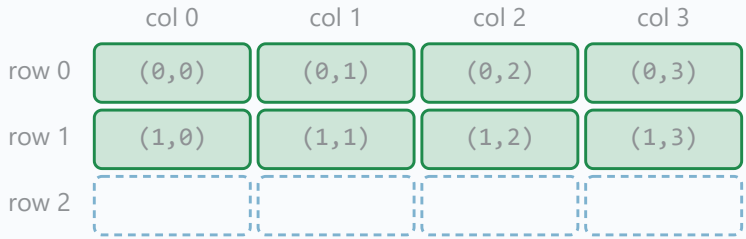
`row = 1` `col = 3`

`printf count = 8`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2) (1,3)
```

Code Trace: a 3 × 4 grid



col = 4 < 4 → false: the inner loop is done. `println` ends the line.

Step 10 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
3     System.out.printf("(%d,%d) ", row, col);
4   }
▶   System.out.println();
6 }
```

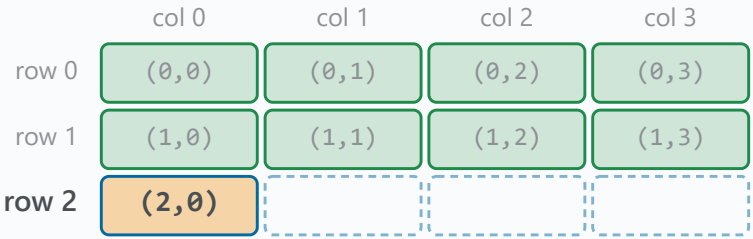
VARIABLES

`row = 1` `printf count = 8`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2) (1,3)
```

Code Trace: a 3 × 4 grid



row++ → **row = 2**; the inner loop **restarts** at col = 0.

Step 11 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

VARIABLES

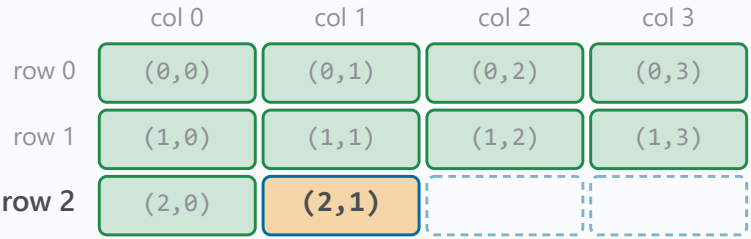
`row = 2` `col = 0`

`printf count = 9`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2) (1,3)
(2,0)
```

Code Trace: a 3 × 4 grid



col++ → 1 < 4 → true → **(2,1)** is printed.

Step 12 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

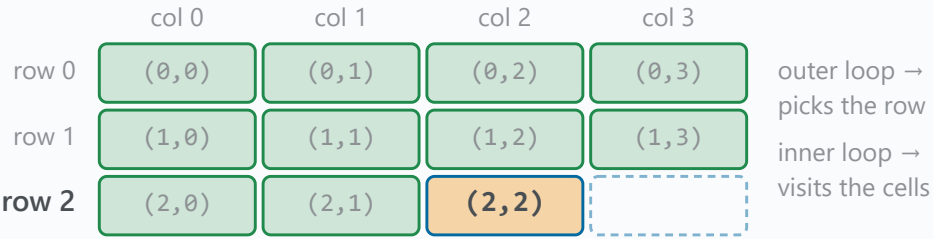
VARIABLES

`row = 2` `col = 1` `printf count = 10`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2) (1,3)
(2,0) (2,1)
```

Code Trace: a 3 × 4 grid



col++ → 2 < 4 → true → **(2,2)** is printed.

Step 13 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

VARIABLES

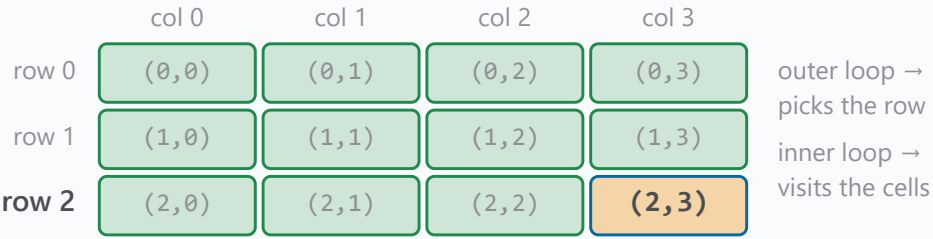
`row = 2` `col = 2`

`printf count = 11`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2) (1,3)
(2,0) (2,1) (2,2)
```

Code Trace: a 3 × 4 grid



col++ → 3 < 4 → true → **(2,3)** is printed.

Step 14 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
▶     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

VARIABLES

`row = 2` `col = 3`

`printf count = 12`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2) (1,3)
(2,0) (2,1) (2,2) (2,3)
```

Code Trace: a 3 × 4 grid



col = 4 < 4 → false: the inner loop is done. `println` ends the line.

Step 15 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
3     System.out.printf("(%d,%d) ", row, col);
4   }
▶   System.out.println();
6 }
```

VARIABLES

`row = 2` `printf count = 12`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2) (1,3)
(2,0) (2,1) (2,2) (2,3)
```

Code Trace: a 3 × 4 grid



row = 3 < 3 → false: the outer loop is done too. 3 × 4 = **12** printf calls in all.

Step 16 / 16

```
1 for (int row = 0; row < 3; row++) {
2   for (int col = 0; col < 4; col++) {
3     System.out.printf("(%d,%d) ", row, col);
4   }
5   System.out.println();
6 }
```

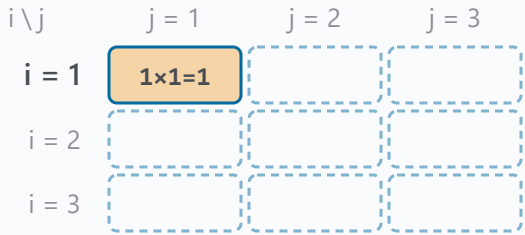
VARIABLES

`printf count = 12`

OUTPUT

```
(0,0) (0,1) (0,2) (0,3)
(1,0) (1,1) (1,2) (1,3)
(2,0) (2,1) (2,2) (2,3)
```

Multiplication Table (nested loops, 3 × 3)



cell (i, j) = i × j

%4d: width of 4 characters

Outer loop **i = 1**; inner loop starts at j = 1: 1 × 1 = 1.

Step 1 / 13

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
▶         System.out.printf("%4d", i * j);
4     }
5     System.out.println();
6 }
```

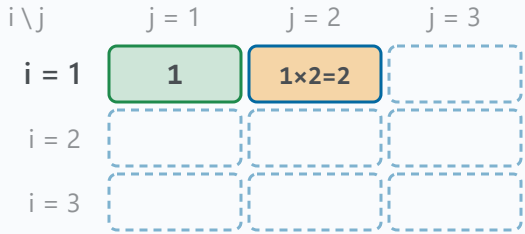
VARIABLES

`i = 1` `j = 1`

OUTPUT

```
   1
```

Multiplication Table (nested loops, 3 × 3)



cell (i, j) = i × j

%4d: width of 4 characters

j = 2: 1 × 2 = **2**, right-aligned in a 4-character field.

Step 2 / 13

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
          System.out.printf("%4d", i * j);
4     }
5     System.out.println();
6 }
```

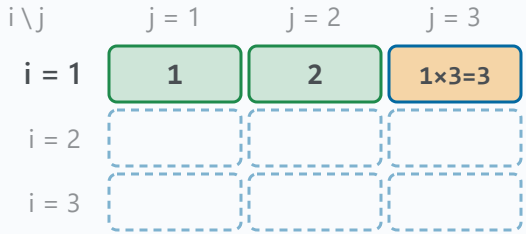
VARIABLES

`i = 1` `j = 2`

OUTPUT

```
   1   2
```

Multiplication Table (nested loops, 3 × 3)



cell (i, j) = i × j

%4d: width of 4 characters

j = 3: 1 × 3 = **3**, right-aligned in a 4-character field.

Step 3 / 13

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
          System.out.printf("%4d", i * j);
4     }
5     System.out.println();
6 }
```

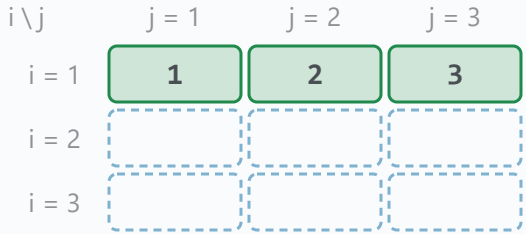
VARIABLES

`i = 1` `j = 3`

OUTPUT

```
   1   2   3
```

Multiplication Table (nested loops, 3 × 3)



cell (i, j) = i × j

%4d: width of 4 characters

j = 4 <= 3 → false; println ends the line.

Step 4 / 13

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
3         System.out.printf("%4d", i * j);
4     }
▶     System.out.println();
6 }
```

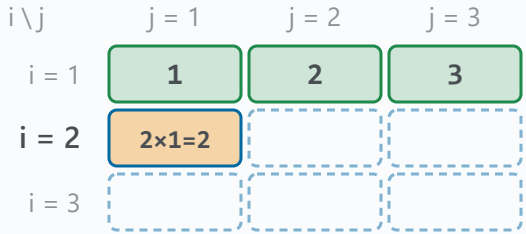
VARIABLES

`i = 1`

OUTPUT

```
   1   2   3
```

Multiplication Table (nested loops, 3 × 3)



cell (i, j) = i × j

%4d: width of 4 characters

Outer loop **i = 2**; inner loop starts at j = 1: 2 × 1 = 2.

Step 5 / 13

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
          System.out.printf("%4d", i * j);
4     }
5     System.out.println();
6 }
```

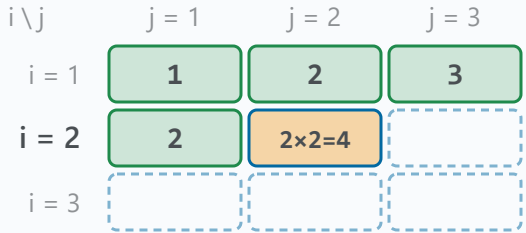
VARIABLES

i = 2 j = 1

OUTPUT

```
   1   2   3
   2
```

Multiplication Table (nested loops, 3 × 3)



cell (i, j) = i × j

%4d: width of 4 characters

j = 2: 2 × 2 = **4**, right-aligned in a 4-character field.

Step 6 / 13

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
          System.out.printf("%4d", i * j);
4     }
5     System.out.println();
6 }
```

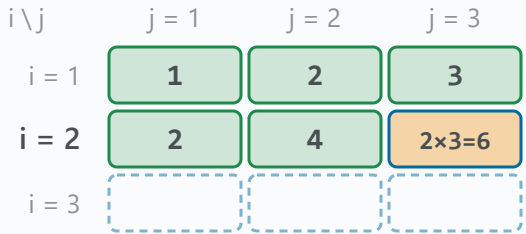
VARIABLES

`i = 2` `j = 2`

OUTPUT

```
   1   2   3
   2   4
```

Multiplication Table (nested loops, 3 × 3)



cell (i, j) = i × j

%4d: width of 4 characters

j = 3: 2 × 3 = **6**, right-aligned in a 4-character field.

Step 7 / 13

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
          System.out.printf("%4d", i * j);
4     }
5     System.out.println();
6 }
```

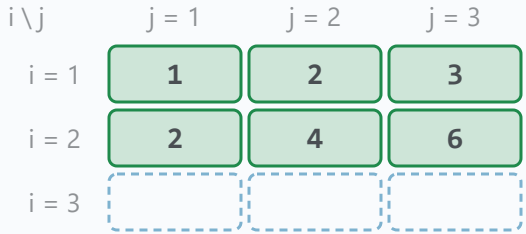
VARIABLES

`i = 2` `j = 3`

OUTPUT

```
   1   2   3
   2   4   6
```

Multiplication Table (nested loops, 3 × 3)



cell (i, j) = i × j

%4d: width of 4 characters

j = 4 <= 3 → false; println ends the line.


```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
3         System.out.printf("%4d", i * j);
4     }
▶     System.out.println();
6 }
```

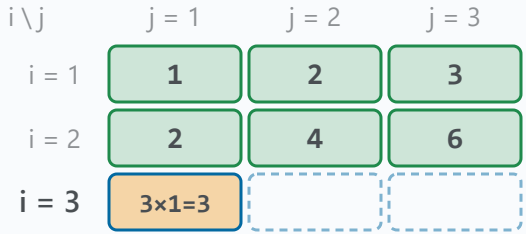
VARIABLES

`i = 2`

OUTPUT

```
   1   2   3
   2   4   6
```

Multiplication Table (nested loops, 3 × 3)



cell (i, j) = i × j

%4d: width of 4 characters

Outer loop **i = 3**; inner loop starts at j = 1: 3 × 1 = 3.

Step 9 / 13

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
          System.out.printf("%4d", i * j);
4     }
5     System.out.println();
6 }
```

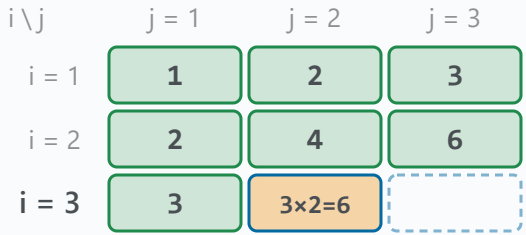
VARIABLES

i = 3 j = 1

OUTPUT

```
   1   2   3
   2   4   6
   3
```

Multiplication Table (nested loops, 3 × 3)



cell (i, j) = i × j

%4d: width of 4 characters

j = 2: 3 × 2 = **6**, right-aligned in a 4-character field.

Step 10 / 13

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
          System.out.printf("%4d", i * j);
4     }
5     System.out.println();
6 }
```

VARIABLES

`i = 3` `j = 2`

OUTPUT

```
   1   2   3
   2   4   6
   3   6
```

Multiplication Table (nested loops, 3 × 3)

i \ j	j = 1	j = 2	j = 3
i = 1	1	2	3
i = 2	2	4	6
i = 3	3	6	3×3=9

cell (i, j) = i × j

%4d: width of 4 characters

j = 3: 3 × 3 = **9**, right-aligned in a 4-character field.

Step 11 / 13

```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
          System.out.printf("%4d", i * j);
4     }
5     System.out.println();
6 }
```

VARIABLES

`i = 3` `j = 3`

OUTPUT

```
   1   2   3
   2   4   6
   3   6   9
```

Multiplication Table (nested loops, 3 × 3)

i \ j	j = 1	j = 2	j = 3
i = 1	1	2	3
i = 2	2	4	6
i = 3	3	6	9

cell (i, j) = i × j

%4d: width of 4 characters

j = 4 <= 3 → false; println ends the line.


```
1 for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
3         System.out.printf("%4d", i * j);
4     }
▶     System.out.println();
6 }
```

VARIABLES

`i = 3`

OUTPUT

```
   1   2   3
   2   4   6
   3   6   9
```

Multiplication Table (nested loops, 3 × 3)

i \ j	j = 1	j = 2	j = 3
i = 1	1	2	3
i = 2	2	4	6
i = 3	3	6	9

cell (i, j) = i × j

%4d: width of 4 characters

i = 4 <= 3 → false. Change the bounds to 10 for the classic 10 × 10 table.

Step 13 / 13

```
▶ for (int i = 1; i <= 3; i++) {
2     for (int j = 1; j <= 3; j++) {
3         System.out.printf("%4d", i * j);
4     }
5     System.out.println();
6 }
```

VARIABLES

—

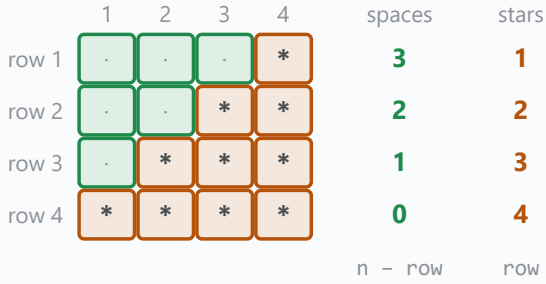
OUTPUT

```
   1   2   3
   2   4   6
   3   6   9
```

EXAMPLE

Solving Pattern Problems

1. Draw the expected output for a small n (e.g. 4) on grid paper.
2. Tabulate how many **spaces** and how many **characters** each row has.
3. Relate the counts to the row number (3, 2, 1, 0 → n − row).
4. Outer loop = row; one inner loop per part.

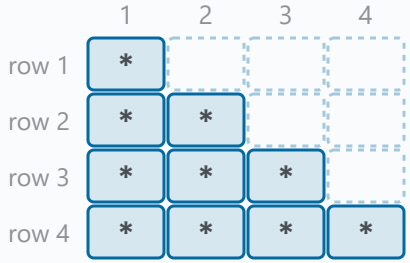


EXAMPLE

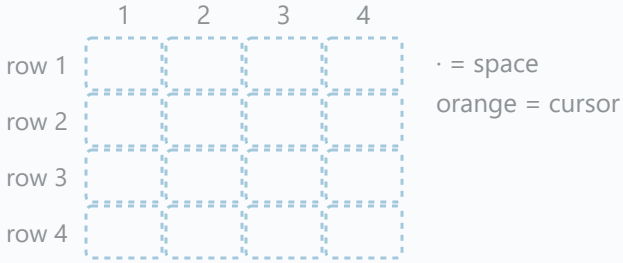
Growing Triangle

```
int n = 4;
for (int row = 1; row <= n; row++) {
    for (int col = 1; col <= row; col++) {
        System.out.print("*");
    }
    System.out.println();
}
```

- Row r gets r stars.
- The inner loop's upper bound is the outer loop variable.



Code Trace: growing triangle (n = 4)



A triangle of n = 4 rows. Each grid cell is one character.

Step 1 / 16

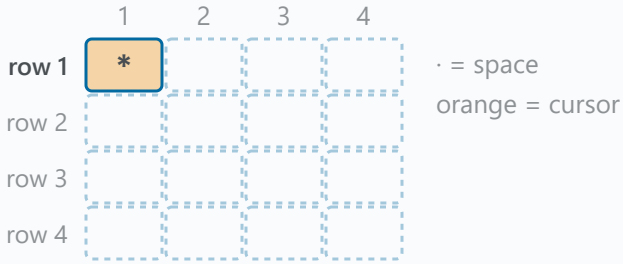
```
▶ int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
4     System.out.print("*");
5   }
6   System.out.println();
7 }
```

VARIABLES

n = 4

OUTPUT

Code Trace: growing triangle (n = 4)



row = 1: inner loop col = 1 … 1 → 1 star(s) on this row.

Step 2 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
▶       System.out.print("*");
5   }
6   System.out.println();
7 }
```

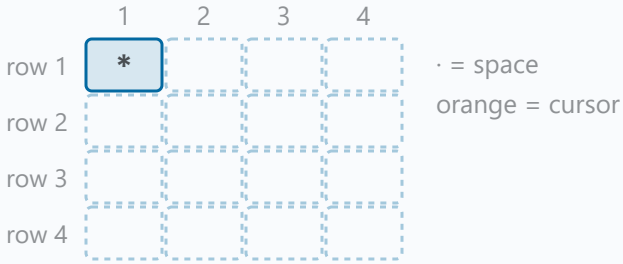
VARIABLES

n = 4 | row = 1 | col = 1

OUTPUT

```
*
```

Code Trace: growing triangle (n = 4)



col = 2 <= 1 → false. println moves to the next line.

Step 3 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
4     System.out.print("*");
5   }
▶   System.out.println();
7 }
```

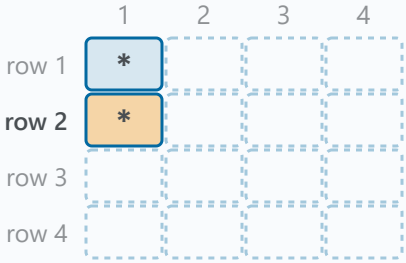
VARIABLES

`n = 4` `row = 1`

OUTPUT

```
*
```

Code Trace: growing triangle (n = 4)



row = 2: inner loop col = 1 … 2 → 2 star(s) on this row.

Step 4 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
▶       System.out.print("*");
5   }
6   System.out.println();
7 }
```

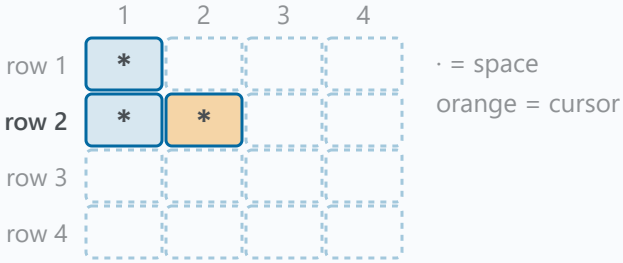
VARIABLES

`n = 4` `row = 2` `col = 1`

OUTPUT

```
*
*
```

Code Trace: growing triangle (n = 4)



col = 2 <= 2 → true → *

Step 5 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
▶     System.out.print("*");
5   }
6   System.out.println();
7 }
```

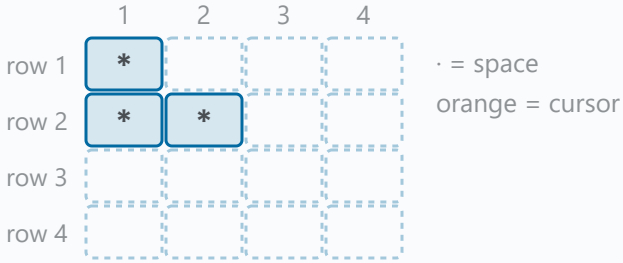
VARIABLES

`n = 4` `row = 2` `col = 2`

OUTPUT

```
*
**
```

Code Trace: growing triangle (n = 4)



col = 3 <= 2 → false. println moves to the next line.

Step 6 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
4     System.out.print("*");
5   }
▶   System.out.println();
7 }
```

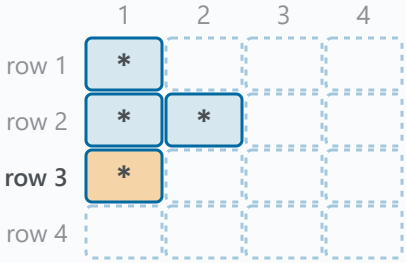
VARIABLES

`n = 4` `row = 2`

OUTPUT

```
*
**
```

Code Trace: growing triangle (n = 4)



row = 3: inner loop col = 1 … 3 → 3 star(s) on this row.

Step 7 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
▶       System.out.print("*");
5   }
6   System.out.println();
7 }
```

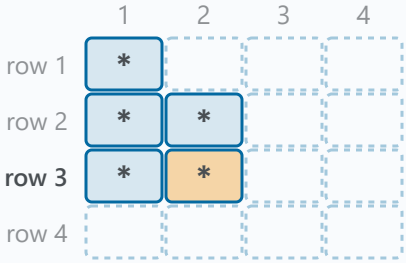
VARIABLES

`n = 4` `row = 3` `col = 1`

OUTPUT

```
*
**
*
```

Code Trace: growing triangle (n = 4)



col = 2 <= 3 → true → *

Step 8 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
▶       System.out.print("*");
5   }
6   System.out.println();
7 }
```

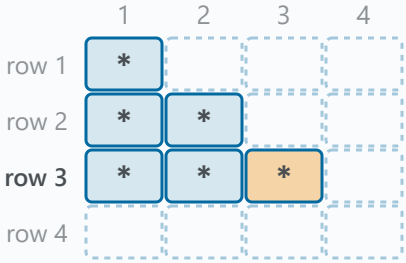
VARIABLES

n = 4 | row = 3 | col = 2

OUTPUT

```
*
**
**
```

Code Trace: growing triangle (n = 4)



col = 3 <= 3 → true → *

Step 9 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
▶     System.out.print("*");
5   }
6   System.out.println();
7 }
```

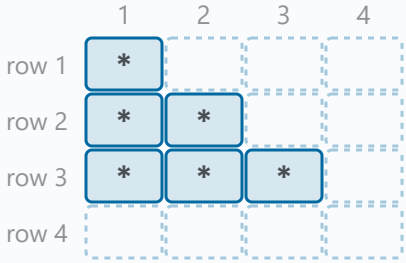
VARIABLES

n = 4 | row = 3 | col = 3

OUTPUT

```
*
**
***
```

Code Trace: growing triangle (n = 4)



col = 4 <= 3 → false. println moves to the next line.

Step 10 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
4     System.out.print("*");
5   }
▶   System.out.println();
7 }
```

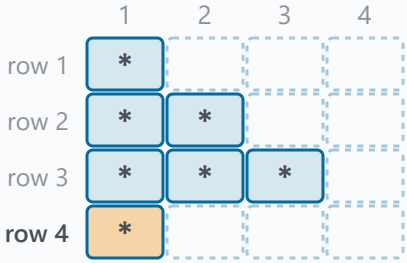
VARIABLES

`n = 4` `row = 3`

OUTPUT

```
*
**
***
```

Code Trace: growing triangle (n = 4)



row = 4: inner loop col = 1 … 4 → 4 star(s) on this row.

Step 11 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
▶       System.out.print("*");
5   }
6   System.out.println();
7 }
```

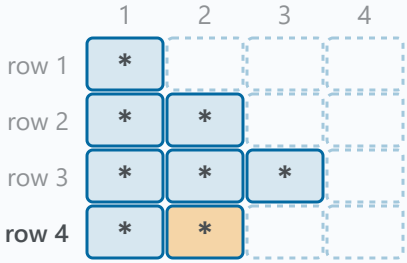
VARIABLES

`n = 4` `row = 4` `col = 1`

OUTPUT

```
*
**
***
*
```

Code Trace: growing triangle (n = 4)



col = 2 <= 4 → true → *

Step 12 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
▶     System.out.print("*");
5   }
6   System.out.println();
7 }
```

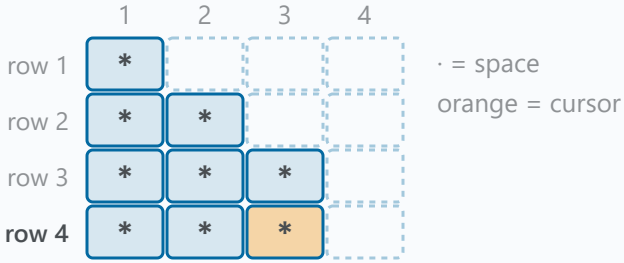
VARIABLES

n = 4 | row = 4 | col = 2

OUTPUT

```
*
**
***
**
```

Code Trace: growing triangle (n = 4)



col = 3 <= 4 → true → *

Step 13 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
▶     System.out.print("*");
5   }
6   System.out.println();
7 }
```

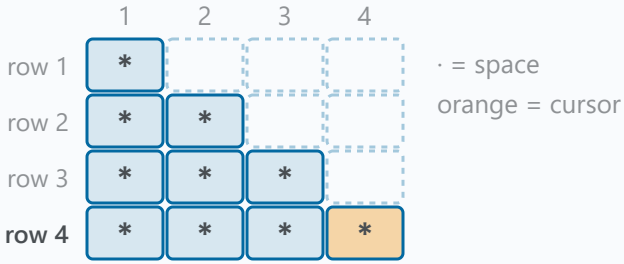
VARIABLES

`n = 4` `row = 4` `col = 3`

OUTPUT

```
*
**
***
***
```

Code Trace: growing triangle (n = 4)



col = 4 <= 4 → true → *

Step 14 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
▶       System.out.print("*");
5   }
6   System.out.println();
7 }
```

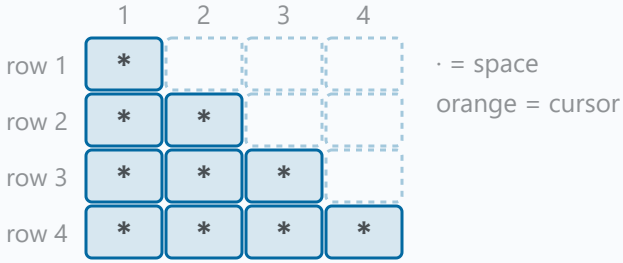
VARIABLES

n = 4 | row = 4 | col = 4

OUTPUT

```
*
**
***
****
```

Code Trace: growing triangle (n = 4)



col = 5 <= 4 → false. println moves to the next line.

Step 15 / 16

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
4     System.out.print("*");
5   }
▶   System.out.println();
7 }
```

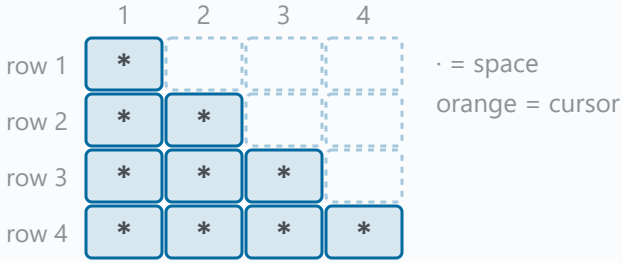
VARIABLES

`n = 4` `row = 4`

OUTPUT

```
*
**
***
****
```

Code Trace: growing triangle (n = 4)



row = 5 <= 4 → false: done. 1 + 2 + 3 + 4 = 10 stars in all.

Step 16 / 16

```
1 int n = 4;
▶ for (int row = 1; row <= n; row++) {
3   for (int col = 1; col <= row; col++) {
4     System.out.print("*");
5   }
6   System.out.println();
7 }
```

VARIABLES

n = 4

OUTPUT

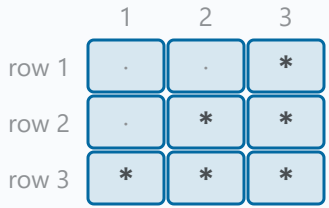
```
*
**
***
****
```

EXAMPLE

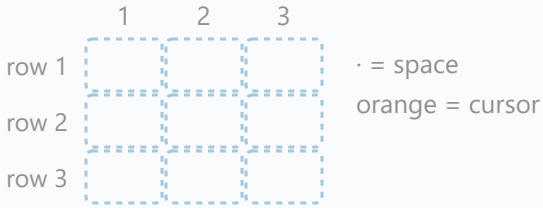
Right-Aligned Triangle

```
int n = 3;
for (int row = 1; row <= n; row++) {
    for (int space = 0; space < n - row; space++)
        System.out.print(" ");
    for (int star = 0; star < row; star++)
        System.out.print("*");
    System.out.println();
}
```

- Each row has a spaces part and a characters part.
- Spaces shrink as stars grow.



Code Trace: right-aligned triangle (n = 3)



n = 3. Each row: first `n - row` spaces, then `row` stars.

Step 1 / 14

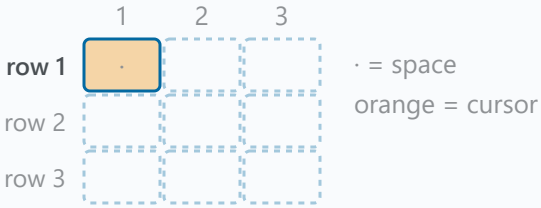
```
▶ int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
4     System.out.print(" ");
5   for (int star = 0; star < row; star++)
6     System.out.print("*");
7   System.out.println();
8 }
```

VARIABLES

n = 3

OUTPUT

Code Trace: right-aligned triangle (n = 3)



row = 1: 2 spaces needed. space = 0 → ' ' is printed.

Step 2 / 14

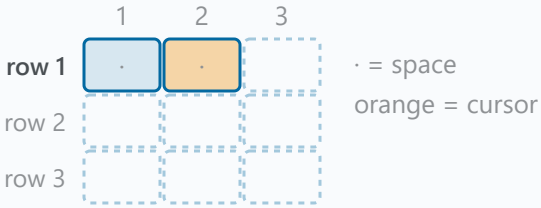
```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
      System.out.print(" ");
5   for (int star = 0; star < row; star++)
6     System.out.print("*");
7   System.out.println();
8 }
```

VARIABLES

`n = 3` `row = 1` `space = 0`

OUTPUT

Code Trace: right-aligned triangle (n = 3)



space = 1 < 2 → ' ' is printed.

Step 3 / 14

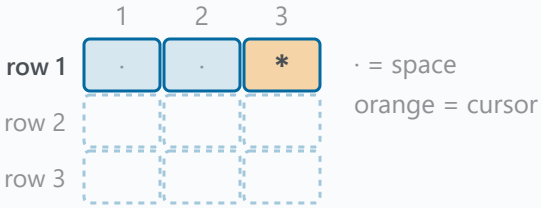
```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
▶       System.out.print(" ");
5   for (int star = 0; star < row; star++)
6     System.out.print("*");
7   System.out.println();
8 }
```

VARIABLES

`n = 3` `row = 1` `space = 1`

OUTPUT

Code Trace: right-aligned triangle (n = 3)



Spaces done; 1 star(s): star = 0 → *

Step 4 / 14

```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
4     System.out.print(" ");
5   for (int star = 0; star < row; star++)
▶     System.out.print("*");
7   System.out.println();
8 }
```

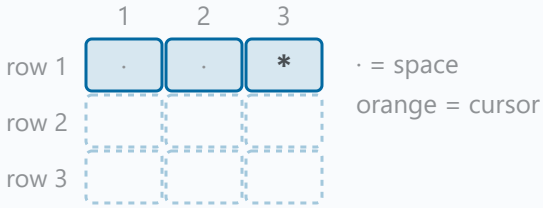
VARIABLES

`n = 3` `row = 1` `star = 0`

OUTPUT

```
  *
```

Code Trace: right-aligned triangle (n = 3)



Row complete (every row has n = 3 characters): println.

Step 5 / 14

```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
4     System.out.print(" ");
5   for (int star = 0; star < row; star++)
6     System.out.print("*");
▶   System.out.println();
8 }
```

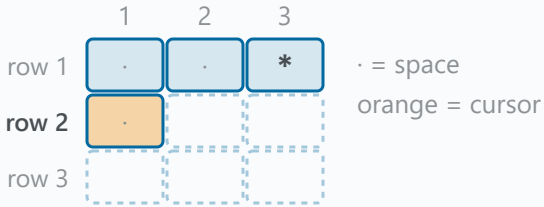
VARIABLES

`n = 3` `row = 1`

OUTPUT

```
  *
```

Code Trace: right-aligned triangle (n = 3)



row = 2: 1 space needed. space = 0 → ' ' is printed.

Step 6 / 14

```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
▶     System.out.print(" ");
5   for (int star = 0; star < row; star++)
6     System.out.print("*");
7   System.out.println();
8 }
```

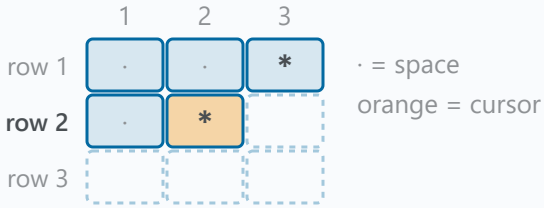
VARIABLES

`n = 3` `row = 2` `space = 0`

OUTPUT

```
  *
```

Code Trace: right-aligned triangle (n = 3)



Spaces done; 2 star(s): star = 0 → *

Step 7 / 14

```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
4     System.out.print(" ");
5   for (int star = 0; star < row; star++)
▶     System.out.print("*");
7   System.out.println();
8 }
```

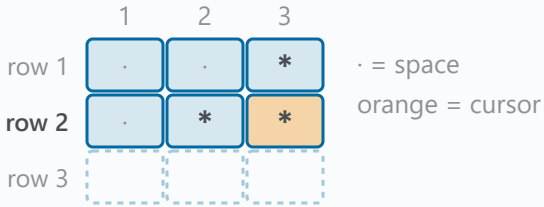
VARIABLES

`n = 3` `row = 2` `star = 0`

OUTPUT

```
  *
 *
```

Code Trace: right-aligned triangle (n = 3)



star = 1 < 2 → *

Step 8 / 14

```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
4     System.out.print(" ");
5   for (int star = 0; star < row; star++)
▶     System.out.print("*");
7   System.out.println();
8 }
```

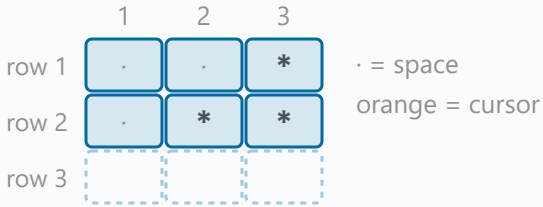
VARIABLES

n = 3 | row = 2 | star = 1

OUTPUT

```
  *
 **
```

Code Trace: right-aligned triangle (n = 3)



Row complete (every row has n = 3 characters): println.

Step 9 / 14

```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
4     System.out.print(" ");
5   for (int star = 0; star < row; star++)
6     System.out.print("*");
▶   System.out.println();
8 }
```

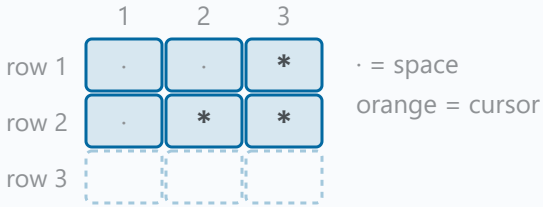
VARIABLES

`n = 3` `row = 2`

OUTPUT

```
  *
 **
```

Code Trace: right-aligned triangle (n = 3)



row = 3: space < 3 − 3 = 0 → false: no spaces on this row.

Step 10 / 14

```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
▶   for (int space = 0; space < n - row; space++)
4     System.out.print(" ");
5   for (int star = 0; star < row; star++)
6     System.out.print("*");
7   System.out.println();
8 }
```

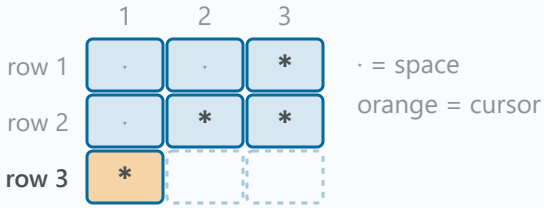
VARIABLES

`n = 3` `row = 3` `space = 0`

OUTPUT

```
  *
 **
```

Code Trace: right-aligned triangle (n = 3)



Spaces done; 3 star(s): star = 0 → *

Step 11 / 14

```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
4     System.out.print(" ");
5   for (int star = 0; star < row; star++)
▶     System.out.print("*");
7   System.out.println();
8 }
```

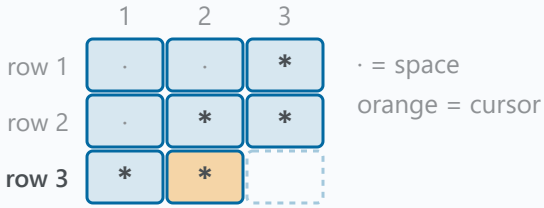
VARIABLES

n = 3 | row = 3 | star = 0

OUTPUT

```
  *
 **
*
```

Code Trace: right-aligned triangle (n = 3)



star = 1 < 3 → *

Step 12 / 14

```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
4     System.out.print(" ");
5   for (int star = 0; star < row; star++)
▶     System.out.print("*");
7   System.out.println();
8 }
```

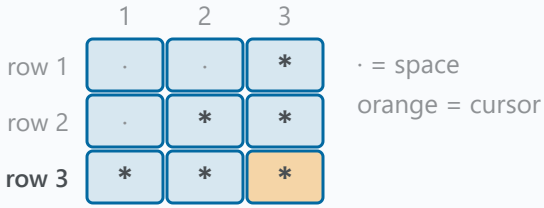
VARIABLES

`n = 3` `row = 3` `star = 1`

OUTPUT

```
  *
 **
**
```

Code Trace: right-aligned triangle (n = 3)



star = 2 < 3 → *

Step 13 / 14

```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
4     System.out.print(" ");
5   for (int star = 0; star < row; star++)
▶     System.out.print("*");
7   System.out.println();
8 }
```

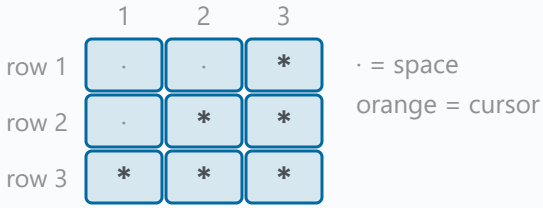
VARIABLES

n = 3 | row = 3 | star = 2

OUTPUT

```
  *
 **
***
```

Code Trace: right-aligned triangle (n = 3)



Row complete (every row has n = 3 characters): println.

Step 14 / 14

```
1 int n = 3;
2 for (int row = 1; row <= n; row++) {
3   for (int space = 0; space < n - row; space++)
4     System.out.print(" ");
5   for (int star = 0; star < row; star++)
6     System.out.print("*");
▶   System.out.println();
8 }
```

VARIABLES

`n = 3` `row = 3`

OUTPUT

```
  *
 **
***
```

EXAMPLE

Counts per Row

Pattern	Spaces	Characters
Growing triangle	0	row
Shrinking triangle	0	n - row + 1
Right-aligned	n - row	row
Pyramid	n - row	2 × row - 1



Growing triangle · Shrinking triangle · Right-aligned · Pyramid

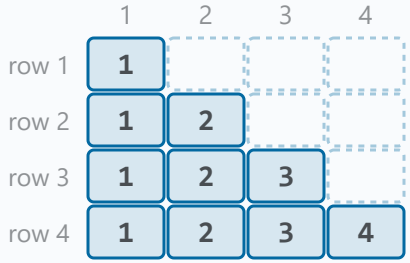
n = 4 · dashed square: space, filled square: character

EXAMPLE

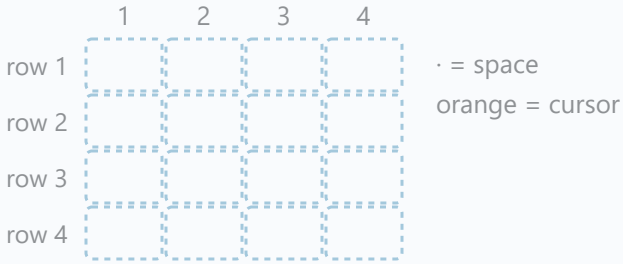
Number Triangle

```
int n = 4;
for (int row = 1; row <= n; row++) {
    for (int value = 1; value <= row; value++) {
        System.out.print(value + " ");
    }
    System.out.println();
}
```

- The printed value comes from the column variable.
- The same structure produces numbers instead of characters.



Code Trace: number triangle (n = 4)



Each cell gets one `value + " "` (number and space).

Step 1 / 15

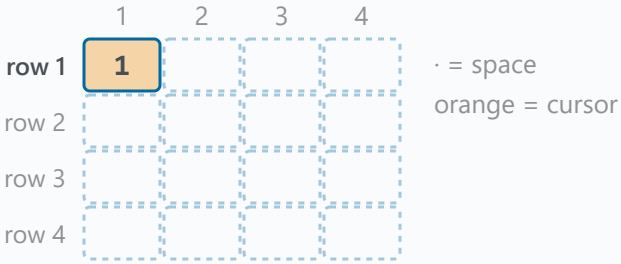
```
▶ int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
4     System.out.print(value + " ");
5   }
6   System.out.println();
7 }
```

VARIABLES

n = 4

OUTPUT

Code Trace: number triangle (n = 4)



row = 1: value runs 1 to 1. value = 1 → “1 ”

Step 2 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
▶     System.out.print(value + " ");
5   }
6   System.out.println();
7 }
```

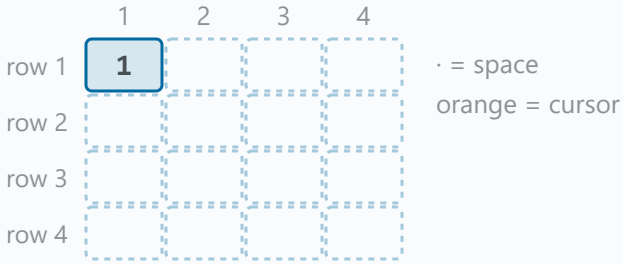
VARIABLES

`n = 4` `row = 1` `value = 1`

OUTPUT

```
1
```

Code Trace: number triangle (n = 4)



value = 2 <= 1 → false; println.

Step 3 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
4     System.out.print(value + " ");
5   }
▶   System.out.println();
7 }
```

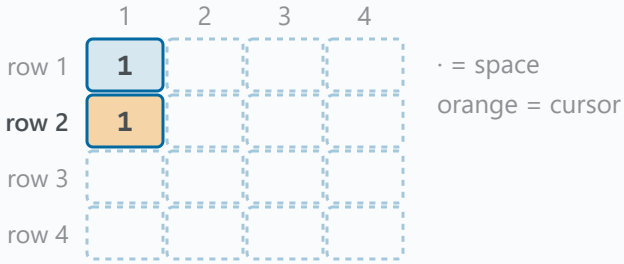
VARIABLES

`n = 4` `row = 1`

OUTPUT

```
1
```

Code Trace: number triangle (n = 4)



row = 2: value runs 1 to 2. value = 1 → “1 ”

Step 4 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
▶     System.out.print(value + " ");
5   }
6   System.out.println();
7 }
```

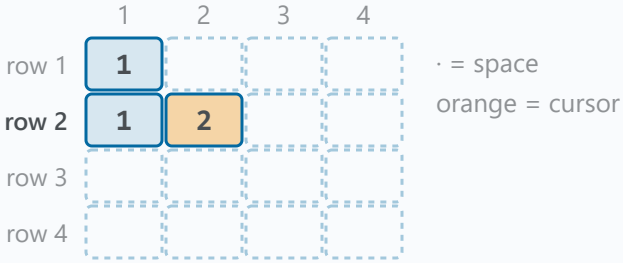
VARIABLES

`n = 4` `row = 2` `value = 1`

OUTPUT

```
1
1
```

Code Trace: number triangle (n = 4)



value = 2 → “2 ” (the printed number is the inner loop variable).

Step 5 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
▶     System.out.print(value + " ");
5   }
6   System.out.println();
7 }
```

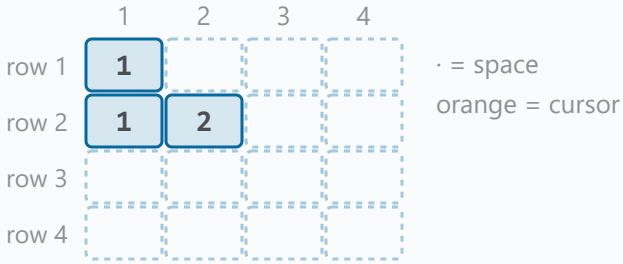
VARIABLES

`n = 4` `row = 2` `value = 2`

OUTPUT

```
1
1 2
```

Code Trace: number triangle (n = 4)



value = 3 <= 2 → false; println.

Step 6 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
4     System.out.print(value + " ");
5   }
    System.out.println();
7 }
```

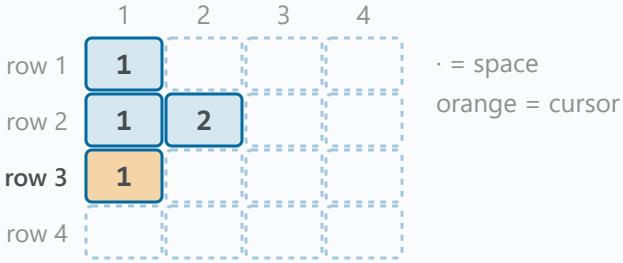
VARIABLES

`n = 4` `row = 2`

OUTPUT

```
1
1 2
```

Code Trace: number triangle (n = 4)



row = 3: value runs 1 to 3. value = 1 → “1 ”

Step 7 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
▶       System.out.print(value + " ");
5   }
6   System.out.println();
7 }
```

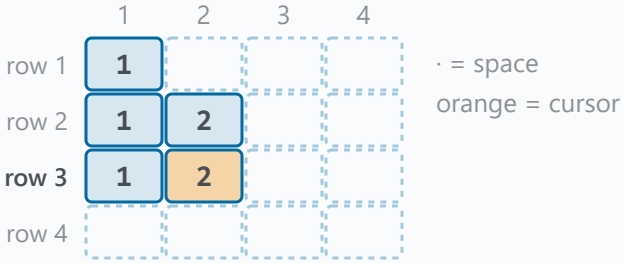
VARIABLES

`n = 4` `row = 3` `value = 1`

OUTPUT

```
1
1 2
1
```

Code Trace: number triangle (n = 4)



value = 2 → “2 ” (the printed number is the inner loop variable).

Step 8 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
▶       System.out.print(value + " ");
5   }
6   System.out.println();
7 }
```

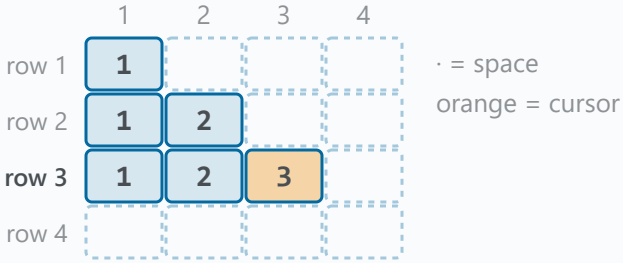
VARIABLES

`n = 4` `row = 3` `value = 2`

OUTPUT

```
1
1 2
1 2
```

Code Trace: number triangle (n = 4)



value = 3 → “3 ” (the printed number is the inner loop variable).

Step 9 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
▶     System.out.print(value + " ");
5   }
6   System.out.println();
7 }
```

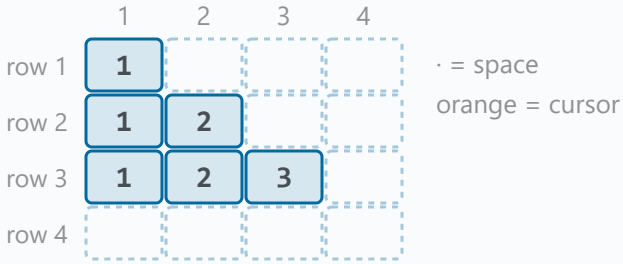
VARIABLES

`n = 4` `row = 3` `value = 3`

OUTPUT

```
1
1 2
1 2 3
```

Code Trace: number triangle (n = 4)



value = 4 <= 3 → false; println.

Step 10 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
4     System.out.print(value + " ");
5   }
▶   System.out.println();
7 }
```

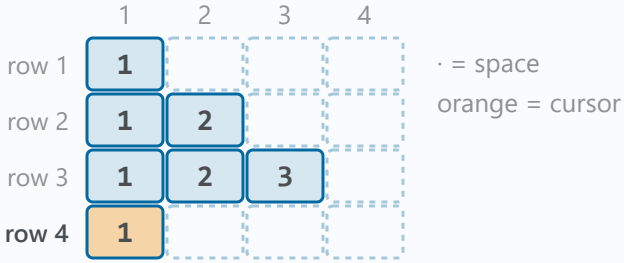
VARIABLES

`n = 4` `row = 3`

OUTPUT

```
1
1 2
1 2 3
```

Code Trace: number triangle (n = 4)



row = 4: value runs 1 to 4. value = 1 → “1 ”

Step 11 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
▶       System.out.print(value + " ");
5   }
6   System.out.println();
7 }
```

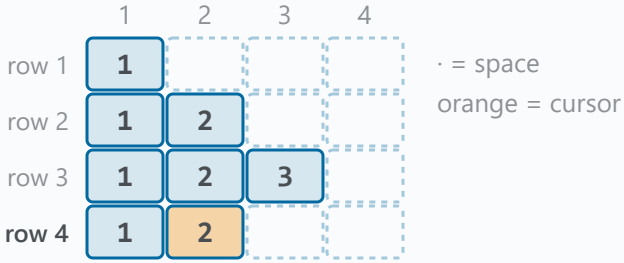
VARIABLES

`n = 4` `row = 4` `value = 1`

OUTPUT

```
1
1 2
1 2 3
1
```

Code Trace: number triangle (n = 4)



value = 2 → “2 ” (the printed number is the inner loop variable).

Step 12 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
▶     System.out.print(value + " ");
5   }
6   System.out.println();
7 }
```

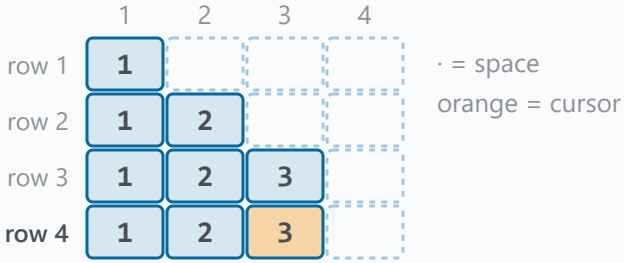
VARIABLES

`n = 4` `row = 4` `value = 2`

OUTPUT

```
1
1 2
1 2 3
1 2
```

Code Trace: number triangle (n = 4)



value = 3 → “3 ” (the printed number is the inner loop variable).

Step 13 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
▶     System.out.print(value + " ");
5   }
6   System.out.println();
7 }
```

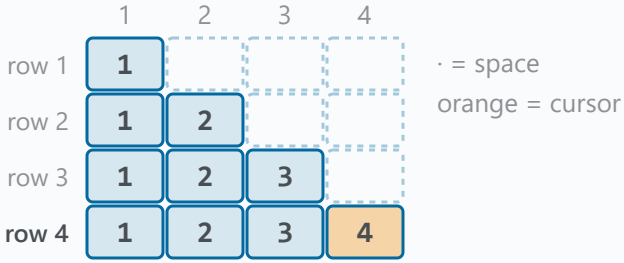
VARIABLES

`n = 4` `row = 4` `value = 3`

OUTPUT

```
1
1 2
1 2 3
1 2 3
```

Code Trace: number triangle (n = 4)



value = 4 → “4 ” (the printed number is the inner loop variable).

Step 14 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
▶     System.out.print(value + " ");
5   }
6   System.out.println();
7 }
```

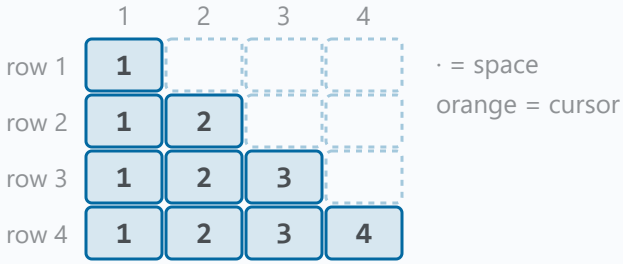
VARIABLES

n = 4 | row = 4 | value = 4

OUTPUT

```
1
1 2
1 2 3
1 2 3 4
```

Code Trace: number triangle (n = 4)



value = 5 <= 4 → false; println.

Step 15 / 15

```
1 int n = 4;
2 for (int row = 1; row <= n; row++) {
3   for (int value = 1; value <= row; value++) {
4     System.out.print(value + " ");
5   }
▶   System.out.println();
7 }
```

VARIABLES

`n = 4` `row = 4`

OUTPUT

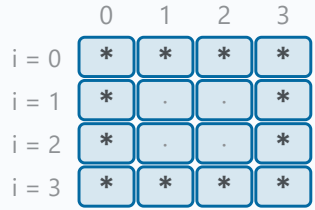
```
1
1 2
1 2 3
1 2 3 4
```

EXAMPLE

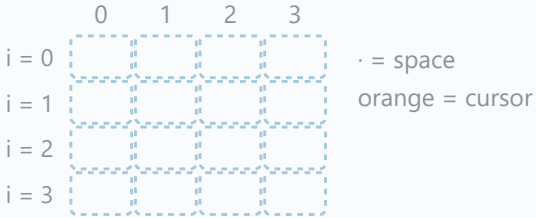
Hollow Square

```
int n = 4;
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
      System.out.print("*");
    else
      System.out.print(" ");
  }
  System.out.println();
}
```

- Only the first/last row or first/last column gets a star.
- `||` : one of the four conditions being true is enough.
- Other cells get a space; otherwise the square shifts.



Code Trace: hollow square (n = 4)



n = 4. Edge: first/last row (i = 0, 3) or first/last column (j = 0, 3).

Step 1 / 11

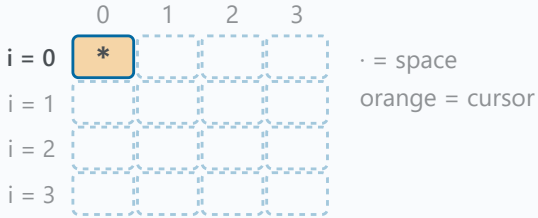
```
▶ int n = 4;
2 for (int i = 0; i < n; i++) {
3   for (int j = 0; j < n; j++) {
4     if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
5       System.out.print("*");
6     else
7       System.out.print(" ");
8   }
9   System.out.println();
10 }
```

VARIABLES

n = 4

OUTPUT

Code Trace: hollow square (n = 4)



i = 0, j = 0: **i == 0** → true → *

Step 2 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

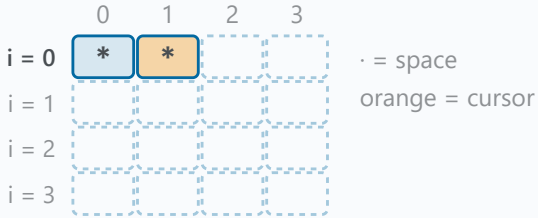
VARIABLES

`n = 4` `i = 0` `j = 0`

OUTPUT

```
*
```

Code Trace: hollow square (n = 4)



i = 0, j = 1: **i == 0** → true → *

Step 3 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

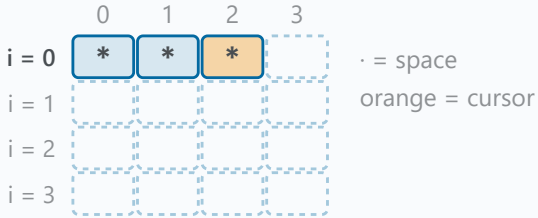
VARIABLES

n = 4 | i = 0 | j = 1

OUTPUT

```
**
```

Code Trace: hollow square (n = 4)



i = 0, j = 2: **i == 0** → true → *

Step 4 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

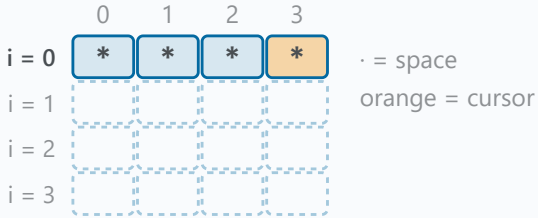
VARIABLES

n = 4 | i = 0 | j = 2

OUTPUT

```
***
```

Code Trace: hollow square (n = 4)



i = 0, j = 3: **i == 0** → true → *

Step 5 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

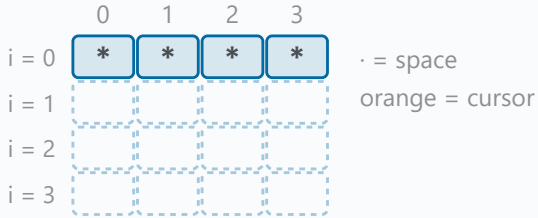
VARIABLES

n = 4 | i = 0 | j = 3

OUTPUT

```
****
```

Code Trace: hollow square (n = 4)



j = 4 < 4 → false; println.

Step 6 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
4      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
5        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
▶    System.out.println();
10 }
```

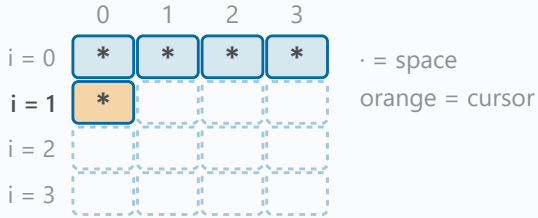
VARIABLES

`n = 4` `i = 0`

OUTPUT

```
****
```

Code Trace: hollow square (n = 4)



i = 1, j = 0: **j == 0** → true → *

Step 7 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

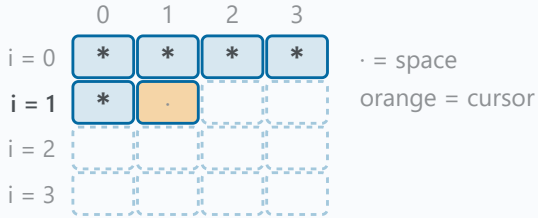
VARIABLES

`n = 4` `i = 1` `j = 0`

OUTPUT

```
****
*
```

Code Trace: hollow square (n = 4)



i = 1, j = 1: all four conditions false → else → space

Step 8 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
5        System.out.print("*");
6      else
▶        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

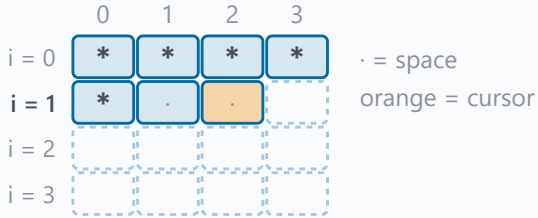
VARIABLES

`n = 4` `i = 1` `j = 1`

OUTPUT

```
****
*
```

Code Trace: hollow square (n = 4)



i = 1, j = 2: all four conditions false → else → space

Step 9 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
5        System.out.print("*");
6      else
▶        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

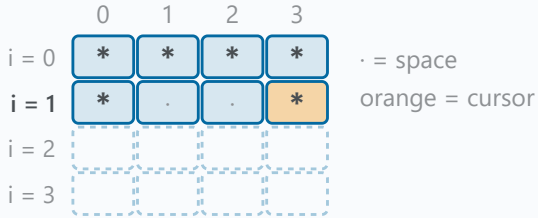
VARIABLES

`n = 4` `i = 1` `j = 2`

OUTPUT

```
****
*
```

Code Trace: hollow square (n = 4)



i = 1, j = 3: **j == 3** → true → *

Step 10 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

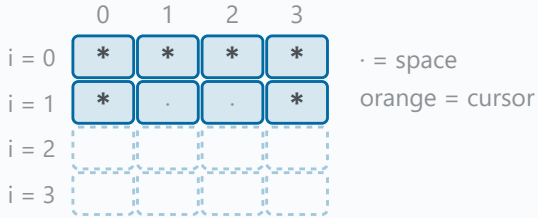
VARIABLES

n = 4 | i = 1 | j = 3

OUTPUT

```
****
*  *
```

Code Trace: hollow square (n = 4)



j = 4 < 4 → false; println.

Step 11 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
4      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
5        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
▶    System.out.println();
10 }
```

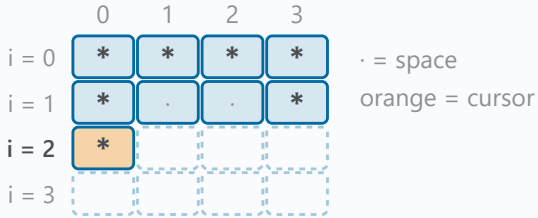
VARIABLES

`n = 4` `i = 1`

OUTPUT

```
****
*  *
```

Code Trace (cont.): rows 2 and 3



i = 2, j = 0: **j == 0** → true → *

Step 1 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

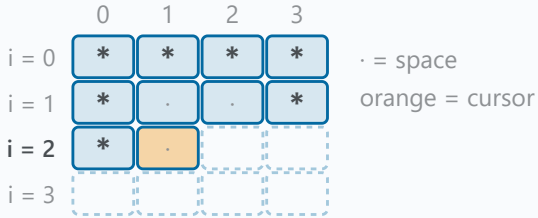
VARIABLES

n = 4 | i = 2 | j = 0

OUTPUT

```
****
*  *
*
```

Code Trace (cont.): rows 2 and 3



i = 2, j = 1: all four conditions false → else → space

Step 2 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
5        System.out.print("*");
6      else
▶        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

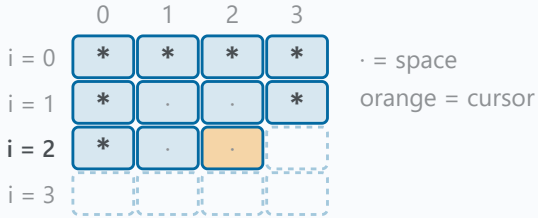
VARIABLES

n = 4 | i = 2 | j = 1

OUTPUT

```
****
*  *
*
```

Code Trace (cont.): rows 2 and 3



i = 2, j = 2: all four conditions false → else → space

Step 3 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
5        System.out.print("*");
6      else
▶        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

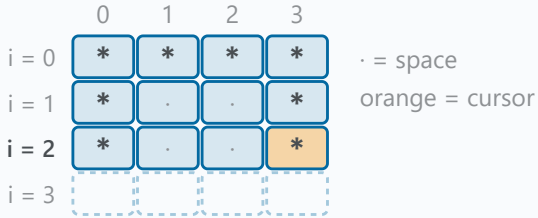
VARIABLES

`n = 4` `i = 2` `j = 2`

OUTPUT

```
****
*  *
*
```

Code Trace (cont.): rows 2 and 3



i = 2, j = 3: **j == 3** → true → *

Step 4 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

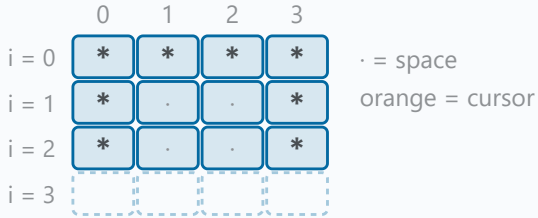
VARIABLES

n = 4 | i = 2 | j = 3

OUTPUT

```
****
*  *
*  *
```

Code Trace (cont.): rows 2 and 3



j = 4 < 4 → false; println.

Step 5 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
4      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
5        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
▶    System.out.println();
10 }
```

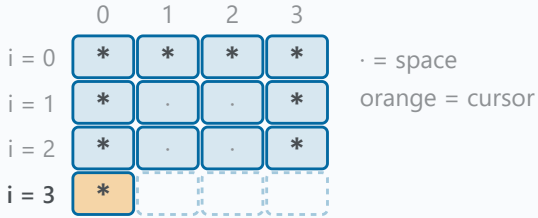
VARIABLES

`n = 4` `i = 2`

OUTPUT

```
****
*  *
*  *
```

Code Trace (cont.): rows 2 and 3



i = 3, j = 0: **i == 3** → true → *

Step 6 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

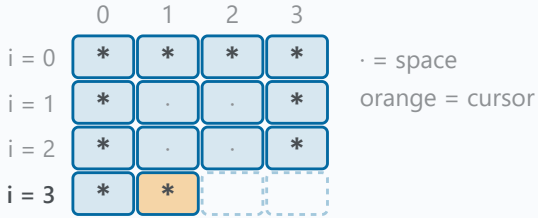
VARIABLES

n = 4 | i = 3 | j = 0

OUTPUT

```
****
*  *
*  *
*
```

Code Trace (cont.): rows 2 and 3



i = 3, j = 1: **i == 3** → true → *

Step 7 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

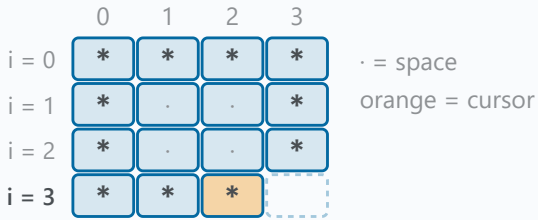
VARIABLES

n = **4** i = **3** j = **1**

OUTPUT

```
****
*  *
*  *
**
```

Code Trace (cont.): rows 2 and 3



i = 3, j = 2: **i == 3** → true → *

Step 8 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

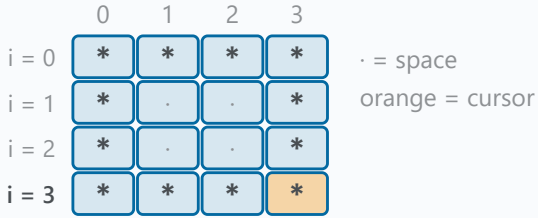
VARIABLES

n = 4 | i = 3 | j = 2

OUTPUT

```
****
*  *
*  *
***
```

Code Trace (cont.): rows 2 and 3



i = 3, j = 3: **i == 3** → true → *

Step 9 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
▶      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
▶        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

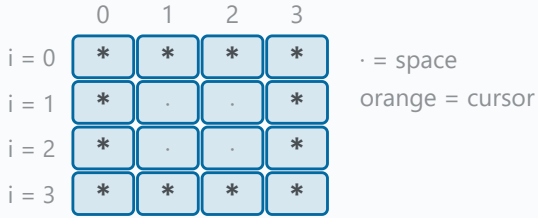
VARIABLES

n = 4 | i = 3 | j = 3

OUTPUT

```
****
*  *
*  *
****
```

Code Trace (cont.): rows 2 and 3



j = 4 < 4 → false; println.

Step 10 / 11

```
1  int n = 4;
2  for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
4      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
5        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
▶    System.out.println();
10 }
```

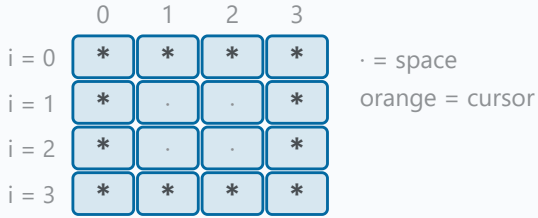
VARIABLES

n = 4 i = 3

OUTPUT

```
****
*  *
*  *
****
```

Code Trace (cont.): rows 2 and 3



i = 4 < 4 → false: the square is done. 12 of the 16 cells are edges.

Step 11 / 11

```
1  int n = 4;
▶ for (int i = 0; i < n; i++) {
3    for (int j = 0; j < n; j++) {
4      if (i == 0 || i == n - 1 || j == 0 || j == n - 1)
5        System.out.print("*");
6      else
7        System.out.print(" ");
8    }
9    System.out.println();
10 }
```

VARIABLES

n = 4

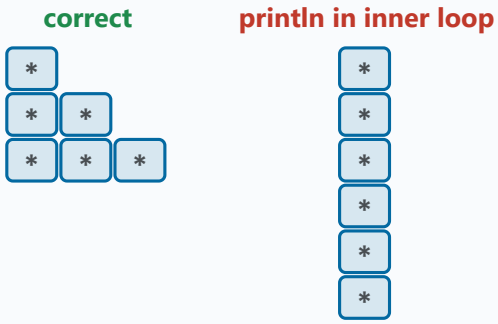
OUTPUT

```
****
*  *
*  *
****
```

EXAMPLE

Common Mistakes

- Putting println in the inner loop prints each character on its own line.
- < vs. <= yields one character too few or too many.
- Printing spaces after the stars flips the alignment.
- Changing the outer variable in the inner loop breaks the flow.



EXAMPLE

Designing Tests

- Pick a normal value for each problem.
- Test 0, 1, negatives and values near the upper limit separately.
- Work out the expected result without running the code.
- Trace the variables pass by pass to find the first deviation.

digit count test	number	expected
Normal	4096	4
Zero	0	1
Negative	−507	3
Extreme	−2147483648	10

SUMMARY

Chapter Summary

- Every loop has initialization, condition, body and update parts.
- Use for when the pass count is known, while while a condition holds, do-while when at least one pass is needed.
- Counter and accumulator patterns recur in digit, power and series problems.
- In nested loops the outer loop produces rows, the inner loop the items in each row.
- Digit pattern (`% 10` , `/ 10`): digit count, reversing, palindromes, Armstrong; divisor pattern: primes and perfect numbers.
- Test loops by tracing them by hand with 0, 1 and boundary values.

SUMMARY

Review Questions

1. What is the key difference between while and do-while?
2. What do break and continue do?
3. What are % 10 and / 10 for in the sum of digits?
4. Why is the inner loop's upper bound the row variable in the growing triangle?
5. How does the output change if println is inside the inner loop?

SUMMARY

Exercises

1. Read numbers until 0 is entered and print their sum and average (while).
2. Find the sum of the numbers from 1 to 100 divisible by 3 or 5 (answer: 2418).
3. Draw an n-row **pyramid**: row r has n − r spaces and 2r − 1 stars (trace it by hand for n = 4).
4. Compute the **factorial** of a number with for; why doesn't 13! fit in an int? Try long.
5. Use do-while to keep asking until the user enters a score from 1 to 5.
6. List all Armstrong (3-digit) and perfect numbers in 1..1000.

References

- Deitel and Deitel, Java: How to Program, Early Objects, 11th ed., Pearson, 2018.
- Liang, Introduction to Java Programming and Data Structures, 13th ed., Pearson, 2024.
- Oracle, The Java Tutorials: **while and do-while**, **for**, **Branching Statements**.
- Sedgewick and Wayne, Computer Science: An Interdisciplinary Approach, Addison-Wesley, 2016: **1.3 Conditionals and Loops**.
- Downey and Mayfield, Think Java, 2nd ed., O'Reilly, 2019 (**free online edition**).

The End

Chapter 8: Loops

Previous: Chapter 7: Conditionals · Next: Chapter 9: Methods