Chapter 7: Conditionals

Computer Fundamentals and Introduction to Programming

Control flow | if-else | else if | switch | Examples

Sercan KÜLCÜ · Giresun University

© Sercan Külcü. All rights reserved. · Interactive HTML version of these slides: https://sercankulcu.github.io/files/computer/slides/Chapter_07_Conditionals.html

INTRODUCTION

Learning Outcomes

Flow

Distinguish sequential, selection and repetition flow.

Conditions

Write conditions with comparison and logical operators.

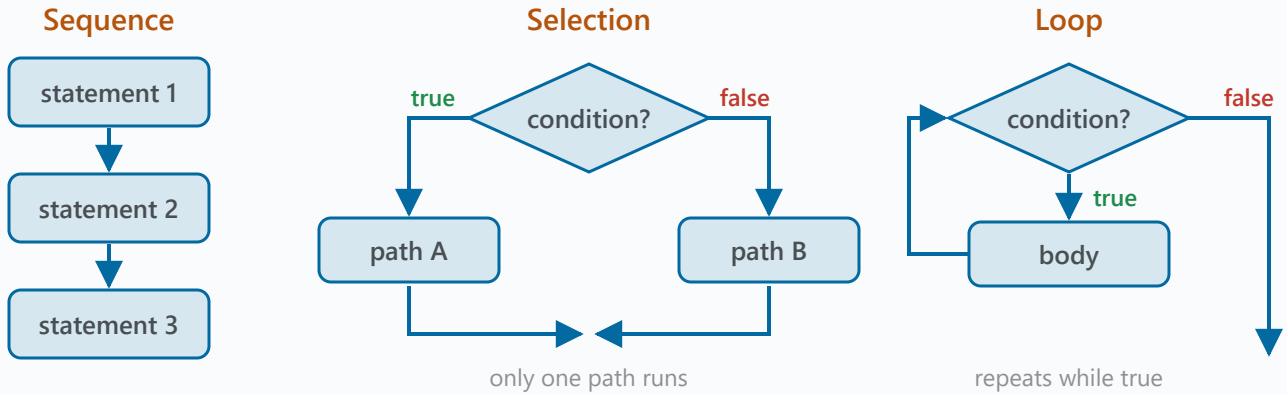
Decisions

Write if, else if and else in the right order.

switch

Use switch for discrete choices.

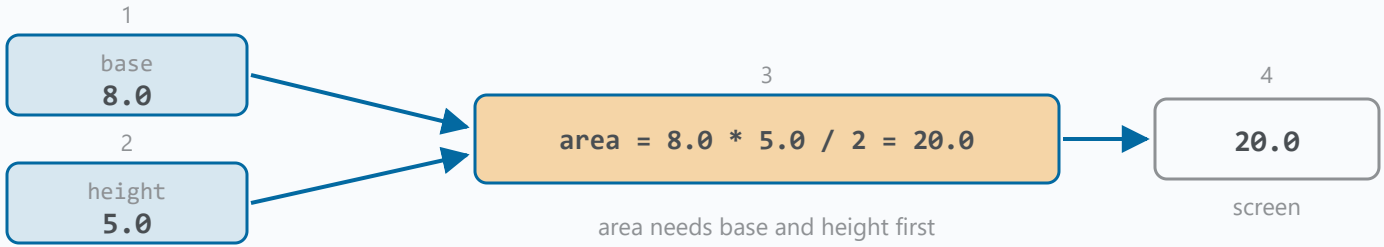
Three Basic Control Structures



- Sequence runs statements in the order written; selection picks a path based on a condition.
- A loop repeats a block of code.
- Every algorithm is built from these three structures.

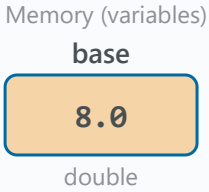
Sequential Statements

```
double base = 8;
double height = 5;
double area = base * height / 2;
System.out.println(area);
```



- Each statement can use values produced by earlier ones.
- Change the order and you may get an undefined or wrong value.

Sequential Statements Step by Step



base = 8 → stored in a double box as 8.0 (widening).

Step 1 / 5

```
▶ double base = 8;
2 double height = 5;
3 double area = base * height / 2;
4 System.out.println(area);
```

VARIABLES

`base = 8.0`

OUTPUT

Sequential Statements Step by Step



height = 5.0.

Step 2 / 5

```
1 double base = 8;
▶ double height = 5;
3 double area = base * height / 2;
4 System.out.println(area);
```

VARIABLES

`base = 8.0` `height = 5.0`

OUTPUT

Sequential Statements Step by Step

Memory (variables)

base	height
8.0	5.0
double	double

Expression

```
base * height / 2
40.0 / 2
```

Left to right: first base * height = 40.0.

Step 3 / 5

```
1 double base = 8;
2 double height = 5;
▶ double area = base * height / 2;
4 System.out.println(area);
```

VARIABLES

`base = 8.0` `height = 5.0`

OUTPUT

Sequential Statements Step by Step



40.0 / 2 = 20.0 → stored in area. This line **uses** the values of the two boxes above.

Step 4 / 5

```
1 double base = 8;
2 double height = 5;
▶ double area = base * height / 2;
4 System.out.println(area);
```

VARIABLES

`base = 8.0` `height = 5.0` `area = 20.0`

OUTPUT

Sequential Statements Step by Step



20.0 is printed. Had line 3 come before line 1, base would not exist yet → compile error.

Step 5 / 5

```
1 double base = 8;
2 double height = 5;
3 double area = base * height / 2;
▶ System.out.println(area);
```

VARIABLES

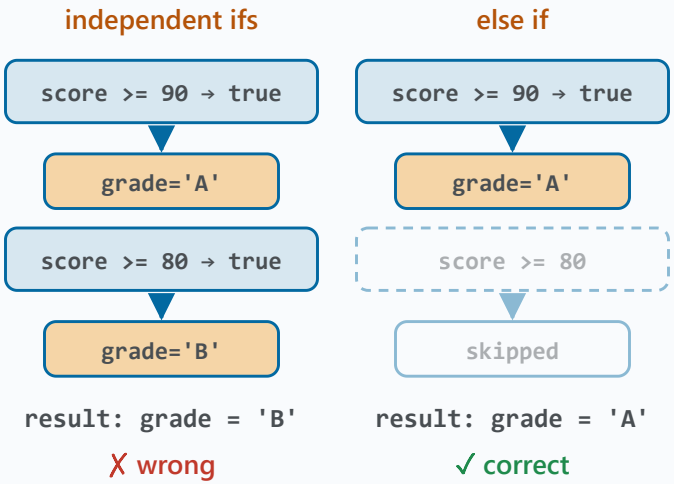
`base = 8.0` `height = 5.0` `area = 20.0`

OUTPUT

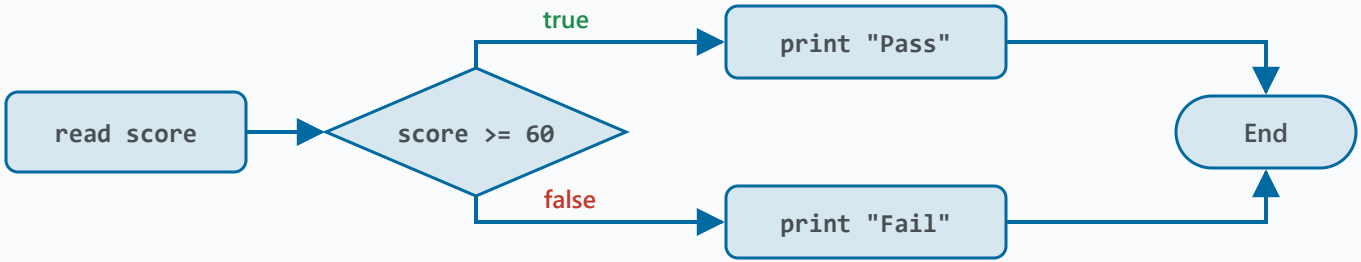
```
20.0
```

Making Decisions

- A condition produces a boolean value.
- if defines the true path, else the false path.
- **Every** independent if is tested; in an else-if chain, once the first true branch runs the rest are **skipped**.
- Example: score = 95. On the right, two versions with different results.



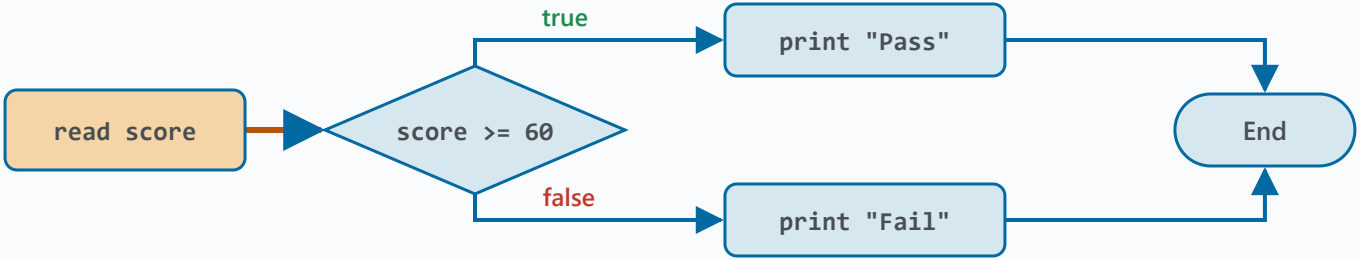
The if-else Statement




```java
if (score >= 60) {
    System.out.println("Pass");
} else {
    System.out.println("Fail");
}
```

- The boundary belongs to the condition: 60 → Pass.
- Every possible value goes to exactly **one** branch.

if-else Step by Step: score = 75, 60, 45



Run 1: the user enters **75**.

Step 1 / 9

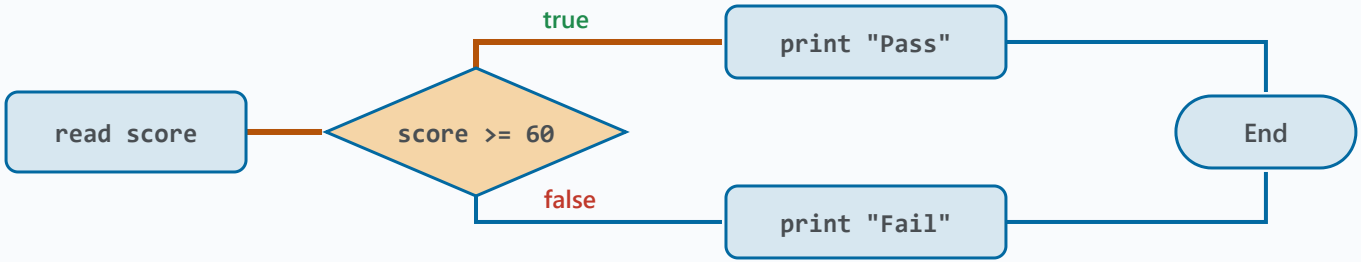
```
1 Scanner in = new Scanner(System.in);
▶ int score = in.nextInt();
3 if (score >= 60) {
4     System.out.println("Pass");
5 } else {
6     System.out.println("Fail");
7 }
```

VARIABLES

`score = 75`

OUTPUT

if-else Step by Step: score = 75, 60, 45



Condition: score >= 60 → 75 >= 60 → **true**.

Step 2 / 9

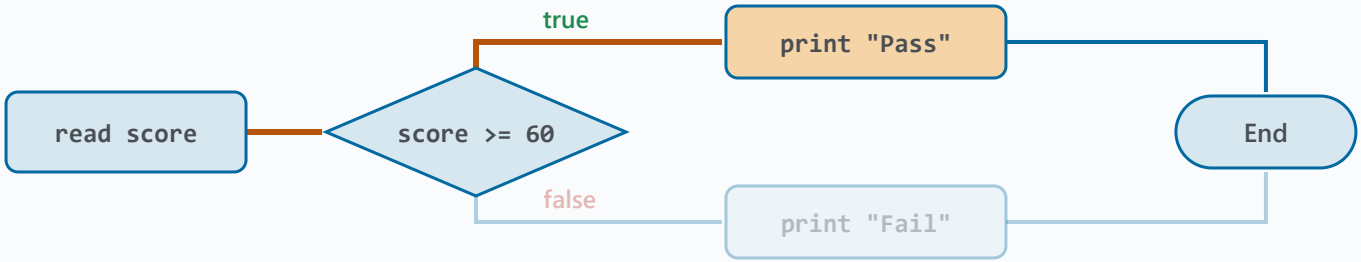
```
1 Scanner in = new Scanner(System.in);
2 int score = in.nextInt();
▶ if (score >= 60) {
4     System.out.println("Pass");
5 } else {
6     System.out.println("Fail");
7 }
```

VARIABLES

`score = 75`

OUTPUT

if-else Step by Step: score = 75, 60, 45



The true branch runs: **Pass**. The else block is skipped.

Step 3 / 9

```
1 Scanner in = new Scanner(System.in);
2 int score = in.nextInt();
3 if (score >= 60) {
▶     System.out.println("Pass");
5 } else {
6     System.out.println("Fail");
7 }
```

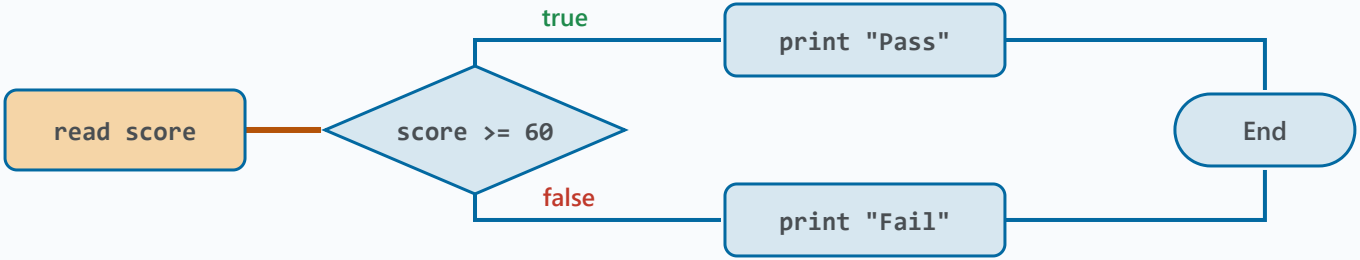
VARIABLES

`score = 75`

OUTPUT

```
Pass
```

if-else Step by Step: score = 75, 60, 45



Run 2: the user enters **60**.

Step 4 / 9

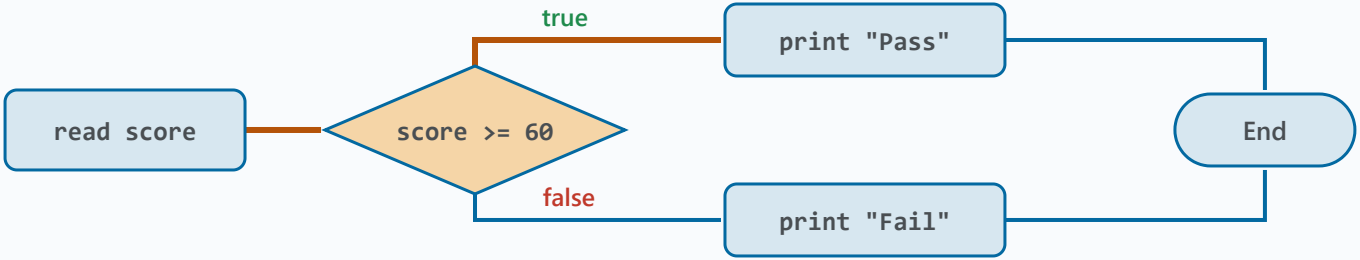
```
1 Scanner in = new Scanner(System.in);
▶ int score = in.nextInt();
3 if (score >= 60) {
4     System.out.println("Pass");
5 } else {
6     System.out.println("Fail");
7 }
```

VARIABLES

`score = 60`

OUTPUT

if-else Step by Step: score = 75, 60, 45



Condition: score >= 60 → 60 >= 60 → **true**. The boundary is included: `>=`.

Step 5 / 9

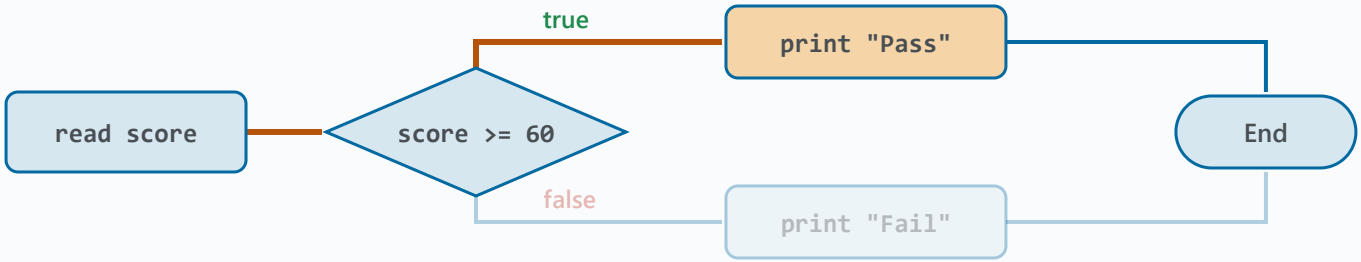
```
1 Scanner in = new Scanner(System.in);
2 int score = in.nextInt();
▶ if (score >= 60) {
4     System.out.println("Pass");
5 } else {
6     System.out.println("Fail");
7 }
```

VARIABLES

`score = 60`

OUTPUT

if-else Step by Step: score = 75, 60, 45



The true branch runs: **Pass**. The else block is skipped.

Step 6 / 9

```
1 Scanner in = new Scanner(System.in);
2 int score = in.nextInt();
3 if (score >= 60) {
▶     System.out.println("Pass");
5 } else {
6     System.out.println("Fail");
7 }
```

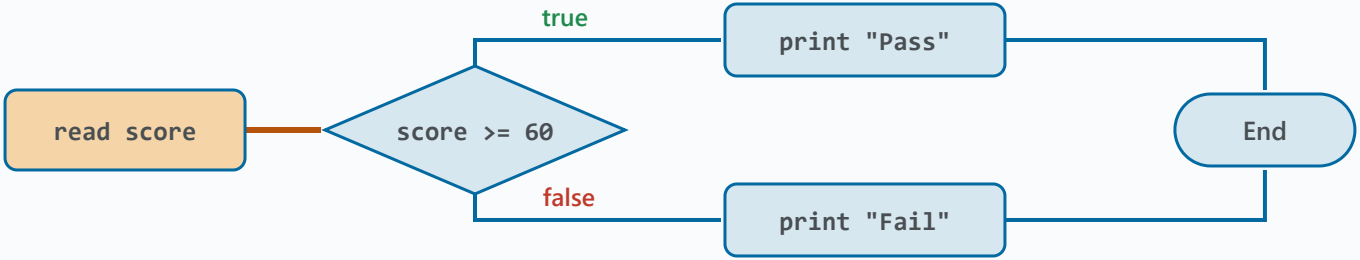
VARIABLES

`score = 60`

OUTPUT

```
Pass
```

if-else Step by Step: score = 75, 60, 45



Run 3: the user enters **45**.

Step 7 / 9

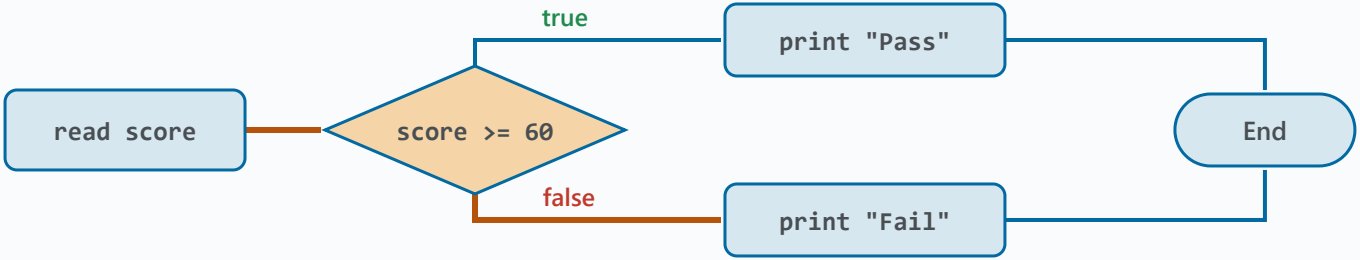
```
1 Scanner in = new Scanner(System.in);
▶ int score = in.nextInt();
3 if (score >= 60) {
4     System.out.println("Pass");
5 } else {
6     System.out.println("Fail");
7 }
```

VARIABLES

`score = 45`

OUTPUT

if-else Step by Step: score = 75, 60, 45



Condition: score >= 60 → 45 >= 60 → **false**.

Step 8 / 9

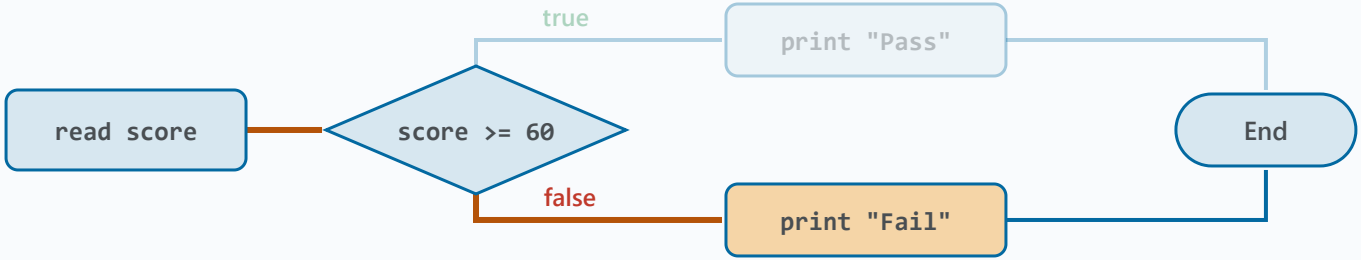
```
1 Scanner in = new Scanner(System.in);
2 int score = in.nextInt();
▶ if (score >= 60) {
4     System.out.println("Pass");
5 } else {
6     System.out.println("Fail");
7 }
```

VARIABLES

`score = 45`

OUTPUT

if-else Step by Step: score = 75, 60, 45



The false branch runs: **Fail**. The if block is skipped.

Step 9 / 9

```
1 Scanner in = new Scanner(System.in);
2 int score = in.nextInt();
3 if (score >= 60) {
4     System.out.println("Pass");
5 } else {
▶     System.out.println("Fail");
7 }
```

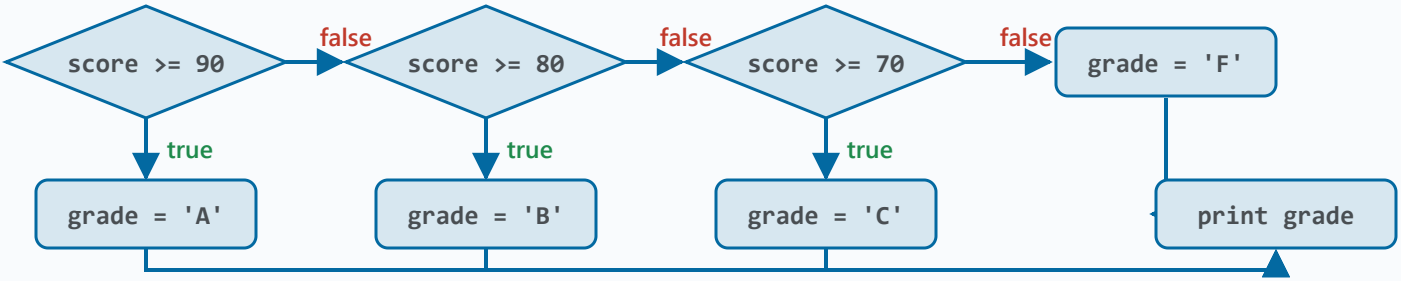
VARIABLES

`score = 45`

OUTPUT

```
Fail
```

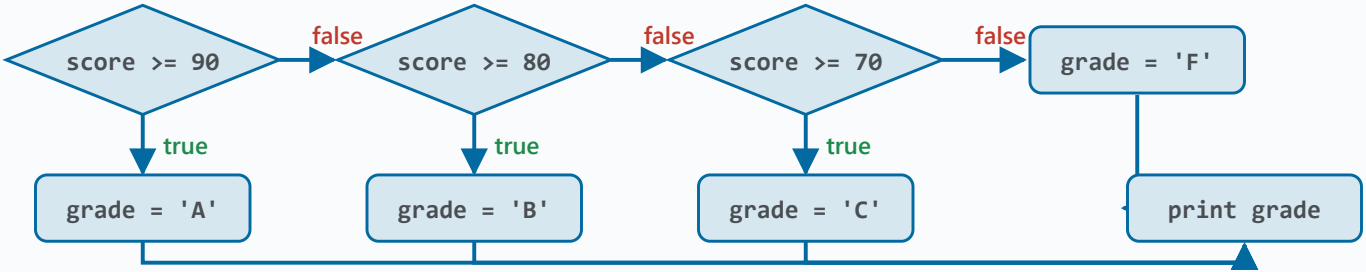
Multiway Decisions: else-if




```
if (score >= 90) {
    grade = 'A';
} else if (score >= 80) {
    grade = 'B';
} else if (score >= 70) {
    grade = 'C';
} else {
    grade = 'F';
}
```

- Conditions are tested top to bottom; the **first true** branch runs and the rest are skipped.
- Put the higher thresholds first: with `score >= 70` on top, 95 would also get a C.
- else runs when no condition holds.

else-if Chain: score = 75, 95, 50



Run 1: score = **75**. grade is declared but has no value yet.

Step 1 / 16

```
▶ int score = in.nextInt();
▶ char grade;
3 if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

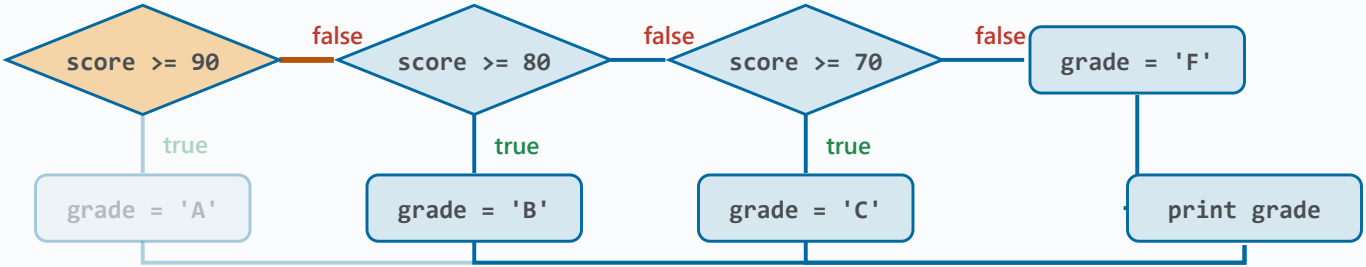
VARIABLES

`score = 75` `grade = ?`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



Condition 1: score >= 90 → 75 >= 90 → **false**. Move on to the next one.

Step 2 / 16

```
1 int score = in.nextInt();
2 char grade;
▶ if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

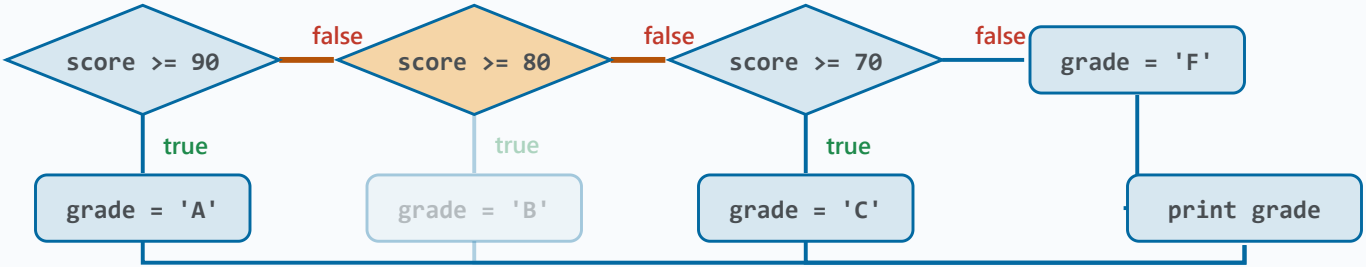
VARIABLES

`score = 75` `grade = ?`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



Condition 2: score >= 80 → 75 >= 80 → **false**. Move on to the next one.

Step 3 / 16

```
1 int score = in.nextInt();
2 char grade;
3 if (score >= 90)      grade = 'A';
▶ else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

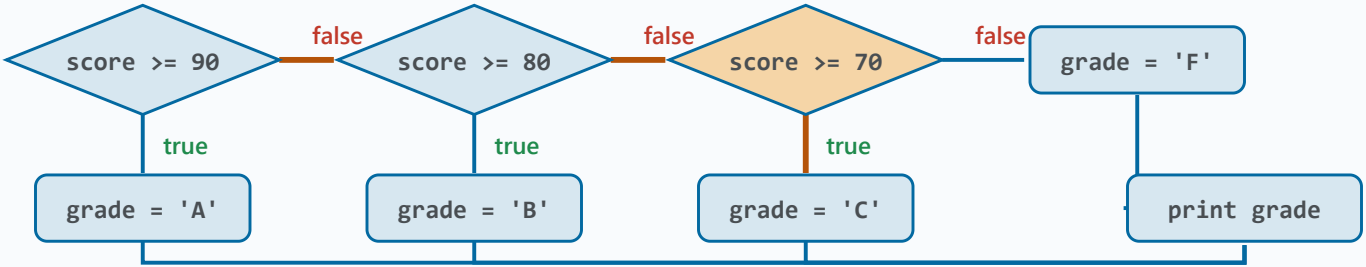
VARIABLES

`score = 75` `grade = ?`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



Condition 3: score >= 70 → 75 >= 70 → **true**. This branch is taken.

Step 4 / 16

```
1 int score = in.nextInt();
2 char grade;
3 if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
▶ else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

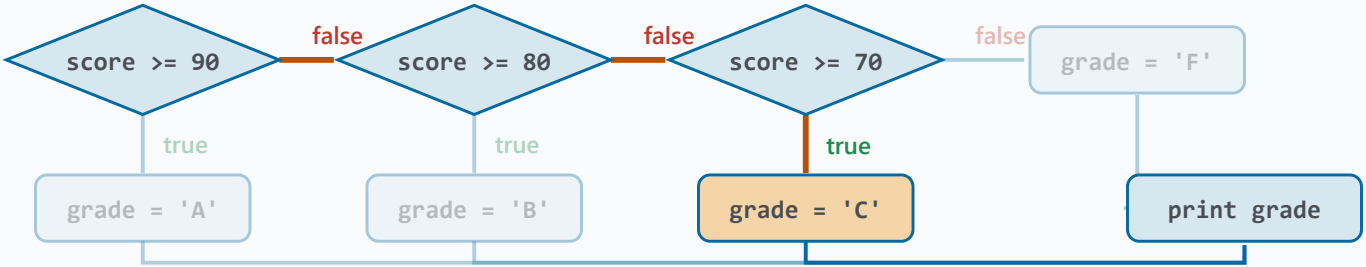
VARIABLES

`score = 75` `grade = ?`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



grade = 'C'. The rest of the chain is **skipped without testing**.

Step 5 / 16

```
1 int score = in.nextInt();
2 char grade;
3 if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
▶ else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

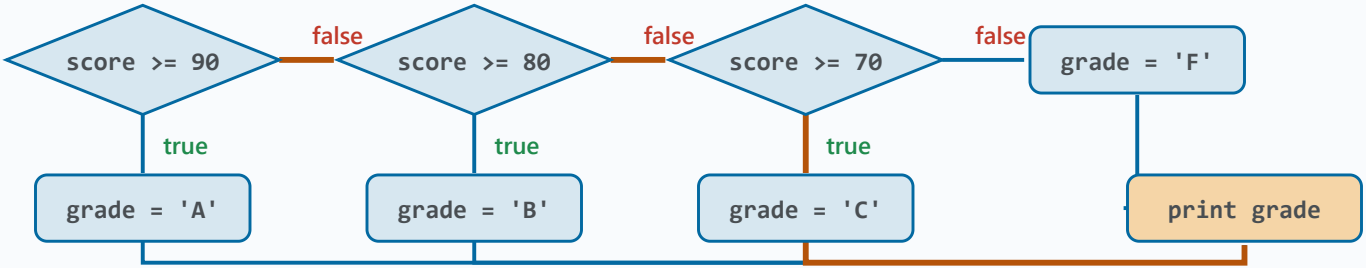
VARIABLES

`score = 75` `grade = 'C'`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



C is printed.

Step 6 / 16

```
1 int score = in.nextInt();
2 char grade;
3 if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
▶ System.out.println(grade);
```

VARIABLES

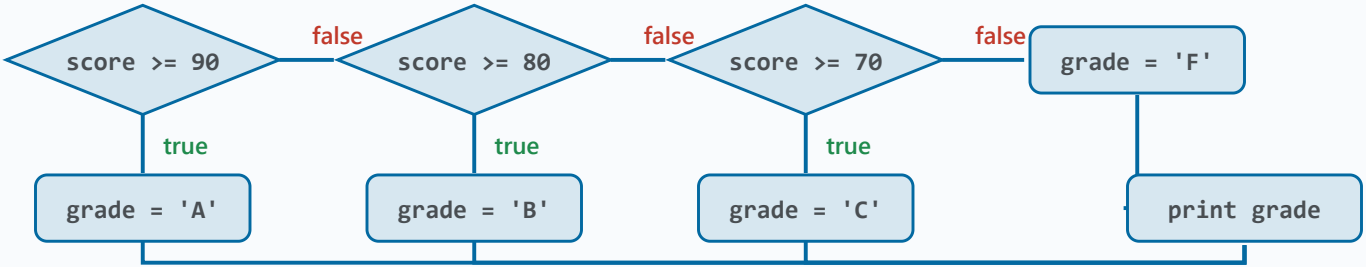
`score = 75` `grade = 'C'`

OUTPUT

```
C
```

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



Run 2: score = **95**. grade is declared but has no value yet.

Step 7 / 16

```
▶ int score = in.nextInt();
▶ char grade;
3 if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

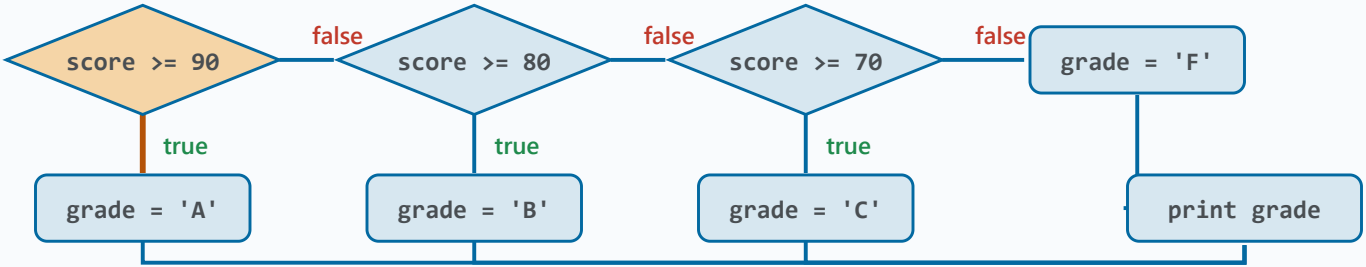
VARIABLES

`score = 95` `grade = ?`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



Condition 1: score >= 90 → 95 >= 90 → **true**. This branch is taken.

Step 8 / 16

```
1 int score = in.nextInt();
2 char grade;
▶ if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

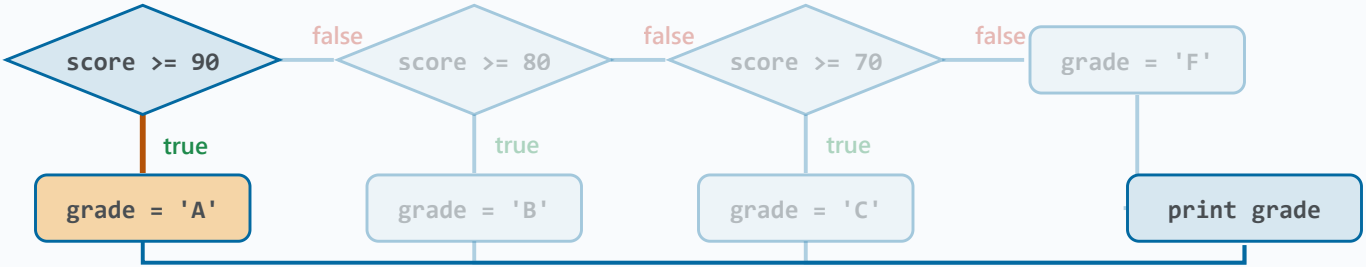
VARIABLES

`score = 95` `grade = ?`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



grade = 'A'. The rest of the chain is **skipped without testing**.

Step 9 / 16

```
1 int score = in.nextInt();
2 char grade;
▶ if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

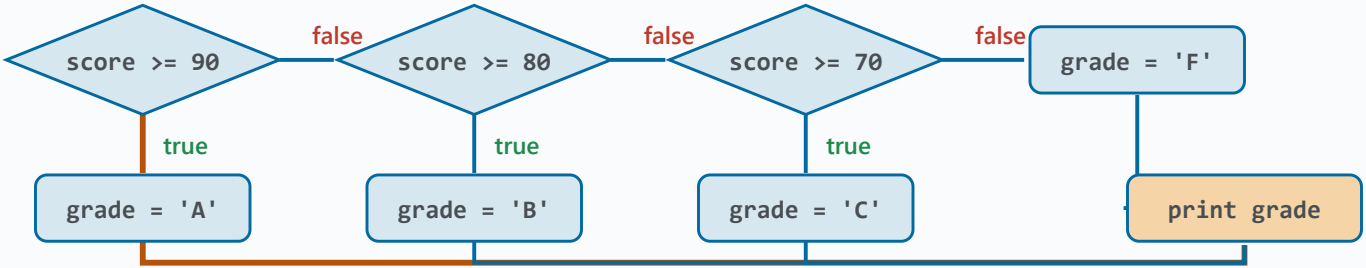
VARIABLES

`score = 95` `grade = 'A'`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



A is printed.

Step 10 / 16

```
1 int score = in.nextInt();
2 char grade;
3 if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
▶ System.out.println(grade);
```

VARIABLES

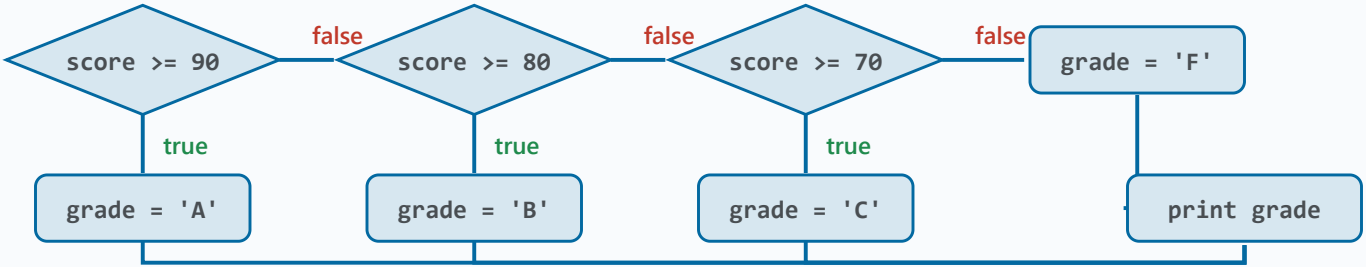
`score = 95` `grade = 'A'`

OUTPUT

```
A
```

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



Run 3: score = **50**. grade is declared but has no value yet.

Step 11 / 16

```
▶ int score = in.nextInt();
▶ char grade;
3 if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

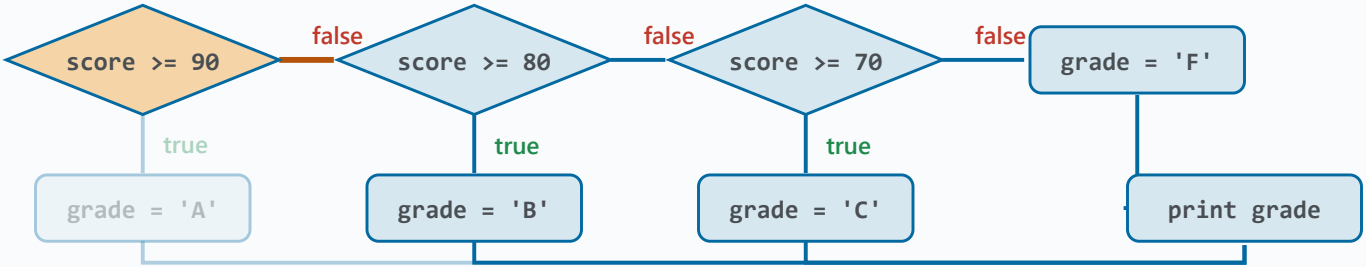
VARIABLES

`score = 50` `grade = ?`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



Condition 1: score >= 90 → 50 >= 90 → **false**. Move on to the next one.

Step 12 / 16

```
1 int score = in.nextInt();
2 char grade;
▶ if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

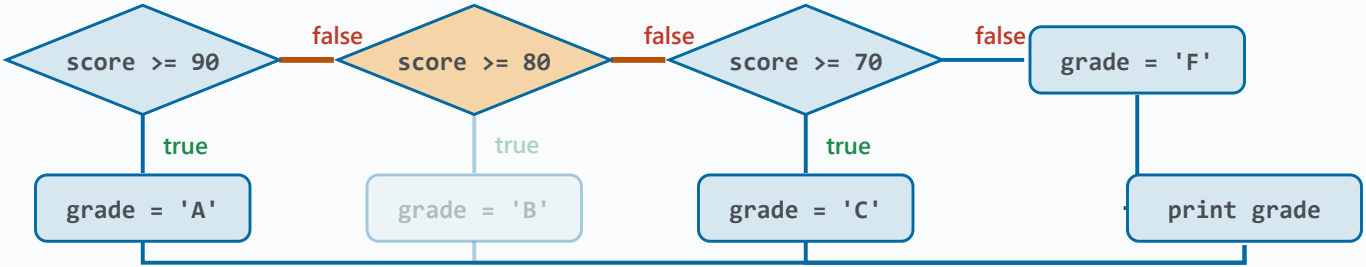
VARIABLES

`score = 50` `grade = ?`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



Condition 2: score >= 80 → 50 >= 80 → **false**. Move on to the next one.

Step 13 / 16

```
1 int score = in.nextInt();
2 char grade;
3 if (score >= 90)      grade = 'A';
▶ else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

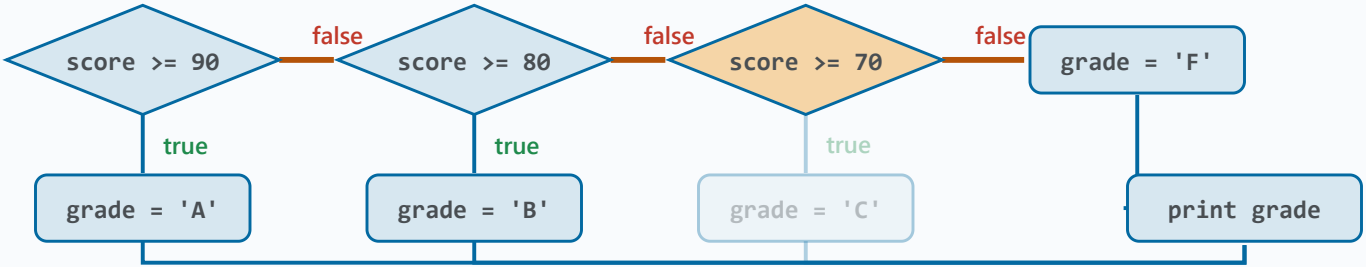
VARIABLES

`score = 50` `grade = ?`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



Condition 3: score >= 70 → 50 >= 70 → **false**. Move on to the next one.

Step 14 / 16

```
1 int score = in.nextInt();
2 char grade;
3 if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
▶ else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
7 System.out.println(grade);
```

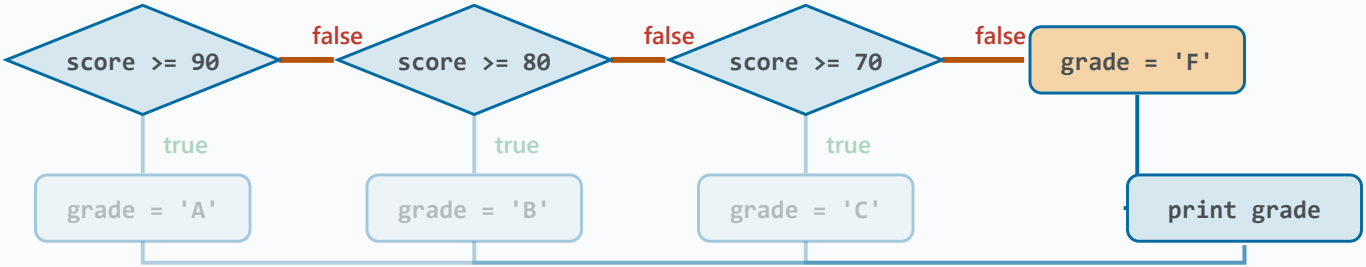
VARIABLES

`score = 50` `grade = ?`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



No condition held → **else**: grade = 'F'.

Step 15 / 16

```
1 int score = in.nextInt();
2 char grade;
3 if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
▶ else                  grade = 'F';
7 System.out.println(grade);
```

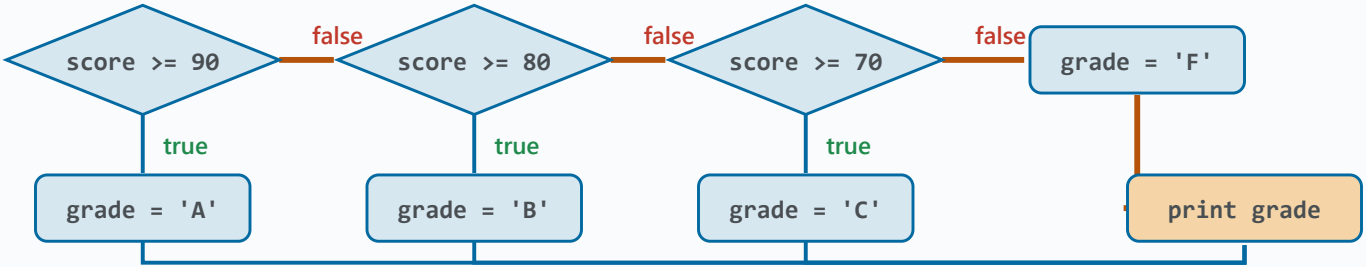
VARIABLES

`score = 50` `grade = 'F'`

OUTPUT

Braces may be omitted when a branch has a single statement.

else-if Chain: score = 75, 95, 50



F is printed.

Step 16 / 16

```
1 int score = in.nextInt();
2 char grade;
3 if (score >= 90)      grade = 'A';
4 else if (score >= 80) grade = 'B';
5 else if (score >= 70) grade = 'C';
6 else                  grade = 'F';
▶ System.out.println(grade);
```

VARIABLES

`score = 50` `grade = 'F'`

OUTPUT

```
F
```

Braces may be omitted when a branch has a single statement.

Independent if Statements: score = 95 → 3 lines

score = 95. The three ifs are independent; none is the else branch of another.

Step 1 / 4

```
▶ int score = 95;
2 if (score >= 90) System.out.println("A");
3 if (score >= 80) System.out.println("B");
4 if (score >= 70) System.out.println("C");
```

VARIABLES

`score = 95`

OUTPUT

Independent if Statements: score = 95 → 3 lines

`95 >= 90` → true → **A** is printed. Because the ifs are independent, the next one is tested **anyway**.

Step 2 / 4

```
1 int score = 95;
▶ if (score >= 90) System.out.println("A");
3 if (score >= 80) System.out.println("B");
4 if (score >= 70) System.out.println("C");
```

VARIABLES

`score = 95`

OUTPUT

```
A
```

Independent if Statements: score = 95 → 3 lines

`95 >= 80` → true → **B** is printed too. The previous if has no effect on this test.

Step 3 / 4

```
1 int score = 95;
2 if (score >= 90) System.out.println("A");
▶ if (score >= 80) System.out.println("B");
4 if (score >= 70) System.out.println("C");
```

VARIABLES

`score = 95`

OUTPUT

```
A
B
```

Independent if Statements: score = 95 → 3 lines

`95 >= 70` → true → **C** is printed too. Result: 3 lines instead of one grade, which is wrong.

Step 4 / 4

```
1 int score = 95;
2 if (score >= 90) System.out.println("A");
3 if (score >= 80) System.out.println("B");
▶ if (score >= 70) System.out.println("C");
```

VARIABLES

`score = 95`

OUTPUT

```
A
B
C
```

Same score = 95, else-if chain → 1 line

Same **score = 95**; this time the conditions are chained with `else if`.

Step 1 / 4

```
▶ int score = 95;
2 if (score >= 90) System.out.println("A");
3 else if (score >= 80) System.out.println("B");
4 else if (score >= 70) System.out.println("C");
```

VARIABLES

`score = 95`

OUTPUT

Same score = 95, else-if chain → 1 line

`95 >= 90` → true → **A** is printed.

Step 2 / 4

```
1 int score = 95;
▶ if (score >= 90) System.out.println("A");
3 else if (score >= 80) System.out.println("B");
4 else if (score >= 70) System.out.println("C");
```

VARIABLES

`score = 95`

OUTPUT

```
A
```

Same score = 95, else-if chain → 1 line

The first branch ran, so the **else if** branches are never evaluated; 80 and 70 are not tested.

Step 3 / 4

```
1 int score = 95;
2 if (score >= 90) System.out.println("A");
▶ else if (score >= 80) System.out.println("B");
▶ else if (score >= 70) System.out.println("C");
```

VARIABLES

`score = 95`

OUTPUT

```
A
```

Same score = 95, else-if chain → 1 line

Result: just **1 line**. Independent ifs print 3 lines (A, B, C); the else-if chain prints 1 (A).

Step 4 / 4

```
1 int score = 95;
2 if (score >= 90) System.out.println("A");
3 else if (score >= 80) System.out.println("B");
▶ else if (score >= 70) System.out.println("C");
```

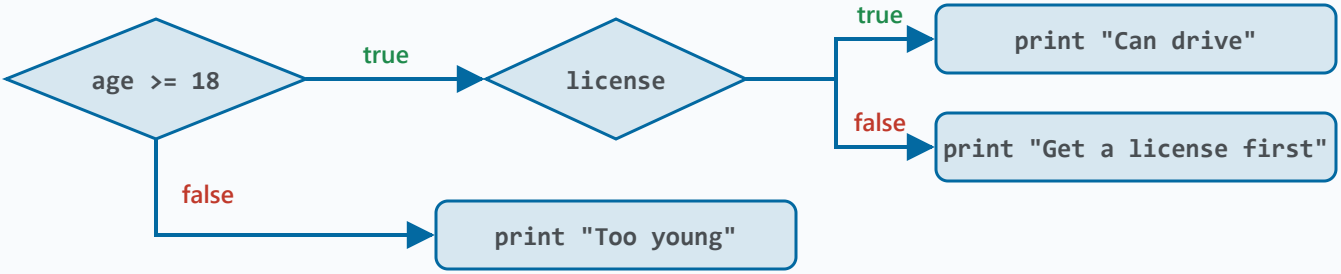
VARIABLES

`score = 95`

OUTPUT

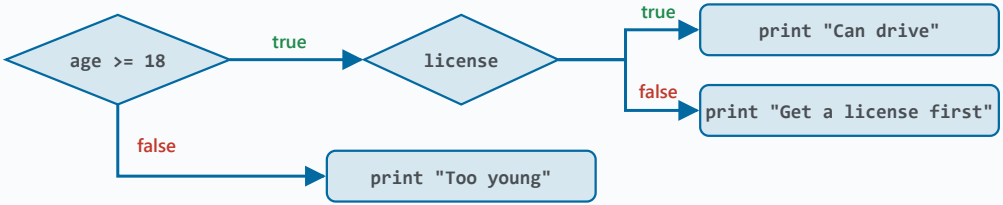
```
A
```

Nested Decisions



- A branch can contain another decision: the inner condition is tested only when the outer one is true.
- Nesting fits cases where the second question only makes sense once the first condition holds.
- Deep nesting hurts readability; conditions can be combined with `&&`.

Nested Decision: 3 different inputs



Run 1: age = 20, license = true.

Step 1 / 11

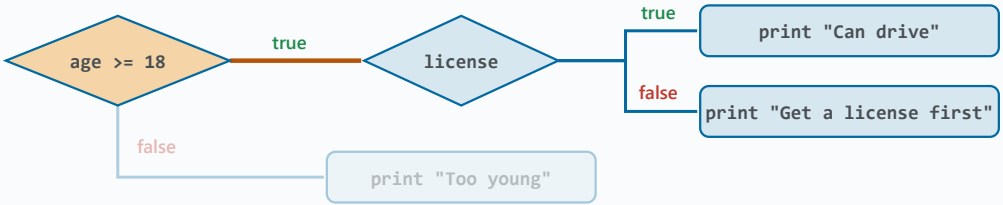
```
▶ int age = in.nextInt();
▶ boolean license = in.nextBoolean();
3 if (age >= 18) {
4   if (license)
5     System.out.println("Can drive");
6   else
7     System.out.println("Get a license first");
8 } else {
9   System.out.println("Too young");
10 }
```

VARIABLES

age = **20** license = **true**

OUTPUT

Nested Decision: 3 different inputs



Outer condition: age >= 18 → 20 >= 18 → **true**. Enter the inner if.

Step 2 / 11

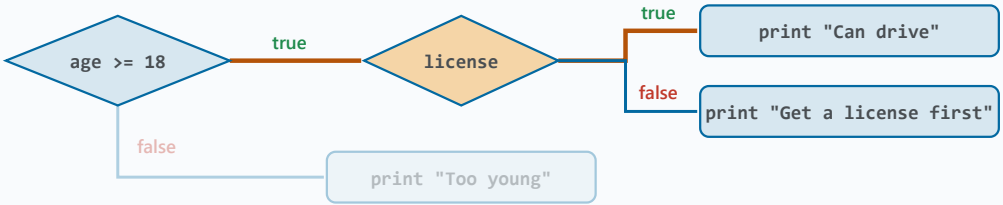
```
1  int age = in.nextInt();
2  boolean license = in.nextBoolean();
▶ if (age >= 18) {
4    if (license)
5      System.out.println("Can drive");
6    else
7      System.out.println("Get a license first");
8  } else {
9    System.out.println("Too young");
10 }
```

VARIABLES

`age = 20` `license = true`

OUTPUT

Nested Decision: 3 different inputs



Inner condition: license → **true**.

Step 3 / 11

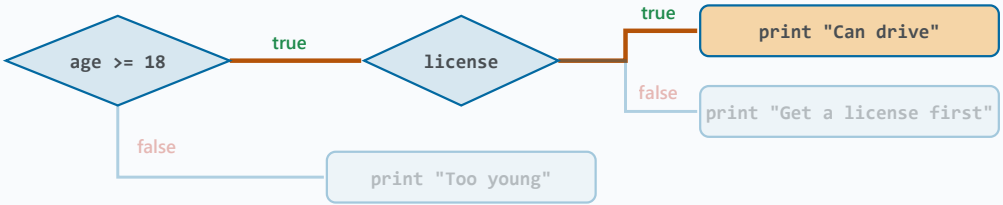
```
1  int age = in.nextInt();
2  boolean license = in.nextBoolean();
3  if (age >= 18) {
▶    if (license)
5      System.out.println("Can drive");
6    else
7      System.out.println("Get a license first");
8  } else {
9    System.out.println("Too young");
10 }
```

VARIABLES

`age = 20` `license = true`

OUTPUT

Nested Decision: 3 different inputs



Can drive is printed.

Step 4 / 11

```
1  int age = in.nextInt();
2  boolean license = in.nextBoolean();
3  if (age >= 18) {
4    if (license)
▶      System.out.println("Can drive");
6    else
7      System.out.println("Get a license first");
8  } else {
9    System.out.println("Too young");
10 }
```

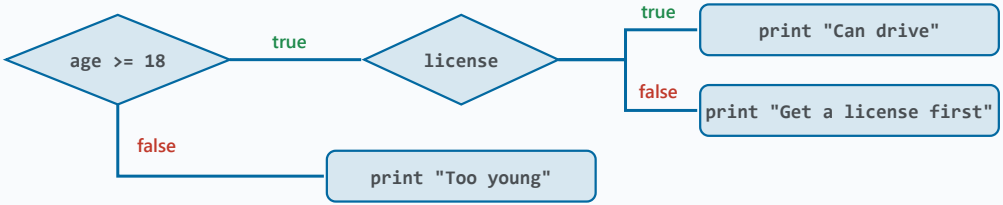
VARIABLES

`age = 20` `license = true`

OUTPUT

```
Can drive
```

Nested Decision: 3 different inputs



Run 2: age = 20, license = false.

Step 5 / 11

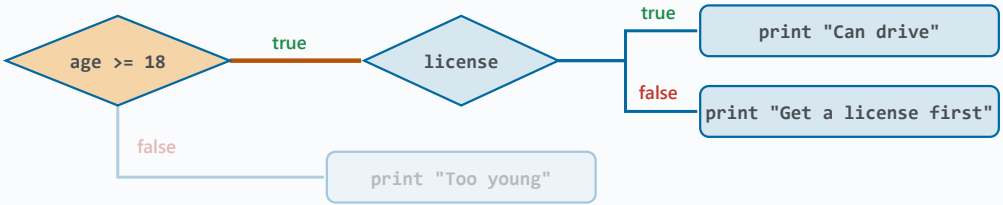
```
▶ int age = in.nextInt();
▶ boolean license = in.nextBoolean();
3 if (age >= 18) {
4   if (license)
5     System.out.println("Can drive");
6   else
7     System.out.println("Get a license first");
8 } else {
9   System.out.println("Too young");
10 }
```

VARIABLES

`age = 20` `license = false`

OUTPUT

Nested Decision: 3 different inputs



Outer condition: age >= 18 → 20 >= 18 → **true**. Enter the inner if.

Step 6 / 11

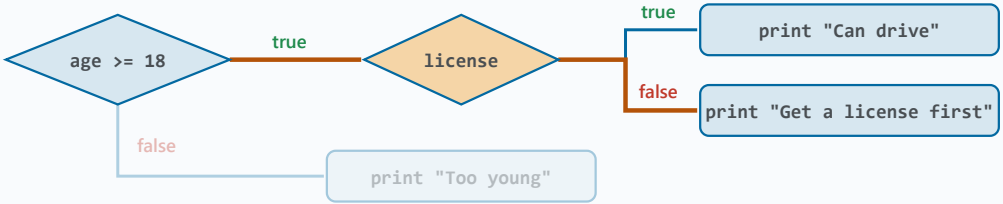
```
1  int age = in.nextInt();
2  boolean license = in.nextBoolean();
▶  if (age >= 18) {
4    if (license)
5      System.out.println("Can drive");
6    else
7      System.out.println("Get a license first");
8  } else {
9    System.out.println("Too young");
10 }
```

VARIABLES

`age = 20` `license = false`

OUTPUT

Nested Decision: 3 different inputs



Inner condition: license → **false**.

Step 7 / 11

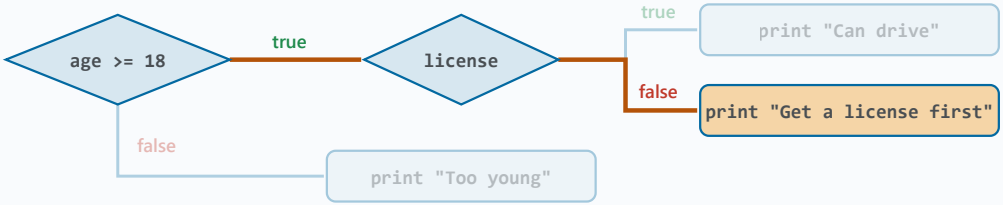
```
1  int age = in.nextInt();
2  boolean license = in.nextBoolean();
3  if (age >= 18) {
▶    if (license)
5      System.out.println("Can drive");
6    else
7      System.out.println("Get a license first");
8  } else {
9    System.out.println("Too young");
10 }
```

VARIABLES

`age = 20` `license = false`

OUTPUT

Nested Decision: 3 different inputs



Get a license first is printed.

Step 8 / 11

```
1  int age = in.nextInt();
2  boolean license = in.nextBoolean();
3  if (age >= 18) {
4    if (license)
5      System.out.println("Can drive");
6    else
▶      System.out.println("Get a license first");
8  } else {
9    System.out.println("Too young");
10 }
```

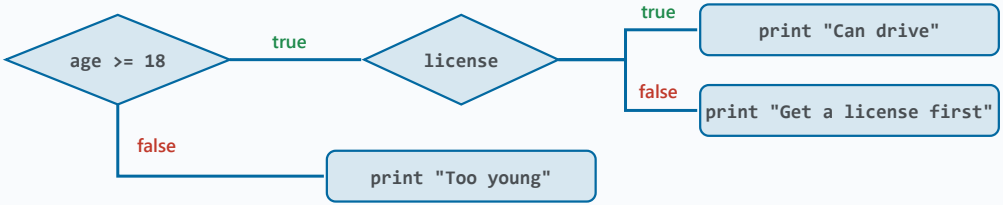
VARIABLES

`age = 20` `license = false`

OUTPUT

```
Get a license first
```

Nested Decision: 3 different inputs



Run 3: age = 15, license = true.

Step 9 / 11

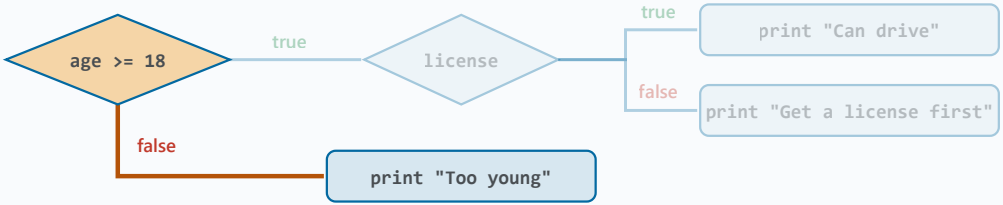
```
▶ int age = in.nextInt();
▶ boolean license = in.nextBoolean();
3 if (age >= 18) {
4   if (license)
5     System.out.println("Can drive");
6   else
7     System.out.println("Get a license first");
8 } else {
9   System.out.println("Too young");
10 }
```

VARIABLES

`age = 15` `license = true`

OUTPUT

Nested Decision: 3 different inputs



Outer condition: 15 >= 18 → **false**. The inner if is **never tested**; license does not matter.

Step 10 / 11

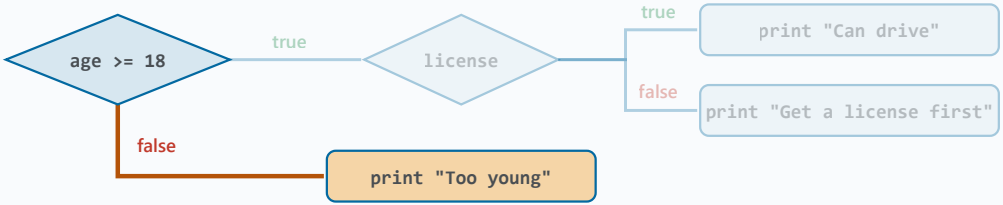
```
1  int age = in.nextInt();
2  boolean license = in.nextBoolean();
▶ if (age >= 18) {
4    if (license)
5      System.out.println("Can drive");
6    else
7      System.out.println("Get a license first");
8  } else {
9    System.out.println("Too young");
10 }
```

VARIABLES

`age = 15` `license = true`

OUTPUT

Nested Decision: 3 different inputs



The outer else runs: **Too young**.

Step 11 / 11

```
1  int age = in.nextInt();
2  boolean license = in.nextBoolean();
3  if (age >= 18) {
4    if (license)
5      System.out.println("Can drive");
6    else
7      System.out.println("Get a license first");
8  } else {
▶    System.out.println("Too young");
10 }
```

VARIABLES

`age = 15` `license = true`

OUTPUT

```
Too young
```

The Dangling else: a = −1

Start: **a = -1**, b = 5, x = 0. The indentation suggests the else belongs to the outer if, but the compiler ignores indentation.

Step 1 / 3

```
1 ▶ int a = -1, b = 5, x = 0;
2   if (a > 0)
3       if (b > 0)
4           x = 1;
5       else
6           x = 2;
7   System.out.println(x);
```

VARIABLES

a = **−1** b = **5** x = **0**

OUTPUT

The Dangling else: a = −1

`a > 0` → −1 > 0 → **false**. The whole body of the outer if (inner if and its else included) is skipped.

Step 2 / 3

```
1 int a = -1, b = 5, x = 0;
▶ if (a > 0)
3     if (b > 0)
4         x = 1;
5     else
6         x = 2;
7 System.out.println(x);
```

VARIABLES

`a = -1` `b = 5` `x = 0`

OUTPUT

The Dangling else: a = −1

else binds to the nearest if (`if (b > 0)`); that if never ran, so neither did the else. x is unchanged: **0** is printed.

Step 3 / 3

```
1 int a = -1, b = 5, x = 0;
2 if (a > 0)
3     if (b > 0)
4         x = 1;
5     else
6         x = 2;
▶ System.out.println(x);
```

VARIABLES

`a = −1` `b = 5` `x = 0`

OUTPUT

```
0
```

Fixing the Dangling else with Braces

Same starting values. This time the inner if is wrapped in braces; **the else belongs to the outer if**.

Step 1 / 4

```
▶ int a = -1, b = 5, x = 0;
2 if (a > 0) {
3     if (b > 0) x = 1;
4 } else {
5     x = 2;
6 }
7 System.out.println(x);
```

VARIABLES

a = −1 | b = 5 | x = 0

OUTPUT

Fixing the Dangling else with Braces

`a > 0` → **false**. The inner if block is skipped and control goes to the `else` block.

Step 2 / 4

```
1 int a = -1, b = 5, x = 0;
▶ if (a > 0) {
3     if (b > 0) x = 1;
▶ } else {
5     x = 2;
6 }
7 System.out.println(x);
```

VARIABLES

`a = -1` `b = 5` `x = 0`

OUTPUT

Fixing the Dangling else with Braces

The else block runs: **x = 2**.

Step 3 / 4

```
1 int a = -1, b = 5, x = 0;
2 if (a > 0) {
3     if (b > 0) x = 1;
4 } else {
▶     x = 2;
6 }
7 System.out.println(x);
```

VARIABLES

a = **−1** b = **5** x = **2**

OUTPUT

Fixing the Dangling else with Braces

2 is printed. Always use braces with nested ifs to remove the ambiguity.

Step 4 / 4

```
1 int a = -1, b = 5, x = 0;
2 if (a > 0) {
3     if (b > 0) x = 1;
4 } else {
5     x = 2;
6 }
▶ System.out.println(x);
```

VARIABLES

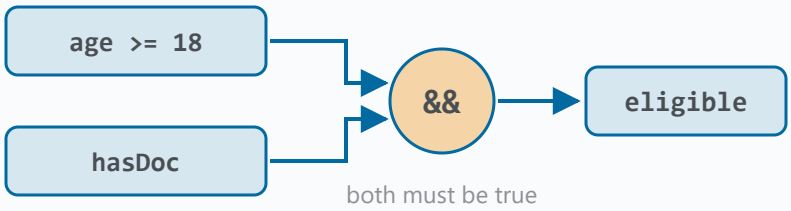
a = **−1** b = **5** x = **2**

OUTPUT

```
2
```

Decisions with Logical Operators

```
boolean eligible = age >= 18 && hasDoc;
if (eligible) {
    System.out.println("Application accepted");
}
```



age >= 18 ↓ \ hasDoc	true	false
true	true	false
false	false	false

value of eligible

- && requires both conditions to be true.
- || requires at least one of them to be true.

Deciding with `&&`: 3 different inputs



Run 1: age = 20, hasDoc = true.

Step 1 / 9

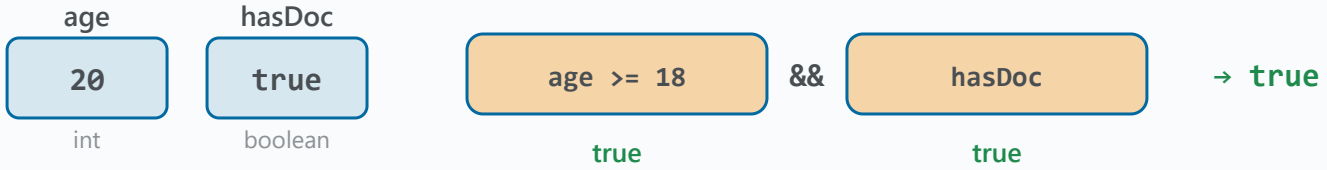
```
▶ int age = in.nextInt();
▶ boolean hasDoc = in.nextBoolean();
3 boolean eligible = age >= 18 && hasDoc;
4 if (eligible) {
5     System.out.println("Application
  accepted");
6 }
```

VARIABLES

age = 20 | hasDoc = true

OUTPUT

Deciding with `&&`: 3 different inputs



20 >= 18 → true; the right side is needed: hasDoc → true. true && true → **true**.

Step 2 / 9

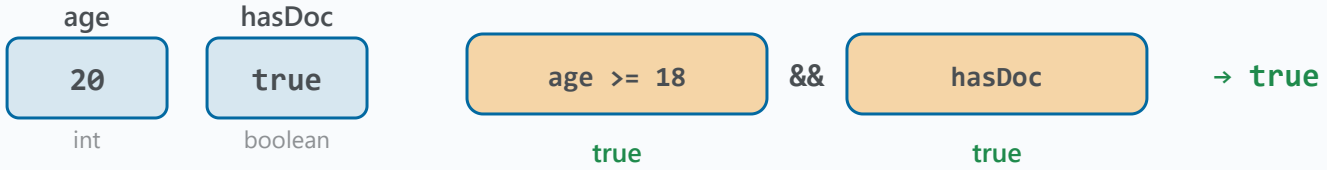
```
1 int age = in.nextInt();
2 boolean hasDoc = in.nextBoolean();
▶ boolean eligible = age >= 18 && hasDoc;
4 if (eligible) {
5     System.out.println("Application
  accepted");
6 }
```

VARIABLES

`age = 20` `hasDoc = true` `eligible = true`

OUTPUT

Deciding with `&&`: 3 different inputs



Application accepted is printed.

Step 3 / 9

```
1 int age = in.nextInt();
2 boolean hasDoc = in.nextBoolean();
3 boolean eligible = age >= 18 && hasDoc;
4 if (eligible) {
▶     System.out.println("Application
  accepted");
6 }
```

VARIABLES

`age = 20` `hasDoc = true` `eligible = true`

OUTPUT

```
Application accepted
```

Deciding with `&&`: 3 different inputs



Run 2: age = 20, hasDoc = false.

Step 4 / 9

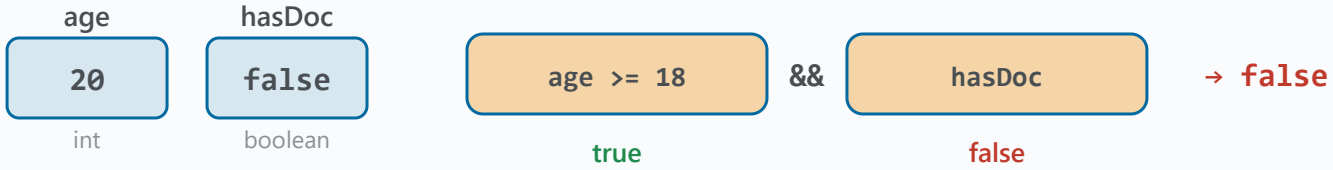
```
▶ int age = in.nextInt();
▶ boolean hasDoc = in.nextBoolean();
3 boolean eligible = age >= 18 && hasDoc;
4 if (eligible) {
5     System.out.println("Application
  accepted");
6 }
```

VARIABLES

age = 20 hasDoc = false

OUTPUT

Deciding with `&&`: 3 different inputs



20 >= 18 → true; the right side is needed: hasDoc → false. true && false → **false**.

Step 5 / 9

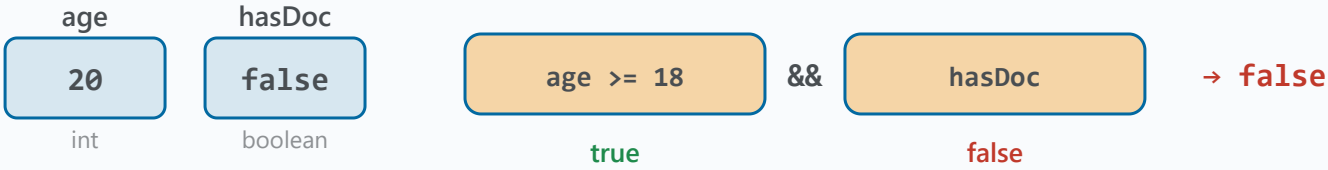
```
1 int age = in.nextInt();
2 boolean hasDoc = in.nextBoolean();
▶ boolean eligible = age >= 18 && hasDoc;
4 if (eligible) {
5     System.out.println("Application
  accepted");
6 }
```

VARIABLES

`age = 20` `hasDoc = false` `eligible = false`

OUTPUT

Deciding with `&&`: 3 different inputs



eligible is false → the block is skipped, no output.

Step 6 / 9

```
1 int age = in.nextInt();
2 boolean hasDoc = in.nextBoolean();
3 boolean eligible = age >= 18 && hasDoc;
▶ if (eligible) {
5     System.out.println("Application
  accepted");
6 }
```

VARIABLES

`age = 20` `hasDoc = false` `eligible = false`

OUTPUT

Deciding with `&&`: 3 different inputs



Run 3: age = 16, hasDoc = true.

Step 7 / 9

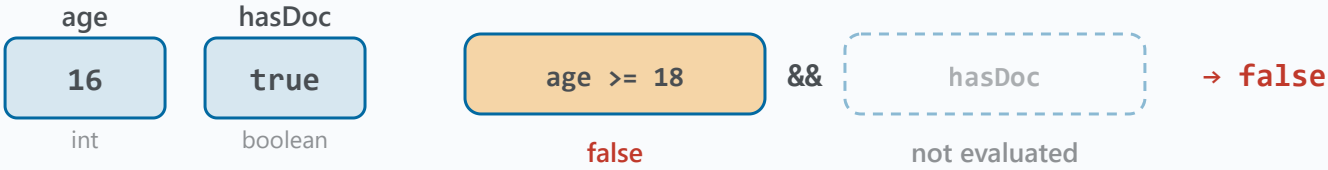
```
▶ int age = in.nextInt();
▶ boolean hasDoc = in.nextBoolean();
3 boolean eligible = age >= 18 && hasDoc;
4 if (eligible) {
5     System.out.println("Application
  accepted");
6 }
```

VARIABLES

`age = 16` `hasDoc = true`

OUTPUT

Deciding with `&&`: 3 different inputs



16 >= 18 → **false**. Short circuit: hasDoc is **not read**; the result is already false.

Step 8 / 9

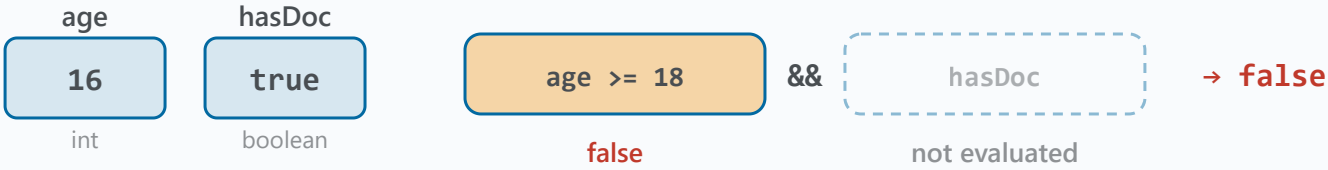
```
1 int age = in.nextInt();
2 boolean hasDoc = in.nextBoolean();
▶ boolean eligible = age >= 18 && hasDoc;
4 if (eligible) {
5     System.out.println("Application
  accepted");
6 }
```

VARIABLES

`age = 16` `hasDoc = true` `eligible = false`

OUTPUT

Deciding with `&&`: 3 different inputs



eligible is false → the block is skipped, no output.

Step 9 / 9

```
1 int age = in.nextInt();
2 boolean hasDoc = in.nextBoolean();
3 boolean eligible = age >= 18 && hasDoc;
▶ if (eligible) {
5     System.out.println("Application
  accepted");
6 }
```

VARIABLES

`age = 16` `hasDoc = true` `eligible = false`

OUTPUT

! and De Morgan's Laws

a	b	!(a && b)	!a || !b	!(a || b)	!a && !b
false	false	true	true	true	true
false	true	true	true	false	false
true	false	true	true	false	false
true	true	false	false	false	false

```
// Reject: under 18 or no document
if (!(age >= 18 && hasDoc)) { … }
// Same condition by De Morgan:
if (age < 18 || !hasDoc) { … }
```

- Pushing `!` inside the parentheses swaps `&&` and `||` and negates each term.
- Negated comparisons: `>=` ↔ `<`, `>` ↔ `<=`, `==` ↔ `!=`.

Non-Short-Circuit `&` and `|`

```
int a = 10, b = 0;
if (b != 0 && a / b > 1) { … }  // safe: a / b never runs
if (b != 0 &  a / b > 1) { … }  // ArithmeticException: / by zero
```

The single-character `&` and `|` also work on booleans, but they **always evaluate both sides**.

```
static int count = 0;
static boolean check(boolean v) { count++; return v; }
```

Expression (count starts at 0)	count
`check(false) && check(true)`	1
`check(false) & check(true)`	2
`check(true) || check(true)`	1
`check(true) | check(true)`	2

The Conditional Operator: `(a > b) ? a : b`



a = 7, b = 12.

Step 1 / 4

```
▶ int a = 7, b = 12;
2 int max = (a > b) ? a : b;
3 System.out.println(max);
```

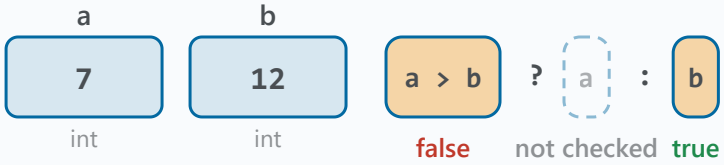
VARIABLES

a = 7 b = 12

OUTPUT

Use it for short two-way value choices; nested ?: hurts readability.

The Conditional Operator: `(a > b) ? a : b`



Condition: a > b → 7 > 12 → **false**, so the value after **:** is chosen.

Step 2 / 4

```
1 int a = 7, b = 12;
▶ int max = (a > b) ? a : b;
3 System.out.println(max);
```

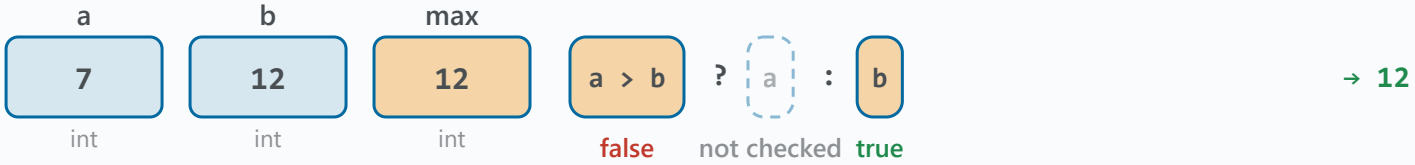
VARIABLES

`a = 7` `b = 12`

OUTPUT

Use it for short two-way value choices; nested ?: hurts readability.

The Conditional Operator: `(a > b) ? a : b`



max = 12. `?:` is an **expression**: it yields a value that can be assigned.

Step 3 / 4

```
1 int a = 7, b = 12;
▶ int max = (a > b) ? a : b;
3 System.out.println(max);
```

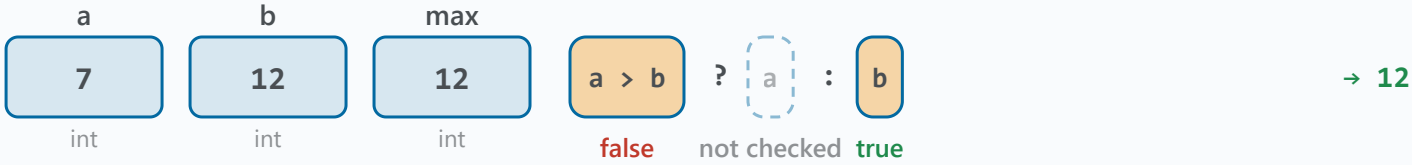
VARIABLES

`a = 7` `b = 12` `max = 12`

OUTPUT

Use it for short two-way value choices; nested ?: hurts readability.

The Conditional Operator: `(a > b) ? a : b`



12 is printed. The same if-else takes 4 lines.

Step 4 / 4

```
1 int a = 7, b = 12;
2 int max = (a > b) ? a : b;
▶ System.out.println(max);
```

VARIABLES

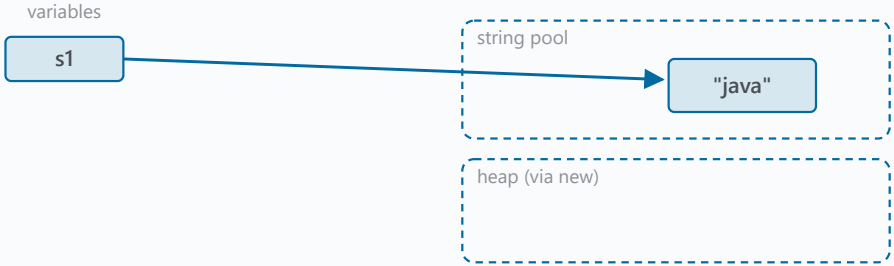
`a = 7` `b = 12` `max = 12`

OUTPUT

```
12
```

Use it for short two-way value choices; nested ?: hurts readability.

Comparing Strings: `==` vs. `equals`



The literal `"java"` is created in the string pool; **s1** holds a reference to it.

Step 1 / 6

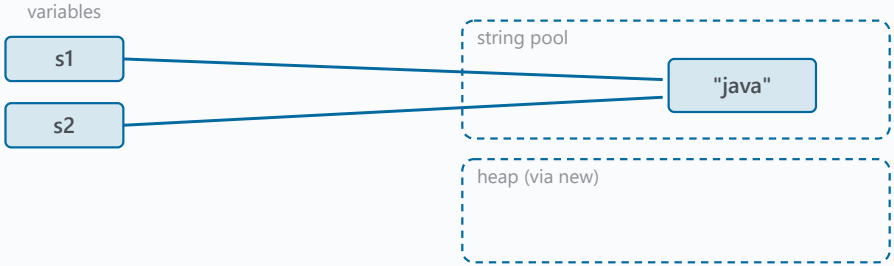
```
▶ String s1 = "java";
2 String s2 = "java";
3 String s3 = new String("java");
4 System.out.println(s1 == s2);
5 System.out.println(s1 == s3);
6 System.out.println(s1.equals(s3));
```

VARIABLES

`s1 = →pool`

OUTPUT

Comparing Strings: `==` vs. `equals`



Writing the same literal again **shares** the pooled object; s2 points to the same object.

Step 2 / 6

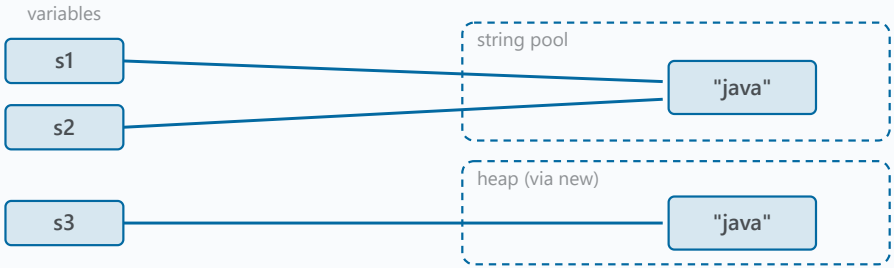
```
1 String s1 = "java";
▶ String s2 = "java";
3 String s3 = new String("java");
4 System.out.println(s1 == s2);
5 System.out.println(s1 == s3);
6 System.out.println(s1.equals(s3));
```

VARIABLES

`s1 = →pool` `s2 = →pool`

OUTPUT

Comparing Strings: `==` vs. `equals`



`new String("java")` always creates **a new object**; s3 points to a separate object on the heap.

Step 3 / 6

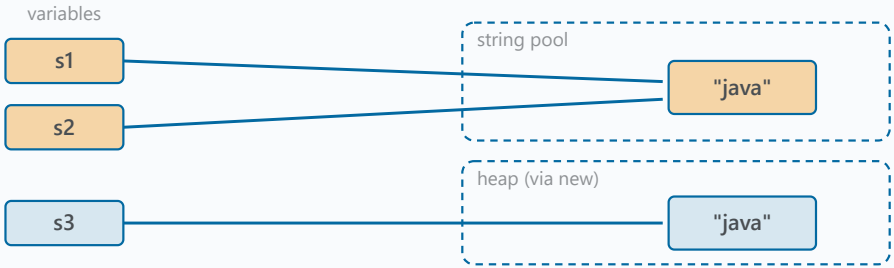
```
1 String s1 = "java";
2 String s2 = "java";
▶ String s3 = new String("java");
4 System.out.println(s1 == s2);
5 System.out.println(s1 == s3);
6 System.out.println(s1.equals(s3));
```

VARIABLES

`s1 = →pool` `s2 = →pool` `s3 = →heap`

OUTPUT

Comparing Strings: `==` vs. `equals`



`==` compares references (addresses). s1 and s2 point to the same object, so **true**.

Step 4 / 6

```
1 String s1 = "java";
2 String s2 = "java";
3 String s3 = new String("java");
▶ System.out.println(s1 == s2);
5 System.out.println(s1 == s3);
6 System.out.println(s1.equals(s3));
```

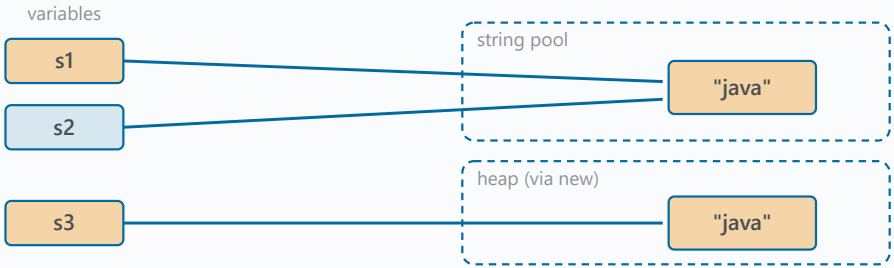
VARIABLES

`s1 = →pool` `s2 = →pool` `s3 = →heap`

OUTPUT

```
true
```

Comparing Strings: `==` vs. `equals`



s1 and s3 have the same contents but are different objects → different addresses → **false**.

Step 5 / 6

```
1 String s1 = "java";
2 String s2 = "java";
3 String s3 = new String("java");
4 System.out.println(s1 == s2);
▶ System.out.println(s1 == s3);
6 System.out.println(s1.equals(s3));
```

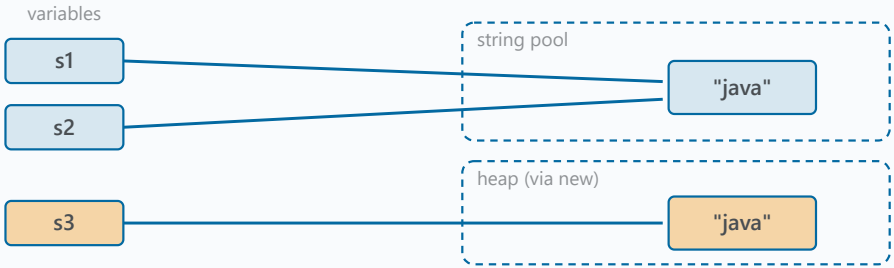
VARIABLES

`s1 = →pool` `s2 = →pool` `s3 = →heap`

OUTPUT

```
true
false
```

Comparing Strings: `==` vs. `equals`



`equals` compares the **contents**: "java" = "java" → **true**. Always compare strings with equals.

Step 6 / 6

```
1 String s1 = "java";
2 String s2 = "java";
3 String s3 = new String("java");
4 System.out.println(s1 == s2);
5 System.out.println(s1 == s3);
▶ System.out.println(s1.equals(s3));
```

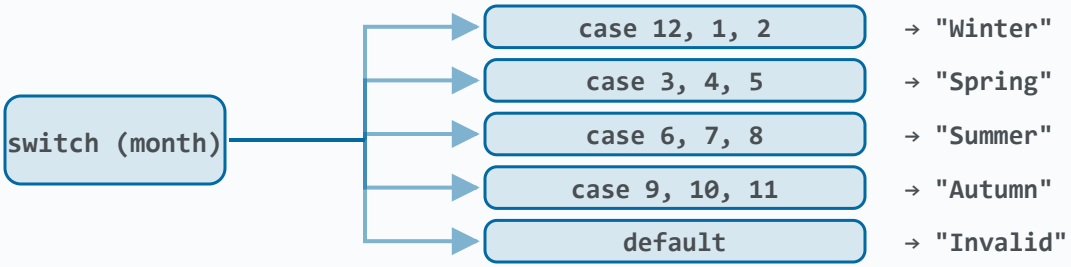
VARIABLES

`s1 = →pool` `s2 = →pool` `s3 = →heap`

OUTPUT

```
true
false
true
```

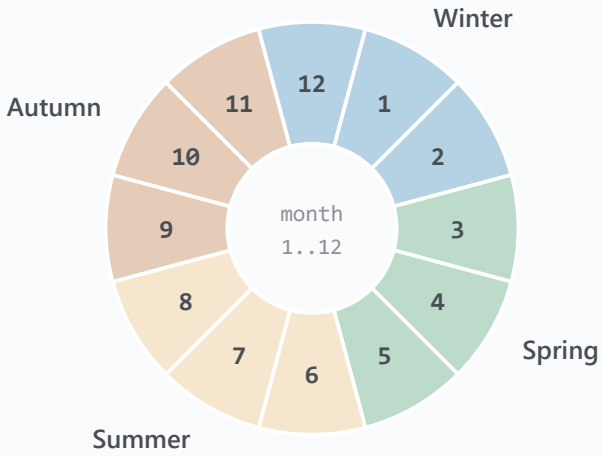
When to Use switch



- It suits a choice among the **discrete values** of a single expression.
- case labels name the fixed options; control jumps directly to the matching case.
- For ranges and complex logical conditions, if-else is usually a better fit.

The switch Expression

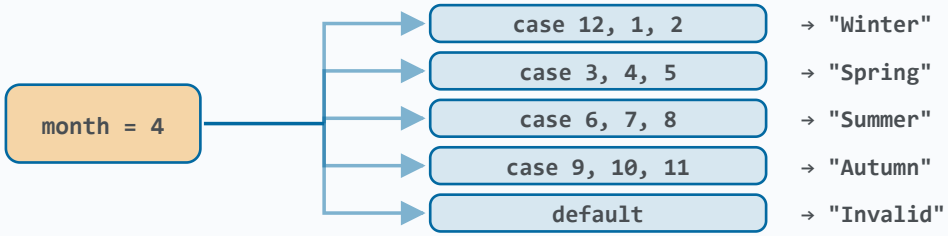
```
String season = switch (month) {
    case 12, 1, 2 -> "Winter";
    case 3, 4, 5 -> "Spring";
    case 6, 7, 8 -> "Summer";
    case 9, 10, 11 -> "Autumn";
    default -> "Invalid";
};
```



a value like 13 → default: "Invalid"

- The arrow form prevents accidental fall-through into the next case.
- default handles unexpected values. A switch **expression** (`x = switch …`) must cover every value, so for int and String `default` is required; in a switch statement it is optional. This syntax needs Java 14 or later.

switch Expression: month = 4, 12, 13



Run 1: month = **4**.

Step 1 / 12

```
1 int month = in.nextInt();
2 String season = switch (month) {
3     case 12, 1, 2 -> "Winter";
4     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
7     default -> "Invalid";
8 };
9 System.out.println(season);
```

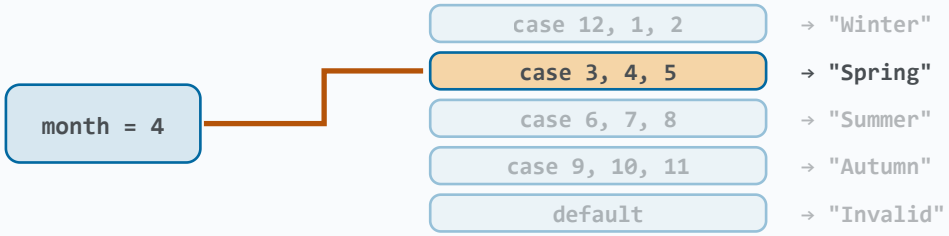
VARIABLES

month = 4

OUTPUT

SWITCH · STEP THROUGH THE CODE

switch Expression: month = 4, 12, 13



switch (month) → 4: 4 is in this case's list. The selected branch yields **"Spring"**.

Step 2 / 12

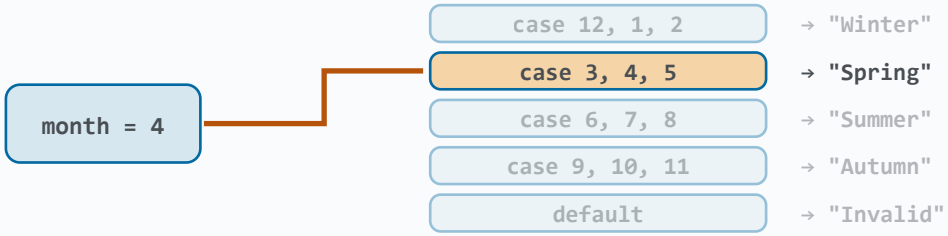
```
1 int month = in.nextInt();
2 String season = switch (month) {
3     case 12, 1, 2 -> "Winter";
▶     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
7     default -> "Invalid";
8 };
9 System.out.println(season);
```

VARIABLES

`month = 4`

OUTPUT

switch Expression: month = 4, 12, 13



The value of the switch expression is assigned: season = "Spring". No other case runs.

Step 3 / 12

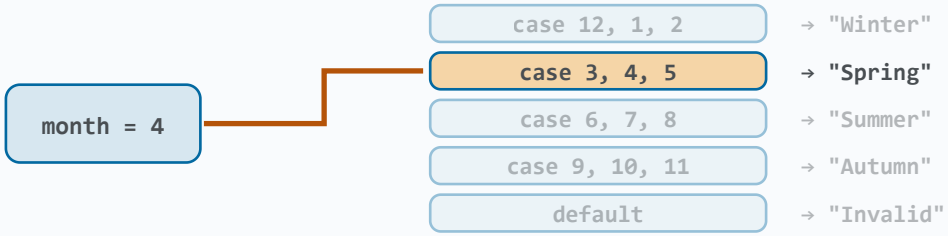
```
1 int month = in.nextInt();
▶ String season = switch (month) {
3     case 12, 1, 2 -> "Winter";
4     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
7     default -> "Invalid";
8 };
9 System.out.println(season);
```

VARIABLES

month = 4 season = "Spring"

OUTPUT

switch Expression: month = 4, 12, 13



Spring is printed.

Step 4 / 12

```
1 int month = in.nextInt();
2 String season = switch (month) {
3     case 12, 1, 2 -> "Winter";
4     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
7     default -> "Invalid";
8 };
▶ System.out.println(season);
```

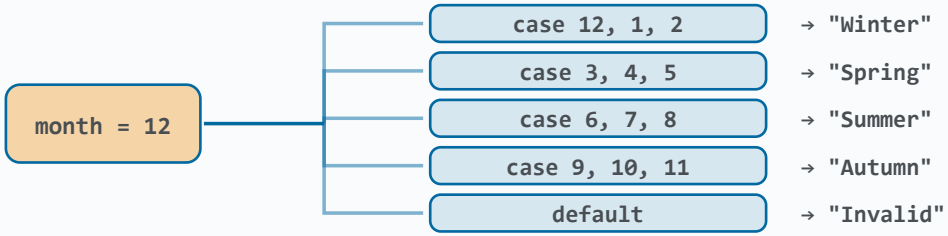
VARIABLES

`month = 4` `season = "Spring"`

OUTPUT

```
Spring
```

switch Expression: month = 4, 12, 13



Run 2: month = **12**.

Step 5 / 12

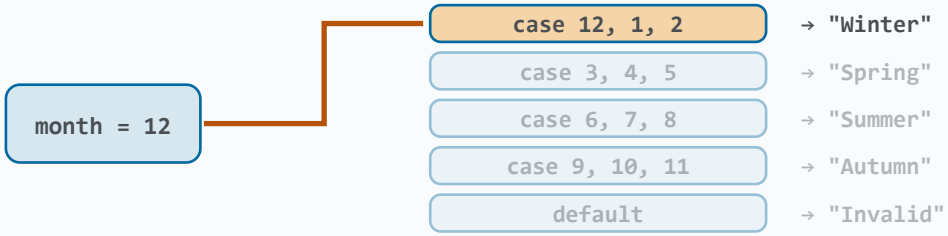
```
▶ int month = in.nextInt();
2 String season = switch (month) {
3     case 12, 1, 2 -> "Winter";
4     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
7     default -> "Invalid";
8 };
9 System.out.println(season);
```

VARIABLES

month = 12

OUTPUT

switch Expression: month = 4, 12, 13



switch (month) → 12: 12 is in this case's list. The selected branch yields **"Winter"**.

Step 6 / 12

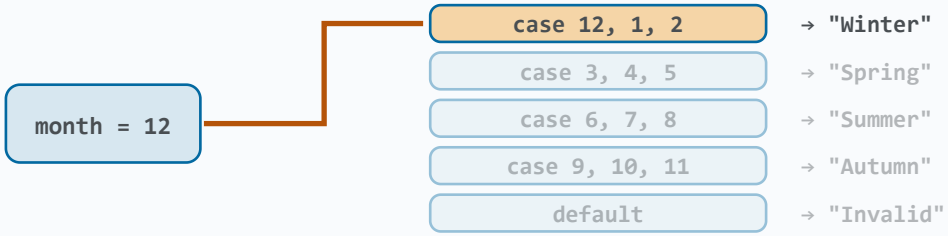
```
1 int month = in.nextInt();
2 String season = switch (month) {
▶     case 12, 1, 2 -> "Winter";
4     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
7     default -> "Invalid";
8 };
9 System.out.println(season);
```

VARIABLES

`month = 12`

OUTPUT

switch Expression: month = 4, 12, 13



The value of the switch expression is assigned: season = "Winter". No other case runs.

Step 7 / 12

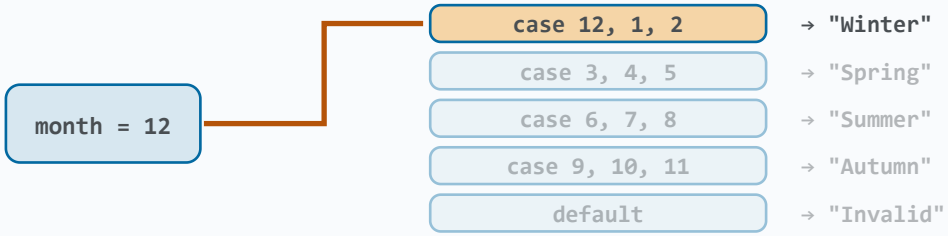
```
1 int month = in.nextInt();
▶ String season = switch (month) {
3     case 12, 1, 2 -> "Winter";
4     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
7     default -> "Invalid";
8 };
9 System.out.println(season);
```

VARIABLES

month = 12 season = "Winter"

OUTPUT

switch Expression: month = 4, 12, 13



Winter is printed.

Step 8 / 12

```
1 int month = in.nextInt();
2 String season = switch (month) {
3     case 12, 1, 2 -> "Winter";
4     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
7     default -> "Invalid";
8 };
▶ System.out.println(season);
```

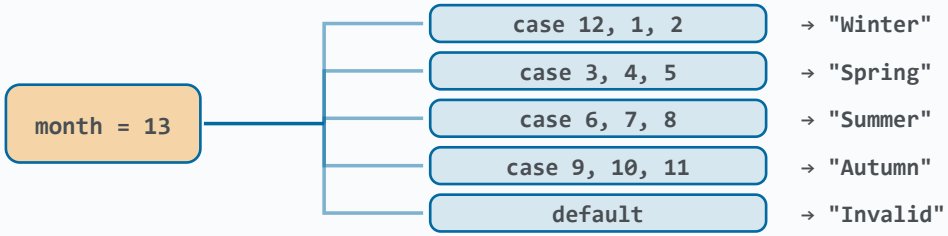
VARIABLES

`month = 12` `season = "Winter"`

OUTPUT

```
Winter
```

switch Expression: month = 4, 12, 13



Run 3: month = **13**.

Step 9 / 12

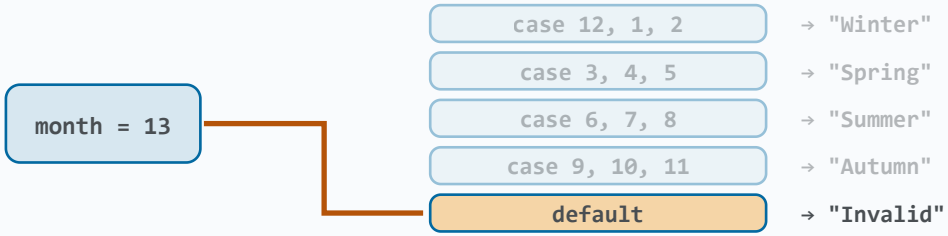
```
▶ int month = in.nextInt();
2 String season = switch (month) {
3     case 12, 1, 2 -> "Winter";
4     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
7     default -> "Invalid";
8 };
9 System.out.println(season);
```

VARIABLES

month = 13

OUTPUT

switch Expression: month = 4, 12, 13



switch (month) → 13: no case lists 13 → **default**. The selected branch yields **"Invalid"**.

Step 10 / 12

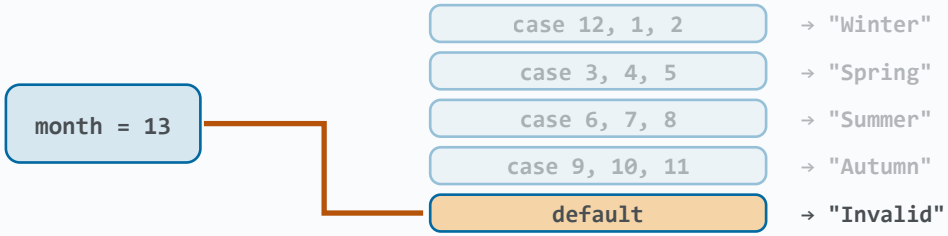
```
1 int month = in.nextInt();
2 String season = switch (month) {
3     case 12, 1, 2 -> "Winter";
4     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
▶     default -> "Invalid";
8 };
9 System.out.println(season);
```

VARIABLES

`month = 13`

OUTPUT

switch Expression: month = 4, 12, 13



The value of the switch expression is assigned: season = "Invalid". No other case runs.

Step 11 / 12

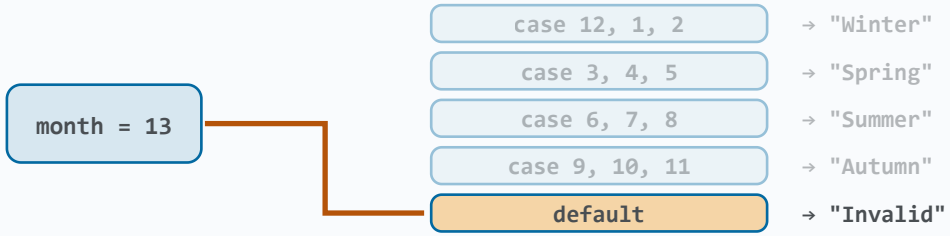
```
1 int month = in.nextInt();
▶ String season = switch (month) {
3     case 12, 1, 2 -> "Winter";
4     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
7     default -> "Invalid";
8 };
9 System.out.println(season);
```

VARIABLES

`month = 13` `season = "Invalid"`

OUTPUT

switch Expression: month = 4, 12, 13



Invalid is printed.

Step 12 / 12

```
1 int month = in.nextInt();
2 String season = switch (month) {
3     case 12, 1, 2 -> "Winter";
4     case 3, 4, 5 -> "Spring";
5     case 6, 7, 8 -> "Summer";
6     case 9, 10, 11 -> "Autumn";
7     default -> "Invalid";
8 };
▶ System.out.println(season);
```

VARIABLES

`month = 13` `season = "Invalid"`

OUTPUT

```
Invalid
```

SWITCH

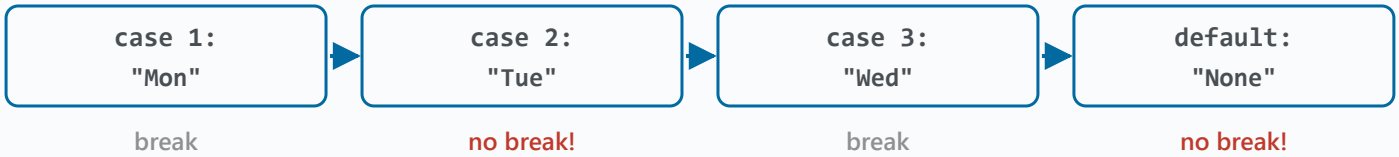
`yield` in a switch Expression

```
int year = 2024, month = 2;
int days = switch (month) {
    case 4, 6, 9, 11 -> 30;
    case 2 -> {
        System.out.println("February");
        yield year % 4 == 0 ? 29 : 28;
    }
    default -> 31;
};
System.out.println(days);
```

Output: `February` and `29` (two lines).

- A single-expression branch gives its value directly; when several statements are needed, wrap the branch in a `{ }` block and give the value with `yield`.
- A block that ends without `yield` is a compile error: `switch rule completes without providing a value`.
- Requires Java 14 or later; the leap-year test is simplified here (no 100/400 rule).

Classic switch: Forgetting `break`



day = 2.

Step 1 / 5

```
▶ int day = 2;
2 switch (day) {
3   case 1: System.out.println("Mon"); break;
4   case 2: System.out.println("Tue");
5   case 3: System.out.println("Wed"); break;
6   default: System.out.println("None");
7 }
```

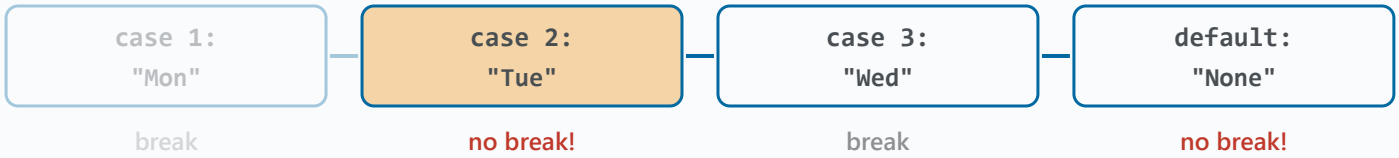
VARIABLES

day = 2

OUTPUT

SWITCH · STEP THROUGH THE CODE

Classic switch: Forgetting `break`



switch (day) → 2: control jumps straight to `case 2`; case 1 is never tested.

Step 2 / 5

```
1 int day = 2;
▶ switch (day) {
3   case 1: System.out.println("Mon"); break;
4   case 2: System.out.println("Tue");
5   case 3: System.out.println("Wed"); break;
6   default: System.out.println("None");
7 }
```

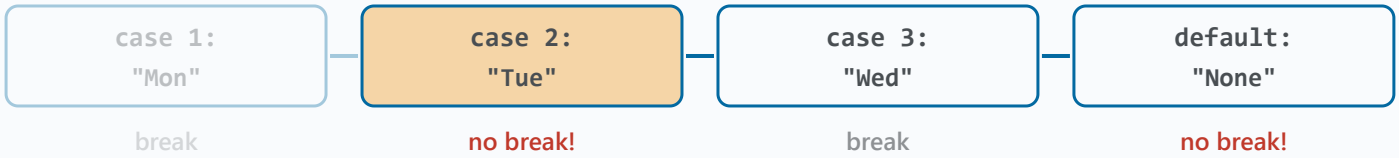
VARIABLES

day = **2**

OUTPUT

SWITCH · STEP THROUGH THE CODE

Classic switch: Forgetting `break`



"Tue" is printed. This case has **no break**!

Step 3 / 5

```
1 int day = 2;
2 switch (day) {
3   case 1: System.out.println("Mon"); break;
▶   case 2: System.out.println("Tue");
5   case 3: System.out.println("Wed"); break;
6   default: System.out.println("None");
7 }
```

VARIABLES

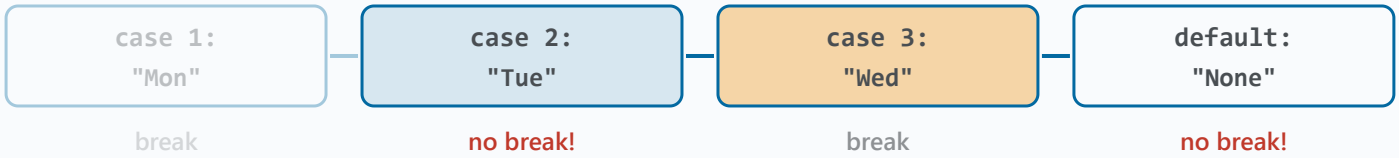
`day = 2`

OUTPUT

```
Tue
```

SWITCH · STEP THROUGH THE CODE

Classic switch: Forgetting `break`



Fall-through: the next case's code runs too, without testing its label → "Wed".

Step 4 / 5

```
1 int day = 2;
2 switch (day) {
3   case 1: System.out.println("Mon"); break;
4   case 2: System.out.println("Tue");
▶   case 3: System.out.println("Wed"); break;
6   default: System.out.println("None");
7 }
```

VARIABLES

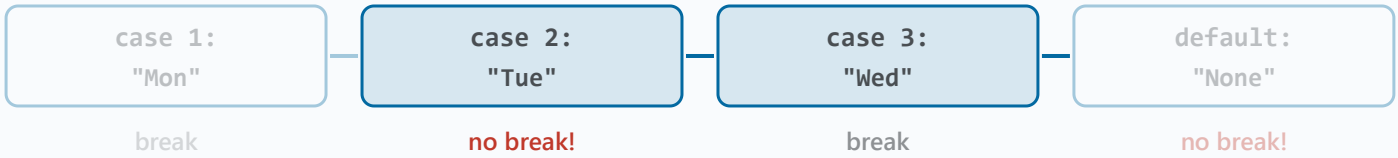
`day = 2`

OUTPUT

```
Tue
Wed
```

SWITCH · STEP THROUGH THE CODE

Classic switch: Forgetting `break`



break leaves the switch; default does not run. The arrow (`->`) form avoids this bug.

Step 5 / 5

```
1 int day = 2;
2 switch (day) {
3   case 1: System.out.println("Mon"); break;
4   case 2: System.out.println("Tue");
▶   case 3: System.out.println("Wed"); break;
6   default: System.out.println("None");
7 }
```

VARIABLES

`day = 2`

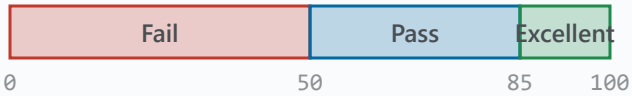
OUTPUT

```
Tue
Wed
```

if-else vs. switch

Criterion	if-else	switch
Range condition	Good fit	Not directly
Many choices on one value	Works	Usually clearer
Complex logic	Good fit	Limited
Each branch yields a value	Needs an assignment	A switch expression can return a value

if-else: range conditions




```
score < 50  ·  score < 85  ·  else
```

each branch covers a range of numbers

switch: discrete values




```
case 1, 2, 3, 4, 5
"Weekday"

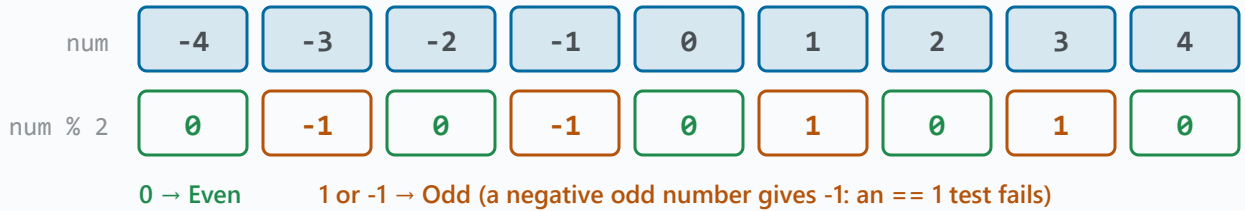
case 6, 7
"Weekend"
```

fixed values of a single variable

EXAMPLE

Odd or Even?

```
int num = 17;
if (num % 2 == 0)
    System.out.println("Even");
else
    System.out.println("Odd");
```



- A number is even if dividing by 2 leaves a remainder of 0.
- This works for negatives too. But in Java -7 % 2 = -1, so testing odd numbers with `num % 2 == 1` is wrong.

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

17

int

Expression

num % 2 == 0

Run 1: num = **17**.

Step 1 / 12

```
▶ int num = in.nextInt();
2 if (num % 2 == 0)
3     System.out.println("Even");
4 else
5     System.out.println("Odd");
```

VARIABLES

num = 17

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

17

int

Expression

```
17 % 2 == 0
1 == 0
```

First the remainder: 17 % 2 = **1**.

Step 2 / 12

```
1 int num = in.nextInt();
▶ if (num % 2 == 0)
3     System.out.println("Even");
4 else
5     System.out.println("Odd");
```

VARIABLES

num = 17

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

17

int

Expression

1 == 0
→ false

1 == 0 → **false**.

Step 3 / 12

```
1 int num = in.nextInt();
▶ if (num % 2 == 0)
3     System.out.println("Even");
4 else
5     System.out.println("Odd");
```

VARIABLES

num = 17

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

17

int

Expression

→ "Odd"

Odd is printed.

Step 4 / 12

```
1 int num = in.nextInt();
2 if (num % 2 == 0)
3     System.out.println("Even");
4 else
▶     System.out.println("Odd");
```

VARIABLES

num = 17

OUTPUT

```
Odd
```

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

-4

int

Expression

```
num % 2 == 0
```

Run 2: num = **-4**.

Step 5 / 12

```
▶ int num = in.nextInt();
2 if (num % 2 == 0)
3     System.out.println("Even");
4 else
5     System.out.println("Odd");
```

VARIABLES

num = −4

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

-4

int

Expression

```
-4 % 2 == 0
0 == 0
```

First the remainder: -4 % 2 = **0**.

Step 6 / 12

```
1 int num = in.nextInt();
▶ if (num % 2 == 0)
3     System.out.println("Even");
4 else
5     System.out.println("Odd");
```

VARIABLES

num = −4

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

-4

int

Expression

0 == 0
→ true

0 == 0 → **true**.

Step 7 / 12

```
1 int num = in.nextInt();
▶ if (num % 2 == 0)
3     System.out.println("Even");
4 else
5     System.out.println("Odd");
```

VARIABLES

num = −4

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

-4

int

Expression

→ "Even"

Even is printed.

Step 8 / 12

```
1 int num = in.nextInt();
2 if (num % 2 == 0)
▶     System.out.println("Even");
4 else
5     System.out.println("Odd");
```

VARIABLES

num = −4

OUTPUT

```
Even
```

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

-7

int

Expression

num % 2 == 0

Run 3: num = **-7**.

Step 9 / 12

```
▶ int num = in.nextInt();
2 if (num % 2 == 0)
3     System.out.println("Even");
4 else
5     System.out.println("Odd");
```

VARIABLES

num = −7

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

-7

int

Expression

```
-7 % 2 == 0
  -1 == 0
```

First the remainder: -7 % 2 = **-1** (in Java the remainder takes the sign of the dividend).

Step 10 / 12

```
1 int num = in.nextInt();
▶ if (num % 2 == 0)
3     System.out.println("Even");
4 else
5     System.out.println("Odd");
```

VARIABLES

num = −7

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

-7

int

Expression

-1 == 0
→ false

-1 == 0 → **false**.

Step 11 / 12

```
1 int num = in.nextInt();
▶ if (num % 2 == 0)
3     System.out.println("Even");
4 else
5     System.out.println("Odd");
```

VARIABLES

num = −7

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Odd or Even? num = 17, −4, −7

Memory (variables)

num

-7

int

Expression

→ "Odd"

Odd is printed. With the test -7 % 2 == 1 the result would have been false!

Step 12 / 12

```
1 int num = in.nextInt();
2 if (num % 2 == 0)
3     System.out.println("Even");
4 else
▶     System.out.println("Odd");
```

VARIABLES

num = −7

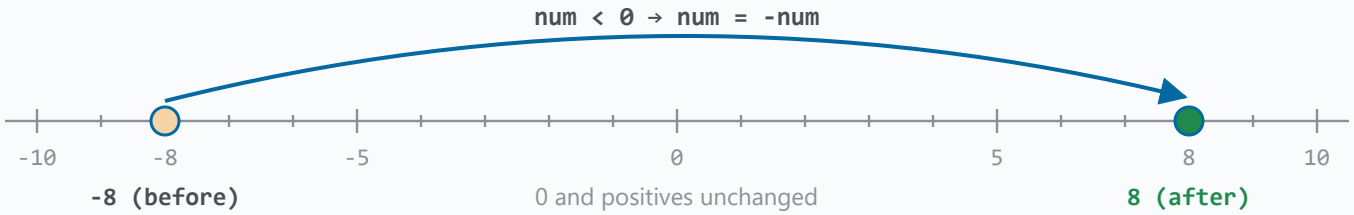
OUTPUT

```
Odd
```

EXAMPLE

Absolute Value

```
int num = -8;
if (num < 0) {
    num = -num;
}
System.out.println(num);
```

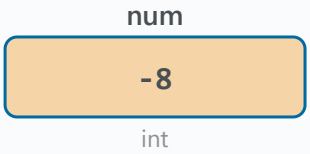


- Zero and positive values stay unchanged.
- Integer.MIN_VALUE is a special overflow case.

EXAMPLE · STEP THROUGH THE CODE

Absolute Value: num = −8, 5, MIN_VALUE

Memory (variables)



Run 1: num = **-8**.

Step 1 / 11

```
▶ int num = in.nextInt();
2 if (num < 0) {
3     num = -num;
4 }
5 System.out.println(num);
```

VARIABLES

num = −8

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Absolute Value: num = −8, 5, MIN_VALUE

Memory (variables)

num

-8

int

Expression

-8 < 0
→ true

Condition: num < 0 → -8 < 0 → **true**.

Step 2 / 11

```
1 int num = in.nextInt();
▶ if (num < 0) {
3     num = -num;
4 }
5 System.out.println(num);
```

VARIABLES

num = −8

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Absolute Value: num = −8, 5, MIN_VALUE

Memory (variables)

num

8

int

num = -num → -(-8) = **8**.

Step 3 / 11

```
1 int num = in.nextInt();
2 if (num < 0) {
▶     num = -num;
4 }
5 System.out.println(num);
```

VARIABLES

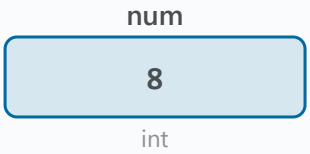
num = 8

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Absolute Value: num = −8, 5, MIN_VALUE

Memory (variables)



8 is printed.

Step 4 / 11

```
1 int num = in.nextInt();
2 if (num < 0) {
3     num = -num;
4 }
▶ System.out.println(num);
```

VARIABLES

`num = 8`

OUTPUT

```
8
```

EXAMPLE · STEP THROUGH THE CODE

Absolute Value: num = −8, 5, MIN_VALUE

Memory (variables)

num

5

int

Run 2: num = **5**.

Step 5 / 11

```
▶ int num = in.nextInt();
2 if (num < 0) {
3     num = -num;
4 }
5 System.out.println(num);
```

VARIABLES

num = 5

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Absolute Value: num = −8, 5, MIN_VALUE

Memory (variables)

num

5

int

Expression

5 < 0
→ false

Condition: num < 0 → 5 < 0 → **false**. The block is skipped; the value stays.

Step 6 / 11

```
1 int num = in.nextInt();
▶ if (num < 0) {
3     num = -num;
4 }
5 System.out.println(num);
```

VARIABLES

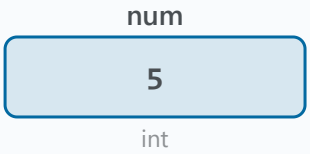
num = 5

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Absolute Value: num = −8, 5, MIN_VALUE

Memory (variables)



5 is printed.

Step 7 / 11

```
1 int num = in.nextInt();
2 if (num < 0) {
3     num = -num;
4 }
▶ System.out.println(num);
```

VARIABLES

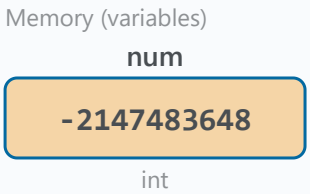
`num = 5`

OUTPUT

```
5
```

EXAMPLE · STEP THROUGH THE CODE

Absolute Value: num = −8, 5, MIN_VALUE



Run 3: num = **-2147483648** (Integer.MIN_VALUE).

Step 8 / 11

```
▶ int num = in.nextInt();
2 if (num < 0) {
3     num = -num;
4 }
5 System.out.println(num);
```

VARIABLES

num = −2147483648

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Absolute Value: num = −8, 5, MIN_VALUE

Memory (variables)

num

-2147483648

int

Expression

-2147483648 < 0

→ true

Condition: num < 0 → -2147483648 < 0 → **true**.

Step 9 / 11

```
1 int num = in.nextInt();
▶ if (num < 0) {
3     num = -num;
4 }
5 System.out.println(num);
```

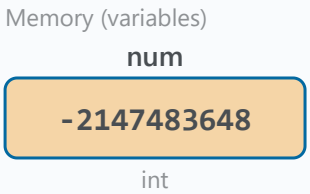
VARIABLES

num = −2147483648

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Absolute Value: num = −8, 5, MIN_VALUE



-(-2147483648) = 2147483648 does not fit in an int → **overflow**: the result is -2147483648 again!

Step 10 / 11

```
1 int num = in.nextInt();
2 if (num < 0) {
      num = -num;
4 }
5 System.out.println(num);
```

VARIABLES

`num = −2147483648`

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Absolute Value: num = −8, 5, MIN_VALUE

Memory (variables)

num

-2147483648

int

-2147483648 is printed. The absolute value came out negative: a special case.

Step 11 / 11

```
1 int num = in.nextInt();
2 if (num < 0) {
3     num = -num;
4 }
▶ System.out.println(num);
```

VARIABLES

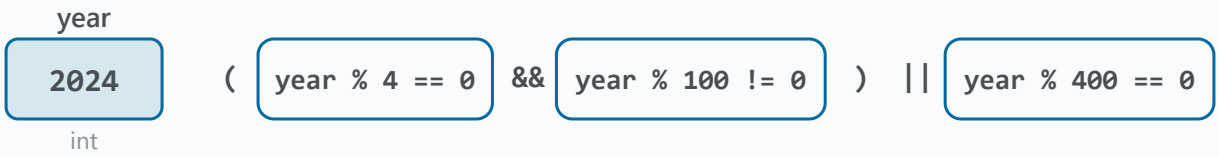
num = **−2147483648**

OUTPUT

```
-2147483648
```

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



Run 1: year = 2024.

Step 1 / 16

```
▶ int year = in.nextInt();
2 boolean leap =
3     (year % 4 == 0 && year % 100 != 0)
4     || year % 400 == 0;
5 if (leap)
6     System.out.println(year + " is a leap
  year");
7 else
8     System.out.println(year + " is not a leap
  year");
```

VARIABLES

year = 2024

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



2024 % 4 = 0 → true; 2024 % 100 = 24 → year % 100 != 0 → true. Parenthesized part: **true**.

Step 2 / 16

```
1  int year = in.nextInt();
▶  boolean leap =
▶      (year % 4 == 0 && year % 100 != 0)
4      || year % 400 == 0;
5  if (leap)
6      System.out.println(year + " is a leap
   year");
7  else
8      System.out.println(year + " is not a leap
   year");
```

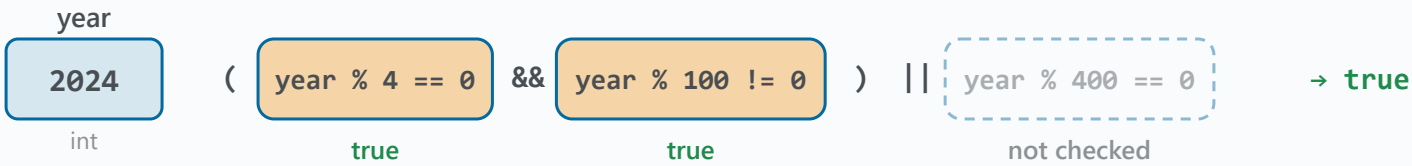
VARIABLES

`year = 2024`

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



Left side true → `||` short-circuits: the 400 rule is not checked. leap = **true**.

Step 3 / 16

```
1  int year = in.nextInt();
▶  boolean leap =
3      (year % 4 == 0 && year % 100 != 0)
▶      || year % 400 == 0;
5  if (leap)
6      System.out.println(year + " is a leap
   year");
7  else
8      System.out.println(year + " is not a leap
   year");
```

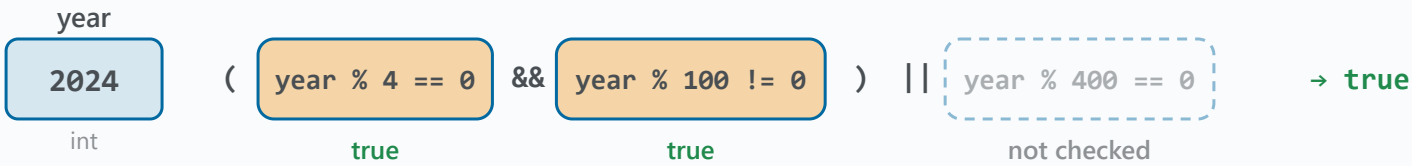
VARIABLES

`year = 2024` `leap = true`

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



2024 is a leap year.

Step 4 / 16

```
1  int year = in.nextInt();
2  boolean leap =
3      (year % 4 == 0 && year % 100 != 0)
4      || year % 400 == 0;
5  if (leap)
▶      System.out.println(year + " is a leap
   year");
7  else
8      System.out.println(year + " is not a leap
   year");
```

VARIABLES

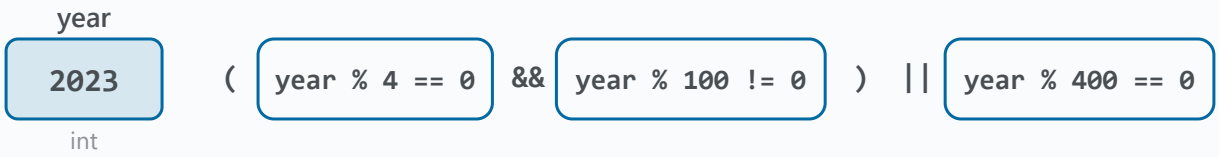
`year = 2024` `leap = true`

OUTPUT

```
2024 is a leap year
```

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



Run 2: year = 2023.

Step 5 / 16

```
▶ int year = in.nextInt();
2 boolean leap =
3     (year % 4 == 0 && year % 100 != 0)
4     || year % 400 == 0;
5 if (leap)
6     System.out.println(year + " is a leap year");
7 else
8     System.out.println(year + " is not a leap year");
```

VARIABLES

year = 2023

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



2023 % 4 = 3 → false; && short-circuits. Parenthesized part: **false**.

Step 6 / 16

```
1  int year = in.nextInt();
▶  boolean leap =
▶      (year % 4 == 0 && year % 100 != 0)
4      || year % 400 == 0;
5  if (leap)
6      System.out.println(year + " is a leap
   year");
7  else
8      System.out.println(year + " is not a leap
   year");
```

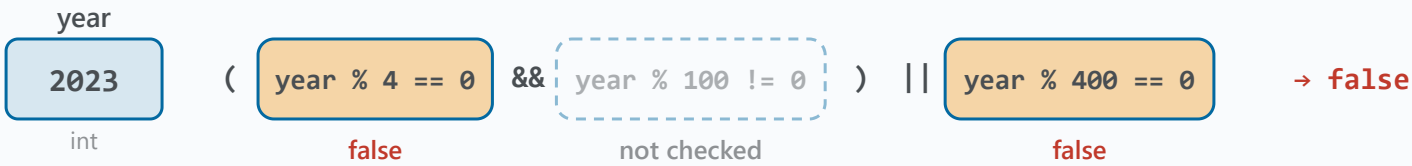
VARIABLES

`year = 2023`

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



Left side false → check the right: 2023 % 400 = 23 → **false**. leap = **false**.

Step 7 / 16

```
1  int year = in.nextInt();
▶  boolean leap =
3      (year % 4 == 0 && year % 100 != 0)
▶      || year % 400 == 0;
5  if (leap)
6      System.out.println(year + " is a leap
   year");
7  else
8      System.out.println(year + " is not a leap
   year");
```

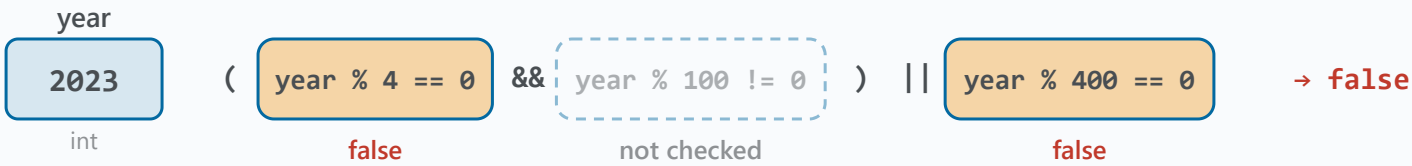
VARIABLES

`year = 2023` `leap = false`

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



2023 is not a leap year.

Step 8 / 16

```
1  int year = in.nextInt();
2  boolean leap =
3      (year % 4 == 0 && year % 100 != 0)
4      || year % 400 == 0;
5  if (leap)
6      System.out.println(year + " is a leap
   year");
7  else
▶      System.out.println(year + " is not a leap
   year");
```

VARIABLES

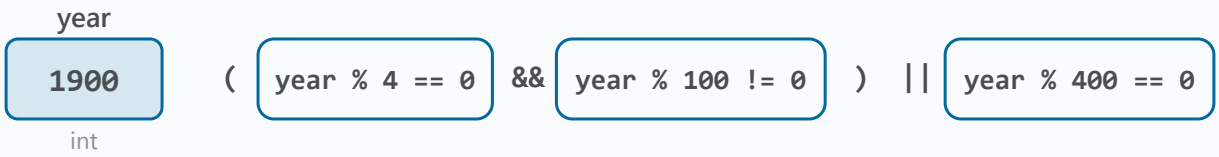
`year = 2023` `leap = false`

OUTPUT

```
2023 is not a leap year
```

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



Run 3: year = 1900.

Step 9 / 16

```
▶ int year = in.nextInt();
2 boolean leap =
3     (year % 4 == 0 && year % 100 != 0)
4     || year % 400 == 0;
5 if (leap)
6     System.out.println(year + " is a leap
  year");
7 else
8     System.out.println(year + " is not a leap
  year");
```

VARIABLES

year = 1900

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



1900 % 4 = 0 → true; 1900 % 100 = 0 → year % 100 != 0 → false. Parenthesized part: **false**.

Step 10 / 16

```
1 int year = in.nextInt();
▶ boolean leap =
▶     (year % 4 == 0 && year % 100 != 0)
4     || year % 400 == 0;
5 if (leap)
6     System.out.println(year + " is a leap
  year");
7 else
8     System.out.println(year + " is not a leap
  year");
```

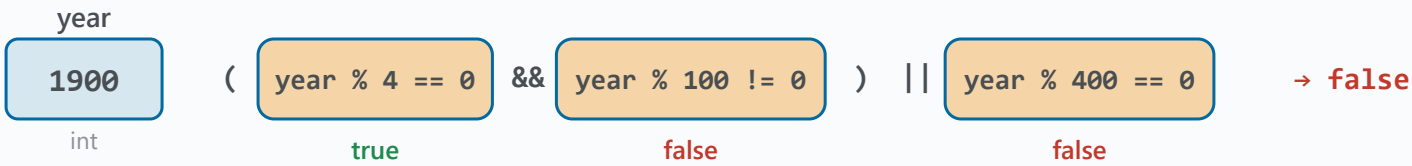
VARIABLES

`year = 1900`

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



Left side false → check the right: 1900 % 400 = 300 → **false**. leap = **false**.

Step 11 / 16

```
1  int year = in.nextInt();
▶  boolean leap =
3      (year % 4 == 0 && year % 100 != 0)
▶      || year % 400 == 0;
5  if (leap)
6      System.out.println(year + " is a leap
   year");
7  else
8      System.out.println(year + " is not a leap
   year");
```

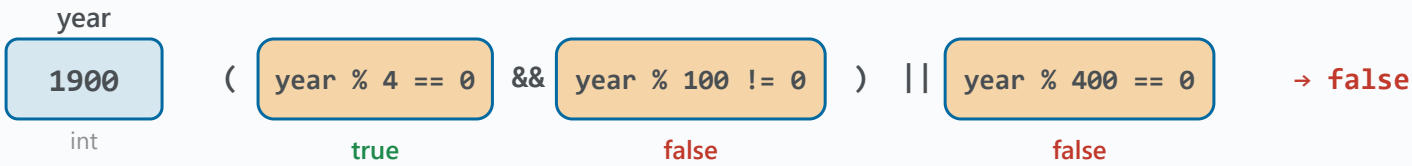
VARIABLES

year = **1900** leap = **false**

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



1900 is not a leap year. Century years are leap years only if divisible by 400.

Step 12 / 16

```
1 int year = in.nextInt();
2 boolean leap =
3     (year % 4 == 0 && year % 100 != 0)
4     || year % 400 == 0;
5 if (leap)
6     System.out.println(year + " is a leap
  year");
7 else
▶     System.out.println(year + " is not a leap
  year");
```

VARIABLES

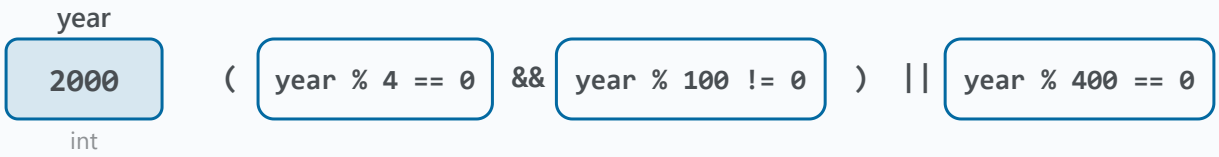
`year = 1900` `leap = false`

OUTPUT

```
1900 is not a leap year
```

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



Run 4: year = 2000.

Step 13 / 16

```
▶ int year = in.nextInt();
2 boolean leap =
3     (year % 4 == 0 && year % 100 != 0)
4     || year % 400 == 0;
5 if (leap)
6     System.out.println(year + " is a leap
  year");
7 else
8     System.out.println(year + " is not a leap
  year");
```

VARIABLES

year = 2000

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



2000 % 4 = 0 → true; 2000 % 100 = 0 → year % 100 != 0 → false. Parenthesized part: **false**.

Step 14 / 16

```
1  int year = in.nextInt();
▶  boolean leap =
▶      (year % 4 == 0 && year % 100 != 0)
4      || year % 400 == 0;
5  if (leap)
6      System.out.println(year + " is a leap
   year");
7  else
8      System.out.println(year + " is not a leap
   year");
```

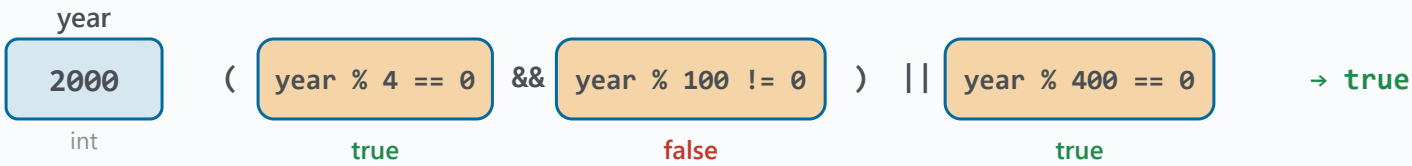
VARIABLES

`year = 2000`

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



Left side false → check the right: 2000 % 400 = 0 → **true**. leap = **true**.

Step 15 / 16

```
1 int year = in.nextInt();
▶ boolean leap =
3     (year % 4 == 0 && year % 100 != 0)
▶     || year % 400 == 0;
5 if (leap)
6     System.out.println(year + " is a leap
  year");
7 else
8     System.out.println(year + " is not a leap
  year");
```

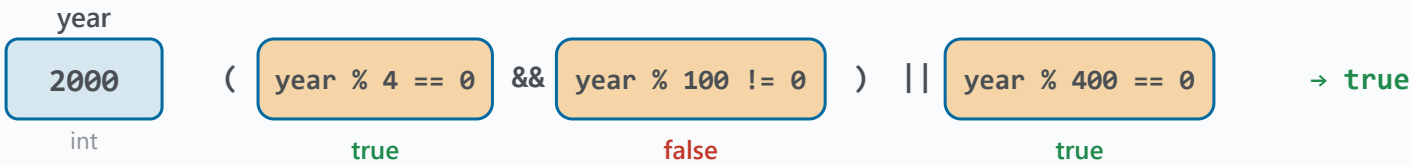
VARIABLES

year = **2000** leap = **true**

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Leap Year: 2024, 2023, 1900, 2000



2000 is a leap year.

Step 16 / 16

```
1  int year = in.nextInt();
2  boolean leap =
3      (year % 4 == 0 && year % 100 != 0)
4      || year % 400 == 0;
5  if (leap)
▶      System.out.println(year + " is a leap
   year");
7  else
8      System.out.println(year + " is not a leap
   year");
```

VARIABLES

`year = 2000` `leap = true`

OUTPUT

```
2000 is a leap year
```

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (-3.0)x + 2.0$

Run 1: a = 1.0, b = -3.0, c = 2.0.

Step 1 / 13

```
▶ double a = in.nextDouble();
▶ double b = in.nextDouble();
▶ double c = in.nextDouble();
4 double d = b * b - 4 * a * c;
5 if (d > 0) {
6   double x1 = (-b + Math.sqrt(d)) / (2 * a);
7   double x2 = (-b - Math.sqrt(d)) / (2 * a);
8   System.out.println(x1 + " and " + x2);
9 } else if (d == 0) {
10  System.out.println(-b / (2 * a));
11 } else {
12  System.out.println("No real roots");
13 }
```

VARIABLES

a = 1.0 | b = -3.0 | c = 2.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (-3.0)x + 2.0$

$\Delta = 1.0$

$d = \Delta = b^2 - 4ac = 9.0 - 8.0 = \mathbf{1.0}$.

Step 2 / 13

```
1  double a = in.nextDouble();
2  double b = in.nextDouble();
3  double c = in.nextDouble();
▶ double d = b * b - 4 * a * c;
5  if (d > 0) {
6    double x1 = (-b + Math.sqrt(d)) / (2 * a);
7    double x2 = (-b - Math.sqrt(d)) / (2 * a);
8    System.out.println(x1 + " and " + x2);
9  } else if (d == 0) {
10   System.out.println(-b / (2 * a));
11 } else {
12   System.out.println("No real roots");
13 }
```

VARIABLES

a = 1.0 | b = -3.0 | c = 2.0 | d = 1.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (-3.0)x + 2.0$

Δ > 0 → two roots

1.0 > 0 → true: two distinct real roots.

Step 3 / 13

```
1  double a = in.nextDouble();
2  double b = in.nextDouble();
3  double c = in.nextDouble();
4  double d = b * b - 4 * a * c;
▶  if (d > 0) {
6    double x1 = (-b + Math.sqrt(d)) / (2 * a);
7    double x2 = (-b - Math.sqrt(d)) / (2 * a);
8    System.out.println(x1 + " and " + x2);
9  } else if (d == 0) {
10   System.out.println(-b / (2 * a));
11 } else {
12   System.out.println("No real roots");
13 }
```

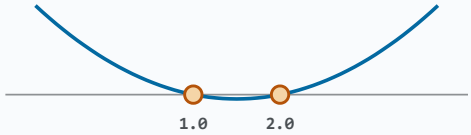
VARIABLES

a = 1.0 | b = -3.0 | c = 2.0 | d = 1.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (-3.0)x + 2.0$

two roots

√Δ = 1.0; x1 = (3.0 + 1.0) / 2.0 = 2.0, x2 = 1.0.

Step 4 / 13

```
1  double a = in.nextDouble();
2  double b = in.nextDouble();
3  double c = in.nextDouble();
4  double d = b * b - 4 * a * c;
5  if (d > 0) {
▶    double x1 = (-b + Math.sqrt(d)) / (2 * a);
▶    double x2 = (-b - Math.sqrt(d)) / (2 * a);
8    System.out.println(x1 + " and " + x2);
9  } else if (d == 0) {
10   System.out.println(-b / (2 * a));
11 } else {
12   System.out.println("No real roots");
13 }
```

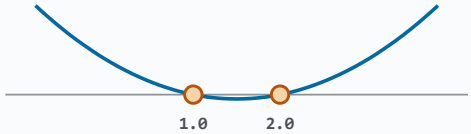
VARIABLES

a = 1.0 | b = -3.0 | c = 2.0 | d = 1.0 | x1 = 2.0 | x2 = 1.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (-3.0)x + 2.0$

two roots

2.0 and 1.0: the parabola crosses the x-axis at two points.

Step 5 / 13

```
1  double a = in.nextDouble();
2  double b = in.nextDouble();
3  double c = in.nextDouble();
4  double d = b * b - 4 * a * c;
5  if (d > 0) {
6    double x1 = (-b + Math.sqrt(d)) / (2 * a);
7    double x2 = (-b - Math.sqrt(d)) / (2 * a);
▶    System.out.println(x1 + " and " + x2);
9  } else if (d == 0) {
10   System.out.println(-b / (2 * a));
11 } else {
12   System.out.println("No real roots");
13 }
```

VARIABLES

a = 1.0 | b = -3.0 | c = 2.0 | d = 1.0 | x1 = 2.0 | x2 = 1.0

OUTPUT

```
2.0 and 1.0
```

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (-2.0)x + 1.0$

Run 2: a = 1.0, b = -2.0, c = 1.0.

Step 6 / 13

```
▶ double a = in.nextDouble();
▶ double b = in.nextDouble();
▶ double c = in.nextDouble();
4 double d = b * b - 4 * a * c;
5 if (d > 0) {
6   double x1 = (-b + Math.sqrt(d)) / (2 * a);
7   double x2 = (-b - Math.sqrt(d)) / (2 * a);
8   System.out.println(x1 + " and " + x2);
9 } else if (d == 0) {
10  System.out.println(-b / (2 * a));
11 } else {
12  System.out.println("No real roots");
13 }
```

VARIABLES

a = 1.0 | b = -2.0 | c = 1.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (-2.0)x + 1.0$

Δ = 0.0

d = Δ = $b^2 - 4ac = 4.0 - 4.0 =$ **0.0**.

Step 7 / 13

```
1  double a = in.nextDouble();
2  double b = in.nextDouble();
3  double c = in.nextDouble();
▶ double d = b * b - 4 * a * c;
5  if (d > 0) {
6    double x1 = (-b + Math.sqrt(d)) / (2 * a);
7    double x2 = (-b - Math.sqrt(d)) / (2 * a);
8    System.out.println(x1 + " and " + x2);
9  } else if (d == 0) {
10   System.out.println(-b / (2 * a));
11 } else {
12   System.out.println("No real roots");
13 }
```

VARIABLES

a = 1.0 | b = -2.0 | c = 1.0 | d = 0.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (-2.0)x + 1.0$

Δ = 0 → one root

Δ > 0 is false; Δ == 0 → **true**: one (double) root.

Step 8 / 13

```
1  double a = in.nextDouble();
2  double b = in.nextDouble();
3  double c = in.nextDouble();
4  double d = b * b - 4 * a * c;
▶  if (d > 0) {
6    double x1 = (-b + Math.sqrt(d)) / (2 * a);
7    double x2 = (-b - Math.sqrt(d)) / (2 * a);
8    System.out.println(x1 + " and " + x2);
▶  } else if (d == 0) {
10   System.out.println(-b / (2 * a));
11 } else {
12   System.out.println("No real roots");
13 }
```

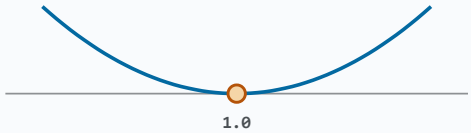
VARIABLES

a = 1.0 | b = -2.0 | c = 1.0 | d = 0.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



1.0x² + (-2.0)x + 1.0

one root

x = −b / 2a = 2.0 / 2.0 = **1.0**: the parabola touches the axis.

Step 9 / 13

```
1  double a = in.nextDouble();
2  double b = in.nextDouble();
3  double c = in.nextDouble();
4  double d = b * b - 4 * a * c;
5  if (d > 0) {
6    double x1 = (-b + Math.sqrt(d)) / (2 * a);
7    double x2 = (-b - Math.sqrt(d)) / (2 * a);
8    System.out.println(x1 + " and " + x2);
9  } else if (d == 0) {
▶    System.out.println(-b / (2 * a));
11 } else {
12   System.out.println("No real roots");
13 }
```

VARIABLES

a = 1.0 | b = -2.0 | c = 1.0 | d = 0.0

OUTPUT

```
1.0
```

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (0.0)x + 1.0$

Run 3: a = 1.0, b = 0.0, c = 1.0.

Step 10 / 13

```
▶ double a = in.nextDouble();
▶ double b = in.nextDouble();
▶ double c = in.nextDouble();
4 double d = b * b - 4 * a * c;
5 if (d > 0) {
6   double x1 = (-b + Math.sqrt(d)) / (2 * a);
7   double x2 = (-b - Math.sqrt(d)) / (2 * a);
8   System.out.println(x1 + " and " + x2);
9 } else if (d == 0) {
10  System.out.println(-b / (2 * a));
11 } else {
12  System.out.println("No real roots");
13 }
```

VARIABLES

a = 1.0 | b = 0.0 | c = 1.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (0.0)x + 1.0$

Δ = -4.0

d = Δ = b^2 − 4ac = 0.0 − 4.0 = **-4.0**.

Step 11 / 13

```
1  double a = in.nextDouble();
2  double b = in.nextDouble();
3  double c = in.nextDouble();
▶ 4  double d = b * b - 4 * a * c;
5  if (d > 0) {
6    double x1 = (-b + Math.sqrt(d)) / (2 * a);
7    double x2 = (-b - Math.sqrt(d)) / (2 * a);
8    System.out.println(x1 + " and " + x2);
9  } else if (d == 0) {
10   System.out.println(-b / (2 * a));
11 } else {
12   System.out.println("No real roots");
13 }
```

VARIABLES

a = 1.0 | b = 0.0 | c = 1.0 | d = -4.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (0.0)x + 1.0$

$\Delta < 0$

-4.0 > 0 false, -4.0 == 0 false → else.

Step 12 / 13

```
1  double a = in.nextDouble();
2  double b = in.nextDouble();
3  double c = in.nextDouble();
4  double d = b * b - 4 * a * c;
▶  if (d > 0) {
6    double x1 = (-b + Math.sqrt(d)) / (2 * a);
7    double x2 = (-b - Math.sqrt(d)) / (2 * a);
8    System.out.println(x1 + " and " + x2);
▶  } else if (d == 0) {
10   System.out.println(-b / (2 * a));
11 } else {
12   System.out.println("No real roots");
13 }
```

VARIABLES

a = 1.0 | b = 0.0 | c = 1.0 | d = -4.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

Quadratic Equation: Δ > 0, Δ = 0, Δ < 0



$1.0x^2 + (0.0)x + 1.0$

no roots

No real roots: the parabola never meets the axis. Math.sqrt of a negative would give NaN; the else prevents that.

Step 13 / 13

```
1  double a = in.nextDouble();
2  double b = in.nextDouble();
3  double c = in.nextDouble();
4  double d = b * b - 4 * a * c;
5  if (d > 0) {
6    double x1 = (-b + Math.sqrt(d)) / (2 * a);
7    double x2 = (-b - Math.sqrt(d)) / (2 * a);
8    System.out.println(x1 + " and " + x2);
9  } else if (d == 0) {
10   System.out.println(-b / (2 * a));
11 } else {
▶    System.out.println("No real roots");
13 }
```

VARIABLES

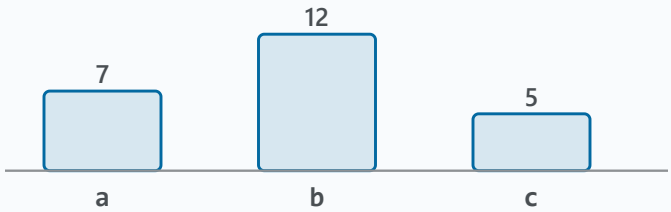
a = 1.0 | b = 0.0 | c = 1.0 | d = -4.0

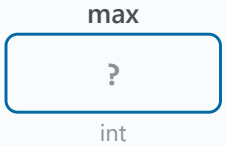
OUTPUT

```
No real roots
```

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





Run 1: 7, 12, 5.

Step 1 / 17

```
▶ int a = in.nextInt();
▶ int b = in.nextInt();
▶ int c = in.nextInt();
4 int max = a;
5 if (b > max)
6     max = b;
7 if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

VARIABLES

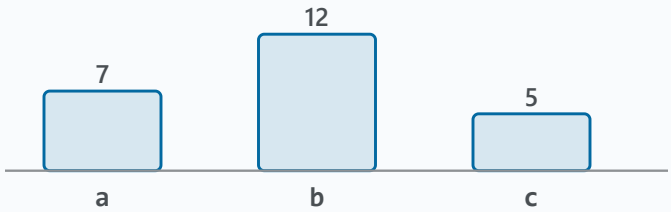
a = 7 | b = 12 | c = 5

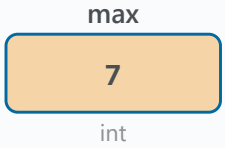
OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





Candidate: max = a = 7.

Step 2 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
▶ int max = a;
5 if (b > max)
6     max = b;
7 if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

VARIABLES

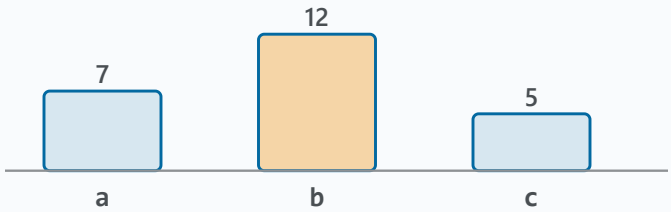
a = 7 | b = 12 | c = 5 | max = 7

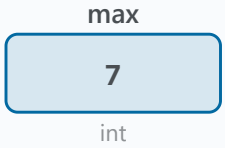
OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





b > max → 12 > 7 → **true**.

Step 3 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
4 int max = a;
▶ if (b > max)
6     max = b;
7 if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

VARIABLES

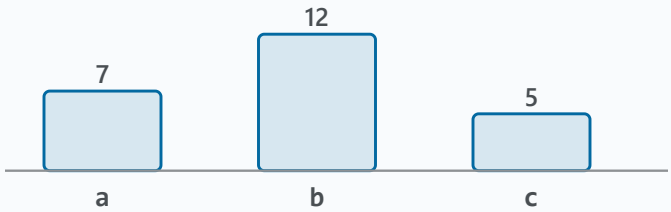
`a = 7` `b = 12` `c = 5` `max = 7`

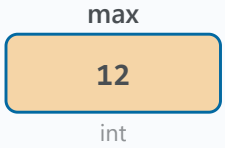
OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





New candidate: max = b = 12.

Step 4 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
4 int max = a;
5 if (b > max)
▶     max = b;
7 if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

VARIABLES

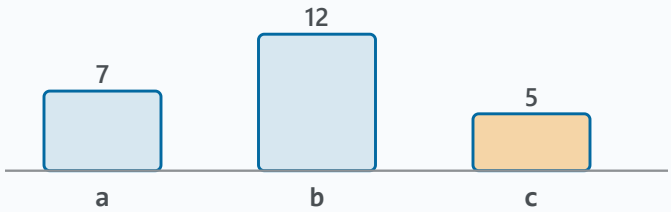
a = 7 | b = 12 | c = 5 | max = 12

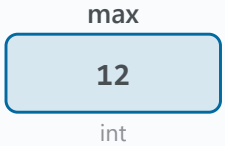
OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





c > max → 5 > 12 → **false**; the candidate stays.

Step 5 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
4 int max = a;
5 if (b > max)
6     max = b;
▶ if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

VARIABLES

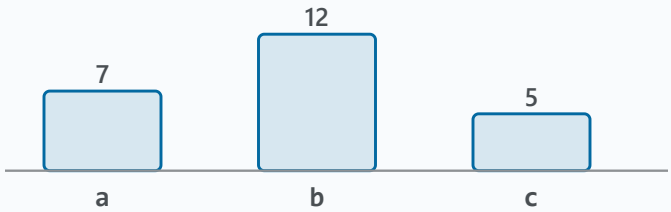
`a = 7` `b = 12` `c = 5` `max = 12`

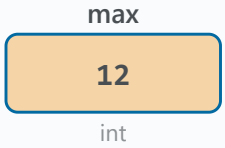
OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





Max: 12.

Step 6 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
4 int max = a;
5 if (b > max)
6     max = b;
7 if (c > max)
8     max = c;
▶ System.out.println("Max: " + max);
```

VARIABLES

a = 7 | b = 12 | c = 5 | max = 12

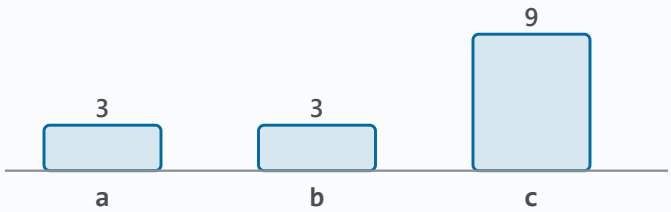
OUTPUT

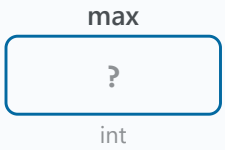
```
Max: 12
```

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





Run 2: 3, 3, 9.

Step 7 / 17

```
▶ int a = in.nextInt();
▶ int b = in.nextInt();
▶ int c = in.nextInt();
4 int max = a;
5 if (b > max)
6     max = b;
7 if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

VARIABLES

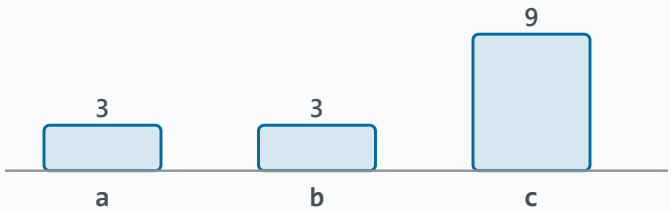
a = 3 b = 3 c = 9

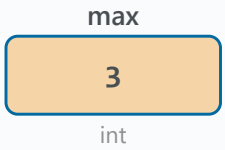
OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





Candidate: max = a = 3.

Step 8 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
▶ int max = a;
5 if (b > max)
6     max = b;
7 if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

VARIABLES

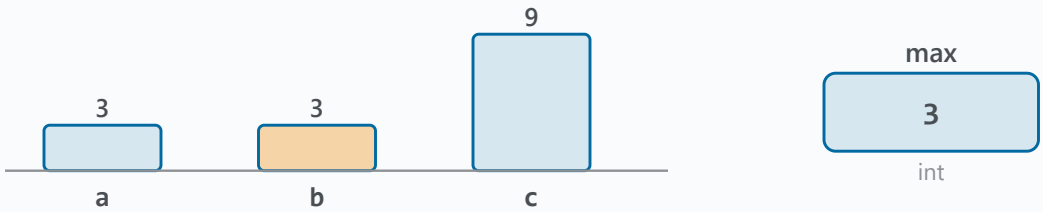
a = 3 | b = 3 | c = 9 | max = 3

OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs



b > max → 3 > 3 → **false**; the candidate stays. An equal value does not replace it.

Step 9 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
4 int max = a;
▶ if (b > max)
6     max = b;
7 if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

VARIABLES

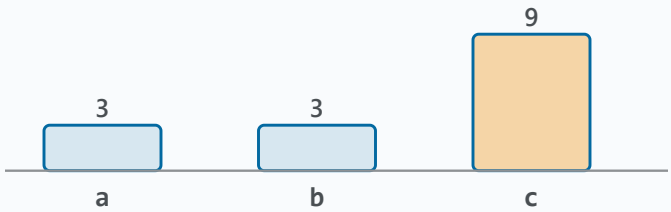
a = 3 | b = 3 | c = 9 | max = 3

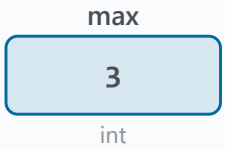
OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





c > max → 9 > 3 → **true**.

Step 10 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
4 int max = a;
5 if (b > max)
6     max = b;
▶ if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

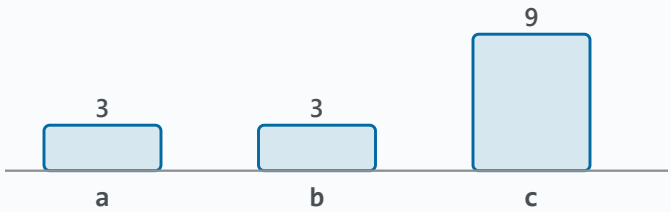
VARIABLES

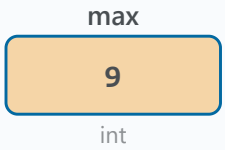
`a = 3` `b = 3` `c = 9` `max = 3`

OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

Largest of Three Numbers: 3 different inputs





New candidate: max = c = 9.

Step 11 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
4 int max = a;
5 if (b > max)
6     max = b;
7 if (c > max)
▶     max = c;
9 System.out.println("Max: " + max);
```

VARIABLES

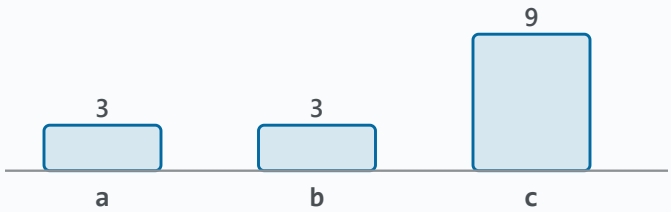
`a = 3` `b = 3` `c = 9` `max = 9`

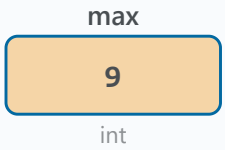
OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





Max: 9.

Step 12 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
4 int max = a;
5 if (b > max)
6     max = b;
7 if (c > max)
8     max = c;
▶ System.out.println("Max: " + max);
```

VARIABLES

a = 3 | b = 3 | c = 9 | max = 9

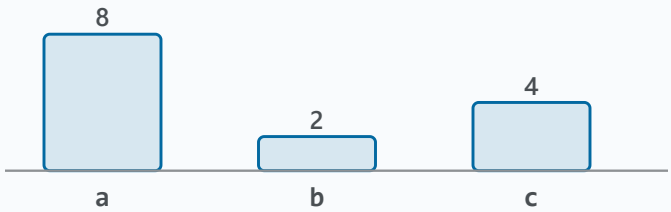
OUTPUT

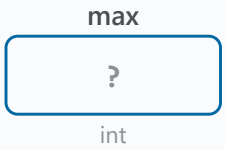
```
Max: 9
```

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





Run 3: 8, 2, 4.

Step 13 / 17

```
int a = in.nextInt();
int b = in.nextInt();
int c = in.nextInt();
int max = a;
if (b > max)
    max = b;
if (c > max)
    max = c;
System.out.println("Max: " + max);
```

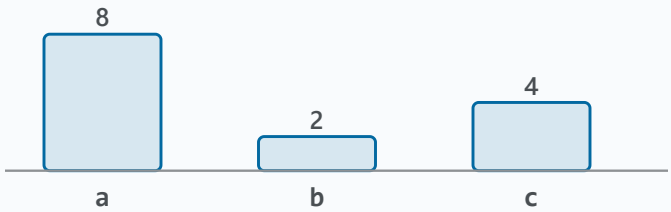
VARIABLES

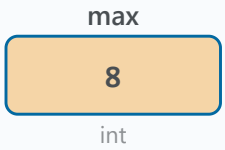
a = 8 b = 2 c = 4

OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

Largest of Three Numbers: 3 different inputs





Candidate: max = a = 8.

Step 14 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
▶ int max = a;
5 if (b > max)
6     max = b;
7 if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

VARIABLES

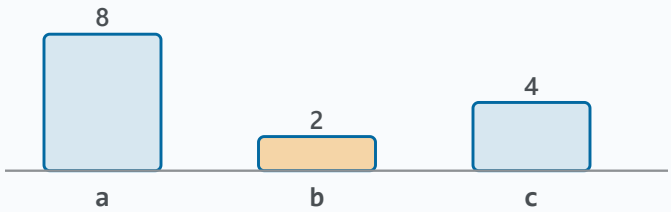
`a = 8` `b = 2` `c = 4` `max = 8`

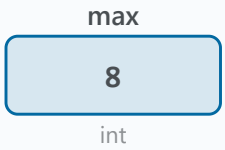
OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





b > max → 2 > 8 → **false**; the candidate stays.

Step 15 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
4 int max = a;
▶ if (b > max)
6     max = b;
7 if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

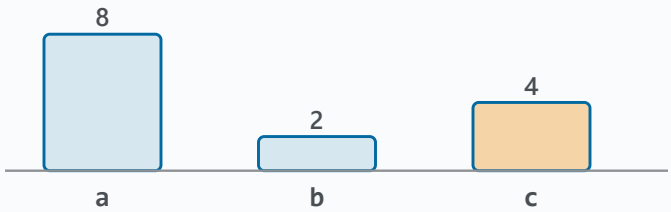
VARIABLES

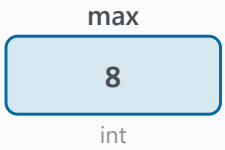
a = 8 | b = 2 | c = 4 | max = 8

OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

Largest of Three Numbers: 3 different inputs





c > max → 4 > 8 → **false**; the candidate stays.

Step 16 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
4 int max = a;
5 if (b > max)
6     max = b;
▶ if (c > max)
8     max = c;
9 System.out.println("Max: " + max);
```

VARIABLES

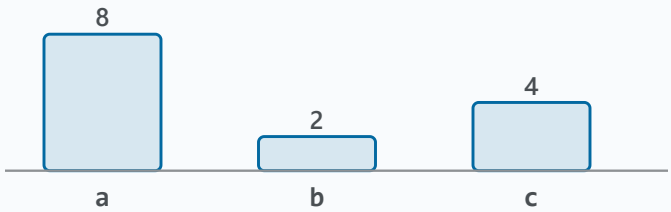
`a = 8` `b = 2` `c = 4` `max = 8`

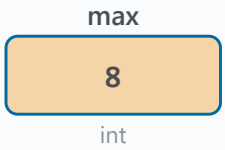
OUTPUT

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

Largest of Three Numbers: 3 different inputs





Max: 8.

Step 17 / 17

```
1 int a = in.nextInt();
2 int b = in.nextInt();
3 int c = in.nextInt();
4 int max = a;
5 if (b > max)
6     max = b;
7 if (c > max)
8     max = c;
▶ System.out.println("Max: " + max);
```

VARIABLES

a = 8 | b = 2 | c = 4 | max = 8

OUTPUT

```
Max: 8
```

Shortcut: `Math.max(a, Math.max(b, c))`. The "candidate + compare" pattern generalizes with a loop.

EXAMPLE · STEP THROUGH THE CODE

A Calculator with switch: *, /0, %

8.0 * 2.5 | "+" | "-" | "*" | "/" | default

Run 1: the user enters `8.0 * 2.5`; next() reads the operator as a string.

Step 1 / 10

```
▶ double x = in.nextDouble();
▶ String op = in.next();
▶ double y = in.nextDouble();
4 switch (op) {
5   case "+" -> System.out.println(x + y);
6   case "-" -> System.out.println(x - y);
7   case "*" -> System.out.println(x * y);
8   case "/" -> {
9     if (y == 0) System.out.println("Error!");
10    else System.out.println(x / y);
11  }
12  default -> System.out.println("Invalid");
13 }
```

VARIABLES

x = 8.0 | op = "*" | y = 2.5

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

A Calculator with switch: *, /0, %



switch (op) → "*": matches **case "*"**.

Step 2 / 10

```
1  double x = in.nextDouble();
2  String op = in.next();
3  double y = in.nextDouble();
▶  switch (op) {
5    case "+" -> System.out.println(x + y);
6    case "-" -> System.out.println(x - y);
7    case "*" -> System.out.println(x * y);
8    case "/" -> {
9      if (y == 0) System.out.println("Error!");
10     else System.out.println(x / y);
11   }
12   default -> System.out.println("Invalid");
13 }
```

VARIABLES

x = 8.0 | op = "*" | y = 2.5

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

A Calculator with switch: *, /0, %



8.0 * 2.5 = **20.0** is printed. No other case runs.

Step 3 / 10

```
1  double x = in.nextDouble();
2  String op = in.next();
3  double y = in.nextDouble();
4  switch (op) {
5    case "+" -> System.out.println(x + y);
6    case "-" -> System.out.println(x - y);
▶    case "*" -> System.out.println(x * y);
8    case "/" -> {
9      if (y == 0) System.out.println("Error!");
10     else System.out.println(x / y);
11   }
12   default -> System.out.println("Invalid");
13 }
```

VARIABLES

x = 8.0 | op = "*" | y = 2.5

OUTPUT

```
20.0
```

EXAMPLE · STEP THROUGH THE CODE

A Calculator with switch: *, /0, %

7.0 / 0.0 | "+" | "-" | "*" | "/" | default

Run 2: the user enters `7.0 / 0.0`; next() reads the operator as a string.

Step 4 / 10

```
▶ double x = in.nextDouble();
▶ String op = in.next();
▶ double y = in.nextDouble();
4 switch (op) {
5   case "+" -> System.out.println(x + y);
6   case "-" -> System.out.println(x - y);
7   case "*" -> System.out.println(x * y);
8   case "/" -> {
9     if (y == 0) System.out.println("Error!");
10    else System.out.println(x / y);
11  }
12  default -> System.out.println("Invalid");
13 }
```

VARIABLES

x = 7.0 | op = "/" | y = 0.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

A Calculator with switch: *, /0, %



switch (op) → "/": matches **case "/"**.

Step 5 / 10

```
1  double x = in.nextDouble();
2  String op = in.next();
3  double y = in.nextDouble();
▶  switch (op) {
5    case "+" -> System.out.println(x + y);
6    case "-" -> System.out.println(x - y);
7    case "*" -> System.out.println(x * y);
8    case "/" -> {
9      if (y == 0) System.out.println("Error!");
10     else System.out.println(x / y);
11   }
12   default -> System.out.println("Invalid");
13 }
```

VARIABLES

x = **7.0** op = **"/"** y = **0.0**

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

A Calculator with switch: *, /0, %



Check before dividing: y == 0 → **true**. (With doubles 7.0 / 0 does not throw; it gives Infinity, so we check.)

Step 6 / 10

```
1  double x = in.nextDouble();
2  String op = in.next();
3  double y = in.nextDouble();
4  switch (op) {
5    case "+" -> System.out.println(x + y);
6    case "-" -> System.out.println(x - y);
7    case "*" -> System.out.println(x * y);
8    case "/" -> {
▶      if (y == 0) System.out.println("Error!");
10     else System.out.println(x / y);
11   }
12   default -> System.out.println("Invalid");
13 }
```

VARIABLES

`x = 7.0` `op = "/"` `y = 0.0`

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

A Calculator with switch: *, /0, %



Error! is printed.

Step 7 / 10

```
1  double x = in.nextDouble();
2  String op = in.next();
3  double y = in.nextDouble();
4  switch (op) {
5    case "+" -> System.out.println(x + y);
6    case "-" -> System.out.println(x - y);
7    case "*" -> System.out.println(x * y);
8    case "/" -> {
▶      if (y == 0) System.out.println("Error!");
10     else System.out.println(x / y);
11   }
12   default -> System.out.println("Invalid");
13 }
```

VARIABLES

x = 7.0 | op = "/" | y = 0.0

OUTPUT

```
Error!
```

EXAMPLE · STEP THROUGH THE CODE

A Calculator with switch: *, /0, %



Run 3: the user enters `5.0 % 3.0`; next() reads the operator as a string.

Step 8 / 10

```
▶ double x = in.nextDouble();
▶ String op = in.next();
▶ double y = in.nextDouble();
4 switch (op) {
5   case "+" -> System.out.println(x + y);
6   case "-" -> System.out.println(x - y);
7   case "*" -> System.out.println(x * y);
8   case "/" -> {
9     if (y == 0) System.out.println("Error!");
10    else System.out.println(x / y);
11  }
12  default -> System.out.println("Invalid");
13 }
```

VARIABLES

`x = 5.0` `op = "%"` `y = 3.0`

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

A Calculator with switch: *, /0, %



switch (op) → "%": no case matches → **default**.

Step 9 / 10

```
1  double x = in.nextDouble();
2  String op = in.next();
3  double y = in.nextDouble();
▶  switch (op) {
5    case "+" -> System.out.println(x + y);
6    case "-" -> System.out.println(x - y);
7    case "*" -> System.out.println(x * y);
8    case "/" -> {
9      if (y == 0) System.out.println("Error!");
10     else System.out.println(x / y);
11   }
12   default -> System.out.println("Invalid");
13 }
```

VARIABLES

x = 5.0 | op = "%" | y = 3.0

OUTPUT

EXAMPLE · STEP THROUGH THE CODE

A Calculator with switch: *, /0, %



Invalid: this calculator has no %; default catches unexpected input.

Step 10 / 10

```
1  double x = in.nextDouble();
2  String op = in.next();
3  double y = in.nextDouble();
4  switch (op) {
5    case "+" -> System.out.println(x + y);
6    case "-" -> System.out.println(x - y);
7    case "*" -> System.out.println(x * y);
8    case "/" -> {
9      if (y == 0) System.out.println("Error!");
10     else System.out.println(x / y);
11   }
▶    default -> System.out.println("Invalid");
13 }
```

VARIABLES

x = 5.0 | op = "%" | y = 3.0

OUTPUT

```
Invalid
```

SUMMARY

Chapter Summary

- Sequence runs statements in the order written; selection picks a path based on a condition.
- A condition must always produce a boolean value.
- In an else-if chain, conditions are tried top to bottom; the first true branch runs.
- Use if-else for ranges and complex conditions, switch for discrete values of one expression.

SUMMARY

Review Questions

1. How do independent if statements differ from an else-if chain?
2. Why are the thresholds written from highest to lowest when converting scores to letter grades?
3. When can && replace a nested decision?
4. Why is default useful in a switch?
5. Which condition tells you that a number is even?

Exercises

1. Print whether an input integer is positive, negative or zero.
2. Compute body mass index (weight / height^2) and print underweight / normal / overweight.
3. Do three side lengths form a triangle? If so, is it equilateral, isosceles or scalene?
4. Map a day number 1–7 to a day name with switch; warn on invalid input.
5. Find the smaller of two numbers with `?:`, then extend it to three numbers.
6. Add an "outside 0–100" check to the letter-grade program. Where in the chain should it go?

SUMMARY

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: **The if-then and if-then-else Statements**, **The switch Statement**.
- Oracle, Java Language Specification SE 21: **Chapter 14, Blocks, Statements, and Patterns**.
- 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 7: Conditionals

Previous: Chapter 6: Variables, Data Types and Operators · Next: Chapter 8: Loops