

2.3 if Statements

Self-Study Guide · About 45 minutes · With answer key

By the end you will be able to

- Explain how an if statement changes the usual top-to-bottom order in which statements run.
- Trace a one-way if statement and say whether its body runs for a given value.
- Write the condition and body of an if statement correctly, including the braces around a body of several statements.
- Trace an if-else statement and explain why exactly one of its two bodies runs.
- Decide whether a situation calls for a one-way if or a two-way if-else, and predict how separate if statements can affect each other.

What to remember from earlier

- A variable holds one value at a time and must be given a value before it is used. ([Topic 1.4: Assignment Statements and Input, 1.4.4 Assign Before You Use](#))
- Compound operators such as += and -= update a variable from its current value. ([Topic 1.6: Compound Assignment Operators, 1.6.1 Adding to a Variable in One Step](#))
- equals compares the characters of two strings, and uppercase and lowercase letters count as different. ([Topic 1.15: String Manipulation, 1.15.9 Comparing Strings with equals and compareTo](#))
- Selection is a step in an algorithm where the next action depends on a true or false decision. ([Topic 2.1: Algorithms with Selection and Repetition, 2.1.2 Selection: Deciding with a True-or-False Question](#))
- The relational operators ==, !=, <, >, <=, and >= compare two values and give a boolean. ([Topic 2.2: Boolean Expressions](#))

2.3.1 Running Code Only When a Condition Is True

Every program you wrote in Unit 1 did the same thing every time it ran. Java started at the first statement in `main`, ran it, moved to the next one, and kept going until the last one. No statement was ever skipped. Running statements one after another, in the order they are written, is called **sequential execution**.

Sequential execution is a problem as soon as a program has to react to its data. Here is a weather app that is supposed to tell you to wear a jacket only when it is cold:

```
1 int temperature = 68;  
2 System.out.println("Wear a jacket.");  
3 System.out.println("Have a good day.");
```

```
Wear a jacket.  
Have a good day.
```

It is 68 degrees, and the program still says to wear a jacket. Line 2 has no way to know it should be skipped: Java runs every line, in order, no matter what `temperature` holds.

In Topic 2.1 you described algorithms with **selection**: a step where what happens next depends on a true or false decision, such as "If it is below 50 degrees, wear a jacket." In Topic 2.2 you learned to write that decision in Java as a Boolean expression, such as `temperature < 50`, which works out to `true` or `false`. This topic connects the two. You will make Java use the value of a Boolean expression to decide whether a piece of code runs.

The order in which a program's statements actually run is called its **flow of control**. In a sequential program, the flow of control goes straight down the page. A **selection statement** is a statement that changes that flow: depending on a Boolean expression, it lets the program run some statements and skip others. Java's main selection statement is the **if statement**.

Think of a sign at a fork in a hiking trail: "Scenic overlook, 200 steps. Skip it if you are short on time." Every hiker reaches the sign. Some take the side trail and some walk past it, and then everyone continues on the same main trail. An if statement is that side trail. (The analogy stops at choice: a hiker decides on a whim, but Java decides only by the value of the Boolean expression.)

Here is the jacket check written correctly:

```
1 int temperature = 45;
2 if (temperature < 50)
3 {
4     System.out.println("Wear a jacket.");
5 }
6 System.out.println("Have a good day.");
```

Take lines 2 to 5 apart:

Piece	What it is
<code>if</code>	A Java keyword that starts the if statement. It is always lowercase.
<code>(temperature < 50)</code>	The condition : a Boolean expression, always inside parentheses. Java works it out first.
<code>{</code> and <code>}</code>	Curly braces that mark where the body starts and ends.
the line between them	The body : the statement or statements that run only when the condition is <code>true</code> .

An if statement with a single body that either runs or is skipped is called a **one-way selection**. There is only one special path: take the side trail, or do not.

KEY IDEA

In a one-way if statement, Java evaluates the condition first. If it is `true`, the body runs; if it is `false`, the body is skipped. Either way, the program then continues with the statement after the closing brace.

This picture shows both paths through the jacket check. Each path joins back up at the last line.

```

    is temperature < 50 ?
      /      \
    true      false
    |         |
  print jacket skip body
    \         /
    print "Have a good day."

```

WORKED EXAMPLE The condition is true

With `temperature` holding 45, follow the flow of control through the code above.

Line	What Java does	Printed
1	Stores 45 in <code>temperature</code> .	
2	Evaluates the condition: <code>45 < 50</code> is <code>true</code> , so it enters the body.	
4	Runs the body.	Wear a jacket.
6	The if statement is finished; continues with the next statement.	Have a good day.

```

Wear a jacket.
Have a good day.

```

Line 6 is not part of the if statement. It sits after the closing brace, so it runs no matter what the condition was.

Now change one thing: the temperature.

WORKED EXAMPLE The condition is false

```

1 int temperature = 68;
2 if (temperature < 50)
3 {
4     System.out.println("Wear a jacket.");
5 }
6 System.out.println("Have a good day.");

```

Line	What Java does	Printed
1	Stores 68 in <code>temperature</code> .	
2	Evaluates the condition: <code>68 < 50</code> is <code>false</code> , so it jumps past the closing brace.	
6	Continues with the next statement.	Have a good day.

```

Have a good day.

```

Line 4 never runs. The code is identical to the first example; only the value in `temperature` changed, and that changed which statements ran. This is what "selection changes sequential execution" means: the lines are still read top to bottom, but some of them can be skipped.

WATCH OUT

A semicolon right after the condition. Most Java lines end with `;`, so students often type one after `if (...)` out of habit:

```
1 int temperature = 68;  
2 if (temperature < 50);  
3 {  
4     System.out.println("Wear a jacket.");  
5 }  
6 System.out.println("Have a good day.");
```

```
Wear a jacket.  
Have a good day.
```

It is 68 degrees, and the jacket message still prints. Java reads the `;` at the end of line 2 as the end of the if statement: the body is an empty statement that does nothing. The braces on lines 3 to 5 are no longer connected to the `if`, so they are just a group of statements that always runs. The code compiles without any warning, which makes this slip hard to spot.

The fix is to delete the semicolon so the braces become the body again:

`if (temperature < 50)` followed directly by `{`. With that change, the program prints only `Have a good day.`, as in the second worked example.

PAUSE AND TRY**Question 1** Multiple choice

What does this code print?

```
1 int speed = 30;  
2 int limit = 35;  
3 if (speed > limit)  
4 {  
5     System.out.println("Slow down!");  
6 }  
7 System.out.println("Speed: " + speed);
```

- A. Slow down!
- B. Slow down! and then Speed: 30
- C. Speed: 30
- D. Nothing

Question 2 Tracing

This is the jacket check again, with `temperature` holding exactly `50`. Which lines run, and what is printed?

```

1 int temperature = 50;
2 if (temperature < 50)
3 {
4     System.out.println("Wear a jacket.");
5 }
6 System.out.println("Have a good day.");

```

2.3.2 Writing the Condition and the Body

The jacket check had a one-line body and a simple comparison. Real decisions often need more: when an order qualifies for free shipping, the program must both change the shipping cost and tell the customer. This subsection looks closely at what can go in each part of an if statement.

The condition can be any expression that works out to a `boolean` value. From Topic 2.2 that includes comparisons such as `total >= 50`. It also includes a `boolean` variable on its own, because the variable already holds `true` or `false`, and a method call that returns a `boolean`, such as `code.equals("SAVE5")` from Topic 1.15. Whatever the condition is, it goes inside parentheses, and the parentheses are required: `if total >= 50` does not compile.

The body can hold as many statements as you need. All of the statements between the braces form one group, called a **block**, and they run together: all of them when the condition is `true`, none of them when it is `false`. The body can do anything a normal statement can do, including changing a variable.

WORKED EXAMPLE A body with two statements

An online store charges \$5.00 shipping, but orders of \$50 or more ship free. Watch which statements the condition controls.

```

1 double total = 62.5;
2 double shipping = 5.0;
3 if (total >= 50)
4 {
5     shipping = 0.0;
6     System.out.println("Free shipping!");
7 }
8 total += shipping;
9 System.out.println("Total: " + total);

```

Line	What Java does	total	shipping	Printed
1–2	Stores the starting values.	62.5	5.0	
3	62.5 >= 50 is true; enter the body.	62.5	5.0	
5	Replaces the shipping cost.	62.5	0.0	
6	Prints the message.	62.5	0.0	Free shipping!
8	After the if statement: 62.5 + 0.0.	62.5	0.0	
9	Prints the total.	62.5	0.0	Total: 62.5

```
Free shipping!
Total: 62.5
```

If `total` had started at 42.5, the condition `42.5 >= 50` would be `false`. Lines 5 and 6 would both be skipped, `shipping` would keep its 5.0, and line 8 would make the total 47.5. The only output would be `Total: 47.5`. The braces are what make lines 5 and 6 a team: they are skipped or run together.

Now vary the kind of condition. This store takes \$2 off for members and \$5 off with a coupon code. Each discount is its own one-way if statement.

WORKED EXAMPLE A boolean variable and a method call as conditions

```
1 double price = 20.0;
2 boolean isMember = true;
3 String code = "SAVE5";
4 if (isMember)
5 {
6     price -= 2.0;
7 }
8 if (code.equals("SAVE5"))
9 {
10    price -= 5.0;
11 }
12 System.out.println("Price: " + price);
```

Line 4's condition is just `isMember`. Java does not need a comparison here: the variable holds `true`, and that value is the condition. (Writing `isMember == true` also works, but it compares `true` with `true` to get `true`, which adds nothing.)

Line 8's condition calls `equals`, which compares the characters of `code` with `"SAVE5"` and returns `true` because they match exactly.

Line	What Java does	price
1	Stores 20.0.	20.0
4	<code>isMember</code> is <code>true</code> ; run line 6.	18.0
8	<code>code.equals("SAVE5")</code> is <code>true</code> ; run line 10.	13.0
12	Prints the price.	13.0

```
Price: 13.0
```

WATCH OUT

Writing `=` instead of `==` in a condition. A single `=` assigns; it does not compare (Topic 2.2). Inside an if condition, that mistake does not compile:

```
1 int score = 100;
2 if (score = 100)
3 {
4     System.out.println("Perfect score!");
5 }
```

error: incompatible types: int cannot be converted to boolean

`score = 100` is an assignment, and its value is the `int` that was stored, `100`. An `if` statement needs a `boolean`, and Java will not treat a number as `true` or `false`, so it stops with this error. The fix is the comparison operator: `if (score == 100)`. (Exact error wording can vary between Java versions.)

WATCH OUT

Leaving out the braces around a body of several statements. Java allows an `if` statement without braces, but then the body is **only the one statement right after the condition**. Indenting the next line does not add it to the body; Java ignores indentation completely.

```
1 int temperature = 68;
2 if (temperature < 50)
3     System.out.println("Wear a jacket.");
4     System.out.println("Bring gloves.");
5 System.out.println("Have a good day.");
```

Bring gloves.
Have a good day.

The programmer meant both lines 3 and 4 to be the body. But with no braces, only line 3 belongs to the `if` statement. The condition `68 < 50` is `false`, so line 3 is skipped, and then line 4, which only looks like part of the body, runs as a normal statement.

The fix is to put braces around every statement that belongs to the body:

```
1 int temperature = 68;
2 if (temperature < 50)
3 {
4     System.out.println("Wear a jacket.");
5     System.out.println("Bring gloves.");
6 }
7 System.out.println("Have a good day.");
```

Now lines 4 and 5 are both skipped, and the program prints only `Have a good day.`. Writing braces even for a one-statement body is a good habit: it makes the body obvious and keeps this bug from appearing when you later add a second statement.

PAUSE AND TRY**Question 3** Debugging

When the player is hit, this code should take away a life and print `Ouch!`. When the player is not hit, it should print only the number of lives. Here `hitByEnemy` is `false`, yet the output is:

```
Ouch!  
Lives: 2
```

Find the error, explain why `Ouch!` prints, and correct the code.

```
1 int lives = 2;  
2 boolean hitByEnemy = false;  
3 if (hitByEnemy)  
4     lives--;  
5     System.out.println("Ouch!");  
6 System.out.println("Lives: " + lives);
```

Question 4 Multiple choice

Assume `count` is an `int`, `isReady` is a `boolean`, and `name` is a `String`, and each has a value. Which if statement does **not** compile?

- A. `if (count == 3)`
- B. `if (isReady)`
- C. `if (name.equals("Sam"))`
- D. `if (count = 3)`

2.3.3 Two-Way Selection with if-else

A one-way if statement handles "do this, or do nothing." Many decisions have two real outcomes instead. A quiz app should say "You passed." for a score of 70 or more, and "Try again next time." otherwise. Every student should see exactly one of those two messages.

You could try two one-way if statements, one with `score >= 70` and one with `score < 70`. That works here, but it makes Java check two conditions when one answer settles everything, and it relies on you writing the second condition as the exact opposite of the first. Java has a statement built for this situation.

An **if-else statement** has two bodies. The first runs when the condition is `true`. The second, after the keyword `else`, runs when the condition is `false`. Because a condition is always either `true` or `false`, exactly one of the two bodies runs: never both, never neither. This is called **two-way selection**. In the trail analogy, the path splits into a left branch and a right branch, every hiker takes one of them, and the branches meet again further on.

```

1 int score = 64;
2 if (score >= 70)
3 {
4     System.out.println("You passed.");
5 }
6 else
7 {
8     System.out.println("Try again next time.");
9 }
10 System.out.println("Score: " + score);

```

Piece	What it is
if (score >= 70)	The condition, exactly as in a one-way if statement.
first { ... } (lines 3–5)	The body of the if: runs only when the condition is <code>true</code> .
else	A keyword meaning "otherwise." It has no condition of its own.
second { ... } (lines 7–9)	The body of the else: runs only when the condition is <code>false</code> .

The `else` has no condition because it does not need one. It covers every case the `if` did not, so the condition is checked once and decides between both bodies.

KEY IDEA

In an if-else statement, Java evaluates the condition once. If it is `true`, the if body runs; if it is `false`, the else body runs. Exactly one of the two bodies runs, and then the program continues after the if-else statement.

```

      is score >= 70 ?
      /      \
    true      false
    |         |
    |         |
  "You passed." "Try again..."
    \         /
    print the score

```

WORKED EXAMPLE Pass or try again

Trace the code above with `score` holding 64.

Line	What Java does	Printed
1	Stores 64.	
2	64 >= 70 is false, so skip the if body and go to the else.	
8	Runs the else body.	Try again next time.
10	After the if-else statement; always runs.	Score: 64

```

Try again next time.
Score: 64

```

Line 4 is skipped. With `score` holding 70 instead, `70 >= 70` is `true`: line 4 prints `You passed.`, line 8 is skipped, and line 10 prints `Score: 70`. A value right at the boundary is the one most worth testing, because it shows whether the operator should be `>` or `>=`.

Now vary the condition. Topic 1.3's remainder operator gives a classic two-way decision: a whole number is even when dividing it by 2 leaves remainder 0, and odd otherwise.

WORKED EXAMPLE Even or odd

```
1 int number = 17;
2 if (number % 2 == 0)
3 {
4     System.out.println(number + " is even.");
5 }
6 else
7 {
8     System.out.println(number + " is odd.");
9 }
```

Line 2 first works out `17 % 2`, which is 1 (two goes into 17 eight times, which is 16, with 1 left over). Then it compares: `1 == 0` is `false`. So the `if` body on line 4 is skipped and the `else` body on line 8 runs:

```
17 is odd.
```

There is no need for a second check like `number % 2 == 1`. If a number is not even, it is odd, and `else` handles every "not" case automatically.

WATCH OUT

A semicolon after the condition, now with an `else`. In 2.3.1 a stray semicolon after `if (...)` compiled and quietly did the wrong thing. With an `else`, the same slip stops the program from compiling:

```
1 int score = 64;
2 if (score >= 70);
3 {
4     System.out.println("You passed.");
5 }
6 else
7 {
8     System.out.println("Try again next time.");
9 }
```

```
error: 'else' without 'if'
```

The `;` on line 2 ends the `if` statement right there. The braces on lines 3 to 5 become a separate group of statements, so by the time Java reaches `else` on line 6, it is no longer attached to any `if`. An `else` must come directly after the closing brace of an `if` body. Delete the semicolon to fix it. (Exact error wording can vary between Java versions.)

PAUSE AND TRY**Question 5** Tracing

What does this code print? Say which body runs and what `price` holds at the end.

```
1 int age = 12;
2 double price;
3 if (age < 13)
4 {
5     price = 8.5;
6     System.out.println("Child ticket");
7 }
8 else
9 {
10    price = 12.0;
11    System.out.println("Adult ticket");
12 }
13 System.out.println("Price: " + price);
```

Question 6 Debugging

This code should print `Not yet.` for anyone younger than 16, but it does not compile. Javac reports `error: not a statement` and points at the `else` line. Find the error, explain it, and correct the line.

```
1 int age = 15;
2 if (age >= 16)
3 {
4     System.out.println("You may drive.");
5 }
6 else (age < 16)
7 {
8     System.out.println("Not yet.");
9 }
```

2.3.4 Choosing One-Way or Two-Way Selection

You now have two tools: a one-way if statement, which runs its body or skips it, and an if-else statement, which always runs exactly one of its two bodies. Picking the right one depends on the question the program is asking.

- Use a **one-way if** when there is something extra to do in one case and nothing special to do otherwise: add a bonus, print a warning, apply a discount.
- Use an **if-else** when there are two outcomes and exactly one of them must happen: pass or try again, even or odd, child ticket or adult ticket.

When a program has several one-way if statements in a row, each one is a separate statement with its own condition. Topic 2.1 noted that the order of the steps in an algorithm affects its result. That matters here: Java runs the if statements one after another, checks every condition, and each

condition sees the variables as earlier statements left them. So any number of their bodies can run: all, some, or none.

KEY IDEA

Separate if statements are independent: each condition is checked in turn, so several bodies, or none, can run. In an if-else statement, exactly one body runs.

WORKED EXAMPLE Two separate if statements can both run

```
1 int score = 93;
2 if (score >= 90)
3 {
4     System.out.println("Excellent work!");
5 }
6 if (score >= 70)
7 {
8     System.out.println("You passed.");
9 }
```

With `score` holding 93, line 2 checks `93 >= 90`, which is `true`, and prints `Excellent work!`. Then line 6 starts a separate if statement and checks `93 >= 70`, which is also `true`, so it prints `You passed.` as well.

```
Excellent work!
You passed.
```

Both messages make sense for a 93, so here two one-way if statements are the right choice. For a score of 80, only the second would print; for 50, neither would.

Now vary the code so the first body changes the variable that the second condition checks. A student with \$12 on a transit card buys a \$10 ticket if they can afford it, and the program warns when the balance drops below \$10.

WORKED EXAMPLE A body that changes the next condition

```
1 int balance = 12;
2 if (balance >= 10)
3 {
4     balance -= 10;
5     System.out.println("Bought a ticket.");
6 }
7 if (balance < 10)
8 {
9     System.out.println("Balance is low.");
10 }
11 System.out.println("Balance: " + balance);
```

Line	What Java does	balance	Printed
1	Stores 12.	12	
2	12 >= 10 is true; enter the body.	12	

4	Subtracts the ticket price.	2	
5	Prints.	2	Bought a ticket.
7	A new if statement: <code>2 < 10</code> is true (uses the new value).	2	
9	Prints.	2	Balance is low.
11	After both if statements.	2	Balance: 2

```
Bought a ticket.
Balance is low.
Balance: 2
```

The conditions on lines 2 and 7 look like exact opposites, so you might expect only one message. But line 7 is checked **after** line 4 changed `balance`, so it sees 2, not 12.

Compare the same code written as an if-else:

```
1 int balance = 12;
2 if (balance >= 10)
3 {
4     balance -= 10;
5     System.out.println("Bought a ticket.");
6 }
7 else
8 {
9     System.out.println("Balance is low.");
10 }
11 System.out.println("Balance: " + balance);
```

```
Bought a ticket.
Balance: 2
```

Here the condition is checked once, with `balance` still 12. It is true, so the if body runs and the else body is skipped, even though `balance` is low afterward. Neither version is wrong; they answer different questions. The two separate if statements warn about the balance **after** the purchase. The if-else chooses between buying and warning based on the balance **before** it.

WATCH OUT

Using a variable that only one path gives a value. Topic 1.4 said a variable must be given a value before it is used. Java checks every possible path through the code, including the path where a one-way if body is skipped:

```
1 int score = 85;
2 String message;
3 if (score >= 70)
4 {
5     message = "Pass";
6 }
7 System.out.println(message);
```

error: variable message might not have been initialized

You can see that `score` is 85, so the body will run. But Java checks the code before it runs and does not rely on the value being stored. It sees that if the condition were `false`, line 5 would be skipped and `message` would have no value on line 7. So it refuses to compile.

An if-else fixes this, because exactly one of its bodies always runs. If both bodies give `message` a value, every path does:

```
1 int score = 85;
2 String message;
3 if (score >= 70)
4 {
5     message = "Pass";
6 }
7 else
8 {
9     message = "Not yet";
10 }
11 System.out.println(message);
```

This prints `Pass`. Giving the variable a starting value when you declare it, such as `String message = "Not yet";`, would also fix the error.

PAUSE AND TRY

Question 7 Tracing

What does this code print? Track `stock` as you go.

```
1 int stock = 4;
2 if (stock < 5)
3 {
4     stock += 10;
5     System.out.println("Restocked.");
6 }
7 if (stock >= 5)
8 {
9     System.out.println("In stock.");
10 }
11 System.out.println("Stock: " + stock);
```

Question 8 Multiple choice

Assume `points` is an `int` that holds some value. Which code segment always prints **exactly one** line, no matter what `points` holds?

- A. Two one-way if statements, `if (points > 10)` and then `if (points > 20)`, each with one `println` in its body
- B. One one-way if statement, `if (points > 10)`, with one `println` in its body

- C. One if-else statement, `if (points > 10)`, with one `println` in the if body and one `println` in the else body
- D. Two one-way if statements, `if (points > 10)` and then `if (points < 10)`, each with one `println` in its body

Putting it together

A library charges 25 cents per day for a late book, up to a maximum of \$2.00. This program reads the number of days late, then uses an if-else statement and a one-way if statement to work out the fee. As in Unit 1, the `Scanner` reads from a string so the output is always the same; reading from the keyboard works the same way.

WORKED EXAMPLE A library late fee

LibraryFee.java

```

1  import java.util.Scanner;
2
3  public class LibraryFee
4  {
5      public static void main(String[] args)
6      {
7          Scanner input = new Scanner("11");
8          int daysLate = input.nextInt();
9          double fee = 0.0;
10
11         if (daysLate > 0)
12         {
13             fee = daysLate * 0.25;
14             System.out.println("Overdue by " + daysLate + " days.");
15         }
16         else
17         {
18             System.out.println("Returned on time. Thank you!");
19         }
20
21         if (fee > 2.0)
22         {
23             fee = 2.0;
24             System.out.println("Fee capped at the maximum.");
25         }
26
27         System.out.println("Fee: $" + fee);
28     }
29 }
```

Before reading the trace, predict the output for 11 days late.

Line	What Java does	daysLate	fee
8	Reads 11.	11	

9	Starts the fee at 0.0.	11	0.0
11	11 > 0 is true: run the if body, skip the else body.	11	0.0
13	11 * 0.25 is 2.75.	11	2.75
14	Prints Overdue by 11 days.	11	2.75
21	A separate if statement: 2.75 > 2.0 is true.	11	2.75
23	Caps the fee.	11	2.0
24	Prints Fee capped at the maximum.	11	2.0
27	Runs on every path.	11	2.0

Overdue by 11 days.
 Fee capped at the maximum.
 Fee: \$2.0

The first decision has two outcomes, late or on time, and exactly one message must print, so it is an if-else. The cap is something extra that happens only sometimes, so it is a one-way if. Notice that line 21 checks the fee **after** line 13 may have changed it, the same way the transit card example did.

Here is how other inputs flow through the same code:

Input	First decision	Cap?	Output
3	if body: fee is 0.75	0.75 > 2.0 is false	Overdue by 3 days. then Fee: \$0.75
0	else body: fee stays 0.0	0.0 > 2.0 is false	Returned on time. Thank you! then Fee: \$0.0

Line 9 matters: it gives `fee` a value before the if-else, so `fee` has a value on line 27 even on the on-time path, where the if body that sets it is skipped.

Vocabulary

Term	Definition	Example
sequential execution	Running statements one after another in the order they are written	Every Unit 1 program
selection	A step where what happens next depends on a true or false decision	"If it is cold, wear a jacket."
flow of control	The order in which a program's statements actually run	Line 4 skipped, line 6 runs
selection statement	A statement that changes the flow of control based on a Boolean expression	an if statement
if statement	A selection statement that runs its body only when its condition is <code>true</code>	<code>if (temperature < 50)</code>
condition	The Boolean expression in parentheses after <code>if</code>	<code>(total >= 50)</code>

body	The statement or block of statements that an if statement (or an <code>else</code>) controls	<code>{ shipping = 0.0; }</code>
block	A group of statements between curly braces that run together	<code>{ ... }</code>
one-way selection	An if statement with one body that either runs or is skipped	<code>if (isMember) { ... }</code>
two-way selection	An if-else statement, where exactly one of two bodies runs	pass or try again
if-else statement	An if statement with a second body after <code>else</code> that runs when the condition is <code>false</code>	<code>if (...) { ... } else { ... }</code>

Chapter summary

- Without selection, Java runs every statement in order. A selection statement changes this flow of control, so some statements run and others are skipped depending on a Boolean expression.
- In a one-way if statement, the body runs only when the condition is `true`; when it is `false`, Java skips to the statement after the closing brace. Statements after the if statement run either way.
- The condition must be a `boolean` expression in parentheses: a comparison, a `boolean` variable, or a method call such as `equals`. Braces group a body of several statements; without them, only the next statement belongs to the body, whatever the indentation.
- In an if-else statement, the condition is checked once: the if body runs when it is `true` and the else body when it is `false`, so exactly one body runs. `else` has no condition.
- Separate if statements are each checked in turn, so several bodies or none may run, and a later condition sees any changes made by an earlier body.
- A semicolon right after `if (...)` ends the if statement: on its own it silently makes the body always run, and before an `else` it causes a compile error.

Review questions

Question 9 Multiple choice

What does this code print?

```
1 String day = "sat";
2 if (day.equals("Sat"))
3 {
4     System.out.println("Weekend");
5 }
6 else
7 {
8     System.out.println("Weekday");
9 }
```

- A. Weekend
- B. Weekday
- C. Weekend and then Weekday

D. Nothing; a `String` method cannot be used as a condition.

Question 10 Tracing

This code rounds a movie's length to the nearest hour and then describes it. Write the exact output and show the values of `hours` and `extra`.

```
1 int minutes = 135;
2 int hours = minutes / 60;
3 int extra = minutes % 60;
4 if (extra >= 30)
5 {
6     hours++;
7 }
8 if (hours > 2)
9 {
10     System.out.println("Long movie");
11 }
12 else
13 {
14     System.out.println("Short movie");
15 }
16 System.out.println("About " + hours + " hours");
```

Question 11 Debugging

A class rule says a score of 70 **or more** passes. This code prints `Try again next time.` for a score of exactly 70. Find the error and correct the line.

```
1 int score = 70;
2 if (score > 70)
3 {
4     System.out.println("You passed.");
5 }
6 else
7 {
8     System.out.println("Try again next time.");
9 }
```

Question 12 Multiple choice

What does this code print?

```
1 double average = 89.6;
2 if ((int) average >= 90)
3 {
4     System.out.println("A");
5 }
6 else
7 {
8     System.out.println("B");
9 }
```

- A. A
- B. B

- C. A and then B
- D. Nothing; a cast cannot appear in a condition.

Question 13 Short answer

Both bodies of this if-else statement end with the same `println`:

```
1 double total = 18.0;
2 if (total >= 25.0)
3 {
4     System.out.println("You earned a coupon.");
5     System.out.println("Thanks for shopping!");
6 }
7 else
8 {
9     System.out.println("Spend $25 for a coupon.");
10    System.out.println("Thanks for shopping!");
11 }
```

A student moves `System.out.println("Thanks for shopping!");` out of both bodies and places it once, after the closing brace of the else body:

```
1 double total = 18.0;
2 if (total >= 25.0)
3 {
4     System.out.println("You earned a coupon.");
5 }
6 else
7 {
8     System.out.println("Spend $25 for a coupon.");
9 }
10 System.out.println("Thanks for shopping!");
```

Does the output change for any value of `total`? Explain why or why not.

Question 14 Multiple choice

Consider these two code segments. Each starts with a different value of `x` in place of `7`.

Segment I:

```
1 int x = 7;
2 if (x > 5)
3 {
4     x -= 3;
5 }
6 if (x <= 5)
7 {
8     System.out.println("small");
9 }
```

Segment II:

```
1 int x = 7;
2 if (x > 5)
3 {
4     x -= 3;
5 }
6 else
7 {
8     System.out.println("small");
9 }
```

For which starting value of x do the two segments print **different** output?

- A. 2
- B. 5
- C. 7
- D. 10

Reflection

Check each statement. If the answer is "not yet," reread the linked subsection.

- ☐ I can explain how an if statement changes the usual top-to-bottom order of a program. Reread: 2.3.1 Running Code Only When a Condition Is True
- ☐ I can trace a one-way if statement and say whether its body runs. Reread: 2.3.1 Running Code Only When a Condition Is True
- ☐ I can write a condition that is a boolean expression and use braces around a body of several statements. Reread: 2.3.2 Writing the Condition and the Body
- ☐ I can trace an if-else statement and explain why exactly one body runs. Reread: 2.3.3 Two-Way Selection with if-else
- ☐ I can choose between a one-way if and an if-else, and trace separate if statements that affect each other. Reread: 2.3.4 Choosing One-Way or Two-Way Selection

Check Your Reasoning

Answers and explanations for every question, numbered to match.

Question 1

C. The condition `speed > limit` is `30 > 35`, which is `false`, so the body on line 5 is skipped. Line 7 comes after the closing brace, so it runs either way and prints `Speed: 30`.

- **A** is wrong: the body runs only when the condition is `true`, and `30 > 35` is `false`. It also forgets line 7, which always runs.
- **B** is what would print if the condition were `true`, for example with `speed` holding `40`.
- **D** is wrong: skipping the body does not end the program. Line 7 is outside the `if` statement.

Reread 2.3.1 *Running Code Only When a Condition Is True* if you chose differently.

Question 2

Lines 1, 2, and 6 run. Only `Have a good day.` is printed.

Line	What Java does	Printed
1	Stores <code>50</code> .	
2	<code>50 < 50</code> is <code>false</code> , so the body is skipped.	
6	Runs, because it is after the <code>if</code> statement.	<code>Have a good day.</code>

`<` means "strictly less than," so a value equal to `50` does not count. If the program should suggest a jacket at exactly 50 degrees too, the condition would need `<=`.

Reread 2.3.1 *Running Code Only When a Condition Is True* if your answer differed.

Question 3

The body has no braces. Without braces, the body of the `if` statement is only line 4, `lives--`; . Line 5 is indented to look like part of the body, but Java ignores indentation, so line 5 is an ordinary statement after the `if` statement and always runs. Because `hitByEnemy` is `false`, line 4 is skipped (so `lives` stays `2`), but line 5 still prints `Ouch!`.

Corrected code:

```
1 int lives = 2;
2 boolean hitByEnemy = false;
3 if (hitByEnemy)
4 {
5     lives--;
6     System.out.println("Ouch!");
7 }
8 System.out.println("Lives: " + lives);
```

With braces, lines 5 and 6 form the body and are skipped together. It prints only:

```
Lives: 2
```

Reread 2.3.2 *Writing the Condition and the Body* if you missed it.

Question 4

D. `count = 3` assigns 3 to `count`, and the value of that expression is the `int` 3. A condition must be a `boolean`, so Java reports incompatible types: `int` cannot be converted to `boolean`.

- **A** compiles: `==` compares two `int` values and gives `true` or `false`.
- **B** compiles: a `boolean` variable already holds `true` or `false`, so it can be a condition by itself.
- **C** compiles: `equals` returns a `boolean`.

Reread 2.3.2 *Writing the Condition and the Body* if you chose differently.

Question 5

The `if` body runs, and `price` ends up holding 8.5.

Line	What Java does	price	Printed
1-2	Stores 12 in <code>age</code> ; declares <code>price</code> .	none	
3	<code>12 < 13</code> is <code>true</code> ; run the <code>if</code> body.		
5	Stores 8.5.	8.5	
6	Prints.	8.5	Child ticket
13	After the <code>if-else</code> statement.	8.5	Price: 8.5

```
Child ticket
Price: 8.5
```

Lines 10 and 11 are skipped because only one body of an `if-else` runs.

Reread 2.3.3 *Two-Way Selection with if-else* if your answer differed.

Question 6

Line 6 gives the `else` a condition. An `else` never has a condition; it runs whenever the `if` condition is `false`. Java reads `else` followed by `(age < 16)` as "otherwise, do the statement `(age < 16)`," and a comparison by itself is not a statement (it works out a value but stores or prints nothing), so it reports `not a statement`. Line 6 is also unnecessary: if `age >= 16` is `false`, then `age < 16` must be `true`.

Corrected code:

```
1 int age = 15;
2 if (age >= 16)
3 {
4     System.out.println("You may drive.");
5 }
6 else
7 {
8     System.out.println("Not yet.");
9 }
```

It prints `Not yet.`, because `15 >= 16` is `false`.

Reread 2.3.3 *Two-Way Selection with if-else* if you missed it.

Question 7

```
Restocked.
In stock.
Stock: 14
```

Line	What Java does	stock	Printed
1	Stores 4.	4	
2	4 < 5 is true; enter the body.	4	
4	4 + 10.	14	
5	Prints.	14	Restocked.
7	Separate if statement: 14 >= 5 is true.	14	
9	Prints.	14	In stock.
11	After both if statements.	14	Stock: 14

If you left out `In stock.`, you checked line 7 with the starting value 4. Each if statement is checked in turn, using the value the earlier statements left.

Reread 2.3.4 *Choosing One-Way or Two-Way Selection* if your answer differed.

Question 8

C. In an if-else statement, exactly one of the two bodies runs, and each body prints one line.

- **A** is wrong: for 25 both conditions are true and two lines print; for 5 neither is, and nothing prints.
- **B** is wrong: when points is 10 or less, the body is skipped and nothing prints.
- **D** is wrong: it looks like a two-way choice, but neither condition covers 10. When points is exactly 10, both are false and nothing prints. An else never misses a case like this.

Reread 2.3.4 *Choosing One-Way or Two-Way Selection* if you chose differently.

Question 9

B. `equals` compares the characters exactly, and uppercase and lowercase letters count as different (Topic 1.15). "sat" and "Sat" differ in the first letter, so `day.equals("Sat")` is false and the else body runs.

- **A** is wrong: it treats `equals` as if it ignored case. It does not.
- **C** is wrong: in an if-else statement, exactly one body runs, never both.
- **D** is wrong: `equals` returns a boolean, and any boolean expression can be a condition.

Reread 2.3.2 *Writing the Condition and the Body* and 2.3.3 *Two-Way Selection with if-else* if you chose differently.

Question 10

```
Short movie
About 2 hours
```

Line	What Java does	hours	extra
2	135 / 60 is 2 (integer division drops the rest).	2	

3	135 % 60 is 15 (two hours use 120 minutes).	2	15
4	15 >= 30 is false; skip line 6.	2	15
8	2 > 2 is false; run the else body.	2	15
14	Prints Short movie.	2	15
16	Prints About 2 hours.	2	15

Two traps here. On line 2, integer division gives 2, not 2.25. On line 8, > means strictly greater, so 2 > 2 is false. If you got Long movie, check both.

Reread 2.3.1 *Running Code Only When a Condition Is True* and 2.3.3 *Two-Way Selection with if-else* if your output differed.

Question 11

Line 2 uses > where the rule needs >=. 70 > 70 is false, because > means strictly greater than, so a score of exactly 70 falls into the else body. The code compiles and works for every score except the boundary, which is why testing the boundary value matters.

Corrected code:

```
1 int score = 70;
2 if (score >= 70)
3 {
4     System.out.println("You passed.");
5 }
6 else
7 {
8     System.out.println("Try again next time.");
9 }
```

Now 70 >= 70 is true, and it prints You passed..

Reread 2.3.3 *Two-Way Selection with if-else* if you missed it.

Question 12

B. Java evaluates the condition before choosing a body. The cast happens first: (int) 89.6 drops the decimal part and gives 89 (Topic 1.5). Then 89 >= 90 is false, so the else body runs.

- **A** is wrong: it rounds 89.6 up to 90. Casting a double to an int truncates; it never rounds.
- **C** is wrong: an if-else runs exactly one body.
- **D** is wrong: the condition can be any expression that works out to a boolean, and (int) average >= 90 does.

Reread 2.3.3 *Two-Way Selection with if-else* if you chose differently.

Question 13

Model answer: No. In the first version, exactly one body runs, and whichever one it is ends by printing Thanks for shopping!, so that line always prints last. In the second version, the line is after the if-else statement, so it runs on every path, also last. For 18.0 both print

Spend \$25 for a coupon. and then Thanks for shopping!; for a total of 25.0 or more, both print the coupon message and then the thanks line. The second version is clearer because a statement that runs in every case does not belong inside a selection statement.

A complete answer must:

- say that the output never changes,
- explain that exactly one body of an if-else runs, so the thanks line printed once in the first version, and
- explain that a statement after the if-else runs no matter which body ran.

Reread 2.3.3 *Two-Way Selection with if-else* if your answer differed.

Question 14

C. With 7, both segments run the first body, so `x` becomes 4. Segment I then checks a separate if statement, `4 <= 5`, which is `true`, and prints `small`. Segment II has an `else`, which is skipped because its if body already ran, so it prints nothing.

Start	Segment I	Segment II
2	<code>2 > 5</code> false; then <code>2 <= 5</code> true: <code>small</code>	<code>2 > 5</code> false: else prints <code>small</code>
5	<code>5 > 5</code> false; then <code>5 <= 5</code> true: <code>small</code>	<code>5 > 5</code> false: else prints <code>small</code>
7	becomes 4; then <code>4 <= 5</code> true: <code>small</code>	becomes 4; else skipped: nothing
10	becomes 7; then <code>7 <= 5</code> false: nothing	becomes 7; else skipped: nothing

- **A** and **B** are wrong: the first condition is `false`, so `x` never changes, and both segments print `small`.
- **D** is wrong: `x` drops to 7, which is still more than 5, so Segment I's second condition is `false` too. Neither segment prints anything.

Reread 2.3.4 *Choosing One-Way or Two-Way Selection* if you chose differently.