2.4 Nested if Statements

Self-Study Guide · About 50 minutes

By the end you will be able to

- Trace code that has an if or if-else statement inside another one, and say which conditions Java checks.
- Explain why an inner condition is checked only when the program reaches the branch that holds it, and use that to keep a risky check safe.
- Work out which if an else belongs to, and use braces to make the pairing you want.
- Trace a multiway selection (if, else if, else) and explain why at most one branch runs.
- Choose the order of conditions in a chain so every value lands in the right branch, and decide when a trailing else is needed.

What to remember from earlier

- The relational operators each evaluate to true or false. (Topic 2.2: Boolean Expressions)
- An if statement runs its body only when its condition is true. (Topic 2.3: if Statements, 2.3.1 Running Code Only When a Condition Is True)
- An if-else statement runs exactly one of its two bodies. (Topic 2.3: if Statements, 2.3.3 Two-Way Selection with if-else)
- Dividing an int by zero crashes the program with an ArithmeticException. (Topic 1.3: Expressions and Output, 1.3.9 Dividing an Integer by Zero)
- Calling substring with an index past the end of a String throws a StringIndexOutOfBoundsException. (Topic 1.15: String Manipulation, 1.15.6 Out-of-Range Indices)

2.4.1 An if Inside Another if

A thermostat app should print "Comfortable" when the room is between 60 and 80 degrees, counting both ends. That is really two questions: is the temperature at least 60, **and** is it at most 80? Every Boolean expression you know from Topic 2.2 asks only one question, such as `temperature >= 60`. With the tools you have so far, how can a program act only when two things are both true?

Think about how a person checks. First they ask "Is it at least 60?" If the answer is no, they stop: the room is too cold, and there is no point asking the second question. Only if the answer is yes do they go on to ask "Is it at most 80?" The second question is asked *inside* the "yes" answer to the first.

Java lets you write exactly that. In Topic 2.3 you saw that the body of an `if` statement can hold any statements. An `if` statement is itself a statement, so it can go inside another `if` statement's body. A **nested if statement** is an `if` or `if-else` statement placed inside the body of another one. (In 2.4.4 you will meet a third kind of selection statement that can be nested the same way.) The one on the outside is the **outer if statement**; the one inside its body is the **inner if statement**.

WORKED EXAMPLE **Checking a range with two nested ifs**

Look at where the second `if` sits: it is between the braces of the first one.

```
1 int temperature = 72;
2 if (temperature >= 60)
3 {
4     if (temperature <= 80)
5     {
6         System.out.println("Comfortable");
7     }
8 }
9 System.out.println("Check done");
```

Follow Java one step at a time with `temperature` equal to `72`:

Line	What Java does	Printed
2	Evaluates the outer condition `72 >= 60`: `true`. Enters the outer body.	
4	Evaluates the inner condition `72 <= 80`: `true`. Enters the inner body.	
6	Prints.	`Comfortable`
9	The whole outer `if` is finished; the next statement runs.	`Check done`

```
Comfortable
Check done
```

Now change only the value and watch which lines Java even looks at:

`temperature`	Outer `temperature >= 60`	Inner `temperature <= 80`	Output
`72`	`true`	`true`	`Comfortable`, `Check done`
`85`	`true`	`false`	`Check done`
`50`	`false`	never evaluated	`Check done`
`60`	`true`	`true`	`Comfortable`, `Check done`

The `50` row is the important one. When the outer condition is `false`, Java skips the entire outer body, and the inner `if` is part of that body. So the inner condition is not evaluated at all; it is not checked and found false, it is simply never reached. The `60` row shows the boundary: `>=` includes 60, so 60 counts as comfortable.

KEY IDEA

The condition of an inner if statement is evaluated only when the outer condition is true, because the inner if is part of the outer body and Java skips that whole body when the outer condition is false.

Why "never evaluated" matters

That rule is not just a detail of tracing. You can use it to protect a check that would crash. Recall from Topic 1.3 that dividing an `int` by zero stops the program with an `ArithmeticException`. Suppose a gradebook should print "Honor roll" when a student's average quiz score is at least 80, but a new student has taken zero quizzes:

```
1  int totalPoints = 255;
2  int quizCount = 0;
3  if (quizCount != 0)
4  {
5      if (totalPoints / quizCount >= 80)
6      {
7          System.out.println("Honor roll");
8      }
9  }
10 System.out.println("Done");
```

```
Done
```

The outer condition `quizCount != 0` is `0 != 0`, which is `false`. Java skips the outer body, so `totalPoints / quizCount` on line 5 is never computed and nothing divides by zero. If `quizCount` were `3` instead, the outer condition would be `true`, the inner condition would compute `255 / 3`, which is `85`, and the program would print `Honor roll` and then `Done`. An outer condition used like this, to make sure the inner condition is safe to evaluate, is called a **guard**.

WATCH OUT

Putting the guard on the inside. A student writes the same two checks in the other order:

```
1  int totalPoints = 255;
2  int quizCount = 0;
3  if (totalPoints / quizCount >= 80)
4  {
5      if (quizCount != 0)
6      {
7          System.out.println("Honor roll");
8      }
9  }
10 System.out.println("Done");
```

Nothing prints. The program crashes on line 3 with a message like this (the exact wording can vary between Java versions):

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

The student's idea was right ("check that there are quizzes"), but the check is in the wrong place. Java evaluates the outer condition first, and that condition divides by `quizCount`, which is `0`. The inner check never gets a chance to help. A guard only protects what is *inside* it, so the safety check must be the outer condition, as in the version above.

PAUSE AND TRY

Question 1 Tracing

Write the exact output when `speed` is `38`. Then write the output when line 1 is changed to `int speed = 20;`.

```java
int speed = 38;
if (speed > 25)
{
    System.out.println("Over the limit");
    if (speed > 35)
    {
        System.out.println("Ticket");
    }
    System.out.println("Slow down");
}
System.out.println("Drive safely");
```

Question 2 Multiple choice

Which statement about running this code is true?

```java
int pagesRead = 120;
int days = 0;
if (days > 0)
{
    if (pagesRead / days > 10)
    {
        System.out.println("Fast reader");
    }
}
System.out.println("Report finished");
```

A. The inner condition is evaluated and is `false`, so `Fast reader` is not printed.
B. The inner condition is never evaluated, so no division happens and the program prints `Report finished`.
C. The program crashes with an `ArithmeticException` because `days` is `0`.
D. The code does not compile because you cannot put an `if` inside another `if`.

2.4.2 Nesting in Either Branch

An online store charges shipping in two different ways. Members get free shipping on orders of $25 or more and pay $3 otherwise. Everyone else gets free shipping at $50 or more and pays $6 otherwise. The first question is "Is this a member?", and the second question depends on the answer. In 2.4.1 the inner `if` sat inside the "true" body. Here each answer needs its own follow-up question.

Recall from Topic 2.3 that an `if-else` statement has two bodies, and exactly one of them runs. Either body can hold a nested `if`. A nested `if` inside the `else` body is reached only when the outer condition is `false`. In general: an inner condition is evaluated only when Java actually enters the body that contains it.

WORKED EXAMPLE **Shipping rules with an if-else in each branch**

Each body of the outer `if-else` holds its own inner `if-else`:

```
 1 boolean isMember = false;
 2 int orderTotal = 40;
 3 if (isMember)
 4 {
 5     if (orderTotal >= 25)
 6     {
 7         System.out.println("Shipping: free");
 8     }
 9     else
10     {
11         System.out.println("Shipping: $3");
12     }
13 }
14 else
15 {
16     if (orderTotal >= 50)
17     {
18         System.out.println("Shipping: free");
19     }
20     else
21     {
22         System.out.println("Shipping: $6");
23     }
24 }
```

The choices form a tree. Java walks down one path, asking at most two questions:

```
isMember?
├── true:  orderTotal >= 25?
│          ├── true:  free
│          └── false: $3
└── false: orderTotal >= 50?
           ├── true:  free
           └── false: $6
```

With `isMember` equal to `false` and `orderTotal` equal to `40`:

Line	What Java does
3	`isMember` is `false`. Skips lines 4 to 13 entirely and enters the `else` body.
16	Evaluates `40 >= 50`: `false`. Skips line 18 and enters the inner `else` on line 20.
22	Prints `Shipping: $6`.

```
Shipping: $6
```

> The condition `orderTotal >= 25` on line 5 was never evaluated, because Java never entered the outer `if` body.

Here are all four kinds of customer. In every row, exactly one of the four `println` statements runs:

isMember	orderTotal	Conditions evaluated	Output
`true`	`30`	`isMember`, then `orderTotal >= 25`	`Shipping: free`
`true`	`10`	`isMember`, then `orderTotal >= 25`	`Shipping: $3`
`false`	`60`	`isMember`, then `orderTotal >= 50`	`Shipping: free`
`false`	`40`	`isMember`, then `orderTotal >= 50`	`Shipping: $6`

Nesting does not have to be in both branches. A parking lot is free on weekends; on weekdays it costs $2 after 6 p.m. (hour 18 on a 24-hour clock) and $5 before. Only the weekday answer needs a second question, so only the `else` body holds a nested `if`:

```
boolean isWeekend = false;
int hour = 19;
if (isWeekend)
{
    System.out.println("Parking: free");
}
else
{
    if (hour >= 18)
    {
        System.out.println("Parking: $2");
    }
    else
    {
        System.out.println("Parking: $5");
    }
}
```

```
Parking: $2
```

`isWeekend` is `false`, so Java enters the `else` body and evaluates `19 >= 18`, which is `true`. Had `isWeekend` been `true`, the program would have printed `Parking: free` and never looked at `hour`, whatever its value.

KEY IDEA

A nested if can sit in the if body, the else body, or both. Its condition is evaluated only when Java enters the body that holds it, so in a nested if-else exactly one path through the tree runs.

WATCH OUT

Thinking the inner `else` catches the outer condition being false. Read this code and predict what prints for a 14-year-old:

```java
int age = 14;
boolean hasLicense = false;
if (age >= 16)
{
    if (hasLicense)
    {
        System.out.println("You may drive");
    }
    else
    {
        System.out.println("Get your license first");
    }
}
System.out.println("Done");
```

Many students say `Get your license first`. The actual output is:

```
Done
```

The inner `else` belongs to the inner `if (hasLicense)`. It runs only when Java is already inside the outer body (so `age >= 16` was `true`) **and** `hasLicense` is `false`. Here `14 >= 16` is `false`, so the whole outer body, inner `else` included, is skipped. Students make this mistake because they read "else" as "otherwise, for anything that failed", but each `else` answers exactly one `if`. If you want a message for drivers who are too young, the outer `if` needs its own `else`:

```java
int age = 14;
boolean hasLicense = false;
if (age >= 16)
{
    if (hasLicense)
    {
        System.out.println("You may drive");
    }
    else
    {
        System.out.println("Get your license first");
    }
}
else
{
    System.out.println("Too young to drive");
}
System.out.println("Done");
```

```
Too young to drive
Done
```

PAUSE AND TRY

Question 3 Tracing

Write the exact output, and list every condition Java evaluates, in order.

```
boolean isWeekend = false;
int age = 30;
if (isWeekend)
{
    if (age < 12)
    {
        System.out.println("$7");
    }
    else
    {
        System.out.println("$11");
    }
}
else
{
    if (age < 12)
    {
        System.out.println("$5");
    }
    else
    {
        System.out.println("$9");
    }
}
```

Question 4 Short answer

In the parking example, a student says: "When `isWeekend` is `true`, Java checks `hour >= 18`, finds it does not matter, and prints `Parking: free`." Explain what is wrong with this description.

2.4.3 Which if an else Belongs To

Every example so far has used braces around every body, so it was easy to see which `else` went with which `if`. Java also lets you leave the braces off when a body is exactly one statement: the `if` (or `else`) then controls just that one statement. That is legal, and you will see such code on tests. But with nested `if`s it raises a real question: when there are two `if`s and one `else`, which `if` does the `else` belong to?

Java's rule: without braces, an `else` belongs to the **nearest** `if` above it that does not already have an `else`. Indentation does not matter. Java ignores spaces and line breaks when it decides how statements fit together, so lining an `else` up under one `if` does not attach it to that `if`. An `else` that could seem to belong to either of two `if`s is called a **dangling else**.

> **WORKED EXAMPLE** **An else that looks like it belongs to the outer if**
>
> The indentation suggests the writer wanted "`Failed`" for scores below 50. Read it the way Java does, not the way it is lined up:
>
> ```
> 1 int score = 70;
> 2 if (score >= 50)
> 3     if (score >= 90)
> 4         System.out.println("Excellent");
> 5 else
> 6     System.out.println("Failed");
> 7 System.out.println("Done");
> ```
>
> The nearest `if` without an `else` above line 5 is `if (score >= 90)` on line 3. So Java reads the code like this, with the `else` paired with the inner `if`:
>
> ```
> if (score >= 50)
> {
>  if (score >= 90)
>  {
>  System.out.println("Excellent");
>  }
>  else
>  {
>  System.out.println("Failed");
>  }
> }
> System.out.println("Done");
> ```
>
> Now trace `score` equal to `70`: `70 >= 50` is `true`, so Java enters the outer body. `70 >= 90` is `false`, so the `else` runs.
>
> ```
> Failed
> Done
> ```
>
> A passing score of 70 printed `Failed`, exactly the opposite of what the writer meant.

Change the score to `45`. Now `45 >= 50` is `false`, so Java skips the outer body, and the `else` is part of that body. The output is only `Done`; the failing score never printed `Failed`. And `95` prints

`Excellent` and `Done`, as expected. The bug only shows for some inputs, which is why it is easy to miss.

To get the pairing you actually want, put braces around the outer body. The closing brace ends the outer body before the `else`, so the `else` can only belong to the outer `if`. This version calls a helper method `report` (an invented class method, like `toMinutes` in Topic 1.10) three times so you can see all three scores:

DanglingElseFixed.java

```java
public class DanglingElseFixed
{
    public static void report(int score)
    {
        if (score >= 50)
        {
            if (score >= 90)
            {
                System.out.println("Excellent");
            }
        }
        else
        {
            System.out.println("Failed");
        }
        System.out.println("Done");
    }

    public static void main(String[] args)
    {
        report(70);
        report(45);
        report(95);
    }
}
```

```
Done
Failed
Done
Excellent
Done
```

`report(70)` prints only `Done` (passing, but not excellent), `report(45)` prints `Failed` and `Done`, and `report(95)` prints `Excellent` and `Done`.

KEY IDEA

Without braces, an else pairs with the nearest earlier if that does not already have an else, no matter how the code is indented. Braces around each body make the pairing you intend.

WATCH OUT

Trusting the indentation. Indentation is for human readers; the compiler never looks at it. Code with a dangling `else` compiles and runs without any error message, so nothing warns you. When you trace code without braces, ignore the indentation, find each `else`, and pair it with the nearest unpaired `if` above it. When you write code, use braces on every body and the question never comes up.

PAUSE AND TRY

Question 5 Multiple choice

What is printed?

```
int balance = -20;
if (balance >= 0)
    if (balance > 1000)
        System.out.println("Premium");
else
    System.out.println("Overdrawn");
System.out.println("Done");
```

A. `Overdrawn` then `Done`
B. `Done`
C. `Premium` then `Done`
D. Nothing; an `else` cannot follow two `if`s, so the code does not compile.

Question 6 Short answer

For the code in Question 5, describe every value of `balance` that makes the program print `Overdrawn`. Explain how you know.

2.4.4 Multiway Selection with else if

A teacher wants to turn a score into a letter: A for 90 and up, B for 80 to 89, C for 70 to 79, D for 60 to 69. That is four possible outcomes, but an `if-else` only chooses between two. You could nest: if the score is not an A, ask whether it is a B; if not a B, ask about C; and so on. Each new question goes inside the previous `else`.

WORKED EXAMPLE **Letter grades as nested if-else statements**

Each `else` body holds the next `if-else`, so the code drifts to the right like a staircase:

```java
int score = 83;
if (score >= 90)
{
    System.out.println("Grade: A");
}
else
{
    if (score >= 80)
    {
        System.out.println("Grade: B");
    }
    else
    {
        if (score >= 70)
        {
            System.out.println("Grade: C");
        }
        else
        {
            if (score >= 60)
            {
                System.out.println("Grade: D");
            }
        }
    }
}
```

With `score` equal to `83`: `83 >= 90` is `false`, so Java enters the first `else`. `83 >= 80` is `true`, so it prints `Grade: B` and skips that inner `else`, along with everything nested inside it.

```
Grade: B
```

The staircase works, but it is hard to read. Look at each `else` body: it contains exactly one statement, the next `if-else`. Since a body with one statement does not need braces, Java lets you drop those braces and write the `else` and the next `if` on the same line. The result is the form programmers actually use:

```java
int score = 83;
if (score >= 90)
{
    System.out.println("Grade: A");
}
else if (score >= 80)
{
    System.out.println("Grade: B");
}
else if (score >= 70)
{
    System.out.println("Grade: C");
}
else if (score >= 60)
{
    System.out.println("Grade: D");
}
```

```
Grade: B
```

This is the same code with fewer braces, so it behaves exactly like the staircase. A chain of conditions written `if`, `else if`, `else if`, ... is called a **multiway selection** (also called an if-else-if statement). `else if` is not a new keyword: it is an `else` whose body is another `if`. That is why a multiway selection counts as a nested `if` statement.

How Java runs a multiway selection follows directly from the staircase:

1. Evaluate the conditions from top to bottom, one at a time.
2. At the **first** condition that is `true`, run its body, then skip the rest of the chain, including any conditions not yet checked.
3. If a condition is `false`, move on to the next `else if`.
4. If every condition is `false`, no body in the chain runs, and Java goes on to the statement after the chain.

Trace `score` equal to `83`:

Condition	Value	What happens
`83 >= 90`	`false`	Move to the next `else if`
`83 >= 80`	`true`	Print `Grade: B`; skip the rest of the chain
`83 >= 70`	not checked	Already finished
`83 >= 60`	not checked	Already finished

Notice that `83 >= 70` is also true, but that never matters, because Java stops at the first true condition. With `score` equal to `95`, the very first condition is `true`, so the output is only `Grade: A`, even though all four conditions would be true for 95. And with `score` equal to `52`, all four conditions are `false`, so this chain prints nothing at all.

KEY IDEA

In a multiway selection, Java checks the conditions in order and runs the body of the first one that is true, then skips the rest. At most one body in the chain runs.

WATCH OUT

Expecting every true condition to run its body. Students who see that `95 >= 80`, `95 >= 70`, and `95 >= 60` are all true sometimes predict four lines for a score of 95. In an `else if` chain, each later condition is reached only through an `else`, that is, only when every condition above it was `false`. Once one body runs, the rest of the chain is skipped. You will see code that really does print four lines in 2.4.5; the difference is that it uses separate `if` statements, not `else if`.

PAUSE AND TRY

Question 7 Tracing

Write the exact output. Then list the conditions Java evaluates, in order.

```
int battery = 35;
if (battery >= 80)
{
    System.out.println("Full");
}
else if (battery >= 50)
{
    System.out.println("Good");
}
else if (battery >= 20)
{
    System.out.println("Low");
}
else if (battery >= 5)
{
    System.out.println("Very low");
}
System.out.println("Status shown");
```

Question 8 Multiple choice

A multiway selection has an `if` followed by three `else if` parts, and each part's body prints one line. For any value of the variable being tested, how many of those lines can print?

A. Exactly one
B. Zero or one
C. Any number from zero to four
D. Exactly four

2.4.5 Order Matters in a Chain

Because Java stops at the first true condition, the order you write the conditions in is part of the program's meaning. Rearranging the lines of a chain can change its output even though each condition is still correct on its own.

WORKED EXAMPLE **The same four conditions in the wrong order**

This chain has exactly the same conditions and messages as the one in 2.4.4, listed from lowest cutoff to highest:

```
int score = 95;
if (score >= 60)
{
    System.out.println("Grade: D");
}
else if (score >= 70)
{
    System.out.println("Grade: C");
}
else if (score >= 80)
{
    System.out.println("Grade: B");
}
else if (score >= 90)
{
    System.out.println("Grade: A");
}
```

```
Grade: D
```

With `score` equal to `95`, the first condition `95 >= 60` is already `true`. Java prints `Grade: D` and skips the rest; the `score >= 90` test at the bottom is never reached. In fact every score of 60 or more lands in the first branch, so this chain can never print A, B, or C.

The fix is not "always put the biggest number first." What matters is that each condition, given that all the ones above it were `false`, describes exactly the values that belong in that branch. With `>=`, that means checking the highest cutoff first. With `<`, it means checking the lowest cutoff first. Here is a correct chain that starts from the bottom and uses `<`:

```java
int score = 95;
if (score < 60)
{
    System.out.println("Grade: F");
}
else if (score < 70)
{
    System.out.println("Grade: D");
}
else if (score < 80)
{
    System.out.println("Grade: C");
}
else if (score < 90)
{
    System.out.println("Grade: B");
}
else
{
    System.out.println("Grade: A");
}
```

```
Grade: A
```

Each `else if` is reached only when everything above it was `false`. So `score < 70` is checked only when `score < 60` was false, which means it catches exactly 60 to 69. A score of 95 fails all four `<` tests and reaches the final `else` (more on that in 2.4.6), which prints `Grade: A`.

KEY IDEA

Each condition in a chain is checked only when every condition above it was false. Order the conditions so that each one, given that, catches exactly the values meant for its branch.

WATCH OUT

Writing separate if statements instead of a chain. Leave out the word `else` and you no longer have one chain; you have four independent `if` statements, and Java checks every one of them:

```java
int score = 95;
if (score >= 90)
{
    System.out.println("Grade: A");
}
if (score >= 80)
{
    System.out.println("Grade: B");
}
if (score >= 70)
{
    System.out.println("Grade: C");
}
if (score >= 60)
{
    System.out.println("Grade: D");
}
```

```
Grade: A
Grade: B
Grade: C
Grade: D
```

Each `if` here starts fresh; nothing connects it to the ones before it, so all four true conditions run their bodies. Separate `if` statements are the right tool when several things should be able to happen. When exactly one category should be chosen, link the conditions with `else if`.

PAUSE AND TRY

Question 9 Debugging

Boxes are small for weights under 5 pounds, medium for under 10, and large for under 20. This code prints `Large box` for a 3-pound parcel. Explain the error and write the corrected chain.

```java
int weight = 3;
if (weight < 20)
{
    System.out.println("Large box");
}
else if (weight < 10)
{
    System.out.println("Medium box");
}
else if (weight < 5)
{
    System.out.println("Small box");
}
```

Question 10 Tracing

Write the exact output of code A and of code B. They differ only in the word `else`.

Code A:

```
int temperature = 72;
if (temperature > 60)
{
    System.out.println("Warm");
}
if (temperature > 40)
{
    System.out.println("Mild");
}
if (temperature > 20)
{
    System.out.println("Cool");
}
```

Code B:

```
int temperature = 72;
if (temperature > 60)
{
    System.out.println("Warm");
}
else if (temperature > 40)
{
    System.out.println("Mild");
}
else if (temperature > 20)
{
    System.out.println("Cool");
}
```

2.4.6 The Trailing else

Recall the score of 52 from 2.4.4. All four conditions (`52 >= 90`, `52 >= 80`, `52 >= 70`, `52 >= 60`) are `false`, so the letter-grade chain runs no body and prints nothing. A student who scored 52 gets no grade at all. The chain needs a way to say "and if none of those were true, do this."

That is what a plain `else` at the end of the chain does. A **trailing else** is an `else` with no condition, placed after the last `else if`. Its body runs exactly when every condition in the chain was `false`. Think of it as the "none of the above" choice on a form.

WORKED EXAMPLE **Adding an F with a trailing else**

The only change from 2.4.4 is the final `else` and its body:

```
int score = 52;
if (score >= 90)
{
    System.out.println("Grade: A");
}
else if (score >= 80)
{
    System.out.println("Grade: B");
}
else if (score >= 70)
{
    System.out.println("Grade: C");
}
else if (score >= 60)
{
    System.out.println("Grade: D");
}
else
{
    System.out.println("Grade: F");
}
```

```
Grade: F
```

All four conditions are `false` for 52, so Java reaches the trailing `else` and runs its body. For any score that makes one of the conditions true, Java stops at that branch and never reaches the `else`. With a trailing `else`, exactly one body in the chain runs, every time.

Now a chain where the trailing `else` does real work in the middle of the range. A city bus is free for children under 5, costs $1 for ages 5 to 17 and for ages 65 and up, and costs $2 for everyone else:

```java
int age = 40;
if (age < 5)
{
    System.out.println("Fare: free");
}
else if (age < 18)
{
    System.out.println("Fare: $1");
}
else if (age >= 65)
{
    System.out.println("Fare: $1");
}
else
{
    System.out.println("Fare: $2");
}
```

```
Fare: $2
```

For `age` equal to `40`: `40 < 5` is `false`, `40 < 18` is `false`, `40 >= 65` is `false`, so the trailing `else` runs. "Everyone else" here is ages 18 to 64, a group the code never has to describe with a condition; the `else` catches whatever is left. Some other ages: `3` prints `Fare: free`, `12` prints `Fare: $1`, `70` prints `Fare: $1`, and `18` prints `Fare: $2`.

A trailing `else` is optional. Leave it out when doing nothing is the right result. This weather program should warn only about strong wind:

```java
int windSpeed = 10;
if (windSpeed >= 74)
{
    System.out.println("Hurricane warning");
}
else if (windSpeed >= 39)
{
    System.out.println("Storm warning");
}
else if (windSpeed >= 25)
{
    System.out.println("Wind advisory");
}
System.out.println("Forecast complete");
```

```
Forecast complete
```

With `windSpeed` equal to `10`, all three conditions are `false` and there is no trailing `else`, so no warning prints, which is exactly right for a calm day. The statement after the chain runs either way.

Chain ends with	When no condition is true	Bodies that run
`else if (…)`	Nothing in the chain runs	zero or one
`else`	The trailing `else` body runs	exactly one

KEY IDEA

When no condition in a multiway selection is true, the trailing else body runs. Without a trailing else, nothing in the chain runs.

WATCH OUT

Putting the plain `else` before an `else if`. The trailing `else` has to be last. Once an `if` has its `else`, the `if` statement is complete, so another `else` cannot attach to it:

ElseInMiddle.java

```
 1 public class ElseInMiddle
 2 {
 3     public static void main(String[] args)
 4     {
 5         int age = 40;
 6         if (age < 5)
 7         {
 8             System.out.println("Fare: free");
 9         }
10         else
11         {
12             System.out.println("Fare: $2");
13         }
14         else if (age >= 65)
15         {
16             System.out.println("Fare: $1");
17         }
18     }
19 }
```

The compiler stops with:

```
ElseInMiddle.java:14: error: 'else' without 'if'
```

Line 14's `else if` has no `if` left to belong to. Move the plain `else` (and its body) to the end of the chain. A related mistake is giving the trailing `else` a condition, as in `else (age >= 65)`. An `else` never has a condition; if the branch needs one, write `else if (age >= 65)`.

PAUSE AND TRY

Question 11 Tracing

Write the exact output.

```java
int correct = 3;
if (correct >= 9)
{
    System.out.println("Expert");
}
else if (correct >= 6)
{
    System.out.println("Skilled");
}
else if (correct >= 4)
{
    System.out.println("Learning");
}
System.out.println("Quiz over");
```

Question 12 Debugging

Workouts of 60 minutes or more are long, 30 to 59 are medium, and anything shorter is short. This program does not compile. Explain why and write the corrected chain.

ElseFirstBug.java

```java
public class ElseFirstBug
{
    public static void main(String[] args)
    {
        int minutes = 25;
        if (minutes >= 60)
        {
            System.out.println("Long workout");
        }
        else
        {
            System.out.println("Short workout");
        }
        else if (minutes >= 30)
        {
            System.out.println("Medium workout");
        }
    }
}
```

Putting it together

A movie theater charges $6 for every matinee ticket, whatever the age. Evening tickets cost $8 for children under 12, $9 for ages 65 and up, and $12 for everyone else. The program reads an age and a show type, here from a string so the output is predictable (keyboard input works the same way, as in Topic 1.4):

TicketPrice.java

```
 1 import java.util.Scanner;
 2
 3 public class TicketPrice
 4 {
 5     public static void main(String[] args)
 6     {
 7         Scanner input = new Scanner("70 evening");
 8         int age = input.nextInt();
 9         String show = input.next();
10         int price;
11         if (show.equals("matinee"))
12         {
13             price = 6;
14         }
15         else
16         {
17             if (age < 12)
18             {
19                 price = 8;
20             }
21             else if (age >= 65)
22             {
23                 price = 9;
24             }
25             else
26             {
27                 price = 12;
28             }
29         }
30         System.out.println("Age " + age + ", " + show + " show");
31         System.out.println("Ticket: $" + price);
32     }
33 }
```

The structure is an outer `if-else` that asks about the show, with a multiway selection nested inside the `else` body. Trace it with the input `70 evening`:

Line	What Java does	Variables
7–9	Reads `70` into `age` and `evening` into `show`.	`age` 70, `show` "evening"
11	`show.equals("matinee")` is `false`. Skips lines 12 to 14 and enters the `else` body.	
17	Inner chain, first condition: `70 < 12` is `false`. Moves on.	
21	`70 >= 65` is `true`. Runs line 23 and skips the rest of the chain (the trailing `else`).	`price` 9

30	Prints the age and show.	
31	Prints the price.	

```
Age 70, evening show
Ticket: $9
```

Three ideas from this chapter made that work:

- **Nesting in the `else` body (2.4.2).** The age checks matter only for evening shows, so they sit inside the `else`. With the input `70 matinee`, line 11 is `true`, `price` becomes `6`, and none of the age conditions is evaluated.
- **First true condition wins (2.4.4, 2.4.5).** `70 >= 65` stopped the chain. Ages 12 to 64 fail both conditions and fall through.
- **The trailing `else` (2.4.6).** It catches everyone the two conditions did not, so `30 evening` costs $12 and `8 evening` costs $8. Because one body runs in every case, `price` always gets a value before line 31 prints it.

Vocabulary

Term	Definition	Example
nested if statement	An `if`, `if-else`, or multiway `if` statement placed inside the body of another one.	`if (t >= 60) { if (t <= 80) { … } }`
outer if statement	The `if` statement whose body contains another `if` statement.	`if (t >= 60)` above
inner if statement	The `if` statement inside another one's body; its condition is evaluated only when Java enters that body.	`if (t <= 80)` above
guard	An outer condition that makes sure the inner condition is safe to evaluate.	`if (count != 0) { if (total / count >= 80) … }`
dangling else	An `else` that could seem to belong to either of two `if`s; Java pairs it with the nearest unpaired `if`.	`if (a) if (b) x(); else y();`
multiway selection	A chain of `if`, `else if`, ... that runs the body of the first true condition and skips the rest.	`if (s >= 90) … else if (s >= 80) …`
trailing else	A plain `else` at the end of a chain; its body runs when no condition in the chain is true.	`else { System.out.println("Grade: F"); }`

Chapter summary

- An `if` statement can go inside the body of another `if` or `else`. This lets a program ask a follow-up question that depends on the answer to the first one, such as checking both ends of a range.
- An inner condition is evaluated only when Java enters the body that holds it: for an inner `if` in the `if` body, only when the outer condition is `true`. An outer condition can therefore guard an inner one that would otherwise crash.
- Without braces, an `else` belongs to the nearest earlier `if` that does not already have one, regardless of indentation. Braces on every body make the pairing clear.
- In a multiway selection (`if`, `else if`, ...), Java runs the body of the first true condition and skips the rest, so at most one body runs. Because of that, the order of the conditions changes the result, and separate `if` statements behave differently from a chain.
- A trailing `else` runs when no condition in the chain is true, so exactly one body runs. Without it, a chain can run nothing.

Review questions

Question 13 Tracing

Write the exact output. Then write the output when line 2 is changed to `boolean isVip = false;`.

```
1  int points = 42;
2  boolean isVip = true;
3  if (isVip)
4  {
5      if (points >= 50)
6      {
7          System.out.println("Gold");
8      }
9      else if (points >= 30)
10     {
11         System.out.println("Silver");
12     }
13     else
14     {
15         System.out.println("Bronze");
16     }
17 }
18 else
19 {
20     System.out.println("Standard");
21 }
22 System.out.println("Thanks");
```

Question 14 Short answer

This code prints `Too short`. The string `"AB"` has only two characters, so `code.substring(2, 3)` would throw a `StringIndexOutOfBoundsException` if it ran. Explain why the program does not crash.

```
String code = "AB";
if (code.length() >= 3)
{
    if (code.substring(2, 3).equals("X"))
    {
        System.out.println("Express code");
    }
    else
    {
        System.out.println("Regular code");
    }
}
else
{
    System.out.println("Too short");
}
```

Question 15 Debugging

A rewards app should print `Reward earned` for members with 100 or more points and `Join today` for anyone who is not a member. A member with 40 points sees `Join today`. Explain the error and write corrected code.

```
boolean isMember = true;
int points = 40;
if (isMember)
    if (points >= 100)
        System.out.println("Reward earned");
else
    System.out.println("Join today");
```

Question 16 Multiple choice

What is printed?

```java
int temperature = 85;
if (temperature > 80)
{
    System.out.println("Hot");
}
else if (temperature > 60)
{
    System.out.println("Warm");
}
if (temperature > 40)
{
    System.out.println("Mild");
}
else
{
    System.out.println("Cold");
}
```

A. `Hot`
B. `Hot` then `Mild`
C. `Hot`, `Warm`, then `Mild`
D. `Mild`

Question 17 Short answer

`day` holds a number from 1 to 7. One student prints the day's name with seven separate `if` statements, `if (day == 1)`, `if (day == 2)`, and so on. Another uses one chain, `if (day == 1)`, `else if (day == 2)`, and so on. Will the two versions ever print different output? What does Java do differently when they run?

Question 18 Multiple choice

In the bus fare program from 2.4.6, a student moves the trailing `else` up so the chain becomes `if (age < 5)`, then `else` (`Fare: $2`), then `else if (age < 18)`, then `else if (age >= 65)`. What happens?

A. It compiles, and every passenger 5 or older pays $2.
B. It compiles, and the fares are the same as before.
C. It does not compile, because once the `if` has its plain `else`, the `else if` that follows has no `if` to belong to.
D. It compiles, but the program crashes when `age` is 18.

Reflection

☐ I can trace an if inside another if and say which conditions Java evaluates. Reread: 2.4.1 An if Inside Another if

- [ ] I can explain how an outer condition guards an inner one that would otherwise crash. Reread: Why "never evaluated" matters
- [ ] I can trace an if-else with a nested if in the if body, the else body, or both. Reread: 2.4.2 Nesting in Either Branch
- [ ] I can pair each else with the right if in code without braces, ignoring the indentation. Reread: 2.4.3 Which if an else Belongs To
- [ ] I can trace a multiway selection and explain why at most one body runs. Reread: 2.4.4 Multiway Selection with else if
- [ ] I can tell whether the conditions in a chain are in the right order, and how a chain differs from separate ifs. Reread: 2.4.5 Order Matters in a Chain
- [ ] I can say what a trailing else does and when a chain should have one. Reread: 2.4.6 The Trailing else