UNIVERSITY OF WASHINGTON

Building Java Programs

Chapter 4:
Conditional Execution

Lecture outline

Lecture 9

- **conditional execution**
  - **the `if` statement and the `if/else` statement**
  - **relational expressions**
  - **nested `if/else` statements**

Lecture 10

- subtleties of conditional execution
  - factoring `if/else` code
- fencepost loops
- methods with conditional execution
  - revisiting return values

if/else statements

- suggested reading: 4.2

The if statement

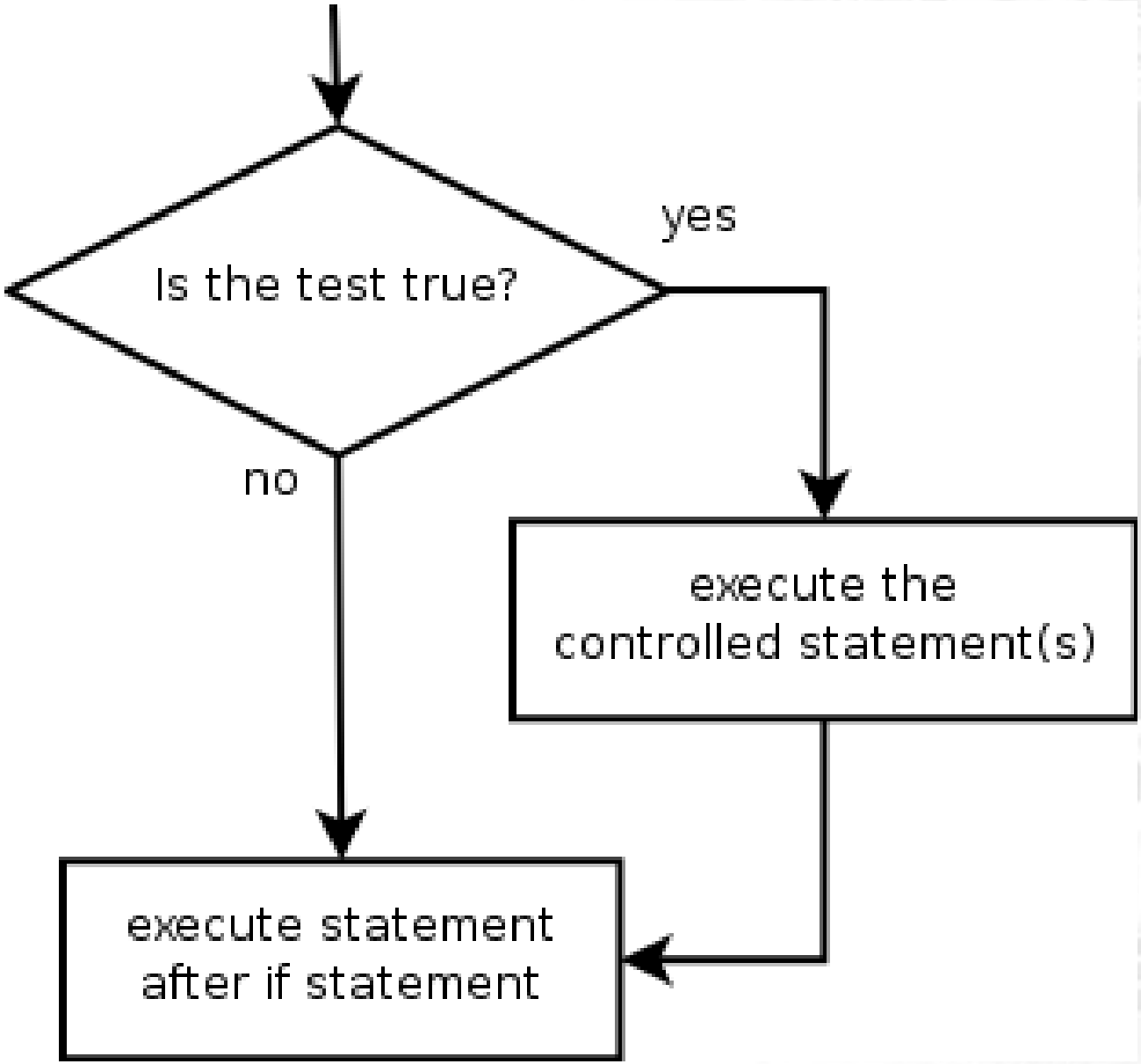
- **`if` statement**: A Java statement that executes a block of statements only if a certain condition is true.
  - If the condition is not true, the block of statements is skipped.
  - General syntax:

```
if (<condition>) {
    <statement> ;
    <statement> ;
    ...
    <statement> ;
}
```

- Example:

```
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Your application is accepted.");
}
```

if statement flow diagram



The if/else statement

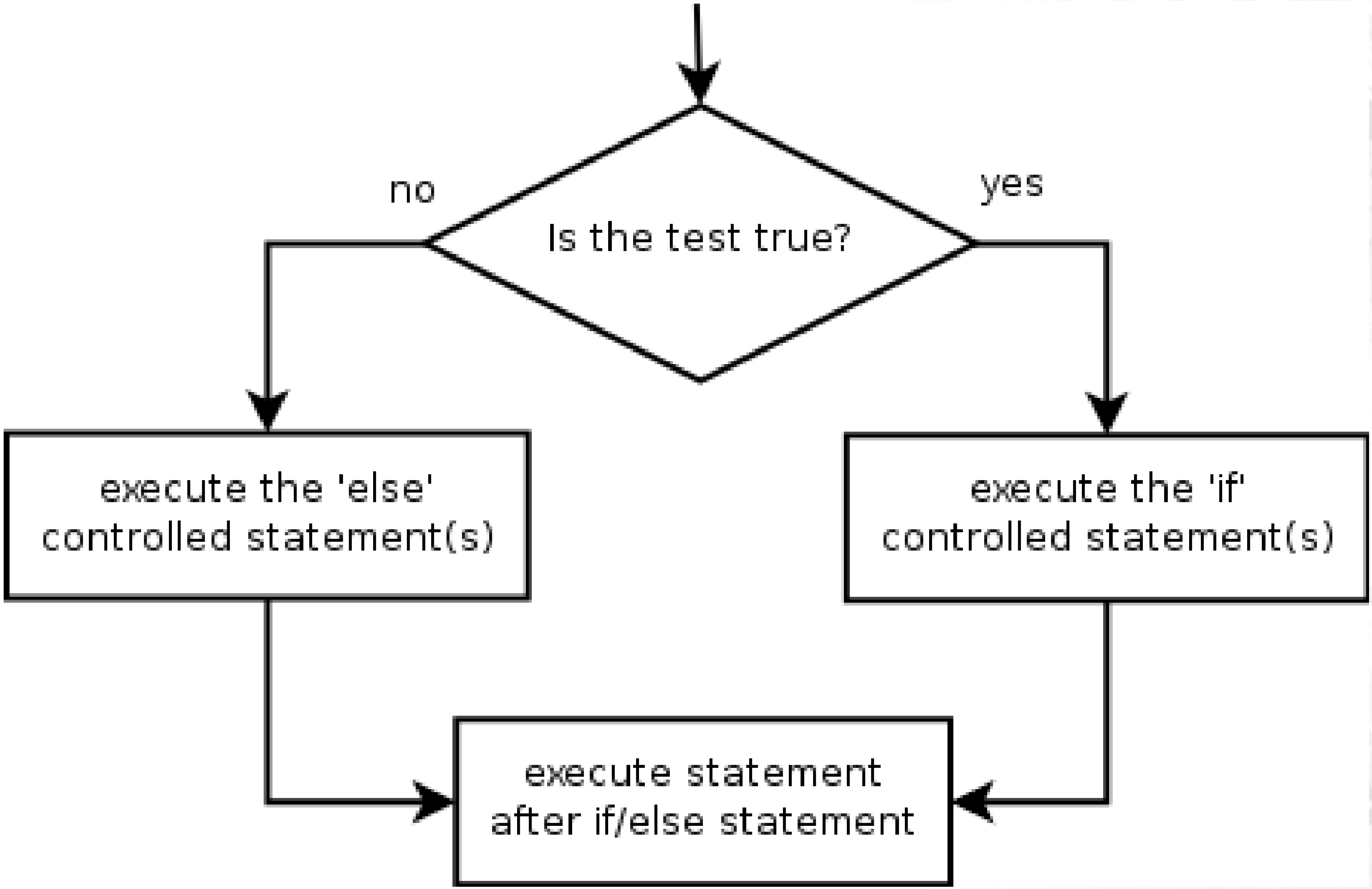
- **`if/else` statement**: A Java statement that executes one block of statements if a certain condition is true, and a second block of statements if it is false.
  - General syntax:

```
if (<condition>) {
    <statement(s)> ;
} else {
    <statement(s)> ;
}
```

- Example:

```
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Welcome to Mars University!");
} else {
    System.out.println("Your application is denied.");
}
```

if/else flow diagram



Relational expressions

- The ***<condition>*** used in an `if` or `if/else` statement is the same kind seen in a `for` loop.

```
for (int i = 1; i <= 10; i++) {
```

  - The conditions are actually of type `boolean`, seen in Ch. 5.

- These conditions are called **relational expressions** and use one of the following six **relational operators**:

Operator	Meaning	Example	Value
`==`	equals	`1 + 1 == 2`	`true`
`!=`	does not equal	`3.2 != 2.5`	`true`
`<`	less than	`10 < 5`	`false`
`>`	greater than	`10 > 5`	`true`
`<=`	less than or equal to	`126 <= 100`	`false`
`>=`	greater than or equal to	`5.0 >= 5.0`	`true`

Evaluating rel. expressions

- Relational operators have lower precedence than math operators.
  - Example:

```
5 * 7 >= 3 + 5 * (7 - 1)
5 * 7 >= 3 + 5 * 6
35    >= 3 + 30
35    >= 33
true
```

- Relational operators cannot be "chained" as they can in algebra.
  - Example:

```
2 <= x <= 10
true   <= 10
   error!
```

if/else question

- Write code to read a number from the user and print whether it is even or odd using an `if/else` statement.

  - Example executions:

```
Type a number: 42
Your number is even

Type a number: 17
Your number is odd
```

Loops with if/else

- Loops can be used with `if/else` statements:

```
int nonnegatives = 0, negatives = 0;
for (int i = 1; i <= 10; i++) {
    int next = console.nextInt();
    if (next >= 0) {
        nonnegatives++;
    } else {
        negatives++;
    }
}
```

```
public static void printEvenOdd(int max) {
    for (int i = 1; i <= max; i++) {
        if (i % 2 == 0) {
            System.out.println(i + " is even");
        } else {
            System.out.println(i + " is odd");
        }
    }
}
```

Nested if/else statements

- **Nested if/else statement**: A chain of `if/else` that can select between many different outcomes based on several conditions.
  - General syntax:

```
if (<condition>) {
    <statement(s)> ;
} else if (<condition>) {
    <statement(s)> ;
} else {
    <statement(s)> ;
}
```

- Example:

```
if (number > 0) {
    System.out.println("Positive");
} else if (number < 0) {
    System.out.println("Negative");
} else {
    System.out.println("Zero");
}
```

Nested if/else variations

- A nested `if/else` can end with an `if` or an `else`.
  - If it ends with `else`, one of the code paths must be taken.
  - If it ends with `if`, the program might not execute any path.

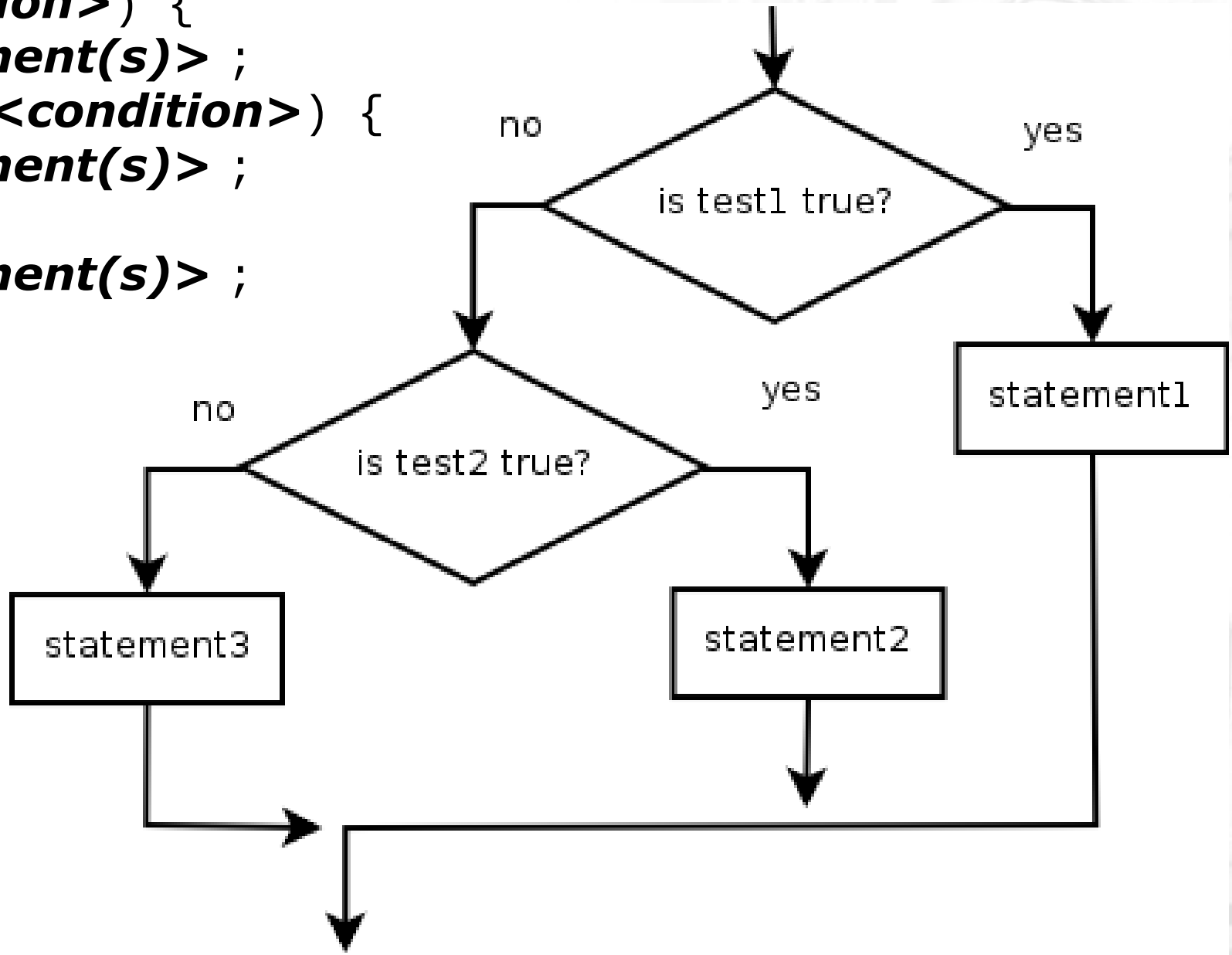
- Example ending with `else`:

```
if (place == 1) {
    System.out.println("You win the gold medal!");
} else if (place == 2) {
    System.out.println("You win a silver medal!");
} else if (place == 3) {
    System.out.println("You earned a bronze medal.");
}
```

  - Are there any cases where this code will not print a message?
  - How could we modify it to print a message to non-medalists?

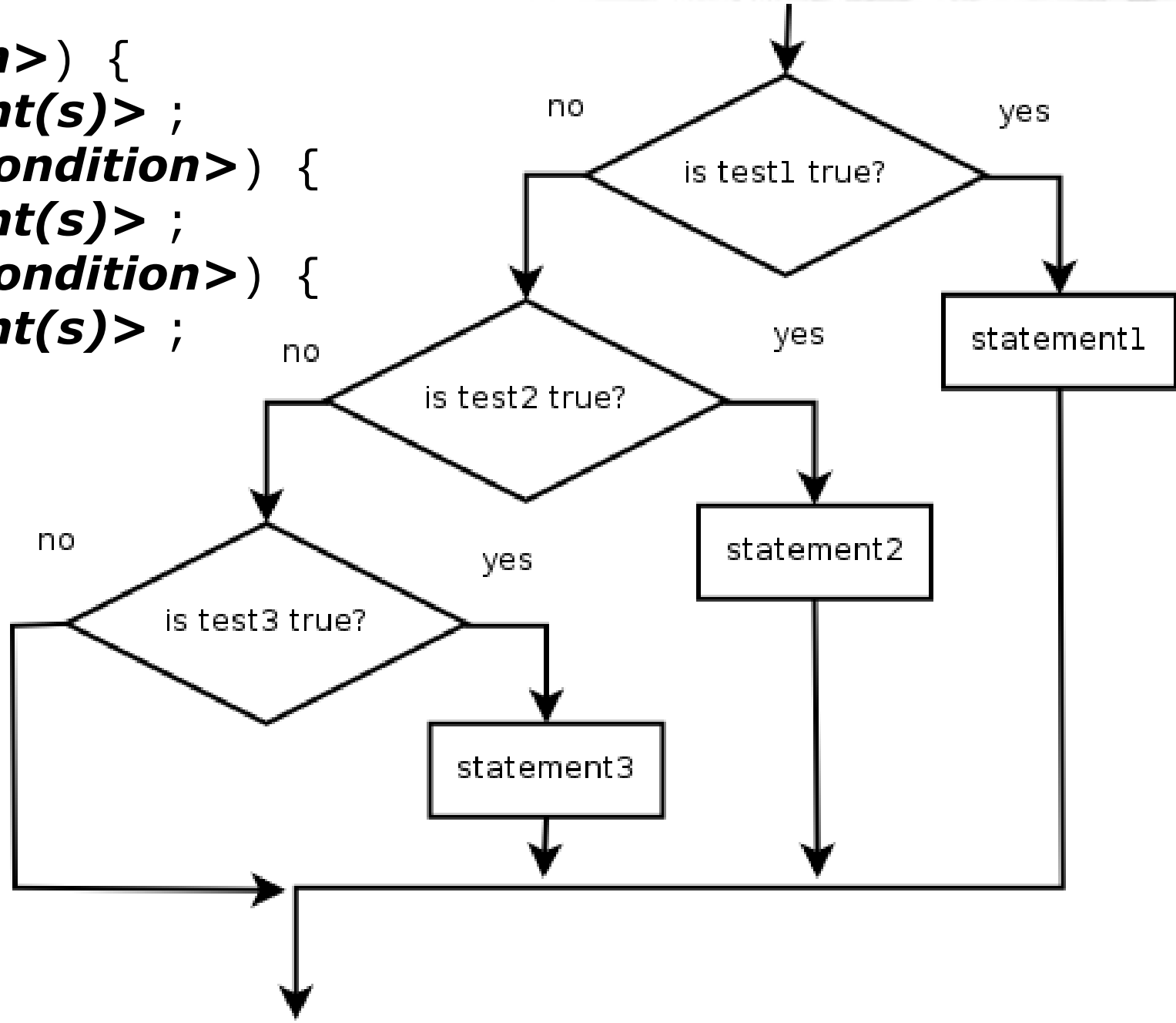
Nested if/else flow diagram

```
if (<condition>) {
    <statement(s)> ;
} else if (<condition>) {
    <statement(s)> ;
} else {
    <statement(s)> ;
}
```



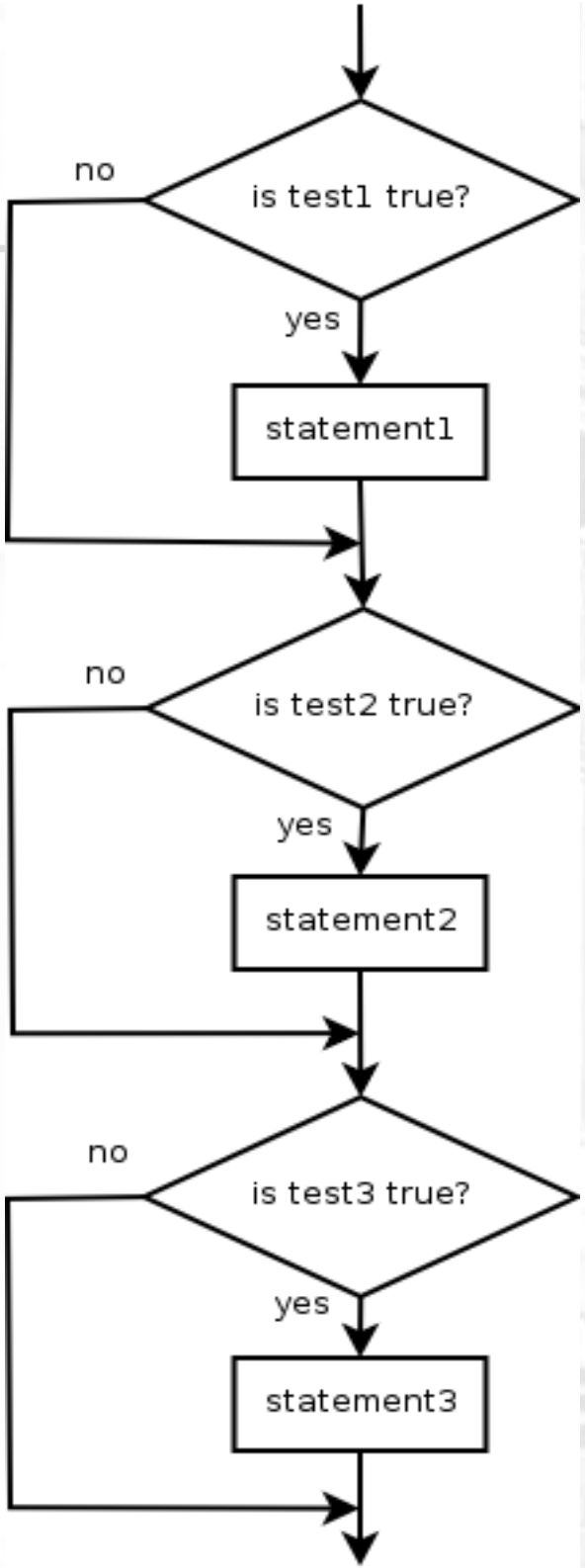
Nested if/else/if flow diagram

```
if (<condition>) {
    <statement(s)> ;
} else if (<condition>) {
    <statement(s)> ;
} else if (<condition>) {
    <statement(s)> ;
}
```



Sequential if flow

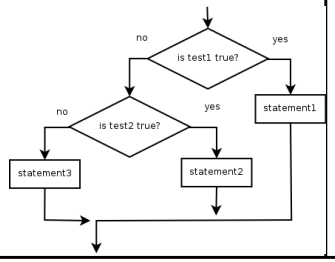
```
if (<condition>) {
    <statement(s)> ;
}
if (<condition>) {
    <statement(s)> ;
}
if (<condition>) {
    <statement(s)> ;
}
```



Structures of if/else code

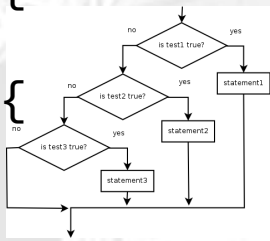
- Choose 1 of many paths:
  (conditions are mutually exclusive)

```
if (<condition>) {
    <statement(s)>;
} else if (<condition>) {
    <statement(s)>;
} else {
    <statement(s)>;
}
```



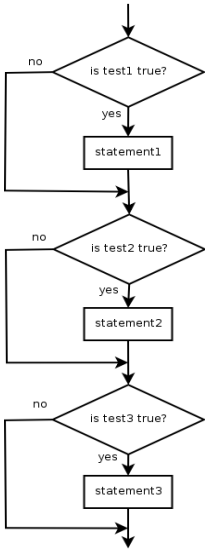
- Choose 0 or 1 of many paths:
  (conditions are mutually exclusive and any action is optional)

```
if (<condition>) {
    <statement(s)>;
} else if (<condition>) {
    <statement(s)>;
} else if (<condition>) {
    <statement(s)>;
}
```



- Choose 0, 1, or many of many paths:
  (conditions/actions are independent of each other)

```
if (<condition>) {
    <statement(s)>;
}
if (<condition>) {
    <statement(s)>;
}
if (<condition>) {
    <statement(s)>;
}
```



Which nested if/else to use?

- Which if/else construct is most appropriate to perform each of the following tasks?
  - Reading the user's GPA and printing whether the student is on the dean's list (3.8 to 4.0) or honor roll (3.5 to 3.8).
  - Printing whether a number is even or odd.
  - Printing whether a user is lower-class, middle-class, or upper-class based on their income.
  - Reading a number from the user and printing whether it is divisible by 2, 3, and/or 5.
  - Printing a user's grade of A, B, C, D, or F based on their percentage in the course.

Which nested if/else to use?

- Which if/else construct is most appropriate to perform each of the following tasks?
  - Reading the user's GPA and printing whether the student is on the dean's list (3.8 to 4.0) or honor roll (3.5 to 3.8).
    - **nested** `if / else if`
  - Printing whether a number is even or odd.
    - **simple** `if / else`
  - Printing whether a user is lower-class, middle-class, or upper-class based on their income.
    - **nested** `if / else if / else`
  - Reading a number from the user and printing whether it is divisible by 2, 3, and/or 5.
    - **sequential** `if / if / if`
  - Printing a user's grade of A, B, C, D, or F based on their percentage in the course.
    - **nested** `if / else if / else if / else if / else`

How to comment: if/else

- Comments on an if statement don't need to describe exactly what the if statement is testing.
  - Instead, they should describe why you are performing that test, and/or what you intend to do based on its result.
  - Bad example:

```
// Test whether student 1's GPA is better than student 2's
if (gpa1 > gpa2) {
    // print that student 1 had the greater GPA
    System.out.println("The first student had the greater GPA.");
} else if (gpa2 > gpa1) {
    // print that student 2 had the greater GPA
    System.out.println("The second student's GPA was higher.");
} else {  // there was a tie
    System.out.println("There has been a tie!");
}
```

  - Better example:

```
// Print a message about which student had the higher grade point average.
if (gpa1 > gpa2) {
    System.out.println("The first student had the greater GPA.");
} else if (gpa2 > gpa1) {
    System.out.println("The second student's GPA was higher.");
} else {  // gpa1 == gpa2 (a tie)
    System.out.println("There has been a tie!");
}
```

How to comment: if/else 2

- If an if statement's test is straightforward, and if the actions to be taken in the bodies of the if/else statement are very different, sometimes putting comments on the bodies themselves is more helpful.
  - Example:

```
if (guessAgain == 1) {
    // user wants to guess again; reset game state and
    // play another game
    System.out.println("Playing another game.");
    score = 0;
    resetGame();
    play();
} else {
    // user is finished playing; print their best score
    System.out.println("Thank you for playing.");
    System.out.println("Your score was " + score);
}
```

Math.max/min vs. if/else

- Many `if/else` statements that choose the larger or smaller of 2 numbers can be replaced by a call to `Math.max` or `Math.min`.
  - ```
    int z;          // z should be larger of x, y
    if (x > y) {
        z = x;
    } else {
        z = y;
    }
    ```
  - ```
    int z = Math.max(x, y);
    ```
  - ```
    double d = a;  // d should be smallest of a, b, c
    if (b < d) {
        d = b;
    }
    if (c < d) {
        d = c;
    }
    ```
  - ```
    double d = Math.min(a, Math.min(b, c));
    ```

Lecture outline

Lecture 9

- conditional execution
  - the `if` statement and the `if/else` statement
  - relational expressions
  - nested `if/else` statements

Lecture 10

- **subtleties of conditional execution**
  - **factoring `if/else` code**
- **fencepost loops**
- **methods with conditional execution**
  - **revisiting return values**

Subtleties of conditional execution

- suggested reading: 4.3

Factoring if/else code

- **factoring**: extracting common/redundant code
  - Factoring `if/else` code reduces the size of the `if` and `else` statements and can sometimes eliminate the need for `if/else` altogether.

- Example:

```
int x;
if (a == 1) {
    x = 3;
} else if (a == 2) {
    x = 5;
} else {  // a == 3
    x = 7;
}
```

→

```
int x = 2 * a + 1;
```

Code in need of factoring

- The following example has a lot of redundant code in the `if/else`:

```
if (money < 500) {
    System.out.println("You have, $" + money + " left.");
    System.out.print("Caution!  Bet carefully.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
} else if (money < 1000) {
    System.out.println("You have, $" + money + " left.");
    System.out.print("Consider betting moderately.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
} else {
    System.out.println("You have, $" + money + " left.");
    System.out.print("You may bet liberally.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
}
```

Code after factoring

- Factoring tips:
  - If the start of each branch is the same, move it *before* the `if/else`.
  - If the end of each branch is the same, move it *after* the `if/else`.

```
System.out.println("You have, $" + money + " left.");

if (money < 500) {
    System.out.print("Caution!  Bet carefully.");
} else if (money < 1000) {
    System.out.print("Consider betting moderately.");
} else {
    System.out.print("You may bet liberally.");
}

System.out.print("How much do you want to bet? ");
bet = console.nextInt();
```

Fencepost loops

- suggested reading: 4.1

The fencepost problem

- Problem: Write a static method named `printNumbers` that prints each number from 1 to a given maximum, separated by commas.

  For example, the method call:

  ```
  printNumbers(5)
  ```

  should print:

  ```
  1, 2, 3, 4, 5
  ```

- Let's write a solution to this problem...

Flawed solution 1

- A flawed solution:

```
public static void printNumbers(int max) {
    for (int i = 1; i <= max; i++) {
        System.out.print(i + ", ");
    }
    System.out.println();  // to end the line of output
}
```

- Output from `printNumbers(5)`:

```
1, 2, 3, 4, 5,
```

Flawed solution 2

- Another flawed solution:

```
public static void printNumbers(int max) {
    for (int i = 1; i <= max; i++) {
        System.out.print(", " + i);
    }
    System.out.println();  // to end the line of output
}
```

- Output from `printNumbers(5)`:

```
, 1, 2, 3, 4, 5
```

Fence post analogy

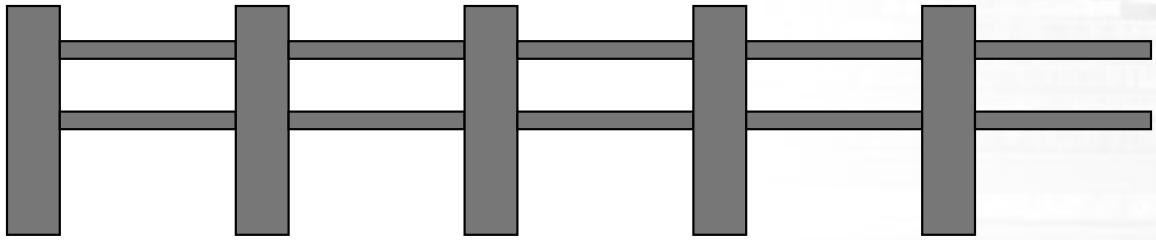
- We print *n* numbers but need only *n* - 1 commas.
- This problem is similar to the task of building a fence with lengths of wire separated by posts.
  - often called a *fencepost problem*
  - If we repeatedly place a post and wire, the last post has an extra dangling wire.

  - A flawed algorithm:

    for (length of fence):

    place some post.

    place some wire.



Fencepost loop

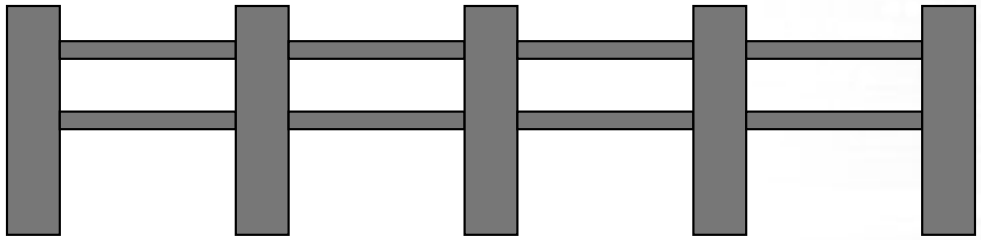
- The solution is to add an extra statement outside the loop that places the inital "post."
  - This is sometimes also called a *fencepost loop* or a "loop-and-a-half" solution.
  - The revised algorithm:

    place a post.

    *for (length of fence **- 1**):*

    place some wire.

    place some post.



Fencepost method solution

- A version of `printNumbers` that works:

```
public static void printNumbers(int max) {
    System.out.print(1);
    for (int i = 2; i <= max; i++) {
        System.out.print(", " + i);
    }
    System.out.println();  // to end the line of output
}
```

OUTPUT from `printNumbers(5)`:

```
1, 2, 3, 4, 5
```

Fencepost practice problem

- Write a method named `printFactors` that, when given a number, prints its factors in the following format (using an example of 24 for the parameter value):

```
[1, 2, 3, 4, 6, 8, 12, 24]
```

Fencepost practice problem

- Write a Java program that reads a base and a maximum power and prints all of the powers of the given base up to that max, separated by commas.

```
Base: 2
Max exponent: 9

The first 9 powers of 2 are:
2, 4, 8, 16, 32, 64, 128, 256, 512
```

Methods with if/else

- suggested reading: 4.5

if/else with return

- Methods can be written to return different values under different conditions using `if/else` statements:

```
public static int min(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```

- An example that maps chess board squares to colors:

```
public static Color chessBoardColor(int row, int column) {
    if ((row + column) % 2 == 0) {
        return Color.WHITE;
    } else {
        return Color.BLACK;
    }
}
```

More examples

- Another example that returns the first word in a string:

```
public static String firstWord(String s) {
    int index = s.indexOf(" ");
    if (index >= 0) {
        return s.substring(0, index);
    } else {   // only one word in String
        return s;
    }
}
```

- It is an error not to return a value in every path:

```
public static int min(int a, int b) {
    if (a > b) {
        return b;
    }
    // Error; not all code paths return a value.
    // What if a <= b ?
}
```

All code paths must return

- The following code does not compile:

```
public static int min(int a, int b) {
    if (a >= b) {
        return b;
    } else if (a < b) {
        return a;
    }
}
```

- It produces the "Not all paths return a value" error.
  - To our eyes, it is clear that all paths (greater, equal, less) do return a value.
  - But the compiler thinks that `if/else/if` code might choose not to execute any branch, so it refuses to accept this code.
  - How can we fix it?

for loops with if/else return

- Methods with loops that return values must consider the case where the loop does not execute the return.

```
public static int indexOf(String s, char c) {
    for (int i = 0; i < s.length(); i++) {
        if (s.charAt(i) == c) {
            return i;
        }
    }
    // error; what if c does not occur in s?
}
```

- A better version that returns -1 when c is not found:

```
public static int indexOf(String s, char c) {
    for (int i = 0; i < s.length(); i++) {
        if (s.charAt(i) == c) {
            return i;
        }
    }
    return -1;   // not found
}
```

if/else return question

- Write a method named `numUnique` that accepts two integers as parameters and returns how many unique values were passed.
  - For example, `numUnique(3, 7)` returns `2` because 3 and 7 are two unique numbers, but `numUnique(4, 4)` returns `1` because 4 and 4 only represent one unique number.
- Write a method named `countFactors` that returns the number of factors of an integer.
  - For example, `countFactors(60)` returns `11` because 1, 2, 3, 4, 5, 6, 10, 15, 20, 30, and 60 are factors of 60.
-

Method return question

- Write a program that prompts the user for a maximum integer and prints out a list of all prime numbers up to that maximum. Here is an example log of execution:

```
Maximum number? 50
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47
14 total primes
```