Building Java Programs

Chapter 4: Conditional Execution

Chapter outline

- loop techniques
  - cumulative sum
  - fencepost loops

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

- subtleties of conditional execution
  - object equality
  - factoring `if/else` code
  - text processing
  - methods with conditional execution: revisiting return values

Cumulative sum

reading: 4.1

Adding many numbers

- How would you write code to find the sum of all integers from 1-1000?

```
int sum = 1 + 2 + 3 + 4 + ... ;
System.out.println("The sum is " + sum);
```

- What if we want the sum of integers from 1-1,000,000? Or to compute the sum up to any maximum?
  - We could write a method that accepts the maximum value as a parameter and returns the sum.
  - How can we generalize code like the above?

A failed attempt

- An incorrect solution for summing 1-100:

```
for (int i = 1; i <= 100; i++) {
    int sum = 0;
    sum = sum + i;
}

// sum is undefined here
System.out.println("The sum is " + sum);
```

  - The scope of `sum` is inside the `for` loop, so the last line of code fails to compile.

- **cumulative sum**: A variable that keeps a sum-in-progress and is updated until summing is finished.
  - The `sum` in the above code is an attempt at a cumulative sum.

Fixed cumulative sum loop

- A corrected version of the sum loop code:

```
int sum = 0;
for (int i = 1; i <= 100; i++) {
    sum = sum + i;
}
System.out.println("The sum is " + sum);
```

The key idea:

- Cumulative sum variables must always be declared *outside* the loops that update them, so that they will continue to live after the loop is finished.

Cumul. sum exercises

- Write a method named `sumSeries` that accepts an integer parameter *k* and computes the sum of the first *k* terms of the following series:
  - 1 + 1/2 + 1/4 + 1/8 + ...

- Write a method named `pow2` that accepts an integer parameter *n* and computes 2^n.

- Write a method named `pow` that accepts integers for a base *a* and an exponent *b* and computes a^b.

Cumul. sum and Scanner

- Consider this code to read and add three values:

```
Scanner console = new Scanner(System.in);
System.out.print("Type a number: ");
int num1 = console.nextInt();

System.out.print("Type a number: ");
int num2 = console.nextInt();

System.out.print("Type a number: ");
int num3 = console.nextInt();

int sum = num1 + num2 + num3;
System.out.println("The sum is " + sum);
```

A cumulative sum

- The variables num1, num2, and num3 are unnecessary:

```
Scanner console = new Scanner(System.in);
System.out.print("Type a number: ");
int sum = console.nextInt();

System.out.print("Type a number: ");
sum += console.nextInt();

System.out.print("Type a number: ");
sum += console.nextInt();

System.out.println("The sum is " + sum);
```

  - The variable sum in the above code is also a cumulative sum.
  - What if we wanted to read and sum 100 numbers?

Fixed cumulative sum loop

- We can use a cumulative sum loop here as well:

```
Scanner console = new Scanner(System.in);
int sum = 0;
for (int i = 1; i <= 100; i++) {
    System.out.print("Type a number: ");
    sum += console.nextInt();
}
System.out.println("The sum is " + sum);
```

User-guided cumulative sum

- User input can control the number of loop repetitions:
  - Desired example output:
    ```
    How many numbers to add? 3
    Type a number: 2
    Type a number: 6
    Type a number: 3
    The sum is 11
    ```
  - Answer:
    ```
    Scanner console = new Scanner(System.in);
    System.out.print("How many numbers to add? ");
    int count = console.nextInt();

    int sum = 0;
    for (int i = 1; i <= count; i++) {
        System.out.print("Type a number: ");
        sum += console.nextInt();
    }
    System.out.println("The sum is " + sum);
    ```

Variation: cumulative product

- The same idea can be used with other operators, such as multiplication which produces a cumulative product:

```
Scanner console = new Scanner(System.in);
System.out.print("Raise 2 to what power? ");
int exponent = console.nextInt();

int product = 1;
for (int i = 1; i <= exponent; i++) {
    product = product * 2;
}
System.out.println("2 to the " + exponent + " = " + product);
```

- Exercises:
  - Change the above code so that it also prompts for the base, instead of always using 2.
  - Change the above code into a method which accepts a base *a* and exponent *b* as parameters and returns a^b.

Cumulative sum question

- Write a program that reads input of the number of hours two employees have worked and displays each employee's total and the overall total hours.
  - The company doesn't pay overtime, so cap any day at 8 hours.
- Example log of execution:

```
Employee 1: How many days? 3
Hours? 6
Hours? 12
Hours? 5
Employee 1's total hours = 19

Employee 2: How many days? 2
Hours? 11
Hours? 6
Employee 2's total hours = 14

Total hours for both = 33
```

Cumulative sum answer

```
// Computes the total paid hours worked by two employees.
// The company does not pay for more than 8 hours per day.
// Uses a "cumulative sum" loop to compute the total hours.

import java.util.*;

public class Hours {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        int hours1 = processEmployee(console, 1);
        int hours2 = processEmployee(console, 2);

        int total = hours1 + hours2;
        System.out.println("Total hours for both = " + total);
    }

    ...
```

Cumulative sum answer 2

```
    ...

    // Reads hours information about one employee with the given number.
    // Returns the total hours worked by the employee.
    public static int processEmployee(Scanner console, int number) {
        System.out.print("Employee " + number + ": How many days? ");
        int days = console.nextInt();

        // totalHours is a cumulative sum of all days' hours worked.
        int totalHours = 0;
        for (int i = 1; i <= days; i++) {
            System.out.print("Hours? ");
            int hours = console.nextInt();
            totalHours += Math.min(hours, 8);   // cap at 8 hours/day
        }

        System.out.println("Employee " + number + "'s total hours = "
                           + totalHours);
        System.out.println();
        return totalHours;
    }
}
```

Fencepost loops

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
  ```

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
}
```

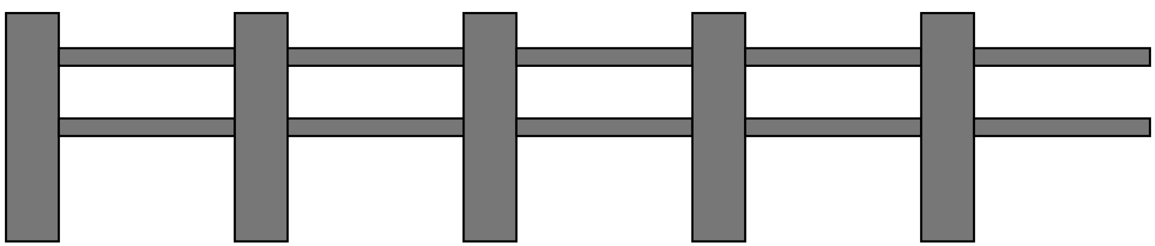
- Output from `printNumbers(5)`:

```
, 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 will have 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:

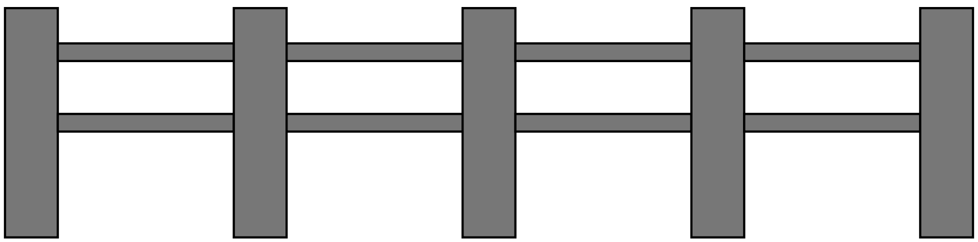
    place a post.

    *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 question

- 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 question

- 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
```

if/else statements

reading: 4.2

The if statement

- **if statement**: Executes a block of statements only if a certain condition is true.
  - Otherwise, the block of statements is skipped.
  - General syntax:

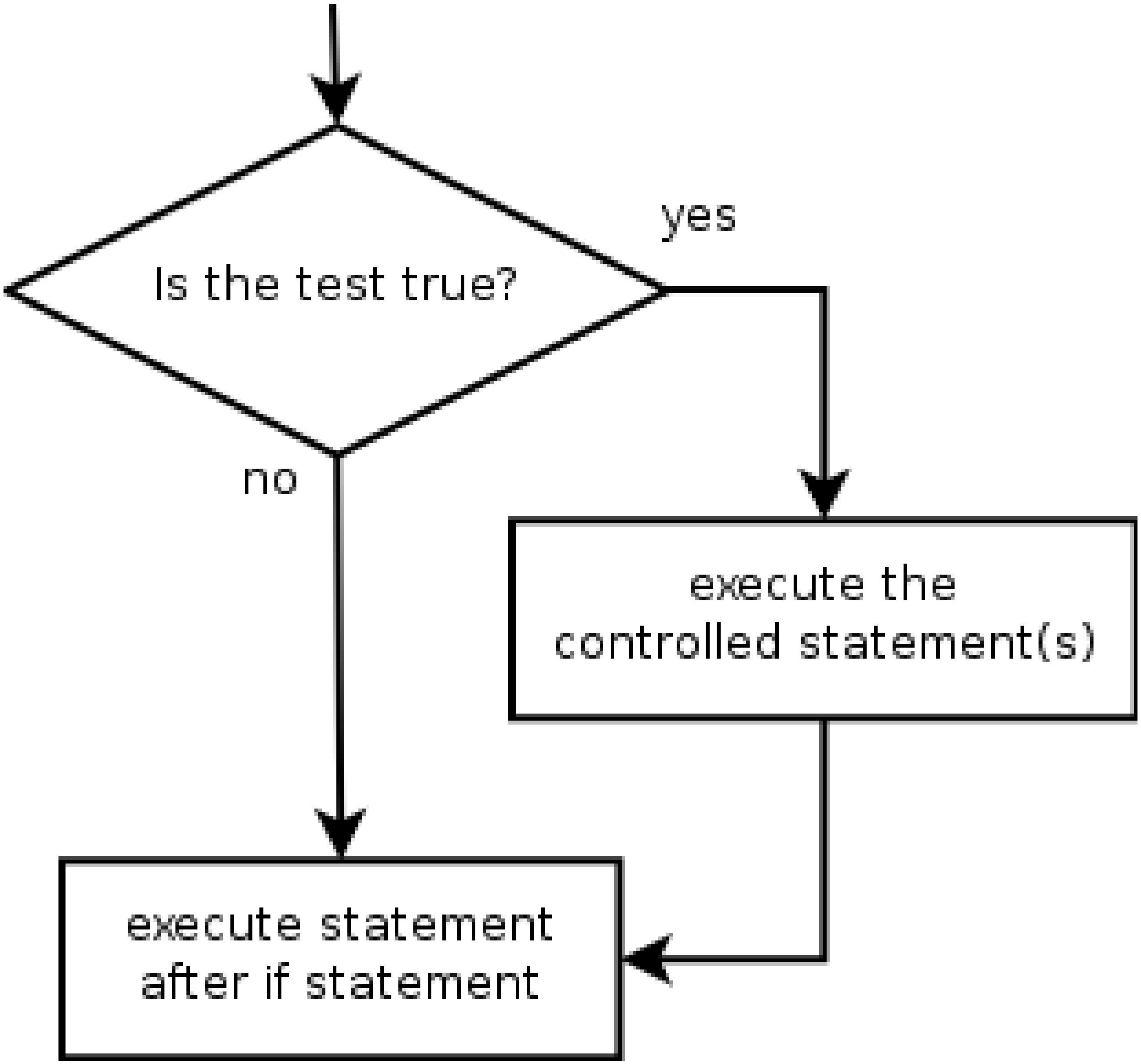
```
if (<condition>) {
    <statement> ;
    <statement> ;
    ...
    <statement> ;
}
```

- Example:

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

if statement flow diagram

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



The if/else statement

- **if/else statement**: Executes one block of statements if a certain condition is true, and another if it is false.
  - General syntax:

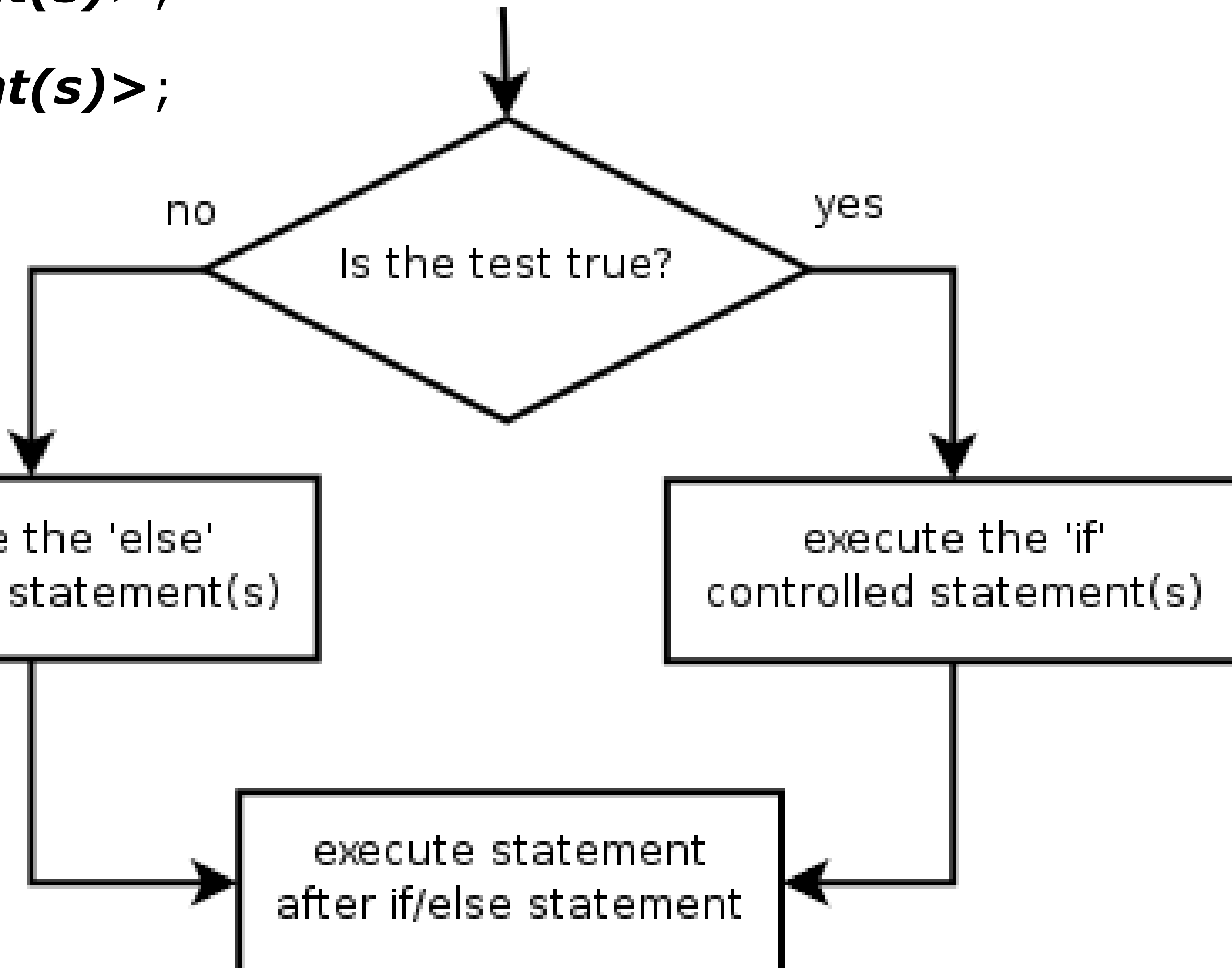
```
if (<condition>) {
    <statement(s)> ;
} else {
    <statement(s)> ;
}
```

- 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

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



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 the following *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`

Logical operators && || !

- Conditions can be combined using *logical operators*:

Operator	Description	Example	Result
`&&`	and	`(9 != 6) && (2 < 3)`	`true`
`||`	or	`(2 == 3) || (-1 < 5)`	`true`
`!`	not	`!(7 > 0)`	`false`

- "Truth tables" for each logical operator, when used with logical values *p* and *q*:

p	q	p && q	p || q
`true`	`true`	`true`	`true`
`true`	`false`	`false`	`true`
`false`	`true`	`false`	`true`
`false`	`false`	`false`	`false`

p	!p
`true`	`false`
`false`	`true`

Evaluating rel. expressions

- Relational operators have lower precedence than math operators.

```
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.

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

(assume that x is 15)

- Instead, combine multiple tests with && or ||

```
2 <= x && x <= 10
true   && false
```

(assume that x is 15)

Logical questions

- What is the result of each of the following expressions?

```
int x = 42;
int y = 17;
int z = 25;
```

  - `y < x && y <= z`
  - `x % 2 == y % 2 || x % 2 == z % 2`
  - `x <= y + z && x >= y + z`
  - `!(x < y && x < z)`
  - `(x + y) % 2 == 0 || !((z - y) % 2 == 0)`

- Answers: `true`, `false`, `true`, `true`, `false`

if/else questions

- 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
```

- Write code to read ten numbers and print how many were negative and non-negative, and the sum of both.
  - Example execution:

```
Type ten numbers: 2 1 -4 7 -19 3 5 -8 -1 6
4 negative, 6 non-negative
negative sum -32, non-negative sum 24
```

Loops with if/else

- if/else statements can be used with loops or methods:

```
int evens = 0,  odds = 0;
for (int i = 1; i <= 10; i++) {
    int next = console.nextInt();
    if (next % 2 == 0) {
        evens++;
    } else {
        odds++;
    }
}
```

```
public static void printEvenOdd(int min, int max) {
    for (int i = min; i <= max; i++) {
        if (i < 0) {
            System.out.println(i + " is negative");
        } else {
            System.out.println(i + " is non-negative");
        }
    }
}
```

Nested if/else statements

- **nested if/else statement**: A chain of `if/else` that chooses between outcomes using many 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.

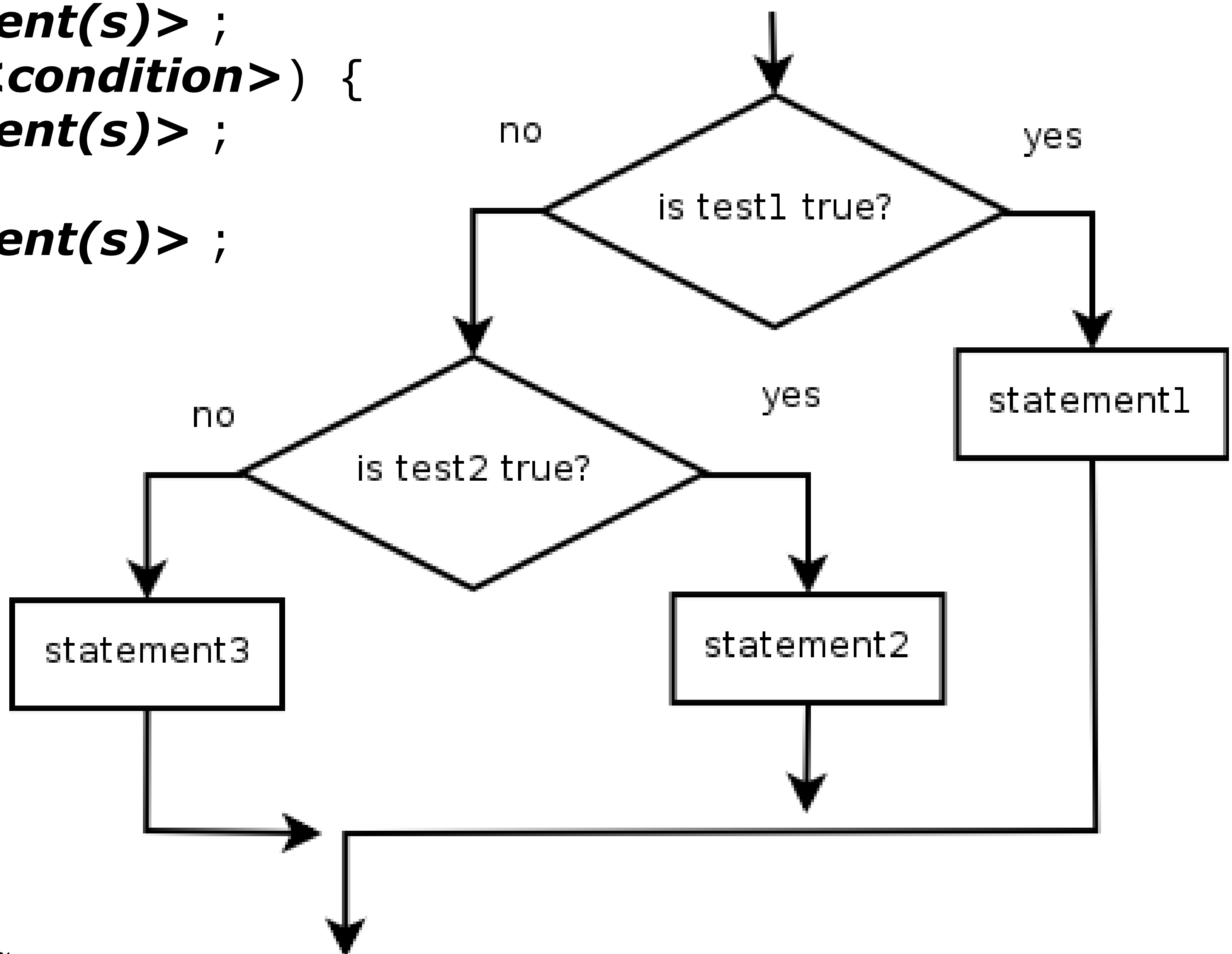
- Example ending with `if`:

```
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?

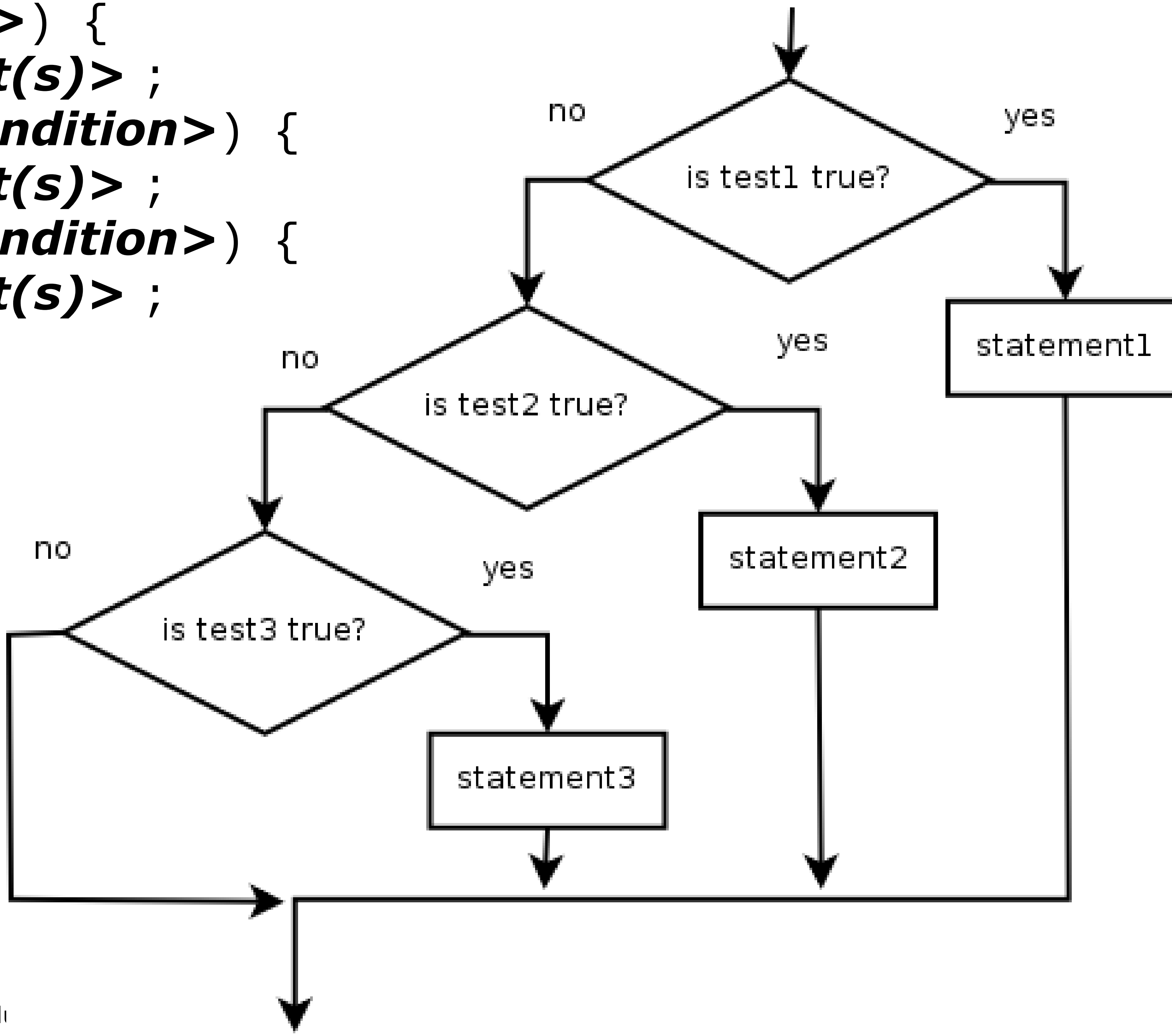
Nested if/else flow diagram

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



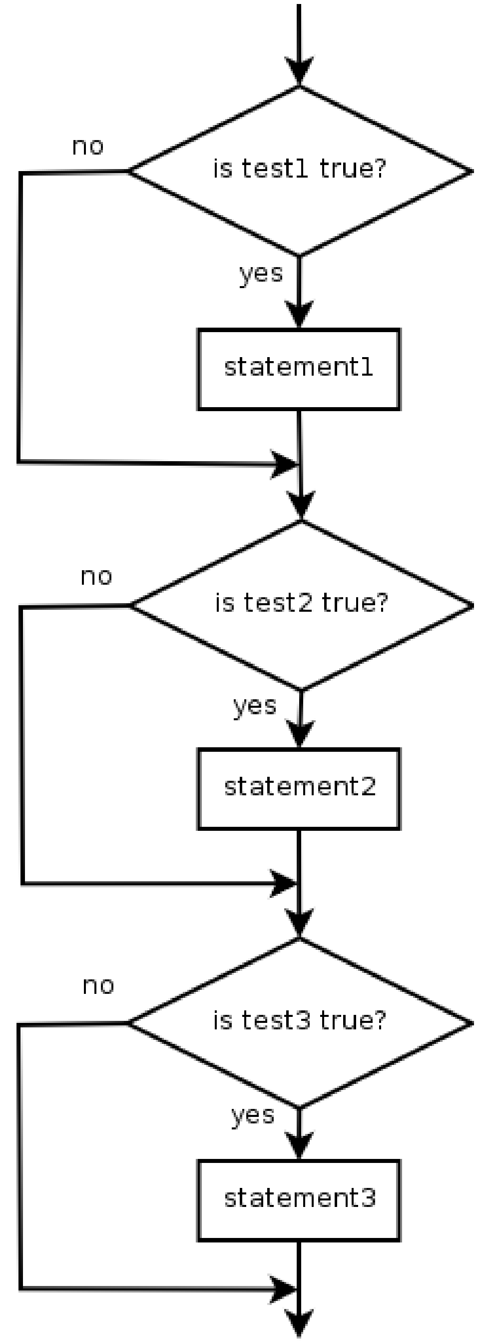
Nested if/else/if diagram

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



Sequential if diagram

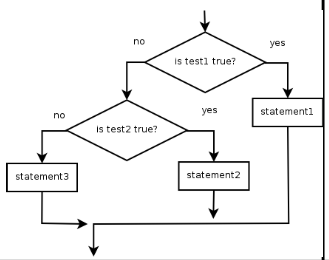
```
if (<condition>) {
      <statement(s)> ;
}
if (<condition>) {
      <statement(s)> ;
}
if (<condition>) {
      <statement(s)> ;
}
```



Structures of if/else code

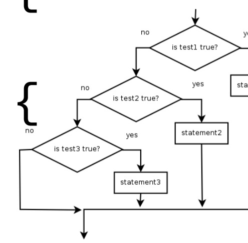
- Choose 1 of many paths:
(conditions are mutually exclusive)

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



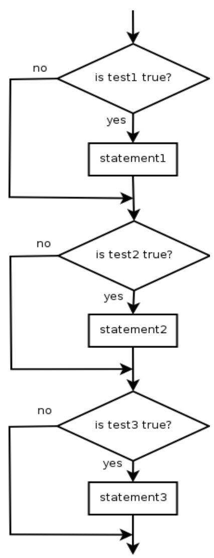
- Choose 0 or 1 of many paths:
(conditions are mutually exclusive and any action is optional)

```
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?
  - 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 answers

- Which `if/else` construct is most appropriate?
  - 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 shouldn't describe the condition being tested.
  - Instead, describe why you are performing that test, 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

- Sometimes putting comments on the `if/else` bodies themselves is more helpful.
  - Example:

```
if (guessAgain == 1) {
    // user wants to guess again; reset game state
    // and start 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));`**

Factoring if/else code

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:

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

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

Code in need of factoring

- The following example has a lot of redundant code:

```
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

- Here is an improved ("factored") version of the same code:

```
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();
```

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

Methods with if/else and return

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 b;
    } else {
        return a;
    }
}
```

All code paths must return

- 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 paths return a value. What if a <= b ?
}
```

- Two fixed versions of the code:

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

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

All code paths must return 2

- The following code also 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 seems that all paths 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.

if/else return question

- 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`.

- Write a method named `min3` that accepts three integers as parameters and returns the smallest of the three. For example, `min3(25, 2, 19)` returns `2`.

if/else return solutions

```
public static int countFactors(int n) {
    int count = 0;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) {
            count++;
        }
    }
    return count;
}

public static int min3(int a, int b, int c) {
    if (a <= b && b <= c) {
        return a;
    } else if (b <= a && b <= c) {
        return b;
    } else {
        return c;
    }
}
```

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
```

Method return answer 1

```
// Prompts for a maximum number and prints each prime up to that maximum.
import java.util.*;

public class Primes {
    public static void main(String[] args) {
        // read max from user
        Scanner console = new Scanner(System.in);
        System.out.print("Maximum number? ");
        int max = console.nextInt();
        printAllPrimes(max);
    }

    public static void printAllPrimes(int max) {
        System.out.print(2);  // print first prime (fencepost)

        // A loop to print the rest of the prime numbers.
        int primes = 1;
        for (int i = 3; i <= max; i++) {
            if (countFactors(i) == 2) {       // i is prime
                System.out.print(", " + i);
                primes++;
            }
        }

        System.out.println();
        System.out.println(primes + " total primes");
    }
```

Method return answer 2

```
    ...

    // Returns how many factors the given number has.
    public static int countFactors(int number) {
        int count = 0;
        for (int i = 1; i <= number; i++) {
            if (number % i == 0) {
                count++;  // i is a factor of number
            }
        }
        return count;
    }
}
```