

Fourth AP* Edition

Java Methods

Object-Oriented Programming
and
Data Structures

Answers and Solutions to Exercises

for Students ✓

Maria Litvin

Phillips Academy, Andover, Massachusetts

Gary Litvin

Skylight Software, Inc.

Skylight Publishing

Andover, Massachusetts

Skylight Publishing
9 Bartlet Street, Suite 70
Andover, MA 01810
(978) 475-1431

e-mail: support@skylit.com
web: <http://www.skylit.com>

**Copyright © 2022 by
Maria Litvin and Gary Litvin**

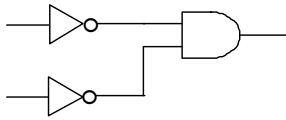
All rights reserved. Students who purchased *Java Methods* are allowed to make one copy. No part of this material may be reproduced for any other purpose without a prior written permission of the authors.

Contents

Chapter 1	An Introduction to Hardware, Software, and the Internet
Chapter 2	An Introduction to Software Development
Chapter 3	Java Syntax and Style
Chapter 4	Objects and Classes
Chapter 5	Data Types, Variables, and Arithmetic
Chapter 6	Boolean Expressions and <code>if-else</code> Statements
Chapter 7	Algorithms and Iterations
Chapter 8	Strings
Chapter 9	Arrays
Chapter 10	Implementing Classes and Using Objects
Chapter 11	<code>java.util.ArrayList</code>
Chapter 12	Class Hierarchies
Chapter 13	Algorithms and Recursion
Chapter 14	Searching and Sorting
Chapter 15	Streams and Files
Chapter 16	Graphics
Chapter 17	GUI Components and Events
Chapter 18	Mouse, Keyboard, Sounds, and Images
Chapter 19	Recursion Revisited
Chapter 20	Sets and Maps

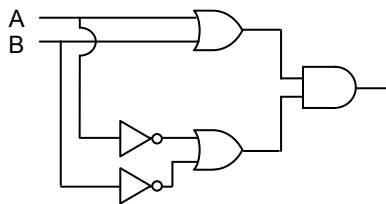
Chapter 1. An Introduction to Hardware, Software, and the Internet

2.

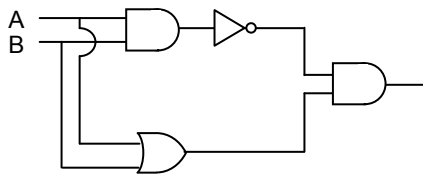


4.

(a)



(b)



6. (b) **F** -- files are created by the operating system
 (c) **F** -- only the boot record is in ROM. Actually, it can boot any operating system that it can find on disk.
8. (c) **S**
10. (b) **T** (however, if you refer to “ASCII” characters as a subset of Unicode, then each “ASCII” character, as all Unicode characters, is represented in two bytes, with the first byte equal to 0)
11. (a) $2^3 = 8$
- 12.
- | | Binary | Decimal | Hex |
|-----|-------------------|---------|------|
| (d) | 00001101 | 13 | 0D |
| (g) | 00000101 10010010 | 1426 | 0592 |
14. $512 * 512 * 8 \text{ bits} = 256 \text{ KB}$. (It takes 8 bits to represent $256 = 2^8$ different values.)
16. Yes. You can use 2 bits per square, for example 00 = empty, 01 = ‘o’, 11 = ‘x’. Then you need $9 * 2 = 18 \text{ bits} = 2.25 \text{ bytes}$.
20. (a) **H** (d) **S** (f) **H**

Chapter 2. An Introduction to Software Engineering

1. (c) **F**
3. (b) **F** (a compiler is needed only for software development)
6. **T**
9. See `J_M\Ch02\Exercises\Solutions\PrintFace.java`.
10. BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, RED, PINK, WHITE, YELLOW.
13. (a) See `J_M\Ch02\Exercises\Solutions\DoubleInput.java`.
(b) `+` adds two numbers, but it concatenates strings and concatenates a number to a string. If you remove the parentheses around `n + n`, then concatenation will be performed first, and instead of, say, 10 you will get 55.
16. See `J_M\Ch02\Exercises\Solutions\HelloGraphics2.java`.

Chapter 3. Java Syntax and Style

3.
 - (a) `import, public, class, extends, implements, private, int, new, this, void, if, else, super, static, false, true`
 - (c) `MovingDisk, time, clock, g, x, y, r, sky, e, args, w, c`
4. (b) **style** (g) **style** (Java is case sensitive, so `IF` and `if` are two different words.)
6. The Java interpreter reports an error:

```
Error: Main method not found in class MovingDisk
```

(Ironically, “Main” is spelled with a capital “M” in interpreter’s error message! It continues in a friendlier manner, though:

```
please define the main method as: public static void main(String[] args)
```
7. The parentheses are required by the syntax, but the braces are optional, since they contain only one statement.

9.

```
public boolean badIndentation(int maxLines)
{
    int lineCount = 3;
    while (lineCount < maxLines)
    {
        System.out.println(lineCount);
        lineCount++;
    }
    return true;
}
```

10. (a) **F** — the compiler ignores indentation and recognizes blocks through braces.
 (c) **T** — such text represents literal strings.
11. (a) The `JFrame`'s constructor that sets the title bar is not called. The program runs, but the title bar is empty.
- (b) Adding `void` confuses the compiler: it now thinks

```
public void HelloGui()
{
    ...
}
```

is a method! Unfortunately, Java allows you to give the same name to a class and a method in that class. Since `HelloGui`'s constructor has been incapacitated, the default constructor is used, which leaves the window blank. This kind of bug can be very frustrating!

Chapter 4. Objects and Classes

1. (c) **F** — it's the other way around: it tells the compiler where it can find classes used by this class.
2. (a) **F** — it also uses `Balloon` and other classes in `balloondraw.jar` (d) **F** — for example, the `Balloon` class does not.
3. (a) Around 50 in `javase` (b) around 360.
4. (b) **T** (e) **F** — an object may not even have an `init` method.
10. (a) **T** (b) **F** — a subclass does not inherit any constructors
11. Deriving `Cylinder` from `Circle` is not appropriate — a bad design decision. It would work, but saving a couple of lines of code is not worth introducing an incorrect IS-A relationship between objects: a `Cylinder` is not a `Circle`.

13. In `BalloonPrint.java`:

```
import java.awt.Color;

public class BalloonPrint
{
    public static void main(String[] args)
    {
        Balloon b = new Balloon(100, 100, 20, Color.RED);
        System.out.println(b);
    }
}
```

In `Balloon.java`:

```
public String toString()
{
    return "Center = (" + xCenter + ", " + yCenter + ")" +
        " radius = " + radius + " color = " + color;
}
```

Chapter 5. Data Types, Variables, and Arithmetic

1. (a) Invalid declaration of local variables: different types should be separated by a semicolon, not a comma.
(b) Field
2. (d) **T** — it is often desirable to give the same name to variables that hold the same types of values for similar purposes in different methods.
(e) **F** — unfortunately the compiler assumes that the code is correct and that the name refers to the local variable where that variable is defined.
5. **compiled**
7. (a) **0** (c) **5.0**
8. (a) **105**
12. See `J_M\Ch05\Exercises\Solutions\FeeToInches.java`.
13.

```
double d = Math.sqrt((double)b * b - 4.0 * a * c);
double x1 = 0.5 * (-b - d) / a;
double x2 = 0.5 * (-b + d) / a;
```
14. Should be: `double temp;`
17. See `J_M\Ch05\Exercises\Solutions\InflatableBalloon.java`.
23. See `J_M\Ch05\Exercises\Solutions\DogsHumanAge.java`.

Chapter 6. Boolean Expressions and `if-else` Statements

2.

```
public static int max(int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

6.

```
(a && !b) || (!a && b)
(a || b) && !(a && b)
a != b
```

8. (a)

```
x && y || !a && !b
```

9. (a)

```
if ((x + 2 > a || x - 2 < b) && y >= 0)
```

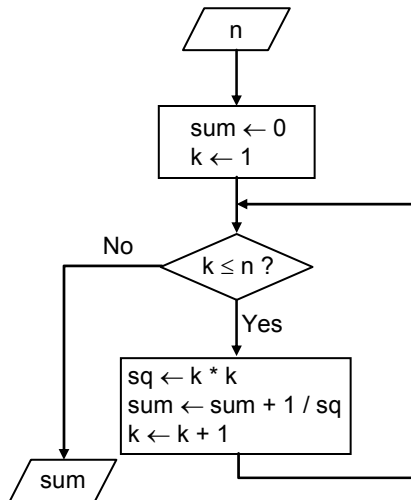
12. (a)

```
boolean inside = (x >= left && x <= right &&
                  y >= top && y <= bottom);
```

14. See `JM\Ch06\Exercises\Solutions\Dates.java`.

Chapter 7. Algorithms and Iterations

1.



Input: n

sum ← 0

k ← 1

Repeat the following
three steps while
 $k \leq n$:

sq = k * k

sum ← sum + 1 / sq

k ← k + 1

Output: sum

```
public class ApproximatePi
{
    private static final int N = 10000;

    public static void main(String[] args)
    {
        double sum = 0;
        for (int k = 1; k <= N; k++)
            sum += 1.0/(k*k);
        System.out.println("Math.PI^2/6 = " + Math.PI*Math.PI/6 +
                           "; Sum approximation = " + sum);
    }
}
```

4. See `JM\Ch07\Exercises\Solutions\QuotientRemainder.java`.6. See `JM\Ch07\Exercises\Solutions\Population.java`.

8.

```
public static int addOdds(int n)
{
    int sum = 0;

    for (int k = 1; k <= n; k += 2)
        sum += k;

    return sum;
}
```

9.

```
public static void main(String[] args)
{
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a positive integer under 10: ");
    int n = input.nextInt();
    input.close();

    int sum = 0;
    for (int k = 1; k <= n; k++)
    {
        if (k > 1)
            System.out.print(" + ");
        System.out.print(k);
        sum += k;
    }
    System.out.println(" = " + sum);
    input.close();
}
```

10. (a)

```
public static boolean isPrime(int n)
{
    if (n < 3)
        return n == 2;
    else if (n % 2 == 0)
        return false;

    int m = 3;

    while (m * m <= n)
    {
        if (n % m == 0)
            return false;
        m += 2;
    }
    return true;
}
```

(b)

```
public static boolean isPrime(int n)
{
    if (n < 5)
        return n == 2 || n == 3;
    else if (n % 2 == 0 || n % 3 == 0)
        return false;

    int m = 5;

    while (m * m <= n)
    {
        if (n % m == 0 || n % (m + 2) == 0)
            return false;
        m += 6;
    }
    return true;
}
```

11.

```
public static boolean isPerfectSquare(int n)
{
    int k = 1, sum = 0;

    while (sum < n)
    {
        sum += k;
        k += 2;
    }
    return sum == n;
}
```

16. **6**

```
public static void main(String[] args)
{
    int n = 37, b = 2;
    int p = 1;

    while (p <= n)
    {
        n -= p;
        p *= b;
    }

    System.out.println(n);
}
```

20.

```
public static void printStarTriangle(int n)
{
    for (int row = 1; row <= n; row++)
    {
        int col = 1;
        while (col <= n - row)
        {
            System.out.print(" ");
            col++;
        }
        while (col < n + row)
        {
            System.out.print("*");
            col++;
        }
        System.out.println();
    }
}
```

Chapter 8. Strings

1. Should be

```
String fileName = "c:\\dictionaries\\words.txt";
```

2. (a)

```
private boolean endsWithStar(String s)
{
    int len = s.length();
    return len > 0 && s.charAt(len - 1) == '*';
}
```

or

```
private boolean endsWithStar(String s)
{
    return s.endsWith("*");
}
```

4. (a)

```
dateStr = dateStr.substring(3,5) + '-' +
          dateStr.substring(0,2) + '-' +
          dateStr.substring(6);
```

5. (a)

```
String last4 = ccNumber.substring(15);
```

- 11.

```
public String cutOut(String s, String s2)
{
    int n = s.indexOf(s2);
    if (n >= 0)
        s = s.substring(0, n) + s.substring(n + s2.length());

    return s;
}
```

- 15.

```
public boolean onlyDigits(String s)
{
    for (int i = 0; i < s.length(); i++)
        if (!Character.isDigit(s.charAt(i)))
            return false;
    return true;
}
```

Chapter 9. Arrays

1. (a) `int a[] = {1, 2, 4};`
2. (a) **F** (c) **T**
 (d) **F** — in arrays, `length` is not a method but works like a public field.

3.

```
public void swapFirstLast(int[] a)
{
    int i = a.length - 1;
    if (i >= 2)
    {
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;
    }
}
```

5.

```
public char getRandomRps()
{
    char[] rps =
        {'r', 'r', 'r', 'p', 'p', 'p', 'p', 'p',
         's', 's', 's', 's', 's', 's'};

    int i = (int)(Math.random() * rps.length);
    return rps[i];
}
```

13.

```
i == j || i + j == n - 1
```

14.

```
private static double positiveMax(double[][] m)
{
    double mMax = 0;

    for (int r = 0; r < m.length; r++)
        for (int c = 0; c < m[0].length; c++)
            if (m[r][c] > mMax)
                mMax = m[r][c];

    return mMax;
}
```

16.

```
private static boolean covers(double[][] m1, double[][] m2)
{
    int count = 0;
    int nRows = m1.length, nCols = m1[0].length;

    for (int r = 0; r < nRows; r++)
        for (int c = 0; c < nCols; c++)
            if (m1[r][c] > m2[r][c])
                count++;

    return 2 * count >= nRows * nCols;
    // return count >= nRows * nCols / 2 doesn't work,
    // for example, nRows = 3, nCols = 3, count = 4
}
```

19.

```
public double average(int[] scores)
{
    int sum = 0;
    for (int s : scores)
        sum += s;
    return (double)sum / scores.length;
}
```

20. (a)

```
public int sumOfValues(int[] a)
{
    int sum = 0;
    for (int x : a)
        sum += x;
    return sum;
}
```

(b)

```
public int[] sumsOfRows(int[][] t)
{
    int[] sums = new int[t.length];

    int k = 0;
    for (int[] row : t)
    {
        sums[k++] = sumOfValues(row);
    }
    return sums;
}
```

27.

```
private static int[] add(int[] a, int[] b)
{
    int[] sum = new int[N];
    int carry = 0;

    for (int i = N-1; i >= 0; i--)
    {
        int d = a[i] + b[i] + carry;
        sum[i] = d % 10;
        carry = d / 10;
    }

    return sum;
}
```

29.

```
public static double averageTopTwo(int[] scores)
{
    int n = scores.length;
    int iMax1 = 0;           // index of the largest element
    int iMax2 = 1;           // index of the second largest element

    // if scores[iMax2] is bigger than scores[iMax1] --
    // swap iMax1 and iMax2
    if (scores[iMax2] > scores[iMax1])
    {
        int i = iMax1;
        iMax1 = iMax2;
        iMax2 = i;
    }

    for (int i = 2; i < n; i++)
    {
        if (scores[i] > scores[iMax1])
        {
            iMax2 = iMax1;
            iMax1 = i;
        }
        else if (scores[i] > scores[iMax2] )
        {
            iMax2 = i;
        }
    }
    return (double)(scores[iMax1] + scores[iMax2]) / 2;
}
```

Chapter 10. Implementing and Using Classes

1.

```
public String replace(String str, char ch)
```

2.

- (a) **F** -- a no-args constructor is not specified.
(b) **T** -- the `int` parameter is promoted to `double`.

4.

Yes for `String`: its documentation describes the following constructor:

“`String(String original)` — Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.”

No for `Color`: its documentation describes several constructors, but none of them is a copy constructor.

9. (b)

```
public Disk(Disk d)
{
    center = new Point(d.center);
    radius = d.radius;
}
```


12. Objects of subclasses of `Integer` or `String` would not be necessarily immutable; if such objects were passed to library methods that rely on immutability, these methods may stop working properly.
14. This class will not compile because the two `swap` methods differ only in their return types. A way to fix it is to rename one of the methods, for example `makeSwappedPair` for the second method.
19. A static method (`main`) calls a non-static method (`hello`). `hello` should be declared `static`.

Chapter 11. `java.util.ArrayList`

1. (a) **T** (c) **F** (e) **T**

4. `[0, 1, 2, 0, 1, 2]`

- 5.

```
public ArrayList<String> reverse(ArrayList<String> list)
{
    ArrayList<String> reversed = new ArrayList<String>(list.size());

    for (int i = list.size() - 1; i >= 0; i--)
        reversed.add(list.get(i));

    return reversed;
}
```

- 7.

```
public void filter(ArrayList<Object> list1, ArrayList<Object> list2)
{
    for (Object obj : list2)
    {
        int j = 0;
        while (j < list1.size())
        {
            if (list1.get(j) == obj)
                list1.remove(j);
            else
                j++;
        }
    }
}
```

14. (c) The number of pairs of (F_1, F_2) with $0 \leq F_1 < p$, $0 \leq F_2 < p$ is finite, so sooner or later the pairs of consecutive values in the Fibonacci list must start repeating. But if $pair_m = pair_n$ with $1 \leq m < n$, then also $pair_{m-1} = pair_{n-1}$. Therefore, a bracelet cannot have a hanging “tail” at the beginning: the first duplicate pair of values must be equal to the first pair.

Chapter 12. Class Hierarchies

1. (a) **T** (e) **F**

3. Only (c) and (d)

4.

```
public class Diploma
{
    private String name, subject;

    public Diploma(String nm, String subj) { name = nm; subject = subj; }
    public String toString()
    {
        return "This certifies that " + name + "\n" +
               "has completed a course in " + subject;
    }
}

public class DiplomaWithHonors extends Diploma
{
    public DiplomaWithHonors(String nm, String subj) { super(nm, subj); }

    public String toString()
    {
        return super.toString() + "\n*** with honors ***";
    }
}
```

6. (b) The program shows that the ratio of the area to the perimeter in the right isosceles triangle (1.757) is greater than that ratio in the equilateral triangle (1.732). Therefore the right isosceles triangle holds a larger inscribed circle.

8.

```
public String toString()
{
    return super.toString().replace("[", "").
        replace(",", " ").replace("]", "");
}
```

13.

```

public class Point1D implements Place
{
    private int x;

    public Point1D(int x) { this.x = x; }
    public int getX() { return x; }

    public int distance(Place other)
    {
        return Math.abs(getX() - ((Point1D)other).getX());
    }
}

public class TestPoint1D
{
    public boolean sameDistance(Place p1, Place p2, Place p3)
    {
        return p1.distance(p2) == p1.distance(p3);
    }

    public static void main(String[] args)
    {
        Point1D p1 = new Point1D(0);
        Point1D p2 = new Point1D(-1);
        Point1D p3 = new Point1D(1);
        TestPoint1D test = new TestPoint1D();
        System.out.println(test.sameDistance(p1, p2, p3));
    }
}

```

16. (a) In Chomp.java, replace

```

HumanPlayer human = new HumanPlayer(this, game, board);
ComputerPlayer computer = new ComputerPlayer(this, game, board);
computer.setStrategy(new Chomp4by7Strategy());

players = new Player[2];
players[0] = human;
players[1] = computer;

```

with

```

players = new Player[2];
players[0] = new HumanPlayer(this, game, board);
players[1] = new HumanPlayer(this, game, board);

```

- (b) In `HumanPlayer.java`, add a name field, a fourth parameter name to the constructor, and a `getName` method; also modify the `getPrompt` and `getWinMessage` methods, adding the name to the messages.

In `Chomp.java`, replace

```
players[0] = new HumanPlayer(this, game, board);
players[1] = new HumanPlayer(this, game, board);
display.setText(" You go first...");
```

with

```
players[0] = new HumanPlayer(this, game, board, "Player 1");
players[1] = new HumanPlayer(this, game, board, "Player 2");
display.setText(players[0].getName() + ", you go first...");
```

- (c) Passing a name as a parameter to `HumanPlayer`'s constructor is more flexible than coding specific names in different subclasses. It also reduces the number of classes. From the object-oriented design point of view, it is more appropriate to treat a name of a player as an attribute of an object, rather than its type.

Chapter 13. Algorithms and Recursion

2. `mysterySum(n)` returns $3n$. `mysterySum(5)` will return 15.

4.

```
public int findMin(int[] list, int n)
{
    if ( n == 1)
        return list[0];
    else
        return Math.min(list[n-1], findMin(list, n-1));
}
```

8. (a)

```
public int sumDigits(int n)
{
    if (n < 10)
        return n;
    else
        return sumDigits(n/10) + n%10;
}
```

9. 010203010

12. (b) E

13. 035. `display` prints all the digits of the number except the most significant digit.

16.

```

public long getSize()
{
    long total = 512 + 128*items.size();

    for (FileItem item : items)
        total += item.getSize();

    return total;
}

```

Chapter 14. Searching and Sorting

2.

```

public int compareTo(Person other)
{
    int diff = getLastName().compareTo(other.getLastName());

    if (diff == 0)
        diff = getFirstName().compareTo(other.getFirstName());
    return diff;
}

```

5. A few target values are much more likely than the rest and these values are placed at the beginning of the array.

6. (a) 6 (b) 7

8.

```

public static int search(int[] a, int m, int n, int target)
{
    if (n <= m)
        return -1;
    int k = (m + n) / 2;
    if (a[k] == target)
        return k;
    int pos = search(a, m, k-1, target);
    if (pos >= 0)
        return pos;
    pos = search(a, k+1, n, target);
    return pos;
}

```

11. (a) T — the number of comparisons in Selection Sort is always the same.
 (b) F — Insertion Sort takes $O(n)$ time if the array is already sorted.

13. 0, 2, 3, 5, 7, 8, 1, 9, 4, 3

15. 6, 9, 11, 10, 2, 22, 81, 74, 54

Chapter 15. Streams and Files

1. **A**
3. (a) Check status — this type of error may happen when the user enters the name of the file and mistypes it (b) Exception (c) Exception
4. See `JM\Ch15\Exercises\Solutions\Braces.java`.
5. See `JM\Ch15\Exercises\Solutions\FileCompare.java`.
7. See `JM\Ch15\Exercises\Solutions\CharImage.java` and `JM\Ch15\Exercises\Solutions\image.txt`.

Chapter 16. Graphics

1. See `JM\Ch16\Exercises\Solutions\Drawings1.java`.
2. See `JM\Ch16\Exercises\Solutions\Drawings2.java`.

Chapter 17. GUI Components and Events

1.

JPanel	none	none
JLabel	none	none
JButton	ActionListener	none
JCheckBox	ActionListener	isSelected
JRadioButton	or ItemListener	
JComboBox	ActionListener or ItemListener	getSelectedIndex or getSelectedItem
JTextField	ActionListener	getText
JSlider	ChangeListener	getValue
JMenuItem	ActionListener	none
2. (a) **T** — why not? It's a regular method.
 (b) **T** — this object's class must implement both `ActionListener` and `ItemListener` interfaces and must supply `actionPerformed` and `itemStateChanged` methods.
 (c) **T** — then all of their respective `actionPerformed` methods are called.
7. See `JM\Ch17\Exercises\Solutions\PizzaGui.java`.

Chapter 18. Mouse, Keyboard, Sounds, and Images

1. See `JM\Ch18\Exercises\Solutions\FourSeasons.java`.
2. See `JM\Ch18\Exercises\Solutions\DrawingPanel.java`.

Chapter 19. Recursion Revisited

2.

```
public boolean isDivisibleBy9(int n)
{
    if (n < 9)
        return false;
    else if (n == 9)
        return true;
    else
        return isDivisibleBy9(sumDigits(n));
}
```
4. (a) `pow(x, n)` executes $n-1$ multiplications. It is easy to prove this fact using mathematical induction. Therefore, this version is no more economical than a simple `for` loop. The answer is 4.
6. 100. `mysterySum(n)` returns n^2 . Indeed, $(n-1)^2 + 2n - 1 = n^2$.

7.

```
public boolean degreeOfSeparation(Set<Person> people,
                                Person p1, Person p2, int n)
{
    if (n == 1)                // Base case
    {
        return p1.knows(p2);
    }
    else                        // Recursive case
    {
        for (Person p : people)
        {
            if (p1.knows(p) && degreeOfSeparation(people, p, p2, n-1))
                return true;
        }
        return false;
    }
}
```

Let $K(n)$ be the number of times `knows` is called for the parameter value of n . Then, for $n = 1$, $K(1) = 1$, and the formula gives $\frac{3N^1 - N^0 - 2N}{N-1} = \frac{3N-1-2N}{N-1} = 1$. For $n > 1$, `knows` is called once for each `Person p` in the group of N people who know `p1` (due to the short-circuit evaluation) and $1 + K(n-1)$ times for each `Person p` in the group of N people who do not know `p1`. Therefore, $K(n) = N + N(1 + K(n-1)) = 2N + N \cdot K(n-1)$. By the induction

hypothesis, $K(n-1) = \frac{3N^{n-1} - N^{n-2} - 2N}{N-1}$. So

$$\begin{aligned} K(n) &= 2N + N \cdot K(n-1) = 2N + N \frac{3N^{n-1} - N^{n-2} - 2N}{N-1} = \\ \frac{2N^2 - 2N + 3N^n - N^{n-1} - 2N^2}{N-1} &= \frac{3N^n - N^{n-1} - 2N}{N-1}, \text{ Q.E.D.} \end{aligned}$$

Chapter 20. Sets and Maps

3.

```
public int busiestHour(List<PhoneCall> dayCalls)
{
    int[] counts = new int[24];

    for (PhoneCall call : dayCalls)
    {
        if (call.getDuration() >= 30)
            counts[call.getStartHour()]++;
    }

    int maxHour = 0;

    for (int hour = 1; hour < 24; hour++)
        if (counts[hour] > counts[maxHour])
            maxHour = hour;

    return maxHour;
}
```

6.

```
public boolean contains(Record record)
{
    int k = record.hashCode();
    if (k >= buckets.size())
        return false;
    ArrayList<Record> bucket = buckets.get(k);
    for (Record rec : bucket)
    {
        if (record.equals(rec))
            return true;
    }
    return false;
}
```