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

Contents

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

2.
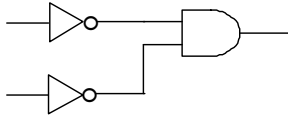
4.
   (a)
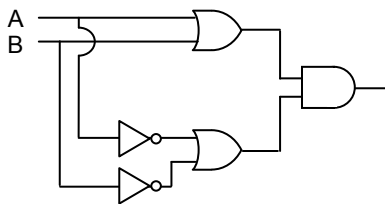
6.   (b) **F** -- files are created by the operating system
    (c) **F** -- only the boot record is in ROM. Actually, it can boot any operating system that it can find
        on disk.

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)	`00001011`	11	0B
(g)	`00000101 10010010`	1426	0592

14.   512 * 512 * 8 bits = 256 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 bits = 2.25 bytes.

20.   (a) **H** (d) **S** (f) **H**

Chapter 2. An Introduction to Software Development

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.

11.   (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.

14.   See J_M\Ch02\Exercises\Solutions\HelloApplet2.java and
       J_M\Ch02\Exercises\Solutions\TestApplet2.html.

Chapter 3. 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 GridWorld's library classes (d) **F**

3.   (b) around 350

4.   (b) **T** (e) **F** — an object may not even have an `init` method.

8.
```
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;
import info.gridworld.actor.Bug;
import info.gridworld.actor.Rock;
import java.awt.Color;

public class BugRunner
{
  public static void main(String[] args)
  {
    ActorWorld world = new ActorWorld();
    world.add(new Location(1, 2), new Bug());
    world.add(new Location(0, 0), new Bug(Color.GREEN));
    world.add(new Rock(Color.GRAY));
    world.show();
  }
}
```

10.   The output is:

```
info.gridworld.actor.Bug[location=null,direction=0,color=java.awt.
Color[r=255,g=0,b=0]]
```

The `toString` method of the `Actor` class defines this output:

```
public String toString()
{
  return getClass().getName() + "[location=" + location +
    ",direction=" + direction + ",color=" + color + "]";
}
```

14.   (a) **T** (b) **F** — a subclass does not inherit any constructors

15.   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 `Cicrle`.

Chapter 4. Algorithms

1.
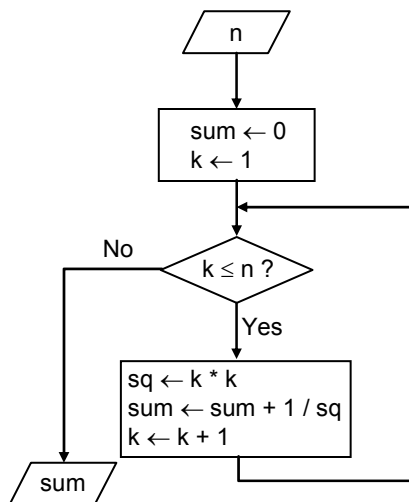

```
Input: n

sum ← 0

k ← 1

Repeat the following
three steps while
k ≤ n:

  sq = k * k
  sum ← sum + 1 / sq
  k ← k + 1

Output: sum
```

3.


```
Input: m, n

q ← 0
r ← m

Repeat the following
steps while r ≥ n:

    r ← r - n
    q ← q + 1

Output: q, r
```
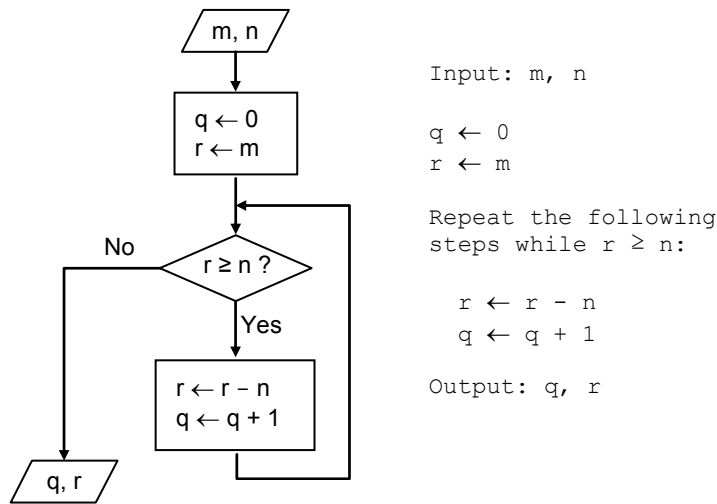
4.   **6**

9.   **15**

13.
```
public double goldenRatioSeq(int n)
{
  if (n == 1)
    return 1;
  else
    return 1 + 1 / goldenRatioSeq(n - 1);
}
```

14.   (b) **E**

17.   If $n = 0$, no radioactive coins found. If $n = 1$, test the coin. If $n > 1$, split the bag into two approximately equal bags. Try to find the radioactive coin in the first bag. If not found, try to find the radioactive coin in the second bag. Using this algorithm you need 10 trials for 1000 coins (h trials for 2^h coins).

18.   If you have three coins, compare the weights of any two. If equal, the third one is the fake; otherwise the lighter one is the fake. For 3^n coins, split them into three groups of 3^{n-1} coins in each group. Compare the weights of any two groups. If equal, the fake is in the third group; otherwise it is in the lighter group. Look for the fake in the identified group of 3^{n-1} coins using the same method. Using this algorithm you need 4 trials for 81 coins.

Chapter 5. Java Syntax and Style

3.

(a) `import`, `public`, `class`, `extends`, `implements`, `private`, `int`, `super`, `new`, `this`, `void`, `if`, `else`, `static`, `false`, `true`

(c) `MovingDisk`, `time`, `clock`, `g`, `x`, `y`, `r`, `sky`, `c`, `e`, `w`, `args`

4.   (b) **style** (g) **style** (Java is case sensitive, so `IF` and `if` are two different words.)

6.   The Java interpreter "throws an exception":
Exception in thread "main" java.lang.NoSuchMethodError: main

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 6. 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**

6.   (a) **0** (c) **5.0**

7.   (a) **105**

11.
```
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;
```

12.   Should be: `double temp;`

18.   See J_M\Ch06\Exercises\Solutions\DogsHumanAge.java.

Chapter 7. Boolean Expressions and `if-else` Statements

1.   5 and 2

3.
```
public static int max(int x, int y)
{
  if (x > y)
    return x;
  else
    return y;
}
```

5.
```
(a && !b) || (!a && b)
(a || b) && !(a && b)
a != b
```

7.   (a)
```
x && y || !a && !b
```

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

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

13. See ^{J}M\Ch07\Exercises\Solutions\<u>Dates.java</u>.

Chapter 8. Iterative Statements: `while, for, do-while`

1.
```java
public class Population
{
  private static final double growthRate = 1.0113; // 1.13 percent growth
                                          //     per year
  public static void main(String[] args)
  {
    double population = 111.2, target = 120.0;
    int year = 2010;

    while (population < target)
    {
      population *= growthRate;
      year++;
    }

    System.out.println("The population will reach " + target
        + " million in " + year);
  }
}
```

3.
```java
public static int addOdds(int n)
{
  int sum = 0;

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

  return sum;
}
```

5.

```java
public static void main(String[] args)
{
  Scanner input = new Scanner(System.in);
  int sum = 0;

  System.out.print("Enter a positive integer under 10: ");
  int n = input.nextInt();

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

6.   (a)

```java
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)

```java
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;
}
```

7.
```
public static boolean isPerfectSquare(int n)
{
  int i = 1, sum = 0;

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

Chapter 9. Implementing Classes and Using Objects

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 value) — 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 sevral 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.

15. 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.

18. A static method (`main`) calls a non-static method (`hello`). `hello` should be declared `static`.

Chapter 10. 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 11. Class Hierarchies and Interfaces

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.   See J_M\Ch11\Exercises\Solutions\SlowBug.java.

8.   (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.

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));
  }
}
```

Chapter 12. 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];
}
```

12.
```
i == j || i + j == n - 1
```

13.
```
private static double positiveMax(double[][] m)
{
  double mMax = 0;
  int rows = m.length, cols = m[0].length;

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

  return mMax;
}
```

15.

```
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
}
```

17. (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.

24.

```
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;
}
```

25.

```
public static double averageTopTwo(int[] scores)
{
  int i, 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])
  {
    i = iMax1;
    iMax1 = iMax2;
    iMax2 = i;
  }

  for (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 13. `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.   (b) No: critters won't "eat" new bugs.

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.

14.   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 J_M\Ch15\Exercises\Solutions\Braces.java.

5.   See J_M\Ch15\Exercises\Solutions\FileCompare.java.

7.   See J_M\Ch15\Exercises\Solutions\CharImage.java and
      J_M\Ch15\Exercises\Solutions\image.txt.

Chapter 16. Graphics

1.   See J_M\Ch16\Exercises\Solutions\Drawings1.java.

2.
```
import ...

public class Drawings extends JPanel
{
  public void paintComponent(Graphics g)
  {
    super.paintComponent(g);

    // (a)
    int x0 = 40;
    int y0 = 40;
    g.setColor(Color.BLACK);
    g.drawRect(x0 - 15, y0, 30, 20);
    g.drawOval(x0 - 10, y0 - 20, 20, 20);

    // (g)
    x0 += 60;
    g.setColor(Color.BLACK);
    g.drawArc(x0 - 20, y0 - 20, 40, 40, 90, 270);
    g.drawLine(x0, y0, x0, y0 - 20);
    g.drawLine(x0, y0, x0 + 20, y0);
  }

  public static void main(String[] args)
  {
    ...
  }
}
```

Chapter 17. GUI Components and Events

1.
```
     JPanel          none                  none
     JLabel          none                  none
     JButton         ActionListener        none
     JCheckBox       \ ActionListener      isSelected
     JRadioButton    |      or             isSelected
     JComboBox       / 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 ^{J}M\Ch17\Exercises\Solutions\<u>PizzaGui.java</u>.

Chapter 18. Mouse, Keyboard, Sounds, and Images

1.   See ^{J}M\Ch18\Exercises\Solutions\<u>FourSeasons.java</u>.

2.   See ^{J}M\Ch18\Exercises\Solutions\<u>DrawingPanel.java</u>.

6.   See ^{J}M\Ch18\Exercises\Solutions\<u>ImagePanel.java</u>.

Chapter 19. Big-O Analysis of Algorithms

1.   (a) **T** (b) **T** — $\log_2 n = \log_{10} n \cdot \log_2 10$

2.   (a) $O(n^2)$ (c) $O(n)$

3.   (c) $O(\log n)$

4.   (a) P
      (c) E. This task is equivalent to finding the largest clique in a graph. (A graph is a set of nodes with edges connecting some of the nodes; a set of nodes in a graph is called a clique if any two nodes in that set are connected with an edge.)   In complexity theory, there is a proof that this is what is called an NP-complete problem: it is equivalent to a whole class of problems for which no polynomial-time algorithms are known and are unlikely to be ever discovered.

7.   (a) always (b) sometimes

10.   (b) **F**

Chapter 20. The Java Collections Framework

2.
```
public <E> void append(List<E> list1, List<E> list2)
{
  for (int i = 0; i < list2.size(); i++)
    list1.add(list2.get(i));
}
```

6.
```
public double sum2(List<Double> list)
{
  double sum = 0;

  ListIterator<Double> iter1 = list.listIterator();
  while (iter1.hasNext())
  {
    double a = iter1.next().doubleValue();
    ListIterator<Double> iter2 = list.listIterator(iter1.nextIndex());

    while (iter2.hasNext())
    {
      sum += a * iter2.next().doubleValue();
    }
  }

  return sum;
}
```

8.   Three-Two
      Three-Two-One
      Three-Two-One

10.   (a) Should be:

```
// Restore cursor position:
double y = stk.pop().doubleValue();
double x = stk.pop().doubleValue();
```

   (b) The following simplification uses `Point`'s copy constructor:

```
Point cursor;
Stack<Point> stk = new Stack<Point>();
...
// Save cursor position:
stk.push(new Point(cursor));

show(new LoginWindow());
...
// Restore cursor position:
cursor = stk.pop();
```

   Recall that a stack holds references to objects. It is necessary to make and push a copy of cursor because subsequent code may change the original.

13.   `0 2 1 3 2 4`

21.   (a) *O*(1) (c) *O*(n)

Chapter 21. Lists and Iterators

1.
```
ListNode node3 = new ListNode("Node 3", null);
ListNode node2 = new ListNode("Node 2", node3);
ListNode node1 = new ListNode("Node 1", node2);
ListNode head = node1;
```

3.
```
public ListNode removeFirst(ListNode head)
{
  if (head == null)
    throw new NoSuchElementException();

  ListNode temp = head.getNext();
  head.setNext(null);
  return temp;
}
```

5.

```
public ListNode add(ListNode head, Object value)
{
  ListNode newNode = new ListNode(value, null);
  if (head == null)
    head = newNode;
  else
  {
    ListNode node = head;
    while (node.getNext() != null)
      node = node.getNext();
    node.setNext(newNode);
  }
  return head;
}
```

9.

```
public ListNode insertInOrder(ListNode head, String s)
{
  ListNode node = head, prev = null;

  while (node != null && s.compareTo(node.getValue()) > 0)
  {
    prev = node;
    node = node.getNext();
  }

  if (node != null && s.equals(node.getValue()))
    return head;

  ListNode newNode = new ListNode(s, node);

  if (prev == null)
    head = newNode;
  else
    prev.setNext(newNode);

  return head;
}
```

Chapter 22. Stacks and Queues

1.   (a) **F**

4. A stack is not needed because we can process `binNum`'s characters in reverse, starting at the end of the string:

```java
public class BinToDecimal
{
  public static int binToInt(String binNum)
  {
    int result = 0, power2 = 1;

    for (int i = binNum.length() - 1; i >= 0; i--)
    {
      char ch = binNum.charAt(i);
      int dig = Character.digit(ch, 2);
      result += dig * power2;
      power2 *= 2;
    }

    return result;
  }
}
```

5.
```java
public boolean moveToTop(Stack<Card> deck, int n)
{
  Stack<Card> temp = new Stack<Card>();

  while (n > 1 && !deck.isEmpty())
  {
    temp.push(deck.pop());
    n--;
  }

  Card nth = null;

  if (!deck.isEmpty())
    nth = deck.pop();

  while (!temp.isEmpty())
  {
    deck.push(temp.pop());
  }

  if (nth != null)
  {
    deck.push(nth);
    return true;
  }
  else
    return false;
}
```

6. (b) This implementation is quite inefficient because `String stack` is reallocated in each `push` and `pop` operation.

9. The integer values stored at `40:1A` and `40:1C` are the same, `0028` (in hex). These offsets represent the front and the rear of the ring buffer. The fact that they are the same indicates that the keyboard queue is currently empty. The last eight ASCII codes, stored in the buffer (going from location `0028` and around) are `64 20 34 30 3A 31 61 0D` (in hex), which corresponds to the string `d40:1a`. Actually, this string is the "dump" command in the *MS-DOS debug* program that was used to produce the memory dump for this question.

10.   **C**

Chapter 23. Recursion Revisited

1.   `010203010`

3.   `009.` `display` prints all the digits of a number except the most significant digit.

6.
```
public boolean isDivisibleBy9(int n)
{
  if (n < 9)
    return false;
  else if (n == 9)
    return true;
  else
    return isDivisibleBy9(sumDigits(n));
}
```

7.   (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.

9.   100. `mysterySum(n)` returns n^2. Indeed, $(n-1)^2 + 2n - 1 = n^2$.

10.
```
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 $\dfrac{3N^1 - N^0 - 2N}{N-1} = \dfrac{3N - 1 - 2N}{N-1} = 1$. 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, $K(n-1) = \dfrac{3N^{n-1} - N^{n-2} - 2N}{N-1}$. So

$$K(n) = 2N + N \cdot K(n-1) = 2N + N\frac{3N^{n-1} - N^{n-2} - 2N}{N-1} =$$

$$\frac{2N^2 - 2N + 3N^n - N^{n-1} - 2N^2}{N-1} = \frac{3N^n - N^{n-1} - 2N}{N-1}, \text{Q.E.D.}$$

Chapter 24. Binary Trees

2. $[\log_2 100000] + 1 = \log_2(1024 \cdot 128) = 17$.

4.   Base case: For $h = 0$ the number of nodes in the tree is 0 and $0 = 2^0 - 1$. Likewise, for $h = 1$ the number of nodes in the tree is 1 and $1 = 2^1 - 1$. Suppose the statement is true for any $q < h$. Take a tree with h levels. By the inductive hypothesis, the numbers of nodes in its left and right subtrees do not exceed $2^{h-1} - 1$. Therefore, the total number of nodes for the tree does not exceed $(2^{h-1} - 1) + (2^{h-1} - 1) + 1 = 2^h - 1$, Q.E.D.

6.
```
public boolean isLeaf(TreeNode node)
{
  return node != null && node.getLeft() == null && node.getRight() == null;
}
```

8.   Leaves.

10.
```
public int depth(TreeNode root)
{
  if (root == null)
    return -1;

  return 1 + Math.max(depth(root.getLeft()), depth(root.getRight()));
}
```
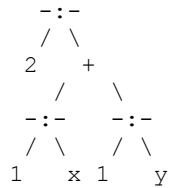
12.   (a)
```
public TreeNode copy(TreeNode root)
{
  if (root == null)
    return null;

  return new TreeNode(root.getValue(), copy(root.getLeft()),
                                       copy(root.getRight()));
}
```
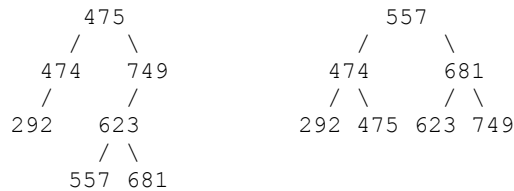
15.   (a)
```
           -:-
          / \
         2   +
          /   \
        -:-    -:-
        / \    / \
       1   x 1   y
```
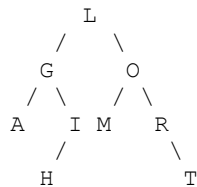
(g)   Inorder, preorder, and postorder traversals of a binary tree visit its leaves in the same
      sequence. You can use mathematical induction (over the total number of nodes) to prove this
      fact: apply the induction hypothesis to the left and the right subtrees.

16.   (b) **T** (c) F — the tree may have degenerated into a near linear shape

18.
```
       475                       557
      /    \                   /      \
    474    749               474      681
    /      /                 / \      / \
  292    623             292 475  623 749
         / \
       557 681
```

19.
```
        L
      /    \
     G      O
    / \    / \
   A   I M   R
      /       \
     H         T
```

Inorder: A G H I L M O R T   Preorder: L G A I H O M R T
Postorder: A H I G M T R O L

22.
```java
public TreeNode maxNode(TreeNode root)
{
  if (root == null)
    return null;

  TreeNode node = root;
  while (node.getRight() != null)
    node = node.getRight();

  return node;
}
```

Chapter 25. Lookup Tables and Hashing

3.
```java
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.   (a) **F** — it is $O(1)$
   (d) **T** — for a reasonably functioning hash table; also, after several removals and additions, a BST may need rebalancing.

7.   (b) A `hashTable` element takes 4 bytes, `ListNode` takes 8 bytes; `Record` takes 20 bytes. With 5 nodes per slot (on average) we need $1000 \cdot 4 + 5000 \cdot (8 + 20) = 144,000$ bytes for the hash table. We need $12000 \cdot 4 + 5000 \cdot 20 = 148,000$ bytes for the lookup table. The lookup table takes less than 3% extra space. Finding a record in a hash table takes one `hashCode` computation plus, on average, three record comparisons. The retrieval operation will run four times faster with the lookup table.

Chapter 26. Heaps and Priority Queues

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

3.   (a)   parent: `x[i/2]`; left child: `x[2*i]`; right child: `x[2*i+1]`
      (b) `2 * i > n`

5.   (a)
```
         3
       /   \
      8     5
     / \   / \
   21 55 34 13
```
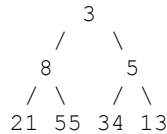
Chapter 27. Design Patterns

1.   See J_M\Ch27\Exercises\Solutions\EasyDate.java.

2.   See the Java files in J_M\Ch27\Goofenspiel\Solution\.

8.   If we make a composite experssion (`SumExpression` and `ProductExpression`) `Observable`, how will it know when its left or right components have changed? In general in MVC, the model must be self-contained. If some of its fields change independently, outside the model, the MVC design breaks down. One possible solution to this problem is to make a composite object both `Observable` and `Observer` and attach it as an `Observer` to all its components. When one of its components changes, the composite will be notified and then it can update its view and/or pass the change along to other composites that hold this one as a component.