Answers to Quick-Check Questions

Chapter 1

Section 1-1

1. CPU, memory, and I/O
2. microprocessor
3. arithmetic: addition, subtraction, multiplication, division
4. logic: equal to, not equal to, less than, greater than, less than or equal to, greater than or equal to
5. arithmetic logic unit (ALU), control unit, and internal registers
6. 134,217,728
7. firmware

Section 1-2

1. Java is strictly an object-oriented language, while C++ is a hybrid structured and object-oriented language.
2. machine language, assembly language, high-level language
3. to manage the resources of the system
4. Java is object-oriented, simple, portable, secure, a network language, supports multithreading, growing, cheap, and is for business.
5. true
6. Oak
7. Sun
Appendix E
Answers to Quick-Check Questions

Section 1-3

1. enter/edit, save, compile, correct errors, translate Java byte code to machine code
2. Java byte code is a low-level code which is generic and not specific to any particular CPU.
3. The generic byte codes generated by the Java compiler make Java an extremely portable language.
4. Java applications are designed to be run on a computer, just like any other application software. Java applets are designed only to be run using an Internet Web browser.
5. A compiler translates the entire program into machine code all at one time, before execution by the CPU. An interpreter translates, then executes, one high-level program statement at a time.
6. source code
7. import and class
8. import myPackage.*;
9. method
10. Inventory.java
11. d. two forward slashes like this //COMMENT.
12. At a minimum, the program should include the following comments:
   The beginning of the program should be commented with the programmer’s name, date the program was written, date the program was last revised, and the name of the person doing the revision. In other words, a brief ongoing maintenance log should be commented at the beginning of the program.
   The beginning of the program should be commented to explain the purpose of the program, which includes the problem definition. This provides an overall perspective by which anyone, including you, the programmer, can begin debugging or maintaining the program.
   Import statements should be commented as to their purpose.
   Constants, variables, and objects should be commented as to their purpose.
   Major sections of the program should be commented to explain the overall purpose of the respective section.
   Individual program lines should be commented when the purpose of the code is not obvious relative to the application.
   All major subprograms (methods in Java) should be commented just like the main program method.
   The end of each program block (right curly brace) should be commented to indicate what the brace is ending.

Chapter 2

Section 2-1

1. pseudocode
2. What output is needed?
What input is needed? What processing is needed to produce the output from the input?

3. Desk-check, compile, debug using a debugger, and run the program.

4. Commenting explains what the program does and self-documents the program, making it easier to read and maintain.

5. A syntax error is any violation of the rules of the programming language.

6. A logic error is an error made by the programmer relative to the way in which the program will execute. A logic error occurs when the program does what you tell it to do, but is not doing what you meant it to do.

Section 2-2

1. To define what steps are needed to produce the desired final result. An algorithm keeps you from “spinning your wheels” at the keyboard.

2. sequence, decision, and iteration

3. The three decision operations are if, if/else, and switch/case.

4. The three iteration operations are while, do/while, and for.

Section 2-3

1. Abstraction allows you to “see the forest over the trees,” because it permits you to initially concentrate on the problem at hand, without agonizing over the implementation details of a computer language.

2. Stepwise refinement begins with the initial abstract algorithm, and step by step divides it into one or more refined algorithms that provide more and more implementation detail.

3. A codeable level of an algorithm is reached when all the statements have been reduced to the pseudocode operations listed in Table 2-1.

Chapter 3

Section 3-1

1. class

2. standard classes

3. An abstract data type, or ADT, describes the data attributes and behavior of its objects.

4. primitive data types, standard classes, and programmer-defined classes

5. Behavior, as related to classes and ADTs, describes how the ADT or class will act and react to a given operation.

6. A method is a subprogram in Java, usually associated with an algorithm of some kind.

7. Because a class allows you to build your own ADTs in a programming language. Classes contain data to provide the ADT attributes, and methods to provide the ADT behavior.
Appendix E  Answers to Quick-Check Questions

Section 3-2

1. approximately −2 billion to +2 billion
2. an overflow error
3. fixed decimal or exponential
4. the character ‘a’
5. the character ‘a’
6. 28 characters (including blanks) @ two bytes per character = 56 bytes
7. boolean

Section 3-3

1. The two reasons for defining constants and variables in a Java program are as follows:
   • The compiler must know the value of a constant before it is used, and must reserve memory locations to store variables.
   • The compiler must know the data type of constants and variables to determine their data attributes and behavior.
2. final char PERIOD = ‘.’;
3. final String BOOK = “Information Systems Programming with Java”;
4. int age = 0;
5. String course = “Accounting”;
6. int length = 0;
   length = course.length();
7. course = course.toUpperCase();
8. true

Section 3-4

1. A class provides the foundation for creating specific objects, each of which shares the general attributes and behavior of the class.
2. An object is an instance, or specimen, of a given class. An object of a given class has the structure and behavior defined by the class that is common to all objects of the same class.
3. By a class defining the behavior of its objects, we mean that the class defines how its objects act and react when they are accessed via the class methods.
4. true
5. CheckingAccount myAccount = new CheckingAccount();
6. Truck myPickup = new Truck();
7. Automobile myConvertible = new Automobile();
Chapter 4

Section 3-5

1. StaugI0 myIO = new StaugIO();
2. myIO.writeInfo("THIS STUFF IS GREAT!");
3. myIO.writeInfo("THIS STUFF \n IS GREAT!");
4. double gpa = 0.0;
   gpa = myIO.readDouble("Enter your grade point average: ");
5. myIO.writeNumber(gpa);
6. import java.text.*;
   NumberFormat currency = NumberFormat.getCurrencyInstance();
   i0.writeInfo(currency.format(mySalary));
7. The NumberFormat class provides for true currency formatting in dollars and cents, which includes a $ sign, commas where required, and trailing 0's.

Section 3-6

1. System.out.println("Andrew C. Staugaard, Jr.");
2. System.out.println(name);
3. \
4. \r
5. The println() method generates a newline after its output to place the cursor on the next line, while the print() method does not generate a newline.

Chapter 4

Section 4-1

1. The order in which Java performs arithmetic operations is as follows:

   ( ) * / % + –

   Any operations inside of parentheses are performed first, then (from left to right) multiplication, division, and modulus, and then (from left to right) addition and subtraction.

2. The value of x is even.
3. The value of y is divisible by 17.
4. 0
5. 0
6. 1
7. --x; or x--; 
8. False, because this happens only when both the numerator and denominator operands are integers.
9. The difference between using the preincrement operator versus the postincrement operator on a variable is that a preincrement operator increments the
variable before any expression involving the variable is evaluated, and a post-
increment operator increments the variable after any expression involving the
variable is evaluated.

10. The result of 10/100 is 0 because both operands are integers, and the / operator
generates an integer result when both operands are integers.

11. System.out.println(Math.random()*100);

12. System.out.println(Math.tan(45*Math.PI/180);

13. Math.PI;

14. The answer to this question depends on your particular Java compiler. Press the
Fl key for JBuilder and Shift+F1 for Sun ONE Studio (Forte).

Section 4-2

1. x+=5;
2. x/=10;
3. This expression cannot be written using a compound assignment operator.
4. Because x can never be equal to itself plus 5 in math class, but you can add 5 to x
and assign the sum to x in Java.

Chapter 5

Section 5-1

1. relational operators
2. The difference between the = operator and the == operator in Java is that the =
operator assigns the value on the right to the variable on the left, and the ==
operator compares two quantities to determine if they are equal.
3. the Boolean value true
4. The Boolean value false is generated, because (5!=5) is false, making the entire
AND statement false.
5. because string values are represented as objects, not primitive data types

Section 5-2

1. False, because the logical opposite of greater than is less than or equal to.
2. False, because the if statements are executed when the test expression is true.
3. The test expression needs to be a comparison, not an assignment. The correct if
statement is:

   if(x == y)
   System.out.println("There is a problem here");

4. The && (AND) operator must be employed to test if all conditions are true.
5. The !& (NOT AND) operators must be employed to test if one or more of sev-
eral conditions is false.
6. The `||` (OR) operator must be employed to test if one of several conditions is true.

7. For all values of \( x \) that are less than or equal to 50

**Section 5-3**

1. True, an `else` must always have a corresponding `if`.
2. Because, without an `else` statement, both strings, “It’s payday” and “It’s not payday”, would be written when the `if` test is true.
3. True, framing can be eliminated when an `if` or `else` has only a single statement, but it is not advisable to do so.
4. Subtract
5. False, `compareTo()` returns an integer value that must be tested against 0.
6. True, the `equals()` method returns a Boolean value.

**Section 5-4**

1. Indentation is important when operations are nested for code readability, and to be able to see at a glance which statements belong to which `if` or `if/else` statement.
2. Misplaced, or dangling, `else`
3. “Red” will be written when the value is greater than –50 and less than 50.
4. “White” will be written when the value is less than or equal to –50.
5. “Blue” will be written when the value is greater than or equal to 50.
6. The equivalent `if-else-if-else` logic is:
   ```java
   if Value >= 50
       Write (“Blue”)
   else
       if Value <= -50
           Write (“White”)
       else
           Write (“Red”)
   ```

**Section 5-5**

1. Matching
2. If you have \( n \) cases in a `switch` statement and there are no `break` statements in any of the cases, all of the subsequent cases will be executed sequentially when a match is made on any given case.
3. False, because there may be times when several subsequent cases need to be executed as the result of a match to a given case.
4. `default`
5. For menu-driven programs
Chapter 6

Section 6-1

1. False, because a while loop repeats until the test expression is false.
2. False, because a while loop is a pretest loop not a posttest loop.
3. The while loop statements need to be framed because without the framing, the value of x is never changed and you will have an infinite loop.
4. The correct code is:

```java
int x = 10;
while (x > 0)
{
    System.out.println("This is a while loop");
    --x;
} //END WHILE
```

5. The loop will never execute because the loop test is false the first time it is tested. Notice that x is initialized to 1 and is never <= 0.
6. Without the limitations of the computer, the loop will execute infinitely because x starts out greater than 0 and is incremented inside of the loop. But, considering the physical limitations of the computer, it will execute about 2 billion times before wrapping around to a negative value, causing the test to be false.

Section 6-2

1. False, because a do/while loop repeats until the test expression is false.
2. true
3. The value of x is never changed within the loop, thereby creating an infinite loop.
4. The correct code is:

```java
int x = 10;
do
{
    System.out.println("This is a do/while loop");
    --x;
} //END DO/WHILE
while (x > 0);
```

5. The loop will execute once because a do/while loop is a posttest loop.
6. Without the limitations of the computer, the loop will execute infinitely because x starts out greater than 0 and is incremented inside of the loop. But, considering the physical limitations of the computer, it will execute about 2 billion times before wrapping around to a negative value, causing the test to be false.
Section 6-3
1. The three things that must appear in the first line of a for loop structure are:
   • The loop counter initialization
   • The loop test expression
   • The changing of the counter value
2. true
3. true, as long as the while loop counter is changed at the end of the loop body
4. The loop will execute zero times because the test condition is false the first time.
5. The loop will execute 11 times.
6. The for loop statements must be framed when there is more than one statement to be executed within the loop.
7. The inner loop will execute 5 times for every outer loop iteration. Thus, there are 5 × 10, or 50, total iterations.
8. decremented

Section 6-4
1. continue
2. if
3. The loop will execute twice. When \( x \) is incremented to 1 at the end of the first iteration, the \( \text{if}(x>0) \) statement will be true in the second iteration, causing the break statement to execute and terminate the loop.

Chapter 7
Section 7-1
1. Methods eliminate the need for duplicate statements in a program.
   Methods make the program easier to design.
   Methods make the program easier to code and test.
   Methods allow for software reusability.
   Methods make the program more clear and readable.
   Methods provide the basis for classes in OOP.
2. non-void, void
3. header, body
4. return
5. what the method accepts and what the method returns to the calling program
6. The method header provides the data interface for the method.
7. optional modifiers, return type, method name, and parameter list
8. parameter
9. to return both a value and control to the calling program
10. An argument is used in the method call, whereas a parameter is used in the method header and receives the value of the corresponding argument when the method is called.
11. public double loanPayment(double amount, double interestRate, double term)
12. public int wheels()
13. The term message is used to describe a call to an instance method with the idea that when we are calling a method, we are sending a message to the class object in which it is defined.

Section 7-2
1. void
2. what the method accepts and what the method returns to the calling program
3. by listing the method name and any required arguments as a statement within the calling program and calling it with an object reference if it is a public method of a class
4. False, a non-void method can be called with an assignment operator, not a void method.
5. to return control to the calling program
6. public void displayEmployee()

Section 7-3
1. In the class from which the object calling the method was created.
2. just after the opening brace of main(), or any place prior to the method call in main()
3. just after the opening brace of the method, or any place prior to their use within the method
4. Two, the class in which the method is defined and the application test class that contains main().
5. The file must have the same name as the application class name, with a .java extension.

Section 7-4
1. the number or types of arguments provided in the method call
2. so that the same operation can behave differently depending on what arguments are provided
3. False, the compiler can only distinguish between parameter lists of overloaded methods.
4. polymorphism
Section 7-5
1. False, because Java supports recursion, which allows a method to call itself.
2. We can describe recursion as a “winding” and “unwinding” process because recursion “winds up” when it places information onto a stack, and “unwinds” by removing information from the stack until the terminating condition is reached.
3. a terminating condition
4. If $n$ is 0
   \[ \text{factorial}(n) = 1. \]
   Else
   \[ \text{factorial}(n) = n \times \text{factorial}(n - 1) \]
5. false, because recursion uses large amounts of memory to keep track of each recursive call in a memory stack
6. true
7. stack
8. last-in, first-out, or LIFO

Chapter 8
Section 8-1
1. By a class defining the behavior of its objects, we mean how an object of a given class acts and reacts when it is accessed via one of its methods.
2. False, because encapsulation dictates only that the data and/or methods are packaged together in a well-defined unit. Encapsulation does ensure information hiding in an OO language, but not in other, non-OO languages.
3. true
4. encapsulation with information, or data, hiding
5. true
6. private
7. implementation

Section 8-2
1. constructor
2. The constructor has the same name as the class and has no return value.
3. false, because constructors must not have a return type, not even void
4. A class constructor is called automatically when an object is defined for that class.
5. false, because any method can be overloaded, including a constructor
6. You call a non-constructor public method of a class by listing the class object name, a dot, and the method name with any required arguments.
7. public MyClass(int value1, int value2)
8. MyClass myObject = new MyClass(5,6);

Chapter 9

Section 9-1

1. superclass
2. subclass
3. family of classes
4. Two reasons for using inheritance are:
   - Inheritance allows you to reuse code without having to start from scratch.
   - Inheritance allows you to build a family hierarchy among classes.
5. IS-A
6. false, because a line IS NOT A point
7. true, because a pixel IS-A point
8. true, because a pickup truck IS-A truck

Section 9-2

1. true
2. super
3. false, because a protected member is inherited by any subclasses of the class in which it resides
4. A private member is only visible within the class in which it is defined.
   A protected member is visible in any subclasses of the class in which it is defined.
   A public member is visible anywhere the class object is visible.
5. The reason that the Savings class should not be derived from the InterestChecking class is that a saving account is not a form of checking account. Such a derivation would violate the IS-A principle.
6. AndrewStaugaard: accountNumber = "0000:
   name = " 
   balance = 0.0
   minimum = 500.0
   charge = 0.5

MichaelJordan: accountNumber = "0010"
   name = "Michael Jordan"
   balance = 10000.0
   minimum = 500.0
   charge = 0.5

PaulAllen: accountNumber = "0020"
   name = "Paul Allen"
Chapter 10

Section 9-3
1. true
2. false, because overloaded methods are statically bound at compile time.
3. true
4. statically
5. dynamically
6. run, or execution, time
7. An abstract class is a superclass for which no objects will ever be created. A concrete class is a subclass for which objects will always be created.
8. true
9. In the subclasses of the class in which the abstract method header appears.

Chapter 10

Section 10-1
1. index and elements
2. false, because the elements within a given array must all be of the same data type or class
3. $1 \times 11$

Section 10-2
1. double testScores[] = new double[15];
2. $1 \times 15$
3. [0]
4. [14]
5. 26, because the first element index is [0]
6. int values[] = { -3, -2, -1, 0, 1, 2, 3};
7. $1 \times 7$
8. String family[] = {"Andy", "Janet", "Ron", "David", "Zane", "Andrew"};
9. zeros
10. blanks

Section 10-3
1. for(int index = 0; index < 15; ++index)
   characters[index] = io.readChar();
(Assumes that io has been defined as an object of the StaugIO class.)
2. for (int index = 0; index < 15; ++index)
   info += characters[index] + "\n"
   io.writeInfo(info);
(assumes that io has been defined as an object of the StaugIO class)

Section 10-4
1. False, an array can be defined using a variable for the size of the array.
2. Employee employees[] = new Employee[10];
3. for(int i = 0; i < 10; ++i)
   
   employees[i] = new Employee();
   employees[i].setName();
   employees[i].setNumber();
   employees[i].setDept();
   
   }//END FOR
4. for(int i = 0; i < 10; ++i)
   
   employInfo = "EMPLOYEE " + (i+1) + " DATA\n"
   employInfo += "\nNAME: " + employees[i].getName() + "\n"
   + "NUMBER: " + employees[i].getNumber() + "\n"
   + "DEPARTMENT: " + employees[i].getDept() + "\n"
   
   io.writeInfo(studentInfo);
   
   }//END FOR

Section 10-5
1. true
2. public void sample(char array[])
3. myObject.sample(characters);
4. public void test(char arrayElement)
5. myObject.test(characters[5]);

Chapter 11
Section 11-1
1. javax.swing
2. a container created from the Container class which has been placed on a frame
   created from the JFrame class
3. JFrame myFrame = new JFrame("This is my first frame");
4. myFrame.setSize(450,250);
5. myFrame.show();
6. Create a GUI component object, add a listener for the object, and implement the corresponding event method to process an event generated by the object.

7. `windowClosing()`

8. `addWindowListener(this);` or `addWindowListener(new WindowHandler());`
   if a `WindowHandler` class is being used.

9. `java.awt.event`

10. An **inner class** is a class which is nested, or contained, inside another class which is usually designated as a **private** class.

11. An **adapter class** is a standard class that implements all the methods in a given standard abstract class.

### Section 11-2

1. `JButton myButton = new JButton("Store");`
2. `container.add(myButton);`
3. `myButton.addActionListener(new ButtonHandler());`
4. `actionPerformed()`
5. `JOptionPane.showMessageDialog(null, "ERROR! ERROR! ERROR!", "INFORMATION", JOptionPane.INFORMATION_MESSAGE);`

### Section 11-3

1. `FlowLayout`
2. `GridLayout`
3. `topPanel.setLayout(new FlowLayout(FlowLayout.LEFT,20,10));`
4. `North, South, East, West, Center`
5. `container.setLayout(null);`
6. at the top left-hand corner of the window
7. `myButton.setLocation(20,30);`
8. `myButton.setSize(50,25);`
9. `getActionCommand()`
10. `bottomPanel.setLayout(new BorderLayout(20,20));`
11. `bottomPanel.add(centerPanel,"Center");`
12. False, the default layout manager is `BorderLayout`.

### Section 11-4

1. `JLabel myLabel = new JLabel("This is my label");`
   `myPanel.add(myLabel);`
2. `JTextField myTextField = new JTextField(20);`
3. JTextArea myTextArea = new JTextArea(10, 50);
4. myPanel.add(myTextField);
   myPanel.add(myTextArea);

Chapter 12
Section 12-1
1. A combo box only allows for one item selection, while a list box allows for multiple item selection, when enabled.
2. isSelected()
3. to act as a container for menu objects
4. JMenuBar myMenuBar = new JMenuBar();
5. JMenu myMenu = new JMenu("My Menu");
6. myMenuBar.add(myMenu);
7. setJMenuBar(myMenuBar);
8. the font type, or name, the font style, and the font size
9. Approximately 1/72 inch.
10. Font myFont = new Font("Arial", Font.BOLD, 16);
11. myTextArea.setFont(myFont);

Section 12-2
1. The Box class is used to implement the BoxLayout manager. A box is like a panel in that it is a container class that can be used to hold other GUI components.
2. BoxLayout arranges GUI components from left to right in a single horizontal row, or from top to bottom in a single vertical column.
3. False, components in a box will not wrap around when the GUI window is resized.
4. A card layout allows you to build a card deck as a panel, consisting of several unique GUI “cards” as panels. When the card deck is first shown, the first card (panel) in the deck appears. From there you can display any of the other cards (panels) within the deck as a separate GUI.
5. first(), next(), previous(), last(), show()
4. a status bar
5. `init()`
6. `paint()`
7. an object of the `Graphics` class
8. `<APPLET codebase=.. code="myApplet.class" width=350 height=200>`
   `</APPLET>`

**Section 13-2**

1. `java.awt`
2. When the “fill” method is called, the component is filled with the color that has been set for the graphics object, `g`. When the “draw” method is called, the object appears as a cartoon-like line drawing whose outline is the color of the graphics object, and whose interior is the color of the applet background.
3. `oval`
4. `R = 0, G = 0, B = 0`
5. `true`
6. A “3D” rectangle appears either raised or sunken.
7. the upper left corner of an imaginary bounding rectangle
8. `three`
9. `true`
10. two arrays, one to store the x-coordinates and one to store the y-coordinates
11. Unlike a polygon, a polyline does not connect the last coordinate to the point of origin.
12. `repaint()`

**Section 13-3**

1. `init(), paint(), start(), stop(), and destroy()`
2. `init()`
3. `stop()`
4. `true`
5. `true`
6. False, panels and layout managers can be used in an applet just like they are used in an application.

**Chapter 14**

**Section 14-1**

1. false, because a file is a sequential access data structure
2. 00000000 01100011
3. When a numeric value is stored in a text file, it must be converted to its machine representation before it can be manipulated arithmetically, after being read by a program.

4. stream
5. System.out
6. FileOutputStream
7. DataOutputStream
8. FileWriter
9. PrintWriter
10. java.io

Section 14-2
1. A communication path, or stream, is established between the logical program object and the physical disk file.
2. DataOutputStream myFile =
   new DataOutputStream(new FileOutputStream("myFile.dat"));
3. DataInputStream yourFile =
   new DataInputStream(new FileInputStream(yourFileName));

Section 14-3
1. exception handling
2. Exception handling is required when opening files to catch a file-not-found or general I/O error.
3. checked
4. true
5. false, because exceptions that are not caught do not cause GUI programs to terminate prematurely
6. try
7. true
8. ArithmeticException
9. false, because a finally block always executes regardless of whether or not an exception is thrown or caught.
10. true

Section 14-4
1. false, because the DataInputStream method provides methods to read binary files.
2. writeUTF()
3. myOutputFile.writeUTF(myName.getText());
4. displayText.append(myInputFile.readUTF());
5. `myOutputFile.writeInt((new 
   Integer(myAge.getText().trim())).intValue());`
6. `displayText.append(String.valueOf(myInputFile.readInt()));`
7. `close`
8. The `flush()` method flushes the output stream causing any internally buffered data to be written to the file.

Section 14-5

1. `PrintWriter myFile = new PrintWriter((
   new BufferedWriter(
   new FileWriter("myFile.txt"))));`
2. `BufferedReader yourFile =
   new BufferedReader(new FileReader(yourFileName));`

Section 14-6

1. true
2. true
3. `myTextFile.print(myText.getText());`
4. `String line;
   while((line = myTextFile.readLine()) != null)
   myText.append(line + '\n');`
5. The `read()` method returns a single character, while the `readln()` method returns a string.
7. `BufferedReader` and `BufferedWriter`.

Chapter 15

Section 15-1

1. The maximum row index is 9, and the maximum column index is 14.
2. The arrays dimension is 10×15 and it will store 150 double floating-point elements.
3. `sample[0][14] = io.readDouble("Enter a decimal value ");`
4. `System.out.print(sample[1][2]);`
5. `for(int row = 0; row < 10; ++row)
   {
      for(int col = 0; col < 15; ++col)
         System.out.print(sample[row][col] + " ");
      System.out.println();
   } //END row FOR`
Section 15-2

1. A problem that might be encountered when defining large multidimensional arrays is the array size too big error. This means that you are attempting to set aside more memory for the array than a particular computer system can allocate.

2. `int myArray[][][] = new int[10][15][3];`

3. $10 \times 15 \times 3 \times 4 = 1800$ bytes

4. ```java
   for(int plane = 0; plane < 10; ++plane)
   {
      for(int row = 0; row < 15; ++row)
      {
         for(int col = 0; col < 3; ++col)
         {
            System.out.print(myArray[plane][row][col]);
            System.out.println();
         } // END row FOR
      } // END plane FOR
   } // END for loop
```