APPENDIX L

Answers to Checkpoints

Chapter 1

1.1 Because the computer can be programmed to do so many different tasks.

1.2 The Central Processing Unit (CPU), main memory, secondary storage devices, input devices, output devices.

1.3 Arithmetic and Logic Unit (ALU), and Control Unit

1.4 Fetch: The CPU's control unit fetches the program's next instruction from main memory.

Decode: The control unit decodes the instruction, which is encoded in the form of a number. An electrical signal is generated.

Execute: The signal is routed to the appropriate component of the computer, which causes a device to perform an operation.

1.5 A unique number assigned to each section of memory. The address is used to identify a location in memory.

1.6 Program instructions and data are stored in main memory while the program is operating. Main memory is volatile, and loses its contents when power is removed from the computer. Secondary storage holds data for long periods of time—even when there is no power to the computer.

1.7 It means that an operating system is capable of running multiple programs at once.

1.8 A key word has a special purpose, and is defined as part of a programming language. A programmer-defined symbol is a word or name defined by the programmer.

1.9 Operators perform operations on one or more operands. Punctuation symbols mark the beginning or ending of a statement, or separates items in a list.

1.10 A line is a single line as it appears in the body of a program. A statement is a complete instruction that causes the computer to perform an action.

1.11 Because their contents may be changed.

1.12 The original value is overwritten.

1.13 A compiler is a program that translates source code into an executable form.

1.14 Syntax errors are mistakes that the programmer has made that violate the rules of the programming language. These errors must be corrected before the compiler can translate the source code.

1.15 The Java compiler translates Java source code into byte code, which is an intermediate language. The Java Virtual Machine executes the byte code instructions.

1.16 The Java Virtual Machine (JVM) is a program that reads Java byte code instructions and executes them as they are read. In other words, it interprets byte code instructions. You can think of the JVM as a program that simulates a computer whose machine language is Java byte code.

1.17 The program's purpose, input, process, and output.

1.18 Before you create a program on the computer, it is often a good idea to imagine what the computer screen will look like while the program is running. If it helps, draw pictures of the screen, with sample input and output, at various points in the program.

1.19 A cross between human language and a programming language. Pseudocode is especially helpful when designing an algorithm. Although the computer can't understand pseudocode, programmers often find it helpful to write an algorithm in a language that's "almost" a programming language, but still very similar to natural language.

1.20 A compiler translates source code into an executable form.

1.21 A runtime error is an error that occurs while the program is running. These are usually logical errors, such as mathematical mistakes.

1.22 Syntax errors are found by the compiler.

1.23 You can provide sample data and predict what the output should be. If the program does not produce the correct output, a logical error is present in the program.

1.24 Data and the code that operates on the data.

1.25 The data contained in an object.

1.26 The procedures, or behaviors, that an object performs.

1.27 Encapsulation refers to the combining of data and code into a single object.

1.28 Data hiding refers to an object's ability to hide its data from code that is outside the object. Only the object's methods may then directly access and make changes to the object's data. An object typically hides its data, but allows outside code to access the methods that operate on the data.

Chapter 2

2.1
```
// A crazy mixed up program
public class Columbus
{
   public static void main(String[] args)
   {
      System.out.println("In 1492 Columbus sailed the ocean blue.");
   }
}
```

2.2 Columbus.java

2.3
```
public class Hello
{
   public static void main(String[] args)
   {
      System.out.println("Hello World");
   }
}
```

2.4
```
// Example
// August 22, 2018
public class MyName
{
   public static void main(String[] args)
   {
      System.out.println("Herbert Dorfmann");
   }
}
```

2.5 C

2.6 A

2.7
```
// Its a mad, mad program
public class Success
{
   public static void main(String[] args)
   {
      System.out.print("Success\n");
      System.out.print("Success ");
      System.out.print("Success\n");
      System.out.println("\nSuccess");
   }
}
```

2.8
```
The works of Wolfgang
include the following
The Turkish March and Symphony No. 40 in G minor.
```

2.9
```
// August 22, 2018
public class PersonalInfo
{
   public static void main(String[] args)
   {
      System.out.println("Herbert Dorfmann");
      System.out.println("123 Elm Street");
      System.out.println("My Town, NC 21111");
      System.out.println("919-555-1234");
   }
}
```

2.10 *Variables:*
`little`
`big`
Literals:
`2`
`2000`
`"The little number is "`
`"The big number is "`

2.11 `The value is number`

2.12 `99bottles` is illegal because it starts with a number.
`r&d` is illegal because the `&` character is illegal.

2.13 They are not the same because one begins with an uppercase S while the other begins with a lowercase s. Variable names are case-sensitive.

2.14 a. `short`
b. `int`
c. 22.1 because it is stored as a `double`.

2.15 6.31E17

2.16 Append the `F` suffix to the numeric literal, such as:

```
number = 7.4F;
```

2.17 `true` and `false`

2.18 a. `char letter;`
b. `letter = 'A';`
c. `System.out.println(letter);`

2.19 The code for 'C' is 67.
The code for 'F' is 70.
The code for 'W' is 87.

2.20 `'B'` is a character literal.

2.21 You cannot assign a string literal to a `char` variable. The statement should be:

```
char letter = 'Z';
```

2.22

Expression	Value
`6 + 3 * 5`	21
`12 / 2 - 4`	2
`9 + 14 * 2 - 6`	31
`5 + 19 % 3 - 1`	5
`(6 + 2) * 3`	24
`14 / (11 - 4)`	2
`9 + 12 * (8 - 3)`	69

2.23 Integer division. The value 23.0 will be stored in `portion`.

2.24
a. `x += 6;`
b. `amount -= 4;`
c. `y *= 4;`
d. `total /= 27;`
e. `x %= 7;`

2.25
a. No
b. Because the result of `basePay + bonus` results in an `int` value, which cannot be stored in the `short` variable `totalPay`. You can fix the problem by declaring `totalPay` as an `int`, or casting the result of the expression to a `short`.

2.26
```
a = (float)b;
```

2.27
```
String city = "San Francisco";
```

2.28
```
stringLength = city.length();
```

2.29
```
oneChar = city.charAt(0);
```

2.30
```
upperCity = city.toUpperCase();
```

2.31
```
lowerCity = city.toLowerCase();
```

2.32 To write a single line comment you begin the comment with `//`. To write a multi-line comment you begin the comment with `/*` and end it with `*/`. To write a documentation comment you begin the comment with `/**` and end it with `*/`.

2.33 Documentation comments can be read and processed by a program named `javadoc`. The `javadoc` program can read Java source code files and generate attractively formatted HTML documentation files. If the source code files contain any documentation comments, the information in the comments becomes part of the HTML documentation.

2.34 A message dialog box is used to display a message to the user. An input dialog box is used to gather keyboard input from the user.

2.35
a. `JOptionPane.showMessageDialog(null, "Greetings Earthling.");`
b. `str = JOptionPane.showInputDialog("Enter a number.");`

2.36
```
String str;
int age;
str = JOptionPane.showInputDialog("Enter your age.");
age = Integer.parseInt(str);
```

2.37
```
import javax.swing.JOptionPane;
```

Chapter 3

3.1
```
if (y == 20)
   x = 0;
```

3.2
```
if (hours > 40)
   payRate *= 1.5;
```

3.3
```
if (sales >= 10000)
   commission = 0.2;
```

3.4
```
if (max)
   fees = 50;
```

3.5
```
if (x > 100)
{
   y = 20;
   z = 40;
}
```

3.6
```
if (a < 10)
{
   b = 0;
   c = 1;
}
```

3.7
```
if (myCharacter == 'D')
   System.out.println("Goodbye");
```

3.8
```
if (x > 100)
   y = 20;
else
   y = 0;
```

3.9
```
if (y == 100)
   x = 1;
else
   x = 0;
```

3.10
```
if (sales >= 50000.0)
   commission = 0.2;
else
   commission = 0.1;
```

3.11
```
if (a < 10)
{
   b = 0;
   c = 1;
}
else
{
   b = -99;
   c = 0;
}
```

3.12 `1 1`

3.13

If the customer purchases this many books . . .	this many coupons are given.
1	1
2	1
3	2
4	2
5	3
10	3

3.14
```
if (amount1 > 10)
   {
     if (amount2 < 100)
     {
        if (amount1 > amount2)
           System.out.println(amount1);
         else
           System.out.println(amount2);
     }
   }
```

3.15
```
if (x > 0)
{
   if (y < 20)
   {
      z = 1;
   }
   else
   {
      z = 0;
   }
}
```

3.16

Logical Expression	**Result (true or false)**
`true && false`	`false`
`true && true`	`true`
`false && true`	`false`
`false && false`	`false`
`true || false`	`true`
`true || true`	`true`
`false || true`	`true`
`false || false`	`false`
`!true`	`false`
`!false`	`true`

3.17 T, F, T, T, T

3.18
```
if (speed >= 0 && speed <= 200)
   System.out.println("The number is valid");
```

3.19
```
if (speed < 0 || speed > 200)
   System.out.println("The number is not valid");
```

3.20
```
if (name.equals("Timothy"))
   System.out.println("Do I know you?");
```

3.21
```
if (name1.compareTo(name2) < 0)
   System.out.println(name1 + " " + name2);
else
   System.out.println(name2 + " " + name1);
```

3.22
```
if (name.equalsIgnoreCase("Timothy"))
   System.out.println("Do I know you?");
```

3.23
```
a. z = x > y ? 1 : 20;
b. population = temp > 45 ? base * 10 : base * 2;
c. wages = hours > 40 ? wages * 1.5 : wages * 1;
d. System.out.println(result >=0 ? "The result is positive" :
                                   "The result is negative");
```

3.24
```
// Here is the switch statement.
switch(userNum)
{
   case 1 : System.out.println("One");
            break;
   case 2 : System.out.println("Two");
            break;
   case 3 : System.out.println("Three");
            break;
   default: System.out.println("Error: invalid number.");
}
```

3.25
```
switch(selection)
{
   case 'A' : System.out.println("You selected A.");
              break;
   case 'B' : System.out.println("You selected B.");
              break;
   case 'C' : System.out.println("You selected C.");
              break;
   case 'D' : System.out.println("You selected D.");
              break;
   default : System.out.println("Not good with letters, eh?");
}
```

3.26 Because it uses greater-than and less-than operators in the comparisons.

3.27 The case expressions must be a literal or a `final` variable, which must be of the `char`, `byte`, `short`, `int`, or `String` types. In this code, relational expressions are used.

3.28
```
That is serious.
```

3.29
```
System.out.printf("%,.2f", number);
```

3.30
```
System.out.printf("%10.1f", number);
```

3.31
```
System.out.printf("%08.1f", number);
```

3.32
```
System.out.printf("%,10d", number);
```

3.33
```
System.out.printf("%-,20.2f", number);
```

3.34
```
System.out.printf("%20s", name);
```

Chapter 4

4.1 a. 2
b. 2
c. 1
d. 8

4.2 0 times

4.3 This must be a trick question. The statement that prints "I love Java programming!" is not in the body of the loop. The only statement that is in the body of the loop is the one that prints "Hello World". Because there is no code in the loop to change the contents of the `count` variable, the loop will execute infinitely, printing "Hello World". So, "I love Java programming!" is never printed.

4.4
```
Scanner keyboard = new Scanner(System.in);
System.out.print("Enter a number in the range of 10 - 24: ");
number = keyboard.nextInt();
while (number < 10 || number > 24)
{
    System.out.println("That number is not in the range.");
    System.out.print("Enter a number in the range of 10 – 24: ");
    number = keyboard.nextInt();
}
```

4.5
```
String input;
Scanner keyboard = new Scanner(System.in);
System.out.print("Enter Y, y, N, or n: ");
input = keyboard.nextLine();
ch = input.charAt(0);
while (ch != 'Y' && ch != 'y' && ch != 'N' && ch != 'n')
{
    System.out.println("Try again.");
    System.out.print("Enter Y, y, N, or n: ");
    input = keyboard.nextLine();
    ch = input.charAt(0);
}
```

4.6
```
String str;
Scanner keyboard = new Scanner(System.in);
System.out.print("Enter Yes or No: ");
str = keyboard.nextLine();
while ((!str.equals("Yes")) && (!str.equals("No")))
{
    System.out.print("Please enter Yes or No: ");
    str = keyboard.nextLine();
}
```

4.7 Initialization, test, and update.

4.8
a. `count = 1`
b. `count <= 50`
c. `count++`
d.
```
for (int count = 1; count <= 50; count++)
   System.out.println("I love to program");
```

4.9 a.
```
 0
 2
 4
 6
 8
10
```

b.
```
-5
-4
-3
-2
-1
 0
 1
 2
 3
 4
```

c.
```
 5
 8
11
14
17
```

4.10
```
// This is how Chloe Ashlyn would write the loop.
for (int i = 1; i <= 10; i++)
   System.out.println("Chloe Ashlyn");
```

4.11
```
for (int i = 1; i <= 49; i += 2)
   System.out.println(i);
```

4.12
```
for (int i = 0; i <= 100; i += 5)
   System.out.println(i);
```

4.13
```
Scanner keyboard = new Scanner(System.in);
int number = 0, total = 0;
for (int i = 1; i <= 7; i++)
{
   System.out.print("Enter a number: ");
   number = keyboard.nextInt();
   total += number;
}
```

4.14 The variable `x` is the loop control variable and `y` is the accumulator.

4.15 You should be careful to choose a value that cannot be mistaken as a valid input value.

4.16 Data is read from an input file, and data is written to an output file.

4.17
```
import java.io.*;
```

4.18 `PrintWriter`

4.19
```
// This is how Kathryn would write the code.
PrintWriter outputFile = new PrintWriter("MyName.txt");
outputFile.println("Kathryn");
outputFile.close();
```

4.20 `File` and `Scanner`

4.21
```
File file = new File("MyName.txt");
Scanner inputFile = new Scanner(file);
if (inputFile.hasNext())
{
   String str = inputFile.nextLine();
   System.out.println(str);
}
inputFile.close();
```

4.22 You create an instance of the `FileWriter` class to open the file. You pass the name of the file (a string) as the constructor's first argument, and the `boolean` value `true` as the second argument. Then, when you create an instance of the `PrintWriter` class, you pass a reference to the `FileWriter` object as an argument to the `PrintWriter` constructor. The file will not be erased if it already exists and new data will be written to the end of the file.

4.23 `throws IOException`

4.24 It assigns the variable `x` a random integer in the range −2,147,483,648 to +2,147,483,647.

4.25 It assigns the variable `x` a random integer in the range 0 through 99.

4.26 It assigns the variable `x` a random integer in the range 1 through 9.

4.27 It assigns the variable `x` a random number in the range 0.0 to 1.0.

Chapter 5

5.1 A value-returning method returns a value back to the code that called it, and a `void` method does not.

5.2 Method call

5.3 Method header

5.4 If the user enters 5 the program will display *Able was I*. If the user enters 10 the program will display *I saw Elba*. If the user enters 100 the program will display *I saw Elba*.

5.5
```
// This is how Mary Catherine Jones would write it.
public static void myName()
{
   System.out.println("Mary Catherine Jones");
}
```

5.6 An argument is a value that is passed into a method. A parameter is a special variable that holds the value being passed into the method.

5.7 b and c will cause a compiler error because the values being sent as arguments (a `double` and a `long`) cannot be automatically converted to an `int`.

5.8 Only d is written correctly.

5.9 Only a copy of an argument's value is passed into a parameter variable. A method's parameter variables are separate and distinct from the arguments that are listed inside the parentheses of a method call. If a parameter variable is changed inside a method, it has no effect on the original argument.

5.10
```
99 1.5
99 1.5
0 0.0
99 1.5
```

5.11 `double`

5.12 `public static int days(int years, int months, int weeks)`

5.13 `public static double distance(double rate, double time)`

5.14 `public static long lightYears(long miles)`

Chapter 6

6.1 To store data

6.2 Operations that the object can perform

6.3 It serves a purpose similar to that of the blueprint for a house. The blueprint itself is not a house, but rather a detailed description of a house. When we use the blueprint to build an actual house, we could say that we are building an instance of the house described by the blueprint. A class is not an object, but a description of an object. When a program is running, it can use the class to create, in memory,

as many objects of a specific type as needed. Each object that is created from a class is called an instance of the class.

6.4 In the Java API

6.5 The `new` operator creates an object in memory, and returns that object's memory address.

6.6 It holds an object's memory address.

6.7 In order to fly a kite, you need a spool of string attached to it. When the kite is airborne, you use the spool of string to hold on to the kite and control it. This is similar to the relationship between an object and the variable that references the object. Think of the object as the kite, and the reference variable as the spool of string.

6.8 Classes are the blueprints.

6.9 The kite represents an object.

6.10 The memory address of the object.

6.11 A `String` object.

6.12
a. `Car`
b. `make` and `yearModel`
c. `setMake`, `setYearModel`, `getMake`, and `getYearModel`
d. `make` and `yearModel`
e. `setMake`, `setYearModel`, `getMake`, and `getYearModel`

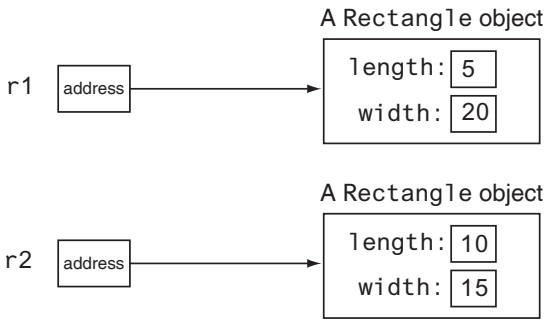
6.13
```
limo.setMake("Cadillac");
```

6.14 Creates an instance of an object in memory.

6.15 An accessor is a method that gets a value from a class's field but does not change it. A mutator is a method that stores a value in a field or in some other way changes the value of a field.

6.16 When the value of an item is dependent on other data and that item is not updated when the other data is changed, it is said that the item has become stale.

6.17



6.18 It has the same name as the class.

6.19 It has no return type, not even `void`.

6.20 a. `ClassAct`
b. `ClassAct myact = new ClassAct(25);`

6.21 Overloaded methods must have different parameter lists. Their return types do not matter.

6.22 A method's signature consists of the method's name and the data types of the method's parameters, in the order that they appear.

6.23
```
This is the second version of the method.
This is the first version of the method.
```

6.24 Only one.

6.25 The problem domain is the set of real-world objects, parties, and major events related to a problem.

6.26 Someone who has an adequate understanding of the problem. If you adequately understand the nature of the problem you are trying to solve, you can write a description of the problem domain yourself. If you do not thoroughly understand the nature of the problem, you should have an expert write the description for you.

6.27 Identify all the nouns (including pronouns and noun phrases) in the problem domain description. Each of these is a potential class. Then, refine the list to include only the classes that are relevant to the problem.

6.28 A class's responsibilities are the things that the class is responsible for knowing and the actions that the class is responsible for doing.

6.29 It is often helpful to ask the questions "In the context of this problem, what must the class know? What must the class do?"

6.30 No. Often responsibilities are discovered through brainstorming.

Chapter 7

7.1 a. `int[] employeeNumbers = new int[100];`
b. `double[] payRates = new double[25];`
c. `float[] miles = new float[14];`
d. `char[] letters = new char[1000];`

7.2 An array's size declarator must be a non-negative integer expression. The first statement is incorrect because the size declarator is negative. The second statement is incorrect because the size declarator is a floating-point number.

7.3 0 through 3

7.4 The size declarator specifies the number of elements in the array. A subscript identifies a specific element in the array.

7.5 The subscript is outside the range of valid subscripts for the array.

7.6 When the statement executes, it crashes the program and displays a runtime error message.

7.7
```
1
2
3
4
5
```

7.8
```
double[] array = { 1.7, 6.4, 8.9, 3.1, 9.2 };
```
There are five elements in the array.

7.9
```
result = numbers1[0] * numbers2[3];
```

7.10
```
for (int i = 0; i < array.length; i++)
   array[i] = -1;
```

7.11
```
// Assume keyboard references a Scanner object.
int size;
System.out.print("Enter the size of the array: ");
size = keyboard.nextInt();
values = new double[size];
```

7.12
```
for (int i = 0; i < a.length; i++)
   b[i] = a[i];
```

7.13
```
myMethod(numbers);
```

7.14
```
public static void zero(int[] array)
{
   for (int i = 0; i < array.length; i++)
      array[i] = 0;
}
```

7.15
a.
```
String[] planets = { "Mercury", "Venus", "Earth", "Mars" };
```
b.
```
for (int i = 0; i < planets.length; i++)
   System.out.println(planets[i]);
```
c.
```
for (int i = 0; i < planets.length; i++)
   System.out.println(planets[i].charAt(0));
```

7.16
```
Rectangle[] rectArray = new Rectangle[5];
for (int i = 0; i <= rectArray.length; i++)
{
   // Initialize each rectangle with the values
   // i and i+1 for length and width.
   rectArray[i] = new Rectangle(i, i+1);
}
```

7.17
```
final int RACKS = 50;
final int SHELVES = 10;
final int VIDEOS = 25;
// Create an array to hold video numbers.
int videoNumbers[][][] = new int[RACKS][SHELVES][VIDEOS];
```

7.18 The selection sort first looks for the smallest value in the array. When it finds it, it moves it to element 0.

7.19 Only once.

7.20 The sequential search steps through each element of the array, starting at element 0, looking for the search value. The binary search requires that the array be sorted in ascending order. It starts by looking at the middle element. If it is not the search value, and is greater than the search value, then the lower half of the array is searched next. If the middle element is not the search value, and is less than the search value, the upper half of the array is searched next. This same technique is repeated on the half of the array being searched until the element is either found or there are no more elements to search.

7.21 10,000 comparisons

7.22 Move the items that are frequently searched for to the beginning of the array.

7.23
```
import java.util.ArrayList;
```

7.24
```
ArrayList frogs = new ArrayList();
```

7.25
```
ArrayList<String> lizards = new ArrayList<String>();
```

7.26 You use the `add` method.

7.27 The `ArrayList` class has a `remove` method that removes an item at a specific index. You pass the index as an argument to the method.

7.28 The `ArrayList` class has a `get` method that retrieves an item at a specific index. You pass the index as an argument to the method.

7.29 Inserting means adding an item at a specific index. The `ArrayList` class has an overloaded version of the `add` method that allows you to add an item at a specific index.

7.30 The `ArrayList` class has a `size` method that returns the number of items stored in the `ArrayList`.

7.31 An `ArrayList`'s size is the number of items stored in the `ArrayList` object. An `ArrayList`'s capacity is the number of items the `ArrayList` object can hold without having to increase its size.

Chapter 8

8.1 Each instance of a class has its own copy of the class's instance fields. A static field does not belong to any instance of the class, and there is only one copy of a static field in memory, regardless of the number of instances of the class.

8.2 It isn't necessary for an instance of the class to be created in order to execute the method.

8.3 They cannot directly refer to non-static members of the class. This means that any method called from a static method must also be static. It also means that if the method uses any of the class's fields, they must be static as well.

8.4 The `this` variable will reference the `stock2` object.

8.5 a. `Flower`
b. The ordinal value for `Rose` is 0. The ordinal value for `DAISY` is 1. The ordinal value for `PETUNIA` is 2.
c. `Flower.Rose`, `Flower.DAISY`, `Flower.PETUNIA`
d. `Flower flora = Flower.PETUNIA;`

8.6 `HOBBIT ELF DRAGON`

8.7 `Z is not greater than X.`

Chapter 9

9.1
```
little = Character.toLowerCase(big);
```

9.2
```
if (Character.isDigit(ch))
   System.out.println("digit");
else
   System.out.println("Not a digit");
```

9.3 `A`

9.4
```
String input;
char ch;
Scanner keyboard = new Scanner(System.in);
System.out.print("Do you want to repeat the " +
                 "program or quit? (R/Q)");
input = keyboard.nextLine();
ch = input.charAt(0);
ch = Character.toUpperCase(ch);
while (ch != 'R' && ch != 'Q')
{
   System.out.print("Do you want to repeat the " +
                    "program or quit? (R/Q)");
   input = keyboard.nextLine();
   ch = input.charAt(0);
   ch = Character.toUpperCase(ch);
}
```

9.5 `$`

9.6
```
int total = 0;
for (int i = 0; i < str.length(); i++)
{
   if (Character.isUpperCase(str.charAt(i)))
      total++;
}
```

9.7
```
public static boolean endsWithGer(String str)
{
   boolean status;

   if (str.endsWith("ger"))
      status = true;
   else
      status = false;
   return status;
}
```

9.8
```
public static boolean endsWithGer(String str)
{
   boolean status;
   String strUpper = str.toUpperCase();

   if (strUpper.endsWith("GER"))
      status = true;
   else
      status = false;
   return status;
}
```

9.9 You would use the `substring` method.

9.10 The `indexOf` method searches for a character or substring, starting at the beginning of a string. The `lastIndexOf` method searches for a character or substring, starting at the end of a string. This means that the `indexOf` method returns the index of the first occurrence of a character or substring, and the `lastIndexOf` method returns the index of the last occurrence of a character or substring.

9.11 The `substring` method returns a reference to a substring. The `getChars` method stores a substring in a `char` array.

9.12 The `concat` method.

9.13 The `toCharArray` method returns all of the characters in the calling object as a `char` array.

9.14 The fellow student is wrong. The `replace` method will return a reference to a `String` which is a copy of `str1`, in which all of the `o` characters have been replaced with `u` characters. The original string in `str1` will not be changed, however. The code will produce the following output:

```
To be, or not to be
Tu be, ur nut tu be
```

9.15 `WilliamtheConqueror`

9.16 `str = String.valueOf(number);`

9.17 Once you create a `String` object, you cannot change the object's value.

9.18 It would be more efficient to use `StringBuilder` objects because they are not immutable. Making a change to a `StringBuilder` object does not cause a new object to be created.

9.19 `StringBuilder city = new StringBuilder("Asheville");`

9.20 The `append` method.

9.21 The `insert` method.

9.22 The `deleteCharAt` method.

9.23 The `setCharAt` method.

9.24 While the `String` class's `replace` method replaces the occurrences of one character with another character, the `StringBuilder` class's `replace` method replaces a specified substring with a string.

9.25 The tokens are "apples", "pears", and "bananas". The delimiter is the space character.

9.26 `4` will be stored in `x`.

`"one"` will be stored in `stuff`.

9.27
```
String str = "/home/rjones/mydata.txt"
String[] tokens = str.split("/");
```

9.28
```
String str = "dog$cat@bird%squirrel"
String[] tokens = str.split("[$@%]");
```

9.29 `str = Integer.toString(i);`

9.30 The `toBinaryString`, `toHexString`, and `toOctalString` methods are static members of the `Integer` and `Long` wrapper classes.

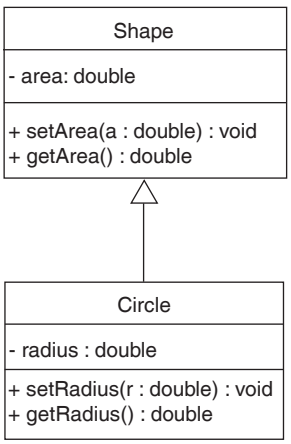
9.31 These variables hold the minimum and maximum values for a particular data type.

Chapter 10

10.1 `Vehicle` is the superclass and `Truck` is the subclass.

10.2 a. `Shape` is the superclass, `Circle` is the subclass.

b.



c. `setArea`, `getArea`, `setRadius`, `getradius`
d. `area`
e.

`c.setRadius(10.0);`	legal
`s.setRadius(10.0);`	illegal
`System.out.println(c.getArea());`	legal
`System.out.println(s.getArea());`	legal

10.3 The superclass constructor (class `A`) will execute first, then the subclass constructor (class `B`) will execute.

10.4
```
You are on the ground.
You are in the sky.
```

10.5
```
The ground is green
The sky is blue
```

10.6 When the superclass method is inadequate for the subclass's purpose.

10.7 A subclass may call an overridden superclass method by prefixing its name with the `super` key word and a dot (`.`).

10.8 It overrides the superclass method.

10.9 It overloads the superclass method.

10.10 You declare the method with the `final` modifier.

10.11 Protected members of class may be accessed by methods in a subclass, and by methods in the same package as the class.

10.12 Private members may be accessed only by methods in the same class. A protected member of a class may be directly accessed by methods of the same class or methods of a subclass. In addition, protected members may be accessed by methods of any class that are in the same package as the protected member's class.

10.13 Because any other class that inherits from the class, or is in the same package, has unrestricted access to the protected member.

10.14 Private members may be accessed only by methods in the same class. Any method in the same package as the class may directly access the class's members that have package access.

10.15 When you accidentally leave out the access specifier, the member will have package access.

10.16 `ClassD` still inherits from `Object`, because all classes ultimately inherit from `Object`.

10.17 Because those methods are members of the `Object` class.

10.18 a. Legal, because a `Cube` is a `Rectangle`.
b.

`System.out.println(r.getLength());`	Legal
`System.out.println(r.getWidth());`	Legal
`System.out.println(r.getHeight());`	Illegal
`System.out.println(r.getSurfaceArea());`	Illegal

c. Illegal, because a `Rectangle` is not a `Cube`. The `Cube` class has capabilities beyond those of the `Rectangle` class, so a `Cube` variable cannot reference a `Rectangle` object.

10.19 Abstract methods are used to ensure that a subclass implements the method.

10.20 Override the abstract method.

10.21 An abstract class serves as a superclass for other classes. It represents the generic or abstract form of all the classes that inherit from it.

10.22 An abstract class cannot be instantiated. It must serve as a superclass.

10.23 To specify behavior for other classes.

10.24 It cannot be instantiated.

10.25 An interface only specifies methods; it does not define them. In addition, all members of an interface are public.

10.26 As `final` and `static`.

10.27 `public class Customer implements Relatable`

10.28 `public class Employee implements Payable, Listable`

Chapter 11

11.1 An exception is an object that is generated as the result of an error or an unexpected event.

11.2 To throw an exception means to generate an exception object.

11.3 Unless an exception is detected by the application and dealt with, it causes the application to halt.

11.4 The `Throwable` class.

11.5 Classes that extend the `Error` class are for exceptions that are thrown when a critical error occurs, such as an internal error in the Java Virtual Machine or running out of memory. An application should not try to handle these exceptions. Exception classes that extend the `Exception` class are general exceptions that an application can handle.

11.6 A try block is one or more statements that are executed and can potentially throw an exception. A catch block is code that appears immediately after a `catch` clause, and is executed in response to a particular exception.

11.7 The program will resume with the code that appears after the entire `try/catch` construct.

11.8 Each exception object has a method named `getMessage` that can be used to retrieve the error message for the exception.

11.9 When an exception is thrown by code in the try block, the JVM begins searching the `try` statement for a `catch` clause that can handle it. It searches the `catch` clauses from top to bottom and passes control of the program to the first `catch` clause with a parameter that is compatible with the exception.

11.10 Statements in the finally block are always executed after the try block has executed and after any catch blocks have executed if an exception was thrown. The finally block executes whether an exception is thrown or not.

11.11 A call stack is an internal list of all the methods that are currently executing. A stack trace is a list of all the methods in the call stack.

11.12 Because method B does not handle the exception, control is passed back to method A. Method A doesn't handle the exception either, so control is passed back to method `main`. Because `main` doesn't handle the exception, the program is halted and the default exception handler handles the exception.

11.13 Unchecked exceptions are those that are extended from the `Error` class or the `RuntimeException` class. A program should not attempt to handle unchecked exceptions. All the remaining exceptions are checked exceptions. Either a checked exception must be handled by the program, or applicable `throws` clauses must be used with all of the methods that can potentially throw the checked exception.

11.14 When code in the method can potentially throw a checked exception, but does not handle the exception.

11.15 The `throw` statement causes an exception object to be created and thrown.

11.16 The argument contains a custom error message that can be retrieved from the exception object's `getMessage` method. If you do not pass a message to the constructor, the exception will have a null message.

11.17 The `throw` statement causes an exception to be thrown. The `throws` clause informs the compiler that a method throws one or more exceptions.

11.18 No. If the method throws an unchecked exception, it does not have to have a `throws` clause.

11.19 You can create a checked exception by extending its class from `Exception`. You can create an unchecked exception by extending its class from `Error`.

11.20 All of the data stored in a text file is formatted as text. Numeric data is converted to text when it is stored in a text file. You can view the data stored in a text file by opening it with a text editor, such as Notepad. In a binary file, data is not formatted as text. Subsequently, you cannot view a binary file's contents with a text editor.

11.21 To write data to a binary file you use the `FileOutputStream` and `DataOutputStream` classes. To read data from a file you use the `FileInputStream` and `DataInputStream` classes.

11.22 With sequential access, when a file is opened for input, its read position is at the very beginning of the file. This means that the first time data is read from the file, the data will be read from its beginning. As the reading continues, the file's read position advances sequentially through the file's contents. In random file access, a program may immediately jump to any location in the file without first reading the preceding bytes.

11.23 `RandomAccessFile`

11.24 The two modes are "r" for reading, and "rw" for reading and writing. When a file is opened with "r" as the mode, the program can only read from the file. When a file is opened with "rw" as the mode, the program can read from the file and write to it.

11.25 The class must implement the `Serializable` interface. You can then use the `ObjectOutputStream` class's `writeObject` method to serialize objects of the class.

Chapter 12

12.1 A computer's user interface is the part of the computer with which the user interacts.

12.2 A command line interface, which is also known as a console interface, requires the user to type commands. If a command is typed correctly, it is executed and the results are displayed. If a command is not typed correctly, an error message is displayed.

12.3 In a text-based environment, such as a command line interface, programs determine the order in which things happen.

12.4 A program that responds to the user's actions, such as the clicking of a button.

12.5 A standard Java library for developing rich applications that employ graphics.

12.6 A scene

12.7 A tree-like hierarchical data structure that holds the objects in a GUI.

12.8 Root, branch, and leaf.

12.9 Branch nodes can have children. Leaf nodes cannot have children.

12.10 `Application`

12.11 The `launch` method creates a `Stage` object that will be the application's window. It then calls the `start` method, passing a reference to the `Stage` object as an argument.

12.12 `start` is the entry point for the application.

12.13 Set the text that is to be displayed in the window's title var.

12.14 It displays the window.

12.15 An HBox arranges controls in a single horizontal row. A VBox arranges controls in a single vertical row.

12.16 `GridPane`

12.17 `javafx.scene`

12.18 With the `setAlignment` method

12.19 `javafx.geometry`

12.20 `javafx.scene.image`

12.21 An `ImageView` object

12.22 You call the `ImageView` class's `setFitWidth` and `setFitHeight` methods.

12.23 The `ImageView` class's `setPreserveRatio` method.

12.24 You call the `ImageView` class's `setImage` method.

12.25 Spacing is space that appears between controls that are inside the container, and padding is space that appears around the inside edge of the container.

12.26 `javafx.geometry`

12.27 With the container's `setHgap` and `setVgap` methods

12.28 You call the container's `add` method. The arguments are: the control being added, the column, and the row.

12.29 An event is an action that takes place while a program is running. For example, each time the user clicks a Button control, an event takes place.

12.30 An object that responds to events. If an event source is connected to an event handler, a specific method in the event handler is called and the event object is passed as an argument to the method.

12.31 `javafx.event`

12.32 You call the `Button` class's `setOnAction` method.

12.33 `javafx.scene.control`

12.34 You call the control's `getText` method. The method returns the value that has been entered into the TextField as a `String`.

12.35
- Writing the definition of an inner class that implements the `EventHandler` interface. Then, instantiating that class, and registering it with a control.
- Instantiating an anonymous inner class that implements the `EventHandler` interface, and registering the object with a control.
- Using a lambda expression to instantiate an anonymous inner class that implements the `EventHandler` interface, and registering the object with a control.

12.36 Top, bottom, left, center, and right

12.37 Constructor #1: The no-arg constructor creates an empty BorderPane container.

Constructor #2: This constructor accepts one argument. The node that passed as the argument is placed in the BorderPane's center region.

Constructor #3: This constructor accepts five nodes as arguments: one to place in each region.

Chapter 13

13.1
```
.button
.text-field
.abel
.image-view
```

13.2 `.menu-button`

13.3 `-fx-font-family`

13.4 `-fx-font-weight`

13.5 `-fx-border-style`

13.6 `scene.getStylesheets().add("stylesheet.css");`

13.7 `.root`

13.8 The `.root` definition

13.9 red, green, blue

13.10 blue

13.11 An ID selector

13.12 You add the `RadioButtons` to a `ToggleGroup`. Only one of the `RadioButton` controls in a `ToggleGroup` may be selected at any time.

13.13 The `RadioButton` has a method, `isSelected`, that returns `true` if it is selected, or `false` otherwise.

13.14 Call the `RadioButton`'s `setSelected` method, passing `true` as the argument.

13.15 `ActionEvent`

13.16 The `CheckBox` control has a method, `isSelected`, that returns `true` if it is selected, or `false` otherwise.

13.17 Call the `CheckBox`'s `setSelected` method, passing `true` as the argument.

13.18 `ActionEvent`

13.19 With the control's `setPrefSize` method.

13.20 There are two ways: (1) You can use the `ListView` class's `getSelectionModel().getSelectedItem()` method get the item that is currently selected, or (2) You can use the `ListView` class's `getSelectionModel().getSelectedIndex()` method get the index of the item that is currently selected.

13.21 In single selection mode, only one item can be selected at a time in a ListView control.

13.22 In multiple interval selection mode, multiple items may be selected in a ListView control.

13.23 You can set the orientation to either vertical or horizontal with the `ListView` class's `setOrientation` method. When you call the method, you pass it one of the following `enum` constants: `Orientation.VERTICAL` or `Orientation.HORIZONTAL`.

13.24 `ObservableList`

13.25 You can use the `ComboBox` class's `getValue()` method get the item that is currently selected.

13.26 `ActionEvent`

13.27 An editable `ComboBox` allows the user to select an item from a drop-down list, or type input into a field that is similar to a `TextField`. By default, `ComboBox`

controls are uneditable, which means that the user cannot type input into the control; he or she can only select items from a drop-down list.

13.28 a. `setMajorTickUnit(`*`value`*`)`
b. `setMinorTickCount(`*`value`*`)`
c. `setShowTickMarks(`*`value`*`)`
d. `setShowTickLabels(`*`value`*`)`

13.29 Change event

13.30 You call the Slider's `getValue()` method.

13.31 A `TextField` can accept multiple lines of input.

13.32 40 columns. You can change the number of columns with the `setPrefColumnCount()` method.

13.33 10 rows. You can change the number of rows with the `setPrefRowCount()` method.

13.34 When text is entered into the control and the end of a line is reached, the text wraps around to the next line. You can enable text wrapping with the `TextArea` class's `setWrapText` method.

13.35 You call the control's `getText()` method.

13.36
- **Menu Bar.** At the top of the window, is a menu bar. The *menu bar* lists the names of one or more menus.
- **Menu.** A *menu* is a drop-down list of menu items. The user may activate a menu by clicking on its name on the menu bar. In the figure, the *Edit* menu has been activated.
- **Menu Item.** A *menu item* can be selected by the user. When a menu item is selected, some type of action is usually performed.
- **Check menu item.** A *check menu item* appears may be selected or deselected. When it is selected, a check mark appears next to the item. When it is deselected, the check does not appear. Check menu items are normally used to turn an option on or off. The user toggles the state of a check menu item each time he or she selects it.
- **Radio menu item.** A *radio menu item* may be selected or deselected. A radio menu item looks like a check menu item. When it is selected, a check mark appears next to the item. When it is deselected, the check does not appear. When a set of radio menu items are grouped with a `ToggleGroup` object, only one of them can be selected at a time. When the user selects a radio menu item, the one that was previously selected is deselected.
- **Submenu.** A menu within a menu is called a *submenu*. Some of the commands on a menu are actually the names of submenus. You can tell when a command is the name of a submenu because a small right arrow appears to its right. Activating the name of a submenu causes the submenu to appear.
- **Separator bar.** A separator bar is a horizontal bar used to separate groups of items on a menu. Separator bars are only used as a visual aid and cannot be selected by the user.

13.37 `MenuBar`

13.38 `Menu`

13.39 `MenuItem`

13.40 `RadioMenuItem`. Call the object's `setSelected` method to select it.

13.41 Add them to a `ToggleGroup`.

13.42 `CheckMenuItem`. Call the object's `setSelected` method to select it.

13.43 `ActionEvent`

13.44 `javafx.stage`

13.45 Call the `FileChooser`'s `showOpenDialog` method.

13.46 Call the `FileChooser`'s `showSaveDialog` method.

13.47 The `showOpenDialog` and `showSaveDialog` methods return a `File` object containing information about the selected file.

Chapter 14

14.1 (0,0)

14.2 (639, 479)

14.3 In the Cartesian coordinate system, the *Y* coordinates decrease as you move downward. In the screen coordinate system, the *Y* coordinates increase as you move downward, toward the bottom of the screen.

14.4 `javafx.scene.shape`

14.5 `javafx.scene.paint`

14.6 `javafx.scene.layout`

14.7 `setStroke()`

14.8 `setFill()`

14.9 `setRotate()`

14.10 `setScaleX()`

14.11 `setScaleY()`

14.12 `Triangle`

14.13 `TranslateTransition`

14.14 `RotateTransition`

14.15 `ScaleTransition`

14.16 `StrokeTransition`

14.17 `FillTransition`

14.18 `FadeTransition`

14.19 With the `Duration` class

14.20 `javafx.util`

14.21 You call the transition class's `setInterpolator` method.

14.22 `javafx.scene.effect`

14.23 `DropShadow`

14.24 `InnerShadow`

14.25 `ColorAdjust`

14.26 `BoxBlur`, `GuassianBlur`, and `MotionBlur`

14.27 `SepiaTone`

14.28 `Glow`

14.29 `Reflection`

14.30 `setInput`

14.31 `Media` and `MediaPlayer`

14.32 `javafx.scene.media`

14.33 If you pass `true` to the method, the sound file will be immediately played once the `MediaPlayer` object is created and ready.

14.34 It specifies the number of times that the sound file should be played.

14.35 `Media`, `MediaPlayer`, and `MediaPlayer`

14.36 `javafx.scene.media`

14.37 With the `MediaView` class's `setFitWidth` and `setFitHeight` methods.

14.38 `KEY_PRESSED`

14.39 `KEY_RELEASED`

14.40 `KEY_TYPED`

14.41 `getCode()`

14.42 The `getCode()` method returns a `KeyCode` value, which is an `enum` in the `javafx.scene.input` package. The `KeyCode` enum contains an extensive set of constants representing the keys on a computer keyboard.

Chapter 15

15.1 A recursive algorithm requires multiple method calls. Each method call requires several actions to be performed by the JVM. These actions include allocating memory for parameters and local variables, and storing the address of the program location where control returns after the method terminates. All of these actions are known as overhead. In an iterative algorithm, which uses a loop, such overhead is unnecessary.

15.2 A case in which the problem can be solved without recursion.

15.3 Cases in which the problem is solved using recursion.

15.4 When it reaches the base case.

15.5 In direct recursion, a recursive method calls itself. In indirect recursion, method A calls method B, which in turn calls method A.

Chapter 16

16.1 Bubble sort

16.2 Insertion sort

16.3 Selection sort

16.4 Quicksort

16.5 The sequential search algorithm simply uses a loop to step through each element of an array, comparing each element's value with the value being searched for. The binary search algorithm, which requires the values in the array to be sorted in order, starts searching at the element in the middle of the array. If the middle element's value is greater than the value being searched for, the algorithm next tests the element in the middle of the first half of the array. If the middle element's value is less than the value being searched for, the algorithm next tests the element in the middle of the last half of the array. Each time the array tests an array element and does not find the value being searched for, it eliminates half of the remaining portion of the array. This method continues until the value is found, or there are no more elements to test. The binary search is more efficient than the sequential search.

16.6 10,000

16.7 15

16.8 The items frequently searched for can be stored near the beginning of the array.

16.9 A basic operation is one that requires constant time, regardless of the size of the problem that is being solved.

16.10 The worst case complexity function $f(n)$ of an algorithm is a measure of the time required by the algorithm to solve a problem instance of size n that requires the most time.

16.11 Because $100n$ and $25n$ differ by a constant factor and constant factors are not significant, the two algorithms are equivalent in efficiency.

16.12 To say that $f(n)$ is not in $O(g(n))$ means that $f(n)$ cannot be bounded by any constant multiple of $g(n)$.

16.13 To say that $f(n)$ is in $O(g(n))$ means that there is a positive constant K such that $f(n) \leq Kg(n)$ for all $n \geq 1$. This means that for large problem sizes, an algorithm with complexity function $f(n)$ is no worse than one with complexity function $g(n)$.

16.14 To show that $100n^3 + 50n^2 + 75$ is in $O(20n^3)$, observe that

$$\frac{100n^3 + 50n^2 + 75}{20n^3} = 5 + \frac{5}{2n} + \frac{75}{20n^3} \leq 5 + 5 + 75 \leq 85$$

for all $n \geq 1$. Take $K = 85$ in Equation 17.1.

16.15 The problem definition and algorithm are as follows:

MAIN ALGORITHM
INPUT: two integer arrays a[ ] and b[ ] of size n
SIZE OF INPUT: The number n of array entries
OUTPUT: true if each element of a[ ] is also an element of b[ ], false otherwise

```
boolean contained = true
int i = 0
While contained && i < n do
   contained = LINSEARCH(a[i], b)
   i++;
End While
print contained
```

SEARCH ALGORITHM
INPUT: an integer X and an array b[ ] of size n
SIZE OF INPUT: The number n of array entries
RETURN: true if X is contained in b[ ], false otherwise

```
boolean LINSEARCH(int X, int [ ] b)
Begin
   int k = 0;
   boolean found = false;
   While !found && k < n do
      If (X == b[k])
         found = true;
      else
         k++
      END If
   End While
   return found
End LINSEARCH
```

16.16 See solution of 16.15 for the algorithm. In the worst case, the main algorithm makes n calls to the `LINSEARCH` procedure. Each call to `LINSEARCH` makes at most n^2 comparisons. Therefore, the algorithm makes a total of comparisons.

16.17 Let $f(n)$ be in O(20). Then there is a positive K such that $f(n) \leq 20K$ for all $n \geq 1$. Let $K_1 = 20K$. Then $f(n) \leq 1K_1$ for all $n \geq 1$, so $f(n)$ is in O(1).

16.18 Assuming that $g(n) \geq 1$ for all $n \geq 1$, we have $100 \leq 100g(n)$ for all $n \geq 1$. This implies that $g(n) + 100 \leq g(n) + 100g(n) = 101g(n)$ for all $n \geq 1$. Now if $f(n)$ is in $O(g(n) + 100)$, there exists a positive K such that $f(n) \leq K(g(n) + 100) \leq 101Kg(n)$ for all $n \geq 1$. Taking $K_1 = 101K$, we see that $f(n) \leq K_1 g(n)$ for all $n \geq 1$.

Chapter 17

17.1 Because it opens an opportunity for the application to throw an exception at run time. If we retrieve an object from the `ArrayList`, but cast it to the wrong type, an exception occurs.

17.2 In non-generic mode, the `get` method returns an `Object` reference. When the `get` method is called, the compiler does not know the exact type of the object being returned, only that it is an `Object`. To assign the reference to any other type of variable we have to cast it to the appropriate type.

17.3 `Rectangle` objects.

17.4 Compile time.

17.5 A type parameter is an identifier in a generic class declaration. It represents an actual type that will be used when a generic class is instantiated.

17.6 Only reference types may be passed to a type parameter.

17.7 Yes, it is possible. Doing so instantiates the class as a raw type. This is not recommended, however, because you give up the benefits of type-safety that you would have otherwise.

17.8 No, it is not required, but it is the standard convention.

17.9 That any type argument can be used in its place.

17.10 It means that any type that extends `Number`, or is `Number`, can be passed as a type argument.

17.11 It means that any type that is `Integer` or a superclass of `Integer` can be passed as a type argument.

17.12 An upper boundary

17.13 A lower boundary

17.14 No, you cannot use both together.

17.15 When you call a generic method, the compiler determines which type to use from the context in which you are using the method.

17.16 To prevent the class from being instantiated with a type argument that does not support some operation that the class is performing.

17.17 A generic class's type can be constrained with either an upper bound or a lower bound.

17.18 A generic class can be a superclass and/or a subclass.

17.19 Yes. The "is-a" relationship is in effect between the classes.

17.20 Yes.

17.21 Yes.

17.22 You would write the type parameter definition as:

```
<T extends Comparable>
```

17.23 When the compiler encounters a generic class, interface, or method with an unbound type parameter, such as `<T>` or `<E>`, it replaces all occurrences of the type parameter with `Object`.

17.24 When the compiler encounters a class, interface, or method with a bound type parameter, such as `<T extends Number>` or `<E extends Comparable>`, it replaces all occurrences of the type parameter with the bound that is applied to the parameter.

17.25 No.

Chapter 18

18.1 The `java.util` package.

18.2 Lists, sets, and maps.

18.3 Elements are stored in a list according to position, and a list can have duplicate values. A set has no concept of position and does not allow duplicates.

18.4 Each element of a map consists of a key associated with a value.

18.5 `List` and `Set` extend the `Collection` interface.

18.6 `forEach()` and `iterator()`.

18.7 An iterator is an object that can move through a collection and provide access to each element of that collection.

18.8 Contiguous allocation.

18.9 Linked allocation.

18.10 So you can change the class that implements the interface without having to change the rest of the code.

18.11 The `Iterator` interface.

18.12 A `ListIterator` object can move forward and backward through a collection, while an `Iterator` object can only move forward.

18.13 Call the `listIterator()` method.

18.14 By calling the `hasNext()` method.

18.15 The `hasPrevious()` method.

18.16 The element returned by the last call to the `next()` method.

18.17 The element returned by the last call to `next()` or `previous()`.

18.18 The capacity is the length of the array that underlies the `ArrayList`, while the size is the number of elements currently stored.

18.19 The `LinkedList` class.

18.20 The `LinkedList` class.

18.21 A hash code of an object is an integer value that is characteristic of that object.

18.22 The `hashCode()` method.

18.23 A collision occurs when two different objects hash to the same value.

18.24 The duplicate item is not added to the set.

18.25 The `hashCode()` method.

18.26 The `LinkedHashSet` class.

18.27 The `TreeSet` class.

18.28 A comparator for a class is an object that can compare two objects of that class and determine whether they are equal, and if not, which comes before the other. A comparator implements the `Comparator` interface.

18.29 Each element of map consists of a key and a value.

18.30 The key.

18.31 Insertion order and access order.

18.32 By key order.

18.33 A stream is an object that permits a pipeline of operations to be performed on a sequence of elements drawn from a source.

18.34 An intermediate operation transforms a stream into another stream, while a terminal operation transforms a stream into a non-stream object or result.

18.35 The `Stream` interface, the Collection interface, and the `Arrays` class all have methods for creating streams.

18.36 A pipeline is a sequence of operations where the output of one operation becomes the input to the next operation in the sequence.

18.37 A terminal operation and a reduction are the same thing.

18.38 A collector is an object that performs a reduction on a stream.

18.39 A stream pipeline can have at most one reduction, which must be the last operation in the pipeline.

Chapter 19

19.1
```
Node greenNode  = new Node("green");
greenNode.next = new Node("blue");
Node redNode = new Node("red", greenNode);
```

19.2 `BrownYellowPurple`

19.3
```
// Let p point to the new Node
Node p = new Node(e, succ, succ.prev);
succ.prev = p;
if (p.prev == null)
   first = p;
else
   p.prev.succ = p;
```

19.4
```
last.next  = new Node(e, null, last);
last = last.next;
```

Chapter 20

20.1 push

20.2 pop

20.3 Throw an exception

20.4 Last in first out

20.5 Some examples are a stack of plates or trays in a cafeteria, a line of cars parked single file in a narrow driveway, and a person wearing an undershirt, a shirt, a jacket, and an overcoat.

20.6 enqueue

20.7 First in first out

20.8 Some examples are a grocery store checkout line, a line of cars going through a toll booth.

20.9 By the time the add operation reaches the end of the array, the remove operation may have emptied the spaces of the beginning of the array, allowing the add operation to wrap around and start storing elements at the beginning. Similarly, when the remove empties spaces at the end of the array, it should wrap around to the beginning.

Chapter 21

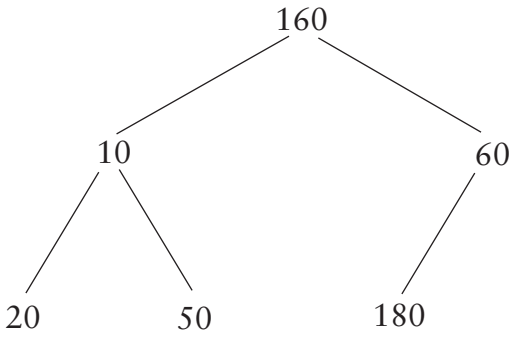
21.1
a. Ancestors of *A* : {*A*}; ancestors of *B*: {*B*, *A*}; ancestors of *C*: {*C*, *A*}; ancestors of *D*: {*D*, *B*, *A*} : ancestors of *E*: {*E*, *C*, *A*}; ancestors of *F*: {*F*, *C*, *A*}; ancestors of *G*: {*G*, *F*, *C*, *A*}.
b. Descendants of *A*: {*A*, *B*, *D*}; descendants of *B*: {*B*, *D*}; descendants of *D*: {*D*}; descendants of *C*: {*C*, *E*, *F*, *G*}; descendants of *E*: {*E*}; descendants of *F*: {*F*, *G*}; descendants of *G*: {*G*}.
c. Leaves of T: {*D*, *E*, *G*}.
d. Non-leaf nodes of *T*: {*A*, *B*, *C*, *F*}.
e. Preorder: {*A*, *B*, *D*, *C*, *E*, *F*, *G*}.
f. Postorder: {*D*, *B*, *E*, *G*, *F*, *C*, *A*}.
g. Inorder: {*B*, *D*, *A*, *E*, *C*, *G*, *F*}.

21.2 The tree on the left : *x* 12 +
`The tree on the right:` *x* 12 + *y* * 2 +

21.3 The tree on the left: + *x* 12
`The tree on the right:` + * + *x* 12 *y* 2

21.4 Nodes in preorder: 160 10 20 50 60 180

Drawing of the tree:



Chapter 22 (Available on the Book's Companion Website)

22.1 Traditional text and binary files are not practical when a large amount of data must be stored and manipulated. Many businesses keep hundreds of thousands, or even millions, of data items in files. When a text or binary file contains this much data, simple operations such as searching, inserting, and deleting become cumbersome and inefficient.

22.2 The DBMS handles the actual reading, writing, and searching of data. The Java application programmer needs only to know how to communicate with the DBMS.

22.3 JDBC, or Java Database Connectivity

22.4 SQL, or Structured Query Language

22.5 A string listing the protocol that should be used to access a database, the name of the database, and potentially other items

22.6 `jdbc:derby:InventoryDB`

22.7 `DriverManager.getConnection`

22.8 The data that is stored in a table is organized into rows and columns. A row is a complete set of information about a single item. The data that is stored in a row is divided into columns. Each column is an individual piece of information about the item.

22.9 A primary key is a column that holds a unique value for each row, and can be used to identify specific rows.

22.10
- `int`
- `int`
- `float`
- `String`
- `String`
- `String`
- `double`

22.11 A `ResultSet` is an object that contains the results of an SQL statement.

22.12 a. The table is `Account`.
b. The column is `Id`.

22.13 a.
```
SELECT * FROM Inventory
```
b.
```
SELECT ProductID FROM Inventory
```
c.
```
SELECT ProductID, QtyOnHand FROM Inventory
```
d.
```
SELECT ProductID FROM Inventory
   WHERE Cost < 17.00
```
e.
```
SELECT * FROM Inventory
   WHERE ProductID LIKE '%ZZ'
```

22.14 The `LIKE` operator can be used in a search criterion to search for a string that contains a pattern of characters.

22.15 The `%` symbol represents any sequence of 0 or more characters. The underscore character (_) represents only a single character.

22.16 You use the `ORDER BY` clause.

23.17
```
Connection conn = DriverManager.getConnection(DB_URL);
Statement stmt = conn.createStatement();
ResultSet result = stmt.executeQuery(sql);
stmt.close();
conn.close();
```

22.18 You create a `Statement` object (`Statement` in an interface in `java.sql`), and you pass a string containing the `SELECT` statement to the `Statement` object's `executeQuery` method.

22.19 The cursor initially points before the first row. To move it forward you call the `ResultSet` object's `next` method.

22.20 You call the `ResultSet` object's `getString` method, passing 3 as an argument.

22.21
```
INSERT INTO Coffee
   (Description, ProdNum, Price)
VALUES
   ('Eastern Blend', '30-001', 18.95)
```

22.22
```
INSERT INTO Coffee
   (Description, ProdNum, Price)
VALUES
   ('Honduran Dark', '22-001', 8.65)
```

22.23
```
UPDATE Coffee
SET Price = 4.95
WHERE Description Like '%Decaf%'
```

22.24
```
DELETE FROM Coffee WHERE Description LIKE '%Decaf%'
```

22.25
```
CREATE TABLE Book
( Publisher CHAR(25),
Author, CHAR(25),
Pages INTEGER,
Isbn CHAR(10) )
```

22.26
```
DROP TABLE Book
```

Chapter 23 (Available on the Book's Companion Website)

23.1 A frame is a basic window that has a border around it, a title bar, and a set of buttons for minimizing, maximizing, and closing the window. In a Swing application, you create a frame from the `JFrame` class.

23.2 With the `setSize` method.

23.3 With the `setVisible` method.

23.4 A content pane is a container that is part of every `JFrame` object. You cannot see the content pane and it does not have a border, but any component that is added to a `JFrame` must be added to its content pane.

23.5 Panels cannot be displayed by themselves.

23.6 It is an object that responds to events.

23.7 The class must implement the `ActionListener` interface, and it must have a method named `actionPerformed`. The method is executed when the user clicks the button.

23.8 With the `addActionListener` method.

23.9 You use the `setBackground` method to set the color of a component, and the `setForeground` method to set the color of text displayed on a component.

23.10 By calling the container's `setLayout` method and passing a reference to a layout manager object as the argument.

23.11 `BorderLayout`

23.12 `FlowLayout`

23.13 `GridLayout`

23.14 Only one, in both cases.

23.15 By placing the component in a panel, then adding the panel to the region. The layout manager resizes the panel, not the component inside the panel.

23.16 By calling the `pack` method.

23.17 `BorderLayout` is the default layout manager for a `JFrame` object's content pane.

`FlowLayout` is the default layout manager for a `JPanel` object.

23.18 Radio buttons

23.19 Check boxes

23.20 A `ButtonGroup` object can contain radio buttons. It creates a mutually exclusive relationship between the radio buttons that it contains.

23.21 Radio buttons.

23.22 An action event.

23.23 An item event.

23.24 You use the `isSelected` method to determine whether a `JRadioButton` component is selected. The method returns a `boolean` value. If the check box is selected, the method returns `true`. Otherwise, it returns `false`.

23.25 You use the `isSelected` method to determine whether a `JCheckBox` component is selected. The method returns a `boolean` value. If the check box is selected, the method returns `true`. Otherwise, it returns `false`.

23.26 `setBorder`

23.27 You should use the `BorderFactory` class to create them for you. The `BorderFactory` class has static methods that return various types of borders.

Chapter 24 (Available on the Book's Companion Website)

24.1 You call its `setEditable` method and pass `false` as the parameter. To store text in the text field, use the `setText` method.

24.2 The index of the first item is 0. The index of the twelfth item would be 11.

24.3 You use the `getSelectedValue` method to retrieve the selected item, and the `getSelectedIndex` method to get the index of the selected item.

24.4 First, set the number of visible rows for the list component. Next, create a scroll pane object and add the list component to it.

24.5 You use the `getSelectedItem` method to retrieve the selected item, and the `getSelectedIndex` method to get the index of the selected item.

24.6 An uneditable combo box combines a button with a list and allows the user to select only items from its list. An editable combo box combines a text field and a list. In addition to selecting items from the list, the user may also type input into the text field. The default type of combo box is uneditable.

24.7 To display an image, you first create an instance of the `ImageIcon` class, which can read the contents of an image file. Next, you display the image in a label by passing the `ImageIcon` object as an argument to the `JLabel` constructor. You can create a `JLabel` with both an image and text by creating the `JLabel` in the usual way, with the text passed to the constructor. Then, you can use the `setIcon` method to display an image.

24.8 To display an image, you first create an instance of the `ImageIcon` class, which can read the contents of an image file. Next, you display the image in a button by passing the `ImageIcon` object as an argument to the `JButton` constructor. To create a button with an image and text, pass a `String` and an `ImageIcon` object to the constructor.

24.9 `setIcon`

24.10 A mnemonic is a key on the keyboard that you press in combination with the Alt key to quickly access a component such as a button. These are sometimes referred to as short-cut keys, or hot keys. You assign an access key to a component through the component's `setMnemonic` method.

24.11 A tool tip is text that is displayed in a small box when the user holds the mouse cursor over a component. The box usually gives a short description of what the component does. You assign a tool tip to a component with the `setToolTipText` method.

24.12 a. **Menu bar.** At the top of the window, just below the title bar, is a menu bar. The menu bar lists the names of one or more menus.
b. **Menu item.** A menu item can be selected by the user. When a menu item is selected, some type of action is usually performed.
c. **Check box menu item.** A check box menu item appears with a small box beside it. The item may be selected or deselected. When it is selected, a check mark appears in the box. When it is deselected, the box appears empty. Check box menu items are normally used to turn an option on or off. The user toggles the state of a check box menu item each time he or she selects it.
d. **Radio button menu item.** A radio button menu item may be selected or deselected. A small circle appears beside it that is filled in when the item is selected and appears empty when the item is deselected. Like a check box menu item, a radio button menu item can be used to turn an option on or off. When a set of radio button menu items are grouped with a `ButtonGroup` object, only one of them can be selected at a time. When the user selects a radio button menu item, the one that was previously selected is deselected.
e. **Submenu.** A menu within a menu is called a submenu. Some of the commands on a menu are actually the names of submenus. You can tell when a command is the name of a submenu because a small right arrow appears to its right. Activating the name of a submenu causes the submenu to appear.
f. **Separator bar.** A separator bar is a horizontal bar used to separate groups of items on a menu. Separator bars are used only as a visual aid and cannot be selected by the user.

24.13 The `JMenuItem` class. You pass a string, which is displayed on the menu item.

24.14 The `JRadioButtonMenuItem` class. You pass a string to the constructor, which is displayed on the menu item. To cause it to be initially selected, you pass `true` as an optional second argument to the constructor.

24.15 You add them to a `ButtonGroup`.

24.16 The `JCheckBoxMenuItem` class. You pass a string to the constructor, which is displayed on the menu item. To cause it to be initially selected, you pass `true` as an optional second argument to the constructor.

24.17 The `JMenu` class. You pass a string to the constructor, which is displayed on the menu.

24.18 The `JMenuBar` class.

24.19 With the `JFrame` object's `setJMenuBar` method.

24.20 Action events.

24.21 With the `setPreferredSize` method.

24.22 The first argument to the `Dimension` class constructor is the component's width, and the second argument is the component's height.

24.23 One constructor accepts the size of the text area, in rows and columns. Another constructor accepts the string that is to be displayed initially in the text area, as well as the size of the text area in rows and columns.

24.24 With the `getText` method.

24.25 `JTextArea` components do not automatically display scroll bars. To display scroll bars on a `JTextArea` component, you must add it to the scroll pane.

24.26 Line wrapping causes text to automatically wrap to the next line when a line is filled. There are two different styles of line wrapping: word wrapping and character wrapping. When *word wrapping* is performed, the line breaks always occur between words, never in the middle of a word. When character wrapping is performed, lines are broken between characters. You use the `JTextArea` class's `setLineWrap` method to turn line wrapping on. The method accepts a `boolean` argument. If you pass `true`, then line wrapping is turned on. If you pass `false`, line wrapping is turned off. You specify the style of line wrapping that you prefer with the `JTextArea` class's `setWrapStyleWord` method. This method accepts a `boolean` argument. If you pass `true`, then the text area will perform word wrapping. If you pass `false`, the text area will perform character wrapping.

24.27 A `Font` object.

24.28 The first argument is the name of a font. The second argument is an `int` that represents the style of the text. The third argument is the size of the text in points.

24.29 It generates a change event.

24.30
a. `setMajorTickSpacing`
b. `setMinorTickSpacing`
c. `setPaintTicks`
d. `setPaintLabels`

Chapter 25 (Available on the Book's Companion Website)

25.1 When a user accesses a Web page with his or her browser, any applet associated with the Web page is transmitted to the user's system. The applet is then executed on the user's system.

25.2 To prevent malicious code from attacking or spying on unsuspecting users.

25.3 `<html>` marks the beginning and `</html>` marks the end.

25.4 `<head></head>`

25.5 You would use: `<title>My Web Page</title>`
This would go in the document head section.

25.6 `<body></body>`

25.7 `<h1>Student Roster</h1>`

25.8 `<center><b>My Resume</b></center>`

25.9 `<b><i>Hello World</i></b>`

25.10 The `<br />` tag causes a line break. The `<p />` tag causes a paragraph break. The `<hr />` tag displays a horizontal rule.

25.11
```
<a href=
"http://java.sun.com">Click Here</a>
```

25.12 `JApplet`

25.13 `init`

25.14 Because the browser creates an instance of the class automatically.

25.15
```
<applet code="MyApplet.class" width=400 height=200>
```

25.16 `Applet`

25.17 You simply add them to the `Frame` or `Applet` object. These classes have an `add` method.

25.18 You should override the `paint` method.

25.19 You override the paintComponent method.

25.20 These methods are automatically called when the component is first displayed and are called again any time the component needs to be redisplayed.

25.21 The superclass's `paint` or `paintComponent` method should be called.

25.22 Call the `repaint` method.

25.23 A rectangle.

25.24 The first array contains the *X* coordinates for each vertex, and the second array contains the *Y* coordinates for each vertex.

25.25
a. `drawLine`
b. `fillRect`
c. `fillOval`
d. `fillArc`
e. `setColor`
f. `drawRect`
g. `drawOval`
h. `drawArc`
i. `drawString`
j. `setFont`

25.26 A mouse press event indicates that the mouse button was pressed. A mouse click event indicates that the mouse button was pressed, and then released.

25.27 To handle a mouse click event: `MouseListener`
To handle a mouse press event: `MouseListener`
To handle a mouse dragged event: `MouseMotionListener`
To handle a mouse release event: `MouseListener`
To handle a mouse move event: `MouseMotionListener`

25.28 They accept `MouseEvent` objects. The `getX` and `getY` methods return the mouse's *X* and *Y* coordinates.

25.29 No, they cannot be left out. Empty definitions can be written, or an adapter class can be used.

25.30 An adapter class implements an interface and provides empty definitions for all of the required methods. When you extend a class from an adapter class, it inherits the empty methods. In your subclass, you can override the methods you want and forget about the others.

25.31 Action events

25.32 In milliseconds.

25.33 By calling its `start` method.

25.34 By calling its `stop` method.

25.35 The `play` method.

25.36 The `Applet` class's `play` method loads a sound file, plays it one time, and then releases it for garbage collection. If you need to load a sound file to be played multiple times, you should use an `AudioClip` object. An `AudioClip` object loads a sound file and retains it in memory. It provides methods for playing, looping, and stopping the audio play.

25.37 The `play` method plays the sound file. The `loop` method plays the sound file repeatedly. The `stop` method stops the sound file from playing.

25.38 The `getDocumentBase` method returns a `URL` object containing the location of the HTML file that invoked the applet. The `getCodeBase` method returns a URL object containing the location of the applet's `.class` file.