APPENDIX

A

Solutions to Exercises

Chapters 1 through 12 close with an "Exercises" section that tests your understanding of the chapter's material through various exercises. Solutions to these exercises are presented in this appendix.

Chapter 1: Getting Started with Java

Chapter 1's exercises test your understanding of applications and language fundamentals:

1. Listing A-1 presents the `EchoArgs` class.

Listing A-1. *Echoing command-line arguments to standard output*

```
class EchoArgs
{
   public static void main(String[] args)
   {
      for (int i = 0; i < args.length; i++)
         System.out.println(args[i]);
   }
}
```

2. Listing A-2 presents the `Circle` class.

Listing A-2. *Calculating a circle's circumference and area*

```
class Circle
{
   public static void main(String[] args)
   {
      double PI = 3.14159, radius = 15;
      System.out.println("Circumference = "+PI*radius*2);
      System.out.println("Area = "+PI*radius*radius);
   }
}
```

3. Listing A-3 presents the `Input` class.

Listing A-3. *Prompting the user to enter C or c*

```
class Input
{
   public static void main(String[] args) throws java.io.IOException
   {
      int ch;
      while (true)
      {
         System.out.println("Press C or c to continue.");
         ch = System.in.read();
         if (ch == 'C' || ch == 'c')
            break;
      }
   }
}
```

4. Listing A-4 presents the `Triangle` class.

Listing A-4. *Outputting a triangle of asterisks*

```
class Triangle
{
   public static void main(String[] args)
   {
      for (int row = 1; row < 20; row += 2)
      {
         for (int col = 0; col < 19-row/2; col++)
            System.out.print(" ");
         for (int col = 0; col < row; col++)
            System.out.print("*");
         System.out.print('\n');
      }
   }
}
```

5. Listing A-5 presents the `OutputReversedInt` class.

Listing A-5. *Reversing an integer's digits*

```
class OutputReversedInt
{
   public static void main(String[] args)
   {
      int x = 876432094;
      int i = 0;
      while (x != 0)
      {
         System.out.print(x%10);
         x /= 10;
      }
      System.out.print('\n');
   }
}
```

Chapter 2: Discovering Classes and Objects

Chapter 2's exercises test your understanding of classes and objects:

1. Listing A-6 presents the `Image` class.

Listing A-6. *Enhancing `Image` by introducing fields and getters*

```
class Image
{
   private int width, height;
   private byte[] image;

   Image()
   {
      this(null);
   }
   Image(String filename)
   {
      this(filename, null);
   }
   Image(String filename, String imageType)
   {
```

```
      if (filename != null)
      {
         System.out.println("reading "+filename);
         if (imageType != null)
            System.out.println("interpreting "+filename+" as storing a "+
                               imageType+" image");
         image = new byte[(int) (Math.random()*100000)];
         width = (int) (Math.random()*1024);
         height = (int) (Math.random()*768);
         return;
      }
      width = -1;
      height = -1;
   }
   int getWidth()
   {
      return width;
   }
   int getHeight()
   {
      return height;
   }
   byte[] getImage()
   {
      return image;
   }
   int getSize()
   {
      return image == null ? 0 : image.length;
   }
   public static void main(String[] args)
   {
      Image image = new Image();
      System.out.println("Image = "+image.getImage());
      System.out.println("Size = "+image.getSize());
      System.out.println("Width = "+image.getWidth());
      System.out.println("Height = "+image.getHeight());
      System.out.println();
      image = new Image("image.png");
      System.out.println("Image = "+image.getImage());
```

```
      System.out.println("Size = "+image.getSize());
      System.out.println("Width = "+image.getWidth());
      System.out.println("Height = "+image.getHeight());
      System.out.println();
      image = new Image("image.png", "PNG");
      System.out.println("Image = "+image.getImage());
      System.out.println("Size = "+image.getSize());
      System.out.println("Width = "+image.getWidth());
      System.out.println("Height = "+image.getHeight());
   }
}
```

2. Listings A-7 through A-13 present the `Animal`, `Bird`, `Fish`, `AmericanRobin`, `DomesticCanary`, `RainbowTrout`, and `SockeyeSalmon` classes.

Listing A-7. *The* `Animal` *class abstracting over birds and fish (and other organisms)*

```
public abstract class Animal
{
   private String kind;
   private String appearance;
   public Animal(String kind, String appearance)
   {
      this.kind = kind;
      this.appearance = appearance;
   }
   public abstract void eat();
   public abstract void move();
   @Override
   public final String toString()
   {
      return kind+" -- "+appearance;
   }
}
```

Listing A-8. *The* `Bird` *class abstracting over American robins, domestic canaries, and other kinds of birds*

```
public abstract class Bird extends Animal
{
   public Bird(String kind, String appearance)
   {
      super(kind, appearance);
```

```
    }
    @Override
    public final void eat()
    {
        System.out.println("eats seeds and insects");
    }
    @Override
    public final void move()
    {
        System.out.println("flies through the air");
    }
}
```

Listing A-9. *The* `Fish` *class abstracting over rainbow trout, sockeye salmon, and other kinds of fish*

```
public abstract class Fish extends Animal
{
    public Fish(String kind, String appearance)
    {
        super(kind, appearance);
    }
    @Override
    public final void eat()
    {
        System.out.println("eats krill, algae, and insects");
    }
    @Override
    public final void move()
    {
        System.out.println("swims through the water");
    }
}
```

Listing A-10. *The* `AmericanRobin` *class denoting a bird with a red breast*

```
public final class AmericanRobin extends Bird
{
    public AmericanRobin()
    {
        super("americanrobin", "red breast");
    }
}
```

Listing A-11. *The* `DomesticCanary` *class denoting a bird of various colors*

```
public final class DomesticCanary extends Bird
{
   public DomesticCanary()
   {
      super("domesticcanary", "yellow, orange, black, brown, white, red");
   }
}
```

Listing A-12. *The* `RainbowTrout` *class denoting a rainbow-colored fish*

```
public final class RainbowTrout extends Fish
{
   public RainbowTrout()
   {
      super("rainbowtrout", "bands of brilliant speckled multicolored "+
            "stripes running nearly the whole length of its body");
   }
}
```

Listing A-13. *The* `SockeyeSalmon` *class denoting a red-and-green fish*

```
public final class SockeyeSalmon extends Fish
{
   public SockeyeSalmon()
   {
      super("sockeyesalmon", "bright red with a green head");
   }
}
```

`Animal`'s `toString()` method is declared `final` because it doesn't make sense to override this method, which is complete in this exercise. Also, each of `Bird`'s and `Fish`'s overriding `eat()` and `move()` methods is declared `final` because it doesn't make sense to override these methods in this exercise, which assumes that all birds eat seeds and insects; all fish eat krill, algae, and insects; all birds fly through the air; and all fish swim through the water.

The `AmericanRobin`, `DomesticCanary`, `RainbowTrout`, and `SockeyeSalmon` classes are declared `final` because they represent the bottom of the `Bird` and `Fish` class hierarchies, and it doesn't make sense to subclass them.

3. Listing A-14 presents the `Animals` class.

Listing A-14. *The* `Animals` *class letting animals eat and move*

```
class Animals
{
   public static void main(String[] args)
   {
      Animal[] animals = { new AmericanRobin(), new RainbowTrout(),
                           new DomesticCanary(), new SockeyeSalmon() };
      for (int i = 0; i < animals.length; i++)
      {
         System.out.println(animals[i]);
         animals[i].eat();
         animals[i].move();
         System.out.println();
      }
   }
}
```

4. Listings A-15 through A-17 present the `Countable` interface, the modified `Animal` class, and the modified `Animals` class.

Listing A-15. *The* `Countable` *interface for use in taking a census of animals*

```
public interface Countable
{
   String getID();
}
```

Listing A-16. *The refactored* `Animal` *class for helping in census taking*

```
public abstract class Animal implements Countable
{
   private String kind;
   private String appearance;
   public Animal(String kind, String appearance)
   {
      this.kind = kind;
      this.appearance = appearance;
   }
   public abstract void eat();
   public abstract void move();
   @Override
   public final String toString()
```

```
   {
      return kind+" -- "+appearance;
   }
   @Override
   public final String getID()
   {
      return kind;
   }
}
```

Listing A-17. *The modified* `Animals` *class for carrying out the census*

```
class Animals
{
   public static void main(String[] args)
   {
      Animal[] animals = { new AmericanRobin(), new RainbowTrout(),
                           new DomesticCanary(), new SockeyeSalmon(),
                           new RainbowTrout(), new AmericanRobin() };
      for (int i = 0; i < animals.length; i++)
      {
         System.out.println(animals[i]);
         animals[i].eat();
         animals[i].move();
         System.out.println();
      }

      Census census = new Census();
      Countable[] countables = (Countable[]) animals;
      for (int i = 0; i < countables.length; i++)
         census.update(countables[i].getID());

      for (int i = 0; i < Census.SIZE; i++)
         System.out.println(census.get(i));
   }
}
```

Chapter 3: Exploring Advanced Language Features

Chapter 3's exercises test your understanding of nested types, packages, static imports, exceptions, assertions, annotations, generics, and enums:

1. Listing A-18 presents the `G2D` class.

Listing A-18. *The `G2D` class with its `Matrix` nonstatic member class*

```
class G2D
{
   private Matrix xform;
   G2D()
   {
      xform = new Matrix(3, 3);
      xform.matrix[0][0] = 1.0;
      xform.matrix[1][1] = 1.0;
      xform.matrix[2][2] = 1.0;
      xform.dump();
   }
   private class Matrix
   {
      private double[][] matrix;
      Matrix(int nrows, int ncols)
      {
         matrix = new double[nrows][ncols];
      }
      void dump()
      {
         for (int row = 0; row < matrix.length; row++)
         {
            for (int col = 0; col < matrix[0].length; col++)
               System.out.print(matrix[row][col]+" ");
            System.out.println();
         }
         System.out.println();
      }
   }
   public static void main(String[] args)
   {
      G2D g2d = new G2D();
   }
```

```
}
```

2. Listings A-19 and A-20 present the `NullDevice` package-private class and the refactored `LoggerFactory` class.

Listing A-19. *Implementing the "bit bucket" class*

```
package logging;


class NullDevice implements Logger
{
   private String dstName;
   NullDevice(String dstName)
   {
   }
   @Override
   public boolean connect()
   {
      return true;
   }
   @Override
   public boolean disconnect()
   {
      return true;
   }
   @Override
   public boolean log(String msg)
   {
      return true;
   }
}
```

Listing A-20. *A refactored* `LoggerFactory` *class*

```
package logging;

public abstract class LoggerFactory
{
   public final static int CONSOLE = 0;
```

```
   public final static int FILE = 1;
   public final static int NULLDEVICE = 2;
   public static Logger newLogger(int dstType, String... dstName)
   {
      switch (dstType)
      {
         case CONSOLE   : return new Console(dstName.length == 0 ? null
                                                                 : dstName[0]);
         case FILE      : return new File(dstName.length == 0 ? null
                                                              : dstName[0]);
         case NULLDEVICE: return new NullDevice(null);
         default        : return null;
      }
   }
}
```

3. Listing A-21 presents the refactored `G2D` class.

Listing A-21. *The refactored `G2D` class takes advantage of static imports*

```
import static java.lang.Math.cos;
import static java.lang.Math.sin;
import static java.lang.Math.toRadians;

class G2D
{
   private Matrix xform;
   G2D()
   {
      xform = new Matrix(3, 3);
      xform.matrix[0][0] = 1.0;
      xform.matrix[1][1] = 1.0;
      xform.matrix[2][2] = 1.0;
      xform.dump();
   }
   void rotate(double angle)
   {
      // Make sure rotation is counter-clockwise.
      angle = -angle;
      Matrix r = new Matrix(3, 3);
      double angRad = toRadians(angle);
```

```
      r.matrix[0][0] = cos(angRad);
      r.matrix[1][0] = sin(angRad);
      r.matrix[0][1] = -sin(angRad);
      r.matrix[1][1] = cos(angRad);
      r.matrix[2][2] = 1.0;
      xform = xform.multiply(r);
      xform.dump();
   }
   private class Matrix
   {
      private double[][] matrix;
      Matrix(int nrows, int ncols)
      {
         matrix = new double[nrows][ncols];
      }
      void dump()
      {
         for (int row = 0; row < matrix.length; row++)
         {
            for (int col = 0; col < matrix[0].length; col++)
               System.out.print(matrix[row][col]+" ");
            System.out.println();
         }
         System.out.println();
      }
      Matrix multiply(Matrix m)
      {
         Matrix result = new Matrix(matrix.length, matrix[0].length);
         for (int i = 0; i < matrix.length; i++)
            for (int j = 0; j < m.matrix[0].length; j++)
               for (int k = 0; k < m.matrix.length; k++)
                  result.matrix[i][j] = result.matrix[i][j]+
                                        matrix[i][k]*m.matrix[k][j];
         return result;
      }
   }
   public static void main(String[] args)
   {
      G2D g2d = new G2D();
      g2d.rotate(45);
```

```
   }
}
```

4. Listings A-22 through A-26 present the `Logger` interface, and the `CannotConnectException`, `Console`, `File`, and `TestLogger` classes.

Listing A-22. *A `Logger` interface whose methods throw exceptions*

```
package logging;

public interface Logger
{
   void connect() throws CannotConnectException;
   void disconnect() throws NotConnectedException;
   void log(String msg) throws NotConnectedException;
}
```

Listing A-23. *An uncomplicated `CannotConnectException` class*

```
package logging;

public class CannotConnectException extends Exception
{
}
```

■ **Note** The `NotConnectedException` class has a similar structure.

Listing A-24. *The `Console` class satisfying `Logger`'s contract without throwing exceptions*

```
package logging;

class Console implements Logger
{
   private String dstName;
   Console(String dstName)
   {
      this.dstName = dstName;
   }
   @Override
   public void connect() throws CannotConnectException
   {
```

```
    }
    @Override
    public void disconnect() throws NotConnectedException
    {
    }
    @Override
    public void log(String msg) throws NotConnectedException
    {
        System.out.println(msg);
    }
}
```

Listing A-25. *The* `File` *class satisfying* `Logger`*'s contract by throwing exceptions as necessary*

```
package logging;

class File implements Logger
{
    private String dstName;
    File(String dstName)
    {
        this.dstName = dstName;
    }
    @Override
    public void connect() throws CannotConnectException
    {
        if (dstName == null)
            throw new CannotConnectException();
    }
    @Override
    public void disconnect() throws NotConnectedException
    {
        if (dstName == null)
            throw new NotConnectedException();
    }
    @Override
    public void log(String msg) throws NotConnectedException
    {
        if (dstName == null)
            throw new NotConnectedException();
        System.out.println("writing "+msg+" to file "+dstName);
```

```
   }
}
```

Listing A-26. *A* `TestLogger` *class that handles thrown exceptions*

```
import logging.CannotConnectException;
import logging.Logger;
import logging.LoggerFactory;
import logging.NotConnectedException;

class TestLogger
{
   public static void main(String[] args)
   {
      try
      {
         Logger logger = LoggerFactory.newLogger(LoggerFactory.CONSOLE);
         logger.connect();
         logger.log("test message #1");
         logger.disconnect();
      }
      catch (CannotConnectException cce)
      {
         System.err.println("cannot connect to console-based logger");
      }
      catch (NotConnectedException nce)
      {
         System.err.println("not connected to console-based logger");
      }
      try
      {
         Logger logger = LoggerFactory.newLogger(LoggerFactory.FILE, "x.txt");
         logger.connect();
         logger.log("test message #2");
         logger.disconnect();
      }
      catch (CannotConnectException cce)
      {
         System.err.println("cannot connect to file-based logger");
      }
      catch (NotConnectedException nce)
```

```
      {
         System.err.println("not connected to file-based logger");
      }
      try
      {
         Logger logger = LoggerFactory.newLogger(LoggerFactory.FILE);
         logger.connect();
         logger.log("test message #3");
         logger.disconnect();
      }
      catch (CannotConnectException cce)
      {
         System.err.println("cannot connect to file-based logger");
      }
      catch (NotConnectedException nce)
      {
         System.err.println("not connected to file-based logger");
      }
   }
}
```

5. Listing A-27 presents the `G2D` class.

Listing A-27. *Verifying a class invariant in the* `G2D` *class*

```
import static java.lang.Math.cos;
import static java.lang.Math.sin;
import static java.lang.Math.toRadians;

class G2D
{
   private Matrix xform;
   G2D()
   {
      xform = new Matrix(3, 3);
      xform.matrix[0][0] = 1.0;
      xform.matrix[1][1] = 1.0;
      xform.matrix[2][2] = 1.0;
      xform.dump();
      assert isIdentity();
   }
```

```
boolean isIdentity()
{
   return xform.matrix[0][0] == 1.0 && xform.matrix[0][1] == 0.0 &&
          xform.matrix[0][2] == 0.0 && xform.matrix[1][0] == 0.0 &&
          xform.matrix[1][1] == 1.0 && xform.matrix[1][2] == 0.0 &&
          xform.matrix[2][0] == 0.0 && xform.matrix[2][1] == 0.0 &&
          xform.matrix[2][2] == 1.0;
}
void rotate(double angle)
{
   // Make sure rotation is counter-clockwise.
   angle = -angle;
   Matrix r = new Matrix(3, 3);
   double angRad = toRadians(angle);
   r.matrix[0][0] = cos(angRad);
   r.matrix[1][0] = sin(angRad);
   r.matrix[0][1] = -sin(angRad);
   r.matrix[1][1] = cos(angRad);
   r.matrix[2][2] = 1.0;
   xform = xform.multiply(r);
   xform.dump();
}
private class Matrix
{
   private double[][] matrix;
   Matrix(int nrows, int ncols)
   {
      matrix = new double[nrows][ncols];
   }
   void dump()
   {
      for (int row = 0; row < matrix.length; row++)
      {
         for (int col = 0; col < matrix[0].length; col++)
            System.out.print(matrix[row][col]+" ");
         System.out.println();
      }
      System.out.println();
   }
   Matrix multiply(Matrix m)
```

```
      {
         Matrix result = new Matrix(matrix.length, matrix[0].length);
         for (int i = 0; i < matrix.length; i++)
            for (int j = 0; j < m.matrix[0].length; j++)
               for (int k = 0; k < m.matrix.length; k++)
                  result.matrix[i][j] = result.matrix[i][j]+
                                        matrix[i][k]*m.matrix[k][j];
         return result;
      }
   }
   public static void main(String[] args)
   {
      G2D g2d = new G2D();
      g2d.rotate(45);
   }
}
```

6. Listing A-28 presents a `ToDo` marker annotation type that annotates only type elements, and that also uses the default retention policy.

Listing A-28. *The `ToDo` annotation type for marking types that need to be completed*

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Target;

@Target(ElementType.TYPE)
public @interface ToDo
{
}
```

7. Listing A-29 presents a rewritten `StubFinder` application that works with Listing 3-43's `Stub` annotation type (with appropriate `@Target` and `@Retention` annotations) and Listing 3-44's `Deck` class.

Listing A-29. *Reporting a stub's ID, due date, and developer via a new version of `StubFinder`*

```
import java.lang.reflect.Method;

class StubFinder
{
   public static void main(String[] args) throws Exception
   {
      if (args.length != 1)
      {
```

```
            System.err.println("usage: java StubFinder classfile");
            return;
         }
         Method[] methods = Class.forName(args[0]).getMethods();
         for (int i = 0; i < methods.length; i++)
            if (methods[i].isAnnotationPresent(Stub.class))
            {
               Stub stub = methods[i].getAnnotation(Stub.class);
               System.out.println("Stub ID = "+stub.id());
               System.out.println("Stub Date = "+stub.dueDate());
               System.out.println("Stub Developer = "+stub.developer());
               System.out.println();
            }
      }
}
```

8. Listing A-30 presents the generic `Stack` class and the `StackEmptyException` and `StackFullException` helper classes.

Listing A-30. *Not all helper classes need to be nested*

```
class Stack<E>
{
   private E[] elements;
   private int top;
   @SuppressWarnings("unchecked")
   Stack(int size)
   {
      if (size < 2)
         throw new IllegalArgumentException(""+size);
      elements = (E[]) new Object[size];
      top = -1;
   }
   void push(E element) throws StackFullException
   {
      if (top == elements.length-1)
         throw new StackFullException();
      elements[++top] = element;
   }
   E pop() throws StackEmptyException
   {
      if (isEmpty())
```

```
            throw new StackEmptyException();
         return elements[top--];
      }
      boolean isEmpty()
      {
         return top == -1;
      }
      public static void main(String[] args)
         throws StackFullException, StackEmptyException
      {
         Stack<String> stack = new Stack<String>(5);
         assert stack.isEmpty();
         stack.push("A");
         stack.push("B");
         stack.push("C");
         stack.push("D");
         stack.push("E");
         // Uncomment the following line to generate a StackFullException.
         //stack.push("F");
         while (!stack.isEmpty())
            System.out.println(stack.pop());
         // Uncomment the following line to generate a StackEmptyException.
         //stack.pop();
         assert stack.isEmpty();
      }
}
class StackEmptyException extends Exception
{
}
class StackFullException extends Exception
{
}
```

9. Listings A-31 and A-32 present the `Compass` enum and the `UseCompass` class.

Listing A-31. *A `Compass` enum with four direction constants*

```
enum Compass
{
   NORTH, SOUTH, EAST, WEST
}
```

Listing A-32. *Using the* `Compass` *enum to keep from getting lost*

```
class UseCompass
{
   public static void main(String[] args)
   {
      int i = (int) (Math.random()*4);
      Compass[] dir = { Compass.NORTH, Compass.EAST, Compass.SOUTH,
                        Compass.WEST };
      switch(dir[i])
      {
         case NORTH: System.out.println("heading north"); break;
         case EAST : System.out.println("heading east"); break;
         case SOUTH: System.out.println("heading south"); break;
         case WEST : System.out.println("heading west"); break;
         default   : assert false; // Should never be reached.
      }
   }
}
```

Chapter 4: Touring Language APIs

Chapter 4's exercises test your understanding of Java's language APIs:

1. Listing A-33 presents the `PrimeNumberTest` class.

Listing A-33. *Testing a positive integer for primality*

```
class PrimeNumberTest
{
   public static void main(String[] args)
   {
      if (args.length != 1)
      {
         System.err.println("usage: java PrimeNumberTest integer");
         System.err.println("integer must be 2 or higher");
         return;
      }
      try
      {
         int n = Integer.parseInt(args[0]);
         if (n < 2)
```

```
         {
             System.err.println(n+" is invalid because it is less than 2");
             return;
         }
         for (int i = 2; i <= Math.sqrt(n); i++)
            if (n%i == 0)
            {
               System.out.println(n+" is not prime");
               return;
            }
         System.out.println(n+" is prime");
      }
      catch (NumberFormatException nfe)
      {
         System.err.println("unable to parse "+args[0]+" into an int");
      }
   }
}
```

The `java.math.BigInteger` class declares a `boolean isProbablePrime(int certainty)` method that you can use to test the primality of very large integers.

2. Listings A-34 through A-36 present the first and second versions of the `Driver` class, and the `DriverDemo` class.

Listing A-34. *A basic driver*

```
class Driver
{
   public String getCapabilities()
   {
      return "basic capabilities";
   }
}
```

Listing A-35. *An extended driver*

```
class Driver
{
   public String getCapabilities()
   {
      return "basic capabilities";
   }
   public String getCapabilitiesEx()
```

```
   {
      return "extended capabilities";
   }
}
```

Listing A-36. *Detecting and invoking a driver with extended or default capabilities*

```
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

class DriverDemo
{
   public static void main(String[] args)
   {
      if (args.length != 0)
      {
         System.err.println("usage: java DriverDemo");
         return;
      }
      try
      {
         Class<?> clazz = Class.forName("Driver");
         Driver driver = (Driver) clazz.newInstance();
         try
         {
            Method methodExt = clazz.getMethod("getCapabilitiesEx",
                                               (Class<?>[]) null);
            System.out.println(methodExt.invoke(driver, (Object[]) null));
         }
         catch (NoSuchMethodException nsme)
         {
            System.err.println("No getCapabilitiesEx() method");
            try
            {
               Method methodDef = clazz.getMethod("getCapabilities",
                                                  (Class<?>[])  null);
               System.out.println(methodDef.invoke(driver, (Object[]) null));
            }
            catch (NoSuchMethodException nsme2)
            {
               System.out.println("No getCapabilities() method");
```

```
            }
            catch (InvocationTargetException ite)
            {
               System.err.println("exception thrown from getCapabilities()");
            }
         }
         catch (InvocationTargetException ite)
         {
            System.err.println("exception thrown from getCapabilitiesEx()");
         }
      }
      catch (ClassNotFoundException cnfe)
      {
         System.err.println("Class not found: "+cnfe.getMessage());
      }
      catch (IllegalAccessException iae)
      {
         System.err.println("Illegal access: "+iae.getMessage());
      }
      catch (InstantiationException ie)
      {
         System.err.println("Unable to instantiate loaded class");
      }
   }
}
```

3. Listing A-37 presents the `GArray` class.

Listing A-37. *Growing a Java array*

```
package ca.tutortutor.collections;

public class GArray<E>
{
   private E[] array;
   private int size;
   @SuppressWarnings("unchecked")
   public GArray(int initCapacity)
   {
      array = (E[]) new Object[initCapacity];
      size = 0;
```

```
   }
   public E get(int index)
   {
      if (index < 0 || index >= size)
         throw new ArrayIndexOutOfBoundsException(index+" out of bounds");
      return array[index];
   }
   public void set(int index, E value)
   {
      if (index < 0)
         throw new ArrayIndexOutOfBoundsException(index+" out of bounds");
      if (index >= array.length)
      {
         @SuppressWarnings("unchecked")
         E[] array2 = (E[]) new Object[index*2];
         System.arraycopy(array, 0, array2, 0, size);
         array = array2;
      }
      array[index] = value;
      if (index >= size)
         size = index+1;
   }
   public int size()
   {
      return size;
   }
}
```

4. Listing A-38 presents the refactored `CountingThreads` class.

Listing A-38. *Counting via daemon threads*

```
class CountingThreads
{
   public static void main(String[] args)
   {
      Runnable r = new Runnable()
                   {
                      @Override
                      public void run()
                      {
```

```
                        String name = Thread.currentThread().getName();
                        int count = 0;
                        while (true)
                           System.out.println(name+": "+count++);
                     }
                  };
      Thread thdA = new Thread(r);
      thdA.setDaemon(true);
      Thread thdB = new Thread(r);
      thdB.setDaemon(true);
      thdA.start();
      thdB.start();
   }
}
```

When you run this application, the two daemon threads start executing and you might see some output. However, the application will end as soon as the default main thread leaves the `main()` method and dies.

5. Listing A-39 presents `StopCountingThreads`, a refactored `CountingThreads` class.

Listing A-39. *Stopping the counting threads when Enter is pressed*

```
import java.io.IOException;


class StopCountingThreads
{
   private static volatile boolean stopped = false;
   public static void main(String[] args)
   {
      Runnable r = new Runnable()
                   {
                      @Override
                      public void run()
                      {
                         String name = Thread.currentThread().getName();
                         int count = 0;
                         while (!stopped)
                            System.out.println(name+": "+count++);
                      }
                   };
      Thread thdA = new Thread(r);
```

```
      Thread thdB = new Thread(r);
      thdA.start();
      thdB.start();
      try { System.in.read(); } catch (IOException ioe) {}
      stopped = true;
   }
}
```

Chapter 5: Collecting Objects

Chapter 5's exercises test your understanding of collections:

1. Listings A-40 and A-41 present the `JavaQuiz` and `QuizEntry` classes.

Listing A-40. *How much do you know about Java? Take the quiz and find out!*

```
import java.io.IOException;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

class JavaQuiz
{
   static QuizEntry[] quizEntries =
   {
      new QuizEntry("What was Java's original name?",
                    new String[] { "Oak", "Duke", "J", "None of the above" },
                    'A'),
      new QuizEntry("Which of the following reserved words is also a literal?",
                    new String[] { "for", "long", "true", "enum" },
                    'C'),
      new QuizEntry("The conditional operator (?:) resembles which statement?",
                    new String[] { "switch", "if-else", "if", "while" },
                    'B')
   };
   public static void main(String[] args)
   {
      // Populate the quiz list.
      List<QuizEntry> quiz = new ArrayList<QuizEntry>();
      for (QuizEntry entry: quizEntries)
```

```
            quiz.add(entry);
         // Perform the quiz.
         System.out.println("Java Quiz");
         System.out.println("---------\n");
         Iterator<QuizEntry> iter = quiz.iterator();
         while (iter.hasNext())
         {
            QuizEntry qe = iter.next();
            System.out.println(qe.getQuestion());
            String[] choices = qe.getChoices();
            for (int i = 0; i < choices.length; i++)
               System.out.println("  "+(char) ('A'+i)+": "+choices[i]);
            int choice = -1;
            while (choice < 'A' || choice > 'A'+choices.length)
            {
               System.out.print("Enter choice letter: ");
               try
               {
                  choice = System.in.read();
                  // Remove trailing characters up to and including the newline
                  // to avoid having these characters automatically returned in
                  // subsequent System.in.read() method calls.
                  while (System.in.read() != '\n');
                  choice = Character.toUpperCase((char) choice);
               }
               catch (IOException ioe)
               {
               }
            }
            if (choice == qe.getAnswer())
               System.out.println("You are correct!\n");
            else
               System.out.println("You are not correct!\n");
         }
      }
   }
```

Listing A-41. *A helper class for storing a quiz's data*

```
class QuizEntry
{
   private String question;
   private String[] choices;
   private char answer;
   QuizEntry(String question, String[] choices, char answer)
   {
      this.question = question;
      this.choices = choices;
      this.answer = answer;
   }
   String[] getChoices()
   {
      // Demonstrate returning a copy of the choices array to prevent clients
      // from directly manipulating (and possibly screwing up) the internal
      // choices array.
      String[] temp = new String[choices.length];
      System.arraycopy(choices, 0, temp, 0, choices.length);
      return temp;
   }
   String getQuestion()
   {
      return question;
   }
   char getAnswer()
   {
      return answer;
   }
}
```

2. Listing A-42 presents the `WC` class.

Listing A-42. *Storing words and their frequency counts in a map*

```
import java.io.IOException;

import java.util.Iterator;
import java.util.Map;
import java.util.TreeMap;
```

```
class WC
{
   public static void main(String[] args) throws IOException
   {
      int ch;
      Map<String, Integer> map = new TreeMap<>();
      // Read each character from standard input until a letter
      // is read. This letter indicates the start of a word.
      while ((ch = System.in.read()) != -1)
      {
         // If character is a letter then start of word detected.
         if (Character.isLetter((char) ch))
         {
            // Create StringBuffer object to hold word letters.
            StringBuffer sb = new StringBuffer();
            // Place first letter character into StringBuffer object.
            sb.append((char) ch);
            // Place all subsequent letter characters into StringBuffer
            // object.
            do
            {
               ch = System.in.read();
               if (Character.isLetter((char) ch))
                  sb.append((char) ch);
               else
                  break;
            }
            while (true);
            // Insert word into map.
            String word = sb.toString();
            if (map.get(word) == null)
                map.put(word, 1);
            else
                map.put(word, map.get(word)+1);
         }
      }
      // Output map in ascending order.
      Iterator i = map.entrySet().iterator();
      while (i.hasNext())
         System.out.println(i.next());
```

```
   }
}
```

3. Listing A-43 presents the `FrequencyDemo` class.

Listing A-43. *Reporting the frequency of last command-line argument occurrences in the previous command-line arguments*

```
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;

class FrequencyDemo
{
   public static void main(String[] args)
   {
      List<String> listOfArgs = new LinkedList<>();
      String lastArg = (args.length == 0) ? null : args[args.length-1];
      for (int i = 0; i < args.length-1; i++)
         listOfArgs.add(args[i]);
      System.out.println("Number of occurrences of "+lastArg+" = "+
                         Collections.frequency(listOfArgs, lastArg));
   }
}
```

Chapter 6: Touring Additional Utility APIs

Chapter 6's exercises test your understanding of the concurrency utilities, `Objects`, and `Random`:

1. Listing A-44 presents the `Pool` and `SemaphoreDemo` classes.

Listing A-44. *Acquiring a resource via a semaphore*

```
import java.util.concurrent.Semaphore;

final class Pool
{
   private static final int MAX_AVAILABLE = 1;
   private final Semaphore available = new Semaphore(MAX_AVAILABLE, true);
   Object getItem() throws InterruptedException
   {
      available.acquire();
      return getNextAvailableItem();
```

```
    }
    void putItem(Object x)
    {
       if (markAsUnused(x))
          available.release();
    }
    // Not a particularly efficient data structure; just for demo
    Object[] items = new String[] { "single pooled resource" };
    boolean[] used = new boolean[MAX_AVAILABLE];
    synchronized Object getNextAvailableItem()
    {
       for (int i = 0; i < MAX_AVAILABLE; ++i)
          if (!used[i])
          {
             used[i] = true;
             return items[i];
          }
       return null; // not reached
    }
    synchronized boolean markAsUnused(Object item)
    {
       for (int i = 0; i < MAX_AVAILABLE; ++i)
       {
          if (item == items[i])
          {
             if (used[i])
             {
                used[i] = false;
                return true;
             }
             else
                return false;
          }
       }
       return false;
    }
}
class SemaphoreDemo
{
    static volatile Pool pool = new Pool();
```

```
   public static void main(String[] args)
   {
      Runnable r = new Runnable()
                   {
                      public void run()
                      {
                         try
                         {
                            String name = Thread.currentThread().getName();
                            while (true)
                            {
                               System.out.println(name+" wants resource");
                               Object o = pool.getItem();
                               System.out.println(name+" gets "+o);
                               System.out.println(name+" using resource");
                               Thread.sleep(3000);
                               System.out.println(name+" returns resource");
                               pool.putItem(o);
                            }
                         }
                         catch (InterruptedException ie)
                         {
                         }
                      }
                   };
      Thread dick = new Thread(r, "Dick");
      Thread jane = new Thread(r, "Jane");
      dick.start();
      jane.start();
   }
}
```

2. Listing A-45 presents the `EqualsDemo`, `Car`, and `Wheel` classes.

Listing A-45. *Discovering arrays that are deeply equal*

```
import java.util.Objects;

class EqualsDemo
{
   public static void main(String[] args)
```

```
   {
      Car[] cars1 = { new Car(4, "Goodyear"), new Car(4, "Goodyear") };
      Car[] cars2 = { new Car(4, "Goodyear"), new Car(4, "Goodyear") };
      Car[] cars3 = { new Car(4, "Michelin"), new Car(4, "Goodyear") };
      Car[] cars4 = { new Car(3, "Goodyear"), new Car(4, "Goodyear") };
      Car[] cars5 = { new Car(4, "Goodyear"), new Car(4, "Goodyear"),
                      new Car(3, "Michelin") };
      System.out.println(Objects.deepEquals(cars1, cars2)); // Output: true
      System.out.println(Objects.deepEquals(cars1, cars3)); // Output: false
      System.out.println(Objects.deepEquals(cars1, cars4)); // Output: false
      System.out.println(Objects.deepEquals(cars1, cars5)); // Output: false
   }
}
class Car
{
   Wheel[] wheels;
   Car(int numWheels, String wheelBrand)
   {
      wheels = new Wheel[numWheels];
      for (int i = 0; i < wheels.length; i++)
         wheels[i] = new Wheel(wheelBrand);
   }
   @Override
   public boolean equals(Object o)
   {
      if (!(o instanceof Car))
         return false;
      Car car = (Car) o;
      if (wheels.length != car.wheels.length)
         return false;
      boolean wheelsEqual = false;
      for (int i = 0; i < wheels.length; i++)
         if (!wheels[i].equals(car.wheels[i]))
            return false;
      return true;
   }
}
class Wheel
{
   String brand;
```

```
   Wheel(String brand)
   {
      this.brand = brand;
   }
   @Override
   public boolean equals(Object o)
   {
      if (!(o instanceof Wheel))
         return false;
      Wheel wheel = (Wheel) o;
      return brand.equals(wheel.brand);
   }
}
```

3. Listing A-46 presents the `Die` class.

Listing A-46. *Rolling a die via the* `Random` *class*

```
import java.util.Random;

class Die
{
   public static void main(String[] args)
   {
      Random r = new Random();
      int die = r.nextInt(6)+1;
      System.out.println(die);
   }
}
```

Chapter 7: Creating and Enriching Graphical User Interfaces

Chapter 7's exercises test your understanding of AWT, Swing, and Java 2D:

1. Listing A-47 presents the `RandomCircles` class.

Listing A-47. *Displaying a randomly colored/positioned/sized filled circle in response to a mouse button press*

```
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
```

```
import java.awt.EventQueue;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.Graphics2D;

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

class RandomCircles extends Frame
{
   RandomCircles()
   {
      super("Random Circles");
      addWindowListener(new WindowAdapter()
                        {
                           @Override
                           public void windowClosing(WindowEvent we)
                           {
                              dispose();
                           }
                        });
      add(new RCCanvas());
      pack();
      setResizable(false);
      setVisible(true);
   }
   public static void main(String[] args)
   {
      Runnable r = new Runnable()
                   {
                      @Override
                      public void run()
                      {
                         new RandomCircles();
                      }
                   };
```

```
      EventQueue.invokeLater(r);
   }
}
class RCCanvas extends Canvas
{
   private Dimension d;
   RCCanvas()
   {
      d = new Dimension(250, 250);
      addMouseListener(new MouseAdapter()
                       {
                          @Override
                          public void mouseClicked(MouseEvent me)
                          {
                             repaint();
                          }
                       });
   }
   @Override
   public Dimension getPreferredSize()
   {
      return d;
   }
   @Override
   public void paint(Graphics g)
   {
      g.setColor(new Color(rnd(256), rnd(256), rnd(256)));
      int extent = 5+rnd(31);
      g.fillOval(rnd(getWidth()), rnd(getHeight()), extent, extent);
   }
   int rnd(int limit)
   {
      return (int) (Math.random()*limit);
   }
}
```

2. Listing A-48 presents the `TempVerter` class with the Nimbus Look and Feel.

Listing A-48. *Giving* `TempVerter` *a modern look and feel*

```
import java.awt.Container;
import java.awt.EventQueue;
import java.awt.FlowLayout;
import java.awt.GridLayout;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.UIManager;

import javax.swing.border.Border;
import javax.swing.border.EtchedBorder;

class TempVerter
{
   static JPanel createGUI()
   {
      JPanel pnlLayout = new JPanel();
      pnlLayout.setLayout(new GridLayout(3, 1));
      JPanel pnlTemp = new JPanel();
      ((FlowLayout) pnlTemp.getLayout()).setAlignment(FlowLayout.LEFT);
      pnlTemp.add(new JLabel("Degrees"));
      final JTextField txtDegrees = new JTextField(10);
      txtDegrees.setToolTipText("Enter a numeric value in this field.");
      pnlTemp.add(txtDegrees);
      pnlLayout.add(pnlTemp);
      pnlTemp = new JPanel();
      ((FlowLayout) pnlTemp.getLayout()).setAlignment(FlowLayout.LEFT);
      pnlTemp.add(new JLabel("Result"));
```

```
final JTextField txtResult = new JTextField(30);
txtResult.setToolTipText("Don't enter anything in this field.");
pnlTemp.add(txtResult);
pnlLayout.add(pnlTemp);
pnlTemp = new JPanel();
ImageIcon ii = new ImageIcon("thermometer.gif");
ActionListener al;
al = new ActionListener()
     {
        @Override
        public void actionPerformed(ActionEvent ae)
        {
           try
           {
              double value = Double.parseDouble(txtDegrees.getText());
              double result = (value-32.0)*5.0/9.0;
              txtResult.setText("Celsius = "+result);
           }
           catch (NumberFormatException nfe)
           {
              System.err.println("bad input");
           }
        }
     };
JButton btnConvertToCelsius = new JButton("Convert to Celsius", ii);
btnConvertToCelsius.addActionListener(al);
pnlTemp.add(btnConvertToCelsius);
al = new ActionListener()
     {
        @Override
        public void actionPerformed(ActionEvent ae)
        {
           try
           {
              double value = Double.parseDouble(txtDegrees.getText());
              double result = value*9.0/5.0+32.0;
              txtResult.setText("Fahrenheit = "+result);
           }
           catch (NumberFormatException nfe)
           {
```

```
                System.err.println("bad input");
             }
          }
       };
   JButton btnConvertToFahrenheit = new JButton("Convert to Fahrenheit", ii);
   btnConvertToFahrenheit.addActionListener(al);
   pnlTemp.add(btnConvertToFahrenheit);
   Border border = BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
   pnlTemp.setBorder(border);
   pnlLayout.add(pnlTemp);
   return pnlLayout;
}
static void fixGUI(Container c)
{
   JPanel pnlRow = (JPanel) c.getComponents()[0];
   JLabel l1 = (JLabel) pnlRow.getComponents()[0];
   pnlRow = (JPanel) c.getComponents()[1];
   JLabel l2 = (JLabel) pnlRow.getComponents()[0];
   l2.setPreferredSize(l1.getPreferredSize());
   pnlRow = (JPanel) c.getComponents()[2];
   JButton btnToC = (JButton) pnlRow.getComponents()[0];
   JButton btnToF = (JButton) pnlRow.getComponents()[1];
   btnToC.setPreferredSize(btnToF.getPreferredSize());
}
public static void main(String[] args)
{
   Runnable r = new Runnable()
                {
                   @Override
                   public void run()
                   {
                      try
                      {
                         String lnf;
                         lnf = "javax.swing.plaf.nimbus.NimbusLookAndFeel";
                         UIManager.setLookAndFeel(lnf);
                      }
                      catch (Exception e)
                      {
                         // ignore if look and feel cannot be set
```

```
                        }
                        final JFrame f = new JFrame("TempVerter");
                        f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                        Border b = BorderFactory.createEmptyBorder(5, 5, 5, 5);
                        f.getRootPane().setBorder(b);
                        f.setContentPane(createGUI());
                        fixGUI(f.getContentPane());
                        f.pack();
                        f.setResizable(false);
                        f.setVisible(true);
                    }
                };
        EventQueue.invokeLater(r);
    }
}
```

3. Listing A-49 presents the `TempVerter` class with button mnemonics.

Listing A-49. *Adding keyboard shortcuts to the Convert to Celsius and Convert to Fahrenheit buttons*

```
import java.awt.Container;
import java.awt.EventQueue;
import java.awt.FlowLayout;
import java.awt.GridLayout;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;

import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

import javax.swing.border.Border;
import javax.swing.border.EtchedBorder;

class TempVerter
```

```
{
   static JPanel createGUI()
   {
      JPanel pnlLayout = new JPanel();
      pnlLayout.setLayout(new GridLayout(3, 1));
      JPanel pnlTemp = new JPanel();
      ((FlowLayout) pnlTemp.getLayout()).setAlignment(FlowLayout.LEFT);
      pnlTemp.add(new JLabel("Degrees"));
      final JTextField txtDegrees = new JTextField(10);
      txtDegrees.setToolTipText("Enter a numeric value in this field.");
      pnlTemp.add(txtDegrees);
      pnlLayout.add(pnlTemp);
      pnlTemp = new JPanel();
      ((FlowLayout) pnlTemp.getLayout()).setAlignment(FlowLayout.LEFT);
      pnlTemp.add(new JLabel("Result"));
      final JTextField txtResult = new JTextField(30);
      txtResult.setToolTipText("Don't enter anything in this field.");
      pnlTemp.add(txtResult);
      pnlLayout.add(pnlTemp);
      pnlTemp = new JPanel();
      ImageIcon ii = new ImageIcon("thermometer.gif");
      ActionListener al;
      al = new ActionListener()
           {
              @Override
              public void actionPerformed(ActionEvent ae)
              {
                 try
                 {
                    double value = Double.parseDouble(txtDegrees.getText());
                    double result = (value-32.0)*5.0/9.0;
                    txtResult.setText("Celsius = "+result);
                 }
                 catch (NumberFormatException nfe)
                 {
                    System.err.println("bad input");
                 }
              }
           };
      JButton btnConvertToCelsius = new JButton("Convert to Celsius", ii);
```

```
        btnConvertToCelsius.setMnemonic(KeyEvent.VK_C);
        btnConvertToCelsius.addActionListener(al);
        pnlTemp.add(btnConvertToCelsius);
        al = new ActionListener()
             {
                @Override
                public void actionPerformed(ActionEvent ae)
                {
                   try
                   {
                      double value = Double.parseDouble(txtDegrees.getText());
                      double result = value*9.0/5.0+32.0;
                      txtResult.setText("Fahrenheit = "+result);
                   }
                   catch (NumberFormatException nfe)
                   {
                      System.err.println("bad input");
                   }
                }
             };
        JButton btnConvertToFahrenheit = new JButton("Convert to Fahrenheit", ii);
        btnConvertToFahrenheit.setMnemonic(KeyEvent.VK_F);
        btnConvertToFahrenheit.addActionListener(al);
        pnlTemp.add(btnConvertToFahrenheit);
        Border border = BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
        pnlTemp.setBorder(border);
        pnlLayout.add(pnlTemp);
        return pnlLayout;
    }
    static void fixGUI(Container c)
    {
        JPanel pnlRow = (JPanel) c.getComponents()[0];
        JLabel l1 = (JLabel) pnlRow.getComponents()[0];
        pnlRow = (JPanel) c.getComponents()[1];
        JLabel l2 = (JLabel) pnlRow.getComponents()[0];
        l2.setPreferredSize(l1.getPreferredSize());
        pnlRow = (JPanel) c.getComponents()[2];
        JButton btnToC = (JButton) pnlRow.getComponents()[0];
        JButton btnToF = (JButton) pnlRow.getComponents()[1];
        btnToC.setPreferredSize(btnToF.getPreferredSize());
```

```
   }
   public static void main(String[] args)
   {
      Runnable r = new Runnable()
                   {
                      @Override
                      public void run()
                      {
                         final JFrame f = new JFrame("TempVerter");
                         f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                         Border b = BorderFactory.createEmptyBorder(5, 5, 5, 5);
                         f.getRootPane().setBorder(b);
                         f.setContentPane(createGUI());
                         fixGUI(f.getContentPane());
                         f.pack();
                         f.setResizable(false);
                         f.setVisible(true);
                      }
                   };
      EventQueue.invokeLater(r);
   }
}
```

4. Listing A-50 presents the `SplashScreen` class with support for antialiasing.

Listing A-50. *Removing the "jaggies" from* `Geometria`*'s pseudo-splash screen*

```
class SplashCanvas extends Canvas
{
   private Dimension d;
   private Font f;
   private String title;
   private boolean invert; // defaults to false (no invert)
   SplashCanvas()
   {
      d = new Dimension(250, 250);
      f = new Font("Arial", Font.BOLD, 50);
      title = "Geometria";
      addMouseListener(new MouseAdapter()
                       {
                          @Override
```

```
                        public void mouseClicked(MouseEvent me)
                        {
                           invert = !invert;
                           repaint();
                        }
                     });
   }
   @Override
   public Dimension getPreferredSize()
   {
      return d;
   }
   @Override
   public void paint(Graphics g)
   {
      ((Graphics2D) g).setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                                        RenderingHints.VALUE_ANTIALIAS_ON);
      int width = getWidth();
      int height = getHeight();
      g.setColor(invert ? Color.BLACK : Color.WHITE);
      g.fillRect(0, 0, width, height);
      g.setColor(invert ? Color.WHITE : Color.BLACK);
      for (int y = 0; y < height; y += 5)
         for (int x = 0; x < width; x += 5)
            g.drawLine(x, y, width-x, height-y);
      g.setColor(Color.YELLOW);
      g.setFont(f);
      FontMetrics fm = g.getFontMetrics();
      int strwid = fm.stringWidth(title);
      g.drawString(title, (width-strwid)/2, height/2);
      g.setColor(Color.RED);
      strwid = fm.stringWidth(title);
      g.drawString(title, (width-strwid)/2+3, height/2+3);
      g.setColor(Color.GREEN);
      g.fillOval(10, 10, 50, 50);
      g.setColor(Color.BLUE);
      g.fillRect(width-60, height-60, 50, 50);
   }
}
```

5. Listing A-51 presents the `SplashScreen` class with improved GUI response.

Listing A-51. *Making the "jaggie"-free pseudo-splash screen more responsive*

```
class SplashCanvas extends Canvas
{
   private Dimension d;
   private Font f;
   private String title;
   private boolean invert; // defaults to false (no invert)
   private BufferedImage bi, biInvert;
   SplashCanvas()
   {
      d = new Dimension(250, 250);
      f = new Font("Arial", Font.BOLD, 50);
      title = "Geometria";
      addMouseListener(new MouseAdapter()
                       {
                          @Override
                          public void mouseClicked(MouseEvent me)
                          {
                             invert = !invert;
                             repaint();
                          }
                       });
      bi = new BufferedImage(250, 250, BufferedImage.TYPE_INT_RGB);
      biInvert = new BufferedImage(250, 250, BufferedImage.TYPE_INT_RGB);
      makeImage(bi, false);
      makeImage(biInvert, true);
   }
   @Override
   public Dimension getPreferredSize()
   {
      return d;
   }
   private void makeImage(BufferedImage bi, boolean invert)
   {
      Graphics2D g2d = bi.createGraphics();
      g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                           RenderingHints.VALUE_ANTIALIAS_ON);
```

```
      int width = d.width;
      int height = d.height;
      g2d.setColor(Color.WHITE);
      g2d.fillRect(0, 0, width, height);
      g2d.setColor(Color.BLACK);
      if (invert)
      {
         g2d.fillRect(0, 0, width, height);
         g2d.setColor(Color.WHITE);
      }
      for (int y = 0; y < height; y += 5)
         for (int x = 0; x < width; x += 5)
            g2d.drawLine(x, y, width-x, height-y);
      g2d.setColor(Color.YELLOW);
      g2d.setFont(f);
      FontMetrics fm = g2d.getFontMetrics();
      int strwid = fm.stringWidth(title);
      g2d.drawString(title, (width-strwid)/2, height/2);
      g2d.setColor(Color.RED);
      strwid = fm.stringWidth(title);
      g2d.drawString(title, (width-strwid)/2+3, height/2+3);
      g2d.setColor(Color.GREEN);
      g2d.fillOval(10, 10, 50, 50);
      g2d.setColor(Color.BLUE);
      g2d.fillRect(width-60, height-60, 50, 50);
      g2d.dispose();
   }
   @Override
   public void paint(Graphics g)
   {
      g.drawImage((invert) ? biInvert : bi, 0, 0, null);
   }
}
```

I specify `int width = d.width;` and `int height = d.height;` instead of `int width = getWidth();` and `int height = getHeight();` because `getWidth()` and `getHeight()` each return 0 at this point; the `getPreferredSize()` method has yet to be called.

6. Listing A-52 presents the `BIP` class with a Blur More menuitem.

Listing A-52. *Adding Blur More to* `BIP`

```
import java.awt.EventQueue;
import java.awt.Graphics2D;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.awt.image.BufferedImage;
import java.awt.image.BufferedImageOp;
import java.awt.image.ConvolveOp;
import java.awt.image.Kernel;
import java.awt.image.LookupOp;
import java.awt.image.ShortLookupTable;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;

class BIP extends JFrame
{
   Picture pic;
   BufferedImage bi;
   BIP()
   {
      super("Buffered Image Processing");
      setDefaultCloseOperation(DISPOSE_ON_CLOSE);
      ImageIcon ii = new ImageIcon("rose.jpg");
      bi = new BufferedImage(ii.getIconWidth(), ii.getIconHeight(),
                             BufferedImage.TYPE_INT_RGB);
      Graphics2D g2d = bi.createGraphics();
      g2d.drawImage(ii.getImage(), 0, 0, null);
      g2d.dispose();
      JMenuBar mb = new JMenuBar();
      JMenu m = new JMenu("Process");
      JMenuItem mi = new JMenuItem("Undo");
```

```
mi.addActionListener(new ActionListener()
                     {
                        @Override
                        public void actionPerformed(ActionEvent ae)
                        {
                           pic.setImage(bi);
                        }
                     });
m.add(mi);
m.addSeparator();
mi = new JMenuItem("Blur");
mi.addActionListener(new ActionListener()
                     {
                        @Override
                        public void actionPerformed(ActionEvent ae)
                        {
                           float ninth = 1.0f/9.0f;
                           float[] blurKernel =
                           {
                              ninth, ninth, ninth,
                              ninth, ninth, ninth,
                              ninth, ninth, ninth
                           };
                           BufferedImageOp blurOp =
                              new ConvolveOp(new Kernel(3, 3,
                                                        blurKernel));
                           BufferedImage biRes = blurOp.filter(bi, null);
                           pic.setImage(biRes);
                        }
                     });
m.add(mi);
mi = new JMenuItem("Blur More");
mi.addActionListener(new ActionListener()
                     {
                        @Override
                        public void actionPerformed(ActionEvent ae)
                        {
                           float tfth = 1.0f/25.0f;
                           float[] blurKernel =
                           {
```

```
                        tfth, tfth, tfth, tfth, tfth,
                        tfth, tfth, tfth, tfth, tfth,
                        tfth, tfth, tfth, tfth, tfth,
                        tfth, tfth, tfth, tfth, tfth,
                        tfth, tfth, tfth, tfth, tfth
                     };
                     BufferedImageOp blurOp =
                        new ConvolveOp(new Kernel(5, 5,
                                                  blurKernel));
                     BufferedImage biRes = blurOp.filter(bi, null);
                     pic.setImage(biRes);
                  }
               });
m.add(mi);
mi = new JMenuItem("Edge");
mi.addActionListener(new ActionListener()
               {
                  @Override
                  public void actionPerformed(ActionEvent ae)
                  {
                     float[] edgeKernel =
                     {
                         0.0f, -1.0f,  0.0f,
                        -1.0f,  4.0f, -1.0f,
                         0.0f, -1.0f,  0.0f
                     };
                     BufferedImageOp edgeOp =
                        new ConvolveOp(new Kernel(3, 3, edgeKernel));
                     BufferedImage biRes = edgeOp.filter(bi, null);
                     pic.setImage(biRes);
                  }
               });
m.add(mi);
mi = new JMenuItem("Negative");
mi.addActionListener(new ActionListener()
               {
                  @Override
                  public void actionPerformed(ActionEvent ae)
                  {
                     short[] invert = new short[256];
```

```
                        for (int i = 0; i < invert.length; i++)
                           invert[i] = (short) (255-i);
                        BufferedImageOp invertOp =
                           new LookupOp(new ShortLookupTable(0,
                                                             invert),
                                        null);
                        BufferedImage biRes = invertOp.filter(bi,
                                                              null);
                        pic.setImage(biRes);
                     }
                  });
m.add(mi);
mi = new JMenuItem("Sharpen");
mi.addActionListener(new ActionListener()
                  {
                     @Override
                     public void actionPerformed(ActionEvent ae)
                     {
                        float[] sharpenKernel =
                        {
                            0.0f, -1.0f,  0.0f,
                           -1.0f,  5.0f, -1.0f,
                            0.0f, -1.0f,  0.0f
                        };
                        BufferedImageOp sharpenOp =
                           new ConvolveOp(new Kernel(3, 3,
                                                     sharpenKernel));
                        BufferedImage biRes = sharpenOp.filter(bi,
                                                               null);
                        pic.setImage(biRes);
                     }
                  });
m.add(mi);
m.addSeparator();
mi = new JMenuItem("Exit");
mi.addActionListener(new ActionListener()
                  {
                     @Override
                     public void actionPerformed(ActionEvent ae)
                     {
```

```
                                dispose();
                            }
                        });
        m.add(mi);
        mb.add(m);
        setJMenuBar(mb);
        setContentPane(pic = new Picture(bi.getWidth(), bi.getHeight(), bi));
        pack();
        setResizable(false);
        setVisible(true);
    }
    public static void main(String[] args)
    {
        Runnable r = new Runnable()
                    {
                        @Override
                        public void run()
                        {
                            new BIP();
                        }
                    };
        EventQueue.invokeLater(r);
    }
}
```

7. Listing A-53 presents the `BIP` class with a Grayscale menuitem.

Listing A-53. *Adding Grayscale to* ***BIP***

```
import java.awt.EventQueue;
import java.awt.Graphics2D;

import java.awt.color.ColorSpace;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.awt.image.BufferedImage;
import java.awt.image.BufferedImageOp;
import java.awt.image.ColorConvertOp;
import java.awt.image.ConvolveOp;
```

```
import java.awt.image.Kernel;
import java.awt.image.LookupOp;
import java.awt.image.ShortLookupTable;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;

class BIP extends JFrame
{
   Picture pic;
   BufferedImage bi;
   BIP()
   {
      super("Buffered Image Processing");
      setDefaultCloseOperation(DISPOSE_ON_CLOSE);
      ImageIcon ii = new ImageIcon("rose.jpg");
      bi = new BufferedImage(ii.getIconWidth(), ii.getIconHeight(),
                             BufferedImage.TYPE_INT_RGB);
      Graphics2D g2d = bi.createGraphics();
      g2d.drawImage(ii.getImage(), 0, 0, null);
      g2d.dispose();
      JMenuBar mb = new JMenuBar();
      JMenu m = new JMenu("Process");
      JMenuItem mi = new JMenuItem("Undo");
      mi.addActionListener(new ActionListener()
                           {
                              @Override
                              public void actionPerformed(ActionEvent ae)
                              {
                                 pic.setImage(bi);
                              }
                           });
      m.add(mi);
      m.addSeparator();
      mi = new JMenuItem("Blur");
      mi.addActionListener(new ActionListener()
                           {
```

```
                        @Override
                        public void actionPerformed(ActionEvent ae)
                        {
                           float ninth = 1.0f/9.0f;
                           float[] blurKernel =
                           {
                              ninth, ninth, ninth,
                              ninth, ninth, ninth,
                              ninth, ninth, ninth
                           };
                           BufferedImageOp blurOp =
                              new ConvolveOp(new Kernel(3, 3,
                                                        blurKernel));
                           BufferedImage biRes = blurOp.filter(bi, null);
                           pic.setImage(biRes);
                        }
                     });
m.add(mi);
mi = new JMenuItem("Edge");
mi.addActionListener(new ActionListener()
                     {
                        @Override
                        public void actionPerformed(ActionEvent ae)
                        {
                           float[] edgeKernel =
                           {
                               0.0f, -1.0f,  0.0f,
                              -1.0f,  4.0f, -1.0f,
                               0.0f, -1.0f,  0.0f
                           };
                           BufferedImageOp edgeOp =
                              new ConvolveOp(new Kernel(3, 3, edgeKernel));
                           BufferedImage biRes = edgeOp.filter(bi, null);
                           pic.setImage(biRes);
                        }
                     });
m.add(mi);
mi = new JMenuItem("Grayscale");
mi.addActionListener(new ActionListener()
                     {
```

```
                        @Override
                        public void actionPerformed(ActionEvent ae)
                        {
                           ColorSpace cs;
                           cs = ColorSpace.getInstance(ColorSpace.CS_GRAY);
                           BufferedImageOp grayOp =
                              new ColorConvertOp(cs, null);
                           BufferedImage biRes = grayOp.filter(bi, null);
                           pic.setImage(biRes);
                        }
                     });
m.add(mi);
mi = new JMenuItem("Negative");
mi.addActionListener(new ActionListener()
                     {
                        @Override
                        public void actionPerformed(ActionEvent ae)
                        {
                           short[] invert = new short[256];
                           for (int i = 0; i < invert.length; i++)
                              invert[i] = (short) (255-i);
                           BufferedImageOp invertOp =
                              new LookupOp(new ShortLookupTable(0,
                                                                invert),
                                           null);
                           BufferedImage biRes = invertOp.filter(bi,
                                                                 null);
                           pic.setImage(biRes);
                        }
                     });
m.add(mi);
mi = new JMenuItem("Sharpen");
mi.addActionListener(new ActionListener()
                     {
                        @Override
                        public void actionPerformed(ActionEvent ae)
                        {
                           float[] sharpenKernel =
                           {
                               0.0f, -1.0f,  0.0f,
```

```
                                -1.0f,  5.0f, -1.0f,
                                 0.0f, -1.0f,  0.0f
                             };
                             BufferedImageOp sharpenOp =
                                new ConvolveOp(new Kernel(3, 3,
                                                          sharpenKernel));
                             BufferedImage biRes = sharpenOp.filter(bi,
                                                                    null);
                             pic.setImage(biRes);
                          }
                       });
      m.add(mi);
      m.addSeparator();
      mi = new JMenuItem("Exit");
      mi.addActionListener(new ActionListener()
                       {
                          @Override
                          public void actionPerformed(ActionEvent ae)
                          {
                             dispose();
                          }
                       });
      m.add(mi);
      mb.add(m);
      setJMenuBar(mb);
      setContentPane(pic = new Picture(bi.getWidth(), bi.getHeight(), bi));
      pack();
      setResizable(false);
      setVisible(true);
   }
   public static void main(String[] args)
   {
      Runnable r = new Runnable()
                   {
                      @Override
                      public void run()
                      {
                         new BIP();
                      }
                   };
```

```
      EventQueue.invokeLater(r);
   }
}
```

Chapter 8: Interacting with Filesystems

Chapter 8's exercises test your understanding of `File` and various stream and writer/reader APIs:

1. Listing A-54 presents the `Touch` class.

Listing A-54. *Setting a file or directory's timestamp to the current or specified time*

```
import java.io.File;

import java.text.ParseException;
import java.text.SimpleDateFormat;

import java.util.Date;

class Touch
{
   public static void main(String[] args)
   {
      if (args.length != 1 && args.length != 3)
      {
         System.err.println("usage: java Touch [-d timestamp] pathname");
         return;
      }
      long time = new Date().getTime();
      if (args.length == 3)
      {
         if (args[0].equals("-d"))
         {
            try
            {
               SimpleDateFormat sdf;
               sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss z");
               Date date = sdf.parse(args[1]);
               time = date.getTime();
            }
            catch (ParseException pe)
```

```
            {
               pe.printStackTrace();
            }
         }
         else
         {
            System.err.println("invalid option: "+args[0]);
            return;
         }
      }
      new File(args[args.length == 1 ? 0 : 2]).setLastModified(time);
   }
}
```

I discuss `java.util.Date`, `java.text.SimpleDateFormat`, and `java.text.ParseException` in Appendix C's "Internationalization" section.

2. Listing A-55 presents the `Split` class.

Listing A-55. *Splitting a large file into numerous smaller part files*

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

class Split
{
   static final int FILESIZE = 1400000;
   static byte[] buffer = new byte[FILESIZE];
   public static void main(String[] args)
   {
      if (args.length != 1)
      {
         System.err.println("usage: java Split pathname");
         return;
      }
      File file = new File(args[0]);
      long length = file.length();
      int nWholeParts = (int) (length/FILESIZE);
      int remainder = (int) (length%FILESIZE);
```

```
System.out.printf("Spliting %s into %d parts%n", args[0],
                  (remainder == 0) ? nWholeParts : nWholeParts+1);
try (BufferedInputStream bis =
       new BufferedInputStream(new FileInputStream(args[0])))
{
   for (int i = 0; i < nWholeParts; i++)
   {
      bis.read(buffer);
      System.out.println("Writing part "+i);
      try (BufferedOutputStream bos =
             new BufferedOutputStream(new FileOutputStream("part"+i)))
      {
         bos.write(buffer);
      }
      catch (IOException ioe)
      {
         ioe.printStackTrace();
      }
   }
   if (remainder != 0)
   {
      int br = bis.read(buffer);
      if (br != remainder)
      {
         System.err.println("Last part mismatch: expected "+remainder
                            +" bytes");
         System.exit(0);
      }
      System.out.println("Writing part "+nWholeParts);
      try (BufferedOutputStream bos =
             new BufferedOutputStream(new FileOutputStream("part"+
                                                           nWholeParts)))
      {
         bos.write(buffer, 0, remainder);
      }
      catch (IOException ioe)
      {
         ioe.printStackTrace();
      }
   }
```

```
      }
      catch (IOException ioe)
      {
         ioe.printStackTrace();
      }
   }
}
```

3. Listing A-56 presents the `CircleInfo` class.

Listing A-56. *Reading lines of text from standard input that represent circle radii, and outputting circumference and area based on the current radius*

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

class CircleInfo
{
   public static void main(String[] args) throws IOException
   {
      InputStreamReader isr = new InputStreamReader(System.in);
      BufferedReader br = new BufferedReader(isr);
      while (true)
      {
         System.out.print("Enter circle's radius: ");
         String str = br.readLine();
         double radius;
         try
         {
            radius = Double.valueOf(str).doubleValue();
            if (radius <= 0)
               System.err.println("radius must not be 0 or negative");
            else
            {
               System.out.println("Circumference: "+Math.PI*2.0*radius);
               System.out.println("Area: "+Math.PI*radius*radius);
            }
         }
         catch (NumberFormatException nfe)
         {
            break;
```

```
         }
      }
   }
}
```

4. Listing A-57 presents the `FindAll` class.

Listing A-57. *An improved* `FindAll` *application that introduces a Stop button and more*

```
import java.awt.EventQueue;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

class FindAll
{
   final static String LINE_SEPARATOR = System.getProperty("line.separator");
   static JTextArea txtSrchResults;
   static JFrame f;
   static volatile String result;
   static volatile boolean stopped;
   static JPanel createGUI()
   {
      JPanel pnl = new JPanel();
      pnl.setLayout(new BoxLayout(pnl, BoxLayout.Y_AXIS));
      JPanel pnlTemp = new JPanel();
```

```
JLabel lblStartDir = new JLabel("Start directory");
pnlTemp.add(lblStartDir);
final JTextField txtStartDir = new JTextField(30);
pnlTemp.add(txtStartDir);
pnl.add(pnlTemp);
pnlTemp = new JPanel();
JLabel lblSrchText = new JLabel("Search text");
pnlTemp.add(lblSrchText);
lblSrchText.setPreferredSize(lblStartDir.getPreferredSize());
final JTextField txtSrchText = new JTextField(30);
pnlTemp.add(txtSrchText);
pnl.add(pnlTemp);
pnlTemp = new JPanel();
final JButton btnSearch = new JButton("Search");
pnlTemp.add(btnSearch);
final JButton btnStop = new JButton("Stop");
btnStop.setEnabled(false);
pnlTemp.add(btnStop);
pnl.add(pnlTemp);
pnlTemp = new JPanel();
txtSrchResults = new JTextArea(20, 30);
pnlTemp.add(new JScrollPane(txtSrchResults));
pnl.add(pnlTemp);
ActionListener al;
al = new ActionListener()
     {
        @Override
        public void actionPerformed(ActionEvent ae)
        {
           btnSearch.setEnabled(false);
           btnStop.setEnabled(true);
           String _startDir = txtStartDir.getText();
           if (_startDir.length() == 0)
           {
              _startDir = System.getProperty("user.dir");
              txtStartDir.setText(_startDir);
           }
           final String startDir = _startDir;
           String _srchText = txtSrchText.getText();
           if (_srchText.length() == 0)
```

```
{
   _srchText = " ";
   txtSrchText.setText(" ");
}
final String srchText = _srchText;
txtSrchResults.setText("");
Runnable r;
r = new Runnable()
    {
       @Override
       public void run()
       {
          stopped = false;
          if (!findAll(new File(startDir), srchText))
          {
             Runnable r;
             r = new Runnable()
                 {
                    @Override
                    public void run()
                    {
                       String msg = "not a directory";
                       JOptionPane.showMessageDialog(f, msg);
                    }
                 };
             EventQueue.invokeLater(r);
          }
          Runnable r;
          r = new Runnable()
              {
                 @Override
                 public void run()
                 {
                    txtSrchResults.append("DONE"+
                                          LINE_SEPARATOR);
                    btnSearch.setEnabled(true);
                    btnStop.setEnabled(false);
                 }
              };
          EventQueue.invokeLater(r);
```

```
                    }
                };
            new Thread(r).start();
         }
      };
   btnSearch.addActionListener(al);
   al = new ActionListener()
      {
         @Override
         public void actionPerformed(ActionEvent ae)
         {
            btnSearch.setEnabled(true);
            btnStop.setEnabled(false);
            stopped = true;
         }
      };
   btnStop.addActionListener(al);
   return pnl;
}
static boolean findAll(File file, String srchText)
{
   File[] files = file.listFiles();
   if (files == null)
      return false;
   for (int i = 0; i < files.length; i++)
      if (stopped)
         return true;
      else
      if (files[i].isDirectory())
         findAll(files[i], srchText);
      else
      if (find(files[i].getPath(), srchText))
      {
         result = files[i].getPath();
         Runnable r = new Runnable()
                     {
                        @Override
                        public void run()
                        {
                           txtSrchResults.append(result+LINE_SEPARATOR);
```

```
                        }
                    };
            EventQueue.invokeLater(r);
        }
    return true;
}
static boolean find(String filename, String srchText)
{
    try (BufferedReader br = new BufferedReader(new FileReader(filename)))
    {
        int ch;
        outer_loop:
        do
        {
            if ((ch = br.read()) == -1)
                return false;
            if (ch == srchText.charAt(0))
            {
                for (int i = 1; i < srchText.length(); i++)
                {
                    if ((ch = br.read()) == -1)
                        return false;
                    if (ch != srchText.charAt(i))
                        continue outer_loop;
                }
                return true;
            }
        }
        while (true);
    }
    catch (IOException ioe)
    {
        System.err.println("I/O error: "+ioe.getMessage());
    }
    return false;
}
public static void main(String[] args)
{
    Runnable r = new Runnable()
                 {
```

```
                    @Override
                    public void run()
                    {
                       f = new JFrame("FindAll");
                       f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                       f.setContentPane(createGUI());
                       f.pack();
                       f.setResizable(false);
                       f.setVisible(true);
                    }
                 };
      EventQueue.invokeLater(r);
   }
}
```

As well as addressing the exercise's requirements, this application includes a few other minor improvements. See if you can find them.

Chapter 9: Interacting with Networks and Databases

Chapter 9's exercises test your understanding of network APIs and JDBC:

1. Listings A-58 through A-60 present the `BJDealer`, `BJPlayer`, and `Card` classes.

Listing A-58. *Dealing cards to players*

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

import java.net.ServerSocket;
import java.net.Socket;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

class BJDealer
{
   final static int PORTNO = 10000;
   public static void main(String[] args)
```

```
   {
      System.out.println("Blackjack server starting...");
      try
      {
         ServerSocket ss = new ServerSocket(PORTNO);
         while (true)
            new ConnectionThread(ss.accept()).start();
      }
      catch (IOException ioe)
      {
         System.out.println("I/O error: "+ioe.getMessage());
      }
   }
}
class ConnectionThread extends Thread
{
   private Socket s;
   private ObjectInputStream ois;
   private ObjectOutputStream oos;
   private List<Card> deck, dhand, phand;
   // Index of top card on deck to retrieve. Assume that a retrieved card is
   // removed from the deck (although this doesn't actually happen) and that
   // topCard refers to the next card on the deck -- the card below the
   // previous top card.
   private int topCard;
   ConnectionThread(Socket s)
   {
      this.s = s;
      deck = Card.newDeck();
      dhand = new ArrayList<Card>();
      phand = new ArrayList<Card>();
   }
   void doDeal(ObjectOutputStream oos) throws IOException
   {
      // Shuffle the deck if the dealer had less than 20 cards in previous
      // round.
      if (dhand.size() < 20)
         Collections.shuffle(deck);
      // Begin dealing cards with the first card on the deck.
      topCard = 0;
```

```
// Reset player's hand to nothing.
phand.clear();
// Deal first two cards to player.
phand.add(deck.get(topCard++));
phand.add(deck.get(topCard++));
// Show these cards to player.
oos.writeObject(phand.get(0));
oos.writeObject(phand.get(1));
// Reset dealer's hand to nothing.
dhand.clear();
// Deal first two cards to dealer.
dhand.add(deck.get(topCard++));
dhand.add(deck.get(topCard++));
// Show dealer's first card to player.
oos.writeObject(dhand.get(0));
// Evaluate player's hand for Blackjack (21). ACE considered to be worth
// 11.
int pvalue1 = phand.get(0).getValue();
if (pvalue1 == Card.Rank.ACE.getValue())
   pvalue1 = 11;
else
if (pvalue1 == Card.Rank.JACK.getValue() ||
    pvalue1 == Card.Rank.QUEEN.getValue() ||
    pvalue1 == Card.Rank.KING.getValue())
   pvalue1 = 10;
int pvalue2 =  phand.get(1).getValue();
if (pvalue2 == Card.Rank.ACE.getValue())
   pvalue2 = 11;
else
if (pvalue2 == Card.Rank.JACK.getValue() ||
    pvalue2 == Card.Rank.QUEEN.getValue() ||
    pvalue2 == Card.Rank.KING.getValue())
   pvalue2 = 10;
int psum = pvalue1+pvalue2;
// Evaluate dealer's hand for Blackjack (21). ACE considered to be worth
// 11.
int dvalue1 = dhand.get(0).getValue();
if (dvalue1 == Card.Rank.ACE.getValue())
   dvalue1 = 11;
else
```

```
        if (dvalue1 == Card.Rank.JACK.getValue() ||
            dvalue1 == Card.Rank.QUEEN.getValue() ||
            dvalue1 == Card.Rank.KING.getValue())
           dvalue1 = 10;
        int dvalue2 = dhand.get(1).getValue();
        if (dvalue2 == Card.Rank.ACE.getValue())
           dvalue2 = 11;
        else
        if (dvalue2 == Card.Rank.JACK.getValue() ||
            dvalue2 == Card.Rank.QUEEN.getValue() ||
            dvalue2 == Card.Rank.KING.getValue())
           dvalue2 = 10;
        int dsum = dvalue1+dvalue2;
        // If dealer has Blackjack and player has Blackjack...
        if (dsum == 21 && psum == 21)
        {
           // Indicate a push -- no one wins.
           oos.writeObject("Status: Dealer and player tie.");
           // Show dealer's second card to player.
           oos.writeObject(dhand.get(1));
        }
        else // If dealer has Blackjack and player does not have Blackjack...
        if (dsum == 21 && psum != 21)
        {
           // Indicate dealer wins. Blackjack!
           oos.writeObject("Status: Blackjack! Dealer wins.");
           // Show dealer's second card to player.
           oos.writeObject(dhand.get(1));
        }
        else // If player has Blackjack and dealer does not have Blackjack...
        if (psum == 21 && dsum != 21)
        {
           // Indicate player wins. Blackjack!
           oos.writeObject("Status: Blackjack! Player wins.");
           // Show dealer's second card to player.
           oos.writeObject(dhand.get(1));
        }
        else
           oos.writeObject(""); // Indicate not-end-of-round.
     }
```

```
void doHit(ObjectOutputStream oos) throws IOException
{
   // Deal card to player.
   phand.add(deck.get(topCard++));
   // Show this card to player.
   oos.writeObject(phand.get(phand.size()-1));
   // Total player's cards in optimal manner. ACE considered 1 or 11.
   int psum = 0;
   Iterator<Card> iter = phand.iterator();
   while (iter.hasNext())
   {
      int value = iter.next().getValue();
      if (value == Card.Rank.ACE.getValue())
         value = 11;
      else
      if (value == Card.Rank.JACK.getValue() ||
          value == Card.Rank.QUEEN.getValue() ||
          value == Card.Rank.KING.getValue())
         value = 10;
      psum += value;
   }
   if (psum > 21)
   {
      psum = 0;
      iter = phand.iterator();
      while (iter.hasNext())
      {
         int value = iter.next().getValue();
         if (value == Card.Rank.JACK.getValue() ||
             value == Card.Rank.QUEEN.getValue() ||
             value == Card.Rank.KING.getValue())
            value = 10;
         psum += value;
      }
   }
   // Dealer automatically wins when player goes over 21.
   if (psum > 21)
   {
      // Indicate dealer wins.
      oos.writeObject("Status: Player > 21. Dealer wins.");
```

```
            // Show dealer's second card to player.
            oos.writeObject(dhand.get(1));
        }
        else
            oos.writeObject(""); // Indicate not-end-of-round.
    }
    void doStand(ObjectOutputStream oos) throws IOException
    {
        // Show dealer's second card to player.
        oos.writeObject(dhand.get(1));
        // The dealer keeps taking hits while sum < 17. When sum >= 17, dealer
        // stands.
        while (true)
        {
            // Sum dealer's cards where ACE is always considered 11.
            int dsum = 0;
            Iterator<Card> iter = dhand.iterator();
            while (iter.hasNext())
            {
                int value = iter.next().getValue();
                if (value == Card.Rank.ACE.getValue())
                    value = 11;
                else
                if (value == Card.Rank.JACK.getValue() ||
                     value == Card.Rank.QUEEN.getValue() ||
                     value == Card.Rank.KING.getValue())
                    value = 10;
                dsum += value;
            }
            if (dsum < 17)
            {
                // Dealer takes a hit. Deal another card to dealer.
                dhand.add(deck.get(topCard++));
                continue;
            }
            break;
        }
        // Evaluate player's hand. ACE is either 1 or 11 -- whichever is optimal.
        int psum = 0;
        Iterator<Card> iter = phand.iterator();
```

```
while (iter.hasNext())
{
   int value = iter.next().getValue();
   if (value == Card.Rank.ACE.getValue())
      value = 11;
   else
   if (value == Card.Rank.JACK.getValue() ||
       value == Card.Rank.QUEEN.getValue() ||
       value == Card.Rank.KING.getValue())
      value = 10;
   psum += value;
}
if (psum > 21)
{
    psum = 0;
    iter = phand.iterator();
    while (iter.hasNext())
    {
       int value = iter.next().getValue();
       if (value == Card.Rank.JACK.getValue() ||
           value == Card.Rank.QUEEN.getValue() ||
           value == Card.Rank.KING.getValue())
          value = 10;
       psum += value;
    }
}
// Evaluate dealer's hand. ACE is either 1 or 11 -- whichever is
// optimal.
int dsum = 0;
iter = dhand.iterator();
while (iter.hasNext())
{
   int value = iter.next().getValue();
   if (value == Card.Rank.ACE.getValue())
      value = 11;
   else
   if (value == Card.Rank.JACK.getValue() ||
       value == Card.Rank.QUEEN.getValue() ||
       value == Card.Rank.KING.getValue())
      value = 10;
```

```
            dsum += value;
        }
        if (dsum > 21)
        {
            dsum = 0;
            iter = dhand.iterator();
            while (iter.hasNext())
            {
               int value = iter.next().getValue();
               if (value == Card.Rank.JACK.getValue() ||
                   value == Card.Rank.QUEEN.getValue() ||
                   value == Card.Rank.KING.getValue())
                  value = 10;
               dsum += value;
            }
        }
        // Figure out whom (if any) won this round.
        if (psum > 21 && dsum > 21)
           oos.writeObject("Status: Dealer > 21 and player > 21.");
        else
        if (psum > 21)
           oos.writeObject("Status: Player > 21. Dealer wins.");
        else
        if (dsum > 21)
           oos.writeObject("Status: Dealer > 21. Player wins.");
        else
        if (dsum > psum)
           oos.writeObject("Status: Dealer wins.");
        else
        if (dsum < psum)
           oos.writeObject("Status: Player wins.");
        else
           oos.writeObject("Status: Player and dealer tie.");
        // Inform client of all cards in dealer's hand. This lets the player
        // see that the server isn't cheating.
        oos.writeInt(dhand.size());
        for (int i = 0; i < dhand.size(); i++)
           oos.writeObject(dhand.get(i));
    }
    // Allow connection thread to run until client sends a Deal command -- or
```

```
   // the client automatically terminates the connection.
   @Override
   public void run()
   {
      try (ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
           ObjectInputStream ois = new ObjectInputStream(s.getInputStream()))
      {
         oos.flush();
         do
         {
            String cmd = (String) ois.readObject();
            switch (cmd)
            {
               case "Deal" : doDeal(oos); break;
               case "Hit"  : doHit(oos); break;
               case "Stand": doStand(oos); break;
               case "Quit" : return;
            }
         }
         while (true);
      }
      catch (ClassNotFoundException cnfe)
      {
         cnfe.printStackTrace();
      }
      catch (IOException ioe)
      {
         ioe.printStackTrace();
      }
   }
}
```

Listing A-59. *Playing Blackjack with the dealer*

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.FlowLayout;
import java.awt.FontMetrics;
import java.awt.Graphics;
```

```
import java.awt.GridLayout;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

import java.net.Socket;

import java.util.ArrayList;
import java.util.List;

import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import javax.swing.border.BevelBorder;

public class BJPlayer extends JFrame
{
   final static int PORTNO = 10000;
   final static String HOST = "localhost";
   static Socket s;
   static ObjectInputStream ois;
   static ObjectOutputStream oos;
   List<Card> dhand, phand;
   // Cards are displayed on the following panel. Player cards are displayed
   // on top and dealer cards are displayed half-way down.
   CardPanel pnlCards;
   JButton btnDeal, btnExit, btnHit, btnStand;
   JLabel lblStatus;
   BJPlayer(String title)
   {
```

```
super(title);
setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
addWindowListener(new WindowAdapter()
                  {
                     @Override
                     public void windowClosing(WindowEvent we)
                     {
                        try
                        {
                           oos.writeObject("Quit");
                           ois.close();
                           oos.close();
                           s.close();
                         }
                         catch (IOException ioe)
                         {
                         }
                         dispose();
                      }
                   });
dhand = new ArrayList<Card>();
phand = new ArrayList<Card>();
pnlCards = new CardPanel(dhand, phand);
pnlCards.setBorder(new BevelBorder(BevelBorder.LOWERED));
getContentPane().add(pnlCards, BorderLayout.NORTH);
btnDeal = new JButton("Deal");
ActionListener al;
al = new ActionListener()
     {
        @Override
        public void actionPerformed(ActionEvent ae)
        {
           dhand.clear();
           phand.clear();
           pnlCards.repaint();
           try
           {
              oos.writeObject("Deal");
              phand.add((Card) ois.readObject());
              phand.add((Card) ois.readObject());
```

```
                    dhand.add((Card) ois.readObject());
                    pnlCards.repaint();
                    // Retrieve status message or not-end-of-round message,
                    // and update GUI status.
                    String status = (String) ois.readObject();
                    lblStatus.setText(status);
                    // If not-end-of-round, disable Deal button and make sure
                    // Hit and Stand buttons are enabled. Otherwise, do the
                    // reverse.
                    if (status.equals(""))
                    {
                       btnDeal.setEnabled(false);
                       btnHit.setEnabled(true);
                       btnStand.setEnabled(true);
                    }
                    else
                    {
                       // Retrieve dealer's second card.
                       dhand.add((Card) ois.readObject());
                       pnlCards.repaint();
                       btnDeal.setEnabled(true);
                       btnHit.setEnabled(false);
                       btnStand.setEnabled(false);
                    }
                 }
                 catch (ClassNotFoundException cnfe)
                 {
                    showMessage(cnfe.toString());
                 }
                 catch (IOException ioe)
                 {
                    showMessage(ioe.toString());
                 }
              }
           };
        btnDeal.addActionListener(al);
        btnExit = new JButton("Exit");
        al = new ActionListener()
           {
              @Override
```

```
        public void actionPerformed(ActionEvent ae)
        {
           try
           {
              oos.writeObject("Quit");
           }
           catch (IOException ioe)
           {
              showMessage(ioe.toString());
           }
           dispose();
        }
     };
btnExit.addActionListener(al);
btnHit = new JButton("Hit");
btnHit.setEnabled(false);
al = new ActionListener()
     {
        @Override
        public void actionPerformed(ActionEvent ae)
        {
           try
           {
              oos.writeObject("Hit");
              // Retrieve player's next-dealt card.
              phand.add((Card) ois.readObject());
              pnlCards.repaint();
              // Retrieve status message or not-end-of-round message,
              // and update GUI status.
              String status = (String) ois.readObject();
              lblStatus.setText(status);
              // If not-end-of-round, disable Deal button and make sure
              // Hit and Stand buttons are enabled. Otherwise, do the
              // reverse.
              if (status.equals(""))
              {
                 btnDeal.setEnabled(false);
                 btnHit.setEnabled(true);
                 btnStand.setEnabled(true);
              }
```

```
                else
                {
                   // Retrieve dealer's second card.
                   dhand.add((Card) ois.readObject());
                   pnlCards.repaint();
                   btnDeal.setEnabled(true);
                   btnHit.setEnabled(false);
                   btnStand.setEnabled(false);
                }
             }
             catch (ClassNotFoundException cnfe)
             {
                showMessage(cnfe.toString());
             }
             catch (IOException ioe)
             {
                showMessage(ioe.toString());
             }
          }
       };
btnHit.addActionListener(al);
btnStand = new JButton("Stand");
btnStand.setEnabled(false);
al = new ActionListener()
     {
        @Override
        public void actionPerformed(ActionEvent ae)
        {
           try
           {
              oos.writeObject("Stand");
              // Retrieve dealer's second card.
              dhand.add((Card) ois.readObject());
              pnlCards.repaint();
              // Retrieve status message and update GUI status.
              // Round is over at this point.
              String status = (String) ois.readObject();
              lblStatus.setText(status);
              // Retrieve all of the dealer's cards, to ensure that the
              // dealer isn't cheating.
```

```
                int count = ois.readInt();
                dhand.clear();
                for (int i = 0; i < count; i++)
                   dhand.add((Card) ois.readObject());
                pnlCards.repaint();
                // End of round. Make sure Deal button is enabled and the
                // Hit and Stand buttons are disabled.
                btnDeal.setEnabled(true);
                btnHit.setEnabled(false);
                btnStand.setEnabled(false);
             }
             catch (ClassNotFoundException cnfe)
             {
                showMessage(cnfe.toString());
             }
             catch (IOException ioe)
             {
                showMessage(ioe.toString());
             }
          }
       };
btnStand.addActionListener(al);
lblStatus = new JLabel("Status:");
JPanel pnlLeft = new JPanel();
pnlLeft.setLayout(new FlowLayout(FlowLayout.LEFT));
pnlLeft.add(lblStatus);
JPanel pnlRight = new JPanel();
pnlRight.setLayout(new FlowLayout(FlowLayout.RIGHT));
pnlRight.add(btnDeal);
pnlRight.add(btnHit);
pnlRight.add(btnStand);
pnlRight.add(btnExit);
JPanel pnlControl = new JPanel();
pnlControl.setLayout(new GridLayout(1, 2));
pnlControl.setBorder(new BevelBorder(BevelBorder.LOWERED));
pnlControl.add(pnlLeft);
pnlControl.add(pnlRight);
getContentPane().add(pnlControl, BorderLayout.SOUTH);
pack();
setResizable(false);
```

```
   setVisible(true);
}
public static void main(String[] args)
{
   Runnable r = new Runnable()
                {
                   @Override
                   public void run()
                   {
                      try
                      {
                         s = new Socket(HOST, PORTNO);
                         oos = new ObjectOutputStream(s.getOutputStream());
                         oos.flush();
                         ois = new ObjectInputStream(s.getInputStream());
                         new BJPlayer("Blackjack Player");
                      }
                      catch (IOException ioe)
                      {
                         showMessage(ioe.toString());
                         try
                         {
                            if (ois != null)
                               ois.close();
                            if (oos != null)
                               oos.close();
                            if (s != null)
                               s.close();
                         }
                         catch (IOException ioe2)
                         {
                         }
                         System.exit(0);
                      }
                   }
                };
   EventQueue.invokeLater(r);
}
static void showMessage(String message)
{
```

```
      JOptionPane.showMessageDialog(null, message, "Blackjack Player",
                                    JOptionPane.INFORMATION_MESSAGE);
   }
}
class CardPanel extends JComponent
{
   private static final int WIDTH = 500;
   private static final int HEIGHT = 500;
   final static int CARD_HEIGHT = 75;
   final static int CARD_WIDTH = 60;
   final static int CARD_SEP = 10;
   private List<Card> dhand, phand;
   CardPanel(List<Card> dhand, List<Card> phand)
   {
      this.dhand = dhand;
      this.phand = phand;
   }
   public Dimension getPreferredSize()
   {
      return new Dimension(WIDTH, HEIGHT);
   }
   @Override
   public void paint(Graphics g)
   {
      g.setColor(Color.WHITE);
      g.fillRect(0, 0, getWidth(), getHeight());
      g.setColor(Color.BLACK);
      // Draw player's cards.
      int col = 5;
      int row = 5;
      for (int i = 0; i < phand.size(); i++)
      {
         drawCard(phand.get(i), g, col, row);
         col += CARD_WIDTH+CARD_SEP;
         if (col+CARD_WIDTH > WIDTH)
         {
            col = 0;
            row += CARD_HEIGHT/2;
         }
      }
```

```
      // Draw dealer's cards.
      col = 5;
      row = HEIGHT/2;
      for (int i = 0; i < dhand.size(); i++)
      {
         drawCard(dhand.get(i), g, col, row);
         col += CARD_WIDTH+CARD_SEP;
         if (col+CARD_WIDTH > WIDTH)
         {
            col = 0;
            row += CARD_HEIGHT/2;
         }
      }
   }
   void drawCard(Card card, Graphics g, int x, int y)
   {
      if (card.getSuit() == Card.Suit.DIAMONDS ||
          card.getSuit() == Card.Suit.HEARTS)
         g.setColor(Color.RED);
      else
         g.setColor(Color.BLACK);
      g.drawRoundRect(x, y, CARD_WIDTH, CARD_HEIGHT, 8, 8);
      FontMetrics fm = g.getFontMetrics();
      int height = fm.getHeight();
      String v;
      switch (card.getRank())
      {
         case ACE  : v = "A"; break;
         case JACK : v = "J"; break;
         case QUEEN: v = "Q"; break;
         case KING : v = "K"; break;
         default   : v = ""+card.getRank().getValue();
      }
      int width = fm.stringWidth(v);
      g.drawString(v, x+5, y+height);
      g.drawString(v, x+CARD_WIDTH-width-5, y+CARD_HEIGHT-5);
      String symbol = "";
      switch (card.getSuit())
      {
         case CLUBS   : symbol = "C"; break;
```

```
            case DIAMONDS: symbol = "D"; break;
            case HEARTS  : symbol = "H"; break;
            case SPADES  : symbol = "S";
         }
         width = fm.stringWidth(symbol);
         g.drawString(symbol, x+(CARD_WIDTH-width)/2, y+CARD_HEIGHT/2+height/4);
      }
   }
```

Listing A-60. *Pick a card, any card*

```
import java.io.Serializable;

import java.util.ArrayList;
import java.util.List;

class Card implements Serializable
{
   enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES }
   enum Rank { ACE, DEUCE, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN,
               JACK, QUEEN, KING;
               int getValue()
               {
                  return ordinal()+1;
               }
             }
   private Suit suit;
   private Rank rank;
   private static final List<Card> initialDeck = new ArrayList<Card>();
   Card(Suit suit, Rank rank)
   {
      this.suit = suit;
      this.rank = rank;
   }
   Rank getRank()
   {
      return rank;
   }
   Suit getSuit()
   {
      return suit;
```

```
   }
   int getValue()
   {
      return rank.ordinal()+1;
   }
   static
   {
      for (Suit suit: Suit.values())
         for (Rank rank: Rank.values())
            initialDeck.add(new Card(suit, rank));
   }
   static List<Card> newDeck() // Return a new unshuffled deck.
   {
      // Copy initial deck to new deck.
      List<Card> deck = new ArrayList<Card>(initialDeck);
      return deck;
   }
}
```

2. Listing A-61 presents the `Planets` class with support for moon stats and notes.

Listing A-61. *A refactored* `Planets` *class also displaying moon counts and notes centered below the image*

```
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.Graphics;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

import java.sql.Blob;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.Statement;
```

```
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.ListSelectionModel;

import javax.swing.border.Border;

import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableModel;

class Planets
{
   static String[] names, notes;
   static double[] diameters, masses, distances;
   static int[] moons;
   static ImageIcon[] iiPhotos;
   static JPanel createGUI()
   {
      JPanel pnlGUI = new JPanel();
      pnlGUI.setLayout(new BorderLayout());
      final SwingCanvas sc = new SwingCanvas(iiPhotos[0]);
      pnlGUI.add(sc, BorderLayout.NORTH);
      final JLabel lblNotes = new JLabel(notes[0], JLabel.CENTER);
      lblNotes.setBorder(BorderFactory.createEmptyBorder(15, 15, 15, 15));
      pnlGUI.add(lblNotes, BorderLayout.CENTER);
      TableModel model = new AbstractTableModel()
      {
         @Override
         public int getColumnCount()
         {
            return 5;
         }
```

```
   @Override
   public String getColumnName(int column)
   {
      switch (column)
      {
         case 0: return "Name";
         case 1: return "Diameter (KM)";
         case 2: return "Mass (KG)";
         case 3: return "Distance (AU)";
         default: return "Moons";
      }
   }
   @Override
   public int getRowCount()
   {
      return names.length;
   }
   @Override
   public Object getValueAt(int row, int col)
   {
      switch (col)
      {
         case 0: return Character.toUpperCase(names[row].charAt(0))+
                 names[row].substring(1);
         case 1: return diameters[row];
         case 2: return masses[row];
         case 3: return distances[row];
         default: return moons[row];
      }
   }
};
final JTable table = new JTable(model);
table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
table.setRowSelectionInterval(0, 0);
ListSelectionListener lsl;
lsl = new ListSelectionListener()
      {
         @Override
         public void valueChanged(ListSelectionEvent lse)
         {
```

```
                sc.setPhoto(iiPhotos[table.getSelectedRow()]);
                lblNotes.setText(notes[table.getSelectedRow()]);
            }
        };
    table.getSelectionModel().addListSelectionListener(lsl);
    // The following scrollpane is need to ensure that table headers are
    // displayed. They aren't displayed when the table is added directly to
    // the content pane.
    JScrollPane sp = new JScrollPane(table);
    // Make sure that the table displays exactly 8 rows.
    Dimension size = sp.getViewport().getPreferredSize();
    size.width *= 1.7;
    size.height = 8*table.getRowHeight();
    table.setPreferredScrollableViewportSize(size);
    pnlGUI.add(sp, BorderLayout.SOUTH);
    return pnlGUI;
}
static void initDB()
{
    String[] planets = { "mercury", "venus", "earth", "mars", "jupiter",
                         "saturn", "uranus", "neptune" };
    double[] diameters = { 4880, 12103.6, 12756.3, 6794, 142984, 120536,
                           51118, 49532 };
    double[] masses = { 3.3e23, 4.869e24, 5.972e24, 6.4219e23, 1.9e27,
                        5.68e26, 8.683e25, 1.0247e26 };
    double[] distances = { 0.38, 0.72, 1, 1.52, 5.2, 9.54, 19.218, 30.06 };
    int[] moons = { 0, 0, 1, 2, 64, 200, 27, 13 };
    String[] notes =
    {
        "closest planet to the sun",
        "covered with opaque layer of highly-reflective sulfuric clouds",
        "supports life",
        "tallest volcanoes in solar system",
        "gas giant, largest planet in solar system",
        "gas giant, most prominent rings",
        "gas giant, coldest planetary atmosphere of solar system planets",
        "gas giant, farthest planet from the sun"
    };
    String url = "jdbc:derby:planets;create=true";
    try (Connection con = DriverManager.getConnection(url))
```

```
{
   try (Statement stmt = con.createStatement())
   {
      String sql = "CREATE TABLE PLANETS(NAME VARCHAR(30),"+
                                        "DIAMETER REAL,"+
                                        "MASS REAL,"+
                                        "DISTANCE REAL,"+
                                        "PHOTO BLOB,"+
                                        "MOONS INTEGER,"+
                                        "NOTES VARCHAR(100))";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO PLANETS VALUES(?, ?, ?, ?, ?, ?, ?)";
      try (PreparedStatement pstmt = con.prepareStatement(sql))
      {
         for (int i = 0; i < planets.length; i++)
         {
            pstmt.setString(1, planets[i]);
            pstmt.setDouble(2, diameters[i]);
            pstmt.setDouble(3, masses[i]);
            pstmt.setDouble(4, distances[i]);
            Blob blob = con.createBlob();
            try (ObjectOutputStream oos =
                   new ObjectOutputStream(blob.setBinaryStream(1)))
            {
               ImageIcon photo = new ImageIcon(planets[i]+".jpg");
               oos.writeObject(photo);
            }
            catch (IOException ioe)
            {
               System.err.println("unable to write "+planets[i]+".jpg");
            }
            pstmt.setBlob(5, blob);
            pstmt.setInt(6, moons[i]);
            pstmt.setString(7, notes[i]);
            pstmt.executeUpdate();
            blob.free();
         }
      }
   }
}
```

```
   catch (SQLException sqlex)
   {
      while (sqlex != null)
      {
         System.err.println("SQL error : "+sqlex.getMessage());
         System.err.println("SQL state : "+sqlex.getSQLState());
         System.err.println("Error code: "+sqlex.getErrorCode());
         System.err.println("Cause: "+sqlex.getCause());
         sqlex = sqlex.getNextException();
      }
   }
}
static boolean loadDB()
{
   String url = "jdbc:derby:planets;create=false";
   try (Connection con = DriverManager.getConnection(url))
   {
      try (Statement stmt = con.createStatement())
      {
         ResultSet rs = stmt.executeQuery("SELECT COUNT(*) FROM PLANETS");
         rs.next();
         int size = rs.getInt(1);
         names = new String[size];
         diameters = new double[size];
         masses = new double[size];
         distances = new double[size];
         iiPhotos = new ImageIcon[size];
         moons = new int[size];
         notes = new String[size];
         rs = stmt.executeQuery("SELECT * FROM PLANETS");
         for (int i = 0; i < size; i++)
         {
            rs.next();
            names[i] = rs.getString(1);
            diameters[i] = rs.getDouble(2);
            masses[i] = rs.getDouble(3);
            distances[i] = rs.getDouble(4);
            Blob blob = rs.getBlob(5);
            try (ObjectInputStream ois =
                   new ObjectInputStream(blob.getBinaryStream()))
```

```
            {
               iiPhotos[i] = (ImageIcon) ois.readObject();
            }
            catch (ClassNotFoundException|IOException cnfioe)
            {
               System.err.println("unable to read "+names[i]+".jpg");
            }
            blob.free();
            moons[i] = rs.getInt(6);
            notes[i] = rs.getString(7);
         }
         return true;
      }
   }
   catch (SQLException sqlex)
   {
      while (sqlex != null)
      {
         System.err.println("SQL error : "+sqlex.getMessage());
         System.err.println("SQL state : "+sqlex.getSQLState());
         System.err.println("Error code: "+sqlex.getErrorCode());
         System.err.println("Cause: "+sqlex.getCause());
         sqlex = sqlex.getNextException();
      }
      return false;
   }
}
public static void main(String[] args)
{
   if (args.length == 1 && args[0].equalsIgnoreCase("INITDB"))
   {
      initDB();
      return;
   }
   Runnable r = new Runnable()
               {
                  @Override
                  public void run()
                  {
                     if (!loadDB())
```

```
                        return;
                    JFrame f = new JFrame("Planets");
                    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                    Border border;
                    border = BorderFactory.createEmptyBorder(5, 5, 5, 5);
                    f.getRootPane().setBorder(border);
                    f.setContentPane(createGUI());
                    f.pack();
                    f.setResizable(false);
                    f.setVisible(true);
                }
            };
        EventQueue.invokeLater(r);
    }
}
class SwingCanvas extends JComponent
{
    private ImageIcon iiPhoto;
    private Dimension d;
    SwingCanvas(ImageIcon iiPhoto)
    {
        d = new Dimension(504, 403); // Create object here to avoid unnecessary
                                     // object creation should getPreferredSize()
                                     // be called more than once.
        this.iiPhoto = iiPhoto;
    }
    @Override
    public Dimension getPreferredSize()
    {
        return d;
    }
    @Override
    public void paint(Graphics g)
    {
        if (iiPhoto != null)
            g.drawImage(iiPhoto.getImage(), (getParent().getWidth()-504)/2, 0,
                        null);
    }
    void setPhoto(ImageIcon iiPhoto)
    {
```

```
      this.iiPhoto = iiPhoto;
      repaint();
   }
}
```

Chapter 10: Parsing, Creating, and Transforming XML Documents

Chapter 10's exercises test your understanding of XML document creation and the SAX, DOM, StAX, XPath, and XSLT APIs:

1. Listing A-62 presents the `books.xml` document file.

Listing A-62. *A document of books*

```
<?xml version="1.0"?>
<books>
   <book isbn="0201548550" pubyear="1992">
      <title>
         Advanced C++
      </title>
      <author>
         James O. Coplien
      </author>
      <publisher>
         Addison Wesley
      </publisher>
   </book>
   <book isbn="9781430210450" pubyear="2008">
      <title>
         Beginning Groovy and Grails
      </title>
      <author>
         Christopher M. Judd
      </author>
      <author>
         Joseph Faisal Nusairat
      </author>
      <author>
         James Shingler
      </author>
```

```
      <publisher>
         Apress
      </publisher>
   </book>
   <book isbn="0201310058" pubyear="2001">
      <title>
         Effective Java
      </title>
      <author>
         Joshua Bloch
      </author>
      <publisher>
         Addison Wesley
      </publisher>
   </book>
</books>
```

2. Listing A-63 presents the `books.xml` document file with an internal DTD.

Listing A-63. *A DTD-enabled document of books*

```
<?xml version="1.0"?>
<!DOCTYPE books [
   <!ELEMENT books (book+)>
   <!ELEMENT book (title, author+, publisher)>
   <!ELEMENT title (#PCDATA)>
   <!ELEMENT author (#PCDATA)>
   <!ELEMENT publisher (#PCDATA)>
   <!ATTLIST book isbn CDATA #REQUIRED>
   <!ATTLIST book pubyear CDATA #REQUIRED>
]>
<books>
   <book isbn="0201548550" pubyear="1992">
      <title>
         Advanced C++
      </title>
      <author>
         James O. Coplien
      </author>
      <publisher>
         Addison Wesley
```

```
      </publisher>
   </book>
   <book isbn="9781430210450" pubyear="2008">
      <title>
         Beginning Groovy and Grails
      </title>
      <author>
         Christopher M. Judd
      </author>
      <author>
         Joseph Faisal Nusairat
      </author>
      <author>
         James Shingler
      </author>
      <publisher>
         Apress
      </publisher>
   </book>
   <book isbn="0201310058" pubyear="2001">
      <title>
         Effective Java
      </title>
      <author>
         Joshua Bloch
      </author>
      <publisher>
         Addison Wesley
      </publisher>
   </book>
</books>
```

3. Listings A-64 and A-65 present the `SAXSearch` and `Handler` classes.

Listing A-64. *A SAX driver class for searching* `books.xml` *for a specific publisher's books*

```
import java.io.FileReader;
import java.io.IOException;

import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
```

```
import org.xml.sax.XMLReader;

import org.xml.sax.helpers.XMLReaderFactory;

class SAXSearch
{
   public static void main(String[] args)
   {
      if (args.length != 1)
      {
         System.err.println("usage: java SAXSearch publisher");
         return;
      }
      try
      {
         XMLReader xmlr = XMLReaderFactory.createXMLReader();
         Handler handler = new Handler(args[0]);
         xmlr.setContentHandler(handler);
         xmlr.setErrorHandler(handler);
         xmlr.setProperty("http://xml.org/sax/properties/lexical-handler", handler);
         xmlr.parse(new InputSource(new FileReader("books.xml")));
      }
      catch (IOException ioe)
      {
         System.err.println("IOE: "+ioe);
      }
      catch (SAXException saxe)
      {
         System.err.println("SAXE: "+saxe);
      }
   }
}
```

Listing A-65. *A SAX callback class whose methods are called by the SAX parser*

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXParseException;

import org.xml.sax.ext.DefaultHandler2;

class Handler extends DefaultHandler2
```

```
{
   private boolean isPublisher, isTitle;
   private String isbn, publisher, pubYear, title, srchText;
   public Handler(String srchText)
   {
      this.srchText = srchText;
   }
   @Override
   public void characters(char[] ch, int start, int length)
   {
      if (isTitle)
      {
         title = new String(ch, start, length).trim();
         isTitle = false;
      }
      else
      if (isPublisher)
      {
         publisher = new String(ch, start, length).trim();
         isPublisher = false;
      }
   }
   @Override
   public void endElement(String uri, String localName, String qName)
   {
      if (!localName.equals("book"))
         return;
      if (!srchText.equals(publisher))
         return;
      System.out.println("title = "+title+", isbn = "+isbn);
   }
   @Override
   public void error(SAXParseException saxpe)
   {
      System.out.println("error() "+saxpe);
   }
   @Override
   public void fatalError(SAXParseException saxpe)
   {
      System.out.println("fatalError() "+saxpe);
```

```
    }
    @Override
    public void startElement(String uri, String localName, String qName,
                             Attributes attributes)
    {
       if (localName.equals("title"))
       {
          isTitle = true;
          return;
       }
       else
       if (localName.equals("publisher"))
       {
          isPublisher = true;
          return;
       }
       if (!localName.equals("book"))
          return;
       for (int i = 0; i < attributes.getLength(); i++)
          if (attributes.getLocalName(i).equals("isbn"))
             isbn = attributes.getValue(i);
          else
          if (attributes.getLocalName(i).equals("pubyear"))
             pubYear = attributes.getValue(i);
    }
    @Override
    public void warning(SAXParseException saxpe)
    {
       System.out.println("warning() "+saxpe);
    }
}
```

4. Listing A-66 presents the `DOMSearch` class.

Listing A-66. *The DOM equivalent of* `SAXSearch` *and* `Handler`

```
import java.io.IOException;

import java.util.ArrayList;
import java.util.List;
```

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

class DOMSearch
{
   public static void main(String[] args)
   {
      if (args.length != 1)
      {
         System.err.println("usage: java DOMSearch publisher");
         return;
      }
      try
      {
         DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
         DocumentBuilder db = dbf.newDocumentBuilder();
         Document doc = db.parse("books.xml");
         class BookItem
         {
            String title;
            String isbn;
         }
         List<BookItem> bookItems = new ArrayList<BookItem>();
         NodeList books = doc.getElementsByTagName("book");
         for (int i = 0; i < books.getLength(); i++)
         {
            Element book = (Element) books.item(i);
            NodeList children = book.getChildNodes();
            String title = "";
            for (int j = 0; j < children.getLength(); j++)
```

```
      {
         Node child = children.item(j);
         if (child.getNodeType() == Node.ELEMENT_NODE)
         {
            if (child.getNodeName().equals("title"))
               title = child.getFirstChild().getNodeValue().trim();
            else
            if (child.getNodeName().equals("publisher"))
            {
               // Compare publisher name argument (args[0]) with text of
               // publisher's child text node. The trim() method call
               // removes whitespace that would interfere with the
               // comparison.
               if (args[0].equals(child.getFirstChild().getNodeValue().
                                  trim()))
               {
                  BookItem bookItem = new BookItem();
                  bookItem.title = title;
                  NamedNodeMap nnm = book.getAttributes();
                  Node isbn = nnm.getNamedItem("isbn");
                  bookItem.isbn = isbn.getNodeValue();
                  bookItems.add(bookItem);
                  break;
               }
            }
         }
      }
   }
   for (BookItem bookItem: bookItems)
      System.out.println("title = "+bookItem.title+", isbn = "+
                         bookItem.isbn);
}
catch (IOException ioe)
{
   System.err.println("IOE: "+ioe);
}
catch (SAXException saxe)
{
   System.err.println("SAXE: "+saxe);
}
```

```
      catch (FactoryConfigurationError fce)
      {
         System.err.println("FCE: "+fce);
      }
      catch (ParserConfigurationException pce)
      {
         System.err.println("PCE: "+pce);
      }
   }
}
```

5. Listing A-67 presents the `ParseXMLDOC` class.

Listing A-67. *A StAX stream-based parser for parsing an XML document*

```
import java.io.FileReader;
import java.io.IOException;

import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;

class ParseXMLDoc
{
   public static void main(String[] args)
   {
      if (args.length != 1)
      {
         System.err.println("usage: java ParseXMLDoc pathname");
         return;
      }
      XMLInputFactory xmlif = XMLInputFactory.newFactory();
      XMLStreamReader xmlsr = null;
      try (FileReader fr = new FileReader(args[0]))
      {
         xmlsr = xmlif.createXMLStreamReader(fr);
         int item = xmlsr.getEventType();
         if (item != XMLStreamReader.START_DOCUMENT)
         {
            System.err.println("START_DOCUMENT expected");
```

```
    return;
}
while ((item = xmlsr.next()) != XMLStreamReader.END_DOCUMENT)
    switch (item)
    {
        case XMLStreamReader.ATTRIBUTE:
            System.out.println("ATTRIBUTE");
            break;
        case XMLStreamReader.CDATA:
            System.out.println("CDATA");
            break;
        case XMLStreamReader.CHARACTERS:
            System.out.println("CHARACTERS");
            break;
        case XMLStreamReader.COMMENT:
            System.out.println("COMMENT");
            break;
        case XMLStreamReader.DTD:
            System.out.println("DTD");
            break;
        case XMLStreamReader.END_ELEMENT:
            System.out.println("END_ELEMENT");
            break;
        case XMLStreamReader.ENTITY_DECLARATION:
            System.out.println("ENTITY_DECLARATION");
            break;
        case XMLStreamReader.ENTITY_REFERENCE:
            System.out.println("ENTITY_REFERENCE");
            break;
        case XMLStreamReader.NAMESPACE:
            System.out.println("NAMESPACE");
            break;
        case XMLStreamReader.NOTATION_DECLARATION:
            System.out.println("NOTATION_DECLARATION");
            break;
        case XMLStreamReader.PROCESSING_INSTRUCTION:
            System.out.println("PROCESSING_INSTRUCTION");
            break;
        case XMLStreamReader.SPACE:
            System.out.println("SPACE");
```

```
                break;
             case XMLStreamReader.START_ELEMENT:
                System.out.println("START_ELEMENT");
                System.out.println("Name = "+xmlsr.getName());
                System.out.println("Local name = "+xmlsr.getLocalName());
                int nAttrs = xmlsr.getAttributeCount();
                for (int i = 0; i < nAttrs; i++)
                   System.out.println("Attribute ["+
                                      xmlsr.getAttributeLocalName(i)+", "+
                                      xmlsr.getAttributeValue(i)+"]");
          }
      }
      catch (IOException ioe)
      {
         ioe.printStackTrace();
      }
      catch (XMLStreamException xmlse)
      {
         xmlse.printStackTrace();
      }
      finally
      {
         if (xmlsr != null)
            try { xmlsr.close(); } catch (XMLStreamException xmlse) {}
      }
   }
}
```

6. Listings A-68 and A-69 present the `contacts.xml` document file and `XPathSearch` class.

Listing A-68. *A contacts document with a titlecased* `Name` *element*

```
<?xml version="1.0"?>
<contacts>
   <contact>
      <Name>John Doe</Name>
      <city>Chicago</city>
      <city>Denver</city>
   </contact>
   <contact>
      <name>Jane Doe</name>
```

```
      <city>New York</city>
   </contact>
   <contact>
      <name>Sandra Smith</name>
      <city>Denver</city>
      <city>Miami</city>
   </contact>
   <contact>
      <name>Bob Jones</name>
      <city>Chicago</city>
   </contact>
</contacts>
```

Listing A-69. *Searching for* `name` *or* `Name` *elements via a multiple selection*

```
import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathException;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

class XPathSearch
{
   public static void main(String[] args)
   {
      try
      {
         DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
         DocumentBuilder db = dbf.newDocumentBuilder();
```

```
            Document doc = db.parse("contacts.xml");
            XPathFactory xpf = XPathFactory.newInstance();
            XPath xp = xpf.newXPath();
            XPathExpression xpe;
            xpe = xp.compile("//contact[city='Chicago']/name/text()|"+
                             "//contact[city='Chicago']/Name/text()");
            Object result = xpe.evaluate(doc, XPathConstants.NODESET);
            NodeList nl = (NodeList) result;
            for (int i = 0; i < nl.getLength(); i++)
               System.out.println(nl.item(i).getNodeValue());
         }
         catch (IOException ioe)
         {
            System.err.println("IOE: "+ioe);
         }
         catch (SAXException saxe)
         {
            System.err.println("SAXE: "+saxe);
         }
         catch (FactoryConfigurationError fce)
         {
            System.err.println("FCE: "+fce);
         }
         catch (ParserConfigurationException pce)
         {
            System.err.println("PCE: "+pce);
         }
         catch (XPathException xpe)
         {
            System.err.println("XPE: "+xpe);
         }
      }
   }
```

7. Listings A-70 and A-71 present the `books.xsl` document stylesheet file and `MakeHTML` class.

Listing A-70. *A stylesheet for converting* `books.xml` *content to HTML*

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/books">
<html>
<head>
<title>Books</title>
</head>
<body>
<xsl:for-each select="book">
<h2>
<xsl:value-of select="normalize-space(title/text())"/>
</h2>
ISBN: <xsl:value-of select="@isbn"/><br/>
Publication Year: <xsl:value-of select="@pubyear"/><br/>
<br/><xsl:text>
</xsl:text>
<xsl:for-each select="author">
<xsl:value-of select="normalize-space(text())"/><br/><xsl:text>
</xsl:text>
</xsl:for-each>
</xsl:for-each>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Listing A-71. *Converting books XML to HTML via a stylesheet*

```
import java.io.FileReader;
import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.OutputKeys;
import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
```

```
import javax.xml.transform.TransformerFactoryConfigurationError;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

import org.w3c.dom.Document;

import org.xml.sax.SAXException;

class MakeHTML
{
   public static void main(String[] args)
   {
      try
      {
         DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
         DocumentBuilder db = dbf.newDocumentBuilder();
         Document doc = db.parse("books.xml");
         TransformerFactory tf = TransformerFactory.newInstance();
         StreamSource ssStyleSheet;
         ssStyleSheet = new StreamSource(new FileReader("books.xsl"));
         Transformer t = tf.newTransformer(ssStyleSheet);
         t.setOutputProperty(OutputKeys.METHOD, "html");
         t.setOutputProperty(OutputKeys.INDENT, "yes");
         Source source = new DOMSource(doc);
         Result result = new StreamResult(System.out);
         t.transform(source, result);
      }
      catch (IOException ioe)
      {
         System.err.println("IOE: "+ioe);
      }
      catch (FactoryConfigurationError fce)
      {
         System.err.println("FCE: "+fce);
      }
      catch (ParserConfigurationException pce)
      {
```

```
            System.err.println("PCE: "+pce);
         }
         catch (SAXException saxe)
         {
            System.err.println("SAXE: "+saxe);
         }
         catch (TransformerConfigurationException tce)
         {
            System.err.println("TCE: "+tce);
         }
         catch (TransformerException te)
         {
            System.err.println("TE: "+te);
         }
         catch (TransformerFactoryConfigurationError tfce)
         {
            System.err.println("TFCE: "+tfce);
         }
      }
}
```

Chapter 11: Working with Web Services

Chapter 11's exercises test your understanding of Java's web services support:

1. Listings A-72 through A-75 present the `Library` interface, and the `LibraryImpl`, `LibraryPublisher`, and `LibraryClient` classes.

Listing A-72. *Library SEI*

```
package ca.tutortutor.lib;

import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public interface Library
{
   @WebMethod void addBook(String isbn, String title);
   @WebMethod String getTitle(String isbn);
}
```

Listing A-73. *Library SIB*

```
package ca.tutortutor.lib;

import java.util.HashMap;
import java.util.Map;

import javax.jws.WebService;

@WebService(endpointInterface = "ca.tutortutor.lib.Library")
public class LibraryImpl implements Library
{
   private Map<String, String> books = new HashMap<>();
   public void addBook(String isbn, String title)
   {
      books.put(isbn, title);
      System.out.println("Number of books in library: "+books.size());
   }
   public String getTitle(String isbn)
   {
      return books.get(isbn);
   }
}
```

Listing A-74. *Publishing the Library web service*

```
import javax.xml.ws.Endpoint;

import ca.tutortutor.lib.LibraryImpl;

class LibraryPublisher
{
   public static void main(String[] args)
   {
      Endpoint.publish("http://localhost:9903/Library",
                       new LibraryImpl());
   }
}
```

Listing A-75. *A client for testing the Library web service*

```
import java.net.URL;

import javax.xml.namespace.QName;

import javax.xml.ws.Service;

import ca.tutortutor.lib.Library;

class LibraryClient
{
   public static void main(String[] args) throws Exception
   {
      URL url = new URL("http://localhost:9903/Library?wsdl");
      QName qname = new QName("http://lib.tutortutor.ca/",
                              "LibraryImplService");
      Service service = Service.create(url, qname);
      qname = new QName("http://lib.tutortutor.ca/", "LibraryImplPort");
      Library lib = service.getPort(qname, Library.class);
//      Library lib = service.getPort(Library.class);
      lib.addBook("9781430234135", "Android Recipes");
      System.out.println(lib.getTitle("9781430234135"));
   }
}
```

2. Listing A-76 presents the `LibraryClientSAAJ` class.

Listing A-76. *A SAAJ client for testing the Library web service*

```
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConnection;
import javax.xml.soap.SOAPConnectionFactory;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
```

```
import javax.xml.namespace.QName;

class LibraryClientSAAJ
{
   final static String ENDPOINT = "http://localhost:9903/Library";
   public static void main(String[] args) throws Exception
   {
      SOAPConnectionFactory soapcf = SOAPConnectionFactory.newInstance();
      SOAPConnection soapc = soapcf.createConnection();
      MessageFactory mf = MessageFactory.newInstance();
      SOAPMessage soapm = mf.createMessage();
      SOAPPart soapp = soapm.getSOAPPart();
      SOAPEnvelope soape = soapp.getEnvelope();
      SOAPBody soapb = soape.getBody();
      soape.getHeader().detachNode();
      soape.addNamespaceDeclaration("xsd", "http://www.w3.org/2001/XMLSchema");
      soape.addNamespaceDeclaration("xsi",
                                 "http://www.w3.org/2001/XMLSchema-instance");
      QName name = new QName("http://lib.tutortutor.ca/", "addBook", "tns");
      SOAPElement soapel = soapb.addBodyElement(name);
      soapel.addChildElement("arg0").addTextNode("9781430234135")
            .setAttribute("xsi:type", "xsd:string");
      soapel.addChildElement("arg1").addTextNode("Android Recipes")
            .setAttribute("xsi:type", "xsd:string");
      soapm.writeTo(System.out);
      System.out.println();
      System.out.println();
      SOAPMessage response = soapc.call(soapm, ENDPOINT);
      response.writeTo(System.out);
      System.out.println();
      System.out.println();
      soapm = mf.createMessage();
      soapm = mf.createMessage();
      soapp = soapm.getSOAPPart();
      soape = soapp.getEnvelope();
      soapb = soape.getBody();
      soape.getHeader().detachNode();
      soape.addNamespaceDeclaration("xsd", "http://www.w3.org/2001/XMLSchema");
      soape.addNamespaceDeclaration("xsi", "http://www.w3.org/2001/XMLSchema-Instance");
      name = new QName("http://lib.tutortutor.ca/", "getTitle", "tns");
```

```
      soapel = soapb.addBodyElement(name);
      soapel.addChildElement("arg0").addTextNode("9781430234135")
            .setAttribute("xsi:type", "xsd:string");
      soapm.writeTo(System.out);
      System.out.println();
      System.out.println();
      response = soapc.call(soapm, ENDPOINT);
      response.writeTo(System.out);
   }
}
```

Examine the XML Schema document that accompanies the WSDL for the Library web service and you'll discover that `addBook`'s parameter names are `arg0` and `arg1`. Similarly, you'll discover that `getTitle`'s parameter name is `arg0`.

If you prefer to use a different parameter name in `LibraryClientSAAJ`, perhaps to follow the naming convention in Listing 11-16 where an `isbn` element and not an `arg0` element is a child element of the `getTitle` element, you can specify a different parameter name with help from the `javax.jws.WebParam` annotation type. For example, in Listing A-72, if you replace `@WebMethod String getTitle(String isbn);` with `@WebMethod String getTitle(@WebParam(name = "isbn") String isbn);`, you can specify in Listing A-76 `soapel.addChildElement("isbn").addTextNode("9781430234135").setAttribute("xsi:type", "xsd:string");` instead of having to specify `soapel.addChildElement("arg0").addTextNode("9781430234135").setAttribute("xsi:type", "xsd:string");`.

3. The RESTful web service described by Listing 11-11's `Library` class is flawed in that the `doDelete()` method doesn't notify the client when requested to delete a nonexistent book. You could modify this method to report this attempt by replacing `library.remove(isbn);` with `if (!library.remove(isbn)) throw new HTTPException(404);`. Response code 404 indicates that the resource was not found.

Chapter 12: Java 7 Meets Android

Chapter 12's exercises test your understanding of Android app development:

1. Assuming Windows and a `C:\prj\dev` hierarchy, execute the following command to create `SimpleApp`:

```
android create project –n SimpleApp -t 1 -p c:\prj\dev\SimpleApp -a SimpleActivity -k↩
ca.tutortutor.simpleapp
```

Replace `C:\prj\dev\SimpleApp\src\ca\tutortutor\simpleapp\SimpleActivity.java` with Listing 12-1.

Assuming that `C:\prj\dev\SimpleApp` is the current directory, execute `ant debug` to create this app's APK.

Assuming that `C:\prj\dev\SimpleApp\bin` is current (and that `adb`'s location has been added to the `PATH` environment variable), execute `adb install SimpleApp-debug.apk` to install this APK on test_AVD.

2. `SimpleActivity` is the name of the default or main activity (as specified by `android`'s `-a` option). When the project is created, `android` stores `<string name="app_name">SimpleActivity</string>` in `strings.xml`, and the `AndroidManifest.xml` file's `application` element assigns `label="@string/app_name"` to its `label` attribute.
3. Execute `adb uninstall ca.tutortutor.simpleapp` to uninstall `SimpleApp` from test_AVD.
4. Listings A-77 and A-78 present the `SimpleActivity` and `SimpleActivity2` classes.

Listing A-77. *Starting `SimpleActivity2` via an implicit intent*

```
package ca.tutortutor.simpleapp;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

public class SimpleActivity extends Activity
{
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState); // Always call superclass method first.
      System.out.println("onCreate(Bundle) called");
      Intent intent = new Intent();
      intent.setAction(Intent.ACTION_VIEW);
      intent.setType("image/jpeg");
      intent.addCategory(Intent.CATEGORY_DEFAULT);
      SimpleActivity.this.startActivity(intent);
   }
   @Override
   public void onDestroy()
   {
      super.onDestroy(); // Always call superclass method first.
      System.out.println("onDestroy() called");
   }
}
```

Listing A-78. *Presenting the contents of the intent used to start* `SimpleActivity2` *via a toast message*

```
package ca.tutortutor.simpleapp;

import android.app.Activity;

import android.os.Bundle;

import android.widget.Toast;

public class SimpleActivity2 extends Activity
{
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      Toast.makeText(this, getIntent().toString(), Toast.LENGTH_LONG).show();
   }
}
```

5. After clicking the `SimpleActivity` icon, `SimpleActivity` runs and a pop-up menu appears. This menu gives you the option of using `SimpleActivity` or `Android`'s default View picture to perform the View action. Clicking `SimpleActivity` runs `SimpleActivity`; `SimpleActivity2` doesn't run because `SimpleActivity2` isn't distinguished from `SimpleActivity` in the manifest. (There probably isn't a `label` attribute on `SimpleActivity2`'s `<activity>` tag or (if present) a `label` attribute that identifies a unique string resource for `SimpleActivity2` – it probably refers to `app_name`). To fix this problem, you must add an entry such as `<string name="simple2">SimpleActivity2</string>` to `strings.xml` and modify `AndroidManifest.xml`'s `<activity>` tag for `SimpleActivity2` to refer to this string resource: `<activity android:name=".SimpleActivity2" android:label="@string/simple2">`. This time, when you install the modified app and click its `StartActivity` icon, the menu will present `SimpleActivity2` instead of `SimpleActivity`. Clicking this menuitem will result in `SimpleActivity2` running and displaying the contents of its `Intent` object via the toast message.

Listings A-79 and A-80 present the modified `strings.xml` and `AndroidManifest.xml` files.

Listing A-79. *Adding an entry to* `strings.xml` *to account for* `SimpleActivity2`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">SimpleActivity</string>
    <string name="simple2">SimpleActivity2</string>
</resources>
```

Listing A-80. *The manifest for making Android aware of* `SimpleActivity` *and* `SimpleActivity2`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="ca.tutortutor.simpleapp">
   <application android:label="@string/app_name" android:icon="@drawable/icon">
      <activity android:name=".SimpleActivity" android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".SimpleActivity2" android:label="@string/simple2">
         <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <data android:mimeType="image/jpeg" />
            <category android:name="android.intent.category.DEFAULT" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```