with a `FileWriter` because writing only one character at a time to a file is tremendously inefficient.

The other interesting processing stream is `PrintWriter`, which has already been discussed. It adds methods such as `print`, `println`, and `printf` to convert types such as `int` and `double` into individual characters.

9.7.3 Byte Streams

The structure of the byte input streams is similar to character input streams. A core class, `InputStream`, is extended four times to provide bytes from files, string buffers, byte arrays, and pipes. A similar set of classes provide processing streams to buffer the stream, count the lines, and push back information previously read.

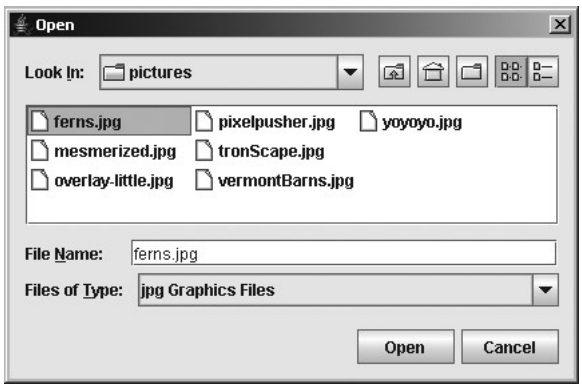
Similarly, the byte output streams mirror the character output streams. The core class, `OutputStream`, is extended to write bytes to files, byte arrays, and pipes. `BufferedOutputStream` and `PrintStream` are analogous to `BufferedWriter` and `PrintWriter`.

9.8 GUI: File Choosers and Images

Files are used with graphical user interfaces in a number of ways. One is to ask the user for a filename with an easy-to-use dialog box, as shown in Figure 9-14. Another is to read an image from a file and paint it on the screen. The image could have been created with a separate editor or taken with a camera.

(figure 9-14)

Dialog box displayed by `JFileChooser` *to choose a file*



9.8.1 Using `JFileChooser`

Using a professional dialog box to obtain a filename from a user is easy, thanks to the libraries that come with Java. The test program in Listing 9-12 displays a dialog box like the one shown in Figure 9-14.

In lines 13–14, the dialog box is created and shown to the user. The method `showOpenDialog` is meant for opening existing files. Another method, `showSaveDialog`, is for saving a file. The difference is the text placed on the buttons and the title bar.

The user's action is returned as an integer from `showOpenDialog`, and could be `APPROVE_OPTION` (the user chose a file), `CANCEL_OPTION` (the user cancelled the dialog without choosing a file), or `ERROR_OPTION` (an error occurred). If the user chooses a file, the full path and filename can be obtained with the code shown in line 19. Of course, your program should open and use the file instead of only printing the name on the console.

Listing 9-12: *A program demonstrating the use of* `JFileChooser`

```
1  import javax.swing.JFileChooser;
2  import java.io.File;
3
4  /** A program testing the operation of JFileChooser.
5  *
6  *  @author Byron Weber Becker */
7  public class Main extends Object
8  {
9    public static void main(String[] args)
10   { System.out.println("Ready to get a filename.");
11
12     // construct the dialog and show it to the user
13     JFileChooser chooser = new JFileChooser();
14     int result = chooser.showOpenDialog(null);
15
16     if (result == JFileChooser.APPROVE_OPTION)
17     { // Open the file and use it
18       System.out.println("You chose '" +
19               chooser.getSelectedFile().getPath());
20     }
21   }
22 }
```

FIND THE CODE

ch09/fileChooser/

Typically your program will be interested only certain kinds of files. For example, the next section shows how to display certain kinds of images on the screen. These images are normally stored in files that end with an extension of either `.gif` or `.jpg`. With the help of the `FileExtensionFilter` class, shown in Listing 9-13, `JFileChooser` will show only the relevant classes. Use it by adding the following lines between lines 13 and 14 in Listing 9-12:

```
chooser.addChoosableFileFilter(
          new FileExtensionFilter(".jpg", "jpg Graphics Files"));
chooser.addChoosableFileFilter(
          new FileExtensionFilter(".gif", "gif Graphics Files"));
```

`FileExtensionFilter` works by overriding the `accept` method in its superclass. `JFileChooser` calls this method once for each file or directory in the current directory. If `accept` returns `true`, the file or directory is displayed so the user can choose it.

FIND THE CODE

ch09/fileChooser/

Listing 9-13: *A filter used by* `JFileChooser` *to show only files with the specified extension*

```
1  import javax.swing.filechooser.FileFilter;
2  import java.io.File;
3
4  /** A class used to filter out some files so that JFileChooser only shows files with a
5  *  specified extension.
6  *
7  *  @author Byron Weber Becker */
8  public class FileExtensionFilter extends FileFilter
9  {
10   private String ext;
11   private String descr;
12
13   /** Accept files ending with the given extension.
14   *  @param extension The extension to accept (e.g., ".jpg")
15   *  @param description A description of the file accepted */
16   public FileExtensionFilter(String extension,
17                              String description)
18   { super();
19     this.ext = extension.toLowerCase();
20     this.descr = description;
21   }
22
23   /** Decide whether or not the given file should be displayed. In our case, include
24   *  directories as well as files with a name ending in the specified extension.
25   *  @param f A description of one file.
26   *  @return True if the file should be displayed to the user; false otherwise. */
```

Listing 9-13: *A filter used by* `JFileChooser` *to show only files with the specified extension* (continued)

```
27    public boolean accept(File f)
28    { return f.isDirectory() ||
29             f.getName().toLowerCase().endsWith(this.ext);
30    }
31
32    /** Return the description of the files accepted.
33    *  @return A description of the files this filter accepts.*/
34    public String getDescription()
35    { return this.descr;
36    }
37 }
```

9.8.2 Displaying Images from a File

The program in Listing 9-12 can be easily modified to display an image from a file. The program currently prints the name of the file selected by the user (see lines 18–19). The new program will replace lines 18–19 with the following code to create a component that displays an image it reads from a file, and then shows that component in a frame. Notice that the filename is obtained from the chooser and passed to `ImageComponent`'s constructor.

```
// Create a component to display an image
ImageComponent imageComp = new
    ImageComponent(chooser.getSelectedFile().getPath());

// Display the image component in a frame
JFrame f = new JFrame("Image");
f.setContentPane(imageComp);
f.setSize(500, 500);
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setVisible(true);
```

LOOKING BACK

This code uses the Display a Frame pattern. See Section 1.7.5.

The source code for `ImageComponent` is more interesting and is shown in Listing 9-14. It is passed the name of the image file via the parameter in the constructor. A class from the Java library, `ImageIcon`, is used to read the image from the file. Supported types of images include `.gif`, `.jpg`, and `.png`. Once the image is loaded, the preferred size for the component is set to the image's size. If the preferred size is not set, the `JFrame` will make it so small that it can't be seen.

As with previous extensions of `JComponent`, the `paintComponent` method is overridden to do the painting. In the past, the `Graphics` parameter, `g`, has been used to call such methods as `drawRect` and `fillOval`. Here, it is used in line 24 to paint the

image read from the file. The second and third parameters give the desired location of the upper-left corner of the image. The zero values shown here put the image in the upper-left corner of the component.

`drawImage` is overloaded. Another version includes two more parameters to specify the painted image's width and height. This is useful if you want to scale the image. It is also possible to use `drawImage` to draw a background image and then add details on top with calls to `drawRect` and similar methods.

FIND THE CODE

ch09/displayImage/

Listing 9-14: *A new kind of component that displays an image from a file*

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  /** A component that paints an image stored in a file.
5  *
6  * @author Byron Weber Becker */
7  public class ImageComponent extends JComponent
8  {
9    private ImageIcon image;
10
11   /** Construct the new component.
12   * @param fileName The file where the image is stored. */
13   public ImageComponent(String fileName)
14   { super();
15     this.image = new ImageIcon(fileName);
16     this.setPreferredSize(new Dimension(
17                             this.image.getIconWidth(),
18                             this.image.getIconHeight()));
19   }
20
21   /** Paint this component, including its image. */
22   public void paintComponent(Graphics g)
23   { super.paintComponent(g);
24     g.drawImage(this.image.getImage(), 0, 0, null);
25   }
26 }
```