1. Filenames and Pathnames

Constructing a Filename Path

A `File` object is used to represent a filename. Creating the `File` object has no effect on the file system; the filename need not exist nor is it created.

On Windows, this example creates the path `\a\b`. On Unix, the path would be `/a/b`.

```
String path = File.separator + "a" + File.separator + "b";
```

Converting Between a Filename Path and a URL

```
// Create a file object
File file = new File("filename");

// Convert the file object to a URL
URL url = null;
try {
    // The file need not exist. It is made into an absolute path
    // by prefixing the current working directory
    url = file.toURL();    //file:/d:/almanac1.4/java.io/filename
} catch (MalformedURLException e) {
}

// Convert the URL to a file object
file = new File(url.getFile());  // d:/almanac1.4/java.io/filename

// Read the file contents using the URL
try {
    // Open an input stream
    InputStream is = url.openStream();

    // Read from is

    is.close();
} catch (IOException e) {
    // Could not open the file
}
```

Getting an Absolute Filename Path from a Relative Filename Path

```
File file = new File("filename.txt");
file = file.getAbsoluteFile();  // c:\temp\filename.txt

file = new File("dir"+File.separatorChar+"filename.txt");
file = file.getAbsoluteFile();  // c:\temp\dir\filename.txt

file = new File(".."+File.separatorChar+"filename.txt");
file = file.getAbsoluteFile();  // c:\temp\..\filename.txt
```

```
// Note that filename.txt does not need to exist
```

Determining If Two Filename Paths Refer to the Same File

A filename path may include redundant names such as `.' or `. .' or symbolic links (on UNIX platforms). `File.getCanonicalFile()` converts a filename path to a unique canonical form suitable for comparisons.

```
File file1 = new File("./filename");
File file2 = new File("filename");

// Filename paths are not equal
boolean b = file1.equals(file2);      // false

// Normalize the paths
try {
    file1 = file1.getCanonicalFile(); // c:\almanac1.4\filename
    file2 = file2.getCanonicalFile(); // c:\almanac1.4\filename
} catch (IOException e) {
}

// Filename paths are now equal
b = file1.equals(file2);              // true
```

Getting the Parents of a Filename Path

```
// Get the parent of a relative filename path
File file = new File("Ex1.java");
String parentPath = file.getParent();      // null
File parentDir = file.getParentFile();     // null

// Get the parents of an absolute filename path
file = new File("D:\\almanac\\Ex1.java");
parentPath = file.getParent();             // D:\almanac
parentDir = file.getParentFile();          // D:\almanac

parentPath = parentDir.getParent();        // D:\
parentDir = parentDir.getParentFile();     // D:\

parentPath = parentDir.getParent();        // null
parentDir = parentDir.getParentFile();     // null
```

Determining If a Filename Path Is a File or a Directory

```
File dir = new File("directoryName");

boolean isDir = dir.isDirectory();
if (isDir) {
    // dir is a directory
```

```
} else {
    // dir is a file
}
```

2. File

Determining If a File or Directory Exists

```
boolean exists = (new File("filename")).exists();
if (exists) {
    // File or directory exists
} else {
    // File or directory does not exist
}
```

e20. Creating a File

```
try {
    File file = new File("filename");

    // Create file if it does not exist
    boolean success = file.createNewFile();
    if (success) {
        // File did not exist and was created
    } else {
        // File already exists
    }
} catch (IOException e) {
}
```

e21. Getting the Size of a File

```
File file = new File("infilename");

// Get the number of bytes in the file
long length = file.length();
```

Deleting a File

```
boolean success = (new File("filename")).delete();
if (!success) {
    // Deletion failed
}
```

Creating a Temporary File

```
try {
    // Create temp file.
    File temp = File.createTempFile("pattern", ".suffix");

    // Delete temp file when program exits.
    temp.deleteOnExit();

    // Write to temp file
    BufferedWriter out = new BufferedWriter(new FileWriter(temp));
    out.write("aString");
    out.close();
} catch (IOException e) {
}
```

Renaming a File or Directory

```
// File (or directory) with old name
File file = new File("oldname");

// File (or directory) with new name
File file2 = new File("newname");

// Rename file (or directory)
boolean success = file.renameTo(file2);
if (!success) {
    // File was not successfully renamed
}
```

Moving a File or Directory to Another Directory

```
// File (or directory) to be moved
File file = new File("filename");

// Destination directory
File dir = new File("directoryname");

// Move file to new directory
boolean success = file.renameTo(new File(dir, file.getName()));
if (!success) {
    // File was not successfully moved
}
```

Getting and Setting the Modification Time of a File or Directory

This example gets the last modified time of a file or directory and then sets it to the current time.

```
File file = new File("filename");
```

```
    // Get the last modified time
    long modifiedTime = file.lastModified();
    // 0L is returned if the file does not exist

    // Set the last modified time
    long newModifiedTime = System.currentTimeMillis();
    boolean success = file.setLastModified(newModifiedTime);
    if (!success) {
        // operation failed.
    }
```

Forcing Updates to a File to the Disk

In some applications, such as transaction processing, it is necessary to ensure that an update has been made to the disk. `FileDescriptor.sync()` blocks until all changes to a file are written to disk.

```
    try {
        // Open or create the output file
        FileOutputStream os = new FileOutputStream("outfilename");
        FileDescriptor fd = os.getFD();

        // Write some data to the stream
        byte[] data = new byte[]{(byte)0xCA, (byte)0xFE, (byte)0xBA,
(byte)0xBE};
        os.write(data);

        // Flush the data from the streams and writers into system
buffers.
        // The data may or may not be written to disk.
        os.flush();

        // Block until the system buffers have been written to disk.
        // After this method returns, the data is guaranteed to have
        // been written to disk.
        fd.sync();
    } catch (IOException e) {
    }
```

3. Directories

Getting the Current Working Directory

The working directory is the location in the file system from where the `java` command was invoked.

```
    String curDir = System.getProperty("user.dir");
```

Creating a Directory

```
    // Create a directory; all ancestor directories must exist
```

```
    boolean success = (new File("directoryName")).mkdir();
    if (!success) {
        // Directory creation failed
    }

    // Create a directory; all non-existent ancestor directories are
    // automatically created
    success = (new File("directoryName")).mkdirs();
    if (!success) {
        // Directory creation failed
    }
```

Deleting a Directory

```
    // Delete an empty directory
    boolean success = (new File("directoryName")).delete();
    if (!success) {
        // Deletion failed
    }
```

If the directory is not empty, it is necessary to first recursively delete all files and subdirectories in the directory. Here is a method that will delete a non-empty directory.

```
    // Deletes all files and subdirectories under dir.
    // Returns true if all deletions were successful.
    // If a deletion fails, the method stops attempting to delete and
returns false.
    public static boolean deleteDir(File dir) {
        if (dir.isDirectory()) {
            String[] children = dir.list();
            for (int i=0; i<children.length; i++) {
                boolean success = deleteDir(new File(dir,
children[i]));
                if (!success) {
                    return false;
                }
            }
        }

        // The directory is now empty so delete it
        return dir.delete();
    }
```

Listing the Files or Subdirectories in a Directory

This example lists the files and subdirectories in a directory. To list all descendant files and subdirectories under a directory, see **e33** Traversing the Files and Directories Under a Directory.

```
    File dir = new File("directoryName");

    String[] children = dir.list();
    if (children == null) {
        // Either dir does not exist or is not a directory
```

```
} else {
    for (int i=0; i<children.length; i++) {
        // Get filename of file or directory
        String filename = children[i];
    }
}

// It is also possible to filter the list of returned files.
// This example does not return any files that start with `.'.
FilenameFilter filter = new FilenameFilter() {
    public boolean accept(File dir, String name) {
        return !name.startsWith(".");
    }
};
children = dir.list(filter);


// The list of files can also be retrieved as File objects
File[] files = dir.listFiles();

// This filter only returns directories
FileFilter fileFilter = new FileFilter() {
    public boolean accept(File file) {
        return file.isDirectory();
    }
};
files = dir.listFiles(fileFilter);
```

Listing the File System Roots

UNIX file systems have a single root, `/'. On Windows, each drive is a root. For example the C drive is represented by the root `C:\`.

```
File[] roots = File.listRoots();
for (int i=0; i<roots.length; i++) {
    process(roots[i]);
}
```

Traversing the Files and Directories Under a Directory

This example implements methods that recursively visits all files and directories under a directory.

```
// Process all files and directories under dir
public static void visitAllDirsAndFiles(File dir) {
    process(dir);

    if (dir.isDirectory()) {
        String[] children = dir.list();
        for (int i=0; i<children.length; i++) {
            visitAllDirsAndFiles(new File(dir, children[i]));
        }
```

```
        }
    }

    // Process only directories under dir
    public static void visitAllDirs(File dir) {
        if (dir.isDirectory()) {
            process(dir);

            String[] children = dir.list();
            for (int i=0; i<children.length; i++) {
                visitAllDirs(new File(dir, children[i]));
            }
        }
    }

    // Process only files under dir
    public static void visitAllFiles(File dir) {
        if (dir.isDirectory()) {
            String[] children = dir.list();
            for (int i=0; i<children.length; i++) {
                visitAllFiles(new File(dir, children[i]));
            }
        } else {
            process(dir);
        }
    }
```

4. Reading and writing

Reading Text from a File

```
    try {
        BufferedReader in = new BufferedReader(new
FileReader("infilename"));
        String str;
        while ((str = in.readLine()) != null) {
            process(str);
        }
        in.close();
    } catch (IOException e) {
    }
```

Reading Text from Standard Input

```
    try {
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        String str = "";
        while (str != null) {
```

```
            System.out.print("> prompt ");
            str = in.readLine();
            process(str);
        }
    } catch (IOException e) {
    }
```

Reading a File into a Byte Array

This example implements a method that reads the entire contents of a file into a byte array.

See also **e35** Reading Text from a File.

```
    // Returns the contents of the file in a byte array.
    public static byte[] getBytesFromFile(File file) throws IOException
{
        InputStream is = new FileInputStream(file);

        // Get the size of the file
        long length = file.length();

        // You cannot create an array using a long type.
        // It needs to be an int type.
        // Before converting to an int type, check
        // to ensure that file is not larger than Integer.MAX_VALUE.
        if (length > Integer.MAX_VALUE) {
            // File is too large
        }

        // Create the byte array to hold the data
        byte[] bytes = new byte[(int)length];

        // Read in the bytes
        int offset = 0;
        int numRead = 0;
        while (offset < bytes.length
               && (numRead=is.read(bytes, offset, bytes.length-offset))
>= 0) {
            offset += numRead;
        }

        // Ensure all the bytes have been read in
        if (offset < bytes.length) {
            throw new IOException("Could not completely read file
"+file.getName());
        }

        // Close the input stream and return bytes
        is.close();
        return bytes;
    }
```

Writing to a File

If the file does not already exist, it is automatically created.

```
    try {
        BufferedWriter out = new BufferedWriter(new
FileWriter("outfilename"));
        out.write("aString");
        out.close();
    } catch (IOException e) {
    }
```

Appending to a File

```
    try {
        BufferedWriter out = new BufferedWriter(new
FileWriter("filename", true));
        out.write("aString");
        out.close();
    } catch (IOException e) {
    }
```

Using a Random Access File

```
    try {
        File f = new File("filename");
        RandomAccessFile raf = new RandomAccessFile(f, "rw");

        // Read a character
        char ch = raf.readChar();

        // Seek to end of file
        raf.seek(f.length());

        // Append to the end
        raf.writeChars("aString");
        raf.close();
    } catch (IOException e) {
    }
```

5. Serialization

Serializing an Object

The object to be serialized must implement `java.io.Serializable`. This example serializes a `javax.swing.JButton` object.

See also **e45** Deserializing an Object.

```
    Object object = new javax.swing.JButton("push me");

    try {
        // Serialize to a file
        ObjectOutput out = new ObjectOutputStream(new
FileOutputStream("filename.ser"));
        out.writeObject(object);
        out.close();

        // Serialize to a byte array
        ByteArrayOutputStream bos = new ByteArrayOutputStream() ;
        out = new ObjectOutputStream(bos) ;
        out.writeObject(object);
        out.close();

        // Get the bytes of the serialized object
        byte[] buf = bos.toByteArray();
    } catch (IOException e) {
    }
```

Deserializing an Object

This example deserializes a `javax.swing.JButton` object.

See also **e44** Serializing an Object.

```
    try {
        // Deserialize from a file
        File file = new File("filename.ser");
        ObjectInputStream in = new ObjectInputStream(new
FileInputStream(file));
        // Deserialize the object
        javax.swing.JButton button = (javax.swing.JButton)
in.readObject();
        in.close();

        // Get some byte array data
        byte[] bytes = getBytesFromFile(file);
        // see e36 Reading a File into a Byte Array for the
implementation of this method

        // Deserialize from a byte array
        in = new ObjectInputStream(new ByteArrayInputStream(bytes));
        button = (javax.swing.JButton) in.readObject();
        in.close();
    } catch (ClassNotFoundException e) {
```

```
    } catch (IOException e) {
    }
```

Implementing a Serializable Singleton

By default, the deserialization process creates new instances of classes. This example demonstrates how to customize the deserialization process of a singleton to avoid creating new instances of the singleton.

```
    public class MySingleton implements Serializable {
        static MySingleton singleton = new MySingleton();

        private MySingleton() {
        }

        // This method is called immediately after an object of this
class is deserialized.
        // This method returns the singleton instance.
        protected Object readResolve() {
            return singleton;
        }
    }
```

6. Encoding

Reading UTF-8 Encoded Data

```
    try {
        BufferedReader in = new BufferedReader(
            new InputStreamReader(new FileInputStream("infilename"),
"UTF8"));
        String str = in.readLine();
    } catch (UnsupportedEncodingException e) {
    } catch (IOException e) {
    }
```

Writing UTF-8 Encoded Data

```
    try {
        Writer out = new BufferedWriter(new OutputStreamWriter(
            new FileOutputStream("outfilename"), "UTF8"));
        out.write(aString);
        out.close();
    } catch (UnsupportedEncodingException e) {
    } catch (IOException e) {
    }
```

Reading ISO Latin-1 Encoded Data

```
    try {
        BufferedReader in = new BufferedReader(
            new InputStreamReader(new FileInputStream("infilename"),
"8859_1"));
        String str = in.readLine();
    } catch (UnsupportedEncodingException e) {
    } catch (IOException e) {
    }
```

Writing ISO Latin-1 Encoded Data

```
    try {
        Writer out = new BufferedWriter(
            new OutputStreamWriter(new FileOutputStream("outfilename"),
"8859_1"));
        out.write(aString);
        out.close();
    } catch (UnsupportedEncodingException e) {
    } catch (IOException e) {
    }
```

7. Parsing

Tokenizing Java Source Code

The `StreamTokenizer` can be used for simple parsing of a Java source file into tokens. The tokenizer can be aware of Java-style comments and ignore them. It is also aware of Java quoting and escaping rules.

```
    try {
        // Create the tokenizer to read from a file
        FileReader rd = new FileReader("filename.java");
        StreamTokenizer st = new StreamTokenizer(rd);

        // Prepare the tokenizer for Java-style tokenizing rules
        st.parseNumbers();
        st.wordChars('_', '_');
        st.eolIsSignificant(true);

        // If whitespace is not to be discarded, make this call
        st.ordinaryChars(0, ' ');

        // These calls caused comments to be discarded
        st.slashSlashComments(true);
        st.slashStarComments(true);

        // Parse the file
        int token = st.nextToken();
        while (token != StreamTokenizer.TT_EOF) {
            token = st.nextToken();
```

```
            switch (token) {
            case StreamTokenizer.TT_NUMBER:
                // A number was found; the value is in nval
                double num = st.nval;
                break;
            case StreamTokenizer.TT_WORD:
                // A word was found; the value is in sval
                String word = st.sval;
                break;
            case '"':
                // A double-quoted string was found; sval contains the
contents
                String dquoteVal = st.sval;
                break;
            case '\'':
                // A single-quoted string was found; sval contains the
contents
                String squoteVal = st.sval;
                break;
            case StreamTokenizer.TT_EOL:
                // End of line character found
                break;
            case StreamTokenizer.TT_EOF:
                // End of file has been reached
                break;
            default:
                // A regular character was found; the value is the
token itself
                char ch = (char)st.ttype;
                break;
            }
        }
        rd.close();
    } catch (IOException e) {
    }
```