VERIFY THAT A FILE OR DIRECTORY EXISTS

When working with files and directories, it is often necessary to verify that a file or directory exists before performing an action. For example, you should verify that a file exists before deleting the file. This is particularly important when working with files and directories located on a network, since events beyond your control can make the files and directories unavailable.

To verify that a file or directory exists, you create a `File` object that uses the path of the file or directory as its argument. The class that is used to create a `File` object is located in the `java.io` package. You must use the `page` directive to import the `java.io` package before you can create a `File` object.

You may want to store the path of the file or directory you want to check in a variable and then use the variable as the argument for the `File` object. A path can also be submitted by a form or retrieved from a database. When specifying the path of a file or directory, you should use slashes (/).

Once you have created a `File` object for a file or directory, you can use methods of the object to determine information about the file or directory. You use the `exists` method to determine if the file or directory exists on the current system. The `exists` method returns a value of `true` if the file or directory exists and a value of `false` if the file or directory does not exist.

The `isFile` method of the `File` object allows you to verify whether a file or directory represented by a `File` object is a file, while the `isDirectory` method lets you verify whether the item is a directory. These methods return a value of `true` or `false`, depending upon the type of the item.

Extra

Permissions may have been set for a file, affecting the types of tasks you can perform while working with the file. For example, a file's permissions can regulate whether you will be able to read or write to the file. To determine whether you have permission to read a file, use the `canRead` method of the `File` object. To determine whether you have permission to write to a file, use the `canWrite` method of the `File` object. If you attempt to read or write to a file that you do not have permission to work with, an error will usually be generated.

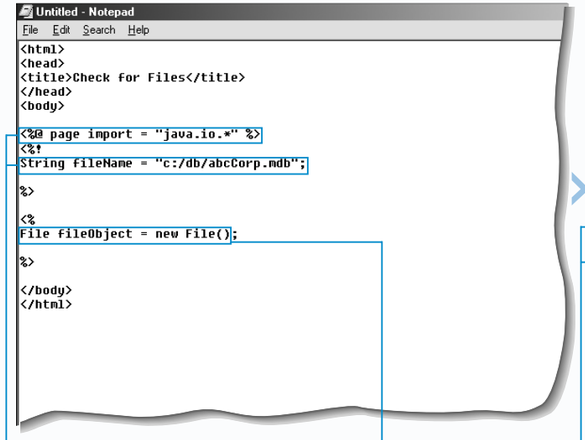
Example:

```
if (fileObject.canRead())
     out.print("You can read the file " + fileName);

if (fileObject.canWrite())
     out.print("You can write to the file " + fileName);
```

When specifying the path to a file or directory for the argument of a `File` object, you can use a relative or absolute path. A relative path specifies the location of the file or directory relative to the current directory. For example, the relative path `../file.txt` refers to a file named file that is located in the parent directory of the current directory. An absolute path specifies the location of a file or directory in relation to the root directory of the storage system in which the file is stored, such as `c:/data/examples/file.txt`.

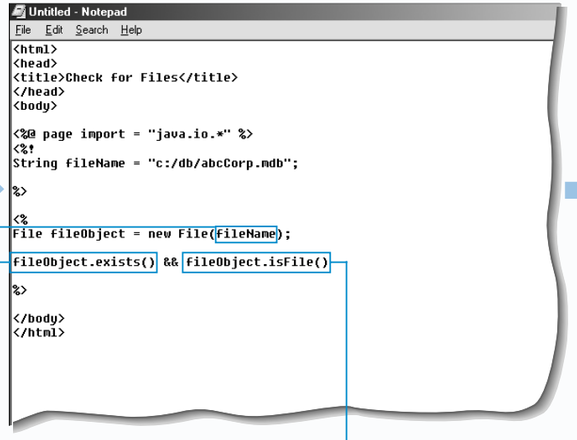
VERIFY THAT A FILE OR DIRECTORY EXISTS



1 To import the `java.io` package, type **<%@ page import = "java.io.*" %>**.

2 To store the path of a file you want to check in a variable, type the code that assigns the path to the variable.

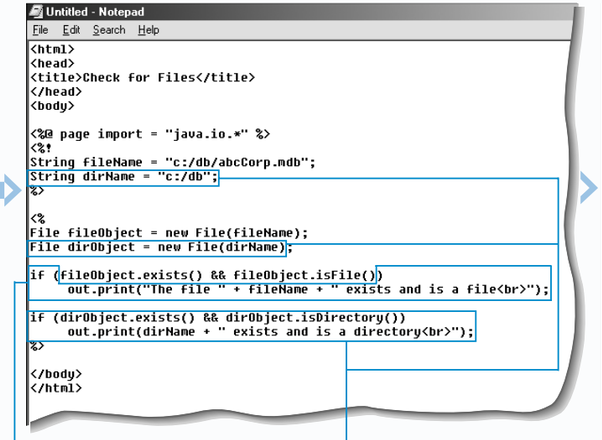
3 To create a `File` object for the file, type **File** followed by a name for the `File` object. Then type **= new File()**.



4 Between the parentheses, type the name of the variable that stores the path of the file you want to check.

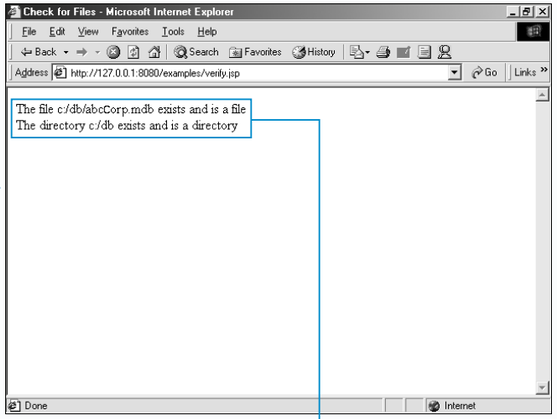
5 To determine whether the file exists, type the name of the `File` object, immediately followed by **.exists()**.

6 To determine whether the file is a file, type the name of the `File` object, immediately followed by **.isFile()**.



7 Type the code that uses the `File` object.

8 To verify if a directory exists and is a directory, repeat steps 2 to 7 for the directory, except use the `.isDirectory()` method in step 6.



9 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the results of determining whether a file and directory exist.

CREATE AND WRITE TO A FILE

A JSP page can be used to create a new file and then write information to the file. A file could be created to track how many times the JSP page has been accessed or store data retrieved from a database. You can also use a JSP page to create and write other JSP files.

You create a `File` object to specify a name and location for the new file. You can then use the `createNewFile` method of the `File` object to create the new file. You must use the `page` directive to import the `java.io` package from the Java class library before creating a new file.

Information is written to a file using an output stream. Stream is the term typically used to describe one continuous line of data. You use a `FileOutputStream` object to create an output stream and specify the name of the `File` object that represents the file you want to write to.

In order to write primitive data types to the output stream, a `DataOutputStream` object must be created. You then associate the `DataOutputStream` object with the `FileOutputStream` object.

You use a write method of the `DataOutputStream` object to write information to the file. The method you should use depends on the type of data you want to write to the file. For example, if you want to write an integer value, you would use the `writeInt` method.

After all the information has been written to the file, you can use the `close` method of the `DataOutputStream` object to close the output stream.

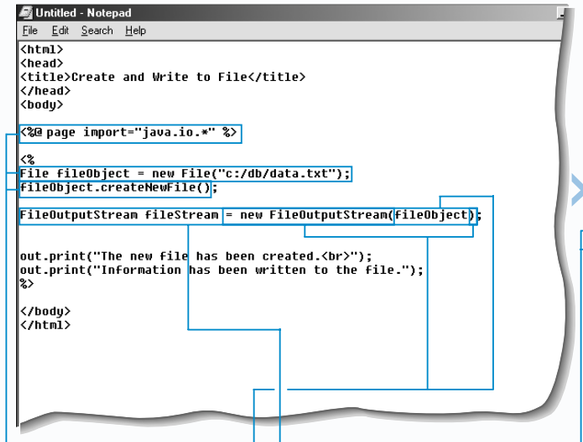
When you display the JSP page in a Web browser, the file will be created and the information you specified will be written to the file. You can open the file with the appropriate program or use a JSP page to read the file.

Extra

The `DataOutputStream` object offers several methods that can be used to write information to a file. Each method writes a different primitive data type or string value to the output stream.

WRITE METHOD:	DESCRIPTION:
`writeBoolean(boolean value)`	Writes a `boolean` value to the output stream.
`writeByte(int value)`	Writes a `byte` value to the output stream.
`writeBytes(String value)`	Writes a string of `byte` values to the output stream.
`writeChar(int value)`	Writes a `char` value to the output stream.
`writeChars(String value)`	Writes a string of `char` values to the output stream.
`writeDouble(double value)`	Converts the `double` argument to a `long` value and writes the `long` value to the output stream.
`writeFloat(float value)`	Converts the `float` argument to an `int` value and writes the `int` value to the output stream.
`writeInt(int value)`	Writes an `int` value to the output stream.
`writeLong(long value)`	Writes a `long` value to the output stream.
`writeShort(int value)`	Writes a `short` value to the output stream.

CREATE AND WRITE TO A FILE



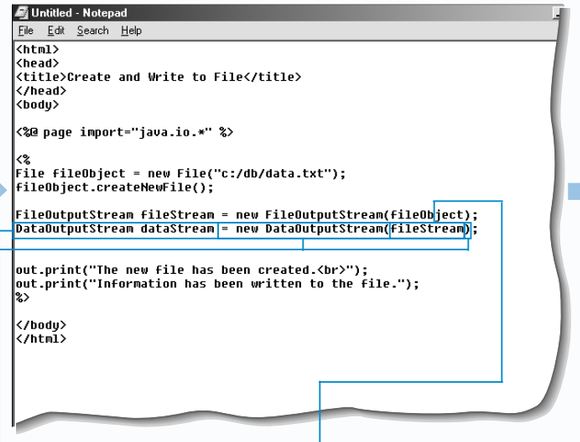
1 Type the code that imports the `java.io` package and creates a `File` object.

2 To create the new file, type the name of the `File` object followed by a dot. Then type **createNewFile**().

3 To create an output stream to write to the file, type **FileOutputStream** followed by a name for the `FileOutputStream` object.

4 Type **= new FileOutputStream**().

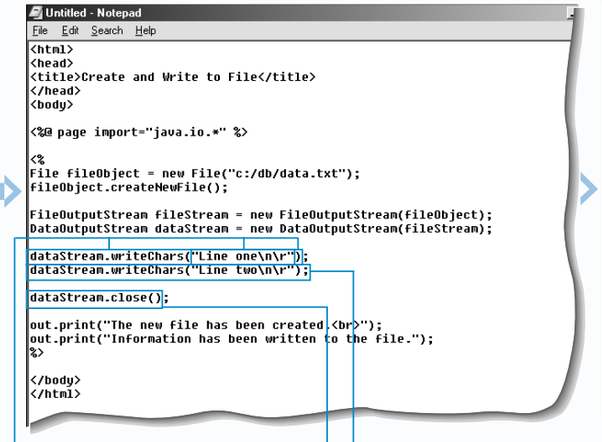
5 Between the parentheses, type the name of the `File` object.



6 To write primitive data types to the file, type **DataOutputStream** followed by a name for the `DataOutputStream` object.

7 To associate the `DataOutputStream` object with the `FileOutputStream` object, type **= new DataOutputStream**().

8 Between the parentheses, type the name of the `FileOutputStream` object.

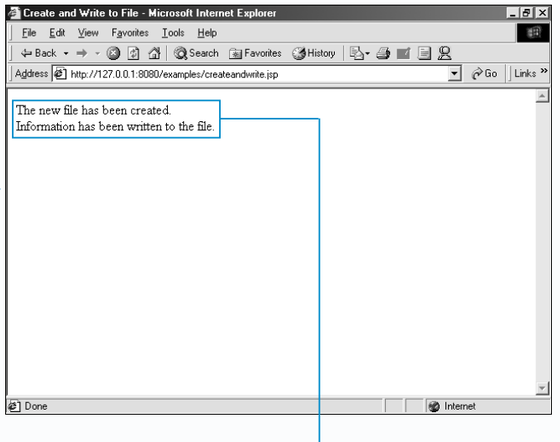


9 To write information to the file, type the name of the `DataOutputStream` object followed by a dot. Then type the write method you want to use followed by ().

10 Between the parentheses, type the information you want to write to the file.

11 Repeat steps 9 and 10 until you have specified all the information you want to write to the file.

12 To close the output stream, type the name of the `DataOutputStream` object followed by a dot. Then type **close**().



13 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The new file is created in the specified directory. You can open the file with the appropriate program to view its contents or use a JSP page to read the file.

READ A FILE

A JSP page can be used to read information from a specific file. The first step in reading information from a file is to create a `File` object that is used to specify the path and the name of the file to be read. Once a `File` object has been created, a `FileReader` object that works with the `File` object must be created. The `FileReader` object is used to convert the information in the file and make it available to the JSP page.

When reading information from a file using a `FileReader` object, the information should be buffered so that it can be read more efficiently. A `BufferedReader` object is used to buffer the information read from a file. For more information about the `BufferedReader` and `FileReader` objects, refer to the `java.io` package information in the Java API specification.

The `readLine` method of the `BufferedReader` object is used to read a single line from a file. The newline character usually indicates the end of a line in a file. A loop is often used to process each line in a file. With each iteration of the loop, the information retrieved from the file using the `readLine` method can be assigned to a variable and displayed to the client using the `print` method of the `out` object.

After reading information from a file, you should close the file using the `close` method of the `FileReader` object.

As with other operations involving accessing a file, the proper permissions must be in place that allows the file to be read. Permissions are typically controlled by the operating system. For information about permissions, you should consult your operating system's documentation.

Extra

You can adjust the size of the input buffer used to process the character stream that is read from a file. The default input buffer size is determined by the Web server and may differ from one system to another. For example, on a Windows platform using the Tomcat Web server, the default input buffer size is typically 512 KB, which is adequate for most needs. However, depending on the size and configuration of the files that are being read, adjusting the size of the input buffer may improve efficiency.

Example:

```
File fileObject = new File("c:/db/data.txt");
FileReader fileRead = new FileReader(fileObject);
BufferedReader buffFileIn = new BufferedReader(fileRead, 1024);
```

If the `FileReader` object does not already exist, you can pass the object creation code for the `FileReader` object as an argument when creating the `BufferedReader` object.

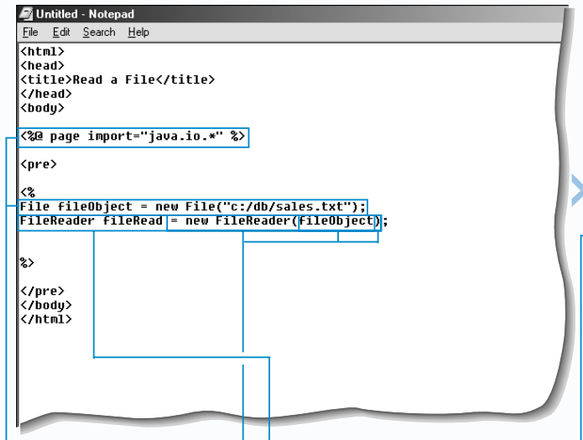
Example:

```
File fileObject = new File("c:/db/data.txt");
FileReader fileRead = new FileReader(fileObject);
BufferedReader buffFileIn = new BufferedReader(fileRead);
```

Can be typed as:

```
File fileObject = new File("c:/db/data.txt");
BufferedReader buffFileIn = new BufferedReader(new FileReader(fileObject));
```

READ A FILE

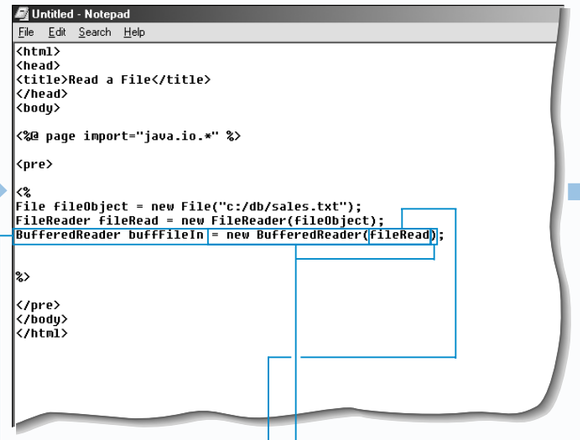


1 Type the code that imports the `java.io` package and creates a `File` object.

2 To create a `FileReader` object to make the information in the file available to the JSP page, type **FileReader** followed by a name for the `FileReader` object.

3 Type **= new FileReader()**.

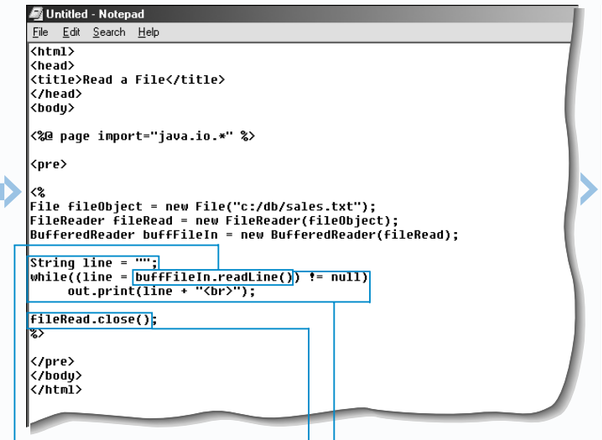
4 Between the parentheses, type the name of the `File` object.



5 To create a `BufferedReader` object to buffer the information read from the file, type **BufferedReader** followed by a name for the `BufferedReader` object.

6 Type **= new BufferedReader()**.

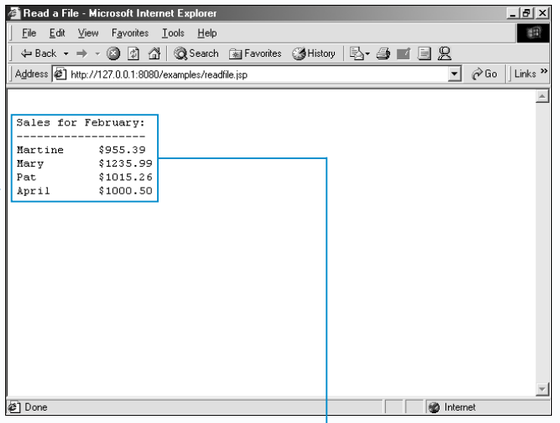
7 Between the parentheses, type the name of the `FileReader` object.



8 To read a line from the file, type the name of the `BufferedReader` object followed by **.readLine()**.

9 Type the code that will use each line of data read from the file.

10 To close the file, type the name of the `FileReader` object followed by **.close()**.



11 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the result of reading a file.

READ A FILE RANDOMLY

A JSP page typically reads and processes a file one line at a time until the entire file is processed. This method of reading a file is referred to as sequential access and is an effective way of working with small text files, but can be inefficient when working with larger files.

You can access a specific area of a file without having to start at the beginning and read each line in the file. Accessing a file at a specific location is referred to as random access. Random access is useful for working with large files that have a set structure, such as files that have the same number of characters in every line.

Once a `File` object that specifies the path and the name of the file to be accessed randomly has been created, a `RandomAccessFile` object must be created. A `RandomAccessFile` object is used to read a file randomly and requires two arguments. The first argument is the name of the `File` object. The second argument is the access mode. Specifying a value of `r` for the access mode indicates the file is read only.

Random access is achieved by positioning an imaginary pointer in the file. The pointer location is measured by the number of bytes the pointer is from the beginning of the file. This distance is known as the offset. The `seek` method of the `RandomAccessFile` object is used to position the pointer. Using the `seek` method, the pointer can be moved forward or backward through a file. When positioning the pointer, you should keep in mind that the carriage return character and the newline character each count as one byte.

When using random access to read data from a file, information is read starting from the location of the pointer. For example, if the `seek` method is set to 13, the data starting at the 13th byte in the file will be read. You can use the `readLine` method to read data up to the next newline character.

Extra

Before you start accessing a file randomly, you may want to determine the length of the file. You can determine the length of a file by using the `length` method of the `RandomAccessFile` object.

Example:

```
The length of the file is 
<%
File fileObject = new File("c:/db/names.txt");
RandomAccessFile myFile = new RandomAccessFile(fileObject, "r");
out.print(myFile.length());
%>
 bytes.
```

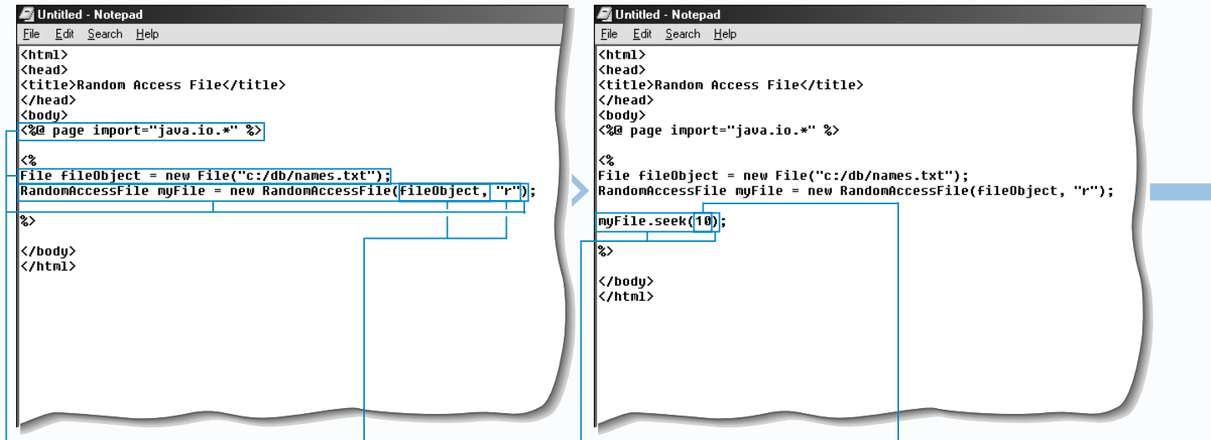
You can also use the `RandomAccessFile` object to write data to a file. To be able to read and write to a file, you must specify an access mode of `rw` when creating the `RandomAccessFile` object.

Example:

```
<%
File fileObject = new File("c:/db/names.txt");
RandomAccessFile myFile = new RandomAccessFile(fileObject, "rw");
myFile.seek(40);
myFile.writeBytes("Barry...");
%>
```

The `RandomAccessFile` class is part of the `java.io` package. You can refer to the Java SDK documentation for more information about the `java.io` package and the methods of the `RandomAccessFile` object.

READ A FILE RANDOMLY



1. Type the code that imports the `java.io` package and creates a `File` object.
2. To create a `RandomAccessFile` object to read the file randomly, type **RandomAccessFile** followed by a name for the `RandomAccessFile` object. Then type **= new RandomAccessFile()**.
3. Between the parentheses, type the name of the `File` object followed by a comma.
4. To specify the access mode is read only, type **r** enclosed in quotation marks.
5. To position the pointer where you want to start reading the file, type the name of the `RandomAccessFile` object followed by a dot. Then type **seek()**.
6. Between the parentheses, type the number of bytes from the beginning of the file where you want to start reading.
7. To read up to the next newline character, type the name of the `RandomAccessFile` object followed by a dot. Then type **readLine()**.
8. Type the code that will use a line of data read from the file.
9. Repeat steps 5 to 8 for each line of data you want to read.
10. To close the file, type the name of the `RandomAccessFile` object followed by a dot. Then type **close()**.
11. Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the result of reading a file randomly.

CREATE A DIRECTORY

JavaServer Pages allows you to create a directory from within a JSP page. You may want to create a directory in a JSP page to help organize files or to store temporary files that will be used by the JSP page.

To create a directory, you must create a `File` object that specifies the name of the directory you want to create. In this case, the `File` object represents a directory, not a file. The name of the new directory is included as the argument of the `File` object. You may want to store the name of the directory in a variable and then use the variable as the argument for the `File` object.

Once the `File` object has been created, you use the `mkdir` method to create the directory. The `mkdir` method will return a `boolean` value of `true` or `false`, depending on whether or not the command to create the directory was successful. You may not be able to use the `mkdir` command to create directories if proper permissions are not in place.

You must have permission to access the parent directory in which you want to create the new directory. You must also have permission to create directories in the JSP page. Permission to create directories is usually controlled by the operating system. For information about access permissions, you should consult your operating system's documentation.

After you create a directory, you can create files and store them in the new directory. For information about creating files, see page 188.

It is good programming practice to verify that a directory was created successfully before using the directory. For information about verifying that a directory exists, see page 186.

Apply It

You can delete a directory you no longer need. This is useful if you frequently create and use temporary directories within your JSP pages. You cannot delete a directory from within a JSP page if the directory contains files. Before deleting a directory, you should remove all the files in the directory.

TYPE THIS:

```
<%!
String dirName = "c:/databases";
%>
<%
File dirObject = new File(dirName);
dirObject.delete();

if (!dirObject.exists())
    out.print("The directory " + dirName
    + " has been deleted.");
%>
```

RESULT:

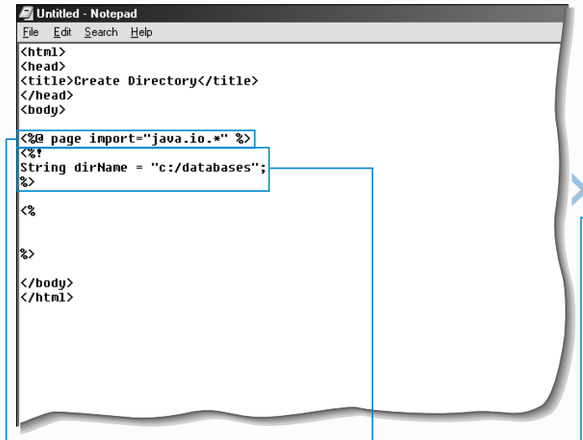
The directory c:/databases has been deleted.

Using the `mkdirs` method instead of the `mkdir` method allows you to create multiple directories at the same time. When you use the `mkdirs` method to create a directory, any directories you specify in the path will also be created if they do not currently exist. For example, if the path specified for the directory is /temp/data and the temp directory does not exist, the Web server will create the temp directory and then the data subdirectory.

TYPE THIS:

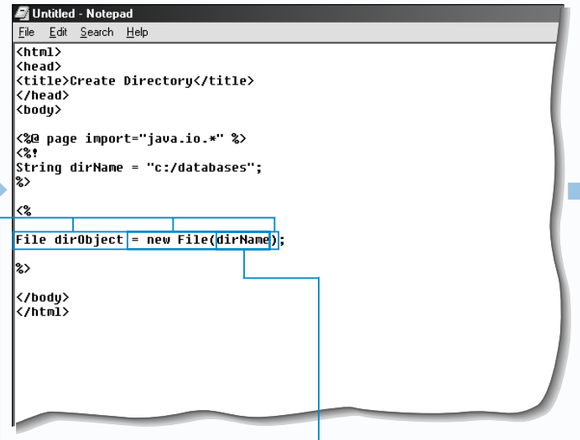
```
File dirObject= new File("/temp/data");
dirObject.mkdirs();
```

CREATE A DIRECTORY



1 To import the `java.io` package, type **<%@ page import="java.io.*" %>**.

2 To store the name or path of the directory you want to create in a variable, type the code that assigns the information to the variable.

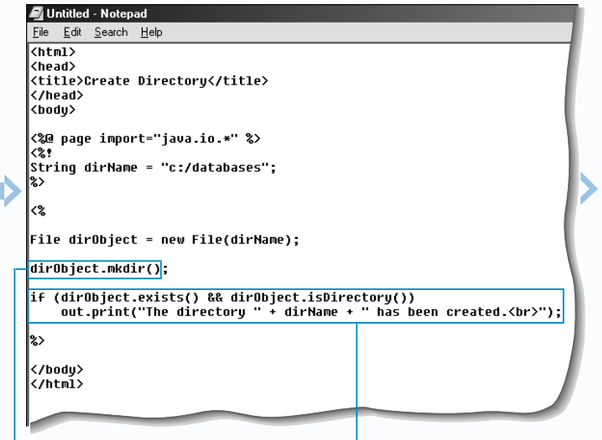


3 To create a `File` object for the directory you want to create, type **File** followed by a name for the `File` object.

4 Type **= new File()**.

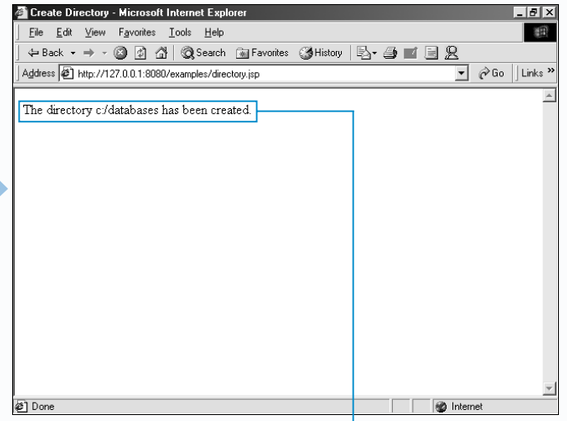
5 Between the parentheses, type the name of the variable that stores the name of the directory you want to create.

■ You can also type the path or name of the directory, enclosed in quotation marks.



6 To create the method that will create the directory, type the name of the `File` object followed by a dot. Then type **mkdir()**.

7 Type the code that will verify whether the directory was created.



8 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the results of creating a directory.

DISPLAY A DIRECTORY LISTING

A JSP page can be used to examine a directory and retrieve the names of the files and subdirectories stored in the directory. Displaying a directory listing is useful when you want to verify that certain support files, such as database files, exist before a JSP page continues processing.

To retrieve the names of the files and subdirectories in a directory, you first create a `File` object that represents the directory. The directory must be accessible from the computer you use to create the `File` object.

The directory you specify as the argument for the `File` object must be a valid directory. If the directory does not exist, an error may be generated when the JSP page is displayed. You may want to use the `exists` method of the `File` object to verify that a directory exists before attempting to display the contents of the directory. For more information about the `exists` method, see page 186.

To retrieve the names of the files and subdirectories stored in the directory, you use the `listFiles` method of the `File` object. The `listFiles` method returns an array of `File` objects that represent the files and subdirectories in the directory. You can then use a `for` loop to display the contents of each `File` object on the JSP page. The path for the files and subdirectories in the directory will be displayed.

The directory listing you display may not contain all the files in the directory, since the `listFiles` method will not return files that the JSP page does not have permission to access. If permissions have been set that prevent the JSP page from reading or listing a file, the file will not appear in the directory listing.

Extra

While working on a computer connected to a network, you may want to access a directory located on another computer on the network and display the contents of the directory in a directory listing. On a Microsoft Windows network, the convention for indicating a computer within a path is to prefix the computer name with two backslashes (\\). Since you must escape backslashes you use in the argument of a `File` object, you must use four backslashes when specifying the computer name. You must also escape backslashes you use before directory names on a Windows network.

Example:

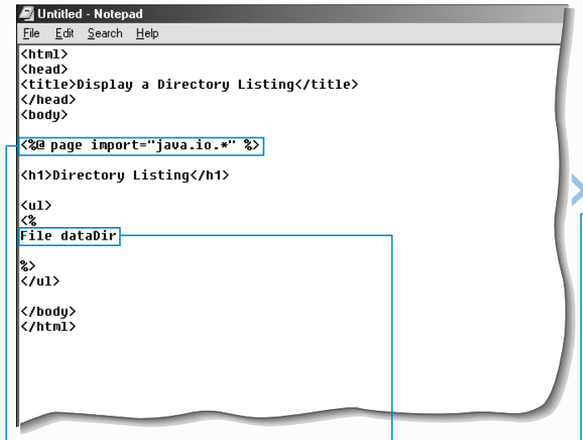
```
File dataDir = new File("\\\\Server\\data");
```

You can delete a file you no longer need from a directory. Deleting files allows you to free up resources on a computer and is useful when you want to delete a directory, since all the files in a directory must be deleted before the directory can be removed. To delete a file, create a `File` object for the file and then use the `delete` method of the `File` object to delete the file.

Example:

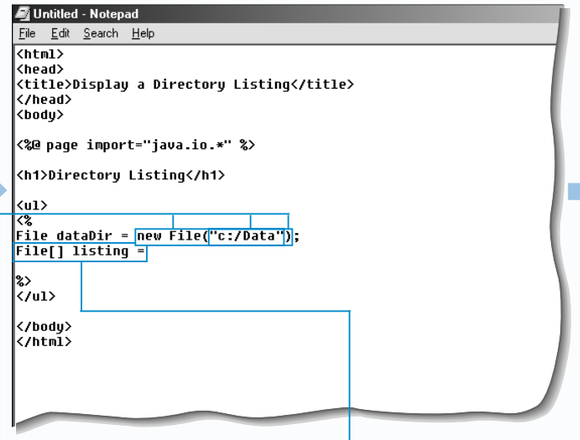
```
File fileObject = new File("c:/Data/file.txt");
fileObject.delete();
```

DISPLAY A DIRECTORY LISTING



1 To import the `java.io` package, type **<%@ page import = "java.io.*" %>**.

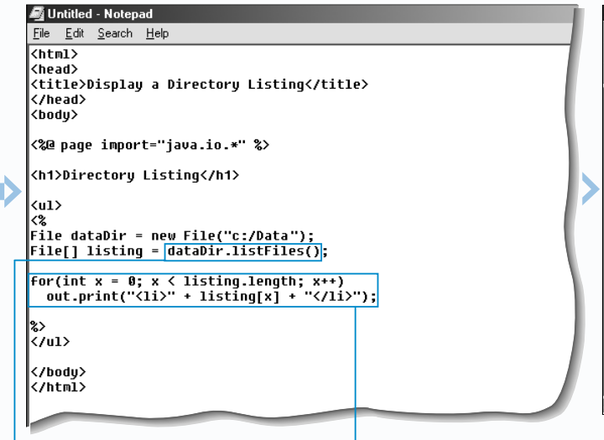
2 To create a `File` object for the directory you want to display a directory listing for, type **File** followed by a name for the `File` object.



3 Type **= new File()**.

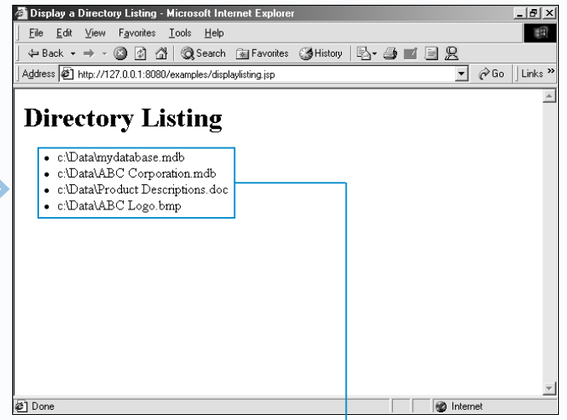
4 Between the parentheses, type the path of the directory you want to display a directory listing for, enclosed in quotation marks.

5 To create an array that will store the files and subdirectories in the directory as an array of `File` objects, type **File[]** followed by a name for the array. Then type **=**.



6 To retrieve the files and subdirectories in the directory, type the name of the `File` object followed by a dot. Then type **listFiles()**.

7 Type the code that creates a `for` loop that will process the elements in the array.



8 Save the page with the .jsp extension and then display the JSP page in a Web browser.

■ The Web browser displays the results of using the `listFiles` method to display a directory listing.