CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

File

Overview

1. Introduction
2. File class declaration
3. File constructors
4. File methods
5. File examples

Introduction

- So far, we have used variables and arrays to store data inside Java programs.

  This approach has two problems:
  1. Data stored in a program is lost when the program is terminated. It means storage is temporary.
  2. Large volumes of data are difficult to handle using variables and arrays.

- To overcome these problems, we use secondary storage devices called hard disks to store large volumes of data.

- The data is stored in the hard disk of a computer by using the concept of files. Data stored in files on a disk is called persistent data. Persistent means lasting.

What is File in Java?

- A **file** is a collection of related records located in a specific area on the disk that stores user and system data. A record consists of many fields.
- A field contains a group of Unicode characters composed of two bytes, each byte containing eight binary digits, 1 or 0.
- In a system, files are organized using directories. A **directory in Java** is a file within a file system that contains several files and other directories.
- A file system is a hierarchy of files and directories, starting from a root directory.
- The root directory is the topmost directory in the file system, from which all files and directories derive. In Windows, it is represented by a drive name c:\, while on Linux it is represented by a single forward slash /.
- In Java, files are used by several applications for storing and retrieving data. Storing and retrieving data using files is called file processing. Java supports the input and output of data through files.

Introducing File Class in Java

The `java.io` package provides a class known as **File class** that provides some methods to know the properties of files or directories.

The `File` class in Java provides several methods to perform operations, such as:

- Creating a file.
- Opening a file.
- Deleting a file.
- Closing a file.
- Getting the file name.
- Getting the size of a file.
- Checking the existence of a file.
- Renaming a file.
- Testing whether a file is writable or not.
- Testing whether a file is readable or not.

File class declaration

`File` class in Java is an abstract representation of file and directory pathnames. It extends Object class and implements Serializable and Comparable<File> interfaces.

The general syntax to declare file class in java is as follows:

```
public class File
       extends Object
       implements Serializable, Comparable<File>
```

`File` class was added in JDK 1.0 version. It is present in the `java.io.File` package.

File Constructors

Java File class provides the following constructors to create File objects. They are as follows:

---

1. File(String pathName):
This constructor creates a `File` object with the specified `pathname`. The `pathname` may be a file or a directory.
It converts the given `pathname` string into an abstract pathname.

---

2. File(String directoryPath, String filename):
This constructor creates a new `File` instance from a directory with the specified directory path and filename.

---

3. File(File dirObject, String filename):
This constructor creates a new `File` instance from a `File` object `dirObject` that specifies a directory and a file having the name `filename`.

---

4. File(URI uriObj):
This constructor creates a new `File` instance from a URI object `uriObj` that specifies a file.

File Constructors

Note:

a. A **path** is a hierarchy of directories that locate a file or directory.

b. A **pathname** is a string representation of the path in which a platform-dependent separator character (/) appears between consecutive names.

c. A **filename** is a unique string of characters that locate a file on the disk. It may consist of two parts: primary name and an optional period with extension. For example, myfile.dat, student.txt, salary, etc.

d. Java `File` class does not contain any methods for reading and writing data in the file.

e. We must know the purpose of using a file. For example, we should know whether the file is built for reading only, or writing only, or both operations.

Create File object in Java

An object of a `File` class signifies the pathname of a particular file or directory on the file system. It is initialized as string that contains either an absolute or relative path to the file or directory within the file system.

Note:
Before covering the methods of Java File class, you should understand the basic concepts of abstract, absolute, relative, and canonical path names.

Create File object in Java

a) The general syntax to create an object of `File` class by passing the filename or directory name:

```
//Here, the filename is a string of characters that identify a file on the disk.
File file = new File(filename);

//For example:
// Creating a File object for file.
File file = new File("myfile.txt");                    // Assigning the filename to the file object.

// Creating a File object for directory /book.
File file = new File("/book");

// Creating a File object of file /book/myfile.dat.
File file = new File("/book/myfile.dat");
```

Create File object in Java

b) The general syntax to create an object of `File` class by passing pathname to it is as follows:

```
File file = new File(String pathName);

//For example:
File file1 = new File("/");                 // Passing directory path.
File file2 = new File("/myfile.txt");       // Passing path of a file.
```

Create File object in Java

c) The general syntax to create an object of `File` class by passing pathname and filename to it as follows:

```
File file = new File(String pathName, String filename);

//For example:
File file = new File("/User/Documents/Projects/", "myfile.txt");
```

File Methods

`File` class in Java provides various useful methods for obtaining the properties of a file/directory and for renaming and deleting a file/directory within the file system.

The most commonly important methods of `File` class are as follows:

Note:

The `File` class in Java has a lot of methods that perform a variety of functions. But it does not provide methods for reading and writing the file contents.

File Methods

Method	Description
boolean exists()	Tests whether the file or directory represented by the File object exists. It returns true if exists.
boolean canExecute()	Tests whether the application can execute the file represented by File object. It returns true if executes.
boolean canRead()	This method returns true if the file represented by the File object exists and can be read.
boolean canWrite()	This method returns true if the file denoted by the File object exists and can be written.
boolean isAbsolute()	This method returns true if the File object is created using an absolute pathname. In other words, it checks whether this abstract pathname is absolute.
boolean isDirectory()	This method returns true if the File object represents a directory. That is, it checks whether the file represented by the abstract pathname is a directory.
boolean isFile()	This method returns true if the File object represents a file. That is, it checks whether the file represented by the abstract pathname is a normal file.
boolean isHidden()	This method returns true if the file represented in the File object is hidden. In other words, it checks whether the file named by this abstract pathname is a hidden file.

File Methods

Method	Description
boolean createNewFile()	It basically constructs a new, empty file named by the abstract pathname if and only if a file with this name does not yet exist.
boolean delete()	This method deletes the file or directory represented by this File object. It returns true if the deletion succeeds.
File getAbsoluteFile()	This method returns the absolute form of the abstract pathname.
String getAbsolutePath()	This method returns the absolute pathname string of the abstract pathname.
File getCanonicalFile()	It returns the canonical form of the abstract pathname.
String getCanonicalPath()	It returns the canonical pathname string of this abstract pathname.
long getFreeSpace()	This method returns the number of unallocated bytes in the partition named by this abstract path name.

File Methods

Method	Description
String getName()	This method returns the last name of the complete directory and file name represented by the File object. For example, new File(“c:\\book\\myfile.dat”).getName() returns myfile.dat.
String getPath()	This method returns the complete directory and file name represented by the File object. For example, new File(“c:\\book\\myfile.dat”).getPath() returns c:\book\myfile.dat.
String getParent()	This method returns the complete parent directory of the current directory or the file represented by the File object. For example, new File(“c:\\book\\myfile.dat”).getParent() returns c:\book.
long getTotalSpace()	This method returns the size of the partition defined by the abstract pathname.
long getUsableSpace()	It returns the number of bytes available to this virtual machine on the partition specified by the abstract pathname.

File Methods

Method	Description
long lastModified()	This method returns the time that the file was last modified.
long length()	It returns the length (size) of the file or 0 if it does not exist or if it is a directory.
String[ ] list()	It returns an array of strings naming the files and directories in the directory represented by the File object.
File[ ] listFiles()	It returns the files under the directory for a directory File object
static File[ ] listRoots()	It list the available file system roots.

File Methods

Method	Description
boolean mkdir()	It creates a directory represented in this File object. It returns true if the directory is created successfully.
boolean mkdirs()	It is the same as mkdir() except that it creates a directory along with its parent directories if the parent directories do not exist.
boolean renameTo(File dest)	The renameTo() method is used to rename the file or directory represented by this File object to the specified name represented in dest. It returns true if the operation succeeds.
URI toURI()	This method creates a file: URI that represents the abstract pathname
boolean setReadOnly()	The setReadOnly() method is used to mark the file or directory as read-only. This method was added in Java 1.2 version to File class.

File Methods

Method	Description
boolean setExecutable(boolean executable)	It is a convenient method to set the execute permission for the owner.
boolean setExecutable(boolean executable, boolean ownerOnly)	It is used to set the execute permission for owner or everybody.
boolean setReadable(boolean readable)	It is a convenient method to set the read permission for the owner.
boolean setReadable(boolean readable, boolean ownerOnly)	It sets the read permission for the owner or everybody.
boolean setWritable(boolean writable)	It is a convenient method to set the write permission for the owner.
boolean setWritable(boolean writable, boolean ownerOnly)	It sets the write permission for the owner or everybody.

Example 1: Check file existence

1. Let’s take an example program where we will check the existence of a file.

Example 1: Check file existence

```
import java.io.File;
public class FileTester1 {
   public static void main(String[] args) {
       // Create an object of File class and pass the path of filename.
       File file1 = new File("./src/myfile.txt");      //exists
       File file2 = new File("myfile.txt");            //not exists

       // Check for the existence of file.
       System.out.println("file1 exists? " + file1.exists() );
       System.out.println("file2 exists? " + file2.exists() );
   }
}
```

Example 1: Check file existence

Output:

```
file1 exists? true
file2 exists? false
```

Note:

Escape sequences (\\) may be included to separate the drive, folder, and file names in the path of file.

Example 2: Display file properties

2. Let’s create a program where we will check the existence of a file and display the properties of that file. Look at the following source code.

Example 2: Display file properties

```
import java.io.File;
import java.io.IOException;
public class FileTester2 {
   public static void main(String[] args) throws IOException {
       // Create an object of File class and pass path of filename.
       File file = new File("./src/myfile.txt");

       // Use File class methods on File object.
       System.out.println("Does myfile.txt exist? " + file.exists());
       System.out.println("File name: " +file.getName());
       System.out.println("File size in bytes: " +file.length());

       System.out.println("Path: " +file.getPath());
       System.out.println("Absolute path: " +file.getAbsolutePath());
       System.out.println("Canonical path: " +file.getCanonicalPath());

       System.out.println("Parent: " +file.getParent());
       System.out.println("Free space: " +file.getFreeSpace());

       System.out.println("Is myfile.txt a file? " + file.isFile());
       System.out.println("Is myfile.txt a directory? " + file.isDirectory());
       System.out.println("Is myfile.txt hidden? " + file.isHidden());

       System.out.println("Can myfile.txt be read? " + file.canRead());
       System.out.println("Can myfile.txt be written? " + file.canWrite());
       System.out.println("Last modified on " +new java.util.Date(file.lastModified()));
   }
}
```

Example 2: Display file properties

Output:

```
Path: ./src/myfile.txt
Absolute path: /Users/ted/IdeaProjects/Lecture23-Files/./src/myfile.txt
Canonical path: /Users/ted/IdeaProjects/Lecture23-Files/src/myfile.txt
Parent: ./src
Free space: 8356220928
Is myfile.txt a file? true
Is myfile.txt a directory? false
Is myfile.txt hidden? false
Can myfile.txt be read? true
Can myfile.txt be written? true
Last modified on Wed Jul 13 16:59:27 CDT 2022
```

Explanation:

a. This program creates a `File` object and prints file properties.

b. The `lastModified()` method of Java `File` class returns the date and time when the file was last modified.

c. The `Date` class is used to print it on the console in a readable format.

END