

# File Handling

Nonika Sharma, Priyanka Sahni

Information Technology, Dronacharya College Of Engineering, Gurgaon, India

nonikasharma1@gmail.com, Priyanka.sahni@yahoo.com

## ABSTRACT-

The *Stack* tool provides a raw data reader. *Stereo* and *Mono* tools perform generic file handling and display for stereo and mono processing respectively. The *Sequence* tool also provides readers for medical image data sets. Files can be flexibly converted directly or by interchanging data via the stack. See appendix for file formats.

The simplest image handler is the *Raw Input Tool* which provides direct access to the data stack within Tina. It allows the user to specify all of the parameters necessary to locate and read a raw binary (or ascii) data block within a file. The parameters of this tool should be quite self explanatory. Typical use involves a trial and error process to select the best parameters while viewing the resulting data in the **Imcalc Tool's** TV. This should be the method of last resort for loading proprietary data sets. The other data input tools support formats which contain useful supplemental header information. The *Sequence Tool* in particular supports common medical volume data sets. Once an data set has been loaded it is possible to write out to any of the supported formats, thus achieving image conversion.

## Introduction

File is a collection of bytes stored in secondary storage device i.e. disk. Thus,

File handling is used to read, write, append or update a file without directly opening it.

Types of File:

- Text File
- Binary File

Text File contains only textual data. Text files may be saved in either a plain text (.TXT) format and rich text (.RTF) format like files in our Notepad while Binary Files contains both textual data and custom binary data like font size, text color and text style etc.

## Why we use File Handling?

The input and output operation that we have performed so far were done through screen and keyboard only. After the termination of program all the entered data is lost because primary memory is volatile. If the data has to be used later, then it becomes necessary to keep it in permanent storage device. So the Java language provides the concept of file through which data can be stored on the disk or secondary storage device. The stored data can be read whenever required.

**Note:** For handling files in java we have to import package named as *java.io* which contains all the required classes needed to perform input and output (I/O) in Java.

## File Class in Java:

Files and directories are accessed and manipulated by the File class. The File class does not actually provide for input and output to files. It simply provides an identifier of files and directories.

**Note:** Always remember that just because a File object is created, it does not mean there actually exists on the disk a file with the identifier held by that File object.

For Defining a file in a File Class there are several types of constructors.

## File Class constructors:

Constructor	Description
File(File parent, String child)	This method creates a new File instance from a parent abstract pathname and a child pathname string.
File(String pathname)	This method creates a new File instance by converting the given pathname string into an abstract pathname
File(String parent, String child)	This method creates a new File instance from a parent pathname string and a child pathname string.
File(String parent, String child)	This method Creates a new File instance by converting the given file: URI into an abstract pathname.

## File class methods:

Method	Description
Boolean canExecute()	This method tests whether the application can execute the file denoted by this abstract pathname.
Boolean canRead()	This method tests whether the application can read the file denoted by this abstract pathname.
Boolean canWrite()	This method tests whether the application can modify the file denoted by this abstract pathname.
int compareTo (File pathname)	This method compares two abstract pathnames.
Boolean createNewFile()	This method atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.

Boolean delete()	This method deletes the file or directory denoted by this abstract pathname.
long getFreeSpace()	This method returns the number of unallocated bytes in the partition named by this abstract path name.
String getName()	This method returns the name of the file or directory denoted by this abstract pathname.
String getParent()	This method returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
File getParentFile()	This method returns the abstract pathname of this abstract pathname's parent, or null if this pathname does not name a parent directory.
String getPath()	This method converts this abstract pathname into a pathname string.
String toString()	This method returns the pathname string of this abstract pathname.
Boolean mkdir()	This method creates the directory named by this abstract pathname.
long getTotalSpace()	This method returns the size of the partition named by this abstract pathname.
long getUsableSpace()	This method returns the number of bytes available to this virtual machine on the partition named by this abstract pathname.
int hashCode()	This method computes a hash code for this abstract pathname.
Boolean isAbsolute()	This method tests whether this abstract pathname is absolute.
Boolean isDirectory()	This method tests whether the file denoted by this abstract pathname is a directory.
Boolean isFile()	This method tests whether the file denoted by this abstract pathname is a normal file.
Boolean isHidden()	This method tests whether the file named by this abstract pathname is a hidden file.
long lastModified()	This method returns the time that the file denoted by this abstract pathname was last modified

long length()	This method returns the length of the file denoted by this abstract pathname.
String[] list()	This method returns an array of strings naming the files and directories in the directory denoted by this abstract pathname
String[] list(FilenameFilter filter)	This method returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
File[] listFiles()	This method returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.

### Listing 1: Check Permission on a File

```
import java.io.File;

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

        File f = null;
        String[] str = {"test.txt", "/test.txt"};
        try{
            // for each string in string array
            for(String s:str )
            {
                // create new file
                f= new File(s);

                // true if the file is executable
                Boolean bool = f.canExecute();

                // find the absolute path
                String a = f.getAbsolutePath();

                // prints absolute path
                System.out.println(a);

                // prints
                System.out.println(" is executable: "+ bool);
            }
        }
    }
}
```

```
// returns true if the file can be read
Boolean w = f.canWrite();

// print
System.out.println("File can be writing: "+w);

// returns true if the file can be read
Boolean r = f.canRead();

// print
System.out.println("File can be read: "+r);

    }
} catch (Exception e) {
    // if any I/O error occurs
    e.printStackTrace();
}
}
```

Output of the program is:

C:\test.txt	is	executable:	True
File	can	be	False
File can be read:	True	writing:	

#### **Listing 2:** Program to Create and Delete a file

```
import java.io.File;

Public class FileDemo {
    Public static void main (String[] args) {

        File f = null;
        Boolean bool = false;

        try {
            // create new file
            f = new File ("test.txt");

            // tries to delete a non-existing file
            bool = f.delete();
        }
    }
}
```

```
// prints
System.out.println ("File deleted: "+bool);

// creates file in the system
f.createNewFile ();

// createNewFile () is invoked
System.out.println ("createNewFile() method is invoked");

// tries to delete the newly created file
bool = f.delete();

// print
System.out.println ("File deleted: "+bool);

}
catch(Exception e){
    // if any error occurs
    e.printStackTrace ();
}
}
```

Output of the Program is:

File	deleted:	false
createNewFile()	method	is invoked
File deleted: true		

### **Listing 3:** Program to Compare Two Files

```
import java.io.File;

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

        File f = null;
        File f1 = null;

        try{
            // create new files
            f = new File("test.txt");
            f1 = new File("File/test1.txt");
```

```
// returns integer value
int value = f.compareTo(f1);

// argument = abstract path name
if(value == 0)
{
    System.out.println (" Both Files are Equal. ");
}

// argument < abstract path name
else if(value > 0)
{
    System.out.println ("First file is greater.");
}

// the argument > abstract path name
else
{
    System.out.println ("Second file is greater.");
}

// prints the value returned by compareTo()
System.out.println("Value returned: "+value);

} catch(Exception e){
    e.printStackTrace();
}
}
```

Output of the Program is:

First file is greater.  
Value returned: 14

**Listing 4:** Program to check whether it is a File or a Directory and check it is a hidden file

```
import java.io.File;

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

```
File f = null;
String path;
boolean bool = false;

try{
    // create new file
    f = new File("c.txt");

    // true if the file path is a file, else false
    bool = f.isFile();

    // get the path
    path = f.getPath();

    // prints
    System.out.println (path+" is file? "+ bool);

    // create new file
    f = new File("c:/test.txt");

    // true if the file path is a file, else false
    p = f.isDirectory();

    // get the path
    path = f.getPath();

    // prints
    System.out.println (path+" is Directory? "+p);

    // create new file
    f = new File("c:/test.txt");

    // true if the file path is a file, else false
    h = f.isHidden();

    // get the path
    path = f.getPath();

    // prints
    System.out.println (path+" is Hidden? "+h);
```



```

    }catch(Exception e){
        // if any error occurs
        e.printStackTrace();
    }
}

```

Output of the Program is:

C.txt	is	File?	False
c:\test.txt	is	Directory?	True
C:\test.txt is hidden? False			

## Conclusion

The java.io package contains many classes that our programs can use to read and write data. The java.io.File package provides extensive support for file and file system I/O. This is a very comprehensive API, but the key entry points are as follows:

- The Path class has methods for manipulating a path.
- The Files class has methods for file operations, such as moving, copy, deleting, and also methods for retrieving and setting file attributes.
- The File System class has a variety of methods for obtaining information about the file system.

## References:

- [1] McIlroy, M. D., & Reeds, J. A. (1991). *U.S. Patent No. 4,984,272*. Washington, DC: U.S. Patent and Trademark Office.
- [2] Barron, D. W., Fraser, A. G., Hartley, D. F., Landy, B., & Needham, R. M. (1967, April). File handling at cambridge university. In *Proceedings of the April 18-20, 1967, spring joint computer conference* (pp. 163-167). ACM.
- [3] Yagi, T., & Takahashi, O. (2002). *U.S. Patent No. 6,393,429*.

Washington, DC: U.S. Patent and Trademark Office.

- [4] Hogg, S. (1996). A review of the validity and variability of the elevated plus-maze as an animal model of anxiety. *Pharmacology Biochemistry and Behavior*, 54(1), 21-30.

- [5] File, S. E., Andrews, N., Wu, P. Y., Zharkovsky, A., & Zangrossi Jr, H. (1992). Modification of chlordiazepoxide's behavioural and neurochemical effects by handling and plus-maze experience. *European journal of pharmacology*, 218(1), 9-14.