File

From java.io package

Read java doc api : //https://docs.oracle.com/javase/8/docs/api/java/io/File.html

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

An abstract representation of file and directory pathnames.

User interfaces and operating systems use system-dependent *pathname strings* to name files and directories. This class presents an abstract, system-independent view of hierarchical pathnames. An *abstract pathname* has two components:

1. An optional system-dependent *prefix* string, such as a disk-drive specifier, `"/"` for the UNIX root directory, or `"\\\\"` for a Microsoft Windows UNC pathname, and
2. A sequence of zero or more string *names*.

The first name in an abstract pathname may be a directory name or, in the case of Microsoft Windows UNC pathnames, a hostname. Each subsequent name in an abstract pathname denotes a directory; the last name may denote either a directory or a file. The *empty* abstract pathname has no prefix and an empty name sequence.

The conversion of a pathname string to or from an abstract pathname is inherently system-dependent. When an abstract pathname is converted into a pathname string, each name is separated from the next by a single copy of the default *separator character*. The default name-separator character is defined by the system property `file.separator`, and is made available in the public static fields `separator` and `separatorChar` of this class. When a pathname string is converted into an abstract pathname, the names within it may be separated by the default name-separator character or by any other name-separator character that is supported by the underlying system.

A pathname, whether abstract or in string form, may be either *absolute* or *relative*. An absolute pathname is complete in that no other information is required in order to locate the file that it denotes. A relative pathname, in contrast, must be interpreted in terms of information taken from some other pathname. By default the classes in the `java.io` package always resolve relative pathnames against the current user directory. This directory is named by the system property `user.dir`, and is typically the directory in which the Java virtual machine was invoked.

The *parent* of an abstract pathname may be obtained by invoking the `getParent()` method of this class and consists of the pathname's prefix and each name in the pathname's name sequence except for the last. Each directory's absolute pathname is an ancestor of any `File` object with an absolute abstract pathname which begins with the directory's absolute pathname. For example, the directory denoted by the abstract pathname `"/usr"` is an ancestor of the directory denoted by the pathname `"/usr/local/bin"`.

The prefix concept is used to handle root directories on UNIX platforms, and drive specifiers, root directories and UNC pathnames on Microsoft Windows platforms, as follows:

- For UNIX platforms, the prefix of an absolute pathname is always `"/"`. Relative pathnames have no prefix. The abstract pathname denoting the root directory has the prefix `"/"` and an empty name sequence.
- For Microsoft Windows platforms, the prefix of a pathname that contains a drive specifier consists of the drive letter followed by `":"` and possibly followed by `"\\"` if the pathname is absolute. The

prefix of a UNC pathname is "\\\\"; the hostname and the share name are the first two names in the name sequence. A relative pathname that does not specify a drive has no prefix.

Instances of this class may or may not denote an actual file-system object such as a file or a directory. If it does denote such an object then that object resides in a *partition*. A partition is an operating system-specific portion of storage for a file system. A single storage device (e.g. a physical disk-drive, flash memory, CD-ROM) may contain multiple partitions. The object, if any, will reside on the partition named by some ancestor of the absolute form of this pathname.

A file system may implement restrictions to certain operations on the actual file-system object, such as reading, writing, and executing. These restrictions are collectively known as *access permissions*. The file system may have multiple sets of access permissions on a single object. For example, one set may apply to the object's *owner*, and another may apply to all other users. The access permissions on an object may cause some methods in this class to fail.

Instances of the `File` class are immutable; that is, once created, the abstract pathname represented by a `File` object will never change.

```
package com.hdfc.file;

import java.io.File;
import java.io.IOException;
import java.util.Scanner;


/**
 *
 *   /  looks like F Forward slash
 *  \ looks like B Backword slash
 *  > look like g greater than
 *  < look like L less than
 *
 *  windows :  backword slash \\
 *  unix/linux: forward slash /
 */


//https://docs.oracle.com/javase/8/docs/api/java/io/File.html
public class FileTest {

    public static void main(String[] args) throws IOException {

        //parent pathname string and a child pathname string.
        //"C:\\Users\\sutha\\Desktop\\test" is directory
        //"Ravi2.txt" is file name

        //or
        // C:\Users\sutha\Desktop parent path
        // \test\Ravi2.txt is child path

        //File file = new File("C:\\Users\\sutha", "\\Desktop\\test\\Ravi2.txt");
        //System.out.println(file.createNewFile());
        //file.mkdir();
        //System.out.println(file.exists());//true?


        //File reprent a path for a file name
        //File file = new File("C:\\Users\\sutha\\Desktop\\test\\MyFolder110\\MyFolder120\\MyFolder130");
       // System.out.println(file.exists());
        //System.out.println(file.mkdirs()); //mkdir for create folder
        //mkdirs to create nexted folders
```

```
        //unix /Linux
        //read write execute
        /*file.canRead(); //to check if file is readable or not?
        file.canWrite();//to check if file is writable or not?
        file.canExecute();//to check if file is executable or not?

        if(true){//admin
            file.setReadable(true);
            file.setWritable(true);
            file.setExecutable(true);
        }else{
            file.setReadable(true);
            file.setWritable(false); //not writable
            file.setExecutable(false); //.sh non executable
        }*/

        //An abstract representation of file and directory pathnames.
        File file = new File("TestJava.txt"); //a file name
        File file2 = new File("C:\\Users\\sutha");//path
        file.createNewFile();
        System.out.println(file.getAbsolutePath());
        System.out.println(file.getCanonicalPath());


        String hello ="  \\Mayank\\  "; // "  \"Mayank\"  "
        System.out.println(hello);


        // Stream based-Stream (Input-read, Output-write)
        //Character based (text based)- Reader/Writer (Input-read, Output-write)


    }
}
```