Theory: Managing files

20 minutes 0 / 5 problems solved

As you already know, the `java.io.File` represents an abstract path to a file or a directory that may not even exist. Apart from just traversing the file hierarchy, we are also able to manage files and directories by creating, deleting and renaming them. Let's consider several methods for doing it. You are able to use considered methods in different operation systems, but in the following example, we work with UNIX-like OS.

§1. Creating files

To create a file in the file system, we'll have to do the following:

1. Create an instance of `java.io.File` with the specified abstract path.
2. Invoke the `createNewFile` method of this instance.

After creating an instance of `File` we should invoke the method `createNewFile`. The method returns `true` if the file was successfully created and `false` if it already exists. It does not erase the content of an existing file.

```
File file = new File("/home/username/Documents/file.txt");
try {
    boolean createdNew = file.createNewFile();
    if (createdNew) {
        System.out.println("The file was successfully created.");
    } else {
        System.out.println("The file already exists.");
    }
} catch (IOException e) {

    System.out.println("Cannot create the file: " + file.getPath());

}
```

Try to play with this code yourself for better understanding.

You may ask: "why does the method return false instead of throwing an exception when the file already exists"? The answer is that sometimes it does not matter for the program if the file was created right now or already existed.

§2. Creating directories

To create a directory we also need to start by creating an instance of `java.io.File`. After that, we should call one of the two methods of this instance:

- `boolean mkdir` creates the directory; it returns `true` only if the directory was created, otherwise, it returns `false`.
- `boolean mkdirs` creates the directory including all necessary non-existing parent directories; it returns `true` only if the directory was created along with all of the specified parent directories.

Both methods do not throw `IOException`, unlike the `createNewFile` method.

The following example demonstrates the `mkdir` method.

```
File file = new File("/home/art/Documents/dir");

boolean createdNewDirectory = file.mkdir();
if (createdNewDirectory) {
    System.out.println("It was successfully created.");
} else {
    System.out.println("It was not created.");
}
```

Current topic:

Managing files

Topic depends on:

× Exception handling Stage 7

× Files

Table of contents:

We recommend you try this code on your computer specifying different paths to the file.

Generally, the code works as follows: if the directory does not exist, this code creates it. If the directory exists, the code will not create it. If there is a non-existing parent directory in the path, the directory also will not be created. No exceptions occur in any case.

Here is another example, demonstrating the `mkdirs` method. It creates the target directory and all parent directories if they do not exist.

```
File file = new File("/home/art/Documents/dir/dir/dir");

boolean createdNewDirectory = file.mkdirs();
if (createdNewDirectory) {
    System.out.println("It was successfully created.");
} else {
    System.out.println("It was not created.");
}
```

The boolean variable is `true`, if the target directory was created, regardless of the existence of the parent directories.

§3. Removing files and directories

Now that we know how to create a directory, let's find out how to get rid of one. There is a method named `delete` to remove a file or a directory. It returns `true` if and only if the file or directory is successfully deleted, otherwise, it returns `false`. The method returns `false` if the file or directory does not exist. It is important to remember that it also returns `false` if the directory contains subdirectories or files. It means that the method will not remove a hierarchy, only a particular file or an empty directory.

```
File file = new File("/home/art/Documents/dir/dir/dir");

if (file.delete()) {
    System.out.println("It was successfully removed.");
} else {
    System.out.println("It was not removed.");
}
```

To delete a directory that is not empty, at first you have to delete all the nested files and directories. Take a look at the code below. It recursively deletes directories with their content. Note that the method assumes that the passed directory `dir` does exist. Otherwise, `children == null` and `NullPointerException` will be thrown.

```
public void deleteDirRecursively(File dir) {
    File[] children = dir.listFiles();
    for (File child : children) {
        if (child.isDirectory()) {
            deleteDirRecursively(child);
        } else {
            child.delete();
        }
    }


    dir.delete();

}
```

The method `delete` never throws an `IOException`.

There's also another method for removing files. It is called `deleteOnExit` and it removes a file or a directory when your program stops. Note that once deletion has been requested, there is no way to cancel it.

§4. Renaming and moving files and directories

Now, let's take a look at renaming files and directories.

The method `renameTo` changes the name of the file by editing it in the abstract path. It returns `true` if and only if the renaming succeeded, otherwise, `false`.

```
File file = new File("/home/art/Documents/dir/filename.txt");


boolean renamed = file.renameTo(new File("/home/art/Documents/dir/newname.txt"));
```

The same method can be used to move the file or directory from the current location to another one:

```
File file = new File("/home/art/Documents/dir/file.txt");


boolean renamed = file.renameTo(new File("/home/art/Documents/another/file.txt"));
```

Many aspects of this method's behavior remain platform-dependent. It might not be able to move a file from one filesystem to another and it might not succeed if a file with the same destination already exists. The return value should always be checked to make sure that the operation was successful.

```
File file = new File("/home/art/Documents/dir/filename.txt");
File renamedFile = new File("/home/art/Documents/dir/newname.txt");

boolean renamed = file.renameTo(renamedFile);
if (renamed) {
    System.out.println("It was successfully renamed.");
} else {
    System.out.println("It was not renamed.");
}
```

The method `renameTo` throws `NullPointerException` in a case when a destination file is null.

Most of the considered methods return false in case you don't have the permission to perform a corresponding operation: rename, move or remove files and directories. However, create method throws the java.lang.IOException in similar cases.

§5. Conclusion

We have considered a set of methods to manage files and directories: creating, deleting and renaming. These methods return a boolean value that depends on the success of the operation. Now you can manage files in the file system using Java programs!

Report a typo

131 users liked this theory. **0** didn't like it. **What about you?**

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion