File IO

AUGUST,2023

expleo

Introduction

- Prior to JDK 7, the **java.io.File** class was the entry point for all **file** and **directory operations**.
- The new file I/O API is called as **NIO.2**

Limitations of java.io.File

- Very basic file system access functionality
- Very limited set of file attributes
- Does not work well with symbolic links
- Performance issues

Introduction

Features of new API

- Works more consistently **across platforms**
- Makes it easier to write programs that gracefully **handle the failure** of file system operations
- Provides more **efficient access** to a larger set of file attributes
- Allows developers of sophisticated applications to take advantage of **platform-specific** features when absolutely necessary
- Allows support for **non-native file systems**, to be "plugged in" to the platform

File Systems, Path, and File

- File systems are **hierarchical (tree) structures**.
- Both **files** and **directories** in NIO.2 are represented by a **path**, which is the file or directory's relative or absolute location.
- **Absolute Path:** An absolute path always starts with the root element and the complete directory list required to locate the file.

  Example: C:\Users\Senthil\eclipse-workspace\Sample\src\FileStream\input.txt
- **Relative Path:** A relative path must be combined with another path in order to access a file.

  Example: Sample\src\FileStream\input.txt

File Systems, Path, and File

Packages and Classes

- **java.nio.file.Path**: Locates a file or a directory by using a system-dependent path
- **java.nio.file.Files:** Using a Path, performs operations on files and directories
- **java.nio.file.FileSystem**: Provides an interface to a file system and a factory for creating a Path and other objects that access a file system
- All the methods that access the file system throw **IOException** or **a subclass**.

Path Interface

- The **java.nio.file.Path** interface provides the **entry point** for the NIO.2 file and directory manipulation.
- To obtain a Path object, obtain an **instance of the default file system**, and then invoke the **getPath** method:

```
FileSystem fs = FileSystems.getDefault();
Path p1 = fs.getPath ("F:\Personal\Training\Example \Test.txt");
```

Path Interface

- A portion of a path can be obtained by using the **subpath method**:

```
Path subpath(int beginIndex, int endIndex);
```

- The element returned by endIndex is one less that the endIndex value.

```
Path p1 = Paths.get (" F:\Personal\Training\Example \Test.txt");
Path p2 = p1.subpath (1, 3);

OutPut: Training\Example
```

Here,
Personal 0
Training 1
Example 2
Test.txt 3

File Systems, Path, and File

The Path interface defines the **methods** used to **locate a file or a directory** in a file system.

- **To access the components of a path**
- getFileName, getParent, getRoot, getNameCount
- **To operate on a path**
- normalize, toUri, toAbsolutePath, subpath, resolve, relativize
- **To compare paths**
- startsWith, endsWith, equals

File Systems, Path, and File

```
// Java program to demonstrate the java.nio.file.Path interface  methods
import java.nio.file.Path;
import java.nio.file.Paths;
class PathMethods{
	public static void main(String args[]) {
		Path p1 = Paths.get ("C:\\Users\\Senthil\\eclipse-
workspace\\...\\Sample\\src\\FileStream\\input.txt");
		Path normalizedPath = p1.normalize();
		Path p2 = Paths.get ("C:\\Users\\Senthil\\eclipse-workspace\\Sample\\src\\FileStream\\input.txt");
		System.out.println("NormalizedPath: "+ normalizedPath);
		Path subPath = p1.subpath (1, 3);
		System.out.println("SubPath: "+ subPath);
		System.out.println("getFileName: "+ p1.getFileName());
		System.out.println("getParent: "+p1.getParent());
		System.out.println("getNameCount: "+p1.getNameCount());
```

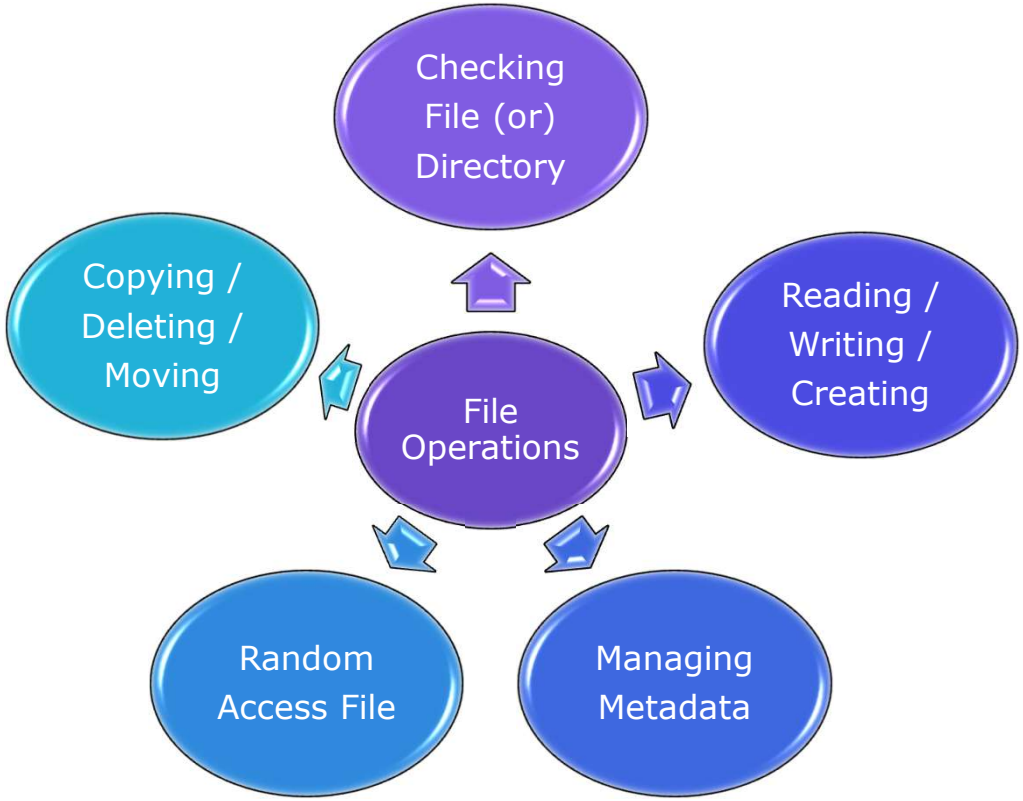
File Systems, Path, and File

```
        System.out.println("getRoot: " + p1.getRoot());
        System.out.println("isAbsolute: "+p1.isAbsolute());
        System.out.println("toAbsolutePath: "+p1.toAbsolutePath());
        System.out.println("toURI: "+p1.toUri());
        if(p1.equals(p2))
                        System.out.println("Both are equal");
            else
                        System.out.println("Both are not equal");
    }
}
```

Output:

```
NormalizedPath: C:\Users\Senthil\eclipse-workspace\Sample\src\FileStream\input.txt
SubPath: Senthil\eclipse-workspace
getFileName: input.txt
getParent: C:\Users\Senthil\eclipse-workspace\Sample\src\FileStream
getNameCount: 7
getRoot: C:\
isAbsolute: true
toAbsolutePath: C:\Users\Senthil\eclipse-workspace\Sample\src\FileStream\input.txt
toURI: file:///C:/Users/Senthil/eclipse-workspace/Sample/src/FileStream/input.txt
Both are equal
```

File Operations



Checking a File or Directory

- A Path object represents the concept of a **file or a directory location**.
- To determine whether it **exists** using the following **Files methods**:
- **exists(Path p, LinkOption... option)**

  Tests to see whether a file exists. By default, symbolic links are followed.
- **notExists(Path p, LinkOption... option)**

  Tests to see whether a file does not exist. By default, symbolic links are followed.

Example:

```
Path p = Paths.get("F:\\Personal\\Training\\Example\\Symbolic_Link\\Test.txt");
Boolean result=Files.exists(p);
System.out.println(“Path" + p + ”exists = “ + result);
```

Checking a File or Directory

- To verify that a **file can be accessed**, the Files class provides the following boolean methods.
- isReadable(Path)
- isWritable(Path)
- isExecutable(Path)
- **Example:**

```
Path p = Paths.get("F:\\Personal\\Training\\Example\\Symbolic_Link\\Test.txt");
boolean result  = Files.isReadable(p);
System.out.println("File " + p + " is Readable = " + result);
```

Creating File or Directory

- Files and directories can be created using one of the following methods:
- Files.createFile (Path dir);
- Files.createDirectory (Path dir);
- The **createDirectories** method can be used to create directories that do not exist, from top to bottom:
- **Example:**

```
Path path = Paths.get("F:\\Personal\\Training\\Example\\Symbolic_Link\\Sample.txt");
Path p= Files.createFile(path);     //creates file at specified location
System.out.println("File Created at Path: "+p);
```

Creating File or Directory

```
// Java program to demonstrate to create directory
import java.nio.file.*;
class CreateDirectory{
      public static void main(String args[]) {
           try{
            Path path = Paths.get("C:\\Users\\Senthil\\eclipse-workspace\\Sample\\src\\FileStream\\SampleDirectory");
                     if (!Files.exists(path)) {
                           Files.createDirectory(path);
                           System.out.println("Directory created");
                     } else {
                           System.out.println("Directory already exists");
                     }
              }catch (IOException e) {
                        System.out.println(e); //Exception details
              } } }
```

Creating File or Directory

```
// Java program to demonstrate to create file
import java.nio.file.*;
class CreateFile{
      public static void main(String args[]) {
          try{
                Path path1 = Paths.get("C:\\Users\\Senthil\\eclipse-workspace\\Sample\\src\\FileStream\\Sample.txt");
                  if (!Files.exists(path1)) {
                       Files.createFile(path1);
                       System.out.println("File created");
                  } else {
                            System.out.println("File already exists");
                  }
            }catch (IOException e) {
                     System.out.println(e); //Exception details
            }
    }  }
```

Deleting a File or Directory

- Can delete files or directories.
- The Files class provides **two methods:**
- delete(Path)
- deleteIfExists(Path)

    - This method deletes a **file if it exists**. It also **deletes a directory** mentioned in the path only if the directory is **not empty.**

Deleting a File or Directory

```
// Java program to demonstrate to delete a file
class DeleteFileDirectory {
      public static void main(String args[]) {
            //File to delete
            Path path = Paths.get("C:\\Users\\Senthil\\eclipse-workspace\\Sample\\src\\FileStream\\Sample.txt");
            //Directory to delete
           Path path = Paths.get("C:\\Users\\Senthil\\eclipse-workspace\\Sample\\src\\FileStream\\SampleDirectory");
       try{
          Files.deleteIfExists(path);
       }
       catch(NoSuchFileException e){
          System.out.println("No such file/directory exists");
       }
```

Deleting a File or Directory

```
        catch(DirectoryNotEmptyException e){
            System.out.println("Directory is not empty.");
        }
        catch(IOException e){
            System.out.println("Invalid permissions.");
        }

        System.out.println("Deletion successful.");
    }
}
```

Copying a File or Directory

- Copy a file or directories use **copy()** method.
- When directories are copied, the **files inside the directory are not copied**.

Synatx:

- copy(sourcePath, targetPath, copy option)

Standard copy option

1. **ATOMIC_MOVE**: Move the file as an atomic file system operation. With an ATOMIC_MOVE you can move a file into a directory and be guaranteed that any process watching the directory accesses a complete file.
2. **COPY_ATTRIBUTES**: Copy attributes to the new file.
3. **REPLACE_EXISTING**: Replace an existing file if it exists.

Copying a File or Directory

```
// Java program to demonstrate to copy a file
class CopyFileDirectory {
      public static void main(String args[]) {
            Path source = Paths.get("C:\\Users\\Senthil\\eclipse-workspace\\Sample\\src\\FileStream\\output.txt");
            Path target = Paths.get("C:\\Users\\Senthil\\eclipse-workspace\\Sample\\src\\FileStream\\Sample.txt");
           try {
               System.out.println(source+"   "+ "Copied to:"+"   "+ Files.copy(source, target,StandardCopyOption.REPLACE_EXISTING));
            }catch (IOException e) {
                 System.out.println(e); //Exception details
            }
   }
}
```

Moving a File or Directory

- Move a file or directories use **move()** method.
- Moving a directory will not move the contents of the directory.

Synatx:

- move(sourcePath, targetPath, move option)

Standard move option

1. ATOMIC_MOVE: Move the file as an atomic file system operation.
2. REPLACE_EXISTING: Replace an existing file if it exists.

Moving a File or Directory

```
// Java program to demonstrate to copy a file
class MoveFileDirectory {
      public static void main(String args[]) {
            Path source = Paths.get("C:\\Users\\Senthil\\eclipse-workspace\\Sample\\src\\FileStream\\output.txt");
           Path target = Paths.get("C:\\Users\\Senthil\\eclipse-workspace\\Sample\\src\\FileStream\\Sample.txt");
           try {
System.out.println(source+"   "+ "Moved to:"+"   "+ Files.move(source,
target,StandardCopyOption.REPLACE_EXISTING));
          } // Catch block to handle the exceptions
          catch (IOException e) {
                System.out.println(e); //Exception details
           }
   }
}
```

Managing Metadata

- Commonly used methods:

Method	Explanation
size	Returns the size of the specified file in bytes
isDirectory	Returns true if the specified Path locates a file that is a directory
isRegularFile	Returns true if the specified Path locates a file that is a regular file
isSymbolicLink	Returns true if the specified Path locates a file that is a symbolic link
isHidden	Returns true if the specified Path locates a file that is considered hidden by the file system
getLastModifiedTime	Returns or sets the specified file's last modified time
setLastModifiedTime	
getAttribute	Returns or sets the value of a file attribute
setAttribute	

Quiz

1. Which one is used to remove redundant elements?

a) normalize

b) toAbsolutePath

c) resolve

d) None of the Above

a) normalize

Quiz

2. Is java.io.file and java.nio.file are same

a) Yes

b) No

b) No

Quiz

3. Given this fragment:

```
Path source = Paths.get("path");
Path target = Paths.get("path");
Files.copy(source, target);
```

Assuming source and target are not directories, how can you prevent this copy operation from generating FileAlreadyExistsException?

a) Delete the target file before the copy.

b) Use the move method instead.

c) Use the copyExisting method instead.

d) Add the REPLACE_EXISTING option to the method

d) Add the REPLACE_EXISTING option to the method

Quiz

4. Which one is not a file operation

a) Files.createFile

b) Files.copy

c) Files.move

d) Files.modify

a) Files.modify

Quiz

5. To copy, move, or open a file or directory using NIO.2, you must first create an instance of:

a) Path

b) FileSystem

c) File

d) None of the Above

a) Path