CHAPTER 10

■ ■ ■

New Input/output 2

QUESTIONS AND EXERCISES

1. What are file systems and file stores?

Answer:

A file store provides storage for files. A file system consists of one or more file stores.

2. How do you obtain an instance of the `FileSystem` class that represents the default file system on the current platform?

Answer:

```
// Obtains a platform-specific default file system object
FileSystem fs = FileSystems.getDefault();
```

3. What is a path, an absolute path, and a relative path in a file system?

Answer:

An object in a file system has a path, typically represented as a string. A path string may contain multiple components separated by a special character called *separator* or *delimiter. If path starts with a root node, it is* an absolute path. No additional information is needed to locate an object referred in a file system by an absolute path. A relative path does not start with a root node. Additional information is needed to locate an object referred in a file system by a relative path.

4. Instances of both the File class and the Path interface represent pathnames. Differentiate between the two. How do you get a File from a Path and vice versa?

Answer:

Instances of the File class and Path interface represent abstract pathnames to files or directories. The File class is part of the legacy I/O API whereas the Path interface is part of new I/O API. The Path interface does not include any methods to perform file I/O operations.

A Path can be obtained from a File object using the toPath() method of the File class. A File object can be obtained from a Path using the toFile() method of the Path interface.

5. What is purpose of the Paths class? Write a snippet of code to get a Path instance using the Paths class to represent a file named test.txt in the current working directory.

Answer:

The Paths class is a utility class, which contains static methods to create instances of the Path interface.

The following snippet of code creates a Path for a file named text.txt in the current working directory:

```
 Path path = Paths.get("test.txt");
```

6. Write a snippet of code to print the path string of the current working directory.

Answer:

```
Path p = Paths.get("");
System.out.println("Current directory is " + p.toAbsolutePath());
```

7. What is the use of the startsWith() and endsWith() methods in the Path interface?

Answer:

The `startsWith()` method tests if the path starts with a specified path. The `endsWith()` method tests if the path ends with a specified path.

8. Suppose you have two instances of the `Path` interface named `p1` and `p2`. What is the difference in calling `p1.equals(p2)` and `Files.isSameFile(p1, p2)`?

Answer:

The `equals()` method tests for equality of two `Path` instances by comparing their string forms, without resolving the actual file references. Whether the equality test is case-sensitive depends on the file system. If `p1.equals(p2)` returns `true`, `Files.isSameFile(p1, p2)` method returns `true` without verifying the existence of the paths in the file system. Otherwise, it checks with the file system whether both paths locate the same file.

9. What is a symbolic link? How do you check if a `Path` represents a symbolic link?

Answer:

A *symbolic link, symlink,* or *soft link* is a special type of file that contains a reference to another file or directory. The `isSymbolicLink(Path p)` static method of the `Files` class can be used to check if the file denoted by the specified path is a symbolic link.

10. What methods of the `Files` class are used to create regular files and temporary files?

Answer:

The `createFile()` method of the `Files` class creates a new regular file. The `createTempFile()` method of the `Files` class creates a temporary file.

11. What is the difference between using the `delete()` and `deleteIfExists()` method of the `Files` class to delete a file?

Answer:

The `delete()` method throws a `NoSuchFileException` if the file being deleted does not exist.

The `deleteIfExists()` method does not throw a `NoSuchFileException` if the file being deleted does not exist. It returns `true` if it deletes the file. Otherwise, it returns `false`.

12. Using the NIO.2 API, how do you check if a file exists?

Answer:

The `exists()` static method of the `Files` class is used to check if a file exists.

13. What methods in the `Files` class are used to copy and move a file?

Answer:

The `copy()` and `move()` static methods of the `Files` class are used to copy and move a file, respectively.

14. Write a program that prints the creation time of a file named `test` in the current directory, changes the creation time of the file to five hours before the original time, and prints the new creation time.

Solution:

```
// UpdateFileAttributeClass.java
package com.jdojo.nio2.exercises;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.attribute.BasicFileAttributeView;
import java.nio.file.attribute.BasicFileAttributes;
import java.nio.file.attribute.FileTime;
import java.time.temporal.ChronoUnit;

public class UpdateFileAttributeClass {
    public static void main(String[] args) {
        try {
            Path path = Paths.get("test");

            // Read create time
            BasicFileAttributes bfa = Files.readAttributes(path, BasicFileAttributes.class);
            FileTime creationTime = bfa.creationTime();
            System.out.format("Create Time: %s %n", creationTime);

            FileTime newCreationTime
                    = FileTime.from(creationTime.toInstant().minus(5, ChronoUnit.HOURS));
            BasicFileAttributeView bfv
                    = Files.getFileAttributeView(path, BasicFileAttributeView.class);
            bfv.setTimes(bfa.lastModifiedTime(), bfa.lastAccessTime(), newCreationTime);

            // Read udpated create time
            BasicFileAttributes bfa2 = Files.readAttributes(path, BasicFileAttributes.class);
            System.out.format("Create Time After Update: %s %n", bfa2.creationTime());

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

15. How do you know the MIME type of a file?

Answer:

The `probeContentType(Path path)` method of the `Files` class returns the MIME content type of a file in string form.

16. How do you know if a `Path` represents a directory, a regular file, or a symbolic link?

Answer:

The `Files` class methods, `isDirectory()`, `isRegularFile()` and `isSymbolicLink()`, can be used to check whether a `Path` represents a directory, a regular file, or a symbolic link, respectively.

17. What file attribute view is guaranteed to be available on all platforms?

Answer:

`BasicFileAttributeView` is guaranteed to be available on all platforms.

18. What types of file system objects can you watch using the watch service in the NIO.2 API?

Answer:

Currently watch service can only watch directories for modifications.

19. Briefly explain the uses of the following classes and interfaces: `Watchable`, `WatchService`, `WatchKey`, `WatchEvent<T>`, `WatchEvent.Kind<T>`, and `StandardWatchEventKinds`.

Answer:

- `Watchable:` A file-system object that can be watched for changes.
- `WatchService:` A service that watches registered objects for changes.
- `WatchKey:` A token that identifies registration of an object with a `WatchService`.
- `WatchEvent<T>:` An event (or a repeated event) on an object registered with a watch service.
- `WatchEvent.Kind<T>:` Represents the kind of event that occurs on a registered object.
- `StandardWatchEventKinds:` A class that defines constants to represent the kind of `WatchEvent<T>`.

20. What is the purpose of the `AsynchronousFileChannel` class?

Answer:

An instance of the `java.nio.channels.AsynchronousFileChannel` class represents an asynchronous file channel that is used to read, write, and perform other operations on a file asynchronously. Multiple I/O operations can be performed simultaneously on an asynchronous file channel.