Java training

Filesystem API - working with files and folders

Session overview

- **Filesystem API -** working with files and folders
- **Hands-on** examples

Filesystem API

- **API** = **A**pplication **P**rogramming **I**nterface
- API → the programmatic way to interact with an application
- The Java filesystem API → the API used to interact with the filesystem, via CRUD operations
  - **CRUD** = Create, Read, Update, Delete
  - >90% of the business logic from any project is a CRUD operation
- Most of the operations that can be made from a OS file management tool can also be done via the API

Core classes

- `FileSystems` - allows obtaining an instance of a file system
- `FileSystemProvider` - searches for suitable FS candidates and creates an instance of it → ready to be worked with
- `FileStore` - an instance of a FS provides methods to retrieve a `FileStore` instance → the concrete physical storage for a file system
  - Provides details about: physical storage, device, partition, memory or specific representation of a file system
  - Primary concern - provide information about storage: capacity, free / allocated space, support for write access, name and storage specific attributes

Filesystem API classes - `Path` class

`Path` - interface used to access files and folders

- It's implementations provide platform specific access to files and folders
- Extends three interfaces:
  - `Comparable<Path>` - allows the comparison of two `Path` objects
  - `Iterable<Path>` - iterate over locations on the path; the iteration:
    - Begins in the location closest to the file system root location
    - Ends in concrete file or folder location
  - `Watchable` - allows registering watchers → watch the path location for changes
- The path entries are hierarchical → sequences of file and folder names, separated by a special character called *path delimiter (/ separator)*

Using `Path` objects

```
String currentUserHomePath = System.getProperty("user.home");
Path path = Paths.get(currentUserHomePath);

System.out.println(path.getFileName());
System.out.println(path.getParent().getFileName());

System.out.println(path.getFileSystem().getSeparator());
```

Hands-on →

Filesystem API classes - `Files` class

`Files` - static methods used to work with files and folders

- Most methods take `Path` objects as parameters
- Aggregates basic files and folders operations on a file system:
  - Creation of temporary files and folders
  - Creation, copying, moving and deleting of files and folders
  - Writing to files
  - Creation of links and symbolic links
  - Reading and writing file attributes
  - Traversing (recursively) the file tree structure
- One of the most useful methods: `walkFileTree` → a tree traversal algorithm for a recursive traversal of a `Path` object

Using the `Files` class

```
String currentUserHomePath = System.getProperty("user.home");

String fileName = "file-name.txt"; // the name of the created file
Path filePath = Paths.get(currentUserHomePath, fileName);

List<String> lines = Arrays.asList("The first", "The second");
Files.write(filePath, lines, StandardCharsets.UTF_8);

System.out.println("The file " + fileName + " was written");
```

Hands-on →

Filesystem API classes - `Paths` class

The **`Paths`** class has just two methods:

- `get(String first, String... more)`
  - Builds a path from the provided strings
  - If there are more parameters - joined together using the file system delimiter
  - Only creates paths for the default file system
- `get(URI uri)`
  - Used when working with more types of file systems, to create path instances
  - Uses concrete file system providers to create paths, based on the provided URI

Using the `Paths` class

```
String fileName = "file-name.txt";     // the name of the created file
Path filePath = Paths.get(fileName);   // will use the current working directory

String currentUserHomePath = System.getProperty("user.home");
Path filePath = Paths.get(currentUserHomePath, fileName);
                                       // will use the user's home directory

Files.write(filePath, lines, StandardCharsets.UTF_8);
System.out.println("The file " + fileName + " was written");
```

File access rights

Two main classes:

- `UserPrincipal` - represent users and groups
  - Used to read and write access rights to file system objects
- `UserPrincipalLookupService` - used to retrieve `UserPrincipal` instances by name (usually OS username)

Filesystem access rights example

```
FileSystem fileSystem = FileSystems.getDefault();
UserPrincipalLookupService lookup = fileSystem.getUserPrincipalLookupService();

String currentUser = System.getProperty(“user.name”);
UserPrincipal userPrincipal = null;
try {
    userPrincipal = lookup.lookupPrincipalByName(currentUser);
} catch (IOException e) {
    throw new RuntimeException(e);
}

Path filePath = Paths.get(System.getProperty(“user.home”), “file-name.txt”);
Files.setOwner(filePath, userPrincipal);
```

Hands-on →

Further reading

https://docs.oracle.com/javase/7/docs/api/java/nio/file/FileSystem.html

https://jakubstas.com/file-system-api/#.Wog3s8ixXRZ

Q&A session

1. You ask, I answer
2. I ask, you answer