CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

Create a File

Overview

1. Introduction
2. Naming Conventions for Creating a File in Java
3. Ways to Create a File in Java
4. File.createNewFile() method
5. Creating a File using FileOutputStream
6. Creating a file using createFile() method

Introduction

- In this lecture, we will learn how to create a file in Java easily using the `File` class.
- `File` class is used to create a file in Java. It belongs to `java.io` package.
- `File` class can perform various operations related to file handling, such as creating, opening, closing, and deleting a file.
- We can also get the name and size of a file using `File` class.

Naming Conventions for Creating a File in Java

There are some naming conventions you must follow while creating a file in Java.

These naming conventions are as follows:

1. File name should be unique and of string type.
2. We can divide the file name into two parts. For example, `myfile.txt` and `myfile.dat`.
3. Before using any file, we should know its purpose, whether it is for writing, reading, or both.
4. Data type of file operations can be in bytes or characters.

Ways to Create a File in Java

In Java, there are three ways to create a file. They are as follows:

- Using `File.createNewFile()` method
- Using `FileOutputStream` class
- By using `File.createFile()` method

Let’s understand all ways one by one with the help of example programs.

File.createNewFile() method

`File` class is a powerful class that provides a static method named `creatNewFile()` to create an empty file automatically. This method belongs to a `java.io` package.

The general signature of this method is as:

```
public boolean createNewFile() throws IOException
```

This method does not take any argument value. The `createNewFile()` automatically creates a new, empty file. This method returns a boolean value:

- true, if the file created successfully.
- false, if the file already exists.

Example 1: File.createNewFile()

1. Let’s create a Java program to create a new, empty text file.

The first execution will create a file named `file1.txt` successfully in the specified location and display a message “`File created at a location`”.

While the second execution will display only a message, “`File already exists at a location`”.

Let’s write the code for it.

Example 1: File.createNewFile()

```
import java.io.File;
import java.io.IOException;
public class CreateFileTester1 {
   public static void main(String[] args) {
       // Create an object of File class and pass the path of filename as argument.
       File file = new File("./src/file1.txt");
       boolean result;
       try {
           result = file.createNewFile(); // creates a new file
           // Check for the existence of a file whether successfully created.
           if(result) {
               System.out.println("File created at location: " +file.getCanonicalPath()); // returns the path string.
           }
           else  {
               System.out.println("File already exists at location: "+file.getCanonicalPath());
           }
       }
       catch (IOException e) {
           e.printStackTrace(); // It will print exception if any.
       }
   }
}
```

Example 1: File.createNewFile()

Output → 1st Execution:

```
File created at location: /Users/ted/IdeaProjects/Lecture23-Files/src/file1.txt
```

Output → 2nd Execution:

```
File already exists at location: /Users/ted/IdeaProjects/Lecture23-Files/src/file1.txt
```

Explanation:

In this example program, we have created an object of `File` class and pass the file name as an argument to `File` class constructor. We can also pass absolute path or relative path as an argument to `File` class constructor.

If we pass a non-absolute path to `File` class constructor, `File` object attempts to locate the file in the current directory. Then, we have invoked `createNewFile()` method of the `File` class to create a new file in Java.

The `createNewFile()` method of `File` class may throw `IOException` if an I/O error occurs. It can also throw `SecurityException` if a security manager exists.

Creating a File using FileOutputStream

`FileOutputStream` in Java is a concrete subclass of `OutputStream` that writes data to a file or to a `FileDescriptor`.

It was added in Java 1.0 version. `FileOutputStream` class is present in the `java.io.FileOutputStream` package. It stores data as individual bytes.

`FileOutputStream` class provides a constructor to create a text file. The general signature is as:

```
public FileOutputStream(String name, boolean append) throws FileNotFoundException
```

This constructor takes two parameters:

- name: is the name of file.
- append: if true, data (in byte) will be written to the end of the file, not in the beginning.

Example 2: FileOutputStream

2. Let’s create a Java program to create a text file using `FileOutputStream` class easily.

Example 2: FileOutputStream

```
import java.io.FileOutputStream;
import java.util.Scanner;
import java.io.IOException;
public class CreateFileTester2 {
   public static void main(String[] args) throws IOException{
       // Creating an object of Scanner class.
       Scanner sc = new Scanner(System.in);
       // Store the filepath into the variable filepath of type String.
       String filepath = "./src/file2.txt";

       // Create FileOutputStream object pass name and true as arguments to constructor.
       FileOutputStream fos = new FileOutputStream(filepath, true);  // true for append mode.
       System.out.print("Enter your file content: ");
       String s = sc.nextLine()+"\n"; // Variable s stores the string which we have entered

       // Converts string into bytes using getBytes() method.
       byte[] b = s.getBytes();
       fos.write(b); // writing bytes into file.
       fos.close();  // closing the file.
       System.out.println("file saved successfully.");
   }
}
```

Example 2: FileOutputStream

Output:

```
Enter your file content: Hello World
file saved successfully.
```

Creating a file using createFile() method

The `createFile()` is a static method of `File` class which belongs to `java.nio.file.Files` package. It creates a new and empty file. It fails if the file already exists.

The main advantage of using this method is that we don't need to close the resources while using this method.

The general signature of this method is:

```
public static Path createFile(Path, FileAttribute) throws IOException
```

This method returns the file. It takes two parameters:

- `Path`: It is the path of the file.
- `FileAttribute`: It is an optional list of file attributes to set atomically when creating the file. Its name identifies each attribute.

If a file of that name already exists, this method throws `FileAlreadyExistsException`. If an I/O error occurs or the parent directory does not exist, this method throws `IOException`.

Example 3: createFile() method

3. Let’s create a Java program to create a file in Java using `createFile()` method.

Example 3: createFile() method

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
public class CreateFileTester3 {
   public static void main(String[] args) {
       // Creating Path instance.
       Path path = Paths.get("./src/file3.txt");
       try {
           Path p = Files.createFile(path); // creating a file at specified location.
           System.out.println("File created successfully at location: " +p);
       }
       catch (IOException e) {
           e.printStackTrace();
       }
   }
}
```

Example 3: createFile() method

Output:

```
File created successfully at location: ./src/file3.txt
```

Explanation:

In this example program, we have created a new and empty file. We have used `get()` method of `Paths` class and assigned it to a variable path of type `Path` to create a `Path` instance.

The following statement creates a Path instance.

```
Path path = Paths.get("./src/file3.txt");
```

In the above statement, `Path` is an interface and `Paths` is a class. Both belongs to the same package. The `Paths.get()` method creates the `Path` object.

END