CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

Read a File

Overview

1. Introduction
2. Read Text File using BufferedInputStream class
3. Read Text File line-by-line using BufferedReader class
4. Read Text File using Scanner class
5. Read Text File using FileReader class
6. Read Text File using Files class

Introduction

In this lecture, we will learn how to read data (or text) of a file using various classes in Java.

There are several ways to read a text file in Java. By using the following classes, we can read data from any text file in Java easily.

They are as:

- `BufferedInputStream` class
- `BufferedReader` class
- `Scanner` class
- `FileReader` class
- Reading the entire file in a list using `readAllLines()` of `Files` class
- Read a text file as `String`

Let's understand these ways to read data from a file in Java with example programs.

Reading Text File using BufferedInputStream

`BufferedInputStream` in Java is a concrete subclass of `FilterInputStream` class. It wraps (buffers) an input stream into a buffered stream and makes read operations on the stream more efficient and fast.

In other words, `BufferedInputStream` adds buffering capabilities to an input stream that stores data (in bytes) temporarily into a memory buffer by reading from the stream.

It adds an additional layer of functionality around the underlying stream. Therefore, it speeds up the input by reducing the number of disk or file reads.

Example 1: BufferedInputStream

1. Let’s create a simple Java program to read data from a file `readfile1.txt` line by line using `BufferedInputStream` class. Look at the source code to understand better..

Example 1: BufferedInputStream

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
public class ReadFileTester1 {
   public static void main(String[] args) throws IOException{
          // Create a FileInputStream object to attach readfile1 to FileInputStream.
          FileInputStream fis = new FileInputStream("./src/readfile1.txt");

          // Create a BufferedInputStream object to wrap around FileInputStream.
          BufferedInputStream bis = new BufferedInputStream(fis);
          int i = 0;
          while ((i = bis.read()) != -1) {
              char ch = (char) i;
              System.out.print(ch);
          }
          bis.close();
          fis.close();
   }
}
```

Example 1: BufferedInputStream

Output:

```
Hello from a file.
```

Explanation:

In this example program,

1. We have created a buffered input stream named bis and connected it to `FileInputStream fis`.

2. Then, we have used a while loop and `read()` method to read all bytes from the internal buffer and display them on the console. Here, we assume that you have the following data in `"readfile1.txt"` file:

```
First Line
Second Line
Third Line
Fourth Line
```

Read text file line by line using BufferedReader

`BufferedReader` in Java is a buffering input character stream that reads text from the buffer rather than directly underlying input stream or other text sources.

It adds the buffering capability to the underlying input character stream, so that there is no need to access the underlying file system for each read and write operation.

`BufferedReader` is a subclass of the `Reader` class that extends `Object` class. It implements `Closeable`, `AutoCloseable`, and `Readable` interfaces. It is also a superclass of `LineNumberReader` class.

Example 2: BufferedReader

2. Let’s create a Java program in which we will read a text of line from an existing file and display it on the console.

Example 2: BufferedReader

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class ReadFileTester2 {
   public static void main(String[] args) throws IOException {
       String filepath = "./src/readfile2.txt";
       // Create an object of FileReader and pass filepath to its constructor.
       FileReader fr = new FileReader(filepath);

       // Create an object of BufferedReader and pass FileReader fr to its constructor.
       BufferedReader br = new BufferedReader(fr);
       String lineOfText;
       // Read a line of text.
       while((lineOfText = br.readLine()) != null) {
           System.out.println(lineOfText);
       }
   }
}
```

Example 2: BufferedReader

Output:

```
First Line
Second Line
Third Line
Fourth Line
```

Explanation:

Assume that you have the following data in the `fileread2.txt` file:

```
First Line
Second Line
Third Line
Fourth Line
```

Read text file using Scanner class

`Scanner` in Java is a predefined class that reads or scans the data dynamically from the keyboard or a text file.

In other words, `Scanner` class allows the user to read all types of numeric values, strings, and other types of data in Java, whether it comes from a disk file, keyboard, or another source.

It is the fastest and easiest way to receive the input from the user in java than `InputStreamReader` and `BufferedReader`.

Example 3: Scanner

3. Let’s take a simple example program where we will read data from the myfile.txt file using `Scanner` class.

Example 3: Scanner

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;
public class ReadFileTester3 {
   public static void main(String[] args) throws IOException {
       // Create an object of File class.
       File file = new File("./src/readfile3.txt");
       // Create an object of Scanner class for the file.
       Scanner sc = new Scanner(file);

       // Reading data from the file.
       while (sc.hasNext()) {
           String firstName = sc.next();
           String mName = sc.next();
           String lastName = sc.next();

           int age = sc.nextInt();
           System.out.println(firstName + " " + mName + " " + lastName + " " + age);
       }
       // Close the file
       sc.close();
   }
}
```

Example 3: Scanner

Output:

```
John T Smith 50
Eric K Smith 45
```

Explanation:

In this program, each iteration in the while loop reads the first name, middle name, last name, and age from the text file. The file is closed using `close()` method.

It is not essential to close the input file, but it is a good practice to do so to release the resources occupied by the file.

Reading text file using FileReader class

`FileReader` in Java is an input stream that reads data in the form of characters from a text file.

In other words, `FileReader` is a character-based input stream that is used to read characters from a file.

`FileReader` class is a subclass of `InputStreamReader` class that extends `Reader` class. It implements `Closeable, AutoCloseable`, and `Readable` interfaces.

To read text file line by line in Java using `FileReader` class, go to this tutorial: `FileReader` class in Java

Reading Entire Content of a File using Files class

The `Files` class provides a static method named `readAllLines()` to read the contents of a file as lines of text. The `readAllLines()` method comes into two variants. The general signature of this method is as:

```
static List<String> readAllLines(Path path)
static List<String> readAllLines(Path path, Charset cs)
```

Both methods may throw an `IOException`. They use a carriage return, a line feed, and a carriage returned followed by a line feed as a line terminator.

Example 4: Files

4. Let’s create a Java program to read whole contents of a file using readAllLines() method of Files class.

Example 4: Files

```java
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
public class ReadFileTester4 {
   public static List readFile(String fileName) {
       List lines = Collections.emptyList();
       try {
           // Read all lines and returns a list of string.
           lines = Files.readAllLines(Paths.get(fileName), StandardCharsets.UTF_8);
       }
       catch (IOException e) {
           e.printStackTrace();
       }
       return lines;
   }
   public static void main(String[] args) throws IOException
   {
       List list = readFile("./src/readfile4.txt");
       // Print each line using iterator() method.
       Iterator itr = list.iterator();
       while (itr.hasNext())
           System.out.println(itr.next());
   }
}
```

Example 4: Files

Output:

```
First Line
Second Line
Third Line
Fourth Line
```

Explanation:

Assume that you have the following data in the `fileread4.txt` file:

```
First Line
Second Line
Third Line
Fourth Line
```

Read a text file as String

Let’s create a Java program to read a text file as string. In this program, we will use `readAllBytes()` method of `Files` class. The `readAllBytes()` method of Files class reads the contents of a file as bytes. The general syntax of this method is as:

```
static byte[ ] readAllBytes(Path path)
```

This method ensures that the file is closed when all bytes have read or I/O error, or other runtime exception is thrown.

Example 5: String

```java
import java.nio.file.Files;
import java.nio.file.Paths;
public class ReadFileTester5 {

   public static String readFile(String fileName)throws Exception {
       String fileContents = "";
       fileContents = new String(Files.readAllBytes(Paths.get(fileName)));
       return fileContents;
   }

   public static void main(String[] args) throws Exception {
       String fileContents = readFile("./src/readfile5.txt");
       System.out.println(fileContents);
   }
}
```

Example 4: Files

Output:

```
This is the first line.
This is the second line.
```

Explanation:

Assume that you have the following data in the `fileread5.txt` file:

```
This is the first line.
This is the second line.
```

END