CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

FileInputStream

Overview

1. Introduction
2. FileInputStream Class
3. FileInputStream Constructors
4. Steps to Read Data from File using FileInputStream class
5. FileInputStream Methods
6. Example 1: Reading a single character
7. Example 2: Read All Bytes
8. Example 3: Read All Characters
9. Example 4: Read a Byte and Array of Bytes

Introduction

- **FileInputStream in Java** is the most basic file input stream class that is designed to read bytes from a file.

- In simple words, it reads data from a text file in the form sequence of bytes. That is, it inputs a stream of bytes from a file.

- FileInputStream was introduced in Java 1.0 version. It is present in `java.io.FileInputStream` package.

FileInputStream Class

`FileInputStream` class is derived from `InputStream` that is abstract superclass of `FileInputStream`. It implements `Closeable` and `AutoCloseable` interfaces.

The general syntax for `FileInputStream` class is as follows:

```
public class FileInputStream
       extends InputStream
       implements Closeable, AutoCloseable
```

FileInputStream Constructors

`FileInputStream` class defines the following constructors that create an `InputStream` and read bytes from a file. Its three most common constructors are as follows:

---

1. FileInputStream(File file):
This constructor creates a `FileInputStream` from a `File` object in the file system.

---

2. FileInputStream(FileDescriptor fdObj):
This form of constructor creates a `FileInputStream` by using the `FileDescriptor` fdObj.

---

3. FileInputStream(String filename):
This form of constructor constructs a `FileInputStream` from a file name.

If we try to create a `FileInputStream` with a nonexistent file, `java.io.FileNotFoundException` will occur.

Steps to Read Data from File using FileInputStream class

There are the following steps to read data from a text file using `FileInputStream` that are as follows:

1. First, we need to add a file to a `FileInputStream` as follows:

```
FileInputStream fis = new FileInputStream("myfile.txt");
```

This `FileInputStream` object enables us to read data from a file in the form of bytes.

Steps to Read Data from File using FileInputStream class

There are the following steps to read data from a text file using `FileInputStream` that are as follows:

2. Next step is to read data from a file. We need to call `read()` method using the file input stream object reference variable. The syntax is as follows:

```
chars = fis.read();
```

Once the `read()` method reads all the characters from a file, it will reach the end of the file, and `-1` will be returned if there is no more data available to read further.

Steps to Read Data from File using FileInputStream class

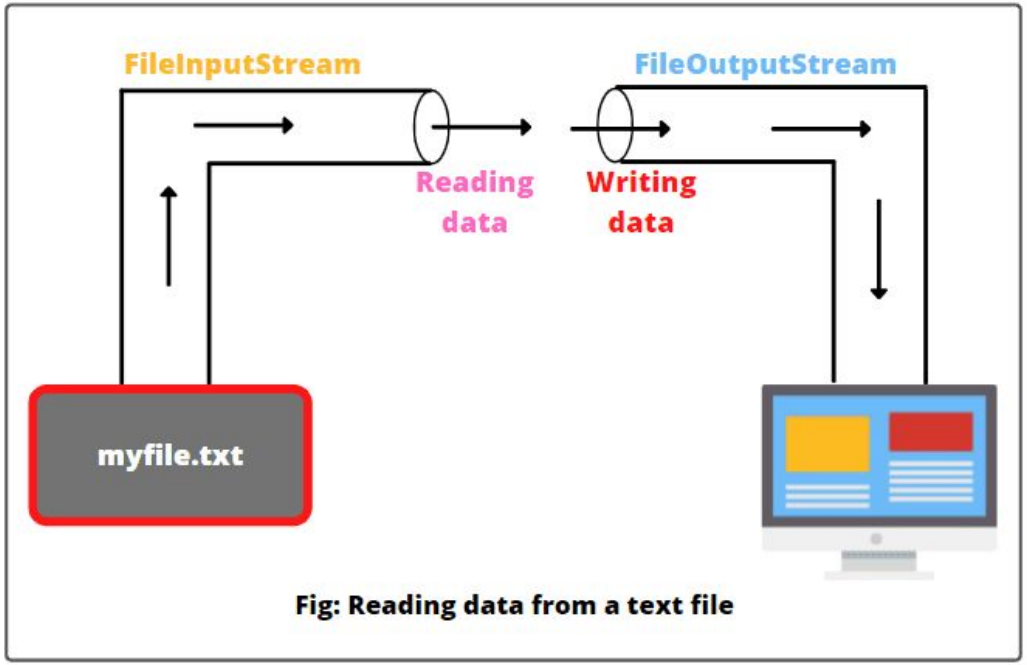

Fig: Reading data from a text file

Look at the figure to understand the two steps for reading data from a text file using Java `FileInputStream` class.

FileInputStream Methods

`FileInputStream` class does not define any new methods. All the methods in this class are inherited from `InputStream`. The most common inherited methods from `InputStream` are as follows:

FileInputStream Methods

Method	Description
int available()	This method is used to get the number of bytes that can be read (or skipped over) from this file input stream without blocking by the next invocation of a method for this input stream.
void close()	This method closes the file input stream and releases any system resources associated with the stream.
FileDescriptor getFD()	This method returns the FileDescriptor object that represents the association to the actual file in the file system being used by this FileInputStream.

FileInputStream Methods

Method	Description
int read()	This method reads a byte of data from the input stream.
int read(byte[ ] b)	This method reads up to b.length bytes of data from the invoking input stream into an array of bytes.
int read(byte[ ] b, int n, int m)	It reads up to m bytes of data from this input stream into an array of bytes from the nth byte.
long skip(long n)	This method skips over and discards n bytes of data from the input stream.

FileInputStream Methods

Almost all the methods in the I/O stream classes throw an exception named IOException. This exception is thrown when an Input/Output operation fails because of an interrupted call.

Therefore, we need to declare to throw java.io.IOException in the method or put the code in a try-catch block, as shown below:

```
//Declaring IOException exception in the method
public static void main(String[] args) throws IOException {
   // Perform I/O operations.
}
//or, Using try-catch block
public static void main(String[] args) {
   try {
       // Perform I/O operations
   }
   catch (IOException ex) {
       ex.printStackTrace();
   }
}
```

Example 1: Reading a single character

1. Let’s take an example program where we will read data from `myfile.txt` using FileInputStream and display it on the monitor. Before doing coding, first, create a file named `myfile.txt`.

In this file, we will get the following content: `Welcome to UNO Computer Science`.

We will write code to read a single character only from this file. Look at the source code to understand better.

Example 1: Read a single character

```
import java.io.FileInputStream;
public class FileInputStreamTester1 {
   public static void main(String[] args) {
       try {
           // Attach a file to FileInputStream for reading data and create an input stream for a file
           FileInputStream fis = new FileInputStream("./src/myfile.txt");

           // Reading a value from a file in the form of byte.
           int value = fis.read();
           System.out.println("Reading a value in byte form: " +value);

           // Converting byte into character to see text.
           System.out.print((char)value); // Displaying a single character on console.
           fis.close(); // Closing input stream automatically.
       } catch(Exception e){
           System.out.println(e);
       }
   }
}
```

Example 1: Reading a single character

Output:

```
Reading a value in byte form: 87
W
```

After executing the preceding program, you will get a value `87` (in byte form) and a single character from the file. To see the read text, we need to translate it into character.

The `java.io.InputStream` class implements `AutoClosable` interface. The `AutoClosable` interface defines the `close()` method that closes a resource automatically.

Note:

When a stream is no longer required, always close it using the `close()` method or automatically close it using a `try-with-resource` statement.

Not closing streams may produce data corruption in the output file or other programming errors.

Example 2: Read All Bytes

2. Let’s take an example program where we will read all byte values from a file and display them on the console.

Example 2: Read All Bytes

```
import java.io.FileInputStream;
public class FileInputStreamTester2 {
   public static void main(String[] args) {
       try {
           FileInputStream fis = new FileInputStream("./src/myfile.txt");

           // Reading all byte values from a file and display on the console.
           int value = 0;
           while((value = fis.read())!=-1) {
               System.out.print(value + " ");
           }
           fis.close();
       }
       catch(Exception e) {
           System.out.println(e);
       }
   }
}
```

Example 2: Read All Bytes

Output:

```
87 101 108 99 111 109 101 32 116 111 32 85 78 79 32 67 111 109 112 117 116 101 114 32 83 99 105 101 110 99 101 10
```

In this example program, values are read from the file and displayed on the console in lines.

The expression `((value = fis.read()) != -1)` reads a byte from `fis.read()`, assigns it to value, and checks whether it is `-1`. The input value of `-1` represents the end of a file (EOF).

Example 3: Read All Characters

3. Reading all characters from a file

Example 3: Read All Characters

```
import java.io.FileInputStream;
public class FileInputStreamTester3 {
   public static void main(String[] args) {
       try {
           FileInputStream fis = new FileInputStream("./src/myfile.txt");

           int value = 0;
           while((value = fis.read())!=-1) {
               System.out.print((char)value);
           }
           fis.close();
       }
       catch(Exception e){
           System.out.println(e);
       }
   }
}
```

Example 3: Read All Characters

Output:

```
Welcome to UNO Computer Science
```

Example 4: Read a Byte and Array of Bytes

4. Let’s create a program where we will see how to read a single byte, an array of bytes, and a subrange array of bytes. We will also determine the number of available bytes remaining using available() method and skip unwanted bytes by using skip() method.

Look at the source code and try to understand it one by one.

Example 4: Read a Byte and Array of Bytes

```
import java.io.FileInputStream;
import java.io.IOException;
public class FileInputStreamTester4 {
   public static void main(String[ ] args) throws IOException {
       FileInputStream fis = new FileInputStream("./src/myfile.txt");
       int size = fis.available();

       System.out.println("Total number of available bytes: " +size);
       int n = 2;

       for (int i = 0; i < n; i++) {
           System.out.print((char) fis.read()); // Reading the first two characters W and e.
       }
       System.out.println("\nStill Available bytes: " + fis.available());
       byte bytearray[ ] = new byte[2];

       if (fis.read(bytearray) != n) {
           System.out.println("Could not get " + n + " bytes");
       }
       String str = new String(bytearray, 0, n);
       System.out.println(str);

       System.out.println("\nStill available bytes: " +fis.available());
       System.out.println("\nSkipping half of remaining bytes using skip() method");
       fis.skip(size/2);

       System.out.println("Still Available bytes: " + fis.available());

       System.out.println("Reading " + n/2 + " from the end of array");
       if (fis.read(bytearray, n/2, n/2) != n/2) {
           System.out.println("couldn't read " + n/2 + " bytes.");
       }
       String str2 = new String(bytearray, 0, bytearray.length);
       System.out.println(str2);

       System.out.println("\nStill Available bytes: " + fis.available());
       fis.close();
   }
}
```

Example 4: Read a Byte and Array of Bytes

Output:

```
Total number of available bytes: 32
We
Still Available bytes: 30
lc

Still available bytes: 28

Skipping half of remaining bytes using skip() method
Still Available bytes: 12
Reading 1 from the end of array
lt

Still Available bytes: 11
```

This example program somewhat contrived demonstrates how to read data in three ways, skip input, and inspect the amount of data available on a stream.

END