CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

FileOutputStream

Overview

Introduction

- A **FileOutputStream in Java** is a concrete subclass of `OutputStream` that provides methods for writing data to a file or to a `FileDescriptor`.
- In simple words, a file output stream is an `OutputStream` that writes data to a file. It stores data in the form of individual bytes.
- A file output stream can be used to create a text file. For example, if you want to write strings into a text file, use `FileOutputStream` object.
- We can write both byte-oriented as well as character-oriented data via `FileOutputStream` class. But, for character-oriented data, it is recommended to use `FileWriter` than `FileOutputStream`.
- `FileOutputStream` was added in Java 1.0 version. It is present in the `java.io.FileOutputStream` package.

FileOutputStream Class declaration

FileOutputStream class is derived from OutputStream class. OutputStream class is an abstract superclass of all classes representing an output stream of bytes like FileOutputStream, ObjectOutputStream, etc.

Java FileOutputStream class implements Closeable, Flushable, and AutoCloseable interfaces.

The general declaration of java.io.FileOutputStream class in java is as follows:

```
public class FileOutputStream
       extends OutputStream
       implements Closeable, Flushable, AutoCloseable
```

FileOutputStream Constructor

To create an instance, `FileOutputStream` class defines five constructors. In all cases, if the file is not opening for writing for any reason, an exception named `FileNotFoundException` will be thrown.

1. FileOutputStream(File file)

This form of constructor creates a `FileOutputStream` to write data to the specified `File` object. The contents of any existing file will be overwritten.

FileOutputStream Constructor

To create an instance, `FileOutputStream` class defines five constructors. In all cases, if the file is not opening for writing for any reason, an exception named `FileNotFoundException` will be thrown.

2. FileOutputStream(File file, boolean append)

This constructor creates a `FileOutputStream` to write data to the specified `File` object. If `append` is `true`, data is appended to the existing file with the following existing contents.

If `append` is `false`, existing data in the file will be cleared when the `FileOutputStream` is constructed.

FileOutputStream Constructor

To create an instance, `FileOutputStream` class defines five constructors. In all cases, if the file is not opening for writing for any reason, an exception named `FileNotFoundException` will be thrown.

3. FileOutputStream(FileDescriptor fdObj)

This constructor creates a `FileOutputStream` for writing data to the specified file descriptor.

FileOutputStream Constructor

To create an instance, `FileOutputStream` class defines five constructors. In all cases, if the file is not opening for writing for any reason, an exception named `FileNotFoundException` will be thrown.

4. FileOutputStream(String filename)

This form of constructor creates a `FileOutputStream` to write to the file with the specified `filename`.

FileOutputStream Constructor

To create an instance, `FileOutputStream` class defines five constructors. In all cases, if the file is not opening for writing for any reason, an exception named `FileNotFoundException` will be thrown.

5. FileOutputStream(String name, boolean append)

This constructor creates a `FileOutputStream` to write to the file with the specified `name`. If `append` is `true`, data will be appended to the file with the following existing contents. If `append` is `false`, the contents of the existing file will be overwritten.

Steps to Create File using FileOutputStream

Here are the following steps to create a text file that will store some characters or text:

1. First, we need to read data from the keyboard. For this purpose, we will have to attach keyboard to an input stream class. The syntax for reading data from keyboard is given below:

```
DataInputStream dis = new DataInputStream(System.in);
```

In the above statement, `System.in` represents the keyboard that is linked with `DataInputStream` object whose reference variable is `dis`.

Steps to Create File using FileOutputStream

Here are the following steps to create a text file that will store some characters or text:

2. Second, attach a file where data is to be stored to an output stream. For this purpose, the syntax for attaching a file `fileout.txt` to `FileOutputStream` is given below:

```
FileOutputStream fos = new FileOutputStream("fileout.txt");
```

Here, for represents object reference variable of `FileOutputStream` class.

Steps to Create File using FileOutputStream

Here are the following steps to create a text file that will store some characters or text:

3. Third step is to read data from `DataInputStream` and write it into `FileOutputStream`. This means that we will read data from `dis` object and write it into `fos` object. The syntax is as follows:

```
ch = (char) dis.read();  // Read a single character into ch.
fos.write(ch);           // write ch into file.
```

Lastly, file must be closed after performing input or output operations on it. Otherwise, data on the file may be corrupted.

Steps to Create File using FileOutputStream

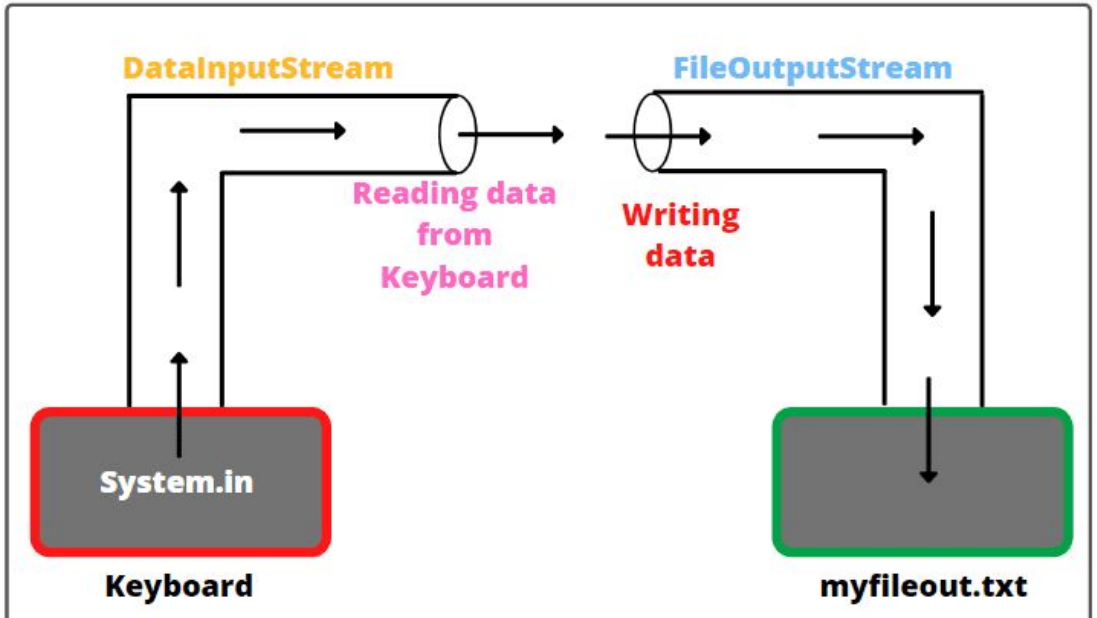

Fig: Creating a text file

Look at the figure to see all these steps.

FileOutputStream Methods

`FileOutputStream` class does not define any new methods. Since `FileOutputStream` class is derived from `OutputStream` class, therefore, all the methods in this class are inherited from `OutputStream`.

The most common useful methods are as follows:

FileOutputStream Methods

Method	Description
void close()	This method is used to close the file output stream and releases any system resources associated with this stream.
protected void finalize()	This method cleans up the connection with the file output stream.
FileChannel getChannel()	This method returns the unique FileChannel object associated with the file output stream.
FileDescriptor getFD()	It returns the file descriptor associated with the stream.

FileOutputStream Methods

Method	Description
void write(int b)	This method writes the specified or single byte to this file output stream.
void write(byte[ ] b)	It writes a complete array of bytes to the file output stream.
void write(byte[ ] b, int off, int numBytes)	This method writes numBytes bytes from the specified byte array starting at offset off to the file output stream.
void flush( )	This method flushes the output stream and forces any buffered output bytes to be written out.

FileOutputStream Methods - Checked Exceptions

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: Writing a single byte

1. Let’s take an example program in which we will write a single byte into a file.

Example 1: Writing a single byte

```
import java.io.FileOutputStream;
import java.io.IOException;
public class FileOutputStreamTester1 {
   public static void main(String[] args) {
       try {
           // Store the filepath into the variable filepath of type String.
           String filepath = "./src/out1.txt";

           // Create FileOutputStream to attach file by using the filepath in its constructor.
           FileOutputStream fos = new FileOutputStream(filepath);
           fos.write(87);
           fos.close(); // Closing file.

           System.out.println("Successfully written");
       }
       catch(IOException e){
           System.out.println(e);
       }
   }
}
```

Example 1: Writing a single byte

Output:

```
Successfully written
```

Data “W” is successfully written into the text file `out1.txt.`

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: Writing a String to Text file

2. Let’s take another example program to write a string into the text file. Look at the source code.

Example 2: Writing a String to Text file

```java
import java.io.FileOutputStream;
public class FileOutputStreamTester2 {
   public static void main(String[] args) {
       try {
           String filepath = "./src/out2.txt";
           FileOutputStream fos = new FileOutputStream(filepath);
           String str = "Welcome to UNO Computer Science!";

           byte bytearray[ ] = str.getBytes(); // Converting string into byte array.
           fos.write(bytearray);
           fos.close();

           System.out.println("Successfully written");
       }
       catch(Exception e){
           System.out.println(e);
       }
   }
}
```

Example 2: Writing a String to Text file

Output:

```
Successfully written
```

The data "`Welcome to UNO Computer Science!`" is successfully written into the text file `out2.txt.`

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 3: Writing from keyboard to Text file

3. Let’s take an example program where we will understand how to read data from the keyboard and write it to `out3.txt` file. Look at the program source code to understand better.

Example 3: Writing from keyboard to Text file

```java
import java.io.DataInputStream;
import java.io.FileOutputStream;
public class FileOutputStreamTester3 {
   public static void main(String[] args) {
       try {
           // Create an object of DataInputStream to attach keyboard to DataInputStream.
           DataInputStream dis = new DataInputStream(System.in);
           // Store the filepath into the variable filepath of type String.
           String filepath = "./src/out3.txt";
           // Create FileOutputStream to attach file by using the filepath in its constructor.
           FileOutputStream fos = new FileOutputStream(filepath);
           System.out.println("Enter the text (@ at the end)");

           int value = 0;
           // Read the values (in byte form) from dis into ch and write them into fos.
           while((value = dis.read()) != '@'){
               char ch = (char)value; // Converting byte values into characters.
               fos.write(ch);
           }

           fos.close(); // Closing file.
           System.out.println("Successfully written...");
       }
       catch(Exception e){
           System.out.println(e);
       }
   }
}
```

Example 3: Writing from keyboard to Text file

Output:

```
Enter the text (@ at the end)
Hello World
Keyboard to Textfile
Across multiple lines
@
Successfully written...
```

In this example program, we read the data from the keyboard and write it into `out3.txt` file. This program takes data from the keyboard as long as the user types `@` to end the statement.

You can observe the output, we have entered three lines of statements and typed `@` to end the statement. Now, open your `out3.txt` file and see that data is successfully written into the file or not.

If the above program is executed again, you will notice that the old data of `out3.txt` file has been lost completely and the new data will be stored in the file. Look at the output of a second execution of the program.

Example 3: Writing from keyboard to Text file

Note:

If you do not want to lose previous data of the file and just want to append the new data at the end of already existing data, you open the file by writing true along with filename. The syntax is as follows:

```
FileOutputStream fos = new FileOutputStream(filepath, true);
```

When you will use this statement in the previous program, execute the program several times, still all the previous data will be preserved and new data will be added to the old data.

Example 4: Copy Data from one File to another File

4. Let’s create a program to copy data from one file to another file using `FileInputStream` and `FileOutputStream` classes. In the first file `myfile.txt`, we will store data as “`Welcome to UNO Computer Science`”.

Then, we will copy it and store it in the second file `out4.txt`. Look at the source code to understand better.

Example 4: Copy Data from one File to another File

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
public class FileOutputStreamTester4 {
   public static void main(String[] args) {
       try {
           FileInputStream fis = new FileInputStream("./src/myfile.txt");
           FileOutputStream fos = new FileOutputStream("./src/out4.txt");

           int i = 0;
           while ((i = fis.read()) != -1){
               char ch = (char)i;
               fos.write(ch);
           }
           fis.close();
           System.out.println("Successfully written...");
       }
       catch(Exception e) {
           System.out.println(e);
       }
   }
}
```

Example 4: Copy Data from one File to another File

Output:

```
Successfully written
```

The data of file `myfile.txt` is copied in the file `out4.txt`. Now open the `out4.txt` file and verify that data is successfully copied or not.

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.

END