CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

SequenceInputStream

Overview

1. Introduction
2. SequenceInputStream class declaration
3. SequenceInputStream constructors
4. SequenceInputStream methods
5. SequenceInputStream Examples

Introduction

- **SequenceInputStream in Java** is a single input stream (file) that is a combination of two or more input streams (files) together in a particular order. It reads data from two or more input streams sequentially, one after the other.

- `SequenceInputStream` begins to read data from a particular order of input streams. It first reads data (in bytes) from the first stream in the sequence, then all the data from the second stream in sequence, then all the data from the third stream, and so on.

- When the end of one stream (file) is reached, that file is closed; the next data comes from the next input stream (file). This process continues until the end of the file is reached on the last of the contained input streams.

- Thus, the connection of two or more files into a single file is called **concatenation of files** and can be achieved by `SequenceInputStream` class in Java.

SequenceInputStream class declaration

`SequenceInputStream` class extends `InputStream` class and implements `Closeable`, and `AutoCloseable` interfaces.

The general declaration of sequence input stream class in Java is given below:

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

`SequenceInputStream` class was added in Java 1.0 version. It is present in `java.io.SequenceInputStream` package.

SequenceInputStream Constructors

`SequenceInputStream` class defines two constructors that are different from any other `InputStream`:

1. SequenceInputStream(InputStream s1, InputStream s2):
This constructor creates a new `SequenceInputStream` object that takes two `InputStream` objects as arguments and combines them to create a single input stream. It reads data (in bytes) in order, first `s1` and then `s2`.

For example, to read data from both `file1` and `file2`, we might do this:

```
FileInputStream file1 = new FileInputStream("text1.txt");
FileInputStream file2 = new FileInputStream("text2.txt");

// Concatenate file1 and file2 into file3.
SequenceInputStream file3 = new SequenceInputStream(file1, file2);
```

SequenceInputStream Constructors

`SequenceInputStream` class defines two constructors that are different from any other `InputStream`:

1. SequenceInputStream(InputStream s1, InputStream s2): Look at the figure below to understand better.

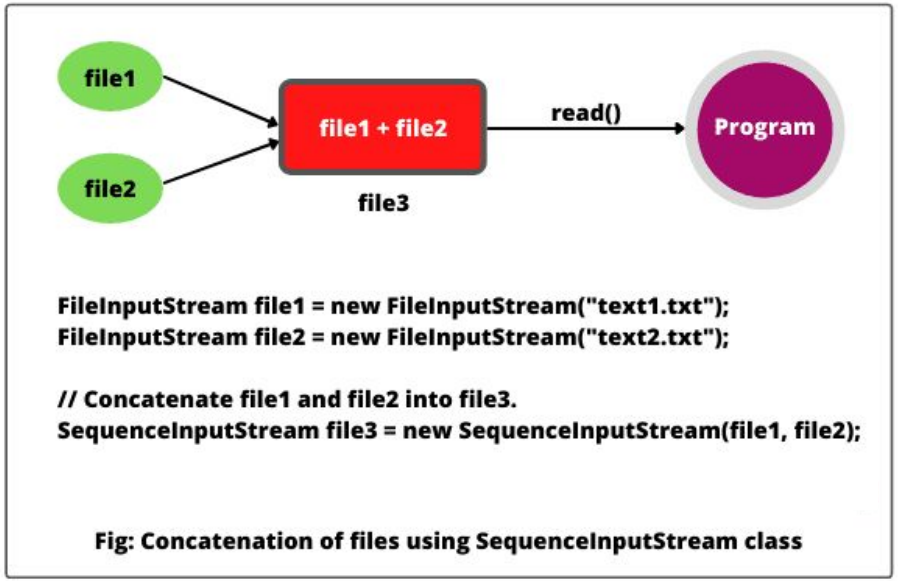

Fig: Concatenation of files using SequenceInputStream class

SequenceInputStream Constructors

`SequenceInputStream` class defines two constructors that are different from any other `InputStream`:

2. SequenceInputStream(Enumeration e):
This constructor creates a new `SequenceInputStream` that takes an `Enumeration` of InputStreams as an argument. It reads the data of an enumeration whose type is `InputStream`.

SequenceInputStream Methods

In addition to methods inherited from `InputStream` class, `SequenceInputStream` class also defines some useful methods in Java. They are as follows:

SequenceInputStream Methods

Method	Description
int available()	This method returns the actual number of bytes that can be read (or skipped over) from the current underlying input stream without blocking by the next invocation of a method.
void close()	This method closes the input stream and releases any system resources associated with the stream.
int read()	This method reads the next byte of data from the underlying input stream.
int read(byte[ ] b, int n, int m)	This method reads up to m bytes of data from the underlying input stream into an array of bytes.

SequenceInputStream 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: Read Two Files to Console

1. Let’s create a program to take data of two files file1 and file2 and display that data on the console. Look at the following source code.

Example 1: Read Two Files to Console

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.SequenceInputStream;
public class SequenceInputStreamTester1 {
   public static void main(String[] args) throws IOException {
       FileInputStream file1 = new FileInputStream("./src/infile1.txt");
       FileInputStream file2 = new FileInputStream("./src/infile2.txt");

       // Concatenate file1 and file2 into file3.
       SequenceInputStream file3 = new SequenceInputStream(file1, file2);
       int i;
       while((i = file3.read()) != -1) {
           System.out.print((char)i);
       }
       file3.close();
       file2.close();
       file1.close();
   }
}
```

Example 1: Read Two Files to Console

Output:

```
Welcome to Java Programming.
This is an example of Java SequenceInputStream class.
```

file1: Welcome to Java Programming.

file2: This is an example of Java SequenceInputStream class.

This program is a simple example of the process of concatenation of two files into a single file. In this program, we have created two objects of class `FileInputStream` for files "`file1.txt`" and "`file2.txt`".

The two `FileInputStream` object reference variables `file1` and `file2` are passed as arguments to the constructor of `SequenceInputStream` to create a single object input stream file.

The file `file3` now contains the combination of `file1` and `file2` and displays data of `file3` on the console after concatenation.

Example 2: Concat, Buffer, & Display Two Files

2. Let’s take an example program where we will concat, buffer, and display the data of two independent files.

Example 2: Concat, Buffer, & Display Two Files

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.SequenceInputStream;
public class SequenceInputStreamTester2 {
   public static void main(String[] args) throws IOException {
       FileInputStream file1 = new FileInputStream("./src/infile1.txt");
       FileInputStream file2 = new FileInputStream("./src/infile2.txt");

       // Concatenate file1 and file2 into file3.
       SequenceInputStream file3 = new SequenceInputStream(file1, file2);

       // Now create buffered input and output stream objects.
       BufferedInputStream bis = new BufferedInputStream(file3);
       BufferedOutputStream bos = new BufferedOutputStream(System.out);

       // Reading and writing until the end of buffers.
       int i;
       while( (i = bis.read()) != -1) {
           bos.write( (char) i );
       }
       bis.close();
       bos.close();
       file1.close();
       file2.close();
   }
}
```

Example 2: Concat, Buffer, & Display Two Files

Output:

```
Welcome to Java Programming.
This is an example of Java SequenceInputStream class.
```

file1: Welcome to Java Programming.

file2: This is an example of Java SequenceInputStream class.

This program shows the entire process of concatenation, buffering, and displaying data of two independent files in a proper manner.

Example 3: Read Two Files & Write into New File

3. Let’s take an example program we will read the data from two files “file1.txt” and “file2.txt” and writes them into another file “fileout.txt”. Look at the source code to understand better.

Example 3: Read Two Files & Write into New File

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.SequenceInputStream;
public class SequenceInputStreamTester3 {
   public static void main(String[] args) throws IOException {
       FileInputStream file1 = new FileInputStream("./src/infile1.txt");
       FileInputStream file2 = new FileInputStream("./src/infile2.txt");
       FileOutputStream fos = new FileOutputStream("./src/outfile.txt");
       SequenceInputStream file3 = new SequenceInputStream(file1,file2);

       // Now create buffered input and output stream objects.
       BufferedInputStream bis = new BufferedInputStream(file3);
       BufferedOutputStream bos = new BufferedOutputStream(fos);

       // Reading and writing until end of buffers.
       int i;
       while((i = bis.read()) != -1){
           bos.write((char)i);
       }
       bis.close();
       bos.close();
       fos.close();
       file1.close();
       file2.close();
       file3.close();
       System.out.println("Successfully written...");
   }
}
```

Example 3: Read Two Files & Write into New File

Output:

```
Successfully written...
```

file1: Welcome to Java Programming.

file2: This is an example of Java SequenceInputStream class.

fileout.txt:

Welcome to Java Programming world.
This is an example of Java SequenceInputStream class.

Example 4: Read with Enumeration

4. Let’s create a program to read data from the files using Enumeration. If you need to read data more than two files, enumeration can be used.

Enumeration object can be created by calling elements() method provided by vector class. Look at the source code.

Example 4: Read with Enumeration

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.SequenceInputStream;
import java.util.Enumeration;
import java.util.Vector;
public class SequenceInputStreamTester4 {
   public static void main(String[] args) throws IOException {
       FileInputStream file1 = new FileInputStream("./src/infile1.txt");
       FileInputStream file2 = new FileInputStream("./src/infile2.txt");
       FileInputStream file3 = new FileInputStream("./src/infile3.txt");
       FileInputStream file4 = new FileInputStream("./src/infile4.txt");
       // Creating an object of Vector class for all the streams.
       Vector v = new Vector();
       v.add(file1);
       v.add(file2);
       v.add(file3);
       v.add(file4);

       // Creating enumeration object by calling the elements method of vector class.
       Enumeration en = v.elements();

       // Passing the enumeration object to the constructor of SequenceInputStream class.
       SequenceInputStream sis = new SequenceInputStream(en);

       int i;
       while((i = sis.read()) != -1){
           System.out.print( (char) i );
       }
       sis.close();
       file1.close();
       file2.close();
       file3.close();
       file4.close();
   }
}
```

Example 4: Read with Enumeration

Output:

```
Welcome to Java Programming.                          ← file1
This is an example of Java SequenceInputStream class. ← file2
Hello World                                           ← file3
Bye now!                                              ← file4
```

END