CSCI 2120:
Software Design & Development II

UNIT3: I/O management

io api

RandomAccessFile

Overview

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

Introduction

- **RandomAccessFile in Java** is a class that allows data to be read from and written to at any location in the file.

- In other simple words, `RandomAccessFile` class allows creating files that can be used for reading and writing data with random access.

- Random access means access to stored data in any order that the user desires. That is, we can access data in the file in any order while using the file. Such a file is known as random access file.

- A **random-access file** is a file that can be used for both reading and writing using `RandomAccessFile` class in java.

Why RandomAccessFile class in Java?

- So far we have discussed all I/O streams that can be used either for “read only” or for “write only” operations and not for both purposes simultaneously.
- That is, all of the I/O streams we have used so far are known as read-only or write-only streams. These streams are called sequential streams in Java.
- A file that is read or written using a sequential stream is called a sequential-access file. The data of a sequential-access file cannot be updated.
- So, for reading and writing data simultaneously, Java provides the `RandomAccessFile` class. Using randomaccessfile class, we can change the location in the file at which the next read or write operation will occur.
- `RandomAccessFile` class was added in Java 1.0 version. It is present in `java.io.RandomAccessFile` package.

RandomAccessFile class declaration

`RandomAccessFile` class extends `Object` class and implements `DataInput` and `DataOutput` interfaces with additional methods to support random access. Therefore, it can be used for both reading and writing simultaneously.

It also implements the `Closeable` and `AutoCloseable` interfaces. It is not inherited from `InputStream` or `OutputStream` superclass.

The general syntax to declare `RandomAccessFile` class in java is as follows:

```
public class RandomAccessFile
       extends Object
       implements DataOutput, DataInput, Closeable, AutoCloseable
```

File Pointer in RandomAccessFile class

- A random-access file consists of a sequence of bytes. It supports a special pointer known as file pointer. A file pointer indicates the current position (location) in the file.

- It is positioned at one of these bytes in the file and can be moved to any arbitrary position in the file prior to reading or writing.

- In other words, a read or write operation takes place at the location of the file pointer. The file pointer can be moved using `seek()` method provided by `RandomAccessFile` class.

- When a file is first created, the file pointer is set to 0, indicating the beginning of the file. When we read or write data to the file using read or write methods, the file pointer moves forward to the next data item (i.e. next byte).

File Pointer in RandomAccessFile class

For example, if we read an int value using `readInt()` method from the file, JVM reads 4 bytes using file pointer, and now the file pointer is 4 bytes ahead of the previous position, as shown in figure below.

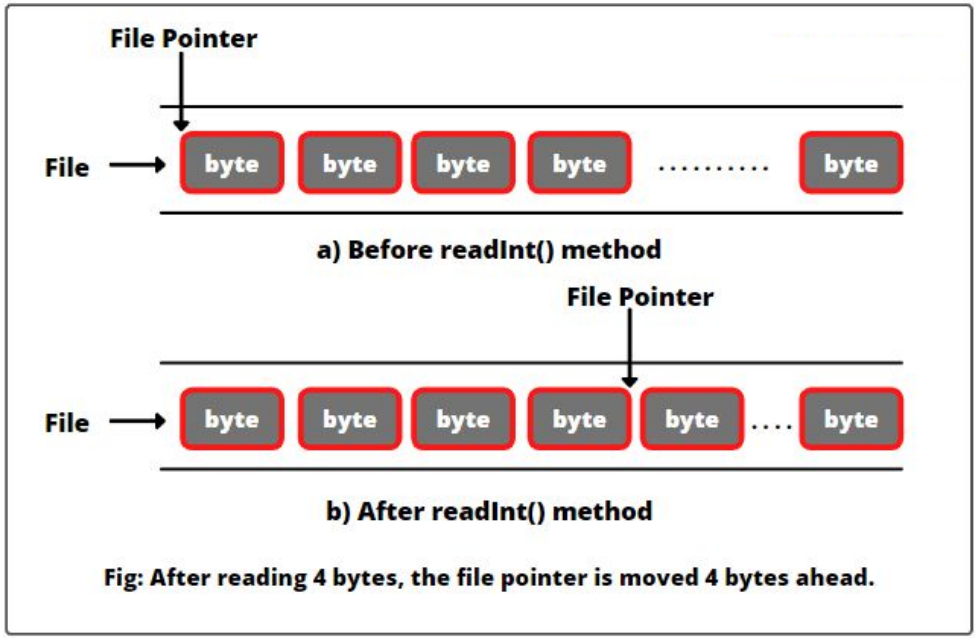

Fig: After reading 4 bytes, the file pointer is moved 4 bytes ahead.

For a `RandomAccessFile raf`,

`raf.seek(position)`
moves the file pointer to a specified location.

`raf.seek(0)`
moves the file pointer to the beginning of the file

`raf.seek(raf.length())`
moves it to the end of the file.

RandomAccessFile Constructors

`RandomAccessFile` class defines the following constructors in Java. They are as follows:

---

1. RandomAccessFile(File fileObj, String mode):
This constructor creates a random access file stream with the specified `File` object and `mode`. Here, `fileObj` defines the name of the file to open as a `File` object.

---

2. RandomAccessFile(String filename, String mode):
This constructor creates a random access file stream with the specified `filename` string and `mode`.

In both cases, the `mode` determines what kind of file access is permitted. For example:

- If the `mode` (in string) is "`r`", the file can be read-only, but not written.
- If it is "`rw`", the file is opened in read-write mode.
- If it is "`rws`", the file is opened for read-write and every change to the file's data will be immediately written to the physical device.

RandomAccessFile Constructors

The general syntax to create a new file stream for both read-write operations is given below:

```
RandomAccessFile raf = new RandomAccessFile("myfile.dat", "rw");
```

Here, “`rw`” is a mode string that has given as a parameter to the constructor of `RandomAccessFile` class. When the file is opened by the above syntax, the file pointer is automatically positioned at the beginning of the file.

The above syntax allows the program to read from and write to the file `myfile.dat`. If `myfile.dat` already exists, `raf` is created to access it; if `myfile.dat` does not exist, a new file named `myfile.dat` is constructed, and `raf` is created to access the new file.

RandomAccessFile Methods

`RandomAccessFile` class in Java defines several important additional methods to support random access. They are as follows:

RandomAccessFile Methods

Method	Description
void close()	This method closes the random access file stream and releases any system resources associated with the stream.
FileChannel getChannel()	This method returns the unique FileChannel object associated with the file.
FileDescriptor getFD()	This method returns the opaque file descriptor object associated with the underlying stream.
long getFilePointer()	This method returns the current offset (in bytes) from the beginning of the file to where the next read or write occurs.
long length()	This method returns the length (number of bytes) of this file.
int read()	It reads a byte of data from this file and returns –1 at the end of the stream.
int read(byte[ ] b)	It reads up to b.length bytes of data from this file into an array of bytes.

RandomAccessFile Methods

Method	Description
int read(byte[ ] b, int n, int m)	It reads up to m bytes of data from this file into an array of bytes, starting from nth byte
boolean readBoolean()	This method reads a boolean value from this file.
byte readByte()	This method reads a signed eight-bit value from this file.
char readChar()	This method reads a character value from this file.
double readDouble()	The readDouble() method reads a double value from this file.
float readFloat()	The readFloat() method reads a float value from this file.
void readFully(byte[ ] b)	It reads b.length bytes from this file into the array of byte, starting at the current file pointer.
void readFully(byte[ ] b, int n, int m)	It reads exactly m bytes from this file into the byte array, starting at the current file pointer.

RandomAccessFile Methods

Method	Description
String readLine()	The readLine() method reads the next line of text from this file.
String readUTF()	The readUTF() method reads in a string from this file.
void seek(long pos)	This method sets the file-pointer (in bytes specified in pos), measured from the beginning of this file, at which the next read or write occurs.
void setLength(long length)	The setLength() sets a new length for this file.
int skipBytes(int n)	This method skips over n bytes of input discarding the skipped bytes.
void write(byte[ ] b)	This method writes b.length bytes from the specified byte array to this file, starting at the current file pointer.
void write(byte[ ] b, int n, int m)	This method writes m bytes from the specified byte array starting at nth byte to this file.

RandomAccessFile Methods

Method	Description
void write(int b)	The write() method writes the specified byte to this file.
void writeBoolean(boolean v)	It writes a boolean value (one byte) to the file.
void writeByte(int v)	It writes a byte value (one byte) to the file.
void writeBytes(String s)	It writes the string value to the file as a sequence of bytes.
void writeChar(int v)	It writes a char to the file as a two-byte value, high byte first.
void writeChars(String s)	It writes a string to the file as a sequence of characters.
void writeDouble(double v)	It first converts the specified double value to a long value using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.

RandomAccessFile Methods

Method	Description
void writeFloat(float v)	It first converts the specified float value to an int value using the floatToIntBits method in class Float and then writes that int value to the file as a four-byte quantity, high byte first.
void writeInt(int v)	It writes an int value to the file as four bytes, high byte first.
void writeLong(long v)	It writes a long value to the file as eight bytes, high byte first.
void writeShort(int v)	It writes a short value to the file as two bytes, high byte first.
void writeUTF(String str)	It writes a string value to the file using modified UTF-8 encoding in a machine-independent manner.

Example 1: Create, Read, Write data to File

1. Let’s take an example programs based on the above methods defined by `RandomAccessFile` class in Java.

Example 1: Create, Read, Write data to File

```
import java.io.IOException;
import java.io.RandomAccessFile;
public class RandomAccessFileTester1 {
   public static void main(String[] args) throws IOException {
       // Create a random access file object.
       RandomAccessFile file = new RandomAccessFile("myfile.dat", "rw");

       // Writing to the file.
       file.writeChar('S');
       file.writeInt(2222);
       file.writeDouble(222.22);
       file.seek(0); // Moving file pointer to the beginning.

       // Reading data from the file.
       System.out.println(file.readChar());
       System.out.println(file.readInt());
       System.out.println(file.readDouble());
       file.seek(2); // Moving the file pointer to the second item.
       System.out.println(file.readInt());

       // Go to the end and append a boolean value to the file.
       file.seek(file.length());
       file.writeBoolean(true);

       // Since pointer is at end, beyond the 4th item, brings the file pointer to the 4th item in the file.
       file.seek(4); // Moving the file pointer to the 4th item.
       System.out.println(file.readBoolean()); // Reading 4th item.
       file.close(); // Closing stream.
   }
}
```

Example 1: Create, Read, Write data to File

Output:

```
S
2222
222.22
2222
true
```

Explanation:

This program opens a random access file and then performs the read-write operations.

a) Writes three items of data to the file using writeChar(), writeInt(), and writeDouble() methods.

b) Brings the file pointer to the beginning of the file using seek() method.

c) Reads all three items from the file and prints them on the console.

d) Now takes the file pointer to the second item (2222) and then reads and prints the second item in the file.

e) Moves the pointer at the end using length() method and then adds the second item to the file.

f) Since there are now four items in the file and pointer is at the end, that is, beyond the fourth item, therefore, brings the file pointer to the fourth item and prints it.

g) At the end, closes the file using close() method. The output would appear on the console.

Example 2: Append data to File

2. Let’s create a program to append an item to an existing file using `RandomAccessFile` class. Look at the following source code.

Example 2: Append data to File

```
import java.io.IOException;
import java.io.RandomAccessFile;
public class RandomAccessFileTester2 {
   public static void main(String[] args) throws IOException {
       // Create a random access file object.
       RandomAccessFile file = new RandomAccessFile("cityfile.txt", "rw");

       file.seek(file.length()); // Moving file pointer to the end.
       file.writeBytes("New Orleans\n"); // Appending New Orleans.
       file.close();
       System.out.println("Successfully written...");
   }
}
```

Example 2: Append data to File

Output:

```
Successfullly written...
```

Explanation:

`cityfile.txt` contents after 1st execution:

```
1  New Orleans
2
```

```
Successfullly written...
```

`cityfile.txt` contents after 2nd execution:

```
1  New Orleans
2  New Orleans
3
```

Example 3: File Pointer for Random Access

3. Let’s take another example program based on the above methods of random access file for practice.

Example 3: File Pointer for Random Access

```
import java.io.IOException;
import java.io.RandomAccessFile;
public class RandomAccessFileTester3 {
   public static void main(String[] args) throws IOException {
       // Create a random access file object.
       RandomAccessFile file = new RandomAccessFile("iofile.dat", "rw");
       file.setLength(0);                                          //Clear file to destroy old data if exists.
       for(int i = 0; i <= 10; i++) {
           file.writeInt(i);                                      // Write new integer values to the file.
       }

       System.out.println("Current length of file: " +file.length()); // Print the current length of the file.
       file.seek(0);                                               // Set file pointer to beginning.
       System.out.println("First number: " +file.readInt());       // Get 1st number from file &
       System.out.println("Second number: " +file.readInt());      // Get the second number from the file.
       file.seek(9 * 4);                                           // Move the file pointer to 9th position.
       System.out.println("Ninth number: " +file.readInt());       // Retrieve the number at 9th position
       file.writeInt(222);                                         // Modify the tenth number
       file.seek(10 * 4);                                          // Moving pointer to 10th position.
       System.out.println("Tenth number: " +file.readInt());
       file.seek(file.length());                                   // Append a new number at the end of the file.
       file.writeInt(333);
       System.out.println("New length of file: " +file.length());  // Print the new length of the file.
       file.seek(11 * 4);                                          // Move the file pointer to new number.
       System.out.println("New number: "+file.readInt());          // Retrieve the new number.
   }
}
```

Example 3: File Pointer for Random Access

Output:

```
Current length of file: 44
First number: 0
Second number: 1
Ninth number: 9
Tenth number: 222
New length of file: 48
New number: 333
```

Explanation:

a) A RandomAccessFile creates a file named infile.dat with mode rw to allow both read and write operations.

b) file.setLength(0) sets the length to 0 to clear file. It destroys the old data of the file.

c) The for loop writes 11 int values from 0 to 10 into the file. Since each int value takes 4 bytes, the total length of the file returned from file.length() is now 44 as shown in the output.

d) Calling file.seek(0) sets the file pointer to the beginning of the file. file.readInt() reads the first value and moves the file pointer to the next number. The second number is read as 1.

e) file.seek(9 * 4) moves the file pointer to the ninth number. file.readInt() reads the ninth number and moves the file pointer to the tenth number.

f) file.write(222) writes a new tenth number at the current position. The previous tenth number is destroyed.

g) file.seek(file.length()) moves the file pointer to the end of the file.

h) file.writeInt(333) writes a 333 to the file. Now the length of the file is increased by 4. So, file.length() returns 48.

i) file.seek(11 * 4) moves the file pointer to the eleventh number. The new eleventh number, 333, is displayed as shown in the above output.

END