The Java™ Tutorials

Trail: Essential Classes
Lesson: Basic I/O
Section: File I/O (Featuring NIO.2)

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

Random Access Files

Random access files permit nonsequential, or random, access to a file's contents. To access a file randomly, you open the file, seek a particular location, and read from or write to that file.

This functionality is possible with the `SeekableByteChannel` interface. The `SeekableByteChannel` interface extends channel I/O with the notion of a current position. Methods enable you to set or query the position, and you can then read the data from, or write the data to, that location. The API consists of a few, easy to use, methods:

- `position` – Returns the channel's current position
- `position(long)` – Sets the channel's position
- `read(ByteBuffer)` – Reads bytes into the buffer from the channel
- `write(ByteBuffer)` – Writes bytes from the buffer to the channel
- `truncate(long)` – Truncates the file (or other entity) connected to the channel

Reading and Writing Files With Channel I/O shows that the `Path.newByteChannel` methods return an instance of a `SeekableByteChannel`. On the default file system, you can use that channel as is, or you can cast it to a `FileChannel` giving you access to more advanced features, such as mapping a region of the file directly into memory for faster access, locking a region of the file, or reading and writing bytes from an absolute location without affecting the channel's current position.

The following code snippet opens a file for both reading and writing by using one of the `newByteChannel` methods. The `SeekableByteChannel` that is returned is cast to a `FileChannel`. Then, 12 bytes are read from the beginning of the file, and the string "I was here!" is written at that location. The current position in the file is moved to the end, and the 12 bytes from the beginning are appended. Finally, the string, "I was here!" is appended, and the channel on the file is closed.

```
String s = "I was here!\n";
byte data[] = s.getBytes();
ByteBuffer out = ByteBuffer.wrap(data);

ByteBuffer copy = ByteBuffer.allocate(12);

try (FileChannel fc = (FileChannel.open(file, READ, WRITE))) {
    // Read the first 12
    // bytes of the file.
    int nread;
    do {
        nread = fc.read(copy);
    } while (nread != -1 && copy.hasRemaining());

    // Write "I was here!" at the beginning of the file.
    fc.position(0);
    while (out.hasRemaining())
        fc.write(out);
    out.rewind();

    // Move to the end of the file.  Copy the first 12 bytes to
    // the end of the file.  Then write "I was here!" again.
    long length = fc.size();
    fc.position(length-1);
    copy.flip();
    while (copy.hasRemaining())
        fc.write(copy);
    while (out.hasRemaining())
        fc.write(out);
} catch (IOException x) {
    System.out.println("I/O Exception: " + x);
}
```

Previous page: Reading, Writing, and Creating Files
Next page: Creating and Reading Directories