Advanced Java Programming

Random Access Files

lec(10)

Learning about Random Access Files

- Sequential access files
  - Access records sequentially from beginning to end
  - Good for batch processing (e.g. customer bills)
    - Same tasks with many records one after the other
  - Inefficient for many applications
- Realtime applications
  - Require immediate record access while client waits

Learning about Random Access Files (cont'd.)

- Random access files
  - Records can be located in any order
  - Also called direct access or instant access files
  - Use **FileChannel** class to create random access files
- File channel object
  - Used for both reading and writing
  - reading/writing can start at any specified position

FileChannel Methods

FileChannel method	Description
`FileChannel open(Path file, OpenOption... options)`	Opens or creates a file, returning a file channel to access the file
`long position()`	Returns the channel's file position
`FileChannel position(long newPosition)`	Sets the channel's file position
`int read(ByteBuffer buffer)`	Reads a sequence of bytes from the channel into the buffer
`long size()`	Returns the size of the channel's file
`int write(ByteBuffer buffer)`	Writes a sequence of bytes to the channel from the buffer

Table 13-7 Selected `FileChannel` methods

ByteBuffer Class

- a ByteBuffer is simply a holding place for bytes waiting to be read or written.
- An array of bytes can be wrapped into a ByteBuffer using the ByteBuffer wrap() method.
- Wrapping a byte array into a buffer causes changes to the buffer to change the array as well, and causes changes to the array to change the buffer.

Accessing a file Randomly

This requires the following steps:

A. use the Path class newByteChannel() method to create a file using StandardOpenOption arguments and return ByteChannel object.

B. The ByteChannel returned by the newByteChannel() method can then be cast to a FileChannel using a statement similar to the following:

```
FileChannel fc = (FileChannel) file.newByteChannel(READ, WRITE);
```

Accessing a file Randomly (Cont.)

C. Create a byte array. For example, a byte array can be built from a String using the getBytes() method as follows:
   - String s = "XYZ";
   - byte[] data = s.getBytes();

D. The byte array can be wrapped into a ByteBuffer as follows:
   - ByteBuffer out = ByteBuffer.wrap(data);

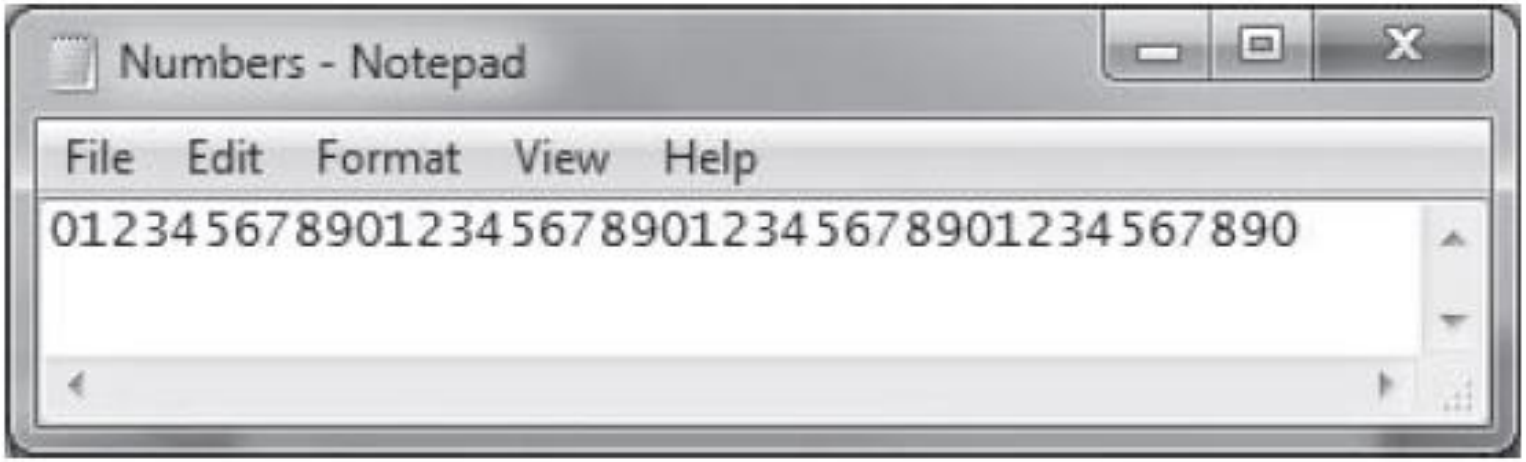
E. Then the filled ByteBuffer can be written to the declared FileChannel with a statement such as the following:
   - fc.write(out);

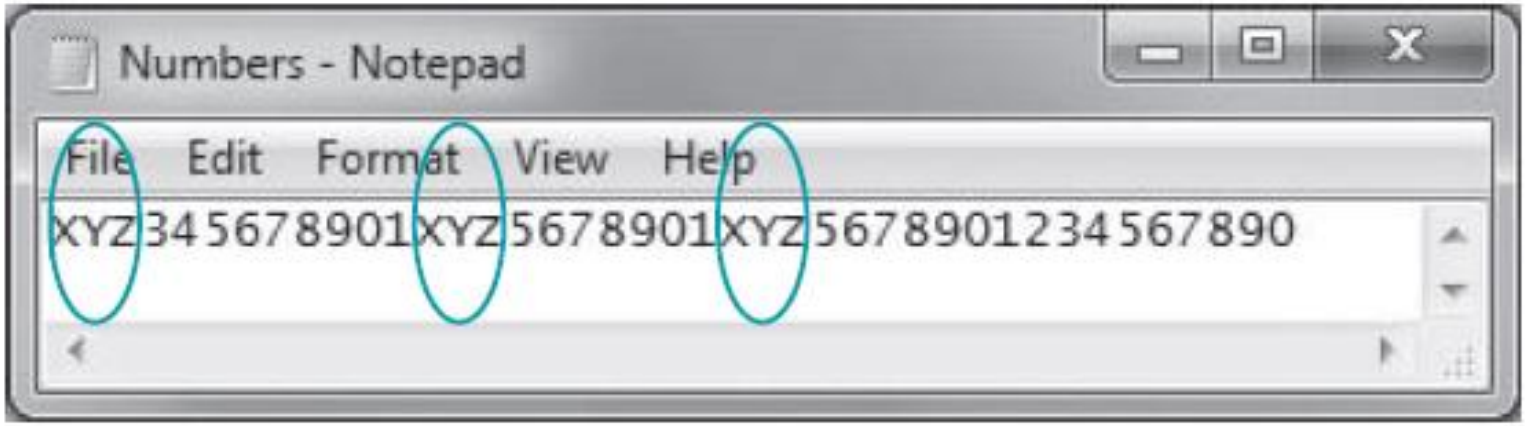
Accessing a file Randomly (Cont.)

e. Then you can test whether a ByteBuffer's contents have been used up by checking the **hasRemaining()** method.

f. After you have written the contents of a ByteBuffer, you can write the same ByteBuffer contents again by using the rewind() method to reposition the ByteBuffer to the beginning of the buffer.

```
import java.nio.file.*;
import java.io.*;
import java.nio.channels.FileChannel;
import java.nio.ByteBuffer;
import static java.nio.file.StandardOpenOption.*;
public class RandomAccessTest
{
   public static void main(String[] args)
   {
      Path file =
         Paths.get("C:\\Java\\Chapter.13\\Numbers.txt");
      String s = "XYZ";
      byte[] data = s.getBytes();
      ByteBuffer out = ByteBuffer.wrap(data);
      FileChannel fc = null;
      try
      {
          fc = (FileChannel)file.newByteChannel(READ, WRITE);
          fc.position(0);
          while(out.hasRemaining())
             fc.write(out);
          out.rewind();
          fc.position(22);
          while(out.hasRemaining())
             fc.write(out);
          out.rewind();
          fc.position(12);
          while(out.hasRemaining())
             fc.write(out);
          fc.close();
      }
      catch (Exception e)
      {
          System.out.println("Error message: " + e);
      }
   }
}
```


Numbers - Notepad
File Edit Format View Help
01234567890123456789012345678901234567890


Numbers - Notepad
File Edit Format View Help
XYZ345678901XYZ5678901XYZ567890123456789O