ABOUT NIKOS MARAVITSAS

Nikos has graduated from the Department of Informatics and Telecommunications of The National and Kapodistrian University of Athens. During his studies he discovered his interests about software development and he has successfully completed numerous assignments in a variety of fields. Currently, his main interests are system's security, parallel systems, artificial intelligence, operating systems, system programming, telecommunications, web applications, human – machine interaction and mobile development.

Java write to File Example

Posted by: Nikos Maravitsas in io April 8th, 2014

In this example we are going to investigate several methods to write a File in Java. We are going to list some of the older, before Java 7 and NIO, methods to write to a text or a binary file as well as some of the modern ones. You are also going to notice that we use some methods to write to larger files and other methods to write to smaller files. Efficiently writing large amounts of data to files usually requires some buffering, which is not necessary for smaller files.

Ok, let's start with the older, before NIO methods.

1. Using classic IO libraries

1.1. Write files using FileWriter and FileOutputStream

Let's see the code and then analyze it:

WriteToFileExample.java:

```
package com.javacodegeeks.core.writeToFile;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.Writer;
import java.util.ArrayList;
import java.util.List;

public class WriteToFileExample {

    private static final String FILEPATH = "C:\\Users\\nikos\\Desktop\\TestFiles\\testFile.txt";

    public static void main(String[] args) throws IOException {

        String str1 = "abc";
        String str2 = "asdasfasfasfa";

        List<String> list = new ArrayList<String>();
        list.add(str1);
        list.add(str2);

        //useBufferedFileOutPutStream(list, FILEPATH);
        useFileOutPutStream(str1,FILEPATH);

    }

    /**
     * Write a small string to a File - Use a FileWriter
     */
    public static void useFileWriter(String content, String filePath) {
        Writer writer = null;
```

GET THE JAVA SKILLS YOU NEED IN 2016. Start Learning N

```
            writer.write(content);

        } catch (IOException e) {

            System.err.println("Error writing the file : ");
            e.printStackTrace();

        } finally {

            if (writer != null) {
                try {
                    writer.close();
                } catch (IOException e) {

                    System.err.println("Error closing the file : ");
                    e.printStackTrace();
                }
            }

        }
    }

    /**
     * Write a big list of Strings to a file - Use a BufferedWriter
     */
    public static void useByfferedFileWriter(List<String> content,
            String filePath) {

        File file = new File(filePath);
        Writer fileWriter = null;
        BufferedWriter bufferedWriter = null;

        try {

            fileWriter = new FileWriter(file);
            bufferedWriter = new BufferedWriter(fileWriter);

            // Write the lines one by one
            for (String line : content) {
                line += System.getProperty("line.separator");
                bufferedWriter.write(line);

                // alternatively add bufferedWriter.newLine() to change line
            }

        } catch (IOException e) {
            System.err.println("Error writing the file : ");
            e.printStackTrace();
        } finally {

            if (bufferedWriter != null && fileWriter != null) {
                try {
                    bufferedWriter.close();
                    fileWriter.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

    }

    /**
     * Write raw data to a small file - use FileOutPutStream
     */
    public static void useFileOutPutStream(String content, String filePath) {

        OutputStream outputStream = null;

        try {

            outputStream = new FileOutputStream(new File(filePath));
            outputStream.write(content.getBytes(), 0, content.length());

        } catch (FileNotFoundException e) {
            System.err.println("Error Opening the file : ");
            e.printStackTrace();
        } catch (IOException e) {
            System.err.println("Error writing  the file : ");
            e.printStackTrace();
        } finally {

            if (outputStream != null) {
                try {
                    outputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

        }

    }
```

CAREER OPPORTUNITIES

Junior Software Developer- Cloud for S
Job**SAP**
Palo Alto, CA
Oct, 26
Engineer-Java**American Express**
Arizona
Oct, 26
Junior Developer- Core Mobile SDK Jo
Dublin, CA
Oct, 25
Database Developer - Server-Side Jav
Stanley
New York, NY
Sep, 02
JAVA DEVELOPER**BirlaSoft**
New Jersey
Oct, 19
Back-End Developer**GE Digital**
Michigan
Oct, 19
Software Engineer - Back End**GE Digi**
Michigan
Oct, 21
Entry Level Software Engineer**Ford M**
Company
Dearborn, MI
Oct, 08
Senior Java Developer**Barclays**
New York, NY
Oct, 17
Java Developer**Configuration Mana**
Englewood Cliffs, NJ
Oct, 25

1 2 ... 5955 »

Keyword ...
Location ...
Country ...

Filter Results

jobs by indeed

```java

    /**
     * Write character data to a big file - use BufferedWriter
     */
    public static void useBufferedFileOutPutStream(List<String> content, String filePath) {
        Writer writer = null;

```

```java
                    new FileOutputStream(filePath), "utf-8"));

            for (String line : content) {
                line += System.getProperty("line.separator");
                writer.write(line);
            }

        } catch (IOException e) {

        } finally {

            if (writer != null) {
                try {
                    writer.close();
                } catch (Exception e) {

                }
            }
        }
    }
      /**
     * Write raw data to a big file - use BufferedOutputStream
     */
    public static void useBufferedOutPutStream(List<String> content,String filePath) {
        BufferedOutputStream bout = null;
        try {


            bout = new BufferedOutputStream( new FileOutputStream(filePath) );

            for (String line : content) {
                line += System.getProperty("line.separator");
                bout.write(line.getBytes());
            }

        } catch (IOException e) {

        } finally {

            if (bout != null) {
                try {
                    bout.close();
                } catch (Exception e) {

                }
            }
        }

    }

}
```

In the above example we basically use two different methods:

```
FileWriter
```

to write to character/text files.

```
FileOutputStream
```

to write raw data.

In order to buffer the writes of the above classes we use a

```
BufferedWriter
```

for character streams and

```
BufferedOutputStream
```

for raw data streams. With

```
BufferedWriter
```

, we simply use an internal buffer to pack the data we want to write and reduce real IO operations, which essentially saves time. So as you can imagine, this is very useful when creating a write-intensive application that writes large amounts of data to files.

```
BufferedWriter
```

is generally created with the default buffer size, which is sufficient for most applications. If you want you can specify the size of the internal buffer using this constructor instead :

```
BufferedWriter(Writer out, int sz)
```

where

```
sz
```

is the size of the buffer in bytes.

to wrap

```
FileOutputStream
```

. Furthermore when writing characters to a file without buffering, on every

```
write
```

invocation the system will perform a conversion from characters to bytes. Buffering will also reduce the amount of conversions performed.

1.2. Using RandomAccessFile

```
RandomAccessFile
```

was introduced to support random accessed to files. As the javadoc says "a random access file behaves like a large array of bytes stored in the file system". It is a very convenient class to write and also read files in arbitrary positions.

Let's see how :

WriteToFileNIO.java:

```java
public static void writeWithRandmoAccessFile( String content, String filePath) {

        try (RandomAccessFile randomAccessFile = new RandomAccessFile(new File(filePath), "rw")) {

        // move the cursor to the end of the file
        // you can move the cursor to any position inside the file to write at random positions
        randomAccessFile.seek(randomAccessFile.length());

        randomAccessFile.write(content.getBytes());

        // alternatively you can use randomAccessFile.writeChars(content)
        // or randomAccessFile.writeUTF(content);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

As you can see we open a

```
RandomAccessFile
```

in read-write mode. Then, we simply move the cursor to the end of the file to append the new content (using

```
seek
```

method) and we simply write some bytes to it with its

```
write
```

method. Note that

```
RandomAccessFile
```

was introduced in JDK1.0.

That's it. Now let's take a see how can write to files in Java using modern NIO classes.

2. Write files using NIO

NIO introduced several classes that made File manipulation easier and highly efficient. We are also going to introduce here the try-with resources syntax.

Let's see the code :

2.1 Using java.nio.file.Files class

WriteToFileNIO.java:

```java
package com.javacodegeeks.core.writeToFile;

import java.io.BufferedOutputStream;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStream;
import java.io.UnsupportedEncodingException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
```

```java
public class WriteToFileNIO {

    private static final String FILEPATH = "C:\\Users\\nikos\\Desktop\\TestFiles\\testFile.txt";

    public static void main(String[] args) throws IOException {

        String str1 = "abc";
        String str2 = "aipcipasincinainsovusdvweviasbdoviuabsudviuadv";

        List<String> list = new ArrayList<String>();
        list.add(str1);
        list.add(str2);

        bufferedWrite(list, FILEPATH);
    }

    /**
     * Write a small string to a File - Use a FileWriter
     */
    public static void simpleWrite(String content, String filePath) {
        Path fileP = Paths.get(filePath);
        try {

            Files.write(fileP, content.getBytes("utf-8"));

        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Write a big list of Strings to a file - Use a BufferedWriter
     */
    public static void bufferedWrite(List<String> content, String filePath) {

        Path fileP = Paths.get(filePath);
        Charset charset = Charset.forName("utf-8");

        try (BufferedWriter writer = Files.newBufferedWriter(fileP, charset)) {

            for (String line : content) {
                writer.write(line, 0, line.length());
                writer.newLine();
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Write raw data to file - use OutputStream
     */
    public static void writeWithOutputStream(String content, String filePath) {

        Path fileP = Paths.get(filePath);

        try (OutputStream outputStream = Files.newOutputStream(fileP)) {

            outputStream.write(content.getBytes());

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Write raw data to file using BufferedOutputStream
     */
    public static void writeWithBufferedOutputStream(List<String> content, String filePath) {

        Path fileP = Paths.get(filePath);

        try (BufferedOutputStream outputStream = new BufferedOutputStream(Files.newOutputStream(fileP))) {

            for (String line : content) {
                outputStream.write(line.getBytes());
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

        /**
     * Write a string list to a File
     */
    public static void simpleWriteListOfString(List<String> content, String filePath) {
        Path fileP = Paths.get(filePath);
        Charset charset = Charset.forName("utf-8");

        try {

            Files.write(fileP, content,charset);

        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (IOException e) {
```

As you can see things are much simpler because of the new NIO `Files` class and the new try-with resource syntax. In try-with resource syntax, when you open a resource in the `try()` clause, the resource will be automatically closed when the flow of the program exits the `try` region. You don't have to create the `finally` block to release the resources as we did previously.

2.2. Using FileChannel

Now let's see how you can use `FileChannel`, which is a very interesting class of the NIO package. It basically connects a channel of bytes to a file and enables both reading and writing from/to files. You can view it as an alternative to `FileOuputStream`. A major difference is that a `FileChannel` connects an allocated byte buffer to the file and it holds the current position of the cursor in the file.

WriteToFileNIO.java:

```
public static void writeWithFileChannel(String content, String filePath) {

    try (RandomAccessFile randomAccessFile = new RandomAccessFile(new File(filePath), "rw")) {

        // move the cursor to the end of the file
        randomAccessFile.seek(randomAccessFile.length());

        // obtain the a file channel from the RandomAccessFile
        try (FileChannel fileChannel = randomAccessFile.getChannel()) {

            ByteBuffer buf = ByteBuffer.allocate(512);
            buf.clear();
            buf.put(content.getBytes());

            buf.flip();

            while (buf.hasRemaining()) {
                fileChannel.write(buf);
            }

        } catch (IOException e) {
            e.printStackTrace();
        }

    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

As you can see we first create a `RandomAccessFile` and obtain a `FileChannel` from it. Then, we allocate a `ByteBuffer` of 512 bytes. Finally we write the contents of the byte buffer to the file. For performance reasons, it's not guaranteed that that the buffer will be written in its entirety to the file in a single

```
write
```

operation. That's why we've used the while loop, so as log as the buffer has remaining bytes in it, we simply append them to the file.

2.3. Using FileChannel and memory mapped file

In the following methods we are also going to use a

```
MappedByteBuffer
```

. This is a direct byte buffer that maps a memory region to a file region.

Let's see how you do this :

WriteToFileNIO.java:

```java
public static void writeWithMemMappedBuffer(String content, String filePath) {

    try (RandomAccessFile randomAccessFile = new RandomAccessFile(new File(filePath), "rw")) {

        // move the cursor to the end of the file
        randomAccessFile.seek(randomAccessFile.length());

        // obtain the a file channel from the RandomAccessFile
        try (FileChannel fileChannel = randomAccessFile.getChannel()) {

            // Map a content.getBytes().length byte region of the file to this memory buffer
            MappedByteBuffer memoryMappedbuffer = fileChannel.map(FileChannel.MapMode.READ_WRITE,
fileChannel.position(),content.getBytes().length);

            memoryMappedbuffer.put(content.getBytes());

        } catch (IOException e) {
            e.printStackTrace();
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

You can use this for performance sensitive applications. As you can see, there is no need to explicitly write the buffer to the file, that is something that the underlying systems does when it deems its necessary. You only have to manipulate the buffer, and the changes will be reflected to the file. Of course, reads are very efficient using this technique as well.

2.4. FileChannel and Direct Memory Access

There is an excellent article form IBM that describes Efficient data transfer through zero copy. In this case we are going to use

```
transferTo()/transferFrom()
```

method of

```
FileChannel
```

class. It's basic characteristics is that the it relies on the underlying system to access its DMA (Direct Memory Access) infrastructure. It might not work in all operating systems, but most modern ones offer such capabilities. What happens is that data are transferred directly from/to disc to the bus, avoiding CPU copies.

It's recommended to use that technique to transfer data from one source channel to another destination channel, e.g from a file to another file, from a file to a socket, from a database to a file and so on. But we are going to show you how to transfer data from a

```
String
```

to a

```
FileChannel
```

. We are going to consider the

```
String
```

as an

```
InputStream
```

from a data source (think hundreds of MB or GB long strings).

WriteToFileNIO.java:

```java
package com.javacodegeeks.core.writeToFile;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.RandomAccessFile;
import java.nio.channels.Channels;
import java.nio.channels.FileChannel;
import java.nio.channels.ReadableByteChannel;
import java.nio.charset.Charset;
```

```java

    public static void main(String[] args) throws IOException {

        String str2 = "aipcipasincinainsovusdvweviasbdoviuabsudviuadv";

        long numBytes = str2.getBytes().length;

        // obtain an inputstream from the string
        InputStream inputStream = new ByteArrayInputStream(str2.getBytes(Charset.forName("UTF-8")));

        writeWithFileChannerDMA(inputStream,FILEPATH,numBytes);

    }

    /**
     * Write to a file using a FileChanel and DMA
     */
    public static void writeWithFileChannerDMA(InputStream inputStream, String outputFile, long count) {

        try (RandomAccessFile randomAccessFile = new RandomAccessFile(new File(
                outputFile), "rw")) {

            // move the cursor to the end of the file
            randomAccessFile.seek(randomAccessFile.length());

            // obtain the a file channel from the RandomAccessFile
            try (
                    FileChannel fileChannel = randomAccessFile.getChannel();
                    ReadableByteChannel inputChannel = Channels.newChannel(inputStream);

                ) {

                fileChannel.transferFrom(inputChannel, 0, count);

            } catch (IOException e) {
                e.printStackTrace();
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

So, in the above example you've seen 4 major things:

1. In `main`, we've used a `ByteArrayInputStream` to obtain an input stream from a `String`.
2. We've used `Channels.newChannel` to obtain an `ReadableByteChannel` from the `InputStream`.
3. You can also see how to open multiple resources in one `try` clause.
4. We've used `transferFrom` to transfer data from one channel to the other. Hopping that the underlying system will offer DMA infrastructure, data can be transferred directly to the file system without any copying in between.

That was it! A lot of solutions there to suit every need. I hope you find this guide useful.

Donwload the Source Code

Tagged with: BUFFEREDOUTPUTSTREAM BUFFEREDWRITER BYTEBUFFER DMA FILECHANNEL FILEOUTPUTSTREAM FILEWRITER MAPPEDBYTEBUFFER RANDOMACCESSFILE

Do you want to know how to develop your skillset to become a Java Rockstar?

Subscribe to our newsletter to start Rocking right now!

To get you started we give you our best selling eBooks for FREE!

1. JPA Mini Book
2. JVM Troubleshooting Guide
3. JUnit Tutorial for Unit Testing
4. Java Annotations Tutorial
5. Java Interview Questions
6. Spring Interview Questions
7. Android UI Design

and many more ....

Email address:

Your email address

Sign up

KNOWLEDGE BASE

- Courses
- News
- Resources
- Tutorials
- Whitepapers

THE CODE GEEKS NETWORK

- .NET Code Geeks
- Java Code Geeks
- System Code Geeks
- Web Code Geeks

HALL OF FAME

- Android Alert Dialog Example
- Android OnClickListener Example
- How to convert Character to String and a String to Character Array in Java
- Java Inheritance example
- Java write to File Example
- java.io.FileNotFoundException – How to solve File Not Found Exception
- java.lang.arrayindexoutofboundsexception – How to handle Array Index Out Of Bounds Exception
- java.lang.NoClassDefFoundError – How to solve No Class Def Found Error
- JSON Example With Jersey + Jackson

ABOUT JAVA CODE GEEKS

JCGs (Java Code Geeks) is an independent online community focused on crea
ultimate Java to Java developers resource center; targeted at the technical a
technical team lead (senior developer), project manager and junior develope
JCGs serve the Java, SOA, Agile and Telecom communities with daily news w
domain experts, articles, tutorials, reviews, announcements, code snippets ar
source projects.

DISCLAIMER

Spring JdbcTemplate Example