

Forum:[Hack Forums](#)

Thread: [\[Specialists\] Java Streams Tutorial](#)

MrOrang - October 16, 2014, 10:17 am

This tutorial has been brought to you by the Specialists.

Difficulty: Easy

Foreword

First off all, to those of you waiting for the bytecode tutorials to continue. I will do those in another thread. They are just more work than this one and need more time.

In this tutorial you will learn how streams work and how to use them. We will also be going over examples such as copying files, downloading files and more.

What are Streams?

A stream is something that can transfer data. These are the basics of streams, there exists many more stream types but all streams rely on the simple definition of transferring data.

Here is a simple picture which shows an example stream:

[img]http://docs.oracle.com/javase/tutorial/figures/essential/io-ins.gif[/img]

Source: [Click](#)

You probably already heard about music streaming services like Spotify. These stream/transfer data to you which in this case is music.

When you save a file you are also streaming as it writes something from your program to a file on a disk.

I hope that I explained what streaming is well enough. If you have any questions you can always comment on the thread and I will be glad to help you out.

Input/Output?

There exist two kind of streams; [input](#) and [output](#) streams. An input stream can only READ and an output stream can only WRITE. You should know the differences because else your program will not work.

Example

I will now give an example on how to download a file using streams. Everything is explained in the code so it should be easy to follow.

Code package the.specialists.looka.testing;

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;
```

```

/**
 * Our test class
 * @author Luca
 *
 */
public class Test {

    /**
     * Code entry point
     * @param args
     */
    public static void main(String[] args) {
        try {
            /**
             * The link we're going to download
             */
            URL downloadUrl = new URL("http://pastebin.com/raw.php?i=xcyMsF8N");

            /**
             * The file we're going to save it to
             */
            File saveFile = new File("thingy.txt");

            /**
             * If the save file already exists delete it(optional)
             */
            if(saveFile.exists())
                saveFile.delete();

            /**
             * Create a new saveFile
             */
            saveFile.createNewFile();

            /**
             * Get the inputstream from the url we're going to download
             */
            InputStream downloadInputStream = downloadUrl.openStream();

            /**
             * Get the outputstream for the file so we can write to it
             */
            FileOutputStream fileOutputStream = new FileOutputStream(saveFile);

            /**
             * Read a single byte
             */
            int read = downloadInputStream.read();

            /**
             * Check if the byte read is equal to -1
             * -1 is most oftenly used as indicator of end of stream so we know we can stop reading.

```

```

    */
while(read != -1) {
    /*
    * Write the read byte to the file
    */
    fileOutputStream.write(read);
    /*
    * Read the next byte
    */
    read = downloadInputStream.read();
}

/*
* Be sure to close BOTH streams when done.
*/
downloadInputStream.close();
fileOutputStream.close();

System.out.println("Done downloading!");
} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}
}

```

This little example will download a text file from pastebin into a file called thingy.txt

Wait a minute, I've heard about readers and writers too, why not use those?

If you did not know yet, there is also something called [readers](#) and [writers](#) in Java. An example of this is the [FileWriter](#). This will also allow you to write just like an output stream. However there is one big difference. Readers and writers are based around reading/writing characters such as text files and streams are there to read bytes. You can choose which one to use yourself. I choose readers/writers for text files and streams for handling io things such as copying files. I will not go into readers and writers in this tutorial so we will forget about those now.

What kind of streams are there?

There are a lot of streams in Java ranging from [sending entire Java objects](#) to just sending a single byte. In the [java.io](#) package you will be able to find (almost)all streams that Java provides.

Obtaining a Stream

If you for example want to write to a file you first need to get an instance of [FileOutputStream](#).

After you got the instance of [FileOutputStream](#) you can write to the file by calling a [write](#) method.

However there are different ways to obtain an instance of a stream.

With the example I gave earlier I used the method [openStream](#) to obtain an [InputStream](#) instance. Most of the time you can just get a stream with the new keyword though like with the [FileOutputStream](#).

Reading & Writing

There are different ways to read and write with streams.

The default read methods are the following:

- [read](#):

This will read the next byte from the input stream.

- [read\(byte\[\] b\)](#)

This will fill the byte array with read bytes. For example; if I have a file containing some information and want to have 10 bytes read I would create a byte array with the size of 10 and then call this method to fill/read it into the byte array.

- [read\(byte\[\] b, int off, int len\)](#)

This is the more advanced version of [read\(byte\[\] b\)](#). Here you can specify where to start reading and how much to read from the input stream. For example if I have a large file and want to read from 500 until 510 I would call the method like this:

```
Code inputStream.read(bytes, 500, 10);
```

The first parameter of this method is the array to store it in. The second parameter is where to start reading from the stream(Also know as offset) and the last parameter is the length to read which in this case is 10.

Now you want to write your information to something using the [OutputStream](#). There are 3 default write methods just like [InputStream](#) has 3 default read methods. These methods are almost identical. The only difference is that the output stream writes and the input stream reads. So lets check the methods out:

- [write\(byte\[\] b\)](#)

When you use this method you are writing to something. Now lets take a simple text file as exmaple. If I have a text file that already says "I love" I would only have to write the bytes of " the specialists!".

This method will write everything at the end of the already written bytes.

- [write\(byte\[\] b, int off, int len\)](#)

This method looks a lot like the [read\(byte\[\] b, int off, int len\)](#) method. However this method will write the byte array at the given location(the off parameter) and will only write the specified length(the len parameter). For example; if I have a file of information and want to write something at index 23 and want to only write 10 bytes I would do the following:

```
Code outputStream.write(bytes, 23, 10);
```

Flushing?

You have probably heard about the term flushing. What is it and why do we need it?

Flushing only exists in output streams. When you call a [flush method](#) it will write all the bytes that were buffered out. When you don't flush you may have the problem of not all bytes being written. For now you don't need to know anything else about flushing as you probably won't need it. You won't have to call flush a lot though. Only if you really need something written like when another program will be reading a file which your program writes.

Closing Streams

The last method you should call when you are done reading or writing is the close method. This method will close the stream preventing any reading or writing. You probably also know the problem when you try to delete a file and it's saying a program is using it? Well that's a stream being busy with that file then. When that stream is closed you will be able to delete the specific file.

If you have an output stream you might think you also have to call the [flush method](#) before calling the close method as some bytes may not be written. This is not the case as the close method in an output stream always calls the [flush](#) method.

Copying

In the example I gave earlier I copy data from an inputstream coming from pastebin to an outputstream which in this case is a file. To copy data from an input stream to an output stream you will have to read a byte then check if it's the indicator for end-of-stream and then write it to the output stream.

Here is another example:

Code /**

* The streams, we will be copying the input stream over to the output stream.

*/

InputStream input = getYourInputStream();

OutputStream output = getYourOutputStream();

/**

* The current read byte

*/

int read = input.read();

/**

* Check if it's end of stream(-1 is end of stream in almost all cases)

*/

while(read != -1) {

/*

*Write the read byte

*/

output.write(read);

/**

* Read the next byte from the input stream

*/

read = input.read();

/**

* Repeat this until the end of stream is reached.

*/

}

AutoClosable

In Java 7 there was a new feature introduced named try-with-resource.

This allows you to skip the closing method. Here is an example that uses try-with-resource:

Code static String readFirstLineFromFile(String path) throws IOException {

try (BufferedReader br =

new BufferedReader(new FileReader(path))) {

return br.readLine();

}

}

I do not have a lot to explain about this rather than showing a piece of code. If you have questions about this feel free to post on my thread!

Conclusion

I hope you enjoyed, if you still don't understand something please post it in this thread and don't pm me about it as some people may be experiencing the same problem and can then find the answer in this thread.

This is my first tutorial for The Specialists so please give me feedback on what I could improve on and tell me if you see any mistakes!

Thanks!

Challenge

Create a program that can backup a folder.

This thread was reviewed by nJoyMe before posting.

Psycho_Coder - October 16, 2014, 10:31 am

Nice tutorial. It seems " Specialists ", is a different and experienced group. I like being member of a programming group, but i have not the post - requirements at this moment.

Btw, thank you for the contribution,
Psycho_Coder.

nOrr - October 16, 2014, 10:32 am

Nip-picking mode activated

- You shouldn't use JavaDoc comments for single line comments. Use // or at the very most multi-line comments where it makes sense (/* comment / and NOT /* Javadoc */))
- If you share catch logic (here, all you do is print the stack trace) you should use try with multi-catch for easier reading. This is a Java 7 feature if I remember right.
- Why do you prefer streams over readers/writes when working with text files?
- You should mention the AutoCloseable interface (<http://docs.oracle.com/javase/7/docs/api/java/lang/AutoCloseable.html>) imo.

Fix these points and you're a step closer to calling yourself a specialist in entry level streams. :P

MrOrang - October 16, 2014, 10:54 am

(Oct 16, 2014 10:32 AM)nOrr Wrote: [\[CITE\]](#) *Nip-picking mode activated*

- You shouldn't use JavaDoc comments for single line comments. Use // or at the very most multi-line comments where it makes sense (/* comment / and NOT /* Javadoc */))
- If you share catch logic (here, all you do is print the stack trace) you should use try with multi-catch for easier reading. This is a Java 7 feature if I remember right.
- Why do you prefer streams over readers/writes when working with text files?
- You should mention the AutoCloseable interface (<http://docs.oracle.com/javase/7/docs/api/java/lang/AutoCloseable.html>) imo.

Fix these points and you're a step closer to calling yourself a specialist in entry level streams. :P
I'll improve on it when I get back on my pc. Maybe I'll add a piece about AutoCloseable. Oh and I choose readers/writers for text files and streams for handling io things such as copying files.

nOrr - October 16, 2014, 10:59 am

(Oct 16, 2014 10:54 AM)?Looka Wrote: [\[CITE\]](#) I'll improve on it when I get back on my pc. Maybe I'll add a piece about AutoCloseable. Oh and I choose readers/writers for text files and streams for handling io things such as copying files.

Mention that in the post then. ;) Currently you have:

Quote: You can choose which one to use yourself. I prefer to use streams most of the time though. I will not go into readers and writers in this tutorial so we will forget about those now.

ObscureCoder - October 16, 2014, 2:21 pm

Nice thread, will be helpful for beginners.

Spoiler ([Click to View](#))

OT: If you guys want a real challenge, actually try and do a JNI thread. JNI is a f*cking sour c*nt.

nOrr - October 16, 2014, 2:24 pm

(Oct 16, 2014 01:00 PM)Bibl Wrote: [\[CITE\]](#) ??????????????:

Uhm... Yaaaarhh? ?

(Oct 16, 2014 02:21 PM)ObscureCoder Wrote: [\[CITE\]](#) OT: If you guys want a real challenge, actually try and do a JNI thread. JNI is a f*cking sour c*nt.

Is it? Why? I haven't actually worked with JNI, but what's "c*nty" about it?

MrOrang - October 16, 2014, 2:38 pm

(Oct 16, 2014 02:21 PM)ObscureCoder Wrote: [\[CITE\]](#) Nice thread, will be helpful for beginners.

Spoiler ([Click to View](#))

OT: If you guys want a real challenge, actually try and do a JNI thread. JNI is a f*cking sour c*nt.

I probably won't do JNI as I don't have any knowledge about the C language. Maybe someone else could do a tutorial about it if they want?

ObscureCoder - October 16, 2014, 2:54 pm

(Oct 16, 2014 02:24 PM)nOrr Wrote: [\[CITE\]](#) Is it? Why? I haven't actually worked with JNI, but what's "c*nty" about it?

Well, for a starter, since it's C++ you have to compile dlls and so files for both 32 bit and 64 bit which in my experience has been very frustrating with the compilers and IDEs I use. It's a real pain in the ass. A lot of people in this section talk about JNI but have never used it and if they had actually used it they'd probably stop talking about how 'great' it is for things like RATs etc. since it's a total nightmare in reality.

That's why. If you have the time and patience for it then go for it but it's a nightmare. I'm currently trying to compile a 64 bit dll using g++ for JNI so it will actually load in the 64bit JRE 1.8. I've had to turn to stackoverflow for help.

Bibl - October 16, 2014, 3:33 pm

urm jni is only a problem is you're terrible at c/c++ otherwise it's perfectly fine

ObscureCoder - October 16, 2014, 3:35 pm

(Oct 16, 2014 03:33 PM)Bibl Wrote: [\[CITE\]](#) urm jni is only a problem is you're terrible at c/c++ otherwise it's perfectly fine
Obviously.

JNI takes a lot of work for 32 bit and 64 bit cross platform use.

(Oct 16, 2014 03:35 PM)ObscureCoder Wrote: [\[CITE\]](#) Obviously.

JNI takes a lot of work for 32 bit and 64 bit cross platform use.

I take this back actually.

JNI is a f*cking doddle if you install the correct f*cking C++ compiler xD

I was compiling for the wrong JVM arch.

nOrr - October 17, 2014, 8:53 am

(Oct 16, 2014 02:54 PM)ObscureCoder Wrote: [\[CITE\]](#) Well, for a starter, since it's C++ you have to compile dlls and so files for both 32 bit and 64 bit which in my experience has been very frustrating with the compilers and IDEs I use. It's a real pain in the ass. A lot of people in this section talk about JNI but have never used it and if they had actually used it they'd probably stop talking about how 'great' it is for things like RATs etc. since it's a total nightmare in reality.

That's why. If you have the time and patience for it then go for it but it's a nightmare. I'm currently trying to compile a 64 bit dll using g++ for JNI so it will actually load in the 64bit JRE 1.8. I've had to turn to stackoverflow for help.

You say this as if SO is a place like HF. :D

It sounds like a fun-but-frustrating project indeed. Surely it must be possible and surely there must exists tools to help you. If it's a common problem, there will be common solutions.

Edit:

(Oct 16, 2014 03:35 PM)ObscureCoder Wrote: [\[CITE\]](#) I take this back actually.

JNI is a f*cking doddle if you install the correct f*cking C++ compiler xD

I was compiling for the wrong JVM arch.

See? Told you!! ? ?

ObscureCoder - October 17, 2014, 9:58 am

(Oct 17, 2014 08:53 AM)nOrr Wrote: [\[CITE\]](#) You say this as if SO is a place like HF. :D

It kind of is though. You can post malware related questions and people will answer and not even care if it's for malicious purposes or not.

Matt AH - October 17, 2014, 10:03 am

Awesome tutorial by a great programmer. Hope this can be used by everyone.

MrOrang - October 17, 2014, 10:27 am

(Oct 17, 2014 10:03 AM)Matt AH Wrote: [\[CITE\]](#) Awesome tutorial by a great programmer. Hope this can be used by everyone.

Thanks man. I will be sure to continue tutorials like this.

whoring for bronze award

I'll also add a piece about AutoCloseable maybe later tonight or tomorrow.

Aussie - October 18, 2014, 3:44 am

This thread needs more recognition. More people have to find these HQ things. Thanks for sharing, it will be useful for a friend of mine who plans on learning Java soon.

MrOrang - October 18, 2014, 4:09 am

(Oct 18, 2014 03:44 AM)n31ln3t Wrote: [\[CITE\]](#) This thread needs more recognition. More people have to find these HQ things. Thanks for sharing, it will be useful for a friend of mine who plans on learning Java soon.

Thanks, I hope this helps your friend.

I also hope that this thread will stay active ??

Aussie - October 18, 2014, 10:49 pm

Yeah, I also do hope this thread stays active. Btw, you should add a challenge, it will help your users remember things better and it will give them a practical use pirate

MrOrang - October 31, 2014, 9:55 am

Just added try-with-resource and added a little challenge.

I removed 1 pic as only 15 pictures are allowed :(

Working on a new tutorial/introduction right now.

nOrr - October 31, 2014, 10:26 am

(Oct 16, 2014 10:17 AM)?Looka Wrote: [\[CITE\]](#) I do not have a lot to explain about this rather than showing a piece of code. If you have questions about this feel free to post on my thread!

How about something along the lines of "Because this particular stream implements the interface AutoCloseable (see this for more info:

<http://docs.oracle.com/javase/8/docs/api/java/lang/AutoCloseable.html>) it is possible to use it this way. The JVM will handle proper closing in the end, so you don't have to worry about it. Should you want to be able to "try with resources" in your own class, make sure you implement the interface." ?

Took me less than a minute to write and it really adds a ton of information to the topic. ;)

MrOrang - October 31, 2014, 11:49 am

(Oct 31, 2014 10:26 AM)nOrr Wrote: [\[CITE\]](#) How about something along the lines of "Because this particular stream implements the interface AutoCloseable (see this for more info:

<http://docs.oracle.com/javase/8/docs/api/java/lang/AutoCloseable.html>) it is possible to use it this way. The JVM will handle proper closing in the end, so you don't have to worry about it. Should you want to be able to "try with resources" in your own class, make sure you implement the interface." ?

Took me less than a minute to write and it really adds a ton of information to the topic. ;)

Thanks, I didn't know what to write anymore so I'll add this info later.

lalalox4 - November 9, 2014, 11:43 pm

Good tutorial, I really like the fact that you can copy a whole range of things with streams. I took on the challenge. My code uses the try-with-resource feature. I like it. It seems to be really resourceful(huehue) in terms of efficiency. Note that no multi-threading is being used. It's just to demonstrate the theory behind my code and should not be viewed in terms of real-world implementation.

Cheers and thanks for the tutorial!

Spoiler ([Click to View](#))Code import java.io.File; // Needed for creating/copying files

import java.util.Scanner; // Needed for user input.

import java.io.FileInputStream; // Needed to read bytes from a file.

import java.io.FileOutputStream; // Needed to write bytes to a file.

import java.io.IOException; // Needed to deal with exceptions.

/**

* Makes a backup of a specified folder if the folder exists.

*

* @author lalalox4

*/

public class Backup {

private File root; // Holds the root folder

private File backup; // Holds the backup folder.

```
/**
 * Initiates the backup process after checking whether or not the folder exists and is a directory.
 *
 * @param rootName name of the folder that is to be copied.
 */
```

```
public void init(String rootName) {
    root = new File(rootName);
    if(root.exists() && root.isDirectory()) {
        backup = new File(root.getParent(), root.getName()+"_backup");
        backup.mkdir();
        start(root);
    } else {
        root.delete();
    }
}
```

```
/**
 * Makes another version of everything in the root folder into the backup folder.
 *
 * @param file File that is to be copied.
 */
```

```
public void start(File file) {
    for(File get: file.listFiles()) {
        File copy = new File(backup.getAbsolutePath()+get.getParent().replaceAll(root.getName(),
""), get.getName());
        if(get.isDirectory()) {
            copy.mkdir();
            start(get);
        }
        else
            fileCopy(get, copy);
    }
}
```

```
/**
 * Copies a single file.
 *
 * @param source source file.
 * @param output new version of the source.
 */
```

```
private void fileCopy(File source, File output) {
    try (
        FileInputStream fin = new FileInputStream(source);
        FileOutputStream fout = new FileOutputStream(output)
    ) {
        int read = fin.read();
        while(read != -1) {
            fout.write(read);
            read = fin.read();
        }
    } catch (IOException e) {
```

```

        e.printStackTrace();
    }

}

public static void main(String[] args) throws IOException {
    System.out.print("Directory to be copied: ");
    try (Scanner getName = new Scanner(System.in)) {
        new Backup().init(getName.next());
    }
}
}

```

nOrr - November 10, 2014, 8:15 am

(Nov 10, 2014 12:43 AM)lalalox4 Wrote: [CITE](#) Good tutorial, I really like the fact that you can copy a whole range of things with streams. I took on the challenge. My code uses the try-with-resource feature. I like it. It seems to be really resourceful(huehue) in terms of efficiency. Note that no multi-threading is being used. It's just to demonstrate the theory behind my code and should not be viewed in terms of real-world implementation.

Cheers and thanks for the tutorial!

Spoiler ([Click to View](#))Code import java.io.File; // Needed for creating/copying files

import java.util.Scanner; // Needed for user input.

import java.io.FileInputStream; // Needed to read bytes from a file.

import java.io.FileOutputStream; // Needed to write bytes to a file.

import java.io.IOException; // Needed to deal with exceptions.

/**

* Makes a backup of a specified folder if the folder exists.

*

* @author lalalox4

*/

public class Backup {

private File root; // Holds the root folder

private File backup; // Holds the backup folder.

/**

* Initiates the backup process after checking whether or not the folder exists and is a directory.

*

* @param rootName name of the folder that is to be copied.

*/

public void init(String rootName) {

root = new File(rootName);

if(root.exists() && root.isDirectory()) {

backup = new File(root.getParent(), root.getName()+"_backup");

backup.mkdir();

start(root);

} else {

root.delete();

}

}

```

/**
 * Makes another version of everything in the root folder into the backup folder.
 *
 * @param file File that is to be copied.
 */
public void start(File file) {
    for(File get: file.listFiles()) {
        File copy = new File(backup.getAbsolutePath()+get.getParent().replaceAll(root.getName(),
""), get.getName());
        if(get.isDirectory()) {
            copy.mkdir();
            start(get);
        }
        else
            fileCopy(get, copy);
    }
}

/**
 * Copies a single file.
 *
 * @param source source file.
 * @param output new version of the source.
 */
private void fileCopy(File source, File output) {
    try (
        FileInputStream fin = new FileInputStream(source);
        FileOutputStream fout = new FileOutputStream(output)
    ) {
        int read = fin.read();
        while(read != -1) {
            fout.write(read);
            read = fin.read();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static void main(String[] args) throws IOException {
    System.out.print("Directory to be copied: ");
    try (Scanner getName = new Scanner(System.in)) {
        new Backup().init(getName.next());
    }
}
}

```

Not all real world applications should be concurrent. The actual copying from one stream to another could be used in a real application.

lalalox4 - November 10, 2014, 2:55 pm

(Nov 10, 2014 09:15 AM)nOrr Wrote: [CITE](#) Not all real world applications should be concurrent. The actual copying from one stream to another could be used in a real application.

Certainly but I was pointing that out because the program is really slow at copying large directories
*

Anyone care to give me their opinion on my code?

lalalox4 - November 11, 2014, 6:25 pm

damn why do i never get feedback when i try to participate...-_-

nOrr - November 12, 2014, 7:48 am

(Nov 11, 2014 07:25 PM)lalalox4 Wrote: [\[CITE\]](#) damn why do i never get feedback when i try to participate...-_-

Dude I just told you that your code is fine if we're looking at the particular example of using the streams. No, the overall structure is not "OK" for real world applications, as you wouldn't rely on something as simple as that.

You could look into the nio library which has some more advanced and useful classes to use instead of streams, but for streams in particular, that is fine.

lalalox4 - November 12, 2014, 10:59 am

(Nov 12, 2014 08:48 AM)nOrr Wrote: [\[CITE\]](#) Dude I just told you that your code is fine if we're looking at the particular example of using the streams. No, the overall structure is not "OK" for real world applications, as you wouldn't rely on something as simple as that.

You could look into the nio library which has some more advanced and useful classes to use instead of streams, but for streams in particular, that is fine.

Thanks :D

It wasn't clear in my mind whether or not you were just saying that it didn't need to be concurrent. Hence, why I thought you weren't directly commenting on the code in general.

nOrr - November 12, 2014, 11:20 am

(Nov 12, 2014 11:59 AM)lalalox4 Wrote: [\[CITE\]](#) Thanks :D

It wasn't clear in my mind whether or not you were just saying that it didn't need to be concurrent. Hence, why I thought you weren't directly commenting on the code in general.
;)

If you ignore all the boiler plate code, and only focus on the stream copying, it's fine.