

Standard concise way to copy a file in Java?

Asked 16 years, 3 months ago Modified 5 months ago

Viewed 282k times



430



It has always bothered me that the only way to copy a file in Java involves opening streams, declaring a buffer, reading in one file, looping through it, and writing it out to the other stream. The web is littered with similar, yet still slightly different implementations of this type of solution.

Is there a better way that stays within the bounds of the Java language (meaning does not involve exec-ing OS specific commands)? Perhaps in some reliable open source utility package, that would at least obscure this underlying implementation and provide a one line solution?

java

file

copy

Share

Improve this question

Follow

asked Sep 20, 2008 at 1:59



Peter

29.8k ● 22 ● 91 ● 126

5 There could be something in Apache Commons [FileUtils](#), Specifically, the **copyFile** methods. – [toolkit](#) Sep 20, 2008 at

24 If using Java 7, use Files.copy instead, as recommended by @GlenBest: stackoverflow.com/a/16600787/44737 – rob Jun 17, 2013 at 21:54

16 Answers

Sorted by:

Highest score (default)



280



I would avoid the use of a mega api like apache commons. This is a simplistic operation and its built into the JDK in the new NIO package. It was kind of already linked to in a previous answer, but the key method in the NIO api are the new functions "transferTo" and "transferFrom".

[http://java.sun.com/javase/6/docs/api/java/nio/channels/FileChannel.html#transferTo\(long,%20long,%20java.nio.channels.WritableByteChannel\)](http://java.sun.com/javase/6/docs/api/java/nio/channels/FileChannel.html#transferTo(long,%20long,%20java.nio.channels.WritableByteChannel))

One of the linked articles shows a great way on how to integrate this function into your code, using the transferFrom:

```
public static void copyFile(File sourceFile, File dest
{
    if(!destFile.exists()) {
        destFile.createNewFile();
    }

    FileChannel source = null;
    FileChannel destination = null;

    try {
        source = new FileInputStream(sourceFile).getCh
        destination = new FileOutputStream(destFile).g
```

```

        destination.transferFrom(source, 0, source.size() - 1);
    }
    finally {
        if(source != null) {
            source.close();
        }
        if(destination != null) {
            destination.close();
        }
    }
}

```

Learning NIO can be a little tricky, so you might want to just trust in this mechanic before going off and trying to learn NIO overnight. From personal experience it can be a very hard thing to learn if you don't have the experience and were introduced to IO via the java.io streams.

Share Improve this answer

edited Aug 15, 2011 at 13:15


Follow

community wiki


3 revs, 3 users 73%

Josh

2 Thanks, useful info. I would still argue for something like Apache Commons, especially if it uses nio (properly) underneath; but I agree it is important to understand the underlying fundamentals. – [Peter](#) Sep 22, 2008 at 17:19

1 Unfortunately, there are caveats. When I copied 1.5 Gb file on Windows 7, 32 bit, it failed to map the file. I had to look for another solution. – [Anton K.](#) Jan 12, 2011 at 8:48 

16 Three possible problems with the above code: (a) if getChannel throws an exception, you might leak an open

stream; (b) for large files, you might be trying to transfer more at once than the OS can handle; (c) you are ignoring the return value of `transferFrom`, so it might be copying just part of the file. This is why `org.apache.tools.ant.util.ResourceUtils.copyResource` is so complicated. Also note that while `transferFrom` is OK, `transferTo` breaks on JDK 1.4 on Linux:
bugs.sun.com/bugdatabase/view_bug.do?bug_id=5056395
– [Jesse Glick](#) Jan 28, 2011 at 0:41 

7 I believe this updated version addresses those concerns:
gist.github.com/889747 – [Mark Renouf](#) Mar 27, 2011 at 23:15

11 This code has a *major* problem. `transferTo()` must be called in a loop. It doesn't guarantee to transfer the entire amount requested. – [user207421](#) Jun 1, 2013 at 1:30



277

As toolkit mentions above, Apache Commons IO is the way to go, specifically [FileUtils.copyFile\(\)](#); it handles all the heavy lifting for you.



And as a postscript, note that recent versions of `FileUtils` (such as the 2.0.1 release) have added the use of NIO for copying files; [NIO can significantly increase file-copying performance](#), in a large part because the NIO routines defer copying directly to the OS/filesystem rather than handle it by reading and writing bytes through the Java layer. So if you're looking for performance, it might be worth checking that you are using a recent version of `FileUtils`.

Follow



Steve Blackwell

5,924 ● 33 ● 49

answered Sep 20, 2008 at 2:23



delfuego

14.3k ● 4 ● 41 ● 39

-
- 1 Very helpful - do you have any insight as to when an official release will incorporate these nio changes? – [Peter](#) Sep 20, 2008 at 3:01
-
- 2 Public release of Apache Commons IO still at 1.4, grrrrrrr – [Peter](#) Sep 2, 2009 at 23:05
-
- 14 As of Dec 2010, Apache Commons IO is at 2.0.1, which has the NIO functionality. Answer updated. – [Simon Nickerson](#) Apr 8, 2011 at 10:08
-
- 4 A warning to Android people: this is NOT included in the standard Android APIs – [IIDan](#) Feb 6, 2012 at 10:46
-
- 18 If using Java 7 or newer, you can use Files.copy as suggested by @GlenBest:
stackoverflow.com/a/16600787/44737 – [rob](#) Jun 17, 2013 at 21:57 ✎
-



Now with Java 7, you can use the following try-with-resource syntax:

183



```
public static void copyFile( File from, File to ) thro  
  
    if ( !to.exists() ) { to.createNewFile(); }  
  
    try (  
        FileChannel in = new FileInputStream( from ).g  
        FileChannel out = new FileOutputStream( to ).g
```

```
        out.transferFrom( in, 0, in.size() );
    }
}
```

Or, better yet, this can also be accomplished using the new Files class introduced in Java 7:

```
public static void copyFile( File from, File to ) thro
    Files.copy( from.toPath(), to.toPath() );
}
```

Pretty snazzy, eh?

Share Improve this answer

edited Jul 29, 2011 at 23:01

Follow

answered Jul 28, 2011 at 18:22



Scott

2,233 ● 2 ● 15 ● 8

15 It's amazing Java hasn't added things like this before today. Certain operations are just the absolute essentials of writing computer software. The Oracle developers of Java could learn a thing or two from operating systems, looking at what services they provide, to make it EASIER for newbies to migrate over. – [Rick Hodgins](#) Oct 18, 2011 at 22:30

2 Ah thanks! I was not aware of the new "Files" class with all of its helper functions. It has exactly what I need. Thanks for the example. – [Chris Cantrell](#) Nov 27, 2012 at 0:26

1 performance wise, java NIO FileChannel is better, read this article journaldev.com/861/4-ways-to-copy-file-in-java – Pankaj Dec 4, 2012 at 0:36

5 This code has a *major* problem. transferTo() must be called in a loop. It doesn't guarantee to transfer the entire amount requested. – user207421 Jun 1, 2013 at 1:30

@Scott: Pete asked for a one-line solution and you're so close...it's unnecessary to wrap Files.copy in a copyFile method. I'd just put the Files.copy(Path from, Path to) at the beginning of your answer and mention that you can use File.toPath() if you have existing File objects:
Files.copy(fromFile.toPath(), toFile.toPath()) – rob Jun 18, 2013 at 17:13



94



- These methods are performance-engineered (they integrate with operating system native I/O).
- These methods work with files, directories and links.
- Each of the options supplied may be left out - they are optional.



The utility class

```
package com.yourcompany.nio;

class Files {

    static int copyRecursive(Path source, Path target,
CopyOptions options...) {
        CopyVisitor copyVisitor = new CopyVisitor(sour
options).copy();
        EnumSet<FileVisitOption> fileVisitOpts;
        if
(Arrays.toList(options).contains(java.nio.file.LinkOpt
```

```

        fileVisitOpts = EnumSet.noneOf(FileVisitOp
    } else {
        fileVisitOpts = EnumSet.of(FileVisitOption
    }
    Files.walkFileTree(source[i], fileVisitOpts, I
copyVisitor);
    }

    private class CopyVisitor implements FileVisitor<P
    final Path source;
    final Path target;
    final CopyOptions[] options;

    CopyVisitor(Path source, Path target, CopyOpti
        this.source = source; this.target = targ
options;
    };

    @Override
    FileVisitResult preVisitDirectory(Path dir, Ba
{
    // before visiting entries in a directory we c
    // (okay if directory already exists).
    Path newdir = target.resolve(source.relativeize
    try {
        Files.copy(dir, newdir, options);
    } catch (FileAlreadyExistsException x) {
        // ignore
    } catch (IOException x) {
        System.err.format("Unable to create: %s: %
        return SKIP_SUBTREE;
    }
    return CONTINUE;
}

    @Override
    public FileVisitResult visitFile(Path file, BasicF
    Path newfile= target.resolve(source.relativeize
    try {
        Files.copy(file, newfile, options);
    } catch (IOException x) {
        System.err.format("Unable to copy: %s: %s%
    }
    return CONTINUE;
}

```



```

    }

    @Override
    public FileVisitResult postVisitDirectory(Path dir
        // fix up modification time of directory when
        if (exc == null && Arrays.toList(options).contains(
            Path newdir = target.resolve(source.relati
        try {
            FileTime time = Files.getLastModifiedT
            Files.setLastModifiedTime(newdir, time
        } catch (IOException x) {
            System.err.format("Unable to copy all
newdir, x);
        }
    }
    return CONTINUE;
}

@Override
public FileVisitResult visitFileFailed(Path file,
    if (exc instanceof FileSystemLoopException) {
        System.err.println("cycle detected: " + fi
    } else {
        System.err.format("Unable to copy: %s: %s%
    }
    return CONTINUE;
}
}

```

Copying a directory or file

```

long bytes = java.nio.file.Files.copy(
    new java.io.File("<filepath1>").toPat
    new java.io.File("<filepath2>").toPat
    java.nio.file.StandardCopyOption.REPL
    java.nio.file.StandardCopyOption.COPY
    java.nio.file.LinkOption.NOFOLLOW_LIN

```

Moving a directory or file

```
long bytes = java.nio.file.Files.move(  
    new java.io.File("<filepath1>").toPath()  
    new java.io.File("<filepath2>").toPath()  
    java.nio.file.StandardCopyOption.ATOMIC_MOVE  
    java.nio.file.StandardCopyOption.REPLACE_EXISTING
```

Copying a directory or file recursively

```
long bytes = com.yourcompany.nio.Files.copyRecursive(  
    new java.io.File("<filepath1>").toPath()  
    new java.io.File("<filepath2>").toPath()  
    java.nio.file.StandardCopyOption.REPLACE_EXISTING  
    java.nio.file.StandardCopyOption.COPY_ATTRIBUTES  
    java.nio.file.LinkOption.NOFOLLOW_LINKS
```

Share Improve this answer

edited Jan 20, 2016 at 17:30

Follow



randers

5,137 ● 5 ● 40 ● 64

answered May 17, 2013 at 3:09



Glen Best

23.1k ● 4 ● 59 ● 74

The package name for Files was wrong (should be java.nio.file not java.nio). I've submitted an edit for that; hope that's OK! – [Stuart Rossiter](#) Dec 3, 2014 at 13:15

- 1 There's no point in writing `new java.io.File("<filepath1>").toPath()` when you can use `Paths.get("<filepath1>")` in the first place. – [Holger](#) Mar 12, 2021 at 7:11
-

In Java 7 it is easy...



52



```
Path src = Paths.get("original.txt");
Path target = Paths.get("copy.txt");

Files.copy(src, target, StandardCopyOption.REPLACE_EXISTING);
```

Share Improve this answer

edited Jun 27 at 13:43

Follow

answered Jun 20, 2014 at 18:38



Kevin Sadler

2,446 ● 28 ● 34

1 What does your answer add to Scott's or Glen's?
– [Uri Agassi](#) Jun 20, 2014 at 19:59

14 It's concise, less is more. Their answers are good and detailed, but I missed them when looking through. Unfortunately there are a lot of answers to this and a lot of them are long, obsolete and complicated and Scott and Glen's good answers got lost in that (I will give upvotes to help with that). I wonder if my answer might be improved by reducing it to three lines by knocking out the exists() and error message. – [Kevin Sadler](#) Jun 21, 2014 at 8:18

This doesn't work for directories. Damn everyone is getting this one wrong. More of an API communication issue your fault. I too got it wrong. – [mjs](#) Jan 17, 2015 at 10:06

4 @momo the question was how to copy a file. – [Kevin Sadler](#) Jan 21, 2015 at 12:01

3 There's no need to go the `File` detour when you need a `Path`. `Files.copy(Paths.get("original.txt"), Paths.get("copy.txt"), ...)` – [Holger](#) Mar 12, 2021 at 7:13



To copy a file and save it to your destination path you can use the method below.

28



```
public void copy(File src, File dst) throws IOException {
    InputStream in = new FileInputStream(src);
    try {
        OutputStream out = new FileOutputStream(dst);
        try {
            // Transfer bytes from in to out
            byte[] buf = new byte[1024];
            int len;
            while ((len = in.read(buf)) > 0) {
                out.write(buf, 0, len);
            }
        } finally {
            out.close();
        }
    } finally {
        in.close();
    }
}
```

Share Improve this answer

Follow

edited Sep 28, 2015 at 20:54



Boris Treukhov

17.8k ● 9 ● 71 ● 92

answered Oct 23, 2013 at 13:08



Rakshi

6,856 ● 3 ● 27 ● 46

-
- 1 This will work, but I don't think it's better than the other answers here? – [Rup](#) Oct 23, 2013 at 14:18
 - 2 @Rup It is considerably better than the other answers here, (a) *because* it works, and (b) because it doesn't rely on third party software. – [user207421](#) Jul 19, 2014 at 9:41
-

- 1 @EJP OK, but it's not very smart. File copying should be an OS or filesystem operation, not an application operation: Java hopefully can spot a copy and turn it into an OS operation except by explicitly reading the file in you're stopping it doing that. If you don't think Java can do that, would you trust it to optimise 1K reads and writes into larger blocks? And if source and destination were on a remote share over a slow network then this is clearly doing unnecessary work. Yes some third party JARs are stupidly large (Guava!) but they do add lots of stuff like this done properly. – [Rup](#) Jul 19, 2014 at 10:02
-

Worked like a charm. Best solution that does not require 3rd party libraries and works on java 1.6. Thanks.
– [James Wierzba](#) Apr 16, 2015 at 19:36

@Rup I agree that it should be an operating system function, but I can't make any other sense of your comment. The part after the first colon is lacking a verb somewhere; I would neither 'trust' not expect Java to turn 1k blocks into something larger, although I would certainly use much larger blocks myself; I would never write an application that used shared files in the first place; and I'm not aware that any third party library does anything more 'proper' (whatever you mean by that) than this code, except probably to use a larger buffer.
– [user207421](#) May 17, 2015 at 18:01



24

Note that all of these mechanisms only copy the contents of the file, not the metadata such as permissions. So if you were to copy or move an executable .sh file on linux the new file would not be executable.



In order to truly a copy or move a file, ie to get the same result as copying from a command line, you actually need to use a native tool. Either a shell script or JNI.



Apparently, this might be fixed in java 7 -

<http://today.java.net/pub/a/today/2008/07/03/jsr-203-new-file-apis.html>. Fingers crossed!

Share Improve this answer

answered Sep 23, 2008 at 3:38

Follow



Brad at Kademi

1,320 ● 8 ● 7



Google's Guava library also has a [copy method](#):

23

```
public static void copy(File from,  
                        File to)  
    throws IOException
```



Copies all the bytes from one file to another.



Warning: If `to` represents an existing file, that file will be overwritten with the contents of `from`. If `to` and `from` refer to the *same* file, the contents of that file will be deleted.

Parameters: `from` - the source file `to` - the destination file

Throws: [IOException](#) - if an I/O error occurs

[IllegalArgumentException](#) - if `from.equals(to)`

Share Improve this answer

edited Feb 26, 2017 at 20:05

Follow



Divyesh Kanzariya

3,789 ● 3 ● 47 ● 46

answered Jul 5, 2010 at 16:17



Andrew McKinlay

2,611 ● 1 ● 26 ● 27



18



Available as standard in Java 7, `path.copyTo`:

<http://openjdk.java.net/projects/nio/javadoc/java/nio/file/Path.html>

<http://java.sun.com/docs/books/tutorial/essential/io/copy.html>



I can't believe it took them so long to standardise something so common and simple as file copying :(

Share Improve this answer

Follow

answered Jun 1, 2010 at 8:33



Ryan

205 ● 2 ● 2

10 There is no `Path.copyTo`; it is `Files.copy`. – Jesse Glick Mar 3, 2012 at 0:59



7



Three possible problems with the above code:

1. If `getChannel` throws an exception, you might leak an open stream.
2. For large files, you might be trying to transfer more at once than the OS can handle.
3. You are ignoring the return value of `transferFrom`, so it might be copying just part of the file.

This is why

`org.apache.tools.ant.util.ResourceUtils.copyResource` is so complicated. Also note that while `transferFrom` is OK, `transferTo` breaks on JDK 1.4 on Linux (see [Bug ID:5056395](#)) – Jesse Glick Jan

Share Improve this answer

edited Nov 10, 2023 at 19:48

Follow



Community Bot

1 • 1

answered Apr 13, 2012 at 9:36



saji

79 • 1 • 1



7



If you are in a web application which already uses Spring and if you do not want to include Apache Commons IO for simple file copying, you can use [FileCopyUtils](#) of the Spring framework.

Share Improve this answer

answered Sep 7, 2012 at 6:43

Follow



Balaji Paulrajan

613 • 5 • 15



6



```
public static void copyFile(File src, File dst) throws
{
    long p = 0, dp, size;
    FileChannel in = null, out = null;

    try
    {
        if (!dst.exists()) dst.createNewFile();
    }
}
```




```
in = new FileInputStream(src).getChannel();
out = new FileOutputStream(dst).getChannel();
size = in.size();

while ((dp = out.transferFrom(in, p, size)) >
{
    p += dp;
}
}
finally {
    try
    {
        if (out != null) out.close();
    }
    finally {
        if (in != null) in.close();
    }
}
}
```

Share Improve this answer

answered Jan 16, 2014 at 1:25

Follow



[user3200607](#)

107 ● 1 ● 3

So the difference from the top accepted answer is that you've got the transferFrom in a while loop? – [Rup](#) Jan 16, 2014 at 11:41

- 1 Doesn't even compile, and the createNewFile() call is redundant and wasteful. – [user207421](#) Jul 19, 2014 at 9:43
-



Here is three ways that you can easily copy files with single line of code!

6

Java7:



[java.nio.file.Files#copy](#)



```
private static void copyFileUsingJava7Files(File source, File dest) throws IOException {
    Files.copy(source.toPath(), dest.toPath());
}
```

Apache Commons IO:

[FileUtils#copyFile](#)

```
private static void copyFileUsingApacheCommonsIO(File source, File dest) throws IOException {
    FileUtils.copyFile(source, dest);
}
```

Guava :

[Files#copy](#)

```
private static void copyFileUsingGuava(File source, File dest) throws IOException {
    Files.copy(source, dest);
}
```

Share Improve this answer

answered Nov 13, 2014 at 11:57

Follow



JaskeyLam

15.8k ● 23 ● 127 ● 158

First one doesn't work for directories. Damn everyone is getting this one wrong. More of an API communication issue your fault. I too got it wrong. – mjs Jan 17, 2015 at 10:07

First one needs 3 parameters. `Files.copy` using only 2 parameters is for `Path` to `Stream`. Just add the parameter `StandardCopyOption.COPY_ATTRIBUTES` or `StandardCopyOption.REPLACE_EXISTING` for `Path` to `Path` – Pimp Trizkit Feb 16, 2016 at 21:18



3



NIO copy with a buffer is the fastest according to my test. See the working code below from a test project of mine at <https://github.com/mhisoft/fastcopy>

```
import java.io.Closeable;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.text.DecimalFormat;

public class test {

    private static final int BUFFER = 4096*16;
    static final DecimalFormat df = new DecimalFormat("#,##");
    public static void nioBufferCopy(final File source, fi
        FileChannel in = null;
        FileChannel out = null;
        double size=0;
        long overallT1 = System.currentTimeMillis();
```

```

try {
    in = new FileInputStream(source).getChannel();
    out = new FileOutputStream(target).getChannel();
    size = in.size();
    double size2InKB = size / 1024 ;
    ByteBuffer buffer = ByteBuffer.allocateDirect(

    while (in.read(buffer) != -1) {
        buffer.flip();

        while(buffer.hasRemaining()){
            out.write(buffer);
        }

        buffer.clear();
    }
    long overallT2 = System.currentTimeMillis();
    System.out.println(String.format("Copied %s KB
df.format(size2InKB), (overallT2 - overallT1)));
}
catch (IOException e) {
    e.printStackTrace();
}

finally {
    close(in);
    close(out);
}
}

private static void close(Closeable closable) {
    if (closable != null) {
        try {
            closable.close();
        } catch (IOException e) {
            if (FastCopy.debug)
                e.printStackTrace();
        }
    }
}
}

```

```

}

```

Share Improve this answer

edited Feb 7, 2015 at 22:57

Follow

answered Feb 7, 2015 at 22:51



Tony

1,528 ● 2 ● 12 ● 13

nice! this one is fast rather than standar java.io stream ..
copying 10GB only in 160 seconds – [aswzen](#) Oct 13, 2016 at 4:44



2



Fast and work with all the versions of Java also Android:

```
private void copy(final File f1, final File f2) throws
    f2.createNewFile();

    final RandomAccessFile file1 = new RandomAccessFil
    final RandomAccessFile file2 = new RandomAccessFil

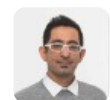
file2.getChannel().write(file1.getChannel().map(FileCh
0, f1.length()));

    file1.close();
    file2.close();
}
```

Share Improve this answer

answered Nov 14, 2013 at 9:43

Follow



user1079877

9,358 ● 5 ● 45 ● 55

1 Not all filesystems support memory mapped files though, and
I think it's relatively expensive for small files. – [Rup](#) Nov 14,

2013 at 10:48

Doesn't work with any version of Java prior to 1.4, and there is nothing that guarantees a single write is sufficient.

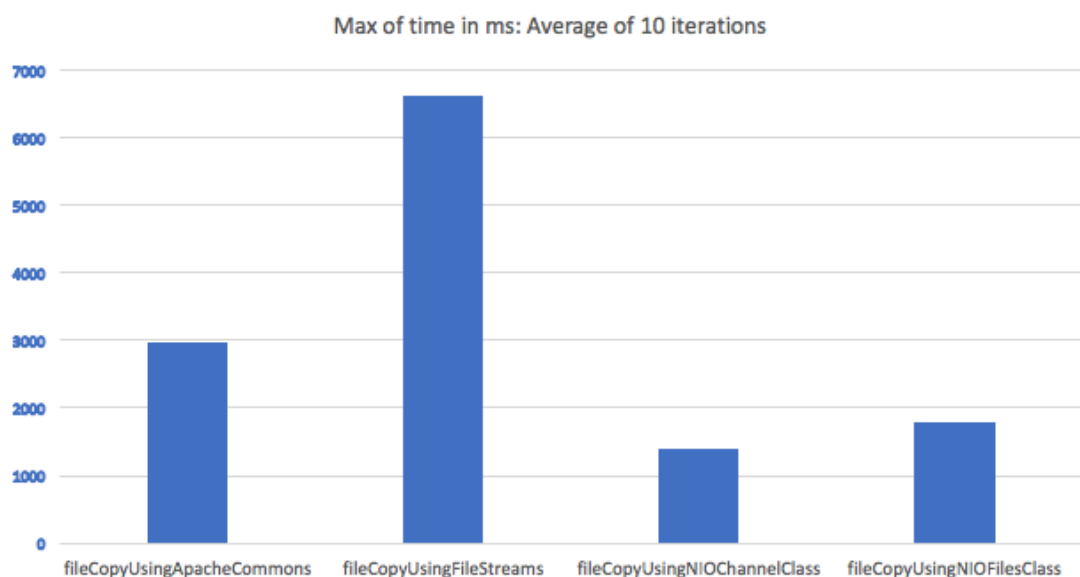
– [user207421](#) Jul 19, 2014 at 9:44 



1



A little late to the party, but here is a comparison of the time taken to copy a file using various file copy methods. I looped in through the methods for 10 times and took an average. File transfer using IO streams seem to be the worst candidate:



Here are the methods:

```
private static long fileCopyUsingFileStreams(File file
throws IOException {
    FileInputStream input = new FileInputStream(fileTo
    FileOutputStream output = new FileOutputStream(new
    byte[] buf = new byte[1024];
    int bytesRead;
    long start = System.currentTimeMillis();
    while ((bytesRead = input.read(buf)) > 0)
    {
```

```

        output.write(buf, 0, bytesRead);
    }
    long end = System.currentTimeMillis();

    input.close();
    output.close();

    return (end-start);
}

private static long fileCopyUsingNIOChannelClass(File fi
throws IOException
{
    FileInputStream inputStream = new FileInputStream(fi);
    FileChannel inChannel = inputStream.getChannel();

    FileOutputStream outputStream = new FileOutputStream(newFile);
    FileChannel outChannel = outputStream.getChannel();

    long start = System.currentTimeMillis();
    inChannel.transferTo(0, fi.length(), outChannel);
    long end = System.currentTimeMillis();

    inputStream.close();
    outputStream.close();

    return (end-start);
}

private static long fileCopyUsingApacheCommons(File fi
throws IOException
{
    long start = System.currentTimeMillis();
    FileUtils.copyFile(fi, newFile);
    long end = System.currentTimeMillis();
    return (end-start);
}

private static long fileCopyUsingNIOFilesClass(File fi
throws IOException
{
    Path source = Paths.get(fi.getPath());
    Path destination = Paths.get(newFile.getPath());
    long start = System.currentTimeMillis();

```

```
Files.copy(source, destination, StandardCopyOption
long end = System.currentTimeMillis());

return (end-start);
}
```

The only drawback what I can see while using NIO channel class is that I still can't seem to find a way to show intermediate file copy progress.

Share Improve this answer

answered May 30, 2018 at 10:50

Follow



Vinit Shandilya

1,643 ● 5 ● 25 ● 46



Highly active question. Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.