

**NIO.2**

Chapter 24

## NIO.2

NIO = Non-blocking Input/Output

NIO.2 provides better support for accessing files and the file system, symbolic links, interoperability, and exceptions among others.

The primary classes of `java.nio.file`, `Path`, `Paths`, and `Files`, are intended to provide an easier way to work with files and to be a replacement for the `java.io.File` class.

# Path Interface

Path interface defines an object that represents the path to a file or a directory.

## Windows-based

Not case sensitive

Backslashes (\)

Root start with letter (c:\)

## Unix-based

Case sensitive

Slashes (/)

Root start with slash (/)

**Path** is the **interface** with methods to work with paths.

**Paths** is the **class** with `static` methods to create a `Path` object.

`java.nio.files.Paths` is this class. It provides two methods to create a `Path` object:

```
static Path get(String first, String... more)
static Path get(URI uri)
```

# Path Interface

```
public static void main(String[] args) {
    // With an absolute path in windows
    Path pathWin = Paths.get("c:\\temp\\file.txt");
    System.out.println("pathWin : " + pathWin);

    // With an absolute path in unix
    Path pathUnix = Paths.get("/temp/file.txt");
    System.out.println("pathUnix : " + pathUnix);

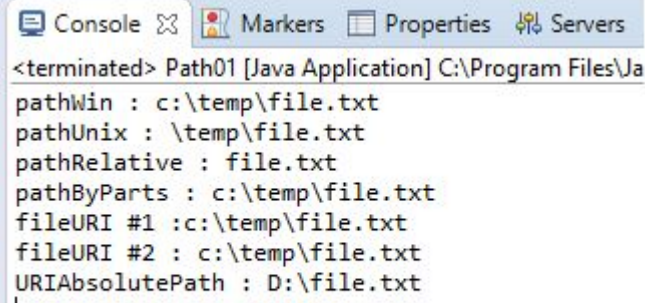
    // With a relative path
    Path pathRelative = Paths.get("file.txt");
    System.out.println("pathRelative : " + pathRelative);

    //Using the varargs parameter
    // (the separator is inserted automatically)
    Path pathByParts = Paths.get("c:", "temp", "file.txt");
    System.out.println("pathByParts : " + pathByParts);

    //use a java.net.URI instance
    try {
        Path fileURI = Paths.get(new URI("file:///c:/temp/file.txt"));
        System.out.println("fileURI #1 : " + fileURI);
    } catch (URISyntaxException e) {
    }

    //use the static method URI.create(String)
    //It wraps the URISyntaxException exception in an IllegalArgumentException
    Path fileURI = Paths.get(URI.create("file:///c:/temp/file.txt"));
    System.out.println("fileURI #2 : " + fileURI);

    //returns the absolute path representation of a Path object
    Path fileURI2 = Paths.get(URI.create("file:///file.txt"));
    System.out.println("URIAbsolutePath : " + fileURI2.toAbsolutePath());
}
```

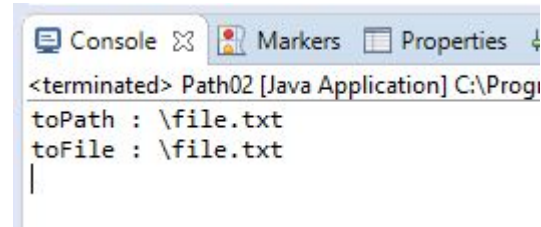


```
<terminated> Path01 [Java Application] C:\Program Files\Ja
pathWin : c:\temp\file.txt
pathUnix : \temp\file.txt
pathRelative : file.txt
pathByParts : c:\temp\file.txt
fileURI #1 : c:\temp\file.txt
fileURI #2 : c:\temp\file.txt
URIAbsolutePath : D:\file.txt
```

Uniform Resource Identifier (URI) = ระบุตัวตนของ Resource

## File - Path

```
public static void main(String[] args) {  
    File file = new File("/file.txt");  
    Path path = file.toPath();  
    System.out.println("toPath : " + path);  
  
    path = Paths.get("/file.txt");  
    file = path.toFile();  
    System.out.println("toFile : " + file);  
}
```



Path instance is system-dependent, let me tell you that `Paths.get()` is actually equivalent to:

```
Path path = FileSystems.getDefault().getPath("c://temp");
```

## Path - method to get information (Absolute Path)

```
public static void main(String[] args) {  
    Path path = Paths.get("C:\\temp\\dir1\\file.txt");  
    // Or Path path = Paths.get("/temp/dir1/file.txt");  
    System.out.println("toString(): " + path.toString());  
    System.out.println("getFileName(): " + path.getFileName());  
    System.out.println("getNameCount(): " + path.getNameCount());  
    // Indexes start from zero  
    System.out.println("getName(0): " + path.getName(0));  
    System.out.println("getName(1): " + path.getName(1));  
    System.out.println("getName(2): " + path.getName(2));  
    // subpath(beginIndex, endIndex) from beginIndex to endIndex-1  
    System.out.println("subpath(0,2): " + path.subpath(0, 2));  
    System.out.println("getParent(): " + path.getParent());  
    System.out.println("getRoot(): " + path.getRoot());  
  
    //invalid index  
    System.out.println("getName(3): " + path.getName(3));  
}
```

Console   Markers   Properties   Servers   Data Source Explorer   Search   Expressions

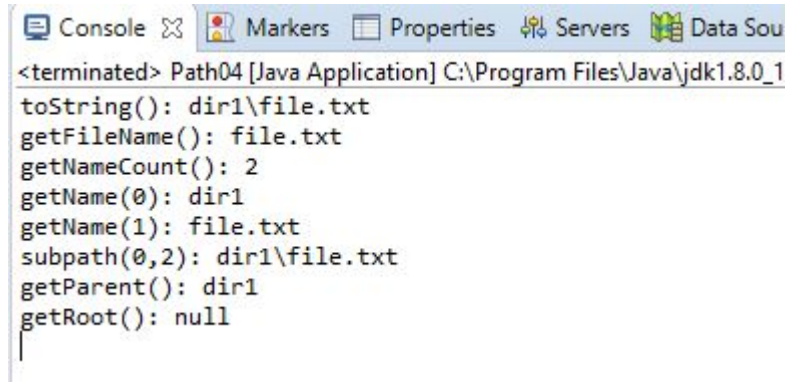
<terminated> Path03 [Java Application] C:\Program Files\Java\jdk1.8.0\_111\bin\javaw.exe (18 n.u. 2560 10:04:56)

toString(): C:\temp\dir1\file.txt  
getFileName(): file.txt  
getNameCount(): 3  
getName(0): temp  
getName(1): dir1  
getName(2): file.txt  
subpath(0,2): temp\dir1  
getParent(): C:\temp\dir1  
getRoot(): C:\

Exception in thread "main" [java.lang.IllegalArgumentException](#)  
at sun.nio.fs.WindowsPath.getName(WindowsPath.java:620)|  
at sun.nio.fs.WindowsPath.getName(WindowsPath.java:44)|  
at com.wealth.certificate.study\_1z0\_809.chapter24.path.Path03.main(Path03.java:23)

## Path - method to get information (Relative Path)

```
public static void main(String[] args) {  
    Path path = Paths.get("dir1\\file.txt");// Or dir1/file.txt  
    System.out.println("toString(): " + path.toString());  
    System.out.println("getFileName(): " + path.getFileName());  
    System.out.println("getNameCount(): " + path.getNameCount());  
    System.out.println("getName(0): " + path.getName(0));  
    System.out.println("getName(1): " + path.getName(1));  
    System.out.println("subpath(0,2): " + path.subpath(0, 2));  
    System.out.println("getParent(): " + path.getParent());  
    System.out.println("getRoot(): " + path.getRoot());  
}
```



The screenshot shows an IDE console window with the following tabs: Console, Markers, Properties, Servers, and Data Source. The console output is as follows:

```
<terminated> Path04 [Java Application] C:\Program Files\Java\jdk1.8.0_1  
toString(): dir1\file.txt  
getFileName(): file.txt  
getNameCount(): 2  
getName(0): dir1  
getName(1): file.txt  
subpath(0,2): dir1\file.txt  
getParent(): dir1  
getRoot(): null
```

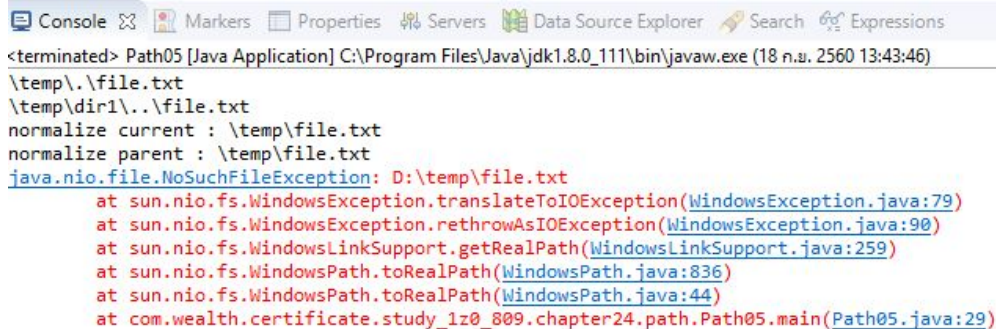


# Paths

When working with paths, you can use:

- `.` to refer to the current directory
- `..` to refer to the parent directory

```
public static void main(String[] args) {  
    // refers to /temp/file.txt  
    Path p1 = Paths.get("/temp/./file.txt");  
    System.out.println(p1);  
  
    // refers to /temp//file.txt  
    Path p2 = Paths.get("/temp/dir1/../file.txt");  
    System.out.println(p2);  
  
    Path path1 = p1.normalize();  
    System.out.println("normalize current : " + path1);  
  
    Path path2 = p2.normalize();  
    System.out.println("normalize parent : " + path2);  
  
    try {  
        // - If LinkOption.NOFOLLOW_LINKS is passed as an argument, symbolic links are not  
        //   followed (by default it does).  
        // - If the path is relative, it returns an absolute path.  
        // - It returns a Path with redundant elements removed (if any).  
        Path realPath = p2.toRealPath();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```



```
Console  [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (18 n.v. 2560 13:43:46)  
<terminated> Path05 [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (18 n.v. 2560 13:43:46)  
\temp\.\file.txt  
\temp\dir1\..\file.txt  
normalize current : \temp\file.txt  
normalize parent : \temp\file.txt  
java.nio.file.NoSuchFileException: D:\temp\file.txt  
    at sun.nio.fs.WindowsException.translateToIOException(WindowsException.java:79)  
    at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:90)  
    at sun.nio.fs.WindowsLinkSupport.getRealPath(WindowsLinkSupport.java:259)  
    at sun.nio.fs.WindowsPath.toRealPath(WindowsPath.java:836)  
    at sun.nio.fs.WindowsPath.toRealPath(WindowsPath.java:44)  
    at com.wealth.certificate.study_1z0_809.chapter24.path.Path05.main(Path05.java:29)
```

Path `toRealPath(LinkOption... options)` throws `IOException`



# Symbolic link








```
C:\WINDOWS\system32>mklink
Creates a symbolic link.
```

```
MKLINK [[/D] | [/H] | [/J]] Link Target
```

```
  /D    Creates a directory symbolic link. Default is a file
        symbolic link.
  /H    Creates a hard link instead of a symbolic link.
  /J    Creates a Directory Junction.
  Link  Specifies the new symbolic link name.
  Target Specifies the path (relative or absolute) that the new link
        refers to.
```

```
mklink softlink.txt target.txt
mklink \H hardlink.txt target.txt
mklink \D softDir dir9
```

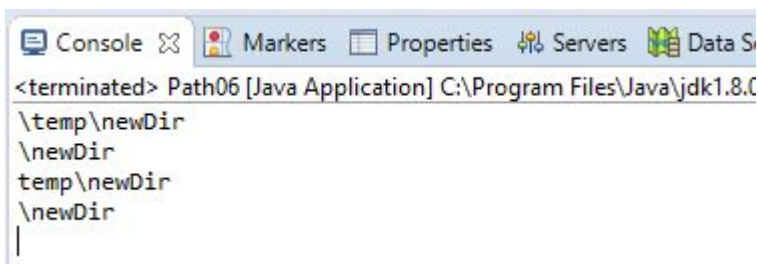
create soft link to File  
create hard link to File  
create soft link to Directory

Name	Date modified	Type	Size
 target.txt - Shortcut	20/9/2017 10:50	Shortcut	3 KB
 softlink.txt	20/9/2017 10:44	.symlink	0 KB
 hardlink.txt	20/9/2017 10:43	Text Document	0 KB
 target.txt	20/9/2017 10:43	Text Document	0 KB
 dir9 - Shortcut	20/9/2017 10:38	Shortcut	3 KB
 softDir	20/9/2017 10:52	File folder	
 dir9	20/9/2017 10:29	File folder	

Ref: <https://www.howtogeek.com/howto/16226/complete-guide-to-symbolic-links-symlinks-on-windows-or-linux/>

## Combine two Paths

```
public static void main(String[] args) {  
    //second path that doesn't have a root element (a partial path), the second path is appended  
    Path path = Paths.get("/temp");  
    System.out.println(path.resolve("newDir")); // \temp\newDir  
  
    //If we have a partial or relative path, and we want to combine it with an absolute path,  
    //this absolute path is returned  
    System.out.println(path.resolve("/newDir")); // \newDir  
  
    Path path2 = Paths.get("temp");  
    System.out.println(path2.resolve("newDir")); // temp\newDir  
    System.out.println(path2.resolve("/newDir")); // \newDir  
}
```



The screenshot shows an IDE console window with tabs for Console, Markers, Properties, Servers, and Data S. The console output is as follows:

```
<terminated> Path06 [Java Application] C:\Program Files\Java\jdk1.8.0_101\bin\java.exe  
\temp\newDir  
\newDir  
temp\newDir  
\newDir  
|
```

## relativize()


`path1.relativize(path2)` is like saying give me a path that shows how to get from `path1` to `path2`.

```
public static void main(String[] args) {
    Path path1 = Paths.get("temp");
    Path path2 = Paths.get("temp/dir1/file.txt");
    Path path1ToPath2 = path1.relativize(path2); // dir1/file.txt
    System.out.println(path1ToPath2);

    //If the paths represent two relatives paths without any other information,
    //they are considered siblings, so you have to go to the parent directory
    //and then go to the other directory
    Path path3 = Paths.get("dir1");
    Path path3ToPath4 = path3.relativize(Paths.get("dir2")); // ../dir2
    System.out.println(path3ToPath4);

    //If both paths are absolute, the result is system-dependent.
    Path path7 = Paths.get("c:\\dir1");
    Path path7ToPath8 = path7.relativize(Paths.get("c:\\dir2")); // ../dir2
    System.out.println(path7ToPath8);

    //If one of the paths is an absolute path, a relative path cannot be constructed
    //because of the lack of information and a llegalArgumentException will be thrown.
    Path path5 = Paths.get("c:\\dir1");
    Path path5ToPath6 = path5.relativize(Paths.get("dir2"));
    System.out.println(path5ToPath6);
}
```



Console | Markers | Properties | Servers | Data Source Explorer | Search | Expressions

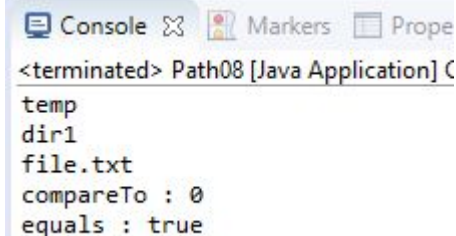
<terminated> Path07 [Java Application] C:\Program Files\Java\jdk1.8.0\_111\bin\javaw.exe (18 n.u. 2560 16:45:11)

dir1\\file.txt  
..\\dir2  
..\\dir2

Exception in thread "main" [java.lang.IllegalArgumentException](#): 'other' is different type of Path  
at sun.nio.fs.WindowsPath.relativize([WindowsPath.java:388](#))  
at sun.nio.fs.WindowsPath.relativize([WindowsPath.java:44](#))  
at com.wealth.certificate.study\_1z0\_809.chapter24.path.Path07.main([Path07.java:29](#))

## Iterable, compareTo, equals

```
public static void main(String[] args) {  
    //Path implements the Iterable  
    Path path1 = Paths.get("c:\\temp\\dir1\\file.txt");  
    for(Path name : path1) {  
        System.out.println(name);  
    }  
  
    Path path2 = Paths.get("c:\\temp\\dir1\\file.txt");  
  
    System.out.println("compareTo : " + path1.compareTo(path2));  
    System.out.println("equals : " + path1.equals(path2));  
}
```



The screenshot shows a Java IDE console window titled "<terminated> Path08 [Java Application] (C". It has tabs for "Console", "Markers", and "Prope". The output text is as follows:

```
temp  
dir1  
file.txt  
compareTo : 0  
equals : true
```

`Path` implements the `Comparable` interface and the `equals()` method to test two paths for equality.

`compareTo()` compares two paths lexicographically. It returns:

- Zero if the argument is equal to the path,
- A value less than zero if this path is lexicographically less than the argument, or
- A value greater than zero if this path is lexicographically greater than the argument.

The `equals()` implementation is system-dependent (for example, it's case insensitive on Windows systems). However, it returns `false` if the argument is not a `Path` or if it belongs to a different file system.

## startsWith, endsWith

```
public static void main(String[] args) {
    Path absPath = Paths.get("c:\\temp\\dir1\\file.txt");
    Path relPath = Paths.get("temp\\dir1\\file.txt");

    // boolean startsWith(Path other)
    System.out.println(absPath.startsWith(Paths.get("c:\\temp\\file.txt"))); // false
    System.out.println(absPath.startsWith(Paths.get("c:\\temp\\dir1\\img.jpg"))); // false
    System.out.println(absPath.startsWith(Paths.get("c:\\temp\\dir1\\"))); // true
    System.out.println(absPath.startsWith(relPath)); // false

    // boolean startsWith(String other)
    System.out.println(relPath.startsWith("t")); // false
    System.out.println(relPath.startsWith("temp")); // true
    System.out.println(relPath.startsWith("temp\\d")); // false
    System.out.println(relPath.startsWith("temp\\dir1")); // true

    // boolean endsWith(Path other)
    System.out.println(absPath.endsWith(Paths.get("file.txt"))); // true
    System.out.println(absPath.endsWith(Paths.get("d:\\temp\\dir1\\file.txt"))); // false
    System.out.println(relPath.endsWith(absPath)); // false

    // boolean endsWith(String other)
    System.out.println(relPath.endsWith("txt")); // false
    System.out.println(relPath.endsWith("file.txt")); // true
    System.out.println(relPath.endsWith("\\dir1\\file.txt")); // false
    System.out.println(relPath.endsWith("dir1\\file.txt")); // true
    System.out.println(absPath.endsWith("dir1\\file.txt")); // true
}
```



# Files

```
public static void main(String[] args) {
    Path absPath = Paths.get(getCurrentPath() + "\\temp\\dir1\\file.txt");
    System.out.println("exists : " + Files.exists(absPath));
    System.out.println("notExists : " + Files.notExists(absPath));
    System.out.println("isReadable : " + Files.isReadable(absPath));
    System.out.println("isWritable : " + Files.isWritable(absPath));
    System.out.println("isExecutable : " + Files.isExecutable(absPath));

    //If cannot find files, throws java.nio.file.NoSuchFileException
    Path absPath2 = Paths.get(getCurrentPath() + "\\temp\\dir1\\file.txt");
    try {
        System.out.println("isSameFile : " + Files.isSameFile(absPath, absPath2));

        //To read a file, we can load the entire file into memory (only useful for small files)
        byte[] bytes = Files.readAllBytes(absPath);

        List<String> readAllLines = Files.readAllLines(absPath);
        System.out.println("readAllLines : " + readAllLines);

        List<String> readAllLinesCharset = Files.readAllLines(absPath, StandardCharsets.UTF_8);
        System.out.println("readAllLinesCharset : " + readAllLinesCharset);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

file.txt

1 line1  
2 line2  
3 line3

Console Markers Properties Servers

<terminated> Files01 (1) [Java Application] C:\Program Files\Java  
exists : true  
notExists : false  
isReadable : true  
isWritable : true  
isExecutable : true  
isSameFile : true  
readAllLines : [line1, line2, line3]  
readAllLinesCharset : [line1, line2, line3]

## Files - readLine

Or to read a file in an efficient way:

```
static BufferedReader newBufferedReader(Path path)
    throws IOException
static BufferedReader newBufferedReader(Path path, Charset cs)
    throws IOException
```

```
public static void main(String[] args) {
    Path path = Paths.get(Files01.getCurrentPath() + "\\temp\\dir1\\file.txt");

    // By default it uses StandardCharsets.UTF_8
    try (BufferedReader reader = Files.newBufferedReader(path, StandardCharsets.ISO_8859_1)) {
        String line = null;
        while ((line = reader.readLine()) != null)
            System.out.println(line);
    } catch (IOException e) {
    }
}
```



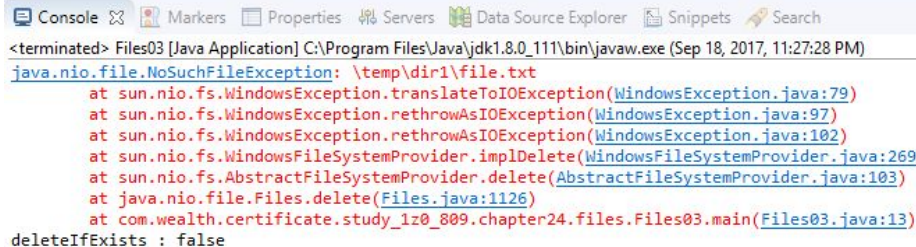
# Files - delete

```
static void delete(Path path) throws IOException
```

```
static boolean deleteIfExists(Path path) throws IOException
```

```
public static void main(String[] args) {
    try {
        Files.delete(Paths.get("/temp/dir1/file.txt"));
        Files.delete(Paths.get("/temp/dir1"));
    } catch (NoSuchFileException nsfe) {
        // If the file/directory doesn't exists
        nsfe.printStackTrace();
    } catch (DirectoryNotEmptyException dnee) {
        // To delete a directory, it must be empty, otherwise, this exception is thrown
        dnee.printStackTrace();
    } catch (IOException ioe) {
        // File permission or other problems
        ioe.printStackTrace();
    }

    try {
        boolean deleteIfExists = Files.deleteIfExists(Paths.get("/temp/dir1/file.txt"));
        System.out.println("deleteIfExists : " + deleteIfExists);
    } catch (DirectoryNotEmptyException dnee) {
        // To delete a directory, it must be empty,
        dnee.printStackTrace();
    } catch (IOException ioe) {
        // File permission or other problems
        ioe.printStackTrace();
    }
}
```



```
Console Markers Properties Servers Data Source Explorer Snippets Search
<terminated> Files03 [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Sep 18, 2017, 11:27:28 PM)
java.nio.file.NoSuchFileException: \temp\dir1\file.txt
    at sun.nio.fs.WindowsException.translateToIOException(WindowsException.java:79)
    at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:97)
    at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:102)
    at sun.nio.fs.WindowsFileSystemProvider.implDelete(WindowsFileSystemProvider.java:269)
    at sun.nio.fs.AbstractFileSystemProvider.delete(AbstractFileSystemProvider.java:103)
    at java.nio.file.Files.delete(Files.java:1126)
    at com.wealth.certificate.study_1z0_809.chapter24.files.Files03.main(Files03.java:13)
deleteIfExists : false
```

# Files - copy

```
static Path copy(Path source, Path target,  
                CopyOption... options) throws IOException
```

return the path to the target file

## Copy File

1. In Target path file doesn't exist - create new one with Target path
2. Existing file in Target - throw `java.nio.file.FileAlreadyExistsException`
3. Existing file in Target, copy with `StandardCopyOption.REPLACE_EXISTING` - new file is created
4. Parent Path doesn't exist - throw `java.nio.file.NoSuchFileException`

## Copy Directory

1. In Target path directory doesn't exist - create new folder with Target path without content (empty folder)
  2. Existing directory in Target - throw `java.nio.file.FileAlreadyExistsException`
  3. Directory in Target is not empty, copy with `StandardCopyOption.REPLACE_EXISTING` - throw `java.nio.file.DirectoryNotEmptyException`
  4. Parent Path doesn't exist - throw `java.nio.file.NoSuchFileException`
- **StandardCopyOption.REPLACE\_EXISTING**  
Performs the copy when the target already exists. If the target is a symbolic link, the link itself is copied and If the target is a non-empty directory, a `FileAlreadyExistsException` is thrown.
  - **StandardCopyOption.COPY\_ATTRIBUTES**  
Copies the file attributes associated with the file to the target file. The exact attributes supported are file system and platform dependent, except for last-modified-time, which is supported across platforms.
  - **LinkOption.NOFOLLOW\_LINKS**  
Indicates that symbolic links should not be followed, just copied.

## Files - copy

```
public static void main(String[] args) {  
    try {  
        Path in = Paths.get(Files01.getCurrentPath() + "\\temp\\dir1\\in.txt");  
        Path out = Paths.get(Files01.getCurrentPath() + "\\temp\\dir2\\out.txt");  
  
        Path result = Files.copy(in, out, StandardCopyOption.REPLACE_EXISTING);  
        System.out.println(result);  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
  
    try {  
        Path in = Paths.get(Files01.getCurrentPath() + "\\temp\\dir3");  
        Path out = Paths.get(Files01.getCurrentPath() + "\\temp\\dir5");  
  
        Path result = Files.copy(in, out, StandardCopyOption.REPLACE_EXISTING, StandardCopyOption.COPY_ATTRIBUTES);  
        System.out.println(result);  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

## Files - copy

```
static long copy(InputStream in, Path target,  
                  CopyOption... options) throws IOException
```

```
static long copy(Path source,  
                  OutputStream out) throws IOException
```

```
public static void main(String[] args) {  
    try (InputStream in = new FileInputStream(Files01.getCurrentPath() + "\\temp\\dir1\\in.txt");  
        OutputStream out = new FileOutputStream(Files01.getCurrentPath() + "\\temp\\dir1\\out.txt")) {  
  
        Path path = Paths.get(Files01.getCurrentPath() + "\\temp\\dir3\\in.csv");  
        // Copy stream data to a file  
        Files.copy(in, path);  
        // Copy the file data to a stream  
        Files.copy(path, out);  
    } catch (IOException e) {  
  
    }  
}
```

# Files - move or rename

```
static Path move(Path source, Path target,  
                CopyOption... options) throws IOException
```

- **StandardCopyOption.REPLACE\_EXISTING**  
Performs the move when the target already exists. If the target is a symbolic link, only the link itself is moved.
- **StandardCopyOption.ATOMIC\_MOVE**  
Performs the move as an atomic file operation. If the file system does not support an atomic move, an exception is thrown.

If the target exists, trying to move a non-empty directory will throw a `DirectoryNotEmptyException`.

```
public static void main(String[] args) {  
    try {  
        String in1 = Files01.getCurrentPath() + "\\temp\\dir6\\in1.txt";  
        String in2 = Files01.getCurrentPath() + "\\temp\\dir6\\in2.txt";  
        String move1 = Files01.getCurrentPath() + "\\temp\\dir7\\move1.txt";  
  
        Files.move(Paths.get(in1), Paths.get(in2), StandardCopyOption.ATOMIC_MOVE);  
        Files.move(Paths.get(in2), Paths.get(move1));  
        Files.move(Paths.get(move1), Paths.get(in1), StandardCopyOption.REPLACE_EXISTING);  
  
        String dir8 = Files01.getCurrentPath() + "\\temp\\dir8";  
        String dir9 = Files01.getCurrentPath() + "\\temp\\dir9";  
        Files.move(Paths.get(dir8), Paths.get(dir9));  
        Files.move(Paths.get(dir9), Paths.get(dir8));  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

When the move is performed as a non-atomic operation, and an `IOException` is thrown, then the state of the files is not defined. The original file and the target file may both exist, the target file may be incomplete or some of its file attributes may not been copied from the original file.



# Files - metadata

```
public static void main(String[] args) {
    Path path = Paths.get(Files01.getCurrentPath() + "\\temp\\dir1\\file.txt");
    try {
        //Returns the size of a file (in bytes).
        System.out.println("size : " + Files.size(path) + " bytes.");

        //Tests whether a file is a directory.
        System.out.println("isDirectory : " + Files.isDirectory(path));

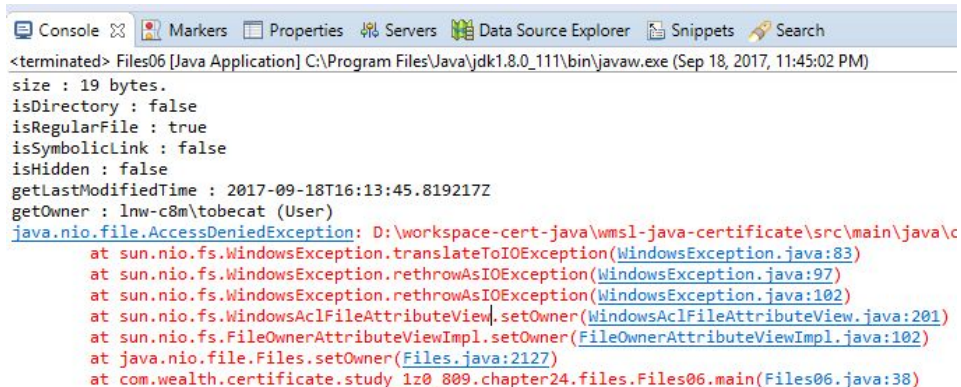
        //Tests whether a file is a regular file.
        System.out.println("isRegularFile : " + Files.isRegularFile(path));

        //Tests whether a file is a symbolic link.
        System.out.println("isSymbolicLink : " + Files.isSymbolicLink(path));

        //Tells whether a file is considered hidden.
        System.out.println("isHidden : " + Files.isHidden(path));

        //Returns or updates a file's last modified time.
        FileTime fileTime = Files.getLastModifiedTime(path);
        System.out.println("getLastModifiedTime : " + fileTime);
        Files.setLastModifiedTime(path, fileTime);

        //Returns or updates the owner of the file.
        UserPrincipal owner = Files.getOwner(path);
        System.out.println("getOwner : " + owner);
        Files.setOwner(path, owner);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



```
<terminated> Files06 [Java Application] C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Sep 18, 2017, 11:45:02 PM)
size : 19 bytes.
isDirectory : false
isRegularFile : true
isSymbolicLink : false
isHidden : false
getLastModifiedTime : 2017-09-18T16:13:45.819217Z
getOwner : lnw-c8m\tobecat (User)
java.nio.file.AccessDeniedException: D:\workspace-cert-java\wmsl-java-certificate\src\main\java\c
    at sun.nio.fs.WindowsException.translateToIOException(WindowsException.java:83)
    at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:97)
    at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:102)
    at sun.nio.fs.WindowsAclFileAttributeView.setOwner(WindowsAclFileAttributeView.java:201)
    at sun.nio.fs.FileOwnerAttributeViewImpl.setOwner(FileOwnerAttributeViewImpl.java:102)
    at java.nio.file.Files.setOwner(Files.java:2127)
    at com.wealth.certificate.study_1z0_809.chapter24.files.Files06.main(Files06.java:38)
```

# FileTime

```
static FileTime from(Instant instant)
static FileTime from(long value, TimeUnit unit)
static FileTime fromMillis(long value)
```

And from a `FileTime` we can get an `Instant` or milliseconds as `long` :

```
Instant toInstant()
long toMillis()
```

```
public static void main(String[] args) {
    try {
        Path path = Paths.get(Files01.getCurrentPath() + "\\temp\\dir1\\file.txt");
        FileTime fileTime = Files.getLastModifiedTime(path);
        System.out.println("getLastModifiedTime : " + fileTime);

        Files.setLastModifiedTime(path, FileTime.from(fileTime.toInstant()));
        System.out.println("from(Instant instant) : " + Files.getLastModifiedTime(path));

        Files.setLastModifiedTime(path, FileTime.from(5, TimeUnit.DAYS));
        System.out.println("from(long value, TimeUnit unit) : " + Files.getLastModifiedTime(path));

        Files.setLastModifiedTime(path, FileTime.fromMillis(fileTime.toMillis() + 1000));
        System.out.println("fromMillis(long value) : " + Files.getLastModifiedTime(path));

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



The screenshot shows an IDE interface with a console window. The console displays the output of the Java application, which includes the file path, the last modified time, and the results of various FileTime operations. The output is as follows:

```
<terminated> Files08 [Java Application] C:\Program Files\Java\jdk1.8.0_11
getLastModifiedTime : 2017-09-18T16:13:50.819Z
from(Instant instant) : 2017-09-18T16:13:50.819Z
from(long value, TimeUnit unit) : 1970-01-06T00:00:00Z
fromMillis(long value) : 2017-09-18T16:13:51.819Z
```



# Attributes

- **java.nio.file.attribute.BasicFileAttributeView**  
Provides a view of basic attributes supported by all file systems.
- **java.nio.file.attribute.DosFileAttributeView**  
Extends `BasicFileAttributeView` to support additionally a set of DOS attribute flags that are used to indicate if the file is read-only, hidden, a system file, or archived.
- **java.nio.file.attribute.PosixFileAttributeView**  
Extends `BasicFileAttributeView` with attributes supported on POSIX systems, such as Linux and Mac. Examples of these attributes are file owner, group owner, and related access permissions.

You can get a file attribute view of a given type to read or update a set of attributes with the method:

```
static <V extends FileAttributeView> V getFileAttributeView(  
    Path path, Class<V> type, LinkOption... options)
```

Most of the time, you'll work with the read-only versions of the file views.  
directly:

```
static <A extends BasicFileAttributes> A  
    readAttributes(Path path, Class<A> type,  
        LinkOption... options)  
    throws IOException
```

# Attributes

```
public static void main(String[] args) {
    try {
        Path path = Paths.get(Files01.getCurrentPath() + "\\temp\\dir1\\in.txt");
        BasicFileAttributeView view = Files.getFileAttributeView(path, BasicFileAttributeView.class);

        FileTime lastModifiedTime = FileTime.from(Instant.now());
        FileTime lastAccessTime = FileTime.from(Instant.now());
        FileTime createTime = FileTime.from(Instant.now());

        // If any argument is null,
        // the corresponding value is not changed
        view.setTimes(lastModifiedTime, lastAccessTime, createTime);

        // Get a class with read-only attributes
        BasicFileAttributes readOnlyAttrs = view.readAttributes();

        // read-only versions of the file views
        BasicFileAttributes attr = Files.readAttributes(path, BasicFileAttributes.class);
        // Size in bytes
        System.out.println("size(): " + attr.size());
        // Unique file identifier (or null if not available)
        System.out.println("fileKey(): " + attr.fileKey());

        System.out.println("isDirectory(): " + attr.isDirectory());
        System.out.println("isRegularFile(): " + attr.isRegularFile());
        System.out.println("isSymbolicLink(): " + attr.isSymbolicLink());
        // Is something other than a file, directory, or symbolic link?
        System.out.println("isOther(): " + attr.isOther());

        // The following methods return a FileTime instance
        System.out.println("creationTime(): " + attr.creationTime());
        System.out.println("lastModifiedTime(): " + attr.lastModifiedTime());
        System.out.println("lastAccessTime(): " + attr.lastAccessTime());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Console | Markers | Properties | Servers | Data Source Explorer

<terminated> Files09 [Java Application] C:\Program Files\Java\jdk1.8.0\_111\bin

```
size(): 13
fileKey(): null
isDirectory(): false
isRegularFile(): true
isSymbolicLink(): false
isOther(): false
creationTime(): 2017-09-19T15:28:06.387Z
lastModifiedTime(): 2017-09-19T15:28:06.387Z
lastAccessTime(): 2017-09-19T15:28:06.387Z
```