I/O

Java Fundamentals

Tõnis Pool

2017, Tallinn

Agenda

- Exceptions
- I/O Streams
- File I/O

Some Examples

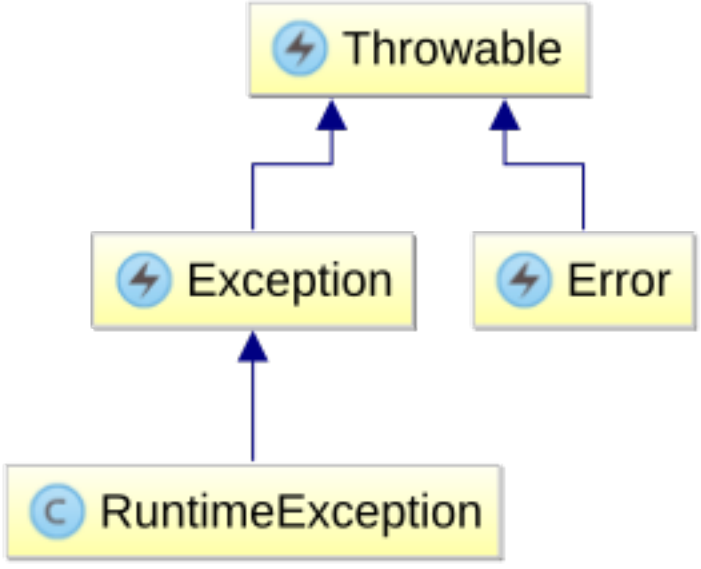
- Github repository http://bit.ly/jf-github
- Checkout the project (or fork)
- Import the **io** sub project to your IDE

I/O

- Input/output or I/O is the communication between the program and outside world.

EXCEPTIONS

Types of Exceptions



Error Handling

- Most I/O methods may throw an `IOException`
- **Good**:
  - Throw an exception up to the caller as much as possible
  - Only wrap an existing exception if it's really necessary or you add extra info to it
  - Handle exceptions centrally
- **Bad**:
  - Catch an exception and do nothing
  - Catch an exception and only print it out

Closing

```
Closeable resource = null;
try {
  resource = obtainResource();
  processResource(resource);
} finally {
  if (resource != null) {
    resource.close();
  }
}
```

Closing

- If you open a resource you **must also close it**.
- Closing must be done in a **finally** block or using **try-with-resources** statement.
- Leaving a resource open might
  - Make your application crash or hang as it may exceed the limit of open resources allowed
  - Keep a file locked so it cannot be deleted

Homework

- Use the following to read lines:

```
Files.lines(Paths.get("lorem-ipsum.txt"))
(Returns a Stream<String>)
```

Unit test with expected output is included in the homework skeleton.

Closing streams

- All **IO backed** streams must be closed
  - `Files.`***`lines`***
  - `Files.`***`list`***
  - `Files.`***`walk`***
  - **…**

Good Practice

- A resource should be closed in the same method where it was opened.
- If a method takes an open resource as input it’s the caller’s responsibility to close it afterwards.

Closing

```
Closeable resource = null;
try {
  resource = obtainResource();
  processResource(resource);     // <-- exception?
} finally {
  if (resource != null) {
    resource.close();            // <-- exception?
  }
}
```

Closing

```
Exception in thread "main" java.io.IOException: errror during closing!
     at exceptions.ClassicHandling.lambda$obtainResource$2(ClassicHandling.java:25)
     at exceptions.ClassicHandling$$Lambda$1/1349393271.close(Unknown Source)
     at exceptions.ClassicHandling.main(ClassicHandling.java:14)
     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
     at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
     at java.lang.reflect.Method.invoke(Method.java:497)
     at com.intellij.rt.execution.application.AppMain.main(AppMain.java:147)
```

try-with-resources

```
try (Closeable resource = obtainResource()) {
  processResource(resource);     // <-- exception?
}
```

try-with-resources

```
Exception in thread "main" java.io.IOException: error during processing!
     at exceptions.TryWithResourcesHandling.processResource(TryWithResourcesHandling.java:15)
     at exceptions.TryWithResourcesHandling.main(TryWithResourcesHandling.java:9)
     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
     at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
     at java.lang.reflect.Method.invoke(Method.java:497)
     at com.intellij.rt.execution.application.AppMain.main(AppMain.java:147)
     Suppressed: java.io.IOException: errror during closing!
          at
exceptions.TryWithResourcesHandling.lambda$obtainResource$0(TryWithResourcesHandling.java:19)
          at exceptions.TryWithResourcesHandling$$Lambda$1/1349393271.close(Unknown Source)
          at exceptions.TryWithResourcesHandling.main(TryWithResourcesHandling.java:10)
          ... 5 more
```

try-with-resources sugar

```
Closeable resource = null;
IOException myException = null;
try {
  resource = obtainResource();
  processResource(resource);    // <-- exception?
}
catch (IOException e) {
  myException = e;
  throw e;
}
finally {
  if (resource != null) {
    if (myException != null) {
      try {
        resource.close();
      }
      catch (Throwable closingException) {
        myException.addSuppressed(closingException);
      }
    }
    else {
      resource.close();
    }
  }
}
```

No Java 7?

- Log / swallow suppressed exceptions yourself
- Use Guava Closer
  https://github.com/google/guava/wiki/ClosingResourcesExplained
  - Logs suppressed exceptions
- Use retrolambda
  https://github.com/orfjackal/retrolambda
  - Swallows suppressed exceptions

I/O STREAMS

Read More

- I/O Streams Java Tutorial
- https://docs.oracle.com/javase/tutorial/essential/io/streams.html

Stream

- Stream is a sequence of data.
- Program uses
  - input stream to read data from a source.
  - output stream to write data to a destination.

Source and Destination

- Disk File
- Device
- Other Program
- Network Socket
- Memory

Kinds of Data

- Bytes
- Primitive data types
- Localized characters
- Objects

Byte Streams

	Input	Output
Base	InputStream	OutputStream
File	FileInputStream	FileOutputStream
Memory	ByteArray-InputStream	ByteArray-OutputStream
Buffer	BufferedInputStream	BufferedOutputStream

Input Stream

```
abstract class InputStream implements Closeable {
  int read()
  int read(byte b[])
  int read(byte b[], int off, int len)
  void close()
  …
}
```

Output Stream

```
abstract class OutputStream implements
  Closeable, Flushable {
  void write(int b)
  void write(byte b[])
  void write(byte b[], int off, int len)
  void flush()
  void close()
}
```

Flushing

```
abstract class OutputStream implements

  /**
   * .. guarantees only that bytes previously written to
   * the stream are passed to the operating system for writing;
   * it does not guarantee that they are actually written to a
   * physical device such as a disk drive.
   */
  void flush()
}
```

To force writing to disk use:

```
FileOutputStream out = new FileOutputStream(filename);
...
out.flush();
out.getFD().sync();
```

RandomAccessFile

- Behaves like an **InputStream** and an **OutputStream**
- Random access: can seek to a position to read/write

```
class RandomAccessFile {
  RandomAccessFile(File file, String mode);
  int read();
  void write(int b);
  void seek(long pos);
}
```

RandomAccessFile (2)

The mode argument specifies the access mode in which the file is to be opened:

“r”	Open for reading only.
“rw”	Open for reading and writing.
“rws”	“**rw**" + require that every update to the file's content or metadata be written to disk.
“rwd”	“**rw**" + require that every update to the file's content be written to disk

Character Streams

	Input	Output
Base	Reader	Writer
File	FileReader	FileWriter
Memory	StringReader CharArrayReader	StringWriter CharArrayWriter
Buffer	BufferedReader	BufferedWriter
Adapter	InputStreamReader	OutputStreamWriter

Reader

```
abstract class Reader implements Readable,
  Closeable {
  int read()
  int read(char cbuf[])
  int read(char cbuf[], int off, int len)
  void close()
  ...
}
```

Writer

```
abstract class Writer implements Appendable,
  Closeable, Flushable {
  void write(int c)
  void write(char cbuf[])
  void write(char cbuf[], int off, int len)
  void flush()
  void close()
  …
}
```

Adapters

```
Reader reader = new FileReader("input.data");
```

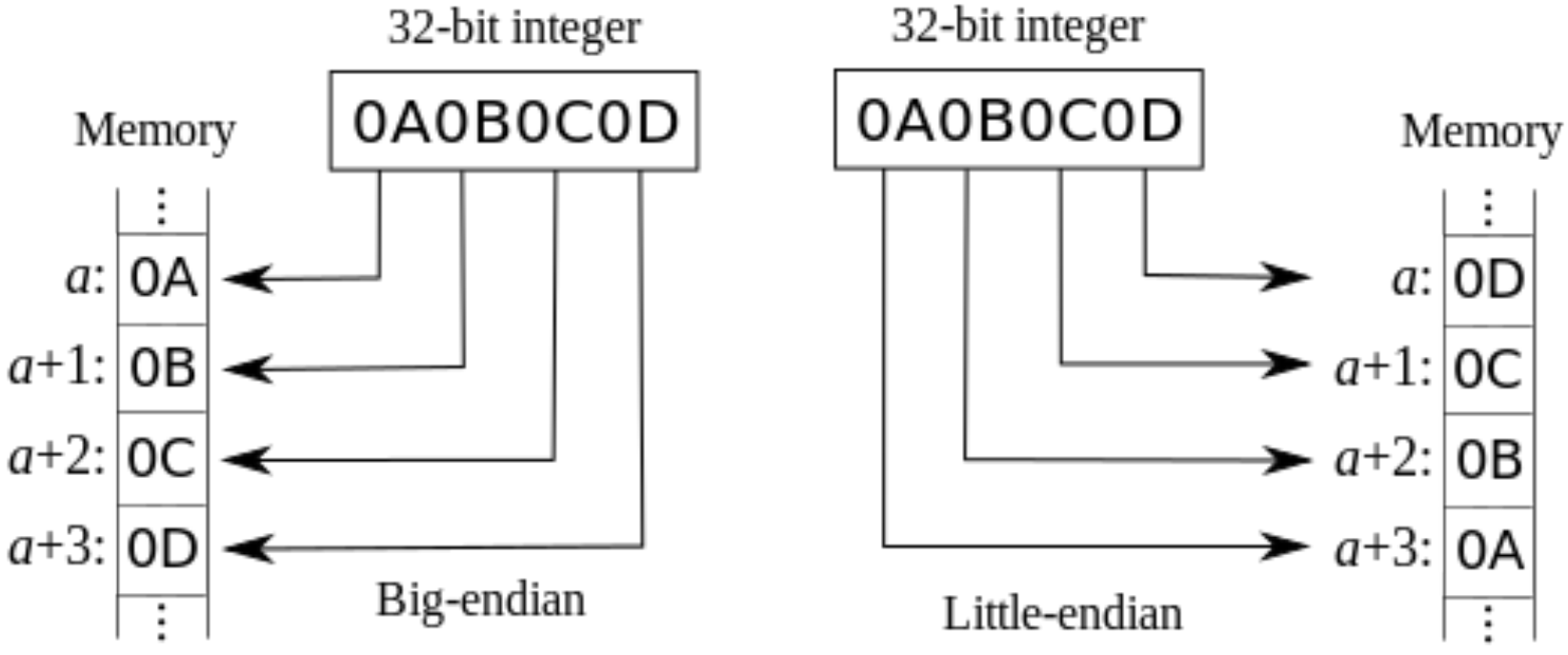
```
InputStream is = new
  FileInputStream("input.data");
Reader reader = new
  InputStreamReader(is, "UTF-8");
```

Primitive Data Streams

	Input	Output
Data	DataInputStream	DataOutputStream

- Reads/writes Java primitives and Strings.
- Numbers are stored as **big-endian**.

Endianness



Internet protocols

Intel CPUs

Object Streams (Serialization)

	Input	Output
Object	ObjectInputStream	ObjectOutputStream

- Reads/writes any serializable Java object.
- Skips all **static** and **transient** fields.
- Can handle object graphs with cycles (values are written only once).
- Uses Java-specific format.

Copying and Transforming Files

- How to copy a file in Java?
- How to change it while copying?
- What if the file does not fit into memory?

Copy Whole File

```
Path src = Paths.get("input.data");
Path dest = Paths.get("output.data");

Files.copy(src, dest);
```

+ Fast (native implementation)
+ Doesn’t load the whole file into memory
– Cannot change the contents

Read + Write Whole File

```
Path src = Paths.get("input.data");
Path dest = Paths.get("output.data");

byte[] data = Files.readAllBytes(src);
Files.write(src, data);
```

+ Can change the contents
– Requires the whole file to fit into memory

Copy by Byte

```
try (InputStream in = Files.newInputStream(src);
     OutputStream out = Files.newOutputStream(dest)) {
   int b;
   while ((b = in.read()) != -1) {
     out.write(b);
   }
}
```

+ Can change the contents
– ~1000x slower

Copy by Byte via Buffered Streams

```
try (InputStream in = new BufferedInputStream(
        Files.newInputStream(src));
    OutputStream out = new BufferedOutputStream(
        Files.newOutputStream(dest))) {
  int b;
  while ((b = in.read()) != -1) {
    out.write(b);
  }
}
```

- \+ Can change the contents
- – ~4x slower

Copy by Buffer via Stream

```
try (InputStream in = Files.newInputStream(src);
     OutputStream out = Files.newOutputStream(dest)) {
  byte[] buffer = new byte[4096];
  int length;
  while ((length = in.read(buffer)) != -1) {
    out.write(buffer, 0, length);
  }
}
```

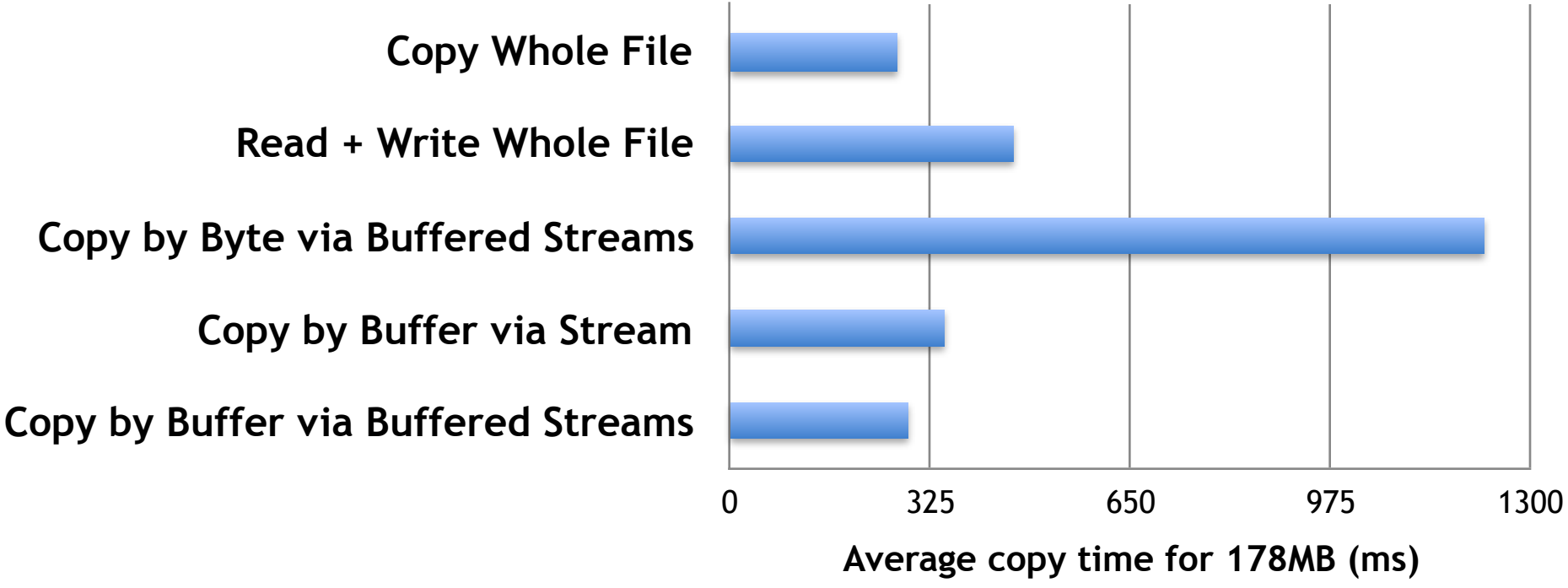
- \+ Can change the contents
- – Still a bit slower

Copy by Buffer via Buffered Streams

```
try (InputStream in = new BufferedInputStream(
        Files.newInputStream(src));
    OutputStream out = new BufferedOutputStream(
        Files.newOutputStream(dest))) {
  byte[] buffer = new byte[4096];
  int length;
  while ((length = in.read(buffer)) != -1) {
    out.write(buffer, 0, length);
  }
}
```

+ Can change the contents
+ Almost as fast a copying the whole file

Copy Time


Copy Whole File
Read + Write Whole File
Copy by Byte via Buffered Streams
Copy by Buffer via Stream
Copy by Buffer via Buffered Streams
0
325
650
975
1300
Average copy time for 178MB (ms)

Copy All Lines

```
List<String> lines = Files.readAllLines(src);
Files.write(dest, lines);
```

+ Can change the contents
– Requires the whole file to fit into memory

Copy All Lines (Java 7)

```
Charset cs = StandardCharsets.UTF_8;
List<String> lines = Files.readAllLines(src, cs);
Files.write(dest, lines, cs);
```

Reading Lines (Java 8)

```
Path file = Paths.get("dog.txt");

try (Stream<String> s = Files.lines(file,
StandardCharsets.UTF_8)) {
  s.forEach(System.out::println);
}

try (Stream<String> s = Files.lines(file)) {
  s.forEach(System.out::println);
}
```

Charset

- … is a named mapping between sequences of characters and sequences of bytes.

- Use **UTF-8** unless needed otherwise.
- Java 8 has some shortcut methods for using it.
- Provide it explicitly to **avoid using system default**.

```
if(System.getProperty("os.name")
    .toUppercase().equals("WINDOWS") {
  doStuffSpecificForWindows();
} else {
  doStuffSpecificForUnix();
}
```

Locale

- A Locale object represents a specific geographical, political or cultural region.
- Locale-sensitive operations:
  - Converting characters to upper and lower case
  - Formatting numbers, currencies, percentages
- Use system default only if you **really** want it.

Locale.ENGLISH

Input	toLowerCase()	toUpperCase()
i	i	I
I	i	I

new Locale("tr", "TR")

Input	toLowerCase()	toUpperCase()
i	i	İ
I	ı	I

- If you don’t provide Locale to those methods they use **system default**.
- So your application may be broken in Turkey.

LiveRebel / LR-1938

Liverebel does not work with Turkish locale

Edit | Comment | Assign | Reopen Issue | Admin

PropertiesEnhancer.invokeReadProperty fails when JVM default locale is Turkish

Reported by Neeme Praks | May 31st, 2012 @ 02:02 PM

Framework version: 1.2.4
Platform you're using: Oracle JDK

Reproduction steps:

https://play.lighthouseapp.com/projects/57987-play-framework/tickets/1541

Formatting

```
int i = 2; double r = Math.sqrt(i);
System.out.format(
  "The square root of %d is %f.%n", i, r);

String key = "foo"; boolean value = true;
System.out.format("%s: %b%n", key, value);
```

See Formatter Javadoc

Printing Text and Bytes to a File

```
try (PrintStream out = new PrintStream(
    new FileOutputStream("print.out"))) {
  out.println("Hello " + name);
  out.format("Hello %s%n", name);
  out.write(bytes);
}
```

Uses platform's default character encoding!

Printing Only Text to a File

```
try (PrintWriter out = new PrintWriter(
     new File("print.txt"), "UTF-8")) {
   out.println("Hello " + name);
   out.format("Hello %s%n", name);
// out.write(bytes); // DOES NOT COMPILE
}
```

File Locks

```
File lockFile = null; // some file to lock
try (RandomAccessFile raf = new
        RandomAccessFile(lockFile, "rw");
     FileLock lock =
        raf.getChannel().lock()) { // #lock blocks!
   System.out.println(
      "Locked: " + lockFile + " with " + lock);
}
```

File Locks

- Locks are **advisory**
- Either *shared* or *exclusive*
- File locks are held on behalf of the **entire Java virtual machine**.
    - Can’t lock the same region from multiple threads

FILE I/O

Read More

- File I/O (Featuring NIO.2) Java Tutorial
- http://docs.oracle.com/javase/tutorial/essential/io/fileio.html

Java Classes

`java.io.File`	File API for Java 1…6
`java.nio.file.Path`	File path instance in Java 7+
`java.nio.file.Files`	File operations in Java 7+
`java.net.URI`	Identity of a resource
`java.net.URL`	Identity + access mechanism (e.g. http://)

File API

- File paths
- File operations
- Managing metadata
- Walking the file tree
- Finding files
- Watching a directory for changes

Paths

```
Paths.get("c:\\example\\data\\input.txt");
Paths.get("/example/data/input.txt");
Paths.get("example/data/input.txt");
Paths.get("example\\data\\input.txt");
Paths.get("example", "data", "input.txt");
Paths.get("/", "example", "data", "input.txt");
Paths.get(URI.create("file:/example/data/input.txt"));
```

For a Relative Path

Method	Output
toString()	example/data/../input
isAbsolute()	false
getFileName()	input
getParent()	example/data/..
getRoot()	null
normalize()	example/input
toAbsolutePath()	/home/User/example/data/../input

For an Absolute Path

Method	Output
toString()	/home/user/../foo
isAbsolute()	true
getFileName()	foo
getParent()	/home/user/..
getRoot()	/
normalize()	/home/foo
toAbsolutePath()	/home/user/../foo

Relative Paths

```
Path home = Paths.get("/Users/mati");

Path docAbsolute = home.resolve(”Dropbox/
Documents”);
// /Users/mati/Dropbox/Documents


Path docRelative = home.relativize(docAbsolute);
// Dropbox/Documents
```

Real Paths

```
Path p = Paths.get("/usr/bin/../bin/java");

Path p2 = p.toRealPath();
// /System/Library/Frameworks/JavaVM.framework/
Versions/A/Commands/java

Path p3 =
p.toRealPath(LinkOption.NOFOLLOW_LINKS);
// /usr/bin/java
```

java.nio.file.Path ⇔ java.io.File

```
Path path = Paths.get("/Users/mati");
File file = path.toFile();
Path path2 = file.toPath();
```

Path ⇔ URI ⇔ URL

```
Path path = Paths.get(“/Users/mati");
URI uri = path.toUri(); // file:///Users/mati
URL url = uri.toURL(); // file:/Users/mati
URI uri2 = url.toURI(); // file:/Users/mati
Path path2 = Paths.get(uri2); // /Users/mati
```

File Checks

```
Path p = Paths.get("/Users/kati");
boolean exists = Files.exists(p); // true
boolean notExists = Files.notExists(p); // false
boolean isDir = Files.isDirectory(p); // true
boolean isFile = Files.isRegularFile(p); // false
```

Where it is not possible to determine if a file exists or not then both *exists*() and *notExists*() methods return **false**.

Checking Symbolic Links

```
Path p = Paths.get(“/usr/bin/java");
boolean isFile = Files.isRegularFile(p); // true
boolean isFile2 = Files.isRegularFile(p,
  LinkOption.NOFOLLOW_LINKS); // false
boolean isLink = Files.isSymbolicLink(p); // true
```

Size and Last Modified Time

```
Path p = Paths.get("/usr/bin/java");

long size = Files.size(p); // 54624

FileTime time = Files.getLastModifiedTime(p);
// 2015-07-14T13:41:40Z
long millis = time.toMillis(); // 1436881300000
```

Resolution depends on OS!

Listing Directory (java.util.stream)

```
Path dir = Paths.get("/Users/herbert");
try (Stream<Path> s = Files.list(dir)) {
  s.forEach(System.out::println);
}
try (Stream<Path> s = Files.list(dir){
  s.filter(Files::isDirectory)
   .forEach(System.out::println);
}
```

Since Java 8

Listing Directory (Java 7)

```
Path dir = Paths.get("/Users/herbert");

try (DirectoryStream<Path> ds =
Files.newDirectoryStream(dir)) {
  ds.forEach(System.out::println);
}
```

Listing Directory (2)

```
try (DirectoryStream<Path> ds =
Files.newDirectoryStream(dir, Files::isDirectory)) {
  ds.forEach(System.out::println);
}
```

Second argument: Filter<? super Path> filter

Listing Directory (3)

```
try (DirectoryStream<Path> ds =
Files.newDirectoryStream(dir, ".*")) {
  ds.forEach(System.out::println);
}
```

Second argument: String glob

Walking File Tree

```
try (Stream<Path> s = Files.walk(root)) {
  s.forEach(System.out::println);
}

try (Stream<Path> s = Files.walk(root)) {
   s.filter(Files::isDirectory)
    .forEach(System.out::println);
}
```

Since Java 8

Walking File Tree (2)

```
Files.walkFileTree(root, new SimpleFileVisitor<Path>() {

   public FileVisitResult preVisitDirectory(Path dir,
BasicFileAttributes attrs) throws IOException {
      System.out.println("Dir: " + dir);
      return FileVisitResult.CONTINUE;
   }
   public FileVisitResult visitFile(Path file,
BasicFileAttributes attrs) throws IOException {
      System.out.println("File: " + file);
      return FileVisitResult.CONTINUE;
   }
});
```

Deleting Files

```
Path file = Paths.get("/Users/laura/delete-file");

Files.delete(file); // May throw
NoSuchFileException

Files.deleteIfExists(file);
```

Also works in case of empty directories.

Deleting Directories

```
Path root = Paths.get("/Users/madis/delete-
dir");


FileUtils.forceDelete(root.toFile());
// May throw FileNotFoundException


if (Files.exists(root)) {
  FileUtils.forceDelete(root.toFile());
}
```

Commons IO

https://commons.apache.org/proper/commons-io/

- IOUtils, FileUtils and more
- Only works with **java.io** not **java.nio.file**
- Use Path.*toFile*() and File.*toPath*() for convertion

Creating a File

```
Path file = Paths.get("/Users/anu/jf/newDir/new.txt");


Files.createFile(file);
// May throw FileAlreadyExistsException
// or NoSuchFileException
```

Creating a File (2)

```
Path file = Paths.get("/Users/anu/jf/newDir/new.txt");

Files.createDirectory(file.getParent());
// May throw FileAlreadyExistsException
// or NoSuchFileException

Files.createFile(file);
```

Creating a File (3)

```
Path file = Paths.get("/Users/anu/jf/newDir/new.txt");

Files.createDirectories(file.getParent());
// It might be either new or existing directory

Files.createFile(file);
// May throw FileAlreadyExistsException
// or NoSuchFileException
```

Creating a File (4)

```
Path parent = file.getParent();
if (Files.exists(parent)) {

FileUtils.cleanDirectory(parent.toFile());
}
else {
  Files.createDirectories(parent);
}
Files.createFile(file); // Works OK
```

Creating a Unique File

```
Path file = Files.createTempFile("homework", ".temp");
Path dir = Files.createTempDirectory("homework");
```

- Uses temporary directory of the user.
- The result path also includes a unique Id.
- Theses files are not deleted automatically.

Creating a Unique File (2)

```
Path parent = Paths.get("/Users/toomas/tmp");
Path file = Files.createTempFile(parent,
"homework", ".temp");
Path dir = Files.createTempDirectory(parent,
"homework");
```

- Using a dedicated folder simplifies cleaning it on startup or shutdown (via a shutdown hook).
- Also might affect atomicity of moving files.

Moving Files

```
Path src = Paths.get("unicorn");
Path dest = Paths.get("unicorn.moved");

Files.move(src, dest);

Files.move(src, dest,
StandardCopyOption.ATOMIC_MOVE);
// May throw AtomicMoveNotSupportedException
```

FILE I/O IN JAVA 6

File Paths

java.nio.file.Path	java.io.File
Path p = Paths.*get*(”files”)	File f = new File(”files”)
Path c = p.resolve("foo.txt”)	File c = new File(f, "foo.txt")
Path name = p.getFileName()	String name = f.getName()
Path parent = p.getParent()	File pf = f.getParentFile() String pp = f.getParent()
Path a = p.toAbsolutePath()	File af = f.getAbsoluteFile() String ap = f.getAbsolutePath()
boolean ia = p.isAbsolute()	boolean ia = f.isAbsolute()

Basic Metadata

java.nio.file.Path	java.io.File
Path p = Paths.*get*(”foo.txt”)	File f = new File(”foo.txt”)
boolean exists = Files.*exists*(p)	boolean exists = f.exists()
boolean f = Files.*isRegularFile*(p)	boolean f = f.isFile()
boolean d = Files.*isDirectory*(p)	boolean d = f.isDirectory()
long size = Files.*size*(p)	long size = f.length()
FileTime lm = Files.*getLastModifiedTime*(p)	long lm = f.lastModified()

Operations

java.nio.file.Path	java.io.File
Path p = Paths.*get*(”foo.txt”)	File f = new File(”foo.txt”)
Files.*createFile*(p)	boolean ok = f.createNewFile()
Files.*createDirectory*(p)	boolean ok = f.mkdir()
Files.*createDirectories*(p)	boolean ok = f.mkdirs()
Files.*move*(p1, p2)	boolean ok = f1.renameTo(f2)
Files.*delete*(p) boolean existed = Files.*deleteIfExists*(p)	boolean ok = f.delete()

Summary

- Remember to **close** the streams.
  - Use **try-with-resources** auto closing.
- Don’t load potentially big files into memory.
- Avoid hard-coding **file separators**.
- Use **java.nio.file** instead of **java.io.File**.
- Also remember FileUtils in **Commons IO**.

QUESTIONS?

Homework

https://github.com/JavaFundamentalsZT/jf-hw-packer

Homework

Write a Java program which packs and unpacks files without compression.

1. Implement the following interface:

```
interface Packer {
  void pack(Path inputDir, Path outputArchive);
  void unpack(Path inputArchive, Path outputDir);
}
```

2. It should accept both relative and absolute paths.

...

Homework (2)

3. Use **DataInputSteam** and **DataOutputStream**
4. It should handle directories recursively.
5. Create missing parent directories automatically.
6. No support for empty directories.
7. Support big files that don’t fit into memory at once.
8. Buffer data for better performance.
9. Close all resources properly.

File Format of the Archive

- Whole Archive = Archive Type + File $Chunk_1$ + File $Chunk_2$ + … + File $Chunk_n$

- Archive Type = 42 (1 fixed byte)

File Format of the Archive (2)

- File Chunk = File Path
  + File Length + File Contents
- File Path – bytes of a String of a relative path in the archive separated by / characters (use readUTF()/writeUTF() methods)
- File Length – 8 bytes showing how many bytes does the File Contents take (big endian long)
- File Contents – actual file in the archive

Unit Tests

- The project has a set of unit tests.
- To get the maximum points all those tests must pass even with limited memory (run `mvn clean test` from command line).
- Existing tests cannot be altered.