PRAKTIKUM SW2

NIO

New IO (NIO)

With NIO und NIO.2 there exist new classes for file management and input and output

Packages

- `java.nio.files` Path and Files for file management
- `java.nio.channel` Channels for buffered and asynchronous input and output
- `java.nio.charset` Charset for encoding in Unicode

NIO

Path and Files

File Walk and WatchService

Channels and Buffers

Non-Blocking Channel Operations

Asynchronous Channels

Miscellaneous

PATH (1/2)

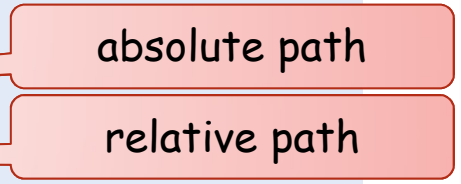
`Path`

- is a class representing a file path (which might not exist)
- provides methods for manipulating file paths

`Paths` has static methods for creating Path objects

```
private static final String SRC = "C:\\Users\\hp\\Java\\src";

Path srcPath = Paths.get(SRC);
Path nioRelPath = Paths.get("nio");
Path javaFilePath = Paths.get(SRC, "nio", "PathDemo.java");
```



`Path` methods

- `resolve`: new path from this and argument path

```
Path nioAbsPath = srcPath.resolve(nioRelPath);
```

- `relativize`: relative path to get from this to other path

```
Path javaRelPath = srcPath.relativize(javaFilePath);
```

- `normalize`: eliminates redundant path parts

```
Path homePath = Paths.get("C:\\Users\\hp\\..\\..\\Users\\hp");
Path homePathNorm = homePath.normalize();
```

PATH (2/2)

- Access to parts of a path
  - by iterator

```
for (Path p : javaFilePath) {
    println(p);
}
```

```
C:\
Users
hp
Java
nio
PathDemo.java
```

  - or by indexed access operation

```
javaFilePath.get(0));
```

```
C:\
```

  - last part (= file name or dir name)

```
println(javaFilePath.getFileName());
```

```
PathDemo.java
```

- Accessing properties

```
println(thisFilePath.endsWith("PathDemo.java"));
println(thisFilePath.startsWith("C:\\"));
```

```
true
true
```

- and others …

FILES (1/3)

`Files` with static methods for accessing file system and manipulating files

- Testing if file or directory exists
- Creating files

```
if (! Files.exists(javaFilePath)) {
    Files.createFile(javaFilePath);
}

if (! Files.exists(nioPath)) {
    Files.createDirectory(nioPath);
}
```

- Copying, moving, deleting files and dirs

multiple options as varargs !

```
Files.copy(javaFilePath, javaCopyPath,
        StandardCopyOption.COPY_ATTRIBUTES, StandardCopyOption.REPLACE_EXISTING);
```

```
Files.move(javaFilePath, javaCopyPath, StandardCopyOption.REPLACE_EXISTING);
```

```
Files.delete(javaFilePath);
Files.deleteIfExists(javaFilePath);
```

FILES (2/3)

Reading and writing files

- lines

```
Stream<String> ls = Files.lines(javaFilePath);
ls.forEach(line -> {
    println(line);
});
```

Stream access is lazy!

- realdAllLines

```
List<String> lines = Files.readAllLines(javaFilePath);
for (String line : lines) {
    println(line);
}
```

eager!

Getting Input/Outputstreams

```
BufferedReader r = Files.newBufferedReader(javaFilePath);
BufferedWriter w = Files.newBufferedWriter(javaFilePath);
```

FILES (3/3)

- Accessing attributes

```
FileTime lastModified = Files.getLastModifiedTime(javaFilePath);
println(lastModified);
```

```
2016-04-10T09:11:23.131982Z
```

```
long size = (long)Files.getAttribute(javaFilePath, "size");
println(size);
```

```
FileTime now =
    FileTime.fromMillis(System.currentTimeMillis());

Files.setLastModifiedTime(javaFilePath, now);
```

BasicFileAttributeView Attribute

Name	Type
"lastModifiedTime"	FileTime
"lastAccessTime"	FileTime
"creationTime"	FileTime
"size"	Long
"isRegularFile"	Boolean
"isDirectory"	Boolean
"isSymbolicLink"	Boolean
"isOther"	Boolean
"fileKey"	Object

Accessing all attributes !

```
BasicFileAttributeView attrs =
            Files.getFileAttributeView(javaFilePath, BasicFileAttributeView.class);

attrs.setTimes(now, now, now);
```