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)

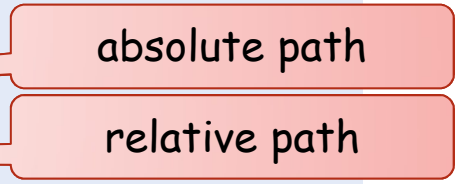
`Path`

- 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);
});
ls.close();
```

Stream access is lazy!

Don't forget to close stream!

- 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);
```

NIO

Path and Files

File Walk and WatchService

Channels and Buffers

Non-Blocking Channel Operations

Asynchronous Channels

Miscellaneous

Iterating Directories and Files

`walk`: Stream of Path objects of a directory tree

```
Path root = Paths.get(SRC);
```

```
try (Stream<Path> paths = Files.walk(root, 3)) {

  paths.forEach(p -> {
    println(p.toString());
  });

}
```

Create Stream as closable resource !

Iterate paths lazily !

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

`PathMatcher`: Finding files using glob pattern

```
PathMatcher javaMatcher = FileSystems.getDefault().getPathMatcher("glob:**.java");
if (javaMatcher.matches(javaFilePath)) println("It‘s a Java file");
```

```
PathMatcher javaMatcher = FileSystems.getDefault().getPathMatcher("glob:**.java");
try (Stream<Path> paths = Files.walk(root)) {
  paths.forEach(p -> {
    if (javaMatcher.matches(p)) {
      println("Java file " + p.toString());
    }
  });
}
```

WALK OVER DIRECTORY TREE

`walkFileTree`: Visitor over directory tree

```
public interface FileVisitor<T> {
    FileVisitResult preVisitDirectory(T dir, BasicFileAttributes attrs) throws IOExce…
    FileVisitResult visitFile(T file, BasicFileAttributes attrs) throws IOException;
    FileVisitResult visitFileFailed(T file, IOException exc) throws IOException;
    FileVisitResult postVisitDirectory(T dir, IOException exc) throws IOException;
}
```

```
Files.walkFileTree(root, new FileVisitor<Path>() {
    @Override
    public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attrs) throws IOException {
        indent = "    " + indent;
        printfln(indent + "START %s ------", dir.toString());
        return FileVisitResult.CONTINUE;
    }
    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs) throws IOException {
        printfln(indent + "FILE %s", file.toString());
        return FileVisitResult.CONTINUE;
    }
    @Override
    public FileVisitResult visitFileFailed(Path file, IOException exc) throws IOException {
        printfln(indent + "++ FAILED  %s", file.toString());
        return FileVisitResult.CONTINUE;
    }
    @Override
    public FileVisitResult postVisitDirectory(Path dir, IOException exc) throws IOException {
        indent = indent.substring(3);
        printfln(indent + "END %s --------", dir.toString());
        return FileVisitResult.CONTINUE;
    }
});
```

```
START C:\Users\hp\greptest ----------------------------------
   START C:\Users\hp\greptest\ff ----------------------------------
      START C:\Users\hp\greptest\ff\ds ----------------------------------
         FILE C:\Users\hp\greptest\ff\ds\File11.txt
         FILE C:\Users\hp\greptest\ff\ds\File12.txt
…
```

WATCHSERVICE (1/3)

WatchService allows watching for changes in directory

```
WatchService  watchService = FileSystems.getDefault().newWatchService();

Path root = Paths.get(SRC);

try (Stream<Path> paths = Files.walk(root)) {
  paths.forEach(p -> {
    if (Files.isDirectory(p)) {
        try {
          WatchKey k = p.register(watchService, ENTRY_CREATE, ENTRY_DELETE, ENTRY_MODIFY);
        } catch (Exception e) {
        }
      }
  });
}
...
```

Create WatchService for FileSystem !

Register Directory at WatchService

WATCHSERVICE (2/3)

Watching for changes in directory

```
watcherThread = new Thread(() -> {
  WatchKey key = null;
  while (!stopWatcher) {
    try {
      key = watchService.take();
      for (WatchEvent<?> evt : key.pollEvents()) {
        WatchEvent<Path> pevt = (WatchEvent<Path>) evt;
        Path relPath = pevt.context();
        Path dir = (Path) key.watchable();
        Path absPath = dir.resolve(relPath);
        if (pevt.kind() == ENTRY_CREATE) {
          if (Files.isDirectory(absPath)) {
            try {
              WatchKey k = absPath.register(watchService, ENTRY_CREATE, ENTRY_DELETE, ENTRY_MODIFY);
            } catch (Exception e) { ... }
            } catch (InterruptedException e) { ... }
          }
        } else if (pevt.kind() == ENTRY_MODIFY) {
        } else if (pevt.kind() == ENTRY_DELETE) {
        }
         ...
      }
    } finally {
       if (key != null) key.reset();
    }
  }
});
watcherThread.start();
```

in new thread

Access WatchKey ➔ blocking!

iterating events

retrieve elements of events

Register new directories with WatchService

distinguish event kinds

Reset key after handling event; otherwise no more events are signaled!

WatchService (3/3)

Summary of approach

- Create global FileWatcher for file system

```
WatchService watcher = FileSystems.getDefault().newWatchService();
```

- Register directory

```
WacthKey k = dirPath.register(watcher, ENTRY_CREATE, ENTRY_DELETE, ENTRY_MODIFY);
```

- Wait for and access changes ➔ WatchKey

```
WatchKey k = watcher.take();
```

- WatchKey allows accessing change events
  - WatchKey contains paths and event type

```
for (WatchEvent<?> evt: k.pollEvents()) {
    WatchEvent<Path> pevt = (WatchEvent<Path>) evt;
    Path relPath = pevt.context();
```

  - directory where event occurred

```
    Path dir = (Path)k.watchable();
```

- Reset WatchKey, if directory should be further observed

```
k.reset();
```

or cancel if not

```
k.cancel();
```