
NIO.2

CONTENT

Java.nio API (non-blocking input/output)

Path

Working with paths and files

File attributes

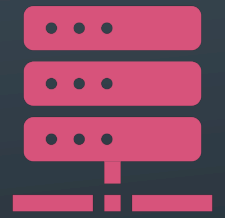
Views

New stream methods

NIO.2 VS IO

Non blocking (NIO) and blocking (IO)

NIO.2 has much more options with files and directories than IO



IO is stream oriented, NIO is buffer oriented

PATH

- In the java.nio package
- An interface
- May refer to file or directory
- Representation of location of file/directory in the filesystem
- Replacement for java.io.File
- Contains symbolic links: special file that points to another file/directory

CREATING PATH

- To create use the Path factory
  - Path p1 = Paths.get("relative/because/no/slash/or/drive/as/first/path.txt");
  - Path p2 = Paths.get("c:\\absolute\\windows\\path.txt");
  - Path p3 = Paths.get("/home/absolute/unix/path.txt");
  - Path p4 = Paths.get("/","home","absolute","path.txt"); //file separator added by the system, works for absolute and relative paths
  - Path p5 = Paths.get(new URI("file:///c://absolute//windows//path.txt")); //only works for absolute paths, also external paths with http or ftp
- Java 11: Path.of

- Question: why is Path an interface?

REWRITE USING PATH

01

Use Java to:

02

Create a new folder “example”

03

Add a file “log.txt” to this folder

04

Add a folder “messages” to the folder example

05

Create a txt file for messages in this folder

06

Print the pathname of the messages.txt to the console

07

Verify both files exist

08

Verify whether example/nother e.txt exists

ACCESSING FILESYSTEM OBJECT

FileSystem is an abstract class

Paths.get() is actually FileSystems.getDefault( ).getPath()

We can access remote file systems with the FileSystems class:

```
FileSystem fs = FileSystems.getFileSystem(new URI("http://www.someuri.com") );
Path p = fs.getPath("somefile.txt");
```

Changing a File to a Path:

```
File f = new File("bla/bla.txt");
Path p = f.toPath();
```

And the other way around:

```
Path p = Paths.get("bla/bla.txt");
File f = p.toFile();
```

INTERACTING WITH PATHS AND FILES

- Optional arguments
  - NOFOLLOW_LINKS > symbolic links will not be traversed
  - FOLLOW_LINKS > symbolic links will be traversed
  - COPY_ATTRIBUTES > metadata will be copied
  - REPLACE_EXISTING > if target already exists, it will be replaced
  - ATOMIC_MOVE > operations are performed in an atomic matter if the file system supports it, e.g. avoid files to be written partially

PATH METHODS

Viewing the Path:

- toString()
- getNameCount()
- getName(int i)

Accessing Path component:

- getFileName()
- getParent()
- getRoot()

Checking Path type:

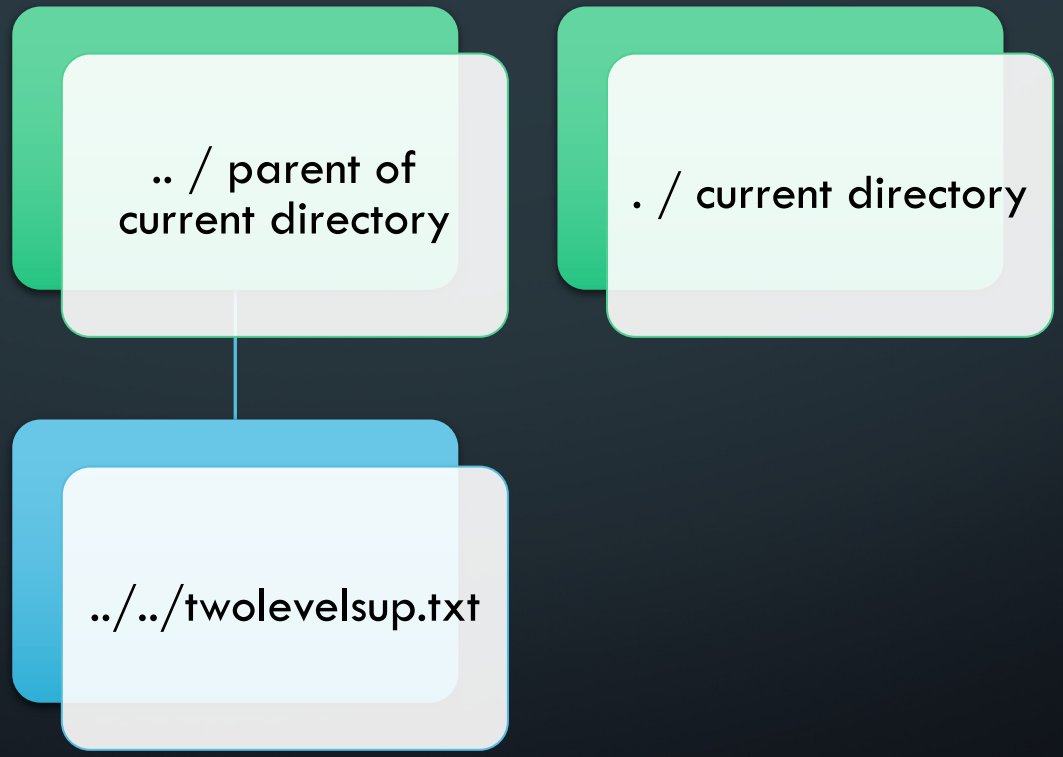
- isAbsolute()
- toAbsolutePath()

subpath(int, int) > create a new path from a path, e.g. :

Path p = Paths.get("/some/path/text.txt");

p.subpath(1,2); // path

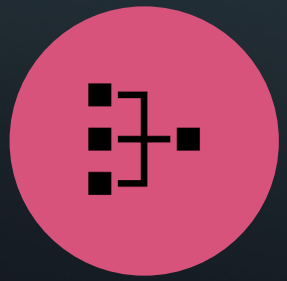
PATH SYMBOLS



SOME MORE FUN METHODS

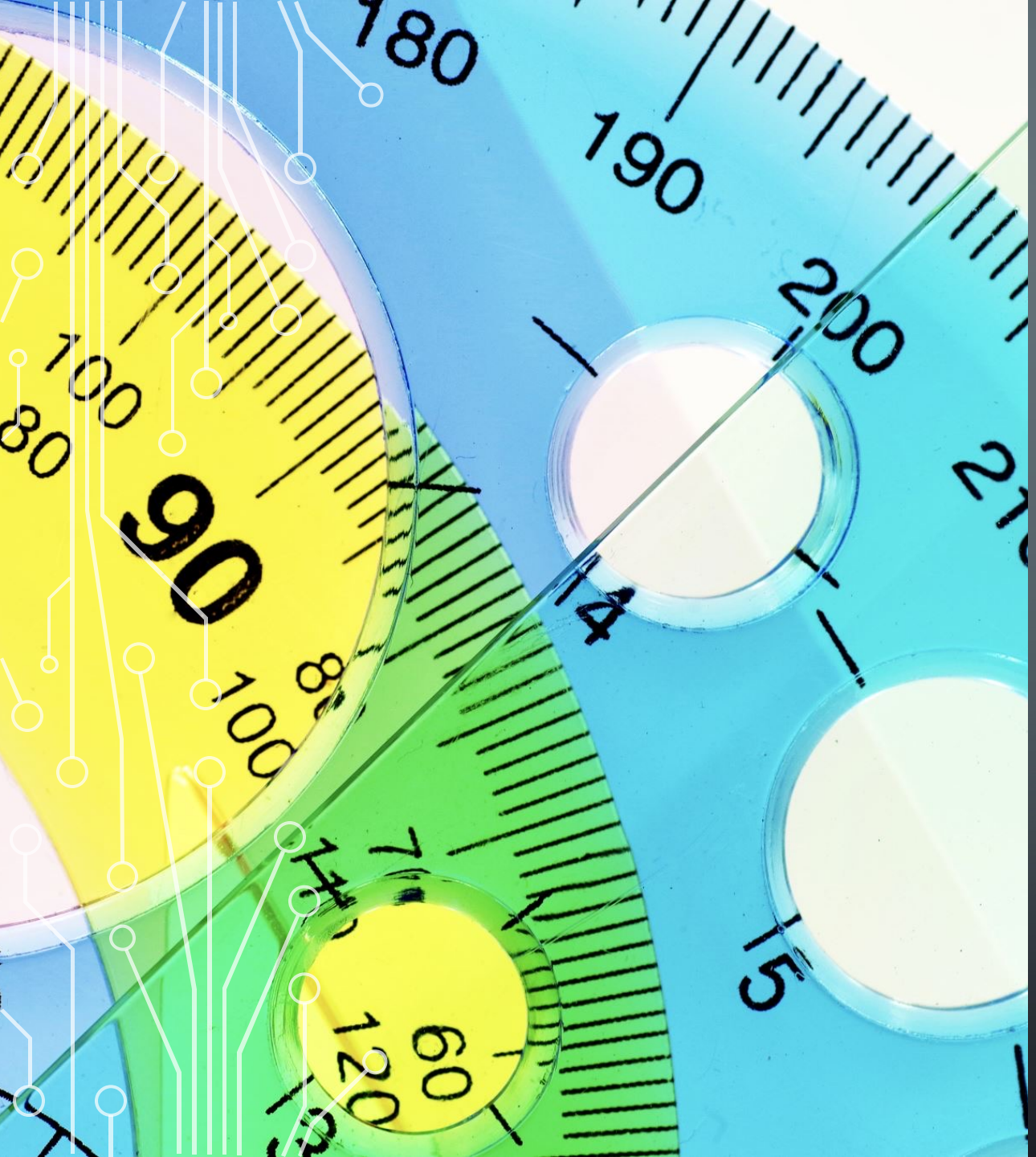
p1.relativize(p2) > creates a relative path from Path p1 to p2, doesn't clean up path symbols that are redundant

p1.resolve(p2) > joins p1 to p2, path concat, doesn't clean up path symbols



.normalize() > cleans up redundant path symbols

.toRealPath(path) > IOException if path doesn't exist, reference to path when it does exist



CONFUSE YOUR COLLEAGUES

- 10 minutes
- Use the methods of the previous slides to create a hard to understand question with paths, e.g.:
  - What is the count?
  - What is the subpath after doing some other magic with path methods?
  - What is the relative path from p1 to p2 who both use path symbols?
  - …?
- Afterwards, we are going to present and answer the questions
- Bonus: be worse than Oracle

INTERACTING WITH FILES

- java.nio.file.Files > has lots of methods to interact with the actual file :
  - exists(pathname) > checks if path exists, returns false if it doesn't
  - isSameFile(pathname, anotherpathname) > checks if they point to same file
  - createDirectory(path) > mkdir, creates a directory
  - createDirectories(path) > mkdirs, creates directory and all parent directories if these don't exist yet
  - copy(path1, path2) > copies from 1 to 2, doesn't copy directory content
  - move(path1, path2) > moves file or directory from 1 to 2
  - delete(path) > deletes path and throws exception if it doesn't exist
  - deleteIfExists(path) > deletes path and returns false if it doesn't exist
  - .newBufferedReader(path, charset) > reads file using java.io.BufferedReader
  - .readAllLines(path) > reads all the lines of a file at once

EXERCISE ON COPYING FILES

- Revisit the first exercise.
- Add example.txt and anotherExample.txt to the example folder
- Create a new folder on the same level as folder example and call it “destination”
- Copy the content and directory of example to destination

FILE ATTRIBUTES

- File attributes: metadata of files and directories
- On the Files class there are methods to get information about file attributes:
  - isDirectory(path)
  - isRegularFile(path) > not a directory, but actual content
  - isSymbolicLink(path)
  - isHidden(path)
  - isReadable(path)
  - isExecutable(path)
  - size(path) > size of file in bytes
  - getLastModifiedTime(path) and setLastModifiedTime(path, FileTime)
  - getOwner() and setOwner(UserPrincipal)

VIEWS

01

View: group of related attributes that can be accessed with a single method

02

readAttributes() > read-only view of attributes (BasicFileAttributes)

03

getFileAttributeView() > gives attributes view and can be directly modified from there (BasicFileAttributeView)

04

All attributes extend from BasicFileAttributes

EXERCISE: MODIFYING ATTRIBUTES

Create a new file: attributesCanBeChanged.txt

Get the attributes

Change some attributes

EXERCISE: LET'S GO FOR A WALK

- Create a directory with subdirectories that represents a nice forest, or perhaps an enchanted forest. Hide a treasure.txt with some treasure content in the forest
- Make sure your forest consists of at least 25 directories
- Walk through the forest

- Let's get lost: create a symbolic link that creates a circular path in the forest, walk in a circle and see what happens
- List the directory contents to create a map of the forest
- Find the treasure and print its content

CASE

- Continue working on the IO exercise, use your new knowledge
- Create a separate folder for every unique person (unique name is fine) in which you store all movements of that person
- Every shipping of goods should be stored in a folder of the owner