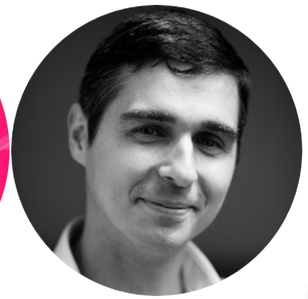
Writing Succinct I/O Code



Andrejs Doronins
Software Developer in Test

Overview

Covering:

- Local file system

Not covering:

- Remote connections, HTTP, etc.

High-level overview

Path separators, current directory

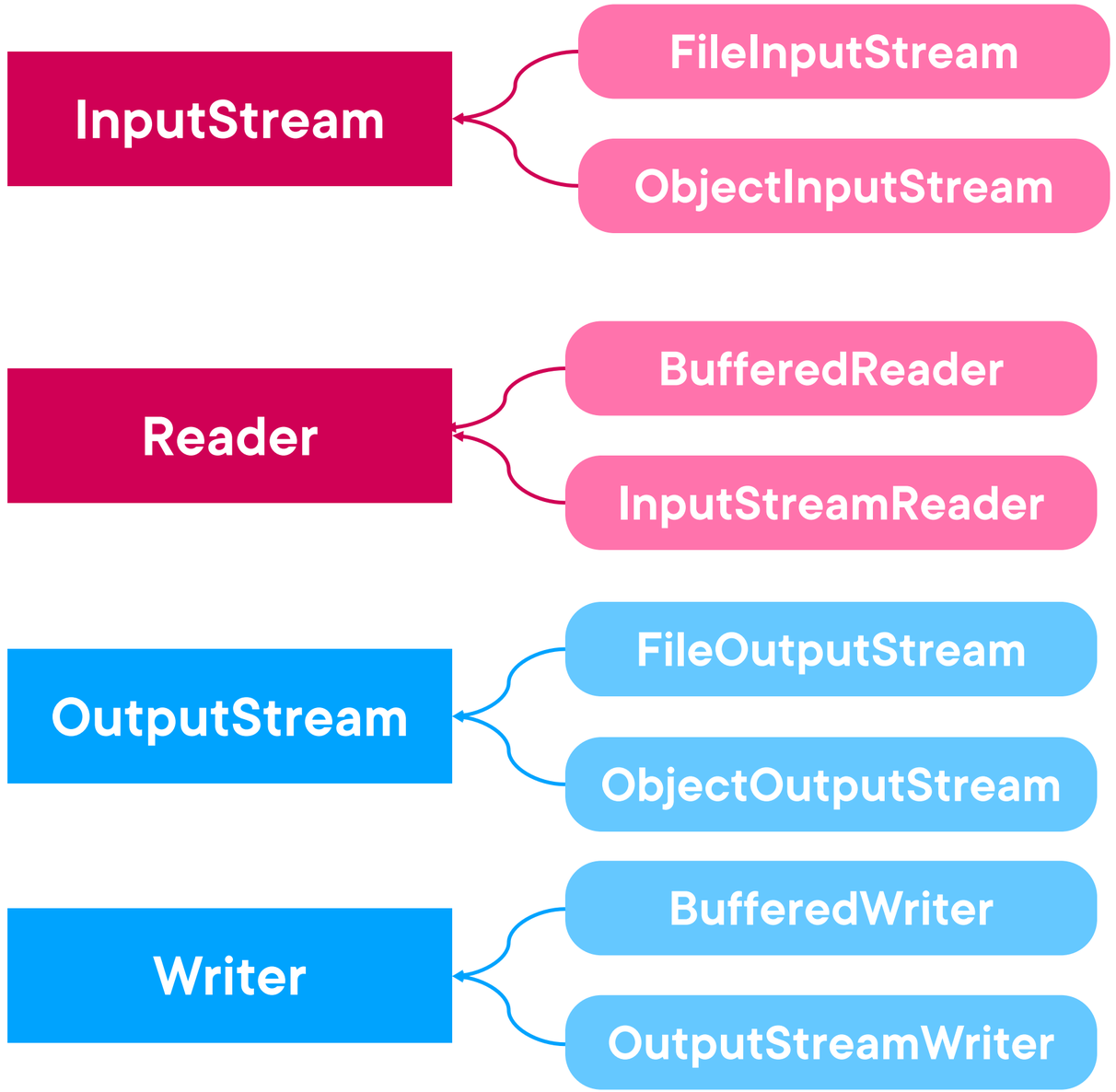
Reading and writing to files

Operations with directories

java.io

└── File.java

```
boolean delete()

boolean exists()

// etc.
```



File Class

Shortcomings:

- No copy() method
- Methods return “boolean”

```
try {
    String strCurrentLine;
     objReader = new BufferedReader(new FileReader(path));
    while ((strCurrentLine = objReader.readLine()) != null)
{                   fileContent.append(strCurrentLine); }
    } catch (IOException e) {
        // handle
     } finally {
            if (objReader != null) {
                try { objReader.close(); }
                 catch (IOException e) { e.printStackTrace();
              }    }   }
```

Java I/O Packages

java.io

File

Streams

Writers and Readers

java.nio

Channels and Buffers

Non-blocking IO

java.nio.file (NIO2)

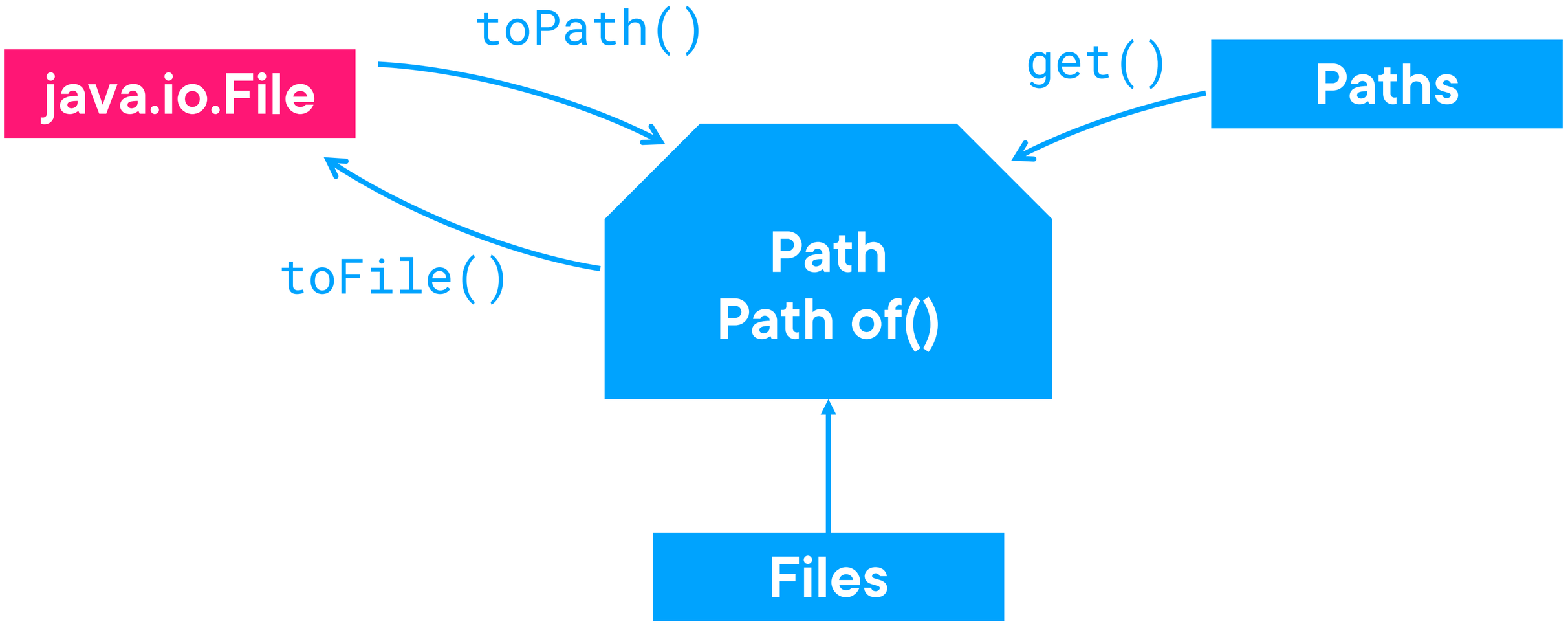
Better API for working with Files and Directories

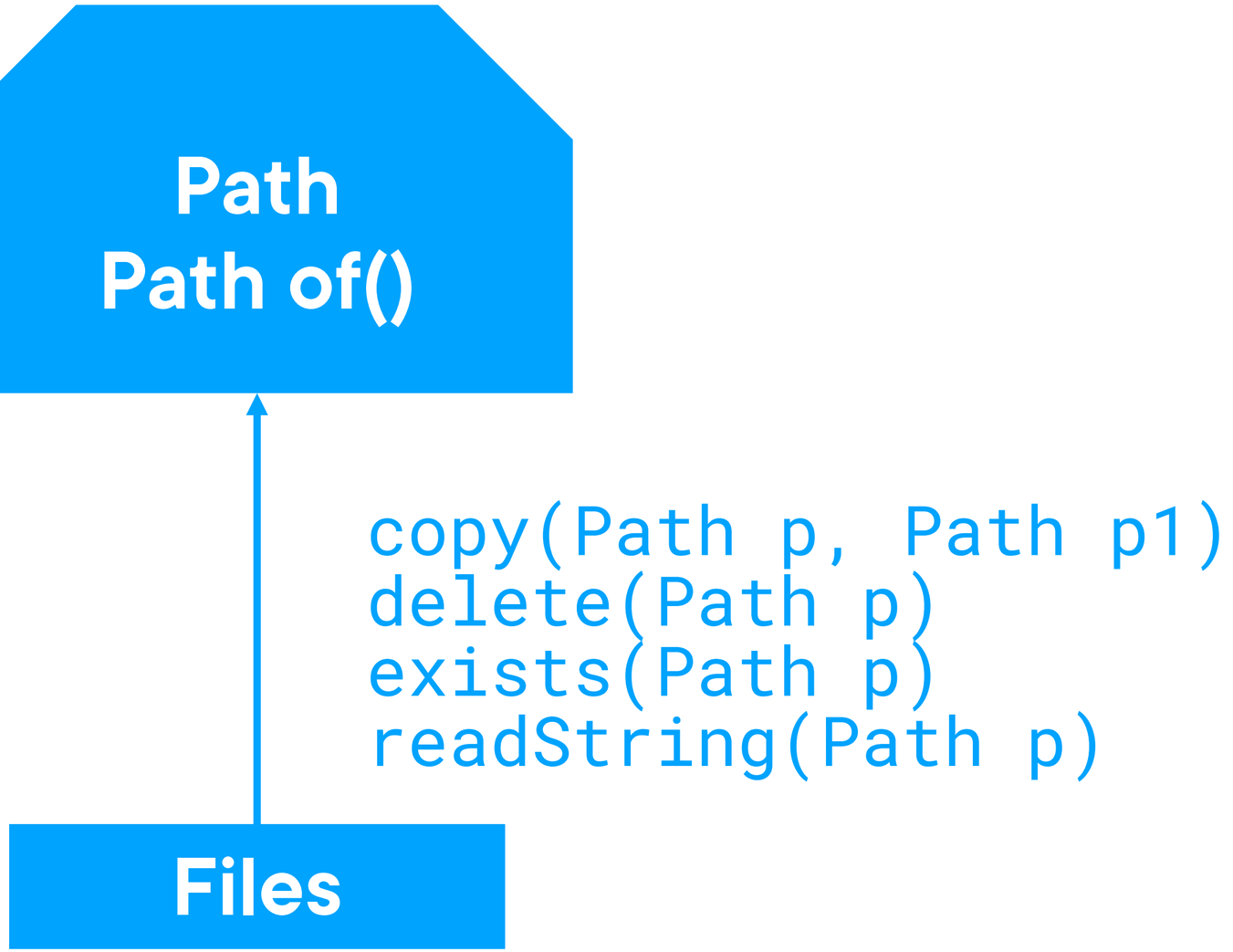
OCP Study Guide, Boyarski and Selikoff

“At its core, NIO2 is a replacement for the legacy java.io.File class. The goal of the API is to provide a more intuitive, more feature-rich API for working with files and directories.”

OCP Study Guide, Boyarski and Selikoff

“So the preferred approach [...] with newer software applications is to use the NIO.2 package. ”


toPath()
java.io.File
get()
Paths
toFile()
Path
Path of()
Files


Path
Path of()
copy(Path p, Path p1)
delete(Path p)
exists(Path p)
readString(Path p)
Files

```
Files.readString(hugeLog);
```

performance

```
try ( BufferedReader br = new BufferedReader(
new FileReader(new File("file.txt"), StandardCharsets.UTF_8))) {


    String line;
    while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
    }
```

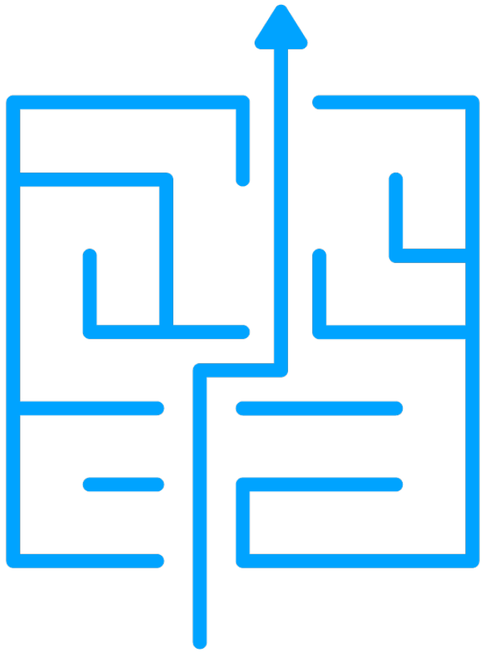
Since Java 11

```
try ( BufferedReader br = Files.newBufferedReader(
            Path.of("file.txt"), StandardCharsets.UTF_8)) {

    String line;
    while ((line = br.readLine()) != null) {
            System.out.println(line);
            }
    }
```

```
Stream<String> lines = Files.lines(Path.of("file.txt"));


lines
    .filter()
    .toList();
```

Read a big file?
You need about 5 lines in modern Java.

Reading from and Writing to Binary Files



Somewhat complicated

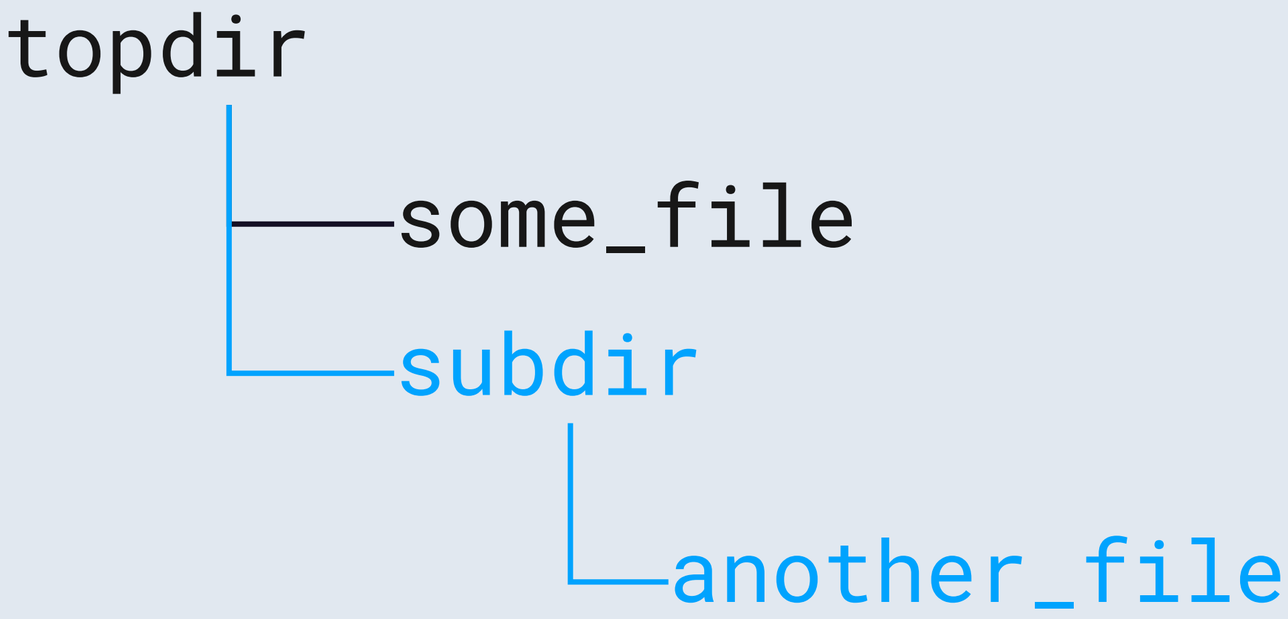
Involves java.io

Course: Java Fundamentals: Input/Output

- Reading and Writing Bytes

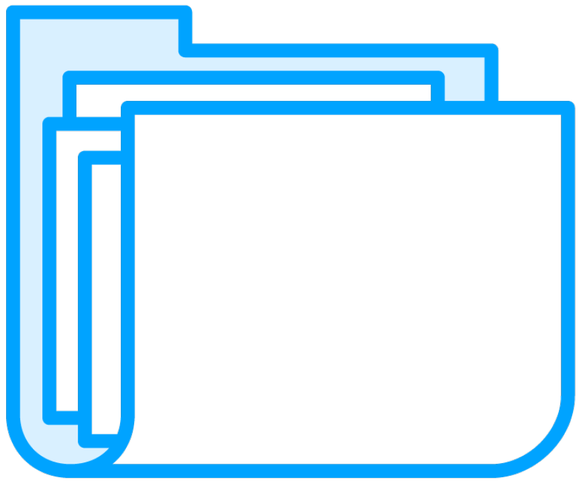
Following Symbolic Links uncontrollably and deleting is dangerous!

```
/**
* [...]
* If the file is a symbolic link then the symbolic link itself,
* not the final target of the link, is deleted.
* [...]
* /
public static void delete(Path path) throws IOException {
    // ...
}
```




```
Files.walkFileTree(Path dir, FileVisitor v)
```

FileVisitor



Visitor Pattern

Iterate over (visit) a node structure (a tree)

Course: Java Design Patterns

```
public interface FileVisitor<T> {


FileVisitResult preVisitDirectory(...)

FileVisitResult visitFile(...)

FileVisitResult visitFileFailed(...)

FileVisitResult postVisitDirectory(...)


}
```

```
new SimpleFileVisitor<Path>() {
    visitFile() { “do this with every file” }
}

Files.walkFileTree(dir, ____ );
```

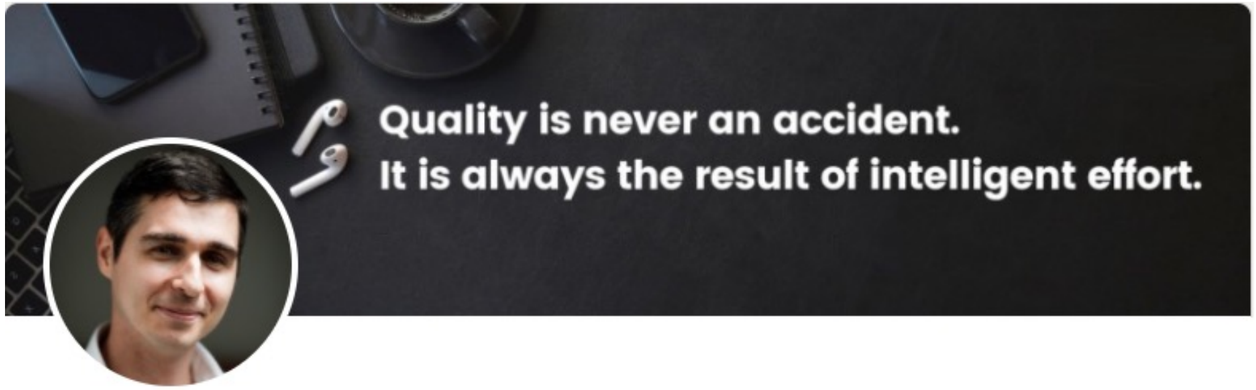
Further Study Summary

Java Fundamentals: Input/Output

Java Fundamentals: NIO and NIO2

Working with Files in Java Using the Java NIO API

Rating



https://www.linkedin.com/in/andrejs-doronins-195125149/

Thank you!

(Happy Coding)