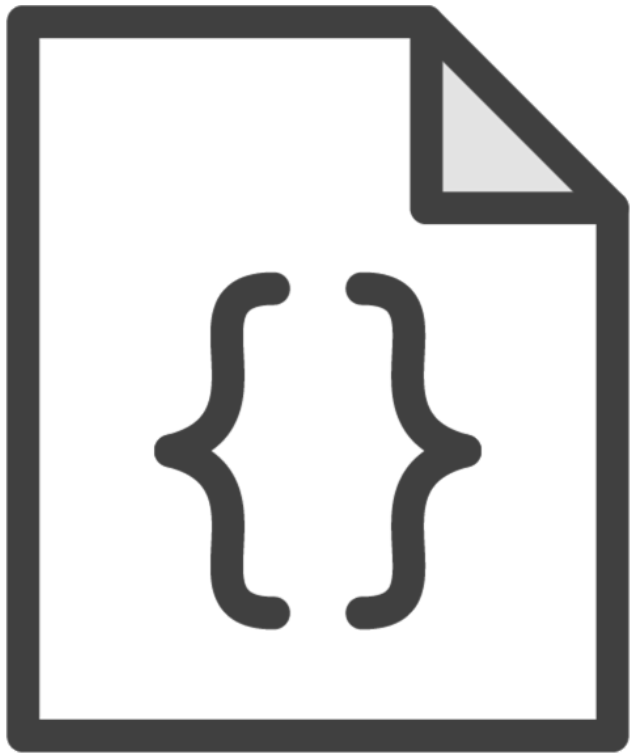
Practicing try-with-resources

Andrejs Doronins



Automated closing of resources

No more “finally {}” block (in most cases)

Overview

try-with-resources (TWR):

- Syntax
- Bare minimum to make it work

Tricky aspects of TWR

- Is catch optional?
- Declaring resources
- Variable scope

AutoCloseable Interface

closeable resources

required semicolon between resource declarations

optional

```
try (Resource1 ; Resource2;) {
    // code
}
```

Resources are closed at this point

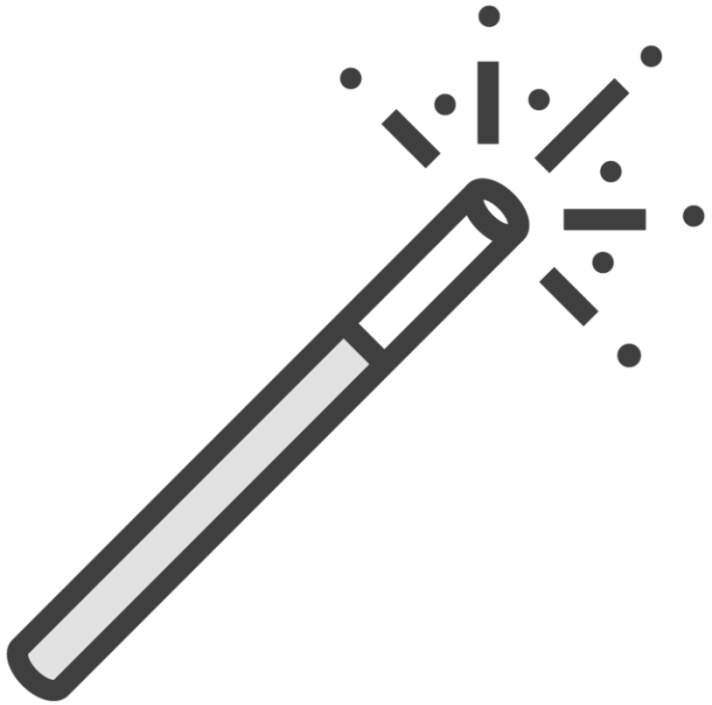
an implicit finally {} is run

The bare minimum

Optional `catch (Exception e) { }`

Optional `finally { }`

(!) If the resource doesn't throw



The tricky bits:

- Declaring resources
- Variable scope
- Order of resource closing

App.java

```
FileInputStream in;
try (in = new FileInputStream("in.txt")) { }
```

```
try (FileInputStream in = new FileInputStream("in.txt")) { }
```

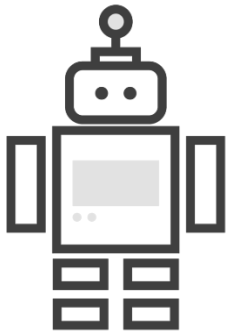
```
try (var in = new FileInputStream("in.txt")) { }
```

```
try (FileInputStream in = new FileInputStream("file.txt")) {
        in.read();
}

catch (Exception e) {
        in.read();  // does not compile!
}

finally {
        in.close(); // does not compile!
}
```

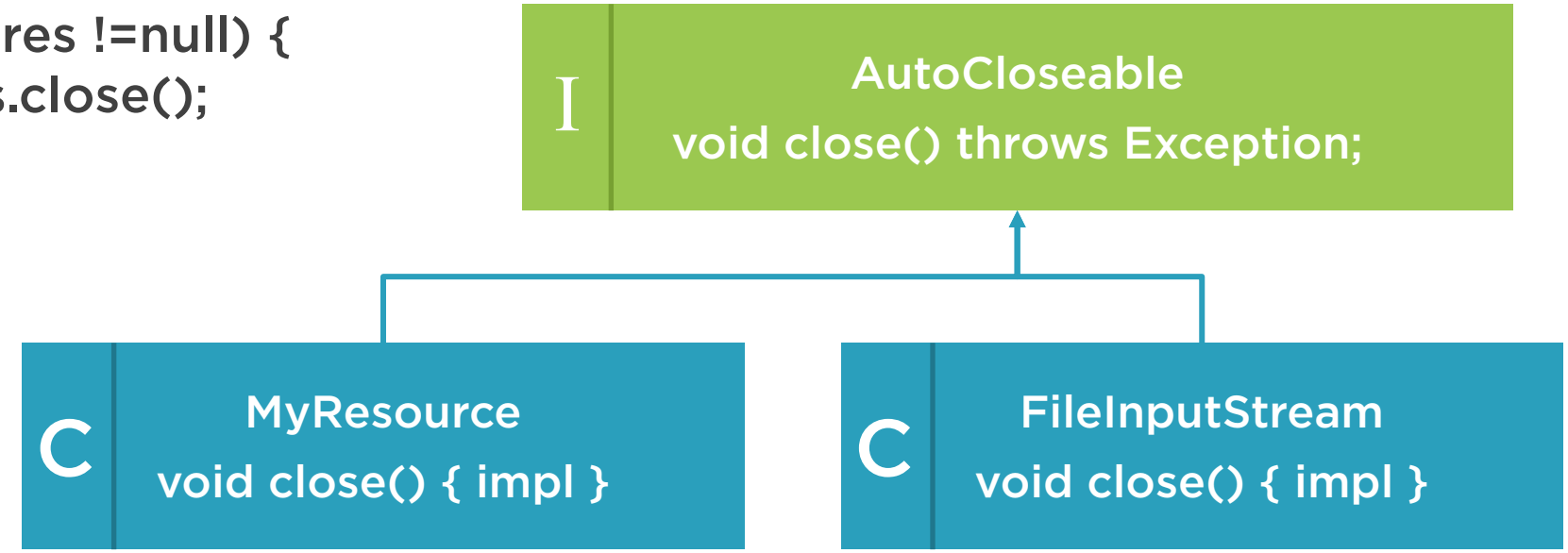
AutoCloseable is mandatory for resources in try-with-resources



I'll do it for you, but...

Java

```
finally {
        if (res !=null) {
        res.close();
        }
}
```



Summary

- Basic syntax: try(R1 r; R2 r){ }
- Catch is (sometimes) optional
- Resources must be declared with type or var
- Resource scope is limited to try

Question 1: will this compile?

```
BufferedReader br = null;
try {
    br = new BufferedReader(new FileReader("file.txt"));
} catch (Exception ex) {

} finally {
    br.close();
}
```

Question 2: will this compile?

```
try (var in = new FileInputStream("in.txt")) {
        // read the input
} catch (FileNotFoundException | IOException e) {
       // handle
}
```

Question 3: will this compile?

```
try (var in = new FileInputStream("in.txt"),
     var out = new FileOutputStream("out.txt")) {
} catch (IllegalArgumentException | IOException e) {


}
```

Answers

All examples fail to compile

Question 1:

- "br.close();" method throws an IOException that is left unhandled

Answer 2:

- Non-disjoint Exceptions in the multi-catch (FileNotFoundException extends IOException)

Answer 3:

- resources separated by "," instead of ";"

Understanding Exception Types