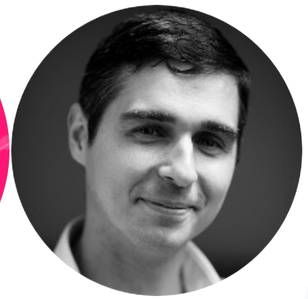
Handling Exceptions Gracefully



Andrejs Doronins
Software Developer in Test

Good Exception Handling

```
File file = new File(“file.txt”);
try (var inputStream = new FileInputStream(file)) {
        // read
} catch (IOException e) {
    log.error(e);
}
```

Good Exception Handling?

```
File file = new File(“file.txt”);
FileInputStream stream;
try {
    // read
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (Throwable e) {
    // should never happen
}
```

Overview

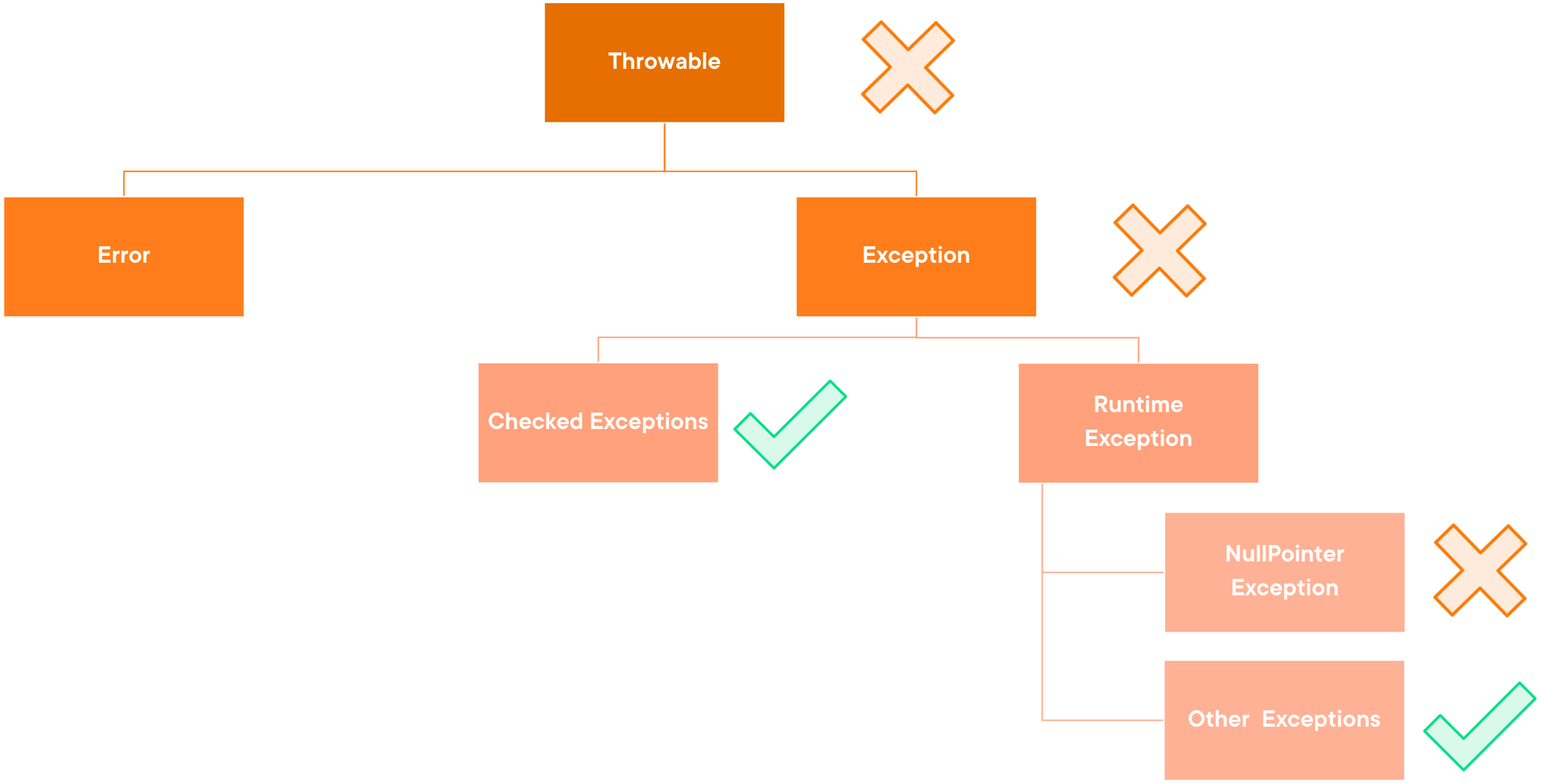
What exceptions not to catch and why

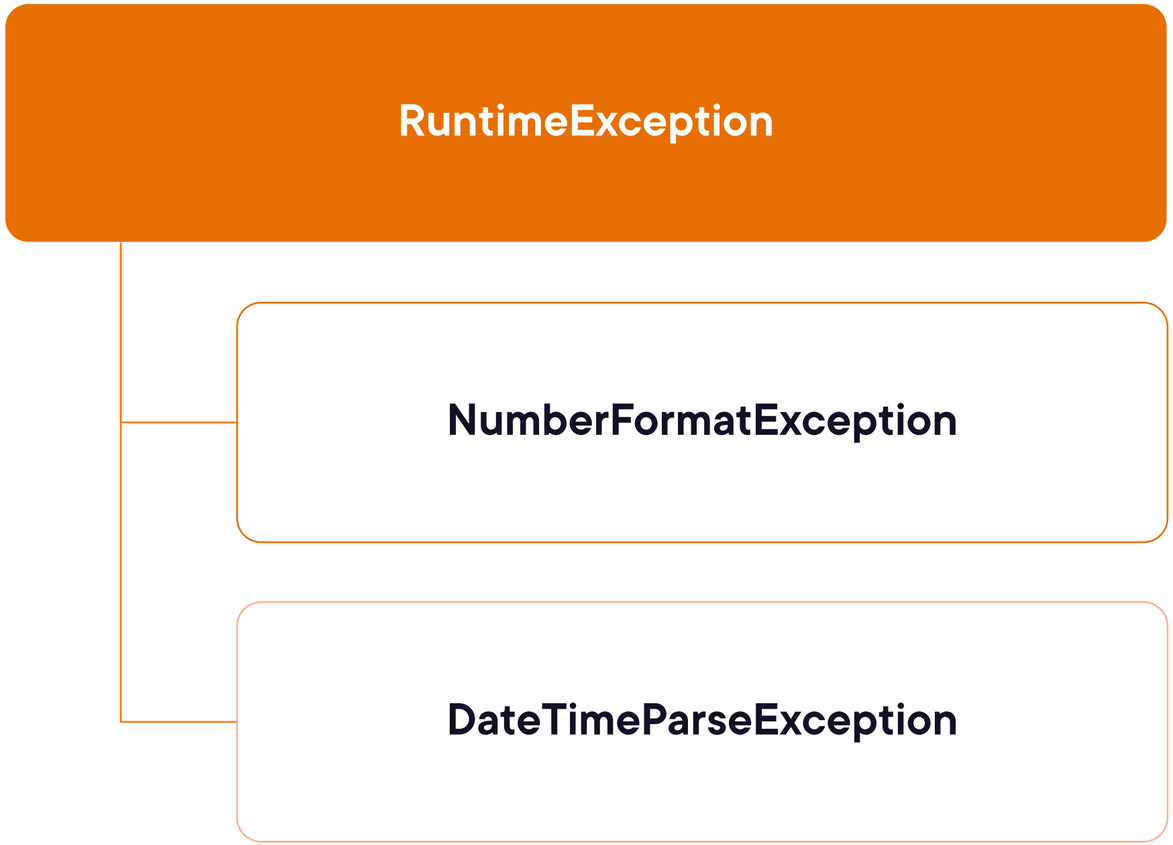
Catch block rules

Leaner exception handling code

```
try {

} catch (SpecificException e) {
    // handle
} catch (AnotherException e) {
    // handle
} catch (Throwable) {
    // handle the rest
}
```

Don’t Catch Them All



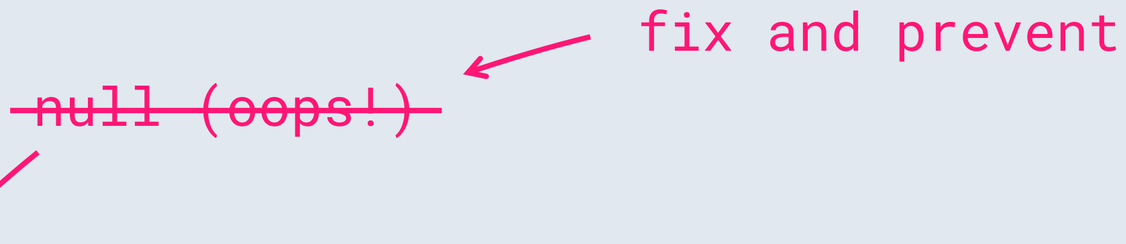

RuntimeException
NumberFormatException
DateTimeParseException

```
try {

} catch (Throwable e) {
    // try to recover
}
// inevitably fail soon after
```

```
try {

} catch (Exception e) { // a bit better

}
```

```
try {

} catch (IOException e) { // even better

}
```

```
try {

} catch (RuntimeException e) { // avoid
    // also handle unwanted exceptions
}
```




```
void search(String date) {
    try {

    } catch (NullPointerException e) { ... }
}
```

null (oops!)

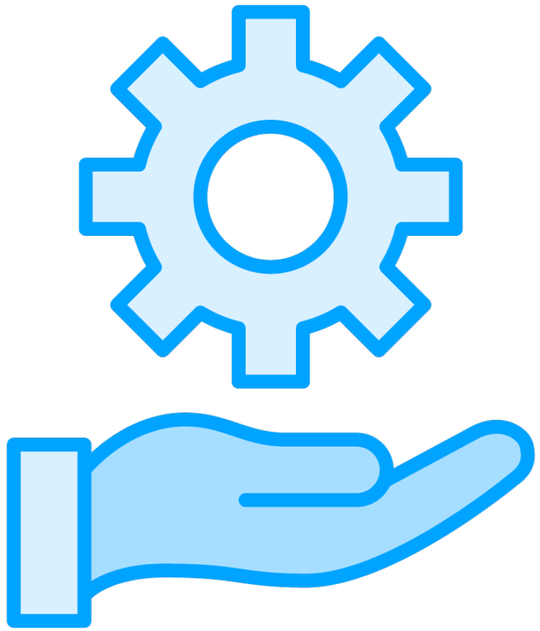
```
void search(String date) {
    if(date == null) {
        // gotcha!
    }
}
```

null

```
void search(String date) {
    try {


    } catch (NullPointerException e) {
        // hide the problem
    }
}
```

```
// accepted an input but failed to do something with it


// Integer
int parseInt(s) throws NumberFormatException {
}


// LocalDate
int parse(text, formatter) throws DateTimeParseException
{
}
```

Handling Runtime Exceptions



Set a default value and continue

Exit gracefully

- (don't use exception handling code as branching)

Rethrow

```
int parseInput(String input) {
    int number = 0;
    try {
        number = Integer.parseInt(input);
    } catch (NumberFormatException e ){
        return 1; // default starting value
    }
        return number;
}
```

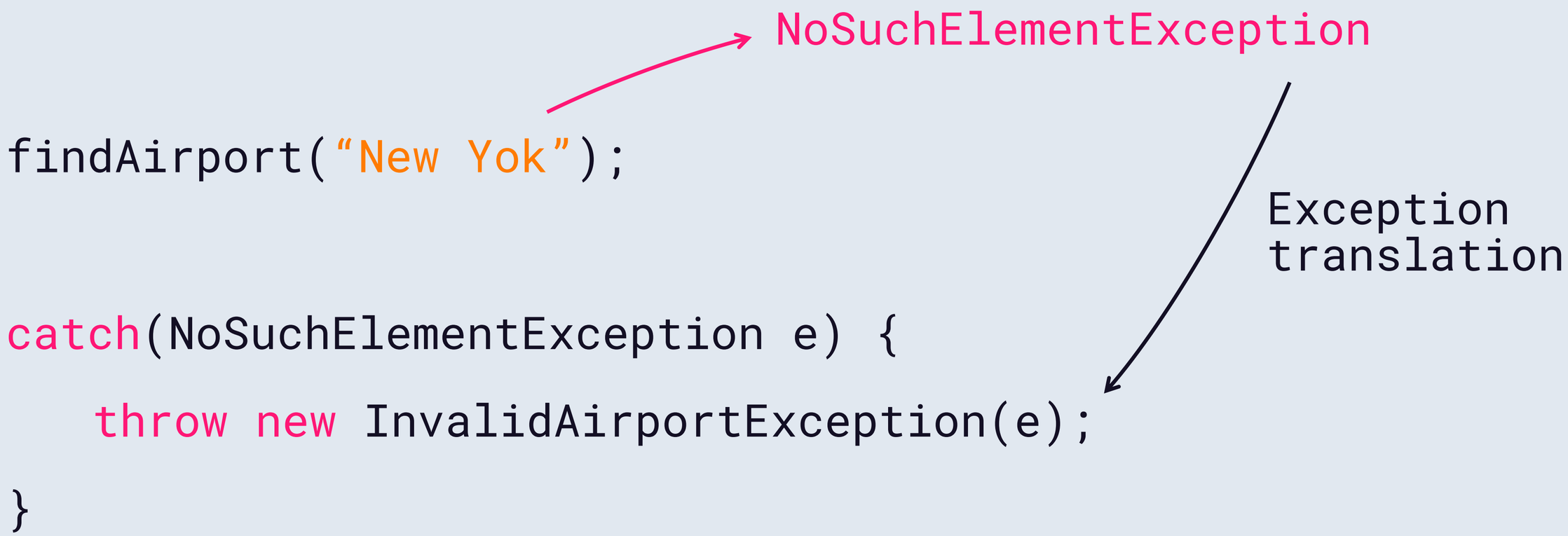
Catch Block Must Not Be Empty

```
try {
    // do things
} catch (SomeException e ){
    // should not happen
    return null;
}
```

Catch Block Must Not Be Empty

```
try {
    // do things
} catch (SomeException e ){
    logger.error(e);
    throw e;
}
```

Translate exceptions to the appropriate abstraction layer


NoSuchElementException
findAirport(“New Yok”);
Exception
translation
catch(NoSuchElementException e) {
throw new InvalidAirportException(e);
}

Pass all pertinent information into your exception message

Practical Tips

To write better exception messages:

- Write the message
- Trigger the exception

Ask yourself:

- Is this informative enough for someone else?
- What information could be missing?

Include:

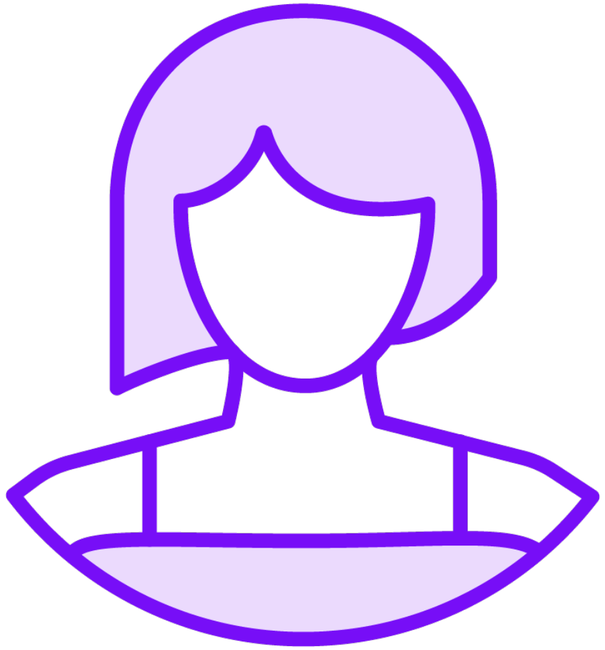
- Expected input or state
- Actual input or state

Practical Tips

Consider:

- The audience (user, programmer, sysadmin?)
- Careful not to include sensitive data

Java exception handling
is verbose

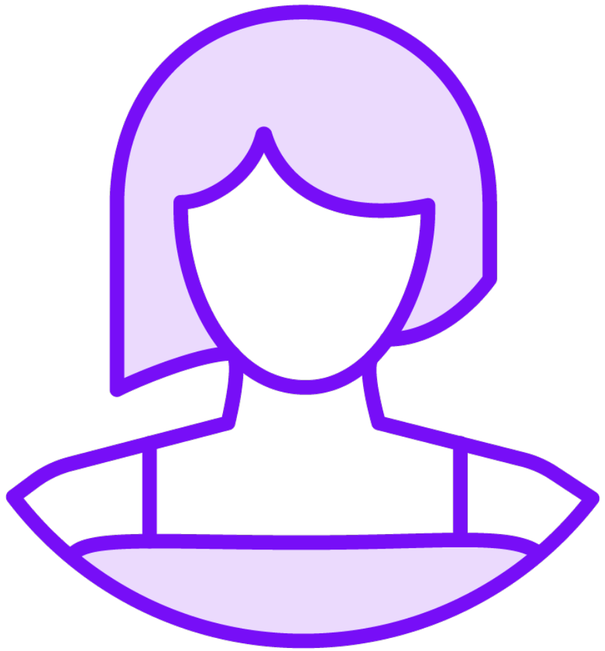


There are ways to make it shorter

```
catch (IOException e) {
  doCleanup();
}
catch (SQLException e) {
  doCleanup();
}
```

```
catch (IOException | SQLException e) {
  doCleanup();
}
```

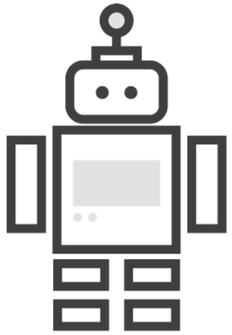
Prefer try-with-resources



Definitely!

```
try (Resource1 ; Resource2) {
    // code
} an implicit finally { } is run

catch (Exception e) {
 }
```

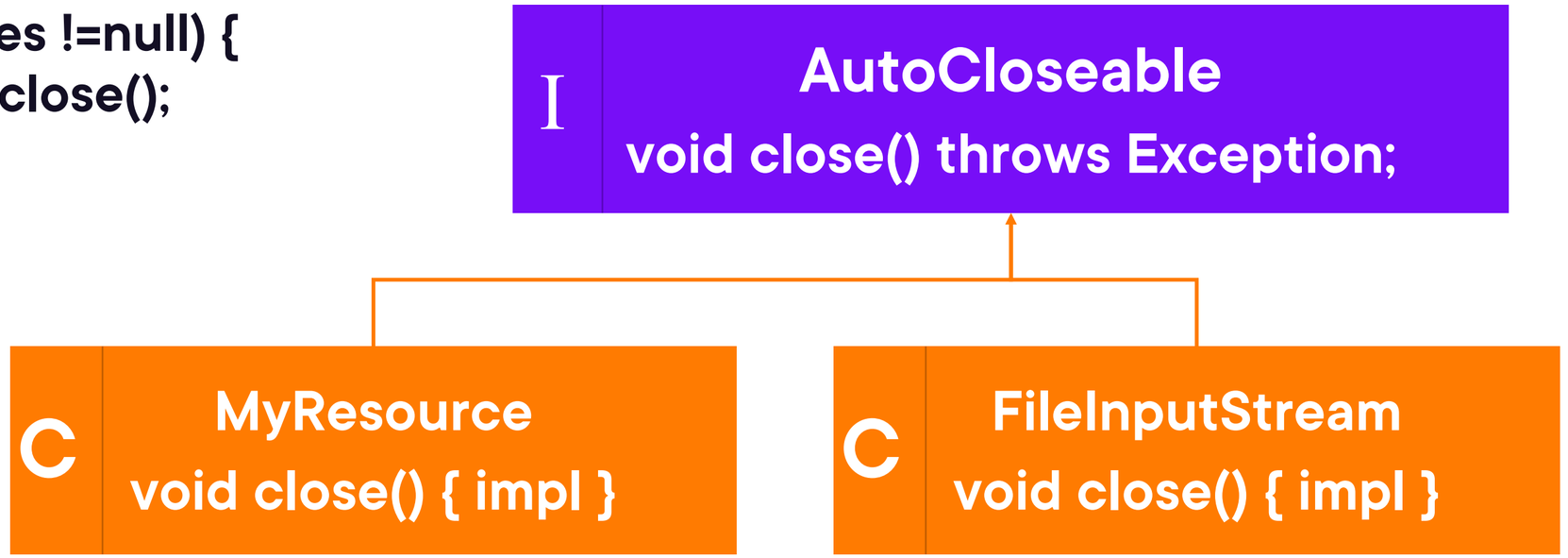
Optional `finally { }`



Java

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

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



Summary

Exception handling is about avoiding bad practices

```
try {

} catch (Throwable e) {
    // try to recover
}
```

```
try {

} catch (SpecificException e) {
    // log OR rethrow
    throw new SomeException("Actual: %s, Expected %s,");
    // or
    throw new TranslatedException("...");
}
```

```
try(resources){

} catch (IOException | SQLException e) {

}
// no need for finally
```

Up Next:

Writing Meaningful Comments Only