Throwing Exceptions

Andrejs Doronins

Overview

- How to throw
- What rethrowing means
- Throwing from the OO perspective:
  - Overriding & overloading
- Custom exceptions
- What the course didn't cover
- Wrap up

```
void setAge(int age) {
    this.age = age;
}
```

```
Person p = new Person();
p.setAge(30);
```

```
void setAge1(int age) throws IllegalArgumentException {
    this.age = age;
}
// OR
```

```
void setAge2(int age) throws IOException {
    this.age = age;
}
```

Declaring but not actually throwing!

Will this compile?

```
void setAge1(int age) throws IllegalArgumentException {
    if(age <= 0) { throw new IllegalArgumentException("…");}
    this.age = age;
}
// OR
void setAge2(int age) throws IOException {
    //check age
    if(checkSomething()) { throw new IOException ("…");}
    this.age = age;
}
```

Should I declare runtime or checked?

```java
Person p = new Person();
// compiles
p.setAge1(30);
```

```java
Person p = new Person();
// fails, unhandled exception
p.setAge2(30);
```

```java
Person p = new Person();
// compiles
p.setAge1(30);
```

```java
Person p = new Person();
try {
  p.setAge2(30);
} catch (...) { }
```

```
if(whatever) { throws new Exception(); }
```

```
void setAge() throw Exception {...}
```

```
if(whatever) { throw new Exception(); }
```

```
void setAge() throws Exception {...}
```

Runtime exceptions can occur anywhere in a program, and in a typical one they can be very numerous.

Having to add runtime exceptions in every method declaration would reduce a program's clarity.

Thus, the compiler does not require that you catch or specify runtime exceptions (although you can).

TLDR: You *can* add Runtime exceptions to the method signature, but avoid it.

```
void calculate() {
    Data d = fetchData();
    // handle data
}
Data fetchData() {
    try {
    Connection conn = openAConnection();
    } catch (IOException e) { ... }
    return  conn.queryDb("...");
 }
```

```
void calculate() {
    Data d = fetchData();
    // handle data
}
Data fetchData() throws IOException {
    Connection conn = openAConnection();
    return  conn.queryDb("...");
 }
```

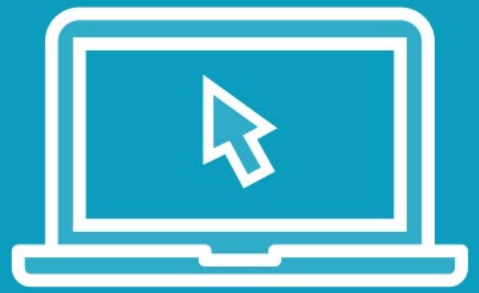
You handle it!

Let the next one do it...

```
void calculate() throws IOException {
    Data d = fetchData();
    // handle data
}
Data fetchData() throws IOException {
    Connection conn = openAConnection();
    return  conn.queryDb("...");
 }
```

You handle it!

Demo



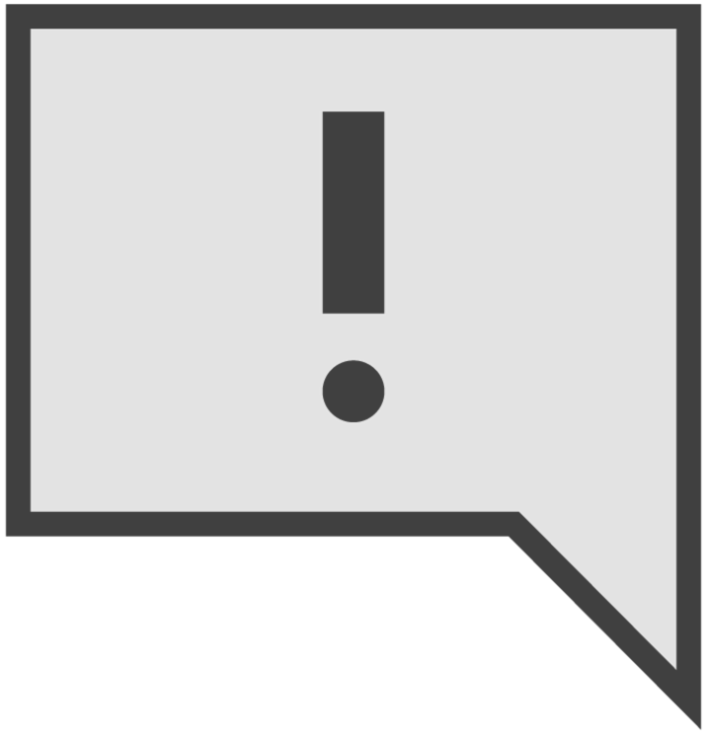
Declaring exceptions in the method signature

Exceptions in Method Signatures

Overriding

Overloading

```java
class Parent {
    void doThing() throws IOException {
    }
}

class Child extends Parent {
    @Override
    void doThing() throws Exception {
    }
}
```



When:

A class overrides a method from a super class or implements a method from an interface

Then:

It's not allowed to add new checked higher-level exceptions to the method signature

```java
class Parent {
    void doThing() { }
}

class Child extends Parent {
    @Override
    void doThing() /* no throwing of checked exceptions */ { }
}
```

```
class Parent {
    void doThing() throws IOException { }
}
class Child extends Parent {
    @Override
    void doThing() throws
            FileNotFoundException,
            IOException,
             Exception {    }
}
```

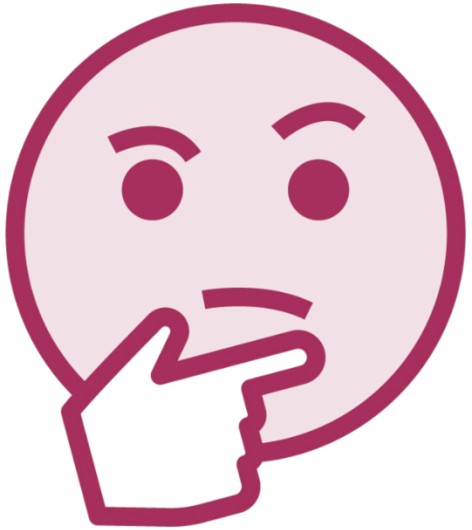
```
class SomeClass {

    void doThing() throws IOException { }

    void doThing() throws RuntimeException { }
}
```

signature

not part of the signature

Common RuntimeException Classes	Common Checked Exception Classes
ArithmeticException	IOException
IllegalArgumentException	FileNotFoundException
NullPointerException	...
...	

Need to better communicate what has gone wrong



```
class MyCustomException {
}
```

directly

```
class MyCustomException extends Exception {
}
```

indirectly

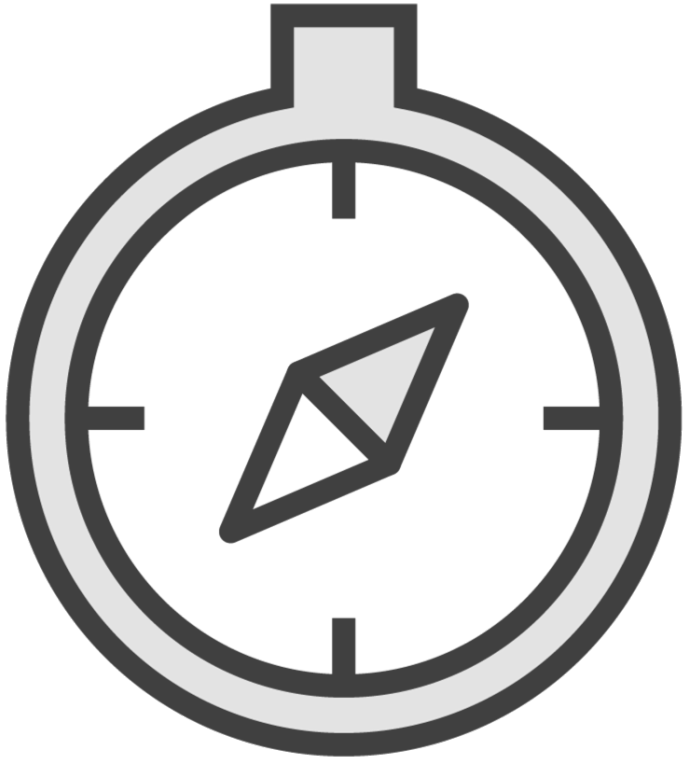
```
class MyCustomException extends RuntimeException {
}
```

...

```
class MyCustomException {
}

class MyCustomException extends IOException {
}
class MyCustomException extends RuntimeException {
}
```

Should I create runtime or checked?

```java
class InsufficientFundsException extends RuntimeException {

    public InsufficientFundsException(String msg){
        super(msg);
    }
}

if(accountFunds < withdrawAmount){
  throw new InsufficientFundsException("Not enough money in the account");
}
```



Creating both checked and unchecked exceptions is OK

It depends...

- Can be expected to recover? Checked!
- Otherwise: unchecked

Evaluated on a case-by-case basis

Exception handling rules!

Clean code principles!

Exception Handling Clean Code Principles

Never catch the Throwable

Prefer catching specific exceptions

Never leave catch blocks empty

Further Study

Book: Effective Java

- Chapter on Exceptions

Summary

Exception handling is indispensable in programming

Syntax and rules of:

- try/catch/finally
- try-with-resources

Catch chaining and multi-catch blocks

Exception class hierarchy

How to throw inbuilt and custom exceptions

Rating

Thank you!

(Happy coding)