Handling Exceptions (Java SE 8 Programmer I Certification 1Z0-808)

Introducing Exception Handling

Andrejs Doronins

App.java

```
try {
    // read a file
} catch (IOException ex) {
   // handle
}
finally { ... }

if(/* invalid input */){
   throw new IllegalArgumentException("ouch…");
}
```

Valid Code?

Filename.here

```
if(checkSomething()){
    throw RuntimeException();
 }
```

Filename.here

```
try {
// do stuff
} catch (IllegalArgumentException ex)
{
 // handle
} catch (NumberFormatException ex) {
// handle
}
```

Be the compiler...

Who This Course Is For

Studying for the Java SE 8 certification exam

Studying for future Java exams

Learning the details of exception handling

Exceptions in Java

2004

Java 5

2011

Java 7

try-with-resources

Catching multiple exception types

2018

Java 11

20xx

Java 15+

Java

=

strong backwards compatibility

Prerequisites

Java fundamentals

1+ years of working with Java

Course Overview

- Advantages of exception handling
- Syntax principles
- Exception types
  - Exception class hierarchy
- Throwing

Overview

Advantages of exception handling

Review try/catch/finally syntax

- Chaining catch blocks

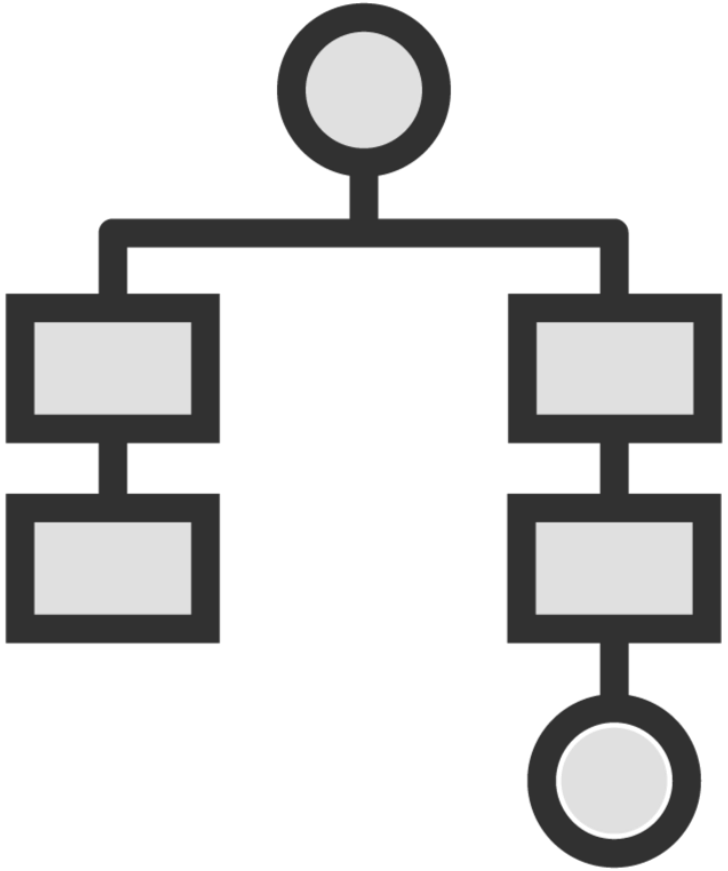
Demo

Internet is down

No disk space left

Access failure

Empty array



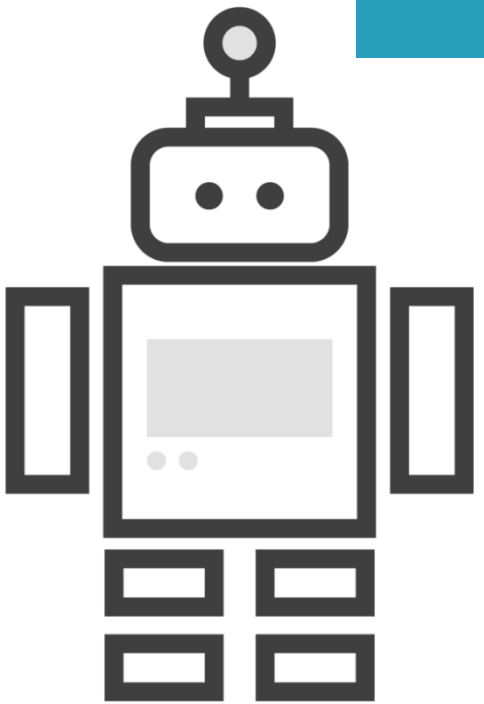
Errors:

- Within your control
- Outside of your control

Exam questions focus on exceptions caused by programming mistakes and errors (within your control)

Will this compile?

I give up. I don't know what to do right now. You deal with it!



```
System.exit(-1);
```

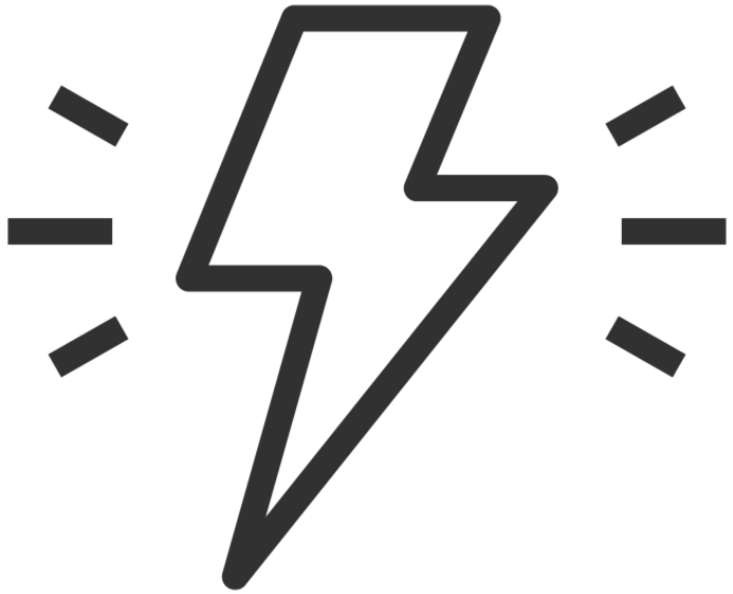
Error codes – ancestors of exceptions

A problem happened...

But what went wrong exactly?

Error: -4?

- **File can't be opened?**
- **Enough memory can't be allocated?**
- **Something else?**



Exceptions:

- Specific names for specific errors
- Example: NumberFormatException
  - “50” -> 50
  - “1a” -> ??

Special syntax to deal with exceptions:

- Try/catch/finally

```
try {
    // risky code
} catch (ExceptionType ex) {
    // handler
}
```

Curly braces are mandatory

Identifier – can have any name

e
err
anyVarName

Interchangeable Terms

Block

Clause

App.java

```java
BufferedReader br;
try {
    br = new BufferedReader(new FileReader("file.txt"));
    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    System.err.format("IOException: %s", e);
}
```

Valid Code?

Unreachable statement

```
try {
    throw new IOException();
    openFile("file.txt");
} catch (IOException ex) {
    // handle
}
```

catch or finally is mandatory

```
try {
    doRiskyStuff();
}
doTheOtherThing();
```

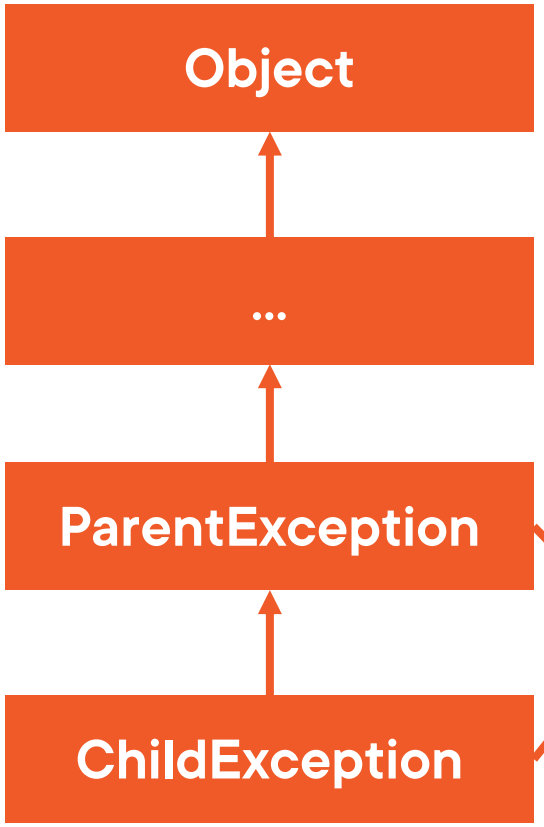
Valid Code?

Will not compile. Mandatory for try-catch

```
try
  openFile("file.txt");
catch (IOException e)
   System.out.println(e);
```

Brackets are optional for if-else blocks

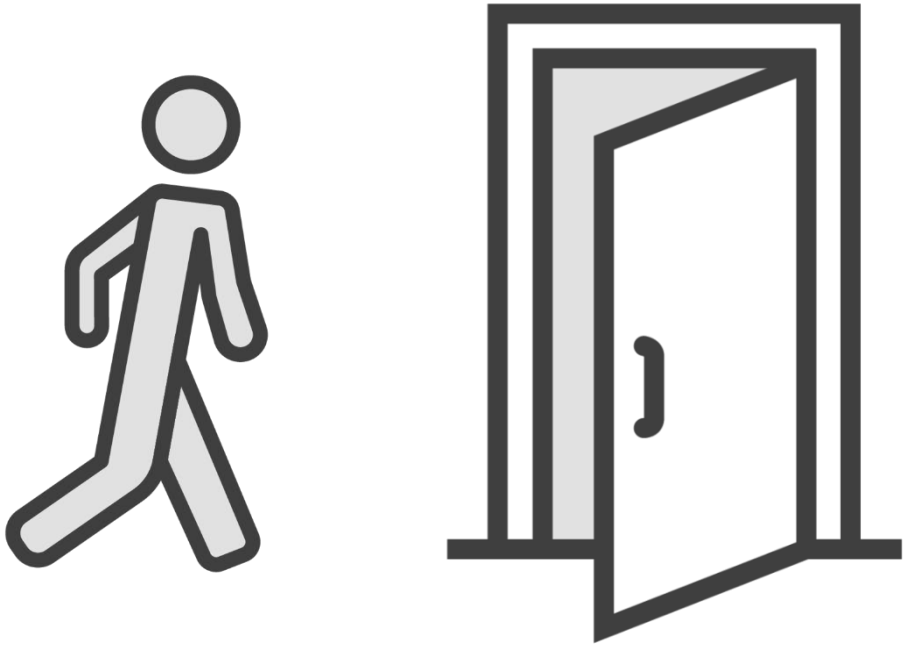
```
if (canReadFile("file.txt"))
    openFile("file.txt");
else
    System.out.println("error!");
```




```
try {
    // code
} catch (ExceptionType1 ex) {
    // handler
} catch (ExceptionType2 ex) {
    // handler
}
```

Must respect
the hierarchy

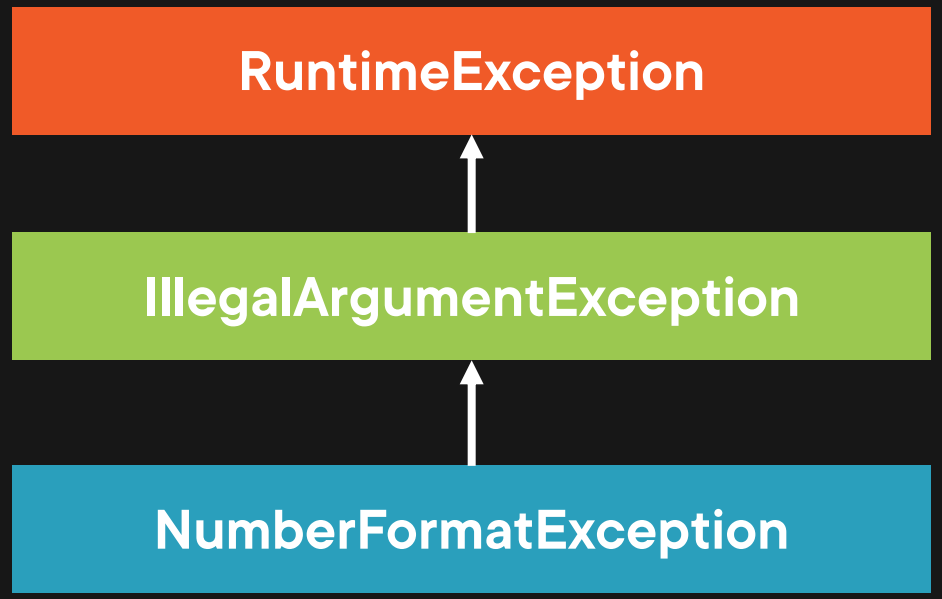
Understanding Exception Types

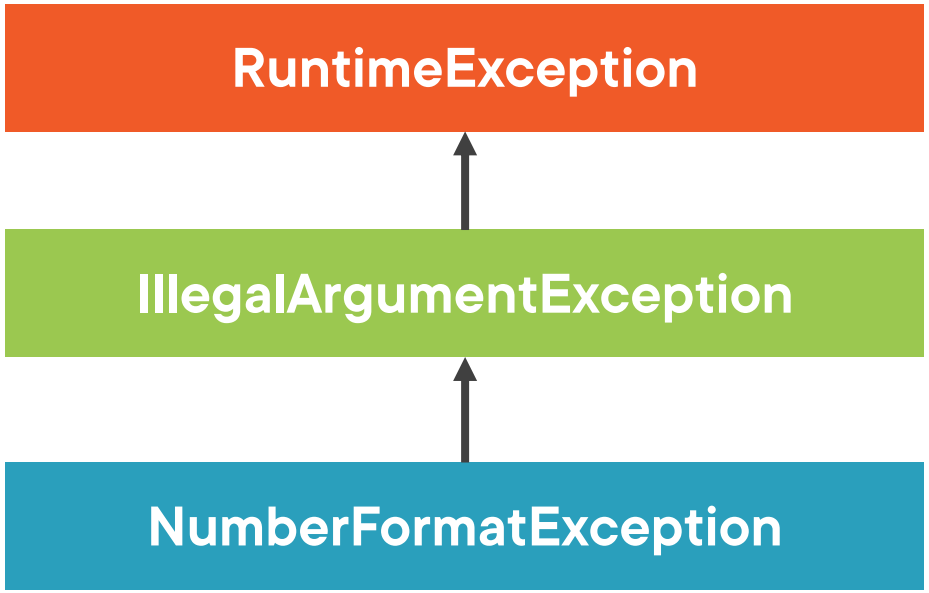


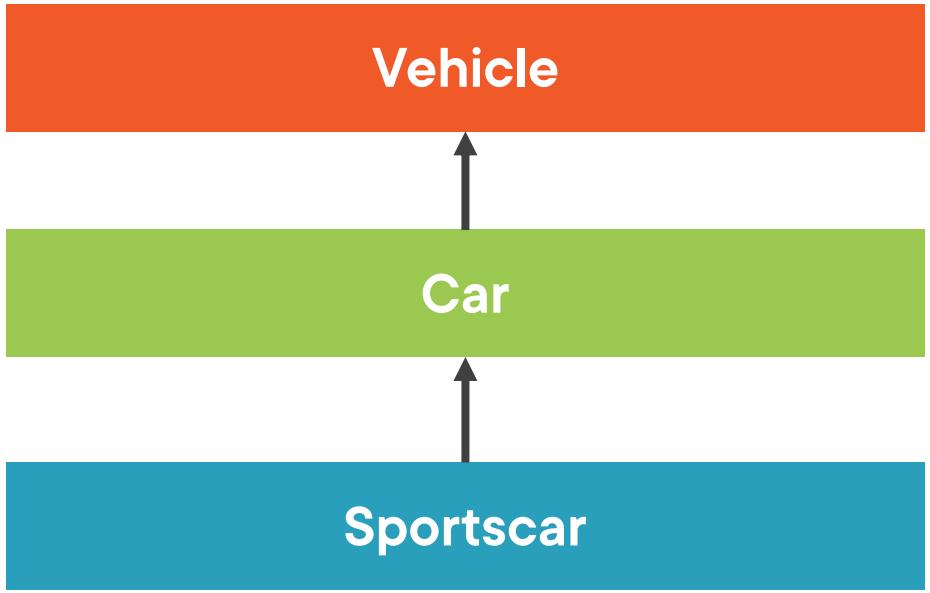
Understanding Exception Types

App.java


```
try {
  // parse user input
} catch (NumberFormatException ex) { }

catch (IllegalArgumentException ex)  { }

catch (RuntimeException ex) { }
```




RuntimeException
IllegalArgumentException
NumberFormatException

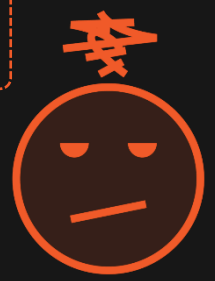

Vehicle
Car
Sportscar

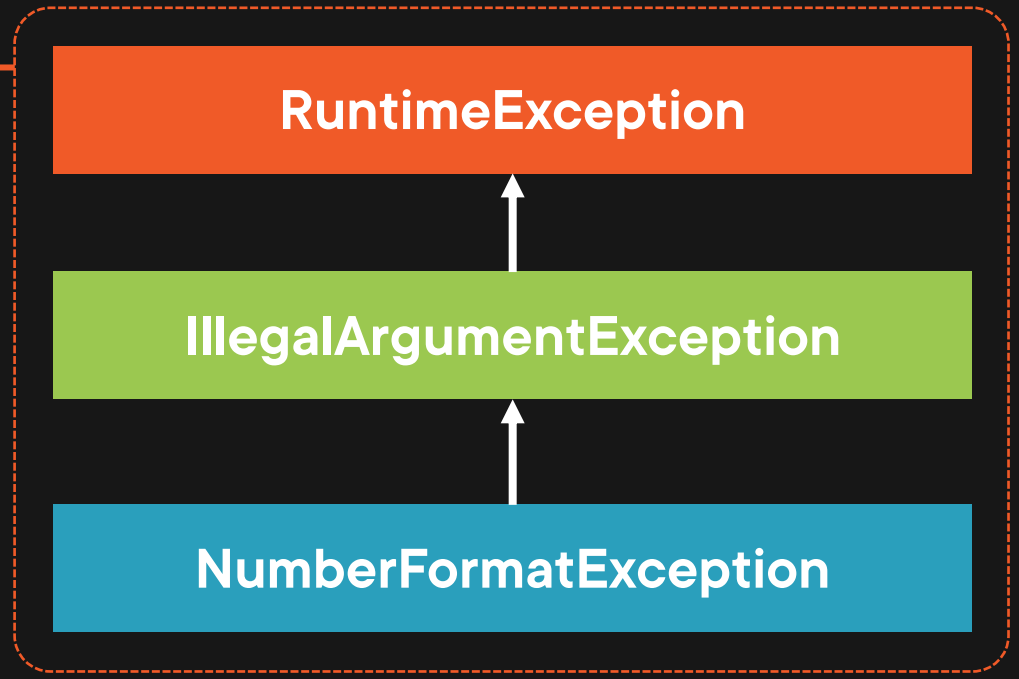
App.java

Catches (almost) everything

```
try {

} catch (RuntimeException ex) { }

catch (IllegalArgumentException ex)  { }

catch (NumberFormatException ex) { }
```

Unreachable statements





App.java

```
try {
  // parse user input
} catch (NumberFormatException ex1) {
    log(ex1);
}
catch (IllegalArgumentException ex2)  {
     log(ex2);
     log(ex1);
}
```

```
try {

    // open file / DB

} catch (ExceptionType ex) {
    // handler
} finally {
    // close file / DB
}
```

Optional if “finally” is present

Always (!) executes

Valid Code?

```
try {

} catch (Exception ex) {

} finally {

}
```

```
try {

} finally {

}
```

Valid Code?

```
try?
catch (Exception ex) {

} finally {

}
```

```
try {

} finally (Exception ex) {

}
```

App.java

Stop right now!

```
try {
    System.exit(0);
} finally {
    System.out.println("This won't print");
}
```

Demo

try/catch/finally syntax

Demo

try/catch/finally flow

Summary

try { } catch () { } finally { }

- syntax
- try/catch OR try/finally

Chaining “catch” blocks

- Specific exceptions first, general last

Exam questions: syntax and flow