Working with Exceptions

Esteban Herrera
JAVA ARCHITECT

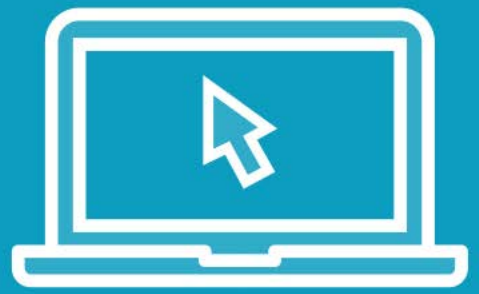
@eh3rrera http://eherrera.net

Overview

- Catch block rules
- Multi-catch block
- Finally block
- Try-with-resources block
- Rethrowing and chaining exceptions
- Throws clause

Catch Block Rules

Demo



Catch block rules

Run-time exception classes are **exempted** [from compile-time checking] because, in the judgment of the designers of the Java programming language, having to declare such exceptions would not aid significantly in establishing the correctness of programs.

The Java® Language Specification 11.2

http://bit.ly/jls11-2

Three Important Rules

Catch exceptions only once.

Subclasses must be caught before their superclasses.

Don’t catch checked exceptions that couldn't be thrown.

```
try {
    // Empty try block
} catch(Exception e) {
    // ...
}
```

- No compilation error

The Multi-catch Block

```
try {
    // ...
} catch (IOException e) {
    log.error(e);
} catch (SQLException e) {
    log.error(e);
} catch (ClassCastException e) {
    log.error(e);
}
```

Before Java 7...

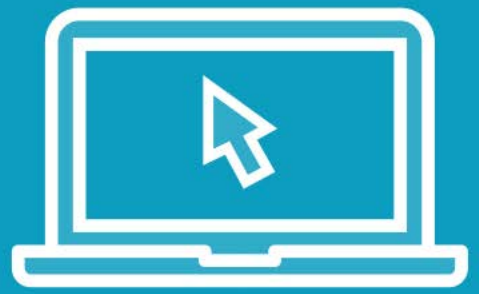
```
try {
    // ...
} catch (IOException | SQLException | ClassCastException e) {
     log.error(e);
}
```

After Java 7...

```
try {
    // Code that can throw ExceptionOne, ExceptionTwo, ExceptionThree
} catch (ExceptionOne | ExceptionTwo | ExceptionThree e) {
     // Do something with the caught exception using reference e
}
```

Multi-catch

Demo



Multi-catch block

Three Important Rules

Exceptions are separated by a pipe and there's only one reference.

Use exceptions not related through inheritance.

The reference is final.

The Finally Block

```
try {
    // Code that may throw an exception
} catch(Exception e) {
    // Do something with the exception
} finally {
    // Block that is always executed
}
```

Finally Block

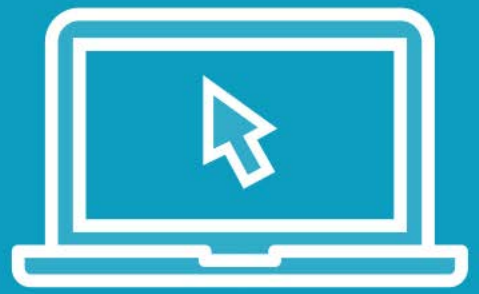
```
try {
    // Code that may throw an exception
} finally {
    // Block that is always executed
}
```

Finally Block

```
try {
    // ...
    System.exit(0);
} catch(Exception e) {
    // ...
} finally {
    // ...
}
```

- This will terminate the program abnormally
- Without executing the finally block

Demo



Finally block

If the finally block completes abruptly for reason S, then the try statement completes abruptly for reason S (and reason R is discarded).

The Java® Language Specification 14.20.2

http://bit.ly/jls14-20-2

Three Important Rules

Finally is always executed.

Finally can hide exceptions.

Returned values from finally.

The Try-With-Resources Block

```
try (AutoCloseableResource r = new AutoCloseableResource()) {
    // Code that may throw an exception
} catch(Exception e) {
    // Do something with the exception
} finally {
    // Block that is always executed
}
```

Try-with-resources Block

```
try (AutoCloseableResource r = new AutoCloseableResource()) {
    // Code that may throw an exception
} catch(Exception e) {
    // Do something with the exception
}
```

Try-with-resources Block

```
try (AutoCloseableResource r = new AutoCloseableResource()) {
    // Code that may throw an exception
}
```

Try-with-resources Block

```
try (AutoCloseableResource r = new AutoCloseableResource();
     AutoCloseableResource r2 = new AutoCloseableResource()) {
    // Code that may throw an exception
}
```

Try-with-resources Block

Interfaces to Implement by Resources

java.lang.AutoCloseable

java.io.Closeable

http://bit.ly/autocloseabledoc
http://bit.ly/closeabledoc

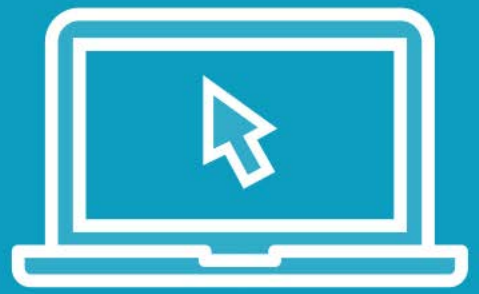
```
void close() throws IOException;
```

java.io.Closeable

```
void close() throws Exception;
```

java.lang.AutoCloseable

Demo



Try-with-resources block

Three Important Rules

Implement either java.lang.AutoCloseable or java.io.Closeable.

Initialized from left to right but closed in reverse order.

Suppressed exceptions.

Rethrowing and Chaining Exceptions

```
try {
    // ...
} catch(Exception e) {
    log.error(e);
}
```

```
try {
    // ...
} catch(Exception e) {
    e.printStackTrace();
}
```

```
try {
    // ...
} catch(Exception e) {
    return null;
}
```

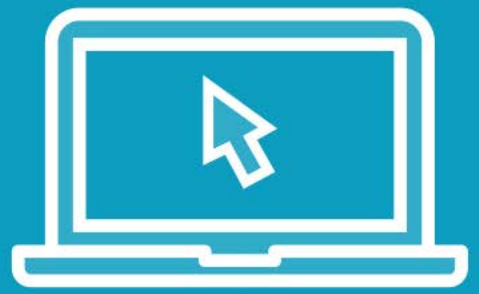
Handling a Caught Exception

Handle the exception locally

Propagate the exception by:

- Rethrowing the exception
- Chaining the exception

Demo



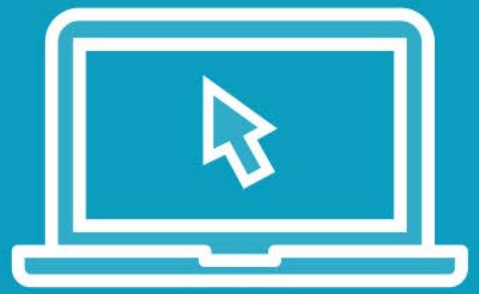
Rethrowing and chaining exceptions

The Throws Clause

```
void method() throws ExceptionOne, ExceptionTwo, ExceptionThree
{
    // Code that could throw ExceptionOne, ExceptionTwo,
    // and ExceptionThree
}
```

Throws Clause

Demo



Throws clause

Three Important Rules

Handled or declared.

Mix subclasses and superclasses.

Only throw the specified checked exceptions when overriding.

Summary

- Catch block rules
- Multi-catch block
- Finally block
- Try-with-resources block
- Rethrowing and chaining exceptions
- Throws clause