Exceptions

Class Error, class RuntimeException, and their derived classes are collectively referred

to as *unchecked exception classes*.

Adding runtime exceptions or errors to a method’s declaration

isn’t required. A method can throw a runtime exception or error

irrespective of whether its name is included in its throws clause.

A METHOD CAN THROW A SUBCLASS OF CHECKED EXCEPTION MENTIONED IN ITS THROWS CLAUSE,

NOT ITS SUPERCLASS

An overriding method can throw any error or runtime exception,

irrespective of whether they’re thrown by the overridden method or not.

A METHOD CAN HANDLE THE EXCEPTION AND STILL DECLARE IT TO BE THROWN

A METHOD THAT DECLARES A CHECKED EXCEPTION TO BE THROWN MIGHT NOT ACTUALLY THROW IT

With Java 7, you can rethrow exceptions with more inclusive

type checking

Custom

Don’t extend class java.lang.Throwable to create your own

exceptions, even though you can. Class Throwable is the superclass of

classes java.lang.Error and java.lang.Exception. The exception handler

for this class will *catch* all types of errors and exceptions.

If an overridden method doesn’t declare to throw any ***checked*** exception,

the overriding method can’t declare to throw a ***checked*** exception.

If an overridden method declares to throw a ***checked*** exception, the overriding method

* can choose to declare to not throw any checked exception,
* throw the same exception,
* throw a more specific exception.
* The overriding method can’t declare to throw a more generic checked exception.



Line1> class MultiCatch {

Line2> void myMethod(Connection con, String fileName) {

Line3> try {

Line4> File file = new File(fileName);

Line5> FileInputStream fin = new FileInputStream(file);

Line6> Statement stmt = con.createStatement();

Line7> }

Line8> catch (FileNotFoundException | SQLException e) {

Line9> System.out.println(e.toString());

Line10> }

Line11> }

Line12>}

EXCEPTIONS THAT YOU CATCH IN A MULTI-CATCH BLOCK CAN’T SHARE AN INHERITANCE RELATIONSHIP

You can combine multi-catch and single-catch blocks

try {

..

}

catch (FileNotFoundException e) {}

catch (IOException | SQLException e) {} // will catch IOExcpetion or any of its subclass except

FileNotFoundException

}

USING A SINGLE EXCEPTION VARIABLE IN THE MULTI-CATCH BLOCK

IN A MULTI-CATCH BLOCK, VARIABLE E IS IMPLICITLY FINAL

}

catch (IOException| SQLException e) {

**e = new FileNotFoundException(); // won’t work, variable is implicitly final**

}

Try with resource

A resource must

implement the java.lang.AutoCloseable interface or any of its subinterfaces to be

eligible to be declared in a try-with-resources statement. Let’s start with an example.

The resources initialized by the try-with-resources statement are automatically

closed, just before the end of execution of the try block. This happens regardless of

whether any exceptions are thrown.

if both the code in the try block and close()

throw an exception, the exception thrown by close() is *suppressed* by the exception

thrown by the try block.

A try-withresources

statement can declare multiple resources, which are separated by

a semicolon. After the last resource declaration, a semicolon is optional.

The variables used to refer to resources are implicitly final variables. You must *declare*

and *initialize* resources in the try-with-resources statement.

The resources declared with the try-with-resources are closed

in the reverse order of their declaration

Assertations

USE ASSERTIONS FOR SITUATIONS THAT CAN NEVER OCCUR (EVEN IN PUBLIC METHODS)

Use assertions for situations that can never occur, even in public methods. Use assertions

to test invariants—internal invariants, control-flow invariants, or class invariants