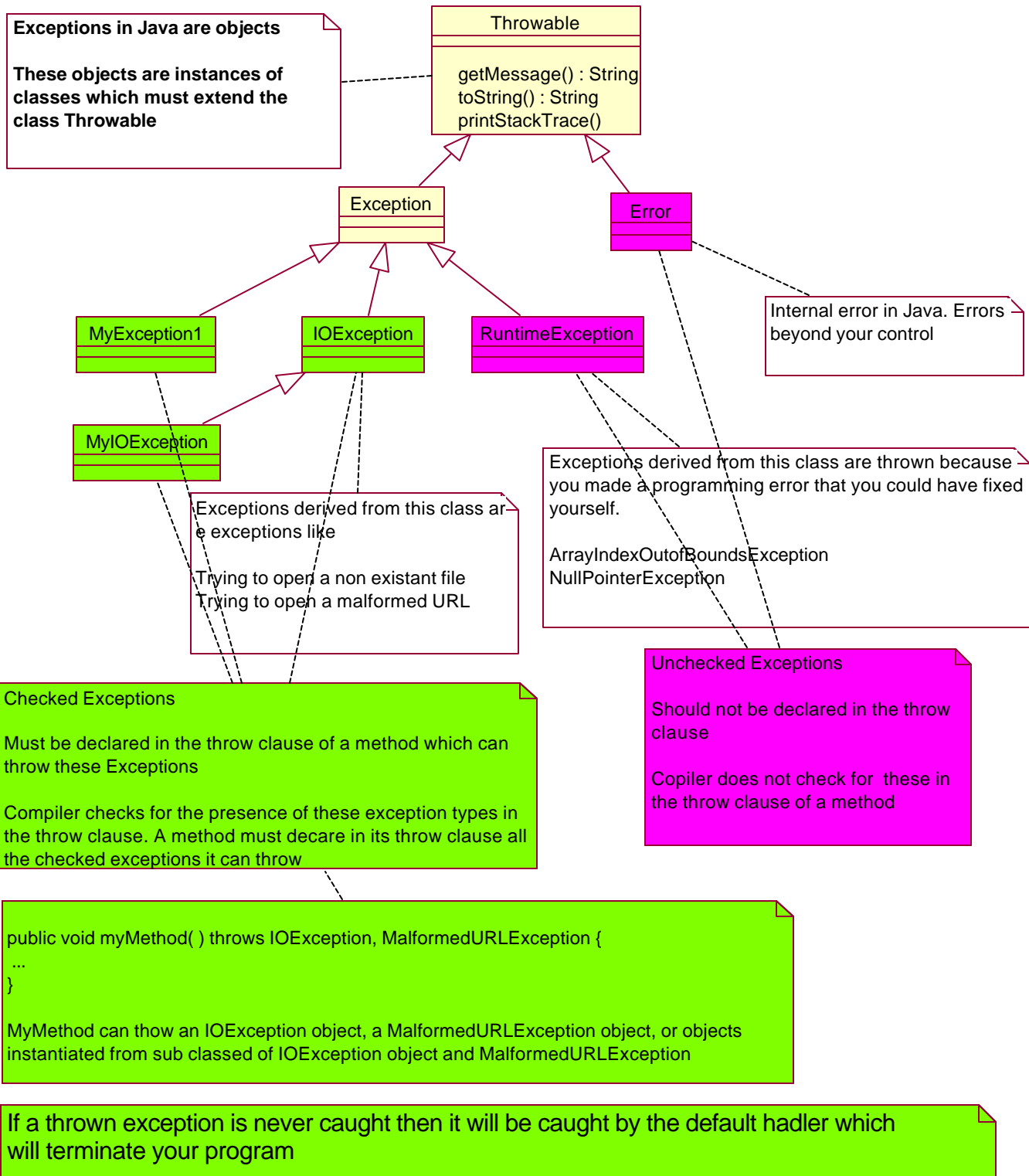


Exception Handling in Java

Experience has shown that programmers forget to handle errors or defer handling them until some future time that never arrives

The Java Programming Language, 2nd ed. Arnold & Gosling



Throwing an Exception

```
public class VelException extends Exception {  
  
    public Vector vel = null;  
  
    public VelException(String str) {  
        super(str);  
    }  
  
    public VelException(String str, Vector vel) {  
        this(str);  
        vel = (Vector) vel.clone();  
    }  
  
}
```

```
public class B {  
  
    Vector velocity = null;  
  
    public methodB1 throws IOException {  
  
        String expString = " .. ";  
        if( ... ) {  
            throw new IOException(expString);  
        }  
    }  
  
    public methodB2 throws IOException, VelException {  
        String str = null;  
  
        str = " .. ";  
        if( ... ) {  
            throw new IOException(str);  
        }  
  
        str = " .. ";  
        if( ... ) {  
            throw new velException(str, velocity);  
        }  
    }  
}
```

Catching an Exception

```
class A {  
  
    public methodA1 throws IOException { // letting an exception percolate through  
        B aB = new B();  
  
        aB.methodB1  
        ...  
    }  
}
```

Exception is percolated to the calling method of methodA1()

```
public methodA2 { // eating an exception. Not recommended  
    B aB = new B();  
  
    try{  
        aB.methodB1();  
        ...  
    }  
    catch (IOException ex) {  
  
    }  
}
```

If an exception is thrown by aB.methodB1 then the rest of the try block is not executed. Control is transferred to the catch block. In this example the exception is eaten (ignored)

At least put statements like this
System.out.println(ex);
ex.printStackTrace();

```
public methodA3 throws IOException { // rethrowing an exception  
    B aB = new B();  
  
    try{  
        aB.methodB1();  
        ...  
    }  
    catch (IOException ex) {  
        System.out.println(ex);  
        ex.printStackTrace();  
        // local cleanup, exception can not be fully  
        // resolved here, so rethrow it  
        throw ex;  
    }  
}
```

```
catch(IOException ex) {  
  
}
```

catches all objects instantiated from class IOException or its subclasses

```
public methodA4 throws velException {  
    B aB = new B();  
  
    try{  
        aB.methodB2();  
    }  
    catch(IOException ioEx) {  
        // handle exception here  
    }  
    catch(velException velEx) {  
        System.out.println(velEx);  
        // do something here  
        throw velEx;  
    }  
}
```

Multiple catch blocks associated with a try block are executed in order of appearance. The first block that matches an exception type is executed. The remaining blocks are not visited.

catch blocks checking for subclasses of an exception type should appear before catch blocks checking for the superclass. Otherwise the compiler will complain

Finally Block and Resource management

```
try {  
    ....  
}  
  
catch (IOException ex) {  
    ...  
}  
  
catch (velException ex) {  
    ...  
}  
  
finally {  
    // release non object resources  
    // clean up stuff  
}
```

A finally clause can be associated with a try statement

A try block may not have any catch clauses associated with it, but it still could have a finally block

The finally block is always executed, wether an exception id thrown in the associated try block or not.

A finally block may also throw an exception.

Two possibilities:

i) Exception Thrown in the try block: The try block is exited at the statement which results in the thrown exception (no further statements in the try block are executed). If any catch clauses are present, then all of them are examined in turn. When the proper catch clause is found its block is executed (this catch block may throw an exception). If no catch clause matches the exception, then none of the catch blocks is executed. Finally block is executed. If the exception has been handled in a catch block then execution continues after the finally block, otherwise the exception percolates to the calling function and control is passed to the calling function, and no statements after the finally block are executed.

ii) No Exception Thrown in the try block: The try block is completed successfully. None of the catch blocks is executed, but the finally block is executed. Execution continues after the finally block.

In both of the above two cases, wether an exception is thrown or not in the try block, the finally block is always executed.

Advice about Exception Handling

Practical Java, Peter Haggart

Never eat an exception

Do not handle exception handling at the end of the development cycle. Design your error handling strategy from the beginning.

Be specific and comprehensive with the throws clause
(don't be lazy and just include the super class in the throw clause, if subclass exceptions are being thrown)

Use finally to avoid resource leaks

Do not return from a try block
(remember, the finally block is always executed)

Place try/catch blocks outside of loops
(exception handling degrades performance)

Do not use exceptions for control flow or to report conditions that are not errors

Do not use exceptions for every error condition

Throw exceptions from constructors

Return objects to a valid state before throwing an exception