When should I create a new Exception class

I notice that a number of Java exception classes differ only in the name of the class and do not add any new functionality. Most exceptions for example just seem to override `Exception() or Exception(String message)` . This goes against the tenets of inheritance ie:- inherit to add **new** functionality.

What are some good reasons to create a new Exception class?

java exception

edited Dec 31 '09 at 19:04

asked Dec 31 '09 at 17:51

VDev
591 4 8 22

2 Inheriting to only change the name of an exception isn't a best practice IMHO. However, doing so to create an exception hierarchy providing for degrees of response at various abstraction layers is a good practice. In other words, there are good reasons and bad reasons to create new Exception classes! – harschware Dec 31 '09 at 19:00

10 Answers

Exceptions are a special case. In their case, the inheritance is not to add new functionality, but to add new classes of errors. This lets your code catch particular kinds of errors while ignoring others.

Say you are writing a large project. You have a Data component, and you have a Display component. They can both fail in various ways, and you want to throw exceptions for these failures. The Display component doesn't care about exceptions arising from the Data component, though, and vice versa. If all the classes just threw `Exception` , there'd be no way to figure out where the exception came from. However, if you subclass `Exception` with `DataException` and `GraphicsException` , even though they don't add new functionality, you can now throw and catch those particular types of exceptions, i.e. a graphics component can catch `GraphicsException` and not have to deal with data exceptions.

answered Dec 31 '09 at 17:54

Claudiu
70.4k 66 256 436

5 don't forget that inheritance (e.g. MyNumericDataException extends MyDataException) lets your exception-handling architecture group various types of exceptions together really cleanly. – Ollie Jones Dec 31 '09 at 18:01

2 Well, using a subclass to distinguish exception cause seems counterintuitive to the use of inheritance. There are other ways one can identify type. However the real issue is a limitation of the language, in the way catch/throw are inherently structured to distinguish based on class type. – VDev Dec 31 '09 at 18:49

3 Yes, but the single inheritance of Java classes is a natural fit for setting up an abstraction hierarchy. Thus, allowing degrees of response to an exception, ie rolling up exceptions for higher level abstractions. – harschware Dec 31 '09 at 18:57

@harschware: Good point, what you mention is a def plus. – VDev Dec 31 '09 at 19:01

You can catch exceptions by type, and having an exception of a particular type that is otherwise identical to the base Exception class allows you to be precise in your exception handling.

I would argue that it does add new functionality directly by increasing the specificity of exception handling code.

answered Dec 31 '09 at 17:54

jball
16k 6 44 77

Use a new exception type when you want to provide error handling for a certain condition. Take a look at java.io.IOException and you will see the following subclasses:

```
ChangedCharSetException
CharacterCodingException
CharConversionException
ClosedChannelException
EOFException
FileLockInterruptionException
FileNotFoundException
FilerException
HttpRetryException
IIOException
InterruptedIOException
InvalidPropertiesFormatException
JMXProviderException
JMXServerErrorException
MalformedURLException
ObjectStreamException
ProtocolException
RemoteException
SaslException
SocketException
SSLException
SyncFailedException
UnknownHostException
UnknownServiceException
UnsupportedDataTypeException
UnsupportedEncodingException
UTFDataFormatException
ZipException
```

Say you want to read from a file, you could get a generic IOException, a FileNotFoundException, and an EOFException. You may want to handle each of those cases differently. If all you had was IOException you would have to do something like look at the error message and then do an if/else statement to figure out what the actual error was so you can display a sensible message to the user. With a hierarchy of exceptions you can deal with those situations much easier.

answered Dec 31 '09 at 17:59

TofuBeer
36.6k 9 72 117

Slight nitpick but "FileNotFoundException, and an" should be "FileNotFoundException, or an" since only one exception can occur at a time. – Chris Dec 31 '09 at 18:50

True but you may wan all 3 catches and or would imply you only need one... Coin toss :-) – TofuBeer Dec 31 '09 at 21:37

In this case the functionality that inheritance adds is in regards to the catch statement, which can distinguish by type. Normally distinguishing subclasses by type is considered a bad practice, but exception handling is, well, an exception.

It definitely leads to odd situations (such as multiple catch statements that have the same logic) as you would expect when things are distinguished by type, but that is a limitation of the language.

answered Dec 31 '09 at 18:03

Yishai
47.8k 8 117 198

I would recommend reading all, or most, of Chapter 9 of Effective Java. Especially Item 60: "Favor the use of standard exceptions" and Item 61: "Throw exceptions appropriate to the abstraction".

EDIT 1: Or Items 42 and 43 in the first edition.

edited Dec 31 '09 at 18:22

answered Dec 31 '09 at 18:11

harschware
4,175 8 32 60

If you need or want to differentiate the particular error situation from other errors *in code*, then it makes sense to subclass the appropriate exception. You should *NOT* need to peek at the error message to determine what went wrong - that is for human eyes only.

Also note, that a properly named exception may be enough for a maintainers eye to deduce what went wrong when having to diagnose a stack trace. A typical example is ArrayOutOfBoundsException.

answered Jan 1 '10 at 2:11


Thorbjørn Ravn Andersen
43.1k 10 98 213

This renaming occurs to remove ambiguity wrt. the error that has occurred. e.g.

```
Foo getFoo() throws FooNotFoundException
```

will be explicit in indicating that a `Foo` cannot be found, as opposed to (what?) some illegal state, or a database connection error or similar, and you can choose to code (or not) for that *particular* error. If you choose to throw a more generic exception, it's less clear (programatically) what's occurred.

answered Dec 31 '09 at 17:55


Brian Agnew
161k 16 198 296

It's bad form to just catch *any* exception because not only will you not know where it came from, but you may be preventing yourself from handling errors correctly when they happen. If you just catch any type exception and handle it the same generic way, you may be hiding other problems in your software, or even introducing new ones. So, technically speaking, subclassing Exception *does* add new functionality.

answered Dec 31 '09 at 17:59


John Ewart
714 4 6

Creating an exception for any type is really dumb. Maybe you should create an exception for different layer (DAO, Service, etc) or different action (create record, duplicate record, etc).

answered Dec 31 '09 at 18:28


fastcodejava
15.3k 12 77 128

You add a new exception class when a user of your API might need to catch it conditionally (or, for checked exceptions specify different, specific types in a `throws`).

edited Dec 31 '09 at 18:49

answered Dec 31 '09 at 17:55


Craig Stuntz
102k 8 194 235

The parenthetical is... confusing. In Java, the "checking" of checked exceptions is done by the compiler, not by your code. – Laurence Gonsalves Dec 31 '09 at 18:31

I'll rephrase that bit. – Craig Stuntz Dec 31 '09 at 18:48
