Do you write exceptions for specific issues or general exceptions?

Asked 16 years, 4 months ago Modified 15 years, 4 months ago Viewed 1k times

10

I have some code that gives a user id to a utility that then send email to that user.

```
emailUtil.sendEmail(userId, "foo");

public void sendEmail(String userId, String message) throws MailException {
    /* ... logic that could throw a MailException */
}
```

`MailException` could be thrown for a number of reasons, problems with the email address, problems with the mail template etc.

My question is this: do you create a new Exception type for every one of these exceptions and then deal with them individually or do you create one MailException and then store something in the exception (something computer-readable, not the description text) that allows us to do different things based on what actually happened.

Edit: As a clarification, the exceptions aren't for logs and what-not, this relates to how code reacts to them. To keep going with the mail example, let's say that when we send mail it could fail because you don't have an email address, or it could because you don't have a **valid** email address, or it could fail.. etc.

My code would want to react differently to each of these issues (mostly by changing the message returned to the client, but actual logic as well).

Would it be best to have an exception implementation for each one of these issues or one umbrella exception that had something internal to it (an enum say) that let the code distinguish what kind of issue it was.

c# java exception

Share

Improve this question

Follow

edited Aug 7, 2009 at 10:28

Mnementh

51.3k ● 48 ● 151 ● 202

asked Aug 22, 2008 at 2:47

SCdF

59.3k ● 24 ● 79 ● 114

11 Answers

Sorted by: Highest score (default)

10

In my code, I find that MOST exceptions percolate up to a UI layer where they are caught by my exception handlers which simply display a message to the user (and write to the log). It's an unexpected exception, after all.

Sometimes, I do want to catch a specific exception (as you seem to want to do). You'll probably find, however, that this is somewhat rare and that it is indicative of using exceptions to control logic -- which is inefficient (slow) and often frowned upon.

So using your example, if you want to run some special logic when the email server is not configured, you may want to add a method to the emailUtil object like:

public bool isEmailConfigured()

... call that first, instead of looking for a specific exception.

When an exception does happen, it really means that the situation was completely unexpected and the code can't handle it -- so the best you can do is report it to the user (or write it to a log or restart)

As for having an exception hierarchy vs exceptions-with-error-codes-in-them, I typically do the latter. It's easier to add new exceptions, if you just need to define a new error constant instead of a whole new class. But, it doesn't matter much as long as you try to be consistent throughout your project.

Share

Improve this answer

Follow

edited Feb 9, 2009 at 18:42

answered Aug 22, 2008 at 3:07

jm.

23.7k ● 23 ● 81 ● 93

Do you also log your exceptions server side? – pjp Aug 7, 2009 at 10:39

@php: Yes, good point, if it's a server code. For real client apps (not web), I also usually have option to log exceptions on client side for debugging. – jm. Aug 8, 2009 at 17:21

8

I usually start with a general exception and subclass it as needed. I always can catch the general exception (and with it all subclassed exceptions) if needed, but also the specific.

An example from the Java-API is IOException, that has subclasses like FileNotFoundException or EOFException (and much more).

This way you get the advantages of both, you don't have throw-clauses like:

```
throws SpecificException1, SpecificException2, SpecificException3 ...
```

a general

```
throws GeneralException
```

is enough. But if you want to have a special reaction to special circumstances you can always catch the specific exception.

Share
Improve this answer
Follow

edited May 19, 2009 at 7:21

answered Sep 26, 2008 at 9:55

Mnementh
51.3k ● 48 ● 151 ● 202

2

@Chris.Lively

You know you can pass a message in your exception, or even the "status codes". You are reinventing the wheel here.

Share Improve this answer Follow

answered Aug 22, 2008 at 4:46

FlySwat
175k ● 75 ● 248 ● 314

2

I have found that if you need to have CODE deciding what to do based on the exception returned, create a well named exception subclassing a common base type. The message passed should be considered "human eyes only" and too fragile to make decisions upon. Let the compiler do the work!

If you need to pass this up to a higher layer through a mechanism not aware of checked exceptions, you can wrap it in a suitable named subclass of RuntimeException (MailDomainException) which can be caught up high, and the original cause acted upon.

Share Improve this answer Follow

answered Jan 11, 2009 at 19:28

Thorbjørn Ravn Andersen
75.3k ● 34 ● 199 ● 352

1

It depends on what your application is doing. You might want to throw individual exceptions in cases like

- The application is high availability

- Sending e-mail is particularly important
- The scope of the application is small and sending e-mail is a large part of it
- The application will be deployed to a site which is remote and you will only get logs for debugging
- You can recover from some subset of the exceptions encapsulated in the mailException but not others

In most cases I would say just log the text of the exception and don't waste your time granularizing already pretty granular exceptions.

Share Improve this answer Follow

answered Aug 22, 2008 at 3:00

stimms

44k ● 31 ● 101 ● 151

1

I think a combination of the above is going to give you the best result.

You can throw different exceptions depending on the problem. e.g. Missing email address = ArgumentException.

But then in the UI layer you can check the exception type and, if need be, the message and then display a appropriate message to the user. I personally tend to only show a informational message to the user if a certain type of exception is thrown (UserException in my app). Of course you should scrub and verify user input as much as possible further up the stack to make sure any exceptions are generated by truly unlikely scenarios, not as a filter for malformed emails which can easily be checked with a regex.

I also wouldn't worry about the performance implications of catching an exception from user input. The only time you are going to see performance problems from exceptions is when they are being thrown and caught in a loop or similar.

Share Improve this answer Follow

answered Aug 26, 2008 at 2:03

abigblackman

1,491 ● 1 ● 9 ● 4

1

Instead of using exceptions, I tend to return a list of status objects from methods that may have problems executing. The status objects contain a severity enum (information, warning, error, ...) a status object name like "Email Address" and a user readable message like "Badly formatted Email Address"

The calling code would then decide which to filter up to the UI and which to handle itself.

Personally, I think exceptions are strictly for when you can't implement a normal code solution. The performance hit and handling restrictions are just a bit too much for me.

Another reason for using a list of status objects is that identifying multiple errors (such as during validation) is MUCH easier. After all, you can only throw one exception which must be handled before moving on.

Imagine a user submitting an email that had a malformed destination address and contained language that you are blocking. Do you throw the malformed email exception, then, after they fix that and resubmit, throw a bad language exception? From a user experience perspective dealing with all of them at once is a better way to go.

UPDATE: combining answers

@Jonathan: My point was that I can evaluate the action, in this case sending an email, and send back multiple failure reasons. For example, "bad email address", "blank message title", etc..

With an exception, you're limited to just percolating the one problem then asking the user to resubmit at which point they find out about a second problem. This is really bad UI design.

Reinventing the wheel.. possibly. However, most applications should analyze the whole transaction in order to give the best possible information to the user. Imagine if your compiler stopped dead at the first error. You then fix the error and hit compile again only to have it stop again for a different error. What a pain in the butt. To me, that's exactly the problem with throwing exceptions and hence the reason I said to use a different mechanism.

Share

Improve this answer

Follow

edited Nov 3, 2008 at 16:22

answered Aug 22, 2008 at 4:35

ChrisLively

88k ● 27 ● 173 ● 247

1 Exceptions are very useful and make code much more readable if used right. Performance of language-features like exception are changing with every release of modern virtual-machines, so you have to measure it every time against the if-clauses (that you clearly must use to check the result of a method). The if-checks to check result of methods clutter your code and make it harder to maintain. Don't get me wrong - sometimes status-objects are a good solution. Especially if you handle multiple errors (as you mentioned) they are more useful than exceptions. But the decision should be well-thought. – Mnementh Aug 14, 2009 at 11:01

I tend to have less Exception types, although it's not really the OO way to do it. Instead I put an enum to my custom Exceptions, which classifies the Exception. Most

0

of the time I have a custom base Exception, which holds on to a couple of members, which can be overridden or customized in derived Exception types.

A couple of months ago I blogged about the idea of how to internationalize Exceptions. It includes some of the ideas mentioned above.

Share Improve this answer Follow

answered Aug 22, 2008 at 5:42

dlinsin

19.6k ● 13 ● 43 ● 53

0

While you can differenciate the code execution looking the exception don't matter if it's done by the "catch exceptionType hierarchy mode" or by "if(...) else...exception code mode"

but if you are developing software wich is going to be used by other people, like a library i think it's usefull create your own exception types to notice the other people that your sofware can throw other exceptions than the normal ones, and they better catch and resolve them.

When i use a library and their methods simply launch an 'Exception' i allways wonder: What can cause this exception?, how must my program react?, if there is a javadoc maybe the cause will be explained, but mustly of times there is not a javadoc or the exception is not explained. Too much overhead witch can be avoided with a WellChossenExceptionTypeName

Share Improve this answer Follow

answered Aug 22, 2008 at 15:13

Telcontar

4,870 ● 7 ● 32 ● 41

0

It depends on whether the code that catches the exception needs to differentiate between exceptions or whether you are just using exceptions to fail out to an error page. If you need to differentiate between a NullReference exception and your custom MailException higher up in the call stack, then spend the time and write it. But most of the time programmers just use exceptions as a catch all to throw up an error on the web page. In this case you are just wasting effort on writing a new exception.

Share Improve this answer Follow

answered Aug 26, 2008 at 1:57

Jim

3,170 ● 4 ● 32 ● 38

I would just go by

-1

```
throw new exception("WhatCausedIt")
```

if you want to handle your exceptions, you could pass a code instead of "WhatCausedIt" an then react to the different answers with a switch statement.

Share Improve this answer Follow

answered Aug 22, 2008 at 5:48

niklasfi

15.9k ● 7 ● 43 ● 55

Instead of a code in the Exception passed as a parameter, you could subclass exception and use instanceof. – Mnementh Aug 14, 2009 at 10:55