Is it really that bad to catch a general exception?

Asked 16 years, 4 months ago Modified 1 year, 4 months ago

Viewed 69k times

77

Whilst analysing some legacy code with FXCop, it occurred to me is it really that bad to catch a general exception error within a try block or should you be looking for a specific exception. Thoughts on a postcard please.

exception

Share

Improve this question

Follow

asked Aug 22, 2008 at 7:41

Triss

847 ● 2 ● 8 ● 11

2 If input is not predictable, and failure is not an option - it's the only way. Of course, there is no good reason not to LOG the error so that it can be properly handled the next time. – krethika Dec 2, 2012 at 1:59

1 Is this a question that would be better suited to programmers.stackexchange? – HardcoreBro Nov 22, 2013 at 16:13

I love that this is also called "**Pokémon catch** — gotta catch 'em all". – Uwe Keim Aug 16, 2016 at 6:54

Duplicate on SE.SE: Is catching general exceptions really a bad thing? – jrh Jun 11, 2018 at 17:31

16 Answers

Sorted by: Highest score (default)

120

Obviously this is one of those questions where the only real answer is "it depends."

The main thing it depends on is where your are catching the exception. In general libraries should be more conservative with catching exceptions whereas at the top level of your program (e.g. in your main method or in the top of the action method in a controller, etc) you can be more liberal with what you catch.

The reason for this is that e.g. you don't want to catch all exceptions in a library because you may mask problems that have nothing to do with your library, like "OutOfMemoryException" which you really would prefer bubbles up so that the user can be notified, etc. On the other hand, if you are talking about catching exceptions inside your main() method which catches the exception, displays it and then exits... well, it's probably safe to catch just about any exception here.

The most important rule about catching all exceptions is that you should never just swallow all exceptions silently... e.g. something like this in Java:

```
try {
    something();
} catch (Exception ex) {}
```

or this in Python:

```
try:
    something()
except:
    pass
```

Because these can be some of the hardest issues to track down.

A good rule of thumb is that you should only catch exceptions that you can properly deal with yourself. If you cannot handle the exception completely then you should let it bubble up to someone who can.

Share Improve this answer

Follow

edited Oct 27, 2011 at 18:44

Matthew Flaschen

284k ● 53 ● 521 ● 552

answered Aug 22, 2008 at 8:53

John

15.3k ● 12 ● 59 ● 57

11 Catching all exceptions at language borders to translate them is also good practice. – Alexandre C. Aug 3, 2011 at 16:12

"The reason for this is that e.g. you don't want to catch all exceptions in a library because you may mask problems that have nothing to do with your library, like "OutOfMemoryException" " The funny thing is, the .NET framework actually catches `OutOfMemoryException` sometimes. – jrh Jun 11, 2018 at 17:29

35

Unless you are doing some logging and clean up code in the front end of your application, then I think it is bad to catch all exceptions.

My basic rule of thumb is to catch all the exceptions you expect and anything else is a bug.

If you catch everything and continue on, it's a bit like putting a sticking plaster over the warning light on your car dashboard. You can't see it anymore, but it doesn't mean everything is ok.

Share Improve this answer Follow

answered Aug 22, 2008 at 8:50

JamesSugrue
15k ● 10 ● 62 ● 93

Sometimes it is important to catch all the exceptions for particular block of code which may not be an important but next lines of code must execute for existing behavior to continue. – Gentleman Jan 3 at 17:22

15

Yes! (except at the "top" of your application)

By catching an exception and allowing the code execution to continue, you are stating that you know how do deal with and circumvent, or fix a particular problem. You are stating that this is **a recoverable situation**.

Catching Exception or SystemException means that you will catch problems like IO errors, network errors, out-of-memory errors, missing-code errors, null-pointer-dereferencing and the likes. It is a lie to say that you can deal with these.

In a well organised application, these unrecoverable problems should be handled high up the stack.

In addition, as code evolves, you don't want your function to catch a new exception that is added *in the future* to a called method.

Share Improve this answer Follow

answered Aug 22, 2008 at 8:59

Cheekysoft
35.6k ● 20 ● 74 ● 86

1 The catch could be used to add additional information and then rethrow, right? – Maarten Bodewes Sep 7, 2016 at 14:50

There is good reason to catch ALL exceptions and wrap them with additional context information and rethrow the new exception. In this case you **absolutely do** want to catch any and all exception types that are added in the future. – AgilePro Apr 15 at 16:36

12

In my opinion you should catch all exceptions you **expect**, but this rule applies to anything but your interface logic. All the way down the call stack you should probably create a way to catch all exceptions, do some logging/give user feedback and, if needed and possible, shut down gracefully.

Nothing is worse than an application crashing with some user unfriendly stacktrace dumped to the screen. Not only does it give (perhaps unwanted) insight into your code, but it also confuses your end-user, and sometimes even scares them away to a competing application.

Share Improve this answer Follow

answered Aug 22, 2008 at 9:02

Erik van Brakel
23.8k ● 3 ● 53 ● 66

Yes, there is something worse: crashing without any indication on the screen. There is good reason to catch ALL exceptions and wrap them with additional context information and rethrow the new exception. In this case you absolutely do want to catch any and all exception types that are added in the future – AgilePro Apr 15 at 16:37

8

There's been a lot of philosophical discussions (more like arguments) about this issue. Personally, I believe the worst thing you can do is swallow exceptions. The next worst is allowing an exception to bubble up to the surface where the user gets a nasty screen full of technical mumbo-jumbo.

Share Improve this answer

Follow

answered Aug 22, 2008 at 12:47

Tundey
2,965 ● 1 ● 24 ● 28

10 Letting an exception bubble up does not mean showing it to the end user. You can (and should) show a generic error page, log it for investigation, and then fix the problem that caused the exception in the first place. – jammycakes Apr 20, 2009 at 8:52

2 Does not answer question asked. – Kenneth K. Feb 12, 2016 at 13:47

Completely agree that swallowing exceptions is bad. As for the next worse: if your exception contain nothing but mumbo jumbo, I think that is the problem to address, not the fact that users see it. – AgilePro Apr 15 at 16:39

6

Well, I don't see any difference between catching a general exception or a specific one, except that when having multiple catch blocks, you can react differently depending on what the exception is.

In conclusion, you will catch both `IOException` and `NullPointerException` with a generic `Exception`, but the way your program should react is probably different.

Share Improve this answer

Follow

edited Feb 20, 2019 at 3:18

surendrapanday
538 ● 4 ● 14

answered Aug 22, 2008 at 8:44

Philippe
4,041 ● 3 ● 39 ● 57

1 Precisely: it depends on what the catch block is doing. Some are designed to handle **all** exceptions, and then catching `Exception` is appropriate. Some catch blocks are designed to response to specific exceptions, and they should list the specific exceptions. – AgilePro Apr 15 at 16:40

3

The point is twofold I think.

Firstly, if you don't know what exception has occurred how can you hope to recover from it. If you expect that a user might type a filename in wrong then you can expect a FileNotFoundException and tell the user to try again. If that same code generated a NullReferenceException and you simply told the user to try again they wouldn't know what had happened.

Secondly, the FxCop guidelines do focus on Library/Framework code - not all their rules are designed to be applicable to EXE's or ASP.Net web sites. So having a global exception handler that will log all exceptions and exit the application nicely is a good thing to have.

Share Improve this answer Follow

answered Aug 22, 2008 at 8:52
samjudson
56.8k ● 7 ● 60 ● 69

It is bad to code logic by exceptions, if you expect that file provided by user could not exist, there is a method to check that: File.exists(). – Krzysztof Cichocki Jun 27, 2016 at 11:02

1 @KrzysztofCichocki - I know this is old, but actually, coding file logic using exceptions in this way is more correct. There's a great explanation by Eric Lippert here. He terms this type "exogenous" exceptions. – Olly Jul 4, 2017 at 15:35

+1 for the answer from @Olly: You can't check if a file exists upfront. There is always a race-condition, where it can be deleted before you access it. – Jonas Benz Oct 8, 2019 at 7:39

3

The problem with catching all exceptions is that you may be catching ones that you don't expect, or indeed ones that you should **not** be catching. The fact is that an exception of any kind indicates that something has gone wrong, and you have to sort it out before continuing otherwise you may end up with data integrity problems and other bugs that are not so easy to track down.

To give one example, in one project I implemented an exception type called CriticalException. This indicates an error condition that requires intervention by the developers and/or administrative staff otherwise customers get incorrectly billed, or other data integrity problems might result. It can also be used in other similar cases when merely logging the exception is not sufficient, and an e-mail alert needs to be sent out.

Another developer who didn't properly understand the concept of exceptions then wrapped some code that could potentially throw this exception in a generic try...catch block which discarded all exceptions. Fortunately, I spotted it, but it could have resulted in serious problems, especially since the "very uncommon" corner case that it was supposed to catch turned out to be a lot more common than I anticipated.

So in general, catching generic exceptions is bad unless you are 100% sure that you know **exactly** which kinds of exceptions will be thrown and under which circumstances. If in doubt, let them bubble up to the top level exception handler instead.

A similar rule here is never throw exceptions of type System.Exception. You (or another developer) may want to catch your specific exception higher up the call stack while letting others go through.

(There is one point to note, however. In .NET 2.0, if a thread encounters any uncaught exceptions it unloads your whole app domain. So you should wrap the main body of a thread in a generic try...catch block and pass any exceptions caught there to your global exception handling code.)

Share Improve this answer Follow

answered Aug 22, 2008 at 9:50

jammycakes
5,866 ● 3 ● 41 ● 49

What would happen if the code which called `CriticalException` also called `Thread.Abort(Thread.CurrentThread)`? Because the thread being aborted would be in a known state (calling `Abort`), the normal hazards of that method wouldn't apply; code which does a `catch(Exception ex)` would catch the resulting thread-abort exception, but wouldn't stop it from propagating up the call chain. It would be better, of course, if there were some way a more meaningfully-named exception could behave likewise, but I know of no such feature in .NET. – supercat Dec 6, 2013 at 23:24

The problem was the code that discarded exceptions. That should never be done. There is good reason to catch ALL exceptions and wrap them with additional context information and rethrow the new exception. In this case you absolutely do want to catch any and all exception types that are added in the future – AgilePro Apr 15 at 16:42

3

There are two completely different use cases. The first is the one most people are thinking about, putting a try/catch around some operation that requires a checked exception. This should not be a catch-all by any means.

The second, however, is to stop your program from breaking when it could continue. These cases are:

- The top of all threads (By default, exceptions will vanish without a trace!)
- Inside a main processing loop that you expect to never exit

- Inside a Loop processing a list of objects where one failure shouldn't stop others
- Top of the "main" thread--You might control a crash here, like dump a little data to stdout when you run out of memory.
- If you have a "Runner" that runs code (for instance, if someone adds a listener to you and you call the listener) then when you run the code you should catch Exception to log the problem and let you continue notifying other listeners.

These cases you ALWAYS want to catch Exception (Maybe even Throwable sometimes) in order to catch programming/unexpected errors, log them and continue.

Share Improve this answer Follow

answered Feb 8, 2018 at 21:33

Bill K
62.8k ● 18 ● 112 ● 158

This is a well considered answer. "It depends" on what you are doing in the catch block in order to say whether the catch block should catch all exceptions or not. You focused here on stopping the exception, however there are more cases when you simply want to wrap with more information and rethrow. – AgilePro Apr 15 at 16:47

2

I would like to play devil's advocate for catching Exception and logging it and rethrowing it. This can be necessary if, for example, you are somewhere in the code and an unexpected exception happens, you can catch it, log meaningful state information that wouldn't be available in a simple stack trace, and then rethrow it to upper layers to deal with.

Share Improve this answer Follow

answered Jan 26, 2015 at 18:59

slashdottir
8,456 ● 7 ● 59 ● 77

1 "log and rethrow" should be discouraged because if you have a global root level exception handler that logs (and you should build this first) then you are only adding multiple redundant log messages. Instead, spent a few moments to make sure the global handler logs correctly. – AgilePro Apr 15 at 16:45

1

Unpopular opinion: Not really.

Catch all of the errors you can meaningfully recover from. Sometimes that's all of them.

In my experience, it matters more *where* the exception came from than which exception is actually thrown. If you keep your exceptions in tight quarters, you won't usually be swallowing anything that would otherwise be useful. Most of the information encoded in the type of an error is ancillary information, so you generally end up effectively catching *all* of them anyway (but you now have to look up the API docs to get the total set of possible Exceptions).

Keep in mind that some exceptions that should bubble up to the top in almost every case, such as Python's

`KeyboardInterrupt` and `SystemExit`. Fortunately for Python, these are kept in a separate branch of the exception hierarchy, so you can let them bubble up by catching `Exception`. A well-designed exception hierarchy makes this type of thing really straightforward.

The main time catching general exceptions will cause serious problems is when dealing with resources that need to be cleaned up (perhaps in a `finally` clause), since a catch-all handler can easily miss that sort of thing. Fortunately this isn't really an issue for languages with `defer`, constructs like Python's `with`, or RAII in C++ and Rust.

Share Improve this answer Follow

edited Feb 6, 2019 at 22:26

answered Feb 6, 2019 at 22:14

Beefster
769 ● 7 ● 22

There is a good case to made for catching **all** exceptions, wrapping them with additional information, and then rethrowing them. – AgilePro Apr 15 at 16:48

@AgilePro it really depends on the use case, as sometimes doing so can hide useful information behind the wrapped exception. For example, sometimes exceptions are used to propagate an error to the HTTP framework to return as an error message, and it's essential that these exceptions are allowed to bubble up. It's almost more useful to catch all exceptions besides a certain type, but that's not really something any language supports – Beefster Apr 16 at 22:49

Indeed it depends on the situation. If a file parser fails, the parse does not include information about the name of the file in the error, because it was parsing a stream and does not have access to the name. It is a GOOD practice to catch this exception, and wrap it with another that cites the file name that failed and RETHROW. This example is NOT about recovering. Since it is situation dependent, we should NOT have a general rule outlawing it. – AgilePro Apr 20 at 18:54

0

I think a good guideline is to catch only specific exceptions from within a framework (so that the host application can deal with edge cases like the disk filling up etc), but I don't see why we shouldn't be able to catch all exceptions from our application code. Quite simply there are times where you don't want the app to crash, no matter what might go wrong.

Share Improve this answer Follow

answered Aug 22, 2008 at 8:45

Matt Hamilton
204k ● 61 ● 392 ● 321

If the catch block is designed to handle **all** exceptions, then it should catch the base class `Exception`. This is particularly in the case of a catch block that wraps the exception in additional context information. You really do want to wrap **all** exceptions in this manner, and catching only some of them leaves open the possibility that in the future there will be new exceptions that are missed. – AgilePro Apr 15 at 16:50

0

Most of the time catching a general exception is not needed. Of course there are situations where you don't have a choice, but in this case I think it's better to check why you need to catch it. Maybe there's something wrong in your design.

Share Improve this answer Follow

answered Aug 22, 2008 at 8:49

jfs
16.7k ● 13 ● 63 ● 90

There is a fairly common situation with a catch block that wraps the exception in additional context information. You really do want to wrap all exceptions in this manner, and catching only some of them leaves open the possibility that in the future there will be new exceptions that are missed.
– AgilePro Apr 15 at 16:51

0

Catching general exception, I feel is like holding a stick of dynamite inside a burning building, and putting out the fuze. It helps for a short while, but dynamite will blow anyways after a while.

Of corse there might be situations where catching a general Exception is necessary, but only for debug purposes. Errors and bugs should be fixed, not hidden.

Share Improve this answer Follow

answered Jun 4, 2014 at 8:21

Tzen
1,426 ● 12 ● 11

There is a fairly common situation with a catch block that wraps the exception in additional context information. You really do want to wrap all exceptions in this manner, and catching only some of them leaves open the possibility that in the future there will be new exceptions that are missed. There is nothing dangerous at all with this, it is not a stick of dynamite. – AgilePro Apr 15 at 16:52

0

One thing with catching specific exception, is that the code can throw a different exception and the catch block will not be executed.

Example you are catching IOException but get StringOutOfBoundException. So, it depends. One thing to ensure is that you are catching every possible bad case inside the try{} statement. Multiple catches with catch exception as the last one. It really depends.

Share Improve this answer Follow

answered Aug 19, 2023 at 4:42

z atef
7,631 ● 3 ● 59 ● 54

If the catch block is designed to handle **all** exceptions, then it should catch the base class Exception. This is particularly in the case of a catch block that wraps the exception in additional context information. You really do want to wrap all exceptions in this manner, and catching only some of them leaves open the possibility that in the future there will be new exceptions that are missed. – AgilePro Apr 15 at 16:51

-1

For my IabManager class, which I used with in-app billing (from the TrivialDrive example online), I noticed sometimes I'd deal with a lot of exceptions. It got to the point where it was unpredictable.

I realized that, as long as I ceased the attempt at trying to consume an in-app product after one exception happens, which is where most of the exceptions would happen (in consume, as opposed to buy), I would be safe.

I just changed all the exceptions to a general exception, and now I don't have to worry about any other random, unpredictable exceptions being thrown.

Before:

```
    catch (final RemoteException exc)
    {
        exc.printStackTrace();
    }
    catch (final IntentSender.SendIntentException exc)
    {
        exc.printStackTrace();
    }
    catch (final IabHelper.IabAsyncInProgressException exc)
    {
        exc.printStackTrace();
    }
    catch (final NullPointerException exc)
    {
        exc.printStackTrace();
    }
    catch (final IllegalStateException exc)
    {
        exc.printStackTrace();
    }
```

After:

```
    catch (final Exception exc)
    {
        exc.printStackTrace();
    }
```

Share Improve this answer Follow

answered Nov 15, 2018 at 22:46

Peter Griffin
769 ● 9 ● 14

printing a stack trace and then continuing is a very bad example which should not be followed. – AgilePro Apr 15 at 16:53