Why are Exceptions not Checked in .NET?

Asked 16 years, 3 months ago Modified 2 years, 1 month ago

Viewed 11k times

43

I know Googling I can find an appropriate answer, but I prefer listening to your personal (and maybe technical) opinions.

What is the main reason of the difference between Java and C# in throwing exceptions?

In Java the signature of a method that throws an exception has to use the "throws" keyword, while in C# you don't know in compilation time if an exception could be thrown.

java .net exception

Share

Improve this question

Follow

edited Sep 26, 2008 at 20:06

Dima

39.3k ● 14 ● 78 ● 116

asked Sep 23, 2008 at 21:52

Enrico Murru

2,331 ● 4 ● 21 ● 24

9 Answers

Sorted by: Highest score (default)

100

In the article The Trouble with Checked Exceptions and in Anders Hejlsberg's (designer of the C# language) own voice, there are three main reasons for C# not supporting checked exceptions as they are found and verified in Java:

- **Neutral on Checked Exceptions**

> “C# is basically silent on the checked exceptions issue. Once a better solution is known—and trust me we continue to think about it—we can go back and actually put something in place.”

- **Versioning with Checked Exceptions**

> “Adding a new exception to a throws clause in a new version breaks client code. It's like adding a method to an interface. After you publish an interface, it is for all practical purposes immutable, …”

> “It is funny how people think that the important thing about exceptions is handling them. That is not the important thing about exceptions. In a well-written application there's a ratio of ten to one, in my opinion, of

> try finally to try catch. Or in C#, `using` statements, which are like try finally."

- **Scalability of Checked Exceptions**

> "In the small, checked exceptions are very enticing…The trouble begins when you start building big systems where you're talking to four or five different subsystems. Each subsystem throws four to ten exceptions. Now, each time you walk up the ladder of aggregation, you have this exponential hierarchy below you of exceptions you have to deal with. You end up having to declare 40 exceptions that you might throw.… It just balloons out of control."

In his article, "Why doesn't C# have exception specifications?", Anson Horton (Visual C# Program Manager) also lists the following reasons (see the article for details on each point):

- Versioning
- Productivity and code quality
- Impracticality of having class author differentiate between *checked* and *unchecked* exceptions
- Difficulty of determining the correct exceptions for interfaces.

It is interesting to note that C# does, nonetheless, support documentation of exceptions thrown by a given method via the `<exception>` tag and the compiler even takes the trouble to verify that the referenced exception type does indeed exist. There is, however, no check made at the call sites or usage of the method.

You may also want to look into the Exception Hunter, which is a commerical tool by Red Gate Software, that uses static analysis to determine and report exceptions thrown by a method and which may potentially go uncaught:

> Exception Hunter is a new analysis tool that finds and reports the set of possible exceptions your functions might throw – before you even ship. With it, you can locate unhandled exceptions easily and quickly, down to the line of code that is throwing the exceptions. Once you have the results, you can decide which exceptions need to be handled (with some exception handling code) before you release your application into the wild.

Finally, Bruce Eckel, author of Thinking in Java, has an article called, "Does Java need Checked Exceptions?", that may be worth reading up as well because the question of why checked exceptions are not there in C# usually takes root in comparisons to Java.

 edited Oct 24, 2022 at 6:37

Follow

answered Sep 24, 2008 at 9:04

Atif Aziz

36.6k ● 17 ● 66 ● 74

I am totally sold on *Versioning with Checked Exceptions* and *Versioning with Checked Exceptions* points. – TheVillageIdiot Dec 20, 2009 at 14:45

3 Interestingly, in my experience many developers tend to ignore the "Once a better solution is known"-part. Now in 2015, almost 12 years after that statement, I haven't seen any "better solution" in C# or any other language like Kotlin etc. (Or did I miss it? I'm not programming in C# currently anymore.) And just removing checked exceptions without a better solution makes thinks worse, IMHO, not better. We had a bad time in one of our C#-applications after some refactorings to verify if the exceptions get handled properly in all cases, because the compiler wouldn't tell. – Puce Jun 1, 2015 at 10:33

With versioning, the answer would just be to not throw a checked exception in a new version unless absolutely necessary. I would disagree and say Java is silent on checked exceptions and c# is opinionated. I.E. Java allows you to throw a checked exception or a runtime, but leaves the decision up to you, whereas c# forces you to throw a runtime exception. It sounds like his argument boils down to "checked exceptions are abused, so we won't let you use them". – TigerBear Oct 17, 2016 at 15:06

if IDE or compiler create warning on exceptions those are not handled with programmer it make software more robust and reduce run time bugs. – Navid_pdp11 Sep 10, 2017 at 9:24

43

Because the response to checked exceptions is almost always:

```
try {
  // exception throwing code
} catch(Exception e) {
   // either
   log.error("Error fooing bar",e);
   // OR
   throw new RuntimeException(e);
}
```

If you actually know that there is something you can do if a particular exception is thrown, then you can catch it and then handle it, but otherwise it's just incantations to appease the compiler.

Share Improve this answer

Follow

answered Sep 23, 2008 at 21:57

noah

21.5k ● 17 ● 67 ● 88

6 But really, it shouldn't be. A lot of exceptions aren't fatal. The main problem is that Checked probably shouldn't be the default exception type, and they're overused in Java. But Checked exceptions can still be useful. – Herms Sep 26, 2008 at 20:11

10 If the programmer is lazy, how is that Java's fault? If the programmer is overusing them, then that is bad API design, and again, how is that Java's fault? Bending a language backwards to appease laziness or stupidity is never a good idea in my book. I, for one, believe that checked exceptions are a useful TOOL, nothing more, nothing less – mcjabberz Sep 28, 2009 at 16:46

24 @mcjabberz the JDK overuses them. That is Java's fault. – noah Sep 30, 2009 at 17:49

Java is chock full of dogma. – Kevin Kostlan Apr 24, 2014 at 11:19

4 some time i wish C# create a warning if i donot catch a exception in my code. this will make life very easier. – Navid_pdp11 Sep 10, 2017 at 9:19

14

The basic design philosophy of C# is that actually catching exceptions is rarely useful, whereas cleaning up resources in exceptional situations is quite important. I think it's fair to say that `using` (the IDisposable pattern) is their answer to checked exceptions. See [1] for more.

1. http://www.artima.com/intv/handcuffs.html

Share Improve this answer

Follow

answered Sep 23, 2008 at 21:57

Derek Slager

13.8k • 3 • 36 • 34

12

By the time .NET was designed, Java had checked exceptions for quite some time and this feature was viewed by Java developers at best as ~~controversial~~ controversial. Thus .NET designers chose not to include it in C# language.

Share Improve this answer

Follow

edited Mar 11, 2016 at 21:47

answered Sep 23, 2008 at 22:08

Constantin

28.1k ● 10 ● 63 ● 79

looks like funny, but I think this is true and is the one of the main reason that C# design language team will never tell to us :) – hqt May 18, 2012 at 17:38

looks like the andersnoras.com link is broke – Amy B Sep 30, 2015 at 4:21

8

Fundamentally, whether an exception should be handled or not is a property of the *caller*, rather than of the function.

For example, in some programs there is no value in handling an IOException (consider ad hoc command-line utilities to perform data crunching; they're never going to be used by a "user", they're specialist tools used by specialist people). In some programs, there is value in handling an IOException at a point "near" to the call (perhaps if you get a FNFE for your config file you'll drop back to some defaults, or look in another location, or something of that nature). In other programs, you want it to bubble up a long way before it's handled (for example you might want it to abort until it reaches the UI, at which point it should alert the user that something has gone wrong.

Each of these cases is dependent on the *application*, and not the *library*. And yet, with checked exceptions, it is the

library that makes the decision. The Java IO library makes the decision that it will use checked exceptions (which strongly encourage handling that's local to the call) when in some programs a better strategy may be non-local handling, or no handling at all.

This shows the real flaw with checked exceptions in practice, and it's far more fundamental than the superficial (although also important) flaw that too many people will write stupid exception handlers just to make the compiler shut up. The problem I describe is an issue even when experienced, conscientious developers are writing the program.

edited Apr 30, 2015 at 9:05

gd1

11.4k ● 7 ● 51 ● 88

answered Sep 27, 2008 at 23:47

DrPizza

18.3k ● 7 ● 42 ● 53

1 +1. Checked-ness should be a function of the relationship between the call- and catch sites. Many methods specify that certain exceptions are used in lieu of a return when specific unexpected conditions occur. For example, `IDictionary.Add()` specifies that an implementation should throw `ArgumentException` if a key already exists. Unfortunately, there's no way to distinguish an exception which is thrown for that reason from an `ArgumentException` which is thrown by a method called by the `Dictionary.Add()` implementations under circumstances it wasn't expecting. – supercat Oct 29, 2012 at 20:04

1 If I were designing an exception system, I would have a few discrete (probably sealed) system-defined exception types, but not use exception types as the primary means of deciding what to catch. Instead, I would have the exception object wrap one or more objects of an abstract `ExceptionInfo` class. The exception thrown by `IDictionary.Add` could start out as an `ExpectedException` holding a `DuplicateKeyExceptionInfo`, but if it percolated through a layer which wasn't expecting it, it would turn into an `UnexpectedException` holding that same object.
– supercat Oct 29, 2012 at 20:11

Checked exceptions may be abused, but I would counter that it can be difficult to know which exceptions are throw by a function in c#. tempting a developers to do catch(Exception)... which is also a sin. If c# or Visualstudio enforced a way to see which exceptions may be thrown, then I would agree that unchecked is the way to go. – TigerBear Oct 17, 2016 at 15:16

Interestingly, the guys at Microsoft Research have added checked exceptions to Spec#, their superset of C#.

4

Share Improve this answer

Follow

edited Jul 9, 2011 at 18:48

Viktor Dahl

1,982 ● 3 ● 25 ● 36

answered Sep 23, 2008 at 22:15

Steve Morgan

13.1k ● 2 ● 43 ● 49

2 Since I don't have enough rep to edit the answer, here is a link to spec# info: research.microsoft.com/SpecSharp – Metro Sep 23, 2008 at 22:38

1

I went from Java to C# because of a job change. At first, I was a little concerned about the difference, but in practice, it hasn't made a difference.

Maybe, it's because I come from C++, which has the exception declaration, but it's not commonly used. I write every single line of code as if it could throw -- always use using around Disposable and think about cleanup I should do in finally.

In retrospect the propagation of the throws declaration in Java didn't really get me anything.

I would like a way to say that a function definitely never throws -- I think that would be more useful.

Share Improve this answer

Follow

answered Sep 23, 2008 at 21:58

Lou Franco

89k ● 14 ● 136 ● 198

2 There's a limited extent to which a function could say it **definitely** never throws - a low memory situation could cause an OutOfMemoryException pretty much anywhere, for example. Maybe a function could claim it never throws 'deliberately' - but how much use would this be? What could you usefully do with such a claim? – AakashM Jun 12, 2009 at 15:26

1

Additionally to the responses that were written already, not having checked exceptions helps you in many situations a lot. Checked exceptions make generics harder to implement and if you have read the closure proposals you will notice that every single closure proposal has to work around checked exceptions in a rather ugly way.

Share Improve this answer

Follow

answered Sep 23, 2008 at 22:06

Armin Ronacher

32.5k ● 14 ● 67 ● 69

1

I sometimes miss checked exceptions in C#/.NET.

I suppose besides Java no other notable platform has them. Maybe the .NET guys just went with the flow...

Share Improve this answer

Follow

answered Sep 23, 2008 at 22:42

Andrei Rînea

20.7k ● 18 ● 121 ● 169

1 The problem is that very few programmers actually use checked exceptions like they are designed. It is a case of ideal vs. practice. Ideally checked exceptions are very helpful, but in practice they mostly add extra work for the programmer. – Guvante Sep 23, 2008 at 22:56

3 Following ideal practices takes ideal programmers. – Constantin Sep 23, 2008 at 23:27

1 @Constantin: ...or a class or two in API design – mcjabberz Sep 28, 2009 at 16:49