Appropriate usage of assertions and exceptions

Asked 7 years, 8 months ago Modified 7 years, 8 months ago Viewed 235 times

I've read a bit around, trying to figure out when to use assertions and exceptions appropriately, but there's still something I'm missing to the big picture. Probably I just need more experience, so I'd like to bring some simple examples to better comprehend in what situations I should use use what.

Example 1: let us start with the classical situation of an invalid value. For example I have the following class, in which both fields must be positive:

```
class Rectangle{
    private int height;
    private int length;

    public int Height{
        get => height;
        set{
            //avoid to put negative heights
        }
    }
    //same thing for length
}
```

Let me remark that I am not talking about how to deal with user input in this example, since I could just make a simple control flow for that. Though, I am facing with the idea that somewhere else there may be some unexpected error and I want this to be detected since I don't want an object with invald values. So I can:

- Use `Debug.Assert` and just stop the program if that happens, so that I can correct possible errors when they come.
- Throw an `ArgumentOutOfRangeException` to basically do the same thing? This feels wrong, so I should use that only if I know I'm going to handle it somewhere. Though, **if I know where to handle the exception, shouldn't I fix the problem where it lies?** Or maybe it is meant for things that may happen but **you cannot control directly in your code**, like an user input (well that can be dealt with, without an exception, but maybe something else can't) or loading data?

Question: did I get the meaning of assertions and exceptions right? Also, could you please give an example in which *handling* exceptions can be useful (because *something you cannot control before* happens)? I cannot figure out what else, beyond the cases I mentioned, can happen, but I clearly still lack experience here.
To expand a bit my question: I can think of a variety of reasons why an exception can be thrown, like a `NullReferenceException`, an `IndexOutOfBoundsException`, the IO

exceptions like `DirectoryNotFoundException` or `FileNotFoundException`, etc. Though, I cannot figure out situations in which **handling** them becomes useful, apart from simply stopping the program (in which case, shouldn't an assertion be used?) or giving a simple message of where the problem has occured. I know even this is useful and exceptions are also meant to categorize "errors" and give clue to how to fix them. Though, **is a simple message really all they are useful to**? *That's sounds fishy*, so I'll stick with the "I've never faced a proper situation, 'cause of experience" mantra.

Example 2: let us now talk about the user input, using the first example. As I've anticipated, I won't use an exception just to check that the values are positive, as that's a simple control flow. But what happens if the user inputs a letter? Should I handle an exception here (maybe a simple `ArgumentException`) and give a message in the `catch` block? Or it can be avoided, too, through control flow (check if input is of type `int`, or something like that)?

Thanks to anyone who will clear my lingering doubts.

c# **debugging** **exception** **assert**

Share Improve this question Follow

asked Apr 24, 2017 at 13:04

Harnak
225 ● 1 ● 3 ● 12

1 There is no one correct answer, it entirely depends on the support system you have in place to deal with client programmers of your code getting it wrong and not knowing why. Users entering invalid data is not exceptional. – Hans Passant Apr 24, 2017 at 13:11

I know that highly depends on the situation, I'm just trying to figure out what kind of situations may arise. Also, sticking to this simple example, instead, if I mean to validate values just to detect errors in the code, I should use `Debug.Assert`. If I intend to implement save/load data and something in the loaded file may fail, I could catch it and stop the program with a "Corrupted data" message. Is this kind of reasoning right or I'm not getting the purpose of assertions and/or exceptions? – Harnak Apr 24, 2017 at 13:21

its an incomplete question. Deal with users by encouraging (or forcing) them to enter correct data. Stopping the program when it probably could just help the user along seems unnecessary – ferday Apr 24, 2017 at 13:34

1 A loaded file is more complex, since you can't control the source as much. – ferday Apr 24, 2017 at 13:58

1 That's primarily an opinion / situation. If it's a single divide by zero event, I would personally change the zero and inform the user that I did. If it's a complex corruption, then tell them the file is garbage. – ferday Apr 24, 2017 at 14:03

2 Answers

Sorted by: Highest score (default)

10

> Throw an ArgumentOutOfRangeException to basically do the same thing? This feels wrong, so I should use that only if I know I'm going to handle it somewhere. Though, if I know where to handle the exception, shouldn't I fix the problem where it lies?

Your reasoning is pretty good here, but not *quite* right. The reason you're struggling is because exceptions are used for *four* things in C#:

- **boneheaded exceptions**. A boneheaded exception is something like "invalid argument" *when the caller could have known that the argument is invalid*. If a boneheaded exception is thrown then the caller has a bug that should be fixed. You never have a `catch(InvalidArgumentException)` outside of a test case because it should never be thrown in production. These exceptions exist to help your callers write correct code by telling them very loudly when they've made a mistake.
- **vexing exceptions** are boneheaded exceptions where *the caller cannot know that the argument is invalid*. These are design flaws in APIs and should be eliminated. They require you to wrap API calls with try-catches to catch what looks like an exception that should be avoided, not caught. If you find that you're writing APIs that require the caller to wrap calls in a try-catch, you're doing something wrong.
- **fatal exceptions** are exceptions like thread aborted, out of memory, and so on. Something terrible has happened and the process cannot continue. There is very little point in catching these because there's not much you can do to improve the situation, and you might make it worse.
- **exogenous exceptions** are things like "the network cable is unplugged". You expected the network cable to be plugged in; it is not, and there is no way you could have checked earlier to see if it was, because the time of checking and the time of using are different times; the cable could be unplugged between those two times. You have to catch these.

Now that you know what the four kinds of exceptions are, you can see what the difference is between an exception and an assertion.

An assertion is something that *must logically be true always*, and if it is not, then you have a bug that should be fixed. You never *assert* that the network cable is plugged in. You never *assert* that a caller-supplied value is not null. There should *never* be a test case that causes an assertion to fire; if there is, then the test case has discovered a bug.

You assert that after your in-place sort algorithm runs, the smallest element in a non-empty array is at the beginning. There should be no way that can be false, and if there is, you have a bug. So assert that fact.

A `throw` by contrast is a statement, and *every statement should have a test case which exercises it*. "This API throws when passed null by a buggy caller" is part of its contract, and that contract should be testable. If you find you're writing throw statements that have no *possible* test case that verifies that they throw, consider changing it to an assertion.

And finally, never pass invalid arguments and then catch a boneheaded exception. If you're dealing with user input, then the UI layer should be verifying that the input is syntactically valid, ie, numbers where numbers are expected. The UI layer should not be passing possibly-unvetted user code to a deeper API and then handling the resulting exception.

Share

Improve this answer

Follow

edited Apr 24, 2017 at 14:37

answered Apr 24, 2017 at 14:18

Eric Lippert

659k ● 183 ● 1.3k ● 2.1k

> Excellent answer, that unraveled many doubts. That addressed exactly what I was looking for. Thanks! Just a last doubt on the unplugged cable example. I mean, I'm not sure I got it right, but I don't feel like you can do very much in a situation like that either, by catching it. Or can you? – Harnak Apr 24, 2017 at 14:43

2

I see it that way. Assertions are for programmers. Exceptions are for users. You can have places in your code that you expect specific value. Then you can put assertion, for example:

```
public int Age
{
    get { return age; }
    set
    {
        age = value;
        Debug.Assert(age == value);
    }
}
```

This is just example. So if age != value there is no exception. But "hey programmer, something strange may have happened, look at this part of code".

You use exception when application don't know how to react in a specific situation. For example:

```
public int Divide(int a, int b)
{
    Debug.Assert(b != 0); //this is for you as a programmer, but if something
bad happened, user won't see this assert, but application also doesn't know
what to do in situation like this, so you will add:

    if(b == 0)
        throw SomeException();
}
```

And SomeException might be handled somewhere else in your application.

Share Improve this answer Follow

answered Apr 24, 2017 at 13:30

Adam Jachocki

2,115 ● 1 ● 15 ● 30

The "Hey programmer" part is really explanatory. I think I got right the purpose of assertions then :) Thanks. Though, I still bear some doubts about handling exceptions. Let's say I have a method which loads some data and then calls Divide to make operations on that data (I use the load data or user input scenarios because they are the only situations I can think of, for now; and maybe **that's** the reason I still have doubts on when **handling** exceptions). I could handle an exception to give a message "Hey user, you put a wrong file there." if something goes wrong? – Harnak Apr 24, 2017 at 13:49

First you should think if you can handle this exception. For example if you divide x / 0 then maybe you could put some kind of default value as the result. But if you really don't know what to do in that "exceptional" situation, then you have to inform user that the file is corrupted or sth. But don't be mislead by what I've written about putting default value. Do not use exception handling to CHECK things. Exceptions are expensive and should be only used when an error occures. – Adam Jachocki Apr 25, 2017 at 7:26

I understand. Thanks for your explanation :) – Harnak Apr 25, 2017 at 9:22