IllegalArgumentException or NullPointerException for a null parameter? [closed]

I have a simple setter method for a property and `null` is not appropriate for this particular property. I have always been torn in this situation: should I throw an `IllegalArgumentException`, or a `NullPointerException`? From the javadocs, both seem appropriate. Is there some kind of an understood standard? Or is this just one of those things that you should do whatever you prefer and both are really correct?

java exception null nullpointerexception illegalargumentexception

edited Jan 17 '14 at 12:18
starsplusplus
924 2 14 29

asked Aug 6 '08 at 19:26

Mike Stone
32k 21 99 133

closed as primarily opinion-based by John Dvorak, Paul Stenne, xenteros, kazanaki, Paul Roub Sep 23 '16 at 13:34

Many good questions generate some degree of opinion based on expert experience, but answers to this question will tend to be almost entirely based on opinions, rather than facts, references, or specific expertise.

If this question can be reworded to fit the rules in the help center, please edit the question.

26 Answers

It seems like an `IllegalArgumentException` is called for if you don't want `null` to be an allowed value, and the `NullPointerException` would be thrown if you were trying to *use* a variable that turns out to be `null`.

edited May 14 '17 at 11:46
vaxquis

answered Aug 6 '08 at 19:29
Greg Hurlman

 **7,044** 5 34 53

 **15.2k** 6 46 81

30 The following answers to this question make persuasive arguments that NullPointerException is the correct exception: stackoverflow.com/a/8160/372926 ; stackoverflow.com/a/8196334/372926 ; and stackoverflow.com/a/6358/372926. – SamStephens Feb 6 '14 at 23:09

You should be using `IllegalArgumentException` (IAE), not `NullPointerException` (NPE) for the following reasons:

First, the NPE JavaDoc explicitly lists the cases where NPE is appropriate. Notice that all of them are thrown *by the runtime* when `null` is used inappropriately. In contrast, the IAE JavaDoc couldn't be more clear: "Thrown to indicate that a method has been passed an illegal or inappropriate argument." Yup, that's you!

Second, when you see an NPE in a stack trace, what do you assume? Probably that someone dereferenced a `null`. When you see IAE, you assume the caller of the method at the top of the stack passed in an illegal value. Again, the latter assumption is true, the former is misleading.

Third, since IAE is clearly designed for validating parameters, you have to assume it as the default choice of exception, so why would you choose NPE instead? Certainly not for different behavior -- do you really expect calling code to catch NPE's separately from IAE and do something different as a result? Are you trying to communicate a more specific error message? But you can do that in the exception message text anyway, as you should for all other incorrect parameters.

Fourth, all other incorrect parameter data will be IAE, so why not be consistent? Why is it that an illegal `null` is so special that it deserves a separate exception from all other types of illegal arguments?

Finally, I accept the argument given by other answers that parts of the Java API use NPE in this manner. However, the Java API is inconsistent with everything from exception types to naming conventions, so I think just blindly copying (your favorite part of) the Java API isn't a good enough argument to trump these other considerations.

edited Sep 15 '16 at 8:11

Toby Speight

14k 13 35 60

answered Sep 6 '08 at 18:29

Jason Cohen

56.5k 24 99 108

111 Effective Java 2nd Edition, Item 60: "Arguably, all erroneous method invocations boil down to an illegal argument or illegal state, but other exceptions are standardly used for certain kinds of illegal arguments and states. If a caller passes null in some parameter for which null values are prohibited, convention dictates that NullPointerException be thrown rather than IllegalArgumentException. Similarly, if a caller passes an out-of-range value in a parameter representing an index into a sequence, IndexOutOfBoundsException should be thrown rather than IllegalArgumentException." – Gili Dec 29 '10 at 20:15

32 The JavaDoc for NPE also states the following: "Applications should throw instances of this class to indicate other illegal uses of the null object." This one could be more clear :(– R. Martinho Fernandes Jan 25 '11 at 16:25

10 Unfortunately, the validation methods `Validate.notNull` (commons lang) and `Preconditions.checkNotNull` (guava) both throw NPE :-(– Aaron Digulla Oct 15 '12 at 15:43

2 Although Guava also has Preconditions.checkArgument() throws IllegalArgumentException ... – michaelok Dec 17 '13 at 0:02

12 @AaronDigulla, from the Guava docs: "We realize there are many valid arguments in favor of throwing IAE on a null argument. In fact, if we had a time machine to go back >15 years, we might even try to push things in that direction. However, we have decided to stick with the JDK and Effective Java preference of NullPointerException. If you're steadfast in your belief that IAE is right, you still have `checkArgument(arg != null)` , just without the convenience of it returning arg, or you can create a local utility to your project." code.google.com/p/guava-libraries/wiki/IdeaGraveyard – Arto Bendiken Oct 28 '14 at 23:07

The standard is to throw the NullPointerException. The generally infallible "Effective Java" discusses this briefly in Item 42 (first edition), Item 60 (second edition), or Item 72 (third edition) "Favor the use of standard exceptions":

> "Arguably, all erroneous method invocations boil down to an illegal argument or illegal state, but other exceptions are standardly used for certain kinds of illegal arguments and states. If a caller passes null in some parameter for which null values are prohibited, convention dictates that NullPointerException be thrown rather than IllegalArgumentException."

edited Jan 13 at 14:15

answered Aug 11 '08 at 20:05

GaryF

19.3k 7 48 69

86 I strongly disagree. NullPointerExceptions should only be thrown if the JVM accidentially follows a null reference. That is the difference that helps you when called in to look at code at 3 in the morning. – Thorbjørn Ravn Andersen May 29 '09 at 20:51

21 I don't necessarily agree with the standard (I could actually go either way on the issue), but that IS what the standard usage throughout the JDK is, hence Effective Java making the case for it. I think this is a case of choosing whether or not to follow the standard, or do the thing you feel is right. Unless you have a very good reason (and this certainly may qualify), it's best to follow standard practice. – GaryF Jun 8 '09 at 10:28

19 The exception trace shows the point of the exception, so if the difference in the type of the exception causes hell for you or is "the difference that helps you", you are doing something very wrong. – Jim Balter Jan 31 '13 at 23:34

6 Fantasies and sophistry. Just below, MB writes "Note that the arguments about hard debugging are bogus because you can of course provide a message to NullPointerException saying what was null and why it shouldn't be null. Just like with IllegalArgumentException." – Jim Balter Jul 7 '14 at 21:37

7 As the original answerer here, let me say it just doesn't matter that much. It certainly doesn't warrant 6 years of conversation. Pick one, whichever you like, and be consistent. The standard, as I pointed out, is NPE. If you prefer IAE for whatever reasons, go for it. Just be consistent. – GaryF Jul 8 '14 at 8:38

I was all in favour of throwing `IllegalArgumentException` for null parameters, until today, when I noticed the `java.util.Objects.requireNonNull` method in Java 7. With that method, instead of doing:

```
if (param == null) {
    throw new IllegalArgumentException("param cannot be null.");
}
```

you can do:

```
Objects.requireNonNull(param);
```

and it will throw a `NullPointerException` if the parameter you pass it is `null`.

Given that that method is right bang in the middle of `java.util` I take its existence to be a pretty strong indication that throwing `NullPointerException` is "the Java way of doing things".

I think I'm decided at any rate.

Note that the arguments about hard debugging are bogus because you can of course provide a message to `NullPointerException` saying what was null and why it shouldn't be null. Just like with `IllegalArgumentException`.

One added advantage of `NullPointerException` is that, in highly performance critical code, you could dispense with an explicit check for null (and a `NullPointerException` with a friendly error message), and just rely on the `NullPointerException` you'll get automatically when you call a method on the null parameter. Provided you call a method quickly (i.e. fail fast), then you have essentially the same effect, just not quite as user friendly for the developer. Most times it's probably better to check explicitly and throw with a useful message to indicate which parameter was null, but it's nice to have the option of changing that if performance dictates without breaking the published contract of the method/constructor.

edited Nov 10 '12 at 12:57

answered Nov 19 '11 at 18:42

MB.

5,002 5 33 38

13 Guava `Preconditions.checkNotNull(arg)` also throws NPE. – assylias Oct 14 '12 at 16:55

13 This isn't really adding more weight to NullPointerException for illegal null ARGUMENTS. Both the JDK requireNonNull() and Guava checkNotNull() can be called anywhere in the code, with any object. You could call them inside a loop for example. requireNotNull() and checkNotNull() could not possibly assume to be invoked with some method arguments and throw IllegalArgumentException. Note that Guava also has Preconditions.checkArgument() which does throw IllegalArgumentException. – Bogdan Calmac Mar 8 '13 at 18:37

2 A fair point by Bogdan, though I suspect the typical (and generally intended) use of requireNonNull is for argument checking. If you needed to check that something wasn't null in a loop I'd have thought an assertion would be the typical way. – MB. Aug 8 '13 at 14:41

13 I think the JavaDoc of Object.requireNonNull() adds weight to the argument: "This method is designed primarily for doing parameter validation in methods and constructors" – Richard Zschech Feb 2 '15 at 14:42

Keep in mind that the performance argument in favor of implicit null checks is often invalid. The JIT is able to recognize user null checks and elide the following implied null check, and/or use the same set of optimizations on either type of null check. See this wiki for more info, in particular: *User-written null checks are in most cases functionally identical to those inserted by the JVM.* Of course, if you are doing something fancier in your null like custom messages, this optimization may not apply. – BeeOnRope Nov 2 '16 at 23:17

I tend to follow the design of JDK libraries, especially Collections and Concurrency (Joshua Bloch, Doug Lea, those guys know how to design solid APIs). Anyway, many APIs in the JDK pro-actively throws `NullPointerException` .

For example, the Javadoc for `Map.containsKey` states:

> @throws NullPointerException if the key is null and this map does not permit null keys (optional).

It's perfectly valid to throw your own NPE. The convention is to include the parameter name which was null in the message of the exception.

The pattern goes:

```
public void someMethod(Object mustNotBeNull) {
    if (mustNotBeNull == null) {
        throw new NullPointerException("mustNotBeNull must not be null");
    }
}
```

Whatever you do, don't allow a bad value to get set and throw an exception later when other code attempts to use it. That makes debugging a nightmare. You should always the follow the "fail-fast" principle.

edited Nov 3 '15 at 21:39

mkobit
17.3k 6 70 88

answered Aug 8 '08 at 20:35

Mark Renouf
21.4k 18 82 116

3 Food for thought: Maybe the reason that NullPointerException doesn't extend IllegalArgumentException is that the former can occur in cases not involving method arguments. – Gili Dec 29 '10 at 20:11

2 @Gili: maybe this problem exists in the first place because Java does not support multiple inheritance. If Java supports MI, you'd be able to throw an exception that inherits from both IllegalArgumentException and NullPointerException. – Lie Ryan Mar 1 '13 at 14:29

7 The message needs not to include the argument, since it would be always null, giving: "null must not be null", not very useful. :-) Otherwise, I agree, you can make a "rich" NPE, with a meaningful message. – PhiLho Jul 22 '13 at 15:21

4 I have to agree - follow the standard API when in doubt. Not everything in the API is optimal mind you, but still it's maintained and iterated by hundreds of developers, and used by millions of programmers. Now in Java 7 we have another example of the NPE being used in this way; the Objects.requireNonNull(T obj) method - clearly specified for checking that object references are non-null, clearly specified for doing parameter validation in methods/constructors, and clearly specified to throw an NPE if the object is null. End of – flamming_python Jun 30 '14 at 17:48

Voted up Jason Cohen's argument because it was well presented. Let me dismember it step by step. ;-)

- The NPE JavaDoc explicitly says, *"other illegal uses of the null object"*. If it was just limited to situations where the runtime encounters a null when it shouldn't, all such cases could be defined far more succinctly.

- Can't help it if you assume the wrong thing, but assuming encapsulation is applied properly, you really shouldn't care or notice whether a null was dereferenced inappropriately vs. whether a method detected an inappropriate null and fired an exception off.
- I'd choose NPE over IAE for multiple reasons
  - It is more specific about the nature of the illegal operation
  - Logic that mistakenly allows nulls tends to be very different from logic that mistakenly allows illegal values. For example, if I'm validating data entered by a user, if I get value that is unacceptable, the source of that error is with the end user of the application. If I get a null, that's programmer error.
  - Invalid values can cause things like stack overflows, out of memory errors, parsing exceptions, etc. Indeed, most errors generally present, at some point, as an invalid value in some method call. For this reason I see IAE as actually the *MOST GENERAL* of all exceptions under RuntimeException.
- Actually, other invalid arguments can result in all kinds of other exceptions. UnknownHostException, FileNotFoundException, a variety of syntax error exceptions, IndexOutOfBoundsException, authentication failures, etc., etc.

In general, I feel NPE is much maligned because traditionally has been associated with code that fails to follow the fail fast principle. That, plus the JDK's failure to populate NPE's with a message string really has created a strong negative sentiment that isn't well founded. Indeed, the difference between NPE and IAE from a runtime perspective is strictly the name. From that perspective, the more precise you are with the name, the more clarity you give to the caller.

edited Jul 11 '12 at 13:53

Rob Kielty
5,996 5 29 46

answered Dec 9 '09 at 6:07

Christopher Smith
4,142 1 24 18

The difference between most unchecked exceptions is just the name. – Thorbjørn Ravn Andersen Jul 7 '14 at 2:33

It's a "Holy War" style question. In others words, both alternatives are good, but people will have their preferences which they will defend to the death.

answered Oct 17 '08 at 13:31

Steve McLeod
31.5k 34 108 162

If it's a `setter` method and `null` is being passed to it, I think it would make more sense to throw an `IllegalArgumentException`. A `NullPointerException` seems to make more sense in the case where you're attempting to actually use the `null`.

So, if you're using it and it's `null`, `NullPointer`. If it's being passed in and it's `null`, `IllegalArgument`.

edited Aug 30 '16 at 10:11

prograshid
185 4 23

answered Aug 6 '08 at 19:31

Jeremy Privett
3,418 2 26 35

Apache Commons Lang has a NullArgumentException that does a number of the things discussed here: it extends IllegalArgumentException and its sole constructor takes the name of the argument which should have been non-null.

While I feel that throwing something like a NullArgumentException or IllegalArgumentException more accurately describes the exceptional circumstances, my colleagues and I have chosen to defer to Bloch's advice on the subject.

answered Jul 2 '09 at 4:47

Brian T. Grant
159 1 2

5 Note they removed it from commons-lang3: apache-commons.680414.n4.nabble.com/… – artbristol Sep 18 '13 at 10:47

Couldn't agree more with what's being said. Fail early, fail fast. Pretty good Exception mantra.

The question about which Exception to throw is mostly a matter of personal taste. In my mind IllegalArgumentException seems more specific than using a NPE since it's telling me that the problem was with an argument I passed to the method and not with a value that may have been generated while performing the method.

My 2 Cents

answered Aug 22 '08 at 11:11

Allain Lalonde
48.4k 63 157 226

The accepted practice if to use the *IllegalArgumentException(String message)* to declare a parameter to be invalid and give as much detail as possible... So to say that a parameters was found to be null while exception non-null, you would do something like this:

```
if( variable == null )
    throw new IllegalArgumentException("The object 'variable' cannot be null");
```

You have virtually no reason to implicitly use the "NullPointerException". The NullPointerException is an exception thrown by the Java Virtual Machine when you try to execute code on null reference (Like *toString()*).

answered Aug 11 '08 at 19:43

Claude Houle
20.3k 8 25 40

Actually, the question of throwing IllegalArgumentException or NullPointerException is in my humble view only a "holy war" for a minority with an incomlete understanding of exception handling in Java. In general, the rules are simple, and as follows:

- argument constraint violations must be indicated as fast as possible (-> fast fail), in order to avoid illegal states which are much harder to debug
- in case of an invalid null pointer for whatever reason, throw NullPointerException
- in case of an illegal array/collection index, throw ArrayIndexOutOfBounds
- in case of a negative array/collection size, throw NegativeArraySizeException
- in case of an illegal argument that is not covered by the above, and for which you don't have another more specific exception type, throw IllegalArgumentException as a wastebasket
- on the other hand, in case of a constraint violation WITHIN A FIELD that could not be avoided by fast fail for some valid reason, catch and rethrow as IllegalStateException or a more specific checked exception. Never let pass the original NullPointerException, ArrayIndexOutOfBounds, etc in this case!

There are at least three very good reasons against the case of mapping all kinds of argument constraint violations to IllegalArgumentException, with the third probably being so severe as to mark the practice bad style:

(1) A programmer cannot a safely assume that all cases of argument constraint violations result in IllegalArgumentException, because the large majority of standard classes use this exception rather as a wastebasket if there is no more specific kind of exception available. Trying to map all cases of argument constraint violations to IllegalArgumentException in your API only leads to programmer frustration using your classes, as the standard libraries mostly follow different rules that violate yours, and most of your API users will use them as well!

(2) Mapping the exceptions actually results in a different kind of anomaly, caused by single inheritance: All Java exceptions are classes, and therefore support single inheritance only. Therefore, there is no way to create an exception that is truly say both a NullPointerException and an IllegalArgumentException, as subclasses can only inherit from one or the other. Throwing an IllegalArgumentException in case of a null argument therefore makes it harder for API users to distinguish between problems whenever a program tries to programmatically correct the problem, for example by feeding default values into a call repeat!

(3) Mapping actually creates the danger of bug masking: In order to map argument constraint violations into IllegalArgumentException, you'll need to code an outer try-catch within every method that has any constrained arguments. However, simply catching RuntimeException in this catch block is out of the question, because that risks mapping documented RuntimeExceptions thrown by libery methods used within yours into IllegalArgumentException, even if they are no caused by argument constraint violations. So you need to be very specific, but even that effort doesn't protect you from the case that you accidentally map an undocumented runtime exception of another API (i.e. a bug) into an IllegalArgumentException of your API. Even the most careful mapping therefore risks masking programming errors of other library makers as argument constraint violations of your method's users, which is simply hillareous behavior!

With the standard practice on the other hand, the rules stay simple, and exception causes stay unmasked and specific. For the method caller, the rules are easy as well: - if you encounter a documented runtime exception of any kind because you passed an illegal value, either repeat the call with a default (for this specific exceptions are neccessary), or correct your code - if on the other hand you enccounter a runtime exception that is not documented to happen for a given set of arguments, file a bug report to the method's makers to ensure that either their code or their documentation is fixed.

answered Dec 31 '12 at 17:44

Sascha Baumeister

172 2 2

Throwing an exception that's exclusive to `null` arguments (whether `NullPointerException` or a custom type) makes automated `null` testing more reliable. This automated testing can be done with reflection and a set of default values, as in Guava's `NullPointerTester`. For example, `NullPointerTester` would attempt to call the following method...

```
Foo(String string, List<?> list) {
  checkArgument(string.length() > 0);
  // missing null check for list!
  this.string = string;
  this.list = list;
}
```

...with two lists of arguments: `""`, `null` and `null`, `ImmutableList.of()`. It would test that each of these calls throws the expected `NullPointerException`. For this implementation, passing a `null` list does *not* produce `NullPointerException`. It does, however, happen to produce an `IllegalArgumentException` because `NullPointerTester` happens to use a default string of `""`. If `NullPointerTester` expects only `NullPointerException` for `null` values, it catches the bug. If it expects `IllegalArgumentException`, it misses it.

answered Nov 9 '12 at 16:30

Chris Povirk

2,573 16 34

In general, a developer should **never** throw a NullPointerException. This exception is thrown by the runtime when code attempts to dereference a variable who's value is null. Therefore, if your method wants to explicitly disallow null, as opposed to just happening to have a null value raise a NullPointerException, you should throw an IllegalArgumentException.

answered Sep 5 '08 at 10:34

Martin OConnor

---

7 JavaDoc on NPE has another opinion: "Applications should throw instances of this class to indicate other illegal uses of the null object.". Don't be so categorical – Donz Nov 9 '10 at 8:56

I wanted to single out Null arguments from other illegal arguments, so I derived an exception from IAE named NullArgumentException. Without even needing to read the exception message, I know that a null argument was passed into a method and by reading the message, I find out which argument was null. I still catch the NullArgumentException with an IAE handler, but in my logs is where I can see the difference quickly.

answered Jan 17 '09 at 0:13

Jason Fritcher

1,355 8 12

I have adopted the "throw new IllegalArgumentException("foo == null")" approach. You need to log the variable name anyway (to be certain that you are looking at the right sttatement etc) – Thorbjørn Ravn Andersen May 29 '09 at 21:05

---

the dichotomy... Are they non-overlapping? Only non-overlapping parts of a whole can make a dichotomy. As i see it:

```
throw new IllegalArgumentException(new
NullPointerException(NULL_ARGUMENT_IN_METHOD_BAD_BOY_BAD));
```

edited May 13 '12 at 0:14

answered Apr 17 '12 at 17:15

Luis Daniel Mesa Velasquez

72 5

1 This would double the overhead for exception creation and wouldn't really help as catching `NullPointerException` would do nothing. The only thing which could help is `IllegalNullPointerArgumentException extends IllegalArgumentException, NullPointerException` , but we have no multiple inheritance. – maaartinus Jan 30 '14 at 11:31

I believe that more specific exceptions should be wrapped by more general exceptions. NPE is for an expression while IAE is for a method. Since methods contain statements that contain expressions, IAE is more general. – Sgene9 Jan 26 at 16:46

Regarding the overhead, I have no idea. But since the stacktraces would basically be identical except that the name of the exception changed in the middle, I think there shouldn't be too much overhead for double exceptions. If one is worried about the overhead, he can use "if" statements to return a null or -1 instead of throwing an exception. – Sgene9 Jan 26 at 16:53

Some collections assume that `null` is rejected using `NullPointerException` rather than `IllegalArgumentException`. For example, if you compare a set containing `null` to a set that rejects `null`, the first set will call `containsAll` on the other and catch its `NullPointerException` -- but not `IllegalArgumentException`. (I'm looking at the implementation of `AbstractSet.equals`.)

You could reasonably argue that using unchecked exceptions in this way is an antipattern, that comparing collections that contain `null` to collections that can't contain `null` is a likely bug that really *should* produce an exception, or that putting `null` in a collection at all is a bad idea. Nevertheless, unless you're willing to say that `equals` should throw an exception in such a case, you're stuck remembering that `NullPointerException` is required in certain circumstances but not in others. ("IAE before NPE except after 'c'...")

edited Nov 9 '12 at 16:30

answered Nov 9 '12 at 16:18

Chris Povirk

2,573 16 34

I don't see how the contains throws a NPE depending on an element inside the collection. Only reason a NPE is thrown (afaict) is if the collection itself is null (in which case the NPE is thrown because it tries to access it's iterator). This does however raise the question if null input should be checked for or if it should propagate through until it's tried to access. – Alowaniak Jul 8 '16 at 12:27

1 `new TreeSet<>().containsAll(Arrays.asList((Object) null));` throws `NPE` because the `List` contains `null`. – Chris Povirk Jul 8 '16 at 14:13

Ah indeed, TreeSet#contains throws NPE "if the specified element is null and this set uses natural ordering, or its comparator does not permit null elements". Only looked at AbstractSet which does allow null, my bad. Personally I find it weird it doesn't just return false though, since in that case there can't possibly be a null added. – Alowaniak Jul 12 '16 at 15:16

NullPointerException thrown when attempting to access an object with a reference variable whose current value is null

IllegalArgumentException thrown when a method receives an argument formatted differently than the method expects

answered Jun 23 '15 at 13:51

Nitesh Soomani

416 3 11

According to your scenario, `IllegalArgumentException` is the best pick, because `null` is not a valid value for your property.

edited Jun 5 '15 at 12:46

lucian.pantelimon
2,933 3 21 44

answered Jun 5 '15 at 9:43

leo
61 2

As an subjective question this should be closed, but as its still open:

This is part of the the internal policy used at my previous place of employment and it worked really well. This is all from memory so I can't remember the exact wording. Its worth noting that they did not use checked exceptions, but that is beyond the scope of the question. The unchecked exceptions they did use fell into 3 main categories.

NullPointerException: Do not throw intentionally. NPEs are to be thrown only by the VM when dereferencing a null reference. All possible effort is to be made to ensure that these are never thrown. @Nullable and @NotNull should be used in conjunction with code analysis tools to find these errors.

IllegalArgumentException: Thrown when an argument to a function does not conform to the public documentation, such that the error can be identified and described in terms of the arguments passed in. The OP's situation would fall into this category.

IllegalStateException: Thrown when a function is called and its arguments are either unexpected at the time they are passed or incompatible with the state of the object the method is a member of.

For example, there were two internal versions of the IndexOutOfBoundsException used in things that had a length. One a sub-class of IllegalStateException, used if the index was larger than the length. The other a subclass of IllegalArgumentException, used if the index was negative. This was because you could add more items to the object and the argument would be valid, while a negative number is never valid.

As I said, this system works really well, and it took someone to explain why the distinction is there: "Depending on the type of error it is quite straight forward for you to figure out what to do. Even if you can't actually figure out what went wrong you can figure out where to catch that error and create additional debugging information."

NullPointerException: Handle the Null case or put in an assertion so that the NPE is not thrown. If you put in an assertion it must one of the other two types. If possible, continue debugging as if the assertion was there in the first place.

IllegalArgumentException: you have something wrong at your callsite. If the values being passed in are from another function, find out why you are receiving an incorrect value. If you are passing in one of your arguments propagate the error checks up the call stack until you find the function that is not returning what you expect.

IllegalStateException: You have not called your functions in the correct order. If you are using one of your arguments, check them and throw an IllegalArgumentException describing the issue. You can then propagate the cheeks up the stack until you find the issue.

Anyway, his point was that you can only copy the IllegalArgumentAssertions up the stack. There is no way for you to propagate the IllegalStateExceptions or NullPointerExceptions up the stack because they had something to do with your function.

answered Sep 23 '16 at 10:07

Ben Seidel

109 3

Ideally runtime exceptions should not be thrown. A checked exception(business exception) should be created for your scenario. Because if either of these exception is thrown and logged, it misguides the developer while going through the logs. Instead business exceptions do not create that panic and usually ignored while troubleshooting logs.

answered Aug 18 '16 at 8:40

vijay

19 3

The definitions from the links to the two exceptions above are IllegalArgumentException: Thrown to indicate that a method has been passed an illegal or inappropriate argument. NullPointerException: Thrown when an application attempts to use null in a case where an object is required.

The big difference here is the IllegalArgumentException is supposed to be used when checking that an argument to a method is valid. NullPointerException is supposed to be used whenever an object being

"used" when it is null.

I hope that helps put the two in perspective.

answered Aug 16 '08 at 18:13

martinatime
2,147 1 13 19

1 The salient bit is that it is the *application* that is using null, not the runtime. So there is a fairly large overlap between "when a method has been passed an illegal or inappropriate argument" and "when an application is using null". In theory, if an app passes a null for a field that requires non-null, both criteria are being met. – Christopher Smith Dec 9 '09 at 6:12

If it's a "setter", or somewhere I'm getting a member to use later, I tend to use IllegalArgumentException.

If it's something I'm going to use (dereference) right now in the method, I throw a NullPointerException proactively. I like this better than letting the runtime do it, because I can provide a helpful message (seems like the runtime could do this too, but that's a rant for another day).

If I'm overriding a method, I use whatever the overridden method uses.

answered Aug 28 '08 at 18:03

erickson
212k 42 318 416

You should throw an IllegalArgumentException, as it will make it obvious to the programmer that he has done something invalid. Developers are so used to seeing NPE thrown by the VM, that any programmer would not immediately realize his error, and would start looking around randomly, or worse, blame your code for being 'buggy'.

answered Sep 9 '08 at 4:13

Will Sargent
3,689 1 22 43

3 Sorry, if a programmer looks around "randomly" upon getting any kind of exception... changing the name of an exception isn't going to help much. – Christopher Smith Dec 9 '09 at 6:09

In this case, IllegalArgumentException conveys clear information to the user using your API that the " should not be null". As other forum users pointed out you could use NPE if you want to as long as you convey the right information to the user using your API.

GaryF and tweakt dropped "Effective Java" (which I swear by) references which recommends using NPE. And looking at how other good APIs are constructed is the best way to see how to construct your API.

Another good example is to look at the Spring APIs. For example, org.springframework.beans.BeanUtils.instantiateClass(Constructor ctor, Object[] args) has a Assert.notNull(ctor, "Constructor must not be null") line. org.springframework.util.Assert.notNull(Object object, String message) method checks to see if the argument (object) passed in is null and if it is it throws a new IllegalArgumentException(message) which is then caught in the org.springframework.beans.BeanUtils.instantiateClass(...) method.

answered Oct 29 '08 at 16:20

If you choose to throw a NPE and you are using the argument in your method, it might be redundant and expensive to explicitly check for a null. I think the VM already does that for you.

answered Aug 11 '08 at 13:20

jassuncao
3,661 1 23 34

The run time wont include a meaningful msg. – mP. Mar 31 '12 at 9:03

Actually this comment could derive some other opinion. Let the VM speak in `NPE` yet the programmers speak in `IAE` before the VM if they want to. – Jin Kwon Jul 13 '12 at 2:22

1 Expensive? I don't think that == null is that expensive... Beside, the null argument can be just stored for latter use, and will throw an exception long after the method call, making the error harder to track. Or you can create an expensive object before using the null argument, and so on. Early detection seems to be a good option. – PhiLho Jul 22 '13 at 15:26

down voting of this answer is misunderstanding of the hardware behind. Certainly the hardware checks (that CPU

does) are cheaper that explicit checking. Dereferencing of null is a SegmentationFault (SegV) special case (access to a page not owned by the process) which CPU/OS checks and JRE handles as a special case throwing NullPointerException. – digital_infinity Aug 11 '17 at 8:31