Begin, Rescue and Ensure in Ruby?

Asked 14 years, 10 months ago Modified 5 months ago Viewed 522k times

632

I was wondering if `ensure` was the Ruby equivalent of `finally` in C#? Should I have:

```
file = File.open("myFile.txt", "w")

begin
  file << "#{content} \n"
rescue
  # handle the error here
ensure
  file.close unless file.nil?
end
```

or should I do this?

```
#store the file
file = File.open("myFile.txt", "w")

begin
  file << "#{content} \n"
  file.close
rescue
  # handle the error here
ensure
  file.close unless file.nil?
end
```

Does `ensure` get called no matter what even if an exception isn't raised?

ruby exception error-handling

Share
Improve this question
Follow

edited Jun 26 at 16:29
Nikita Fedyashev
18.9k ● 13 ● 56 ● 103

asked Feb 3, 2010 at 11:54
Lloyd Powell
18.8k ● 18 ● 90 ● 126

3 Neither is good. As a rule, when dealing with external resources, you **always** want the resource opening` to be inside the `begin` block. – Nowaker Oct 4, 2018 at 17:38

7 Answers

Sorted by: Highest score (default)

1343

Yes, `ensure` ensures that the code is always evaluated. That's why it's called `ensure`. So, it is equivalent to Java's and C#'s `finally`.

The general flow of `begin`/`rescue`/`else`/`ensure`/`end` looks like this:

```
begin
  # something which might raise an exception
rescue SomeExceptionClass => some_variable
  # code that deals with some exception
rescue SomeOtherException => some_other_variable
  # code that deals with some other exception
else
  # code that runs only if *no* exception was raised
ensure
  # ensure that this code always runs, no matter what
  # does not change the final value of the block
end
```

You can leave out `rescue`, `ensure` or `else`. You can also leave out the variables in which case you won't be able to inspect the exception in your exception handling code. (Well, you can always use the global exception variable to access the last exception that was raised, but that's a little bit hacky.) And you can leave out the exception class, in which case all exceptions that inherit from `StandardError` will be caught. (Please note that this does not mean that *all* exceptions are caught, because there are exceptions which are instances of `Exception` but not `StandardError`. Mostly very severe exceptions that compromise the integrity of the program such as `SystemStackError`, `NoMemoryError`, `SecurityError`, `NotImplementedError`, `LoadError`, `SyntaxError`, `ScriptError`, `Interrupt`, `SignalException` or `SystemExit`.)

Some blocks form implicit exception blocks. For example, method definitions are implicitly also exception blocks, so instead of writing

```
def foo
  begin
    # ...
  rescue
    # ...
  end
end
```

you write just

```
def foo
  # ...
rescue
  # ...
end
```

or

```
def foo
  # ...
ensure
  # ...
end
```

The same applies to `class` definitions and `module` definitions.

However, in the specific case you are asking about, there is actually a much better idiom. In general, when you work with some resource which you need to clean up at the end, you do that by passing a block to a method which does all the cleanup for you. It's similar to a `using` block in C#, except that Ruby is actually powerful enough that you don't have to wait for the high priests of Microsoft to come down from the mountain and graciously change their compiler for you. In Ruby, you can just implement it yourself:

```
# This is what you want to do:
File.open('myFile.txt', 'w') do |file|
  file.puts content
end

# And this is how you might implement it:
def File.open(filename, mode='r', perm=nil, opt=nil)
  yield filehandle = new(filename, mode, perm, opt)
ensure
  filehandle&.close
end
```

And what do you know: this is *already* available in the core library as `File.open`. But it is a general pattern that you can use in your own code as well, for implementing any kind of resource cleanup (à la `using` in C#) or transactions or whatever else you might think of.

The only case where this doesn't work, if acquiring and releasing the resource are distributed over different parts of the program. But if it is localized, as in your example, then you can easily use these resource blocks.

BTW: in modern C#, `using` is actually superfluous, because you can implement Ruby-style resource blocks yourself:

```
class File
{
    static T open<T>(string filename, string mode, Func<File, T> block)
    {
        var handle = new File(filename, mode);
        try
```

```
            {
                return block(handle);
            }
            finally
            {
                handle.Dispose();
            }
        }
    }

    // Usage:

    File.open("myFile.txt", "w", (file) =>
    {
        file.WriteLine(contents);
    });
```

Share

Improve this answer

Follow

edited Jul 28, 2023 at 2:00

answered Feb 3, 2010 at 13:04

Jörg W Mittag
369k ● 79 ● 453 ● 661

107 Note that, although the `ensure` statements are executed last, they are not the return value. – Chris Dec 11, 2013 at 18:47

47 I love seeing rich contributions like this on SO. It goes above and beyond what the OP asked such that it applies to many more developers, yet is still on topic. I learned a few things from this answer + edits. Thank you for not just writing "Yes, `ensure` gets called no matter what." – Dennis Oct 17, 2014 at 11:11

6 Note, that ensure is NOT guaranteed to complete. Take the case of where you have a begin/ensure/end inside of a thread, and then you call Thread.kill when the first line of the ensure block is being called. This will cause the rest of the ensure to not execute. – Teddy Jan 6, 2015 at 14:47

7 @Teddy: ensure is guaranteed to begin executing, not guaranteed to complete. Your example is overkill - a simple exception inside the ensure block will cause it to exit as well. – Martin Konecny May 4, 2015 at 23:35

3 Please note that the C# example doesn't remove the need for `using`. The `open` method still needs to do cleanup. The example just does this the verbose (and not 100% bullet-proof) way instead of using the `using` shorthand. I recommend `using` whenever possible in place of `try-finally`. – Mashmagar Aug 19, 2015 at 12:55

50

FYI, even if an exception is re-raised in the `rescue` section, the `ensure` block will be executed before the code execution continues to the next exception handler. For instance:

```
begin
  raise "Error!!"
rescue
  puts "test1"
  raise # Reraise exception
```

```
ensure
  puts "Ensure block"
end
```

Share

Improve this answer

Follow

edited Sep 25, 2015 at 23:24
the Tin Man
160k ● 44 ● 221 ● 306

answered Oct 29, 2012 at 12:18
alup
2,981 ● 1 ● 23 ● 12

15

If you want to ensure a file is closed you should use the block form of `File.open`:

```
File.open("myFile.txt", "w") do |file|
  begin
    file << "#{content} \n"
  rescue
  #handle the error here
  end
end
```

Share

Improve this answer

Follow

edited Sep 25, 2015 at 23:25
the Tin Man
160k ● 44 ● 221 ● 306

answered Feb 3, 2010 at 12:08
Farrel
2,381 ● 18 ● 14

3 I guess if you don't want to handle the error but just raise it, and close the file handle, you don't need the begin rescue here? – rogerdpack Feb 1, 2013 at 21:34

9

This is why we need `ensure`:

```
def hoge
  begin
    raise
  rescue
    raise # raise again
  ensure
    puts 'ensure' # will be executed
  end
  puts 'end of func' # never be executed
end
```

Share

Improve this answer

Follow

edited Sep 25, 2015 at 23:29
the Tin Man
160k ● 44 ● 221 ● 306

answered Jan 23, 2014 at 7:27
kuboon
10.2k ● 3 ● 45 ● 33

Yes, `ensure` is called in any circumstances. For more information see "[Exceptions, Catch, and Throw]" of the Programming Ruby book and search for "ensure".

7

Share

Improve this answer

Follow

edited Sep 25, 2015 at 23:27

the Tin Man
160k ● 44 ● 221 ● 306

answered Feb 3, 2010 at 11:57

Milan Novota
15.6k ● 7 ● 56 ● 62

5

Yes, `ensure` ENSURES it is run every time, so you don't need the `file.close` in the `begin` block.

By the way, a good way to test is to do:

```
begin
  # Raise an error here
  raise "Error!!"
rescue
  #handle the error here
ensure
  p "=========inside ensure block"
end
```

You can test to see if "=========inside ensure block" will be printed out when there is an exception. Then you can comment out the statement that raises the error and see if the `ensure` statement is executed by seeing if anything gets printed out.

Share

Improve this answer

Follow

edited Sep 25, 2015 at 23:29

the Tin Man
160k ● 44 ● 221 ● 306

answered Feb 3, 2010 at 12:10

Aaron Qian
4,485 ● 2 ● 25 ● 27

5

Yes, `ensure` like `finally` **guarantees that the block will be executed**. This is very useful for making sure that critical resources are protected e.g. closing a file handle on error, or releasing a mutex.

Share

Improve this answer

Follow

edited Oct 2, 2015 at 14:30

answered Feb 3, 2010 at 12:09

Chris McCauley
26.3k ● 9 ● 50 ● 68

1 Except in his/her case, there's no guarantee for the file to be closed, because `File.open` part is NOT inside the begin-ensure block. Only `file.close` is but it's not enough.
– Nowaker Oct 4, 2018 at 17:39