Defensive Programming

Writing fault-tolerant Ruby

Before we begin...

- Hi, I'm Johnny Boursiquot
- Audience: Beginner-ish

What We'll Cover

- Definition
- Purpose
- Basics
- Example

Definition

Defensive programming is a form of defensive design intended to ensure the continuing function of a piece of software in spite of unforeseeable usage of said software. (Wikipedia)

Purpose

Resiliency

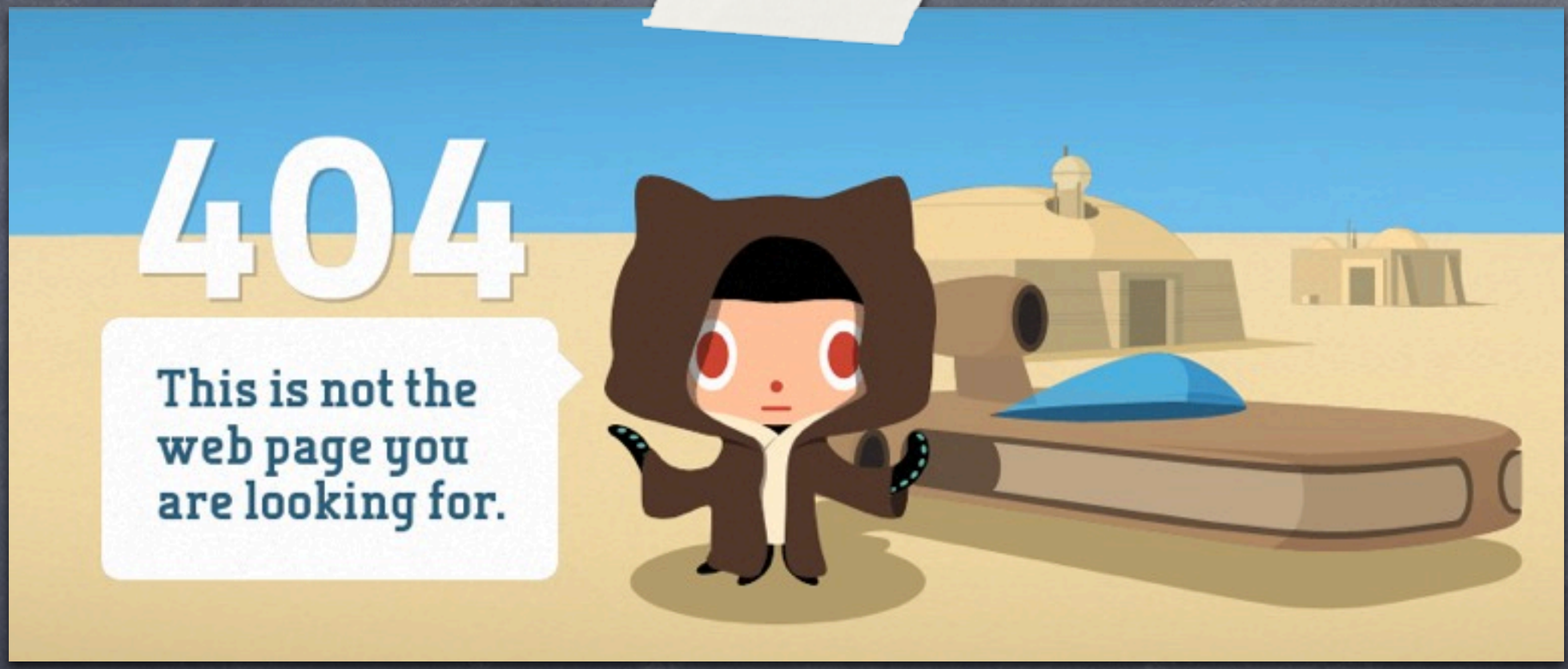
Fault-tolerance

Predictability

Sooner or later...

...failure happens

When failure is anticipated...


404
This is not the web page you are looking for.

When failure is not anticipated...

Doing it (in Ruby)

Basic Concepts

- Exception class and subclasses NoMemoryError, LoadError, NotImplementedError, SignalException, Interrupt, ScriptError
- Those tend to be unrecoverable

Basic Concepts

- StandardError and its subclasses ArgumentError, EncodingError, FiberError, IOError, IndexError, LocalJumpError, NameError, RangeError, RegexpError, RuntimeError, SystemCallError, ThreadError, TypeError, ZeroDivisionError
- You will encounter those more often

Basic Concepts

- Your custom error classes that inherit from StandardError
- They can provide as much granularity as you need

Raising Exceptions

- Exceptions are raised with `raise`
  - aka `Kernel.raise`
  - aka `Kernel.fail`
- `raise` or `fail` is a matter of preference, use whichever makes sense given the context
- `raise [class], [message], [backtrace]`

Raising Exceptions

```
irb> raise TypeError
TypeError: TypeError
  from (irb):1
  from /Users/jboursiquot/.rvm/rubies/
ruby-1.9.3-p194/bin/irb:16:in `<main>'
```

Raising Exceptions

```
irb> raise TypeError, "I don't like your type"
TypeError: I don't like your type
  from (irb):4
  from /Users/jboursiquot/.rvm/rubies/
ruby-1.9.3-p194/bin/irb:16:in `<main>'
```

Raising Exceptions

```
irb> raise "Watch me run...oops"
RuntimeError: Watch me run...oops
  from (irb):5
  from /Users/jboursiquot/.rvm/rubies/
ruby-1.9.3-p194/bin/irb:16:in `<main>'
```

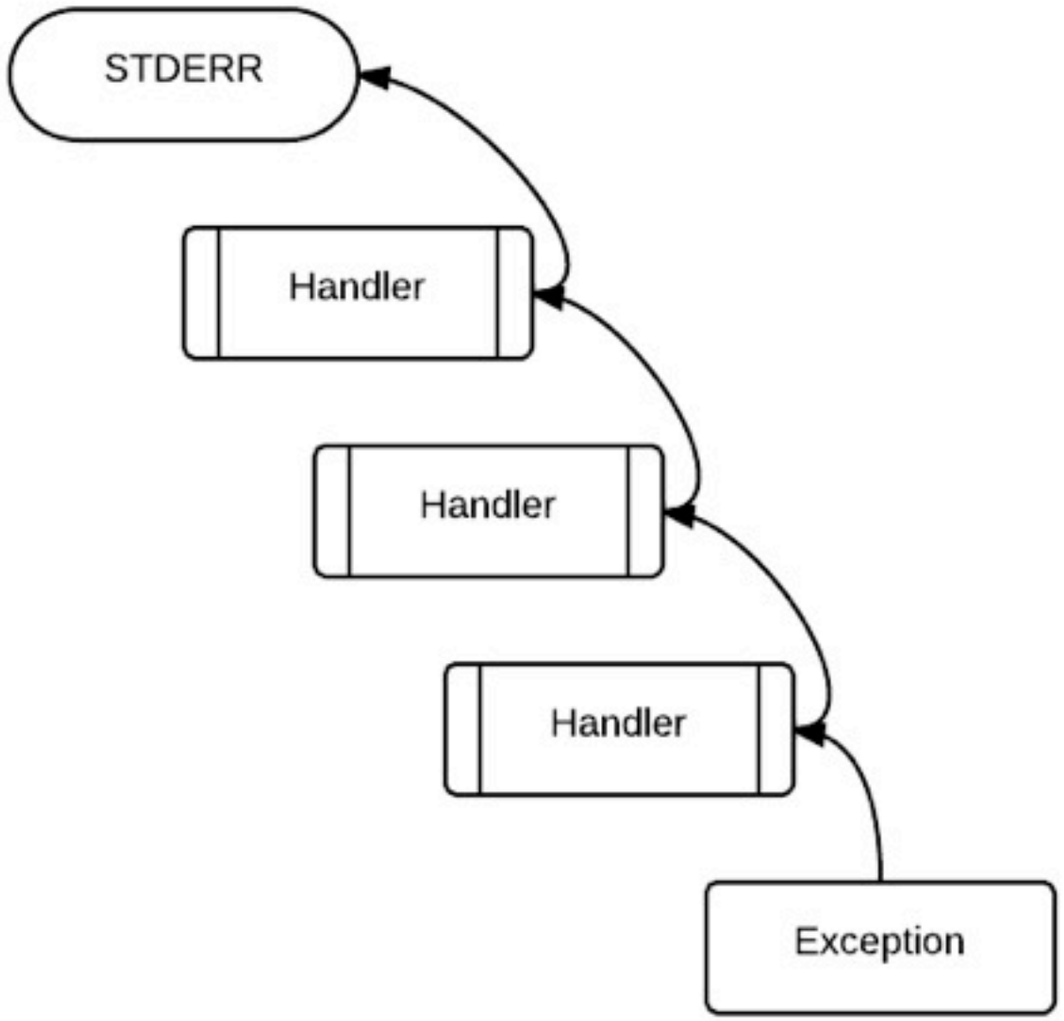
Raising Exceptions

```
irb> raise RuntimeError, "Whiskey!!!",
["Bourbon:1","Corn:2","Malt:3","Wheat:4"]
RuntimeError: Whiskey!!!
  from Bourbon:1
  from Corn:2
  from Malt:3
  from Wheat:4
```

Raising Exceptions

```
irb> raise RuntimeError, "Whiskey!!!",
["Bourbon:1","Corn:2","Malt:3","Wheat:4"]
RuntimeError: Whiskey!!!
  from Bourbon:1
  from Corn:2
  from Malt:3
  from Wheat:4
```

Bubbling



Bubbling

- Raised exceptions travel up the call stack in search of a matching exception handler
- If the call stack is exhausted without a matching handler, the exception is printed to STDERR and program aborts <-- You want to avoid this

What do Dennis Leary and Fontella Bass share in common?

What do Dennis Leary and Fontella Bass share in common?

`rescue` to the, um, rescue

- Exceptions are handled by `rescue` in one of two ways:
  - as a clause within a `begin...end` block and class, method or module definitions
  - as a statement modifier

rescue Example 1

```
irb> begin
irb>   raise "You did me wrong"
irb> rescue => e
irb>   e.message
irb> end
 => "You did me wrong"
```

rescue Example 2

```
irb> begin
irb>   raise TypeError, "me no like"
irb> rescue TypeError => e
irb>   "Why u no like?"
irb> rescue RuntimeError => e
irb>   "RuntimeError: #{e.message}"
irb> end
 => "Why u no like?"
```

rescue Example 3

```
irb> begin
irb>   raise TypeError, "me no like"
irb> rescue NameError => e
irb>   "Bad Name"
irb> rescue TypeError, RuntimeError => e
irb>   "Error: #{e.message}"
irb> end
 => "Error: me no like"
```

rescue Example 4

```
[0,1,2].map{|d| 1/d rescue nil}
 => [nil, 1, 0]
```

```
[0,1,2].map{|d| 1/d rescue ZeroDivisionError}
 => [ZeroDivisionError, 1, 0]
```

```
[0,1,2].map{|d| 1/d rescue $!}
 => [#<ZeroDivisionError: divided by 0>, 1, 0]
```

else

- Used in a group of `rescue` statements
- Triggered when none of the handlers are

else Example 1

```
irb> begin
irb>   p "All your base are belong to us"
irb> rescue => e
irb>   p e.message
irb> else
irb>   p "Not rescued"
irb> end
 => "All your base are belong to us"
 => "Not rescued"
```

else Example 2

```
irb> begin
irb>   p "All your base are belong to us"
irb>   raise "Oh no!"
irb> rescue TypeError => e
irb>   p "TypeError: #{e.message}"
irb> rescue RuntimeError => e
irb>   p "RuntimeError: #{e.message}"
irb> else
irb>   p "Nothing to see here"
irb> end
 => "All your base are belong to us"
 => "RuntimeError: Oh no!"
```

ensure

- Statements within an `ensure` clause are guaranteed to execute
- Generally a good place to clean up after yourself

`ensure`'s behavior

- If an `else` is present, its statements are executed followed by `ensure`'s statements
- If a `return` statement appears before an `else` clause, `else` statements (if present) are skipped but `ensure`'s are executed
- If an exception is raised, any matching handler will execute followed by `ensure`'s statements

ensure Example

```
irb> begin
irb>   youtube_gangnam_style_views = 1196472881
irb>   my_remaining_patience = 0
irb>   youtube_gangnam_style_views / my_remaining_patience
irb> rescue ZeroDivisionError => e
irb>   p "ZeroDivisionError"
irb> else
irb>   p "Nothing to see here"
irb> ensure
irb>   p "Never watch Gangnam Style again"
irb> end
 => "ZeroDivisionError"
 => "Never watch Gangnam Style again"
```

Reflect on...

- Exceptions for exceptional cases
- Ruby provides `throw` and `catch` mechanisms for expected failures which can be more appropriate based on the use case

Let's program something defensively, shall we?

Exercise also available on GitHub:

https://github.com/jboursiquot/defensive-ruby-talk

Quick Recap

- Defensive Programming is about anticipating the unexpected
- Ruby lets you know of the unexpected by raising exceptions based on a pre-defined set of classes and subclasses for you to handle
- You can have your own custom exception classes that inherit from Ruby's (StandardError) for more granular captures

Quick Recap

- Keywords and clauses: `raise`, `rescue`, `else` (within a rescue), `ensure`
- For further study: `throw`/`catch` and differentiating the exceptional from the expected

Where to from here

- Rubydoc's Exception class documentation (http://www.ruby-doc.org/core-1.9.3/Exception.html)
- Pragmatic Programmer's Guide to Ruby (Earlier edition available online free: http://www.ruby-doc.org/docs/ProgrammingRuby/html/tut_exceptions.html)
- Avdi Grimm's Exceptional Ruby (http://exceptionalruby.com/)

Reach Out

Johnny Boursiquot
jboursiquot@gmail.com
@jboursiquot

Credits

- "Wait For It" from failblog.org
- Coca Cola fail from http://thedailywtf.com
- GitHub 404 from GitHub (duh)
- Photo of Dennis Leary from Google Plus https://plus.google.com/u/0/+DenisLeary/posts
- Photo of Fontella Bass from http://darkjive.com/2011/12/27/fontella-bass-sassy-soulful-siren-in-the-first-degree/