SPEED UP YOUR APP WITH

CRYSTAL

INTRODUCTION

Todd Resudek

Backend Engineer at Weedmaps

github: supersimple

WHAT IS CRYSTAL?

WHAT IS CRYSTAL?

WHAT IS CRYSTAL?

"Crystal is a great tool for your toolkit where high performance is paramount."

- Mike Perham

author of Sidekiq

WHAT IS CRYSTAL?

- ✓ Started in 2012
- ✓ Currently on v 0.18.0
- ✓ Statically type-checked but without having to specify the type of variables or method arguments
- ✓ Compiles to efficient native code
- ✓ Ruby-inspired syntax

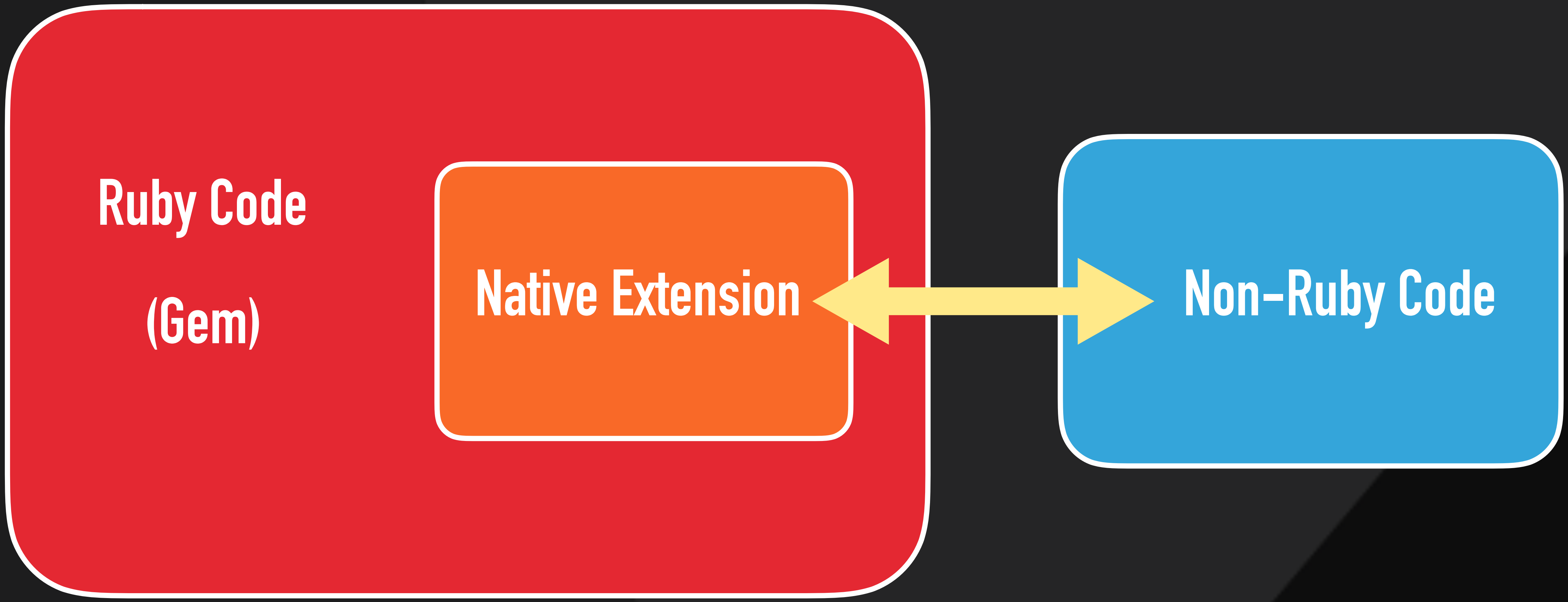
BUT THIS IS A RUBY MEETUP

...DO YOU HAVE A MOMENT TO TALK ABOUT NATIVE EXTENSIONS?

NATIVE EXTENSIONS

“Native extensions” are the glue that connects a Ruby gem with some other non-Ruby software component or library present on your machine.

NATIVE EXTENSIONS



NATIVE EXTENSIONS

- ✓ Usually written in C
- ✓ Included in Ruby Gem package
- ✓ Provides interface for Gem code to interact with C code

NATIVE EXTENSIONS

This means that Ruby gem authors can use Ruby to do what Ruby is best at, but switch to C or some other programming language or library when that makes sense.

WHAT ABOUT CRYSTAL?

AVAILABLE TYPES

Nil

Bool

Int - Int8, Int16, Int32, Int64 & Unsigned counterparts (UInt8, etc.)

Float - Float32, Float64

Char - 32bit UTF-8 character

String

Symbol

Array

Hash

Range

Regex

Tuple

NamedTuple

Proc

AVAILABLE TYPES - NIL

```
foo = nil
foo # nil
foo.class # Nil
```

AVAILABLE TYPES - BOOL

```
foo = true
foo # true
foo.class # Bool
@foo : Bool = true
```

AVAILABLE TYPES - INTEGERS

```
1      # Int32

1_i8   # Int8
1_i16  # Int16
1_i32  # Int32
1_i64  # Int64

1_u8   # UInt8
1_u16  # UInt16
1_u32  # UInt32
1_u64  # UInt64

+10    # Int32
-20    # Int32

2147483648          # Int64
9223372036854775808 # UInt64
```

```
42.to_s            # "42"
42.even?           # true
42.remainder 5     # 2
42.pred            # 41
42.downto 36       # …
```

AVAILABLE TYPES - FLOAT

```
1.0        # Float64
1.0_f32    # Float32
1_f32      # Float32

1e10       # Float64
1.5e10     # Float64
1.5e-7     # Float64

+1.3       # Float64
-0.5       # Float64
```

```
42.42.round  # 42.0
42.42.ceil   # 43.0
42.42.to_i   # 42
```

AVAILABLE TYPES – STRING

```
“hello”                                        # “hello”
‘hello’                                        # ERROR
<←TEXT
…
TEXT                                           # …
a, b = “foo”, “bar”
“#{a} #{b}”                                    # “foo bar”
```

AVAILABLE TYPES – STRING

```
“foo” * 3                                   # “foofoofoo”
“bar”.chars                           # [‘b’, ‘a’, ‘r’]
“BAZ”.downcase                                  # “baz”
“World”.gsub(“or”, “i”)                        # “Wild”
“Hello” + “World”                        # “HelloWorld”
“Hello” << “World”                              # ERROR
```

AVAILABLE TYPES - ARRAY

```
arr = [] of String                                       # []
arr : Array(String | Int32)                              # []
["r", 42, true, 'x'].class      # Array(Bool | Char | Int32
                                                  | String)
%w(foo bar baz)                       # ["foo","bar","baz"]
%i(foo bar baz)                          # [:foo,:bar,:baz]
arr << "foo"                                        # ["foo"]
arr << 42                                           # ERROR
```

CLASS DEFINITIONS

```
class Person
  @@brain = 1

  def initialize(name : String)
    @name = name
  end

  def name
    @name
  end

  def self.brain
    @@brain
  end

end
```

CLASS DEFINITIONS

A method's return type is always inferred by the compiler.

However, you might want to specify it for two reasons:

- ✓ To make sure that the method returns the type that you want
- ✓ To make it appear in documentation comments

```
def some_method : String
  "hello"
end
```

OVERLOADING

you can have different methods with the same name and different number of arguments and they will be considered as separate methods. This is called ***method overloading***.

Methods overload by several criteria:

- ✓ The number of arguments
- ✓ The type restrictions applied to arguments
- ✓ The names of required named arguments
- ✓ Whether the method accepts a block or not

OVERLOADING

```
class Roster
  @attendees : String = “”

  def add_attendee
    @attendees += “Anonymous”
   end

  def add_attendee(name : String)
     @attendees += name
   end

   def add_attendee(names : Array(String))
     names.each{ |name| @attendees += name }
   end

  def add_attendee(name : String)
    @attendees += yield name
  end
end
```

ARGUMENTS

NAMED ARGUMENTS

All arguments can also be specified, in addition to their position, by their name.

```
john.become_older by: 5
```

When there are many arguments, the order of the names in the invocation don't matter, as long as all required arguments are covered:

```
def some_method(x, y = 1, z = 2, w = 3)
  # do something...
end

some_method 10                    # x: 10, y: 1, z: 2, w: 3
some_method 10, z: 10             # x: 10, y: 1, z: 10, w: 3
some_method 10, w: 1, y: 2, z: 3 # x: 10, y: 2, z: 3, w: 1
some_method y: 10, x: 20          # x: 20, y: 10, z: 2, w: 3

some_method y: 10                 # Error, missing argument: x
```

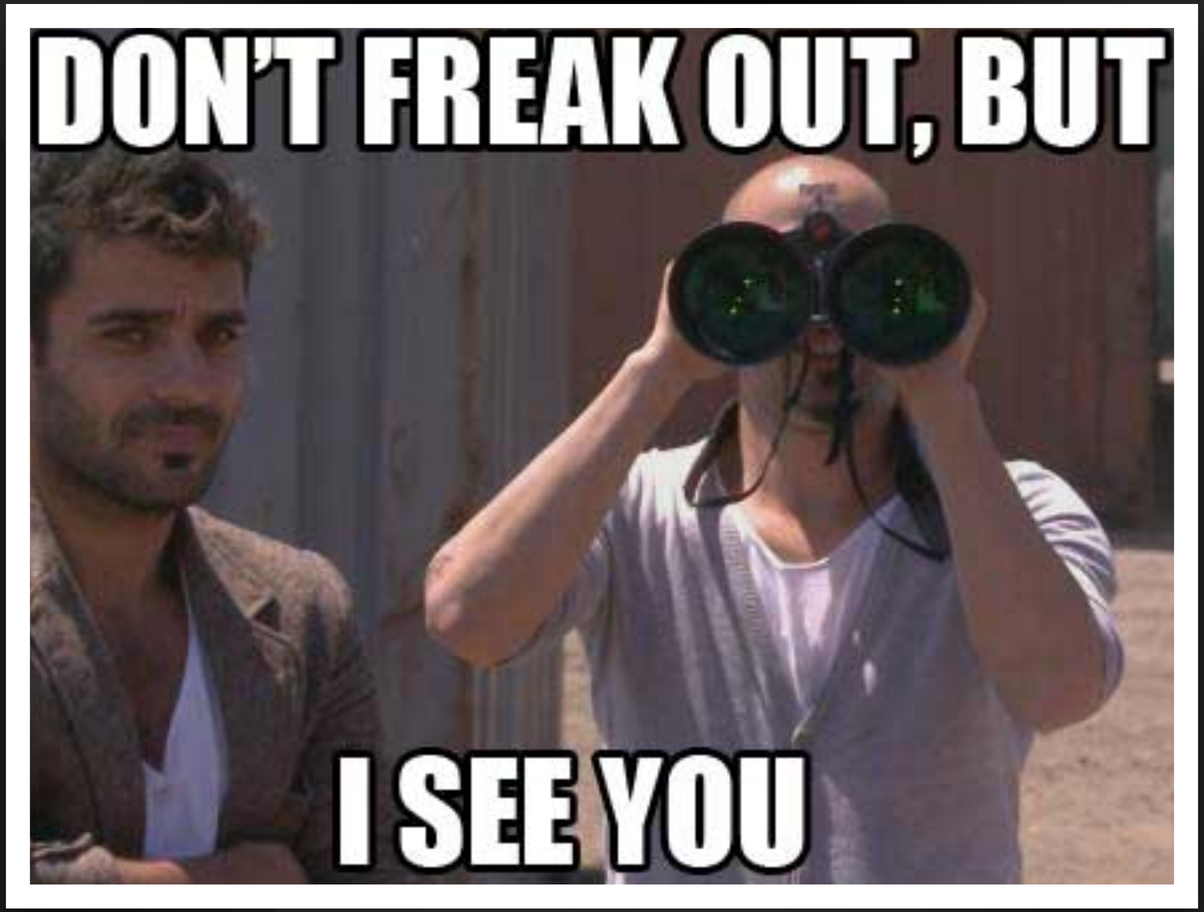
SPLATS AND TUPLES

```
def sum(*elements)
  elements.reduce(0){|sum,i| sum += i }
end

sum 1, 2, 3     #=> 6
sum 1, 2, 3, 4.5 #=> 10.5


def coords(**points)
  puts points
end

coords(x: 7, y: 8, z: 0)    #=> {x: 7, y: 8, z: 0}
```



VISIBILITY

VISIBILITY

Methods are public by default: the compiler will always let you invoke them. Because public is the default, there is no public keyword.

There are also protected and private methods.

```
class Visibility
  def small
    @foo.downcase
  end

 private def large
   @foo.upcase
 end
end
```

ACCESSORS

Crystal uses the keywords: getter, setter, and property rather than the ruby attr syntax.

RUBY

```
attr_accessor :foo
attr_writer :bar
attr_accessor :baz
```

CRYSTAL

```
getter :foo
setter :bar
property :baz
```

BLOCK SHORTHAND

RUBY

```
["a", "b", “c"].map(&:upcase)
["a", "b", “c"].map(&.*(3))  # ERROR
```

CRYSTAL

```
["a", "b", "c"].map(&.upcase)
["a", "b", "c"].map(&.*(3))
```

MACROS

Macros are methods that receive AST nodes at compile-time and produce code that is pasted into a program.

```
macro define_method(name, content)
  def {{name}}
    {{content}}
  end
end

# This generates:
def foo
 1
end
define_method foo, 1

foo #=> 1
```

MACROS - CONDITIONALS

```
macro define_method(name, content)
  def {{name}}
    {% if content == 1 %}
      "one"
    {% else %}
      {{content}}
    {% end %}
  end
end
```

MACROS - ITERATING

```
macro define_dummy_methods(names)
  {% for name, index in names %}
    def {{name.id}}
      {{index}}
    end
  {% end %}
end

define_dummy_methods [foo, bar, baz]

foo #=> 0
bar #=> 1
baz #=> 2
```

CONCURRENCY / PARALLELISM

A concurrent system is one that can be in charge of many tasks, although it is not necessarily executing them at the same time.

You can think of yourself being in the kitchen cooking: you chop an onion, put it to fry, and while it's being fried you chop a tomato, but you are not doing all of those things at the same time: you distribute your time between those tasks.

Parallelism would be to stir fry onions with one hand while with the other one you chop a tomato.

CONCURRENCY / PARALLELISM

- ✓ Crystal currently supports concurrency, but not parallelism.
- ✓ A Crystal program executes in a single operating system thread, except the Garbage Collector (GC) which implements a concurrent mark-and-sweep (currently Boehm GC).

FIBERS

To achieve concurrency, Crystal has fibers.

A fiber is in a way similar to an operating system thread except that it's much more lightweight and its execution is managed internally by the process. So, a program will spawn multiple fibers and Crystal will make sure to execute them when the time is right.

EVENT LOOP

For everything I/O related there's an event loop.

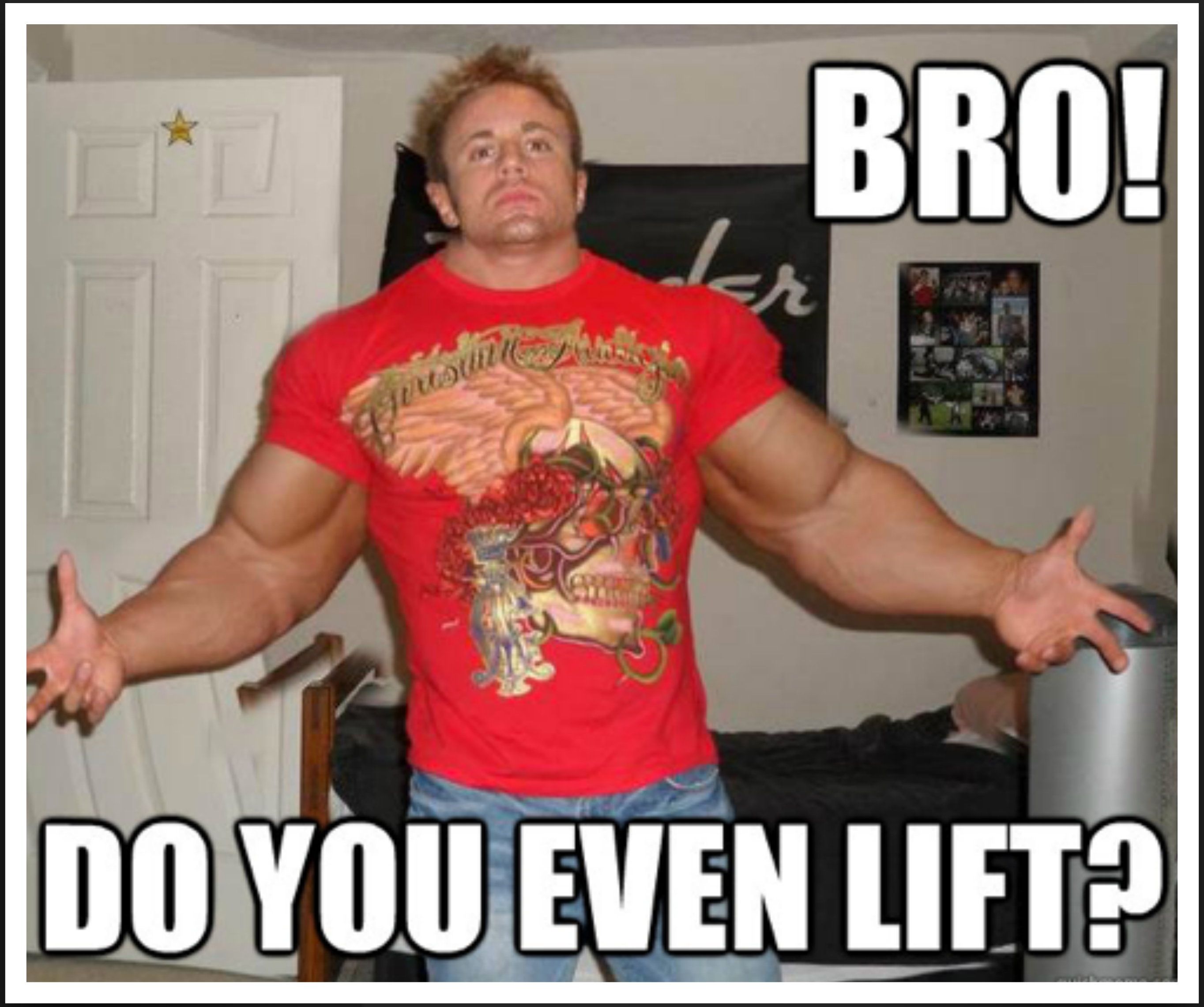
Some time-consuming operations are delegated to it, and while the event loop waits for that operation to finish the program can continue executing other fibers.

CHANNELS

Crystal has Channels inspired by CSP*.

They allow communicating data between fibers without sharing memory and without having to worry about locks, semaphores or other special structures.

*communicating sequential processes (CSP) is a formal language for describing patterns of interaction in concurrent systems.



CODING EXERCISE

BACK TO NATIVE EXTENSIONS

FAST BLANK

fast_blank is a simple C extension which provides a fast implementation of Active Support's String#blank? method written by Sam Saffron

```
96 Lines of C code
```

```
BENCHMARK (string length: speed)
0   : +13.75x
6   : +7.53x
14  : +11.80x
24  : +9.03x
136 : +9.08x
```

https://github.com/SamSaffron/fast_blank

ACTIVESUPPORT::INFLECTOR

Handles string operations, including transforms, and pluralization.

AVAILABLE METHODS (20)

camelize, classify, constantize

dasherize, deconstantize, demodulize

foreign_key

humanize

inflections

ordinal, ordinalize

parameterize, pluralize

safe_constantize, singularize

tableize, titleize, transliterate

underscore, upcase_first

ACTIVESUPPORT::INFLECTOR
VS
CRYSTAL::INFLECTOR

ActiveSupport::Inflector

```ruby
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)

  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match] || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string.gsub!(/(?:_|(\/))([a-z\d]*)/i) { "#{$1}#{inflections.acronyms[$2] || $2.capitalize}" }
    string.gsub!('/'.freeze, '::'.freeze)
    string
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub('::'.freeze, '/'.freeze)
    word.gsub!(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { "#{$1 && '_'.freeze }
#{$2.downcase}" }
    word.gsub!(/([A-Z\d]+)([A-Z][a-z])/, '\1_\2'.freeze)
    word.gsub!(/([a-z\d])([A-Z])/, '\1_\2'.freeze)
    word.tr!("-".freeze, "_".freeze)
    word.downcase!
    word
  end
  def humanize(lower_case_and_underscored_word, options = {})
    result = lower_case_and_underscored_word.to_s.dup
    inflections.humans.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
    result.sub!(/\A_+/, ''.freeze)
    result.sub!(/_id\z/, ''.freeze)
    result.tr!('_'.freeze, ' '.freeze)
    result.gsub!(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match] || match.downcase}"
    end
    if options.fetch(:capitalize, true)
      result.sub!(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string)
    string.length > 0 ? string[0].upcase.concat(string[1..-1]) : ''
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, ''.freeze)))
  end
  def dasherize(underscored_word)
    underscored_word.tr('_'.freeze, '-'.freeze)
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex('::')
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex('::') || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).include?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  def apply_inflections(word, rules)
    result = word.to_s.dup
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
      result
    end
  end
end
```

Crystal::Inflector

```crystal
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)
  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match]? || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string = string.gsub(/(?:_|(\/))([a-z\d]*)/i) { |match| "#{match[0]}#{inflections.acronyms[match[1..-1]]? ||
(match[1..-1].capitalize)}" }
    string = string.gsub("/", "::")
    string = string.gsub("_", "")
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub("::", "/")
    word = word.gsub(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { |match| "#{'_' if !
word.downcase.starts_with?(match.downcase)}#{match.downcase}" }
    word = word.gsub(/([A-Z\d]+)([A-Z][a-z])/, "\\1_\\2")
    word = word.gsub(/([a-z\d])([A-Z])/, "\\1_\\2")
    word = word.gsub(/\W_/) { |match| match[0] }
    word = word.tr("-", "_")
    word.downcase
  end
   def humanize(lower_case_and_underscored_word, capitalize = true)
    original = lower_case_and_underscored_word.to_s
    result = original
    inflections.humans.find do |arr, _|
      rule, replacement = arr
      result = original.sub(rule, replacement)
      result != original
    end
    result = result.sub(/\A_+/, "")
    result = result.sub(/_id\z/, "")
    result = result.tr("_", " ")
    result = result.gsub(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match]? || match.downcase}"
    end
    if capitalize
      result = result.sub(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string : String)
    string.size > 0 ? string[1..-1].insert(0, string[0].upcase) : ""
  end
  def upcase_first(char : Char)
    char.upcase
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, "")))
  end
  def dasherize(underscored_word)
    underscored_word.tr("_", "-")
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex("::")
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex("::") || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).includes?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  private def apply_inflections(word, rules)
    original = word.to_s.dup
    result = original
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.find do |arr, _|
        rule, replacement = arr
        result = original.sub(rule, replacement)
        result != original
      end
      result
    end
  end
end
```

ActiveSupport::Inflector

```ruby
  module Inflector
    extend self
    def pluralize(word, locale = :en)
      apply_inflections(word, inflections(locale).plurals)

    end
    def singularize(word, locale = :en)
      apply_inflections(word, inflections(locale).singulars)
    end
    def camelize(term, uppercase_first_letter = true)
      string = term.to_s
      if uppercase_first_letter
        string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match] || match.capitalize }
      else
        string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
      end
      string.gsub!(/(?:_|(\/))([a-z\d]*)/i) { "#{$1}#{inflections.acronyms[$2] || $2.capitalize}" }
      string.gsub!('/'.freeze, '::'.freeze)
      string
    end
    def underscore(camel_cased_word)
      return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
      word = camel_cased_word.to_s.gsub('::'.freeze, '/'.freeze)
      word.gsub!(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { "#{$1 && '_'.freeze }
#{$2.downcase}" }
      word.gsub!(/([A-Z\d]+)([A-Z][a-z])/, '\1_\2'.freeze)
      word.gsub!(/([a-z\d])([A-Z])/, '\1_\2'.freeze)
      word.tr!("-".freeze, "_".freeze)
      word.downcase!
      word
    end
    def humanize(lower_case_and_underscored_word, options = {})
      result = lower_case_and_underscored_word.to_s.dup
      inflections.humans.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
      result.sub!(/\A_+/, ''.freeze)
      result.sub!(/_id\z/, ''.freeze)
      result.tr!('_'.freeze, ' '.freeze)
      result.gsub!(/([a-z\d]*)/i) do |match|
        "#{inflections.acronyms[match] || match.downcase}"
      end
      if options.fetch(:capitalize, true)
        result.sub!(/\A\w/) { |match| match.upcase }
      end
      result
    end
    def upcase_first(string)
      string.length > 0 ? string[0].upcase.concat(string[1..-1]) : ''
    end
    def titleize(word)
      humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
    end
    def tableize(class_name)
      pluralize(underscore(class_name))
    end
    def classify(table_name)
      camelize(singularize(table_name.to_s.sub(/.*\./, ''.freeze)))
    end
    def dasherize(underscored_word)
      underscored_word.tr('_'.freeze, '-'.freeze)
    end
    def demodulize(path)
      path = path.to_s
      if i = path.rindex('::')
        path[(i+2)..-1]
      else
        path
      end
    end
    def deconstantize(path)
      path.to_s[0, path.rindex('::') || 0]
    end
    def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
      underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
    end
    def ordinal(number)
      abs_number = number.to_i.abs
      if (11..13).include?(abs_number % 100)
        "th"
      else
        case abs_number % 10
          when 1; "st"
          when 2; "nd"
          when 3; "rd"
          else    "th"
        end
      end
    end
    def ordinalize(number)
      "#{number}#{ordinal(number)}"
    end
    def apply_inflections(word, rules)
      result = word.to_s.dup
      if word.empty? || inflections.uncountables.uncountable?(result)
        result
      else
        rules.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
        result
      end
    end
  end
```

Crystal::Inflector

```crystal
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)
  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match]? || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string = string.gsub(/(?:_|(\/))([a-z\d]*)/i) { |match| "#{match[0]}#{inflections.acronyms[match[1..-1]]? ||
(match[1..-1].capitalize)}" }
    string = string.gsub("/", "::")
    string = string.gsub("_", "")
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub("::", "/")
    word = word.gsub(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { |match| "#{'_' if !
word.downcase.starts_with?(match.downcase)}#{match.downcase}" }
    word = word.gsub(/([A-Z\d]+)([A-Z][a-z])/, "\\1_\\2")
    word = word.gsub(/([a-z\d])([A-Z])/, "\\1_\\2")
    word = word.gsub(/\W_/) { |match| match[0] }
    word = word.tr("-", "_")
    word.downcase
  end
   def humanize(lower_case_and_underscored_word, capitalize = true)
    original = lower_case_and_underscored_word.to_s
    result = original
    inflections.humans.find do |arr, _|
      rule, replacement = arr
      result = original.sub(rule, replacement)
      result != original
    end
    result = result.sub(/\A_+/, "")
    result = result.sub(/_id\z/, "")
    result = result.tr("_", " ")
    result = result.gsub(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match]? || match.downcase}"
    end
    if capitalize
      result = result.sub(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string : String)
    string.size > 0 ? string[1..-1].insert(0, string[0].upcase) : ""
  end
  def upcase_first(char : Char)
    char.upcase
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, "")))
  end
  def dasherize(underscored_word)
    underscored_word.tr("_", "-")
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex("::")
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex("::") || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).includes?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  private def apply_inflections(word, rules)
    original = word.to_s.dup
    result = original
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.find do |arr, _|
        rule, replacement = arr
        result = original.sub(rule, replacement)
        result != original
      end
      result
    end
  end
end
```

ActiveSupport::Inflector

```
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)

  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[mat[illegible]tch.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)[illegible] { |m[illegible]tch| ma[illegible]ch.[illegible]wn[illegible]ase }
    end
    string.gsub!(/(?:_|(\/))([a-z\d]*)/i) { "#{$1}#{inflections.acronyms[$2] || $2.capital[illegible] }
    string.gsub!('/'.freeze, '::'.freeze)
    string
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub('::'.freeze, '/'.freeze)
    word.gsub!(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { "#{$1 && '_'.freeze }
#{$2.downcase}" }
    word.gsub!(/([A-Z\d]+)([A-Z][a-z])/, '\1_\2'.freeze)
    word.gsub!(/([a-z\d])([A-Z])/, '\1_\2'.freeze)
    word.tr!("-".freeze, "_".freeze)
    word.downcase!
    word
  end
  def humanize(lower_case_and_underscored_word, options = {})
    result = lower_case_and_underscored_word.to_s.dup
    inflections.humans.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
    result.sub!(/\A_+/, ''.freeze)
    result.sub!(/_id\z/, ''.freeze)
    result.tr!('_'.freeze, ' '.freeze)
    result.gsub!(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match] || match.downcase}"
    end
    if options.fetch(:capitalize, true)
      result.sub!(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string)
    string.length > 0 ? string[0].upcase.concat(string[1..-1]) : ''
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, ''.freeze)))
  end
  def dasherize(underscored_word)
    underscored_word.tr('_'.freeze, '-'.freeze)
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex('::')
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex('::') || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).include?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  def apply_inflections(word, rules)
    result = word.to_s.dup
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
      result
    end
  end
end
```

85%
copied

Crystal::Inflector

```
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)
  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match]? || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string = string.gsub(/(?:_|(\/))([a-z\d]*)/i) { |match| "#{match[0]}#{inflections.acronyms[match[1..-1]]? ||
(match[1..-1].capitalize)}" }
    string = string.gsub("/", "::")
    string = string.gsub("_", "")
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub("::", "/")
    word = word.gsub(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { |match| "#{'_' if !
word.downcase.starts_with?(match.downcase)}#{match.downcase}" }
    word = word.gsub(/([A-Z\d]+)([A-Z][a-z])/, "\\1_\\2")
    word = word.gsub(/([a-z\d])([A-Z])/, "\\1_\\2")
    word = word.gsub(/\W_/) { |match| match[0] }
    word = word.tr("-", "_")
    word.downcase
  end
   def humanize(lower_case_and_underscored_word, capitalize = true)
    original = lower_case_and_underscored_word.to_s
    result = original
    inflections.humans.find do |arr, _|
      rule, replacement = arr
      result = original.sub(rule, replacement)
      result != original
    end
    result = result.sub(/\A_+/, "")
    result = result.sub(/_id\z/, "")
    result = result.tr("_", " ")
    result = result.gsub(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match]? || match.downcase}"
    end
    if capitalize
      result = result.sub(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string : String)
    string.size > 0 ? string[1..-1].insert(0, string[0].upcase) : ""
  end
  def upcase_first(char : Char)
    char.upcase
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, "")))
  end
  def dasherize(underscored_word)
    underscored_word.tr("_", "-")
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex("::")
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex("::") || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).includes?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  private def apply_inflections(word, rules)
    original = word.to_s.dup
    result = original
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.find do |arr, _|
        rule, replacement = arr
        result = original.sub(rule, replacement)
        result != original
      end
      result
    end
  end
end
```

ActiveSupport::Inflector

```ruby
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)

  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[mat[illegible]h] [illegible]tch capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)[illegible] { |m[illegible]tch| ma[illegible]ch.[illegible]wn[illegible]ase }
    end
    string.gsub!(/(?:_|(\/))([a-z\d]*)/i) { "#{$1}#{inflections.acronyms[$2] || $2.capital[illegible] }
    string.gsub!('/'.freeze, '::'.freeze)
    string
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub('::'.freeze, '/'.freeze)
    word.gsub!(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { "#{$1 && '_'.freeze }
#{$2.downcase}" }
    word.gsub!(/([A-Z\d]+)([A-Z][a-z])/, '\1_\2'.freeze)
    word.gsub!(/([a-z\d])([A-Z])/, '\1_\2'.freeze)
    word.tr!("-".freeze, "_".freeze)
    word.downcase!
    word
  end
  def humanize(lower_case_and_underscored_word, options = {})
    result = lower_case_and_underscored_word.to_s.dup
    inflections.humans.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
    result.sub!(/\A_+/, ''.freeze)
    result.sub!(/_id\z/, ''.freeze)
    result.tr!('_'.freeze, ' '.freeze)
    result.gsub!(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match] || match.downcase}"
    end
    if options.fetch(:capitalize, true)
      result.sub!(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string)
    string.length > 0 ? string[0].upcase.concat(string[1..-1]) : ''
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, ''.freeze)))
  end
  def dasherize(underscored_word)
    underscored_word.tr('_'.freeze, '-'.freeze)
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex('::')
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex('::') || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).include?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  def apply_inflections(word, rules)
    result = word.to_s.dup
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
      result
    end
  end
end
```

85%

copied

Crystal::Inflector

```crystal
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)
  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match]? || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string = string.gsub(/(?:_|(\/))([a-z\d]*)/i) { |match| "#{match[0]}#{inflections.acronyms[match[1..-1]]? ||
(match[1..-1].capitalize)}" }
    string = string.gsub("/", "::")
    string = string.gsub("_", "")
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub("::", "/")
    word = word.gsub(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { |match| "#{'_' if !
word.downcase.starts_with?(match.downcase)}#{match.downcase}" }
    word = word.gsub(/([A-Z\d]+)([A-Z][a-z])/, "\\1_\\2")
    word = word.gsub(/([a-z\d])([A-Z])/, "\\1_\\2")
    word = word.gsub(/\W_/) { |match| match[0] }
    word = word.tr("-", "_")
    word.downcase
  end
   def humanize(lower_case_and_underscored_word, capitalize = true)
    original = lower_case_and_underscored_word.to_s
    result = original
    inflections.humans.find do |arr, _|
      rule, replacement = arr
      result = original.sub(rule, replacement)
      result != original
    end
    result = result.sub(/\A_+/, "")
    result = result.sub(/_id\z/, "")
    result = result.tr("_", " ")
    result = result.gsub(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match]? || match.downcase}"
    end
    if capitalize
      result = result.sub(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string : String)
    string.size > 0 ? string[1..-1].insert(0, string[0].upcase) : ""
  end
  def upcase_first(char : Char)
    char.upcase
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, "")))
  end
  def dasherize(underscored_word)
    underscored_word.tr("_", "-")
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex("::")
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex("::") || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).includes?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  private def apply_inflections(word, rules)
    original = word.to_s.dup
    result = original
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.find do |arr, _|
        rule, replacement = arr
        result = original.sub(rule, replacement)
        result != original
      end
      result
    end
  end
end
```

ActiveSupport::Inflector

```ruby
  module Inflector
    extend self
    def pluralize(word, locale = :en)
      apply_inflections(word, inflections(locale).plurals)

    end
    def singularize(word, locale = :en)
      apply_inflections(word, inflections(locale).singulars)
    end
    def camelize(term, uppercase_first_letter = true)
      string = term.to_s
      if uppercase_first_letter
        string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match] || match.capitalize }
      else
        string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
      end
      string.gsub!(/(?:_|(\/))([a-z\d]*)/i) { "#{$1}#{inflections.acronyms[$2] || $2.capitalize}" }
      string.gsub!('/'.freeze, '::'.freeze)
      string
    end
    def underscore(camel_cased_word)
      return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
      word = camel_cased_word.to_s.gsub('::'.freeze, '/'.freeze)
      word.gsub!(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { "#{$1 && '_'.freeze }
#{$2.downcase}" }
      word.gsub!(/([A-Z\d]+)([A-Z][a-z])/, '\1_\2'.freeze)
      word.gsub!(/([a-z\d])([A-Z])/, '\1_\2'.freeze)
      word.tr!("-".freeze, "_".freeze)
      word.downcase!
      word
    end
    def humanize(lower_case_and_underscored_word, options = {})
      result = lower_case_and_underscored_word.to_s.dup
      inflections.humans.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
      result.sub!(/\A_+/, ''.freeze)
      result.sub!(/_id\z/, ''.freeze)
      result.tr!('_'.freeze, ' '.freeze)
      result.gsub!(/([a-z\d]*)/i) do |match|
        "#{inflections.acronyms[match] || match.downcase}"
      end
      if options.fetch(:capitalize, true)
        result.sub!(/\A\w/) { |match| match.upcase }
      end
      result
    end
    def upcase_first(string)
      string.length > 0 ? string[0].upcase.concat(string[1..-1]) : ''
    end
    def titleize(word)
      humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
    end
    def tableize(class_name)
      pluralize(underscore(class_name))
    end
    def classify(table_name)
      camelize(singularize(table_name.to_s.sub(/.*\./, ''.freeze)))
    end
    def dasherize(underscored_word)
      underscored_word.tr('_'.freeze, '-'.freeze)
    end
    def demodulize(path)
      path = path.to_s
      if i = path.rindex('::')
        path[(i+2)..-1]
      else
        path
      end
    end
    def deconstantize(path)
      path.to_s[0, path.rindex('::') || 0]
    end
    def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
      underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
    end
    def ordinal(number)
      abs_number = number.to_i.abs
      if (11..13).include?(abs_number % 100)
        "th"
      else
        case abs_number % 10
          when 1; "st"
          when 2; "nd"
          when 3; "rd"
          else    "th"
        end
      end
    end
    def ordinalize(number)
      "#{number}#{ordinal(number)}"
    end
    def apply_inflections(word, rules)
      result = word.to_s.dup
      if word.empty? || inflections.uncountables.uncountable?(result)
        result
      else
        rules.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
        result
      end
    end
  end
```

85% copied

4% fixed

Crystal::Inflector

```crystal
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)
  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match]? || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string = string.gsub(/(?:_|(\/))([a-z\d]*)/i) { |match| "#{match[0]}#{inflections.acronyms[match[1..-1]]? ||
(match[1..-1].capitalize)}" }
    string = string.gsub("/", "::")
    string = string.gsub("_", "")
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub("::", "/")
    word = word.gsub(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { |match| "#{'_' if !
word.downcase.starts_with?(match.downcase)}#{match.downcase}" }
    word = word.gsub(/([A-Z\d]+)([A-Z][a-z])/, "\\1_\\2")
    word = word.gsub(/([a-z\d])([A-Z])/, "\\1_\\2")
    word = word.gsub(/\W_/) { |match| match[0] }
    word = word.tr("-", "_")
    word.downcase
  end
   def humanize(lower_case_and_underscored_word, capitalize = true)
    original = lower_case_and_underscored_word.to_s
    result = original
    inflections.humans.find do |arr, _|
      rule, replacement = arr
      result = original.sub(rule, replacement)
      result != original
    end
    result = result.sub(/\A_+/, "")
    result = result.sub(/_id\z/, "")
    result = result.tr("_", " ")
    result = result.gsub(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match]? || match.downcase}"
    end
    if capitalize
      result = result.sub(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string : String)
    string.size > 0 ? string[1..-1].insert(0, string[0].upcase) : ""
  end
  def upcase_first(char : Char)
    char.upcase
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, "")))
  end
  def dasherize(underscored_word)
    underscored_word.tr("_", "-")
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex("::")
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex("::") || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).includes?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  private def apply_inflections(word, rules)
    original = word.to_s.dup
    result = original
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.find do |arr, _|
        rule, replacement = arr
        result = original.sub(rule, replacement)
        result != original
      end
      result
    end
  end
end
```

ActiveSupport::Inflector

```
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)

  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match] || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string.gsub!(/(?:_|(\/))([a-z\d]*)/i) { "#{$1}#{inflections.acronyms[$2] || $2.capitalize}" }
    string.gsub!('/'.freeze, '::'.freeze)
    string
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub('::'.freeze, '/'.freeze)
    word.gsub!(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { "#{$1 && '_'.freeze }
#{$2.downcase}" }
    word.gsub!(/([A-Z\d]+)([A-Z][a-z])/, '\1_\2'.freeze)
    word.gsub!(/([a-z\d])([A-Z])/, '\1_\2'.freeze)
    word.tr!("-".freeze, "_".freeze)
    word.downcase!
    word
  end
  def humanize(lower_case_and_underscored_word, options = {})
    result = lower_case_and_underscored_word.to_s.dup
    inflections.humans.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
    result.sub!(/\A_+/, ''.freeze)
    result.sub!(/_id\z/, ''.freeze)
    result.tr!('_'.freeze, ' '.freeze)
    result.gsub!(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match] || match.downcase}"
    end
    if options.fetch(:capitalize, true)
      result.sub!(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string)
    string.length > 0 ? string[0].upcase.concat(string[1..-1]) : ''
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, ''.freeze)))
  end
  def dasherize(underscored_word)
    underscored_word.tr('_'.freeze, '-'.freeze)
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex('::')
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex('::') || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).include?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  def apply_inflections(word, rules)
    result = word.to_s.dup
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.each { |(rule, replacement)| break if result.sub!(rule, replacement) }
      result
    end
  end
end
```

85% copied

4% fixed

Crystal::Inflector

```
module Inflector
  extend self
  def pluralize(word, locale = :en)
    apply_inflections(word, inflections(locale).plurals)
  end
  def singularize(word, locale = :en)
    apply_inflections(word, inflections(locale).singulars)
  end
  def camelize(term, uppercase_first_letter = true)
    string = term.to_s
    if uppercase_first_letter
      string = string.sub(/^[a-z\d]*/) { |match| inflections.acronyms[match]? || match.capitalize }
    else
      string = string.sub(/^(?:#{inflections.acronym_regex}(?=\b|[A-Z_])|\w)/) { |match| match.downcase }
    end
    string = string.gsub(/(?:_|(\/))([a-z\d]*)/i) { |match| "#{match[0]}#{inflections.acronyms[match[1..-1]]? ||
(match[1..-1].capitalize)}" }
    string = string.gsub("/", "::")
    string = string.gsub("_", "")
  end
  def underscore(camel_cased_word)
    return camel_cased_word unless camel_cased_word =~ /[A-Z-]|::/
    word = camel_cased_word.to_s.gsub("::", "/")
    word = word.gsub(/(?:(?<=([A-Za-z\d]))|\b)(#{inflections.acronym_regex})(?=\b|[^a-z])/) { |match| "#{'_' if !
word.downcase.starts_with?(match.downcase)}#{match.downcase}" }
    word = word.gsub(/([A-Z\d]+)([A-Z][a-z])/, "\\1_\\2")
    word = word.gsub(/([a-z\d])([A-Z])/, "\\1_\\2")
    word = word.gsub(/\W_/) { |match| match[0] }
    word = word.tr("-", "_")
    word.downcase
  end
   def humanize(lower_case_and_underscored_word, capitalize = true)
    original = lower_case_and_underscored_word.to_s
    result = original
    inflections.humans.find do |arr, _|
      rule, replacement = arr
      result = original.sub(rule, replacement)
      result != original
    end
    result = result.sub(/\A_+/, "")
    result = result.sub(/_id\z/, "")
    result = result.tr("_", " ")
    result = result.gsub(/([a-z\d]*)/i) do |match|
      "#{inflections.acronyms[match]? || match.downcase}"
    end
    if capitalize
      result = result.sub(/\A\w/) { |match| match.upcase }
    end
    result
  end
  def upcase_first(string : String)
    string.size > 0 ? string[1..-1].insert(0, string[0].upcase) : ""
  end
  def upcase_first(char : Char)
    char.upcase
  end
  def titleize(word)
    humanize(underscore(word)).gsub(/\b(?<!['’`])[a-z]/) { |match| match.capitalize }
  end
  def tableize(class_name)
    pluralize(underscore(class_name))
  end
  def classify(table_name)
    camelize(singularize(table_name.to_s.sub(/.*\./, "")))
  end
  def dasherize(underscored_word)
    underscored_word.tr("_", "-")
  end
  def demodulize(path)
    path = path.to_s
    if i = path.rindex("::")
      path[(i+2)..-1]
    else
      path
    end
  end
  def deconstantize(path)
    path.to_s[0, path.rindex("::") || 0]
  end
  def foreign_key(class_name, separate_class_name_and_id_with_underscore = true)
    underscore(demodulize(class_name)) + (separate_class_name_and_id_with_underscore ? "_id" : "id")
  end
  def ordinal(number)
    abs_number = number.to_i.abs
    if (11..13).includes?(abs_number % 100)
      "th"
    else
      case abs_number % 10
        when 1; "st"
        when 2; "nd"
        when 3; "rd"
        else    "th"
      end
    end
  end
  def ordinalize(number)
    "#{number}#{ordinal(number)}"
  end
  private def apply_inflections(word, rules)
    original = word.to_s.dup
    result = original
    if word.empty? || inflections.uncountables.uncountable?(result)
      result
    else
      rules.find do |arr, _|
        rule, replacement = arr
        result = original.sub(rule, replacement)
        result != original
      end
      result
    end
  end
end
```

~15 minutes to fix

BASIC DIFFERENCES

```
word.gsub!(/([a-z\d])([A-Z])/, '\1_\2'.freeze)
```

BASIC DIFFERENCES

```
word = word.gsub(/([a-z\d])([A-Z])/, '\1_\2'.freeze)
```

- ✓ Replace bang methods

BASIC DIFFERENCES

```
word = word.gsub(/([a-z\d])([A-Z])/, '\1_\2')
```

- ✓ Replace bang methods
- ✓ Remove freeze

BASIC DIFFERENCES

```
word = word.gsub(/([a-z\d])([A-Z])/, “\1_\2”)
```

- ✓ Replace bang methods
- ✓ Remove freeze
- ✓ Replace single quotes

BASIC DIFFERENCES

```
word = word.gsub(/([a-z\d])([A-Z])/, “\\1_\\2”)
```

- ✓ Replace bang methods
- ✓ Remove freeze
- ✓ Replace single quotes
- ✓ Add an extra \ to regex backrefs

REMAINING STEPS

- ✓ Convert between Ruby data and Crystal data
- ✓ Wrap Crystal methods
  - this lets us to use pure Crystal libraries
- ✓ Initialize Crystal methods for Ruby
  - (make these methods available via C API)
- ✓ Write some Ruby!

WRAP CRYSTAL METHODS

```
module Wrapper

  def self.ordinal(self : LibRuby::VALUE)
    int = Int.from_ruby(self)
    int.ordinal.to_ruby
  end

  def self.ordinalize(self : LibRuby::VALUE)
    int = Int.from_ruby(self)
    int.ordinalize.to_ruby
  end

  def self.squish(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.squish.to_ruby
  end

  def self.blank?(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.blank?.to_ruby
  end

  def self.titleize(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.titleize.to_ruby
  end

  def self.titlecase(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.titlecase.to_ruby
  end

  def self.dasherize(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.dasherize.to_ruby
  end


  def self.deconstantize(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.deconstantize.to_ruby
  end

  def self.tableize(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.tableize.to_ruby
  end

  def self.classify(self : LibRuby::VALUE)
    str = String.from_ruby(self)
    str.classify.to_ruby
  end
end
```

INIT C FUNCTIONS FOR RUBY

```
require "../lib_ruby"
require “./wrapper"

fun init = Init_inflector
  GC.init
  LibCrystalMain.__crystal_main(0, Pointer(Pointer(UInt8)).null)

  string = LibRuby.rb_define_class("String", LibRuby.rb_cObject)

  LibRuby.rb_define_method(string, "cr_squish",        ->Wrapper.squish,        0)
  LibRuby.rb_define_method(string, "cr_blank?",        ->Wrapper.blank?,        0)
  LibRuby.rb_define_method(string, "cr_pluralize",     ->Wrapper.pluralize,     0)
  LibRuby.rb_define_method(string, "cr_humanize",      ->Wrapper.humanize,      0)

  integer = LibRuby.rb_define_class("Integer", LibRuby.rb_cNumeric)
  LibRuby.rb_define_method(integer, "cr_ordinal",      ->Wrapper.ordinal,       0)
  LibRuby.rb_define_method(integer, "cr_ordinalize",   ->Wrapper.ordinalize,    0)

end
```

RUBY USAGE

```ruby
require "./inflector"

puts 1.cr_ordinalize                                            # => "1st"
puts 2.cr_ordinalize                                            # => "2nd"
puts ''.cr_blank?                                               # => true
puts '   '.cr_blank?                                            # => true
puts "apple".cr_pluralize                                      # => "apples"
puts "apples".cr_singularize                                    # => "apple"
puts "active_record/errors".cr_camelize                         # => "ActiveRecord::Errors"
puts "fancyCategory".cr_tableize                                # => "fancy_categories"
puts "employee_salary".cr_humanize                              # => "Employee salary"
puts "author_id".cr_humanize                                    # => "Author"
```

BENCHMARK-IPS RESULTS

iterations/second	ActiveSupport	Crystal	Improvement
ordinal	418,430	2,027,814	4.85x
ordinalize	140,863	556,205	3.95x
blank?	241,471	785,621	3.25x
squish	206,708	735,772	3.56x
pluralize	5,985	25,061	4.19x
singularize	6,276	28,546	4.55x
camelize	36,658	79,380	2.17x
titleize	14,837	38,707	2.61x
underscore	20,560	73,844	3.59x
demodulize	608,325	788,773	1.30x
deconstantize	532,506	797,424	1.50x
tableize	8,302	28,792	3.47x
classify	14,909	56,535	3.79x
humanize	40,904	82,314	2.01x
upcase_first	987,707	1,423,886	1.44x
foreign_key	13,642	66,009	4.84x

REFERENCES AND RESOURCES

Crystal-lang: crystal-lang.org/api/

Crystalized Ruby: https://github.com/phoffer/crystalized_ruby

Introduction to Crystal: http://leopard.in.ua/presentations/brug_2015/index.html

Introduction to Native Extensions: http://patshaughnessy.net/2011/10/31/dont-be-terrified-of-building-native-extensions