Ruby 2.1.0 New Features

Presented By: Shota Bakuradze

Github/StackOverflow: headshota

Agenda

- Exception cause
- Rational and complex literals
- Garbage collection with RGenGC
- `def` return value
- Required keyword arguments
- Refinements
- Object allocation tracing

Exception cause

- Often when dealing with low level API-s we don't want the client to see its exceptions.
- But we need to give a client an ability to see what the root of the exception was in the first place.
- When exception is raised from rescue block, Exception#cause refers to an exception for which was caught in rescue block.
- Note: Prior to 2.1.0, use `cause`gem for similar goal.

Exception cause

```
begin
  raise StandardError.new "low level exception"
rescue
  begin
    raise StandardError.new "high level exception"
  rescue StandardError => e
    puts "#{e}"
    puts "#{e.cause.message}"
  end
End
```

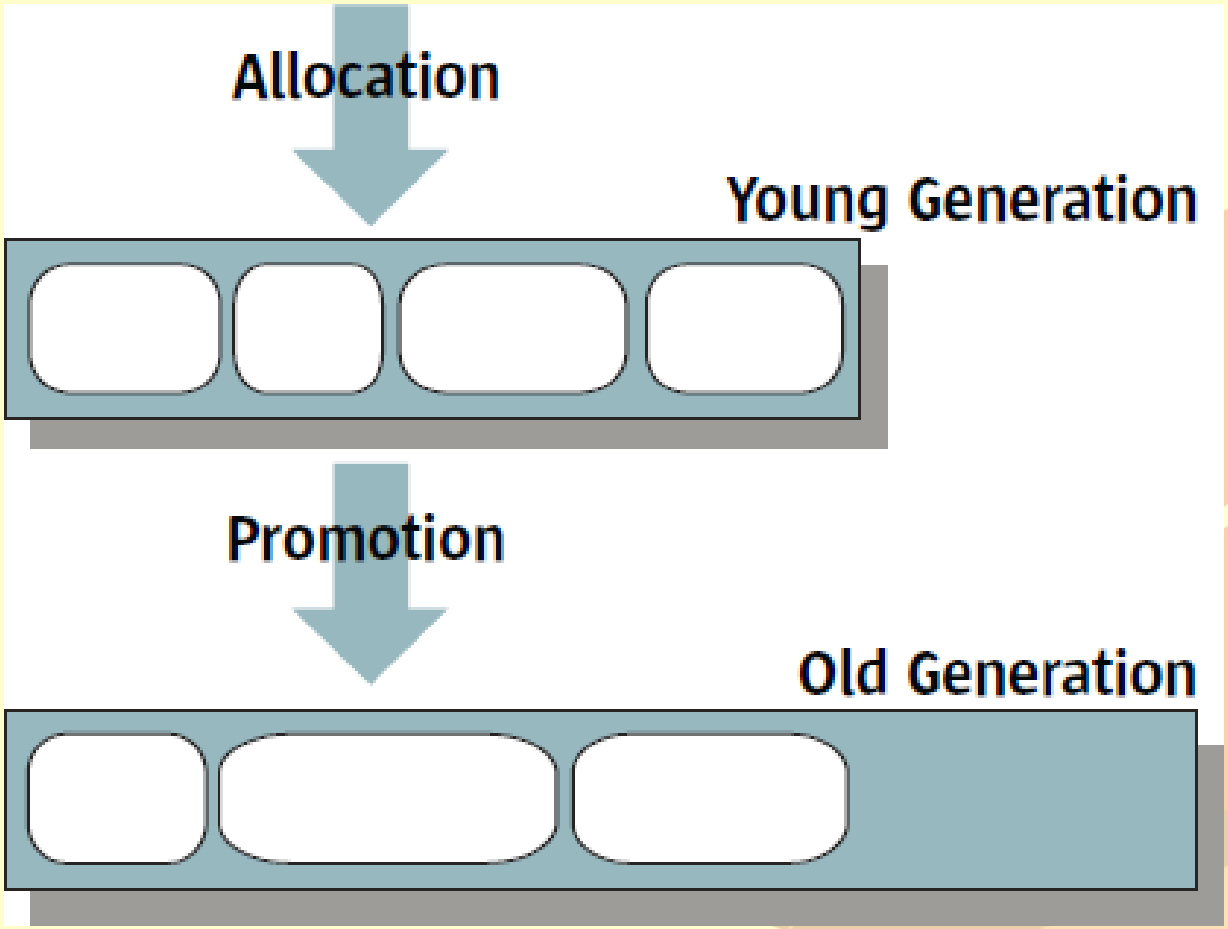
Complex and Rational Literals

- We can now use literals for complex and rational numbers, which are shorthands of `Complex` and `Rational` classes respectively.
- `5 + 6i` for complex numbers
- `5 / 6r` for rational numbers

Garbage Collection With RGenGC

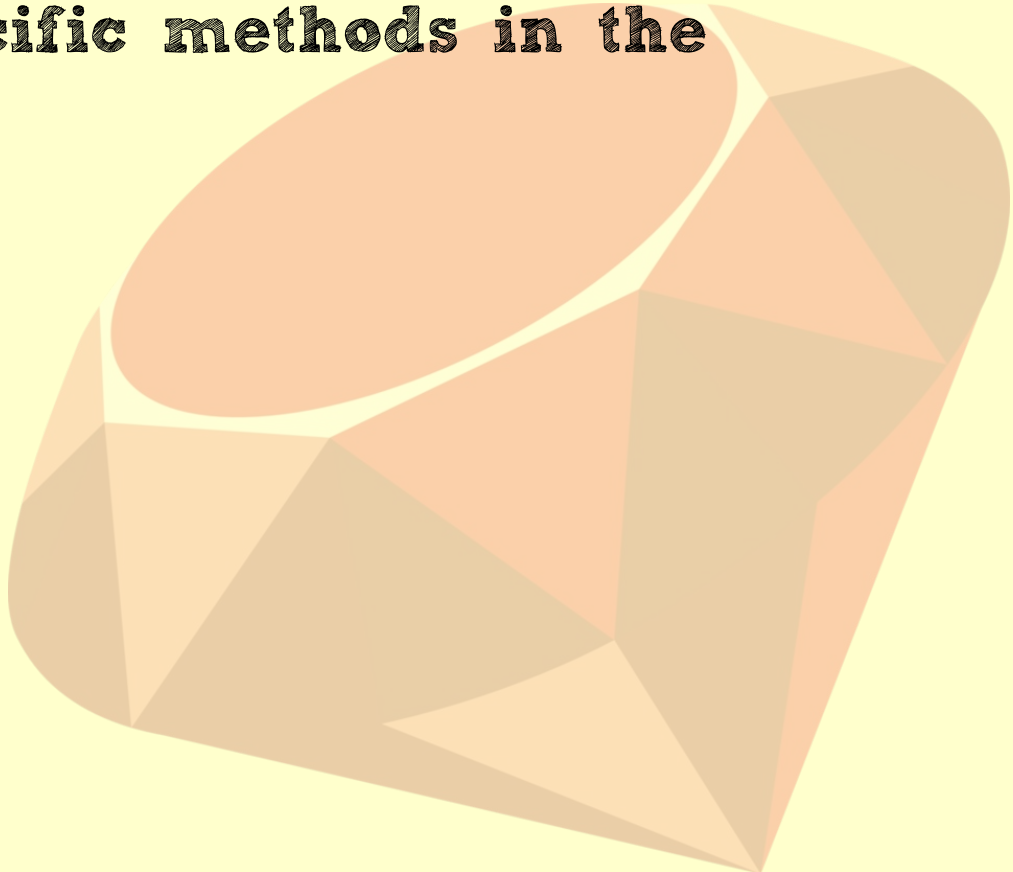
- Ruby 2.1 adds “restricted” generational garbage collection.
- Assumption is that more recently created objects usually die sooner.
- Create two generations of objects, young and old.
- On each GC, promote survived objects from young to old.
- On each GC try to mark-and-sweep only young generation.
- If no objects are young or memory's not enough, mark and sweep old generation.

Garbage collection with RGenGC



`def` return value

- `*def*` now returns a symbol of method name.
- This comes handy for explicitly specifying `*private*` modifier on specific methods in the class.



Required keyword arguments

- In Ruby 2.0 keyword arguments were introduced.
- But it was not complete! No way of specifying required arguments.
- Ruby 2.1 adds the feature. In order to specify the keyword argument as required, just *omit* the default value for it.

Refinements

- Ruby always loved monkey patching.
- Monkey patching is Evil!!!
- Problem: when monkey patching a feature, it is global!
- Refinements introduce context to monkey patching.

Refinements

```
module MyPatch
 refine String do
   def greet
     “welcome #{self}”
   end
 end
end

class MyClass
  using MyPatch

  def self.foo
   “home”.greet  # welcome home
  end
end

puts MyClass.foo # welcome home
```

Object Allocation Tracing

- Before 2.1 ObjectSpace class provided few methods for object allocation management. e.g. ::count_objects, ::each_object, ::define_finalizer...
- In 2.1, ObjectSpace adds Object Allocation Tracing. Which enables developers to trace individual objects and retrieve their allocation information.

Some Links

- http://www.sitepoint.com/look-ruby-2-1/
- http://tmml.net/ruby21-rgengc/
- https://www.youtube.com/watch?v=hVqoX4QE2O0
- http://www.confreaks.com/videos/2870-rubyconf2013-new-ruby-2-1-awesomeness-fine-grained-object-allocation-tracing

Thank You!