EXÁMEN PARCIAL – PARTE 1
Desarrollo de Software

Cesar Jesús Lara Avila

Calagua Mallqui Jairo Andre

20210279F

Facultad de Ciencias, Universidad Nacional de Ingeniería

octubre del 2023

Fundamentos de Ruby

Comando grep

Mi trabajo es implementar un comando grep simplificado, que admita la búsqueda de cadenas fijas. Se nos indica los tres argumentos que grep toma además de los indicadores que admite, entonces planteo lo siguiente:

simple_grep.builder

C: > Users > calag > Desktop > simple_grep.builder

```ruby
#-------------------------------------------------------------------------------#
# CALAGUA MALLQUI JAIRO ANDRE - DESARROLLO DE SOFTWARE - CESAR LARA
#-------------------------------------------------------------------------------#
# Comando 'grep' simplificado

def simple_grep(search_string, flags, *files)
  result = []
  files.each do |file|
    begin
      lines = File.readlines(file)
      lines.each_with_index do |line, index|
        match = line.include?(search_string) || (flags.include?('-i') && line.downcase.include?(search_string.downcase))
        match = !match if flags.include?('-v')
        match = line.chomp == search_string if flags.include?('-x')

        if match
          if flags.include?('-l')
            result << file
            break
          else
            output = flags.include?('-n') ? "#{file}:#{index + 1}: " : "#{file}: "
            output += line.chomp
            result << output
          end
        end
      end
    rescue Errno::ENOENT
      result << "File not found: #{file}"
    end
  end

  result
end
```

```ruby

# Uso del comando simple_grep
search_string = ARGV.shift
flags = ARGV.select { |arg| arg.start_with?('-') }
files = ARGV.reject { |arg| arg.start_with?('-') }

if search_string.nil? || files.empty?
  puts "Uso: ruby simple_grep.rb <search_string> [flags] <file1> <file2> ..."
else
  result = simple_grep(search_string, flags, *files)
  result.each { |line| puts line }
end
```

Tal y cual se nos indica, mi script toma una cadena de búsqueda, cero o más indicadores y uno o más archivos como argumentos de la línea de comandos.

Luego, lee el contenido de los archivos, busca las líneas que coinciden con la cadena de búsqueda y devuelve las líneas coincidentes en el orden en que fueron encontradas.

Principio abierto-cerrado

Se me brindan unas indicaciones para poder completar la clase OpenClosed, entonces:

simple_grep.builder | OpenClosed.builder

C: > Users > calag > Desktop > OpenClosed.builder

```ruby
#-------------------------------------------------------------------------------#
# CALAGUA MALLQUI JAIRO ANDRE - DESARROLLO DE SOFTWARE - CESAR LARA
#-------------------------------------------------------------------------------#

class OpenClosed
  # Este método devuelve una lista de métodos de instancia de una clase, incluyendo los privados.
  def self.meths(m)
    m.instance_methods + m.private_instance_methods
  end
end

# Sobreescribo el método 'include'
class Class
  alias_method :original_include, :include

  def include(*modules)
    if modules.any? { |m| (self.meths(self) & OpenClosed.meths(m)).any? }
      raise "Error: Intento de incluir módulo con métodos que se superponen."
    else
      original_include(*modules)
    end
  end
end

# Sobreescribo el método 'extend'
class Object
  alias_method :original_extend, :extend

  def extend(*modules)
    if modules.any? { |m| (self.singleton_class.meths(self.singleton_class) & OpenClosed.meths(m)).any? }
      raise "Error: Intento de extender objeto con módulo que contiene métodos superpuestos."
    else
      original_extend(*modules)
    end
  end
end

```

```ruby
# Define un método para controlar el evento method_added
class Class
  def method_added(method_name)
    if method_name.to_s[0] == "_"
      if self.meths(self).include?(method_name)
        raise "Error: Intento de sobreescribir el método #{method_name} con alias."
      end
    end
  end
end

# Pruebas
module MyModule
  def my_method
    puts "Mi metodo"
  end
end

class MyClass
  include MyModule
end
```

```ruby
# Esto provocará una excepción ya que el módulo se incluye con un método que se superpone.
module AnotherModule
  def my_method
    puts "Otro metodo"
  end
end

begin
  class MyClass2
    include AnotherModule
  end
rescue => e
  puts e.message
end

# Esto también provocará una excepción ya que se intenta sobreescribir el método 'my_method' con alias.
class MyClass3
  def my_method
    puts "Metodo original"
  end

  def _my_method
    puts "Metodo con alias"
  end
end

# Si se intenta modificar un objeto con un módulo que contiene métodos superpuestos provocará una excepción.
obj = Object.new

```

```ruby
begin
  obj.extend(AnotherModule)
rescue => e
  puts e.message
end
```

Como se podrá notar, el realizar este script ha sido una tarea muy tediosa para poder implementar todo lo pedido.

Me parece importante mencionar que, tal y cual se describe el principio abierto-cerrado, es muy interesante, pero quizás hay un “problema” y es que como sabemos Ruby está diseñado para permitir la modificación de clases y objetos en tiempo de ejecución, por lo tanto, no es loco pensar que habrá formas de poder eludir estas restricciones debido a la “flexibilidad” del lenguaje.

Preguntas:

Se nos pide responder a las siguientes preguntas:

1. **Si los métodos de clase son métodos de instancia, ¿qué clase contiene esos métodos de instancia?**

   Los métodos de clase son métodos de instancia de la propia clase en la que están definidos. En otras palabras, pertenecen a la clase en sí y no a las instancias de esa clase.

2. **¿Cuándo es una mala idea implementar una abstracción mediante metaprogramación?**

Se me ocurren 3 casos específicos en los que es una mala idea implementar una abstracción mediante metaprogramación:

- Cuando se utiliza en exceso y de manera innecesaria, lo que puede llevar a una complejidad innecesaria.
- Cuando la metaprogramación hace que el código sea difícil de entender y mantener.
- Cuando se utiliza en un equipo de desarrollo donde no todos los miembros tienen experiencia con metaprogramación, lo que puede dificultar la colaboración y la comprensión del código**.**

3. **¿Siempre es malo usar abstracciones en el código de prueba?**

No, no siempre es malo usar abstracciones en el código de prueba. De hecho, el uso de abstracciones en las pruebas puede ser beneficioso para mejorar la legibilidad, reutilización y mantenimiento de las pruebas. Pero es importante encontrar un "equilibrio" y realizar abstracciones en exceso, ya que esto puede hacer que las pruebas sean difíciles de entender y mantener.

4. **¿Cuál es el requisito previo más importante antes de comenzar una refactorización?**

El requisito previo más importante antes de comenzar una refactorización es tener un conjunto sólido de pruebas automatizadas (test) que cubran el código existente. Estas pruebas actúan como una especie de "salvavidas" para garantizar que la refactorización no introduzca nuevos errores o cambie el comportamiento esperado. Además, las pruebas nos proporcionan lo que conocemos como retroalimentación inmediata sobre si la refactorización se realizó de manera exitosa.

5. **¿Es mejor utilizar un enfoque de desarrollo del lado del cliente o del lado del servidor al diseñar una aplicación que implica la entrada de datos a través de formularios HTML?**

Me parece que esa decisión depende de varios factores, incluyendo la complejidad de la aplicación y los requisitos específicos.

Plantearé los enfoques tanto del lado del cliente como del lado del servidor y luego un punto medio al que llamaré "enfoque equilibrado", entonces:

- **Desarrollo del lado del cliente:** Supongamos que el uso es de JavaScript, Es eficaz para crear interfaces de usuario altamente interactivas. Puede reducir la carga del servido al realizar validaciones en el navegador y proporcionar retroalimentación en tiempo real. Pero puede ser menos seguro y depende de la capacidad del cliente para ejecutar el lenguaje.

- **Desarrollo del lado del servido:** Garantiza la seguridad y la consistencia de los datos. El servidor puede validar los datos antes de almacenarlos en la base de datos y proporcionar una capa adicional de seguridad.

- **Enfoque equilibrado:** ¿Por qué no combinar ambas estrategias?, utilizando JavaScript en el cliente para así mejorar la experiencia del usuario y el servidor para validar y procesar los datos de manera segura.

Dicho todo esto, creo que al final la elección dependerá de los requisitos específicos del proyecto y las consideraciones de rendimiento, seguridad y usabilidad.