Nombre _____

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

9 de noviembre de 2018

IIC2113 – Diseño Detallado de Software

Interrogación 2

Instrucciones:

- Sea preciso: no es necesario escribir extensamente pero sí ser preciso.
- En caso de ambigüedad, utilice su criterio y explicite los supuestos que considere convenientes.
- Responda y entregue cada pregunta en hojas separadas. Si no responde una pregunta debe entregar de todas formas la hoja correspondiente a la pregunta.
- Indique su nombre en cada hoja de respuesta.
- Esta interrogación fue diseñada para durar 120 minutos.

1. (0.8 pts) Relacione los nombres de los siguientes patrones *GoF* con su descripción correspondiente (utilice la letra que lista cada patrón):

a. Visitor
b. Template Method
c. Strategy
d. Interpreter
e. Mediator
f. State
g. Chain of Responsability

c Encapsula la implementación de un algoritmo en una clase
d Dado un contexto, especifica cómo evaluar expresiones
__ Captura y restaura el estado interno de un objeto
__ Agrega comportamiento a un objeto de manera dinámica
f Altera el comportamiento de un objeto cuando cambia su estado
b Delega la implementación de los pasos de un algoritmo a subclases
e Define una comunicación simplificada entre clases
__ Encapsula una acción como un objeto
a Define una nueva operación para una clase sin modificarla
__ Accede de forma secuencial los elementos de una colección
g Genera una jerarquía dinámica de objetos capaces de procesar una solicitud

Nombre _____

2. (0.8 pts) Considere el siguiente artículo sobre GitLab escrito por Aricka Flowers:

Why we use Ruby on Rails to build GitLab

When our Co-founder and Engineering Fellow Dmitriy Zaporozhets decided to build GitLab, he chose to do it with Ruby on Rails, despite working primarily in PHP at the time. GitHub, a source of inspiration for GitLab, was also based on Rails, making it a logical pick considering his interest in the framework. GitLab CEO Sid Sijbrandij thinks his co-founder made a good choice:

"It's worked out really well because the Ruby on Rails ecosystem allows you to shape a lot of functionality at a high quality," he explained. "If you look at GitLab, it has an enormous amount of functionality. Software development is very complex and to help with that, we need a lot of functionality and Ruby on Rails is a way to do it. Because there's all these best practices that are on your happy path, it's also a way to keep the code consistent when you ship something like GitLab. You're kind of guided into doing the right thing."

Depending on useful gems

Ruby gems play an integral role in the building of GitLab, with it loading more than a thousand non-unique gems, according to Sid. Calling the Ruby on Rails framework "very opinionated," he thinks it's a strong environment in which to build a complex app like GitLab.

"There's a great ecosystem around it with gems that can make assumptions about how you're doing things and in that regard, I think the Ruby on Rails ecosystem is still without par," he says. "If you look at our Gemfile, it gives you an indication of how big the tower is of dependencies that we can build on. Ruby on Rails has amazing shoulders to stand on and it would have been much slower to develop GitLab in any other framework."

Overcoming challenges

All of this is not to say there haven't been challenges in building GitLab with Ruby on Rails. Performance has been an issue that our developers have made strides to improve in a number of ways, including rewriting code in Go and using the Vue framework. The latter is being used to rewrite frequently accessed pages, like issues and merge requests, so they load faster, improving user experience.

Go is being used to address other issues affecting load times and reduce memory usage.

"Ruby was optimized for the developer, not for running it in production," says Sid. "For the things that get hit a lot and have to be very performant or that, for example, have to wait very long on a system IO, we rewrite those in Go … We are

still trying to make GitLab use less memory. So, we'll need to enable multithreading. When we developed GitLab that was not common in the Ruby on Rails ecosystem. Now it's more common, but because we now have so much code and so many dependencies, it's going to be a longer path for us to get there. That should help; it won't make it blazingly fast, but at least it will use less memory."

Adding Go to GitLab's toolbox led to the creation of a separate service called Gitaly, which handles all Git requests.

Building on GitLab's mission

The organized, structured style of Ruby on Rails' framework falls in line with our core mission. Because Rails is streamlined, anyone can jump into GitLab and participate, which made it especially attractive to Sid from the start.

"Our mission is that everyone can contribute," he explains. "Because Ruby on Rails is really opinionated about which pieces go where, it's much easier for new developers to get into the codebase, because you know where people have put stuff. For example, in every kitchen you enter, you never know where the knives and plates are located. But with Ruby on Rails, you enter the kitchen and it's always in the same place, and we want to stick to that.

"I was really encouraged when I opened the project and saw it for the first time a year after Dmitriy started it. I opened it up and it's idiomatic Rails. He followed all the principles. He didn't try to experiment with some kind of fad that he was interested in. He made it into a production application. Dmitriy carefully vetted all the contributions to make sure they stick to those conventions, and that's still the case. I think we have a very nice codebase that allows other people to build on top of it. One of our sub-values is boring solutions: don't do anything fancy. This is so that others can build on top it. I think we've done that really well … and we're really thankful that Ruby has been such a stable, ecosystem for us to build on."

a) En base a la lectura, comente ventajas y desventajas al utilizar un *framework* en un proyecto de *software*.

- En el texto se dice que *GitHub*, la competencia de *Gitlab*, utiliza el mismo *framework*. Esto presenta 2 ventajas de utilizar el *framework*:
    - En teoría se podrían implementar todas las funcionalidades que tiene la competencia.
    - La transición al contratar gente que ha trabajado en *GitHub* debería ser más fácil que si se utilizara otro *framework*. Sin embargo, este último punto es un "arma de doble filo" porque podría implicar lo contrario también.
- Sid explica que al utilizar un *framework* como *RoR* uno "está guiado a hacer las cosas bien." Esto se debe a que la herramienta incluye buenas prácticas para mantener consistencia, que están respaldadas por una comunidad.

- En el texto explican que utilizar un *framework* como *RoR* también tiene desventajas, como comprometer el rendimiento del programa. Esto se debe a que cada *framework* potencia algunos conceptos en desmedro de otros.
- Por otra parte, un *framework* cambia constantemente (como se ejemplifica con el soporte de *RoR* para *multithreading*), lo que implica que hay que mantenerse actualizado para beneficiarse de las últimas mejoras, lo cual no siempre es factible. También, estos cambios son fomentados por una comunidad que utiliza el *framework*, lo que puede resultar en que no sea lo mejor para una empresa en particular que lo utiliza.
- Utilizar un *framework* y seguir sus convenciones hace mucho más fácil que nuevos desarrolladores entiendan el código. Esto se ejemplifica en que Sid pudo entender fácilmente el proyecto luego de un año que Dmitriy lo empezara ya que siguió al pie de la letra las convenciones de *RoR*.

b) Comente en base a conceptos vistos en el curso la siguiente frase: “Ruby was optimized for the developer, not for running it in production.”

Para algunos proyectos es más importante la mantenibilidad, legibilidad y extensibilidad del *software* más que el rendimiento o escalabilidad de este. Esto se puede deber a la necesidad de tener una gran velocidad de desarrollo que sea capaz de adaptarse a las necesidades del negocio rápidamente. Por esta razón existen lenguajes de programación con mayor nivel de abstracción y con sintaxis más intuitiva que permiten a los desarrolladores entender y programar con mayor rápidez, a pesar de ocultar configuraciones que se podrían utilizar para optimizar una solución en particular.

Nombre _____

3. (1.0 pts) Para cada uno de los siguientes *code smells* indique a qué familia pertenecen, explique a qué se refiere y cuáles son los beneficios al evitar su presencia en el código:

a) *Refused Bequest*
- *Object-Orientation Abusers*
- Una subclase utiliza solamente algunas de las propiedades y/o métodos heredados de sus padres.
- Código más claro y organizado

b) *Shotgun Surgery*
- *Change Preventers*
- Hacer cualquier modificación resulta en muchos cambios pequeños en distintas clases
- Mejora la organización del código, disminuye código duplicado, simplifica la mantención

c) *Data Clumps*
- *Bloaters*
- Diferentes partes de una aplicación contienen una definición recurrente de variables
- Código más fácil de entender y organizar, menos código

d) *Inappropriate Intimacy*
- *Couplers*
- Una clase utiliza los métodos y atributos internos de otra clase
- Código más organizado
- Simplifica el mantenimiento del código

e) Lazy Class
- *Dispensables*
- Una clase no cumple un rol que realmente justifique su existencia
- Disminuye la cantidad de código
- Simplifica la mantención de la aplicación

Nombre _____

4. (1.5 pts) Considere el siguiente código:

```ruby
# A movie that can be rented
class Movie
  REGULAR = 0
  NEW_RELEASE = 1
  CHILDRENS = 2

  attr_reader :title, :price_code

  def initialize(title, price_code)
    @title, @price_code = title, price_code
  end
end
```

```ruby
# A rental event
class Rental
  attr_reader :movie, :days_rented

  def initialize(movie, days_rented)
    @movie, @days_rented = movie, days_rented
  end
end
```

Nombre _____

```ruby
# A customer of the store
class Customer
  attr_reader :name

  def initialize(name)
    @name = name
    @rentals = []
  end

  def add_rental(rental)
    @rentals << rental
  end

  def summary
    total_amount, frequent_renter_points = 0, 0
    result = "Rental records for #{@name}\n"

    @rentals.each do |rental|
      this_amount = 0
      # determine amounts for each line
      case rental.movie.price_code
      when Movie::REGULAR
        this_amount += 2
        this_amount += (rental.days_rented - 2) * 1.5 if rental.days_rented > 2
      when Movie::NEW_RELEASE
        this_amount += rental.days_rented * 3
      when Movie::CHILDRENS
        this_amount += 1.5
        this_amount += (rental.days_rented - 3) * 1.5 if rental.days_rented > 3
      end

      # add frequent renter points
      frequent_renter_points += 1

      # add bonus for a two day new release rental
      if rental.movie.price_code == Movie::NEW_RELEASE && rental.days_rented > 1
          frequent_renter_points += 1
      end

      # show detail for this rental
      result += "\t" + rental.movie.title + "\t" + this_amount.to_s + "\n"
      total_amount += this_amount
    end

    # add footer lines
    result += "Amount owed is #{total_amount}\n"
    result += "You earned #{frequent_renter_points} frequent renter points"
    result
  end
end
```

a) Identifique los *code smells* presentes, detallando a qué familia pertenecen. Para cada uno especifique por qué es un problema para el código.

- (*Customer#21*) *Switch Statements – Object-Orientation Abusers*
    - Dificulta realizar cambios en la lógica para calcular la cantidad adeudada para distintos tipos de películas (como también agregarlos).

Nombre _____

- (*Rental*, *Movie*) *Data Class – Dispensables*
  - Dificulta la legibilidad del código ya que la lógica que utiliza estos datos está en otras clases con más responsabilidades.
- (*Customer#14*) *Long Method – Bloaters*
  - Complejiza la lectura del código con un método que tiene más de una responsabilidad definida y oculta código duplicado
- (*Customer#21*) *Feature Envy – Couplers*
  - Dificulta la legibilidad y modificación del código, ya que para calcular *this_amount* se accede a más información de *rental* que de *customer*, por lo que no debería estar en esa clase.
- (*Customer#20,32,35,40,45*) *Comments – Dispensables*
  - Estos comentarios no aportan ya que el código es suficientemente claro, lo que hace que sea más tedioso leerlo.

b) Proponga una versión mejorada del código que no presente los *code smells* detectados. Puede utilizar pseudo-código, pero debe ser explicito en lo que quiere expresar.

```ruby
class Movie
  attr_reader :title

  def initialize(title)
    @title = title
  end

  def get_amount(days_rented)
  end

  def get_points(days_rented)
    1
  end
end
```

```ruby
class ChildrensMovie < Movie
  def get_amount(days_rented)
    amount = 1.5
    amount += (days_rented - 3) * 1.5 if days_rented > 3
    amount
  end
end
```

Nombre _____

```ruby
class NewRelease < Movie
  def get_amount(days_rented)
    days_rented * 3
  end

  def get_points(days_rented)
    days_rented > 1 ? 2 : 1
  end
end
```

```ruby
class RegularMovie < Movie
  def get_amount(days_rented)
    amount = 2
    amount += (days_rented - 2) * 1.5 if days_rented > 2
    amount
  end
end
```

```ruby
# A rental event
class Rental
  attr_reader :movie, :days_rented

  def initialize(movie, days_rented)
    @movie, @days_rented = movie, days_rented
  end
end
```

Nombre _____

```ruby
class Customer
  attr_reader :name

  def initialize(name)
    @name = name
    @rentals = []
  end

  def add_rental(rental)
    @rentals << rental
  end

  def summary
    total_amount, frequent_renter_points = 0, 0
    result = "Rental records for #{@name}\n"

    @rentals.each do |rental|
      movie = rental.movie
      this_amount = movie.get_amount(rental.days_rented)
      frequent_renter_points += movie.get_points(rental.days_rented)

      # show detail for this rental
      result += "\t" + movie.title + "\t" + this_amount.to_s + "\n"
      total_amount += this_amount
    end

    # add footer lines
    result += "Amount owed is #{total_amount}\n"
    result += "You earned #{frequent_renter_points} frequent renter points"
    result
  end
end
```

Nombre _____

5. (0.5 pts)

a) Explique 3 beneficios y 2 costos de utilizar *tests* unitarios para un proyecto.

Beneficios:

- Permiten hacer cambios al código con algún nivel de seguridad
- Ayudan a entender y diseñar funcionalidades
- Sirven como apoyo a la documentación

Costos:

- Requieren tiempo de diseño, creación y mantención
- No todos agregan el mismo valor al código

b) ¿Cuál es la diferencia entre *mock* y *stub*? ¿Cuál es el propósito principal de aplicar estos conceptos al realizar *tests*?

Mock: imitación de un objeto, de la cual se espera cierto comportamiento durante un *test*. Si esta expectativa no se cumple el *test* falla.
Stub: imitación de un objeto, la cual tiene resultados predefinidos para ciertas invocaciones.

El propósito principal de estas herramientas es eliminar la dependencia entre *tests*, para así hacerlos realmente unitarios.

Nombre _____

6. (1.4 pts)

Diseñe *tests* unitarios para las 2 funciones implementadas en el siguiente extracto de código. No es necesario que implemente código, aunque puede utilizar pseudo-código si lo desea. Se pide como mínimo que describa las pruebas con una breve descripción del escenario (qué se quiere probar), un contexto (configuración inicial del escenario) y un resultado esperado. Debe ser explícito en todas condiciones del contexto de sus *tests*. Considere que se descontará puntaje por *tests* redundantes.

```ruby
class UF < ApplicationRecord
  def self.get(date = Time.zone.today)
    uf = UF.find_by(date: date)
    if uf.blank?
      published = value_published?(date)
      save_from_api(get_from_api(date.year)) if published || UF.empty?
      uf = UF.find_by(date: date)
      return uf.value if uf
      logger.error 'Error: UF not available' if uf.blank? && published
      uf = UF.where('date <= ?', date).order(:date).last
    end
    uf.value
  end

  def self.value_published?(date = Time.zone.today)
    today = Time.zone.today
    date_month = date.month
    date_day = date.day
    months_difference = months_difference(today, date)
    return false if months_difference > 1
    if date_day > 9 && (months_difference == 1 || months_difference == 0)
      return false
    end
    true
  end

  def self.get_from_api(year = Time.zone.now.year)
    # Solicita los valores de la UF publicados en un año
  end

  def self.save_from_api(ufs)
    # Guarda los valores de la UF recibidos en el formato de la API
  end

  def self.months_difference(from, to)
    # retorna un número que representa la diferencia de meses entre (to - from)
  end
end
```

Nombre _____

Lo importante de este ejercicio es que no se detallen *tests* redundantes y que se tomen las medidas necesarias para que los *tests* sean unitarios (es decir que las pruebas de una función no dependan de los resultados de otra función que tiene pruebas).