Nombre _____

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

14 de septiembre de 2018

IIC2113 – Diseño Detallado de Software

Interrogación 1

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 150 minutos.

1. (1.0 pts)

a. Explique a qué se refieren los conceptos "cohesión" y "acoplamiento" en el diseño de *software*. ¿Existe alguna correlación entre ellos?

Cohesión se refiere al grado en que los elementos de un módulo/componente permanecen juntos.

Acoplamiento se refiere al nivel de interdependencia entre módulos/componentes de *software*.

Un *software* con alto acoplamiento implica baja cohesión.

b. ¿Cómo afecta un *software* altamente acoplado en la extensión de este?

Un software altamente acoplado significa que tiene gran dependencia entre sus módulos/componentes. Esto implica que si se desea realizar una modificación se deberá intervenir varias partes del código, lo que hace que sea más difícil agregar funcionalidades.

c. Explique qué problemas conlleva el mal uso de herencia en la programación orientada a objetos.

- Aumenta el acoplamiento entre clases de una jerarquía, lo que dificulta realizar modificaciones
- Oculta lógica del sistema, lo que puede dificultar su lectura y análisis

d. *Python* ofrece el módulo ***urllib*** que sirve para trabajar con URLs (abrir, leer, *parsear*). También, existe la librería externa ***requests*** que ofrece las mismas funcionalidades y basa su implementación en ***urllib***. Sin embargo, ***requests*** es una de las librerías con más descargas en *Python* e incluso tiene mayor popularidad que ***urllib***. Basado en un principio fundamental del diseño, comente a qué se puede deber esta situación.

Probablemente la popularidad de ***requests*** se debe a que presenta un mejor nivel de abstracción que ***urllib***, lo que facilita su uso.

2. (0.8 pts)

Considere el siguiente contexto:

Mercado Público es un sitio web del gobierno de Chile donde las instituciones que dependen del Estado publican licitaciones para satisfacer sus necesidades. En ese contexto, *Chile Crece Contigo* (parte del Ministerio de Desarrollo social) ha generado una licitación para el desarrollo de un *software* de gestión interna.

Considerando el modelo 4+1, señale la vista y diagrama más atingente para presentar a los siguientes actores. Justifique su elección.

i. Analista de negocio de *Chile Crece Contigo*

Vista de procesos – diagrama de actividad: representa las reglas y lógica del negocio

ii. Jefe tecnológico de *Chile Crece Contigo*

Vista de implementación – diagrama de componentes: entrega una visión de alto nivel del sistema

iii. Potenciales usuarios del sistema

Vista de lógica – diagrama de secuencia: representa las tareas que realizará la aplicación y cómo están involucrados los distintos actores

iv. Arquitecto de software de *Chile Crece Contigo*

Vista física – diagrama de despliegue: representa cómo de compone el *hardware* y *software* del sistema

3. (1.2 pts)

a. Considere el siguiente extracto de código:

```
1   class FileParser
2     def initialize(client, file)
3       @client = client
4       @file = file
5     end
6
7     def parse
8       # which format uses the client?
9       case @client.file_format
10        when :xml
11          parse_xml
12        when :csv
13          parse_csv
14      end
15
16      @client.last_parse = Time.now
17      @client.save!
18    end
19
20    private
21
22    def parse_xml
23      # parse xml
24    end
25
26    def parse_csv
27      # parse csv
28    end
29  end
```

Identifique el principio *SOLID* predominante que no se respeta y modifique el código para que sí lo haga.

El principio que no se respeta es "*Open-Closed*", ya que si se quisiera agregar otro *parser* se debería modificar la clase *FileParser*.

Nombre _____

```ruby
class FileParser
  def initialize (client, file, parser)
    @client = client
    @file = file
    @parser = parser
  end

  def parse
    @parser.parse
    @client.last_parse = Time.now
    @client.save!
  end
end

class XMLParser
  def parse
    # Parse XML
  end
end

class CSVParser
  def parse
    # Parse CSV
  end
end
```

b. Explique a qué se refiere el principio *Liskov Substitution*. Acompañe su explicación junto con un ejemplo concreto en *ruby* (o *pseudo-ruby*) que no cumpla con el principio. Luego, realice una versión modificada de su ejemplo que sí cumpla con el principio.

Este principio se refiere a que en una correcta jerarquía de herencia cualquier clase hija podría sustituir a su clase base.

Nombre _____

```ruby
class Bird
  def fly
    # fly
  end

  def eat
    # eat
  end
end

class Bluejay < Bird
end

class Ostrich < Bird
  def fly
    raise "Cannot fly"
  end
end
```

a - ejemplo que no respeta LSP

```ruby
class Bird
  def eat
    # eat
  end
end

class FlyingBird < Bird
  def fly
    # fly
  end
end

class FlightlessBird < Bird
end

class Bluejay < FlyingBird
end

class Ostrich < FlightlessBird
end
```

b - ejemplo que respeta LSP

Nombre _____

4. (3.0 pts)

Considere el siguiente contexto:

"Alan busca cumplir el sueño de toda su vida: dedicarse a la composición de música electrónica, para ser usado como base en canciones de trap latino. A raíz de esto decidió dejar su trabajo *full-time* como *Senior SQL Injector* en Radius. Sin embargo, como este empleo no estaba bien remunerado, Alan no contaba con el dinero suficiente para adquirir un software para componer música. Esto lo llevó a la siguiente determinación: desarrollar su propio *audio editing software*, ajustado a sus necesidades musicales. Este editor —bautizado como *Droptable*— será una aplicación de escritorio, de código abierto y desarrollada en Electron. Este es un *framework* que permite crear aplicaciones multiplataforma, con el uso de tecnologías de la web: HTML, CSS y JavaScript.

Droptable debe soportar *multitracking edition*. En otras palabras, una canción estará compuesta por uno o más *tracks*, donde cada uno de ellos representa una fuente de sonido, que generalmente es un instrumento musical. El principal beneficio de este sistema es la posibilidad de grabar cada instrumento de forma aislada, para luego conseguir un sonido cohesivo con la ayuda del software. Además, cada uno de estos *tracks* está compuesto por notas musicales. Para una primera versión de Droptable, estas notas sólo tendrán un nombre (e.g. *do mayor* en piano), una duración en milisegundos, una altura (representado con una frecuencia en Hertz), una posición temporal en el *track* y un archivo de formato FLAC con el sonido.

Una funcionalidad esencial que Droptable debe ofrecer es la opción de aplicar distintos filtros y efectos como, por ejemplo, reducción de ruido, normalización, ecualización y *reverb*. Para no reinventar la rueda, la mayoría de los efectos musicales se obtendrá con el uso de la popular librería *brytiago.js*. Sin embargo, Alan deberá ajustar estas funciones de la librería para poder aplicarlas tanto en una nota en particular, como en una sección específica del *track*, o en una canción de forma completa.

Entre otros *features* que mejoran la usabilidad, Droptable debe ofrecer la posibilidad de copiar, cortar y pegar secciones de un *track*. Además, para corregir potenciales errores, debe contar con la funcionalidad de deshacer y rehacer las acciones realizadas. Junto con esto, el software debe ser capaz de importar y exportar archivos en distintos formatos de audio (*e.g.* OGG, WAV, MP3, OZUNA, FLAC), o con distintos niveles de calidad del sonido, para trabajar en un formato común.

Al crear un primer prototipo de la aplicación, Alan notó un problema: cada uno de los archivos en formato FLAC, asociados a una nota en particular, tiene un tamaño cercano a la centena de kilobytes. De esta forma, al crear un *track* de más de diez minutos, Droptable se volvía inutilizable; esto ocurría aunque se estuviese trabajando únicamente con la misma nota musical.

Nombre _____

Motivado por este inconveniente a solucionar, Alan también evaluó que, en una fase posterior al primer MVP, sería una buena idea extender el *back-end* de la aplicación para registrar métricas sobre el rendimiento de Droptable. Esto permitiría saber, por ejemplo, cuánto tiempo y memoria está tomando aplicar cada uno de los efectos y filtros. La idea es desarrollar estas funcionalidades una vez que la primera parte esté completa."

a. Identifique exactamente 5 patrones de diseño que podrían ser utilizados en la resolución del problema enunciado. Además, por cada uno de ellos justifique de forma concisa su elección, basada en los requisitos de la aplicación.

   Algunas alternativas de patrones son:

   - **Façade**: Para ajustar las funciones de la librería brytiago.js y poder aplicarlas en la lógica de la aplicación. Bajo esta idea, se podría diseñar una interfaz que sólo se preocupe de obtener el subconjunto de funcionalidades relevantes al problema, para mantener el código limpio. Otro punto a considerar es que esta interfaz ofrezca un nivel de abstracción adecuado al contexto del problema. Por ejemplo, podría mezclar varias funciones de bajo nivel de la librería, en un método que sí tenga sentido en Droptable. Esto hará que el código sea más fácil de mantener.
   - **Composite**: Para formar estructuras en forma de árbol a partir de objetos básicos (i.e. notas musicales) y aplicar funciones en estructuras más complejas, como en una sección específica de un track, o una canción de forma completa. El beneficio de aplicar este patrón es que ofrece una interfaz más limpia, ya que las funciones sacadas desde la librería pueden ser llamadas en niveles distintos. En otras palabras, esto permite trabajar con estructuras complejas como si fueran objetos individuales.
   - **Command/Memento**: Para implementar la posibilidad de deshacer y rehacer las acciones realizadas en Droptable. Para lograr esto, existen dos alternativas: por acciones o por estados. Si esta funcionalidad se consigue a través de acciones, el patrón Command es de utilidad para manejar este stack de acciones. Y si se hace con el almacenamiento de estados —que podrían ser vistos como snapshots de una canción—, entonces el patrón a utilizar es Memento.
   - **Adapter**: Para aceptar distintos formatos de audio (e.g. OGG, WAV, MP3, FLAC) y cargar las canciones en FLAC, que es el formato en común de Droptable. Además, este patrón también es útil para transformar las canciones entre diferentes bit rates, lo que permite manejar distintos niveles de calidad del sonido. El beneficio de este patrón es que permite, entonces, encapsular detalles de implementación relacionados a la conversación de datos.
   - **Flyweight**: Para conseguir un uso más eficiente de la RAM, al compartir las partes en común de un objeto en la aplicación. En el contexto del enunciado, esto se podía notar en el problema que Alan sufrió al crear su primer prototipo de Droptable. Cada objeto en memoria —que simbolizaba una nota— mantenía un archivo FLAC. Por lo tanto, para alivinar el uso de memoria, el

propósito de este patrón es que todas las notas que produzcan un mismo sonido carguen el mismo archivo FLAC. Dicho de otra forma, cada instancia de una nota debería preocuparse de almacenar sólo lo necesario como, por ejemplo, su posición en el track.

- **Decorator**: Para extender el back-end después de terminar el MVP. La idea es no hacerle modificaciones a las funciones originales, sino decorarlas con nuevas funciones que permitan registrar métricas acerca del rendimiento de Droptable. El beneficio es que estas funciones serán, de cierta forma, agnósticas a la funciones que serán desarrolladas en la primera fase. Ellas sólo buscarán medir y registrar cuántos recursos utiliza un efecto o un filtro en particular.

b. A partir de los patrones identificados, escoja 2 de ellos y realice un diagrama de clases *UML 2* que refleje la combinación en **el contexto de la aplicación descrita**. Se debe realizar un solo diagrama que refleje la interacción cohesiva entre ambos patrones. Su diagrama debe detallar todo lo necesario para implementar la solución, además de señalar claramente dónde se implementa cada patrón en el diagrama y explicar el rol de cada participante.