Ejercicio 1 (Ex 2021-2)

Un desarrollador experimentado está explicando a un colega algunos aspectos sobre el diseño del producto. A continuación se presenta una transcripción de lo que le dice reemplazando algunas palabras por números. Cada una de esas palabras corresponde a un patrón de diseño conocido.
Para cada uno de los números 1 al 8 encuentre el nombre correcto del patrón de diseño que corresponda y explique en no más de 3 líneas el por qué lo eligió.

Para los algoritmos criptográficos, afortundamente, teníamos ya una clase que alguien había desarrollado. Por supuesto que no tenía la interfaz correcta pero lo solucionamos rápidamente con un **1**. Para facilitar a la empresa externa, el conectarse con nuestro producto, simplemente le proporcionamos una **2** de modo de que pueda tener una visión simplificada sin tener que entender todos los detalles de la aplicación. Mas complejo resultó la implementación del esquema de "undo's" ilimitados que el cliente insistió en tener, finalmente optamos por usar para ello un patrón **3** y así poder manejar las operaciones como si fuesen objetos. Evidentemente que el repositorio de datos se manejó como una instancia única para lo cual el patrón **4** fue de gran ayuda.

Una complicación extra de este software es que el conjunto de objetos es ligeramente distinto dependiendo del país donde será utilizado. Esto nos hizo pensar en utilizar una **5** de modo que la creación de nuevos objetos no quede tan amarrada al código.

El procedimiento, de 8 pasos, para lograr una conexión con la base de datos, en términos generales, era siempre el mismo, aunque algunos de los pasos podían ser distintos dependiendo de la situación. Aquí el patrón **6** nos cayó del cielo.

También queríamos que nuestros algoritmos de inteligencia artificial pudieran ser cambiados con facilidad en el futuro sin afectar mucho al código por lo que nos decidimos por usar el patrón **7**.
Finalmente, una cosa bastante astuta fue darse cuenta que, al final de cuentas, el documento tiene una estructura de árbol, por lo que es posible aplicar la misma operación sobre el documento completo o sobre subárboles del documento, cosa que por supuesto estaba pintado para usar el **8.**

Ejercicio 2 (I2 2021-2)

Se tiene el siguiente código Ruby que permite calcular el salario neto en distintos países. *amount* es el sueldo bruto y lo que cambia es el tratamiento de los impuestos en cada país:

```
def net_salary(amount, country)
    taxes = case country
         when "Ukraine"
             (amount * 0.05) + 313
         when "U.S."
             (amount * 0.2) + 100
         when "Poland"
             amount * 0.3
         else
             0
         end
    amount - taxes
end
```

Por ejemplo:

```
net_salary(1000, "Poland")   => 700.0
net_salary(1000, "Ukraine")  => 637.0
```

a) Efectue un proceso de *refactoring* de este código utilizando el patrón de diseño "Estrategia" (Strategy) para que el código sea mas extensible. Escriba un segmento de código que ilustre el funcionamiento para calcular el sueldo en Polonia y en Ukrania como lo hacía el antiguo código.

b) Haga diagramas de clases y de secuencia que ilustren la forma en que funciona el código de ejemplo de la parte a).

Ejercicio 3 (Ex 2021-2)

A continuación, se presenta un código Ruby en el cual se han utilizado 3 patrones de diseño distintos

```
class C1
  def m1
    raise 'abstract method'
  end
end

class C2
  def m2(f)
    raise 'abstract method'
  end
  def m3
    raise 'abstract method'
  end
end

class C3
  def m4
    raise 'abstract method'
  end
end

class C4 < C1
  @@x = C4.new
  def self.x
    return @@x
  end
  def m1
    return C5.new
  end
  private_class_method :new
end
```

```
class C5 < C2
  def initialize
    @s = ''
    @arr = []
  end
  def s=(new_s)
    @s = new_s
    puts @s
    m3()
  end
  def m2(f)
    @arr.push(f)
  end
  def m3
    for f in @arr do
      f.m4
    end
  end
end

class C6 < C3
  def initialize(msg)
    @msg = msg
  end
  def m4
    puts @msg
  end
end
```

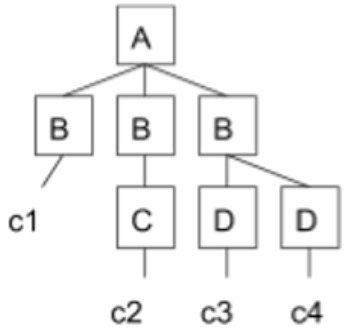
```
one = C6.new("software engineering")
two = C6.new("design partterns")
three = C4.x
four = three.m1
four.m2(one)
four.m2(two)
four.s = "I hate"
four.s = "I love"
```

a) Dibuje un diagrama de clases que corresponda a este código (incluya métodos y atributos).
b) Identifique a continuación cada uno de los 3 patrones utilizados. Para cada uno de ellos indicar el nombre del patrón y los elementos que actúan en él. Por ejemplo, si se tratara del patrón método plantilla, decir cual es la clase abstracta, cuales las clases concretas y cual el método plantilla.
c) ¿Cual es el output que se produce cuando este programa corre? Explíquelo haciendo un diagrama de secuencia desde que inicia hasta que termina la ejecución del código

Ejercicio 4 (I2 2021-1)

Un documento xml tiene estructura de árbol. Un elemento xml (etiqueta) puede contener otros elementos xml o bien solo contenido. Por ejemplo, el árbol de la figura a la derecha corresponde al documento xml de la izquierda

```
<A>
  <B> c1 </B>
  <B>
    <C> c2 </C>
  </B>
  <B>
    <D> c3 </D>
    <D> c4 </D>
  </B>
</A>
```



a) Utiliza el patrón de diseño conocido como *Composite* para implementar una clase XmlTree con un método *display* que genere la versión en texto del árbol. Haga un diagrama de clases con todo lo necesario (métodos y atributos) y escriba (Ruby) el método display. No te preocupes por identación o espacios, puedes mostrar el árbol anterior como:

<A> <B> c1 </B> <B> <C> c2 </C> </B> <B> <D> c3 </D> <D> c4 </D> </B> </A>

b) Escribe un trozo de código Ruby que construye el árbol de la figura como un objeto compuesto y luego lo imprime como documento *xml* usando el método display

Ejercicio 5 (I2 2021-1)

Supon que se dispone de una componente gráfica *TextView* (clase) capaz de desplegar un texto en una ventana gráfica. El problema es que necesitamos ventanas con 3 distintos tipos de bordes (Plain, 3D, y Fancy) y también poder agregar *scrollbars* tanto verticales como horizontales (una, otra, o ambas).

Un programador ha propuesto una solución que se basa en generar subclases con todas las especializaciones posibles de *TextView*. Así tendríamos, por ejemplo, las siguientes subclases:

TextView-Plain
TextView-Plain-Horizontal
TextView-Plain-Vertical
TextView-Plain-HorizontalVertical
TextView-Fancy
TextView-Fancy-Horizontal
TextView-Fancy-Vertical
TextView-Fancy-HorizontalVertical

a) Propone un mejor diseño basado en un patrón que estudiamos en clases. Dibuje el diagrama de clases UML que muestre la solución completa y explique por qué este diseño sería superior al propuesto

b) Escribe un pequeño programa Ruby de prueba que muestre como se usaría su solución para obtener una ventana 3D con *scrollbars* horizontal y vertical y otra ventana *Fancy* con *scrollbar* vertical solamente. No necesitas escribir el código de la definición de las clases del diagrama mostrado en a)