Recuperatorio I2: Ingeniería de Software
Semestre 2- 2022

Nombre Completo: _____

Sección (profesor): _____

Pregunta I (20 puntos). En clases estudiamos el modelo C4 usado para describir la arquitectura de un producto de software. **Cada inciso vale 5 puntos.**

A. (10 pts) Describa en la forma más completa posible el diagrama de Contexto (no se piden ejemplos no cual es su objetivo sino qué es lo que se dibuja aquí)
B. (5 pts) ¿Cuál es la diferencia entre un container y una componente? ¿Por qué se necesitan diagramas distintos ? Un microservicio corresponde a una componente, a un container o a ninguno de ellos.
C. (5 pts) ¿Qué representan las líneas que conectan a los containers y a las componentes ?

Solución

A. En el diagrama de contexto se presenta un rectángulo que representa todo el sistema o aplicación y un conjunto de íconos que lo rodean que representa a quienes interactúan con el . Estos son de dos tipos: usuarios (pueden haber de varios tipos) y sistemas externos o APIS que nuestra aplicación va a consumir. Las líneas que van entre los usuarios y APIS van etiquetadas con el tipos de información de entrada o salida del sistema.
B. El container representa un subsistema completo implementado de alguna forma o con alguna tecnología en particular y normalmente comprende muchas componentes de software. Ejemplos de containers son por ejemplo un motor de bases de datos o un servidor Web. Una componente es una pieza de software que está caracterizada por su interfaz y que puede ser intercambiada sin problemas si se preserva la interfaz. Un microservicio corresponde a una forma particular de implementar una componente en que el acoplamiento es solo por medio de protocolos de internet.
C. Las líneas representan comunicación pero en este caso se debe ser mucho mas específico indicando el protocolo que se usará (http, smtp, jdbc, etc)

Pregunta II (20 puntos). Considere el siguiente código.

```
class AbstractMac
    def print()
        puts "Macbook,
                #{self.inch} inch,
                #{self.gpu} gpu,
                #{self.ram} ram."
    end
    def inch()
        return 14
    end
    def ram()
        return 16
    end
    def gpu()
        return 16
    end
end
```

```
class MacBase < AbstractMac

end

class MacDecorator < AbstractMac
    def initialize(mac)
        @decoratedMac = mac
    end
end
```

```
class GPU16Upgrade < MacDecorator
    def gpu()
     return @decoratedMac.gpu + 16
    end
end
```

```
class GPU32Upgrade < MacDecorator
    def gpu()
      return @decoratedMac.gpu + 32
    end
end
```

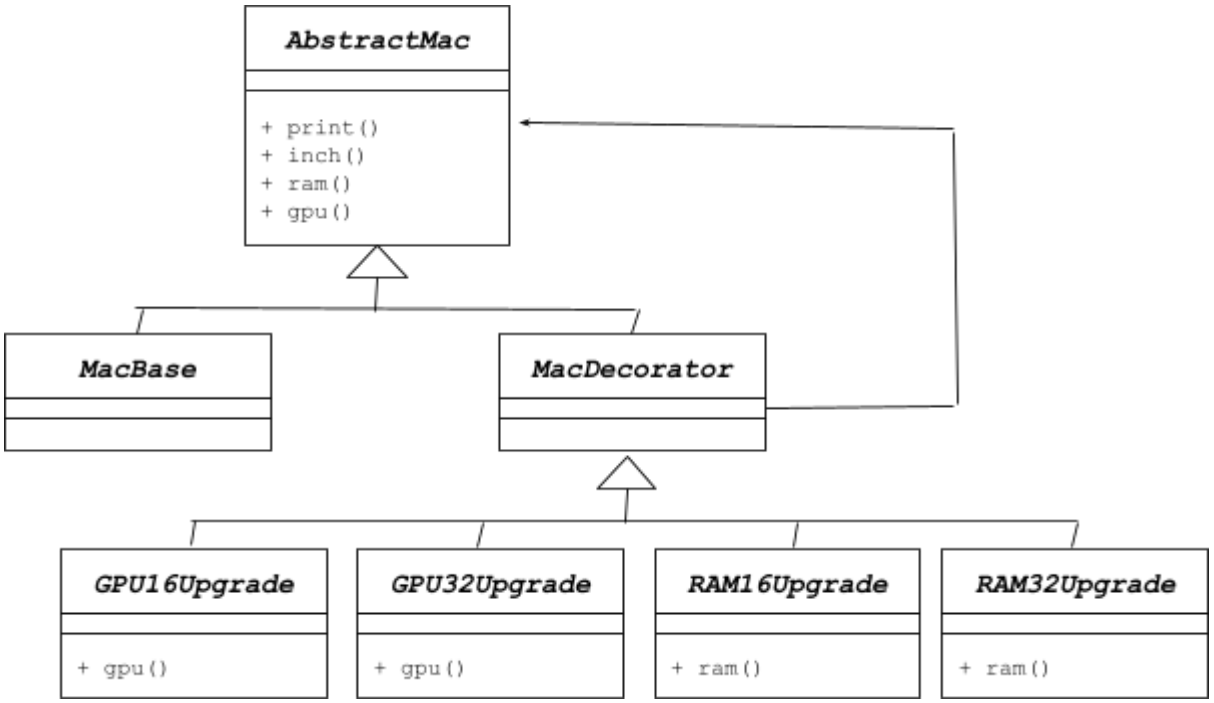
```
class RAM16Upgrade < MacDecorator
    def ram()
     return @decoratedMac.ram + 16
    end
end
```

```
class RAM32Upgrade < MacDecorator
    def ram()
     return @decoratedMac.ram + 32
    end
end
```

A. (5 puntos) Dibuje el diagrama de clases del código anterior.
B. (15 puntos) Dibuje el diagrama de secuencia, asociado a la ejecucion del siguiente código e indique que imprime.

```
mac3 = GPU16Upgrade.new(RAM16Upgrade.new(GPU16Upgrade.new(mac)))
mac3.print
```

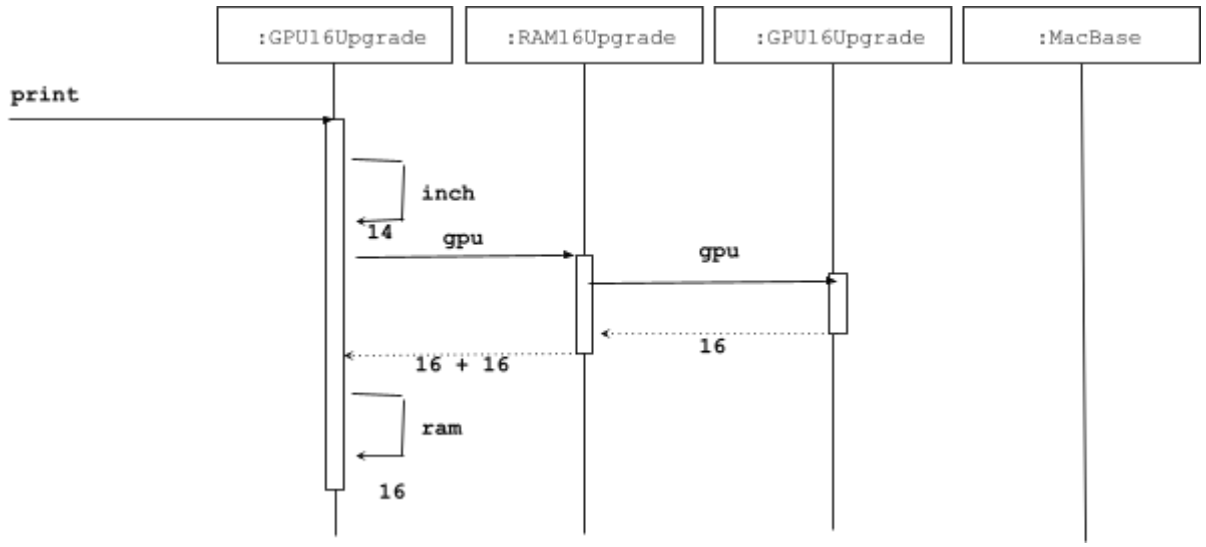
A. Solución



A. Pauta

- **1 punto,** por la clase AbstractMac y sus 4 métodos. No descontar si se olvidaron el +
- **1 punto**, por la relación entre MacDecorator y AbstractMac (la flecha)
- **1 punto**, por la relación de herencia entre MacBase, MacDecorator y AbstractMac
- **1 punto,** por la relación de herencia entre los upgrades y el MacDecorator
- **1 punto** por los métodos correspondientes en cada upgrade, en RAM update es el método ram() y en GPU es el método *GPU*.

B.Solución.



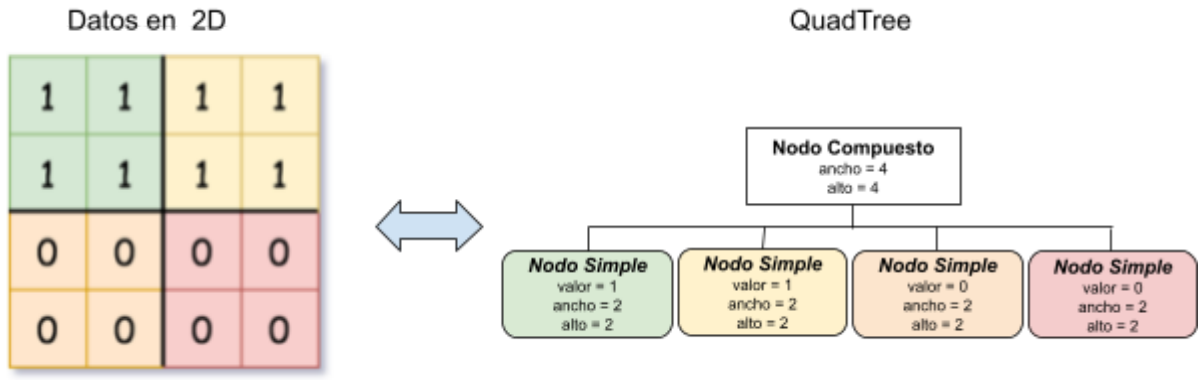
B.Aclaraciones

- El print llama a los métodos inch, gpu y ram utilizando el objeto GPU16Upgrade.
- El método inch no existe en la clase GPU16Upgrade, entonces ejecuta el método inch de la clase padre que devuelve 14.
- El método gpu, primero llama al método GPU de GPU16Upgrade, él mismo llama al método gpu de objeto de la clase RAM16Upgrade, como la clase RAM16Upgrade no tiene el metodo gpu, se ejecuta el método de la clase padre que devuelve 16.
- El método ram no existe en la clase GPU16Upgrade, llama al de la clase padre, el mismo devuelve 16.
- La clase AbstractMac nunca se instancia, entonces no aparece en el diagrama porque el diagrama muestra los objetos instanciados.

B. Pauta

- **3 puntos** por dibujar los objetos que intervienen en la ejecución. Es válido si solo dibujan los primeros 3 (GPU16Upgrade, RAM16Upgrade, y GPU16Update), también es válido si dibujan los 4.
- **4 puntos** por cada mensaje bien dibujado
    - El mensaje print que se ejecuta en el objeto GPU16Upgrade
    - El mensaje gpu, que se ejecuta en el objecto GPU16Upgrade, luego se propaga al RAM16.
    - El mensaje ram, que se ejecuta sobre el objeto GPU16Upgrade.

Pregunta III (30 puntos). Un quadtree es una estructura de datos jerárquica cuya propiedad común es su composición recursiva del espacio. Por ejemplo, considere el siguiente gráfico donde se muestra una matriz de datos en 2D y su representación equivalente en un QuadTree.

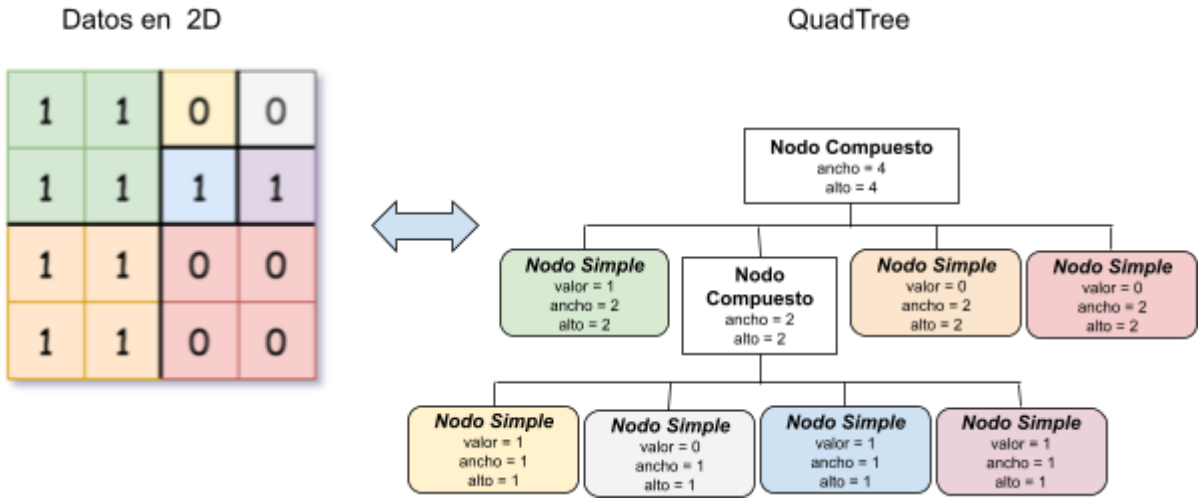


Este ejercicio trata de modelar un QuadTree que represente una matriz bi-dimencional (2D) de números enteros utilizando el patrón composite. El QuadTree a modelar tiene dos tipos de nodos:

- **Nodo Compuesto**, este nodo tiene el ancho y el alto de espacio 2D contenido. Además, este nodo tiene exactamente 4 sub-nodos (ni uno más, ni uno menos). Los sub-nodos pueden ser nodos simples o nodos compuestos.
- **Nodo Simple**, este nodo tiene el valor que se repite dentro del espacio 2D contenido, el ancho y el alto del espacio 2D.

Recuerde que el patrón composite debe proporcionar una interfaz común entre todas las clases parte de la estructura. Es decir, en este ejemplo en particular, se debe poder ejecutar las mismas operaciones tanto en nodos compuestos como en nodos simples. Para este ejercicio se le pide implementar dos operaciones:

- ***contains(value)***, cada nodo dentro de la estructura debe poder verificar si dentro de su espacio bi-dimensional existe un valor pasado como argumento.
- ***count(value)***, cada nodo dentro de la estructura debe poder contar cuántas veces aparece de un valor pasado como argumento dentro del espacio bi-dimensional que el nodo encapsula.



Implemente el patrón composite para representar una matriz de datos 2d en un quadtree. Como ayuda se le proporciona el siguiente código de ejemplo, el mismo crea los objetos que representan la matriz del ejemplo anterior.

```
yellow = SimpleNode.new(1,1,0)
gray = SimpleNode.new(1,1,0)
purple = SimpleNode.new(1,1,1)
blue = SimpleNode.new(1,1,1)
composed = ComplexNode.new(2,2,[yellow,gray,purple,blue])
green = SimpleNode.new(2,2,1)
orange = SimpleNode.new(2,2,1)
red = SimpleNode.   new(2,2,0)
main = ComplexNode.new(4,4, [green,composed,orange,red])

# operation contains
puts '=contains='
puts yellow.contains(1) # imprime false
puts gray.contains(1) # imprime false
puts purple.contains(1) # imprime verdad
puts blue.contains(1) # imprime verdad
puts composed.contains(1) # imprime verdad
puts green.contains(1) # imprime verdad
puts orange.contains(1) # imprime verdad
puts red.contains(1) # imprime false
puts main.contains(1) # imprime verdad

# operation count
puts '=count='
puts yellow.count(1) # imprime 0
puts gray.count(1) # imprime 0
puts purple.count(1) # imprime 1
puts blue.count(1) # imprime 1
puts composed.count(1) # imprime 2
puts green.count(1) # imprime 4
puts orange.count(1) # imprime 4
puts red.count(1) # imprime 0
puts main.count(1) # imprime 10
```

Solución

```ruby
class Node
    def initialize(ancho,alto)
        @ancho = ancho
        @alto = alto
    end
    def contains(v)
        raise NotImplementedError
    end
    def count(v)
        raise NotImplementedError
    end
end
```

```ruby
class SimpleNode < Node
    def initialize(ancho,alto, value)
        super(ancho,alto)
        @value = value
    end
    def contains(v)
        return v == @value
    end
    def count(v)
        if v == @value
            return @ancho*@alto
        else
            return 0
        end
    end
end
```

```ruby
class ComplexNode < Node
    def initialize(ancho,alto,child)
        super(ancho,alto)
        @child=child
    end
    def contains(v)
        res = false
        @child.each do |c|
            res = res || c.contains(v)
        end
        return res
    end
    def count(v)
        res = 0
        @child.each do |c|
            res = res + c.count(v)
        end
        return res
    end
end
```

Pauta.

- **10 puntos.** Crear una clase Node, Crear SimpleNode y ComplexNode como clases hijas de Node. La clase Node debe tener los atributos ancho y alto. La clase ComplexNode, tiene que tener un atributo con una lista de nodes (o subnodes)..
- **10 puntos.** Crear la operacion count, el método debe estar al menos en las 2 clases hijas, idealmente también en la clase padre como abstracto. La implementación debe funcionar, si no funciona al 100% se puede asignar un poco de puntaje.
- **10 puntos.** Crear la operación contains, el método debe estar al menos en las 2 clases hijas, idealmente también en la clase padre como abstracto. La implementación debe funcionar, si no funciona al 100% se puede asignar un poco de puntaje.

Pregunta IV (30 puntos). La empresa *DC-Code* quiere implementar un sistema de blogs simple, donde una persona puede realizar un blog y otros usuarios pueden comentar el mismo. Un estudiante de programación avanzada escribió un pequeño código orientado a objeto (ver al final de la pregunta). El mismo permite crear objetos blogs, agregar comentarios y automáticamente ayuda a moderar los comentarios. El programa automáticamente detecta:

- comentarios que contengan un número de teléfono y los marca como "*under-review*".
- comentarios que contengan la palabra "F" y los denega (*deny*). Ya que los comentarios no deben contener malas palabras.
- comentarios que contengan un enlace a un recurso externo. En particular, si el comentario tiene una subcadena que empieza con "htttp*://*".

El código siguiente muestra un ejemplo de cómo se pueden crear objetos de dicho programa. Al imprimir un blog y sus comentarios, se muestran los comentarios que fueron aceptados, los comentarios *under-review*, y los comentarios denegados.

```
#main
blog = Blog.new('Toyota is the best car brand in the world...')
blog.addComment(Comment.new('yes, it is buddy'))
blog.addComment(Comment.new('need a car? call me 5691234567'))
blog.addComment(Comment.new('F, no way this brand sucks'))
blog.addComment(Comment.new('Hey check mazda cars at http://www.mazda.cl'))
blog.addComment(Comment.new('I have a toyota since 1980, guest what? it is still
working...'))
blog.addComment(Comment.new('holly molly wakamolly'))
blog.print
```

```
#output
Toyota is the best car brand in the world...
> yes, it is buddy
> under review
> deleted
> under review
> I have a toyota since 1980, guest what? it is still working...
> holly molly wakamolly
```

Al arquitecto de *DC-Code* le gustó el código inicial pero hizo una observación. Observó que si en el futuro agregan nuevas formas de moderar los comentarios[1], se tendría que modificar el método *addComment,* lo cual rompe el principio de cerrado para modificaciones abierto para

[1] Por ejemplo, agregar moderadores para detectar comentarios que contengan direcciones personales, o comentarios ofensivos.

extensiones. Además, el método addComment crecerá mucho y tendremos un método muy largo (un code-smell). Para mejorar el diseño, el arquitecto le dio la tarea de aplicar el *patrón observer*. Donde cada tipo moderador sería un observador.

Implemente el ***patrón observer.*** Puede modificar y crear las clases necesarias al código actual. Después de implementar el patrón el código de ejemplo anterior (#main) debe seguir mostrando el mismo texto de salida (#output). Es posible modificar ligeramente el código en #main, pero el #output debe ser el mismo.

código escrito por *DC-Code*

```
class Comment
   attr_accessor :text
   def initialize(text)
       @text = text
       @state = "approved"
   end
   def to_review
       @state = "under review"
   end
   def deny
       @state = "deny"
   end
   def print
       if @state == "under review"
           puts "> under review"
       elsif @state == "approved"
           puts "> #{@text}"
       else
           puts "> deleted"
       end
   end
end
```

```
class Blog
   def initialize(message)
     @message = message
     @comments = []
   end
   def lastComment()
       return @comments.last

   end


   def addComment(comment)
       # moderador 1: detecta si alguien comparte numeros de telefono
       if comment.text.match(/[5-5][6-6]\d+/)
           comment.to_review
       end
       # moderador 2: detecta si alguien escribe laa palabra con F
       if comment.text.include?('F')
           comment.deny
       end
       # moderador 3: detecta si alguien escribe un link a otro sitio
```

```
        if comment.text.include?('http://')
            comment.to_review
        end
        @comments.push(comment)
    end
    def print
        puts @message
        @comments.each do |c|
            c.print
        end
    end
end
```

Solución

```
class Observer
    def update(blog)
        raise NotImplementedError
    end
end
```

```
class PhoneModerator < Observer
    def update(blog)
        comment = blog.lastComment()
        if
comment.text.match(/[5-5][6-6]\d+/)
            comment.to_review
        end
    end
end
```

```
class LinkModerator < Observer
    def update(blog)
        comment = blog.lastComment()
        if
comment.text.include?('http://')
            comment.to_review
        end
    end
end
```

```
class FModerator < Observer
    def update(blog)
        comment = blog.lastComment()
        if comment.text.include?('F')
            comment.deny
        end
    end
end
```

```
class Blog
    …
    def register(obs)
        @observers.push(obs)
    end
    def notifyAll()
        @observers.each do |obs|
          obs.update(self)
        end
    end
    def addComment(comment)
        @comments.push(comment)
        notifyAll()
    end
    …
end
```

```
blog = Blog.new('Toyota is the best car brand in the world...')
blog.register(PhoneModerator.new)
blog.register(FModerator.new)
blog.register(LinkModerator.new)
```

```
blog.addComment(Comment.new('yes, it is buddy'))
blog.addComment(Comment.new('need a car? call me 5691234567'))
blog.addComment(Comment.new('F, no way this brand sucks'))
blog.addComment(Comment.new('Hey check mazda cars at http://www.mazda.cl'))
blog.addComment(Comment.new('I have a toyota since 1980, guest what? it is still
working...'))
blog.addComment(Comment.new('holly molly wakamolly'))
blog.print
```

Pauta.

- **15 puntos**. Crear la clase Observer y crear 3 clases que hereden de la clase observer. Cada una debe sobreescribir el método update, y tener el código asociado a cada moderador para que funcione.
- **10 puntos.** Agregar la lista de observadores a la clase blog (o comment) y notificar adecuadamente a los observadores cuando un nuevo comentario se agrega.
- **5 puntos.** Crear y agregar los observadores.

- Se puede hacer valer parte del puntaje a criterio.