Trabajo Práctico – Composite

Objetivo: Comprender el patrón de diseño Composite. Discutir alternativas de diseño e implementación.

Ejercicio 1 - Desde el libro

Basándose en el capítulo del patrón de diseño del libro Design Patterns de Gamma et al. desarrolle los siguientes puntos.

1. En la intención del patrón se indica que el cliente debe poder tratar de manera
2. uniforme tanto a objetos individuales como compuestos. ¿ Cómo se logra esto con el patrón Composite?
3. Explique en qué casos aplicaría el patrón.
4. Dentro de las consecuencias, se indica que los clientes deben mantenerse simples y no deben
5. ser conscientes si están interactuando con un objeto hoja o una composición. ¿Qué elementos de la programación orientada a objetos hacen posible esto?

Ejercicio 2 - Cultivos

Diseñe una solución para el siguiente problema.
Se debe modelar un sistema que controla el uso que se le da a porciones productivas de tierra. Las porciones pueden ser puras de un cultivo o mixtas, estas ultimas se dividen en cuatro regiones iguales. En cada región se pueden plantar solamente soja, trigo o subdividir la porción en cuatro siguiendo las mismas reglas anteriores. La figura muestra un ejemplo de esto. A partir del cultivo, el sistema debe determinar las ganancias anuales de cada tipo de cultivo para una región. La región cultivada con soja da una ganancia anual de $500, la de trigo $300. En caso que la región este subdividida, la ganancia será proporcional a la cantidad de sub parcelas que posean ese cultivo. El sistema debe responder a la ganancia anual que dará la parcela tanto para soja como para trigo.

1. ¿Qué impacto genera en su diseño el hecho de compartir instancias de parcelas entre composiciones? Es decir, que la misma instancia se encuentre en más de una composición.
2. Implemente en Smalltalk su solución del problema.

3. Describa las alternativas que puede utilizar para calcular la ganancia anual. Relacione las alternativas.
4. Explique la discusión sobre quiénes deben implementar las operaciones de agregado y borrado de hijos.
5. Sobre su implementación, indique los roles que desempeñan cada una de las clases diseñadas en relación al patrón Composite.

Ejercicio 3 - Juego de estrategia

Se debe diseñar un juego de estrategia del estilo Warcraft. En este juego, el jugador dispone de diferentes caracteres que puede caminar desde un punto hasta otro en el mapa. Existen los siguientes tipos de personajes:

- Ingeniero:
  - Al caminar de un punto a otro lo hace de la forma más corta posible. Mientras lo hace va construyendo un camino. Para construirlo va dejando en el piso lajas que contiene en una bolsa. Las lajas son finitas, por lo cual las coloca solamente si en el camino no han colocado alguna laja antes. Cuando se terminan sus lajas, camina normalmente.
- Caballero:
  - Al caminar debe ir vigilando la zona. Por lo tanto, para ir de un punto a otro debe hacerlo en zigzag para controlar que no haya enemigos.
- Ejercito:
  - Están conformados por cualquiera de los dos personajes anteriores y también otros ejércitos. La acción de caminar de un ejercito esta basada en mover al punto destino, todos los personajes que se incluyen en el ejercito de la forma que cada un lo realiza.

1. Modele una solución orientada a objetos que permita la creación de los personajes y ejércitos.
2. Implemente en java la solución de manera que cualquiera de los personajes pueda moverse de una posición a otra en un mapa.
3. Indique los participantes y roles de aquellos patrones de diseño que utilizó en su solución.

Ejercicio 4 - Sistema de Archivos

Un *File System* esta conformado por Archivos y Directorios en una organización jerárquica y de inclusión. Un archive debe ser capaz de responder al nombre, la cantidad de espacio en disco que ocupa y la fecha de la ultima modificación. De un Directorio debe poder decirse el nombre, la fecha en que fue creado y su contenido: archivos y directorios.

El *File System* debe proveer la funcionalidad descripta en la siguiente interfaz:

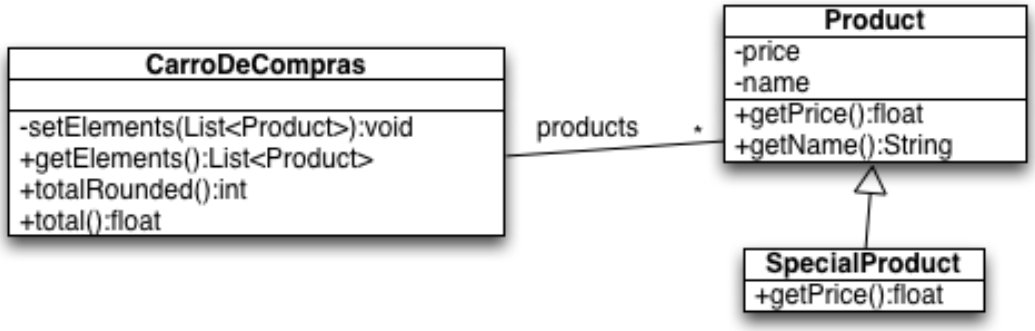
```
public interface FyleSystem{
	/**
	* Retorna el total ocupado en disco del receptor. Expresado en
	*cantidad de bytes.
	*/
	public int totalSize();

	/**
	* Imprime en consola el contenido indicando el nombre del elemento
```

```
        * e indentandolo con tantos espacios como profundidad en la
        estructura.
        */
        public void printStructure();

        /**
        * Elemento mas nuevo
        */
        public XXXXXXXXX lastModified();

        /** Elemento mas antiguo
        */
        public XXXXXXXX oldestElement();
}
```

Nota: Tenga en cuenta que los tipos de retorno indicados con XXXXXXX en la interfaz, deben ser completados por el que crea conveniente.

Ejercicio 5 - UML

Escriba en lenguaje Java todo aquello que puede establecer a partir del siguiente diagrama de clases UML.



Ejercicio 6 - ShapeShifter

En el grupo de investigación de la UNQ se requiere construir un nuevo tipo objetos que posee una estructura del tipo arbórea al que denominaron ShapeShifter. Estos objetos deben poder componerse, detectar la mayor profundidad y "achatarse". En esta primera etapa es necesario que simplemente trabajen con Integer. Como la idea es un tanto difusa, el equipo de investigación les provee a los desarrolladores una interfaz java y una serie de descripciones con ejemplos de cada una de las funcionalidades necesarias.

```
public interface IShapeShifte{
public IShapeShifter compose(IShapeShifter);
public int deepest();
public IShapeShifter flat();
public List<Integer> values()
}
```

1. **public IShapeShifter compose(IShapeShifter);**
   Este método recibe un IShapeShifter y retorna un nuevo IShapeShifter que es el resultado de componer al receptor con el que es enviado como parámetro. La composición se realiza al mismo nivel de profundidad. En la figura se muestran

esquemas de IShapeShifter, cada letra sibolizaria un nombre de variable. Entonces las siguientes expresiones deberían ser verdaderas:

- a.compose(b) es igual a c
- d.compose(c) es igual a d
- d.compose(e) es igual a f.

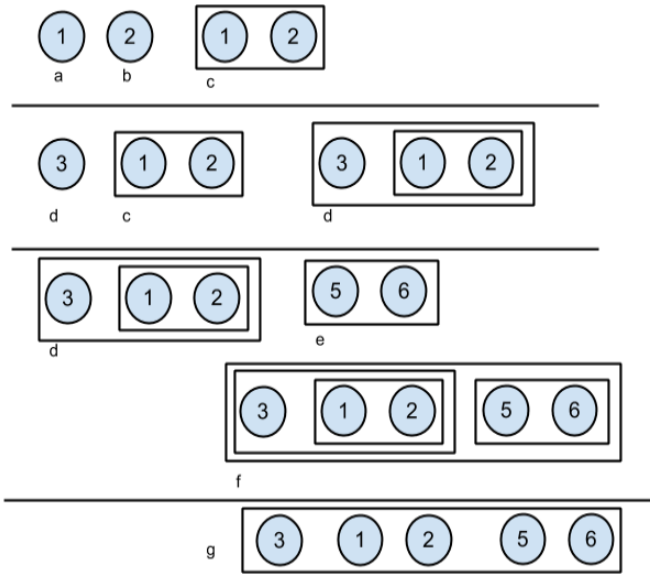
2. **public int deepest(IShapeShifter)**
   Retorna un numero que representa la profundidad máxima alcanzada en composiciones que contiene. Continuando con las figuras de ejemplo, las siguienes expresiones deben ser validas.
   - a.deepest() es igual a 0.
   - c.deepest() es igual a 1.
   - f.deepest() es igual a 3.

3. **public IShapeShifter flat()**
   Achata un IShapeShifter. Si el IShapeShifter posee una prfundidad maxima >= 1, entonces retorna un IShapeShifter de profundidad maxima 1 con todos los IShapeShifter de profundidad 0 contenidos. En cualquier otro caso, retorna el mismo IShapeShifter. Siguiendo los ejemplos
   - a.flat() es igual a a.
   - f.flat() es igual a g.

   Importante: No es necesario establecer un orden particular de los IShapeShifter en los resultantes.

4. **public List<Integer> values()**
   Retorna una lista de enteros con los valores incluidos en el IShapeShifter. Siguiendo el ejemplo,
   - a.values retorna una lista con el entero 1.
   - d.values retorna una lista con los enteros 3,1 y 2.

   Importante: No es necesario establecer un orden particular de los enteros en la lista.



Ejercicios:

1. Realizar un diagrama de clases UML donde muestre un diseño orientado a objetos que resuelva el problema aplicando el patron Composite.
2. Indicar en el diseño cuales son los roles del patrón.
3. Implementar en java todo lo necesario para resolver la funcionalidad de la interface.
4. Escribir un metodo en el cual muestre como se instancia el ejemplo a y c.