Trabajo Práctico 6 – Composite y Template Method

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

Ejercicio 1 – Sistema de Archivos

Un *File System* está 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 última 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 FileSystem{
        /**
         * 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
        * identandolo con tantos espacios como profundidad en la estructura.
        */
        public void printStructure();

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

        /**
        * Elemento más antiguo
        */
        public XXXXXXXX oldestElement();

        /**
        * Retorna la fecha de la última modificación.
        */
        public DateTime lastModification();
}
```

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

Ejercicio 2 – 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. a.compose(b) es igual a c
   b. d.compose(c) es igual a d
   c. 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. a.deepest() es igual a 0.
   b. c.deepest() es igual a 1.
   c. 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 siguientes ejemplos:

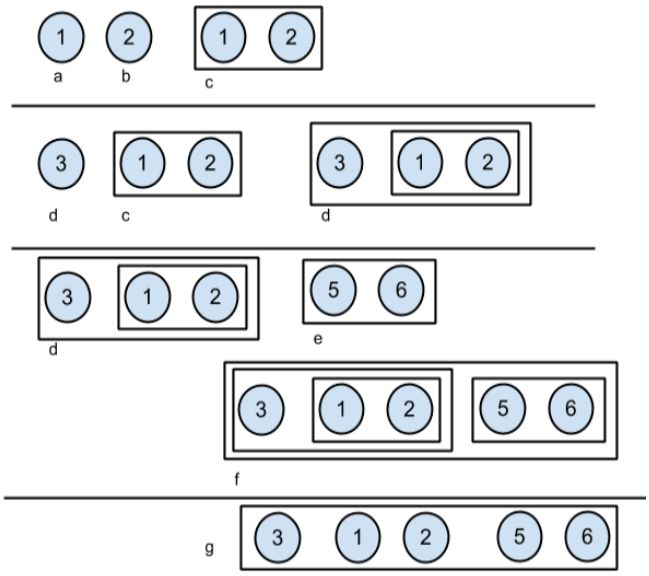
   a. a.flat() es igual a a.
   b. f.flat() es igual a g.

   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. a.values retorna una lista con el entero 1.
b. 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 patrón 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 con clases que la implementen
4. Escribir un método en el cual muestre como se instancia el ejemplo a y c.

Ejercicio 3 - Lea el pattern Template Method y...

1. Utilizando las ideas propuestas por el pattern resuelva los ejercicios siguientes.
2. Responda:
    a. ¿Dónde se define el esqueleto del algoritmo?
    b. ¿Se puede redefinir el esqueleto?
    c. ¿Qué es lo que se puede redefinir?
    d. ¿Qué es un hook method?
3. Busque dos ejemplos del patrón en las clases de la imagen de Smalltalk.
4. Para cada ejemplo, realice un diagrama de clases indicando:
    a. La clase abstracta.
    b. El template method.
    c. Al menos dos clases concretas.
    d. Los métodos en la clase abstracta que se redefinen en la clase concreta.
    e. Indique hooks methods.

Ejercicio 4 - Ayudando al soberano

Un banco tiene dos tipos de cuentas bancarias. Cajas de ahorro y Cuentas corrientes. Ambas cuentas se manejan en forma similar excepto para la extracciones de dinero. En las caja de ahorro es posible realizar la extracción solamente si el saldo de la cuenta es superior o igual a la cantidad de dinero que quiero extraer y no supera el límite por extracción, si esto sucede, entonces realiza efectivamente la extracción (actualizando el saldo) y luego registra que se hizo una extracción simplemente agregando un texto a una colección. Por otra parte, las cuentas corrientes pueden realizar una extracción solamente cuando la cantidad de dinero a extraer no supera el saldo más el límite en rojo que posee la cuenta. Si esto ocurre entonces realiza la extracción y luego registra que se hizo una extracción simplemente agregando un texto a una colección.
Un compañero del curso, realizó el ejercicio de la siguiente manera:

```
import java.util.ArrayList;
import java.util.List;

public abstract class CuentaBancaria {
    private String titular;
    private int saldo;
    private List<String> movimientos;

    public CuentaBancaria(String titular){
        this.titular=titular;
        this.saldo=0;
        this.movimientos=new ArrayList<String>();
    }
        public String getTitular(){
        return this.titular;
    }
    public int getSaldo(){
        return this.saldo;
    }
    protected void setSaldo(int monto){
        this.saldo=monto;
    }
    public void agregarMovimientos(String movimiento){
        this.movimientos.add(movimiento);
    }
    public abstract void extraer(int monto);
}

public class CuentaCorriente extends CuentaBancaria {
    private int descubierto;

    public CuentaCorriente(String titular, int descubierto){
        super(titular);
        this.descubierto=descubierto;
    }
    public int getDescubierto(){
        return this.descubierto;
    }
    @Override
    public void extraer(int monto) {
        if(this.getSaldo()+this.getDescubierto()>=monto){
```

```
            this.setSaldo(this.getSaldo()-monto);
            this.agregarMovimientos("Extraccion");
        }
    }
}

public class CajaDeAhorro extends CuentaBancaria {
    private int limite;

    public CajaDeAhorro(String titular, int limite){
        super(titular);
        this.limite=limite;
    }
    public int getLimite(){
        return this.limite;
    }
    @Override
    public void extraer(int monto) {
        if(this.getSaldo()>=monto && this.getLimite()>=monto){
            this.setSaldo(this.getSaldo()-monto);
            this.agregarMovimientos("Extraccion");
        }
    }
}
```

El cual compila y retorna los resultados esperados. Su compañero se da cuenta que hay código repetido y además no utilizar template method, el cual es indispensable para que apruebe la entrega.

Su objetivo es el siguiente:

1. Debe realizar los cambios necesarios en el código para que el método extraer sea un template method. (Escriba solamente los cambios, el resto se asume que queda intacto)
2. Analice en el código resultante los elementos que indica Gamma et. al que pueden aparecer en su solución, por ejemplo: Clase abstracta, clase concreta, cual es el template method, cuales son las operaciones primitivas, cuales son las operaciones concretas, cuales los hook methods.

Importante: No necesariamente aparecen todos los elementos que lista Gamma, ud. debe explicar solamente aquellos que están en su solución.