Documentación Examen II Cantones de Costa Rica

Manuel Arias Medina – 2017119039

Justin Bogantes Rodríguez – 2017110613

Introducción

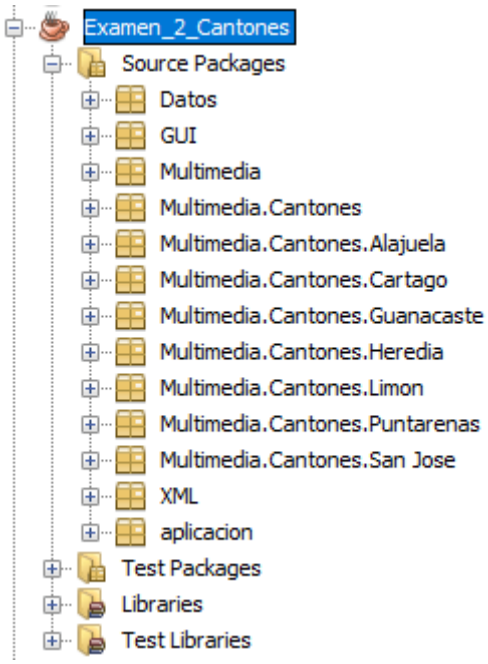
En el desarrollo de este examen programado utilizamos los patrones vistos en clases con el fin de agilizar el proceso de codificación y para ejercer una buena practica ante el problema al cual nos vimos enfrentados. Los patrones que utilizamos nos permitieron un mejor manejo de la información, así como una manera de distinguir las situaciones en las cuales es mejor utilizar cada patrón. Entre los patrones que utilizamos se encuentran los siguientes: Patrón de Observador, patrón de multicapas, patrón controlador y patrón fabrica.

Patrón Multicapas

Un sistema orientado a objetos multicapa está organizados en varias capas, cada una de las cuales contiene un conjunto de clases con responsabilidades relacionadas con la capa a la que pertenecen. El ejemplo más habitual de sistemas multicapa se da en las aplicaciones que permiten a las personas utilizar datos almacenados en una base de da tos: los usuarios manejan la aplicación a través de una serie de pantallas, que son realmente clases situadas en la denominada “capa de presentación”. Cuando un usuario desea hacer una operación sobre la base de datos, realiza las operaciones sobre el objeto correspondiente de la capa de presentación; este objeto, en lugar de realizar él mismo la

operación, le pasa un mensaje a un objeto de una capa más interna, denominada habitualmente capa de “dominio”, “negocio”, “procesamiento”, etc. La capa de dominio está ocupada por las clases que están presentes en el enunciado del problema (por ejemplo: “factura”, “empleado” o “vehículo”). Estas clases son las que se encargan realmente de realizar las operaciones necesarias para resolver el problema (por ejemplo: “calcula total de la factura”, “contar número de empleados”, “subir el sueldo a los empleados de tal departamento”, etc.). Algunas de las clases de la capa de dominio serán persistentes, lo que significa que sus instancias estarán almacenadas en la base de datos. De este modo, estas clases deberán, de alguna manera, incorporar las operaciones relacionadas con la persistencia de sus instancias (por ejemplo: insertarse, borrarse, actualizarse, etc.). La base de datos ocupará la tercera y más profunda capa del sistema, denominada capa “de persistencia”, de “almacenamiento”, etc. Esta tercera capa incorporará quizás una serie de clases que actuarán como mediadores entre las clases de la capa de dominio y la base de datos.

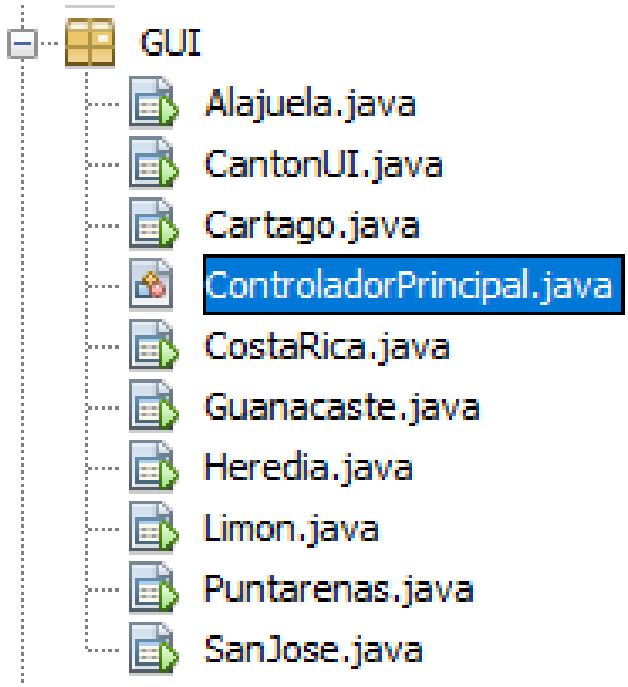
Nosotros hicimos uso de este patrón a como se evidencia aquí.



Patrón Controlador

Modelo-vista-controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado, define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

Nosotros hicimos uso de este patrón a como se evidencia aquí.



Patrón Observador

El patrón observador es un patrón de diseño de software que define una dependencia del tipo uno a muchos entre objetos, de manera que cuando uno de los

objetos cambia su estado, notifica este cambio a todos los dependientes. Se trata de un patrón de comportamiento (existen de tres tipos: creación, estructurales y de comportamiento), por lo que está relacionado con algoritmos de funcionamiento y asignación de responsabilidades a clases y objetos.

Los patrones de comportamiento describen no solamente estructuras de relación entre objetos o clases sino también esquemas de comunicación entre ellos y se pueden clasificar en función de que trabajen con clases (método plantilla) u objetos (cadena de responsabilidad, comando, iterador, recuerdo, observador, estado, estrategia, visitante).

La variación de la encapsulación es la base de muchos patrones de comportamiento, por lo que cuando un aspecto de un programa cambia frecuentemente, estos patrones definen un objeto que encapsula dicho aspecto. Los patrones definen una clase abstracta que describe la encapsulación del objeto.

Este patrón también se conoce como el patrón de publicación-inscripción o modelo-patrón. Estos nombres sugieren las ideas básicas del patrón, que son: el objeto de datos, que se le puede llamar Sujeto a partir de ahora, contiene atributos mediante los cuales cualquier objeto observador o vista se puede suscribir a él pasándole una referencia a sí mismo. El Sujeto mantiene así una lista de las referencias a sus observadores. Los observadores a su vez están obligados a implementar unos métodos determinados mediante los cuales el Sujeto es capaz de notificar a sus observadores suscritos los cambios que sufre para que todos ellos tengan la oportunidad de refrescar el contenido representado. De manera que cuando se produce un cambio en el Sujeto, ejecutado, por ejemplo, por alguno de los observadores, el objeto de datos puede recorrer la lista de observadores avisando a cada uno. Este patrón suele utilizarse en los entornos de trabajo de interfaces gráficas orientados a objetos, en los que la forma de capturar los eventos es suscribir listeners a los objetos que pueden disparar eventos.

Nosotros hicimos uso de este patrón a como se evidencia aquí.

```
private void cantonSarapiquiActionPerformed(java.awt.event.ActionEvent evt) {
    CantonUI ui = new CantonUI();
    ui.getlblNombre().setText("Sarapiqui");
    ui.getLblCodigo().setText("410");
```

Patrón Fabrica

En diseño de software, el patrón de diseño Factory Method consiste en utilizar una clase constructora (al estilo del Abstract Factory) abstracta con unos cuantos métodos definidos y otro(s) abstracto(s): el dedicado a la construcción de objetos de un subtipo de un tipo determinado. Es una simplificación del Abstract Factory, en la que la clase abstracta tiene métodos concretos que usan algunos de los abstractos; según usemos una u otra hija de esta clase abstracta, tendremos uno u otro comportamiento.

Nosotros usamos este patrón a como se evidencia aquí.

```
private List<Canton> cantones;

public ListaCantones(){

}

public Canton MostrarCanton(int codigo) {

    Canton salida = new Canton();
    for (int i = 0; i < cantones.size(); i++) {
        salida = cantones.get(i);
        if (salida.getCodigo()== codigo) {
            return salida;
        }
    }
    return null;
}
```