Universidad Nacional de Quilmes

Trabajo Práctico Integrador

Programación con Objetos II - 2023 - 1er Semestre

A la caza de las vinchucas

INTEGRANTES:

Fuentes Tomas (tomasrfuentes@gmail.com)

Gomez Walter (walterggomez@gmail.com)

Peppe Leandro (leandro.peppe@alu.unq.edu.ar)

Decisiones de diseño

En el siguiente apartado se detallarán a continuación las decisiones de diseño tomadas para satisfacer los puntos requeridos en el trabajo practico. Los mismos estarán divididos por las distintas clases que componen el diseño en base a la programación orientada a objetos.

Muestra

Para esta clase se determinó utilizar el patrón de comportamiento State ya que la misma pasa por diferentes estados en base a las opiniones que recibe. Los roles de este patrón se especifican en (*1).
Se instancia con la fecha de creación del día actual. Creamos el método setFechaCreación() para poder realizar test en donde las mismas tengan diferente día.

Opinión

Al momento de realizar una opinión los atributos que reciben valores son; fecha actual, *TipoOpinion. El* atributo *DatosDelCreador* se setea antes de realizar la opinión.
Como el estado del participante puede cambiar (participante dinámico), para no alterar las opiniones realizadas con anterioridad se creó la clase *DatosDelCreador* para capturar el estado del participante al momento de realizar una opinión. Es decir, si un participante al momento previo a opinar su estado es *Dinamico* y una vez realizada la opinión su estado cambia a *Experto*, la opinión no se verá alterada por este cambio.
Determinamos que la opinión no conozca a la muestra en la cual fue agregada ya que esta misma no es usada en su vida útil.

Como las opiniones tienen una clasificación, que indica cómo fue clasificada la muestra según el participante, se creó un Enumerativo *TipoOpinion* donde se declaran los tipos.

Ubicación

Ubicación es una clase que contiene dos atributos; longitud o latitud, con la combinación de estos dos se determina una coordenada en el espacio la cual no sirve para identificar un punto determinado en el mapa.
Según lo pedido en el enunciado del trabajo práctico deberíamos haber realizado un método para esta clase, el cual dada una muestra determina las muestras que se encuentran a x metros o kilómetros de la misma. Este método no se implementó en esta clase ya que la ubicación de la instancia particular no interfiere en el algoritmo para determinar todas las muestras que están a x distancia de una muestra dada. Por esta razón decidimos implementarlo en la clase Sistema debido a que esta misma conoce a todos los participantes, por ende, conoce a todas las muestras y a su vez a todas las ubicaciones de las mismas.

Participante

Determinamos que tanto los Participantes básicos, como los Participantes Expertos Externos comparten estructura y comportamiento (cargan y opinan sobre una muestra de la misma

manera) decidimos por lo tanto tener una clase padre *Participante* y aplicar herencia con las clases *Dinamico* y *ExpertoExterno.*
Para la implementación en la evolución del conocimiento de los participantes, se determina la utilización del patrón de comportamiento State (*2).
Se decidió usar la clase abstracta EstadoUsuario en vez de una interfaz, debido a que los algoritmos para hacer el filtrado tanto de opiniones como de muestras son exactamente iguales entre los distintos estados del usuario. Al tener una clase abstracta tenemos la posibilidad de hacer herencia y de esta forma evitar la duplicación de código en común.

Zona De Cobertura

Se implementa mediante una clase para poder cumplir con los requerimientos solicitados, por lo tanto, se tomó la decisión de que esta clase no tenga una lista de muestra como atributo, sino que se obtenga esta lista a través de un método mediante su colaborador interno sistema. De esta manera las muestras que perecen a estas zonas se mantienen actualizadas.
Mediante el empleo de una fórmula matemática se determinó el algoritmo para determinar el solapamiento de las distintas zonas de cobertura.
Esta clase comparte junto a la clase *Organización* la implementación del patrón Observer (*3).

Organización

Esta clase resuelve lo pedido con respecto a la funcionalidad externa, utilizando dos atributos debido a que en el ítem se menciona la capacidad de poder usar dos funcionalidades externas diferentes ya sea para el ingreso o verificación de una muestra.
Esta clase comparte junto a la clase *ZonaDeCobertura* la implementación del patrón Observer (*3).

Sistema

En esta clase para resolver lo solicitado con la búsqueda de muestra se empleó un atributo *Filtro* el cual puede ser seteado.
Posee una lista de *Participante* ya que a través de ellos puede llegar a todas las muestras que hay en la aplicación web(*Sistema*), debido a que las muestras posee un creador, Participante.

Filtro

Según lo mencionado en el enunciado, determinamos que hay una composición de filtros, lo cual claramente nos indicaba la utilización del patrón de diseño Composite (*4).
Para incluir al operador AND y OR en el filtrado se decidido crear la clase abstracta *FiltroCompuesto* y de están heredan dos clases que lo resuelven.
Por el otro lado debíamos hacer un filtro para las muestras verificadas o votadas, decidimos separar el filtro por muestras verificadas en una clase y votadas en otra.

Patrones de Diseño

Los patrones de diseño utilizados en el trabajo son : Composite , Observer y State.

State

Este patrón fue utilizado para representar el cambio de estado de la clase Participante y para los cambios de estado de la Muestra. Los roles según la definición de Gamma son los siguientes:

(*2)Participante

Contexto: Es la clase Participante, la cual cambia de estado en base a ciertas condiciones dadas en la cantidad de muestras y opiniones hechas por un participante en los últimos 30 días.
Estado: Esta representado por la clase abstracta *EstadoUsuario* que nos define el protocolo para encapsular el comportamiento asociado a los estados.
EstadoConcreto: Uno de los estados concretos es el implementado por la clase *EstadoBasico* donde está el comportamiento para el estado básico de un participante.
EstadoConcreto: El otro estado concreto es el implementado por la clase *EstadoExperto* donde está el comportamiento para el estado experto de un participante.

(*1) Muestra

Contexto: Esta representado por la clase *Muestra* la cual va a pasar por diferentes estados, estos se determinan en base a las opiniones realizadas.
Estado: Esta representado por la clase abstracta *EstadoMuestra* que nos define el protocolo para encapsular el comportamiento asociado a los estados.
EstadoConcreto: Uno de los estados concretos es el implementado por la clase *Inicial* donde se representa el comportamiento de la muestra al ser instanciada hasta que sea comentada por un participante experto.
EstadoConcreto: El otro estado concreto es el implementado por la clase *SemiVerificada* donde está el comportamiento de la muestra cuando esta ya fue opinada por un participante experto.
EstadoConcreto: El ultimo estado concreto es el implementado por la clase *Verificada* donde el comportamiento de la muestra es el de no aceptar más opiniones de participantes, lo cual se cumple cuando la opinión de dos participantes expertos coinciden.

(*3) Observer

Este patrón fue utilizado para representar el aviso a las organizaciones que se suscriben a las zonas de cobertura cuando necesiten saber si hubo un nuevo ingreso de muestra o una validación de una muestra. Los roles según la definición de Gamma son los siguientes:
Observador Concreto: En el diseño está dado por las organizaciones ya que son las que antes un evento en la muestra son notificadas por el sujeto(*zonaDeCobertura*).
Sujeto Concreto: En el diseño planteado la Zona de cobertura es el Sujeto que conoce a los observadores(Organizaciones) y proporciona el comportamiento para asignar y quitar objetos Observador.

(*4) **Composite**

Este patrón fue realizado para La clase *Filtro* el cual se encarga de la búsqueda de Muestras. Los roles según la definición de Gamma son los siguientes:
Componente: Este rol lo cumple filtro ya que implementa el comportamiento en común entre todos los filtros.
Hoja: Esta representado por las clases *PorFecha*, *PorFechaUltimaOpinion*, *PorTipoVinchuca*, *NivelDeVerificacion(abstracta), NivelDeVerificacionVotada, NivelDeVerificacionVerificada*.
Compuesto: Representado por la clase *FiltroCompuesto* la cual es abstracta y de ella heredan las clases *FiltroCompuestoAnd* y *FiltroCompuestoOr.*
Cliente: En el diseño el *Sistema* cumple este rol ya que utiliza los filtros.