15/06/2023

Profesores:

- Diego Torres
- Diego Cano
- Matias Butti

Integrantes:

- Jonatan Hermosilla
- Julián Villarroel
- Hector Villavicencio

E-mails

jonatanhermosilla1989@gmail.com
villarroel.julian.esteban@gmail.com
fvillavicencio2@gmail.com

Toma de decisiones de diseño:

Calificación y tipoDeOrganización:

A la hora de implementar la Calificación y los tipos de Organización de la muestra, utilizamos Enumerativos, siendo así, la opinión utiliza ese enumerativo Calificación.

Filtro compuesto:

Antes de implementar los filtros nos encontramos con un diseño de patrón composite al ver que se podían anidar las búsquedas. En torno a eso, decidimos utilizar una interfaz de búsqueda con ramas y el nodo composite, el cual tendría dos ramas más que serían: "**and**" y "**or**" qué son las pedidas. Al avanzar con la implementación encontramos el patrón template method, por el cual modificamos la interfaz por una clase abstracta.

Estado de la muestra:

En primer lugar decidimos usar Enumerativos porque pensamos que solamente cambiaba el nombre del estado, pero a la hora de implementar nos dimos cuenta que había comportamientos distintos al momento de actualizar los estados. Por lo que finalmente nos quedamos con el patrón Estate para abordarlo.

Especialista experto:

Inicialmente quisimos que implemente especialista a estadoDeUsuario, pero, nos dimos cuenta que especialista y experto tenían casi el mismo comportamiento, salvo en su forma de actualizarse. Finalizando con el diseño de qué especialista herede de experto.

Muestra:

- Muestra tendrá un atributo que representa a la "fotoVinchuca". Inicialmente la abordamos como un String, pero luego nos percatamos que podría ser tipada como un BufferedImage para poder representarla.
- Contiene un historial de opiniones, en la cual cada opinión estará asociada a un usuario en cuestión.
- Puede conocer en todo momento su resultado actual, obteniendo la opinión con más votos adquiridos de acuerdo al tipo de Calificación que obtuvo.
- Puede obtener las muestras cercanas entre dos muestras, dependiendo del kilometraje. De forma implementativa, esto se lo delegó a la ubicación que conoce la muestra.

AplicaciónWeb:

- Esta clase decidimos implementarla ya que es necesaria para que pasen la mayoría de las funcionalidades de las muestras. De esta forma se libera a la muestra de la responsabilidad de administrar y accionar funcionalidades extras delegando eso a la AplicacionWeb.

Ubicación:

- Utilizamos la fórmula de Haversine extraída de la web, para resolver la distancia entre dos ubicaciones geográficas conociendo su latitud y longitud.
- Puede obtener su distancia respecto de otra ubicación; así también como las muestras cercanas de tal muestra.

Patrones de diseño:

A continuación dejamos imágenes, las cuales pueden observarse en el UML para mejor calidad de visualización.

Design Patterns: State

Roles de los estados de muestras:

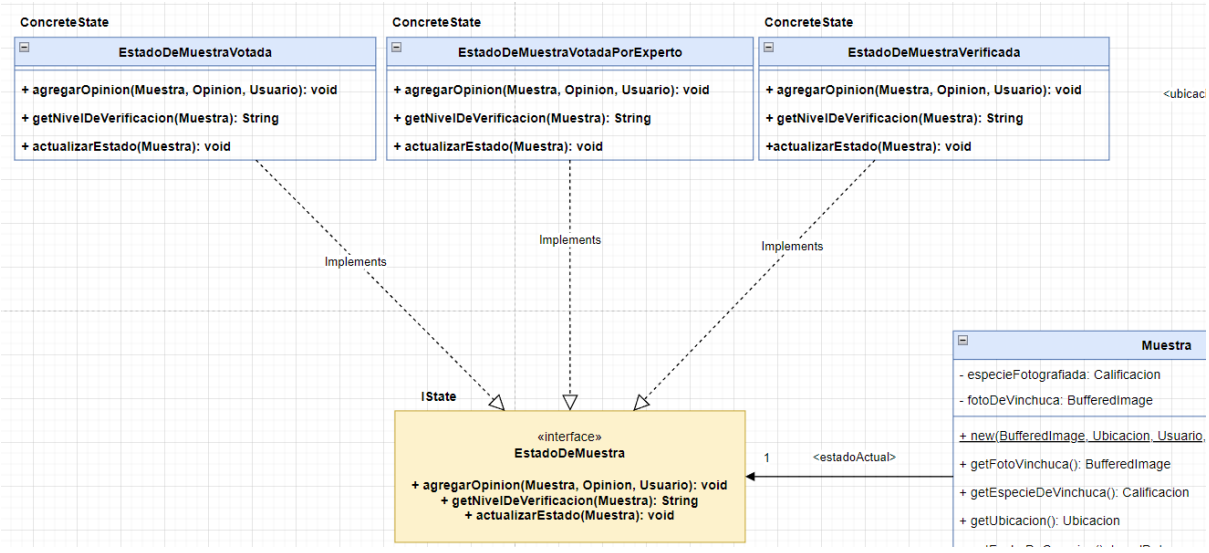
Muestra: Context

EstadoDeMuestra: State

EstadoDeMuestraVotada: ConcreteState

EstadoDeMuestraVotadaPorExperto: ConcreteState

EstadoDeMuestraVerificada: ConcreteState



Design Patterns: State

Roles de los estados de usuario:

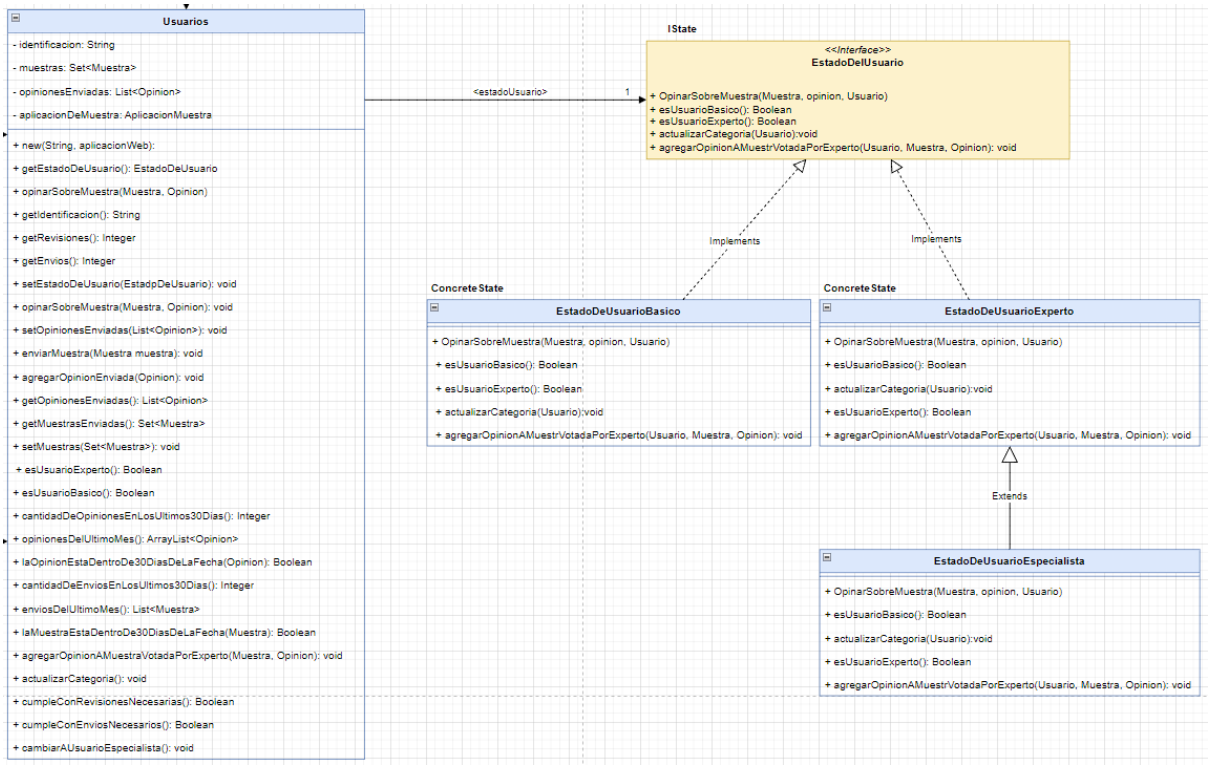
Usuario: Context

EstadoDelUsuario: State

EstadoDeUsuarioBasico: ConcreteState

EstadoDeUsuarioExperto: ConcreteState

EstadoDeUsuarioEspecialista: ConcreteState



Design Patterns: Composite

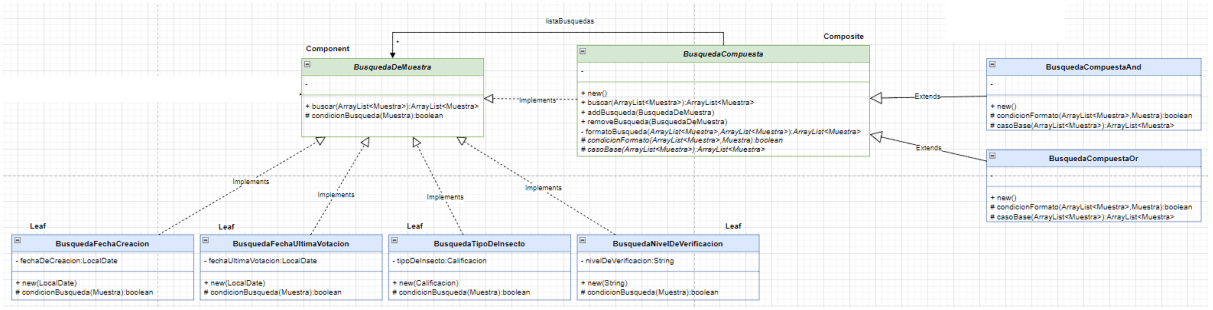
Roles de composite:

BusquedaDeMuestra: Component.

BusquedaCompuesta: Composite.

BusquedaCompuestaAnd,BusquedaCompuestaOr: Composite.

BusquedaFechaCreacion, BusquedaFechaUltimaVotacion, BusquedaTipoDeInsecto, BusquedaNivelDeVerificacion: Leaf.



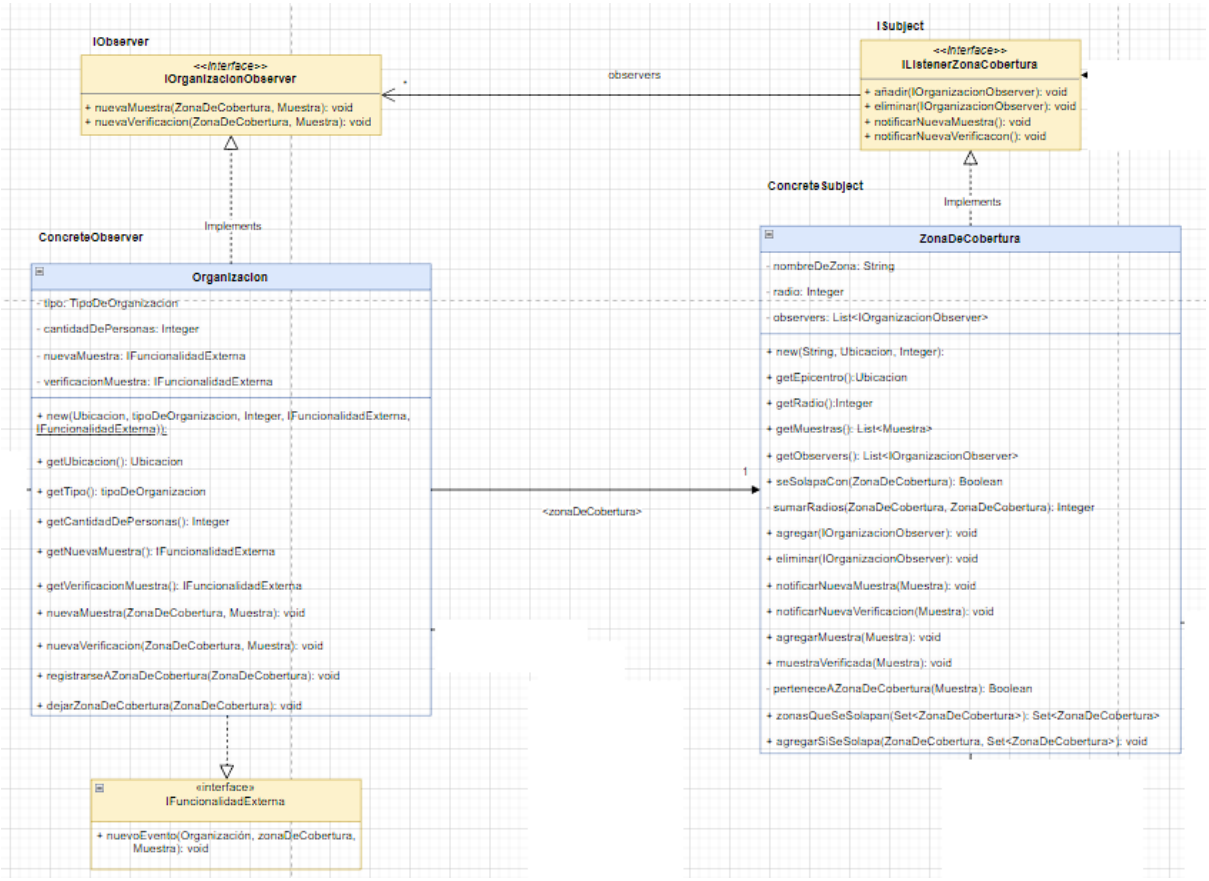
Design Patterns: Observer

Roles del observer:

ZonaDeCobertura: ConcreteSubject. El mismo envía la notificación a sus observadores cuando cambia su estado.

Organizacion: ConcreteObserver. Implementa la interfaz de actualización para mantener su estado consistente con el del sujeto.

IOrganizacionObserver: Observer.


IObserver
<<Interface>>
IOrganizacionObserver
+ nuevaMuestra(ZonaDeCobertura, Muestra): void
+ nuevaVerificacion(ZonaDeCobertura, Muestra): void
observers
ISubject
<<Interface>>
ILIstenerZonaCobertura
+ añadir(IOrganizacionObserver): void
+ eliminar(IOrganizacionObserver): void
+ notificarNuevaMuestra(): void
+ notificarNuevaVerificacon(): void
Implements
ConcreteObserver
Organizacion
- tipo: TipoDeOrganizacion
- cantidadDePersonas: Integer
- nuevaMuestra: IFuncionalidadExterna
- verificacionMuestra: IFuncionalidadExterna
+ new(Ubicacion, tipoDeOrganizacion, Integer, IFuncionalidadExterna, IFuncionalidadExterna);
+ getUbicacion(): Ubicacion
+ getTipo(): tipoDeOrganizacion
+ getCantidadDePersonas(): Integer
+ getNuevaMuestra(): IFuncionalidadExterna
+ getVerificacionMuestra(): IFuncionalidadExterna
+ nuevaMuestra(ZonaDeCobertura, Muestra): void
+ nuevaVerificacion(ZonaDeCobertura, Muestra): void
+ registrarseAZonaDeCobertura(ZonaDeCobertura): void
+ dejarZonaDeCobertura(ZonaDeCobertura): void
*
1
<zonaDeCobertura>
ConcreteSubject
Implements
ZonaDeCobertura
- nombreDeZona: String
- radio: Integer
- observers: List<IOrganizacionObserver>
+ new(String, Ubicacion, Integer):
+ getEpicentro():Ubicacion
+ getRadio():Integer
+ getMuestras(): List<Muestra>
+ getObservers(): List<IOrganizacionObserver>
+ seSolapaCon(ZonaDeCobertura): Boolean
- sumarRadios(ZonaDeCobertura, ZonaDeCobertura): Integer
+ agregar(IOrganizacionObserver): void
+ eliminar(IOrganizacionObserver): void
+ notificarNuevaMuestra(Muestra): void
+ notificarNuevaVerificacion(Muestra): void
+ agregarMuestra(Muestra): void
+ muestraVerificada(Muestra): void
- perteneceAZonaDeCobertura(Muestra): Boolean
+ zonasQueSeSolapan(Set<ZonaDeCobertura>): Set<ZonaDeCobertura>
+ agregarSiSeSolapa(ZonaDeCobertura, Set<ZonaDeCobertura>): void
«interface»
IFuncionalidadExterna
+ nuevoEvento(Organización, zonaDeCobertura, Muestra): void