

Programación orientada a objetos ||
1° cuatrimestre, 2025

A LA CAZA DE LAS VINCHUCAS

trabajo práctico integrador



Universidad
Nacional
de Quilmes

| NOMBRES | MAILS |
|-----------------|-----------------------------|
| Hernan Brito | hernanbritotps@gmail.com |
| Ivan Leguizamon | ivanleguizamon939@gmail.com |
| Sofía Natanson | sofiarnatanson@gmail.com |

Índice

| | |
|---|-----------|
| 1.Introducción..... | 3 |
| 2.¿Cómo funciona?..... | 3 |
| 3.Estructura del proyecto..... | 4 |
| Paquetes por funcionalidad - clases de src..... | 4 |
| 4.Patrones de diseño utilizados:..... | 7 |
| 4.1 Filtros - Composite..... | 7 |
| 4.2 Criterios Lógica - Strategy..... | 8 |
| 4.3 Criterios Fecha - Strategy..... | 9 |
| 4.4 Muestra - State..... | 10 |
| 4.5 Usuario - State..... | 12 |
| 4.6 AvisoOrganizaciones - Observer..... | 13 |
| 5.Bibliografía | 14 |

1.Introducción

El objetivo de este proyecto es poder realizar un mapeo de la presencia de vinchucas en el territorio argentino y así poder encontrar relaciones entre la presencia de este insecto, los casos de chagas detectados, las organizaciones que se encuentran en la región, y la posibilidad de minimizar los casos de contagio de la enfermedad de chagas; realizando esto mediante dos aplicaciones conectadas, una móvil donde se realiza la toma de la información y una web donde se reciben los datos de las muestras y se hace el procesamiento.

Este trabajo se centra solamente en la lógica de negocio de la aplicación Web.

2.¿Cómo funciona?


Se convoca la participación de las personas que viven o se encuentran en argentina para que puedan enviar fotos y una serie de respuestas a un cuestionario para informar de la aparición de este insecto. A esto se le llama envío de muestras. Como la participación es de cualquier persona, las muestras deben poder ser votadas por otros usuarios y finalmente verificadas por expertos.

En este caso, las muestras las toman las personas utilizando una aplicación móvil,, la cual fotografía a las vinchucas, indica en qué posición geográfica se encuentra y envía una serie de respuestas de un cuestionario. Todo esto es recibido por el sistema Web. Desde allí comienza el trabajo práctico.

3.Estructura del proyecto

Se ha adoptado una estructura de carpetas estándar compuesta por dos directorios principales:

 **src** : contiene el código fuente del proyecto

 **tests** : contiene los tests del proyecto

Paquetes por funcionalidad - clases de src

- **ubicación**: contiene la clase Ubicacion que representa una ubicación geográfica mediante latitud y longitud. Permite calcular la distancia entre ubicaciones usando la fórmula de Haversine y filtrar ubicaciones o muestras dentro de un radio determinado.
- **criterio**: contiene la clase BuscadorMuestra, que se encarga de filtrar muestras aplicando distintos criterios de búsqueda.
- **sistema**: contiene la clase Sistema, encargada de coordinar y gestionar los distintos componentes de la aplicación, permitiendo la administración centralizada de usuarios, muestras, zonas de cobertura y organizaciones.
- **zonaCobertura**: contiene las clases e interfaces que permiten representar zonas geográficas mediante un epicentro y un radio de alcance. Facilita la detección de muestras dentro de la zona y gestiona la notificación de observadores cuando se registran o validan muestras en dichas áreas.

-
- **organizacion:** contiene la clase Organizacion interesada en conocer eventos que ocurren dentro de una o varias zonas de cobertura, sobre las cargas de nuevas muestras o muestras validadas, estas organizaciones pueden configurar cómo reaccionan a estos eventos mediante funcionalidades externas.
 - **muestra:** contiene la clase Muestra, que representa una serie de respuestas a un cuestionario para informar de la posible aparición de las vinchucas junto con una imagen adjunta. Esta clase permite centralizar el proceso de identificación de un muestra de posible caso de aparición de vinchucas, conteniendo opiniones tanto de profesionales como no profesionales.

También contiene los diferentes estados de una muestra, EstadoVerificacionMuestra, EstadoMuestraVerificandose y EstadoMuestraVerificada, los cuales pueden confirmar la identificación del animal; estados como una muestra no verifica, en proceso de verificación o ya verificada.

- **usuario:** Contiene las clases relacionadas con la gestión de usuarios en el sistema: Nombre del usuario, estado actual (básico, experto o verificado), métodos para calcular envíos y revisiones en los últimos 30 días.(Usuario/UsuarioVerificado serían el contexto según el libro Gamma)
- **estadoUsuario:** implementa el patrón State para manejar los diferentes estados de un usuario mediante la interfaz EstadoUsuario, La cual define métodos para: Verificar si el usuario es experto, actualizar su estado según actividad, determinar si cumple condiciones para cambiar de estado. Los estados son: UsuarioBasico, UsuarioExperto y UsuarioVerificado, este último siempre es verificado sin importar si cumple con las condiciones para ser experto(UsuarioBasico/UsuarioExperto/UsuarioVerificado serían el state según el libro Gamma)
- **opinion:** Modela una opinión que se agrega en una muestra y guarda: El usuario que la emitió, el tipo de opinión , la fecha y si está misma es verificada. También

tiene el enum TipoOpinion que define los tipos: Especies de vinchucas, especies de chinches y casos especiales (imagen poco clara y no definida)

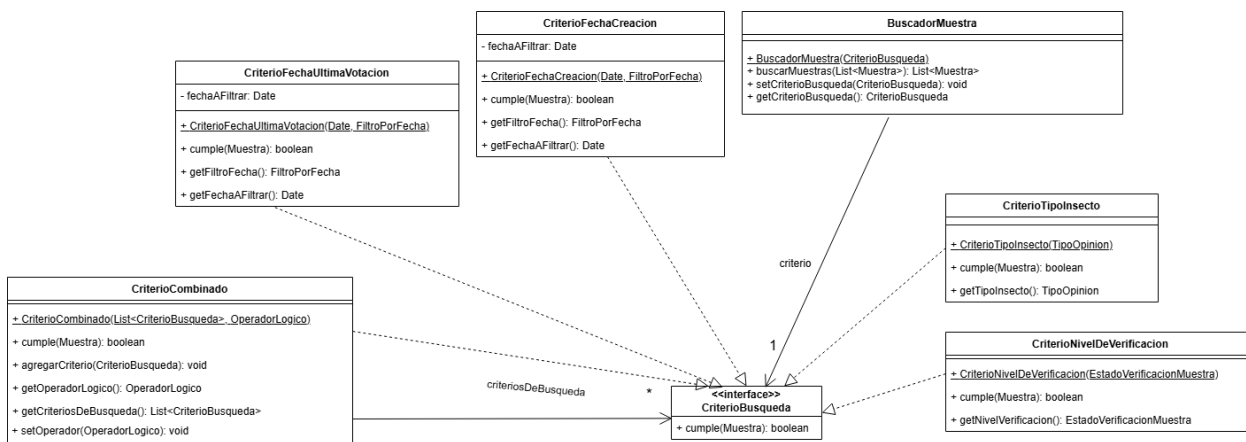
- **avisoOrganizaciones:** Contiene la clase ObservadorMuestra el cual es el puente entre las muestras y las zonas de cobertura, asegurando que los eventos críticos se comuniquen a las áreas geográficas relevantes. Su diseño sigue el patrón Observer, donde las zonas son "observadoras" que reaccionan a cambios en las muestras.

4. Patrones de diseño utilizados:

A lo largo del diseño de este trabajo se detectaron problemas a los cuales les eran aplicables estrategias y patrones de diseños orientados a objetos vistos en clase, a continuación se nombran algunos de estos que se quieren resaltar.

4.1 Filtros - Composite

Consta de búsquedas de muestras basadas en filtros en la que se aplicó el patrón.



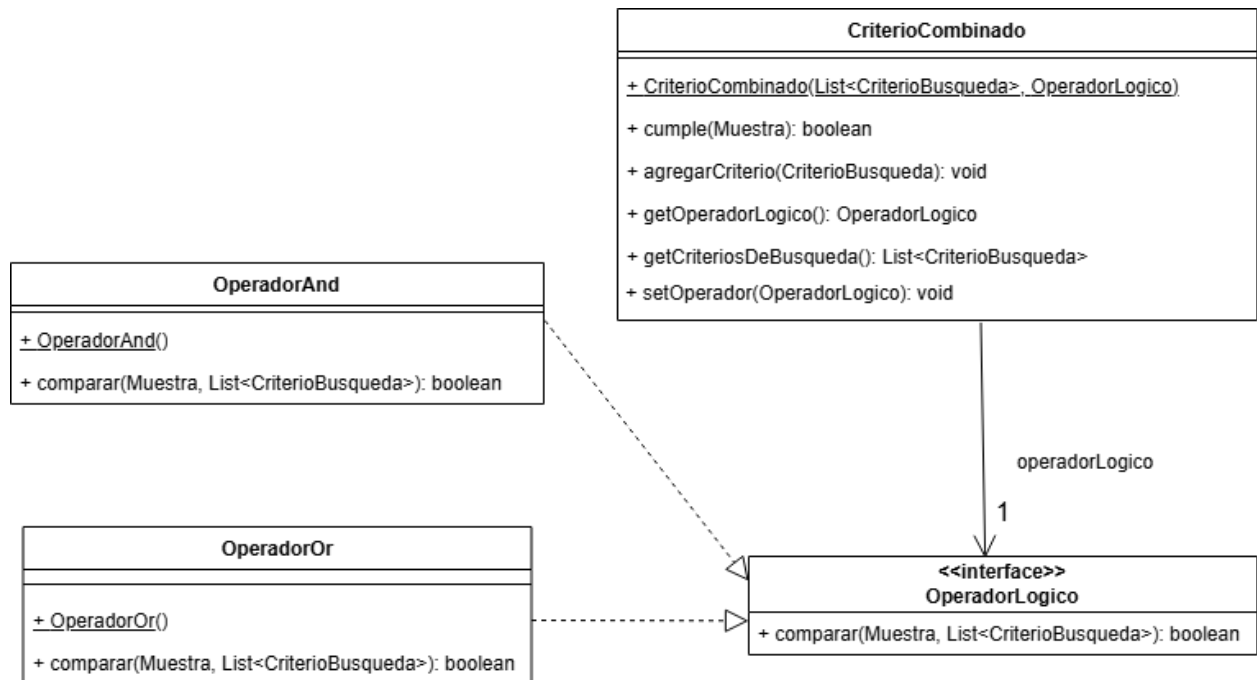
El patrón **Composite** se utiliza en el diseño de los criterios de búsqueda de muestras porque permite tratar objetos individuales (criterios simples) y composiciones de objetos (criterios combinados) de manera uniforme.

Participantes:

- **Client:** BuscadorMuestra
- **Component:** CriterioBusqueda
- **Leaf:** CriterioNivelVerificacion, CriterioTipoinsecto, CriterioFechaCreacion, CriterioFechaUltimaVotacion
- **Composite:** CriterioCombinado

4.2 Criterios Lógica - Strategy

Se utilizan criterios combinados con una estrategia (Operador lógico)



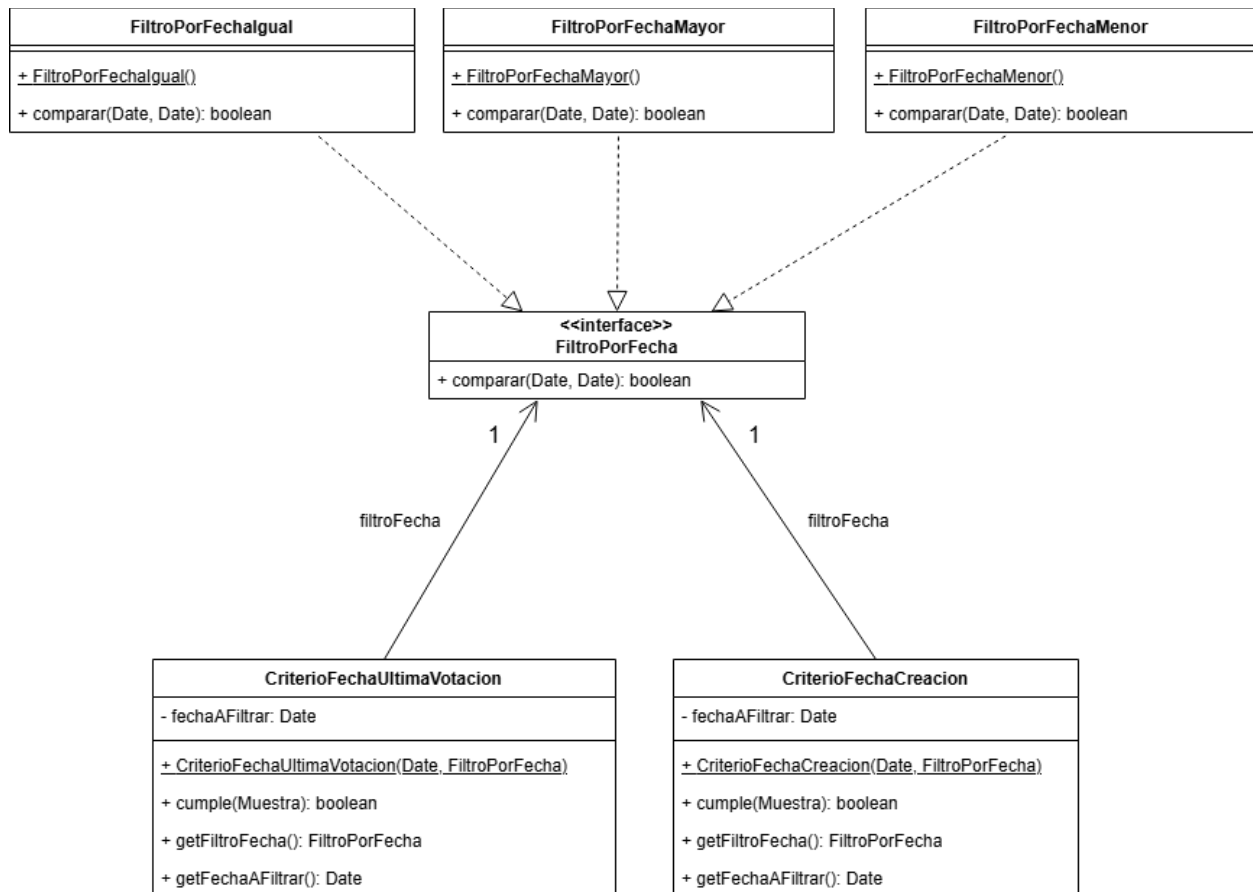
El patrón **Strategy** se aplica en el diseño de los criterios de búsqueda para encapsular las diferentes formas en que se pueden combinar múltiples criterios. En particular, la clase **CriterioCombinado** necesita combinar los resultados de varios criterios hijos utilizando una lógica booleana que puede ser **AND** o **OR**.

Participantes:

- **Strategy:** OperadorLogico
- **ConcreteStrategy:** OperadorAnd, OperadorOr
- **Context:** CriterioCombinado

4.3 Criterios Fecha - Strategy

Consta de criterios de fechas según una estrategia (Filtro por Menor, Mayor o Igual).



El patrón **Strategy** se aplica en el diseño de los criterios de búsqueda de fechas para encapsular las diferentes formas en que una fecha puede ser comparada (igual, mayor, menor).

Participantes:

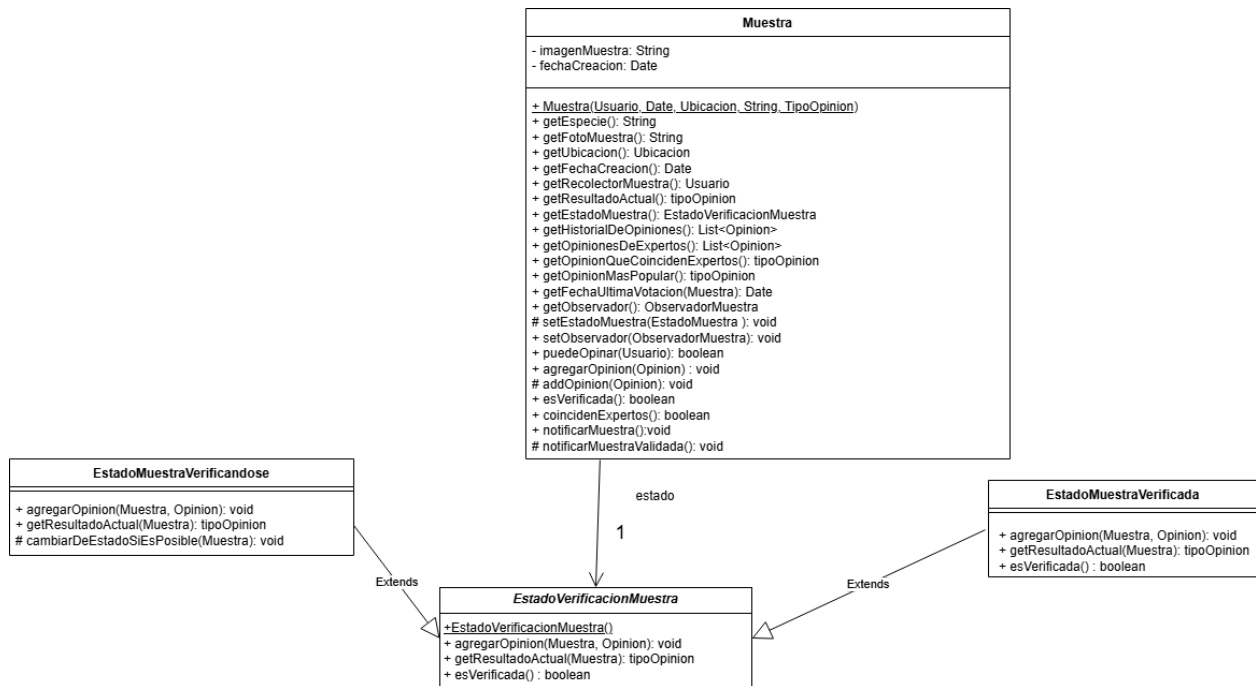
- **Strategy:** FiltroPorFecha
- **ConcreteStrategy:** FiltroPorFechaIgual, FiltroPorFechaMayor, FiltroPorFechaMenor
- **Context:** CriterioFechaUltimaVotacion, CriterioFechaCreacion

Justificación de la Decisión de Diseño (Uso de Strategy para las fechas):

Para los criterios de búsqueda por **fecha de creación** y **fecha de última votación**, el enunciado no especificaba el tipo de comparación a realizar. Por ello, se decidió aplicar el **patrón Strategy** para permitir distintas formas de comparar fechas (mayor, menor o igual).

4.4 Muestra - State

Se aplica el **patrón State** sobre la clase Muestra y sus estados.



Se decidió utilizar este patrón de diseño sobre las muestras y sus estados ya que una muestra a lo largo de su vida pasa por diferentes etapas, estados, y estos estados son los que determinan varios de sus comportamientos. Este patrón permite encapsular esta variación de comportamientos de una clase que depende de su estado

Participantes:

- **Context** : Muestra
- **State** : EstadoVerificacionMuestra
- **ConcreteStates** : EstadoMuestraVerificandose, EstadoMuestraVerificada

Justificación de la Decisión de Diseño (Uso de State para las Muestras):

cumple con la intención de este patrón y los casos de aplicabilidad.

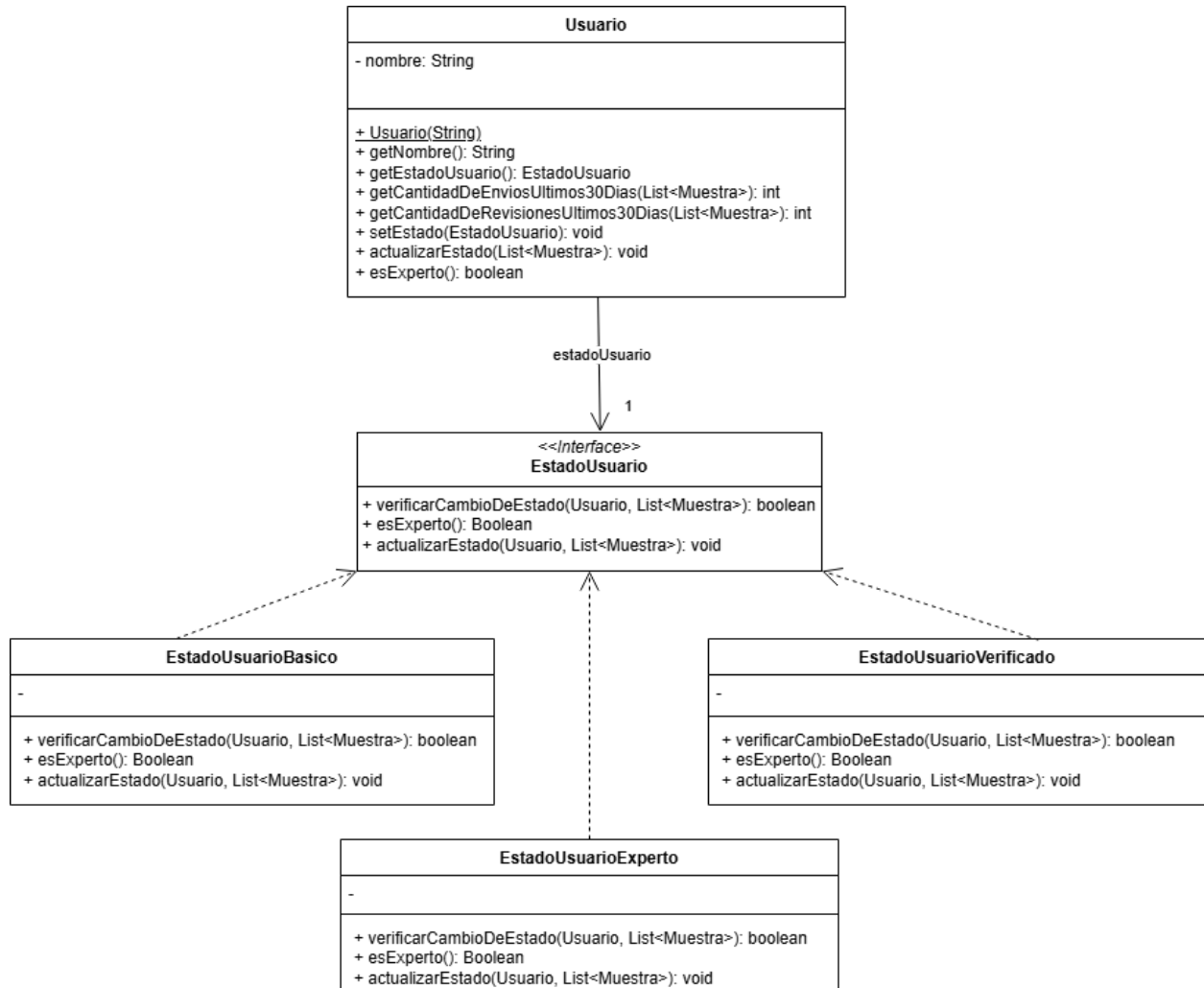
“Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.”

Gamma, Erich, et al. *Design Patterns*. 1995. Accessed 17 June 2025.

El enunciado hace una clara diferencia de los comportamientos de la muestra en función de ciertos criterios, que determinan un estado de esta, denotando un comportamiento diferente en la muestra según estos estados. Una muestra puede reaccionar de diferentes maneras a un mismo mensaje, por esto, si no se usase este patrón para varios comportamientos en esta clase se tendría una larga lista de condicionales repitiendo un patrón de condiciones dentro de cada método de esta clase.

4.5 Usuario - State

En la clase usuario se implementa el patrón **State** para manejar los diferentes estados de un usuario.

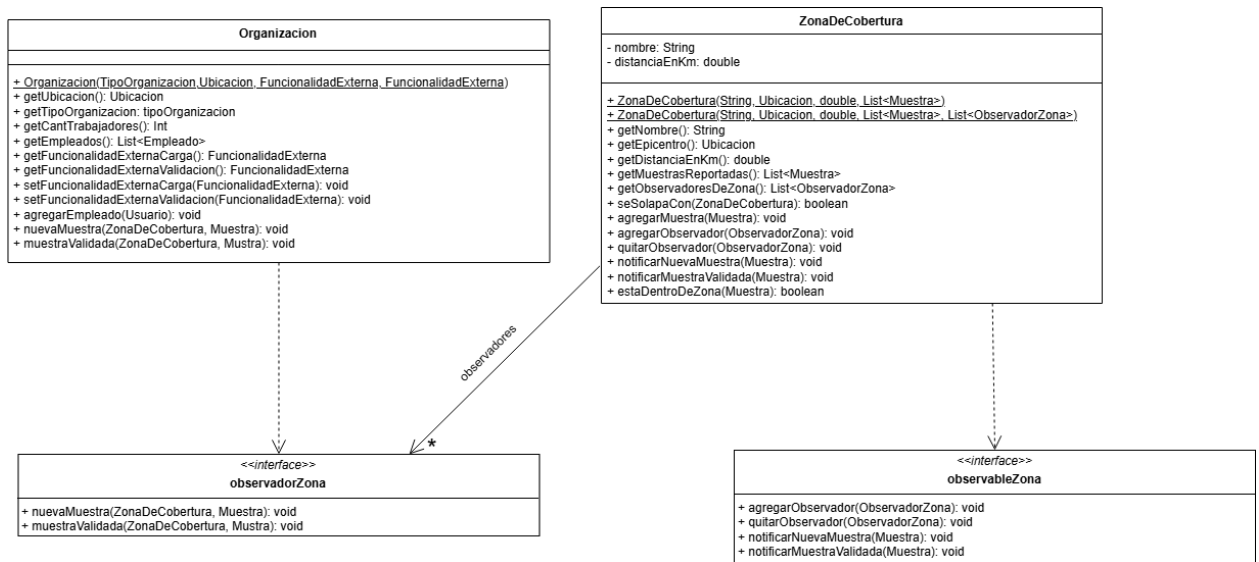


Participantes:

- **Context** : Usuario
- **State** : EstadoUsuario
- **ConcreteStates** : UsuarioBasico, UsuarioExperto y UsuarioVerificado

4.6 Aviso Organizaciones - Observer

En la clase AvisoOrganizaciones se sigue el patrón **Observer**, donde las zonas son "observadoras" que reaccionan a cambios en las muestras.



Participantes:

- **Subject** : ObservableZona
- **Observer** : ObservadorZona
- **ConcreteSubject** : ZonaDeCobertura
- **ConcreteObserver** : Organizacion

5. Bibliografía

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.