PROGRAMACIÓN ORIENTADA A OBJETOS II

TRABAJO GRUPAL FINAL

A LA CAZA DE LAS VINCHUCAS

Integrantes:

Misiukowiec Pablo Moises (Comisión 2)

- pablo.misiukowiec@gmail.com

Cáceres Daniel (Comisión 1)

- dragonaladodera2002@hotmail.com

Troche Leonardo Martin (Comisión 1)

- leo.m.troche@gmail.com

MUESTRAS

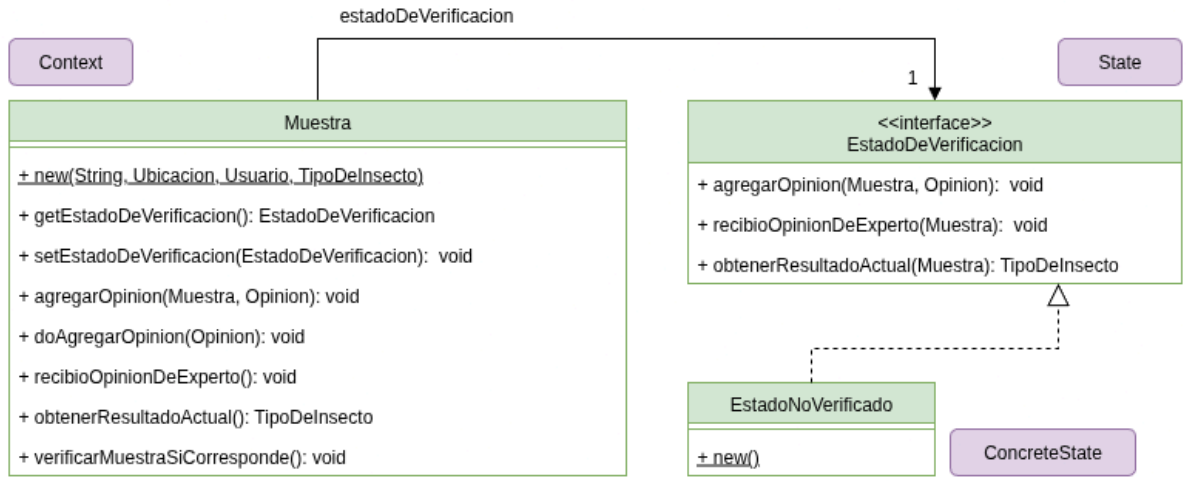
El ciclo de vida de una muestra incluye distintas fases que reflejan su nivel de validación dentro del sistema.

A medida que las opiniones de los usuarios —ya sean básicos o expertos— son recibidas, la muestra puede cambiar de estado, lo que modifica su comportamiento y las reglas aplicables en cada etapa.

Patrones de Diseño

Para modelar el ciclo de vida de una muestra y su comportamiento cambiante frente a las opiniones recibidas, se utilizó el **Patrón State**.

Este patrón permite que el objeto Muestra delegue su comportamiento a diferentes objetos que representan su estado actual, lo que evita condicionales complicados y mejora tanto la claridad como la extensibilidad del diseño.



Se definieron tres estados concretos:

- **EstadoNoVerificado**: Estado inicial, donde cualquier usuario (básico o experto) puede opinar.
- **EstadoEnProceso**: Se activa cuando ya ha opinado al menos un experto. A partir de este punto, solo se permiten opiniones de expertos.
- **EstadoVerificado**: Estado final, alcanzado cuando dos expertos coinciden en su opinión. No se aceptan nuevas opiniones.

Cada uno de estos estados implementa una interfaz común que define las operaciones relevantes, como agregarOpinion y obtenerResultadoActual.

Decisiones de Diseño

Se optó por separar completamente la lógica de cada estado en clases distintas, para evitar que la clase Muestra asumiera múltiples responsabilidades.

Esta decisión mejora la cohesión interna del sistema, facilita el mantenimiento y permite futuras extensiones. Por ejemplo, si se quisiera incorporar un nuevo estado (como "Muestra contaminada"), bastaría con definir una nueva clase de estado sin modificar las existentes.

Detalles de Implementación

El cálculo del resultado actual de una muestra depende de su estado y se delega a cada implementación concreta del patrón State, siguiendo estos criterios:

- **EstadoNoVerificado**:

El resultado se calcula agrupando las opiniones por TipoDeInsecto. La agrupación con mayor cantidad de votos se toma como resultado actual. Esta lógica está encapsulada dentro del estado, evitando condicionales externos.

- **EstadoEnProceso**:

La primera opinión de un experto se toma como resultado inicial. Mientras no haya coincidencia entre al menos dos expertos, el resultado se considera "no definido" por empate. Una estructura interna gestiona el historial de opiniones expertas y controla las transiciones.

- **EstadoVerificado**:

Una vez verificada la muestra (cuando dos expertos coinciden), se bloquean nuevas opiniones. El resultado final corresponde a la última opinión registrada que coincide con el criterio de verificación.

USUARIOS

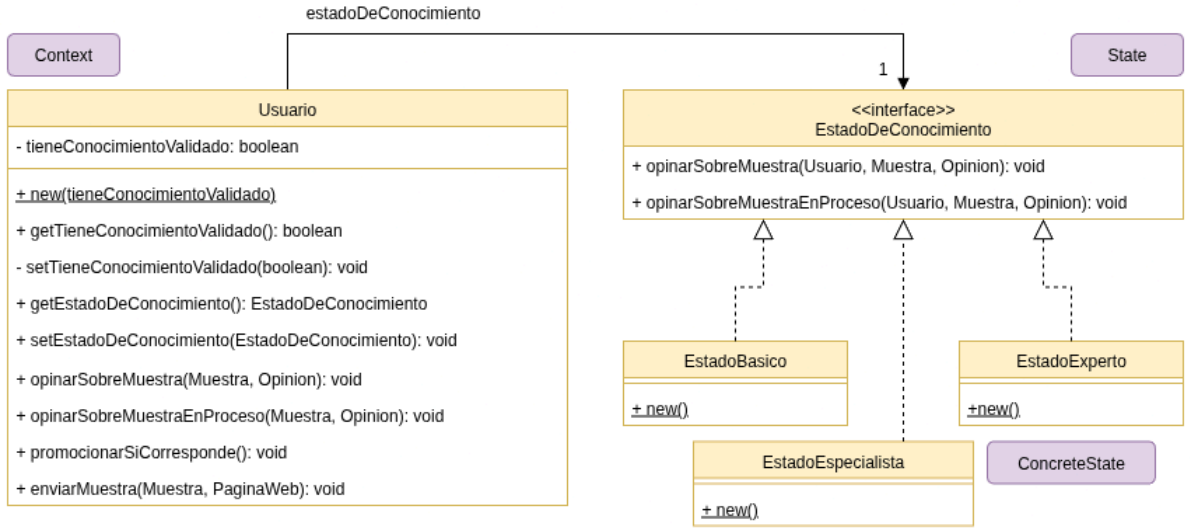
En el sistema, los usuarios desempeñan un rol central en el proceso de validación de muestras mediante la emisión de opiniones. Sin embargo, no todos poseen el mismo nivel de autoridad, ya que su comportamiento varía según su experiencia y validación institucional.

Para reflejar estas diferencias de manera estructurada y flexible, se implementó una lógica que permite adaptar dinámicamente las acciones permitidas a cada usuario, de acuerdo con su estado actual dentro del sistema.

Patrones de Diseño

Para modelar este comportamiento dinámico, se aplicó el **Patrón State**.

Este patrón permite que el objeto Usuario delegue su comportamiento a objetos que representan su estado actual, evitando estructuras condicionales extensas y favoreciendo un diseño claro y extensible.



Se definieron tres estados concretos:

- **EstadoBasico**: Los usuarios en este estado pueden opinar sobre muestras, pero sus votos pierden efecto una vez que interviene un experto.
- **EstadoExperto**: Representa a usuarios con conocimientos validados en el dominio. Sus opiniones tienen prioridad y reemplazan a las de usuarios básicos.
- **EstadoEspecialista**: Es un nivel superior al de experto, reservado para usuarios con validación científica o institucional. Este estado puede incluir permisos adicionales en futuras extensiones del sistema.

Cada estado implementa una interfaz común que define las operaciones habilitadas, permitiendo que el objeto Usuario modifique su comportamiento interno sin necesidad de lógica condicional explícita.

Decisiones de Diseño

Además del uso del patrón State, se tomaron las siguientes decisiones clave:

- El estado inicial del usuario se asigna en el momento de su instanciación, según la existencia o no de validación institucional de conocimientos:
    - Si no posee validación externa, se lo inicializa en **EstadoBasico**.
    - Si cuenta con validación institucional, se lo instancia directamente en **EstadoEspecialista**.

- Se implementó una lógica de cambio de estado dinámico aplicable únicamente a los usuarios sin validación institucional.
  Estos usuarios pueden alternar entre **EstadoBasico** y **EstadoExperto**, de acuerdo con su participación, la calidad de sus opiniones u otras métricas internas del sistema.

- Los usuarios en **EstadoEspecialista** no están sujetos a esta lógica, ya que su validación es externa, permanente y estable. Por lo tanto, mantienen su estado fijo durante todo el ciclo de vida del sistema.

De este modo, el sistema respeta y diferencia claramente los niveles de experiencia y responsabilidad asignados a cada tipo de usuario.

Detalles de Implementación

No se requieren elementos adicionales a los ya descritos.

La implementación del patrón State es directa: cada clase de estado encapsula el comportamiento específico correspondiente, y el objeto Usuario delega en ella sus operaciones relevantes.

AVISO A ORGANIZACIONES

Las organizaciones interesadas pueden suscribirse a distintos eventos geográficos mediante las zonas de cobertura. Actualmente, el sistema contempla dos eventos principales: la **carga de una nueva muestra** y **la validación de una muestra**, aunque está diseñado para soportar más eventos en el futuro.

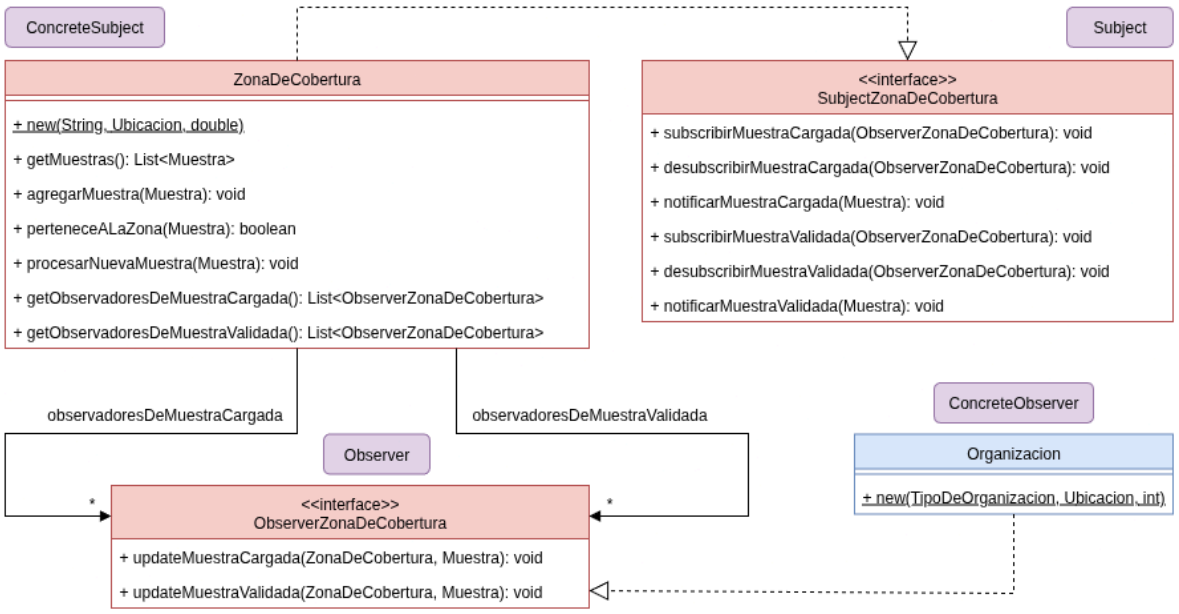
Patrones de Diseño

Para modelar la comunicación entre entidades emisoras de eventos y sus observadores, se implementó el **Patrón Observer**, aplicado en distintos niveles del sistema.

Flujo de Eventos — Carga de Muestra

Cuando se carga una nueva muestra desde la interfaz web, el sistema delega el procesamiento a todas las instancias de ZonaDeCobertura. Cada una evalúa si la ubicación de la muestra se encuentra dentro de su área geográfica. En caso afirmativo:

1. La muestra se agrega a la lista interna de la zona.
2. La ZonaDeCobertura se suscribe como observadora de la muestra para recibir futuras notificaciones, como la validación.
3. Se notifica a las organizaciones suscritas a la zona correspondiente que una nueva muestra ha sido cargada, de modo que puedan ejecutar sus funcionalidades externas asociadas en respuesta al evento.



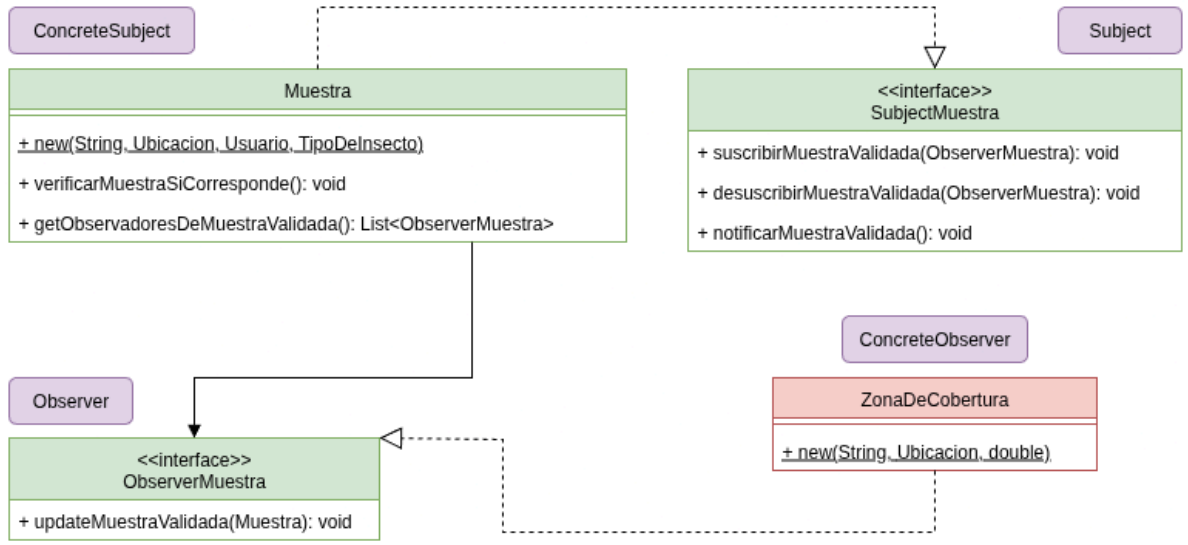
Flujo de Eventos — Validación de Muestra

La suscripción al evento de validación está reservada para aquellas entidades que implementen la interfaz ObserverMuestra, como lo hace ZonaDeCobertura.

Esto significa que, al momento de cargarse una muestra dentro del área geográfica de una zona, dicha zona comienza a observarla en espera de eventos relevantes.

Cuando la muestra cambia su estado a Verificada, según los criterios establecidos, se notifica a todas las zonas de cobertura que la estén observando.

A su vez, la ZonaDeCobertura ejecuta el método updateMuestraValidada(Muestra) y notifica a todas las organizaciones suscritas que la muestra ha sido validada, permitiéndoles ejecutar sus funcionalidades externas asociadas.



Decisiones de Diseño

Inicialmente, se evaluó la incorporación de un gestor de eventos (event manager) con el objetivo de centralizar la lógica relacionada con la implementación del patrón Observer.

No obstante, para la reentrega del trabajo práctico, se decidió eliminar dicha abstracción a fin de simplificar la arquitectura del sistema y hacer más explícita la responsabilidad de cada entidad dentro del patrón.

Esta decisión permitió una mejor separación de responsabilidades y facilitó la comprensión del flujo de eventos.

Detalles de Implementación

La lógica resultante es más sencilla y clara. Se implementaron dos listas diferenciadas de observadores:

- Una para el evento de muestra cargada
- Otra para el evento de muestra validada

De esta manera, cada entidad se suscribe únicamente a los eventos que le son relevantes, evitando notificaciones innecesarias y manteniendo un bajo acoplamiento entre componentes.

Esto mejora la mantenibilidad del código y la escalabilidad del sistema frente a la incorporación de nuevos eventos o tipos de observadores.

FUNCIONALIDADES EXTERNAS

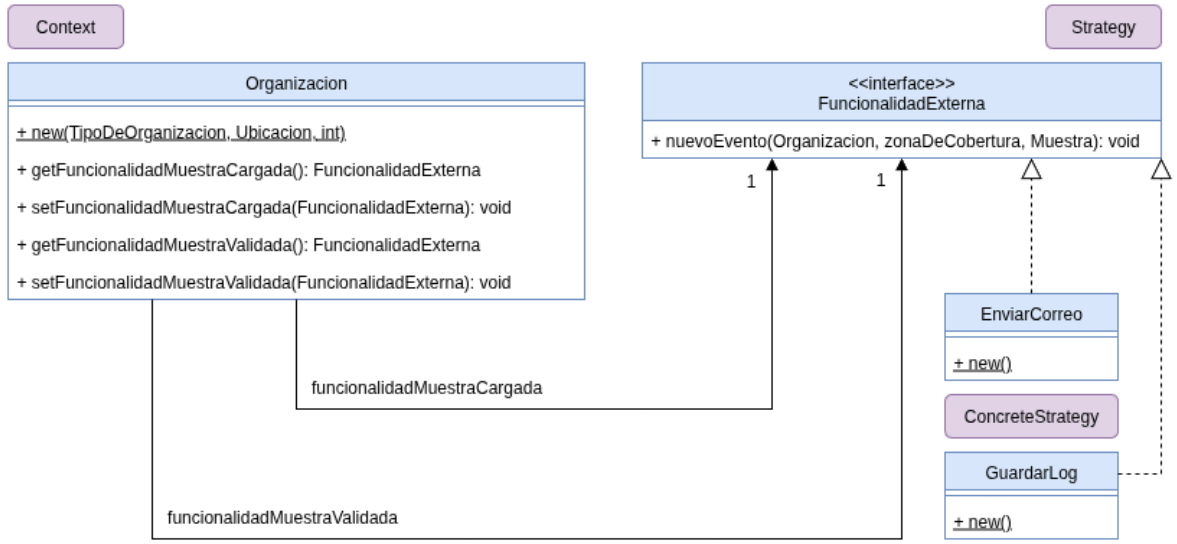
Cuando una Organización es notificada de un evento —como la carga o validación de una Muestra—, se ejecuta una funcionalidad externa asociada a dicho evento, lo que le permite realizar acciones específicas, como enviar notificaciones por correo, registrar información en un log, entre otras.

Cada tipo de evento puede tener una única funcionalidad activa, la cual puede ser modificada dinámicamente en tiempo de ejecución.

Patrones de Diseño

Para modelar las funcionalidades externas como comportamientos intercambiables, se utilizó el **Patrón Strategy**.

Cada funcionalidad concreta implementa una interfaz común: FuncionalidadExterna.



Decisiones de Diseño

Al igual que en el caso del patrón Observer, inicialmente se consideró centralizar la lógica en un gestor de funcionalidades externas.

Sin embargo, con el objetivo de simplificar la arquitectura y hacer más evidente el rol de cada entidad dentro del patrón Strategy, se optó por eliminar dicha abstracción.

Detalles de Implementación

No se requieren elementos adicionales a los ya descritos.

Por defecto, las organizaciones no poseen funcionalidades externas cargadas; estas deben ser asignadas manualmente una vez instanciada la entidad.

BÚSQUEDA DE MUESTRAS

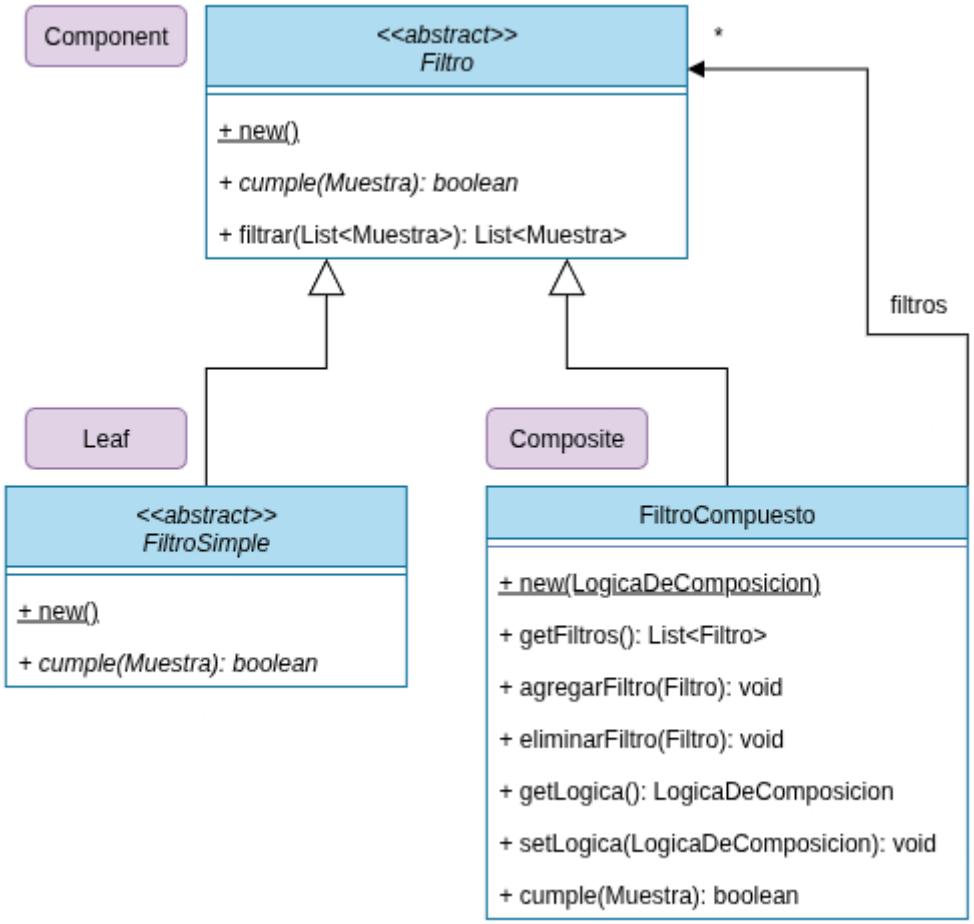
Patrones de Diseño

Para implementar un sistema flexible y extensible de búsqueda de muestras, se utilizaron dos patrones de diseño: **Composite** y **Strategy**.

La elección de estos patrones permitió construir filtros reutilizables, combinables y fácilmente adaptables a diferentes necesidades.

Patrón Composite

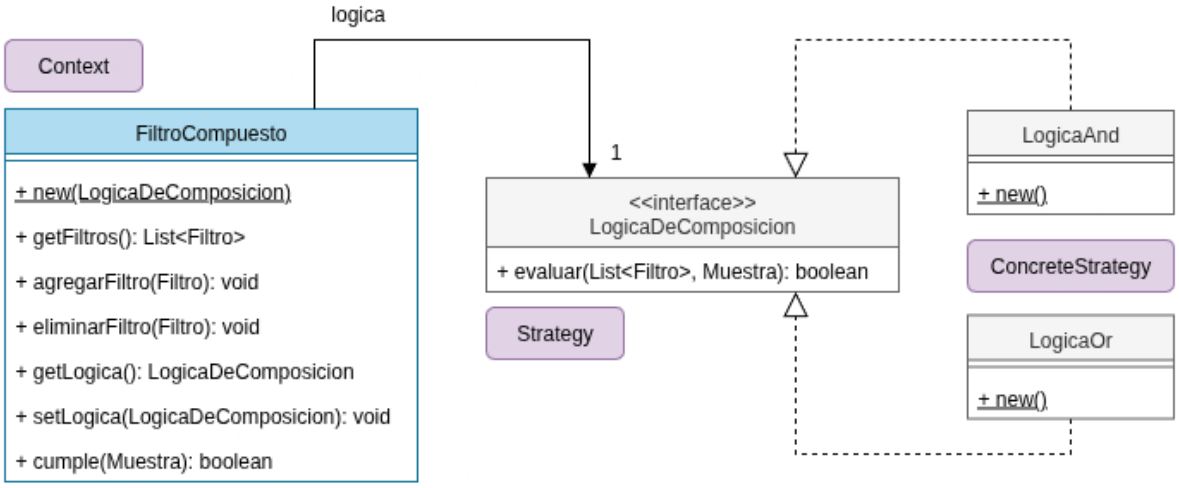
Este patrón se usó para representar filtros de búsqueda que pueden ser simples o compuestos. Es decir, filtros que funcionan por sí solos (como buscar por fecha o por categoría) y filtros que combinan varios criterios a la vez.



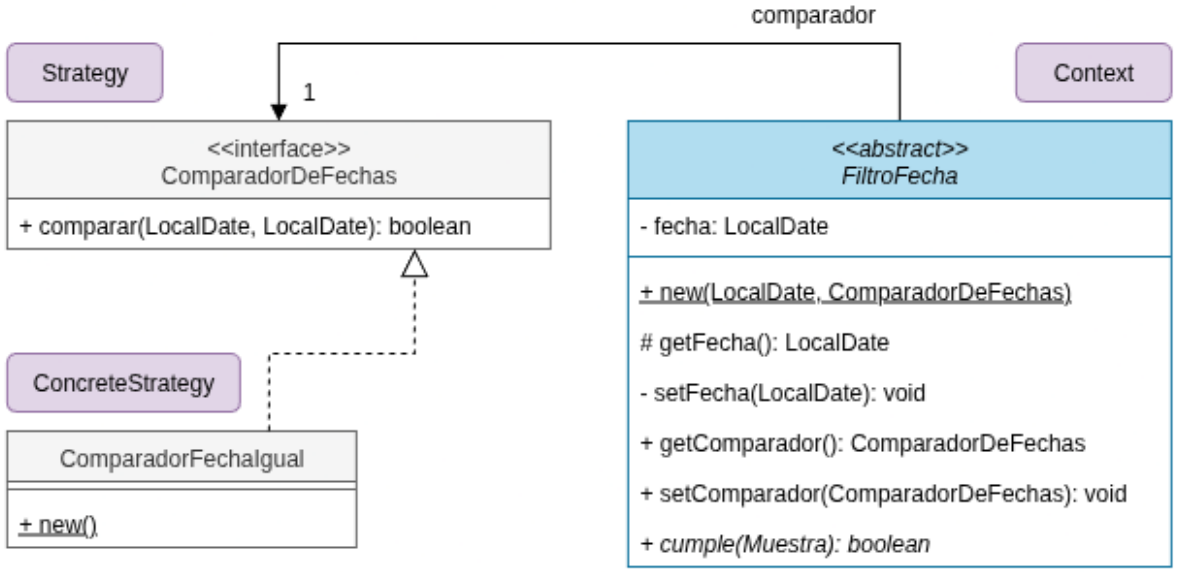
El patrón permite trata a todos los filtros de la misma forma, sin importar si son simples o compuestos. Esto facilita su uso y combinación en distintos escenarios de búsqueda.

Patrón Strategy

Este patrón se aplicó para poder cambiar de forma dinámica cómo se combinan los filtros, según la necesidad del usuario. Por ejemplo, si se quieren muestras que cumplan todas las condiciones (AND) o alguna de ellas (OR).



También se usó este patrón para los filtros de fecha, un tipo de filtro simple, permitiendo cambiar fácilmente el tipo de comparación a realizar: si la fecha es anterior, posterior o igual a una determinada.



Como se puede apreciar, el método, heredado de la clase Filtro, sigue siendo abstracto, de modo que las clases que extienden FiltroFecha —como FiltroFechaDeCreacion y FiltroFechaDeUltimaVotacion— implementen la lógica específica según la fecha que les corresponde.

Decisiones de Diseño

En la clase abstracta Filtro, se implementó un método filtrar marcado como final, encargado de determinar qué muestras cumplen con el criterio del filtro.

De este modo, cada subclase —ya sea un filtro simple o compuesto— debe implementar su propia lógica en el método cumple para evaluar si una muestra satisface el filtro correspondiente.

Detalles de Implementación

En esta sección no hay aspectos destacables, ya que el código de cada clase es bastante sencillo y directo, sin complicaciones ni detalles que necesiten una explicación adicional.

OTRAS ENTIDADES

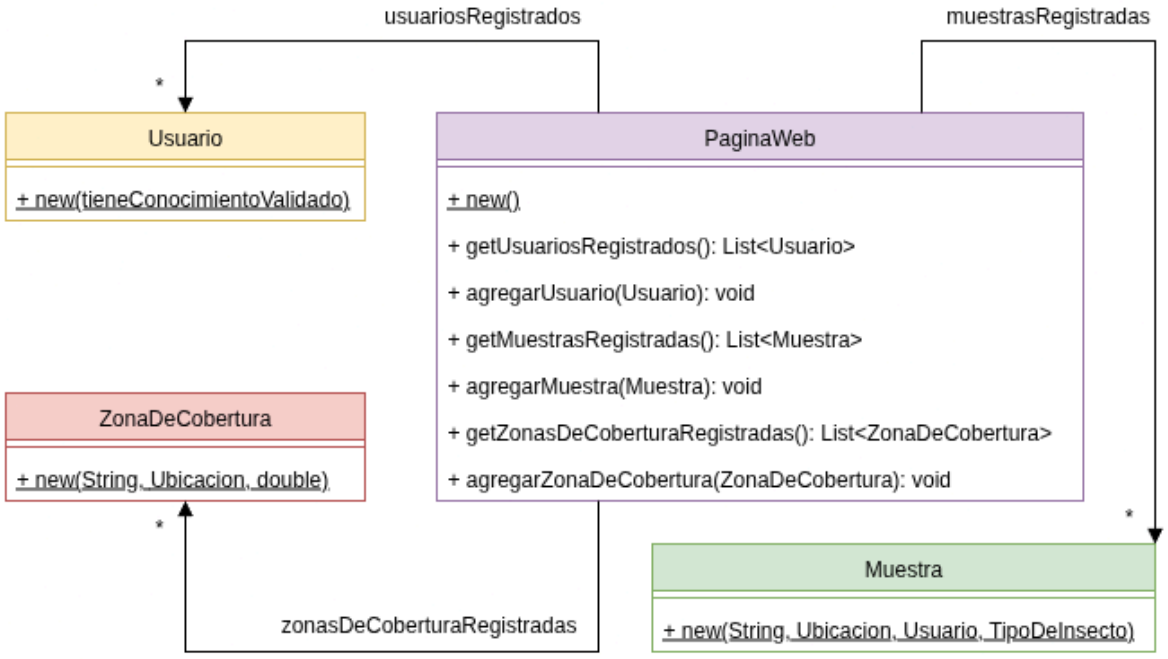
Además de las entidades principales, el sistema cuenta con otros componentes fundamentales para su correcto funcionamiento:

PaginaWeb

Representa el núcleo central del sistema, ya que concentra y gestiona a todos los usuarios, muestras y zonas de cobertura. Actúa como punto de entrada principal y coordinador general de las operaciones del sistema.

Sus responsabilidades principales incluyen:

- Registrar y administrar a todos los usuarios del sistema.
- Almacenar y administrar todas las muestras recibidas.
- Registrar y gestionar las zonas de cobertura geográfica.



Además, desde esta entidad se gestionan las notificaciones hacia las organizaciones suscritas a eventos en zonas específicas.

Cuando se carga una nueva muestra, PáginaWeb delega su procesamiento a las zonas de cobertura. Una vez identificada la zona correspondiente, será esta la encargada de notificar a las organizaciones interesadas.

De este modo, PáginaWeb funciona como el nexo central y repositorio global de toda la información y funcionalidad del sistema.

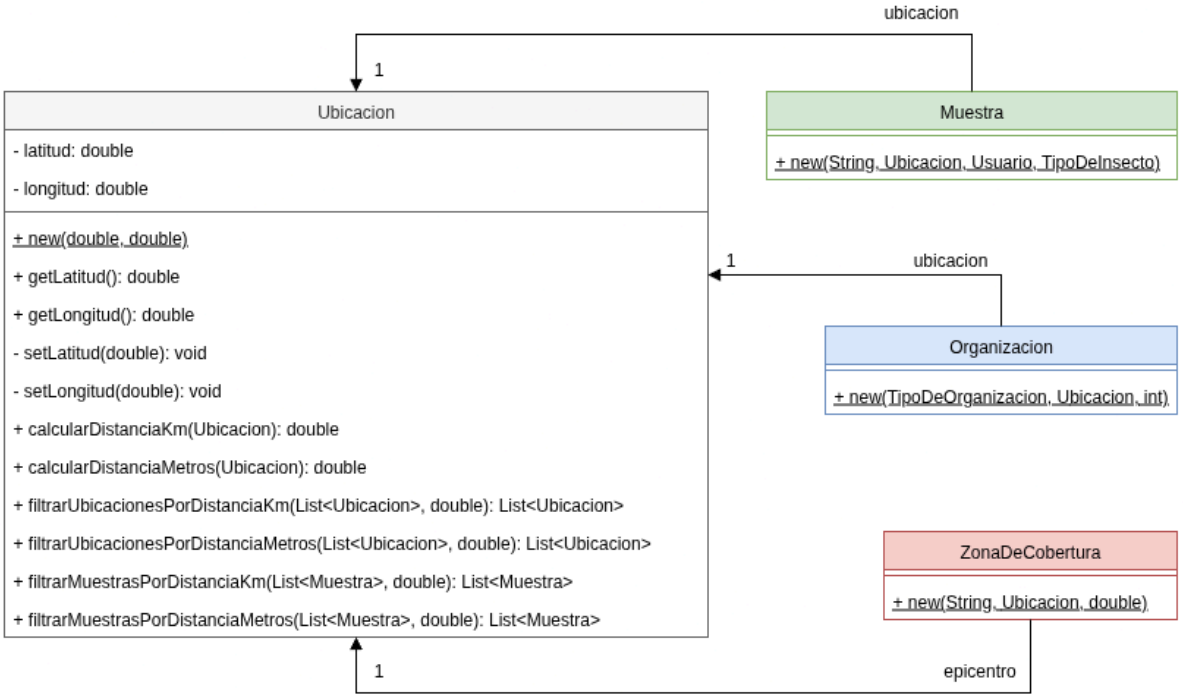
Ubicacion

Modela un punto geográfico mediante sus coordenadas de latitud y longitud.

Esta entidad es fundamental para calcular distancias entre muestras, zonas de cobertura y organizaciones.

Sus responsabilidades principales incluyen:

- Calcular la distancia entre dos ubicaciones, devolviendo el resultado en metros o kilómetros según se requiera.
- Filtrar listas de ubicaciones o muestras, devolviendo solo aquellas que se encuentren dentro de un radio determinado respecto a una ubicación de referencia.



Esta funcionalidad permite saber qué tan cerca están dos lugares, lo cual es fundamental para saber si una muestra pertenece a una zona de cobertura o para buscar muestras que estén dentro de un área determinada.

CONCLUSIONES

Realizar este trabajo práctico fue todo un desafío.

Durante la primera entrega, **el principal obstáculo fue la falta de tiempo**, lo cual, combinado con **algunas dificultades iniciales de coordinación**, nos impidió llegar con todos los aspectos completamente resueltos.

Si bien logramos entregar una parte importante del sistema funcionando correctamente —especialmente en lo relacionado con **la interacción entre el estado de las muestras y el conocimiento de los usuarios**, que fue uno de los aspectos más complejos—, **quedó la sensación de que el trabajo podría haberse desarrollado con mayor profundidad y mejor organización**.

Para la reentrega, **el equipo mejoró notablemente la organización y el trabajo colaborativo**. Usamos **Git con mayor profundidad**, lo cual nos permitió distribuir tareas de forma más clara, llevar un mejor control de los avances y evitar pisarnos entre nosotros. **Esta mejora fue clave para poder avanzar de manera más ordenada y enfocada**.

En cuanto al desarrollo, enfrentamos varios desafíos de diseño.

Uno de los más relevantes fue construir **un sistema de búsqueda flexible y combinable**, que resolvimos mediante una combinación de estrategias que aprendimos durante el curso.

También tomamos decisiones importantes como **simplificar estructuras que, aunque teóricamente correctas, sumaban complejidad innecesaria al código**.

Por otro lado, nos ocupamos de resolver con claridad la lógica de verificación de muestras y de **modelar correctamente las ubicaciones geográficas**, aplicando buenas prácticas sin complicar la implementación.

En resumen, **el trabajo fue una oportunidad de aprendizaje integral**, tanto desde lo técnico como en lo grupal. **Pudimos superar los obstáculos iniciales y presentar una solución robusta, bien pensada y funcional**, en la que todos los integrantes del grupo aportaron activamente.