INFORME TP INTEGRADOR “LA CAZA DE LAS VINCHUCAS”

INTEGRANTES: Escalante Joaquin, Pasquale Alexander, Laghezza Luca

COMISIÓN: 2

FECHA DE ENTREGA: 26 de junio

CLASES CREADAS Y SUS COMPORTAMIENTOS

Aplicacion: Clase principal de la lógica del sistema. Actúa como fachada que coordina la interacción entre usuarios, muestras, zonas y organizaciones.

SistemaDeUsuarios: Administra los usuarios del sistema. Sus comportamientos incluyen el alta de usuarios con validación de nombre único y la consulta de usuarios registrados.

SistemaDeExcepciones: Clase personalizada para lanzar excepciones relacionadas al dominio, encapsulando errores como nombre de usuario duplicado o nombre vacío.

Muestra: Representa una denuncia enviada por un usuario sobre un posible insecto. Almacena foto, ubicación, fecha y autor, gestiona el estado (verificada o no) según las opiniones recibidas y coordina con SistemaDeOpiniones.

Opinion: Representa un voto u opinión sobre una muestra, emitido por un usuario. Sus atributos incluyen el usuario que opina, el tipo de opinión (enum TipoDeOpinion) y si era experto al momento de opinar.

SistemaDeOpiniones: Gestiona el proceso de votación sobre una muestra. Sus responsabilidades son agregar y validar opiniones, llevar un historial textual, calcular el resultado final y verificar si la muestra debe considerarse verificada.

RepositorioDeMuestras: Implementa IRepositorioDeMuestras. Almacena y filtra muestras, devuelve estadísticas (total, verificadas/no verificadas) y aplica filtros individuales o compuestos.

RepositorioDeOpiniones: Contenedor y gestor de opiniones asociadas a muestras. Cada usuario tiene uno, es donde se aloja cada opinion realizada.

ValidadorDeOpiniones: Encapsula la lógica de validación para decidir si un usuario puede opinar. Valida que el usuario no haya opinado ya y que la muestra no esté verificada.

Usuario: La clase Usuario representa a una persona que interactúa activamente con el sistema, ya sea enviando muestras, emitiendo opiniones o accediendo a funcionalidades según su nivel.

NivelDeUusario: Permite distinguir entre diferentes grados de participación y capacidades de los usuarios dentro del sistema.

Nivel Básico: Es el nivel inicial de todos los usuarios. Los usuarios con este nivel no pueden verificar muestras. Se identifica con el nombre. Este nivel se actualiza automáticamente cuando el usuario cumple los requisitos para convertirse en experto.

Nivel Experto: Los usuarios que alcanzan este nivel adquieren la capacidad de verificar muestras. Para alcanzar este nivel automáticamente, el usuario debe haber enviado al menos 20 muestras y emitido 20 opiniones en los últimos 30 días.

Nivel Investigador: Es un nivel especial que también permite verificar muestras. Se identifica como "Nivel Investigador" y aunque esNivelExperto() devuelve true, a diferencia del nivel experto, este no se obtiene automáticamente sino que debe ser asignado manualmente. Una vez asignado, el usuario mantiene este nivel independientemente de su actividad, ya no puede bajar de categoría.

SistemaDeZonas: La clase SistemaDeZonas implementa la interfaz ISistemaDeZonas y se encarga de administrar todas las zonas de cobertura del sistema. Mantiene un registro de las zonas existentes y ofrece funcionalidades para gestionarlas. Permite agregar y eliminar zonas, obtener la lista completa de zonas registradas, y determinar qué zonas cubren una muestra o ubicación específica. Además, procesa nuevas muestras registrándolas en las zonas correspondientes y maneja las notificaciones cuando una muestra es verificada. Para determinar la cobertura, utiliza cálculos geográficos basados en la ubicación y radio de cada zona.

Ubicacion: La clase Ubicacion representa coordenadas geográficas con latitud y longitud. Proporciona métodos para crear ubicaciones a partir de strings con formato específico y para calcular distancias entre dos puntos usando la fórmula de Haversine, que considera la curvatura terrestre. Las instancias de Ubicacion son inmutables, garantizando la consistencia de los datos de ubicación.

ZonaDeCobertura: La clase ZonaDeCobertura modela áreas geográficas de monitoreo con un epicentro central y un radio de cobertura. Cada zona mantiene registros de las organizaciones suscritas a ella y de las muestras reportadas dentro de su área. Ofrece métodos para gestionar suscripciones de organizaciones, registrar nuevas muestras que caen dentro de su cobertura, y notificar eventos a las organizaciones suscritas. Además, puede identificar otras zonas con las que solapa su área de cobertura. La verificación de si una muestra pertenece a la zona se realiza calculando la distancia entre la ubicación de la muestra y el epicentro de la zona.

Organizacion Impl: Implementa la interfaz Organizacion y representa a una organización real dentro del sistema. Esta clase maneja información básica como el nombre, la ubicación, el tipo de organización (definido mediante el enum TipoOrganizacion) y la cantidad de personas que trabajan en ella.

FiltroPorFecha: Implementa Filtro. Filtra las muestras cuya fecha de creación está dentro de un rango (fechaInicio a fechaFin).

FiltroPorTipoInsecto: Implementa Filtro. Filtra las muestras cuyo resultado coincide con un tipo de insecto (TipoDeOpinion).

FiltroPorUltimaVotacion: Implementa Filtro. Filtra las muestras cuya fecha de última votación está dentro de un rango (fechaInicio a fechaFin). Si la muestra no tiene fecha de última votación, no la incluye.

OperadorOR: Implementa la estrategia de operador lógico OR para filtros compuestos. Evalúa si al menos uno de los filtros dados acepta la muestra.

OperadorAND: Implementa OperadorStrategy. Evalúa si una muestra cumple con todos los filtros de la lista (operador lógico AND).

FiltroCompuesto: Implementa la interfaz Filtro y permite combinar varios filtros simples usando un operador lógico (AND u OR). Usa una lista de filtros y una estrategia de operador para decidir cómo combinar los resultados de los filtros.

OperadorStrategyFactory: Es una clase factory que, dado un OperadorLogico (AND u OR), retorna la instancia correspondiente de OperadorStrategy (OperadorAND o OperadorOR).

FiltroPorNivelVerificacion: Permite filtrar una lista de muestras según su nivel de verificación. Al crear una instancia de esta clase, se le indica si se desean obtener solo las muestras verificadas o solo las no verificadas, mediante un valor booleano. Cuando se utiliza el método filtrar, la clase recorre la lista de muestras y selecciona únicamente aquellas cuyo estado de verificación coincide con el valor especificado al crear el filtro.

FiltroPorZona: Filtra las muestras en función de si se encuentran dentro de una zona de cobertura determinada. Al instanciar esta clase, se le pasa una zona específica, y al aplicar el filtro, se seleccionan sólo aquellas muestras cuya ubicación está dentro del radio de esa zona. Para determinar esto, la clase calcula la distancia entre la ubicación de cada muestra y el epicentro de la zona, incluyendo solo las que se encuentran dentro del área definida. Si ocurre algún error al obtener la ubicación de una muestra, esta se descarta del resultado.

INTERFACES CREADAS

IRepositorioDeMuestras: Define las operaciones que debe cumplir un repositorio de muestras, incluyendo agregar, buscar, obtener por estado y contar.

ISistemaDeZonas: Define operaciones relacionadas con las zonas de cobertura, como agregar/eliminar zonas, determinar cobertura y procesar muestras y validaciones.

ISistemaDeOrganizaciones: Maneja el registro, consulta y suscripción de organizaciones a zonas, incluyendo filtrado por tipo o cercanía y gestión de suscripciones.

Votable: Marca que una entidad tiene un resultado de votación.

INivelDeUsuario: Define el contrato básico que deben implementar los diferentes niveles de usuario en el sistema.

Organizacion: Define el contrato que toda implementación concreta debe cumplir. Esto garantiza que cualquier tipo futuro de organización pueda integrarse fácilmente sin alterar el resto del sistema.

FuncionalidadExterna: Define el contrato para notificar eventos a organizaciones mediante el método nuevoEvento(), que recibe una organización, una zona de cobertura y una muestra relacionada al evento.

Filtro: Define el contrato para todos los filtros. Tiene un método filtrar(List<Muestra> muestras) que retorna una lista de muestras filtradas según algún criterio.

OperadorStrategy: Es una interfaz que define el método evaluar (Muestra muestra, List<Filtro> filtros), usado para implementar la lógica de combinación de filtros (por ejemplo, si todos los filtros deben cumplirse o solo alguno).

ENUMS CREADOS

EstadoMuestra: Representa el estado de la muestra que puede ser NO_VERIFICADA, VERIFICADA.

TipoDeOpinion: Enum con todas las clasificaciones posibles, más NO_DEFINIDO (caso de empate), VINCHUCA_INFESTANS, VINCHUCA_SORDIDA, VINCHUCA_GUASAYANA, CHINCHE_FOLIADA, PHTIA_CHINCHE, NINGUNA, IMAGEN_POCO_CLARA.

TipoOrganizacion: Permite clasificar las organizaciones en categorías predefinidas (SALUD, EDUCATIVA, CULTURAL, ASISTENCIA), facilitando búsquedas y estadísticas.

OperadorLogico: Es un enum que define los operadores lógicos posibles para combinar filtros: AND y OR. Se utiliza para tener un conjunto cerrado de opciones aportando así seguridad de tipo en tiempo de compilación ya que si en lugar de un enum usaramos un String o un int, podríamos equivocarnos al teclear ("And" vs "AND", o un valor fuera de rango) y no te enterarías hasta la ejecución, además facilita la extensibilidad futura.

PATRONES DE DISEÑO IMPLEMENTADOS

Strategy: Se utiliza para para la implementación de operadores lógicos en filtros.

Observer: Se utiliza en el sistema de notificaciones entre zonas de cobertura y organizaciones.

State: Se utiliza para para el comportamiento dinámico de los usuarios según su nivel.

Composite: Se utiliza para para combinar filtros simples en filtros compuestos.

Factory: Se utiliza para la creación de estrategias lógicas (AND/OR) mediante una expresión switch.

Template Method: Se utiliza para la estructura del algoritmo de actualización de nivel de usuario.

Repository: Se utiliza para la abstracción del acceso a datos de muestras y opiniones, este no pertenece al libro "Design Patterns: Elements of Reusable Object-Oriented Software".

Facade: Como el sistema está compuesto por muchas clases y su interacción es compleja, la "Fachada" actúa como una "puerta de entrada" sencilla para que los clientes puedan usar el sistema sin necesidad de conocer o manejar toda la complejidad, en este caso, se utiliza para la Aplicación, este tampoco pertenece al libro "Design Patterns: Elements of Reusable Object-Oriented Software".

Patrón Observer agregado para re-entrega: El patrón Observer se utilizó para implementar una relación de observación entre la clase Muestra y las instancias de ZonaDeCobertura. La clase Muestra actúa como el Sujeto Concreto (Subject) y es responsable de mantener una lista de observadores, los cuales son instancias que implementan la interfaz ObservadorMuestra. Esta clase Muestra también ofrece métodos que permiten agregar o remover observadores, y cuando su estado cambia, se encarga de notificar automáticamente a todos los observadores registrados.

Por otro lado, la interfaz ObservadorMuestra define el contrato que deben seguir todos los observadores. Es decir, cualquier clase que desee observar a una muestra debe implementar esta interfaz y definir el comportamiento que se ejecutará cuando reciba una notificación de cambio.

Una de las clases que implementa esta interfaz es ZonaDeCobertura, que funciona como Observador Concreto. Esta clase reacciona a los cambios en la Muestra de una forma específica, adaptando su comportamiento cada vez que recibe una actualización. Aunque la referencia al sujeto observado (la muestra) no se almacena explícitamente, la relación queda establecida a través del mecanismo de registro en la lista de observadores.

Patrón state agregado para re-entrega: En este diseño se aplicó el patrón State para modelar los distintos comportamientos que puede tener una instancia de la clase Muestra dependiendo de su estado interno. La clase Muestra representa el Contexto y mantiene una referencia a un objeto que implementa la interfaz IEstadoMuestra, el cual representa su estado actual. Toda la lógica que depende del estado es delegada a este objeto, permitiendo que el comportamiento de Muestra varíe dinámicamente sin necesidad de utilizar condicionales.

La interfaz IEstadoMuestra actúa como el Estado Abstracto y define los métodos que deben implementar todos los estados concretos. Esta interfaz encapsula las acciones que varían de acuerdo con el estado en que se encuentra la muestra.

Las clases EstadoAbierto, EstadoExperto y EstadoVerificada son los Estados Concretos que implementan dicha interfaz. Cada una de estas clases define un comportamiento específico según el momento del ciclo de vida de la muestra. Además pueden modificar el estado del contexto, es decir, actualizar el estado de la muestra para reflejar una nueva etapa.

Template method agregado para re-entrega: Se creó una nueva clase abstracta FiltroAbstracto que implementa el patrón Template Method. Esta clase define el algoritmo común de filtrado y delega la lógica específica a las subclases. Todos los filtros existentes fueron refactorizados para heredar de FiltroAbstracto en lugar de implementar directamente Filtro. Esto incluye a FiltroPorFecha, FiltroPorNivelVerificacion, FiltroPorTipoInsecto, FiltroPorUltimaVotacion y FiltroPorZona. Como resultado, se simplificó el código: todos los filtros ahora solo necesitan implementar el método cumpleCriterio(Muestra muestra),y así se eliminó la duplicación de código del método filtrar en cada filtro.

VALIDACIÓN DE CASOS ESPECÍFICOS DEL ENUNCIADO

Se validaron todos los casos específicos solicitados en el enunciado. Se verificó el proceso completo que va desde el envío de una muestra por parte de un usuario con nivel básico hasta su posterior verificación por parte de usuarios expertos, lo cual fue cubierto en el test testEnvioYVerificacionCompletaDeMuestras. También se comprobó el correcto manejo de empates durante la votación de opiniones. Además, se garantizó la nivelación automática de los usuarios según sus niveles de participación en el sistema, cumpliendo las condiciones establecidas. Se validaron las restricciones impuestas por el sistema, como la imposibilidad de opinar sobre muestras propias, emitir opiniones repetidas o intervenir en muestras ya verificadas; todo esto está contemplado en el test testCasosLimiteYExcepciones. Por último, se implementaron y confirmaron búsquedas complejas a través de la combinación de múltiples filtros, lo cual fue verificado mediante el test testCasosFuncionalesAvanzadosDelEnunciado.

CONCLUSIÓN

El sistema implementado cubre la totalidad de los requisitos funcionales especificados en el enunciado. A través de pruebas integrales y unitarias, logramos validar:

- Todas las especies de vinchuca y tipos de opinión.
- La evolución de los usuarios y restricciones de participación.
- El sistema completo de verificación y gestión de muestras.
- Cálculos geográficos y zonas de cobertura con detección de solapamientos.
- El motor de búsquedas con filtros complejos y operadores lógicos.
- El mecanismo de notificaciones automáticas y funcionalidades externas.
- La implementación efectiva de patrones de diseño.
- Una arquitectura limpia, desacoplada y mantenible.

La cobertura de código alcanzada fue del **97,9%**, lo que refleja un alto grado de confiabilidad y robustez en la solución propuesta.