Integrantes

- Jonatan Lezcano
- Leandro Tittarelli
- Ramiro Otermin

Patrones utilizados:

- Composite (Se utiliza en la búsqueda)
- Observer (Sistema de muestras como observable de zona de muestras, zona de muestras como observable de organización)
- State (Usuarios, EstadoDeMuestra)
- Strategy (Filtros de búsqueda)
- Facade (Sistema de muestras)

Sobre el sistema de muestras

Se decidió crear un sistema de muestras, para poder tener un punto de acceso al sistema, el cual nos va a permitir tener las muestras y poder disparar los eventos en las zonas de cobertura. Se podría pensar esta clase como una mezcla entre un patrón facade y un repositorio.

Sobre los usuarios

Los usuarios se crean con un id y un tipo, este último hace las veces de “State” en el cual se delega el algoritmo de re-categorización de usuarios. Por ejemplo, si el tipo de usuario actual es Usuario Basico y en el proceso de recategorización se descubre que cumple los requisitos para ascender a experto, se cambia el tipo de usuario al usuario que disparó el proceso. A su vez, el tipo sirve para responder si es experto, flag que nos va a ayudar a lo largo del proceso de verificación de una muestra.

Sobre las opiniones

Las opiniones se crean con un usuario, solo para guardarse un booleano y así poder responder el mensaje esDeExperto().

Las opiniones poseen un tipo en base a dos enumerativos: Especie y NoVinchuca, los cuales implementan la interfaz Opinable, la cual no tiene protocolo. Esto se hizo con motivo de que el enumerativo Especie se pueda utilizar tanto para registrar una muestra como para opinar, lo cual permite que si el día de mañana surgiera otra especie, no haya que modificar en varias partes del código.

Sobre la verificación de una muestra

Para el estado de una muestra (Votada, Verificada) se decidió implementar un patrón State, el cual nos permite iniciar la muestra como Votada y a posteriormente preguntarle a la muestra si es verificada, cuya respuesta se obtiene delegando en estos objetos que representan el estado de una muestra.

Sobre el ContadorDeOpiniones

A la hora de contar las opiniones para devolver el restultadoActual() de una Muestra, nos parecía que quedaba una implementación muy algorítmica y que “ensuciaba” el modelo de la muestra. Por lo que se decidió reificar ese comportamiento en el objeto ContadorDeOpiniones, el cual se crea con una lista de Opinable(s) y puede responder al mensaje opinionMasVotada().

Sobre el buscador

Para el buscador se decidió implementar un patron Composite donde la clase Filtro hace de rol Componente mientras que FiltroCompuesto de Compuesto, FiltroNivelDeValidacion, FiltroTipoDeEspecie y FiltroFecha (Clase abstracta para FiltroFechaDeCreacion y FiltroFechaDeVotacion) son las Hojas y SistemaDeMuestra es el Cliente

Sobre la combinacion de filtros

Para la combinación de filtros se decidió implementar el patrón Strategy para cada tipo de combinación (AND, OR) dando lugar a la Estrategia (FiltroEstrategia) y sus EstrategiasCompuestas (AndEstrategia, OrEstrategia)

Sobre el filtro de fechas

Para el filtro de fechas se decidio implementar el patrón Strategy para cada tipo de combinación (>, >=, <, <=) dando lugar a la Estrategia (FechaEstrategia) y sus EstrategiasCompuestas (FechaMayorEstrategia, FechaMayorIgualEstrategia, FechaMenorEstrategia, FechaMenorIgualEstrategia)

Sobre Las Funciones Externas

Para implementar funcionalidades externas que luego pueden ser utilizadas por una organización, simplemente hay que implementar la interface FuncionalidadExterna y definir el protocolo mínimo para que pueda responder a un nuevo evento que arribe en la organización.