Trabajo Grupal “A la caza de las vinchucas”

Miembros:

- Insaurralde, Juan Cruz
- Provvidenza, Ivana (vanndere@gmail.com)
- Romero Muñoz, Fernando Mario

Link al repositorio: https://github.com/Juan-Insa/TP-INTEGRADOR-POO2

Decisiones de diseño:

En el siguiente documento explicaremos las decisiones de diseño tomadas al inicio del desarrollo del trabajo, así como también, los patrones utilizados y los detalles que merecen ser destacados.

Nuestro primer paso en el desarrollo del diseño fue realizar un diagrama UML con las clases principales, buscando entender sus responsabilidades así como también sus interacciones con las demás. Durante esta tarea designamos los mensajes que cada clase debía comprender, sus colaboradores internos y los patrones que se podían aplicar sobre algunas de ellas.

A continuación, se detallan aquellas clases que destacan alguna característica particular o forman parte de uno de los patrones aplicados:

Sobre el sistema (*Patrón Singleton*)

El sistema es implementado como un Singleton para asegurar que la clase tenga una única instancia y brindar un punto de acceso global a esa única instancia. Este sistema se encarga de gestionar las muestras, las opiniones, las zonas de cobertura, los usuarios, y darle de alta a estos últimos, para poder realizar búsquedas de muestras utilizando distintos filtros.

Sobre las opiniones y sus clasificaciones

Cada opinión que realiza un usuario es una instancia de la clase homónima. En ella se guarda la clasificación (el valor propiamente dicho de la opinión) así como también quien dió tal opinión y la fecha en la que fue realizada. Dado que hay una cierta cantidad limitada de posibles valores a dar en una clasificación, se decidió crear un enumerativo que los represente. De modo tal que existe un enumerativo con todos los valores válidos para una opinión al cual se le llamó “clasificaciones”.

Sobre las distancias y ubicaciones

Para representar ubicaciones a partir de una latitud y una longitud se creó una clase llamada Ubicación cuya tarea es describir una ubicación válida en el planeta Tierra. Por este motivo y para evitar un exceso de comportamiento por parte de las ubicaciones (pudiendo

así tener una clara *single responsability*), se decidió crear además, una clase CalculadoraDistancias que funcione de manera similar a la clase “Math” de java. El objetivo de esta última clase es realizar operaciones matemáticas utilizando ubicaciones o distancias en km, considerando la forma de la tierra (al ser un esferoide las distancias son curvadas por lo que no basta con triangular los puntos en el plano) a través de métodos estáticos que puedan ser accedidos por objetos que necesiten operar con distancias o instancias de Ubicación.

Sobre las muestras y sus posibles estados (*Patrón State*)

Las muestras tienen un comportamiento particular dependiendo de las personas que ya hayan opinado al respecto de ella. Para poder desacoplar estos comportamientos particulares de la clase se definió utilizar un patrón state sobre el “estado” actual de la muestra y el cual se va cambiando a medida que corresponda que la muestra actúe distinto (por ejemplo que un experto opine en una muestra que no tenía opiniones de expertos previamente). De este modo nos quedan los roles divididos del siguiente modo:

- Muestra: Es el contexto del state.
- EstadoMuestra: Es la clase abstracta state, donde heredan los estados concretos.
- SinOpinionExperto, ConOpinionExperto y Verficiada: Son los distintos estados concretos.

Cabe destacar que en clases se preguntó cómo representar la foto de la muestra y se explicó que bastaba con representarla con algo como un String para no tener que crear una clase vacía sin comportamiento que fuese la foto en si.

Sobre los usuarios y sus posibles categorías (*Patrón State*)

Al igual que con las muestras, los usuarios tienen comportamientos específicos a la hora de opinar dependiendo de si son usuarios básicos (no expertos), expertos o especialistas verificados. Nuevamente para desacoplar esta situación de la clase Usuario en cuestión se utilizó un patrón State, donde la categoría del usuario sea su estado actual. De este modo quedan los roles divididos del siguiente modo:

- Usuario: Es el contexto del state.
- Categoría: Es la clase abstracta state, donde heredan los estados concretos.
- Básico, Experto y Especialista: Son los distintos estados concretos.

Sobre las organizaciones (*Patrón Observer*)

Las organizaciones dependen de ser notificadas por las zonas de cobertura a la que estén suscritas para poder realizar eventos correspondientes a una muestra determinada (si es verificada o no). Este comportamiento nos llevó a implementar un observer complex que en vez de utilizar un String como código de notificación, simplemente consulta el estado de verificación de la muestra enviada por parámetro para definir su comportamiento en el update. Los roles quedan divididos del siguiente modo:

- Organización: Es el concrete observer.
- OrganizaciónObserver: Es la interfaz observer.
- ZonaDeCobertura: Es el subject del observador.
- ZonaDeCoberturaSubject: Es el concretSubject.

Sobre las zonas de cobertura (*Patrón Observer*)
Listener con parte de complex

Las zonas de cobertura deben ser notificadas cuando se agregan zonas nuevas o muestras que se encuentran dentro de su alcance por parte del sistema. Por este motivo se decidió usar un observer listener que entienda los mensajes updateMuestra y updateZona. Los roles quedan divididos del siguiente modo:

- ZonaDeCobertura: Es el concreteObserver.
- ZonaDeCoberturaObserver: Es la interfaz observer.
- Sistema: El sistema es el concreteSubject del observador.

Sobre las búsquedas de muestras en el sistema (*Patrón Composite*)

Como se requiere realizar búsquedas de muestras basadas en filtros se incorporó al modelo el patrón Composite para poder realizar filtros complejos combinando los distintos criterios pedidos, los distintos criterios de operaciones de comparación, y las operaciones lógicas OR y AND. Los roles quedan divididos del siguiente modo:

- BusquedaComponent es la interfaz Component.
- ANDComposite, ORComposite: son los objetos compuestos Composite.
- FechaCreacionMuestra, FechaUltimaVotacion, NivelValidacion, TipoInsecto: son las hojas u objetos simples.

Sobre las búsquedas de muestras en el sistema (*Patrón Strategy*)

Para poder realizar operaciones de comparación entre fechas (FechaCreacionMuestra y FechaUltimaVotacion) se decidió usar el patrón Strategy para aplicar criterios lógicos específicos. Así cada subclase de la clase Criterio define un comportamiento dependiendo de si la comparación va a ser: mayor, mayor e igual, menor, menor e igual, igual.