Trabajo grupal integrador

Alumnos: - Axel Agüero
- Carlos Saldaña
- Selene Madrid

Repositorio: https://gitlab.com/carlosjoel95/tp-integrador-vinchucas-ag-ero-madrid-salda-a

Uml: el uml decidimos hacerlo en la herramienta UMlet porque se nos hizo más cómodo para manejar. Sin embargo, esto dificultaba un poco las modificaciones al uml, ya que teníamos que estar pasándonos el archivo con la última edición disponible; ya que no tiene una forma de hacerlo “online” y que cada uno pueda editar ese UML. Sin embargo, se nos hizo más fácil la escritura del mismo en este programa.

Librerías utilizadas: decidimos utilizar mockito-all-2.0.2-beta.jar .

Diseño, implementación y decisiones: En cuánto al diseño al decidimos hacer una clase SitioWeb que representaría la aplicación, la cuál posee las muestras, participantes y organizaciones.

Cada participante debe registrarse al **SitioWeb** para poder enviar. También decidimos que cada participante conozca las muestra que envío tanto como las que verificó.

A la hora de implementar las muestras, más bien dicho cuando un participante la envíaba la muestra su opinión cuenta a la hora de pedir qué tipo de insecto que es detectado en muestra por mayoría de voto. Pero esto no quiere decir que su opinión cuenta para que dicha muestra **esté verificada**. Esto nos trajo mucha confusión al principio ya que de primeras no habíamos tomado en cuenta su opinión, después la tomamos en cuenta pero en forma de voto pero no debería ser así hasta que pudimos implementar sin que sea implementada ó una ó la otra. (Ó al menos es lo que interpretamos nosotros del enunciado, esta parte es media contradictoria por un lado dice que no puede “votar” su propia muestra pero su “opinion” de la muestra debe ser contada).

El nivel de conocimiento al principio habíamos decidido que se dividía en dos (**ConocimientoBasico** y **ConocimientoExperto**) pero la consigna decía que también pueden existir expertos de forma externa, ya sea por estudios o títulos que tuviesen y a ellos lo habíamos identificado con un boolean pero después decidimos que sea una clase más llamada **ConocimientoEspecialista**. En este caso el conocimiento especialista, logra las mismas acciones y/o derechos que tenga un participante de **ConocimientoExperto**, pero no puede volver a ser un participante de **ConocimientoBasico** aunque no cumpla los requisitos mensuales.

Al momento de los test nos dimos cuenta que una persona con **ConocimientoBasico** cuando cumplia las metas para ser **ConocimientoExperto** , provocaba que todas las

votaciones suyas que hayan sido como **ConocimientoBasico.** Se modificaban a **ConocimientoExperto,** lo cual provocaba una error en las verificaciones que quieran ser enviadas por otros usuarios. Esto lo arreglamos guardando el nivel con el que se votó al momento.

En participante con los distintos niveles de conocimiento se aplica el **patrón strategy.**

En los filtros de búsqueda al tener 4 tipo de búsqueda (por fecha de creación de muestra, fecha de última votación, tipo de insecto detectado en la muestra y nivel de verificación) había que poder combinarlos con los operadores OR y AND.
Para ello decidimos hacer una clase abstracta llamada **FiltroCompuesto** que tenga cómo subclases a **FiltroCompuestoOR** y **FiltroCompuestoAND,** para así con ellas poder combinar los otros 4 tipo de búsqueda y hasta ellas mismas.
Lo dicho anteriormente aplica el **patrón composite**.

Cabe recalcar que al principio tuvimos unos problemas de entendimientos de algunos de los filtros de búsqueda como el de " fecha de la última votación" ya que no sabíamos bien a qué se refería. Los ejemplos eran medio confusos.
Con respecto a los filtros decidimos separarlos en dos package distintos, para no perdernos cuando los buscamos y queda en claro cuales son implementaciones ajenas a los filtros. La misma separación está hecha en los test.

La **Funcionalidad Externa** Lleva al nuevoEvento que deberia ser usado por cada organizacion en el momento que sea implementado (en este tp **no** se pedía que le creemos algun evento), asi que queda en vacio. Para los test utilizamos el Spy del Mockito y solo verificamos que se “lance” el mensaje ya que no ibamos a ejecutar nada.

Las fechas en **Muestra** son 2, la fechaObtencion y la fechaCreacion. La fechaObtencion se refiere a cuando un usuario consiguió los datos para la muestra en sí y la fechaCreacion es cuando es creada la muestra.

Coverage: Tenemos un test de unidad de un **99,5%** de cobertura en la TOTALIDAD del proyecto y 97,8% en El SRC (source) del proyecto.