Universidad Nacional de Quilmes

Programación con Objetos II

Trabajo Práctico

1° y 2° Hito

Integrantes:

Cicovich, Esteban

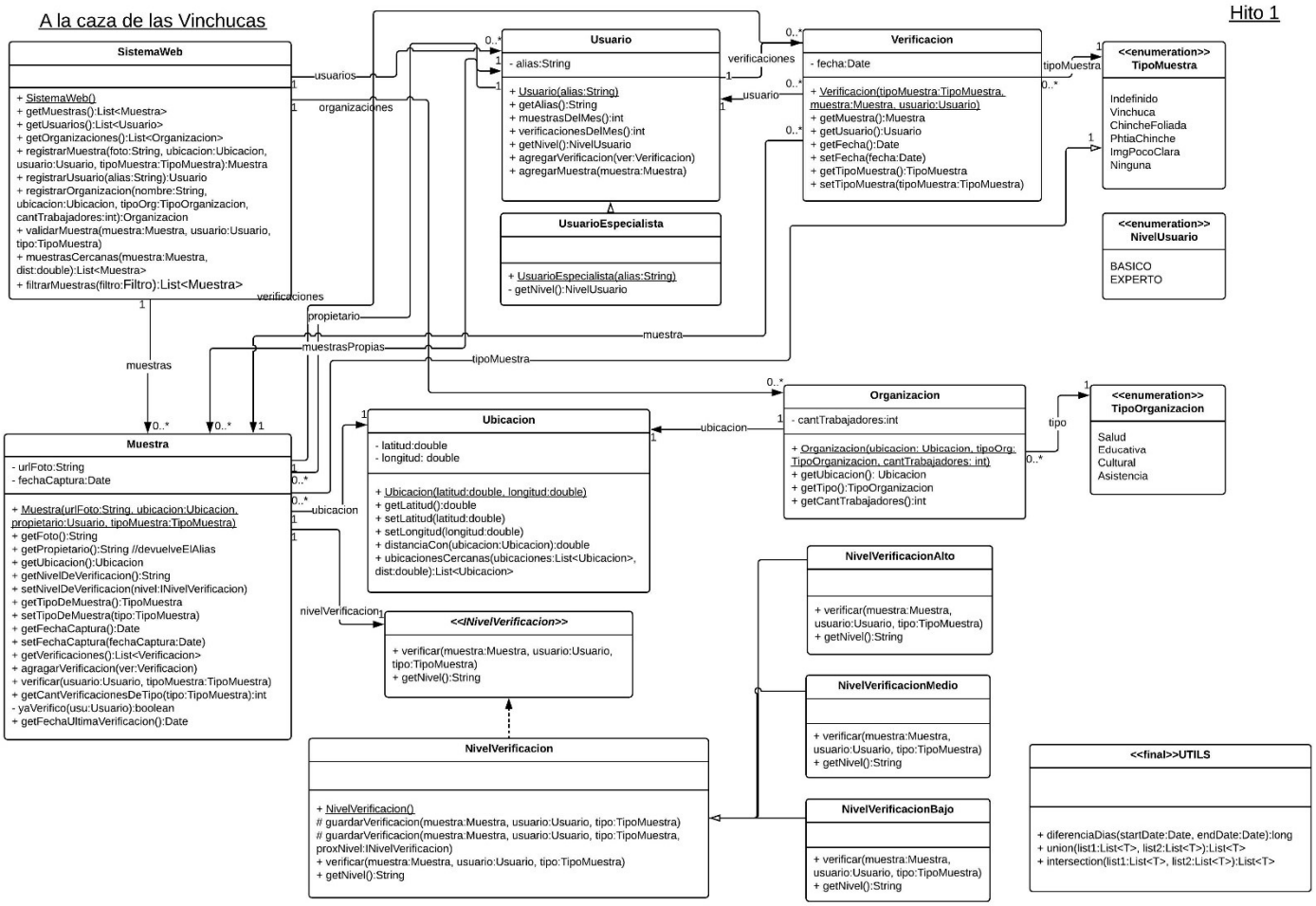
Di Carlo, Leonardo

Mur, Lucas

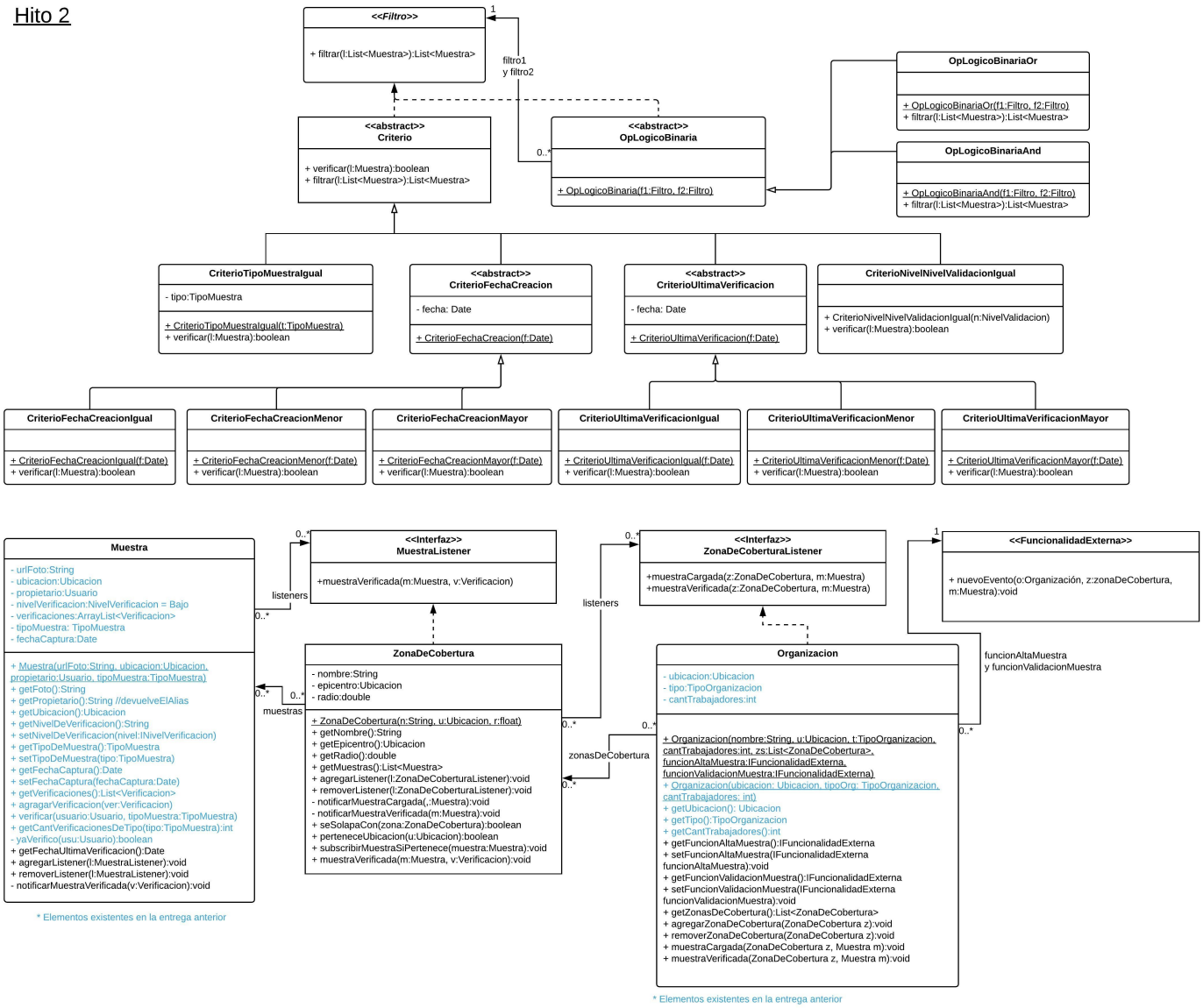
“A la caza de las Vinchucas”

A fin de hacer más clara la resolución que tomamos para este trabajo, utilizamos un proyecto compartido de UML desde la web www.lucidchart.com .

El mismo, nos permitió ir diagramando las diferentes relaciones entre los objetos, sus herencias y su jerarquía. A continuación, adjuntamos dos diseños UML: El primero es la estructura con la que realizamos el Primer Hito, y el segundo muestra la resolución para la extensión solicitada en el Segundo Hito.



Hito 2



Patrones de Diseño

1° Hito

En cuanto a los patrones de diseño implementados, cabe destacar el uso del Patrón “State” para la implementación del Nivel de Verificación de cada muestra. Dicho Nivel, puede ir cambiando de estado entre los valores Bajo, Medio y Alto. El cambio de estado está relacionado a la cantidad y tipo de usuarios que van validando la muestra.

A su vez, el nivel de verificación también aplica el Patrón “Template Method”, ya que posee una clase abstracta en la cual cada estado particular redefine sus procedimientos.

El resto de la estructura del trabajo no sigue un patrón en particular, pero fue siguiendo cierta lógica que deriva del enunciado establecido.

2° Hito

Para la segunda parte del proyecto, el problema general se resolvía implementando un patrón Observer. Dicho patrón, tenía la dificultad de involucrar no sólo a dos objetos, sino a tres: Organización - Zona de Cobertura – Muestra.

En rigor, se trataba de dos patrones Observer, ya que las Zonas de Cobertura se suscribían a las muestras, y a su vez las Organizaciones se suscribían a las Zonas de Cobertura.

Por otra parte, para la resolución de lo que respecta a los Filtros se utilizó un patrón Composite para hacer recursivo el hecho de que un filtro podía a la vez tener una concatenación de filtros adentro. Para los Filtros sólo se utilizaron operadores binarios: AND y OR.

Relación entre Objetos

1° Hito

El Sistema Web conoce a casi todos los demás objetos, uno o varios de cada uno. Más precisamente, este sistema conoce a una o muchas muestras, uno o muchos usuarios y una o muchas organizaciones.

A su vez, la Muestra conoce a un nivel de verificación, el cual hace las veces de “Estado” de esta muestra. También la muestra conoce al usuario que la cargó, y la ubicación desde donde la cargó.

Por el lado del Usuario, éste conoce las muestras que él cargó y las verificaciones que hizo. Existe un “Usuario Especialista” que hereda de este usuario base, que nace y muere con esta condición, la cual genera que cualquier muestra que verifique este usuario se vuelva con un nivel de verificación alto inmediatamente.

Siguiendo este esquema, la Verificación conoce a la muestra que está siendo verificada, al usuario que la está verificando, y a los tipos de muestra existentes.

Finalmente, la Organización conoce a una ubicación que le será propia, y a los distintos tipos de Organización existentes (Salud, Educativa, Cultura, etc).

2° Hito

Para el segundo tramo del trabajo, se implementaron interfaces para la Organización (con una interfaz ZonaDeCoberturaListener que utilizan los observadores de cada Zona), para los diferentes Filtros (con la interfaz Filtro, que obliga a los diferentes criterios/operadores a implementar el filtrado en las listas de muestras), y para las Zonas de Cobertura (con la interfaz MuestraListener que utilizan los observadores de cada Muestra).

Para el caso de los filtros, se crearon dos clases abstractas (Criterio y OpLogicoBinaria) que derivan en sus subclases la implementación ya sea de un Criterio, como de un Operador. Cabe destacar que la Clase abstracta OpLogicoBinaria es quien permite realizar la recursión de sumar diferentes filtros.

Casos de Prueba

1° Hito

Al momento de crear los tests, se intentó utilizar TDD para la clase que más conoce a las demás (Sistema Web), ya que dicha clase es quien implementará el uso de todos los demás participantes.

En el resto de los tests, comenzamos por los que poseían menor cantidad de dependencias, con el objetivo de cubrir el testeo de la mayor parte del código posible. Nos restaría implementar el Plugin ECL EMMA, para de esta forma asegurar el mayor porcentaje posible de utilización de testeo código de una forma concreta.

Se utilizaron los estilos de test Double clásicos de “Mockito”, tales como “Verify”, “Mock”, “Dummy” y “Stub”.

2° Hito

En esta segunda entrega, se realizaron Test para probar la funcionalidad de los observadores en todo su proceso: desde que se suscriben los observadores al observado, pasando por el momento en que se notifique correctamente a los observadores, hasta llegar a que el observador implemente lo que desee una vez notificado (Para esto ya venía implementada una "Funcionalidad Externa").

Código JAVA

Se adjuntan los archivos "source" y "java" con el código implementado de resolución, dentro de un .ZIP junto con el presente informe.

Observaciones

La resolución de este trabajo intenta cubrir lo solicitado en el enunciado, sin embargo hemos tomado algunas suposiciones como estándar:

- Cuando la Muestra toma un nivel de verificación "Alto", no acepta más validaciones.
- Los Usuarios pueden tener un nivel "Básico" o "Experto", pero existe una excepción: Hay un tipo de usuario ("Especialista") que nace y muere con esta condición, y al verificar una muestra la eleva a un nivel de verificación "Alto" inmediatamente.
- Las dependencias del proyecto fueron implementadas con Maven 3.7.0 . Por su parte, la versión de Mockito utilizada es 2.7.9 . Por último se utilizó JUnit 4 para la creación de tests.