PROGRAMACIÓN CON OBJETOS II

TRABAJO FINAL: “A la caza de vinchucas”

Alumnos:

- De Maio, Julian Daniel
- Acosta, Federico Antonio
- Beltrame, Franco

Emails:

julidema82@gmail.com
federicoacosta002@gmail.com
beltramefranco97@gmail.com

Fecha: 15/06/2023

Introducción:

A continuación detallamos todas las decisiones de diseño, detalles de implementación, patrones de diseño que elegimos a la hora de realizar el trabajo propuesto en la materia Programación con Objetos II, como así también las dificultades que atravesamos como grupo y conclusiones que sacamos en el proceso y al finalizarlo.

Decisiones de diseño

Entre las decisiones de diseño tomadas en este sistema, se destacan:

-El diseño de los estados del usuario utilizando un Patron State. Esto se llevó a cabo debido a que se encontraron distintos comportamientos para los Usuarios, dependiendo de los Estados. Se deben realizar distintas verificaciones a la hora de recibir una opinión por parte de un Usuario hacia una Muestra.

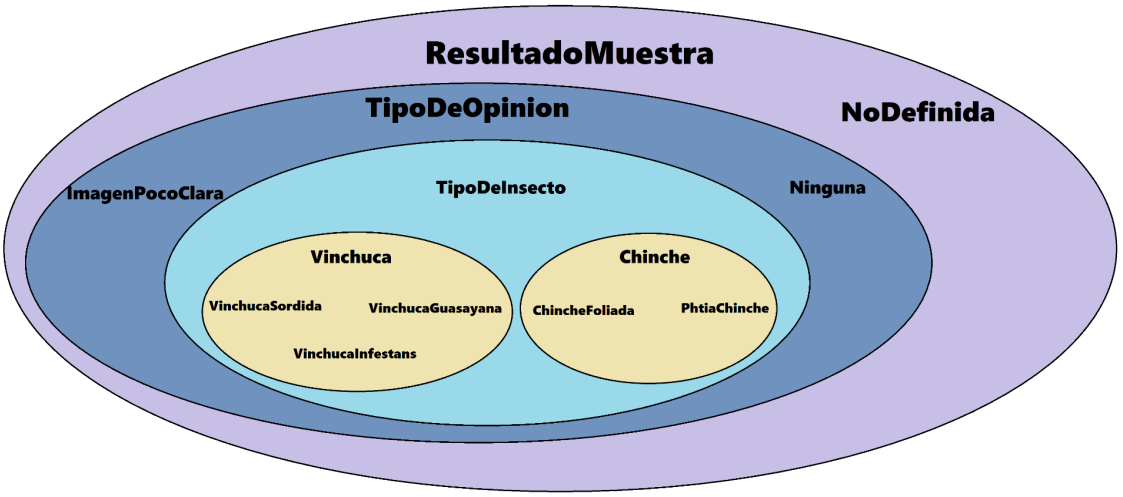
-El diseño de los Filtros de Muestra utilizando un Patrón Composite. Esto se llevó a cabo debido a que se requiere la combinación y/o anidación entre distintos tipos de filtros, con operadores lógicos (Or, And). En adición, se requiere no tener un orden en los filtros especificables. Esto se resolvía mediante el Patrón Composite, que comprende a cualquier tipo de búsqueda (And, Or, Filtro individual) como un mismo objeto.

-El diseño de las zonas de cobertura y organización utilizando un Patrón Observer. Esto se llevó a cabo debido a que se encontraron varios mensajes específicos (muestraSubida, muestraValidada) que debían ser emitidos por un Observable y ser implementados por los Listeners específicos.

-El diseño de los estados de muestra utilizando un Patrón State. Esto se realizó de esta manera, con motivo de tener distintas implementaciones para calcular el resultado de cada muestra, dependiendo del estado de la misma. Por ende, se realizaron 4 estados distintos, en los que cada uno calcula de manera distinta el resultado de la muestra en cuestión. En adición a esto, se requirió la implementación de un double-dispatch entre los estados de usuario y los propios estados de muestra, para eliminar condicionales y obtener la categoría del usuario de forma más fiable y legible. A partir de este double-dispatch se obtiene que tipo de verificación se debe realizar (usuario básico o experto).

-La implementación de un Template Method en los mensajes de filtrado de los filtros. Esto se llevó a cabo debido a que el comportamiento para todos los filtros individuales era el mismo, con una mínima diferencia en los criterios de filtrado. Cada criterio depende de cada filtro, por ende se especificó una Primitive Operation (pasaElFiltro(Muestra)) que era definido por cada filtro.

-La jerarquía de enumerativos correspondientes a los tipos de opinion y resultados de muestra. Esto se realizó implementando distintas interfaces para poder tipar y generar una jerarquía entre los enumerativos.
Vinchuca, Chinche, TipoDeInsecto, TipoDeOpinion, ResultadoMuestra



Se tomó esta decisión de diseño, debido a que se encontraron muchas similitudes entre los distintos enumerativos y había incompatibilidad entre los tipos de los mismos. Por ejemplo, un TipoDeOpinion no se podía convertir a un ResultadoMuestra. Por lo que era necesario crear un tipado que engloba a los mismos. Además, se evitó una leve duplicación de código.

Detalles de Implementación

Entre los detalles de implementación, hay varias deudas técnicas que quedaron pendientes por falta de tiempo y/o falta de una solución eficiente.

1. Mensaje verificarMuestra(Muestra) en EstadoMuestraOpinadaPorExpertos. El mismo realiza una verificación del estado que debería corresponder (a través de un condicional), lo cual no es adecuado según el Patrón State.

2. El mensaje obtenerTipoDeOpinionMayoritaria() tiene una implementación bastante compleja, debería ser refactorizada en distintos mensajes que vayan realizando las tareas de a una. Tener un método tan largo es confuso, poco legible y poco comunicativo.

3. No pudimos cubrir los tests que verificaban excepciones en nuestra implementación debido a que las mismas cortan la ejecución del programa sin dejar actuar al test, según

lo que investigamos. Probamos varias soluciones pero en ninguna pudimos solucionarlo, la pérdida de cobertura total en el trabajo debido a este inconveniente fue de 1,1%.

4. En la clase ubicación, para representar el kilometraje, investigamos formas de importar algún tipo de clase de unidades de mediciones que lo represente, pero nos pareció bastante engorroso y decidimos representarlo con un double (parte entera kilómetros; parte decimal, metros). La englobamos en el listado de deudas técnicas, ya que alguna clase de unidades de medición sería interesante y aportaría mucha más legibilidad a la clase ubicación.

5. En la aplicación del patrón Template Method sobre la clase Filtro no pudimos colocar la modificación final al template method (filtrarMuestras(Set<Muestra>) propiamente dicho ya que daba error al momento de testearlo.

Patrones de Diseños

Los Patrones de Diseño utilizados en el proyecto fueron los siguientes:

State (EstadoUsuario):
Context -> Usuario
State -> EstadoUsuario
ConcreteStateA -> EstadoUsuarioBasico
ConcreteStateB -> EstadoUsuarioExperto

State (EstadoMuestra):
Context -> Usuario
State-> IEstadoMuestra
ConcreteStateA -> EstadoMuestraVotadaPorBasicos
ConcreteStateB -> EstadoMuestraVotadaPorUnExperto
ConcreteStateC -> EstadoMuestraVotadaPorExpertos
ConcreteStateD -> EstadoMuestraVerificada

Composite:
El Patrón de Diseño Composite, fue implementado para la filtración de muestras, por parte de la AppWeb.
Client -> AppWeb

Component -> IBusqueda
Composite 1 -> And
Composite 2 -> Or
Leaf -> Filtro

Observer:
El Patrón de Diseño Observer, fue implementado para la notificación que querían recibir las organizaciones, por parte de las zonas de cobertura.
Observable/Subject -> Subject
Observer -> IZonaDeCoberturaListener
ConcreteObserver -> Organizacion
ConcreteObservable/Subject -> ZonaDeCobertura

Template Method:
El Patrón de Diseño Template Method fue implementado en el Patrón Composite, de los filtros.
Template Method -> filtrarMuestras(Set<Muestra>)
Primitive Operation -> pasaElFiltro(Muestra).

Singleton:
El Patron de Diseño Singleton fue implementado en la clase AppWeb, para asegurarnos de tener una única instancia de la misma en todo el sistema.
Singleton -> AppWeb

Dificultades

Entre las dificultades afrontadas en este trabajo, se destacan:

1. El planteamiento y diseño del Patrón State para los estados de Usuario, ya que en un primer instante se intuyó que sería un Patrón Strategy, debido a que contaba con las categorías que eran una característica de cada Usuario que cambiaría de forma dinámica. Sin embargo, se descartó esta idea debido a que no se encontraba suficiente justificación para aplicarla debido a que los States no tenían mensajes que contestaran

de diferentes maneras. Fue decisiva la elección del State cuando nos dimos cuenta que los distintos estados se conocían entre sí y eran intercambiables.

2. La implementación de la Muestra, fue uno de los mayores problemas debido a que es una clase muy grande y con bastante comportamiento, que está involucrada en muchos aspectos del sistema, desde los filtros hasta las zonas de cobertura. Se encontraron diversas dificultades a la hora de delegar responsabilidades e implementar métodos complejos que involucran el uso de Streams.

3. El planteamiento y diseño de los Estados de Muestra, involucrando el Double-dispatch entre los estados de muestra y las categorías del usuario, que realiza la gestión para la posterior verificación de los resultados de cada Muestra.

4. La realización del UML, se realizaron bastantes modificaciones mientras se iba implementando el sistema, debido a los diversos refactors que se iban realizando. El UML prácticamente no se finalizó hasta el último instante.

5. El testeo de las excepciones del que ya hablamos previamente en detalles de la implementación. Probamos varias alternativas de testeo pero ninguna nos resultó eficaz.

Conclusiones

Como conclusión, pensamos que este trabajo abarcó en su totalidad los temas vistos en la materia. Principalmente los conceptos orientados al diseño de sistemas, aplicando Patrones de Diseño (Observer, State, Composite, Template Method) y con foco en el Refactoring y las buenas prácticas (escalabilidad, mantenibilidad, etc) utilizando el UML como herramienta de diseño.

Uno de los conceptos que se aplicaron en mayor medida, fue el Patrón de Diseño State, tanto para la representación de las categorías del Usuario y de la Muestra.

También tuvimos en cuenta la aplicación de los principios SOLID a la hora del diseño del trabajo. Consideramos que nuestro diseño es comprensible, escalable, flexible y resistente a nuevos cambios gracias a la aplicación de los mismos.

El trabajo complementa todos los conceptos vistos en la materia, ya que a pesar de tener conocimiento de ellos, nos pareció muy interesante aplicarlos todos a un sistema grande y con un dominio con mucho para charlar como equipo, con grandes decisiones de diseño y dificultades con las que tuvimos que lidiar y pudimos atravesar.

Nos encantó trabajar en equipo, creemos que fue un gran desafío del que nos llevamos un gran aprendizaje como grupo.