TRABAJO PRÁCTICO

Programación orientada a objetos 2

Leandro Antúnez
Fernando Castro
Darío Gutiérrez

Viernes 1 de julio, 2016
Primera comisión

Tabla de contenidos:

1. Mails de los integrantes:

- Leandro Antúnez: leandro_antunez@outlook.com
- Fernando Castro: fernando.castro.unq@gmail.com
- Darío Gutiérrez: dariofg93@gmail.com

2. Problemáticas y soluciones:

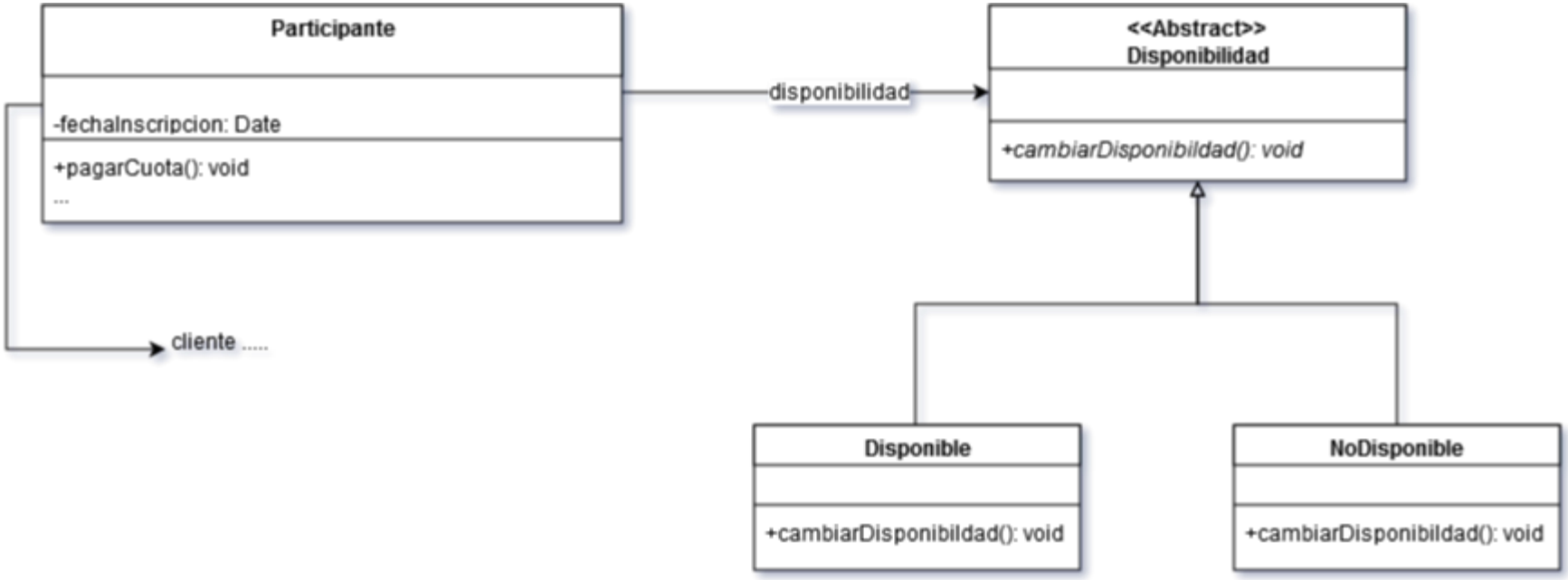
A lo largo del TP se discutieron varias alternativas para llevarlo a cabo, sobre todo en el momento del diseñar un diagrama UML final.

Entraron en discusión varias clases que no forman parte del material entregado, como por ejemplo un AdministradorDeConcesionaria, cuyo propósito era llevar a cabo las tareas que la Concesionaria tiene, pero fue descartada rápidamente al ver que no era necesario agregar una "capa" más de implementación si no se ganaba nada (creímos que era sobre diseñar el TP), también tenía responsabilidades que quizás un administrador no debería tener.

Los patrones de diseño que se fueron proponiendo tuvieron el siguiente ciclo, en la mayoría de los casos:
-Aparecer como propuesta: Alguien pensaba que en tal lugar sería buena idea un patrón de diseño y se lo comunicaba a los otros integrantes.

-Implementación en java (o pensado en seudocódigo): Se implementaba o pensaba la alternativa para sacar conclusiones y ventajas y desventajas de la misma.
-Desaparecer: En base a las conclusiones conseguidas del paso anterior, quitábamos el patrón del diagrama UML y por consiguiente de la implementación.

Un claro ejemplo de este ciclo ocurrió al querer modelar la disponibilidad de los participantes del plan de ahorro en cuestión, se propuso que la disponibilidad sea considerada un state. Al suscribir un cliente a un plan, éste tendría un estado Disponible y si es adjudicado pasaría a NoDisponible, pero jamás podría pasar a un estado Disponible de nuevo porque se considera que una vez que el participante paga todas las cuotas y es adjudicado desaparece del plan de ahorro.



Alternativa a la problemática de la disposición del participante en el plan de ahorro.

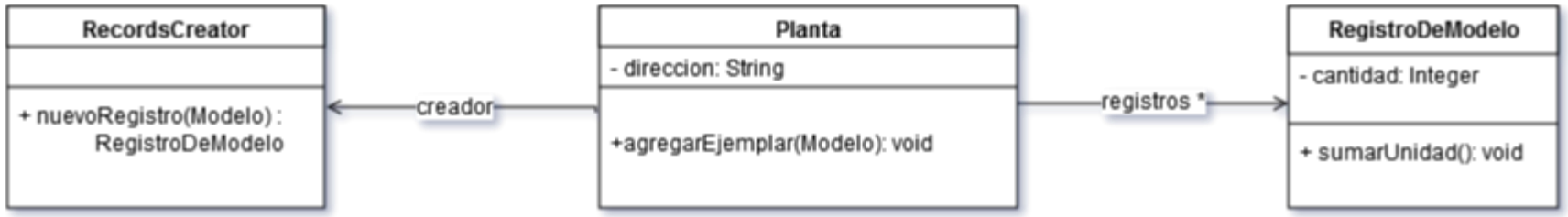
Algunos casos si pasaron la revisión del segundo caso, como fue el patrón de diseño strategy que sí se quedó en el Financiamiento.

3. Los principios SOLID:

El diagrama UML se pensó en base a los principios SOLID:

Al momento de crear un nuevo modelo en la Planta se tiene en cuenta el primer principio, Single Responsibility Principle, y entra en consideración una nueva clase encargada de crear los registros de los modelos (RecordsCreator).

En el mismo ejemplo también se tiene en cuenta el segundo principio Open Closed Principle, ya que el código propuesto en agregarUnidad(Modelo) de Planta no se debe modificar ante un eventual cambio de los registros de modelos.
Otro claro ejemplo del segundo principio es el modelar el equipamiento de un modelo como una nueva jerarquía de clases, en vez de solo tratarlo como un String.



El tercer principio, Liskov Substitution Principle, es fácil de verlo, se usa al momento de crear un modelo, cuando se le designa un Equipamiento puede ser de Full o Base. Este principio está fuertemente ligado al concepto de herencia y es lo que se utiliza para poder llevarlo a cabo, vale aclarar que gracias a éste principio ganamos más abstracción y polimorfismo en el diseño, además también se tiene en cuenta el segundo principio.

El cuarto principio, Interface Segregation Principle, está presente en el modo de adjudicación de un plan, ya que MayorCobertura y Sorteo implementan una interface que tiene una única responsabilidad, el mensaje elegirGanador(PlanDeAhorro). De haber algún nuevo mensaje que tenga una responsabilidad posiblemente haya una nueva interface u otra clase para llevarla a cabo, todo depende de lo que se necesite.

El quinto principio, Dependency Inversion, se respeta en todos los sentidos donde podría haber una eventual extensión del diseño entregado, por ejemplo al diseñar el Financiamiento se tiene en cuenta que pueden haber otros financiamientos además de los ya propuestos que son Plan100 y Plan70Y30, no se sabe con cual clase concreta interactúa, en este caso, el PlanDeAhorro, logrando un desacoplamiento de las clases.

Hay muchos otros ejemplos en donde se siguieron los principios SOLID pero basto con estos para representar cada uno de los principios.

4. Conclusiones:

El trabajo práctico fue una muy buena propuesta para terminar de "absorber" todos los conocimientos que se adquieren en la materia. Incluso quienes recursan la materia aprenden mucho al volver a hacer un diseño UML y discutirlo con otros integrantes de su grupo.

Aprendimos a usar una poderosa herramienta de trabajo grupal Git, o más precisamente eGit en este caso, la cual ahora conocemos un poco más sobre los riesgos y ventajas de usarla.

En cuanto a los patrones de diseños, no hay tantos en el UML dado como se hubiese querido en un principio.

Por ultimo las adiciones en Java SE 8 de las Expresiones Lambda y la API Stream nos dio una excelente manera de pasar de un enfoque imperativo a un enfoque funcional cuando es necesario, lo que nos permitió, por ejemplo, modificar listas de Participante, PlanesDeAhorro, etc… Aunque para poder usarla en su máximo potencial es necesario sentarse y practicar!