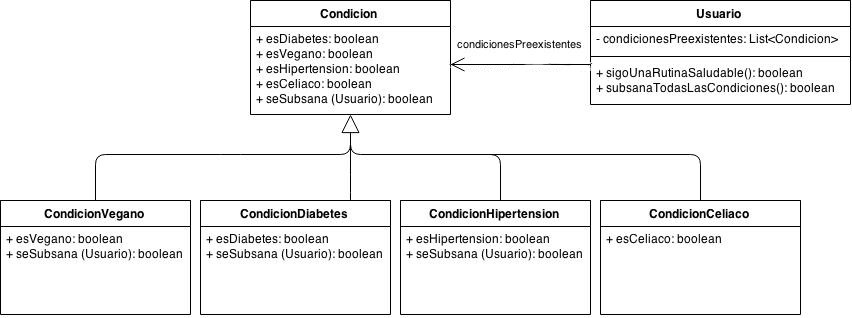
**Documentación – TP “Qué Comemos Hoy?”**

**Alumnos:** Cantarini Juan - Gagliardo Miguel - Perez Gabriel - Toja Daniel

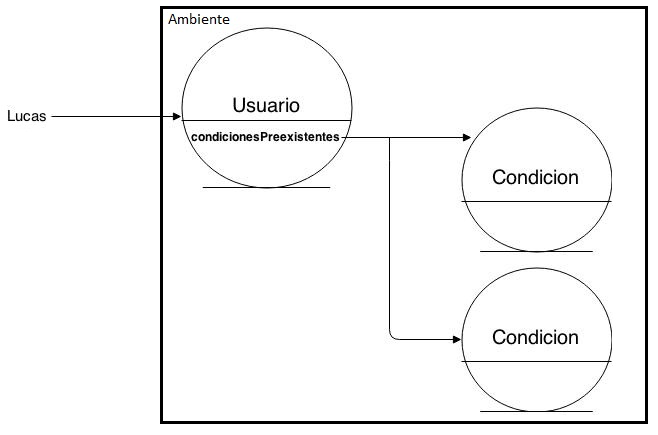
**Grupo:** 3

**Diagrama de Clases para Condición(Vegano, Diabetes, Hipertensión, Celíaco).**

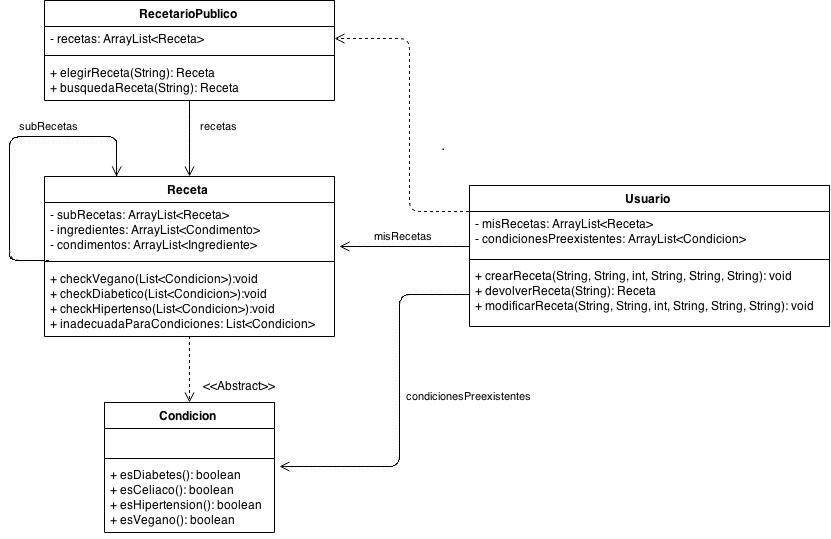


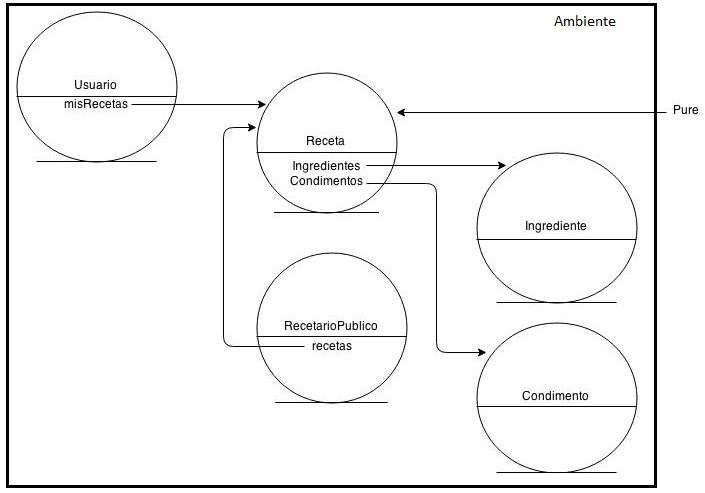
Decidimos implementar este caso utilizando como base el patrón de diseño **Strategy.** La razón por la cual decidimos hacer esto es que se cuenta con una Clase Abstracta (Condición) y 4 clases que van a implementar algunos de sus métodos (CondicionVegano, CondiciónDiabetes, CondicionHipertension y CondicionCeliaco). Como se puede ver, se factoriza el comportamiento común de las subclases (método seSubsana) en una única superclase.

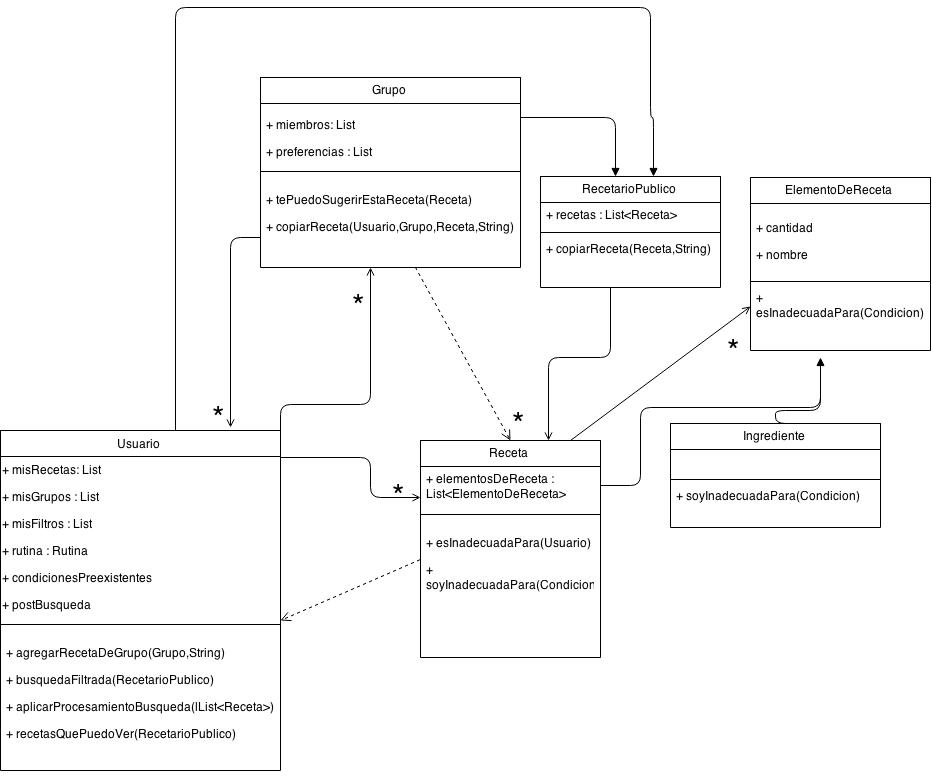
A su vez, la clase Usuario “conoce” a la clase Condición a través de una variable de instancia, la colección **condicionesPreexistentes**.



**Diagrama de clases para Receta.**







En estos dos diagramas de clase, se decidió dividir las comidas en dos tipos distintos para facilitar luego la elección de recetas según sean aptas o no para las condiciones actuales incluidas en el programa.

**Entrega Número 2.**

**Strategy.**

Diagrama de clases Filtros.

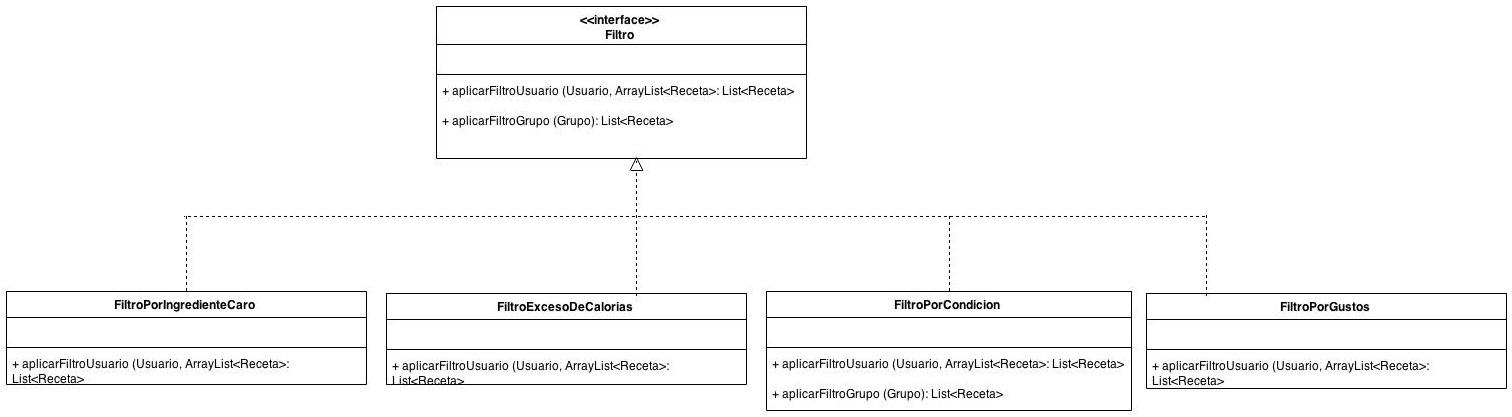
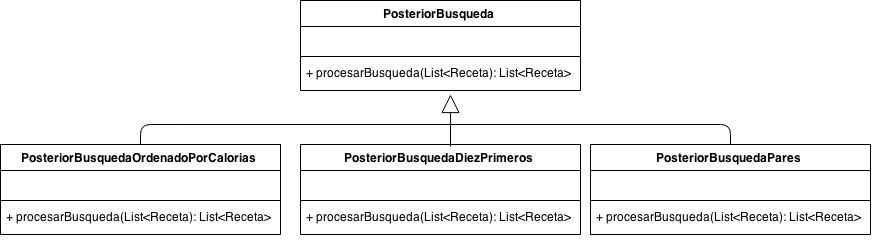
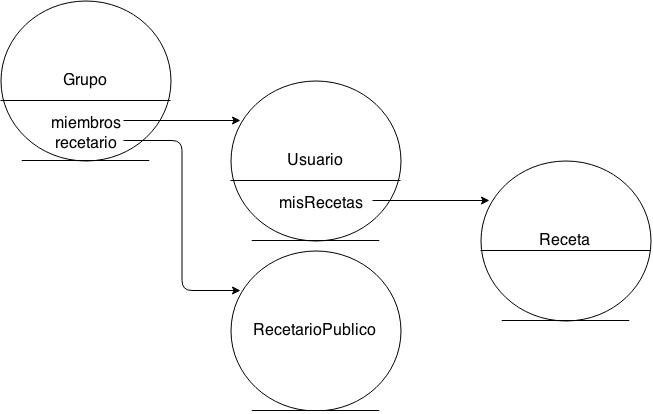


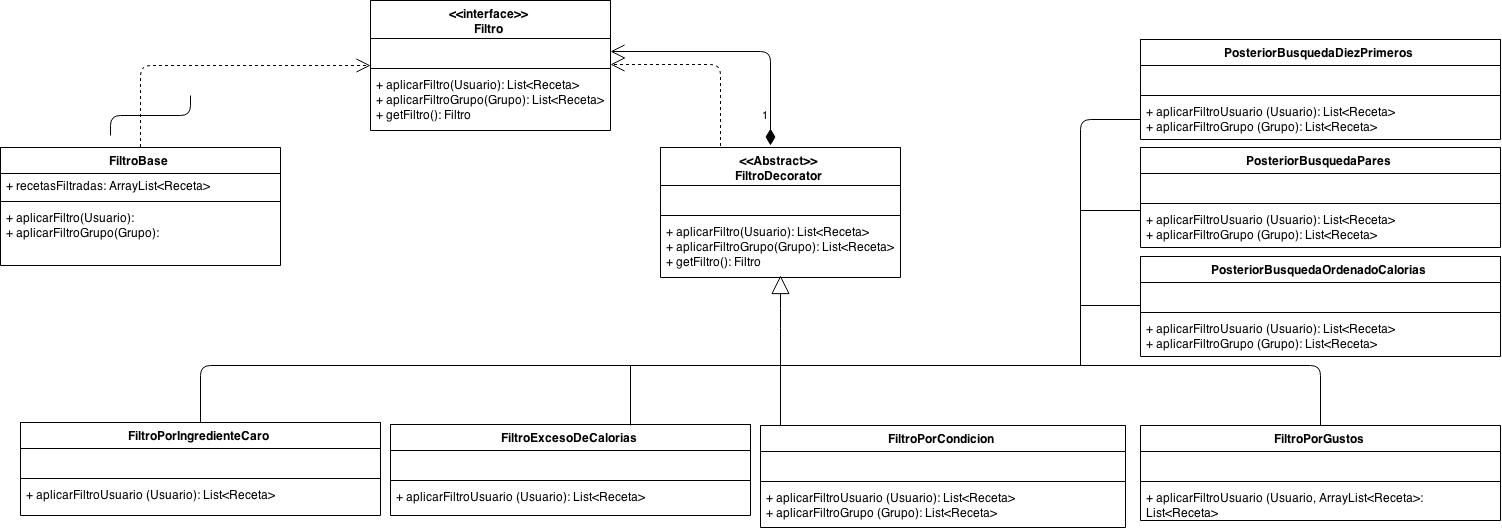
Diagrama de clases de Busquedas Posteriores.





**Decorator**

Diagrama de clases Filtros y Busquedas Posteriores.



|  |  |  |
| --- | --- | --- |
|  | Strategy | Decorator |
| Facilidad para agregar nuevos criterios de filtro | Más dificil que Decorator.  Crear una nueva clase e implementar el mensaje. | Más fácil que Strategy.  Se agrega la nueva clase y se le ingresa comportamiento deseado al método. |
| Posibilidad de eliminar filtros dinámicamente | Más fácil, se elimina el objeto no deseado. | Más dificil, hay que recrear el objeto Filtro. |
| Posibilidad de intercambiar el orden de los filtros | Hay que reimplementar los mensajes de los filtros deseados en el orden que se quiere. | Hay que recrear el objeto filtro con el orden deseado. |
| Simplicidad | Más fácil de crear. | Más dificil de crear, hay que implementar y replicar todos los nuevos mensajes en todas las clases y subclases. |
| Cohesión | Menos cohesivo. | Más cohesivo. |
| Mantenibilidad | Mayor dificultad. | Menor dificultad, aunque el debugging se puede volver engorroso. |