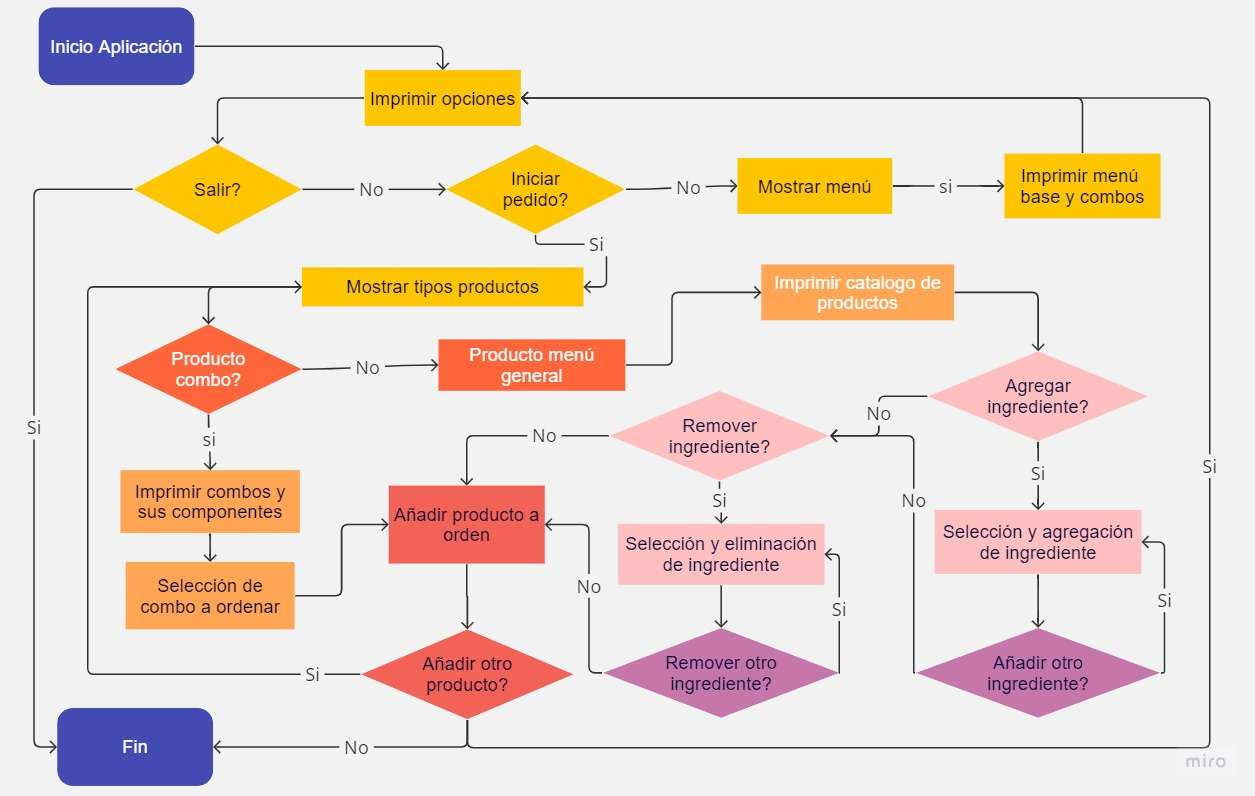
**Taller 1**

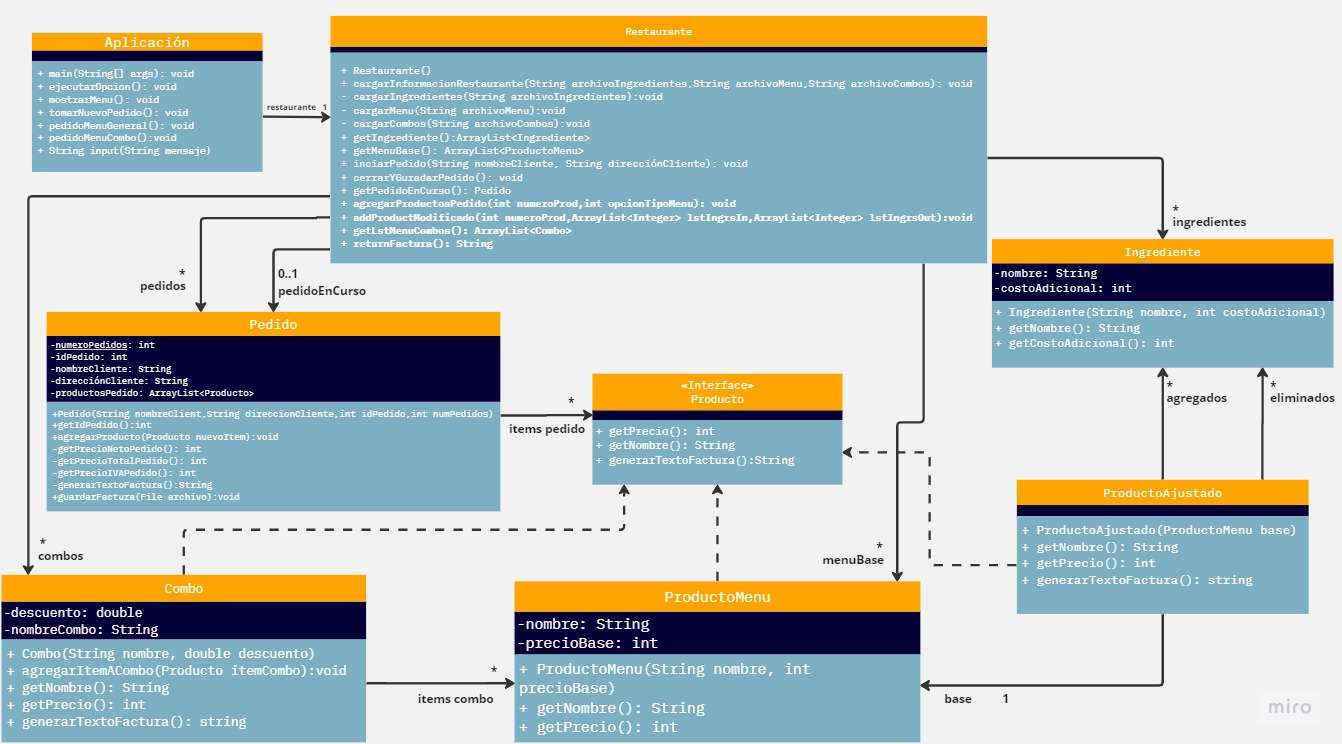
María José Amorocho, 202220179, m.amorocho@uniandes.edu.co

# **Modificaciones primera y segunda parte**

## **Diagrama de opciones y UML**



Para ver con mejor resolución: <https://miro.com/app/board/uXjVMmpwVCI=/?share_link_id=643884613794>



Para ver el diagrama con mejor resolución: <https://miro.com/app/board/uXjVMmkSCMc=/?share_link_id=431457928078>

## **Descripción de las modificaciones**

Las modificaciones implementadas se dividen en dos grandes categorías: los ajustes hechos en el paquete de consola, y aquellos implementados en la clase ‘Restaurante’.

Por su parte, a la clase restaurante se le incluyeron métodos adicionales; estos fueron:

* **agregarProductosPedido(int numeroProd,int opcionTipoMenu): void, cuya función es agregar un producto de las clases ProductoMenu o Combo a el pedido en curso. ‘numeroProd’ hace referencia al número de producto base o combo seleccionado, pues cada item del menú base y cada combo tiene un número asociado cuando se imprime en pantalla (por ejemplo, 1 en el menú de productos base corresponde a una hamburguesa corral). ‘opcionTipoMenu’ hace referencia al tipo de producto que el usuario seleccionó; 1 para productos del menú base y 2 para combos.**
* **addProductModificado(int numeroProd,ArrayList<Integer> lstIngrsIn, ArrayList<Integer> lstIngrsOut):void, que sirve para construir un objeto de la clase ProductoModificado a medida que el usuario va seleccionando los ingredientes que desea añadir o remover, y agregar este en el pedido en curso. ‘numeroProd’ al número de producto base seleccionado. ‘lstIngrsIn’ es una lista con los números correspondientes a los ingredientes que el usuario quiere añadir a su orden, y ‘lstIngrsOut’ es otra lista con los números asociados a los ingredientes que el usuario desea remover de su pedido.**
* **getLstMenuCombos(): ArrayList<Combo>, que se encarga de retornar una lista con todos los combos incluidos en el menú.**
* **returnFactura(): String, cuyo propósito es retornar un String con la información de un pedido una vez finalizado.**

Los dos primeros métodos fueron creados ante la necesidad de agregar varios ítems de diferentes clases productos al pedido en curso. Así mismo, el segundo método descrito le da la posibilidad al usuario de ordenar un producto ajustado en el que puede añadir o remover más de un único ingrediente.

Los dos últimos métodos tienen el objetivo de retornar información relevante para cuando se ejecuta la consola, de manera que su utilidad funcional está asociada al retorno impreso de datos.

Por otro lado, en la clase ‘Aplicación’ los métodos añadidos fueron:

* tomarNuevoPedido(): void, que invoca a métodos ‘Restaurante’ para inicializar un nuevo pedido.
* pedidoMenuGeneral(): void, que se encarga usar métodos de ‘Restaurante’ para agregar al pedido que pertenezca a las clases de ‘ProductoMenu’ o ‘ProductoModificado’.
* pedidoMenuCombo():void, que implementa métodos de ‘Restaurante’ para añadir al pedido un producto de la clase ‘Combo’.
* String input(String mensaje): String, que sirve como método “auxiliar” para que el usuario pueda ingresar información al programa.

Los anteriores métodos hubiesen podido estar escritos dentro del método ‘ejecutarOpicion()’; sin embargo, se consideró que era mejor trabajarlos por aparte para evitar posibles errores ante la cantidad de opciones que el usuario puede elegir. Dado a que el cliente no tiene un límite de cantidad o clases de productos que puede ordenar, así como puede elegir cuantos ingredientes extra o a remover quiera, se requieren varias estructuras iterativas que pueden ser complicadas de implementar en un solo método.

Como observación final vale la pena mencionar que no se eliminaron características incluidas en el modelo original y se conservaron las relaciones existentes entre cada clase.