Fecha: 5/ 12/ 2024

**Comedor UNQ - TP Final Programación con Objetos I**

Grupo 3 - Integrantes:

Dores Joaquín; Sandoval Thomas.

**Índice:**

**Agrega encabezados (Formato > Estilos de párrafo) y aparecerán en el índice.**

**Introducción**

En este informe estaremos brindando una observación completa sobre el desarrollo y composición de nuestro proyecto final para la materia de Programación con Objetos I, de la Universidad de Quilmes.

En esta ocasión se nos pidió que implementemos una nueva aplicación para poder gestionar el comedor de la universidad, desarrollando esta en el lenguaje smalltalk, bajo el marco de Cuis University.

Dentro del desarrollo de este gestor se pide que cumpla diversas funciones para poder manejar tanto la parte de organización de menús y pedidos, los consumos de todos los comensales y el stock de todos los productos y menús.

**Características generales**

Para el desarrollo del sistema mencionado, implementamos un sistema donde existen 3 tipos de comensales: el estudiante y el personal de la UNQ, que están registrados en el comedor como tales, y el comensal externo, que es aquel que no pertenece a la universidad pero quiere acceder a los servicios del comedor.

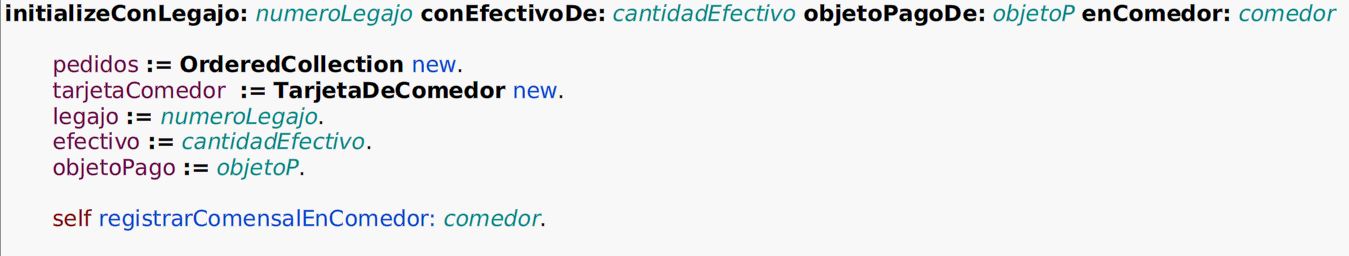
Cada comensal puede realizar un pedido que puede incluir 3 tipos de producto distinto, siendo estos productos de góndola, el menú estudiantil(solo para estudiantes) y el menú del día. El comensal tiene la opción de realizar el pago en efectivo o con “Objeto Pago”, y en el caso de los estudiantes y el personal de la UNQ, disponen de la opción de realizar los pagos con su Tarjeta UNQ.

También, existen distintos tipos de descuentos y recargas en el precio del pedido, dependiendo de los productos seleccionados en el pedido, el tipo de comensal que realizó el pedido y el método de pago elegido.

**Alcance**

El proyecto llega a cumplir con todas las características que pedía el enunciado, tanto en funcionamiento general del sistema, como en las funciones más específicas, por ejemplo que el comedor pueda saber cuales son los pedidos hechos los lunes pagando en efectivo.

Cabe aclarar que el proyecto fue llevado a cabo tratando de darle el enfoque más realista posible, teniendo en cuenta que no se rompa la encapsulación en los métodos, y creando los objetos de la manera que a nosotros nos pareciera óptima. Por ejemplo, cuando se inicializa un objeto de la clase estudiante o personal de la UNQ, este se registra en el sistema del comedor (siendo este el sistema de la universidad) en el momento que se inicializa, ya que, bajo nuestro punto de vista, un estudiante o un personal de la UNQ ya forma parte del sistema de la universidad al ser parte de esta como tal.



Otro caso es el de la inicialización de los pedidos, estos existen en el sistema como pedidos una vez que el comensal realiza el pedido. El pedido no existe por sí solo, sino que es el comensal el que lo crea; si bien se puede inicializar un objeto de la clase pedido por si solo, no lo llevamos a cabo de esa manera en los test, ya que no consideramos que sea el enfoque correcto.

Como característica adicional (ajena al enunciado), agregamos una función para cargar la tarjeta UNQ, ya sea con efectivo o con “Objeto Pago”, de esta manera, ocurre lo lógico al cargarle dinero a la tarjeta: ese dinero se descuenta, ya sea del efectivo del comensal, o de su saldo en Objeto Pago.

**Modelo**

El sistema cuenta con un objeto comedor, este tiene un registro de su stock, es decir sus productos disponibles; también lleva la cuenta de todos los pedidos realizados y de todos los comensales que están registrados en el comedor, es decir los que son personal de la UNQ o estudiantes. El comedor se encuentra abierto los días hábiles (de lunes a viernes), y los sábados y domingos permanece cerrado.

Todo esto hace al comedor indispensable a la hora de realizar un pedido, ya que se tiene en cuenta si este está abierto en el día del pedido, si tiene stock disponible, y si el comensal está registrado o no en sistema, para así poder chequear si un comensal registrado realmente puede realizar un pedido como tal.

Por otro lado, se modeló el objeto comensal; como ya se mencionó antes, este puede ser un externo a la universidad, es decir que no está registrado, o puede ser un comensal registrado en sistema, es decir que pertenece a la universidad, por lo tanto este tendrá su tarjeta de la UNQ para poder pagar con esta sus pedidos si así lo desea. Existen dos tipos de comensales registrados: los estudiantes y los empleados o personal de la UNQ (bajo el nombre de “Personal UNQ”). Las únicas dos diferencias entre estos dos es que el estudiante puede incluir el menú estudiantil en sus pedidos y el personal UNQ no, y que el personal UNQ posee descuentos para el menú del día y los productos de góndola que son distintos a los descuentos del estudiante.

Todo comensal cuenta con una cantidad de efectivo y un saldo en objeto pago, para realizar los pagos de sus pedidos.

Los comensales registrados tienen un número de legajo que los identifica y además se les otorga una tarjeta de comedor, que también es un objeto, para poder realizar el pago de sus pedidos, pagando con esta, pueden acceder a diversos descuentos.

Un comensal puede realizar un pedido con los productos que este quiera seleccionar, estos deben estar en el stock del comedor y además deben ser productos que el comensal tenga permitido acceder(por ejemplo un comensal externo no puede hacer un pedido con un menú estudiantil); luego este debe pagar el pedido con cualquiera de sus métodos de pago disponibles. Dependiendo del tipo de comensal que sea, los productos que lleve, el dia de la semana que se realice el pedido y el método de pago con el que se abone, el precio puede variar, ya que existen diversas recargas y descuentos para cada producto e incluso para el precio del pedido en general, dependiendo de los factores recién mencionados.

El objeto pedido es inicializado por el comensal cuando este realiza un pedido; cada pedido tiene un comensal que lo ordenó, una lista de productos que se incluyen en el pedido, el precio total de estos sumados, el precio que el comensal pagó por los productos llevados y el dia en que fue realizado el pedido; también se puede saber si el pedido fue abonado o no, y el método de pago con que fue pagado, en caso de que este haya sido abonado.

Existen 3 tipos de productos disponibles: los productos de góndola, el menú del día y el menú estudiantil. Cada producto tiene un precio general distinto, y algunos tienen distintos precios dependiendo del tipo de comensal que realice el pedido con este producto.

**Progreso**

El proyecto fue realizado de manera iterativa e incremental utilizando TDD, para más información sobre nuestro progreso, se puede consultar los test realizados en la clase testCase “ComedorTest” de nuestro proyecto.

Para comenzar definimos la clase comedor, probando si un comedor estaba abierto un día de semana y cerrado un día no hábil. Luego definimos a los comensales, modelando cada tipo y creando un objeto comensal de cada tipo en el setup de los testCase, así como el comedor; en un principio estos responden solo a funciones básicas como ver su saldo o su cantidad de efectivo.

Luego modelamos los distintos tipos de productos con sus precios, para después poder añadir estos al stock del comedor, y así comenzar a tener un comedor que puede o no tener productos disponibles.

Una vez hecho esto, comenzamos a testear características de los comensales, como por ejemplo que un estudiante estaba registrado en el comedor y un comensal externo no lo estaba.

Luego definimos la tarjeta de comedor y sus características, para que pueda cumplir el rol de método de pago de los comensales registrados. Y luego testeamos si estos podían cargar la tarjeta ya sea con efectivo o con Objeto Pago.

Tras tener definidos los comensales y el comedor y sus características, empezamos a testear la posibilidad de que un producto tenga o no un descuento, para luego decidir que un producto puede tener distinto precio dependiendo el método de pago con el que sea pagado y del tipo de comensal que lo incluya en su pedido. Así fuimos definiendo los distintos tipos de descuentos y recargas que podían llegar a tener cada uno.

También tuvimos en cuenta que existen productos que no pueden ser agregados al pedido de cualquier comensal, por ende añadimos y testeamos sus debidas restricciones.

Una vez que definimos cómo responde cada producto a su precio o restricción, empezamos a modelar los pedidos; estos se conformaban principalmente por una lista de productos, que determinarían, dependiendo también del comensal y el método de pago elegido, su precio final. En un principio inicializamos un pedido en un test para comprobar que este exista y funcione como debía; pero luego de esto comenzamos a inicializarlo mediante el comensal, ya que él es el que encarga el pedido y decide todas sus características.

Así, fuimos testeando distintos casos donde distintos tipos de comensales realizaban pedidos con distintos productos, o abonaban con distintos métodos de pago. De esta manera evaluamos si se cumplian las restricciones antes mencionadas, si se aplicaban correctamente los descuentos y las recargas, y si el pedido se registraba de manera correcta en el comedor y tenia sus datos bien, como por ejemplo, si un comensal no tenía suficiente dinero para pagar el pedido, este último quedaba como no pago y no se registraba en el comedor.

**Dificultades y Observaciones**

Entre las dificultades que pudimos llegar a encontrarnos a medida que íbamos haciendo el trabajo fue el de cómo hacer los métodos de pagos de los comensales, los filtros que se pedían para los comedores y los descuentos que tenían los productos, dependiendo de qué tipo de comensales eran. Para el 1er problema lo solucionamos creando un objeto de “TarjetaUNQ” que sólamente lo tienen asignado los estudiantes y los empleados de la UNQ, y con los otros dos métodos se los asignamos directamente con una cantidad de pesos determinada que se inicializaba en el momento en que se creaba, además de que sólamente los registrados podían tener la TarjetaUNQ en su posesión, así que por eso, los registrados tenían una inicialización diferente a la de los externos, que no podían tener ésa tarjeta. Para los filtros tuvimos que buscar algunos que otros mensajes en las diapositivas de la materia, para refrescarlos y poder hacerlos más fáciles. Creamos distintos tipos de filtros en donde se pueden mezclar entre sí y que son simples y concisos cada uno de estos. Y para la última dificultad tuvimos que usar muchos double dispatch entre los comensales y los productos, tantos que, a la hora de llegar a el punto de aplicar el descuento cuando se paga en efectivo los lunes, nos quedó un ifTrue y un ifFalse, que fue necesario para el punto siguiente.

**Condicionales e Ifs**

Para éste punto, como se mencionó previamente tuvimos que dejar, obviando los casos de ifFalse, que son principalmente para los mensajes de error, habremos dejado al menos 2 ifs en el momento de que el comensal hace un pedido y paga el mismo. El tema es que en estos 2 casos no podríamos supimos cómo hacer double dispatch, que creemos que no se podía por cómo hicimos el trabajo, y tampoco se podía hacer un Hundred o un Gestor de pago porque tampoco teníamos la cantidad de condiciones necesarias para hacer un buen uso del mismo, ya que estos se implementan cuando hay varias pasos a completar para poder devolver diferentes resultados, por lo que hemos decidido dejar los mismísimos ifs en el proyecto.

Nuestra idea principal era hacer hundreds para que gestione los métodos de pagos, los comensales y sus descuentos pero, nuevamente, por cómo habíamos avanzado con el proyecto, no era algo muy viable y lindo para resolver.

**Conclusiones**

Por conclusión, pudimos ver la importancia que tienen todos y cada uno de los objetos, además de cuáles es su función en el programa, lo importante que son y nos fueron los double dispatch, la gran cantidad de mensajes y funciones que tienen las colecciones y por último, y más importante, lo incremental y progresivo siempre es lo mejor a la hora de programar, separar las tareas entre objetos y las responsabilidades que tiene cada uno, si corresponde que tenga ésa responsabilidad y si tiene coherencia en el mismo. Lo que no pudimos usar y, que queríamos implementar, eran los hundreds y/o gestores para tener las cosas más ordenadas y sin tantos dobles dispatches y, sobre todo, la eliminaciones de los ifs que podrían generar ambigüedades y errores en el futuro. Al terminarlo también pudimos observar que podríamos haber hecho a todos los métodos de pagos como objetos, y así podríamos haber usado más felizmente los gestores.