**Algoritmos y Programación II**

**Trabajo Práctico 2**

**Restaurant**

**Integrantes:**

* **Bellotti, Matias**
* **De Alesandro, Franco**
* **Hansen, Nicolas**
* **Juguera, Matias**

*Introducción*

En este informe se detalla los procedimientos llevados a cabo para desarrollar el programa pedido. En este caso, un administrador para un restaurante dividido en tres módulos: módulo de precio, módulo de mesas y módulo de caja. En el primer módulo, se administran los productos y los combos ofrecidos con sus respectivos precios. En el módulo de mesas, se administran las consumiciones de las mesas y sus estados (ocupada, disponible y cerrada). Y en el módulo de caja, se genera el ticket con las consumiciones por mesa y lista los tickets ya generados a partir de ciertas fechas. Para este trabajo no utilizamos consola por lo que desarrollamos tests para comprobar el buen funcionamiento del programa. En los tests se probó cada método de cada clase y, en caso de haber, sus respectivas excepciones.

Para comenzar el trabajo práctico, desarrollamos un diagrama de clases que en un principio tuvo ciertos errores. A medida que fuimos desarrollando el programa, fuimos cambiando también los métodos del diagrama.

Para este programa, se nos pidió que utilizaramos dos de los tres patrones de diseño aprendidos. En este caso, utilizamos el patrón singleton y el patrón composite.

*Desarrollo*

Como se dijo anteriormente, el programa está principalmente dividido en tres módulos. Cada uno de estos módulos representa una clase en la que se utilizó el patrón singleton, para que, de esta manera, solo pueden ser instanciados una vez.

El módulo de precios administra los productos únicos y los combos. Se utilizó una clase abstracta llamada Producto, de la cual extienden estas dos clases. El combo debía contener una lista de productos únicos que contiene, y a su vez también podría contener otros combos dentro. Para que esto sea posible, implementamos el patrón composite. Cada producto único tiene su propio precio de costo y precio de venta (el precio de venta nunca debe ser menor al precio de costo), además de una descripción del producto y su categoría (bebida con alcohol, bebida sin alcohol, minutas, pizzas o hamburguesas), la cual fue programada como un enum. El producto también tiene un id único generado por un contador estático que sumaba cada vez que se creaba un nuevo producto. El combo por su parte tiene su descripción y un descuento que se aplicará al precio de venta total de los productos que contiene. La clase Combo tiene un HashMap de Productos donde se guarda los productos ingresados como clave y su cantidad como valor.

El módulo de mesas genera la cantidad de mesas que habrá en el restaurant. Se encarga de registrar todas las consumiciones de una mesa, puede cambiarle los estados (mesa disponible, ocupada o cerrada). En las mesas se almacenarán las consumiciones con un HashMap, donde su clave será el producto consumido y el valor será la cantidad de estos productos. Las mesas tendrán un número de mesa y un estado, el cual siempre empezará como disponible al crearse. El estado de las mesas se programó con un enum y sus condiciones están dentro de la clase Mesa.

El módulo de caja se encargará de generar los tickets para una mesa (la cual debe estar ocupada y tener consumiciones) y los muestra por consola, donde imprime los productos consumidos y el total a pagar. Además los tickets serán almacenados en un ArrayList. El módulo también puede imprimir los tickets generados entre dos fechas específicas, devolviéndonos la suma de todos los ingresos. La clase Ticket tendrá el número de la mesa, un monto total, y un código de identificador único generado por un contador estático que aumenta cada vez que se crea un ticket. El ticket también tiene fecha y hora en que se creó, para esto utilizamos la clase “Date”.

Para ciertos casos donde un parámetro o un método podía generar algún error, utilizamos excepciones creadas por nosotros mismos que imprimen un mensaje especificando el error ocurrido.

Desarrollamos tres clases de Junit para hacer los tests de cada módulo. Para los tests del módulo de mesas utilizamos un método Before que limpia los atributos de los singletone. En estos tests probamos cada método de cada clase y, en caso de haber, sus excepciones.

*Directorio*

<https://github.com/Tps-UNTREF/Restaurante-TP>

*Conclusión*

En síntesis, en este TP se pudo aprender como es el uso de patrones de diseño en un situación cotidiana (es decir en el desarrollar una aplicación para un restaurante). En este trabajo hubieron muy pocas equivocaciones aunque al generar el uml se generó una interfaz con atributos y una clase solo con métodos que éste heredaría a dos clases (esta clase era la de productos) pero se pudo cambiar esa interfaz con una clase abstracta juntando esos atributos y métodos.

**Lista de archivos .Java:**

**Clases:**

* ModuloCaja.java
* Ticket.java
* ModuloMesa.java
* Mesa.java (enum Estado)
* ModuloPrecios.java
* Producto.java (enum Categoria)
* Combo.java
* ProductoUnico.java

**Excepciones:**

* CantidadDeProductosInvalidosExcepcion.java
* DescuentoInvalidoExcepcion.java
* MesaEstadoInvalidoExcepcion.java
* MesaNoDisponibleExcepcion.java
* MesaNoOcupadaExcepcion.java
* MesasYaGeneradasExcepcion.java
* PrecioDeCostoInvalidoExcepcion.java
* PrecioDeVentaInvalidoExcepcion.java
* ProductoADarDeBajaNoExistenteExcepcion.java
* ProductoYaExistenteExcepcion.java
* ProductoNoEncontradoExcepcion.java

**Test:**

* ModuloCajaTest.java
* ModuloMesasTest.java
* ModuloPreciosTest.java