Proyecto Final

ITAM | Otoño-2020

Algoritmos y prgramas - 004

MAURICIO VERDUZCO CHAVIRA - 195106

2020

Índice

[Problemática Por Resolver 2](#_Toc58257436)

[Análisis del problema 3](#_Toc58257437)

[Descripción de la Solución 4](#_Toc58257438)

[Diagramas UML 6](#_Toc58257439)

[Recetas 6](#_Toc58257440)

[Meseros 7](#_Toc58257441)

[Mesa 8](#_Toc58257442)

[Restaurante 9](#_Toc58257443)

[Implementación de Eclipse 12](#_Toc58257444)

[Recetas 12](#_Toc58257445)

[Meseros 14](#_Toc58257446)

[Mesa 17](#_Toc58257447)

[Restaurante 21](#_Toc58257448)

[RestauranteVista 29](#_Toc58257449)

[RestauranteControlador 36](#_Toc58257450)

[Pruebas y Resultados 41](#_Toc58257451)

[Primera prueba 41](#_Toc58257452)

[Segunda prueba 44](#_Toc58257453)

[Tercera prueba 46](#_Toc58257454)

[Cuarta prueba 47](#_Toc58257455)

[Quinta prueba 48](#_Toc58257456)

[Sexta prueba 49](#_Toc58257457)

[Manual de Usuario 52](#_Toc58257458)

[Partes del programa 52](#_Toc58257459)

[Instructivo 53](#_Toc58257460)

[Conclusiones 55](#_Toc58257461)

# Problemática Por Resolver

El mundo de la administración de negocios gastronómicos es un reto en todos los sentidos y cualquier aspecto que se pueda simplificar le será útil a un restaurantero. Entre los problemas más importantes es la contabilidad básica y correcta del día a día, el control de los pedidos, de los clientes y la repartición correcta de las propinas.

El objetivo de este programa es ayudar a la simplificación de esta problemática de una manera simple, pues al trabajar en un restaurante, no hay mucho tiempo para descifrar un programa. Este suele ser el problema de la implementación de software a un negocio gastronómico y el problema que me propongo resolver.

Mi familia, desde que tengo memoria, ha sido una apasionada por la comida, tanto así que hemos tenido más de un restaurante. Toda mi experiencia laborar es en este ámbito y me considero apasionado por la cocina.

A lo largo de este proceso, me ha tocado estar encerrado en una oficina, horas después del cierre del local, intentando que las cuentas cuadren, buscar *tickets* y recibos e intentar hacer memoria de lo que sucedió en el día. Es algo muy desgastante y que considero que, con una buena organización, no debería de suceder.

Mi familia y yo tenemos el objetivo de abrir un restaurante de cocina de autor y es por eso que me propongo escribir un programa funcional para la situación que se nos aproxima. Para que de esta manera no haya inconvenientes desde el primer día.

# Análisis del problema

Para un restaurante “común” la solución sería más compleja, porque implicaría una base de datos completa de todos los platillos. Pero en el caso de la cocina de autor, el platillo cambia todos los días, así como el precio, por lo tanto, ese es el primer reto. Agregado a esto, es importante que exista la posibilidad de eliminar platillos, pues en muchas ocasiones, los clientes cambian de opinión y lo encargados no pueden dar de baja lo ya registrado.

Acto seguido, poder tener control de los meseros, los cuales subdividen las propinas al finalizar el día puesto que no se pueden poner a contar en el momento. De esta manera suelen haber discrepancias al terminar la jornada. Este es el segundo reto por solucionar.

Un restaurante de autor suele tener reservaciones con días, semanas y a veces meses de anticipación, así que el problema deberá tener una manera de implementarlo, de tal manera que “si el usuario no se acordara” aún así no hubiese margen de error.

Por último, es importante poder conocer todos los pedidos que se hicieron en un día. Saber quien los atendió, en que mesa estuvieron y de cuanto fue su cuenta. Estos son los detalles que consideramos intrascendentes y son los que nos rompen la espalda al final del día. Agregado a esto, hay que tener en claro cuanto fue el ingreso del restaurante ese día.

# Descripción de la Solución

Para el primer problema, el de los platillos, se creó una clase “Recetas”, la cual tiene como atributos un nombre y un precio. Estas recetas se podrán dar de alta en un futuro en cada mesa cuando el usuario “tome el pedido”.

Para los meseros, se creó una clase “Mesero”, que distingue el nombre de cada mesero, les asigna una clave estática, una variable de propinas, que se inicializará en cero cuando se corra el programa e irá aumentando a lo largo del día. Y una variable booleana de ocupación pues un mesero ocupado no podrá atender una mesa.

De esto se creó la clase “Mesa” que será crucial para el desarrollo del programa. Dicha clase tiene como atributos el nombre del titular, el número de la mesa asignada (en un restaurante los meseros les ponen nombres a las mesas de manera arbitraría), el numero foliado del pedido, un arreglo de “Recetas” que se usará para registrar el pedido, una variable para identificar si ya se lo tomó el pedido a la mesa y un mesero asignado.

Esta clase aparte de las funciones básicas tiene también cuatro particulares de la clase:

1. tomarPedido: Se da el nombre del platillo y su precio y se agrega al arreglo. De esta manera estamos “tomando el pedido”.

eliminarPedido: Con ayuda del “ManejadorArreglosGenerico” se busca un platillo y si existe, se elimina del arreglo del pedido.

1. calculaTotal: Esta función busca todos los precios de todos los platillos en el arreglo y los suma.
2. ImprimirCuenta: Esta función manda a llamar la información de cada receta dentro del arreglo y lo despliega de manera agradable.

NOTA: las funciones 2 y 3 funcionan solo si ya se les tomó el pedido

Por ultimo la clase “Restaurante” la cual tiene nombre, domicilio y teléfono. Así como una matriz de mesas y un arreglo de meseros. También tiene una lista en la que se van registrando todos los pedidos hasta ahora, una variable que almacena las ganancias y otra que evalúa si el restaurante está habilitado. Entre las funciones de la clase hay:

1. iniciarMesa: Función que, según los atributos de la mesa y las evaluaciones de la mesa, la puede “inicializar”.
2. leerHojaReservas: Utiliza el Scanner de archivos para leer un documento con las reservaciones e inicializa todas esas mesas.
   1. **NOTA**: para leer un archivo con el programa es necesario que el archivo esté guardado en el mismo lugar que el programa. Es posible hacer que el archivo esté dentro de la información general del jar. Pero si este fuera el caso y el usuario no tuviera acceso a modificarlo todos los días según las reservaciones diarias, el programa no tendría mucho sentido y el botón de reservas solo serviría el primer día.
3. tomarElPedido: Con ayuda de las funciones de la clase “Mesa”, el usuario puede tomar el pedido de la mesa.
4. eliminarElPedido: con ayuda de las funciones de la clase “Mesa”, el usuario puede eliminar el pedido de la mesa.
5. imprimeCuenta: El usuario imprime la cuenta de la mesa.
6. pagarCuenta: El usuario paga la cuenta y le asigna al mesero su propina, lo declara como desocupado, regresa el cambio del cliente, agrega la información de la mesa al registro y libera la mesa, es decir, elimina la información para que se pueda volver a usar.
7. Hacer el corte: Esta función pregunta si todas las mesas se desocuparon correctamente y despliega toda la información del día de una manera fácil de entender. Esta función cambia la variable de “habilitado” a falso. De esta manera el restaurante, después del corte NO podrá seguir funcionando. Claro, hasta el siguiente día.

Por último, para la correcta ejecución del programa, se creó una interfaz, la cual permite el acceso y la manipulación de todas las funciones anteriores según lo que al usuario le convenga.

## Diagramas UML

### Recetas

|  |
| --- |
| Recetas |
| - nombre: String  - precio: double |
| + Recetas (String, double)  + getNombre (): String  + setNombre (String)  + getPrecio (): double  + setPrecio (double)  + toString (): String  + hashCode (): int  + equals (Receta): boolean  + compareTo (Recetas): int |

### Meseros

|  |
| --- |
| Meseros |
| - nombre: String  - clave: int = c  - c: int = 100  - propinas: double = 0  - boolean: ocupado = false |
| + Meseros ()  + Meseros (String)  + getNombre (): String  + setNombre (String)  + getPropinas (): double  + setPropinas (double)  + getClave (): int  + setOcupado (boolean)  + getOcupado (): boolean  + toString (): String  + hashCode (): int  + equals (Meseros): boolean  + compareTo (Meseros): int |

### Mesa

|  |
| --- |
| Mesa |
| - nombre: String  - numMesa: int  - numPedido: int = contador  - contador: int = 1  - pedido: [] Recetas  - MAXN: int = 100  - numA: int = 0  - pedidoTomado: Boolean = false  - mes: Meseros |
| + Mesa ()  + Mesa (String, int, Mesero)  + getMes (): Meseros  + setMes (Mesero)  + getPedidoTomado (): boolean  + setPedidoTomado(boolean)  + getNombre (): String  + setNombre (String)  + getNumMesa (): int  + getNumPedido (): int  + tomarPedido (String, double): boolean  + eliminarPedido (String, double): boolean  + calculaTotal (): double  + imprimirCuenta (): String  + toString (): String |

### Restaurante

|  |
| --- |
| Restaurante |
| - nombre: String  - domicilio: String  - telefono: String  - estrellas: int  - mesas: [] [] Mesas  - MAXR: int = 3  - MAXC: int = 3  - ganancias: double = 0  - tiempo: double = LocalDateTime.*now*  - horario: String = tiempo  - fecha: String = tiempo  - cuentas: ArrayList<String>  - mess: [] Meseros  - habil: Boolean = true |
| + Restaurante ()  + Restaurante (String, String, String, int)  + iniciarMesa (String, int, int, int): Mesa  + leerHojaReservas (): String  + tomarElPedido (int, String, double): String  + eliminarElPedido (int, String, double): String  + imprimeCuenta (int): String  + pagarCuenta (double, int, int): String  + hacerCorte (): String  + toString (): String |

### Restaurante Vista

|  |
| --- |
| RestauranteVista extends JFrame |
| - frmShnippenketer: JFrame  - txNomTit: JTextField  - txNumMesa: JTextField  - txRow: JTextField  - txCol: JTextField  - txPlatillo: JTextField  - txPrecio: JTextField  - txPropina: JTextField  - txPago: JTextField  - btnIniciarMesa: JButton  - btnTomarPedido: JButton  - btnImprimirCuenta: JButton  - btnPagarCuenta: JButton  - btnHacerCorte: JButton  - btnLeerReservas: JButton  - btnClear: JButton  - btnInformacion: JButton  - btnEliminarPedido: JButton  - lblInicializacin: JLabel  - lblNombreDelTitular: JLabel  - lblNoMesa: JLabel  - lblNoCol: JLabel  - lblNoRow: JLabel  - lblPedido: JLabel  - lblPlatillo: JLabel  - lblPrecio: JLabel  - lblFilipinos: JLabel  - lblPagar: JLabel  - lblPropina: JLabel  - textArea: JTextAtea |
| + RestauranteVista (String)  + main (String [] args): static void |

### RestauranteControlador

|  |
| --- |
| RestauranteControlador extends RestauranteVista |
| - r: Restaurante (String, String, String, int) |
| + RestauranteVista (String)  + EscuchadorIniciaMesa ()  + EscuchadorLeerReservas ()  + EscuchadorTomarPedido ()  + EscuchadorImprimirCuenta ()  + EscuchadorPagarCuenta ()  + EscuchadorHacerCorte ()  + EscuchadorClear ()  + EscuchadorInformacion ()  + EscuchadorEliminarPedido ()  + main (String [] args): static void |

# Implementación de Eclipse

## Recetas

**package** librerias;

/\*\*

\*

\* **@author** Mauricio Verduzco Chavira

\* **@version** 2.0

\* **@Fecha**: 06/12/2020

\* **@Clase**: Recetas

\* **@Descripción**: Primera clase primaria del programa "Restaurantes". Consta de dos atributos, nombre y precio.

\*/

**public** **class** Recetas {

**private** String nombre;

**private** **double** precio;

/\*\*

\*

\* Para construir una receta siempre se necesitará el nombre y el precio

\*/

**public** Recetas(String nombre, **double** precio) {

**this**.nombre=nombre;

**this**.precio=precio;

}

/\*\*

\*

\* Esta clase solo tiene como funciones getters y setters así como las de funcionalidad básica.

\* toString, compareTo & Equals

\* Cabe destacar que en este programa solo se utilizó getPrecio. Pero por efectos prácticos o de actualizaciones futuras dejamos los demás getters y setters.

\*/

**public** String getNombre() {

**return** nombre;

}

**public** **void** setNombre(String nombre) {

**this**.nombre = nombre;

}

**public** **double** getPrecio() {

**return** precio;

}

**public** **void** setPrecio(**double** precio) {

**this**.precio = precio;

}

@Override

**public** String toString() {

**return** (nombre + ".............. $" + precio);

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + ((nombre == **null**) ? 0 : nombre.hashCode());

**long** temp;

temp = Double.*doubleToLongBits*(precio);

result = prime \* result + (**int**) (temp ^ (temp >>> 32));

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Recetas other = (Recetas) obj;

**if** (nombre == **null**) {

**if** (other.nombre != **null**)

**return** **false**;

} **else** **if** (!nombre.equals(other.nombre))

**return** **false**;

**if** (Double.*doubleToLongBits*(precio) != Double.*doubleToLongBits*(other.precio))

**return** **false**;

**return** **true**;

}

**public** **int** compareTo(Recetas otro) {

**return** nombre.compareTo(otro.nombre);

}

}

## Meseros

**package** librerias;

/\*\*

\*

\* **@author** Mauricio Verduzco Chavira

\* **@version** 2.0

\* **@Fecha**: 06/12/2020

\* **@Clase**: Meseros

\* **@Descripcion**: Segunda clase primaria del programa "Restaurantes". Consta de cuatro atributos, nombre, clave, propinas y ocupado.

\*/

**public** **class** Meseros {

**private** String nombre;

**private** **int** clave;

**private** **static** **int** *c*=100;

**private** **double** propinas;

**private** **boolean** ocupado;

**public** Meseros() {

**this**.clave=*c*;

*c*++;

}

/\*\*

\*

\* Un mesero se construye solo con su nombre, la clave es autogenerada con el constructor vacío, las propinas inician en cero y el mesero se inicia desocupado

\*/

**public** Meseros(String nombre) {

**this**();

**this**.nombre=nombre;

propinas = 0;

ocupado=**false**;

}

/\*\*

\*

\* La clase meseros tiene solo funciones básicas de la clase: getters, setters, compareTo, toString & Equals.

\* Al igual que en recetas, no se usan todos los getters y los setters, pero dejamos todos. (Excepto set clave).

\*/

**public** String getNombre() {

**return** nombre;

}

**public** **void** setNombre(String nombre) {

**this**.nombre = nombre;

}

**public** **double** getPropinas() {

**return** propinas;

}

**public** **void** setPropinas(**double** propinas) {

**this**.propinas = propinas;

}

**public** **int** getClave() {

**return** clave;

}

**public** **boolean** isOcupado() {

**return** ocupado;

}

**public** **void** setOcupado(**boolean** ocupado) {

**this**.ocupado = ocupado;

}

@Override

**public** String toString() {

**return** ( nombre + " $" + propinas);

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + clave;

result = prime \* result + ((nombre == **null**) ? 0 : nombre.hashCode());

**long** temp;

temp = Double.*doubleToLongBits*(propinas);

result = prime \* result + (**int**) (temp ^ (temp >>> 32));

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Meseros other = (Meseros) obj;

**if** (clave != other.clave)

**return** **false**;

**if** (nombre == **null**) {

**if** (other.nombre != **null**)

**return** **false**;

} **else** **if** (!nombre.equals(other.nombre))

**return** **false**;

**if** (Double.*doubleToLongBits*(propinas) != Double.*doubleToLongBits*(other.propinas))

**return** **false**;

**return** **true**;

}

**public** **int** compareTo(Meseros otros) {

**return** clave-otros.clave;

}

}

## Mesa

**package** librerias;

/\*\*

\*

\* **@author** Mauricio Verduzco Chavira

\* **@version** 2.0

\* **@Fecha**: 06/12/2020

\* **@Clase**: Mesa

\* **@Descripción**: Clase secundaria del programa "Restaurantes". Consta varios atributos, entre ellos, un mesero y un ArrayList de Recetas.

\*/

**import** java.util.\*;

**public** **class** Mesa {

**private** String nombre;

**private** **int** numMesa;

**private** **int** numPedido;

**private** **static** **int** *contador* = 1;

**private** Recetas [] pedido;

**private** **final** **int** MAXN = 100;

**private** **int** numA;

**private** **boolean** pedidoTomado;

**private** Meseros mes;

**public** Mesa() {

pedido = **new** Recetas[MAXN];

numA = 0;

numPedido = *contador*;

*contador*++;

}

/\*\*

\*

\* **@param** nombre

\* **@param** numMesa

\* **@param** alguien

\* Una mesa se construye con el nombre del titular, el número de la mesa que se le asigna y un mesero.

\* La mesa tiene un contador foliado del número del pedido, se inicia el ArrayList de Recetas y el pedido tomado se inicializa en falso.

\*/

**public** Mesa(String nombre, **int** numMesa, Meseros alguien) {

**this**();

**this**.numMesa=numMesa;

**this**.nombre=nombre;

**this**.pedidoTomado = **false**;

**this**.mes=alguien;

}

/\*\*

\*

\* **@return**

\* Funciones básicas de getters & setters

\*/

**public** Meseros getmes() {

**return** mes;

}

**public** **void** setmes(Meseros mes) {

**this**.mes = mes;

}

**public** **boolean** isPedidoTomado() {

**return** pedidoTomado;

}

**public** **void** setPedidoTomado(**boolean** pedidoTomado) {

**this**.pedidoTomado = pedidoTomado;

}

**public** String getNombre() {

**return** nombre;

}

**public** **void** setNombre(String nombre) {

**this**.nombre = nombre;

}

**public** **int** getNumMesa() {

**return** numMesa;

}

**public** **int** getNumPedido() {

**return** numPedido;

}

/\*\*

\*

\* Esta función agrega una nueva receta a la lista de pedido.

\* Recibe como parámetro el nombre y el precio del platillo y regresa un booleano.

\*/

**public** **boolean** tomarPedido(String nombre, **double** precio) {

**boolean** res = **false**;

**if**(numA<MAXN){

pedido[numA] = (**new** Recetas(nombre, precio));

pedidoTomado = **true**;

res = **true**;

numA++;

}

**return** res;

}

/\*\*

\* Esta función elimina un pedido. Primero evalúa la existencia del objeto tipo "Recetas" y luego lo elimina

\*/

**public** **boolean** eliminarPedido(String nombre, **double** precio) {

**boolean** res = **false**;

**int** Z;

**if**(numA>0) {

Recetas aux = **new** Recetas(nombre, precio);

Z = ManejadorArreglosGenerico.*eliminaEnDesordenado*(pedido, numA, aux);

**if** (numA>Z) {

numA=Z;

res=**true**;

}

}

**return** res;

}

/\*\*

\*

\* Esta función suma el precio de todos los platillos que estén en la lista de pedido.

\* No recibe nada como parametro y regresa el total

\*/

**public** **double** calculaTotal() {

**double** total=0;

**for**(**int** i=0; i<numA; i++)

total+=pedido[i].getPrecio();

**return** total;

}

/\*\*

\*

\* Esta función imprime la cuenta del usuario a través de un StringBuilder

\* No recibe parámetros y regresa el número del pedido y el nombre del mesero,

\* seguido de todos los platillos y su respectivo precio,

\* por último el total de la cuenta.

\*

\* Para que esta función se ejecute correctamente, la mesa debe tener su pedido tomado, caso contrario regresa un mensaje adecuado.

\*/

**public** String imprimirCuenta() {

StringBuilder cad = **new** StringBuilder();

**if**(pedidoTomado) {

cad.append("\nEl numero del pedido es "+numPedido);

cad.append("\nLos antendió: " + mes.getNombre());

**for**(**int** i = 0; i < numA; i++)

cad.append("\n= " + pedido[i].toString());

cad.append("\nEl total es de: $"+ calculaTotal() + "\n");

}

**else**

cad.append("<<El pedido no ha sido tomado>>");

**return** cad.toString();

}

/\*\*

\* toString adaptado

\*/

@Override

**public** String toString() {

**return** "\nMesa #" + numMesa + " [Titular: " + nombre + ", No. de pedido " + numPedido + ", pedido tomado: " + pedidoTomado + "]";

}

}

## Restaurante

**package** librerias;

/\*\*

\*

\* **@author** Mauricio Verduzco Chavira

\* **@version** 2.0

\* **@Fecha**: 06/12/2020

\* **@Clase**: Restaurante

\* **@Descripción**: Clase final del programa "Restaurantes". Consta de varios atributos, entre ellos, los más importantes son una matriz de mesas y un arreglo de meseros.

\*

\*/

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.time.LocalDateTime;

**import** java.time.Month;

**import** java.time.Year;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**public** **class** Restaurante {

**private** String nombre;

**private** String domicilio;

**private** String telefono;

**private** **int** estrellas;

**private** Mesa [][] mesas;

**private** **final** **int** MAXR=3;

**private** **final** **int** MAXC=3;

**private** **double** ganancias;

**private** **double** tiempo;

**private** String horario;

**private** String fecha;

**private** ArrayList<String> cuentas;

**private** Meseros [] mess;

**private** **boolean** habil;

/\*\*

\* Un restaurante con constructor vacío inicializa su matriz de mesas con un máximo de 9 y todos los meseros.

\* Se utilizan nueve meseros, en este caso, la nómina de un restaurante es constante y los meseros ya son conocidos,

\* esto no le es importante al usuario en las operaciones futuras.

\* En caso de que cambiaran los meseros esta parte del código cambiaría, lo cual es considerablemente extraño y de momento no es necesario.

\*/

**public** Restaurante() {

mesas = **new** Mesa[MAXR][MAXC];

mess = **new** Meseros [MAXR\*MAXC];

mess[0]=**new** Meseros("Frank");

mess[1]=**new** Meseros("Jaime");

mess[2]=**new** Meseros("Roberto");

mess[3]=**new** Meseros("Adrian");

mess[4]=**new** Meseros("Martín");

mess[5]=**new** Meseros("Efraín");

mess[6]=**new** Meseros("Nancy");

mess[7]=**new** Meseros("Ricky");

mess[8]=**new** Meseros("Lore");

}

/\*\*

\*

\* Un restaurante se construye con su nombre, el domicilio, el teléfono y el número de estrellas que tenga.

\* Se manda a llamar el constructor vacío.

\* Se inicicializan las ganancias del restaurante en 0.

\* Se utilizan las librarías "time" para darle función a la fecha y al horario, respectivamente.

\* Se inicializa un ArrayList con las cuentas del día

\* La variable booleana 'habil' se utiliza para evaluar si en el restaurante no se ha hecho el corte. Después de esto, no se podrá hacer ninguna función.

\*/

**public** Restaurante(String nombre, String domicilio, String telefono, **int** estrellas) {

**this**();

**this**.nombre = nombre;

**this**.domicilio = domicilio;

**this**.telefono = telefono;

**this**.estrellas = estrellas;

**this**.ganancias=0;

LocalDateTime tiempo = LocalDateTime.*now*();

**int** years = tiempo.getYear();

**int** months = tiempo.getMonthValue();

**int** days = tiempo.getDayOfMonth();

**int** hours = tiempo.getHour();

**int** minutes = tiempo.getMinute();

**int** seconds = tiempo.getMinute();

**this**.fecha=("Date: " + days+"/"+months+"/"+years + "/" );

**this**.horario=("Time: " + hours + ":" + minutes + ":" + seconds );

**this**.cuentas= **new** ArrayList<String>();

**this**.habil=**true**;

}

/\*\*

\*

\* Función que inicializa una mesa.

\* Para iniciar una mesa se necesita el nombre del usuario, el número de la mesa se le va a asignar por efectos prácticos y la coordenada de la mesa.

\* Primero se tiene que buscar un mesero desocupado para asignarle a la mesa.

\* Luego se tienen que validar las coordenadas de la mesa, así como que la coordenada esté desocupada y que el número asignado no esté siendo utilizado.

\* Si todas las condiciones se cumplen se iniciará una mesa con esas características y el mesero se mostrará como "ocupado".

\* Esta función regresa la mesa inicializada. En caso de que no se pudo hacer la inicialización, regresa una Mesa=null.

\*/

**public** Mesa iniciarMesa(String nombre, **int** numMesa, **int** numR, **int** numC) {

**boolean** res = **false**;

**int** i,j;

i=numR-1;

j=numC-1;

**int** k=0;

Mesa aux=**null**;

**while**(k<mess.length && mess[k].isOcupado()==**true**)

k++;

**if**(habil && i>=0 &&j >=0 && i<MAXR && j <MAXC && numMesa>=1 && numMesa<=9 && mesas[i][j]==**null** && MandejadorMatricesGenerico.*busquedaDesordenadaMesas*(mesas, numMesa)==**null**) {

aux = **new** Mesa (nombre, numMesa, mess[k]);

mesas[i][j]= aux;

res = **true**;

mess[k].setOcupado(**true**);

}

**return** aux;

}

/\*\*

\*

\* Con esta función el usuario puede leer la "hoja de reservaciones", a partir de un textDocument.

\* Esta función recibe como parámetro el nombre de la hoja y regresa un String con la información de las mesas reservadas.

\* Primero se crean un StringBuilder y un ArrayList de Mesas.

\* El archivo lee primero la cantidad de reservaciones y luego inicializa esa cantidad de mesas según la información del documento.

\* La información de cada mesa se agrega al ArrayList y para luego agregarla al StringBuilder y poderla regresar con un mensaje adecuado.

\*

\*/

**public** String leerHojaReservas() {

StringBuilder res = **new** StringBuilder();

ArrayList <Mesa > rsvp = **new** ArrayList<Mesa>();

**if**(habil) {

Scanner archivo;

Mesa aux;

**int** n;

**try** {

archivo = **new** Scanner (**new** File("reservas.txt"));

n = archivo.nextInt();

**for**(**int** i=0;i<n; i++) {

aux = iniciarMesa(archivo.next(),archivo.nextInt(), archivo.nextInt(), archivo.nextInt());

rsvp.add(aux);

}

res.append("<<Mesas inicializadas correctamente>>");

res.append("\nLas mesas inicializadas/reservadas son: ");

res.append(rsvp.toString());

}**catch**(FileNotFoundException fnfe) {

System.***out***.println(fnfe);

res.append(fnfe);

res.append("\n <<No se encontró el archivo>>");

res.append("\n <<Hubo un error en inicializar las mesas>>");

res.append("\n\n <<Asegurate de tener el archivo 'reservas.txt' guardado en el mismo lugar que tu \n programa>>");

//System.exit(-1);

}

}**else** {

res.append("<<Hubo un error en inicializar las mesas>>");

}

**return** res.toString();

}

/\*\*

\*

\* Función que toma el pedido de una mesa.

\* Esta función recibe como parámetros el número de la mesa, el platillo y le precio del platillo.

\* (Un programa de administración restaurantera suele tener una base de datos con todos los platillos que tiene, en este caso, es una cocina de autor,

\* lo que significa que los platillos nunca son los mismos y que el precio dependerá del día y del pedido).

\* Primero se tiene que validar que exista una mesa con ese número dentro del arreglo de mesas.

\* después se buscan las coordenadas de dicha mesa. Es importante confirmar que sean válidas.

\* Se toman en cuenta los errores como nullPointerException y que la mesa en cuestión sea la del arreglo.

\* Si es el caso, se invoca la función "tomarPedido" de la clase Mesa y se ejecuta con los parámetros de la función.

\* La función regresa un mensaje adecuado según el caso.

\*/

**public** String tomarElPedido(**int** numMesa, String platillo, **double** precio) {

String resp = "";

**int** [] coordenadas = **new** **int** [2];

**int** i,j;

Mesa otro = MandejadorMatricesGenerico.*busquedaDesordenadaMesas*(mesas, numMesa);

coordenadas = MandejadorMatricesGenerico.*busquedaDesordenada*(mesas, otro);

i=coordenadas[0];

j=coordenadas[1];

**if**(habil && i!=-1 && j!=-1) {

**if**(mesas[i][j]!=**null** && mesas[i][j].getNumMesa()==otro.getNumMesa()) {

mesas[i][j].tomarPedido(platillo, precio);

resp=("<<Se tomó el pedido correctamente>>\n");

}**else**

resp=("<<Hubo algún error al tomar el pedido>>");

}**else**

resp=("<<Hubo algún error al tomar el pedido>>");

**return** resp;

}

**public** String eliminarElPedido(**int** numMesa, String platillo, **double** precio) {

String resp = "";

**int** [] coordenadas = **new** **int** [2];

**int** i,j;

Mesa otro = MandejadorMatricesGenerico.*busquedaDesordenadaMesas*(mesas, numMesa);

coordenadas = MandejadorMatricesGenerico.*busquedaDesordenada*(mesas, otro);

i=coordenadas[0];

j=coordenadas[1];

**if**(habil && i!=-1 && j!=-1) {

**if**(mesas[i][j]!=**null** && mesas[i][j].getNumMesa()==otro.getNumMesa()) {

mesas[i][j].eliminarPedido(platillo, precio);

resp=("<<Se eliminó el pedido correctamente>>\n");

}**else**

resp=("<<Hubo algún error al eliminar el pedido>>");

}**else**

resp=("<<Hubo algún error al eliminar el pedido>>");

**return** resp;

}

/\*\*

\*

\* Función que imprime la cuenta de una mesa.

\* Esta función necesita como parámetros solo el número de la mesa.

\* Primero se evalúa la existencia de la mesa seguido de las coordenadas de la misma.

\* Si estas son válidas y no hay otros errores, se invoca la función "imprimirCuenta" de la clase Mesa

\* y se agrega al StringBuilder del resultado.

\* En caso de que la condiciones no se hayan cumplido correctamente, se regresa un mensaje adecuado.

\*/

**public** String imprimeCuenta(**int** numMesa) {

StringBuilder res = **new** StringBuilder();

**int** i,j;

**int** [] coordenadas = **new** **int** [2];

Mesa otro = MandejadorMatricesGenerico.*busquedaDesordenadaMesas*(mesas, numMesa);

coordenadas = MandejadorMatricesGenerico.*busquedaDesordenada*(mesas, otro);

i=coordenadas[0];

j=coordenadas[1];

**if**(habil && i!=-1 && j!=-1) {

**if**(mesas[i][j]!=**null** && mesas[i][j].getNumMesa()==otro.getNumMesa()) {

res.append("La CUENTA de la mesa " + otro.getNumMesa() + " es: \n");

res.append( mesas[i][j].imprimirCuenta());

}**else**

res.append("<<Hubo algún problema al imprimir la cuenta>>");

}**else**

res.append("<<Hubo algún problema al imprimir la cuenta>>");

**return** res.toString();

}

/\*\*

\*

\* Esta función pagará una cuenta desocupando una mesa o en otras palabras eliminándola de la matriz.

\* Recibe como parámetros la cantidad de dinero con la que el usuario pagará su cuenta, el porcentaje de propina que dejará y el número de la mesa en cuestión.

\* Primero se evalúa la existencia de la mesa y de sus coordenadas.

\* Se calcula la propina del mesero en relación con el total de la cuenta.

\* Se hacen las evaluaciones pertinentes de errores básicos, así como que el "pago" sea suficiente para pagar la cuenta y dar la propina.

\* Si es el caso, se calcula el cambio del usuario y se le suma el total de la cuenta como ganancia para el restaurante.

\* Se agrega al ArrayList de pedidos del día, la información de la mesa que se está pagando, para tener el registro.

\* Al mesero se le suma su propina y se le declara como "desocupado".

\* Seguido a esto se elimina le mesa de la matriz declarándola como null, en otras palabras, desocupándola.

\* Si todo lo anterior funcionó correctamente, se regresa un mensaje adecuado junto con la hora a la que se desocupó.

\* Caso contrario, regresa un mensaje de problema adecuado.

\*

\*/

**public** String pagarCuenta(**double** pago, **int** propina, **int** numMesa) {

StringBuilder cad = **new** StringBuilder();

**boolean** res = **false**;

**double** cambio =0;

**double** tips;

String cuentitas = "";

**int** [] coordenadas = **new** **int** [2];

**int** i,j;

Mesa otro = MandejadorMatricesGenerico.*busquedaDesordenadaMesas*(mesas, numMesa);

coordenadas = MandejadorMatricesGenerico.*busquedaDesordenada*(mesas, otro);

tips = otro.calculaTotal()\*propina/100;

i=coordenadas[0];

j=coordenadas[1];

**if**(habil && i!=-1 && j!=-1) {

**if**(mesas[i][j]!=**null** && mesas[i][j].isPedidoTomado() && mesas[i][j].getNumMesa()==otro.getNumMesa() && pago>=(mesas[i][j].calculaTotal()+tips)) {

cambio = pago-(mesas[i][j].calculaTotal()+tips);

ganancias+=mesas[i][j].calculaTotal();

cuentitas=("Mr. " + otro.getNombre() + ", mesa:" +otro.getNumMesa() + ", cuenta: $" + otro.calculaTotal() + ", mesero: " + otro.getmes().getNombre() + ", hora: " + horario + ", FOLIO: " + otro.getNumPedido() + "\n");

cuentas.add(cuentitas);

otro.getmes().setPropinas((otro.getmes().getPropinas()+tips));

otro.getmes().setOcupado(**false**);

mesas[i][j]=**null**;

res=**true**;

cad.append("La mesa #" + otro.getNumMesa() + " se pagó y desocupó correctamente correctamente. mat\n" +"El cambio del señor " + otro.getNombre() + " es: $" + cambio + "\n");

cad.append(horario + "\n");

}**else**

cad.append("<<Hubo algún problema al pagar la cuenta>>");

}**else**

cad.append("<<Hubo algún problema al buscar la mesa>>");

**return** cad.toString();

}

/\*\*

\*

\* La función de hacer el corte, regresa toda la información del día del restaurante y de los meseros.

\* Primero es importante evaluar que le matriz de mesas esté vacía, pues si hay una mesa habilitada, no se puede hacer el corte.

\* Si todas las mesas están desocupadas, se agrega a un StringBuilder toda la información pertinente.

\* Por último el atributo de la clase habil se convertirá en falso.

\* Esto deshabilitará TODAS las funciones del programa dando así el corte definitivo del día.

\*/

**public** String hacerCorte() {

StringBuilder cad = **new** StringBuilder();

**if**(habil && MandejadorMatricesGenerico.*matrizVacia*(mesas)){

cad.append("\nLa información del restaurante es:");

cad.append("\n" + toString());

cad.append("\n");

cad.append("\nLa información de los pedidos de hoy es: \n");

cad.append(cuentas.toString() + "\n");

cad.append("\n");

cad.append("\nEl ingreso del restaurante del día fue de: $" + ganancias);

cad.append("\n");

cad.append("\n PROPINAS:");

**for** (**int** i = 0; i<mess.length; i++)

cad.append("\n"+mess[i].toString());

habil = **false**;

}**else**

cad.append("<<No se puedo hacer el corte>>");

**return** cad.toString();

}

/\*\*

\* toString

\*/

**public** String toString() {

StringBuilder cad = **new** StringBuilder();

cad.append(fecha + " && " + horario);

cad.append("\nEl nombre del restaurante es: "+nombre);

cad.append("\nEl domicilio es "+domicilio);

cad.append("\nEl numero de telefono es "+telefono);

cad.append("\nEl numero de estrellas es "+estrellas);

**for**(**int** i = 0; i < MAXR; i++)

**for**(**int** j = 0; j < MAXC; j++)

**if**(mesas[i][j] != **null**) {

cad.append("\n= " + mesas[i][j].toString());

cad.append("\nEl total es de: $" + mesas[i][j].calculaTotal());

}

**return** cad.toString();

}

}

## RestauranteVista

**package** librerias;

/\*\*

\*

\* **@author** Mauricio Verduzco Chavira

\* **@version** 2.0

\* **@Fecha**: 06/12/2020

\* **@Clase**: RestauranteVista

\* **@Descripción**: Clase con extends JFrame para crear la vista gráfica de la intefaz.

\*

\*/

/\*\*

\* Primero se importan todas las librerías pertinentes: java.awt y javax.swing

\*/

**import** java.awt.EventQueue;

**import** javax.swing.JFrame;

**import** javax.swing.JScrollPane;

**import** javax.swing.JTextArea;

**import** javax.swing.JLabel;

**import** javax.swing.JTextField;

**import** javax.swing.JButton;

**import** javax.swing.JPanel;

**import** javax.swing.JSeparator;

**import** javax.swing.SwingConstants;

**import** java.awt.Color;

**import** java.awt.Font;

**import** java.awt.SystemColor;

**import** javax.swing.JComboBox;

**import** javax.swing.ImageIcon;

**import** javax.swing.JScrollBar;

**import** javax.swing.DropMode;

**import** java.awt.Toolkit;

**public** **class** RestauranteVista **extends** JFrame{

/\*\*

\* Todas las futuras partes del interfaz como atributos de la clase. TextFields, Labels, Buttons, JFrame y TextArea.

\*/

**private** JFrame frmShnippenketer;

**protected** JTextField txNomTit, txNumMesa, txRow, txCol, txPlatillo, txPrecio, txPropina, txPago;

**protected** JButton btnIniciarMesa, btnTomarPedido, btnImprimirCuenta, btnPagarCuenta, btnHacerCorte, btnLeerReservas, btnClear, btnInformacion, btnEliminarPedido;

**private** JLabel lblInicializacin, lblNombreDelTitular, lblNoMesa, lblNoCol, lblPago, lblNoRow, lblPedido, lblPlatillo, lblPrecio, lblFilipinos, lblPagar, lblPropina;

**protected** JTextArea textArea;

//protected JScrollPane infoScrol;

/\*\*

\* Con este constructor se "crea" la aplicaición.

\*/

**public** RestauranteVista(String titulo) {

**super**(titulo);

/\*\*

\* Se inicializan TODOS los contenidos de la aplicación.

\* En este caso decidí no estar un panel e ingresar todos mis atributos como parte del JFrame.

\* Para esto, di las coordenadas y el tamaño de cada objeto.

\*/

/\*\*

\* Primero el JFrame

\*/

frmShnippenketer = **new** JFrame();

frmShnippenketer.setIconImage(Toolkit.*getDefaultToolkit*().getImage(RestauranteVista.**class**.getResource("/com/sun/javafx/scene/control/skin/caspian/pattern-transparent.png")));

frmShnippenketer.setTitle("Shnippenk\u00F6eter");

/\*\*

\* Labels

\*/

lblInicializacin = **new** JLabel("de la inicializaci\u00F3n");

lblInicializacin.setFont(**new** Font("Times New Roman", Font.***ITALIC***, 12));

lblInicializacin.setBounds(36, 121, 94, 16);

frmShnippenketer.getContentPane().add(lblInicializacin);

lblNombreDelTitular = **new** JLabel("Nombre del Titular");

lblNombreDelTitular.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 14));

lblNombreDelTitular.setBounds(79, 186, 116, 16);

frmShnippenketer.getContentPane().add(lblNombreDelTitular);

lblNoMesa = **new** JLabel("No. Mesa");

lblNoMesa.setFont(**new** Font("Calibri", Font.***BOLD***, 14));

lblNoMesa.setBounds(521, 71, 56, 16);

frmShnippenketer.getContentPane().add(lblNoMesa);

lblNoRow = **new** JLabel("No. Row:");

lblNoRow.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 14));

lblNoRow.setBounds(276, 153, 56, 16);

frmShnippenketer.getContentPane().add(lblNoRow);

lblNoCol = **new** JLabel("No. Col:");

lblNoCol.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 14));

lblNoCol.setBounds(276, 199, 56, 16);

frmShnippenketer.getContentPane().add(lblNoCol);

lblPedido = **new** JLabel("del pedido");

lblPedido.setFont(**new** Font("Times New Roman", Font.***ITALIC***, 12));

lblPedido.setBounds(36, 261, 106, 16);

frmShnippenketer.getContentPane().add(lblPedido);

lblPlatillo = **new** JLabel("Platillo");

lblPlatillo.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 14));

lblPlatillo.setBounds(117, 318, 56, 16);

frmShnippenketer.getContentPane().add(lblPlatillo);

lblPrecio = **new** JLabel("Precio");

lblPrecio.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 14));

lblPrecio.setBounds(316, 318, 56, 16);

frmShnippenketer.getContentPane().add(lblPrecio);

lblPropina = **new** JLabel("% Propina");

lblPropina.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 14));

lblPropina.setBounds(103, 444, 67, 16);

frmShnippenketer.getContentPane().add(lblPropina);

lblPago = **new** JLabel("Pago");

lblPago.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 14));

lblPago.setBounds(321, 444, 39, 16);

frmShnippenketer.getContentPane().add(lblPago);

lblPagar = **new** JLabel("del pago");

lblPagar.setFont(**new** Font("Times New Roman", Font.***ITALIC***, 12));

lblPagar.setBounds(36, 386, 56, 16);

frmShnippenketer.getContentPane().add(lblPagar);

lblFilipinos = **new** JLabel("Philp\u00B4s House of Gold");

lblFilipinos.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 45));

lblFilipinos.setBounds(36, 43, 409, 41);

frmShnippenketer.getContentPane().add(lblFilipinos);

/\*\*

\* TextBoxes

\*/

txNomTit = **new** JTextField();

txNomTit.setFont(**new** Font("Times New Roman", Font.***PLAIN***, 14));

txNomTit.setBackground(Color.***WHITE***);

txNomTit.setForeground(Color.***BLACK***);

txNomTit.setBounds(79, 163, 116, 22);

frmShnippenketer.getContentPane().add(txNomTit);

txNomTit.setColumns(10);

txNumMesa = **new** JTextField();

txNumMesa.setFont(**new** Font("Times New Roman", Font.***PLAIN***, 14));

txNumMesa.setBackground(Color.***WHITE***);

txNumMesa.setBounds(589, 54, 50, 50);

frmShnippenketer.getContentPane().add(txNumMesa);

txNumMesa.setColumns(10);

txRow = **new** JTextField();

txRow.setFont(**new** Font("Times New Roman", Font.***PLAIN***, 14));

txRow.setBackground(Color.***WHITE***);

txRow.setForeground(Color.***BLACK***);

txRow.setBounds(335, 150, 60, 22);

frmShnippenketer.getContentPane().add(txRow);

txRow.setColumns(10);

txCol = **new** JTextField();

txCol.setFont(**new** Font("Times New Roman", Font.***PLAIN***, 14));

txCol.setBackground(Color.***WHITE***);

txCol.setForeground(Color.***BLACK***);

txCol.setBounds(335, 196, 60, 22);

frmShnippenketer.getContentPane().add(txCol);

txCol.setColumns(10);

txPlatillo = **new** JTextField();

txPlatillo.setFont(**new** Font("Times New Roman", Font.***PLAIN***, 14));

txPlatillo.setBackground(Color.***WHITE***);

txPlatillo.setForeground(Color.***BLACK***);

txPlatillo.setBounds(79, 294, 116, 22);

frmShnippenketer.getContentPane().add(txPlatillo);

txPlatillo.setColumns(10);

txPrecio = **new** JTextField();

txPrecio.setFont(**new** Font("Times New Roman", Font.***PLAIN***, 14));

txPrecio.setBackground(Color.***WHITE***);

txPrecio.setForeground(Color.***BLACK***);

txPrecio.setBounds(276, 294, 116, 22);

frmShnippenketer.getContentPane().add(txPrecio);

txPrecio.setColumns(10);

txPropina = **new** JTextField();

txPropina.setFont(**new** Font("Times New Roman", Font.***PLAIN***, 14));

txPropina.setBackground(Color.***WHITE***);

txPropina.setForeground(Color.***BLACK***);

txPropina.setBounds(79, 418, 116, 22);

frmShnippenketer.getContentPane().add(txPropina);

txPropina.setColumns(10);

txPago = **new** JTextField();

txPago.setFont(**new** Font("Times New Roman", Font.***PLAIN***, 14));

txPago.setBackground(Color.***WHITE***);

txPago.setForeground(Color.***BLACK***);

txPago.setBounds(276, 418, 116, 22);

frmShnippenketer.getContentPane().add(txPago);

txPago.setColumns(10);

/\*\*

\* Buttons

\*/

btnIniciarMesa = **new** JButton("Iniciar Mesa");

btnIniciarMesa.setBackground(Color.***WHITE***);

btnIniciarMesa.setForeground(SystemColor.***infoText***);

btnIniciarMesa.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 18));

btnIniciarMesa.setBounds(521, 220, 152, 47);

frmShnippenketer.getContentPane().add(btnIniciarMesa);

btnTomarPedido = **new** JButton("Tomar Pedido");

btnTomarPedido.setBackground(Color.***WHITE***);

btnTomarPedido.setForeground(SystemColor.***infoText***);

btnTomarPedido.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 18));

btnTomarPedido.setBounds(521, 302, 152, 47);

frmShnippenketer.getContentPane().add(btnTomarPedido);

btnImprimirCuenta = **new** JButton("Imprimir Cuenta");

btnImprimirCuenta.setBackground(Color.***WHITE***);

btnImprimirCuenta.setForeground(SystemColor.***infoText***);

btnImprimirCuenta.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 18));

btnImprimirCuenta.setBounds(521, 464, 152, 47);

frmShnippenketer.getContentPane().add(btnImprimirCuenta);

btnPagarCuenta = **new** JButton("Pagar Cuenta");

btnPagarCuenta.setBackground(Color.***WHITE***);

btnPagarCuenta.setForeground(SystemColor.***infoText***);

btnPagarCuenta.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 18));

btnPagarCuenta.setBounds(521, 542, 152, 47);

frmShnippenketer.getContentPane().add(btnPagarCuenta);

btnHacerCorte = **new** JButton("CORTE");

btnHacerCorte.setIcon(**new** ImageIcon(RestauranteVista.**class**.getResource("/com/sun/javafx/scene/control/skin/modena/HTMLEditor-Cut-Black.png")));

btnHacerCorte.setToolTipText("");

btnHacerCorte.setFont(**new** Font("Calibri", Font.***PLAIN***, 30));

btnHacerCorte.setBackground(Color.***WHITE***);

btnHacerCorte.setForeground(Color.***BLACK***);

btnHacerCorte.setBounds(59, 496, 146, 92);

frmShnippenketer.getContentPane().add(btnHacerCorte);

btnLeerReservas = **new** JButton("R.S.V.P.");

btnLeerReservas.setBackground(Color.***WHITE***);

btnLeerReservas.setForeground(SystemColor.***infoText***);

btnLeerReservas.setFont(**new** Font("Calibri", Font.***PLAIN***, 30));

btnLeerReservas.setBounds(292, 496, 146, 92);

frmShnippenketer.getContentPane().add(btnLeerReservas);

btnClear = **new** JButton("Clear");

btnClear.setBackground(Color.***WHITE***);

btnClear.setForeground(Color.***BLACK***);

btnClear.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 18));

btnClear.setBounds(521, 137, 152, 47);

frmShnippenketer.getContentPane().add(btnClear);

btnInformacion = **new** JButton("");

btnInformacion.setBackground(Color.***WHITE***);

btnInformacion.setIcon(**new** ImageIcon(RestauranteVista.**class**.getResource("/javax/swing/plaf/metal/icons/Inform.gif")));

btnInformacion.setBounds(1151, 605, 50, 50);

frmShnippenketer.getContentPane().add(btnInformacion);

btnEliminarPedido = **new** JButton("Eliminar Pedido");

btnEliminarPedido.setBackground(Color.***WHITE***);

btnEliminarPedido.setFont(**new** Font("Calibri Light", Font.***PLAIN***, 18));

btnEliminarPedido.setBounds(521, 381, 152, 47);

frmShnippenketer.getContentPane().add(btnEliminarPedido);

/\*\*

\* Los separadores funcionan como parte de la estética de la app y no tienen que formar parte de los atributos

\*/

JSeparator separator = **new** JSeparator();

separator.setForeground(Color.***BLACK***);

separator.setBounds(36, 276, 430, 2);

frmShnippenketer.getContentPane().add(separator);

JSeparator separator\_1 = **new** JSeparator();

separator\_1.setForeground(Color.***BLACK***);

separator\_1.setBounds(36, 400, 430, 2);

frmShnippenketer.getContentPane().add(separator\_1);

JSeparator separator\_2 = **new** JSeparator();

separator\_2.setBounds(479, 196, 0, 384);

frmShnippenketer.getContentPane().add(separator\_2);

JSeparator separator\_3 = **new** JSeparator();

separator\_3.setBounds(36, 470, 430, -7);

frmShnippenketer.getContentPane().add(separator\_3);

JSeparator separator\_5 = **new** JSeparator();

separator\_5.setBounds(36, 328, 424, -25);

frmShnippenketer.getContentPane().add(separator\_5);

JSeparator separator\_6 = **new** JSeparator();

separator\_6.setForeground(Color.***BLACK***);

separator\_6.setBounds(36, 135, 430, 2);

frmShnippenketer.getContentPane().add(separator\_6);

/\*\*

\* TextArea

\*/

textArea = **new** JTextArea();

textArea.setFont(**new** Font("Calibri", Font.***PLAIN***, 14));

textArea.setForeground(Color.***BLACK***);

textArea.setBackground(Color.***WHITE***);

textArea.setBounds(685, 13, 501, 642);

frmShnippenketer.getContentPane().add(textArea);

/\*\*

\* Características del JFrame

\*/

frmShnippenketer.getContentPane().setForeground(Color.***BLACK***);

frmShnippenketer.getContentPane().setFont(**new** Font("Calibri", Font.***PLAIN***, 30));

frmShnippenketer.getContentPane().setBackground(Color.***WHITE***);

frmShnippenketer.getContentPane().setLayout(**null**);

frmShnippenketer.setBounds(100, 100, 1231, 715);

frmShnippenketer.setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);

frmShnippenketer.setVisible(**true**);

}

/\*\*

\* Para correr la aplicación

\*/

**public** **static** **void** main(String[] args) {

RestauranteVista prueba = **new** RestauranteVista("Shippenköeter");

}

}

## RestauranteControlador

**package** librerias;

/\*\*

\*

\* **@author** Mauricio Verduzco Chavira

\* **@version** 2.0

\* **@Fecha**: 06/12/2020

\* **@Clase**: RestauranteControlador

\* **@Descripción**: Clase con extends a la vista para darle función a todos los botones y operaciones.

\*

/\*\*

\* Se importan todas las librerias

\*/

**import** java.awt.event.\*;

**import** javax.swing.\*;

**import** librerias.\*;

/\*\*

\*

\* Se crea la clase Controlador con un extends de la Vista

\*/

**public** **class** RestauranteControlador **extends** RestauranteVista{

/\*\*

\* Se crean un restaurante con los atributos pertinentes.

\*/

Restaurante r = **new** Restaurante("Filipinos", "Av. Naciones Unidas 7275", "3337275007", 5);

/\*\*

\*Se crea el controlador con su nombre y todos los botones con sus rrespectivos ActionListeners

\*/

**public** RestauranteControlador (String t) {

**super**(t);

**this**.btnIniciarMesa.addActionListener(**new** EscuchadorIniciarMesa());// Inicializa una mesa

**this**.btnLeerReservas.addActionListener(**new** EscuchadorLeerReservas());// Lee las reservaciones

**this**.btnTomarPedido.addActionListener(**new** EscuchadorTomarPedido());//Toma el pedido

**this**.btnImprimirCuenta.addActionListener(**new** EscuchadorImprimirCuenta());//Imprime la cuenta

**this**.btnPagarCuenta.addActionListener(**new** EscuchadorPagarCuenta());//Paga la cuenta

**this**.btnHacerCorte.addActionListener(**new** EscuchadorHacerCorte());//Hace el corte

**this**.btnClear.addActionListener(**new** EscuchadorClear());//Limpia los campos

**this**.btnInformacion.addActionListener(**new** EscuchadorInformacion());//Da información del programa

**this**.btnEliminarPedido.addActionListener(**new** EscuchadorEliminarPedido());//Elimina un pedido

}

/\*\*

\*

\* Para iniciar una mesa se tienen que tomar los valores de los textBoxes y convertirlos en el tipo de variable correspondiente.

\* Se inicializa la mesa con la función de restaurante y se evalúa la inicialización para imprimir un mensaje adecuado

\*/

**public** **class** EscuchadorIniciarMesa **implements** ActionListener{

**public** **void** actionPerformed(ActionEvent ae) {

Mesa resp;

String nomTit;

**int** numMesa, row, col;

nomTit = txNomTit.getText();

numMesa = Integer.*valueOf*(txNumMesa.getText());

row = Integer.*valueOf*(txRow.getText());

col = Integer.*valueOf*(txCol.getText());

resp = r.iniciarMesa(nomTit, numMesa, row, col);

**if**(resp!=**null**)

textArea.setText("<<Mesa inicializada correctamente>> \n " + resp.toString());

**else**

textArea.setText("<<Sucedió algún error al inicializar la mesa>>");

}

}

/\*\*

\*

\* Se lee la hoja de reservaciones e imprime el mensaje adecuado.

\*/

**public** **class** EscuchadorLeerReservas **implements** ActionListener{

**public** **void** actionPerformed(ActionEvent ae) {

String resp;

resp = r.leerHojaReservas();

**if** (resp!="")

textArea.setText(resp);

**else**

textArea.setText("<<Error al leer las hojas de reservaciones");

}

}

/\*\*

\*

\* Convierte la información de los textBoxes y la usa para tomar un pedido, imprime el mensaje adecuado.

\*/

**public** **class** EscuchadorTomarPedido **implements** ActionListener{

**public** **void** actionPerformed(ActionEvent ae) {

String resp, platillo;

**double** precio;

**int** numMesa;

numMesa = Integer.*valueOf*(txNumMesa.getText());

platillo = txPlatillo.getText();

precio = Double.*valueOf*(txPrecio.getText());

resp = r.tomarElPedido(numMesa, platillo, precio);

textArea.setText(resp);

}

}

/\*\*

\* Registra la información de los textBoxes como el pedido que se quiere eliminar, imprime un mensaje adecuado

\*/

**public** **class** EscuchadorEliminarPedido **implements** ActionListener{

**public** **void** actionPerformed(ActionEvent ae) {

String resp, platillo;

**double** precio;

**int** numMesa;

numMesa = Integer.*valueOf*(txNumMesa.getText());

platillo = txPlatillo.getText();

precio = Double.*valueOf*(txPrecio.getText());

resp = r.eliminarElPedido(numMesa, platillo, precio);

textArea.setText(resp);

}

}

/\*\*

\*

\* Convierte la información de los textBoxes y la usa para imprimir la cuenta.

\*/

**public** **class** EscuchadorImprimirCuenta **implements** ActionListener{

**public** **void** actionPerformed(ActionEvent ae) {

String resp;

**int** numMesa;

numMesa = Integer.*valueOf*(txNumMesa.getText());

resp = r.imprimeCuenta(numMesa);

textArea.setText(resp);

}

}

/\*\*

\*

\* Convierte la información de los textBoxes y la usa para pagar la cuenta.

\*/

**public** **class** EscuchadorPagarCuenta **implements** ActionListener{

**public** **void** actionPerformed(ActionEvent ae) {

String resp;

**int** numMesa, propina, pago;

numMesa = Integer.*valueOf*(txNumMesa.getText());

propina = Integer.*valueOf*(txPropina.getText());

pago = Integer.*valueOf*(txPago.getText());

resp = r.pagarCuenta(pago, propina, numMesa);

textArea.setText(resp);

}

}

/\*\*

\*

\* Para hacer el corte no se necesita ningún parámetro.

\*/

**public** **class** EscuchadorHacerCorte **implements** ActionListener{

**public** **void** actionPerformed(ActionEvent ae) {

String resp;

resp = r.hacerCorte();

textArea.setText(resp);

}

}

/\*\*

\*

\* El escuchador clear declara todos los textBoxes como "", es decir los vacía o los limpia.

\*/

**public** **class** EscuchadorClear **implements** ActionListener{

**public** **void** actionPerformed(ActionEvent ae) {

txNomTit.setText("");

txNumMesa.setText("");

txRow.setText("");

txCol.setText("");

txPlatillo.setText("");

txPrecio.setText("");

txPropina.setText("");

txPago.setText("");

textArea.setText("");

}

}

/\*\*

\*

\* El botón de información, ubicado en la esquina inferior derecha, imprime toda la información necesaria para que el usuario puede hacer un uso adecuado del programa.

\*

\*/

**public** **class** EscuchadorInformacion **implements** ActionListener{

**public** **void** actionPerformed(ActionEvent ae) {

StringBuilder resp = **new** StringBuilder();

resp.append("\nBienvenid@ al programa para administrar el restaurante Philp's House of Gold");

resp.append("\n- Primero deberá usar el boton R.S.V.P., para dejar esas mesas apartadas.");

resp.append("\n- Acto seguido puede inciar distintas mesas.");

resp.append("\n- A una mesa incializada le puede tomar el pedido según los recuadros correspondientes.");

resp.append("\n- Se puede eliminar un pedido siempre y cuando exista el pedido y se llenen los recuadros.");

resp.append("\n- Cuando a una mesa le haya tomado el pedido, puede también imprimir la cuenta.");

resp.append("\n- Una mesa con el pedido tomado puede pagar su cuenta.");

resp.append("\n- Para pagar una cuenta indique el porcentaje de propina y con cuanto dinero pagará la \n cuenta.");

resp.append("\n- Una mesa pagada se desocupa y puede volver a usarse.");

resp.append("\n- Al terminar la jornada, use el botón de 'corte' para terminar el día.");

resp.append("\n\nNOTAS:");

resp.append("\n\*Todas las mesas apartadas, ya están inicializadas.");

resp.append("\n\*Para inicializar una mesa no pueden usar coordenadas ocupadas ni números oupados.");

resp.append("\n\*Mínimo(1,1)-Máximo(3,3) & Número máximo para une mesa: 9");

resp.append("\n\*No se puede eliminar un platillo que no existe. ");

resp.append("\n\*Usted no podrá ver la cuenta de una mesa, si a esta no se le ha tomado el pedido.");

resp.append("\n\*Usted no podrá pargar la cuenta de una mesa, si a esta no se le ha tomado el pedido.");

resp.append("\n\*No se podrá hacer el corte si tiene mesas inicializadas.");

resp.append("\n\*Después de hacer el corte, no podrá seguir manejando el programa");

resp.append("\n\*Puede apoyarse del boton 'clear' para limpiar los campos más rapidamente.");

resp.append("\n\*Recuerde que el recuadro 'No. Mesa' debe ser usado para cualquier función referente a \n una mesa.");

textArea.setText(resp.toString());

}

}

/\*\*

\*

\* Para ejecutar la app finalizada.

\*/

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

RestauranteControlador Final = **new** RestauranteControlador("Shippenköeter");

}

}

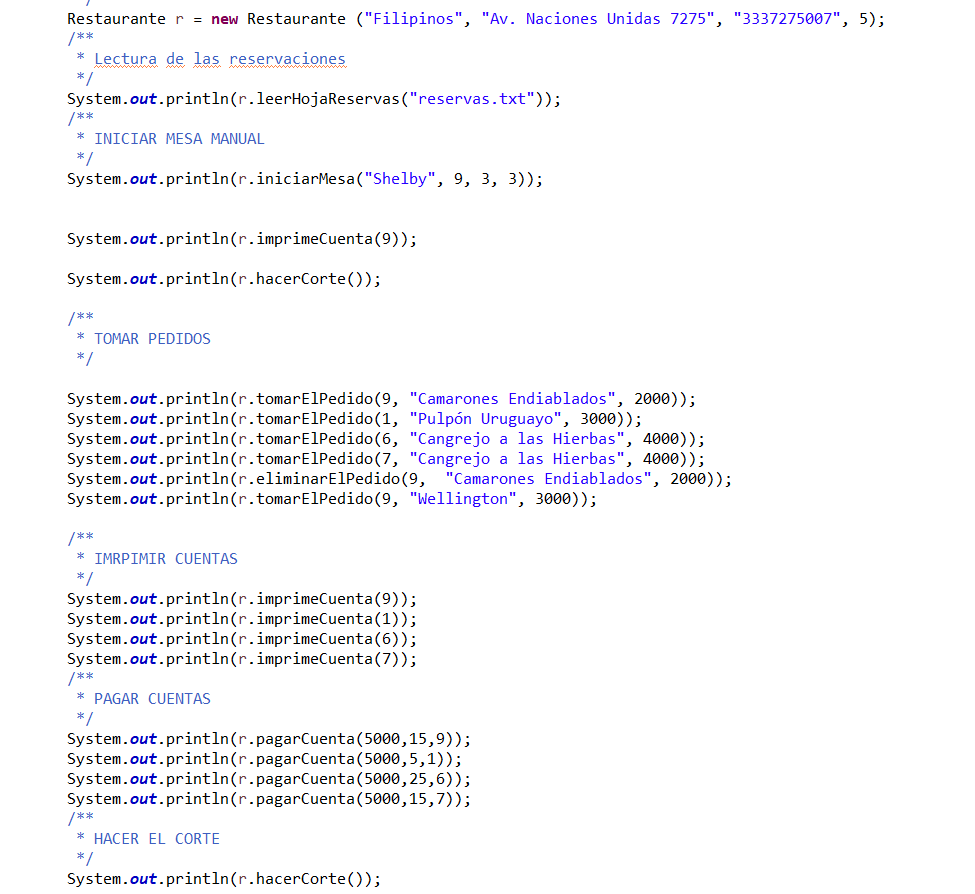
# Pruebas y Resultados

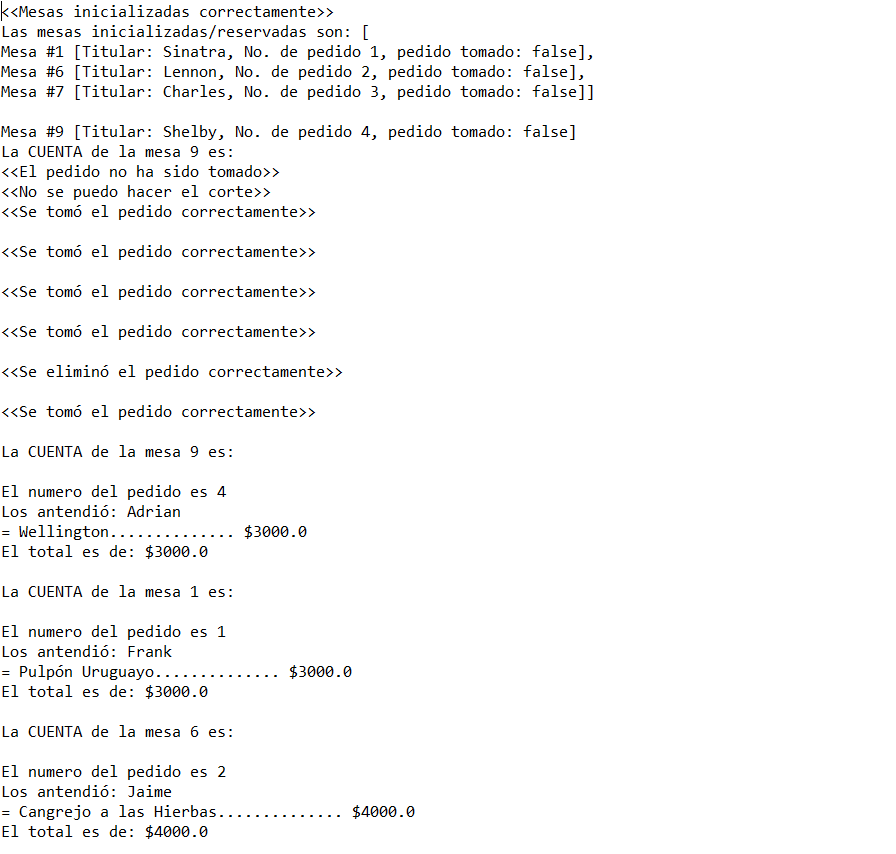
Para las pruebas primero creé una clase “pruebaRestaurante”. En la cual utilizo todas mis funciones. A continuación, explico y adjunto evidencia de dos posibles escenarios con la clase de prueba y otras cuatro con la interfaz.

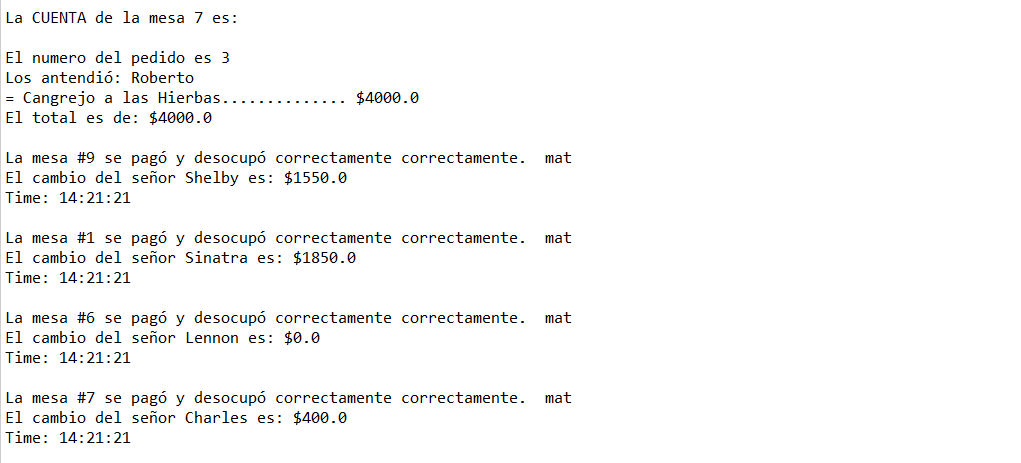
## Primera prueba

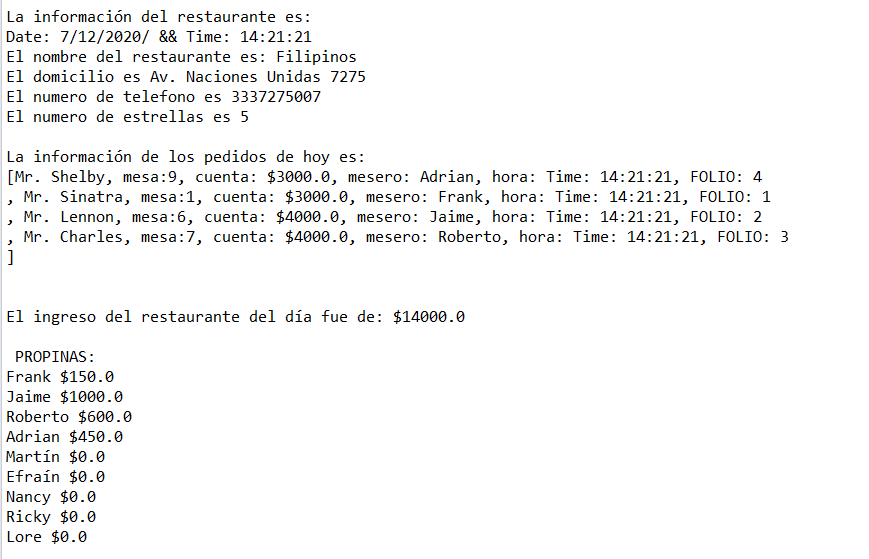
En este primero escenario procuré usar todas mis funciones y utilizando información adecuada para probar su funcionamiento óptimo.

Primero creé un restaurante, luego leí la hoja de reservaciones. A continuación, inicié una mesa manualmente. Tomé el pedido de todas las mesas, eliminé uno, imprimí sus cuentas y las pagué. Por último, hice el corte.





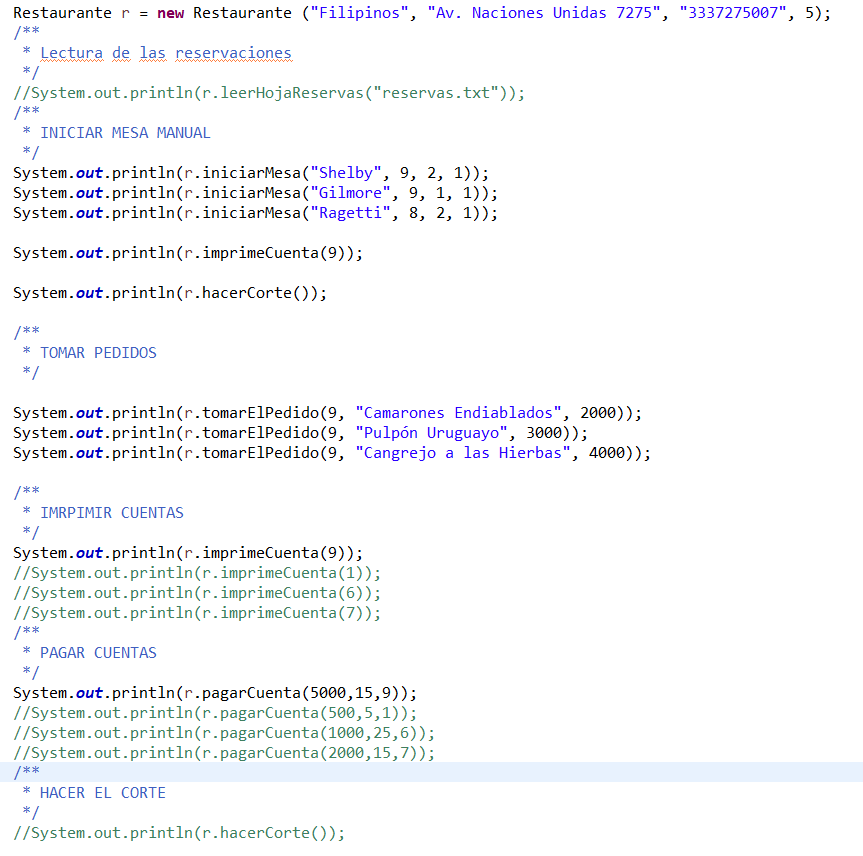


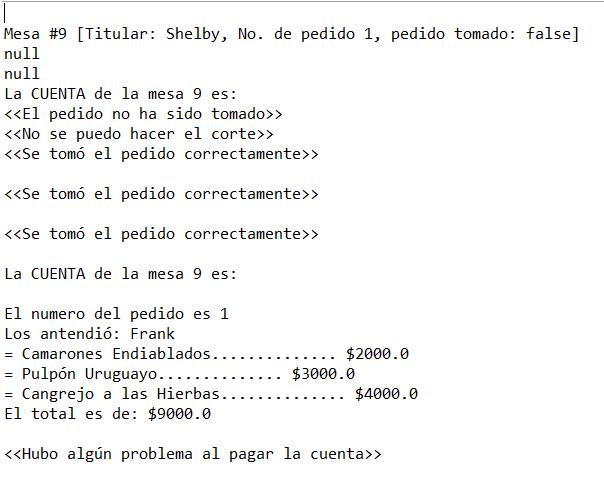


En estos resultados podemos observar cómo todo corrió como debería. Con mensajes adecuados y fáciles de entender. Desde crear el restaurante, leer reservaciones, dar de alta una mesa, tomar y eliminar pedidos, imprimir cuentas pagarlas y hacer el corte.

## Segunda prueba

En esta ocasión me aseguré de que ninguna de mis funciones pudiera correr correctamente de acuerdo con lo ingresado, como iniciar mesas ocupadas, hacer el corte antes de tiempo, imprimir la cuenta de una mesa a la que no se le ha tomado el pedido e intentar pagar una cuenta con insuficiente dinero.

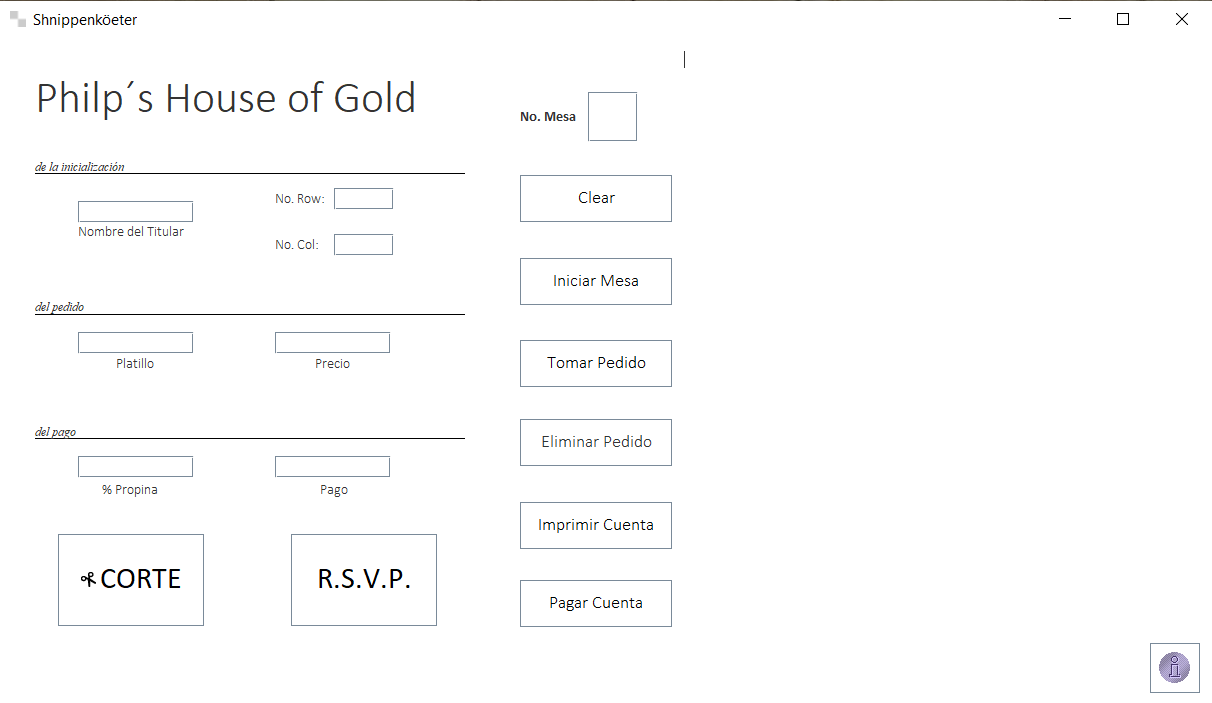


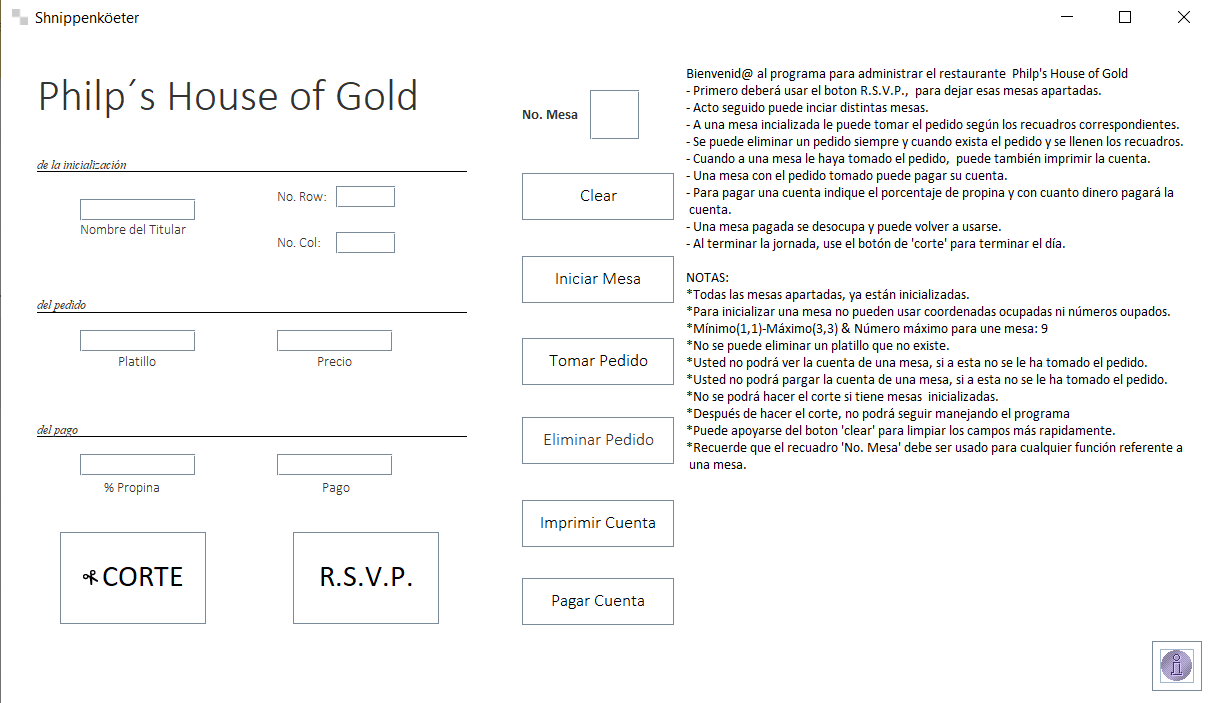


Podemos ver cómo en esta ocasión como la segunda y la tercera mesa no se pudieron inicializar, pues el número de la mesa y la coordenada estaban ocupadas. Luego cuando intentamos imprimir la cuenta de la mesa nos deja en claro que no ha sido tomado el pedido. Acto seguido el corte no se puede hacer, pues siguen mesas habilitadas. Para finalizar se tomó la cuenta de tres platillos que suman una cuenta de $9000 y se intentó pagar esa cuanta +15% de propina con tan solo $5000. Cosa, que como nos dice nuestro ultimo mensaje, no fue posible.

## Tercera prueba

En las pruebas de la vista es complejo demostrar varios eventos simultáneamente. En este caso mostraré la vista original y el botón de dudas.

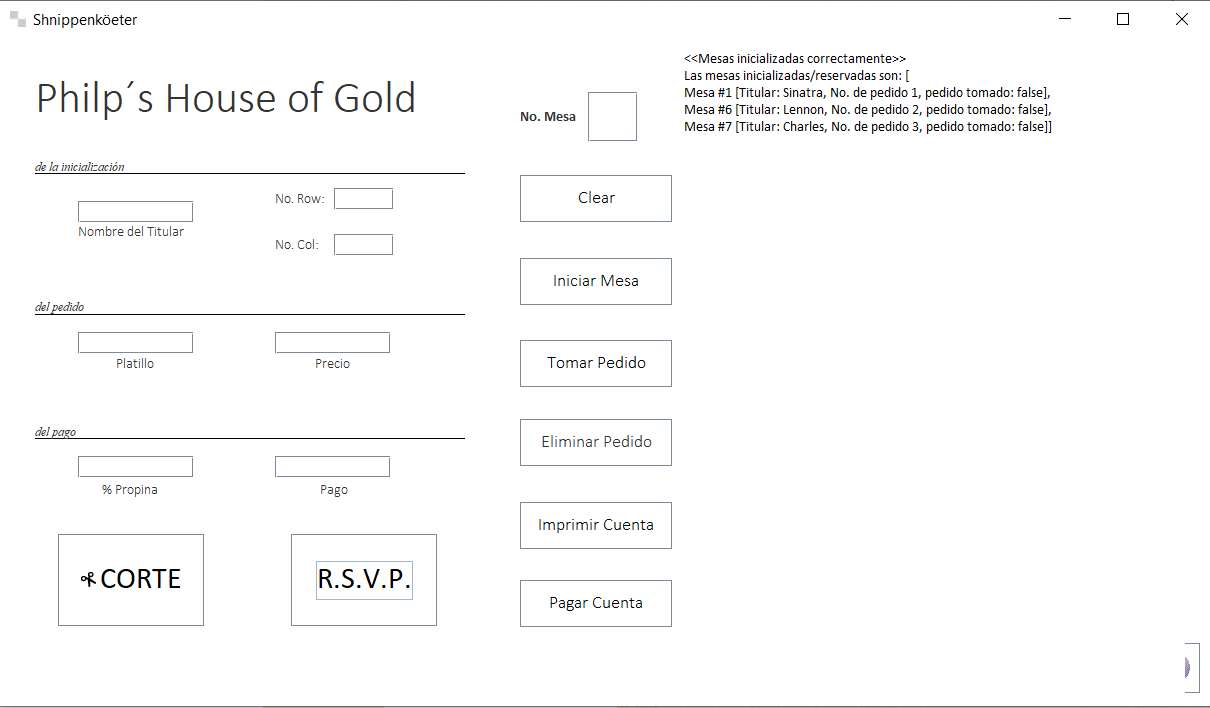


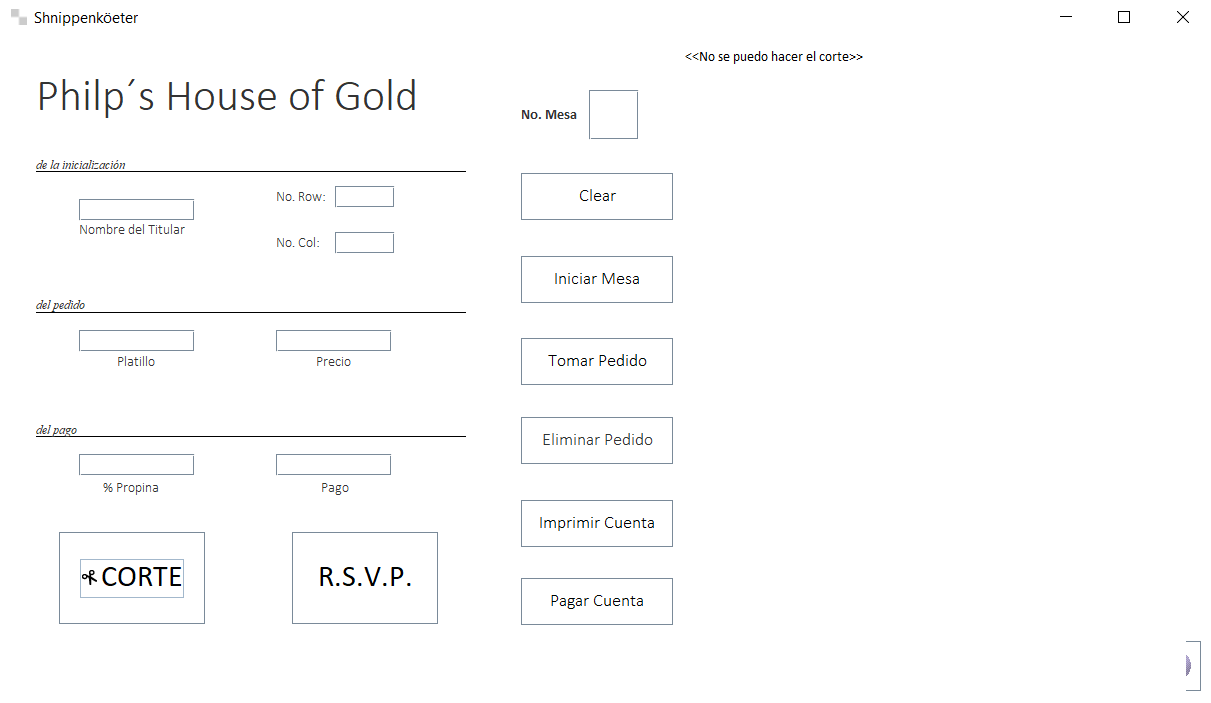


Podemos observar como el botón ubicado en la parte inferior derecha le despliega a usuario una pequeña guía del funcionamiento del programa, así como las cosas que debe evitar hacer.

## Cuarta prueba

En esta ocasión primero uso el botón de R.S.V.P. para leer las reservaciones y luego intento hacer el corte con las mesas todavía ocupadas.

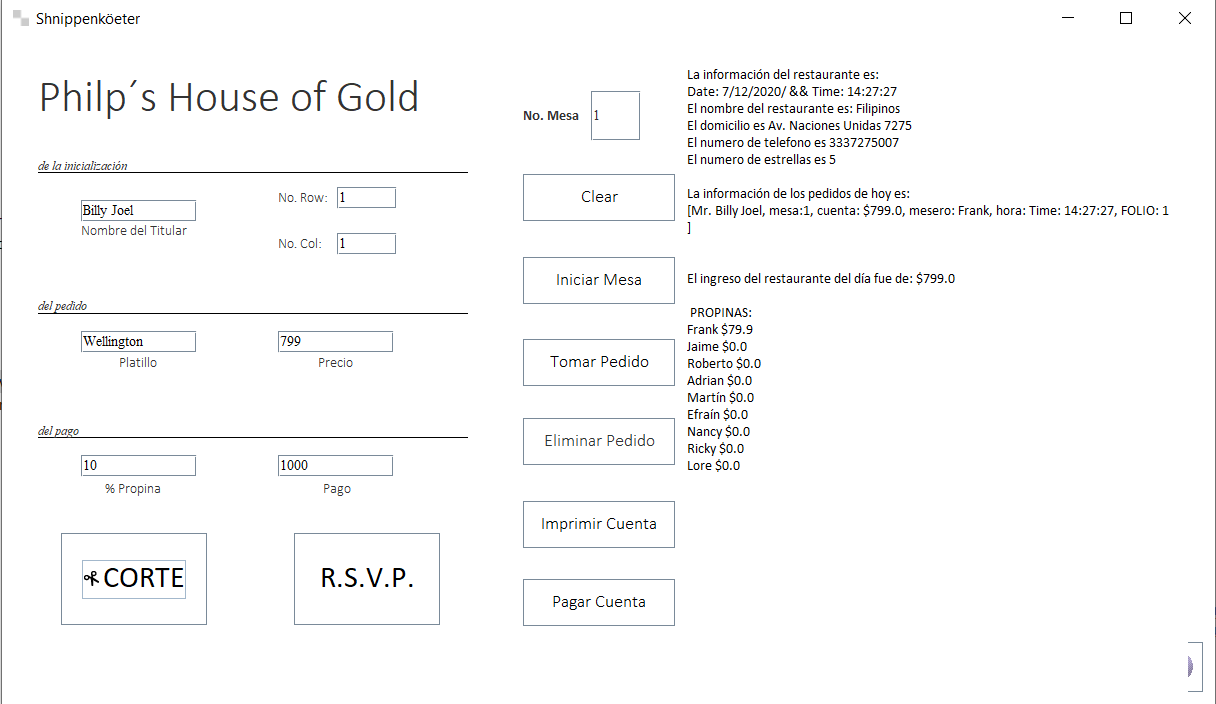


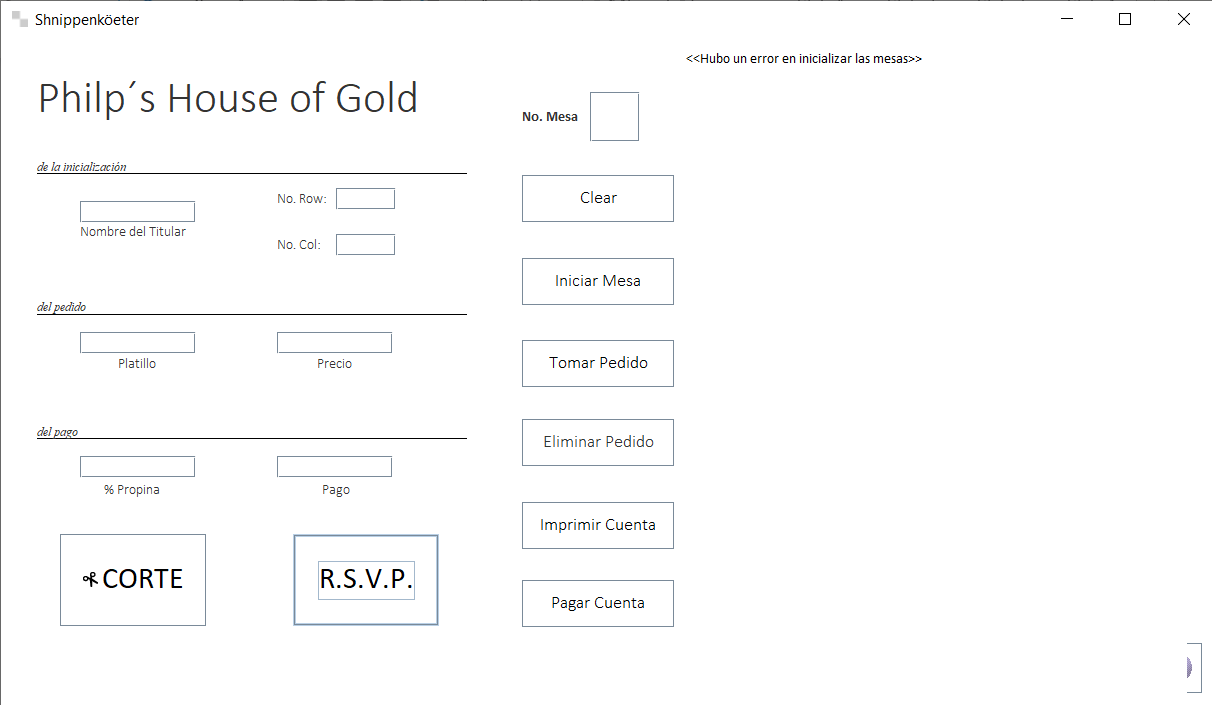


Podemos observar como el lector de archivos funcionó correctamente y cómo el programa no nos permitió hacer el corte con las mesas ocupadas.

## Quinta prueba

Ahora, hago el corte del programa después de crear, una mesa, tomar el pedido, eliminar algunos elementos y haber pegado la cuenta. Después del corte intentaré ejecutar más funciones.





En la primera imagen podemos observar claramente como el programa cumple los objetivos deseados al hacer el corte. Imprime la información del restaurante, la información de los pedidos del día, las ganancias el día y las propinas de los meseros.

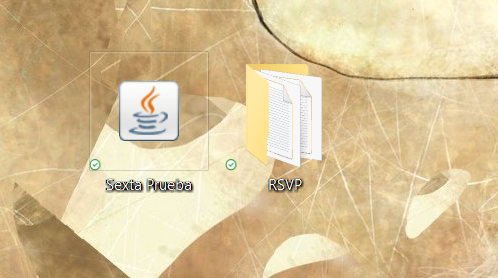
En la segunda imagen, podemos observar claramente como después de hacer el corte, las demás funciones quedan inhabilitadas. En este caso el botón de R.S.V.P nos muestra que no se pueden inicializar las mesas.

## Sexta prueba

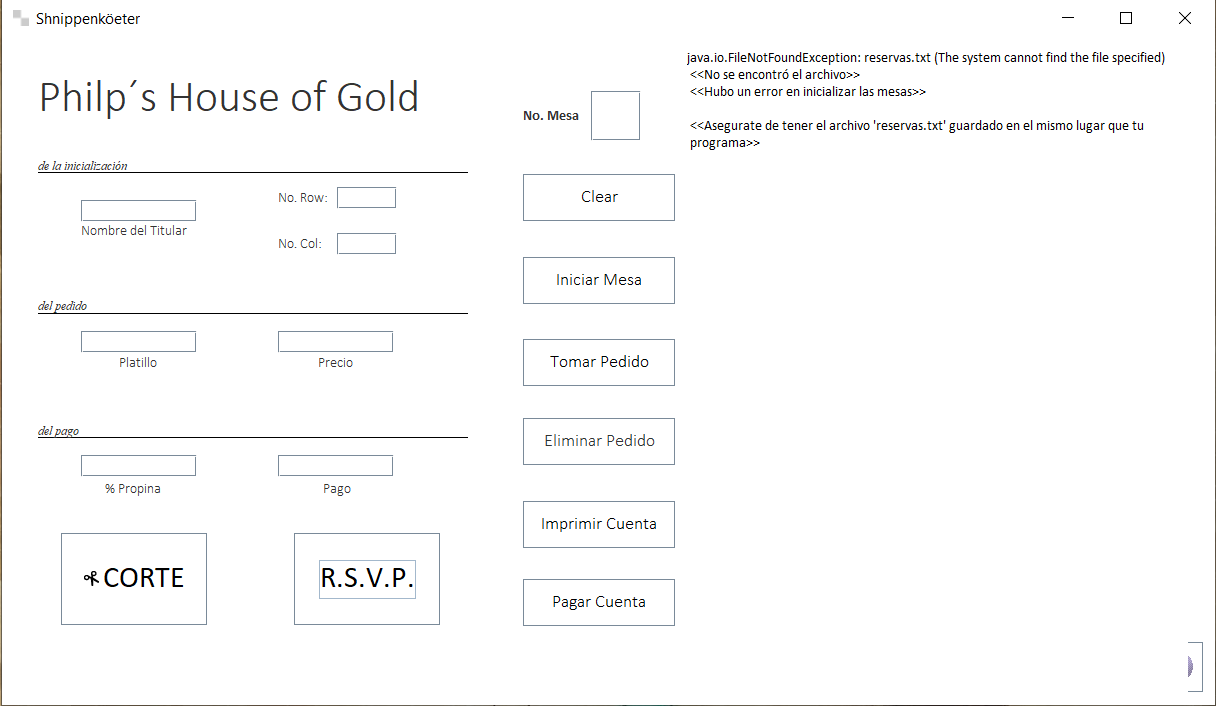
Por último, tenemos las situaciones referentes al archivo de texto.

Es posible hacer que el archivo esté dentro de la información general del jar. Pero si este fuera el caso y el usuario no tuviera acceso a modificarlo todos los días según las reservaciones diarias, el programa no tendría mucho sentido y el botón de reservas solo serviría el primer día.

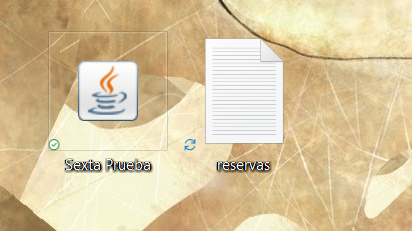
De esta manera primero vamos a demostrar los distintos escenarios según dónde esté guardado el documento de texto.



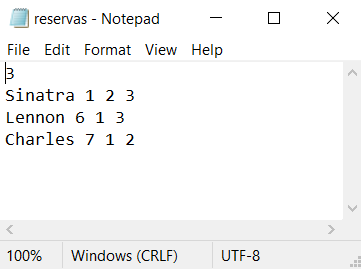
En el escritorio tenemos guardado el programa y una carpeta, dentro de la cual hay un archivo de texto titulado “reservas.txt”.



Cuando intentamos ejecutar el botón R.S.V.P. con el documento en una ubicación distinta a la indicada, el programa nos regresa un mensaje adecuado, explicándonos como el archivo no es encontrado, no se pudo inicializar la mesa y que nos aseguremos que el archivo esté guardado a la misma altura del programa.

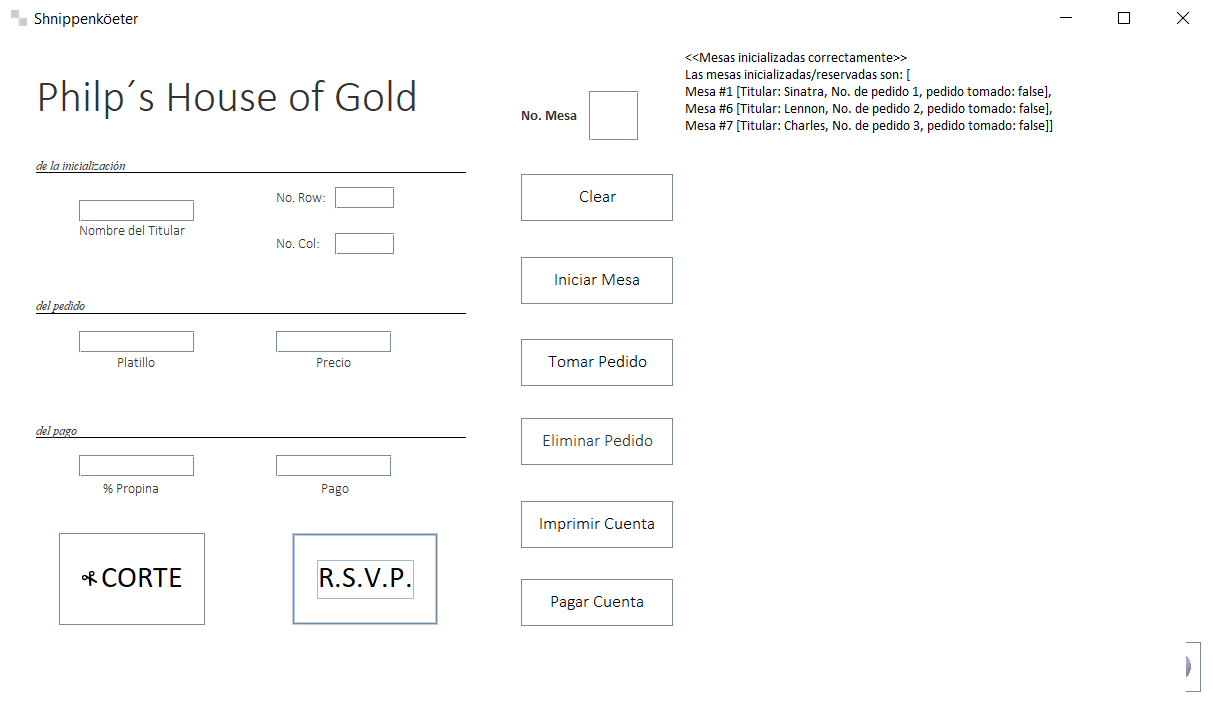


Ahora, a la misma altura que la aplicación, tenemos el archivo reservas.



Este es el contenido del archivo.

Primero la cantidad de reservaciones, seguido del nombre del titular, el número de mesa asignada y las coordenadas de dicha mesa.



Gracias al cambio de la ubicación del archivo de texto, podemos ver como nuestro programa generó el registro correctamente.

De la misma manera, el formato necesario para llenar el archivo de texto es muy simple y será fácil para un usuario modificarlo en el día a día.

# Manual de Usuario

Estimado Usuario:

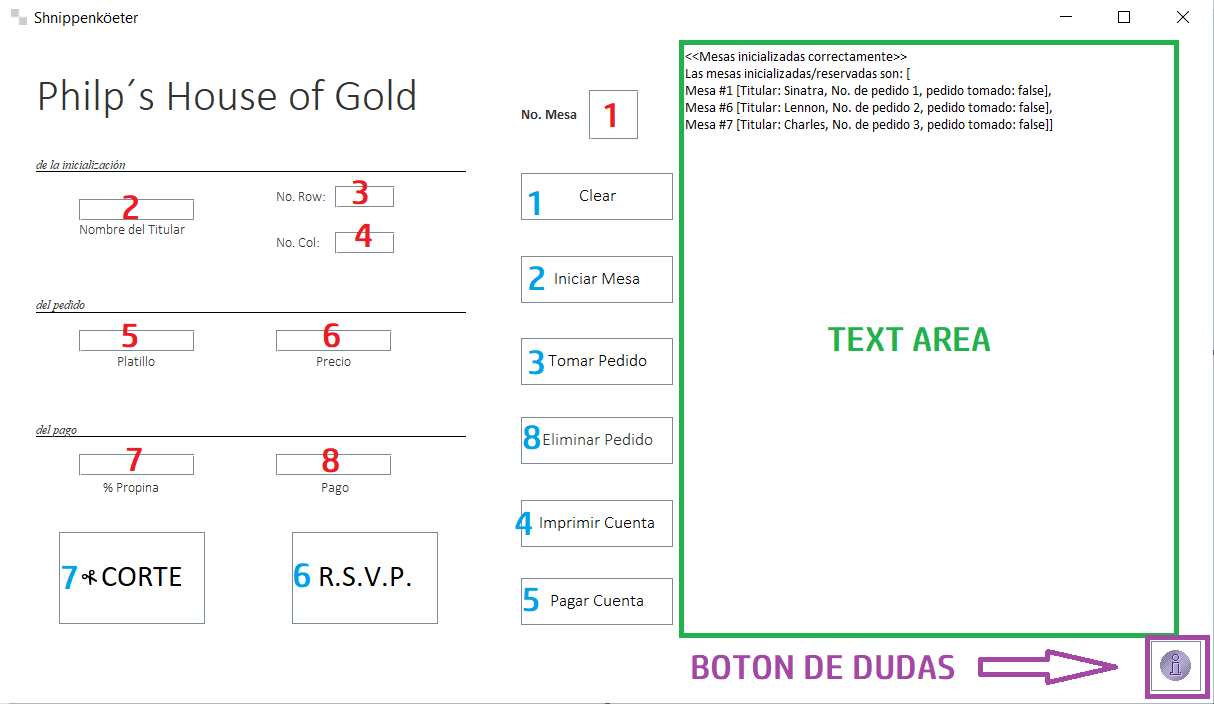
Felicidades por adquirir el nuevo software de administración de negocios gastronómicos de *Schinppenköeter developments*. Ahora, para su comodidad, es nuestro placer explicarle con detalle el funcionamiento del programa, así como las instrucciones para correrlo. También hay que destacar unos cuantos detalles que el usuario deberá tomar en cuenta al momento de trabajar con él. Por último, recomendamos que toda persona que trabaje con el programa en un futuro lea este instructivo antes de manejarlo.

## Partes del programa

El programa cuenta con ocho casillas de texto, ocho botones, un panel de impresión y un botón extra de dudas.

Cada casilla deberá ser llenada según la operación a ejecutar y cada botón ejecutará la función, respectivamente. El área de la derecha es panel de impresión, por debajo a la derecha está el botón de dudas.

Por efectos prácticos, las instrucciones se basan en la numeración de la imagen a continuación.



## Instructivo

1. Al correr el programa, lo primero que quiere hacer el usuario es presionar el botón #6 “R.S.V.P.” para que el programa lea todas las reservaciones y el usuario no pueda iniciar mesas que ya están apartadas.
   1. NOTA: las mesas reservadas se “inicializan” automáticamente al picar el botón.
2. Después de esto, el usuario puede inicializar mesas con el botón #2. Para inicializarla, los recuadros 1, 2, 3 y 4 deberán estar llenos.
   1. El número 1 o número de la mesa deberá ser entre uno y nueve y no se puede usar un número que ya esté siendo usado.
   2. El nombre del titular se llena de preferencia con el apellido del usuario, seguido de la inicial de su nombre.
   3. Los recuadros 1, 3 y 4 se llenan con números entre el uno y el tres. Estos hacen referencia a las coordenadas de las mesas y no pueden usarse coordenadas que estén siendo ocupadas.
3. El usuario puede tomar el pedido de las mesas que ya estén inicializadas. Para esto usará el botón #3 y tendrá que llenar los recuadros 1, 5 y 6.
   1. El número 1 se llena con el número que se le asignó a la mesa al inicializarla.
   2. El número 5 se llena con el nombre del platillo.
   3. El número 6 con el precio del platillo.
4. El usuario puede eliminar el pedido de las mesas a las que ya se les haya tomado el pedido. Para esto se usará el botón #8 y se deben llenar los recuadros 1, 5 y 6.
   1. No se puede eliminar un pedido que no existe.
   2. Para eliminar un pedido, el nombre y el precio de dicho pedido deberán estar escritos exactamente como estaban al tomar el pedido.
5. El usuario puede imprimir la cuenta de una mesa con el botón #4 y solo tiene que llenar el recuadro #1 con el número de la mesa en cuestión
   1. NOTA: El usuario no puede imprimir la cuenta de una mesa a la que no se le ha tomado el pedido.
6. Para pagar una cuenta de utiliza el botón #5 y se deben llenar los recuadros 1, 7 y 8.
   1. El número 1 se llena con el número que se le asignó a la mesa al inicializarla.
   2. El número 7 se llena con el porcentaje de propina que el usuario le quiere dejar a su mesero.
   3. El número 8 se llena con la cantidad de dinero con la que el usuario paga su cuenta.
   4. NOTA: El usuario no puede pagar la cuenta de una mesa a la que no se le ha tomado el pedido.
   5. NOTA: Al pagar una cuenta, la mesa de está desocupando, por lo tanto, el número y la coordenada que se utilizaron al inicializarla, pueden volver a utilizarse.

* El botón #7 hace el corte y no se necesita llenar ninguna casilla para esto.
  + 1. Es importante destacar que no se puede hacer el corte si aún hay mesas habilitadas. Todas tienen que pagarse para poder hacer el corte.
    2. Después de hacer el corte, todo el software pierde funcionalidad. Hasta que se vuelve a correr.
* NOTA: El usuario puede apoyarse del botón #1 para limpiar todos los campos
* NOTA: En la parte inferior derecha está el botón de dudas, con el cual el usuario puede ayudarse en caso de ser necesario.
* NOTA: Para leer un archivo con el programa es necesario que el archivo esté guardado en el mismo lugar que la aplicación. De esta manera es necesario que el usuario actualice cada día su documento de texto para que el programa puede funcionar el día siguiente.

\*Para cualquier duda o comentario comuníquese al 3337275007 o mande un correo a mverduz2@itam.mx

# Conclusiones

Al terminar el programa, el proyecto y el escrito puedo afirmar que solucioné las problemáticas que me propuse, hice un código funcional y amigable. Estos eran tan solo los objetivos del proyecto, pero ahora, después de todo el proceso, puedo concluir otras dos cosas.

La primera es lo mucho que complicamos las actividades diarias y lo sencillo que sería simplificarlas con una implementación de software. Claro que esto implica un reto nuevo, la disciplina del usuario, la paciencia del programador, etcétera. Ahora que terminé el programa, me permití compartirlo con mis familiares, para encontrarme con muchas opiniones distintas. Algunas emocionadas por el desarrollo y la presunta solución, otras un poco asustadas pues no son “fans” de la tecnología. Y otras indiferentes.

Entre los comentarios que recibí uno fue respecto al precio del programa antiguo que tenían los restaurantes familiares y cuando escuché la cifra creí que era chiste, pero era cierto y el programa era poco funcional y, sobre todo, no era específico para nuestro local. Cosa que encontraron fascinante de este.

La segunda cosa que aprendí durante este proyecto es la infinidad de posibilidades que permite la programación. Retomando los comentarios de mis familiares, muchos de ellos eran respecto a qué tanto más se podría hacer. “¿Se puede hace esto?” “¿y aquello?” “¿puedes poner esto acá?” “¿Puedes hacer que haga esto también?”. Todos estos comentarios me motivaron a buscar distintas opciones y con las cuales cada vez fui solucionando más problemas. En realidad, yo creo que este podría ser un proyecto interminable. La cantidad de conocimientos que adquirí en el proceso fue más de lo que me hubiera imaginado.

Uno de esos conocimientos y que creo que es mi favorito es que después de un semestre, mi primer semestre. Reconozco que puedo hacer tantas cosas pero que en realidad no se casi nada. Me queda mucho por aprender y por experimentar, las posibilidades las delimita el programador.