


# taller de programacion I

Trabajo Práctico Especial

GRUPO 1 → SUBGRUPO 1

Bruses –Angelico



“Si se quiere y espera que un programa funcione, lo más probable es que se vea un programa funcionando (y que se pasen por alto los fallos).”

---

CEM KANER

# RESUMEN DE TEST

## PUNTOS PRINCIPALES

Pruebas Unitarias de Caja Negra

Test de Cobertura

Pruebas Unitarias de Caja Blanca

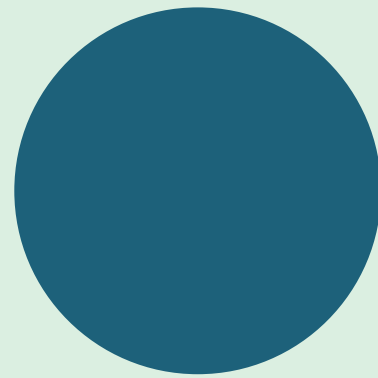
Test de Persistencia

Test de Interfaces Gráficas de Usuario

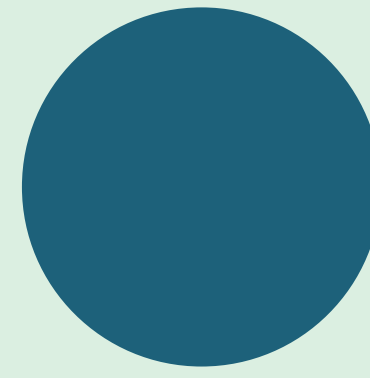
Test de Integración

# TEST UNITARIOS DE CAJA NEGRA

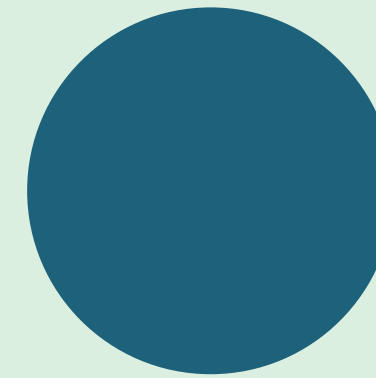
## PRUEBAS DE COMPORTAMIENTO



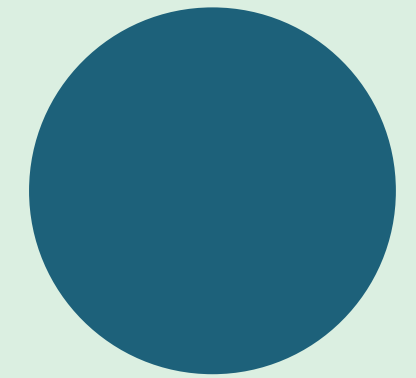
Módulos  
Unitarios



Especificación  
de  
Requerimientos



Pruebas  
Automatizadas



JUnit

# TEST UNITARIOS DE CAJA NEGRA

## ESCENARIOS

Plantear los diferentes contextos donde puede ser llamado módulo unitario.

## TABLAS DE PARTICIONES

División del campo de entrada al módulo en clases de datos .  
Clases de Equivalencia

## BATERÍA DE PRUEBAS

Casos de prueba que se ejecutan con un solo valor perteneciente a su clase de Equivalencia

## RESULTADOS

Encontramos errores en el método agregar Producto ya que permite crear objetos Productos con stock Negativo

# COBERTURA



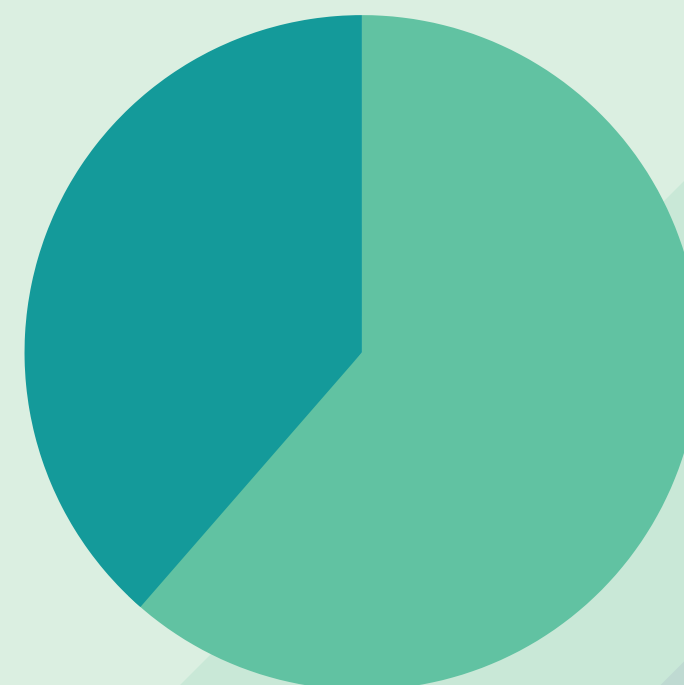
COVERAGE REPORT



Clases  
100%

COBERTURA DE  
CLASES

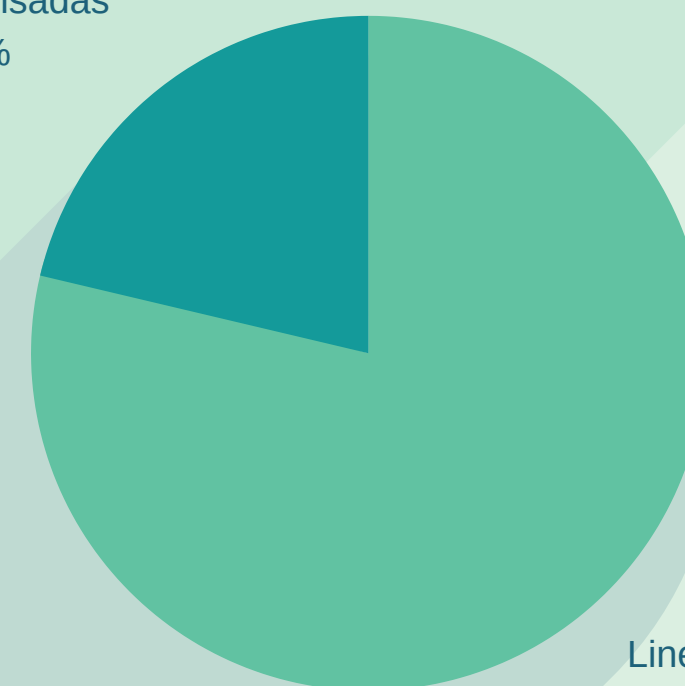
No Testeados  
38.6%



Testeados  
61.4%

COBERTURA DE  
MÉTODOS

Lineas no pisadas  
21.3%



Lineas pisadas  
78.7%

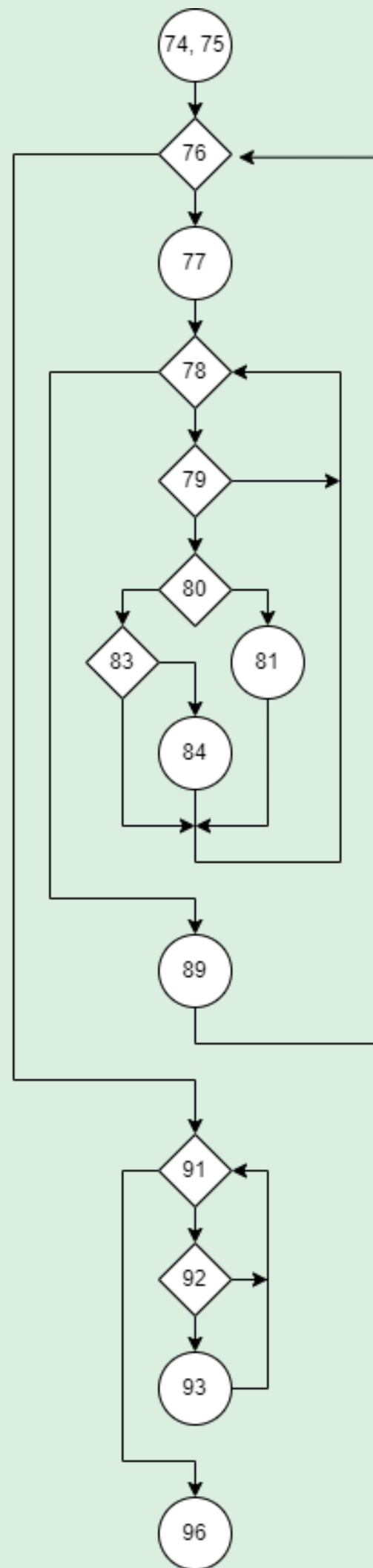
COBERTURA DE  
LINEAS

# TEST UNITARIOS DE CAJA BLANCA

Decidimos realizar una prueba de caja blanca al método `setTotal()`, dentro de la clase `Factura`.

```
/**
 * El metodo permite calcular el total a facturar, recorre todos los pedidos realizados y hace un calculo parcial al que posteriormente se
 * a aplicar descuentos en caso de ser necesario. Primero se recorre el arreglo de promociones de producto para saber si es necesario
 * aplicarle un descuento, para esto la promocion debera estar activa. Luego recorreremos el arreglo de promociones temporales para
 * saber si corresponde aplicarselas.
 * @return devuelve un flotante que debera ser mayor a cero, representa el total a cobrar a la mesa correspondiente
 */
public double setTotal() {
    double total = 0;
    double parcial;
    for (int i = 0; i < pedidos.size(); i++) {
        parcial = pedidos.get(i).getCantidad() * pedidos.get(i).getProducto().getPrecioVenta();
        for (int j = 0; j < promocionesProductos.size(); j++) {
            if (promocionesProductos.get(j).isActiva()) {
                if (promocionesProductos.get(j).isAplicaDosPorUno()) {
                    parcial /= 2.;
                } else {
                    if (promocionesProductos.get(j).isAplicaDtoPorCantidad() && pedidos.get(i).getCantidad() >= promocionesProductos.get(j).getDtoPorCantidad()) {
                        parcial = promocionesProductos.get(j).getDtoPorCantidad_PrecioUnitario() * pedidos.get(i).getCantidad();
                    }
                }
            }
        }
        total += parcial;
    }
    for (int k = 0; k < promocionesTemporales.size(); k++) {
        if (promocionesTemporales.get(k).isActivo() && promocionesTemporales.get(k).getFormaDePago().equals(this.getFormaDePago()) && promocionesTemporales.get(k).getPorcentajeDescuento() > 0) {
            total = total - total * (double) promocionesTemporales.get(k).getPorcentajeDescuento() / 100;
        }
    }
    return total;
}
```

# GRAFO DE CONTROL



## COMPLEJIDAD CICLOMATICA

$$V(G)=8$$



8

LÍMITE SUPERIOR DE NÚMERO DE  
CASOS DE PRUEBA PARA UN  
PROGRAMA

3

CAMINOS NECESARIOS PARA  
ABARCAR EL 100% DE COBERTURA  
DEL CODIGO

## ESCENARIO 1

Arraylist de promocionesTemporales con una promoción temporal activa, aplicable a un pedido. Arraylist de promocionesProductos vacía.

## ESCENARIO 2

Arraylist de promocionesTemporales vacía. Arraylist de promocionesProductos con 2 promociones de producto activa. aplicable a 2 pedidos de la comanda.

# BATERIA DE PRUEBAS

Camino	Escenario	Parámetros de entrada	Salida esperada
C4	2	pedidos = {p1 = {"Coca-Cola", 100, 150, 100}, 2}, FormaDePago = "Efectivo",	total = 150
C6	2	pedidos = {p2 = {"Pepsi", 200, 250, 100}, 2}, FormaDePago = "Efectivo",	total = 400
C8	1	pedidos = {p1 = {"Coca-Cola", 100, 150, 100}, 2}, FormaDePago = "Efectivo"	total = 150

# ERRORES ENCONTRADOS



PROMOCIONES  
TEMPORALES

DESCUENTO  
APLICADOS POR UNO



ATRIBUTO  
DIAS DE PROMO

PRECONDICIONES  
ESCASAS



# TEST DE PERSISTENCIA

## ÉSCENARIOS

Archivo existente

Archivo Inexistente

### ESC 1

Caso Prueba 1 -> Empresa sin datos

Caso Prueba 2 -> Empresa con datos

### ESC 2

Caso Prueba 1 -> Crear archivo

Caso Prueba 2 -> Leer archivo -> excepción

Caso Prueba 3 -> Escribir archivo -> excepción

Caso Prueba 4 -> Leer archivo -> excepción

## RESULTADOS

Todos las pruebas realizadas sobre la persistencia fueron correctas

# TEST DE INTERFACES GRÁFICAS

## MODELO MVC

Separar reglas del negocio de la GUI

## VENTANA PRINCIPAL

Se testearon los sus componentes reaccionaran coherentemente a sus estímulos

## CLASE ROBOT

Permitió simular eventos de teclado y mouse

## RESULTADOS

Todas las pruebas, en cada escenario, fueron correctas y no arrojaron errores

# TEST DE INTEGRACIÓN

## ORIENTADO A OBJETOS

Definición de caso de uso

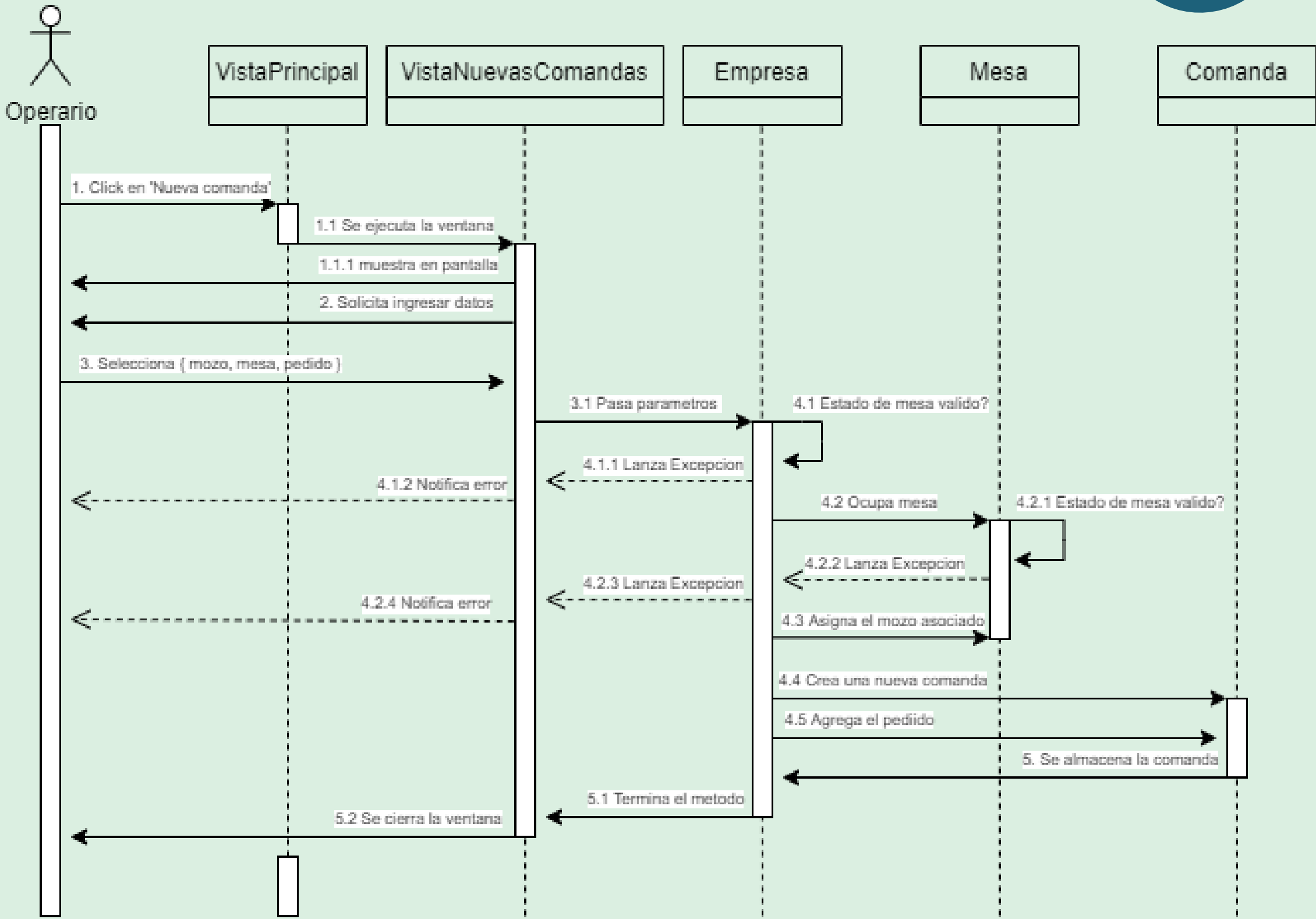
Diagrama de secuencia

Generación de casos de prueba

# CU - CREAR COMANDA

El operario desea asociar una nueva comanda con pedidos a una mesa libre con un mozo activo asociado.

## DIAGRAMA DE SECUENCIA





# GENERACIÓN DE CASOS DE PRUEBA

## ESTADO DE LOS ELEMENTOS

Que elementos afectan a la ejecución del software?

## VALORES DE LOS PARÁMETROS

En caso de existir parámetros, existirán valores aceptables y no aceptables

## CONJUNTO DE DATOS ADECUADO

Con cada estado y cada representante de conjunto de datos se forma un caso de prueba, combinando los diferentes valores de cada categoría.

# CASOS DE PRUEBA

Entrada	Condiciones de entrada	Salida esperada
Mesa = {...,"Libre"}, Mozo = {...,"Activo"}, pedido = {Producto,...}	VistaPrincipal ausente	Falla de ejecución
Mesa = {...,"Libre"}, Mozo = {...,"Activo"}, pedido = {Producto,...}	VistaNuevasComandas ausente	Falla de ejecución
Mesa = {...,"Libre"}, Mozo = {...,"Activo"}, pedido = {Producto,...}	Empresa ausente	Falla de ejecución
Mesa = {...,"Libre"}, Mozo = {...,"Activo"}, pedido = {Producto,...}	Mesa ausente	Falla de ejecución
Mesa = {...,"Libre"}, Mozo = {...,"Activo"}, pedido = {Producto,...}	Comanda ausente	Falla de ejecución
Mesa = {...,"Libre"}, Mozo = {...,"Activo"}, pedido = {Producto,...}	VistaPrincipal visible <u>VistaNuevasComandas</u> visible Empresa presente Mesa presente	Nueva comanda creada
Mesa = {...,"Ocupada"}, Mozo = {...,"Activo"}, pedido = {Producto,...}	VistaPrincipal visible <u>VistaNuevasComandas</u> visible Empresa presente Mesa presente	Mensaje de error
Mesa = {...,"Libre"}, Mozo = {...,"Franco"}, pedido = {Producto,...}	VistaPrincipal visible <u>VistaNuevasComandas</u> visible Empresa presente Mesa presente	Mensaje de error
Mesa = {...,"Libre"}, Mozo = {...,"Ausente"}, pedido = {Producto,...}	VistaPrincipal visible <u>VistaNuevasComandas</u> visible Empresa presente Mesa presente	Mensaje de error

- NOTAMOS QUE EXISTEN CONTRADICCIONES ENTRE EL CONTRATO Y LA DOCUMENTACIÓN DEL MÉTODO.
- SIN EMBARGO SE REALIZARON LOS TEST PERTINENTES PARA ABARCAR TODOS LOS CASOS DE PRUEBA DEFINIDOS.

# CONCLUSIONES

## DOCUMENTACIÓN

Es indispensable contar con una documentación extensa, precisa y completa

## ESPECIFICACIÓN REQUERIMIENTOS DE SOFTWARE

detallada y que no deje zonas grises

## TESTING

que ambos documentos no se contradigan. Para poder realizar un buen trabajo de TESTING