Trabajo Práctico Especial

Taller de programación 1 – Facultad de ingeniería UNMDP

Cursada 2022

Grupo n°: 7

Integrantes:

Subgrupo 1 (Testing e informe en cuestión):

Vázquez, Imanol

Isaías Nievas, Nahuel

Subgrupo 2 (Desarrollo):

Giacri, Tobias

Olave, Alejandro Mikel

Fecha de entrega: 16/11/2022

Link a video de youtube subgrupo1:
https://www.youtube.com/watch?v=LyzE0lBTdXE

Índice

Introducción:

Se nos solicitó como Trabajo Practico Especial de cursada de Taller de programación 1 crear un programa para gestión de un restaurante desde cero. Una vez desarrollado el programa lo entregamos a un grupo de dos compañeros y recibimos de su parte un programa al cual le realizamos testing. En este informe se presenta la documentación y reporte de testing realizado.

Test a realizar:

- Test caja negra
- Test caja blanca
- Test de persistencia
- Test de GUI
- Test de integración

Nota: Para la realización de estos test se utilizó la versión del paquete JUnit4.

Test de caja negra:

Para realizar el test de caja negra comenzamos por ejecutar el programa recibido por parte del otro Sub-grupo. Al hacer esto nos dimos cuenta que había diversos campos de la interfaz gráfica que no presentaban validaciones. Ante esto fuimos a la documentación correspondiente pero no encontramos excepciones que tratasen estas validaciones, a excepción de una que valida la contraseña incorrecta (ContraException).

Como grupo consideramos que el hecho de que no se tratasen estas validaciones es erróneo. Para el test de caja negra entonces asumimos que cuando un caso de prueba era incorrecto, el flujo del programa no debería de continuar y debería de lanzar una excepción, pero al no haber excepciones documentadas entonces definimos que el flujo del programa se corta cuando se recibe una excepción del tipo “Exception”. Al ser esta esta excepción padre de todas las hijas, logramos capturar cualquier excepción posible y cortar el flujo del método.

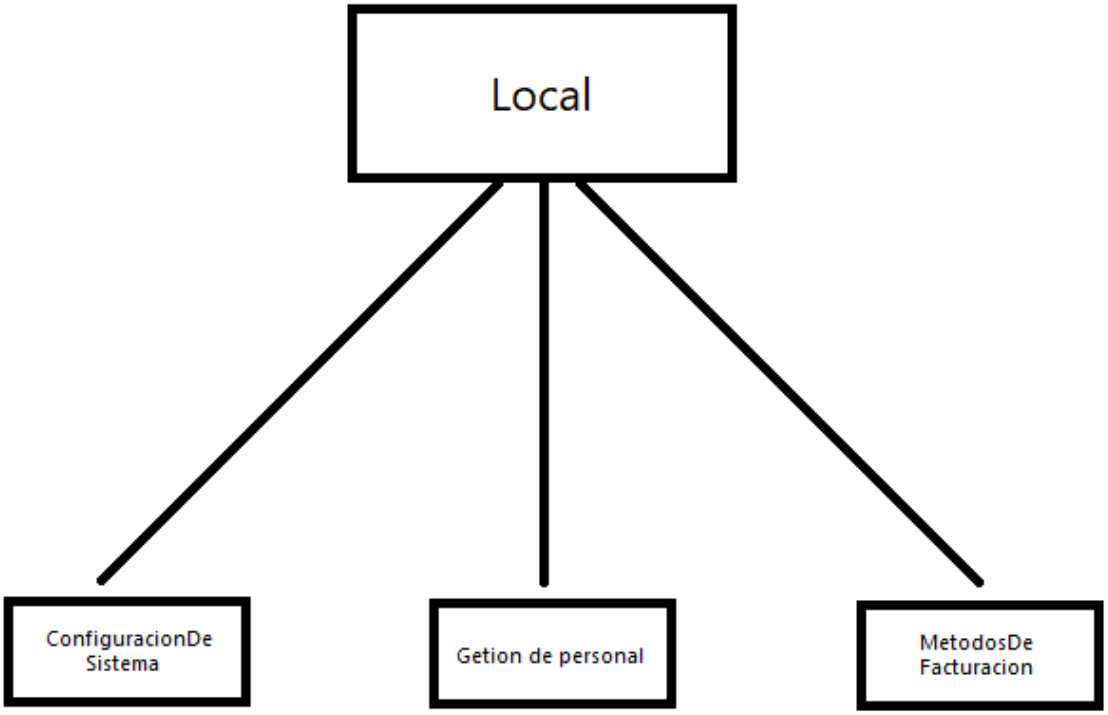
Un ejemplo de test sigue la siguiente estructura básica:

```
@Test
	public void test () {
		try {
			metodoATestear();
			Assert.fail("No deberia seguir con la ejecucion");

		}
		catch(Exception e) {

		}
```

}

Esto trae un inconveniente. Si bien capturamos excepciones, se pueden modificar datos antes de largar la excepción ya que no es controlada y esto puede ocasionar problemas, por lo tanto, no es condición suficiente para afirmar que si un método corta el flujo es correcto, puede o no ser un problema. Por otro lado, sí podemos afirmar que cuando un método no corta su flujo en un caso invalido, efectivamente hay un error en todos los casos.

La capa de negocios está compuesta por un patrón facade, tiene como principal la clase Local que deriva en tres clases hijas



Por documentación se sabe que la clase padre posee patrón Singleton y las siguientes invariantes los cuales se aplican a las clases hijas:

```
/**
<b>inv: </b> <br>
* formasDePago != null <br>
* mesas != null <br>
* mozos != null <br>
* facturas != null <br>
* promocionesProductos != null <br>
* promocionesTemporales != null <br>
* operarios != null <br>
* operarioAdministrador != null <br>
* operarioAdministrador nunca es vacío <br>
* comandas != null <br>
* productos != null <br>
* asignacionDiaria != null <br>
*/
```

Nota: En los métodos de java de test unitarios de caja negra que siguen se utiliza la siguiente notación:

Public void “test” + nombreMetodo + número de caso de la tabla de batería de pruebas {}

A continuación, se determinan los casos de prueba de test unitario de caja negra.

CLASE Local:

```
/**
	* Obtiene una comanda de la lista de comandas a partir de la mesa.
	* <b>Pre: </b> mesa != null<br>
	* @param mesa no puede ser null
	* @return comanda asociada a la mesa
	*/
```

Local → getComandaByMesa(Mesa mesa)

Escenario

Nro escenario	Descripción
1	Lista de comandas con al menos 1 comanda
2	Lista de comandas vacía

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
mesa	mesa es instancia de clase Mesa (1)	mesa = null (2)

(2) sale del contrato por eso no se la considera en la batería de pruebas.

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(mesa1) Escenario 1	Devuelve comanda de la lista asociada a mesa1.	1
2	Incorrecta	(mesa2) Escenario 1	Error	1
3	Incorrecta	(mesa1) Escenario 2	Error	1

mesa1 se encuentra asignada a una comanda de la lista.

mesa2 no se encuentra asignada a una comanda de la lista.

```
/**
	* Obtiene un mozo de la lista de asignaciones diaria a partir de la mesa asignada.
	* <b>Pre: </b> mesa != null<br>
	* @param mesa no puede ser null
	* @return mozo que tiene asignada la mesa
	*/
```

Local → getMozoByMesa(Mesa mesa)

Escenario

Nro escenario	Descripción
1	Lista de mozos con al menos 1 mozo
2	Lista de mozos vacía

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
mesa	mesa es instancia de clase Mesa (1)	mesa = null (2)

(2) sale del contrato por eso no se la considera en la batería de pruebas.

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(mesa1) Escenario 1	Devuelve mozo de la lista asociada a mesa1.	1
2	Incorrecta	(mesa2) Escenario 1	Error	1
3	Incorrecta	(mesa1) Escenario 2	Error	1

mesa1 se encuentra asignada a un mozo de la lista.

mesa2 no se encuentra asignada a un mozo de la lista.

```
/**
	* Obtiene el mozo de la lista de mozos que tiene el maximo acumulado de ventas.
	* @return mozo que tiene maximo acumulado de ventas
```

```
*/
```

Local → getMozoMaxVentas()

Escenario

Nro escenario	Descripción
1	Lista de mozos con al menos 1 mozo.
2	Lista de mozos vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	--- Escenario 1	Retorna mozo con mayor volumen de ventas .	
2	Incorrecta	--- Escenario 2	Error	

```
/**
	* Obtiene el mozo de la lista de mozos que tiene el minimo acumulado
de ventas.
	* @return mozo que tiene minimo acumulado de ventas
	*/
```

Local → getMozoMinVentas()

Escenario

Nro escenario	Descripción
1	Lista de mozos con al menos 1 mozo.
2	Lista de mozos vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	--- Escenario 1	Retorna mozo con menor volumen de ventas.	
2	Incorrecta	--- Escenario 2	Error	

```
/**
	* Obtiene el mozo de la lista de mozos que tiene el maximo promedio
de ventas por mesas atendidas.
	* @return el mozo con el promedio maximo
	*/
```

Local → getMozoMaxPromedio()

Escenario

Nro escenario	Descripción
1	Lista de mozos con al menos 1 mozo.
2	Lista de mozos vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	--- Escenario 1	Retorna mozo con máximo promedio de ventas por mesas atendidas.	
2	Incorrecta	--- Escenario 2	Error	

```
/**
	* Obtiene el mozo de la lista de mozos que tiene el minimo promedio de ventas por mesas atendidas.
	* @return el mozo con el promedio minimo
	*/
```

Local → getMozoMinPromedio()

Escenario

Nro escenario	Descripción
1	Lista de mozos con al menos 1 mozo.
2	Lista de mozos vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	--- Escenario 1	Retorna mozo con mínimo promedio de ventas por mesas atendidas.	
2	Incorrecta	--- Escenario 2	Error	

Por SRS:

Un usuario inactivo (activo = 0) NO puede ingresar al sistema.

En el momento del despliegue inicial, el usuario predeterminado será ADMIN y el password predeterminado será ADMIN1234.

```
/**
	* A traves del nombre de usuario y password se verifica que se encuentre registrado como operario administrador o si se encuentra registrado en la lista de operarios.
	* Si los datos son validos y no pertenece a ninguno de los dos tipos de operarios, no hace nada.
	* Si es operario administrador settea atributo booleano admin de clase Local en true.
```

```
        * @param nombreUsuario: Parametro de tipo String que representa el
nombre de usuario.
        * @param password: Parametro de tipo String que representa la
contrasenia del usuario.
        */
```

Local → login(String nombreUsuario, String password)

Escenario

Nro escenario	Descripción
1	Hay un operario administrador registrado; Lista de operarios con al menos 1 operario registrado.
2	Hay un operario administrador registrado; Lista de operarios vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
1	nombreUsuario != null (1)	nombreUsuario = null (2)
2	password != null (3)	password = null (4)

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(“ADMIN”, “ADMIN1234”) Escenario 1	Se permite ingresar al sistema al operario administrador activo. Atributo admin pasa a valer true.	1, 3
2	Correcta	(“Miguel”, “Oper123”) Escenario 1	Se permite ingresar al sistema al operario activo. Atributo admin continua valiendo false.	1, 3
3	Incorrecta	(“Walter”, “Oper1234”) Escenario 1	Error	1, 3
4	Incorrecta	(“Juan”, “Admin1234”) Escenario 2	Error	1, 3

5	Incorrecta	(“Miguel”, “Oper123”) Escenario 2	Error	1, 3
6	Incorrecta	(null, “Oper123”) Escenario 1	Error	2, 3
7	Incorrecta	(“Miguel”, null) Escenario 1	Error	1,4

CLASE MetodosFacturacion:

Por SRS:

Producto no puede ser nulo.

Activa debe ser true o false.

Por documentación:

```
/**
	* Se crea la promocion de un producto de la lista de productos y se agrega a lista de productos del sistema.
	* diaProm deberia ser un dia de la semana. cantidadMinima deberia ser mayor a 0.
	* descCantMin deberia ser mayor a 0.
	*
	* @param producto: Parametro de tipo Producto que representa producto del cual se hara promocion.
	* @param diaProm: Parametro de tipo String que indica el dia de la semana que es valida la promocion.
	* @param dosXuno: Parametro de tipo boolean que indica si la promocion incluye 2x1.
	* @param descuentoCantMin: Parametro de tipo boolean que indica si la promocion incluye descuento de acuerdo a una cantidad minima.
	* @param cantidadMinima: Parametro de tipo entero que indica la cantidad minima a partir de la cual se aplica la promocion.
	* @param descCantMin: Parametro de tipo float que indica el porcentaje de descuento por cantidad minima.
	* @param activa: Parametro de tipo boolean que indica si la promocion se encuentra activa o no.
	*/
```

MetodosFacturacion → altaPromocionProducto(Producto producto, String diaProm, boolean dosXuno, boolean descuentoCantMin, int cantidadMinima, float descCantMin, boolean activa)

Escenario

Nro escenario	Descripción
1	Lista de productos con al menos 1 producto. Lista de promociones de producto con al menos 1 promoción.
2	Lista de productos vacía. Lista de promociones de producto vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
producto	producto es instancia de la clase Producto (1)	producto = null (2)
diaProm	diaProm = “Lunes” (3.1) diaProm = “Martes” (3.2) diaProm = “Miercoles” (3.3) diaProm = “Jueves” (3.4) diaProm = “Viernes” (3.5) diaProm = “Sabado” (3.6) diaProm = “Domingo” (3.7)	diaProm != “Lunes” (4.1) diaProm != “Martes” (4.2) diaProm != “Miercoles” (4.3) diaProm != “Jueves” (4.4) diaProm != “Viernes” (4.5) diaProm != “Sabado” (4.6) diaProm != “Domingo” (4.7) diaProm = null (5)
dosXuno	dosXuno = true (6) dosXuno = false (7)	
descuentoCantMin	descuentoCantMin = true (8) descuentoCantMin = false (9)	
cantidadMin	cantidadMin > 0 (10)	cantidadMin <= 0 (11)
descCantMin	descCantMin > 0 (12)	descCantMin <= 0 (13)
activa	activa = true (14) activa = false (15)	

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(producto1, “Lunes”, true, true, 2, 50, true) Escenario 1	Se da de alta una promoción activa del producto1 para los días lunes que aplica 2x1 y aplica descuento de 50% por cantidad mínima de 2.	1, 3, 6, 8, 10, 12, 14
2	Correcta	(producto1, “Lunes”, false, true, 2, 50, true) Escenario 1	Se da de alta una promoción activa del producto1 para los días lunes que aplica descuento de 50% por cantidad mínima de 2.	1, 3, 7, 8, 10, 12, 14
3	Correcta	(producto1, “Lunes”, true, false, 2, 50, true) Escenario 1	Se da de alta una promoción activa del producto1 para los días lunes que aplica 2x1.	1, 3, 6, 9, 10, 12, 14
4	Correcta	(producto1, “Lunes”, true, false, 2, 50, false) Escenario 1	Se da de alta una promoción inactiva del producto1 para los días lunes que aplica 2x1.	1, 3, 6, 9, 10, 12, 15
5	Incorrecta	(producto2, “Lunes”, true, true, 2, 50, true) Escenario 1 Escenario 2	Error	1, 3, 6, 8, 10, 12, 14
6	Incorrecta	(null, “Lunes”, true, false, 2, 50, false) Escenario 1 Escenario 2	Error	2, 3, 6, 9, 10, 12, 15
7	Incorrecta	(producto1, “ninguno”, true, false, 0, 0, false) Escenario 1	Error	1, 4, 6, 9, 10, 12, 15
8	Incorrecta	(producto1, null, true, false, 0, 0, false) Escenario 1	Error	1, 5, 6, 9, 10, 12, 15

9	Incorrecta	(producto1, “Lunes”, false, true, -1, 50, true) Escenario 1	Error	1, 3, 7, 8, 11, 12, 14
10	Incorrecta	(producto1, “Lunes”, false, true, 2, -100, true) Escenario 1	Error	1, 3, 7, 8, 11, 13, 14

producto1 se encuentra en la lista de productos.
producto2 no se encuentra en la lista de productos.

```
/**
	* Se elimina la promocion de la lista de promociones de producto. Si promocion no esta en la lista, no hace nada.
	*
	* @param prom: Parametro de tipo PromocionProducto a eliminar de la lista.
	*/
```

MetodosDeFacturacion → bajaPromocionProducto(PromocionProducto prom)

Escenario

Nro escenario	Descripción
1	Lista de promociones de productos con al menos 1 promoción de producto.
2	Lista de promociones de productos vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
prom	prom es una instancia de PromocionProducto (1)	prom = null (2)

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	prom1 Escenario 1	Se da de baja prom1 de la lista de promociones de productos.	1
2	Incorrecta	prom2 Escenario 1	No hace nada	1

3	Incorrecta	null Escenario 1 Escenario 2	No hace nada	2

prom1 se encuentra en la lista de promociones de productos antes de eliminarla.
prom2 no se encuentra en la lista de promociones de productos antes de eliminarla.

```
/**
	* Se modifican todos los parametros de la promocion de producto pasada por parametro.
	* diaProm deberia ser un dia de la semana. cantidadMinima deberia ser mayor a 0.
	* descCantMin deberia ser mayor a 0.
	*
	* @param prom: Parametro de tipo Promocion que representa la promocion del cual se modificaran los atributos.
	* @param diaProm: Parametro de tipo String que indica el dia de la semana que es valida la promocion.
	* @param dosXuno: Parametro de tipo boolean que indica si la promocion incluye 2x1.
	* @param descuentoCantMin: Parametro de tipo boolean que indica si la promocion incluye descuento de acuerdo a una cantidad minima.
	* @param porcentajeCantMin: Parametro de tipo Float que indica el porcentaje de descuento que se aplica a promociones tipo descuentoCantMin
	* @param cantidadMinima: Parametro de tipo entero que indica la cantidad minima a partir de la cual se aplica la promocion.
	* @param activa: Parametro de tipo boolean que indica si la promocion se encuentra activa o no.
	*/
```

MetodosFacturacion → modificacionPromocionProducto(PromocionProducto prom, String diaProm, boolean dosXuno, boolean descuentoCantMin, float porcentajeCantMin, int cantidadMinima, boolean activa)

Escenario

Nro escenario	Descripción
1	Lista de promociones de productos con al menos 1 promoción de producto.
2	Lista de promociones de productos vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
prom	prom es instancia de la clase PromocionProducto (1)	prom = null (2)
diaProm	diaProm = “Lunes” (3.1) diaProm = “Martes” (3.2) diaProm = “Miercoles” (3.3)	diaProm != “Lunes” (4.1) diaProm != “Martes” (4.2) diaProm != “Miercoles” (4.3)

	diaProm = “Jueves” (3.4) diaProm = “Viernes” (3.5) diaProm = “Sabado” (3.6) diaProm = “Domingo” (3.7)	diaProm != “Jueves” (4.4) diaProm != “Viernes” (4.5) diaProm != “Sabado” (4.6) diaProm != “Domingo” (4.7) diaProm = null (5)
dosXuno	dosXuno = true (6) dosXuno = false (7)	dosXuno = null (8)
descuentoCantMin	descuentoCantMin = true (9) descuentoCantMin = false (10)	
porcentajeCantMin	descCantMin > 0 (11)	descCantMin <= 0 (12)
cantidadMin	cantidadMin > 0 (13)	cantidadMin <= 0 (14)
activa	activa = true (15) activa = false (16)	

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(promocion1, “Lunes”, true, true, 50, 2, true) Escenario 1	Se modifica promocion1 a activa para los días lunes que aplica 2x1 y aplica descuento de 50% por cantidad mínima de 2.	1, 3, 6, 9, 11, 13, 15
2	Correcta	(promocion1, “Martes”, false, true, 50, 2, true) Escenario 1	Se modifica promocion1 a activa para los días martes que aplica descuento de 50% por cantidad mínima de 2.	1, 3, 7, 9, 11, 13, 15
3	Correcta	(promocion1, “Martes”, true, false, 50, 2, true) Escenario 1	Se modifica promocion1 a activa para los días martes que aplica 2x1.	1, 3, 6, 10, 11, 13, 15

4	Correcta	(promocion1, “Martes”, true, false, 50, 2, false) Escenario 1	Se modifica promocion1 a promoción inactiva del producto1 para los días martes que aplica 2x1.	1, 3, 6, 10, 11, 13, 16
5	Incorrecta	(null, “Martes”, true, false, 50, 2, false) Escenario 1 Escenario 2	Error	2, 3, 6, 10, 11, 13, 16
6	Incorrecta	(promocion1, “ninguno”, true, false, 50, 2, false) Escenario 1	Error	1, 4, 6, 10, 11, 13, 16
7	Incorrecta	(promocion1, null, true, false, 50, 2, false) Escenario 1	Error	1, 5, 6, 10, 11, 13, 16
8	Incorrecta	(promocion1, “Martes”, false, true, -1, 2, true) Escenario 1	Error	1, 3, 7, 9, 12, 13, 15
9	Incorrecta	(promocion1, “Martes”, false, true, 50, -100, true) Escenario 1	Error	1, 3, 7, 9, 12, 14, 15
10	Incorrecta	(promocion2, “Martes”, true, false, 50, 2, false) Escenario 1 Escenario 2	Error	1, 3, 6, 10, 11, 13, 16

promocion1 está incluida en la lista de promociones de productos.
promocion2 no está incluida en la lista de promociones de productos.

Por SRS:

formaPago = “efectivo” || formaPago = “tarjeta” || formaPago = “mercPago” || formaPago = “ctaDNI”

Por documentación:

```
/**
	* Se crea y da de alta una nueva promocion temporal a la lista de promociones temporales.
	* porcentajeDesc deberia ser mayor a 0
	* diasDePromo deberia indicar los dias de la semana.
	* activa deberia ser true o false.
	* acumulable deberia ser true o false.
	*
```

```
        * @param nombre: parametro de tipo String que indica el nombre de la
nueva promocion temporal.
        * @param formaPago: parametro de tipo String que indica forma de
pago.
        * @param porcentajeDesc: parametro de tipo entero que indica
porcentaje de descuento.
        * @param diasDePromo: parametro de tipo String que indica dias de
promocion.
        * @param activa: parametro de tipo Boolean que indica si promocion
esta activa o no.
        * @param acumulable: parametro de tipo boolean que indica si es
acumulable con otras promociones.
        */
```

MetodosFacturacion → altaPromocionTemporal(String nombre, String formaPago, int porcentajeDesc, String diasDePromo, boolean activa, boolean acumulable)

Escenario

Nro escenario	Descripción
1	Lista de promociones temporales con al menos 1 promoción temporal.
2	Lista de promociones temporales vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
nombre	nombre != null (1)	nombre = null (2)
formaPago	formaPago = “efectivo” (3.1) formaPago = “tarjeta” (3.2) formaPago = “mercPago” (3.3) formaPago = “ctaDNI”(3.4)	formaPago != “efectivo” (4.1) formaPago != “tarjeta” (4.2) formaPago != “mercPago” (4.3) formaPago != “ctaDNI” (4.4) formaPago = null (5)
porcentajeDesc	porcentajeDesc > 0 (6)	porcentajeDesc <= 0 (7)
diasDePromo	diasDePromo = “Lunes” (8.1) diasDePromo = “Martes” (8.2) diasDePromo = “Miercoles” (8.3) diasDePromo = “Jueves” (8.4) diasDePromo = “Viernes” (8.5) diasDePromo = “Sabado” (8.6) diasDePromo = “Domingo” (8.7)	diasDePromo != “Lunes” (9.1) diasDePromo != “Martes” (9.2) diasDePromo != “Miercoles” (9.3) diasDePromo != “Jueves” (9.4) diasDePromo != “Viernes” (9.5) diasDePromo != “Sabado” (9.6) diasDePromo != “Domingo” (9.7)

		diasDePromo = null (10)
activa	activa = true (11) activa = false (12)	
acumulable	acumulable = true (13) acumulable = false (14)	

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(“Gran promo”, “efectivo”, 30, “Martes”, true, true)	Se agrega a la lista de promociones temporales del sistema y se da de alta una promocion temporal activa en efectivo de 30% acumulable los días martes.	1, 3, 6, 8, 11, 13
2	Correcta	(“Gran promo”, “efectivo”, 30, “Martes”, false, true)	Se agrega a la lista de promociones temporales del sistema y se da de alta una promocion temporal inactiva en efectivo de 30% acumulable los días martes.	1, 3, 6, 8, 12, 13
3	Correcta	(“Gran promo”, “efectivo”, 30, “Martes”, true, false)	Se agrega a la lista de promociones temporales del sistema y se da de alta una promocion temporal activa en efectivo de 30% no acumulable los días martes.	1, 3, 6, 8, 12, 14
4	Incorrecta	(null, “efectivo”, 30, “Martes”, true, true)	Error	2, 3, 6, 8, 11, 13
5	Incorrecta	(“Gran promo”, “ninguno”, 30, “Martes”, true, true)	Error	1, 4, 6, 8, 11, 13
6	Incorrecta	(“Gran promo”, null, 30, “Martes”, true, true)	Error	1, 5, 6, 8, 11, 13
7	Incorrecta	(“Gran promo”, “efectivo”, -1000, “Martes”, true, true)	Error	1, 4, 7, 8, 11, 13

8	Incorrecta	(“Gran promo”, “efectivo”, 30, “ninguno”, true, true)	Error	1, 3, 6, 9, 11, 13
9	Incorrecta	(“Gran promo”, “efectivo”, 30, null, true, true)	Error	1, 3, 6, 10, 11, 13

```
/**
	* Se elimina una promocion temporal de la lista de promociones temporales.
	*
	* @param prom: Parametro de tipo PromocionTemporal a eliminar.
	*/
```

MetodosFacturacion → bajaPromocionTemporal(PromocionTemporal prom)

Escenario

Nro escenario	Descripción
1	Lista de promociones temporales con al menos 1 promoción temporal.
2	Lista de promociones temporales vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
prom	prom es una instancia de la clase PromocionTemporal (1)	prom = null (2)

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	prom1 Escenario 1	Se da de baja prom1 de la lista de promociones temporales.	1
2	Incorrecta	prom2 Escenario 2	Error	1
3	Incorrecta	null	Error	2

prom1 se encuentra en la lista de promociones temporales.
prom2 no se encuentra en la lista de promociones temporales.

Por SRS:

formaPago = “efectivo” || formaPago = “tarjeta” || formaPago = “mercPago” || formaPago = “ctaDNI”

Por documentación:

```
/**
	* Se modifican todos los parametros de la promocion temporal pasada por parametro.
	* porcentajeDesc deberia ser mayor a 0
	* diasDePromo deberia indicar los dias de la semana.
	* activa deberia ser true o false.
	* acumulable deberia ser true o false.
	*
	* @param nombre: parametro de tipo String que indica el nombre de la nueva promocion temporal.
	* @param formaPago: parametro de tipo String que indica forma de pago.
	* @param porcentajeDesc: parametro de tipo entero que indica porcentaje de descuento.
	* @param diasDePromo: parametro de tipo String que indica dias de promocion.
	* @param activa: parametro de tipo Boolean que indica si promocion esta activa o no.
	* @param acumulable: parametro de tipo boolean que indica si es acumulable con otras promociones.
	*/
```

MetodosFacturacion → modificacionPromocionTemporal(PromocionTemporal prom, String nombre, String formaPago, int porcentajeDesc, String diasDePromo, boolean activa, boolean acumulable)

Escenario

Nro escenario	Descripción
1	Lista de promociones temporales con al menos 1 promoción temporal.
2	Lista de promociones temporales vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
prom	prom es una instancia de la clase PromocionTemporal (1)	prom = null (2)
nombre	nombre != null (3)	nombre = null (4)

formaPago	formaPago = “efectivo” (5.1) formaPago = “tarjeta” (5.2) formaPago = “mercPago” (5.3) formaPago = “ctaDNI”(5.4)	formaPago != “efectivo” (6.1) formaPago != “tarjeta” (6.2) formaPago != “mercPago” (6.3) formaPago != “ctaDNI” (6.4) formaPago = null (7)
porcentajeDesc	porcentajeDesc > 0 (8)	porcentajeDesc <= 0 (9)
diasDePromo	diasDePromo = “Lunes” (10.1) diasDePromo = “Martes” (10.2) diasDePromo = “Miercoles” (10.3) diasDePromo = “Jueves” (10.4) diasDePromo = “Viernes” (10.5) diasDePromo = “Sabado” (10.6) diasDePromo = “Domingo” (10.7)	diasDePromo != “Lunes” (11.1) diasDePromo != “Martes” (11.2) diasDePromo != “Miercoles” (11.3) diasDePromo != “Jueves” (11.4) diasDePromo != “Viernes” (11.5) diasDePromo != “Sabado” (11.6) diasDePromo != “Domingo” (11.7) diasDePromo = null (12)
activa	activa = true (13) activa = false (14)	
acumulable	acumulable = true (15) acumulable = false (16)	

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(promo1, “Promo”, “ctaDNI”, 30, “Jueves”, true, true)	Se modifica la promo1 a nombre Promo y a activa en ctaDNI de 30% acumulable los días jueves.	1, 3, 5, 8, 10, 13, 15
2	Correcta	(promo1, “Promocion1”, “ctaDNI”, 40, “Jueves”, false, true)	Se modifica la promo1 a nombre Promocion1 y a inactiva en ctaDNI de 40%	1, 3, 5, 8, 10, 14, 15

			acumulable los días jueves.	
3	Correcta	(promo1, “Promocion1”, “ctaDNI”, 40, “Jueves”, true, false)	Se modifica la promo1 a nombre Promo y a activa en ctaDNI de 40% no acumulable los días jueves.	1, 3, 5, 8, 10, 13, 16
4	Incorrecta	(promo2, “Promocion2”, “tarjeta”, 25, “Sabado”, true, true)	Error	1, 3, 5, 8, 10, 13, 15
5	Incorrecta	(null, “Promocion1”, “ctaDNI”, 40, “Jueves”, true, true)	Error	2, 3, 5, 8, 10, 13, 15
6	Incorrecta	(promo1, null, “ctaDNI”, 40, “Jueves”, true, true)	Error	1, 4, 5, 8, 10, 13, 15
7	Incorrecta	(promo1, “Promocion1”, “ninguno”, 40, “Jueves”, true, true)	Error	1, 3, 6, 8, 10, 13, 15
8	Incorrecta	(promo1, “Promocion1”, null, 40, “Jueves”, true, true)	Error	1, 3, 7, 8, 10, 13, 15
9	Incorrecta	(promo1, “Promocion1”, “ctaDNI”, -1000, “Jueves”, true, true)	Error	1, 3, 5, 9, 10, 13, 15
10	Incorrecta	(promo1, “Promocion1”, “ctaDNI”, 40, “ninguno”, true, true)	Error	1, 3, 5, 8, 11, 13, 15

11	Incorrecta	(promo1, “Promocion1”, “ctaDNI”, 40, null, true, true)	Error	1, 3, 5, 8, 12, 13, 15

promo1 se encuentra en la lista de promociones temporales.
promo2 no se encuentra en la lista de promociones temporales.

```
/**
	* Se genera una factura. Se pasa como parametro la comanda a cerrar a partir de la cual se calculara el total de la factura y a mesa a cambiar de estado. Se indica el metodo de pago, fecha y dia de la semana.
	* A partir de la lista de pedidos de la comanda se calcula el total. De la lista de pedidos se obtienen los productos.
	* Se verifica que los productos se encuentren o no en la lista de promociones temporales y/o en la lista de promociones de producto.
	* A traves de la mesa de la comanda se obtiene el mozo asignado a la misma. Al mozo se le actualiza el acumulado y se le aumenta en 1 la cantidad de mesas atendidas.
	*
	* <b>Pre: </b> comanda != null<br>
	* @param fecha: Parametro de tipo Calendar que indica la fecha del sistema.
	* @param diaSemana: Parametro de tipo String que indica el dia de la semana que se emitio la factura.
	* @param comanda: Parametro de tipo comanda que indica la comanda a cerrar y a partir de la cual se obtendran los datos para cerrar la factura.
	* @param metodoDePago: Parametro de tipo String que indica el metodo de pago de la factura.
	* @return Factura generada
	*/
```

MetodosFacturacion → generacionDeFactura(Calendar fecha, String diaSemana, Comanda comanda, String metodoDePago)

Escenarios

Nro escenario	Descripción
1	Lista de promociones de producto con al menos 1 elemento; Lista de promociones temporales con al menos 1 elemento;
2	Lista de promociones de producto vacía; Lista de promociones temporales vacía;

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
fecha	fecha != null (1)	fecha = null (2)
diaSemana	diaSemana = “Lunes” (3.1) diaSemana = “Martes” (3.2)	diaSemana != “Lunes” (4.1) diaSemana != “Martes” (4.2)

	diaSemana = “Miercoles” (3.3) diaSemana = “Jueves” (3.4) diaSemana = “Viernes” (3.5) diaSemana = “Sabado” (3.6) diaSemana = “Domingo” (3.7)	diaSemana != “Miercoles” (4.3) diaSemana != “Jueves” (4.4) diaSemana != “Viernes” (4.5) diaSemana != “Sabado” (4.6) diaSemana != “Domingo” (4.7) diaSemana = null (5)
comanda	comanda != null (6)	comanda = null (7)
metodoDePago	metodoDePago = “efectivo” (8.1) metodoDePago = “tarjeta” (8.2) metodoDePago = “mercPago” (8.3) metodoDePago = “ctaDNI”(8.4)	metodoDePago != “efectivo” (9.1) metodoDePago != “tarjeta” (9.2) metodoDePago != “mercPago” (9.3) metodoDePago != “ctaDNI” (9.4) metodoDePago = null (10)

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(fecha, “Jueves”, comanda1, “ctaDNI”) Escenario 1	Se aplica promoción temporal del día jueves en ctaDNI. Se aplica promoción de producto a los productos de la lista de pedidos participantes. Calcula el total de la factura. Se acumula el total al mozo que atiende la mesa y aumenta su cantidad de mesas atendidas. Retorna la factura creada.	1, 3, 6, 8
2	Correcta	(fecha, “Lunes”, comanda2, “efectivo”) Escenario 1	No aplica promoción temporal porque no hay. Se aplica promoción de producto a los productos de la lista de pedidos participantes. Calcula el total de la factura. Se	1, 3, 6, 8

			acumula el total al mozo que atiende la mesa y aumenta su cantidad de mesas atendidas. Retorna la factura creada.	
3	Correcta	(fecha, “Jueves”, comanda3, “ctaDNI”) Escenario 1	Se aplica promoción temporal del día jueves en ctaDNI. No aplica promoción de producto porque no hay. Calcula el total de la factura. Se acumula el total al mozo que atiende la mesa y aumenta su cantidad de mesas atendidas. Retorna la factura creada.	1, 3, 6, 8
4	Correcta	(fecha, “Martes”, comanda4, “efectivo”) Escenario 1	Calcula el total de la factura. Se acumula el total al mozo que atiende la mesa y aumenta su cantidad de mesas atendidas. Retorna la factura creada.	1, 3, 6, 8
5	Incorrecta	(null, “Jueves”, comanda1, “ctaDNI”) Escenario 1	Error	2, 3, 6, 8
6	Incorrecta	(fecha4, “dia”, comanda1, “ctaDNI”) Escenario 1	Error	1, 4, 6, 8
7	Incorrecta	(fecha4, null, comanda1, “ctaDNI”) Escenario 1	Error	1, 5, 6, 8
8	Incorrecta	(fecha4, “Jueves”, comanda1, “Lavado de platos”) Escenario 1	Error	1, 3, 6, 9
9	Incorrecta	(fecha4, “Jueves”, comanda1, null) Escenario 1	Error	1, 3, 6, 10

Por SRS:

cantidad > 0

fecha debe ser la del día actual.

Por documentación:

```
/**
	* Se crea y da de alta un pedido a la lista de pedidos si la cantidad
pedida de ese producto esta disponible.
	* Se actualiza el stock del producto pedido.
	*
	* <b>Pre: </b> producto != null<br>
	* <b>Pre: </b> cantidad > 0<br>
	* @param hoy: Parametro de tipo String convertido de un
GregorianCalendar.
	* @param cantidad: Parametro de tipo int que representa la cantidad
del producto pedido
	* @param producto: Parametro de tipo Producto que representa el
producto pedido
	* @return el nuevo pedido o null en caso de que no haya stock
suficiente.
	*/
```

MetodosFacturacion → altaPedido(String hoy, int cantidad, Producto producto)

Escenarios

Nro escenario	Descripción

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
hoy	hoy = fecha del día actual (1)	hoy = null (2)
cantidad	cantidad > 0 (3)	cantidad <= 0 (4)
producto	producto es instancia de la clase Producto. (5)	producto = null (6)

(4) y (6) sale del contrato por eso no se la considera en la batería de pruebas.

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	GregorianCalendar.toString(), 2, producto1	Devuelve el pedido dato de alta y actualiza el stock del producto.	1, 3, 5

2	Incorrecta	null, 2, producto1	Error	2, 3, 5
3	Incorrecta	GregorianCalendar.toString(), 11, producto1	Devuelve null	1, 3, 5

Stock producto1 = 10.

```
/**
	* Se da de baja un pedido de la lista de pedidos de una comanda.
	* @param comanda: Parametro de tipo comanda de la cual se da de baja
el pedido.
	* @param pedido: Parametro de tipo pedido que indica pedido que se da
de baja de la lista de una comanda.
	*/
```

MetodosFacturacion → bajaPedido(Comanda comanda, Pedido pedido)

Escenarios

Nro escenario	Descripción
1	Lista de pedidos de la comanda con al menos 1 pedido
2	Lista de pedidos de la comanda vacía

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
comanda	comanda instancia de la clase Comanda (1)	comanda != null (2)
pedido	pedido instancia de la clase Pedido (3)	pedido != null (4)

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	comanda1, pedido1 Escenario 1	Se da de baja de la lista de pedidos de la comanda.	1, 3

2	Incorrecta	comanda1, pedido2 Escenario 1	Error	1, 3
3	Incorrecta	null, pedido1 Escenario 1	Error	2, 3
4	Incorrecta	comanda1, null Escenario 1	Error	1, 4
5	Incorrecta	null, null Escenario 1 Escenario 2	Error	2, 4

pedido1 se encuentra en la lista de pedidos de la comanda1.
pedido2 no se encuentra en la lista de pedidos de la comanda1.

Por SRS:

Mesa debe tener un mozo asociado.
Mesa debe estar en estado libre.
Al crear la comanda, mesa debe pasar a estado ocupada.
Mesa asociada debe tener un mozo activo asociado

Por documentación:

```
/**
	* Se crea y se da de alta una nueva comanda como activa en la lista de comandas del local.

	*
	* <b>Pre: </b> mesa != null<br>
	* <b>Pre: </b> pedido != null<br>
	*
	* @param mesa: Parametro de tipo Mesa a la cual se le asignara la comanda.
	* @param pedido: Parametro de tipo Pedido que indica pedido inicial de la comanda.
	* @return nueva instancia de Comanda
	*/
```

MetodosFacturacion → altaComanda(Mesa mesa, Pedido pedido)

Escenarios

Nro escenario	Descripción

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
mesa	mesa es instancia de la clase Mesa (1)	mesa = null (2)
pedido	pedido es instancia de la clase Pedido (3)	pedido = null (4)

(2) y (4) sale del contrato por eso no se la considera en la batería de pruebas.

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	mesa1, pedido1	Se da de alta una nueva comanda en la lista de comandas.	1, 3

```
/**
	* Se agrega o se remueve un pedido de la lista de pedidos de la comanda.
	* <b>Pre: </b> comanda != null<br>
	* <b>Pre: </b> pedido != null<br>
	* @param comanda: Parametro de tipo Comanda que representa la comanda de la cual se actualizara la lista de pedidos.
	* @param pedido: Parametro de tipo Pedido que representa el pedido a dar de alta o baja de la lista de pedidos de la comanda.
	* @param agregar: Parametro de tipo boolean que representa accion a realizar.
	*/
```

MetodosFacturacion → modificacionComanda(Comanda comanda, Pedido pedido, boolean agregar)

Escenarios

Nro escenario	Descripción
1	Lista de comandas con al menos 1 comanda.
2	Lista de comandas vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
comanda	comanda es instancia de la clase Comanda (1)	comanda != null (2)

pedido	pedido es instancia de la clase Pedido (3)	pedido != null (4)
agregar	agregar = true (5) agregar = false (6)	

(2) y (4) sale del contrato por eso no se la considera en la batería de pruebas.

Batería de pruebas:

Número de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	comanda1, pedido2, true Escenario 1	Agrega pedido2 a la lista de pedidos de la comanda1.	1, 3, 5
2	Correcta	comanda1, pedido1, false Escenario 1	Da de baja pedido1 de la lista de pedidos de la comanda1.	1, 3, 6
3	Incorrecta	comanda1, pedido2, false Escenario 1	Error	1, 3, 6
4	Incorrecta	comanda1, pedido1, true Escenario 1	Error	1, 3, 5

pedido1 se encuentra en la lista de pedidos de la comanda1.
pedido2 no se encuentra en la lista de pedidos de la comanda2.

CLASE GestionDePersonal

```
/**
	*  Da de alta un operario
	* @param nombreApellido : Parametro de tipo String que representa el
nombre y apellido del operario
	* @param nacimiento : Parametro de tipo String que representa la
fecha de nacimiento del operario
	* @param nombreUsuario : Parametro de tipo String que representa el
nombre de usuario de la cuenta del operario
	* @param password : Parametro de tipo String que representa el
passworde de la cuenta del operario
	*/
```

GestionDePersonal → altaOperario(String nombreApellido, String nacimiento, String

Escenario

Nro escenario	Descripción
1	Lista de operarios con al menos 1 operario.
2	Lista de operarios vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
nombreApellido	nombreApellido!=null (1)	nombreApellido==null(2)
nacimiento	nacimiento!=null(3)	NombreApellido==null(4)
nombreUsuario	nombreUsuario!=null(5)	NombreUsuario==null(6)
password	Password tiene entre 6 y 12 caracteres, contiene al menos un digito y una mayúscula. (7)	Password ==null (8.1) Password tiene menos de 6 caracteres (8.2) Password tiene más de 12 caracteres (8.3) Password no tiene al menos un digito (8.4) Password no tiene al menos una mayúscula (8.5)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	{“Nahuel”,”14/01/2000”, “Nahuel”, “Nahuel38}	Se crea un operario en el sistema con su información	1,3,5,7
2	Incorrecta	{null,14/01/2000”, “Nahuel”, “Nahuel38}	Error	2,3,5,7
3		{“Nahuel”,”null”, “Nahuel”, “Nahuel38}	Error	1,4,5,7
4		{“Nahuel”,”14/01/2000”, “null”, “Nahuel38}	Error	1,3,6,7

5		{“Nahuel”,”14/01/2000”, “Nahuel”, “null”}	Error	1,3,5,8.1
6		{“Nahuel”,”14/01/2000”, “Nahuel”, “Nah38”}	Error	1,3,5,8.2
7		{“Nahuel”,”14/01/2000”, “Nahuel”, “Nahuasdasdael38”}	Error	1,3,5,8.3
8		{“Nahuel”,”14/01/2000”, “Nahuel”, “Nahuelll”}	Error	1,3,5,8.4
9		{“Nahuel”,”14/01/2000”, “Nahuel”, “nahuel38”}	Error	1,3,5,8.5

```
/**
	* Da de baja un operario
	* @param operario : Parametro de tipo operario que representa el
operario a eliminar
	*/
	}
```

GestionDePersonal → bajaOperario(Operario operario)

Escenario

Nro escenario	Descripción
1	Lista de operarios con al menos 1 operario {operario1}.
2	Lista de operarios vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Operario	Operario es instancia de la clase Operario (1)	Operario==null (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	{operario1}	Da de baja el operario de la lista de operarios.	1

2	Incorrecta	{operario2} Escenario 1	Error	1
3	Incorrecta	{null}	Error	2

```
/**
	* Modifica un parametro de un operario , recibe un String
{nombreApellido,nombreUsuario,password,nacimiento} y en base a ello modifica
el atributo
	* pre: accion!=null
	* @param operario : Parametro de tipo operario que representa el
operario a modificar
	* @param accion : Parametro de tipo String que representa el
parametro a modificar
	* @param valor : Parametro de tipo String que representa el valor del
parametro a modificar
	*/
```

GestionDePersonal → modificaOperario(Operario operario, String accion, String valor)

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Operario	operario!=null (1)	operario==null(2)
Valor	valor!=null(3)	valor==null(4)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	{operario,”nombreApellido”,”hola”}	Modifica los parámetros del operario	1,3
2		{operario,”nombreUsuario”,”hola”}	Modifica los parámetros del operario	1,3
3		{operario,” password”,”hola”}	Modifica los parámetros del operario	1,3
4		{operario,” nacimiento”,”hola”}	Modifica los parámetros del operario	1,3
5	Incorrecta	{null,” nacimiento”,”hola”}	Error	2,3
6		{operario,” nacimiento”,null}	Error	1,4

```
/**
```

```
	* Da de alta un mozo y setea el id del mozo con id del ultimo mozo
agregado mas 1. Si no hay mozos agregados setea id en 20000
	* @param nombreApellido : Parametro que representa el nombre y
apellido del mozo
	* @param nacimiento : Parametro que representa la fecha de nacimiento
del mozo
	* @param cantHijos : Parametro que representa la cantidad de hijos
del mozo
	*/
```

GestionDePersonal → altaMozo(String nombreApellido, String nacimiento, int cantHijos)

Escenario

Nro escenario	Descripción
1	Lista de mozos con al menos 1 mozo {Mozo, id=5}
2	Lista de mozos vacia.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
nombreApellido	nombreApellido!=null (1)	nombreApellido==null(2)
nacimiento	nacimiento!=null y con fecha de nacimiento de una persona mayor a 18 años(3)	Nacimiento==null(4.1) Nacimiento con fecha de una persona menor a 18 años (4.2)
CantHijos	cantHijos>=0 (5)_	CantHijos<0 (6)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	{“NahuelNievas”,”14/01/2000”, 5} Ambos escenarios	Se crea un mozo con su id y se agrega a la lista con id 6	1,3,5
2	Incorrecta	{null,”14/01/2000”, 5} Escenario 1	Error	2,3,5

3		{“NahuelNievas”, null, 5} Escenario 1	Error	1,4.1,5
4		{“NahuelNievas”,”14/01/2021”, 5} Escenario 1	Error	1,4.2,5
5		{“NahuelNievas”,”14/01/2000”, -3} Escenario 1	Error	1,3,6

```
/**
	* Da de baja un mozo
	* @param mozo : Mozo a eliminar
	*/
```

GestionDePersonal → bajaMozo(Mozo mozo)

Escenario

Nro escenario	Descripción
1	Lista de mozos con al menos 1 mozo {Mozo1}
2	Lista de mozos vacia.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Mozo	Mozo!=null (1)	Mozo==null (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	{mozo1}	Se borra el mozo de la lista	1
2	Incorrecta	Mozo=null	Error	2
3		{Mozo2}	Error	1

```
/**
	* Modifica un parametro de un mozo a excepcion de la cantidad de
hijos
	* pre: acción no puede ser nula y debe ser {“nombreApellido”,
“estado”)
	* @param mozo : Mozo a modificar
	* @param accion : String que representa el parametro a modificar
	* @param valor : String que representa el valor del parametro a
modificar
	*/
```

GestionDePersonal → modificaMozo(Mozo mozo, String accion, String valor)

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
mozo	mozo!=null (1)	mozo==null(2)
Valor	valor!=null(3)	valor==null(4)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	{mozo,”nombreApellido”,”hola”}	Modifica el nombre del mozo	1,3
2		{mozo,” estado”,”hola”}	Modifica el estado del mozo	1,3
3	Incorrecta	{null,”estado”,”hola”}	Error	2,3
4		{mozo,” estado”,null}	Error	1,4

```
	/**
	* Modifica la cantidad de hijos de un mozo. En acción debe recibir el
String “cantHijos”
	* @param mozo : Mozo a modificar
	* @param accion : String que representa el parametro a modificar
	* @param valor : String que representa el valor del parametro a
modificar
	*/
```

GestionDePersonal → modificaMozo(Mozo mozo, String accion, int valor)

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
mozo	mozo!=null (1)	mozo==null(2)
Valor	Valor>=0 (3)	Valor<0 (4)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	{mozo,"cantHijos",3}	Modifica la cantidad de hijos del mozo	1,3
2	Incorrecta	{null,"cantHijos",3}	Error	2,3
3		{mozo,"cantHijos",-4}	Error	1,4

CLASE ConfiguracionDeSistema

```
/**
       * Se da de alta una mesa en la lista de mesas del sistema. Si la
lista de mesas del sistema esta vacia se le agrega como
       * id al producto el prefijo de mesas del local. Si la lista tiene al
menos 1 mesa, se le suma 1 al id de la ultima mesa de la lista.
       *
       */
```

ConfiguracionDeSistema → altaMesa()

Escenario

Nro escenario	Descripción
1	Lista de productos con al menos 1 producto.
2	Lista de productos vacía.

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas

1	Correcta	Escenario 1	Alta de una mesa con id un número mayor al último de la lista.	
2	Correcta	Escenario2	Alta de una mesa con id igual al prefijo de mesas del local	2, 3, 5, 6, 9

```
/**
        * Se elimina una mesa pasada por parametro de la lista de mesas
        *
        * @param mesa: Parametro de tipo mesa a eliminar
        */
```

ConfiguracionDeSistema → bajaMesa(Mesa mesa)

Escenario

Nro escenario	Descripción
1	Lista de mesas con al menos 1 mesa
2	Lista de mesas vacía

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
mesa	mesa es instancia de clase Mesa (1)	mesa = null (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(mesa1) Escenario 1	Da de baja la mesa de la lista de mesas.	1
2	Incorrecta	(mesa2) Escenario 1	Error	1
3		Mesa ==null	Error	2

mesa1 se encuentra en la lista.

mesa2 no se encuentra en la lista.

Por SRS:

valor >= 2 cuando nromesa > 0

```
/**
	*
	* Se modifica una mesa de acuerdo a la accion especificada y se settea con parámetro valor. Accion permitida es "comensales", en otro caso, no hace nada. Valor debería ser un entero mayor a cero.
	* @param mesa: Parametro de tipo mesa a modificar
	* @param accion: Parametro tipo String que indica tipo de modificacion de mesa a realizar.
	* @param valor: Parametro tipo entero que indica la cantidad de comensales de la mesa a settear.
	*/
```

ConfiguracionDeSistema → modificaMesa(Mesa mesa, String accion, int valor)

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
mesa	mesa es instancia de clase Mesa (1)	mesa = null (2)
accion	accion = “comensales” (3)	accion != “comensales” (4) accion = null (5)
valor	valor > 0 (6)	valor <= 0 (7)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(mesa1, “comensales”, 2)	Se modifica la cantidad de comensales con valor 2 de la mesa1.	1, 3, 6
2	Incorrectas	(mesa1, “cubiertos”, 3)	Error	1, 4, 6
3		(null, “comensales”, 2)	Error	2, 3, 6
4		(mesa1, null, 2)	Error	1, 5, 6

5		(mesa1, “comensales”, -1)	Error	1, 3, 7

Por SRS:

valor = “estado” || valor = “libre”

Por documentación:

```
/**
	*
	* Se modifica el estado de una mesa y se settea con el valor pasado por parametro. La accion debería ser estado, en otro caso.
	* @param mesa: Parametro de tipo mesa distinto de null
	* @param accion: Parametro tipo String que indica tipo de modificacion de mesa a realizar.
	* @param valor: Parametro tipo entero que indica la cantidad de comensales de la mesa a settear.
	*/
```

ConfiguracionDeSistema → modificaMesa(Mesa mesa, String accion, String valor)

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
mesa	mesa es instancia de clase Mesa (1)	mesa = null (2)
accion	accion = “estado” (3)	accion != “estado” (4) accion = null (5)
valor	valor = “Libre” (6) valor = “Ocupada” (7)	valor != “Libre” (8) valor != “Ocupada” (9) valor = null (10)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(mesa1, “estado”, “Libre”)	Se modifica el estado de la mesa1 a libre.	1, 3, 6
2		(mesa1, “estado”, “Ocupada”	Se modifica el estado de la mesa1 a ocupada.	1, 3, 7

3	Incorrectas	(mesa1, “cantidad”, “Libre”)	Error	1, 4, 6
4		(mesa1, “estado”, “Nada”)	Error	1, 3, 8, 9
5		(null, “estado”, “Libre”)	Error	2, 3, 6
6		(mesa1, null, “Ocupada”)	Error	1, 5, 7
7		(mesa1, “estado”, null)	Error	1, 3, 10

Por SRS:

precioCosto > 0

precioVenta > 0

precioCosto > precioVenta.

Stock > 0

Por documentación:

```
/**
	* Se da de alta un producto en la lista de productos del sistema. Si la lista de productos del sistema esta vacia se le agrega como
	* id al producto el prefijo de producto del local. Si la lista tiene al menos 1 producto, se le suma 1 al id del ultimo
	* producto de la lista.
	*
	* @param stock: Parametro de tipo entero que representa el stock inicial del producto
	* @param nombre: Parametro de tipo string que representa el nombre del producto.
	* @param precioCosto: Parametro de tipo float que representa el precio de costo del producto.
	* @param precioVenta: Parametro de tipo float que representa el precio de venta del producto.
	*/
```

ConfiguracionDeSistema → altaProducto(int stock, String nombre, float precioCosto, float precioVenta)

Escenario

Nro escenario	Descripción
1	Lista de productos con al menos 1 producto.
2	Lista de productos vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
stock	stock > 0 (1)	stock <= 0 (2)
nombre	nombre != null (3)	nombre = null (4)
precioCosto	precioCosto > 0 (5) precioCosto < precioVenta (6)	precioCosto <= 0 (7) precioCosto > precioVenta (8)
precioVenta	precioVenta > 0 (9)	precioVenta <= 0 (10)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(10, “Pepsi”, 150, 300) Escenario 1	Alta de producto con id un número mayor al último de la lista.	1, 3, 5, 6, 9
2	Incorrecta	(-1, “Fanta”, 150, 300) Escenario 1	Error	2, 3, 5, 6, 9
3		(13, null, 150, 300) Escenario 2	Error.	1, 4, 5, 6, 9
4		(13, “Coca”, -10, 300) Escenario 2	Error	1, 3, 7, 6, 9
5		(13, “Torta”, 100, -10) Escenario 2	Error	1, 3, 5, 8, 10
6		(13, “Torta”, 100, 50) Escenario 2	Error	1, 3, 5, 8, 9

```
/**
```

```
        * Se elimina un producto pasado por parametro de la lista de
productos.
        *
        * @param producto: Parametro de tipo producto a eliminar.
        */
```

ConfiguracionDeSistema → bajaProductos(Producto producto)

Escenario

Nro escenario	Descripción
1	Lista de productos con al menos 1 producto.
2	Lista de productos vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
producto	producto es instancia de la clase Producto. (1)	producto = null (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	producto1 Escenario 1	Se da de baja el producto de la lista.	1
2	Incorrecta	producto2 Escenario 2	Error	1
3		null Escenario 1	Error	2

producto1 se encuentra en la lista de productos.

producto2 no se encuentra en la lista de productos.

```
/**
        * Se modifica el stock de un producto pasado por parametro. La accion
debería ser stock. valor debería ser un valor mayor o igual a 0.
        * @param producto: Parametro de tipo Producto a modificar.
        * @param accion: Parametro de tipo string que indica atributo a
modificar.
        * @param valor: Parametro de tipo entero que indica el valor del
atributo a modificar.
        */
```

ConfiguracionDeSistema → modificaProducto(Producto producto, String accion, int valor)

Escenario

Nro escenario	Descripción

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
producto	producto es instancia de la clase Producto. (1)	producto = null (2)
accion	accion = “stock” (3)	accion != “stock” (4) accion = null (5)
valor	valor >= 0 (6)	valor < 0 (7)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(producto1, “stock”, 10)	Se modifica el stock del producto1 a 10.	1, 3, 6
2	Incorrecta	(producto1, “cantidad”, 10)	Error	1, 4, 6
3		(null, “stock”, 10)	Error	2, 3, 6
4		(producto1, null, 10)	Error	1, 5, 6
5		(producto1, “stock”, -10)	Error	1, 3, 7

```
/** Se modifican el nombre del producto pasado por parametro. La accion deberia ser nombre, en otro caso, no hace nada. valor debería ser distinto de null.
	* @param producto: Parametro de tipo Producto a modificar.
	* @param accion: Parametro de tipo string que indica atributo a modificar.
	* @param valor: Parametro de tipo String que indica el valor del atributo a modificar.
	*/
```

ConfiguracionDeSistema → modificaProducto(Producto producto, String accion, String valor)

Escenario

Nro escenario	Descripción

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
producto	producto es instancia de la clase Producto. (1)	producto = null (2)
accion	accion = “nombre” (3)	accion != “nombre” (4) accion = null (5)
valor	valor != null (6)	valor = null (7)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(producto1, “nombre”, “PepsiMax”)	Se modifica el nombre del producto1.	1, 3, 6
2	Incorrecta	(producto1, “etiqueta”, “PepsiMax”)	Error	1, 4, 6
3		(null, “nombre”, “Pepsi”)	Error	2, 3, 6
4		(producto1, null, “Pepsi”)	Error	1, 5, 6
5		(producto1, “nombre”, null)	Error	1, 3, 7

```
/** Se modifican los precios del producto pasado por parametro. La accion debería ser precioCosto o precioVenta. valor deberia ser mayor a 0.
	* @param producto: Parametro de tipo Producto a modificar.
	* @param accion: Parametro de tipo string que indica atributo a modificar.
	* @param valor: Parametro de tipo float que indica el valor del atributo a modificar.
	*/
```

ConfiguracionDeSistema → modificaProducto(Producto producto, String accion, float valor)

Escenario

Nro escenario	Descripción

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
producto	producto es instancia de la clase Producto. (1)	producto = null (2)
accion	accion = “precioCosto” (3) accion = “precioVenta” (4)	accion != “precioCosto” (5) accion != “precioVenta” (6) accion = null (7)
valor	valor > 0 (8)	valor <= 0 (9)

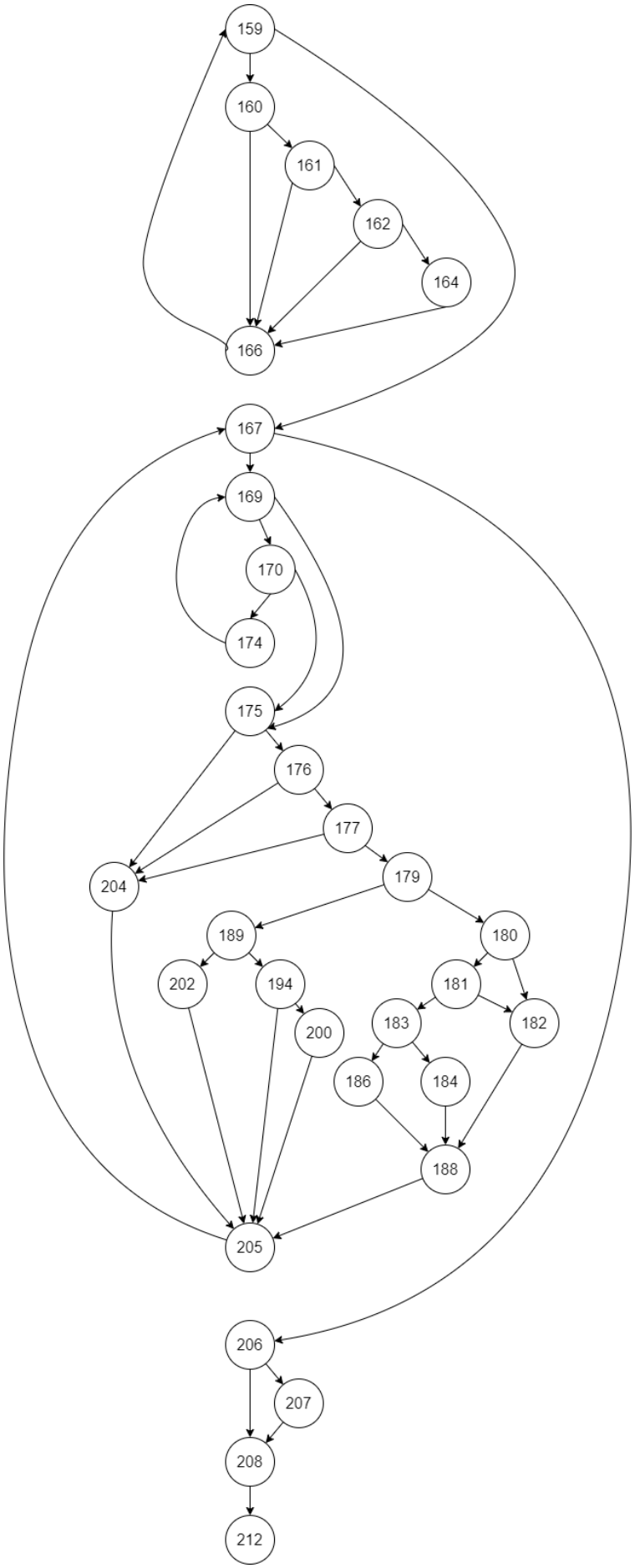
Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correcta	(producto1, “precioCosto”, 10)	Se modifica el precio de costo del producto1 a 10.	1, 3, 8
2		(producto1, “precioVenta”, 100)	Se modifica el precio de venta del producto1 a 100.	1, 4, 8
3	Incorrecta	(producto1, “precio”, 50)	Error	1, 5, 6, 8
4		(null, “precioCosto”, 50)	Error.	2, 3, 8
5		(producto1, “precioCosto”, -10)	Error	1, 3, 9
6		(producto1, null, 50)	Error	1, 7, 8

Test de caja blanca:

Método de la clase MetodosFacturacion del cual se realiza caja blanca:

GeneracionDeFactura();


159
160
161
162
164
166
167
169
170
174
175
176
177
204
179
189
180
202
194
181
183
182
200
186
184
188
205
206
207
208
212

Análisis complejidad ciclomática de McCabe:

V(G) = arcos – nodos + 2 = 47 – 31 + 2 = 18

V(G) = número de regiones = 18

V(G) = número de nodos condición + 1 = 17 + 1 = 18

Análisis de caminos por métodos simplificado:

N° camino independiente	Camino
C1	159-167-206-208-212
C2	159-160-166-159-167-206-208-212
C3	159-160-161-166-159-167-206-208-212
C4	159-160-161-162-166-159-167-206-208-212
C5	159-160-161-162-164-166-159-167-206-208-212
C6	159-167-169-175-204-205-167-206-208-212
C7	159-167-169-170-175-204-205-167-206-208-212
C8	159-167-169-170-174-169-170-175-204-205-167-206-208-212
C9	159-167-169-170-175-176-204-205-167-206-208-212
C10	159-167-169-170-175-176-177-204-205-167-206-208-212
C11	159-167-169-170-175-176-177-179-189-202-205-167-206-208-212
C12	159-167-169-170-175-176-177-179-189-194-205-167-206-208-212
C13	159-167-169-170-175-176-177-179-189-194-200-205-167-206-208-212
C14	159-167-169-170-175-176-177-179-180-182-188-205-167-206-208-212
C15	159-167-169-170-175-176-177-179-180-181-182-188-205-167-206-208-212
C16	159-167-169-170-175-176-177-179-180-181-183-186-188-205-167-206-208-212
C17	159-167-169-170-175-176-177-179-180-181-183-184-188-205-167-206-208-212
C18	159-160-161-162-164-166-159-167-206-207-208-212

Escenario 1:

Lista promociones temporales vacía.
Lista de pedidos de la comanda vacía.

Camino independiente	Parámetros de entrada	Resultado esperado
C1 159-167-206-208-212	Calendar.getInstance(), “Lunes”, comanda1, “efectivo”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda1, lista de

		pedidos vacía, total = 0. Método de pago efectivo, lista de promociones temporales aplicadas vacía, lista de promociones productos aplicadas vacía.

Escenario 2:

Lista de promociones temporales con una promoción temporal día Jueves en “ctaDNI” inactiva.
Lista de pedidos de la comanda vacía.

Camino independiente	Parámetros de entrada	Resultado esperado
C2 159-160-166-159-167-206-208-212	Calendar.getInstance(), “Lunes”, comanda1, “efectivo”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda1, lista de pedidos vacía, total = 0. Método de pago efectivo, lista de promociones temporales aplicadas vacía, lista de promociones productos aplicadas vacía. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado.
C3 159-160-161-166-159-167-206-208-212	Calendar.getInstance(), “Lunes”, comanda1, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda1, lista de pedidos vacía, total = 0. Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos aplicadas vacía. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado.

Escenario 3:

Lista de promociones temporales con una promoción temporal día Jueves en “ctaDNI” activa.
Lista de pedidos de la comanda vacía.

Camino independiente	Parámetros de entrada	Resultado esperado
C4 159-160-161-162-166-159-167-206-208-212	Calendar.getInstance(), “Lunes”, comanda1, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda1, lista de pedidos vacía, total = 0. Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos aplicadas vacía. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado.
C5 159-160-161-162-164-166-159-167-206-208-212	Calendar.getInstance(), “Jueves”, comanda1, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda1, lista de pedidos vacía, total = 0. Método de pago ctaDNI, lista de promociones temporales aplicadas con 1 promoción temporal, lista de promociones productos aplicadas vacía. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado.
C18 159-160-161-162-164-166-159-167-206-207-208-212	Calendar.getInstance(), “Jueves”, comanda1, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda1, lista de pedidos vacía, total = 0. Método de pago ctaDNI, lista de promociones temporales aplicadas con 1 promoción temporal, lista de promociones productos aplicadas vacía. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado.

comanda1 tiene lista de pedidos vacía.

Escenario 4:

Lista de promociones temporales vacía.
Lista de pedidos de la comanda con un pedido.
Lista de promociones de producto vacía.

Camino independiente	Parámetros de entrada	Resultado esperado
C6 159-167-169-175-204-205-167-206-208-212	Calendar.getInstance(), “Lunes”, comanda2, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda1, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida. Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos aplicadas vacía. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado y se actualiza su acumulado.

comanda2 tiene una lista de pedidos con 1 pedido.

Escenario 5:

Lista de promociones temporales vacía.
Lista de pedidos de la comanda con un pedido.
Lista de promociones de producto con 1 promoción de producto.

Camino independiente	Parámetros de entrada	Resultado esperado
C7 159-167-169-170-175-204-205-167-206-208-212	Calendar.getInstance(), “Martes”, comanda2, “ctaDNI” //Producto en la lista de pedidos es el mismo de promocion de producto pero distinto día.	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda2, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida, Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos aplicadas vacía.

		Se aumenta en 1 la cantidad de mesas atendida al mozo asignado y se actualiza su acumulado.
C8 159-167-169-170-174-169-175-204-205-167-206-208-212	Calendar.getInstance(), “Lunes”, comanda2, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda2, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida. Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos aplicadas vacía. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado y se actualiza su acumulado.
C9 159-167-169-170-175-176-204-205-167-206-208-212	Calendar.getInstance(), “Martes”, comanda2, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda2, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida. Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos aplicadas vacía. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado y se actualiza su acumulado.
C10 159-167-169-170-175-176-177-204-205-167-206-208-212	Calendar.getInstance(), “Martes”, comanda2, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda2, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida. Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos aplicadas vacía. Se aumenta en 1 la cantidad de mesas atendida al mozo

		asignado y se actualiza su acumulado.
C11 159-167-169-170-175-176-177-179-189-202-205-167-206-208-212	Calendar.getInstance(), “Martes”, comanda2, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda2, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida. Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos aplicadas vacía. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado y se actualiza su acumulado.
C12 159-167-169-170-175-176-177-179-189-194-205-167-206-208-212	Calendar.getInstance(), “Martes”, comanda2, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda2, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida. Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos aplicadas vacía. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado y se actualiza su acumulado.
C13 159-167-169-170-175-176-177-179-189-194-200-205-167-206-208-212	Calendar.getInstance(), “Martes”, comanda2, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda2, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida * %descuento promoción producto por cantidad mínima. Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos con 1 promoción de producto.

		Se aumenta en 1 la cantidad de mesas atendida al mozo asignado y se actualiza su acumulado.
C14 159-167-169-170-175-176-177-179-180-182-188-205-167-206-208-212	Calendar.getInstance(), “Martes”, comanda2, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda2, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida * %descuento promoción producto por 2X1. Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos con 1 promoción de producto. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado y se actualiza su acumulado.
C15 159-167-169-170-175-176-177-179-180-181-182-188-205-167-206-208-212	Calendar.getInstance(), “Martes”, comanda3, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda3, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida * %descuento promoción producto por 2X1 (pero no hace efecto). Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos con 1 promoción de producto. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado y se actualiza su acumulado.
C16 159-167-169-170-175-176-177-179-180-181-183-186-188-205-167-206-208-212	Calendar.getInstance(), “Martes”, comanda4, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda4, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida-1 * %descuento promoción producto por 2X1 + precio del producto pedido * cantidad pedida.

		Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos con 1 promoción de producto. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado y se actualiza su acumulado.
C17 159-167-169-170-175-176-177-179-180-181-183-184-188-205-167-206-208-212	Calendar.getInstance(), “Martes”, comanda5, “ctaDNI”	Devuelve una factura con la fecha pasada por parámetro, mesa de la comanda5, lista de pedidos con 1 pedido, total = precio del producto pedido * cantidad pedida * %descuento promoción producto por 2X1. Método de pago ctaDNI, lista de promociones temporales aplicadas vacía, lista de promociones productos con 1 promoción de producto. Se aumenta en 1 la cantidad de mesas atendida al mozo asignado y se actualiza su acumulado.

Errores detectados por Caja blanca:

-Caso de prueba Camino 18:

Es imposible acceder a las líneas 163-164 debido a que en el código se compara mediante la función equals() a un String con un Calendar.

Debido a esto, no se logra cambiar el valor del atributo bandera, por lo que tampoco se puede acceder a la línea 207.

-Caso de prueba Camino 11:

El estado de la mesa no cambia una vez hecha la factura.

Test de persistencia:

El test de persistencia se plantea por dos escenarios:

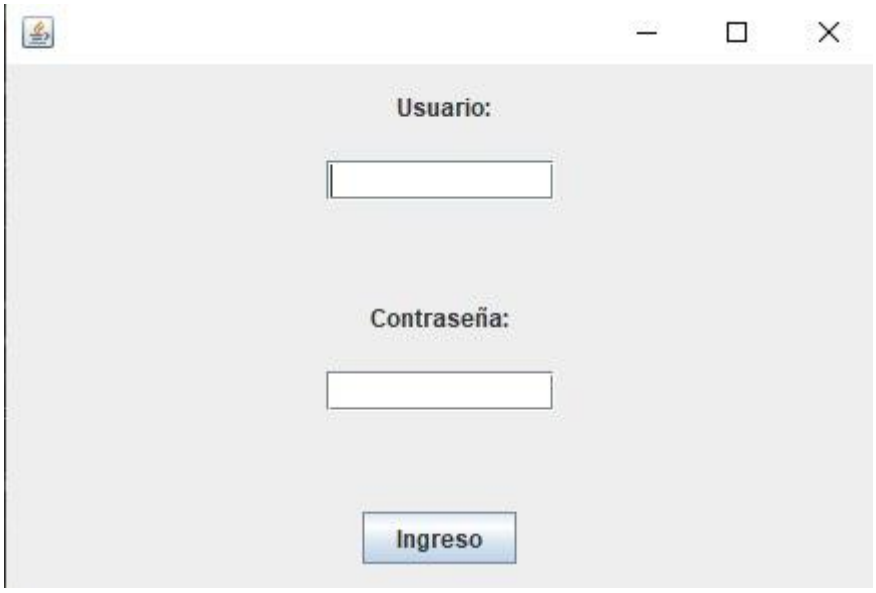
Nro escenario	Descripción
1	Archivo de persistencia inexistente
2	Archivo de persistencia existente

En el test se realiza el test de persistencia para cada uno de los atributos de la clase Local, a excepción de el atributo iniciarDia, ya que ese valor no requiere ser persistido.

Test de GUI:

El test de GUI utilizado se realizó con la clase java.awt.Robot. En este caso decidimos testear la clase VistaLogin debido a que es la interfaz que realiza validaciones en forma de excepciones.

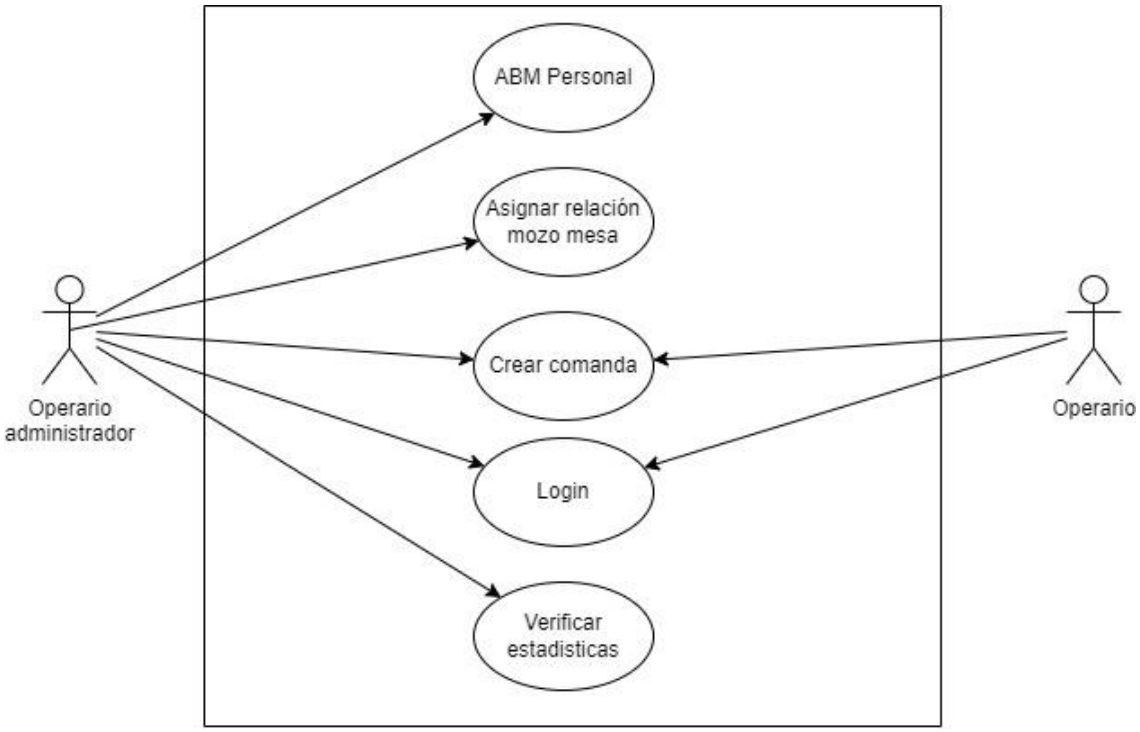
La vista es la siguiente:



Nro escenario	Descripción
1	Lista de operarios con al menos 1 operario.
2	Lista de operarios vacía.

Test de Integración:

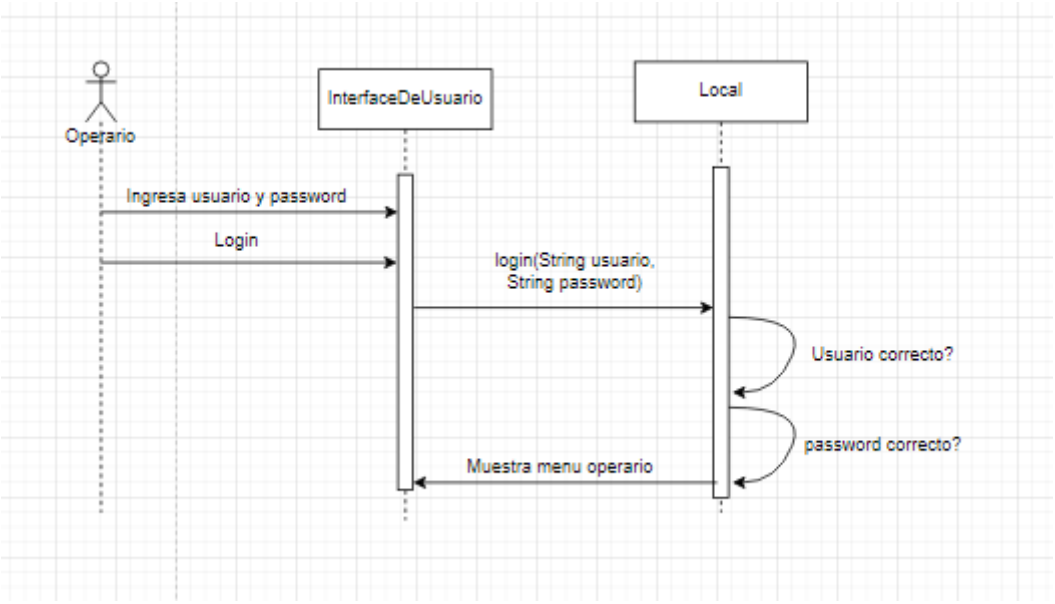
Para comenzar el test de integración se realizó el correspondiente diagrama de casos de uso



A partir de este diagrama se plantearon los casos de prueba en diagramas de secuencia:

Nota: Se realizó el test del caso de prueba de verificar estadísticas debido a que el resto fueron testeados en los test unitarios.

Login operario:



Escenario

Nro escenario	Descripción
1	Lista de operarios con al menos 1 operario registrado.
2	Lista de operarios vacia

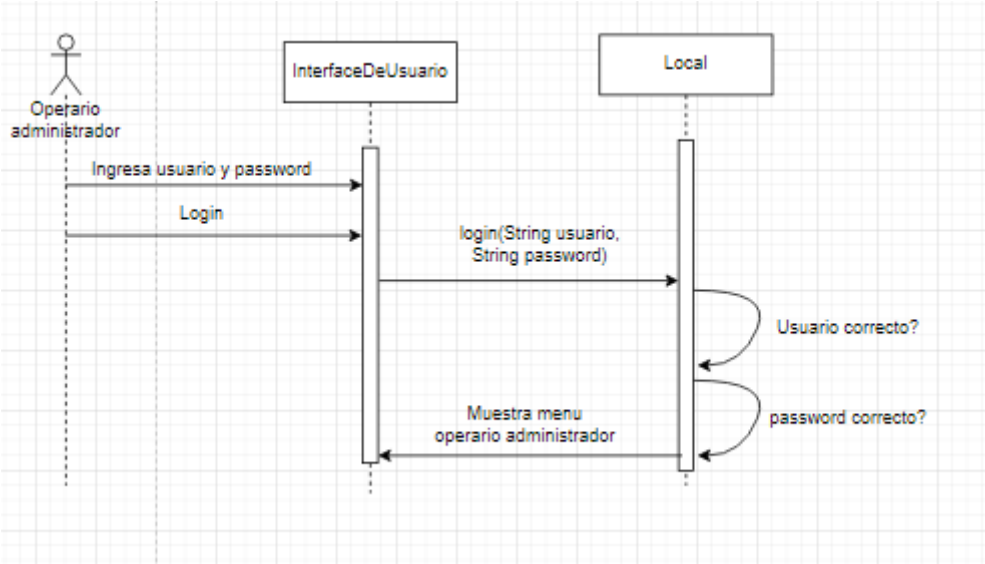
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Usuario y password	Usuario y password validos(1)	Usuario y password invalidos(2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Usuario y password validos	Muestra el menú operario	1
2	Incorrecta	Usuario==null, password==null	error	2

Login operario administrador:



Escenario

Nro escenario	Descripción
1	Operario administrador registrado
2	Operario administrador no registrado

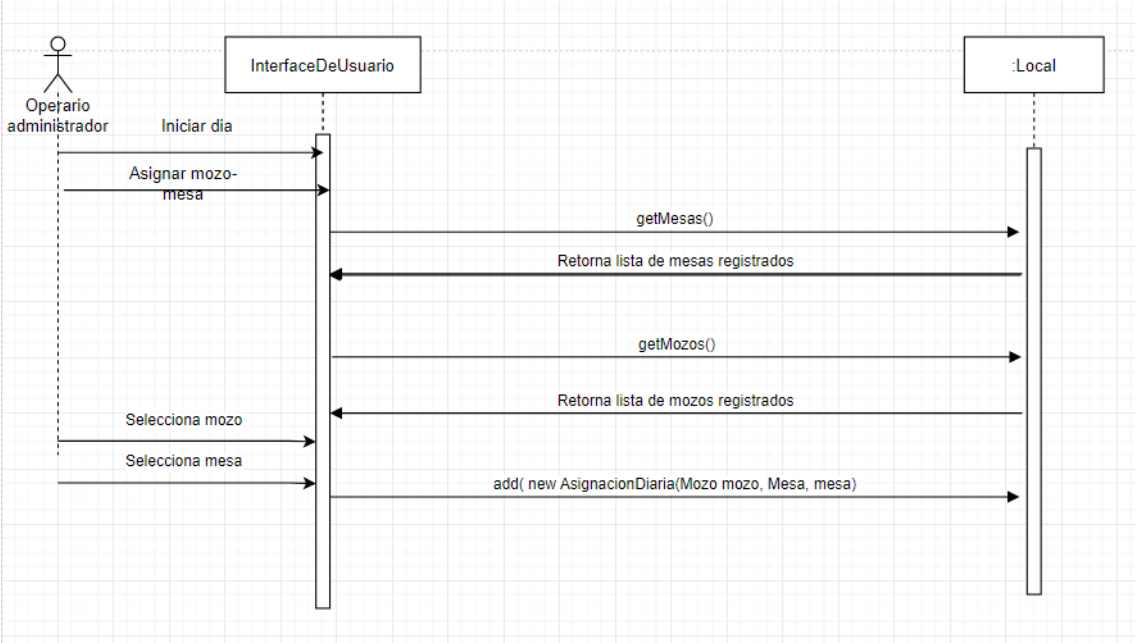
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Usuario y password	Usuario y password validos(1)	Usuario y password invalidos(2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Usuario y password validos	Muestra el menú operario administrador	1
2	Incorrecta	Usuario==null, password==null	error	2

Asignar mozo a mesa:



Escenario

Nro escenario	Descripción
1	Lista de mesas y mozos con al menos 1 mesa y mozo registrado.
2	Lista de mesas y mozos vacia

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Mozo	Mozo valido (1)	Mozo invalido (2)
Mesa	Mesa valida (3)	Mesa invalida (4)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Mesa1, Mozo2	Devuelve lista de mesas y mozos del Local. Crea una nueva relación en el local con la mesa y mozo seleccionado	1,3
2	Incorrecta	null, Mozo2	error	1,4
3	Incorrecta	Mesa1, null	error	2,3

Crear comanda y pedido:

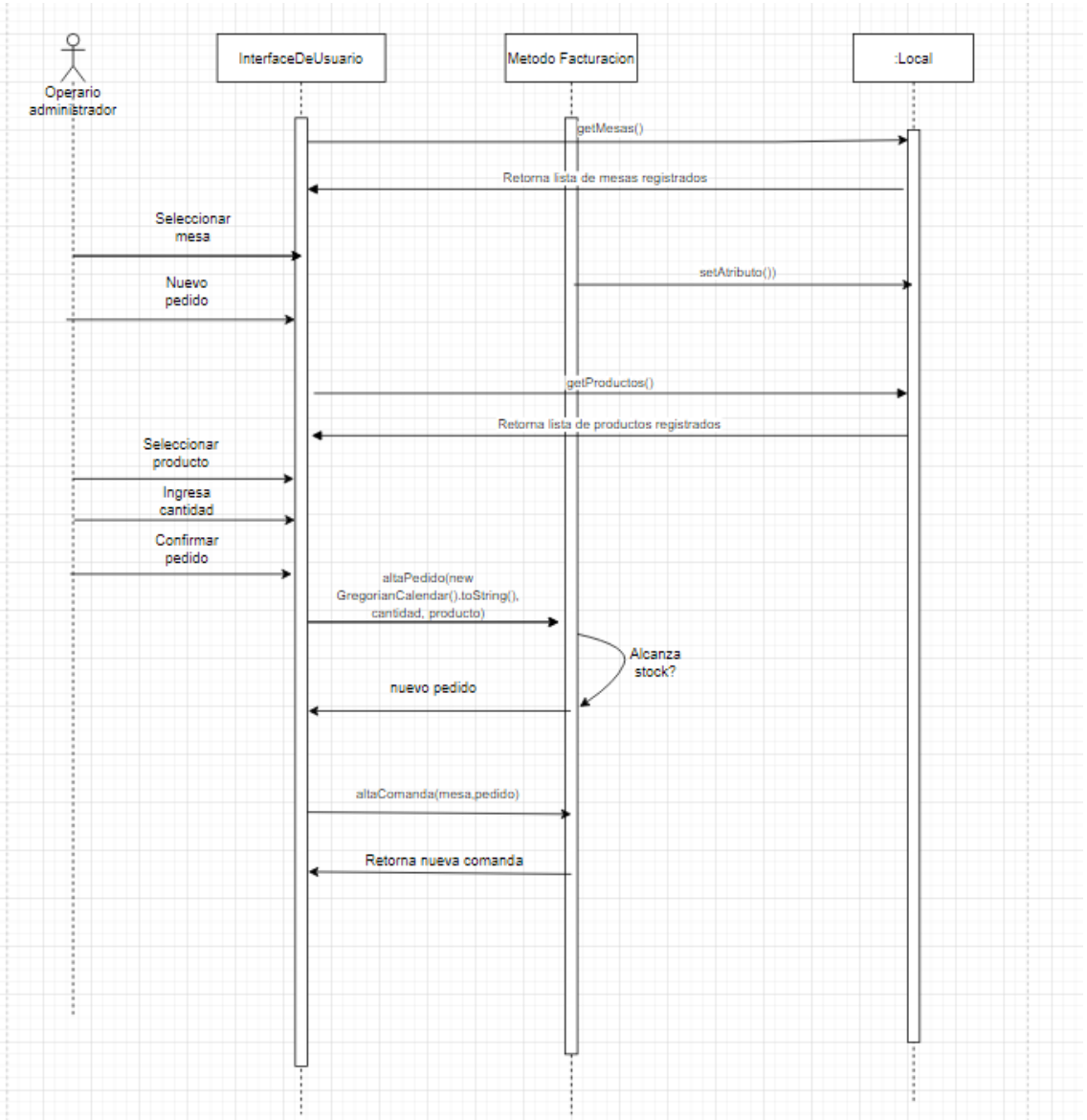


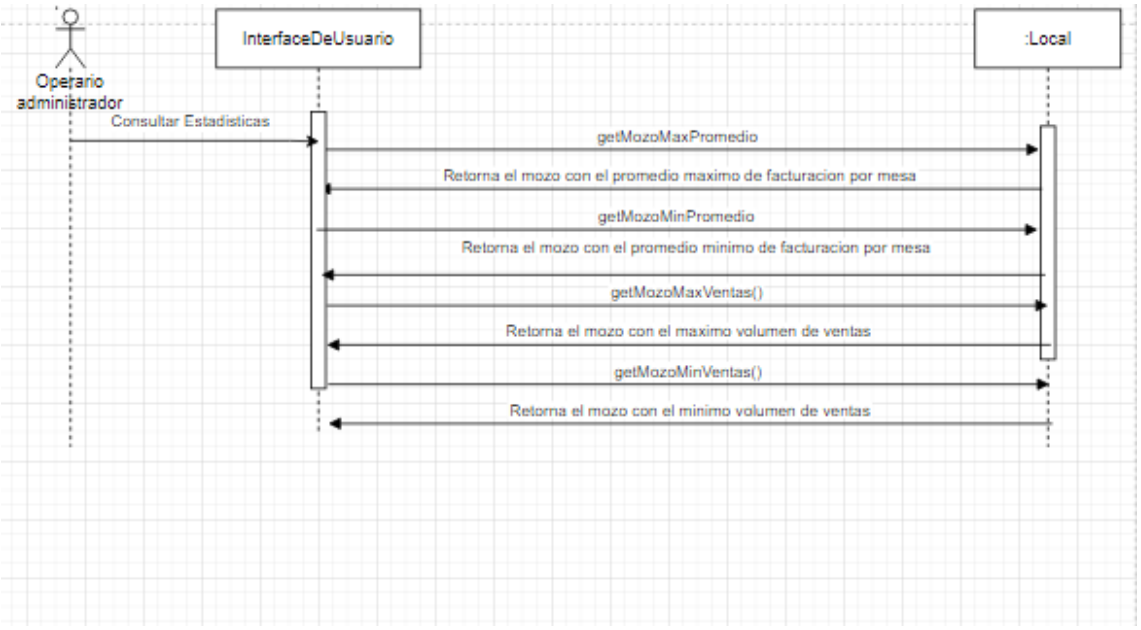
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Parametros de pedido	Parametros validos(1)	Parametros invalidos(2)
Mesa	Mesa valida (3)	Mesa invalida (4)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Mesa1, Producto1, cantidad=3	Devuelve lista de mesas del Local. Devuelve lista de productos del local. Da de alta un nuevo pedido. Devuelve una nueva comanda	1,3
2	Incorrecta	null, Producto1, cantidad=3	error	1,4
3	Incorrecta	Mesa1, null, cantidad=2	error	2,3

Consultar estadísticas:



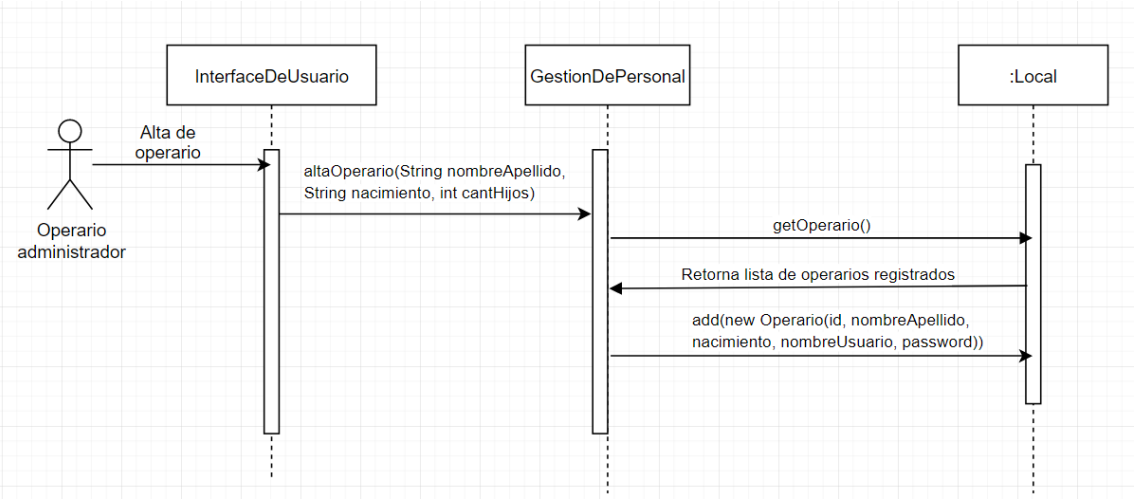
Escenario

Nro escenario	Descripción
1	Lista de mozos con al menos 1 mozo registrado.
2	Lista de mozos vacía.

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Escenario1	Muestra el mozo con el promedio máximo y minimo de facturación por mesa y el mozo con máximo y minimo volumen de ventas	
2	Incorrecta	Escenario2	No disponible	

Alta de operario:



Escenario

Nro escenario	Descripción
1	Lista de operarios con al menos 1 operario registrado.

2	Lista de operarios vacía.

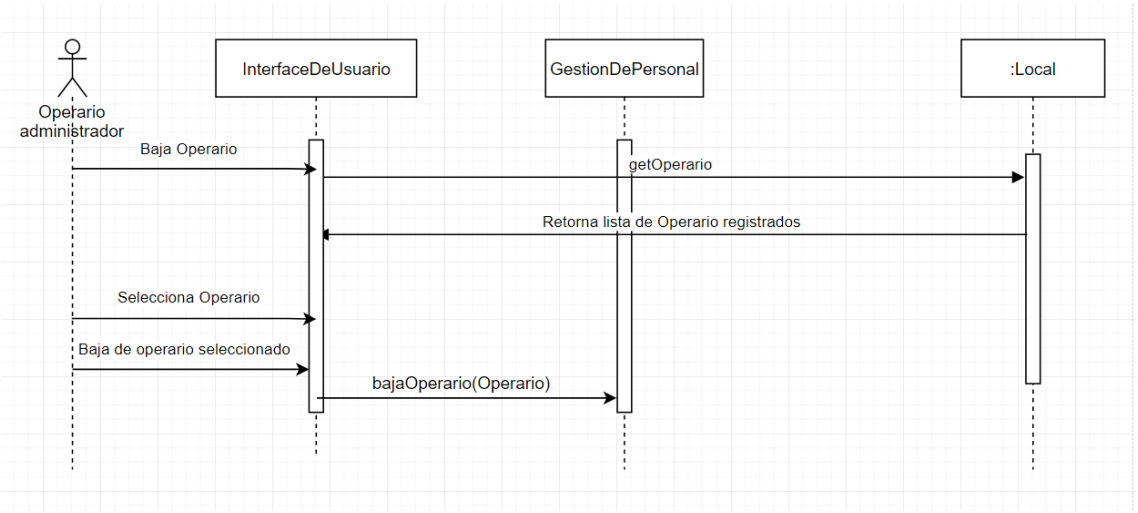
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Parametros de operario	Parametros validos (1)	Parametros invalidos (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	{“Nahuel”,”14/01/2000”, “Nahuel”, “Nahuel38}	Agrega operario al sistema	1
2	Incorrectas	{null,”14/01/2000”, “Nahuel”, “Nahuel38}	Error	2

Baja de operario:



Escenario

Nro escenario	Descripción
1	Lista de operarios con al menos 1 operario registrado.
2	Lista de operarios vacía.

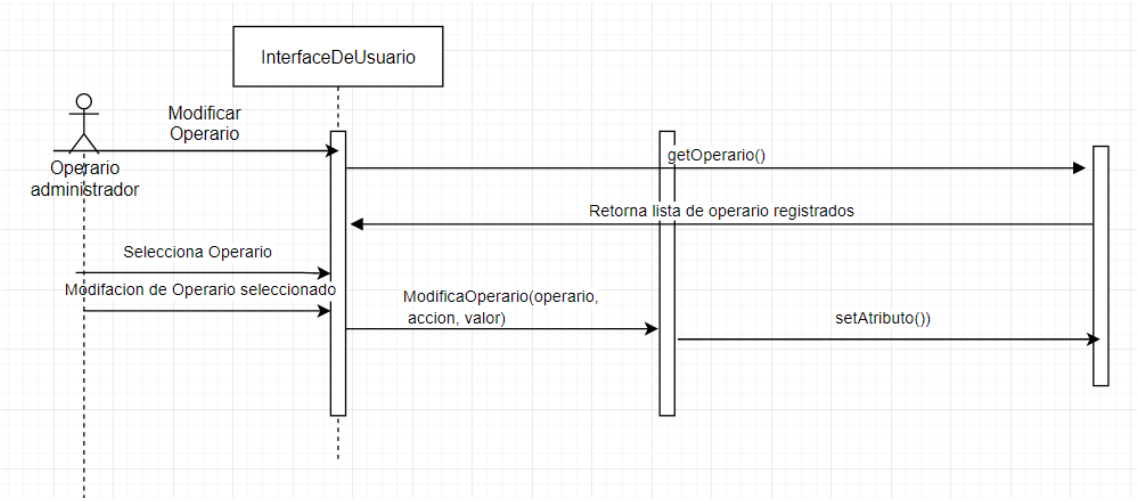
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Operario	Operario valido (1)	Operario invalido (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Operario1	Devuelve lista de operarios Registrados. Da de baja el operario seleccionado.	1
2	Incorrectas	null	Error	2

Modificación operario:



Escenario

Nro escenario	Descripción
1	Lista de operarios con al menos 1 operario registrado.
2	Lista de operarios vacía.

Tabla de particiones

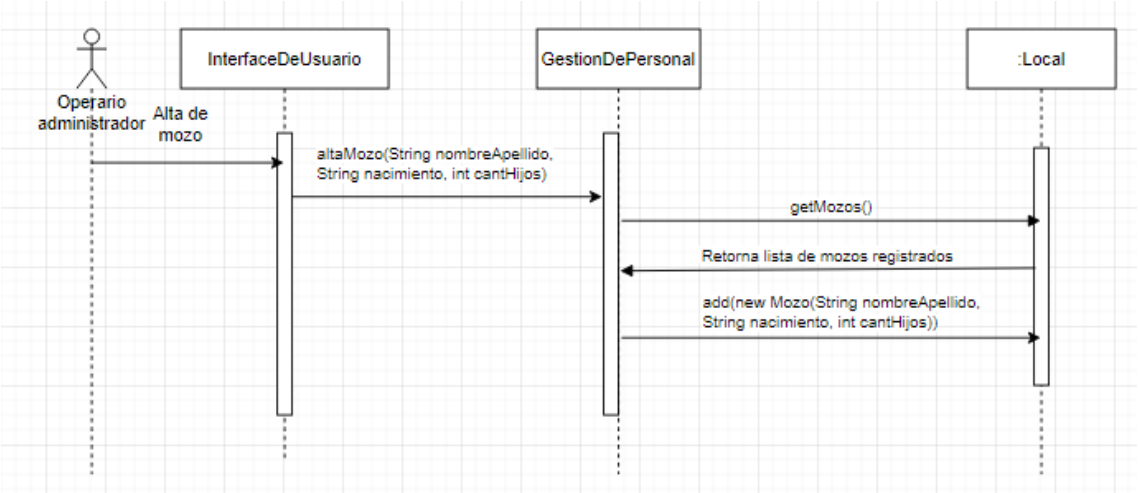
Condición de entrada	Clases válidas	Clases inválidas
Operario	Operario valido (1)	Operario invalido (2)

Accion	Accion valida (3)	Acción invalida (4)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Operario1, acción={nombreApellido}	Devuelve lista de operarios Registrados. Modifica el nombre de operario seleccionado.	1,3
2	Correctas	Operario1, acción={nombreUsuario}	Devuelve lista de operarios Registrados. Modifica el nombre de usuario de operario seleccionado.	1,3
3	Correctas	Operario1, acción={password}	Devuelve lista de operarios Registrados. Modifica el password de operario seleccionado.	1,3
4	Correctas	Operario1, acción={nacimiento}	Devuelve lista de operarios Registrados. Modifica el nacimiento de operario seleccionado.	1,3
5	Incorrectas	Null, acción={nacimiento}	Error	2,3
6	Incorrectas	Operario1, null	Error	1,4

Alta mozo:



Escenario

Nro escenario	Descripción
1	Lista de mozos con al menos 1 mozo registrado.
2	Lista de mozos vacía.

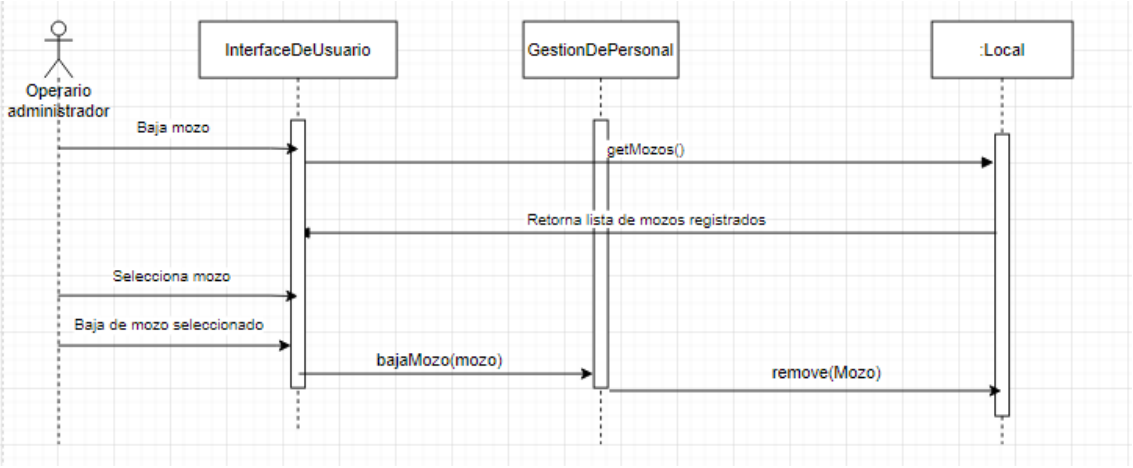
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Parametros de mozo	Parametros validos (1)	Parametros invalidos (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	{“NahuelNievas”,”14/01/2000”, 5}	Agrega mozo al sistema	1
2	Incorrectas	{null,”14/01/2000”, 5}	Error	2

Baja mozo:



Escenario

Nro escenario	Descripción
1	Lista de mozos con al menos 1 mozo registrado.
2	Lista de mozos vacía.

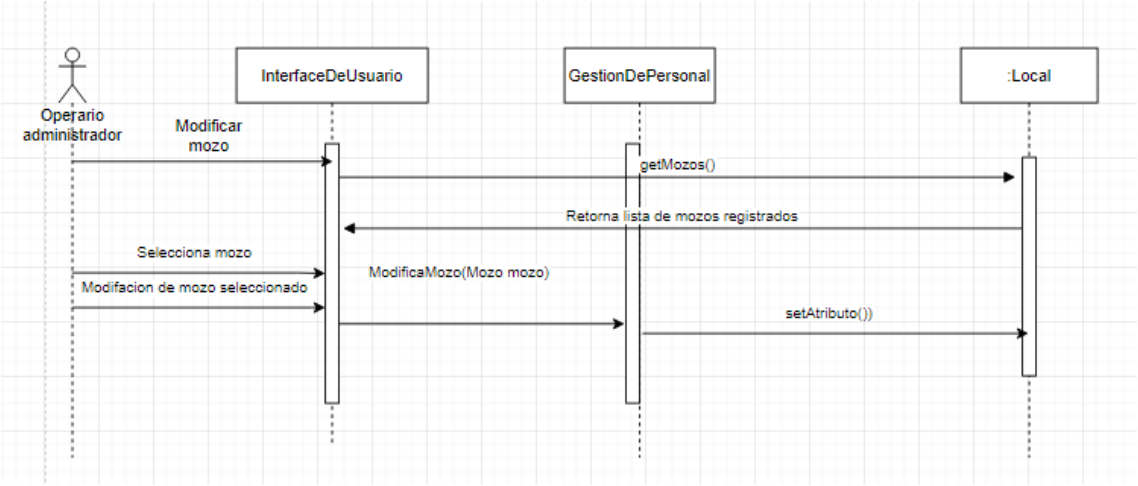
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Mozo	Mozo valido (1)	Mozo invalido (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Mozo1	Devuelve lista de mozos registrados. Da de baja el mozo seleccionado.	1
2	Incorrectas	null	Error	2

Modificar mozo:



Escenario

Nro escenario	Descripción
1	Lista de mozos con al menos 1 mozo registrado.
2	Lista de mozos vacía.

Tabla de particiones

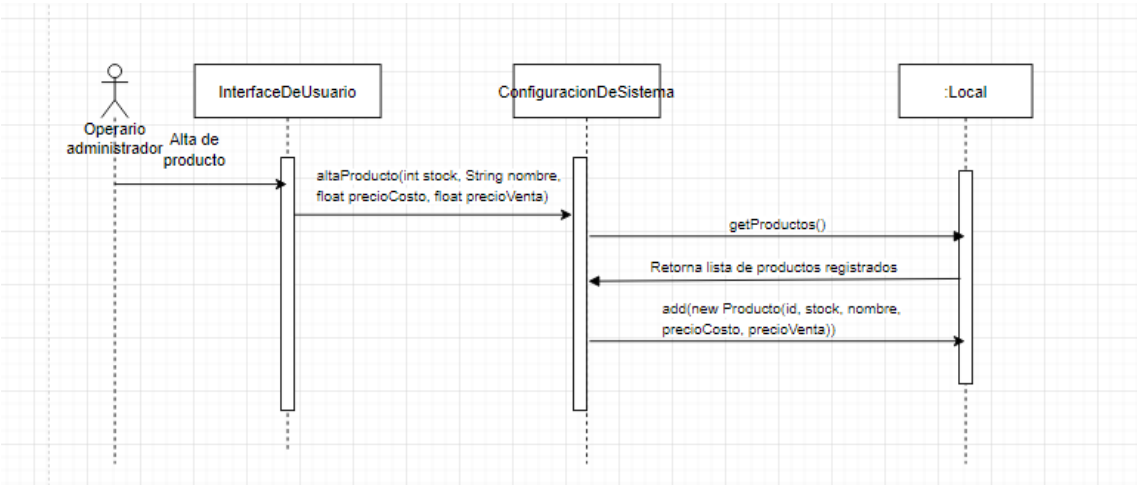
Condición de entrada	Clases válidas	Clases inválidas
Mozo	Mozo valido (1)	Mozo invalido (2)
Accion	Accion valida (3)	Acción invalida (4)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Mozo1, acción={nombreApellido}	Devuelve lista de mozo registrados. Modifica el nombre de mozo seleccionado.	1,3
2	Correctas	Mozo1, acción={estado}	Devuelve lista de mozo Registrados. Modifica el estado de mozo seleccionado.	1,3
3	Correctas	mozo1, acción={cantHijos}	Devuelve lista de mozos registrados.	1,3

			Modifica la cantidad de hijos de mozo seleccionado.	
4	Incorrectas	Null, acción={cantHijos}	Error	2,3
5	Incorrectas	Mozo1, null	Error	1,4

Alta de producto:



Escenario

Nro escenario	Descripción
1	Lista de productos con al menos 1 producto registrado.
2	Lista de mozos vacía.

Tabla de particiones

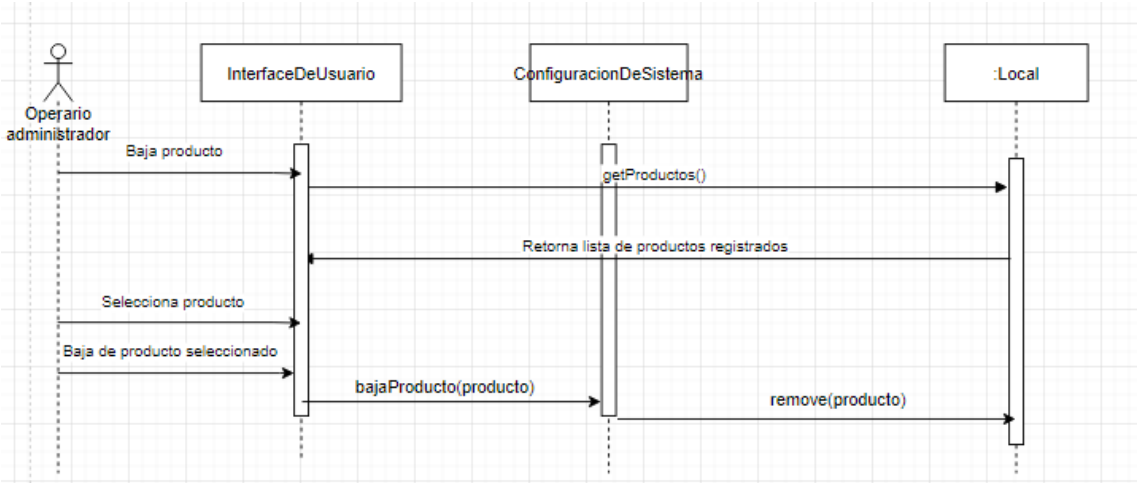
Condición de entrada	Clases válidas	Clases inválidas
Parametros de producto	Parametros validos (1)	Parametros invalidos (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(10, “Pepsi”, 150, 300)	Agrega producto al sistema	1

2	Incorrectas	(10, null, 150, 300)	Error	2

Baja de producto:



Escenario

Nro escenario	Descripción
1	Lista de Productos con al menos 1 Productos registrado.
2	Lista de Productos vacía.

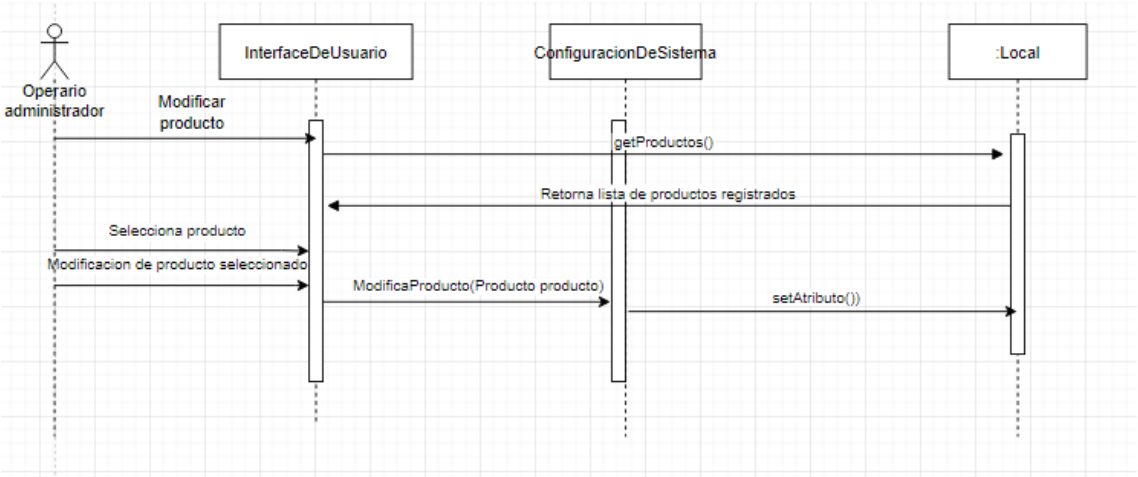
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Producto	Producto valido (1)	Producto invalido (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Producto1	Devuelve lista de Productos registrados. Da de baja el mozo seleccionado.	1
2	Incorrectas	null	Error	2

Modificacion de producto:



Escenario

Nro escenario	Descripción
1	Lista de Productos con al menos 1 Productos registrado.
2	Lista de Productos vacía.

Tabla de particiones

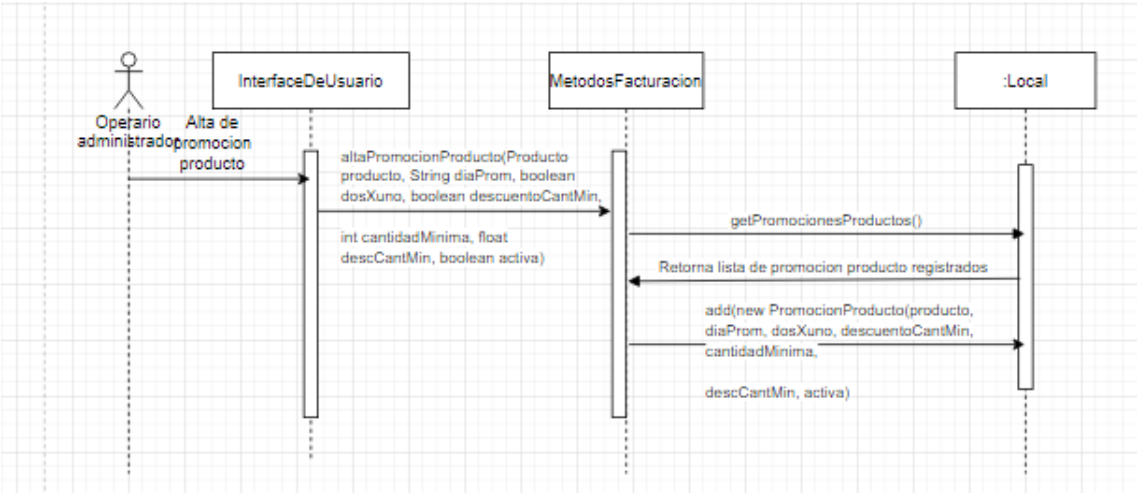
Condición de entrada	Clases válidas	Clases inválidas
Productos	Productos valido (1)	Productos invalido (2)
Accion	Accion valida (3)	Acción invalida (4)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Producto1, acción={stock}	Devuelve lista de productos registrados. Modifica el stock de producto seleccionado.	1,3
2	Correctas	Producto1, acción={nombre}	Devuelve lista de producto Registrados. Modifica el nombre de producto seleccionado.	1,3

3	Correctas	Producto1, acción={precioCosto}	Devuelve lista de Productos registrados. Modifica el precio de costo de producto seleccionado.	1,3
4	Correctas	Producto1, acción={precioVenta}	Devuelve lista de Productos registrados. Modifica el precio de venta de producto seleccionado.	1,3
5	Incorrectas	Null, acción={precioVenta}	Error	2,3
6	Incorrectas	Producto1, null	Error	1,4

Alta promoción producto



Escenario

Nro escenario	Descripción
1	Lista de promoción productos con al menos 1 promoción producto registrado.

2	Lista de promoción productos vacía.

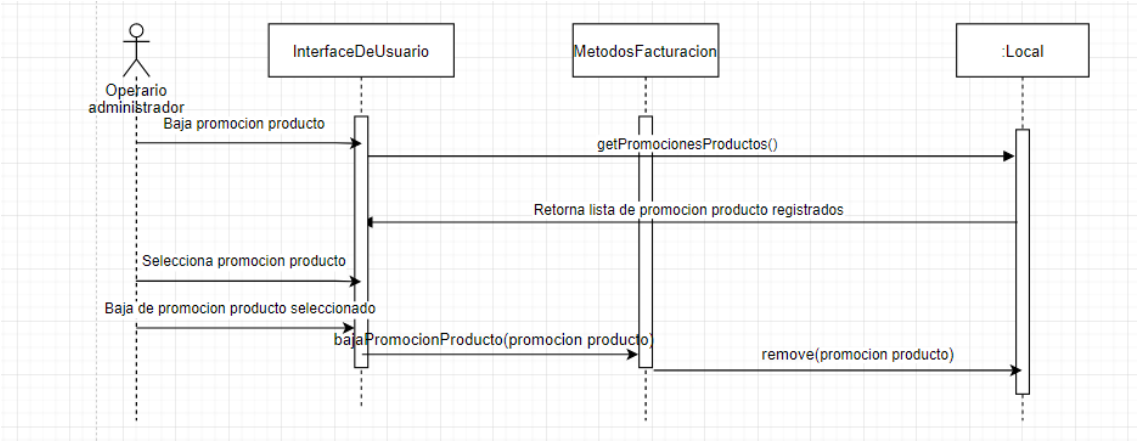
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Parametros de producto	Parametros validos (1)	Parametros invalidos (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(producto1, “Lunes”, true, true, 2, 50, true) Escenario 1	Agrega promoción producto al sistema	1
2	Incorrectas	(null, “Lunes”, true, true, 2, 50, true) Escenario 1	Error	2

Baja promoción producto



Escenario

Nro escenario	Descripción
1	Lista de promoción productos con al menos 1 promoción producto registrado.
2	Lista de promoción productos vacía.

Tabla de particiones

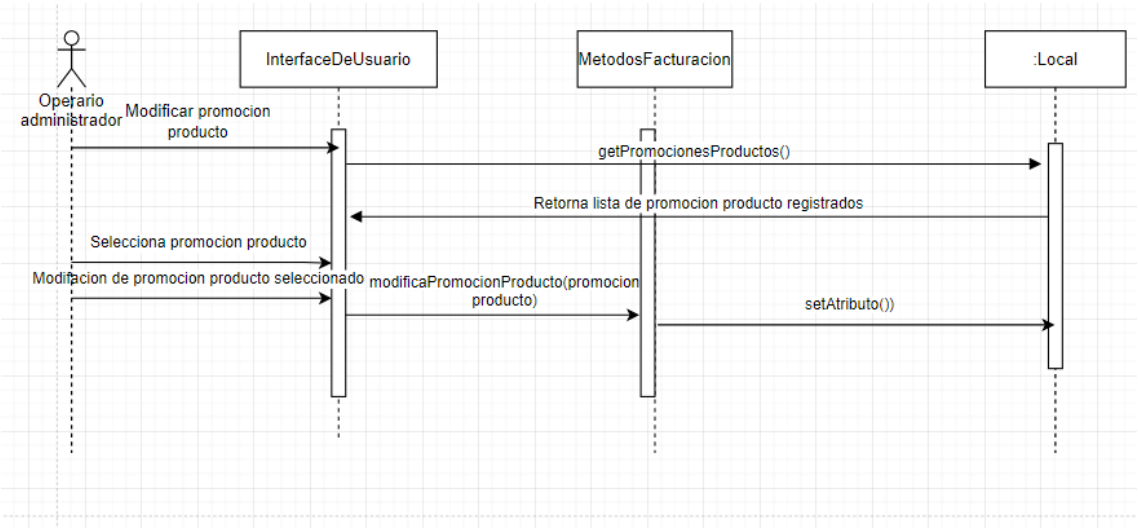
Condición de entrada	Clases válidas	Clases inválidas

Parametros de promoción producto	Parametros validos (1)	Parametros invalidos (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(10, “Pepsi”, 150, 300)	Agrega promoción producto al sistema	1
2	Incorrectas	(10, null, 150, 300)	Error	2

Modificación promoción producto



Escenario

Nro escenario	Descripción
1	Lista de promoción productos con al menos 1 promoción productos registrado.
2	Lista de promoción productos vacía.

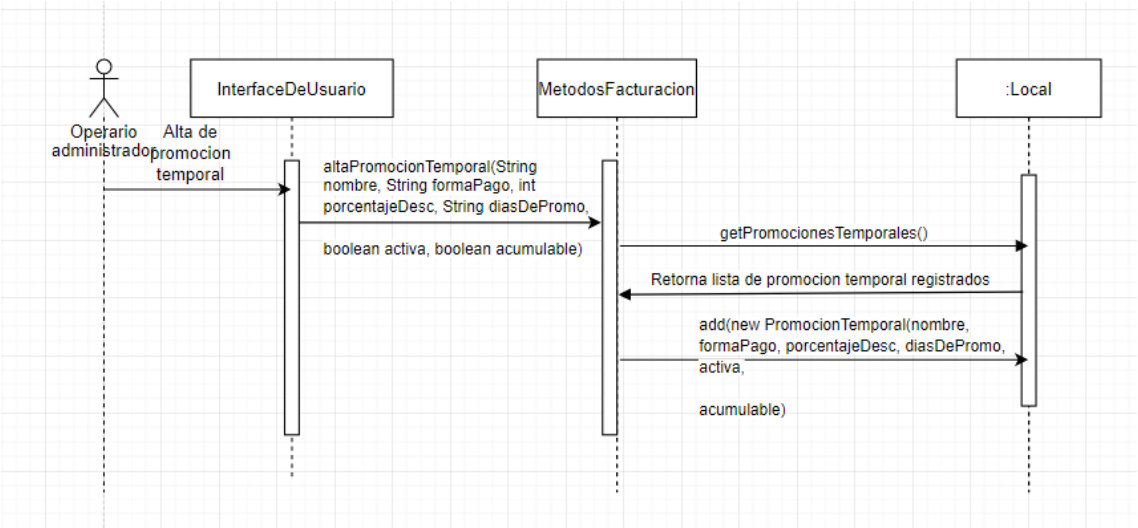
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Promoción productos	Promoción productos valido (1)	promoción productos invalido (2)
Atributos promoción producto	Atributos validos (3)	Atributos invalidos(4)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(promocion1, “Lunes”, true, true, 50, 2, true)	Devuelve lista de promoción producto registrados. Modifica los atributos de la promoción producto.	1,3
2	Incorrectas	(null, “Lunes”, true, true, 50, 2, true)	Error	2,3
3	Incorrectas	(promocion1, null,true, true, 50, 2, true)	Error	1,4

Alta promoción temporal



Escenario

Nro escenario	Descripción
1	Lista de promoción temporal con al menos 1 promoción temporal registrado.
2	Lista de promoción temporal vacía.

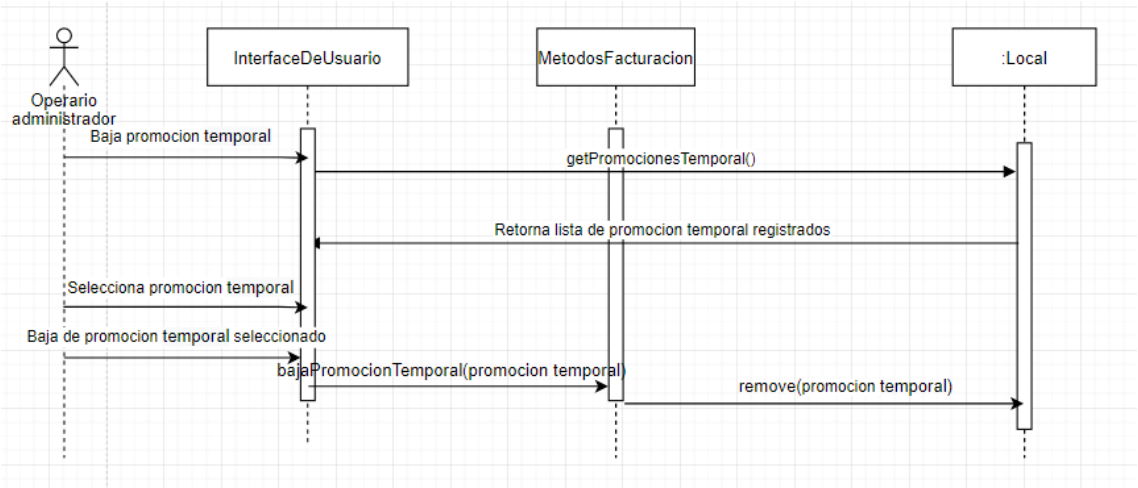
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Parametros de producto	Parametros validos (1)	Parametros invalidos (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(“Gran promo”, “efectivo”, 30, “Martes”, true, true)	Agrega promoción temporal al sistema	1
2	Incorrectas	(“Gran promo”,null, 30, “Martes”, true, true)	Error	2

Baja promoción temporal



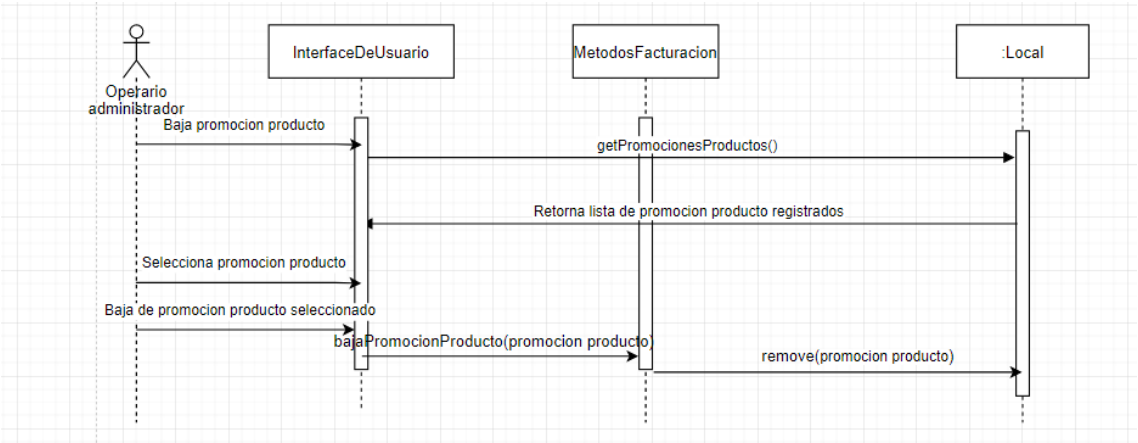
Condición de entrada	Clases válidas	Clases inválidas
Parametros de producto	Parametros validos (1)	Parametros invalidos (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(producto1, “Lunes”, true, true, 2, 50, true) Escenario 1	Agrega promoción producto al sistema	1

2	Incorrectas	(null, “Lunes”, true, true, 2, 50, true) Escenario 1	Error	2

Baja promoción producto



Escenario

Nro escenario	Descripción
1	Lista de promoción temporal con al menos 1 promoción temporal registrado.
2	Lista de promoción temporal vacía.

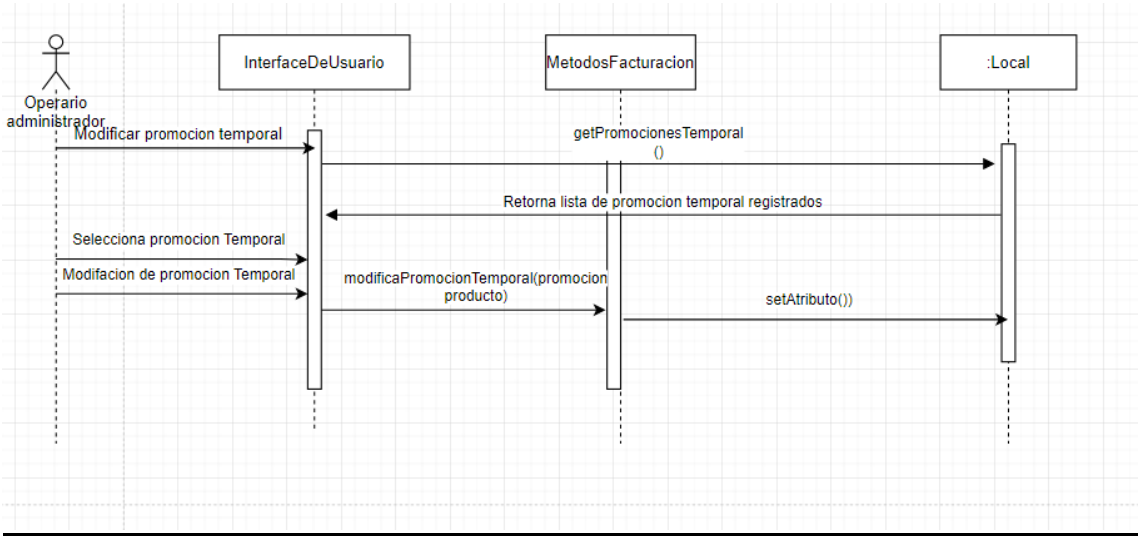
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Parametros de promoción temporal	Parametros validos (1)	Parametros invalidos (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(10, “Pepsi”, 150, 300)	Agrega promoción temporal al sistema	1
2	Incorrectas	(10, null, 150, 300)	Error	2

Modificacion promoción temporal



Escenario

Nro escenario	Descripción
1	Lista de promoción temporal con al menos 1 promoción temporal registrado.
2	Lista de Productos vacía.

Tabla de particiones

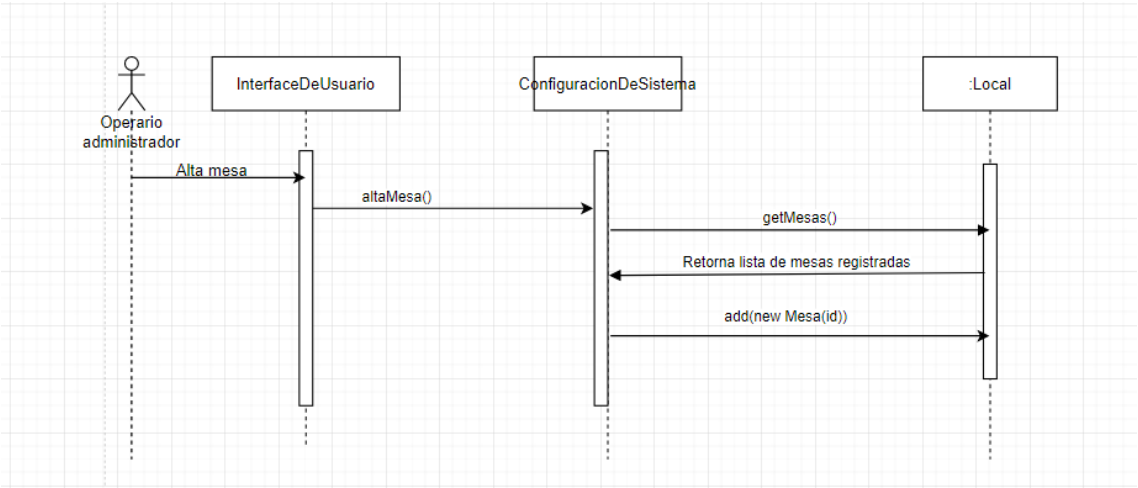
Condición de entrada	Clases válidas	Clases inválidas
Promoción temporal	Promoción temporal valido (1)	Promoción temporal invalido (2)
Atributos promoción temporal	Atributos validos (3)	Atributos invalidos(4)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(promo1, “Promo”, “ctaDNI”, 30, “Jueves”, true, true)	Devuelve lista de promoción temporal registrados. Modifica los atributos de la promoción temporal.	1,3

2	Incorrectas	(null, “Promo”, “ctaDNI”, 30, “Jueves”, true, true)	Error	2,3
3	Incorrectas	(promo1, null,c “ctaDNI”, 30, “Jueves”, true, true)	Error	1,4

Alta mesa



Escenario

Nro escenario	Descripción
1	Lista de mesas con al menos 1 mesas registrado.
2	Lista de mesas vacía.

Tabla de particiones

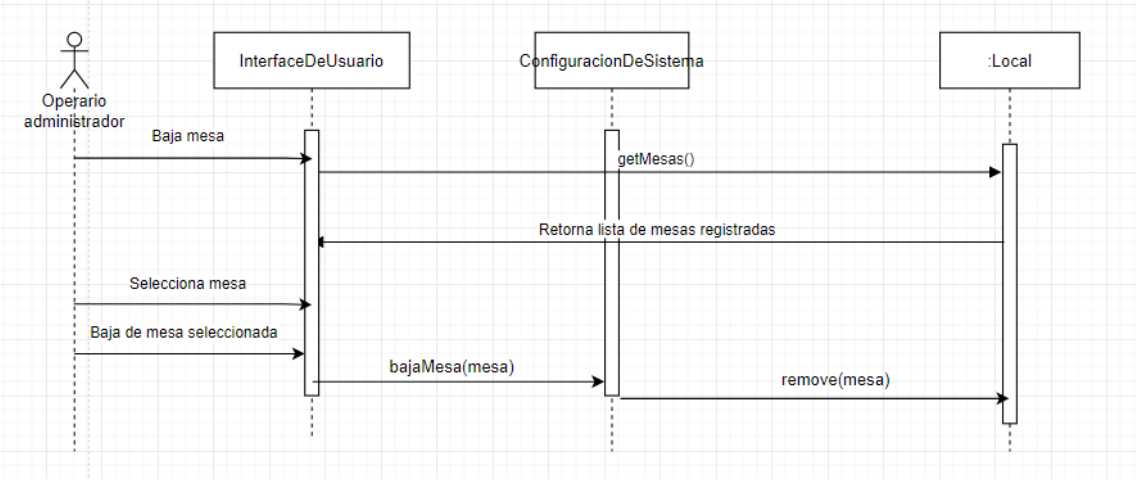
Condición de entrada	Clases válidas	Clases inválidas
Parametros de producto	Parametros validos (1)	Parametros invalidos (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(“Gran promo”, “efectivo”, 30, “Martes”, true, true)	Agrega promoción temporal al sistema	1

2	Incorrectas	(“Gran promo”,null, 30, “Martes”, true, true)	Error	2

Baja mesa



Escenario

Nro escenario	Descripción
1	Lista de mesa con al menos 1 mesa registrada.
2	Lista de mesas vacía.

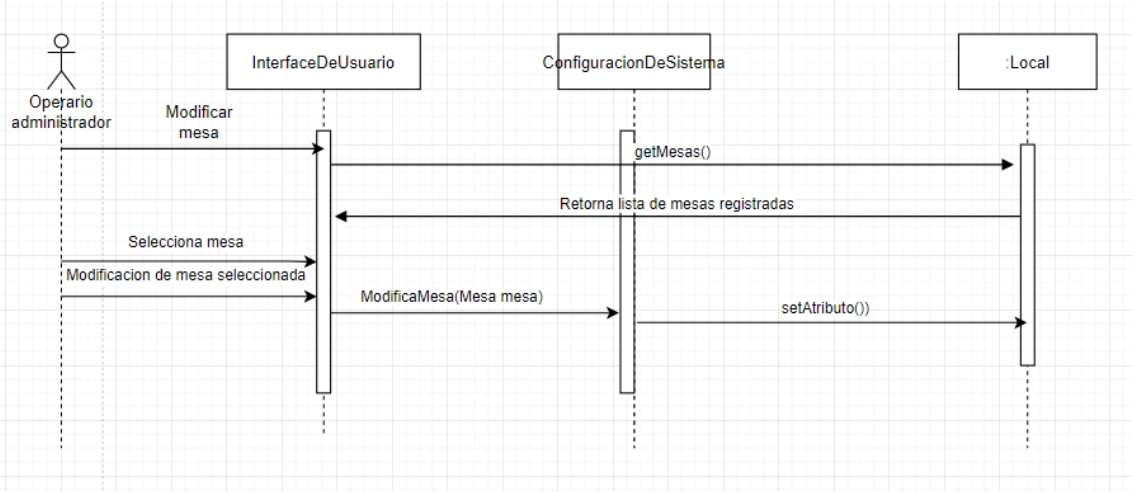
Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Parametros de mesa	Parametros validos (1)	Parametros invalidos (2)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	(10, “Pepsi”, 150, 300)	Agrega mesa al sistema	1
2	Incorrectas	(10, null, 150, 300)	Error	2

Modificacion mesa



Escenario

Nro escenario	Descripción
1	Lista de mesas con al menos 1 mesas registrado.
2	Lista de mesas vacía.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
mesas	mesas valido (1)	mesas invalido (2)
Accion	Accion valida (3)	Acción invalida (4)

Batería de pruebas:

Numero de caso de prueba	Tipo de clase (correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
1	Correctas	Mesa1, acción={comensales}	Devuelve lista de mesas registrados. Modifica la cantidad de comensales de mesa seleccionada.	1,3
2	Correctas	Mesa1, acción={estado}	Devuelve lista de mesas Registrados. Modifica el estado de mesa seleccionada.	1,3

3	Incorrectas	Null, acción={estado}	Error	2,3
4	Incorrectas	Mesa1, null	Error	1,4

Resultados:

- 286 métodos de test ejecutados – 165 fallas – 0 errores
  Caja negra:
  - Local: 90.4% cobertura - 21 métodos de test ejecutados – 12 fallas
  - GestionDePersonal: 97.8% cobertura - 64 métodos de test ejecutados – 38fallas
  - ConfiguracionDeSistema: 100% cobertura - 75 métodos de test ejecutados – 36 fallas
  - MetodosFacturacion: 88.5% cobertura - 86 métodos de test ejecutados – 49 fallas
- Cobertura:
  - Capa de negocios: 91.2%
  - Persistencia: 88.3%
  - Interfaz gráfica: 8,2%
  - Capa de datos: 72.3%
  - Excepciones: 0% (la excepción creada no se utiliza)

Conclusión:

Un código es imposible que esté 100% libre de errores pero habiendo utilizado las distintas herramientas de testing presentadas durante la cursada logramos realizar el test de un programa entregado por nuestros compañeros con el objetivo de reducir la mayor cantidad de estos errores. Logramos detectar distintos errores, fallas e inconsistencias con la documentación otorgada. Detectamos más fallas de lo esperado, pero consideramos que al ser la primera vez que se lleva a cabo el testeo del programa y siendo este test en su totalidad, es dentro de todo normal encontrar tantas fallas.

Hubiese sido más efectivo un enfoque de programación orientado al testing a fin de facilitar y mejor la fase de test. También hubiese sido efectivo realizar pruebas estáticas a medida que nuestros compañeros iban desarrollando su programa, con el fin de ahorrar retrabajo para la corrección de errores.