SISTEMA DE CERVECERÍA

TALLER DE PROGRAMACIÓN I
UNMDP

SUBGRUPO

-RODRÍGUEZ, JUAN GABRIEL
-UZQUIANO, TOMÁS EMANUEL

INTEGRANTES

-BOOLLS, NICOLÁS
-GUTIÉRREZ, ALAN
-RODRÍGUEZ, JUAN GABRIEL
-UZQUIANO, TOMÁS EMANUEL

ÍNDICE

Introducción

En el presente informe se detallarán los resultados obtenidos tras aplicar técnicas de testing en un sistema que simula el funcionamiento de una cervecería. El sistema de la cervecería fue desarrollado por otros dos alumnos de la materia Taller de Programación I , nuestro trabajo con el mismo fue aplicar los siguientes métodos de testing:

- **Pruebas de caja negra:** Basadas en verificar si el código cumple los contratos especificados, sin observar el código fuente que resuelve el problema
- **Pruebas de caja blanca:** Se analiza el código fuente, prestando atención a los posibles caminos que puede tomar el código por las estructuras de decisión.
- **Prueba de persistencia:** Se observará si los datos de la cervecería se almacenan y se recuperan correctamente.
- **Prueba de Interfaz Gráfica:** También llamado Test de GUI, consiste en revisar la parte gráfica del proyecto, específicamente en este trabajo se analiza el módulo de ventana login a la cervecería.
- **Prueba de Integración:** Diseñadas para probar la interacción entre los distintos componentes de un sistema.

Caja Negra

El presente trabajo se encuentra separado en distintas capas las cuales son : la capa de negocio, la de presentación y las de datos. Para el siguiente test de caja negra hemos decidido testear la capa más representativa al funcionamiento general del sistema, la cual es la capa de negocio. Por lo tanto, a continuación testeamos los métodos correspondientes a las clases de dicha capa.

Dichas clases son: ConfiguracionDeSistema , GestionDePersonal, Local, y por ultimo MetodosFacturacion.

Antes de empezar con el respectivo testeo nos percatamos de que en el paquete de excepciones solo existe una única Exception, por lo tanto para las salidas esperadas de las clases incorrectas decidimos marcarlas como Error, y dentro del código cortamos el flujo con una (Exception e) genérica.

Para las clases que fueron testeadas determinamos escenarios, tabla de particiones y batería de prueba para cada método de las mismas tal cual se muestra a continuación, acompañado de su respectiva documentación para cada método.

CLASE ConfiguracionDeSistema

-public void modificaMesa(Mesa mesa,String accion, int valor)

modificaMesa

```
public void modificaMesa(Mesa mesa,
                         java.lang.String accion,
                         int valor)
```

Se modifica una mesa de acuerdo a la accion especificada y se settea con parÃimetro valor. Accion permitida es "comensales", en otro caso, no hace nada. Valor deberÃa ser un entero mayor a cero.

Parameters:

`mesa` - : Parametro de tipo mesa a modificar

`accion` - : Parametro tipo String que indica tipo de modificacion de mesa a realizar.

`valor` - : Parametro tipo entero que indica la cantidad de comensales de la mesa a settear.

Escenarios

Nro Escenario	Descripción
1	Colección de Mesas con Mesa n°2 en la colección y n°0 en la colección

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
mesa	mesa valida(1)	mesa null(2)
accion	accion = "comensales"(3)	accion != "comensales"(4)
		accion null (5)
valor	valor >=2 (6)	valor <= 0 (7)
	valor >=0 && valor <2 (8)	

Batería de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{mesa[numero=2],"comensales",8}	Se modifica una mesa	1 - 3 - 6
	{mesa[numero=0],"comensales",1}	Se modifica una mesa	1 - 3 - 8
Incorrecta	{mesa[numero=0],"PEPA",1}	Error	1-4- 8
	{null,"comensales",2}	Error	2-3- 6
	{mesa[numero=2],null,8}	Error	1-5- 6
	{mesa[numero=2],"comensales",-1}	Error	1-3- 7

`public void modificaMesa(Mesa mesa,String action,String valor)`

modificaMesa

```
public void modificaMesa(Mesa mesa,
                         java.lang.String accion,
                         java.lang.String valor)
```

Se modifica el estado de una mesa y se settea con el valor pasado por parametro. La accion deberÃa ser estado, en otro caso.

Parameters:

`mesa` - : Parametro de tipo mesa distinto de null

`accion` - : Parametro tipo String que indica tipo de modificacion de mesa a realizar.

`valor` - : Parametro tipo entero que indica la cantidad de comensales de la mesa a settear.

Escenarios

Nro Escenario	Descripción
1	Colección de Mesas con Mesa n°2 en la colección

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
mesa	mesa valida(1)	mesa null(2)
accion	accion = "estado"(3)	accion != "estado"(4)
		accion null (5)
valor	valor ="libre" (6)	valor !="libre" && valor!="ocupada" (7)
	valor ="ocupada" (8)	valor null(9)

Batería de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{mesa[numero=2],"estado","libre"}	Se modifica una mesa	1 - 3 - 6

	{mesa[numero=2],"estado","ocupada"}	Se modifica una mesa	1 - 3 - 8
Incorrecta	{mesa[numero=2],"PEPA","libre"}	Error	1-4- 6
	{mesa[numero=2],"estado","PEPA"}	Error	1-3- 7
	NULL,"estado","libre"}	Error	2- 3- 6
	{mesa[numero=2],null,"libre"}	Error	1- 5- 6
	{mesa[numero=2],"estado",null}	Error	1- 3- 9

`public void altaProducto(int stock,String nombre,float precio Costo, float precioVenta)`

altaProducto

```
public void altaProducto(int stock,
                         java.lang.String nombre,
                         float precioCosto,
                         float precioVenta)
```

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.

Parameters:

`stock` - : Parametro de tipo entero que representa el stock inicial del producto

`nombre` - : Parametro de tipo string que representa el nombre del producto.

`precioCosto` - : Parametro de tipo float que representa el precio de costo del producto.

`precioVenta` - : Parametro de tipo float que representa el precio de venta del producto.

Escenarios

Nro Escenario	Descripción
1	Colección de productos con al menos un producto en la colección
2	Colección 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 <= 0 (6)
precioVenta	precioCosto < precioVenta (7)	precioCosto > precioVenta (8)
	precioVenta > 0 (9)	precioVenta <= 0 (10)

Batería de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{234,"Macchiato",100,400} Escenario 1	Se dio de alta nuevo producto	1- 3- 5- 7- 9
Incorrecta	{-5,"Macchiato",100,400}Escenario 1	Se modifica una mesa	2- 3- 5 -7- 9
	{234,null,100,400}Escenario 2	Error	1-4-5-7-9
	{234,"Macchiato",-100,400}Escenario 2	Error	1-3-6-7-9
	{234,"Macchiato",100,-400}Escenario 2	Error	1-3-5-8-10
	{234,"Macchiato",1000,4}Escenario 2	Error	1-3-5-8-9

```
public void bajaProductos(Producto producto)
```

bajaProductos

```
public void bajaProductos(Producto producto)
```

Se elimina un producto pasado por parametro de la lista de productos.

Parameters:

`producto` - : Parametro de tipo producto a eliminar.

Escenarios

Nro Escenario	Descripción
1	Colección de productos con al menos un producto en la colección
2	Colección de productos vacía

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
producto	producto valido(1)	producto null(2)

Batería de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{producto[] }Escenario 1	Se elimina el producto de la colección	1
Incorrecta	{producto2[] }Escenario 2	Error	1
	{null}Escenario 1	Error	2

```
public void modificaProducto(Producto producto, String
accion, int valor)
```

modificaProducto

```
public void modificaProducto(Producto producto,
                             java.lang.String accion,
                             int valor)
```

Se modifica el stock de un producto pasado por parametro. La accion deberÃa ser stock. valor deberÃa ser un valor mayor o igual a o.

Parameters:

producto - : Parametro de tipo Producto a modificar.

accion - : Parametro de tipo string que indica atributo a modificar.

valor - : Parametro de tipo entero que indica el valor del atributo a modificar.

Escenarios

Nro Escenario	Descripción
1	Colección de productos con al menos un producto en la colección
2	Colección de productos vacía

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
producto	producto valido(1)	producto null(2)
accion	accion = “stock” (3)	accion != “stock” (4)
		accion null (5)
valor	valor >= 0 (6)	valor < 0 (7)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{producto[],"stock", 245 }Escenario 1	Stock actualizado	1-3- 6
Incorrecta	{producto[],"stock", 245 }Escenario 2	Error	1-3- 6

	{producto[],"ASD", 245 }Escenario 1	Error	1-4- 6
	{null,"stock", 245 }Escenario 1	Error	2-3 -6
	{producto[],null, 245 }Escenario 1	Error	1-5- 6
	{producto[],"stock", -245 }Escenario 1	Error	1-3- 7

```
public void modificaProducto(Producto producto, String accion, String valor)
```

modificaProducto

```
public void modificaProducto(Producto producto,
                             java.lang.String accion,
                             java.lang.String valor)
```

Se modifican el nombre del producto pasado por parametro. La accion deberia ser nombre, en otro caso, no hace nada. valor deberÃa ser distinto de null.

Parameters:

`producto` - : Parametro de tipo Producto a modificar.

`accion` - : Parametro de tipo string que indica atributo a modificar.

`valor` - : Parametro de tipo String que indica el valor del atributo a modificar.

Escenarios

Nro Escenario	Descripción
1	Colección de productos con al menos un producto en la colección
2	Colección de productos vacía

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
producto	producto valido(1)	producto null(2)
accion	accion = “nombre” (3)	accion != “nombre” (4)
		accion null (5)
valor	nuevo nombre del producto (6)	valor null (7)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{producto[],"nombre", "Cerveza Antares" }Escenario 1	Stock actualizado	1-3- 6
Incorrecta	{producto[],"nombre", "Cerveza Antares" }Escenario 2	Error	1-3- 6
	{producto[],"ASD", "Cerveza Antares" }Escenario 1	Error	1-4- 6
	{null,"nombre", "Cerveza Antares" }Escenario 1	Error	2-3 -6
	{producto[],null, "Cerveza Antares" }Escenario 1	Error	1-5- 6
	{producto[],"nombre", null }Escenario 1	Error	1-3- 7

`public void modificaProducto(Producto producto, String accion, float valor)`

modificaProducto

```
public void modificaProducto(Producto producto,
                             java.lang.String accion,
                             float valor)
```

Se modifican los precios del producto pasado por parametro. La accion deberÃa ser precioCosto o precioVenta. valor deberia ser mayor a 0.

Parameters:

`producto` - : Parametro de tipo Producto a modificar.

`accion` - : Parametro de tipo string que indica atributo a modificar.

`valor` - : Parametro de tipo float que indica el valor del atributo a modificar.

Escenarios

Nro Escenario	Descripción
1	Colección de productos con al menos un producto en la colección
2	Colección de productos vacía

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
producto	producto valido(1)	producto null(2)
accion	accion = “precioCosto” (3)	accion != “precioCosto” && accion!= precioVenta (4)
	accion = “precioVenta” (5)	accion null (6)
valor	valor>0(7)	valor <=0 (8)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{producto[],"precioCosto",50 }Escenario 1	Costo actualizado	1-3- 7
	{producto[],"precioVenta",50 }Escenario 1	Venta actualizada	1-5- 7
Incorrecta	{producto[],"ASD",50 }Escenario 1	Error	1-4- 7
	{null,"precioCosto",50 }Escenario 1	Error	2-3- 7
	{producto[],"precioCosto",0 }Escenario 1	Error	1- 3- 8
	{producto[],null,50 }Escenario 1	Error	1- 6- 7

CLASE GestionDePersonal

`public void altaOperario(java.lang.String nombreApellido, String nacimiento, String nombreUsuario, String password)`

altaOperario

```
public void altaOperario(java.lang.String nombreApellido,
                         java.lang.String nacimiento,
                         java.lang.String nombreUsuario,
                         java.lang.String password)
```

Da de alta un operario

Parameters:

`nombreApellido` - : Parametro de tipo String que representa el nombre y apellido del operario

`nacimiento` - : Parametro de tipo String que representa la fecha de nacimiento del operario

`nombreUsuario` - : Parametro de tipo String que representa el nombre de usuario de la cuenta del operario

`password` - : Parametro de tipo String que representa el passworde de la cuenta del operario

Escenarios

Nro Escenario	Descripción
1	Colección de operarios puede contener algún operario o no

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
nombreApellido	nombreApellido valido(1)	nombreApellido null(2)
nacimiento	nacimiento valido (3)	nacimiento null (4)
nombreUsuario	nombreUsuario valido (5)	NombreUsuario null (6)
password	password valido (7)	password invalido (8)
		password null (9)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{"Juan","19/12/1995","Juan8","Promocion2022"}	Operario creado	1-3-5-7
Incorrecta	{null,"19/12/1995","Juan8","Promocion2022"}	Error	2-3-5-7
	{"Juan",null","Juan8","Promocion2022"}	Error	1-4-5-7
	{"Juan","19/12/1995","null","Promocion2022"}	Error	1-3-6-7
	{"Juan","19/12/1995","Juan8",null}	Error	1-3-5-9
	{"Juan","19/12/1995","Juan8","a"}	Error	1-3-5-8

```
public void bajaOperario(Operario operario)
```

bajaOperario

```
public void bajaOperario(Operario operario)
```

Da de baja un operario

Parameters:

operario - : Parametro de tipo operario que representa el operario a eliminar

Escenarios

Nro Escenario	Descripcion
1	Colección de operarios con al menos un operario en la colección
2	Colección de operarios vacía

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
operario	operario valido(1)	operario null(2)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{operario[]} Escenario 1	Operario eliminado	1
Incorrecta	{operario[]} Escenario 2	Error	1
	{null} Escenario 1	Error	2

```
public void modificaOperario(Operario operario, String
accion, String valor)
```

modificaOperario

```
public void modificaOperario(Operario operario,
                             java.lang.String accion,
                             java.lang.String valor)
```

Modifica un parametro de un operario , recibe un String {nombreApellido,nombreUsuario,password,nacimiento} y en base a ello modifica el atributo pre: accion!=null

Parameters:

`operario` - : Parametro de tipo operario que representa el operario a modificar

`accion` - : Parametro de tipo String que representa el parametro a modificar

`valor` - : Parametro de tipo String que representa el valor del parametro a modificar

Escenarios

Nro Escenario	Descripción
1	Colección de operarios con al menos un operario en la colección
2	Colección de operarios vacía

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
operario	operario valido(1)	operario null(2)
valor	valor valido (3)	valor null(4)

Batería de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{operario,"nombreApellido","Gabriel"}Escenario1	Operario modificado	1-3
	{operario,"nacimiento","19/12/2021"}Escenario1	Operario modificado	1-3
	{operario,"nombreUsuario","Juan10"}Escenario1	Operario modificado	1-3
	{operario,"password","Gabriel2022"}Escenario1	Operario modificado	1-3
Incorrecta	{operario,"password","Gabriel"}Escenario1	Error	1-3
	{null,"nombreApellido","Gabriel"}Escenario1	Error	2-3
	{operario,"nombreApellido",null}Escenario1	Error	1-4
	{operario,"nombreApellido","Gabriel"}Escenario2	Error	1-3

`public void altaMozo(String nombreApellido,String nacimiento, int cantHijos)`

altaMozo

```
public void altaMozo(java.lang.String nombreApellido,
                     java.lang.String nacimiento,
                     int cantHijos)
```

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

Parameters:

`nombreApellido` - : Parametro que representa el nombre y apellido del mozo

`nacimiento` - : Parametro que representa la fecha de nacimiento del mozo

`cantHijos` - : Parametro que representa la cantidad de hijos del mozo

Escenarios

Nro Escenario	Descripcion
1	Colección de mozos puede contener algún mozo o no

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
nombreApellido	nombreApellido valido(1)	nombreApellido null(2)
nacimiento	nacimiento valido de persona mayor de 18 años (3)	nacimiento de una persona menor de 18 años(4)
		nacimiento null(5)
cantHijos	cantHijos>=0(6)	cantHijos<0(7)

Batería de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{"Lionel Messi","24/06/1987",3}	Vamos Mesi!!!!!!!	1-3- 6
Incorrecta	{null,"24/06/1987",3}	Error	2-3- 6
	{"Lionel Messi",null,3}	Error	1-5- 6
	{"Lionel Messi","24/06/2022",3}	Error	1-4- 6
	{"Lionel Messi","24/06/1987",-3}	Error	1-3- 7

```
public void bajaMozo(Mozo mozo)
```

bajaMozo

```
public void bajaMozo(Mozo mozo)
```

Da de baja un mozo

Parameters:

mozo - : Mozo a eliminar

Escenarios

Nro Escenario	Descripción
1	Colección de mozos con al menos un mozo en la colección
2	Colección de mozos vacía

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
mozo	mozo valido(1)	mozo null(2)

Batería de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{mozo[]}Escenario1	Mozo eliminado	1
Incorrecta	{mozo[]}Escenario2	Error	1
	{null}Escenario 1	Error	2

`public void modificaMozo(Mozo mozo, String accion, String valor)`

modificaMozo

```
public void modificaMozo(Mozo mozo,
                         java.lang.String accion,
                         java.lang.String valor)
```

Modifica un parametro de un mozo a excepcion de la cantidad de hijos pre: accion no puede ser nula y debe ser nombreApellido, estado

Parameters:

`mozo` - : Mozo a modificar

`accion` - : String que representa el parametro a modificar

`valor` - : String que representa el valor del parametro a modificar

Escenarios

Nro Escenario	Descripción
1	Colección de mozos con mozo "Lionel Messi" en la colección
2	Colección de mozos vacía

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
mozo	mozo valido(1)	mozo null(2)
valor	valor {activo,de franco, ausente}(3)	valor != {activo,de franco, ausente, nuevo NyA}(4)
	valor con nuevo nombre y apellido (5)	valor null(6)

Batería de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{mozo[],"NombreApellido","Lionel Andres Messi"}Escenario1	Mozo modificado	1- 5
	{mozo[],"estado","activo"}Escenario 1	Mozo modificado	1- 3
Incorrecta	{mozo[],"estado","a"}Escenario1	Error	1- 4
	{null,"NombreApellido","Lionel Andres Messi"}Escenario1	Error	2- 5
	{mozo[],"estado",null}Escenario1	Error	1- 5
	{mozo[],"NombreApellido","Lionel Andres Messi"}Escenario2		

CLASE Local

```
public Comanda getComandaByMesa(Mesa mesa)
```

getComandaByMesa

```
public Comanda getComandaByMesa(Mesa mesa)
```

Obtiene una comanda de la lista de comandas a partir de la mesa. **Pre:** mesa != null

Parameters:

mesa - no puede ser null

Returns:

comanda asociada a la mesa

Escenarios

Nro Escenario	Descripcion
1	Colección comandas con por lo menos una comanda.
2	Colección de comandas vacía.

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
mesa	mesa valido(1)	mesa null(2)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{mesa[]}Escenario 1	Devuelve mesa	1
Incorrecta	{mesa3[]}Escenario 1	Error	1
Incorrecta	{mesa[]}Escenario 2	Error	1

```
public Mozo getMozoByMesa(Mesa mesa)
```

getMozoByMesa

```
public Mozo getMozoByMesa(Mesa mesa)
```

Obtiene un mozo de la lista de asignaciones diaria a partir de la mesa asignada. **Pre:** mesa != null

Parameters:

`mesa` - no puede ser null

Returns:

mozo que tiene asignada la mesa

Escenarios

Nro Escenario	Descripcion
1	Colección mozos con por lo menos una comanda.
2	Colección de mozos vacía.

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
mesa	mesa valido(1)	

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{mesa[]}Escenario 1	Devuelve mozo	1
Incorrecta	{mesaX[]}Escenario 1	Error	1
Incorrecta	{mesa[]}Escenario 2	Error	1

`public void login(java.lang.String nombreUsuario,String password)`

login

```
public void login(java.lang.String nombreUsuario,
                  java.lang.String password)
```

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.

Parameters:

`nombreUsuario` - : Parametro de tipo String que representa el nombre de usuario.

`password` - : Parametro de tipo String que representa la contrasenia del usuario.

Escenarios

Nro Escenario	Descripcion
1	Existe un administrador registrado. Colección operarios con por lo menos un operario.
2	Existe un administrador registrado. Colección de operarios vacía.

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
nombreUsuario	nombreUsuario != null(1)	nombreUsuario null(2)
password	password != null(3)	password null(4)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{"ADMIN", "ADMIN1234"}Escenario 1	Se registra el administrador	1-3
Correcta	{"Meolans", "QWERTY1234"}Escenario 1	Ingresa el operario	1-3
Incorrecta	{null, "QWERTY1234"}Escenario 1	Error	1-2
Incorrecta	{"Meolans", "QWERTY1234"}Escenario 2	Error	1-3
Incorrecta	{"Meolans", null}Escenario 1	Error	1-4

CLASE MetodosFacturacion

`public void altaPromocionProducto(Producto producto,String diaProm,boolean dosXuno,boolean descuentoCantMin,int cantidadMinima,float descCantMin,boolean activa)`

altaPromocionProducto

```
public void altaPromocionProducto(Producto producto,
                                  java.lang.String diaProm,
                                  boolean dosXuno,
                                  boolean descuentoCantMin,
                                  int cantidadMinima,
                                  float descCantMin,
                                  boolean activa)
```

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.

Parameters:

`producto` - : Parametro de tipo Producto que representa producto del cual se hara promocion.

`diaProm` - : Parametro de tipo String que indica el dia de la semana que es valida la promocion.

`dosXuno` - : Parametro de tipo boolean que indica si la promocion incluye 2x1.

`descuentoCantMin` - : Parametro de tipo boolean que indica si la promocion incluye descuento de acuerdo a una cantidad minima.

`cantidadMinima` - : Parametro de tipo entero que indica la cantidad minima a partir de la cual se aplica la promocion.

`descCantMin` - : Parametro de tipo float que indica el porcentaje de descuento por cantidad minima.

`activa` - : Parametro de tipo boolean que indica si la promocion se encuentra activa o no.

Escenarios

Nro Escenario	Descripcion
1	Colección de productos con por lo menos un producto. Colección de promociones de producto con por lo menos una promocion de producto.
2	Colección de productos vacía. Colección de promociones de producto vacía.

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
producto	producto valido(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 != Dias Semana (4), 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)	

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{producto[], "Lunes", true, true, 6, 25, true}Escenario 1	Alta promocion	1-3-6-8-10-12-14
Correcta	{producto[], "Lunes", true, false, 6, 25, true}Escenario 1	Alta promocion	1-3-6-9-10-12-14
Correcta	{producto[], "Lunes", false, true, 6, 25, true}Escenario 1	Alta promocion	1-3-7-8-10-12-14
Correcta	{producto[], "Lunes", true, true, 6, 25, false}Escenario 1	Alta promocion	1-3-6-8-10-12-15
Incorrecta	{null, "Lunes", true, true, 6, 25, true}Escenario 2	Error	2-3-6-8-10-12-14
Incorrecta	{producto[], != Dias Semana , true, true, 6, 25, true}Escenario 1	Error	1-4-6-8-10-12-14
Incorrecta	{producto[], null , true, true, 6, 25, true}Escenario 1	Error	1-5-6-8-10-12-14
Incorrecta	{producto[], "Lunes" , true, true, -999, -25, true}Escenario 1	Error	1-3-6-8-11-13-14

Public void bajaPromocionProducto(PromocionProducto prom)

bajaPromocionProducto

`public void bajaPromocionProducto(PromocionProducto prom)`

Se elimina la promocion de la lista de promociones de producto. Si promocion no esta en la lista, no hace nada.

Parameters:

`prom` - : Parametro de tipo PromocionProducto a eliminar de la lista.

Escenarios

Nro Escenario	Descripcion
1	Colección de promociones de producto con por lo menos una promocion de producto.
2	Colección de promociones de producto vacía.

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
prom	producto valido(1)	producto null(2)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{promocion[]}Escenario 1	Baja promocion	1
Incorrecta	{null}Escenario 2	No se da de baja	2

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

modificacionPromocionProducto

```
public void modificacionPromocionProducto(PromocionProducto prom,
                                          java.lang.String diaProm,
                                          boolean dosXuno,
                                          boolean descuentoCantMin,
                                          float porcentajeCantMin,
                                          int cantidadMinima,
                                          boolean activa)
```

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.

Parameters:

`prom` - : Parametro de tipo Promocion que representa la promocion del cual se modificaran los atributos.

`diaProm` - : Parametro de tipo String que indica el dia de la semana que es valida la promocion.

`dosXuno` - : Parametro de tipo boolean que indica si la promocion incluye 2x1.

`descuentoCantMin` - : Parametro de tipo boolean que indica si la promocion incluye descuento de acuerdo a una cantidad minima.

`porcentajeCantMin` - : Parametro de tipo Float que indica el porcentaje de descuento que se aplica a promociones tipo descuentoCantMin

`cantidadMinima` - : Parametro de tipo entero que indica la cantidad minima a partir de la cual se aplica la promocion.

`activa` - : Parametro de tipo boolean que indica si la promocion se encuentra activa o no.

Escenarios

Nro Escenario	Descripcion
1	Colección de productos con por lo menos un producto.
2	Colección de productos vacía.

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
prom	promocion valido(1)	promocion 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 != Dias Semana (4), 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)	

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{promocion[], "Lunes", true, true, 6, 25, true}Escenario 1	Alta promocion	1-3-6-9-11-13-15
Correcta	{promocion[], "Lunes", false, false, 6, 25, false}Escenario 1	Alta promocion	1-3-7-10-11-13-16
Incorrecta	{null], "Lunes", false, true, 6, 25, true}Escenario 1	Error	2-3-7-8-10-12-14
Incorrecta	{promocion[], != Dias semana, false, false, 6, 25, false}Escenario 1	Error	1-4-7-10-11-13-16
Incorrecta	{promocion[], "Lunes", null, false, 6, 25, false}Escenario 1	Error	1-3-8-10-11-13-16
Incorrecta	{promocion[], "Lunes", true, true, -999, 25, true}Escenario 1	Error	1-3-6-9-12-13-15

`public void altaPromocionTemporal(java.lang.String nombre,String formaPago,int porcentajeDesc,String diasDePromo,boolean activa,boolean acumulable)`

altaPromocionTemporal

```
public void altaPromocionTemporal(java.lang.String nombre,
                                  java.lang.String formaPago,
                                  int porcentajeDesc,
                                  java.lang.String diasDePromo,
                                  boolean activa,
                                  boolean acumulable)
```

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

Parameters:

`nombre` - : parametro de tipo String que indica el nombre de la nueva promocion temporal.

`formaPago` - : parametro de tipo String que indica forma de pago.

`porcentajeDesc` - : parametro de tipo entero que indica porcentaje de descuento.

`diasDePromo` - : parametro de tipo String que indica dias de promocion.

`activa` - : parametro de tipo Boolean que indica si promocion esta activa o no.

`acumulable` - : parametro de tipo boolean que indica si es acumulable con otras promociones.

Escenarios

Nro Escenario	Descripcion
1	Colección de promociones temporales con por lo menos una promocion temporal.
2	Colección de promociones temporales vacía.

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
nombre	nombre valido(1)	nombre null(2)
formaPago	formaPago = “efectivo” (3.1) formaPago = “tarjet” (3.2) formaPago = “mercPago” (3.3) formaPago = “ctaDNI” (3.4)	formaPago != pagos existentes (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 != Dias Semana (9), diasDePromo = null (10)
activa	activa = true (11) activa = false (12)	
acumulable	acumulable = true (13)	

	acumulable = false (14)	

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{"Promo Dia", "efectivo", 50, "Lunes", true, true}Escenario 1	Alta promocion temporal	1-3-6-8-11-13
Correcta	{"Promo Dia", "efectivo", 50, "Lunes", false, false}Escenario 1	Alta promocion temporal	1-3-6-8-12-14
Incorrecta	{null, "efectivo", 50, "Lunes", true, true}Escenario 1	Error	2-3-6-8-11-13
Incorrecta	{"Promo Dia", "Bitcoin", 50, "Lunes", true, true}Escenario 1	Error	1-4-6-8-11-13
Incorrecta	{"Promo Dia", null, 50, "Lunes", true, true}Escenario 2	Error	1-5-6-8-11-13
Incorrecta	{"Promo Dia", "efectivo", -999, "Lunes", true, true}Escenario 1	Error	1-3-7-9-11-13
Incorrecta	{"Promo Dia", "efectivo", 50, "LuneZ", true, true}Escenario 1	Error	1-3-6-9-11-13
Incorrecta	{"Promo Dia", "efectivo", 50, null, true, true}Escenario 1	Error	1-3-6-10-11-13

```
public void bajaPromocionTemporal(PromocionTemporal prom)
```

bajaPromocionTemporal

```
public void bajaPromocionTemporal(PromocionTemporal prom)
```

Se elimina una promocion temporal de la lista de promociones temporales.

Parameters:

prom - : Parametro de tipo PromocionTemporal a eliminar.

Escenarios

Nro Escenario	Descripcion
1	Colección de promociones temporales con por lo menos una promocion temporal.
2	Colección de promociones temporales vacía.

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
prom	promocion valido(1)	promocion null(2)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{promocion[]}Escenario 1	Baja promocion temporal	1
Incorrecta	{null}Escenario 1	Error	2
Incorrecta	{promocionX[]}Escenario 2	Error	1

`public void modificacionPromocionTemporal(PromocionTemporal prom,String nombre,String formaPago,int porcentajeDesc,String diasDePromo,boolean activa,boolean acumulable)`

modificacionPromocionTemporal

```
public void modificacionPromocionTemporal(PromocionTemporal prom,
                                          java.lang.String nombre,
                                          java.lang.String formaPago,
                                          int porcentajeDesc,
                                          java.lang.String diasDePromo,
                                          boolean activa,
                                          boolean acumulable)
```

Se modifican todos los parametros de la promocion temporal pasada por parametro. porcentajeDesc deberia ser mayor a o diasDePromo deberia indicar los dias de la semana. activa deberia ser true o false. acumulable deberia ser true o false.

Parameters:

`nombre` - : parametro de tipo String que indica el nombre de la nueva promocion temporal.

`formaPago` - : parametro de tipo String que indica forma de pago.

`porcentajeDesc` - : parametro de tipo entero que indica porcentaje de descuento.

`diasDePromo` - : parametro de tipo String que indica dias de promocion.

`activa` - : parametro de tipo Boolean que indica si promocion esta activa o no.

`acumulable` - : parametro de tipo boolean que indica si es acumulable con otras promociones.

Escenarios

Nro Escenario	Descripción
1	Colección de promociones temporales con por lo menos una promoción temporal.
2	Colección de promociones temporales vacía.

Tabla de Particiones

Condición de entrada	Clases Validas	Clases Invalidas
prom	promocion valido(1)	promocion null(2)
nombre	nombre != null(3)	nombre null(4)
formaPago	formaPago = “efectivo” (5.1) formaPago = “tarjet” (5.2) formaPago = “mercPago” (5.3) formaPago = “ctaDNI” (5.4)	formaPago != pagos existentes (6), 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 != Dias Semana (11), diasDePromo = null (12)
activa	activa = true (13) activa = false (14)	
acumulable	acumulable = true (15) acumulable = false (16)	

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{promocion[], "Promo Dia", "efectivo", 50, "Lunes", true, true}Escenario 1	Modifica promocion temporal	1-3-5-8-10-13-15
Correcta	{promocion[], "Promo Dia", "efectivo", 50, "Lunes", false, false}Escenario 1	Modifica promocion temporal	1-3-5-8-10-14-16
Incorrecta	{null, "Promo dia", "efectivo", 50, "Lunes", false, false}Escenario 1	Error	2-3-5-8-10-14-16
Incorrecta	{promocion[], null, "efectivo", 50, "Lunes", false, false}Escenario 1	Error	1-4-5-8-10-13-15
Incorrecta	{promocion[], "Promo Dia", "Bitcoin", 50, "Lunes", true, true}Escenario 1	Error	1-3-6-8-10-13-15
Incorrecta	{promocion[], "Promo Dia", null, 50, "Lunes", true, true}Escenario 1	Error	1-3-7-8-10-13-15
Incorrecta	{promocion[], "Promo Dia", "efectivo", 50, "Lunes", true, true}Escenario 1	Error	1-3-5-9-10-13-15
Incorrecta	{promocion[], "Promo Dia", "efectivo", 50, "LuneZ", true, true}Escenario 1	Error	1-3-5-8-11-13-15
Incorrecta	{promocion[], "Promo Dia", "efectivo", 50, null, true, true}Escenario 1	Error	1-3-5-8-12-13-15

```
public Factura generacionDeFactura(java.util.Calendar
fecha,diaSemana,Comanda comanda,String metodoDePago)
```

generacionDeFactura

```
public Factura generacionDeFactura(java.util.Calendar fecha,
                                   java.lang.String diaSemana,
                                   Comanda comanda,
                                   java.lang.String metodoDePago)
```

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. **Pre:** comanda != null

Parameters:

`fecha` - : Parametro de tipo Calendar que indica la fecha del sistema.

`diaSemana` - : Parametro de tipo String que indica el dia de la semana que se emitio la factura.

`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.

`metodoDePago` - : Parametro de tipo String que indica el metodo de pago de la factura.

Returns:

Factura generada

Escenarios

Nro Escenario	Descripción
1	Colección de productos con por lo menos un producto. Colección de promociones temporales con por lo menos una promoción temporal.
2	Colección de productos vacía. Colección de promociones temporales vacía.

Tabla de Particiones

Condición de entrada	Clases Válidas	Clases Inválidas
fecha	fecha valido(1)	fecha null(2)
diasSemana	diasSemana = “Lunes” (3.1) diasSemana = “Martes” (3.2) diasSemana = “Miercoles” (3.3) diasSemana = “Jueves” (3.4) diasSemana = “Viernes” (3.5) diasSemana = “Sabado” (3.6) diasSemana = “Domingo” (3.7)	diasDePromo != Dias Semana (4), diasDePromo = null (5)
comanda	comanda valido(6)	comanda null(7)
metodoDePago	metodoDePago = “efectivo” (8.1) metodoDePago = “tarjet” (8.2) metodoDePago = “mercPago” (8.3) metodoDePago = “ctaDNI” (8.4)	metodoDePago != pagos existentes (9), metodoDePago = null (10)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{fecha[], "Lunes", comanda[], "efectivo"}Escenario 1	Retorna factura esperada	1-3-6-8
Correcta	{fecha[], "Lunes", comandaX[], "efectivo"}Escenario 2	No retorna factura esperada	1-3-6-8
Incorrecta	{null, "Lunes", comanda[], "efectivo"}Escenario 1	Error	2-3-6-8
Incorrecta	{fecha[], "LuneZ", comanda[], "efectivo"}Escenario 1	Error	1-4-6-8
Incorrecta	{fecha[], null, comanda[], "efectivo"}Escenario 1	Error	1-5-6-8
Incorrecta	{fecha[], "Lunes", null, "efectivo"}Escenario 1	Error	1-3-7-8
Incorrecta	{fecha[], "Lunes", comanda[], "Bitcoin"}Escenario 1	Error	1-3-6-9

```
public Pedido altaPedido(java.lang.String hoy,int
cantidad,Producto producto)
```

altaPedido

```
public Pedido altaPedido(java.lang.String hoy,
                         int cantidad,
                         Producto producto)
```

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. **Pre:** producto != null
Pre: cantidad > 0

Parameters:

hoy - : Parametro de tipo String convertido de un GregorianCalendar.

cantidad - : Parametro de tipo int que representa la cantidad del producto pedido

producto - : Parametro de tipo Producto que representa el producto pedido

Returns:

el nuevo pedido o null en caso de que no haya stock suficiente.

Escenarios

Nro Escenario	Descripcion

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
hoy	hoy = dia actual(1)	hoy null(2)
cantidad	cantidad > 0(3)	cantidad <= 0(4)
producto	producto valido(5)	producto null(6)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{hoy[], 10, producto[]}	Alta pedido	1-3--5
Incorrecta	{null, 10, producto[]}	Error	2-3--5
Incorrecta	{hoy[], -999, producto[]}	Error	1-4--5
Correcta	{hoy[], 10, null}	Error	1-3--6

```
public void bajaPedido(Comanda comanda,Pedido pedido)
```

bajaPedido

```
public void bajaPedido(Comanda comanda,
                       Pedido pedido)
```

Se da de baja un pedido de la lista de pedidos de una comanda.

Parameters:

`comanda` - : Parametro de tipo comanda de la cual se da de baja el pedido.

`pedido` - : Parametro de tipo pedido que indica pedido que se da de baja de la lista de una comanda.

Escenarios

Nro Escenario	Descripcion
1	Colección de pedidos de comanda con por lo menos un pedido.
2	Colección de pedidos de comanda vacía.

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
comanda	comanda valido(1)	comanda null(2)
pedido	pedido valido(3)	pedido null(4)

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{comanda[], pedido[]}Escenario 1	Baja pedido	1-3
Incorrecta	{null, pedidod[]}Escenario 1	Error	2-3
Incorrecta	{comanda[], null}Escenario 2	Error	1-4

```
public Comanda altaComanda(Mesa mesa,Pedido pedido)
```

altaComanda

```
public Comanda altaComanda(Mesa mesa,
                           Pedido pedido)
```

Se crea y se da de alta una nueva comanda como activa en la lista de comandas del local. **Pre:** mesa != null
Pre: pedido != null

Parameters:

`mesa` - : Parametro de tipo Mesa a la cual se le asignara la comanda.

`pedido` - : Parametro de tipo Pedido que indica pedido inicial de la comanda.

Returns:

nueva instancia de Comanda

Escenarios

Nro Escenario	Descripcion

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
mesa	mesa valido(1)	
pedido	pedido valido(2)	

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{mesa[], pedido[]}	Baja pedido	1-2

`public void modificacionComanda(Comanda comanda,Pedido pedido,boolean agregar)`

modificacionComanda

```
public void modificacionComanda(Comanda comanda,
                                Pedido pedido,
                                boolean agregar)
```

Se agrega o se remueve un pedido de la lista de pedidos de la comanda. **Pre:** comanda != null
Pre: pedido != null

Parameters:

`comanda` - : Parametro de tipo Comanda que representa la comanda de la cual se actualizara la lista de pedidos.

`pedido` - : Parametro de tipo Pedido que representa el pedido a dar de alta o baja de la lista de pedidos de la comanda.

`agregar` - : Parametro de tipo boolean que representa accion a realizar.

Escenarios

Nro Escenario	Descripcion
1	Colección comandas con por lo menos una comanda.
2	Colección de comandas vacía.

Tabla de Particiones

Condicion de entrada	Clases Validas	Clases Invalidas
comanda	comanda valido(1)	
pedido	pedido valido(2)	
agregar	agregar = true (3) agregar = false (4)	

Bateria de pruebas

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{comanda[], pedido2[], true}Escenario 1	Modifica pedido, agrega	1-2--3
Correcta	{comanda[], pedido[], false}Escenario 1	Modifica pedido, baja	1-2--4
Incorrecta	{comanda[], pedido2[], false}Escenario 1	Error	1-2--4
Incorrecta	{comanda[], pedido[], true}Escenario 1	Error	1-2--3

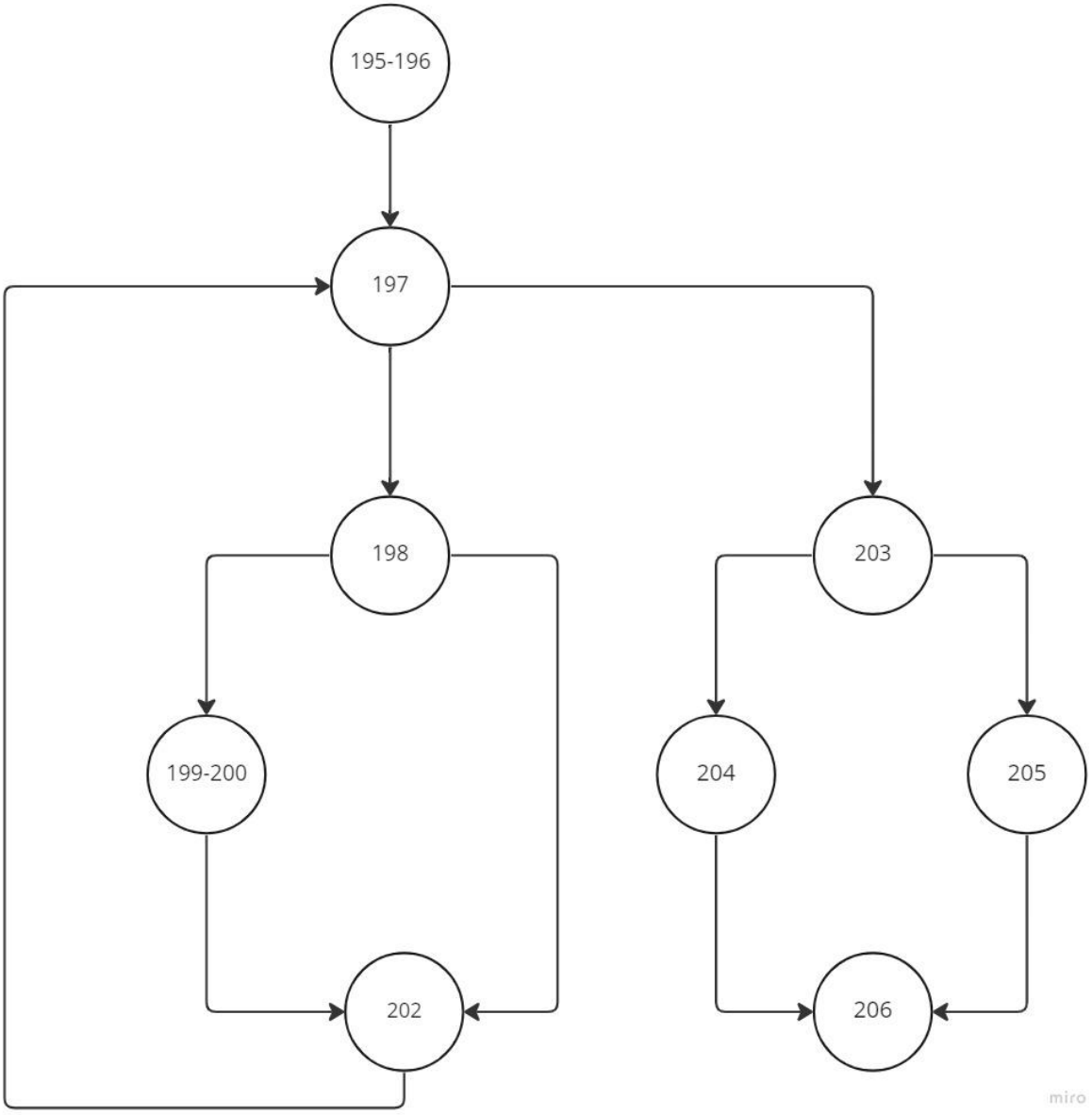
Caja Blanca

Las pruebas de caja blanca se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. En este trabajo evaluamos los distintos flujos de ejecución del programa y nos cercioramos de que se devuelven los valores de salida adecuados. Por lo tanto, a diferencia de la caja negra, en estas pruebas nos centramos en analizar el código fuente, intentando que todas las líneas de código se ejecuten.

En el caso particular de este trabajo,al realizar el test de cobertura se decidió analizar métodos correspondientes a la clase *local* relacionados a obtener datos estadísticos de los mozos, para esto se aplicaron pruebas de caja blanca sobre 4 métodos fundamentales para esto, que son : getMozoMaxVentas, getMozoMinVentas, getMozoMaxPromedio y getMozoMinPromedio.

En todos estos métodos se ha realizado el grafo ciclomatico, se ha calculado la complejidad ciclomática, determinamos caminos, casos y salidas esperadas. Posteriormente según la tabla que armamos codificamos los métodos correspondientes según nuestras salidas esperadas, para de esta manera obtener conclusiones después de aplicar dichas pruebas.

public Mozo getMozoMaxVentas()



Complejidad ciclomática = arcos - nodos + 2 =11-9+2= 4

Por lo tanto tendremos como máximo **4** caminos.

C1:(195-196)-197-203-204-206
C2:(195-196)-197-203-204-205-206
C3:(195-196)-197-198-202-203-205-206
C4:(195-196)-197-198-(199-200)-202-203-204-206

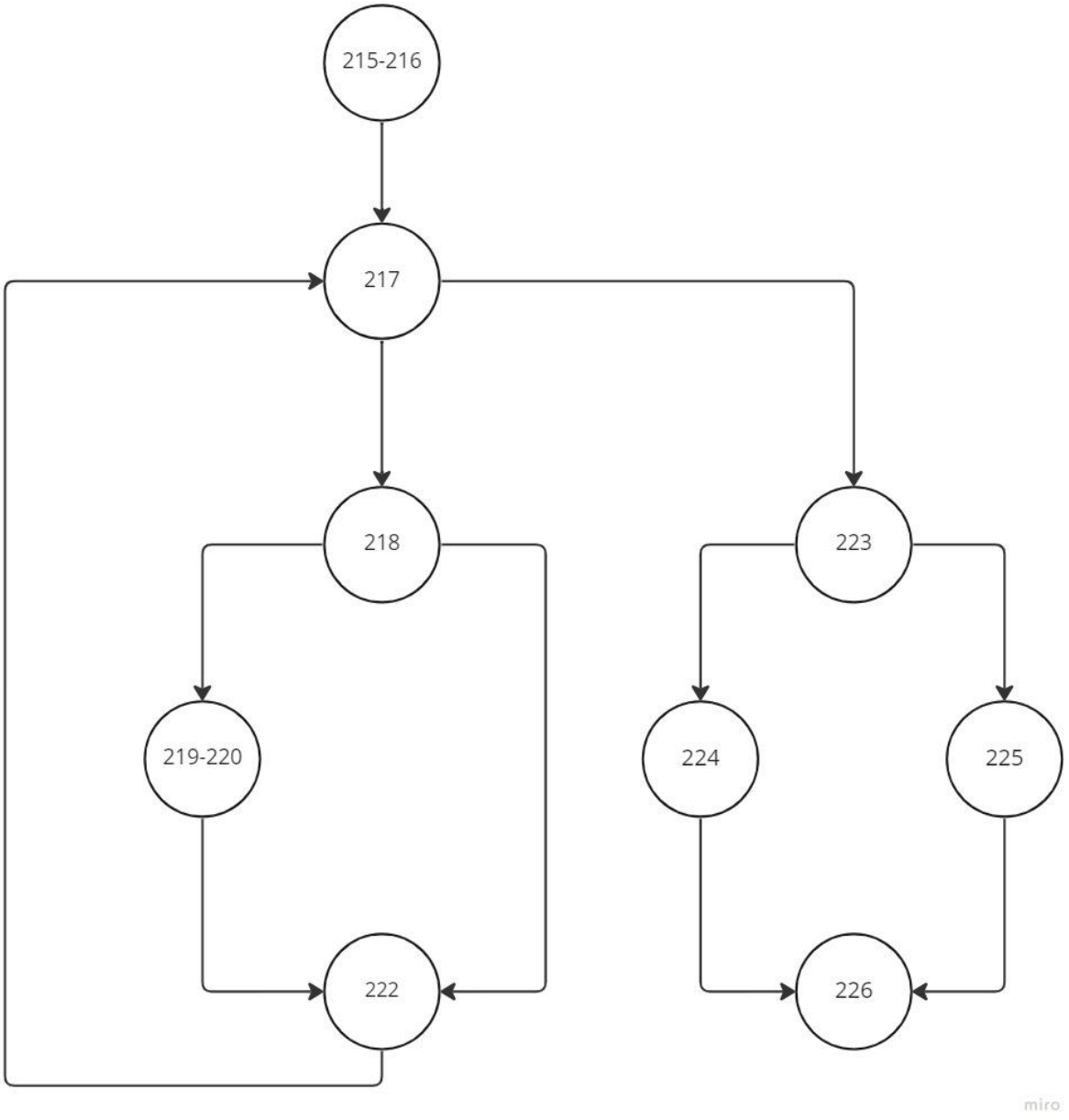
Escenario 1: Colección de mozos con al menos 1 mozo en la colección

Camino	Caso	Salida esperada
1	Camino imposible	Camino imposible
2	No hay mozos, imposible en este escenario	Imposible en este escenario
3	El mozo actual no es el que tiene el máximo acumulado de ventas	retorna mozo con máximo de ventas
4	El mozo actual es el que tiene el máximo acumulado de ventas	retorna mozo con máximo de ventas

Escenario 2: Colección de mozos vacía

Camino	Caso	Salida esperada
1	Camino imposible	Camino imposible
2	No hay mozos	retorna null
3	Error , mozo es null	Error , mozo es null
4	Error , mozo es null	Error , mozo es null

public Mozo getMozoMinVentas()



Complejidad ciclomática = arcos - nodos + 2 =11-9+2= 4

Por lo tanto tendremos como máximo **4** caminos.

C1:(215-216)-217-223-224-226
C2:(215-216)-217-223-224-225-226
C3:(215-216)-217-218-222-223-225-226
C4:(215-216)-217-218-(219-220)-222-223-224-226

Escenario 1: Colección de mozos con al menos 1 mozo en la colección

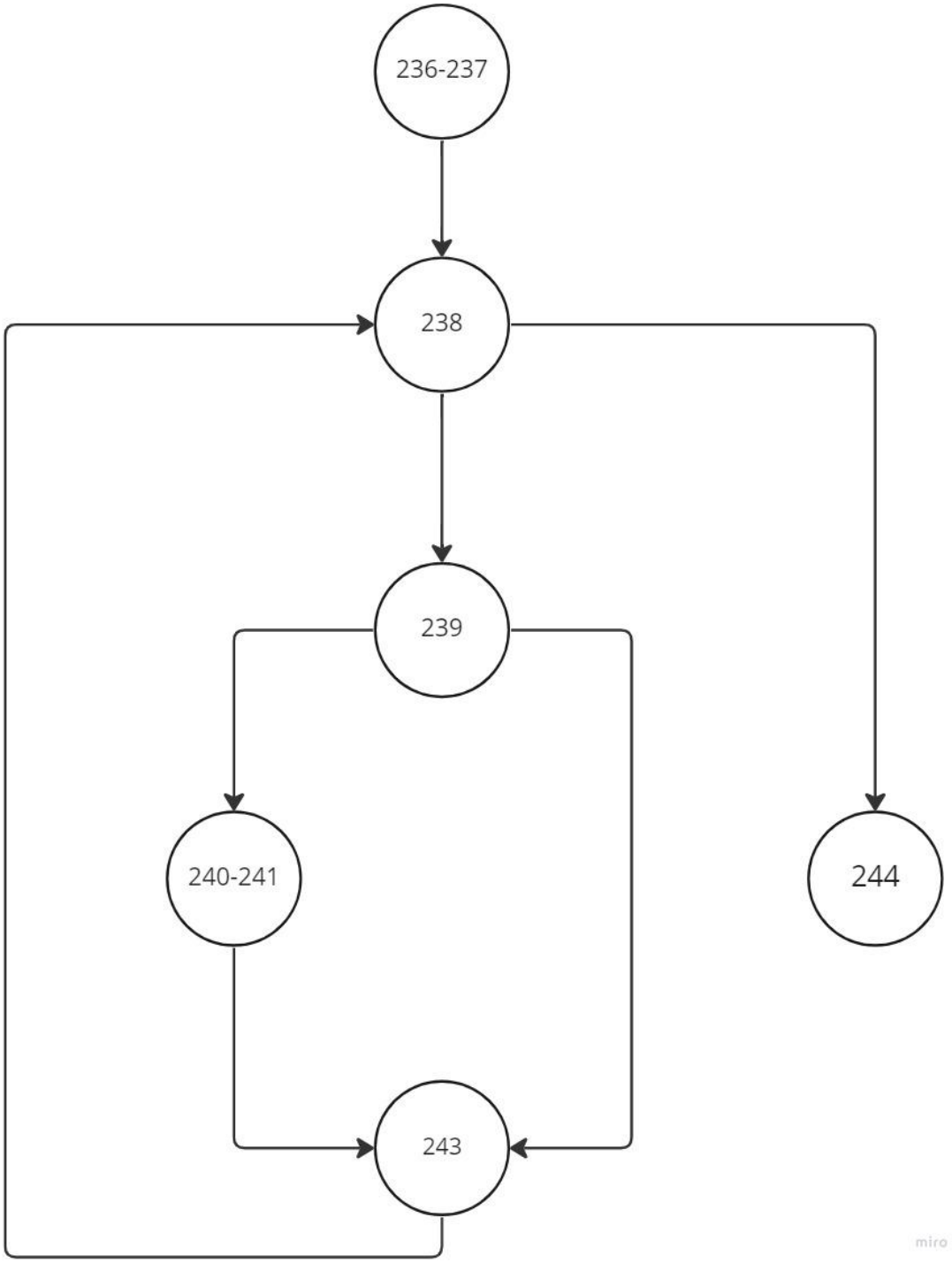
Camino	Caso	Salida esperada
1	Camino imposible	Camino imposible
2	No hay mozos, imposible en este escenario	Imposible en este escenario
3	El mozo actual no es el que tiene el máximo acumulado de ventas	retorna mozo con máximo de ventas
4	El mozo actual es el que tiene el máximo acumulado de ventas	retorna mozo con máximo de ventas

Escenario 2: Colección de mozos vacía

Camino	Caso	Salida esperada
1	Camino imposible	Camino imposible
2	No hay mozos	retorna null
3	Error , mozo es null	Error , mozo es null
4	Error , mozo es null	Error , mozo es null

Para estos dos métodos similares en su estructura podemos concluir que gracias al análisis de caja blanca que se realizó, no se contempla el caso de una colección de mozos vacía, donde entrando en las líneas 203 y 223 respectivamente, el código se rompería ya que no se considera la posibilidad de que mozo = null.
En cuanto a la colección completa, concluimos en que el método pasa satisfactoriamente la prueba de cobertura realizada en el código de caja blanca.

public Mozo getMozoMaxPromedio()



Complejidad ciclomática = arcos - nodos + 2 =7-6+2= 3

Por lo tanto tendremos como máximo **3** caminos.

C1:(236-237)-238-244

C2:(236-237)-238-239-243-244

C3:(236-237)-238-239-(240-241)-243-244

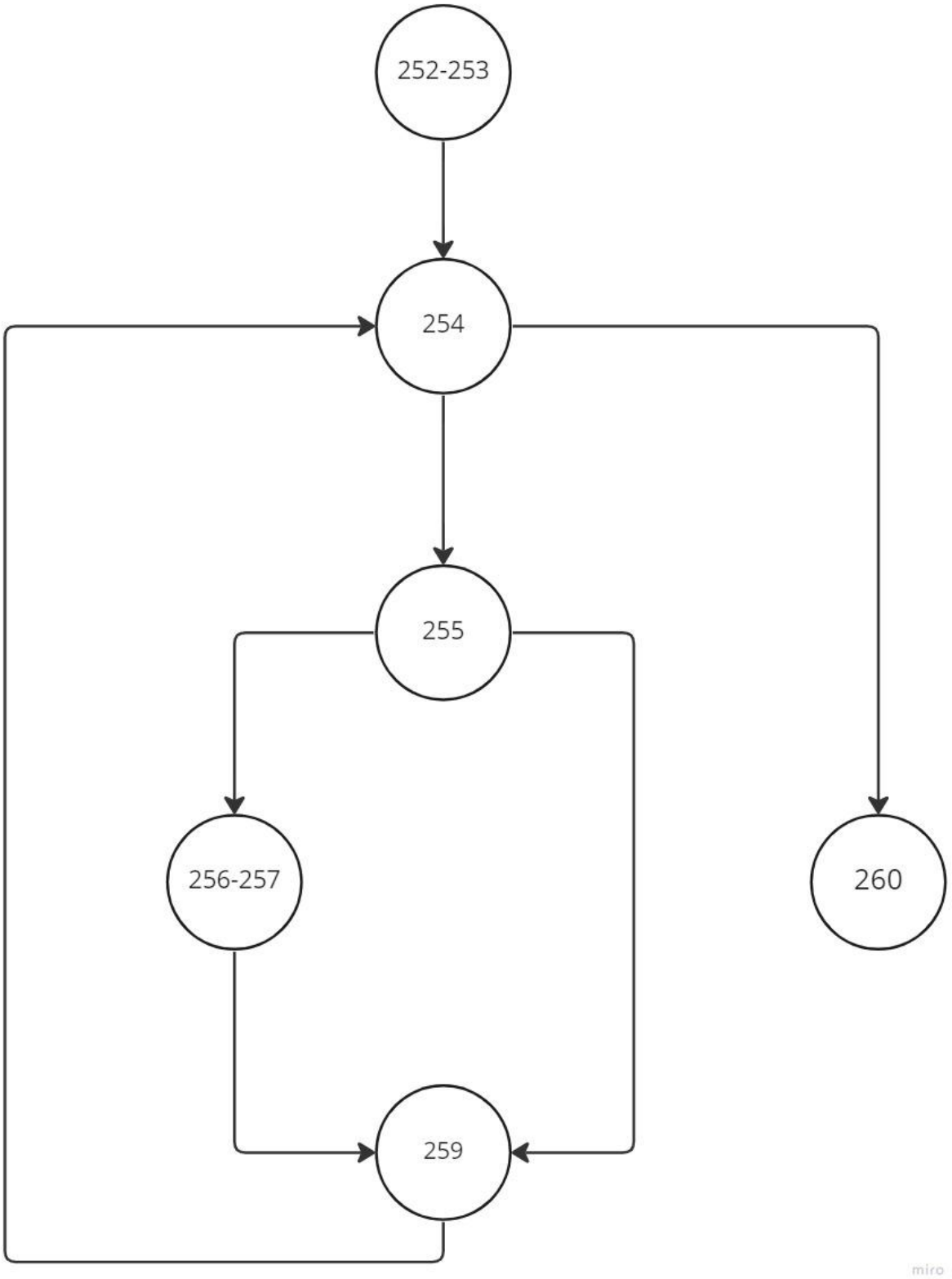
Escenario 1: Colección de mozos con al menos 1 mozo en la colección

Camino	Caso	Salida esperada
1	No hay mozos, imposible en este escenario	No hay mozos, imposible en este escenario
2	El mozo actual no es el que tiene el máximo promedio	retorna mozo con máximo promedio
3	El mozo actual es el que tiene el máximo promedio	retorna mozo con máximo promedio

Escenario 2: Colección de mozos vacía

Camino	Caso	Salida esperada
1	No entra al ciclo porque no hay mozos en la colección	retorna null
2	Camino imposible	Camino imposible
3	Camino imposible	Camino imposible

public Mozo getMozoMinPromedio()



Complejidad ciclomática = arcos - nodos + 2 =7-6+2= 3

Por lo tanto tendremos como máximo **3** caminos.

C1:(252-253)-254-260

C2:(252-253)-254-255-259-260

C3:(252-253)-254-255-(256-257)-259-260

Escenario 1: Colección de mozos con al menos 1 mozo en la colección

Camino	Caso	Salida esperada
1	No hay mozos, imposible en este escenario	No hay mozos, imposible en este escenario
2	El mozo actual no es el que tiene el mínimo promedio	retorna mozo con mínimo promedio
3	El mozo actual es el que tiene el mínimo promedio	retorna mozo con mínimo promedio

Escenario 2: Colección de mozos vacía

Camino	Caso	Salida esperada
1	No entra al ciclo porque no hay mozos en la colección	retorna null
2	Camino imposible	Camino imposible
3	Camino imposible	Camino imposible

A diferencia de los primeros dos métodos testeados, estos últimos no presentan errores en su ejecución y pasan satisfactoriamente la prueba de caja blanca.

Test de Persistencia

Para el test de persistencia se eligió el módulo *local*. El método de persistencia usado en *local* fue la persistencia xml.

Durante las pruebas se testean los siguientes escenarios:

- La creación correcta del archivo.
- La escritura en el archivo, tanto con el arreglo de mesas vacías como con

datos cargados.

- Despersistir con un archivo incorrecto que no existe.
- Despersistir con el archivo correcto.

En todos estos escenarios no se encontraron errores, pero sí hemos detectado algunas fallas dado que mediante estos tests comprobamos que la persistencia se crea y se guarda correctamente, sin embargo hay fallas a la hora de cargar datos. Por lo que se concluyó que la persistencia del módulo local no está bien implementada.

Test de GUI

Implementamos pruebas para la ventana de login, donde pueden iniciar sesion los operarios, siendo el primero de estos quien queda como Administrador logueandose por única vez con el usuario “ADMIN” contraseña “ADMIN1234” , también para esta prueba sometimos al programa al ingreso de un operario de nombre de usuario “Meolans” y de contraseña “QWERTY1234” , así como los logueos fallidos, los cuales serían usuario incorrecto, y contraseña incorrecta.

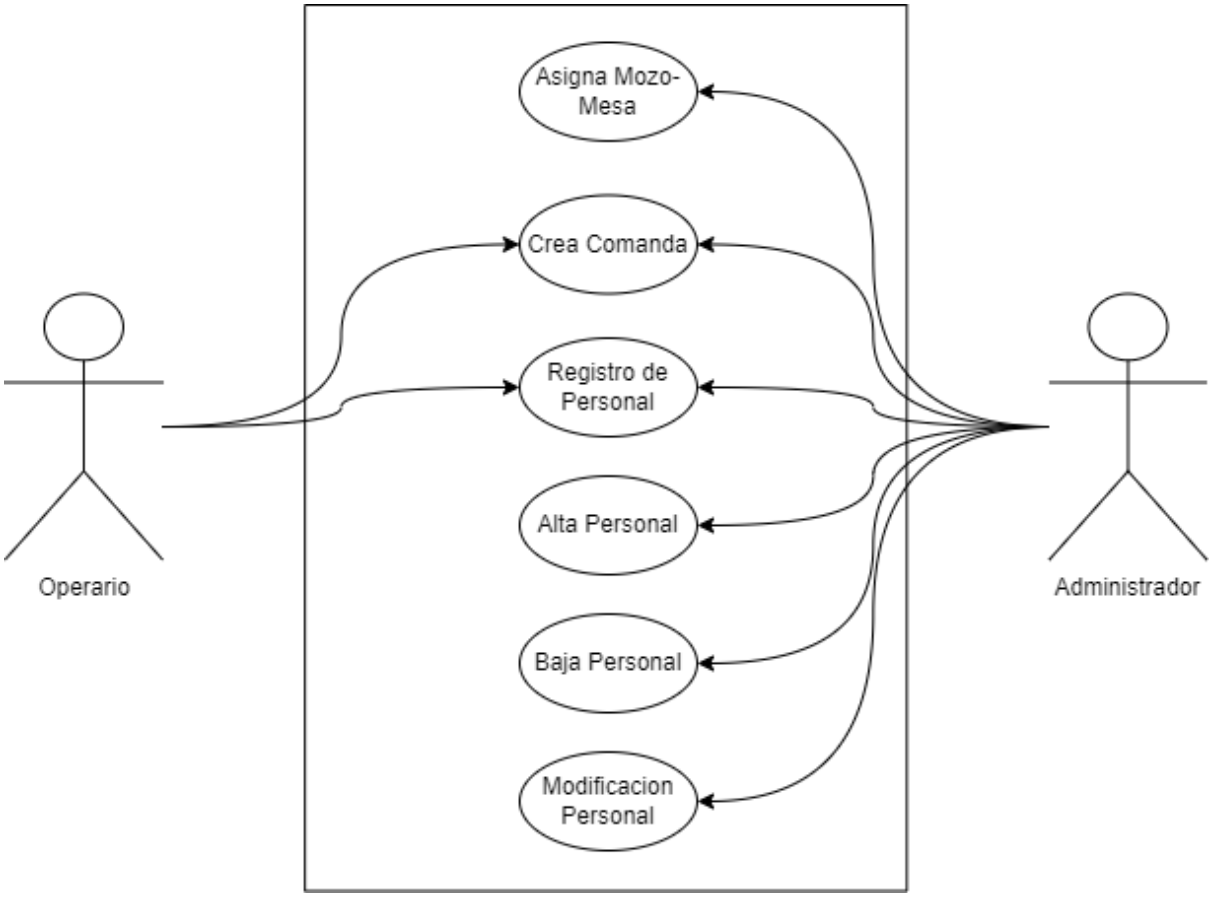
El testeo de interfaces gráficas lo hicimos utilizando JUnit 4 y la clase Robot perteneciente al paquete AWT de Java.

El método de testeo automatizado usado consiste en simular la interacción del usuario, utilizando la clase robot para clickear y completar los componentes de la ventana (cuadros de texto, botones, listas, etc).

Como objetivo a testear, nos enfocamos en verificar que los ingresos por teclado estén validados, y que todo funcione como es esperado.

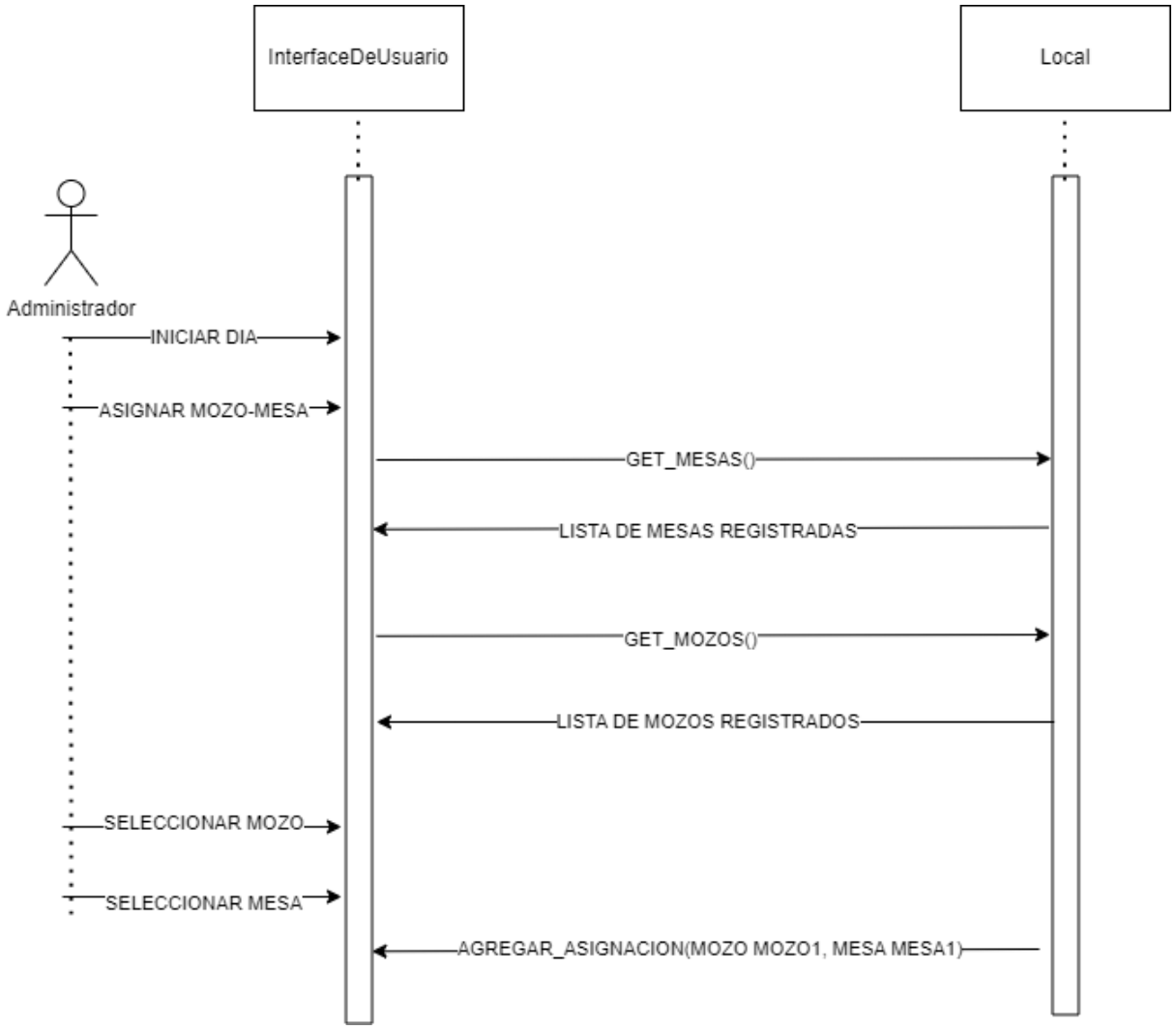
Test de Integración

Diagrama de casos de uso:



Diagramas de casos de secuencia:

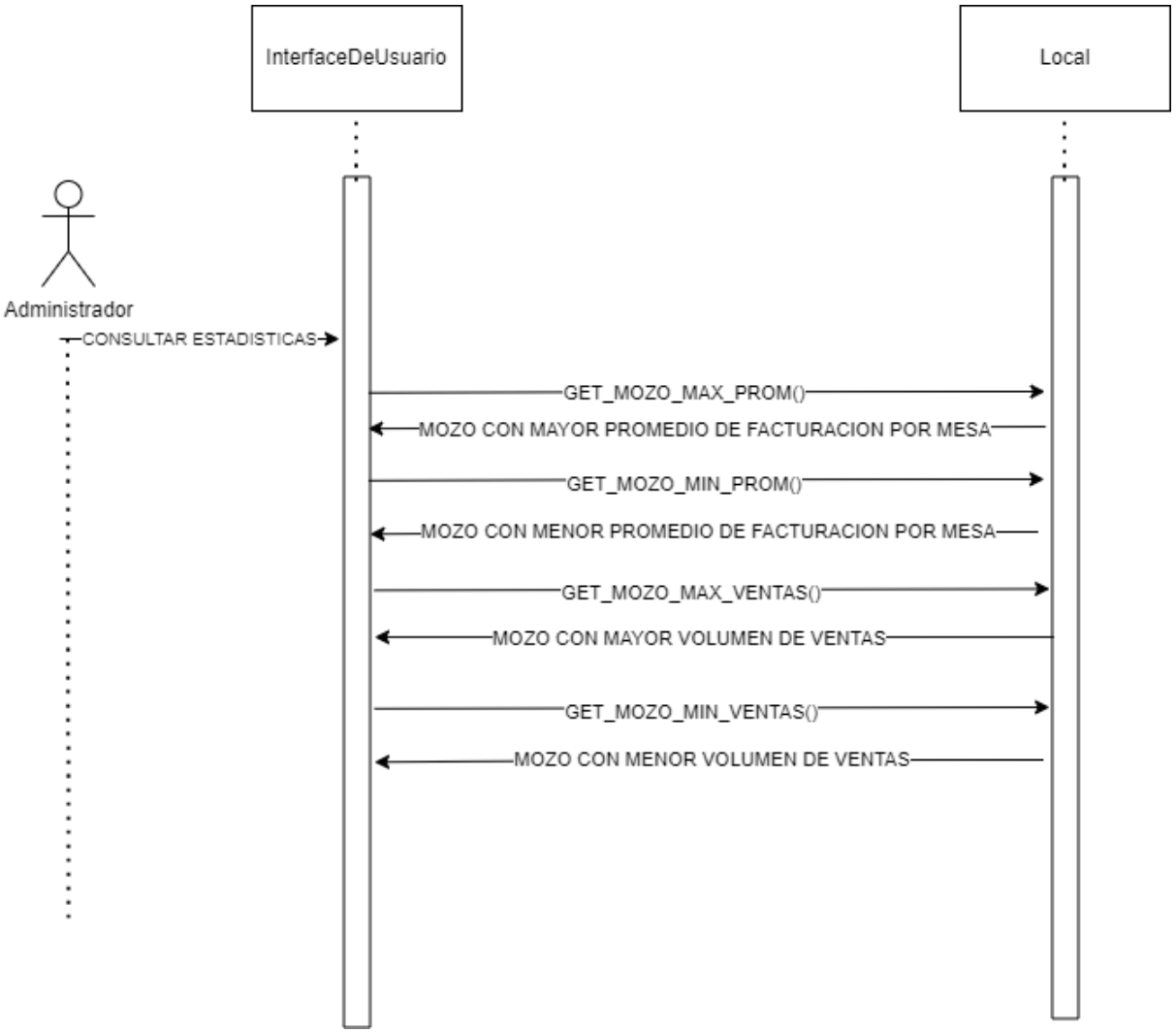
Asigna Mozo-Mesa:



Nro Escenario	Descripción
1	Colección de mesas con por lo menos una mesa. Colección de mozos con por lo menos algún mozo
2	Colección de mozos vacía. Colección de mesas vacía.

Condición de entrada	Clases Válidas	Clases Inválidas
mozo	mozo valido(1)	mozo invalido(2)
mesa	mesa valido(3)	mesa invalida(4)

Consulta estadísticas:

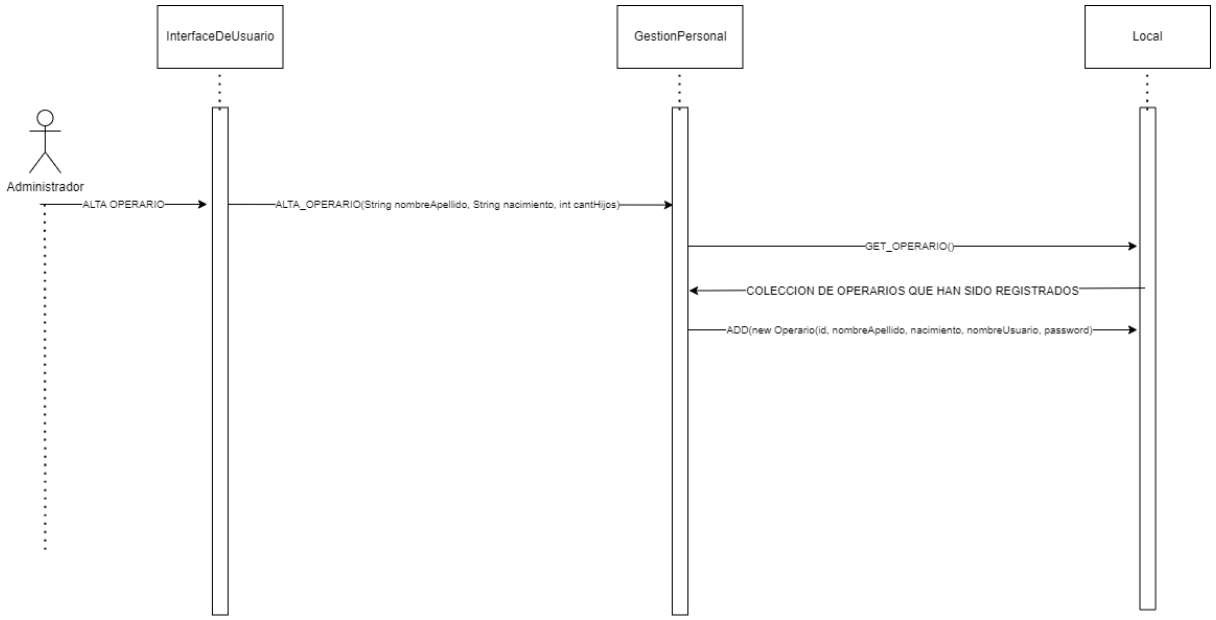


Nro Escenario	Descripcion
1	Colección de mozos con por lo menos algun mozo
2	Colección de mozos vacía

Condicion de entrada	Clases Validas	Clases Invalidas
mozo	mozo se encuentra en coleccion(1)	mozo no se encuentra en coleccion(2)

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{mozo1[]}Escenario 1	Se muestran las estadisticas	1
Incorrecta	{mozo1[]}Escenario 2	No se muestran las estadisticas	2

Alta Operario:

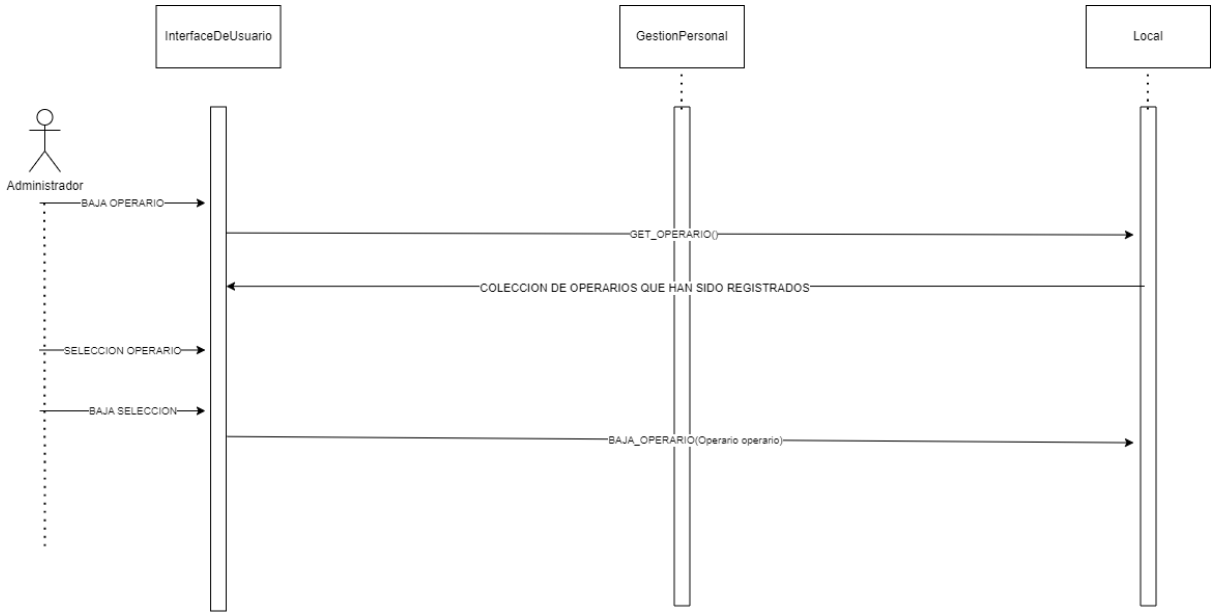


Nro Escenario	Descripcion
1	Colección de operarios con por lo menos algun operario
2	Colección de operarios vacía

Condicion de entrada	Clases Validas	Clases Invalidas
Estado de Parametros	Parametros validos(1)	Parametros validos(2)

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{"Jose", "22/06/1978", "Meolans", "QWERTY1234"}Escenario 1	Se da de alta operario	1
Incorrecta	{null, "22/06/1978", "Meolans", "QWERTY1234"}Escenario 2	Error	2

Baja Operario:



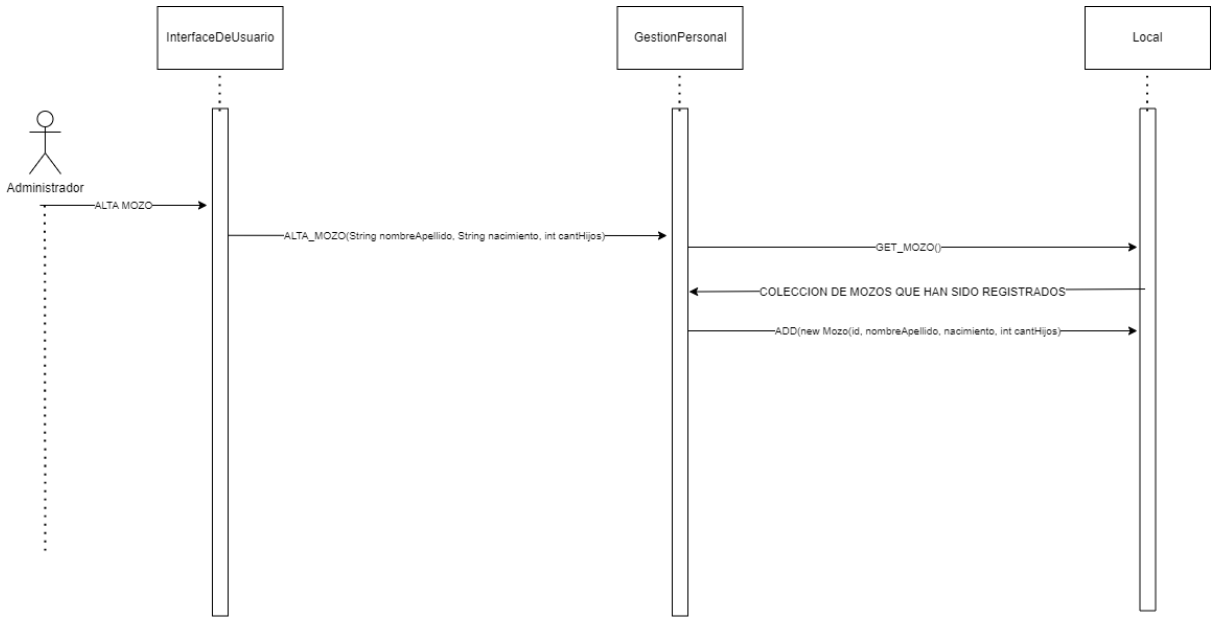
Nro Escenario	Descripcion
1	Colección de operarios con por lo menos algun operario
2	Colección de operarios vacía

Condicion de entrada	Clases Validas	Clases Invalidas
operario	operario valido(1)	operario invalida(2)

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

Correcta	{"Jose", "22/06/1978", "Meolans", "QWERTY1234"}Escenario 1	Se da de baja operario	1
Incorrecta	{null}Escenario 2	Error	2

Alta Mozo:

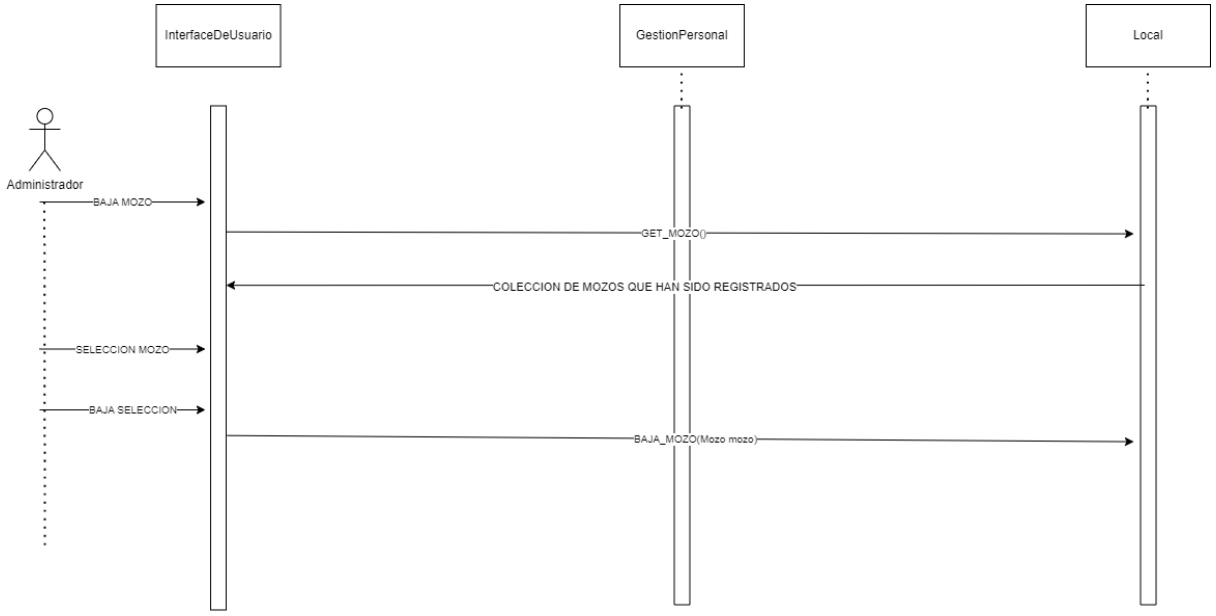


Nro Escenario	Descripcion
1	Colección de mozos con por lo menos algun mozo
2	Colección de mozos vacía

Condicion de entrada	Clases Validas	Clases Invalidas
Estado de Parametros	Parametros validos(1)	Parametros validos(2)

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada
Correcta	{"Walter White", "07/09/1958", 1}Escenario 1	Se da de alta al mozo
Incorrecta	{NULL, "07/09/1958", 1}Escenario 2	Error

Baja Mozo:

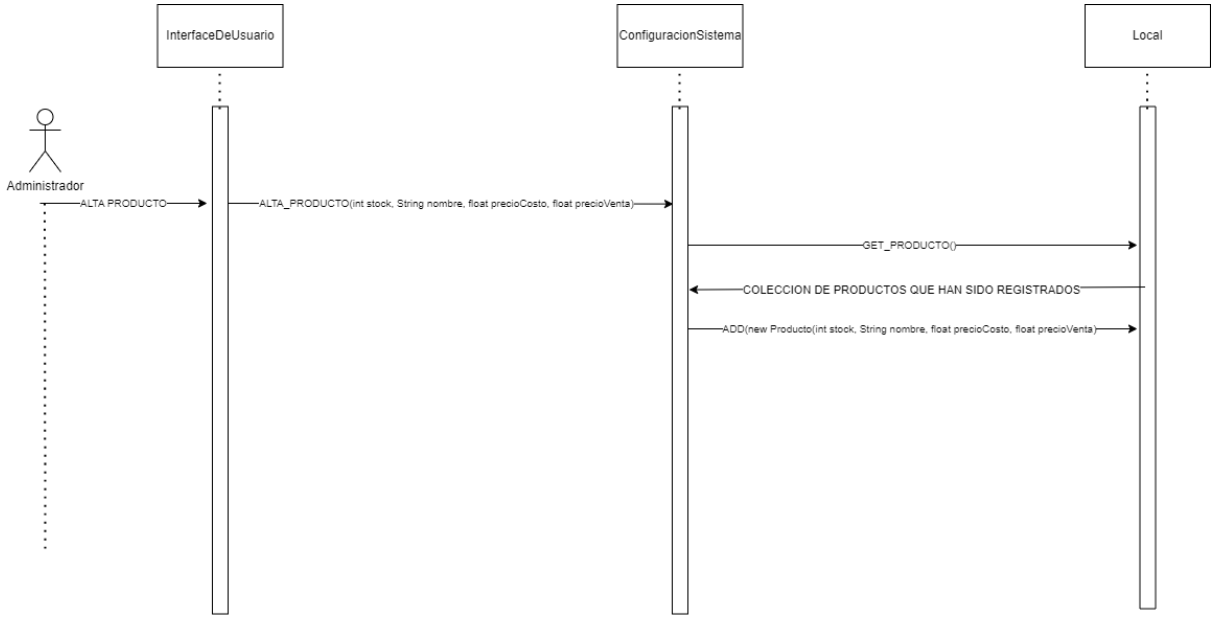


Nro Escenario	Descripcion
1	Colección de mozos con por lo menos algun mozo
2	Colección de mozos vacía

Condicion de entrada	Clases Validas	Clases Invalidas
mozo	mozo valido(1)	mozo invalido(2)

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{mozo[]}Escenario 1	Se da de baja operario	1
Incorrecta	{null}Escenario 2	Error	2

Alta Producto:

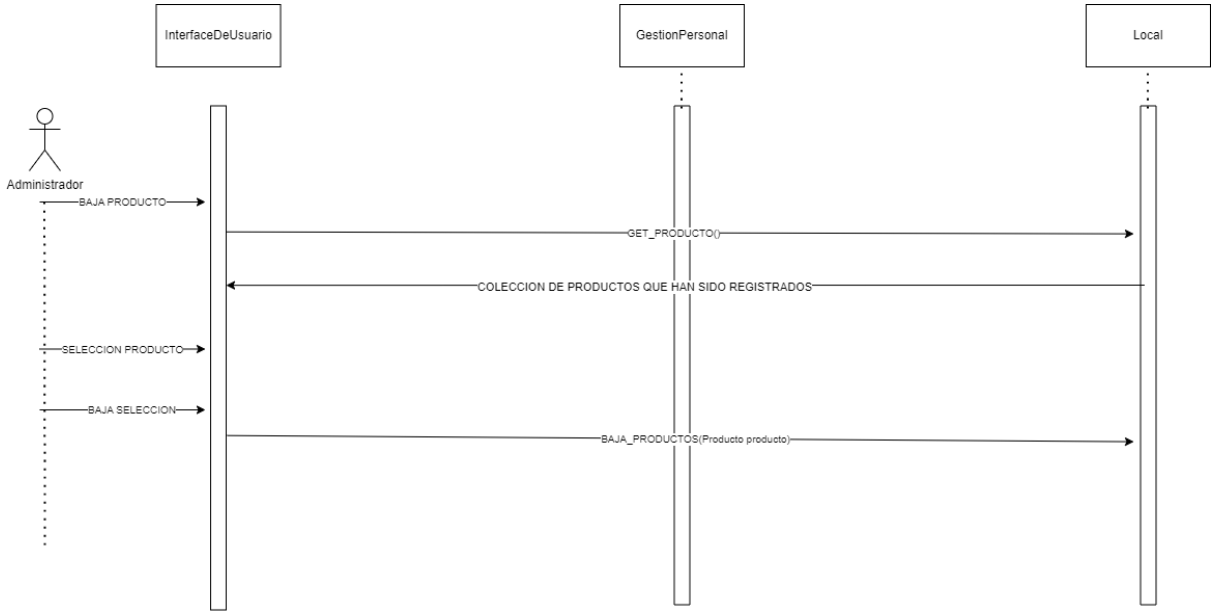


Nro Escenario	Descripcion
1	Colección de productos con por lo menos algun producto
2	Colección de productos vacía

Condicion de entrada	Clases Validas	Clases Invalidas
Estado de Parametros	Parametros validos(1)	Parametros validos(2)

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{0,255,"Agua Mineral", 50, 250}Escenario 1	Se da de alta al producto	1
Incorrecta	{0,255, null, 50, 250}Escenario 2	Error	2

Baja Producto:

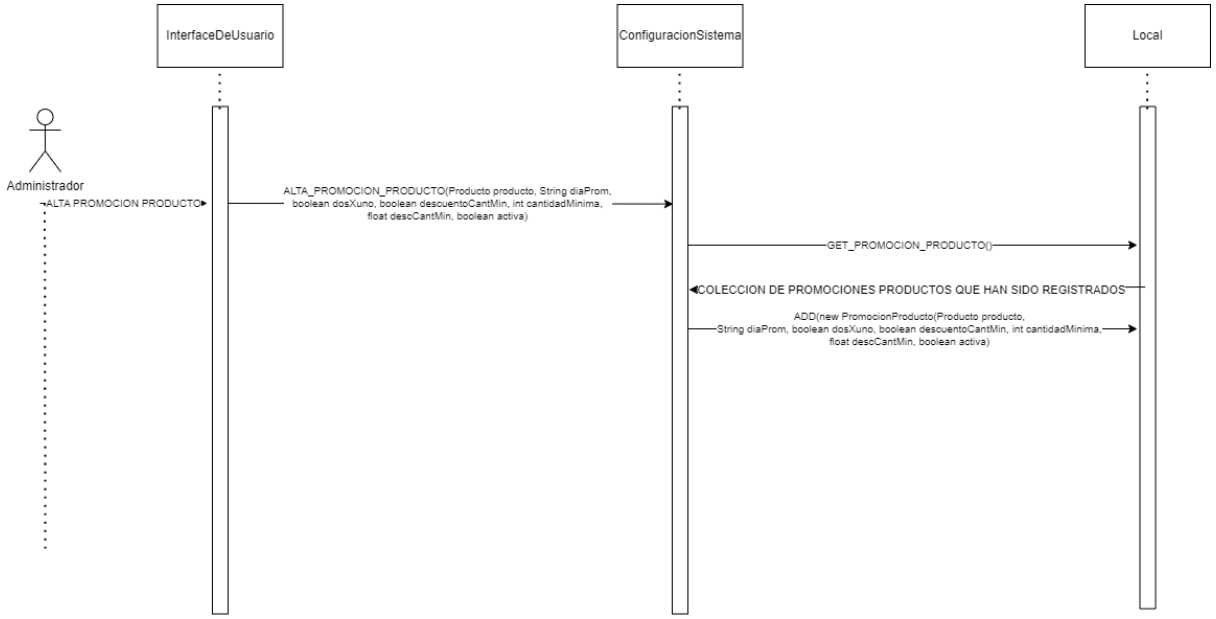


Nro Escenario	Descripcion
1	Colección de productos con por lo menos algun producto
2	Colección de productos vacía

Condicion de entrada	Clases Validas	Clases Invalidas
producto	producto valido(1)	producto invalido(2)

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{producto[]}Escenario 1	Se da de baja al producto	1
Incorrecta	{null}Escenario 2	Error	2

Alta PromocionProducto:

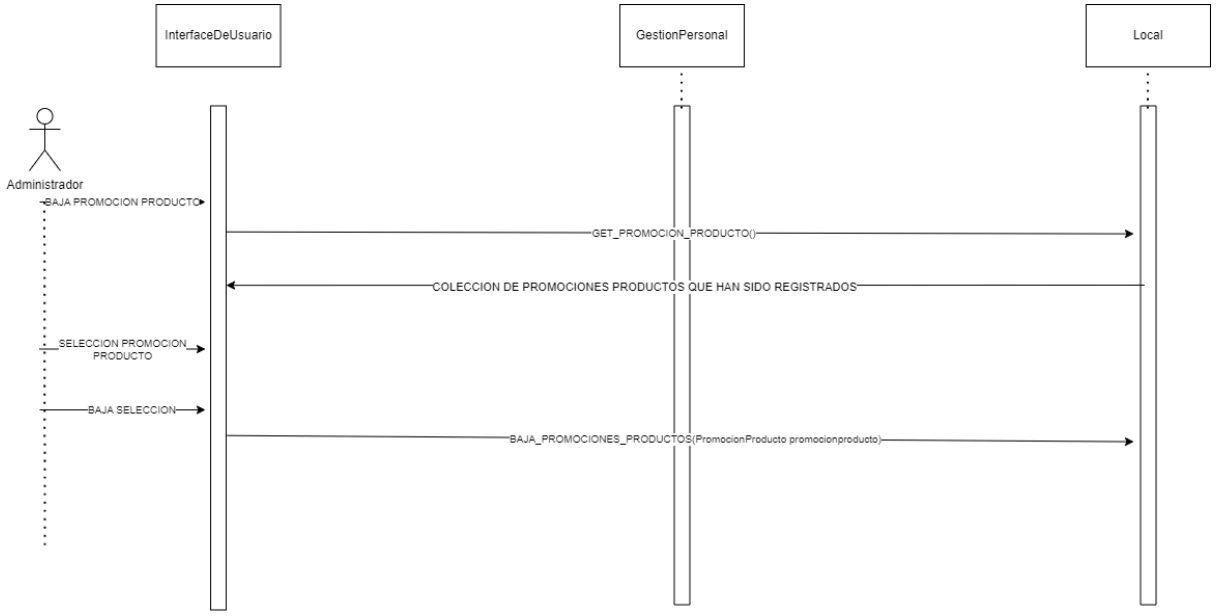


Nro Escenario	Descripcion
1	Colección de promociones productos con por lo menos algun promocion producto
2	Colección de promociones productos vacia

Condicion de entrada	Clases Validas	Clases Invalidas
Estado de Parametros	Parametros validos(1)	Parametros validos(2)

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{0,255,"Agua Mineral", 50, 250}Escenario 1	Se agrega la promocion al producto	1
Incorrecta	{0,255, null, 50, 250}Escenario 2	Error	2

Baja PromocionProducto:



Nro Escenario	Descripcion
1	Colección de promociones productos con por lo menos algun promocion producto
2	Colección de promociones productos vacia

Condicion de entrada	Clases Validas	Clases Invalidas
promocion producto	promocion producto valido(1)	promocion producto invalido(2)

Tipo de clase(correcta o incorrecta)	Valores de entrada	Salida esperada	Clases de prueba cubiertas
Correcta	{producto[], “Lunes”, true, true, 6, 25, true}Escenario 1	Se da de baja al producto	1
Incorrecta	{null, “Lunes”, true, true, 6, 25, true}Escenario 2	Error	2

Conclusión

A lo largo de esta re entrega pudimos implementar diversos métodos de testing. Se detectó mediante la aplicación de distintas técnicas de testeo cuales eran los errores del sistema de software implementado.

Al momento de hacer las pruebas de caja negra priorizamos a los métodos estructurales de la capa de negocio dado que no es viable realizar caja negra a la totalidad de los métodos. Algunos de los métodos no cubiertos por la caja negra fueron testeados utilizando el test de caja blanca.

Utilizamos el Javadoc lo cual nos facilitó la comprensión del código y nos ayudó al realizar las pruebas de caja negra.

Utilizamos las pruebas de caja blanca para verificar el funcionamiento de las líneas de código no ejecutadas hasta el momento. Para eso realizamos el gráfico ciclomático del código. Calculamos la complejidad del mismo, y concluimos haciendo los caminos básicos con su correspondiente salida esperada.

Durante la ejecución del test de GUI no todos los casos se ejecutaron de la mejor manera.

Se trabajó sobre un programa que tenía bastantes fallas, lo que nos dio un mejor acercamiento al proceso del testing y se pudieron detectar muchos errores ya sea por la documentación como por la implementación de la misma