Trabajo Práctico

“Testing de código del subgrupo Número 6”

Integrantes de subgrupo:

Bedini Tomas y Firmani Gregorio

Repositorio en el que codificamos nuestro proyecto

https://github.com/Greg1704/TallerDeProgramacionTP/tree/branch-login

Integrantes del subgrupo cuyo código será testeado:

Acuña Joaquin y Etchepare Mateo

Repositorio en el que se realizó el testing:

https://github.com/mateoetchepare/TPTaller1/tree/Test-Branch

Link del video:

https://www.youtube.com/watch?v=4aIEotnL47o

Introducción

El testing aplicado al sistema que nos fue provisto fue en gran parte aplicando el método de Caja Negra, habiendo realizado gran parte del proceso aplicando ese enfoque. Pensamos los casos de uso que se podrían aplicar y, basándonos en el Javadoc que se nos fue entregado junto con el trabajo, desarrollamos las clases de equivalencia que podrían darse, aplicando cuando se podía un análisis en los valores límite.

Que gran parte del trabajo se haya dedicado a pruebas de integración no quita que no hayamos hecho test unitarios y de caja blanca. Se analizaron dos métodos de forma exhaustiva, aplicando primero el método de caja blanca, y a partir de los casos de pruebas obtenidos se realizaron pruebas unitarias a los mismos utilizando la herramienta JUnit.

La selección de caminos utilizada en el método de caja blanca fue a través de la utilización del método general, eligiendo en primera instancia el camino con sentido funcional y luego los caminos con errores. Se aplicaron estos caminos también en las pruebas unitarias, en las que en ambos casos se logró recorrer todos los caminos de las mismas.

Para el testeo de GUI, elegimos la ventana de login debido a la complejidad que tienen las demás interfaces, por lo que optamos por tomar una ventana simple y aplicar todos los casos de prueba que podían suceder en esa ventana, de esta forma, generamos un testeo robusto y completo de una interfaz que, si bien es simple, fue enriquecedora porque pudimos detectar una gran cantidad de escenarios y casos de prueba en la misma.

Para el testeo de persistencia, verificamos la creación del archivo binario, la lectura del mismo para chequear que se recupere la información de forma correcta y por último la persistencia y posterior despersitencia para corroborar que lo que se crea se guarda y es posible recuperarlo nuevamente.

Datos de Login del archivo binario Sistema.bin

Para Admin:
Usuario: ADMIN1234
Contraseña: Gomez11

Para Operario:
Usuario: guccionel
Contraseña: Leonel1

Pruebas de Caja Negra

Casos de uso:

- *Creación de administrador*

Tabla de Particiones

Condición de entrada	Clases válidas	Clases inválidas
Nombre y Apellido	String **(1)**	-------
Nueva Contraseña	String. 6<=Tamaño<=12 Incluye al menos una mayúscula y un dígito.(**3**)	Tamaño<6 **(4.1)** Tamaño>12 **(4.2)** Ausencia de mayúscula **(4.3)** Ausencia de dígito **(4.4)**
Repetir nueva contraseña	String Nueva contraseña == Repetir nueva contraseña **(5)**	String Nueva Contraseña!= Repetir nueva contraseña **(6)**
Nombre del restaurante	String **(7)**	-------
Sueldo básico	Double **(9)**	Cualquier otro tipo de dato. **(2)**

Batería de Pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	Nombre y Apellido: “Juan Gomez” Nueva Contraseña: “Gomez11” Repetir nueva Contraseña:	Administrador creado. Acceso a la ventana de login.	**1, 3, 5, 7, 9**	Administrador creado. Acceso a la ventana de login.

	“Gomez11” Nombre de restaurante: “Cervecería” Sueldo básico: “60000.00”			
Incorrecta	Nombre y Apellido: “Juan Gomez” Nueva Contraseña: “Gomez11” Repetir nueva Contraseña: “Gomez22” Nombre de restaurante: “Cervecería” Sueldo básico: “60000.00”	Excepción: “Las contraseñas no coinciden” El administrador no es creado. Permanecemos en la ventana.	**6**	Excepción: “Las contraseñas no coinciden” El administrador no es creado. Permanecemos en la ventana.
Incorrecta	Nombre y Apellido: “Juan Gomez” Nueva Contraseña: “Gomez” Repetir nueva Contraseña: “Gomez” Nombre de restaurante: “Cervecería” Sueldo básico: “60000.00”	ContraseniaLongitudInvalidaException El administrador no es creado. Permanecemos en la ventana.	**4.1**	ContraseniaLongitudInvalidaException El administrador no es creado. Permanecemos en la ventana.
Incorrecta	Nombre y Apellido: “Juan Gomez” Nueva Contraseña: “Gomez12345678”	ContraseniaLongitudInvalidaException El administrador	**4.2**	ContraseniaLongitudInvalidaException El administrador

	Repetir nueva Contraseña: “Gomez12345678” Nombre de restaurante: “Cervecería” Sueldo básico: “60000.00”	no es creado. Permanecemos en la ventana.		no es creado. Permanecemos en la ventana.
Incorrecta	Nombre y Apellido: “Juan Gomez” Nueva Contraseña: “gomezzz” Repetir nueva Contraseña: “gomezzz” Nombre de restaurante: “Cervecería” Sueldo básico: “60000.00”	ContraseniaReqNoCumplidosException El administrador no es creado. Permanecemos en la ventana.	**4.3, 4.4**	ContraseniaReqNoCumplidosException El administrador no es creado. Permanecemos en la ventana.
Incorrecta	Nombre y Apellido: “Juan Gomez” Nueva Contraseña: “Gomez11” Repetir nueva Contraseña: “Gomez11” Nombre de restaurante: “Cervecería” Sueldo básico: “Jose”	Excepción: “Las contraseñas no coinciden” El administrador no es creado. Permanecemos en la ventana.	**2**	Excepción: “Las contraseñas no coinciden” El administrador no es creado. Permanecemos en la ventana.

- *Creación de un operario*

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Nombre de Usuario	String tamaño <= 10 caracteres **(1)**	tamaño > 10 caracteres **(2)**
Contraseña	String tamaño <= 12 caracteres **(3)**	Tamaño<6 **(4.1)** Tamaño>12 **(4.2)** Ausencia de mayúscula **(4.3)** Ausencia de dígito **(4.4)**
Nombre	String **(5)**	----------
Apellido	String **(7)**	----------

Batería de Pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	Nombre de Usuario: firmanig Contraseña: Gregorio1 Nombre: “Gregorio” Apellido: “Firmani”	Se crea el operario correctamente.	**1, 3, 5, 7**	Se crea el operario correctamente.
Incorrecta	Nombre de Usuario: firmanigggg Contraseña: Gregorio1 Nombre: “Gregorio” Apellido: “Firmani”	Excepción o aviso de que el nombre de usuario es mayor a 10 caracteres	**2**	Se crea el operario de igual manera.
Incorrecta	Nombre de Usuario:	ContraseniaLongi	**4.1**	ContraseniaLongit

	firmanig Contraseña: Greg1 Nombre: “Gregorio” Apellido: “Firmani”	tudInvalidaExcep tion El administrador no es creado. Permanecemos en la ventana.		udInvalidaExcepti on El administrador no es creado. Permanecemos en la ventana.
Incorrecta	Nombre de Usuario: firmanig Contraseña: Gregorio12345 Nombre: “Gregorio” Apellido: “Firmani”	ContraseniaLongi tudInvalidaExcep tion El administrador no es creado. Permanecemos en la ventana.	**4.2**	ContraseniaLongit udInvalidaExcepti on El administrador no es creado. Permanecemos en la ventana.
Incorrecta	Nombre de Usuario: firmanig Contraseña: gregorio Nombre: “Gregorio” Apellido: “Firmani”	ContraseniaReqN oCumplidosExce ption El administrador no es creado. Permanecemos en la ventana.	**4.3, 4.4**	ContraseniaReqNo CumplidosExcepti on El administrador no es creado. Permanecemos en la ventana.

- *Creación de un mozo*

Tabla de Particiones

Condición de entrada	Clases válidas	Clases inválidas
Nombre	String **(1)**	**----------**
Apellido	String **(3)**	----------

Cantidad de hijos	Integer Cantidad de hijos >= 0 **(5)**	Otro tipo de dato **(6.1)** Cantidad de hijos < 0 **(6.2)**
Fecha de Nacimiento	LocalDate formato: “dd/MM/yyyy” **(7)**	Otro tipo de dato **(8.1)** LocalDate con formato distinto **(8.2)**

Batería de Pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	Nombre: “Gregorio” Apellido: “Firmani” Cantidad de hijos: 2 Fecha de nacimiento: 17/04/2001	Se crea el mozo correctamente.	**1,3,5,7**	Se crea el mozo correctamente.
Incorrecta	Nombre: “Gregorio” Apellido: “Firmani” Cantidad de hijos: “aaa” Fecha de nacimiento: 17/04/2001	Excepción o aviso sobre la diferencia en formato en la cantidad de hijos	**6.1**	No salta ningún aviso o excepción, pero el programa sigue funcionando sin problema y sin crear al mozo.
Incorrecta	Nombre: “Gregorio” Apellido: “Firmani” Cantidad de hijos: -1 Fecha de nacimiento: 17/04/2001	Excepción o aviso sobre la cantidad de hijos menor a 0.	**6.2**	Aviso: “Ingresa una cantidad de hijos válida y una fecha de nacimiento válida(dd/MM/yyyy)”
Incorrecta	Nombre: “Gregorio”	Excepción o	**8.1**	Aviso: “Ingresa

	Apellido: “Firmani” Cantidad de hijos: 2 Fecha de nacimiento: 17 de abril de 2001	aviso sobre la diferencia en el tipo de dato.		una cantidad de hijos válida y una fecha de nacimiento válida(dd/MM/yyyy)”
Incorrecta	Nombre: “Gregorio” Apellido: “Firmani” Cantidad de hijos: 2 Fecha de nacimiento: 2001-04-17	Excepción o aviso sobre la diferencia de formato.	**8.2**	Aviso: “Ingresa una cantidad de hijos válida y una fecha de nacimiento válida(dd/MM/yyyy)”

- *Creación de un producto*

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Nombre	String **(1)**	-----
Stock	Integer Stock >= 0 **(3)**	Stock < 0 **(4)**
Id	Integer Id no repetido **(5)**	Id repetido **(6)**
Precio de Compra	Double Precio de compra > 0 **(7)**	Precio de costo > 0 **(8)**
Precio de Venta	Double Precio de venta >= Precio de compra **(9)**	Precio de venta<Precio de Compra **(10)**

Batería de Pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	Nombre: “Milanesa” Stock: 30 Id: 5 Precio de Compra: 200.0 Precio de Venta: 500.0	Se crea el producto correctamente.	**1,3,5,7,9**	Se crea el producto correctamente.
Incorrecta	Nombre: “Milanesa” Stock: -5 Id: 5 Precio de Compra: 200.0 Precio de Venta: 500.0	Excepción: “El stock es negativo” No se crea el producto.	**4**	Se crea el producto de igual manera..
Incorrecta	Nombre: “Milanesa” Stock: 30 Id: 5(Se crea previamente otro producto con Id: 5) Precio de Compra: 200.0 Precio de Venta: 500.0	ProductoExistenteException No se crea el producto.	**6**	ProductoExistenteException
Incorrecta	Nombre: “Milanesa” Stock: 30 Id: 5	Excepción que avise que el precio es	**8**	ProductoPreciosInvalidosException

	Precio de Compra: -200.0 Precio de Venta: 500.0	negativo.		
Incorrecta	Nombre: “Milanesa” Stock: 30 Id: 5 Precio de Compra: 500.0 Precio de Venta: 300.0	Excepción que avise que el precio de venta es menor al de compra.	**10**	ProductoPreciosInvalidosException

Aclaraciones:

- No se consideró de manera directa que el Precio de Venta sea negativo, ya que en ese caso se incumpliría previamente que el precio de compra sea negativo o mayor al de venta
- Tanto para los casos nombrados en la aclaración anterior como para los casos de las clases inválidas **(8) y (10)** salta el mismo mensaje de error, proveniente de la misma excepción. Lo cual no solo es confuso, sino que contradice el contrato del javadoc, donde la excepción “ProductoPreciosInvalidosException” se supone que solo cubre el caso en el que el Precio de Compra sea mayor al Precio de Venta.

- *Baja de un producto*

Escenario 1: El producto no se encuentra asociado a una comanda

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
producto	Objeto del tipo Producto producto != null **(1)**	----------

Batería de Pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	producto	Se elimina el producto de la lista de productos.	**1**	Se elimina el producto de la lista de productos.

Escenario 2: El producto se encuentra asociado a una comanda

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
producto	Objeto del tipo Producto producto != null **(1)**	----------

Batería de Pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	producto	Excepción o aviso sobre que el producto se encuentra asociado a una comanda.	**1**	Se elimina el producto de la lista de productos de igual manera.

- *Estadísticas del mozo*

Escenario 1: El mozo no tiene ventas asociadas a sus estadísticas

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
mozo	Objeto del tipo Mozo **(1)**	----------

Batería de Pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	mozo: Objeto mozo	Mensaje que indique que el mozo no realizó ninguna venta.	**(1)**	MozoSinVentasException

Escenario 1: El mozo tiene ventas asociadas a sus estadísticas

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
mozo	Objeto del tipo Mozo **(1)**	----------

Batería de Pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	mozo: Objeto del tipo mozo	Muestra en pantalla de las ventas hechas por el mozo	**(1)**	Ventas hechas por el mozo y el promedio de las mismas.

- *Estadísticas de empleado con mayor/menor volumen de venta*

Aclaración: No se armó la tabla de particiones ni baterías de prueba para estos casos de uso debido a que no hay un parámetro de entrada.

Escenario 1: No hay mozos en el sistema

-Salida esperada: Aviso en pantalla de que no existen mozos.
-Salida obtenida: Salta la excepción NoHayFacturasException.

Escenario 2: Hay mozos pero no hay facturas

-Salida esperada: Aviso en pantalla de que no existen facturas.
-Salida obtenida: Salta la excepción NoHayFacturasException.

Escenario 3: Hay mozos y facturas

-Salida esperada: Muestra en pantalla de mozo con mayor y menor volumen de venta.
-Salida obtenida: Muestra en pantalla de mozo con mayor y menor volumen de venta.

Caso particular:
-Si se llegara a crear 2 mozos, asignarles una factura a ambos. Luego eliminar uno y al otro asignarle otra factura nueva, al pedir las estadísticas nuevamente al mozo que quedó se le asignan sus facturas y las del mozo que ya no existe.

- *Creación de Promociones Permanentes*

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas

producto	Objeto del tipo Producto **(1)**	Null
Promo tipo	Boolean Promo tipo = 2x1 || Por cantidad **(3)**	Null **(4)**
Cantidad Mínima	Integer Cantidad Mínima > 0 **(5)**	Null **(6.1)** String **(6.2)** Cantidad mínima <= 0 **(6.3)**
Precio(unitario)	Double Precio > 0 **(7)**	Null **(8.1)** String **(8.2)** Precio <= 0 **(8.3)**
Dias de Promo	String **(9)**	----------

Batería de Pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	producto: Objeto del tipo Producto PromoTipo = 2x1 Cantidad mínima = 1 Precio = 150 Dias de Promo: Lunes	Se genera la promoción	**1,3,5,7,9**	Se genera la promoción.
Incorrecta	producto: Null PromoTipo = 2x1 Cantidad mínima = 1 Precio = 150 Dias de Promo: Lunes	Excepción o aviso sobre la ausencia de un producto seleccionado.	**2**	Aviso: “Seleccione un producto antes de agregarlo a promoción.”
Incorrecta	producto: Objeto del tipo	Excepción o	**4**	Aviso: “Seleccione

	Producto PromoTipo = Null Cantidad mínima = 1 Precio = 150 Dias de Promo: Lunes	aviso sobre la ausencia de una promoción seleccionada.		el tipo de promoción".
Incorrecta	producto: Objeto del tipo Producto PromoTipo = 2x1 Cantidad mínima = Null Precio = Null Dias de Promo: Lunes	Excepción o aviso sobre la ausencia del dato de la Cantidad mínima.	**6.1, 8.1**	Aviso: "Completa los campos de cantidad y precio"
Incorrecta	producto: Objeto del tipo Producto PromoTipo = 2x1 Cantidad mínima = "perro" Precio = "gato" Dias de Promo: Lunes	Excepción o aviso sobre el ingreso de tipo incorrecto de datos	**6.2, 8.2**	Aviso: "Número mal ingresado, ingrese un número entero"
Incorrecta	producto: Objeto del tipo Producto PromoTipo = 2x1 Cantidad mínima = 1 Precio = 150 Dias de Promo: Lunes	Excepción o aviso de que el precio unitario es menor o igual a 0.	**8.3**	Aviso: "usted ingresó un precio unitario supuestamente en descuento que es mayor al precio de venta del producto"

Observaciones:

-Se puede intentar generar la promoción permanente sin haber completado varios campos, aunque están cubiertos con avisos o excepciones que evitan que el programa deje de funcionar.

-Se puede crear correctamente una promoción permanente sin haberle asignado el día. Pero esta no se puede activar hasta que se le haya asignado uno.
-Se puede asignar un mismo día múltiples veces en una promoción.

- *Creación de Promos Temporales*

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
Nombre	String **(1)**	--------
Acumulable	Boolean **(3)**	--------
Descuento	Integer **(5)**	String **(6.1)** Descuento <= 0 **(6.2)** Descuento >=100 **(6.3)**
Método de pago	String Método de pago = Efectivo || Tarjeta || Cuenta DNI || Mercado pago **(7)**	--------
Dias de Promo	String **(9)**	--------

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	Nombre: “Lunes de promo” Acumulable: True Descuento: 1 Método de pago:Efectivo	Se crea la promoción temporal correctamente.	**1, 3, 5, 7, 9**	Se crea la promoción temporal correctamente.

	Dias de promo: Lunes			
Incorrecta	Nombre: “Lunes de promo” Acumulable: False Descuento: “uno” Método de pago:Efectivo Dias de promo: Lunes	Excepción o aviso sobre la diferencia en tipo de dato.	**6.1**	Aviso por pantalla “Numero mal ingresado, ingrese un número entre 1 y 100”
Incorrecta	Nombre: “Lunes de promo” Acumulable: False Descuento: 0 Método de pago:Efectivo Dias de promo: Lunes	Excepción o aviso sobre el valor incorrecto de descuento.	**6.2**	Se crea la promoción temporal de igual manera.
Incorrecta	Nombre: “Lunes de promo” Acumulable: False Descuento: -2 Método de pago:Efectivo Dias de promo: Lunes	Excepción o aviso sobre el valor incorrecto de descuento.	**6.2**	Aviso por pantalla “Numero mal ingresado, ingrese un número entre 1 y 100”
Incorrecta	Nombre: “Lunes de promo” Acumulable: False Descuento: 100 Método de pago:Efectivo Dias de promo: Lunes	Excepción o aviso sobre el valor incorrecto de descuento.	**6.3**	Se crea la promoción temporal de igual manera.
Incorrecta	Nombre: “Lunes de promo” Acumulable: False Descuento: 101 Método de pago:Efectivo	Excepción o aviso sobre el valor incorrecto de descuento.	**6.3**	Aviso por pantalla “Numero mal ingresado, ingrese un número entre 1 y 100”

	Dias de promo: Lunes			

Observaciones:

- Como se puede ver, en algunas clases de prueba, si el descuento es igual a 0 o 100, se crea la promoción de igual manera, pero para valores límite como -1 o 101, la promoción no se llega a crear.
- Se puede crear correctamente una promoción temporal sin haberle asignado el día. Pero esta no se puede activar hasta que se le haya asignado uno.
- Se puede asignar un mismo día múltiples veces en una promoción.

- *Agregar pedido a Comanda*

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
producto	Objeto del tipo Producto producto != null **(1)**	----------
cantidad	cantidad<=stock del producto **(3)**	cantidad>stock del producto**(4)**

Batería de Pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	producto: -nombre “milanesa” -stock: 100 cantidad: 1	Se agrega una milanesa a la comanda.	**1, 3**	Se agrega una milanesa a la comanda.
Correcta	producto:	Se agregan 99	**1, 3**	Se agregan 99

	-nombre “milanesa” -stock: 99 cantidad: 99	milanesas a la comanda.		milanesas a la comanda.
Incorrecta	producto: -nombre “pasta” -stock: 100 cantidad: 101	Error o aviso sobre el intento de agregar más pasta de la que se tiene en stock.	**4**	PedidoInvalidoException

- *Sacar pedido de comanda*

  Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
pedido	Objeto del tipo Pedido producto != null **(1)**	----------

Batería de Pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	pedido: -Nombre Producto: “Milanesa” -cantidad: 20	Se elimina el pedido de la comanda.	**1**	Se elimina el pedido de la comanda.

Observaciones:

- Al eliminar el pedido de la comanda, no se recupera la cantidad del producto que se hizo en el pedido.

- *Cierre de mesa y facturación*

Escenario 1: Cierra mesa con producto con promoción 2x1 activa.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
comanda	comanda != null **(1)**	----------
forma de pago	forma de pago = “Efectivo” || “Tarjeta” || “Mercado Pago” || “Cuenta DNI” **(3)**	----------

Batería de Pruebas

(Pedido de la comanda: Producto: Milanesa(precio: 200) cantidad: 5)

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	comanda forma de pago: “Efectivo”	Precio de la factura: 600$	**1, 3**	Precio de la factura: 1000$

Escenario 2: Cierra mesa con producto con promoción Descuento por Cantidad activa.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
comanda	comanda != null **(1)**	----------
forma de pago	forma de pago = “Efectivo” || “Tarjeta” ||“Mercado Pago” || “Cuenta DNI” **(3)**	----------

Batería de Pruebas

(Pedido de la comanda: Producto: Pasta(precio: 300) cantidad: 5)

(Promoción aplicada: Cantidad mínima: 3 Precio unitario: 190$)

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	comanda forma de pago: “Efectivo”	Precio de factura: 950$	**1, 3**	Precio de factura: 1500$

Escenario 3: Cierra mesa con promoción temporal activa.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
comanda	comanda != null **(1)**	----------
forma de pago	forma de pago = “Efectivo” ‖ “Tarjeta” ‖“Mercado Pago” ‖ “Cuenta DNI” **(3)**	----------

Batería de Pruebas

(Pedido de la comanda: Producto: Milanesa(precio: 200) cantidad: 5)

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	comanda Forma de pago: “Efectivo”	Precio de factura: 500$	**1, 3**	Precio de factura: 1000$

Escenario 4: Cierra mesa con producto en promoción 2x1 activa y promoción temporal activa no acumulable.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas

comanda	comanda != null **(1)**	----------
forma de pago	forma de pago = “Efectivo” || “Tarjeta” ||“Mercado Pago” || “Cuenta DNI” **(3)**	----------

Batería de Pruebas

(Pedidos de la comanda: -Producto: Milanesa(precio: 200) cantidad: 5

-Producto: Pasta(precio: 300) cantidad: 2)

(Producto con promoción 2x1: Milanesa)

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	comanda Forma de pago: “Efectivo”	Precio de factura:900$	**1, 3**	Precio de factura:1600$

Escenario 5: Cierra mesa con producto en promoción 2x1 activa y promoción temporal activa acumulable.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
comanda	comanda != null **(1)**	----------
forma de pago	forma de pago = “Efectivo” || “Tarjeta” ||“Mercado Pago” || “Cuenta DNI” **(3)**	----------

Batería de Pruebas

(Pedidos de la comanda: -Producto: Milanesa(precio: 200) cantidad: 5

-Producto: Pasta(precio: 300) cantidad: 2)

(Producto con promoción 2x1: Milanesa)

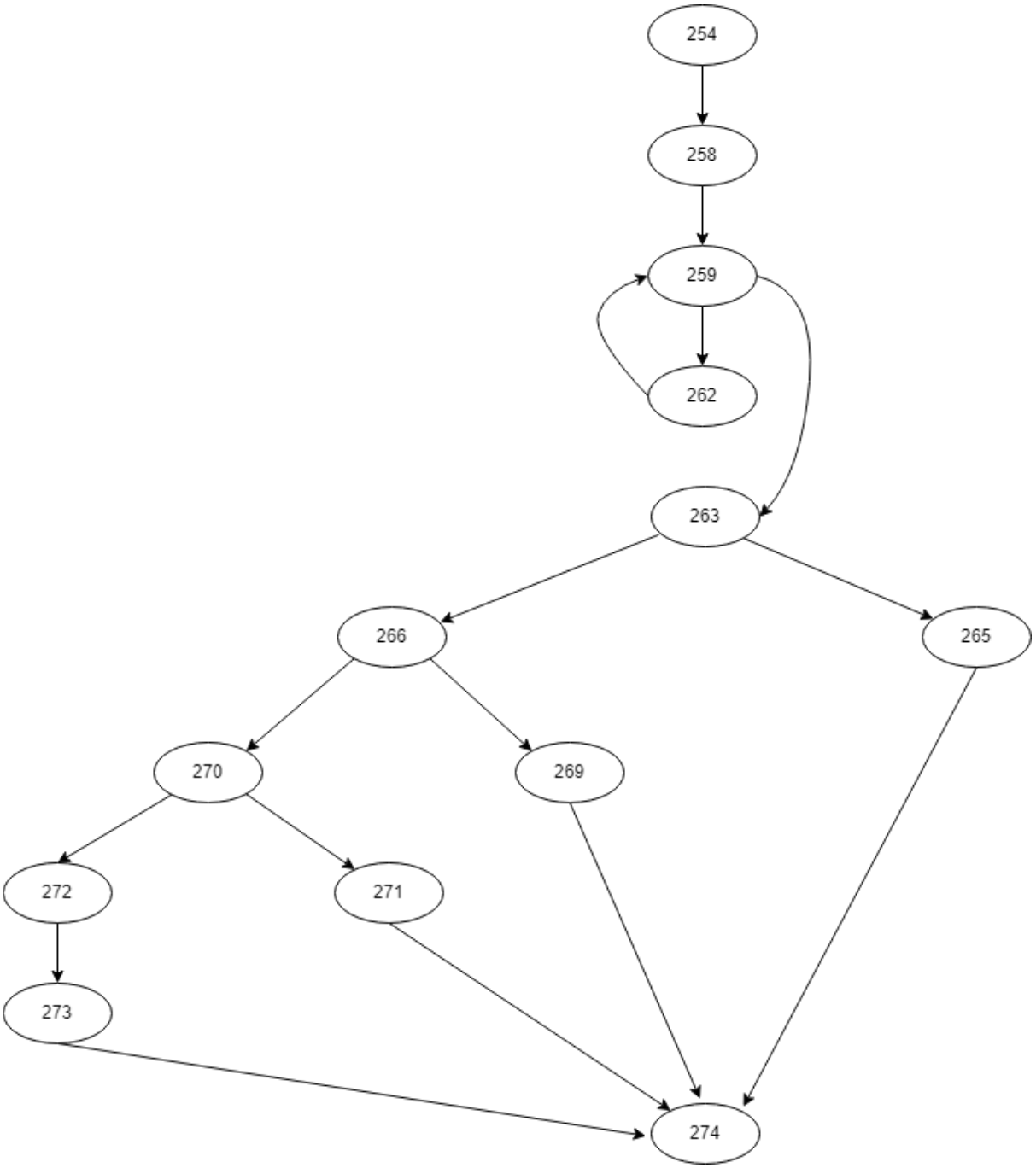
Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salida obtenida
Correcta	comanda Forma de pago: “Efectivo”	Precio de factura:600$	**1, 3**	Precio de factura:1600$

Observaciones generales de los escenarios:

- Como podemos ver, en ningún caso se aplicaron las promociones solicitadas, los totales obtenidos son simplemente la suma de los precios de venta de cada producto.

Pruebas de Caja blanca

Método loginOperario de clase Sistema



Complejidad Ciclomática: arcos - nodos + 2 = 16 - 13 + 2 = 5 // Número de nodos condición + 1 = 4 + 1 = 5.

Caminos:

1. 254 - 258 - 259 - 262 - 259 - 263 - 266 - 270 - 272 - 273 - 274

2. 254 - 258 - 259 - 262 - 259 - 263 - 265 - 274
3. 254 - 258 - 259 - 262 - 259 - 263 - 266 - 269 - 274
4. 254 - 258 - 259 - 262 - 259 - 263 - 266 - 270 - 270 - 271 - 274

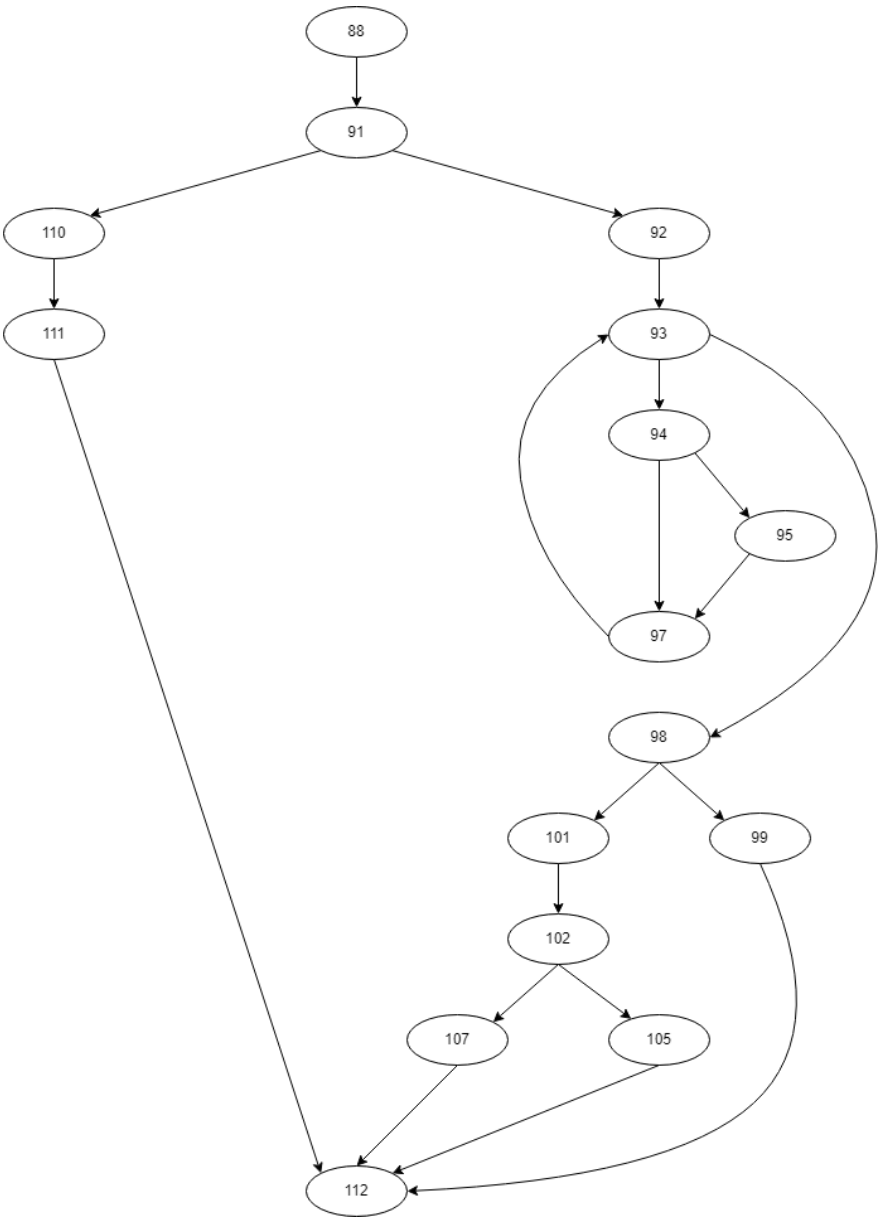
Camino indep	Datos de entrada	Procedimiento	Salida Esperada
1	Usuario y contraseña que se encuentren en el sistema	Se ingresan datos correctos, sabiendo que el operario se encuentra registrado en el sistema.	Se retorna el Operario que se buscaba.
2	Usuario no se encuentra en el sistema, el dato que contenga contraseña es indistinto porque no afecta al desarrollo	Se ingresa un nombre de usuario que no se encuentra en el sistema, por lo que se recorre el while hasta cumplirse la condición “i < j”	Se lanza excepción UsuarioIncorrectoException.
3	Usuario se encuentra en el sistema pero la contraseña es incorrecta.	El nombre es correcto por lo que se procede a corroborar que la contraseña sea correcta pero no lo es.	Se lanza excepción ContraseniaIncorrectaException
4	Usuario y contraseña son datos correctos.	Se encuentra al operario en el sistema, pero el estado del mismo es “Inactivo”.	Se lanza excepción UsuarioInactivoException.

(No se analiza si no hay operarios ya que es una precondición que haya operarios en el sistema).

Imagen del código del método analizado:

```java
public Operario loginOperario(String usuario, String contrasenia)
        throws LoginIncorrectoException, CambioObligatorioContraseniaException {
    int j, i = 0;

    j = operarios.size();
    while (i < j && (!operarios.get(i).getNombreUsuario().equals(usuario))) {

        i++;
    }
    if (i == j) {

        throw new UsuarioIncorrectoException();
    } else if (operarios.get(i).getNombreUsuario().equals(usuario)
            && !operarios.get(i).getPassword().equals(contrasenia)) {

        throw new ContraseniaIncorrectaException();
    } else if (operarios.get(i).isActivo() == false)
        throw new UsuarioInactivoException();
    else
        return operarios.get(i);
}
```

Método agregarMesa de clase Mozo



Complejidad Ciclomática: arcos - nodos + 2 = 20 - 16 + 2 = 6 // Número de nodos condición + 1 = 5 + 1 = 5.

Caminos:

1. 88 - 91 - 92 - 93 - 94 - 97 - 93 - 98 - 101 - 102 - 105 - 112
2. 88 - 91 - 110 - 111 - 112
3. 88 - 91 - 92 - 93 - 94 - 97 - 93 - 98 - 99 - 112
4. 88 - 91 - 92 - 93 - 94 - 97 - 93 - 98 - 101 - 102 - 107 - 112

Camino indep	Datos de entrada	Procedimiento	Salida Esperada
1	Objeto mesa con atributo nombreMozo en Null y habilitado en true.	La mesa que se quiere asignar no está asignada a ningún mozo	Se le asigna la mesa al mozo.
2	Objeto mesa	Es indiferente los valores que tengan los atributos de mesa pues el mozo no se encuentra activo.	Se lanza excepción MozoInvalidoExceptio n.
3	Objeto mesa con atributo nombreMozo distinto de Null.	La mesa se encuentra ya asignada a un mozo.	Se lanza excepción MesaYaAsignadaExcep tion.
4	Objeto mesa con atributo nombreMozo en Null y deshabilitado en true.	La mesa no está asignada, pero al no estar habilitada no puede ser asignada a un mozo.	Se lanza excepción MesaDeshabilitadaExce ption.

Imagen del código del método analizado:

```java
public void agregarMesa(Mesa mesa) throws MozoInvalidoException,MesaDeshabilitadaException, MesaYaAsignadaException{
    boolean yaAsignada=false;

    if (this.getEstado() == "Activo") {
        Iterator<Mozo> itMozo= Sistema.getInstancia().getMozos().iterator();
        while(itMozo.hasNext() && !yaAsignada) {
            if(itMozo.next().getMesas().contains(mesa)) {//algun mozo ya tiene esta mesa asignada
                yaAsignada=true;
            }
        }
        if(yaAsignada) {
            throw new MesaYaAsignadaException();
        }
        else {
            if (mesa.getHabilitado() == true) {
                mesa.setNombreMozo(nombreYApellido);
                this.mesas.add(mesa);
            }
            else
                throw new MesaDeshabilitadaException("La mesa esta deshabilitada");
            System.out.println("Se agrego la mesa al mozo");
        }
    } else
        throw new MozoInvalidoException();
}
```

Pruebas Unitarias con JUnit

Prueba en método loginOperario de clase Sistema

Se realizó el testeo en la clase SistemaTest en el paquete test.

Se realizaron métodos de test para cada uno de los cuatro escenarios especificados en la caja blanca con sus respectivas condiciones, partiendo de la precondición de que no puede ser nulo el arraylist de operarios.

Se observa que el método funciona y cumple con lo especificado. Como corrección, hay una excepción especificada en throws que realmente no tiene motivo para estar ahí, pues todas las excepciones que se encuentran dentro del método heredan de LoginIncorrectoException(). Es por esto que CambioObligatorioContraseniaException() debería borrarse del código.

En cuanto a recorridos, no se dejó ninguna línea sin controlarse, por lo que el código no dispone de partes a las que no es posible llegar. Se adjunta imagen debajo.

```
253
254     public Operario loginOperario(String usuario, String contrasenia)
255             throws LoginIncorrectoException, CambioObligatorioContraseniaException {
256         int j, i = 0;
257
258         j = operarios.size();
259         while (i < j && (!operarios.get(i).getNombreUsuario().equals(usuario))) {
260
261             i++;
262         }
263         if (i == j) {
264
265             throw new UsuarioIncorrectoException();
266         } else if (operarios.get(i).getNombreUsuario().equals(usuario)
267                 && !operarios.get(i).getPassword().equals(contrasenia)) {
268
269             throw new ContraseniaIncorrectaException();
270         } else if (operarios.get(i).isActivo() == false)
271             throw new UsuarioInactivoException();
272         else
273             return operarios.get(i);
274     }
275
```

Prueba en metodo agregarMesa de clase Mozo

Se realizó el testeo en la clase SistemaTest en el paquete test.

Se realizaron test para cada uno de los 4 caminos que se plantearon en el método de caja blanca.

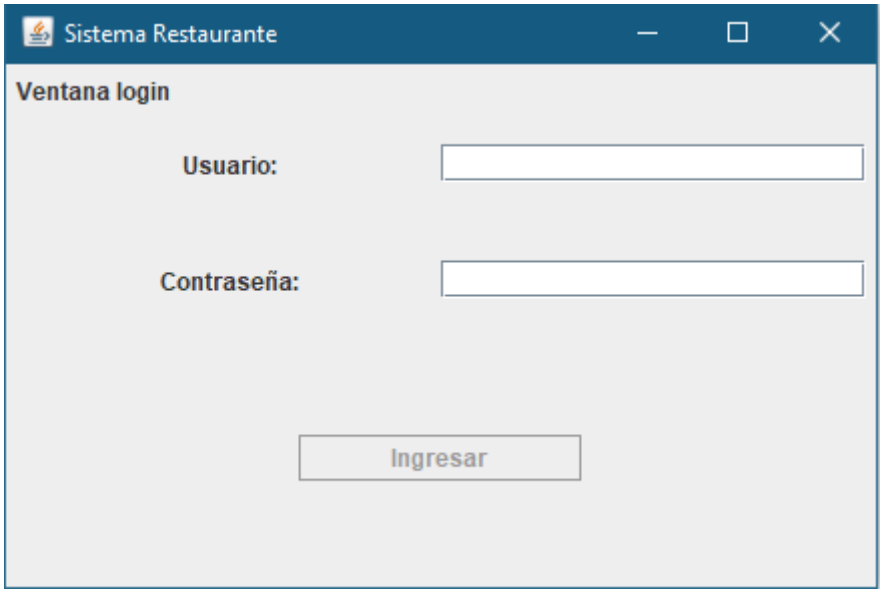
Como puede verse, pudo recorrerse el código en su totalidad, pasando por todos los condicionales y cumpliendo todas las situaciones en las que podía darse, corroborando cada excepción y el caso de éxito,

```
88    public void agregarMesa(Mesa mesa) throws MozoInvalidoException,MesaDeshabilitadaException, MesaYaAsignadaException{
89        boolean yaAsignada=false;
90
91        if (this.getEstado() == "Activo") {
92            Iterator<Mozo> itMozo= Sistema.getInstancia().getMozos().iterator();
93            while(itMozo.hasNext() && !yaAsignada) {
94                if(itMozo.next().getMesas().contains(mesa)) {//algun mozo ya tiene esta mesa asignada
95                    yaAsignada=true;
96                }
97            }
98            if(yaAsignada) {
99                throw new MesaYaAsignadaException();
100           }
101           else {
102               if (mesa.getHabilitado() == true) {
103                   mesa.setNombreMozo(nombreYApellido);
104                   this.mesas.add(mesa);
105               }
106               else
107                   throw new MesaDeshabilitadaException("La mesa esta deshabilitada");
108               System.out.println("Se agrego la mesa al mozo");
109           }
110       } else
111           throw new MozoInvalidoException();
112   }
```

Pruebas de GUI

Para la prueba de la interfaz gráfica de usuario, decidimos testear el login de usuario. Esto por dos razones, por un lado, consideramos que es una ventana vital, ya que si esta ventana falla, se inutiliza completamente el programa, y por el otro lado, nos pareció también una ventana simple con algunos detalles interesantes.

Adjuntamos foto de la ventana:



GuiTestEnabledDisabled

Para este tipo de pruebas, previo a los tests, se creó el sistema, un operario, y las ventanas y controladores necesarias para el testeo. Además, se asignaron previamente los textFields y botones necesarios para que pueda funcionar todo correctamente.

Se nos presentaron 5 tests distintos:

1. *testA_IngresoSoloUsuario*

   En el primer caso, se ingresó solamente un usuario y se intentó presionar el botón. Para corroborar que todo funcionara y el botón no se habilitará, se aplico el siguiente aserto en el test:

   Assert.assertTrue("El boton no deberia estar habilitado",!boton.isEnabled());

El test arrojó resultados positivos, no llegó a presentar ningún problema.

2. *testB_IngresoSoloPassword*

   Para el segundo caso, en cambio, se ignoró el campo del usuario y se completó solo la contraseña. Y al igual que el caso anterior, se intentó presionar el botón esperando como resultado que no ocurriera nada.
   Para corroborar que todo funcionara y el botón no se habilitará, se aplico el siguiente aserto en el test:
   Assert.assertTrue("El boton no deberia estar habilitado",!boton.isEnabled());

   El test arrojó resultados positivos, no llegó a presentar ningún problema.

3. *testC_IngresoUsuarioYPasswordMenor6Car*

   En el tercer caso, se ingresó tanto el usuario como la contraseña, pero con la peculiaridad de que la contraseña será menor a 6 caracteres. Este caso surgió inicialmente al probar el programa por arriba, donde al ingresar una contraseña de ese tamaño, el botón de login permanecía inactivo, por lo cuál nos pareció interesante probarlo en los tests de GUI.
   Para corroborar que todo funcionara y el botón no se habilitará, se aplico el siguiente aserto en el test:

   Assert.assertTrue("El boton no deberia estar habilitado",!boton.isEnabled());

   El test arrojó resultados positivos, no llegó a presentar ningún problema.

4. *testD_IngresoUsuarioYPasswordMayor12Car*

   Como el caso anterior, si la contraseña superará los 12 caracteres, el botón tampoco debería ser activado, por lo que probamos un caso donde se ingresa el usuario, una contraseña mayor a 12 caracteres, y se clickea el botón.
   Para corroborar que todo funcionara y el botón no se habilitará, se aplico el siguiente aserto en el test:

Assert.assertTrue("El boton no deberia estar habilitado",!boton.isEnabled());

El test arrojó resultados positivos, no llegó a presentar ningún problema.

5. *testE_IngresoDatosUsuarioYPasswordCorrectos*

   En este último caso, se ingresó todo correctamente, un usuario y una contraseña que se encuentre entre los 6 y 12 caracteres y se clickeo el botón, el cual se espera que estuviera habilitado.
   Para corroborar que todo funcionara y el botón no se habilitará, se aplico el siguiente aserto en el test:

   Assert.assertTrue("El botón deberia estar habilitado", boton.isEnabled());

   El test arrojó resultados positivos, no llegó a presentar ningún problema.

En caso de que requieran ver el código completo, adjuntamos el link de la clase donde fueron realizados los tests en el repositorio de GitHub:
https://github.com/mateoetchepare/TPTaller1/blob/Test-Branch/src/test_Gui/GuiTestEnabledDisabled.java

GuiTestConjuntoVacio

Este escenario no fue generado ya que si la lista de operarios se encontrara vacía, los test a realizar serían exactamente los mismos que algunos de la siguiente parte, por lo cual decidimos omitir esta parte.

GuiTestConjuntoConDatos

Para este tipo de escenario, previo a los tests, se creó el sistema, un operario, y las ventanas y controladores necesarias para el testeo. Además, se asignaron previamente los textFields y botones necesarios para que pueda funcionar todo correctamente.
Tuvimos que modificar ligeramente el código para poder obtener los mensajes de los JOptionPane aplicando el método mostrado en la práctica de test de GUI.

1. *testA_UsuarioIncorrecto*

   En este caso, ingresamos los datos del operario pero con el usuario mal escrito.
   Se espera que una vez clickeado el botón de login, salte una notificación sobre el ingreso de un usuario incorrecto, para verificar que ocurriera esto, se ingresaron los siguientes asertos:

   Assert.assertTrue("El boton deberia estar habilitado", boton.isEnabled());
   Assert.assertEquals("Deberia haber salido el siguiente mensaje : Usuario no encontrado ", "Usuario no encontrado", op.getMensaje());

   El test arrojó resultados positivos, se lanzó el mensaje esperado.

2. *testB_ContraseniaIncorrecto*

   Para este caso, en cambio, se ingresa una contraseña mal escrita. Se espera que una vez clickeado el botón de login, salte una notificación sobre el ingreso de una contraseña errónea, para verificar que ocurriera esto, se ingresaron los siguientes asertos:

   Assert.assertTrue("El boton deberia estar habilitado", boton.isEnabled());
   Assert.assertEquals("Deberia haber salido el siguiente mensaje : Contrasenia Incorrecta ", "Contrasenia Incorrecta", op.getMensaje());

   El test arrojó resultados positivos, se lanzó el mensaje esperado.

3. *testC_OperarioInactivo*

   Para este caso, los datos del operario se ingresaron correctamente, pero se cambió su estado a falso(Inactivo) antes del inicio del test para corroborar que un operario inactivo no puede conectarse al sistema, como lo implica el contrato. Se espera que una vez clickeado el botón de login, salte una notificación sobre que el operario no puede acceder al sistema debido a su estado, para verificar que ocurriera esto, se ingresaron los siguientes asertos:

   Assert.assertTrue("El boton deberia estar habilitado", boton.isEnabled());
   Assert.assertEquals("Deberia haber salido el siguiente mensaje : El operario NO esta activo, no puede ingresar al sistema. Consulte con el administrador ", "El operario NO esta activo, no puede ingresar al sistema. Consulte con el administrador", op.getMensaje());

   El test arrojó resultados positivos, se lanzó el mensaje esperado. Pero por el otro lado, esto solo puede ocurrir si se hardcodea el estado del operario antes de iniciar el test. Ya que la ventana por si sola, puede crear operarios, pero no puede cambiar su estado en ningún momento, por lo cual permanecen siempre activos.

4. *testD_LoginCorrecto*

   El procedimiento de este test fue exactamente el mismo que el último test de los testEnabledDisabled(*testE_IngresoDatosUsuarioYPasswordCorrectos*), por lo cuál no hay mucho que agregar, fuera de que el test arrojó nuevamente resultados positivos.

En caso de que requieran ver el código completo, adjuntamos el link de la clase donde fueron realizados los tests en el repositorio de GitHub:
https://github.com/mateoetchepare/TPTaller1/blob/Test-Branch/src/test_Gui/GuiTestConjuntoConDatos.java

Test de Persistencia

Para el test de persistencia, se realizaron 3 casos distintos:

1. *testA_CrearArchivoCorrectamente*

   Para este primer test, se verificó que el método encargado de persistir cree correctamente el archivo. Para corroborar esto, se utilizó el siguiente aserto:

   Assert.assertTrue("Tendria que existir el archivo", archivo.exists());

   El test arrojó resultados positivos, el archivo se generó sin problemas.

2. *testB_LecturaCorrectaSistema*

   En este segundo test, persistimos un sistema y luego lo despersistimos, para verificar que el sistema siga existiendo y que mantenga su referencia a las estructuras encargadas de guardar los distintos tipos de datos(mozos, operarios, productos, etc.)
   Para corroborar esto, se utilizaron los siguientes asertos:

   Assert.assertNotEquals("Deberia haberse recuperado el Sistema", sist, null);
   Assert.assertNotEquals("Deberia haberse recuperado el ArrayList", sist.getProductos(), null);
   Assert.assertNotEquals("Deberia haberse recuperado el ArrayList", sist.getPromociones(), null);
   Assert.assertNotEquals("Deberia haberse recuperado el ArrayList", sist.getOperarios(), null);
   Assert.assertNotEquals("Deberia haberse recuperado el ArrayList", sist.getMozos(), null);
   Assert.assertNotEquals("Deberia haberse recuperado el ArrayList", sist.getMesas(), null);
   Assert.assertNotEquals("Deberia haberse recuperado el ArrayList", sist.getComandas(), null);

Assert.assertNotEquals("Deberia haberse recuperado el ArrayList", sist.getFacturas(), null);

El test arrojó resultados positivos, se recuperó el sistema y sus distintas estructuras de datos correctamente.

3. *testC_escrituraYLecturaCorrectaDeElementoDeArrayList*

Por último, se decidió tomar el siguiente paso, ya sabemos que se guardan las estructuras de datos del sistema, por lo que verificamos también que los elementos de al menos una de estas también se guarden.
Para este caso, creamos un Operario, lo guardamos en el sistema, persistimos el mismo, luego lo despersistimos, y verificamos que no se haya perdido el operario y sus datos.
Para corroborar esto, se utilizaron los siguientes asertos:

Assert.assertEquals("Los operarios deberian tener un mismo nombre y apellido",sist.getOperarios().get(0).getNombreYApellido(),operario.getNombreYApellido());
Assert.assertEquals("Los operarios deberian tener un mismo usuario",sist.getOperarios().get(0).getNombreUsuario(),operario.getNombreUsuario());
Assert.assertEquals("Los operarios deberian tener una misma contrasenia",sist.getOperarios().get(0).getPassword(),operario.getPassword());

El test arrojó resultados positivos, se pudo persistir y despersistir sin ningún problema los elementos de las estructuras de datos del Sistema.

En caso de que requieran ver el código completo, adjuntamos el link de la clase donde fueron realizados los tests de persistencia en el repositorio de GitHub:
https://github.com/mateoetchepare/TPTaller1/blob/Test-Branch/src/testPersistencia/TestPersistencia.java

Sección de incumplimientos del contrato

- El usuario del administrador no es ADMIN, sino ADMIN1234.
- En ningún momento la contraseña ADMIN1234 existiría siendo que primero crear al usuario administrador y luego te logueas.
- Las mesas se encuentran hardcodeadas, el sistema no te permite ingresar, eliminar o editar mesas.
- El sistema no permite cambiar el estado de los operarios.
- Un operario puede dar de alta a un producto.
- Un operario puede pedir estadísticas del sistema.
- Se puede eliminar un producto asociado a una comanda.

Conclusiones

Para concluir el trabajo nos sentimos conformes con nuestra labor realizada, ya que a pesar de las complicaciones que se nos presentaron durante la realización del trabajo, pudimos seguir adelante con el mismo logrando su realización.

Logramos entender la importancia del testing y lo vital que resulta para aplicar en las situaciones reales y cotidianas de la vida laboral, ya que al principio de la materia ninguno de nosotros entendíamos que el proceso de testeo era tan vital como el proceso de codear un programa.

Aprendimos a trabajar con contratos, a comprenderlos y a aplicarlos a la realidad, ya sea tanto para desarrollar un código como para testear el mismo. También entendimos la importancia de que éste sea bien documentado y no deje lugar a ambigüedades, ya que en los momentos que sucedió, sabíamos que esos requerimientos podrían ser problemas que deberíamos volver a tratar a futuro.

También comprendimos la necesidad de ser cuidadosos con nuestros códigos, a comentarlos en lugares donde podría haber dudas y a documentarlos de manera correcta, ya que la persona que reciba ese código ya sea para continuar nuestra labor como programadores, como para testearlo, puede tener sus dudas acerca de lo que hicimos, y la forma correcta de evacuarlas es viendo una documentación de los mismos que no deje lugar a ambigüedades.