Trabajo Práctico Especial

Número de grupo: 6
Integrantes del grupo: Mateo Etchepare, Joaquín Acuña Destreé, Gregorio Firmani, Tomás Bedini
Integrantes del subgrupo: Mateo Etchepare, Joaquín Acuña Destreé
Materia: Taller de Programación I
Universidad: Universidad Nacional de Mar del Plata
Repositorio de GitHub del programa codificado por el subgrupo: https://github.com/mateoetchepare/TPTaller1
Repositorio de GitHub del programa testeado por el subgrupo: https://github.com/Greg1704/TallerDeProgramacionTP
En este último repositorio, en el package "Testing" estarán nuestras clases de test.

Índice

Caja Negra: Test de integración

Todas las salidas obtenidas que sean salidas erróneas se marcarán con texto de color rojo.

Caso de uso #1: El admin se loguea por primera vez.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
nombre de usuario default	Cualquier string **1**	-
password default	Cualquier string **2**	-

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	usuario: "ADMIN" password: "ADMIN1234"	Mensaje emergente de "Ingrese nueva contraseña"	1, 2	Mensaje emergente de "Ingrese nueva contraseña"
Incorrecta	usuario: "ADMIN" password: "ADMIN"	excepción con Mensaje emergente de "Contraseña incorrecta"	1, 2	Mensaje emergente de "contraseña incorrecta"
Incorrecta	usuario: "AD" password: "ADMIN1234"	excepción con Mensaje emergente de "Usuario incorrecto"	1, 2	Mensaje emergente de "nombre de usuario incorrecto"

Caso de uso #2: Cambio obligatorio de contraseña por primera vez del ADMIN.

Tabla de particiones

Condición de entrada	Clases válidas	Clases inválidas
contraseña	Cualquier string diferente a la que viene por default **1**	{"ADMIN1234"} **2**

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	contraseña: {"pepe"} (en realidad, cualquier contraseña diferente a {"ADMIN1234"})	ventana emergente de "ingrese nombre y apellido"	1	ventana emergente de "ingrese nombre y apellido"
Incorrecta	contraseña: {"ADMIN1234"}	excepción con ventana emergente de "Ingrese una contraseña diferente a la anterior" (ya que el contrato pide un cambio obligatorio de contraseña)	2	ventana emergente de "ingrese nombre y apellido"

Caso de uso #3: Agregar operarios al sistema

Condición de entrada	Clases válidas	Clases inválidas
usuario	Cualquier string menor o igual a 10 caracteres **1**	- String mayor a 10 caracteres **2**
password	String entre 6 y 12 caracteres con al menos una mayúscula y un dígito **3**	String menor a 6 caracteres **4** String mayor a 12 caracteres **5** String sin mayúscula **6** String sin dígito **7**
nombreYApellido	Cualquier string **8**	-

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	usuario: Jose password: Josesito124 nombreYApellido: Josesito	Se agrega a la lista	1, 3, 8	Se agrega a la lista
Incorrecta	usuario: Jose password: Jose1 nombreYApellido: Josesito	excepción con ventana emergente de "la contraseña es menor a 6 caracteres"	1, 4, 8	excepción con ventana emergente de "la contraseña es menor a 6 caracteres"
Incorrecta	usuario: Jose	excepción con ventana emergente	1, 5, 8	excepción con

	password: Josesito1Josesito nombreYApellido: Josesito	de "la contraseña es mayor a 12 caracteres"		ventana emergente de "la contraseña es mayor a 12 caracteres"
Incorrecta	usuario: Jose password: JosesitoJJ nombreYApellido: Josesito	excepción con ventana emergente de "la contraseña debe tener al menos un número"	1, 7, 8	excepción con ventana emergente de "la contraseña debe tener al menos un numero"
Incorrecta	usuario: Jose password: josesito123 nombreYApellido: Josesito	excepción con ventana emergente de "la contraseña no cuenta con al menos una mayúscula"	1, 6, 8	excepción con ventana emergente de "la contraseña debe tener al menos una mayúscula"
Incorrecta	usuario: Jose password: Josesito124 nombreYApellido: Josesito (siendo que ya se agregó previamente)	excepción con ventana emergente de "Error: el operario ya existe en el sistema"	1, 3, 8	excepción con ventana emergente de "Error: el operario ya existe en el sistema"
Incorrecta	usuario: Joseee222233 password: Josesito124 nombreYApellido: Josesito	ventana emergente de "error: el nombre de usuario no puede superar los 10 caracteres" o un mensaje similar	2, 3, 8	Se agrega a la lista

Caso de uso #4: Modificación de operarios

Condición de entrada	Clases válidas	Clases inválidas
usuario	Cualquier string menor o igual a 10 caracteres **1**	- Cualquier string mayor a 10 caracteres **2**
contraseña	String entre 6 y 12 caracteres con al menos una mayúscula y un dígito **3**	String que no cumpla las condiciones impuestas en la sección de clases válidas **4**
nombreYApellido	Cualquier string **5**	-
estado	Cualquier estado que aparezca en el ComboBox: {"Activo", "Inactivo"} **6**	-

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba	Salidas obtenidas

			cubiertas	
Correcta	usuario: Josesito1 contraseña: Jose122 nombreYApellido: Jose Hernandez estado: Activo	Se modifica el operario	1, 3, 5, 6	Se modifica el operario
Incorrecta	usuario: Josesito1 contraseña: Jose122 nombreYApellido: Jose Hernandez estado: Activo	excepción con ventana emergente de "contraseña que no cumple con los requisitos"	1, 4, 5, 6	Al volver a intentar modificar el operario, se puede observar que la contraseña no queda guardada, pero en el momento el sistema no lo informa, por lo que el usuario efectivamente piensa que la contraseña ha sido modificada.
Incorrecta	usuario: Josesito1235 contraseña: Jose122 nombreYApellido: Jose Hernandez estado: Activo	ventana emergente de "error: el nombre de usuario no puede superar los 10 caracteres" o un mensaje similar	2, 3, 5, 6	Se modifica el operario

Caso de uso #5: Log-In de operarios

Condición de entrada	Clases válidas	Clases inválidas
usuario	Cualquier string **1**	-
password	Cualquier string **2**	-

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	usuario: Jose password: Josesito124	Que sólo se habilite la tab de "General". El operario solo puede	1, 2	Solo la tab "General" habilitada.
Incorrecta	usuario: Jose password: Josesito111	excepción con ventana emergente de "contraseña incorrecta"	1, 2	excepción con ventana emergente de "contraseña incorrecta"
Incorrecta	usuario: Joseee password:	excepción con ventana emergente de "Usuario no	1, 2	excepción con ventana emergente

	Josesito124	encontrado"		de "Usuario no encontrado"

Caso de uso #6: Agregar mesas al sistema

Condición de entrada	Clases válidas	Clases inválidas
Número de mesa	Número de mesa >= 0 **1**	-
Cantidad de comensales	Cantidad de comensales > 0 **2**	-

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	Número de mesa=0 Cantidad de comensales = 1	Se agrega la mesa a la lista de la ventana	1, 2	Se agrega la mesa a la lista de la ventana
Incorrecta	(teniendo en cuenta la lista de mesas vacía) Número de mesa=0 Cantidad de comensales = 2	Ventana emergente que muestre error, ya que la mesa 0 representa la barra y en la barra puede haber como máximo 1 persona según contrato	1, 2	Se agrega la mesa a la lista de la ventana
Incorrecta	(teniendo en cuenta la lista de mesas existente) Número de mesa=0 Cantidad de comensales = 1	Excepción con ventana emergente de "El número de mesa que quiere agregar ya se encuentra en el sistema"	1, 2	Excepción con ventana emergente de "El número de mesa que quiere agregar ya se encuentra en el sistema"

Caso de uso #7: Modificación de las mesas del sistema

Según la documentación, para las modificaciones no hay ningún tipo de precondición / excepción que se encargue de los valores erróneos que se pueden llegar a ingresar, por lo que aquí se reportarán varios errores al momento del testeo:

Condición de entrada	Clases válidas	Clases inválidas
Número de mesa	Número de mesa >= 0 **1**	Número de mesa < 0 **2**
Cantidad de comensales	Cantidad de comensales > 0 **3**	Cantidad de comensales <= 0 **4**

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	Número de mesa (correspondiente a una mesa existente) = 2 Cantidad de comensales = 2	Atributos de la mesa cambiados correctamente	1, 3	Atributos de la mesa cambiados correctamente
Incorrecta	Número de mesa (correspondiente a una mesa existente) = 2 Cantidad de comensales = -2	Ventana emergente indicando el error de valor inválido de comensales	1, 4	Atributos de la mesa cambiados correctamente
Incorrecta	Número de mesa (correspondiente a una mesa inexistente) = -2 Cantidad de comensales = 2	Ventana emergente indicando el error de valor inválido de número de mesa	2, 3	Atributos de la mesa cambiados correctamente
Incorrecta	Número de mesa (correspondiente a una mesa inexistente) = -1 Cantidad de comensales = -1	Ventana emergente indicando el error de valores inválidos	2, 4	Atributos de la mesa cambiados correctamente

Caso de uso #8: Agregar mozos al sistema y ver si el sueldo corresponde con la fórmula de la cantidad de hijos

Condición de entrada	Clases válidas	Clases inválidas
NombreYApellido	Cualquier string **1**	-
FechaDeNacimiento (yyyy-mm-dd)	Fecha de nacimiento que dé edad >18 **2**	- Fecha de nacimiento que dé edad < 18 **3**
Hijos	Hijos >= 0 **4**	Hijos < 0 **5**

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	NombreYApellido: Pablo Gomez FechaDeNacimiento: 2000–02-02 Hijos = 2	Se agrega el mozo a la lista	1, 2, 3	Se agrega el mozo a la lista

Incorrecta	NombreYApellido: Marcelo Sanchez FechaDeNacimiento: 2008-02-02 Hijos = 2	Excepción con ventana emergente de "el mozo no es mayor a 18 años"	1, 3, 4	Excepción con ventana emergente de "el mozo no es mayor a 18 años"
Incorrecta	NombreYApellido: Pepe Garcia FechaDeNacimiento : 2000-02-02 Hijos = -1	Excepción con ventana emergente de "Cantidad de hijos menor que 0"	1, 2, 5	Excepción con ventana emergente de "Cantidad de hijos menor que 0"
Incorrecta	NombreYApellido: Pablo Gomez FechaDeNacimiento : 2002-05-02 Hijos = 2	Excepción con ventana emergente de "Mozo ya existente"	1, 2, 4	Excepción con ventana emergente de "Mozo ya existente"

Incumplimiento del contrato / Inconsistencia del sistema: Nunca se le pide al usuario el sueldo básico, y al ingresar la cantidad de hijos nunca se informa, en ningún apartado, el sueldo total que cobraría cada mozo.

Caso de uso #9: Agregar productos al sistema

Condición de entrada	Clases válidas	Clases inválidas
Nombre	Cualquier string **1**	-
precioCosto	precioCosto > 0 **2**	precioCosto <= 0 **3**
precioVenta	precioVenta > 0 **4**	precioVenta <= 0 **5**
StockInicial	StockInicial > 0 **6**	Stock <= 0 **7**

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	Nombre: Milanesa precioCosto = 200 precioVenta = 600 StockInicial = 2	Se agrega el producto a la lista	1, 2, 4, 6	Se agrega el producto a la lista
Incorrecta	(suponiendo lista vacia) Nombre: Milanesa precioCosto = 0 precioVenta = 5 Stockinicial = 2	Ventana emergente con "El precioCosto no puede ser 0" (por contrato de consigna)	1, 3, 4, 6	Se agrega el producto a la lista (No hay excepción o preCond que verifique el contrato de la consigna)

Incorrecta	Nombre: Milanesa precioCosto = 0 precioVenta = 0 StockInicial = 3	Ventana emergente de "el precioCosto no puede ser 0" o "el precioVenta no puede ser 0" o ambas (por contrato de consigna)	1, 3, 5, 6	Se agrega el producto a la lista (No hay excepción o preCond que verifique el contrato de la consigna)
Incorrecta	Nombre: Milanesa precioCosto = 200 precioVenta = 300 StockInicial = 0	Ventana emergente de "Stock a por lo menos un producto"	1, 2, 4, 7	Ventana emergente de "Stock a por lo menos un producto"

Caso de uso #10: Eliminar productos del sistema

Escenario 1: El producto no está asociado a una comanda. El producto se elimina correctamente. Esto es correcto.
Escenario 2: El producto está asociado a una comanda. El producto NO se elimina, se muestra un mensaje de error explicando que el producto está asociado a una comanda activa. Esto es correcto.

Caso de uso #11: Agregar promociones de producto al sistema

Condición de entrada	Clases válidas	Clases inválidas
diasDePromo	{"Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"} **1**	-
Producto	Cualquier producto que se muestre en el ComboBox **2**	-
dosPorUno	dosPorUno = true **3** dosPorUno = false **4**	-
porCantidad	porCantidad = true **5** porCantidad = false **6**	
cantMinima	cantMinima >= 0 **7**	cantMinima < 0 **8**
precioUnitario	precioUnitario >= 0 **9**	precioUnitario < 0 **10** precioUnitario > precioVentaProducto **11**

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas

Correcta	diasDePromo: "Miércoles" Producto: Pepino dosPorUno = true porCantidad = false cantMinima = 0 (forzado por dosPorUno) precioUnitario = 0 (forzado por dosPorUno)	Se agrega la promoción a la lista	1, 2, 3, 6, 7, 9	Se agrega la promoción a la lista
Correcta	diasDePromo: "Miércoles" Producto: Pepino dosPorUno = true porCantidad = true cantMinima = 2 precioUnitario = 100	Se agrega la promoción a la lista	1, 2, 3, 5, 7, 9	Se agrega la promoción a la lista
Incorrecta	diasDePromo: "Miércoles" Producto: Pepino dosPorUno = false porCantidad = true cantMinima = 1 precioUnitario = -1	Excepción con ventana emergente "precio unitario negativo"	1, 2, 4, 5, 7, 10	Excepción con ventana emergente "precio unitario negativo"
Incorrecta	diasDePromo: "Miércoles" Producto: Pepino dosPorUno = false porCantidad = false cantMinima = 0 (forzado) precioUnitario = 0 (forzado)	Excepción con ventana emergente "No pueden ser false ambos tipos de descuento"	1, 2, 4, 6, 7, 9	Excepción con ventana emergente "No pueden ser false ambos tipos de descuento"
Incorrecta	diasDePromo: "Miércoles" Producto: Pepino dosPorUno = false porCantidad = true cantMinima = 1 precioUnitario = un precio mayor al de venta del pepino, por ejemplo, 2000	Excepción con ventana emergente "el precio unitario del producto en descuento no puede ser mayor al precio de venta del producto en sí"	1, 2, 4, 5, 7, 9, 11	Se agrega la promoción a la lista

Caso de uso #12: Modificación de promociones de producto

Según la documentación, los métodos que modifican las promociones de producto no tienen excepciones y tampoco por contrato se especifican precondiciones, por lo que cualquier valor de cantidad mínima y precio unitario ingresado se guardarán, independientemente de si sean valores positivos o negativos, por lo que esto es un error.

Caso de uso #13: Agregar promociones temporales al sistema

Condición de entrada	Clases válidas	Clases inválidas
diasDePromo	{"Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"} **1**	-
Nombre	Cualquier string **2**	-
porcDescuento	porcDescuento> 0 **3**	porcDescuento<= 0 **4** porcDescuento> 100 **5**
Acumulable	Acumulable = true **6** Acumulable = false **7**	-
metodoDePago	{"Efectivo", "Tarjeta", "CtaDNI", "MercPago"} **8**	-

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	diasDePromo: Martes Nombre: promoConModo porcDescuento = 15 Acumulable = true metodoDePago = CtaDNI	Se agrega la promoción a la lista	1, 2, 3, 5, 7, 8	Se agrega la promoción a la lista
Incorrecta	diasDePromo: Martes Nombre: promoConModo porcDescuento = -1 Acumulable = true metodoDePago = CtaDNI	Ventana emergente de error "el porcentaje de descuento debe ser mayor a 0"	1, 2, 4, 6, 8	Se agrega la promoción a la lista
Incorrecta	diasDePromo: Martes Nombre: promoConModo porcDescuento = 120 Acumulable = true metodoDePago = CtaDNI	Ventana emergente de error con "el porcentaje de descuento debe ser menor a 100"	1, 2, 5, 6, 8	Se agrega la promoción a la lista

Caso de uso #14: Modificación de promociones temporales

Según la documentación, los métodos que modifican las promociones temporales no tienen excepciones y tampoco por contrato se especifican precondiciones, por lo que cualquier valor de porcentaje de descuento se guardará, independientemente de si sea negativo o mayor o igual a 100, por lo que esto es un error.

Caso de uso #15: Ocupar una mesa + Crear la comanda (todo se hace a la vez)

Condición de entrada	Clases válidas	Clases inválidas
numMesa	numMesa >= 0 **1**	**- NO hay clases inválidas de numMesa ya que se elige desde una lista (incosistencia con el método ocupaMesa y la implementación)**
cantComensales	cantComensales >= 1 **2**	cantComensales < 1 **3**

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	numMesa = 0 cantComensales = 1	La mesa pasa a estar en estado "ocupada" y la comanda se crea	1, 2	La mesa pasa a estar en estado "ocupada" y la comanda se crea
Incorrecta	numMesa = 0 cantComensales = 0	ventana emergente de "ingrese una cantidad de comensales válida" o algo similar	1, 3	La mesa pasa a estar en estado "ocupada" y la comanda se crea

Escenarios particulares:

1_ Si la mesa está ocupada previamente y se ingresa una cantidad de comensales válida, se lanza la excepción MesaOcupadaException y se muestra en una ventana emergente, por lo que esto es **correcto.**
2_ Si no hay productos, siempre se lanza NoHayProductosException. Esto es **correcto.**
3_ Si la mesa no está asignada, siempre se lanza MesaNoAsignadaException. Esto es **correcto.**
4_ Si no hay mozos en el sistema, siempre se lanza NoHayMozosException. Esto es **correcto.**
5_ Si no hay al menos dos productos diferentes en promoción, siempre se lanza NoHayDosPromosException. El sistema comprueba que el día de las promociones coincidan con el día actual. Esto es **correcto.**

Caso de uso #16: Agregar productos a la comanda de la mesa

Condición de entrada	Clases válidas	Clases inválidas
cantidad	cantidad > 0 **1**	cantidad < 0 **2** cantidad = 0 **3**
Producto	El producto siempre es válido, es	

	elegido desde una lista, **4**	

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	cantidad = 1 Producto: pan	Se agrega el pedido a la comanda	1, 4	Se agrega el pedido a la comanda
Incorrecta	cantidad = 0 Producto: pan	Ventana emergente de pedir una cantidad válida o algo similar	1, 3	Se agrega el pedido a la comanda
Incorrecta	cantidad = -1 Producto: pan	Ventana emergente de pedir una cantidad válida o algo similar	1, 2	Se suma al stock del producto la cantidad ingresada (es decir, stockActual + 1 en este caso) (!!!)

Escenario particular: Si la cantidad del producto es 1, y en el pedido se agrega esa unidad restante, se tira una excepción de Stock negativo, cuando en realidad el stock pasa a 0. Esto es un error, es incorrecto.

Caso de uso #17: Facturar el total de la comanda

Escenario 1: Promoción de 2x1 en pan y promoción por cantidad de zapallo. La cantidad mínima del zapallo es 2 y el precio unitario es 150$ y cada unidad de pan vale 50$.

Condición de entrada	Clases válidas	Clases inválidas
Comanda	Comanda != null **1**	**-** ninguna (ingresada por lista)
formaDePago	{"Efectivo", "Tarjeta", "CtaDNI", "MercPago"} **2**	- ninguna (ingresada por un redButton)

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	Comanda con: cantidad = 2 de Producto pan cantidad = 1 de zapallo formaDePago: Efectivo	2x1 en pan = 50 + 150x2 del zapallo debería dar 350	1, 2	total = 150$
Correcta	Comanda con	2x1 en pan = 50 + pan	1, 2	total = 100

	cantidad = 3 de Producto pan formaDePago: MercPago	restante = 100		

Escenario 2: Promoción temporal no acumulable del 15% con efectivo.

Condición de entrada	Clases válidas	Clases inválidas
Comanda	Comanda != null **1**	**-** ninguna (ingresada por lista)
formaDePago	{"Efectivo", "Tarjeta", "CtaDNI", "MercPago"} **2**	- ninguna (ingresada por un redButton)

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	Comanda: cantidad = 1 de producto: pan cantidad = 1 de producto: zapallo formaDePago: Efectivo	zapallo (200) * 0,85 + pan (50) * 0,85 = 212,50	1, 2	total = 212,50
Correcta	Comanda: cantidad = 1 de producto: pan cantidad = 1 de producto: zapallo formaDePago: MercPago	zapallo (200) + pan (50) = 250	1, 2	total = 250

Escenario 3: Promoción temporal del 15% acumulable con efectivo. También una promoción de 2x1 en pan y una promoción por cantidad en zapallo con cantidad mínima = 2 y precio unitario = 150.

Condición de entrada	Clases válidas	Clases inválidas
Comanda	Comanda != null **1**	**-** ninguna (ingresada por lista)
formaDePago	{"Efectivo", "Tarjeta", "CtaDNI", "MercPago"} **2**	- ninguna (ingresada por un redButton)

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba	Salidas obtenidas

			cubiertas	
Correcta	Comanda con: cantidad = 2 de pan cantidad = 2 de zapallo formaDePago: Efectivo	(Pan (50) + Zapallo*2 (150)) * 0,85 = 297,50	1, 2	total = 127,50
Correcta	Comanda con: cantidad = 2 de pan cantidad = 2 de zapallo formaDePago: MercPago	Pan (50) + Zapallo*2 (150) = 350	1, 2	total = 150

La conclusión de las pruebas de las promociones es que hay un error en el cálculo de la promoción por **cantidad mínima**, estando el resto de las promociones bien facturadas

Escenario particular: Si en la comanda se ingresa un pedido con una cantidad negativa como se informó en el **Caso de uso #10**, el resultado es erróneo.

Caso de uso #18: Estadísticas de un empleado

Escenario: El mozo "Adolfo" hace una venta de 220$ en la mesa 1 y otra venta de 660$ más tarde en la mesa 2.

Condición de entrada	Clases válidas	Clases inválidas
nombre	Cualquier string **1**	- ninguna, el string no puede ser inválido porque se elige desde una lista

Batería de pruebas

Tipo de clase	Valores de entrada	Salida esperada	Clases de prueba cubiertas	Salidas obtenidas
Correcta	nombre: Adolfo	consumo total de 880$	1	consumo total de 880$

Caso de uso #19: Empleado con mayor volumen de venta

Escenario: El mozo "Adolfo" hace una venta de 220$ en la mesa 1 y otra venta de 660$ más tarde en la mesa 2, y el mozo "Pedro" hace una venta de 400$ en la mesa 1

Salida esperada: Adolfo es el mozo con mayor volumen de venta con 880$
Salida obtenida: Adolfo es el mozo con mayor volumen de venta con 880$

Caso de uso #20: Empleado con menor volumen de venta

Escenario: El mozo "Adolfo" hace una venta de 220$ en la mesa 1 y otra venta de 660$ más tarde en la mesa 2, y el mozo "Pedro" hace una venta de 400$ en la mesa 1

Salida esperada: Pedro es el mozo con menor volumen de venta con 400$
Salida obtenida: Pedro es el mozo con menor volumen de venta con 400$

Caso de uso #21: Consumo promedio por mesa

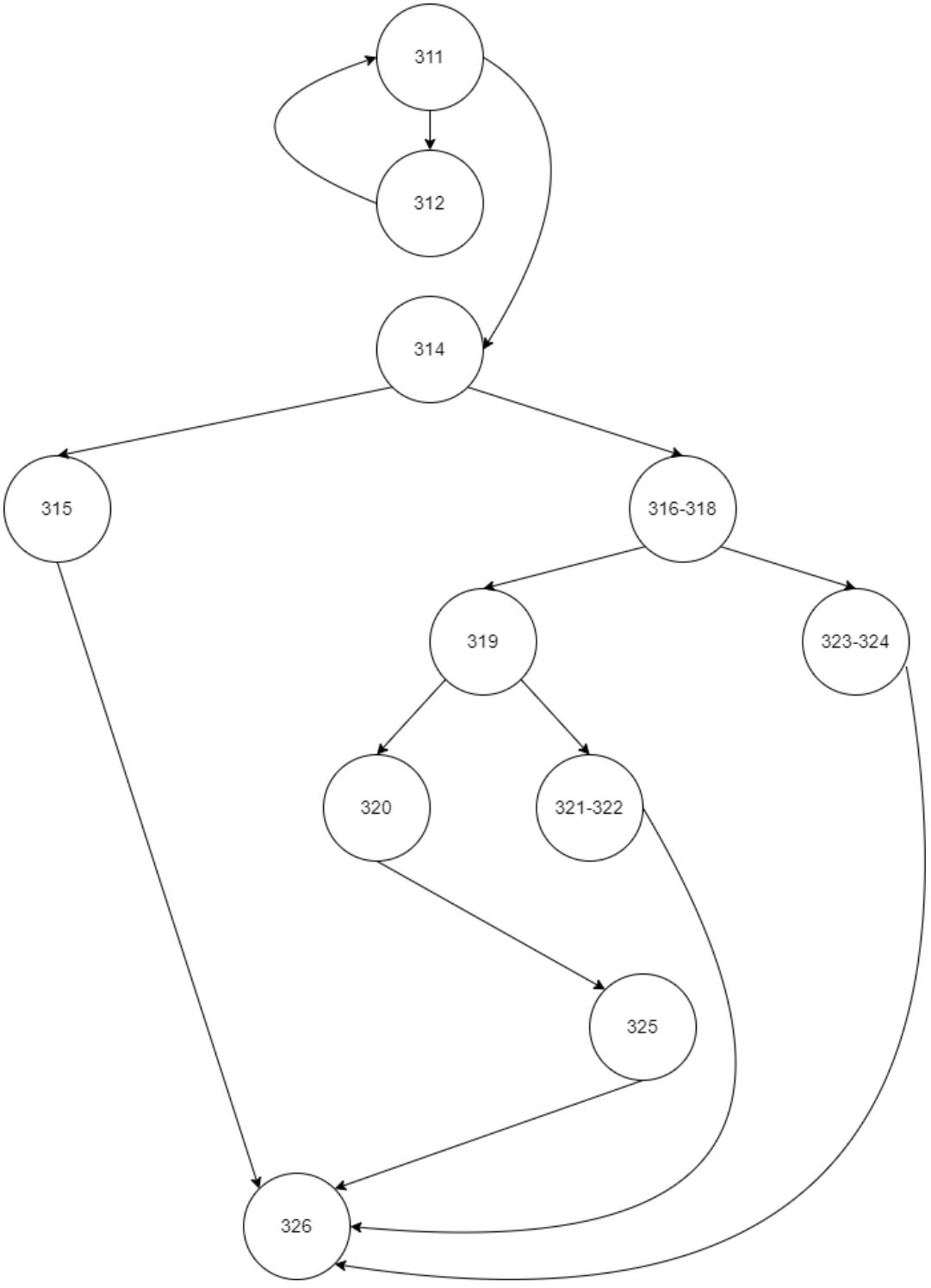
Escenario: El mozo "Adolfo" hace una venta de 220$ en la mesa 1 y otra venta de 660$ más tarde en la mesa 2, y el mozo "Pedro" hace una venta de 400$ en la mesa 0

Salida esperada: La mesa 0 tiene un consumo promedio de 400
La mesa 1 tiene un consumo promedio de 440
Salida obtenida: La mesa 0 tiene un consumo promedio de 400
La mesa 1 tiene un consumo promedio de 440

Caja Blanca

Método ocupaMesa(int numero, int cantComensales) de la clase Sistema.

Grafo ciclomático:



Complejidad:
V(G)= nodos decisión +1= 5
V(G)= regiones= 5
V(G)= arcos - nodos + 2= 5

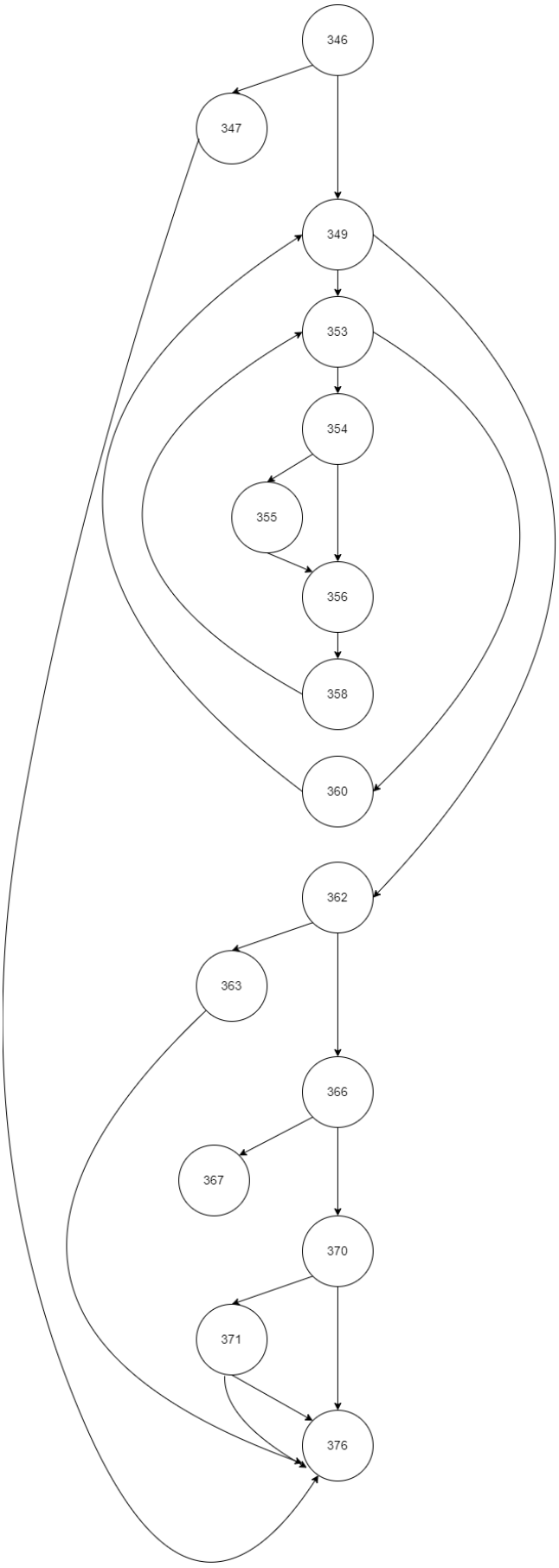
Caminos por método simplificado:

1) 311 - 314 - 315 - 326 (NO HAY MESAS)
2) 311 - 312 - 311 - 314 - 315 - 326 (NO HAY MESAS)
3) 311 - 314 - (316-318) - (323-324) - 326 (MESA OCUPADA)
4) 311 - 314 - (316-318) - 319 - 321-322 - 326 (COMENSALES INSUFICIENTES)
5) 311 - 314 - 316-318 - 319 - 320 - 325 - 326 (SE EJECUTA EL MÉTODO CUMPLIENDO SU PROPÓSITO)

CAMINO	ENTRADA (int numero, int cantComensales)	RESULTADO ESPERADO
1	{número de mesa inexistente, cantComensales es indiferente su valor}, el array de mesas puede estar vacío o no e igual se ejecutará de esta manera.	MesaNoExistenteException()
2	{número de mesa inexistente, cantComensales es indiferente su valor}, el array de mesas no puede estar vacío y no debe encontrar una mesa con el número pasado por parametro..	MesaNoExistenteException()
3	{número de mesa existente, cantComensales es indiferente su valor} array de mesas no vacío, con la mesa indicada por número de mesa (parámetro)	MesaOcupadaException()
4	{número de mesa existente, cantComensales mayor a la cantidad de comensales permitidos por la mesa}array de mesas no vacío, con la mesa indicada por número de mesa (parámetro)	ComensalesInsuficientesExceptio n()
5	{número de mesa existente, cantComensales menor o igual a la cantidad de comensales permitidos por la mesa} array de mesas no vacío, con la mesa indicada por número de mesa (parámetro)	this.comandas.add(creaComanda(mesa)) , se cumple el propósito del método.

Método creaComanda(Mesa mesa) de la clase Sistema.

Grafo ciclomático:


346
347
349
353
354
355
356
358
360
362
363
366
367
370
371
376

Complejidad:
V(G)= nodos decisión +1= 8
V(G)= regiones= 8
V(G)= arcos - nodos + 2= 8

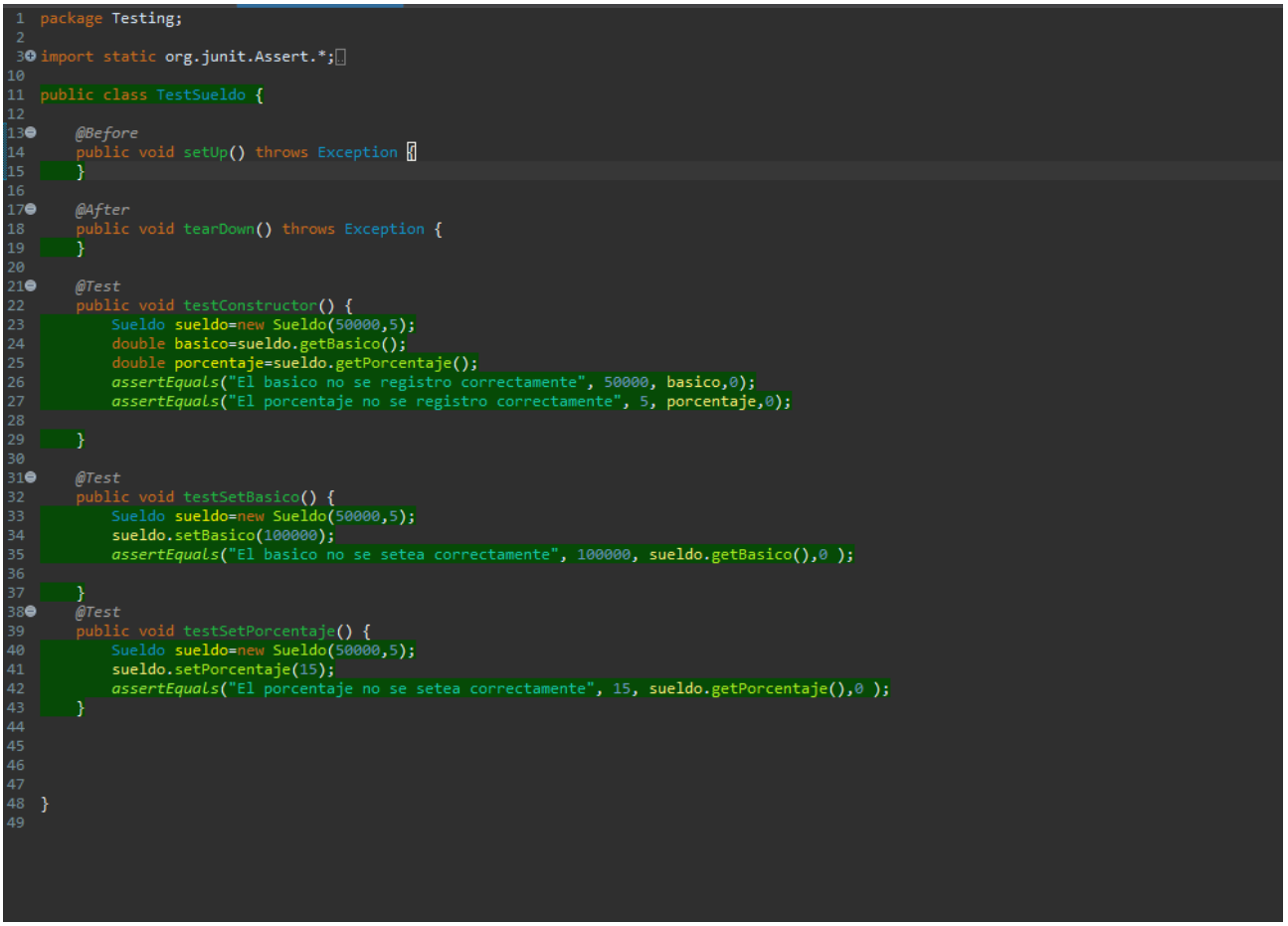
Caminos por método simplificado

1) 346 - 347 - 376 (NO HAY PRODUCTOS)
2) 346 - 349 - 362 - 363 - 376 (NO HAY MOZOS)
3) 346 - 349 - 353 - 354 - 356 - 358 - 360 - 349 - 362 - 366 - 367 (NO ESTÁ LA MESA ASIGNADA)
4) 346 - 349 - 353 - 354 - 355 - 356 - 353 - 358 360 - 349 - 362 - 366 - 370 - 371 - 376 (NO HAY DOS O MÁS PROMOS)
5) 346 - 349 - 353 - 354 - 355 - 356 - 353 - 358 360 - 349 - 362 - 366 - 370 - 376 (SE CREA LA COMANDA Y SE SETEA EL ESTADO DE MESA)

CAMINO	ENTRADA (Mesa mesa)	RESULTADO ESPERADO
1)	{mesa != null} Arreglo de productos vacío en sistema, puede haber o no mozos en sistema.	NoHayProductosException()
2	{mesa != null} Arreglo de productos en sistema con un producto o más, pero el arreglo de mozos está vacío.	NoHayMozosException()
3	{mesa != null} Arreglo de productos en sistema con 1 producto o más, arreglo de mozos en sistema con 1 mozo o más, y el mozo no tiene la mesa asignada.	MesaNoAsignadaException()
4	{mesa != null} Arreglo de productos en sistema con 1 producto o más, arreglo de mozos en sistema con 1 mozo o más, y el mozo tiene la mesa asignada. NO hay al menos dos productos diferentes en promoción.	NoHayDosPromosException()
5	{mesa != null} {mesa != null} Arreglo de productos en sistema con 1 producto o más, arreglo de mozos en sistema con 1 mozo o más, y el mozo tiene la mesa asignada. SÍ hay al menos dos productos diferentes en promoción.	Se setea la mesa como ocupada y se crea la comanda.

Pruebas unitarias

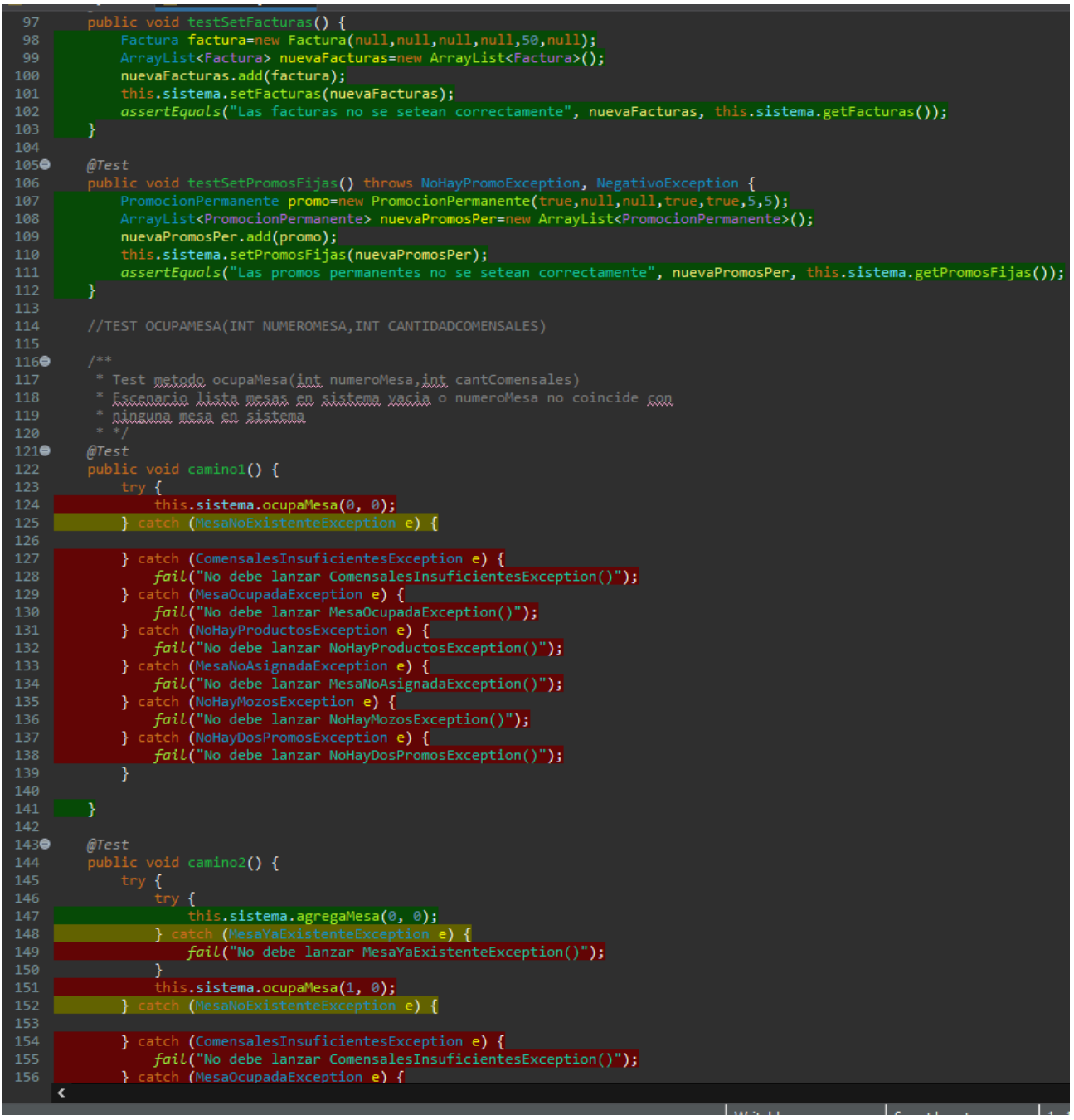
Clase Sueldo: Constructor, setSueldoBasico, setPorcentaje



```java
package Testing;

import static org.junit.Assert.*;

public class TestSueldo {

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testConstructor() {
        Sueldo sueldo=new Sueldo(50000,5);
        double basico=sueldo.getBasico();
        double porcentaje=sueldo.getPorcentaje();
        assertEquals("El basico no se registro correctamente", 50000, basico,0);
        assertEquals("El porcentaje no se registro correctamente", 5, porcentaje,0);

    }

    @Test
    public void testSetBasico() {
        Sueldo sueldo=new Sueldo(50000,5);
        sueldo.setBasico(100000);
        assertEquals("El basico no se setea correctamente", 100000, sueldo.getBasico(),0 );

    }
    @Test
    public void testSetPorcentaje() {
        Sueldo sueldo=new Sueldo(50000,5);
        sueldo.setPorcentaje(15);
        assertEquals("El porcentaje no se setea correctamente", 15, sueldo.getPorcentaje(),0 );
    }



}
```

Clase Sistema: Log-In, setFacturas, setPromosFijas, y método ocupaMesa(int NumMesa, int CantComensales) con sus 5 caminos posibles.

```
39  public class TestSistema {
40      Sistema sistema;
41
42      @Before
43      public void setUp() throws Exception {
44          this.sistema= Sistema.getInstancia();
45          System.out.println(this.sistema.getMesas().size());
46
47      }
48
49      @After
50      public void tearDown() throws Exception {
51          this.sistema.setMesas(new ArrayList<Mesa>());
52      }
53
54
55      @Test
56      public void testLoginAdminExitoso() {
57          Operario admin=null;
58          try {
59              admin=this.sistema.loginAdmin("ADMIN", "ADMIN1234");
60          } catch (ContraseniaIncorrectaException  e) {
61              fail("No deberia lanzar contraseniaIncorectaException()" );
62          }catch (UsuarioIncorrectoException e) {
63              fail("No deberia lanzar UsuarioIncorrectoException()" );
64          }
65          assertEquals("No se logeo correctamente", sistema.getAdmin(), admin);
66      }
67      @Test
68      public void testLoginAdminContraseniaIncorrecta() {
69          Operario admin=null;
70          try {
71              admin=this.sistema.loginAdmin("ADMIN", "ADMIN124");
72          } catch (ContraseniaIncorrectaException  e) {
73
74          }catch (UsuarioIncorrectoException e) {
75              fail("No deberia lanzar UsuarioIncorrectoException()" );
76          }
77
78          assertNotEquals("No deberia legearse correctamente", sistema.getAdmin(), admin);
79
80      }
81
82      @Test
83      public void testLoginAdminUsuarioIncorrecto() {
84          Operario admin=null;
85          try {
86              admin=this.sistema.loginAdmin("ADMINN", "ADMIN1234");
87          } catch (ContraseniaIncorrectaException  e) {
88              fail("No deberia lanzar ContraseniaIncorrectaException()" );
89          }catch (UsuarioIncorrectoException e) {
90
91          }
92
93          assertNotEquals("No deberia legearse correctamente", sistema.getAdmin(), admin);
94          }
95
96      @Test
97      public void testSetFacturas() {
98          Factura factura=new Factura(null,null,null,null,50,null);
```



```java
    public void testSetFacturas() {
        Factura factura=new Factura(null,null,null,null,50,null);
        ArrayList<Factura> nuevaFacturas=new ArrayList<Factura>();
        nuevaFacturas.add(factura);
        this.sistema.setFacturas(nuevaFacturas);
        assertEquals("Las facturas no se setean correctamente", nuevaFacturas, this.sistema.getFacturas());
    }

    @Test
    public void testSetPromosFijas() throws NoHayPromoException, NegativoException {
        PromocionPermanente promo=new PromocionPermanente(true,null,null,true,true,5,5);
        ArrayList<PromocionPermanente> nuevaPromosPer=new ArrayList<PromocionPermanente>();
        nuevaPromosPer.add(promo);
        this.sistema.setPromosFijas(nuevaPromosPer);
        assertEquals("Las promos permanentes no se setean correctamente", nuevaPromosPer, this.sistema.getPromosFijas());
    }

    //TEST OCUPAMESA(INT NUMEROMESA,INT CANTIDADCOMENSALES)

    /**
     * Test metodo ocupaMesa(int numeroMesa,int cantComensales)
     * Escenario lista mesas en sistema vacia o numeroMesa no coincide con
     * ninguna mesa en sistema
     * */
    @Test
    public void camino1() {
        try {
            this.sistema.ocupaMesa(0, 0);
        } catch (MesaNoExistenteException e) {

        } catch (ComensalesInsuficientesException e) {
            fail("No debe lanzar ComensalesInsuficientesException()");
        } catch (MesaOcupadaException e) {
            fail("No debe lanzar MesaOcupadaException()");
        } catch (NoHayProductosException e) {
            fail("No debe lanzar NoHayProductosException()");
        } catch (MesaNoAsignadaException e) {
            fail("No debe lanzar MesaNoAsignadaException()");
        } catch (NoHayMozosException e) {
            fail("No debe lanzar NoHayMozosException()");
        } catch (NoHayDosPromosException e) {
            fail("No debe lanzar NoHayDosPromosException()");
        }

    }

    @Test
    public void camino2() {
        try {
            try {
                this.sistema.agregaMesa(0, 0);
            } catch (MesaYaExistenteException e) {
                fail("No debe lanzar MesaYaExistenteException()");
            }
            this.sistema.ocupaMesa(1, 0);
        } catch (MesaNoExistenteException e) {

        } catch (ComensalesInsuficientesException e) {
            fail("No debe lanzar ComensalesInsuficientesException()");
        } catch (MesaOcupadaException e) {
```

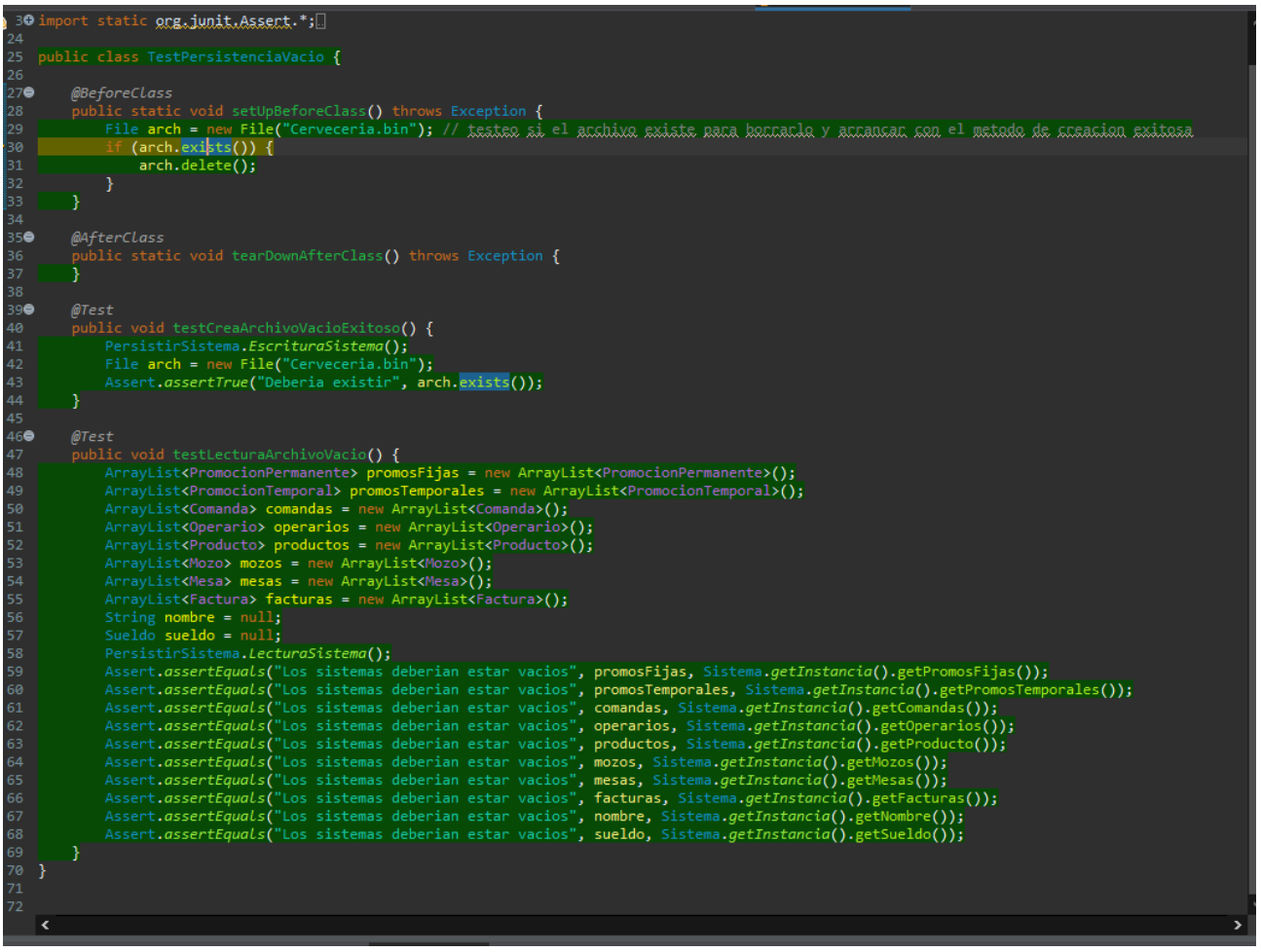
```
155             fail("No debe lanzar ComensalesInsuficientesException()");
156         } catch (MesaOcupadaException e) {
157             fail("No debe lanzar MesaOcupadaException()");
158         } catch (NoHayProductosException e) {
159             fail("No debe lanzar NoHayProductosException()");
160         } catch (MesaNoAsignadaException e) {
161             fail("No debe lanzar MesaNoAsignadaException()");
162         } catch (NoHayMozosException e) {
163             fail("No debe lanzar NoHayMozosException()");
164         } catch (NoHayDosPromosException e) {
165             fail("No debe lanzar NoHayDosPromosException()");
166         }
167 
168     }
169 
170     /**
171      * Test metodo ocupaMesa(int numeroMesa,int cantComensales)
172      * Escenario lista mesas en sistema con elementos, y parametro numeroMesa igual a
173      * el numero de mesa de las del sistema y las mesas con estado distinto a libre.
174      */
175     @Test
176     public void camino3() {
177         try {
178             this.sistema.agregaMesa(0, 0);
179             this.sistema.getMesas().get(0).setEstado("");
180             try {
181                 this.sistema.ocupaMesa(0, 0);
182             } catch (MesaNoExistenteException e) {
183                 fail("No debe lanzar MesaNoExistenteException" );
184             } catch (ComensalesInsuficientesException e) {
185                 fail("No debe lanzar ComensalesInsuficientesException" );
186             } catch (MesaOcupadaException e) {
187 
188             } catch (NoHayProductosException e) {
189                 fail("No debe lanzar NoHayProductosException" );
190             } catch (MesaNoAsignadaException e) {
191                 fail("No debe lanzar MesaNoAsignadaException" );
192             } catch (NoHayMozosException e) {
193                 fail("No debe lanzar NoHayMozosException" );
194             } catch (NoHayDosPromosException e) {
195                 fail("No debe lanzar NoHayDosPromosException" );
196             }
197         } catch (MesaYaExistenteException e) {
198             fail("No debe lanzar MesaYaExistenteException" );
199         }
200     }
201 
202 
203     /**
204      * Test metodo ocupaMesa(int numeroMesa,int cantComensales)
205      * Escenario lista mesas en sistema con elementos, y parametro numeroMesa igual a
206      * el numero de mesa de una del sistema y estado libre.
207      * */
208     @Test
209     public void camino4() {
210         try {
211             this.sistema.agregaMesa(0, 0);
212             try {
213                 this.sistema.ocupaMesa(0, 5);
214             } catch (MesaNoExistenteException e) {
```

```
            fail("No debe lanzar MesaNoExistenteException()");
        } catch (ComensalesInsuficientesException e) {

        } catch (MesaOcupadaException e) {
            fail("No debe lanzar MesaOcupadaException()");
        } catch (NoHayProductosException e) {
            fail("No debe lanzar NoHayProductosException()");
        } catch (MesaNoAsignadaException e) {
            fail("No debe lanzar MesaNoAsignadaException()");
        } catch (NoHayMozosException e) {
            fail("No debe lanzar NoHayMozosException()");
        } catch (NoHayDosPromosException e) {
            fail("No debe lanzar NoHayDosPromosException()");
        }
    } catch (MesaYaExistenteException e) {
        fail("No debe lanzar MesaYaExistenteException" );
    }

}

/**
 * Test metodo ocupaMesa(int numeroMesa,int cantComensales)
 * Escenario lista mesas en sistema con elementos, y parametro numeroMesa igual a
 * el numero de mesa de una del sistema con estado libre, con mozo en sistema, con un mozo
 * ue tenga asignada la mesa y ue haya dos promos activas en sistema
 */

@Test
public void camino5() {
    try {
        this.sistema.agregaMesa(0, 2);
        //this.sistema.agregaMesa(1, 3);
        try {
            try {
                this.sistema.agregaProductos(new Producto("hola",50,100,20));
                this.sistema.agregaProductos(new Producto("chau",50,100,20));
                try {
                    this.sistema.agregaMozo(new Mozo("Activo",0,"nombre", LocalDate.of(1998,04,04)));
                    this.sistema.getMozos().get(0).agregarMesa(this.sistema.getMesas().get(0));
                    try {
                        this.sistema.getPromosFijas().add(new PromocionPermanente(true,null,this.sistema.getProducto().get(0),false,true,5,5));
                        this.sistema.getPromosFijas().add(new PromocionPermanente(true,null,this.sistema.getProducto().get(1),true,false,5,5));
                    } catch (NoHayPromoException e1) {
                        fail("No debe lanzar NoHayPromoException");
                    }

                    try {
                        this.sistema.ocupaMesa(0, 2);
                    } catch (MesaNoExistenteException e) {
                        fail("No debe lanzar MesaNoExistenteException");
                    } catch (ComensalesInsuficientesException e) {
                        fail("No debe lanzar ComensalesInsuficientesException");
                    } catch (MesaOcupadaException e) {
                        fail("No debe lanzar MesaOcupadaException");
                    } catch (NoHayProductosException e) {
                        fail("No debe lanzar NoHayProductosException");
                    } catch (MesaNoAsignadaException e) {
                        fail("No debe lanzar MesaNoAsignadaException");
                    } catch (NoHayMozosException e) {
                        fail("No debe lanzar NoHayMozosException");
```

```
                    } catch (NoHayMozosException e) {
                        fail("No debe lanzar NoHayMozosException");
                    } catch (NoHayDosPromosException e) {
                        fail("No debe lanzar NoHayDosPromosException");
                    }
                } catch (MozoDuplicadoException e) {
                    fail("No debe lanzar MozoDuplicadoException");
                } catch (HijosNegativosException e) {
                    fail("No debe lanzar HijosNegativosException");
                } catch (MenorDeDieciochoException e) {
                    fail("No debe lanzar MenorDeDieciochoException");
                }
            } catch (NegativoException e) {
                fail("No debe lanzar NegativoException" );
            } catch (PrecioVentaMenorCostoException e) {
                fail("No debe lanzar PrecioVentaMenorCostoException" );
            }
        } catch (ProductoDuplicadoException e1) {
            fail("No debe lanzar ProductoDuplicadoException" );
        }

    } catch (MesaYaExistenteException e) {
        fail("No debe lanzar MesaYaExistenteException" );
    }

}
```

Test de persistencia

Prueba de creación del archivo verificación en caso de que se cree sin datos



```java
import static org.junit.Assert.*;

public class TestPersistenciaVacio {

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
        File arch = new File("Cerveceria.bin"); // testeo si el archivo existe para borrarlo y arrancar con el metodo de creacion exitosa
        if (arch.exists()) {
            arch.delete();
        }
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Test
    public void testCreaArchivoVacioExitoso() {
        PersistirSistema.EscrituraSistema();
        File arch = new File("Cerveceria.bin");
        Assert.assertTrue("Deberia existir", arch.exists());
    }

    @Test
    public void testLecturaArchivoVacio() {
        ArrayList<PromocionPermanente> promosFijas = new ArrayList<PromocionPermanente>();
        ArrayList<PromocionTemporal> promosTemporales = new ArrayList<PromocionTemporal>();
        ArrayList<Comanda> comandas = new ArrayList<Comanda>();
        ArrayList<Operario> operarios = new ArrayList<Operario>();
        ArrayList<Producto> productos = new ArrayList<Producto>();
        ArrayList<Mozo> mozos = new ArrayList<Mozo>();
        ArrayList<Mesa> mesas = new ArrayList<Mesa>();
        ArrayList<Factura> facturas = new ArrayList<Factura>();
        String nombre = null;
        Sueldo sueldo = null;
        PersistirSistema.LecturaSistema();
        Assert.assertEquals("Los sistemas deberian estar vacios", promosFijas, Sistema.getInstancia().getPromosFijas());
        Assert.assertEquals("Los sistemas deberian estar vacios", promosTemporales, Sistema.getInstancia().getPromosTemporales());
        Assert.assertEquals("Los sistemas deberian estar vacios", comandas, Sistema.getInstancia().getComandas());
        Assert.assertEquals("Los sistemas deberian estar vacios", operarios, Sistema.getInstancia().getOperarios());
        Assert.assertEquals("Los sistemas deberian estar vacios", productos, Sistema.getInstancia().getProducto());
        Assert.assertEquals("Los sistemas deberian estar vacios", mozos, Sistema.getInstancia().getMozos());
        Assert.assertEquals("Los sistemas deberian estar vacios", mesas, Sistema.getInstancia().getMesas());
        Assert.assertEquals("Los sistemas deberian estar vacios", facturas, Sistema.getInstancia().getFacturas());
        Assert.assertEquals("Los sistemas deberian estar vacios", nombre, Sistema.getInstancia().getNombre());
        Assert.assertEquals("Los sistemas deberian estar vacios", sueldo, Sistema.getInstancia().getSueldo());
    }
}
```

Prueba de persistencia correcta con datos

```java
package testArchivos;

import static org.junit.Assert.*;

public class TestPersistenciaConDatos {

    @Before
    public void setUp() throws Exception {
        Sistema sist = Sistema.getInstancia();
        Producto producto = new Producto("Hamburguesa", 100, 200, 5);
        sist.agregaProductos(producto);
        PersistirSistema.EscrituraSistema();
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void testEscrituraLecturaConDatos() throws NegativoException, PrecioVentaMenorCostoException, ProductoDuplicadoException {
        Producto producto = new Producto("Panceta", 10, 30, 6);
        Sistema.getInstancia().agregaProductos(producto); // propago las excepciones porque estoy creando bien
        PersistirSistema.LecturaSistema();
        Assert.assertNotEquals("Deberian ser diferentes", producto, Sistema.getInstancia().getProducto());
    }

}
```

Prueba de GUI

Test de disabled / enabled

En esta sección se va a testear

Escenarios sin empleado logueado:

Escenario 1: Para el botón de log-in y log-out, no ingresamos ningún dato en los campos nombre de usuario y contraseña. Ambos botones permanecen disabled.

Escenario 2: Para el botón de log-in y log-out, ingresamos datos sólo en el campo nombre de usuario.Ambos botones permanecen disabled.

Escenario 3: Para el botón de log-in y log-out, ingresamos datos sólo en el campo contraseña. Ambos botones permanecen disabled.

Escenario 4: Para el botón de log-in y log-out, ingresamos datos en los campos nombre usuario y contraseña. El botón log-out permanece disabled y el botón log-in pasa a enabled.

Todos los escenarios fueron testeados de forma correcta, por lo que no se detectaron errores.

Escenarios con un empleado logueado:

Escenario 5: Para el botón de log-out. El botón log-out pasa a enabled.

Independientemente de si hay un empleado logueado o no:

Escenario 6: Para el botón de nuevo sistema, no ingresamos datos en el campo nombre del sistema. El botón nuevo sistema permanece disabled.

Escenario 7: Para el botón de nuevo sistema, ingresamos datos en el campo nombre del sistema. El botón nuevo sistema pasa a enabled.

Todos los escenarios fueron testeados de forma correcta, por lo que no se detectaron errores.

Test con datos

Se testeó la ventana principal, donde se puede guardar el sistema o recuperar los datos, y posteriormente loguearse tanto como admin como operario.

Escenario 1: Se ingresa un usuario correcto y una contraseña correcta. Se comprueba que el usuario logueado es el esperado.

Escenario 2: Se ingresa un usuario correcto y una contraseña incorrecta. Se lanza una ventana emergente de contraseña incorrecta.

Escenario 3: Se ingresa un usuario incorrecto y una contraseña incorrecta. Se lanza una ventana emergente de nombre de usuario incorrecto.

Para estos test tuvimos la dificultad de las ventanas emergentes JOptionPane ya que de la forma que estaba realizado el código no pudimos recuperar una referencia a las mismas como para poder comparar el mensaje obtenido con el esperado.Por esto en los últimos dos test, no hay ningún mensaje de assert, comprobamos su funcionamiento viendo ue mensaje mostraba la emergente.

Todos los escenarios fueron testeados de forma correcta, por lo que no se detectaron errores.