Trabajo Práctico Especial

Taller de Programación 1

Grupo 3

Testing realizado por
Teo Ramos Kees
Camila Belen Cacace.

Otros compañeros del grupo
Renzo Agustin Romeo
Lucas de Lellis

Github del subgrupo -> Branch: grupo_a_test
https://github.com/RenzoRomeo/taller-de-programacion-1/tree/grupo_a_test
Video: https://youtu.be/ktGHfLmAybY

Introducción

El objetivo de este trabajo es detallar el proceso de testeo de un sistema que gestiona todos los procesos pertinentes a un local gastronómico. Para ello exploramos distintos tipos de test.

- Test de Caja Negra: Se basa en el suministro de entradas y el análisis del comportamiento de la salida desconociendo la estructura del código.
- Test de Caja Blanca: En este caso se necesita del código fuente
- Test de Persistencia: Compara la información que se persiste en memoria secundaria con la que luego se carga en memoria al despersistir,
- Test de GUI: Buscan validar el comportamiento de la interfaz. Es decir las pruebas de GUI se orientan a detectar errores propios de la interfaz gráfica.
- Test de Integración: Analiza la interacción entre distintos módulos y ve cómo se integran para llevar a cabo procesos más complejos.

Algunos de estos tests no pudieron ser realizados.

Caja Negra

Los tests caja negra se definen como una técnica de análisis de la funcionalidad de un sistema que no tiene en cuenta la estructura interna del código. Es decir, no se conoce información del código. Los casos de prueba se basan sólo en el comportamiento de entrada/salida, por lo que el testeador simplemente se limita a suministrar datos.

Para ello se necesita contar con la especificación de requerimientos de cada módulo a testear, que fue provista por el subgrupo al cual le testeamos el código.

En nuestro caso, para realizar los tests de caja negra tomamos los métodos que consideramos más significativos para el programa, es decir aquellos que tenían que ver con la clase sistema y su uso. Con esto pudimos testear todas las excepciones que se encontraban en el trabajo para controlar su buen uso e implementación.

La tabla de particiones y baterías de prueba planteadas para los testing con sus métodos correspondientes y sus distintos escenarios fueron las siguientes:

Clase Sistema

metodo: agregarMesa(Mesa mesa)

Tabla de particiones

Condición	Clases Correctas	Clases Erróneas
Condición del parámetro nuevo	Mesa con datos válidos	Mesa con datos válidos
Atributos del parámetro nuevo	Nro mesa	Nro mesa
	Capacidad	Capacidad
Estado de la colección	Colección sin esa mesa (1)	Colección con esa mesa (2)

Modo del sistema	Usuario autorizado	Usuario autorizado
-	-	-
Condición del parámetro nuevo		Mesa con datos válidos
Atributos del parámetro nuevo		Nro mesa
		Capacidad
Estado de la colección		Colección con esa mesa
Modo del sistema		Usuario no autorizado (3)

Escenario 1: Colección de mesas con la mesa nro 1 y sin una mesa nro 0.

Escenario 2: Usuario en modo operario.

Batería de pruebas

	Entradas	**Salidas**		**Clases Cubiertas**
		Esperada	**Obtenida**	
Clases correctas	{0,3}	Mesa se agrego correctamente	PASS	1
Clases incorrectas	{1,4}	No se agrego en la coleccion porque ya existe (MesaRepetidaException)	PASS	2
	{1,4}	No se agrego en la coleccion porque	PASS	3

		el modo del sistema no es correcto (OperacionNoAutorizadaException)		

metodo agregarMozo(Mozo mozo)

Tabla de particiones

Condición	Clases Correctas	Clases Erróneas
Condición del parámetro nuevo	Mozo con datos validos	Mozo con datos validos
Atributos del parámetro nuevo	nombre	nombre
	apellido	apellido
	fecha de nacimiento	fecha de nacimiento
	hijos a cargo	hijos a cargo
	sueldo	sueldo
Estado de la colección	Coleccion sin ese mozo (1)	Coleccion con ese mozo (2)
Cantidad de mozos en sistema	Menor a la maxima	Menor a la maxima
Modo del sistema	Usuario autorizado	Usuario autorizado

-	-	-
Condición del parámetro nuevo		Mozo con datos validos
Atributos del parámetro nuevo		nombre
		apellido
		fecha de nacimiento
		hijos a cargo
		sueldo
Estado de la colección		Coleccion sin ese mozo
Cantidad de mozos en sistema		Menor a la maxima
Modo del sistema		Usuario no autorizado (3)
-	-	-
Condición del parámetro nuevo		Mozo con datos validos
Atributos del parámetro nuevo		nombre
		apellido
		fecha de nacimiento
		hijos a cargo
		sueldo
Estado de la colección		Coleccion sin ese mozo
Cantidad de mozos en sistema		Mayor a la maxima (4)
Modo del sistema		Usuario autorizado

Escenario 1: Colección de mozos con un mozo “Martin Perez” pero sin “Juan Perez”

Escenario 2: Usuario en modo operario.

Escenario 3: Colección de mozos con 6 mozos cargados.

Bateria de pruebas

	Entradas	Salidas		Clases Cubiertas
		Esperadas	Obtenidas	
Clases correctas	{"Juan","Perez","13/10/98",2,sueldo"}	Mozo se agrego correctamente	PASS	1
Clases incorrectas	{"Martin","Perez","13/10/98",4,sueldo"}	No se agrego a la coleccion porque ya existe (MozoExistenteException)	PASS	2
	{"Juan","Perez","13/10/98",4,sueldo"}	No se agrego en la coleccion porque el modo del sistema no es correcto (OperacionNoAutorizadaException)	PASS	3
	{"Juan","Perez","13/10/98",9,sueldo"}	No se agrega a la coleccion porque se supero la cantidad maxima de mozos	PASS	4

		(MaximaCantidad MozozException)		

metodo: agregarOperario(Operario operario)

Tabla de particiones

Condición	Clases Correctas	Clases Erróneas
Condicion del parametro nuevo	Operario con datos validos	Operario con datos validos
Atributos del parametro nuevo	nombre	nombre
	apellido	apellido
	nombreUsuario	nombreUsuario
	contrasenia	contrasenia
Estado de la colección	Coleccion sin ese operario (1)	Coleccion con ese operario (2)
Modo del sistema	Usuario autorizado	Usuario autorizado
-	-	-
Condicion del parametro nuevo		Operario con datos validos
Atributos del parametro nuevo		nombre

		apellido
		nombreUsuario
		contrasenia
Estado de la colección		Coleccion sin ese operario (3)
Modo del sistema		Usuario no autorizado

Escenario 1:Colección sin operario “Teo Ramos”

Escenario 2: Colección con operario “Teo Ramos” y en modo operario.

Bateria de pruebas

	Entradas	**Salidas**		**Clases Cubiertas**
		Esperada	**Obtenida**	
Clases correctas	{Teo,Ramos,teo1,Teo1234}	Operario se agrego correctamente	PASS	1
Clases incorrectas	{Teo,Ramos,teoramites,Teo1234}	No se agrego en la coleccion porque ya existe (OperarioExistenteException)	PASS	2
	{Teo,Ramos,teo2,Teo1234}	No se agrego en la coleccion porque	PASS	3

		el modo del sistema no es correcto (OperacionNoAutorizadaException)		

metodo: agregarPedido(Mesa mesa, Pedido pedido)

Tabla de particiones

Condición	Clases Correctas	Clases Erróneas
Condición del parámetro nuevo	Mesa y Pedido con parametros validos	Mesa y Pedido con parametros validos
Atributos del parámetro nuevo	nro Mesa	nro Mesa
	Capacidad	Capacidad
	Producto	Producto
	Cantidad	Cantidad
	Fecha	Fecha
Stock	Suficiente	Suficiente

Composicion de la mesa	Mesa con comanda (1)	Mesa sin comanda (2)
-	-	-
Condición del parámetro nuevo		Mesa y Pedido con parametros validos
Atributos del parámetro nuevo		nro Mesa
		Capacidad
		Producto
		Cantidad
		Fecha
Stock		Insuficiente (3)
Composición de la mesa		Mesa con comanda

Escenario 1: Colección de mesas con mesa nro 0 con comanda y mesa nro 1 sin comanda

Escenario 2: Colección de mesas con mesa nro 0 con comanda pero el stock del producto a agregar no tiene suficiente stock.

Batería de pruebas

	Entradas	**Salidas**		**Escenario**	**Clases Cubiertas**
		Esperadas	**Obtenidas**		

Clases correctas	({0,1},{producto ,5,15/11/22})	Se agrega el pedido correctamente	PASS	1	1
Clases incorrectas	({1,4},{producto ,5,15/11/22})	No se agrega a la mesa porque no tiene una comanda (ComandaInexistenteExceptio n)	FAIL	1	2
	({0,1},{producto ,5,15/11/22})	No se agrega a la mesa porque el stock es	FAIL	2	3

metodo: agregarProducto(Producto producto)

Tabla de particiones

Condición	Clases Correctas	Clases Erróneas
Condición del parámetro nuevo	Producto con parametros validos	Producto con parametros validos
Atributos del parámetro nuevo	nombre	nombre

	precio costo	precio costo
	precio venta	precio venta
	stock	stock
Estado de la colección	Coleccion sin ese producto (1)	Coleccion con ese producto (2)
Modo del sistema	Usuario autorizado	Usuario autorizado
-	-	-
Condición del parámetro nuevo		Producto con parametros validos
Atributos del parámetro nuevo		nombre
		precio costo
		precio venta
		stock
Estado de la colección		Coleccion sin ese producto
Modo del sistema		Usuario no autorizado (3)

Escenario 1: Coleccion de producto con “papas” pero sin “hamburguesa”

Escenario 2: Colección con papas pero el usuario no está en el modo autorizado.

Bateria de pruebas

	Entradas	**Salidas**	**Escenarios**	**Clases Cubiertas**

		Esperadas	Obtenidas		
Clases correctas	{papas,100,150,20}	Producto se agrego correctamente	PASS	1	1
Clases incorrectas	{hamburguesa, 80,100,10}	No se agrego en la coleccion porque ya existe	PASS	1	2
	{papas,100,150,20}	No se agrego en la coleccion porque el modo del sistema no es correcto (OperacionNoAutorizadaException)	PASS	2	3

metodo: crearComanda(Mesa mesa)

Tabla de particiones

Condición	Clases Correctas	Clases Erróneas

Condición del parámetro	Mesa con parametros validos	Mesa con parametros validos
Atributos del parámetro	nro mesa	nro mesa
	capacidad	capacidad
Estado de la colección	Coleccion con esa mesa (1)	Coleccion sin esa mesa (2)
Composicion de la mesa	Mesa sin comanda	Mesa sin comanda
-	-	-
Condición del parámetro nuevo		Mesa con parametros validos
Atributos del parámetro nuevo		nro mesa
		capacidad
Estado de la colección		Coleccion con esa mesa
Composicion de la mesa		Mesa con comanda (3)

Escenario unico: Colección de mesas con mesa nro 2 sin comanda y nro 3 con comanda.

Bateria de pruebas

	Entradas	**Salidas**		**Clases Cubiertas**
		Esperadas	**Obtenidas**	
Clases correctas	Mesa con {2,5}	Comanda se agrego correctamente a la mesa	PASS	1
Clases incorrectas	Mesa con {4,4}	No se agrego en la	PASS	2

		coleccion porque no existe la mesa (MesaInexistenteException)		
	Mesa con {3,5}	No se agrego en la mesa porque ya existe una comanda (MesaOcupadaException)	PASS	3

metodo: asignarMesa(Mozo mozo, Mesa mesa)

Tabla de particiones

Condición	Clases Correctas	Clases Erróneas
Condición del parámetro	Mesa y mozo con parametros validos	Mesa y mozo con parametros validos
Atributos de los parametros	nro mesa	nro mesa
	capacidad	capacidad
	nombre	nombre

	apellido	apellido
	fecha de nacimiento	fecha de nacimiento
	hijos a cargo	hijos a cargo
	sueldo	sueldo
Estado de las colecciones	Colecciones con mesa y mozo (1)	Coleccion sin mesa(2.1) o Coleccion sin mozo (2.2)

Escenario único: Colección con mesa nro 2, capacidad 5 pero no una con nro 4, capacidad 4 y colección con mozo "Juan Perez" pero no "Matias Perez"

Bateria de pruebas

	Entradas	**Salidas**		**Clases Cubiertas**
		Esperadas	**Obtenidas**	
Clases correctas	({2,5},{"Juan","Perez","13/10/98",2,sueldo"})	Se asigno correctamente el mozo a la mesa	FAIL	1
Clases incorrectas	({2,5},{"Matias","Perez","13/10/00",1,sueldo"})	No se asigno el mozo a la mesa porque este no existe	PASS	2.1
	({4,4},{"Juan","Perez","13/10/98",2,su	No se asigno el mozo a la mesa	PASS	2.2

	eldo"})	porque no existe la mesa (MesaInexistenteException)		

metodo: agregarPromocionProducto(Producto producto, PromocionProducto promocion)

Tabla de particiones

Condición	Clases Correctas	Clases Erróneas
Condición del parámetro	Producto y promocion con parametros validos	Producto y promocion con parametros validos
Atributos de los parametros	nombre	nombre
	precio costo	precio costo
	precio venta	precio venta
	stock	stock
	aplica dos por uno	aplica dos por uno
	aplica dto por cantidad	aplica dto por cantidad

	dto por cantidad minima	dto por cantidad minima
	dto por cantidad precio unitario	dto por cantidad precio unitario
	dias promo	dias promo
Estado de la coleccion producto	Coleccion con ese producto (1)	Coleccion sin ese producto (2)

Escenario unico: Colección con producto papas pero sin hamburguesa

Bateria de pruebas

	Entradas	**Salidas**		**Clases Cubiertas**
		Esperada	**Obtenida**	
Clases correctas	({papas,100,150,20},{true,true,10,15,lunes,miercoles})	La promocion se asocio al producto y se agrego correctamente	PASS	1
Clases incorrectas	({hamburguesa,80,100,10},{true,true,10,15,lunes,miercoles})	La promocion no pudo ser agregada porque no existe el producto (ProductoInexistenteException)	PASS	2

metodo: eliminarProducto(Producto producto)

Tabla de particiones

Condición	Clases Correctas	Clases Erróneas
Condición del parámetro	Producto con parametros validos	Producto con parametros validos
Atributos de los parametros	nombre	nombre
	precio costo	precio costo
	precio venta	precio venta
	stock	stock
Estado del producto	Existe (1)	No existe (2)
Relacion del producto	No esta ascociado a una comanda	
Modo del sistema	Usuario autorizado	Usuario autorizado
-	-	-
Condición del parámetro		Producto con parametros validos
Atributos de los parametros		nombre
		precio costo
		precio venta
		stock
Estado del producto		Existe

Relacion del producto		En una comanda (3)
Modo del sistema		Usuario autorizado
-	-	-
Condición del parámetro		Producto con parametros validos
Atributos de los parametros		nombre
		precio costo
		precio venta
		stock
Estado del producto		Existe
Relacion del producto		En una comanda
Modo del sistema		Usuario no autorizado (4)

Escenario 1: Existe el producto papas pero no hamburguesa y el producto pancho existe y esta asociado a una comanda.

Escenario 2: Existe el producto papas pero el usuario no está en el modo correcto.

Bateria de pruebas

	Entradas	**Salidas**		**Escenario**	**Clases Cubiertas**
		Esperada	**Obtenida**		
Clases correctas	{papas,100,150,20}	Se elimina el producto	PASS	1	1

		correctamente			
Clases incorrectas	{hamburguesa, 80,100,10}	El producto no se elimina porqueno existe (ProductoInexistenteException)	FAIL	1	2
	{pancho, 60,80,15}	El producto no se elimina porque esta asociado a una comanda (ProductoEnComandaExceptio n)	PASS	1	3
	{papas,100,150 ,20}	El producto no se elimina porque el modo del sistema no es correcto (OperacionNoAutorizadaExcep tion)	FAIL	2	4

Clase Operario

metodo: iniciarSesion(String contrasenia)

Tabla de particiones

Condición	Clases Correctas	Clases Erróneas
Estado del usuario	Activo	Activo
Parametro de entrada	Contrasenia coincide (1)	Contrasenia no coincide (2)
Estado del usuario	-	Inactivo (3)
Parametro de entrada	-	Contrasenia coincide

Escenario 1: El usuario está activo con contraseña “Teo1234”

Escenario 2: El usuario con contraseña “Teo1234” está inactivo.

Bateria de pruebas

	Entradas	Salidas	Escenario	Clases Cubiertas

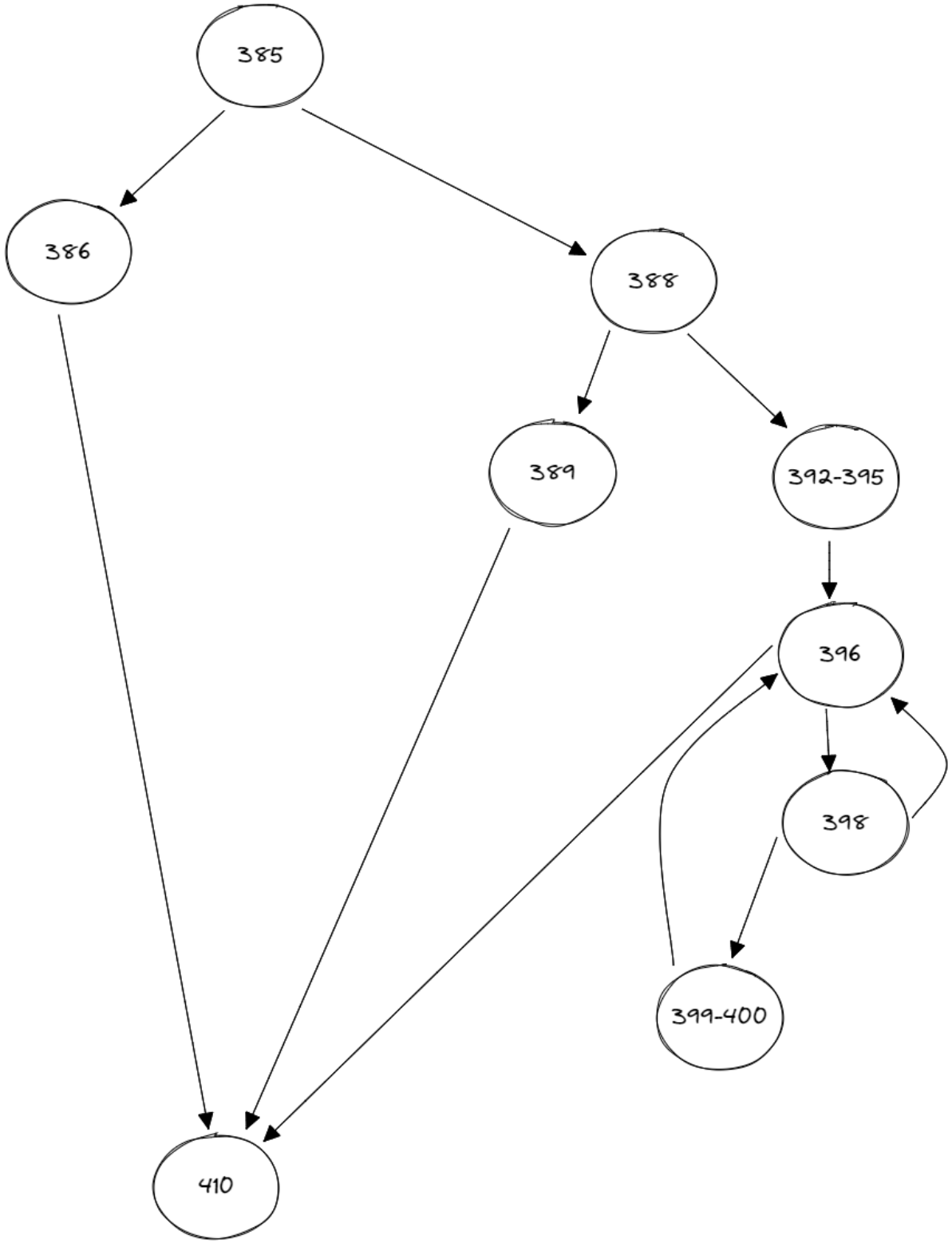
		Esperadas	Obtenidas		
Clases correctas	{Teo1234}	Se inicio sesion	PASS	1	1
Clases incorrectas	{Teo123}	ContraseniaIncorrectaException	PASS	1	2
	{Teo1234}	UsuarioInactivoException	PASS	2	2

Caja Blanca

En el caso de los test de caja blanca , se basan en información sobre cómo el software ha sido diseñado o codificado. Es decir el testeador tiene el código a disposición para poder probarlo y analizar el diseño, código y estructura interna. Por ende las decisiones que tome en cuanto a los datos a suministrar estarán condicionadas por la estructura del código y no por los requerimientos como sucede en caja negra.

En nuestro caso, decidimos realizar este test a dos métodos que no pudimos testear en caja negra. Tomamos métodos que consideramos que tenían suficientes condicionales como para que su testeo se justifique.

metodo: cerrarComanda(Mesa mesa)



Complejidad ciclomática = arcos(12) - nodos(9) + 2 = 5

Complejidad ciclomática = 4 condicionales + 1 = 5

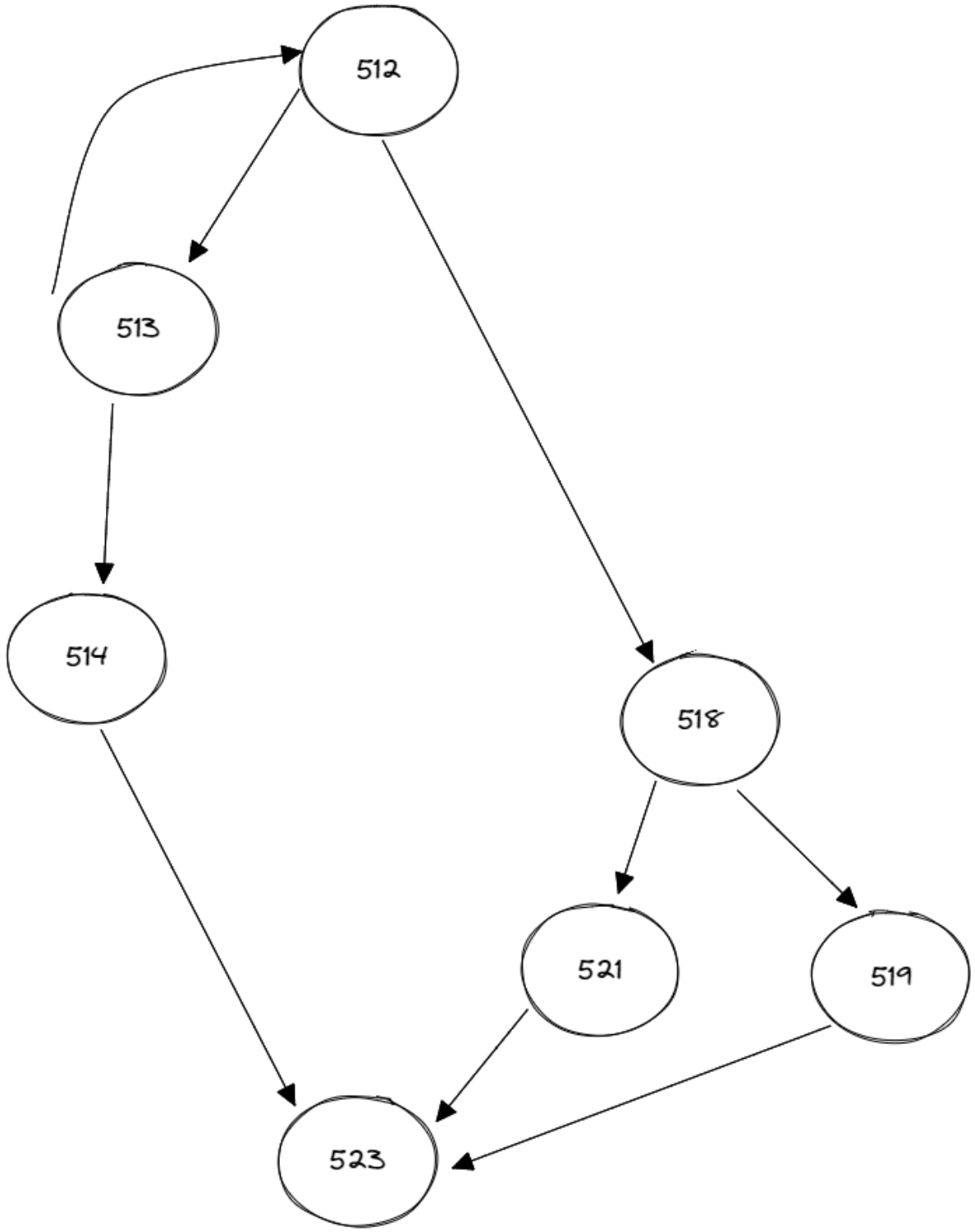
Caminos independientes

C1	385-386-410
C2	385-388-389-410
C3	385-388-395-396-410
C4	385-388-395-396-398-396-410
C5	385-388-395-396-398-400-396-410

Camino independiente N	**Parámetros de entrada**	**Resultado esperado**
1	mesa no esta en en la colección de mesas mesas.contains(mesa) == false	Ingresa al primer if y lanza la excepcion MesaInexistenteException
2	mesas.contains(mesa) == true mesa está ocupada mesa.estaOcupada == false	Ingresa al segundo if y lanza la excepción MesaNoOcupadaEx

		ception
3	mesas.contains(mesa) == true mesa.estaOcupada == true Iterator vacio. Sin embargo,camino no posible ya que si mesa ocupada, significa que se encuentra en el iterator porque cuando crea comanda es cuando se ocupa..	Camino no posible, forzado.
4	mesas.contains(mesa) == true mesa.estaOcupada == true El primer mozo no tiene la mesa asignada. Camino no posible ya que si mesa ocupada, significa que se encuentra en el iterator.	Camino no posible, forzado.
5	mesas.contains(mesa) == true mesa.estaOcupada == true Se encuentra en el iterator encontrado == true	Ingresa al ciclo while y encuentra la mesa, sale del ciclo, la saca de la lista y la desocupa

metodo: buscarOperario(String nombreUsuario)



Complejidad ciclomática: arcos(10) - nodos(8) + 2 = 4

Complejidad ciclomática: 3 condicionales + 1 = 4

Caminos independientes

C1	512-513-514-523
C2	512-518-519-523
C3	512-518-521-523

Camino independiente N	**Parámetros de entrada**	**Resultado esperado**
1	Lista de operarios no vacia Nombre usuario perteneciente a un operario	Entra al for y cuando encuentra el operario buscado entra al if para retornar su valor
2	Nombre de usuario perteneciente a un administrador	No entra al for y cuando sale entra al if por ser administrador y devuelve su valor
3	Nombre de usuario no perteneciente a ningún operario	No entra al for y entra por el else final tirando la

		excepción OperarioInexistemte Exception

Conclusión

Para la realización de los diferentes tipos de testeo se tuvieron en cuenta las distintas metodologías vistas en la materia. Con el uso de diferentes métodos se pudo analizar arduamente el código encontrando algunos errores que inevitablemente se le pueden escapar al programador.

Concluyendo, podemos afirmar que pusimos en práctica los conceptos vistos en la materia y que de esta manera pudimos experimentar otro lado de la programación la cual desconociamos.