



Trabajo Práctico especial

—Taller de la programación I

Grupo 4

Testing realizado por Valentina Reale y Martiniano Presa

Otros integrantes:

Paula Bonifazi Aquino

Aureliano Vega Imbalde

Testeo de caja negra

A partir de la srs y de los contratos, determinamos los casos de prueba de caja negra. Lo hicimos para los métodos centrales del programa, para que este cumpla con sus funcionalidades más importantes de una manera robusta. En el test de Unidad fueron detallados todos los métodos con todos los casos de prueba posibles.

Como aclaración para una buena comprensión, no realizamos testeo directo a métodos privados ni tampoco testeamos lo que se encuentra especificado en las precondiciones.

 CajaNegraTP.xlsx

Una observación que podemos hacer es que en el método login() hay una excepcion que no se lanza nunca ya que en la precondición asegura que es imposible. La excepción es "OperarioNoActivo_Exception" y la precondición afirma que el operario debe estar activo.

Otra observación de la misma índole la vemos en el test del método abreComanda(int nroMesa). El mismo declara que lanza una excepcion de tipo "MozoNoActivo_Exception" pero la misma nunca se lanza ya que es 'pisada' por la excepcion de "TodosMozosInactivos_Exception". Ya que el abreComanda(int nroMesa) debería asignar un mozo si o si, busca en la colección de mozos a cualquiera que esté activo. El único caso en que el mozo no estaría activo sería que todos lo estén.

Test de cobertura

Test de unidad blabla






















Al finalizar verificamos que los errores que se nos daban en las baterías de prueba, coinciden con la mayoría de los vistos en este test.

A partir de los resultados del test de cobertura podemos notar que se cubrió un **61.9%** del total de sentencias del paquete "test". Además se realizaron **129** runs de los cuales **25** contienen fallas y se presenta **1** error. Del paquete "modelo" se cubrió un **88.9%**, del paquete "negocio" se cubrió un **80.4%** y del paquete "excepciones" se cubrió un **48.6%**.

Conformando una cobertura del proyecto total de **63.3%**.

Element	Coverage
▼ Taller1-Grupo4	 66,3 %
▼ src	 66,3 %
> test	 61,9 %
> negocio	 80,4 %
> persistencia	 18,1 %
> prueba	0,0 %
> modelo	 88,9 %
> excepciones	48,6 %

Con un poco más de detalle:

▼ test	 61,9 %
> FuncionalidadOperariosTest2.java	 70,3 %
> FuncionalidadOperariosTest.java	 51,5 %
> FuncionalidadAdminTest2.java	 64,3 %
> SistemaTest2_ColeccionNoVacia.java	 48,2 %
> PersistenciaXMLTest.java	 62,8 %
> FuncionalidadAdminTest.java	 62,9 %
> SistemaTest_ColeccionVacia.java	 52,1 %
> AllTests.java	0,0 %
▼ negocio	 80,4 %
> FuncionalidadOperarios.java	 75,1 %
> Sistema.java	 87,1 %
> GestionProdPromo.java	0,0 %
> GestionProdTemp.java	0,0 %
> GestionComandas.java	0,0 %
> GestionProdVenta.java	0,0 %
> FuncionalidadAdmin.java	 100,0 %
▼ persistencia	 18,1 %
> PersistenciaBIN.java	 0,0 %
> MozoDTO.java	 0,0 %
> MesaAtendidaDTO.java	 0,0 %
> CerveceriaDTO.java	 0,0 %
> MesaDTO.java	 0,0 %
> ComandaDTO.java	 0,0 %
> UtilCerveceria.java	 0,0 %
> PersistenciaXML.java	 100,0 %

Observaciones

A partir de estos resultados podemos concluir que el porcentaje de cobertura del paquete “negocio”, “persistencia” y “excepciones” no se completa debido a razones tales como:

- No se testean métodos privados y contiene algunos

- Las clases de Gestión... nunca se llaman ya que no tienen ninguna funcionalidad importante, por lo que no se recorren.
- La persistencia solo se testea para el caso XML.
- La mitad de las excepciones no se usan nunca.

En la clase Sistema no se testearon los métodos loginOperario() y loginAdmin() de manera directa. Pero si se testan indirectamente. Notamos que resulta imposible el login de un Admin por primera vez debido a lo siguiente:

```
private FuncionalidadAdmin loginAdmin(String password, String NyAdmin, String nuevaPasswordAdmin)
{
    FuncionalidadAdmin fA = null;
    if (password.equals(this.passwordADMIN))
        fA = new FuncionalidadAdmin(new Administrador(NyAdmin, nuevaPasswordAdmin));
    else if (this.operarios.get("ADMIN").getPassword().equals(password))
        fA = new FuncionalidadAdmin(this.operarios.get("ADMIN"));
    else
        throw new ContraseñaIncorrecta_Exception();

    return fA;
}
```

La colección de operarios está vacía desde un principio por lo que nunca logrará hacer el login correcto, ya que al “logear” al administrador no se lo agrega a la colección de operarios para luego poder logearse con una contraseña distinta a la predeterminada.

Otra observación que podemos agregar es que en el método para modificar una mesa que pertenece a FuncionalidadOperario.class tiene la declaración de la excepción NroMesaRepetido_Exception pero nunca es lanzada, ya que solo contiene un parámetro con nroMesa y se usa para buscar una mesa ya existente. No para modificar el mismo.

```
public void modificaMesa(int nroMesa, int cantSillas, Enumerados.estadoMesa estado)
    throws NroMesaRepetido_Exception, NoExisteEnLaColeccion_Exception
{
    if (!Sistema.getInstance().getMesas().containsKey(nroMesa))
        throw new NoExisteEnLaColeccion_Exception(Integer.toString(nroMesa));
    Mesa mesa = Sistema.getInstance().getMesas().get(nroMesa);
    mesa.setCantSillas(cantSillas);
    mesa.setEstado(estado);
}
```

Recién comentamos que el testSuite() nos devuelve un error. El mismo se debe a que no se realiza bien la persistencia de los objetos y al intentar leer el archivo persistido se produce un error. Esto se debe a que algunos de los objetos no cumplen con las condiciones necesarias para la correcta persistencia XML.

```

123 @Test
124 public void testLeer() throws ClassNotFoundException {
125     try {
126         this.persiste.abrirInput("Archivo.xml");
127         Mozo mozo = (Mozo) this.persiste.leer();
128         Producto prod = (Producto) this.persiste.leer();
129         Mesa mesa = (Mesa) this.persiste.leer();
130         MesaAtendida mesaAt = (MesaAtendida) this.persiste.leer();
131         Comanda com = (Comanda) this.persiste.leer();
132         Assert.assertTrue("No se leyo correctamente el producto",
133             prod.getNombre().equals(prod1.getNombre()) && prod1.getIdProd() == prod.getIdProd());
134         Assert.assertTrue("No se leyo correctamente el mozo",
135             mozo.getNyA().equals("Juan") && mozo.getCantHijos() == 4);
136         Assert.assertTrue("No se leyo correctamente la mesa atendida",
137             mesaAt.getTotal() == 200 && mesaAt.getMesa().getNroMesa() == 0);
138         Assert.assertTrue("No se leyo correctamente la comanda", com.getMesa().getNroMesa() == 0);
139         Assert.assertTrue("No se leyo correctamente la mesa",
140             mesa.getNroMesa() == this.ml.getNroMesa() && mesa.getCantSillas() == this.ml.getCantSillas());
141         this.persiste.cerrarInput();
142     } catch (IOException e) {
143         Assert.fail("No deberia lanzarse excepcion");
144     }
145 }

```

Otro gran problema ocurrió con el método abreComanda(), que fue testeado con varios casos de prueba y en la mayoría se lanza una excepción que no debería ser lanzada. Esto lo detallamos a continuación:

Failures resultantes del test de cobertura:

test.FuncionalidadAdminTest

Presenta una falla en el siguiente método:

1. testAgregaMozoIncorrecto3() → Se intenta agregar un Mozo con día y mes de nacimiento < 0. No son tratados estos errores.

test.FuncionalidadAdminTest2

Presenta una falla en el siguiente método:

1. testAgregaMozoIncorrecto1() → Se intenta agregar un mozo con día y mes de nacimiento menores a cero. No tira ningún error.

test.FuncionalidadOperarioTest

Presenta tres fallas en total en los siguientes métodos:

1. testAgregaPromocionProdIncorrecta() → Se intenta agregar una promo con los dos parámetros de aplica2x1 y aplicaDtoPorCant en falso y no se lanza excepcion. Por regla no debería suceder.
2. testAbreComandaIncorrecto2() → Se intenta abrir una comanda con mesa existente y en estado libre, con mozo existente y en estado activo y se lanza una excepción de

tipo "TodasMesasInhabilitadas_Exception". Según el contrato esta se lanza cuando el sistema no tiene ninguna mesa libre, no es el caso. Por lo tanto está mal lanzada.

Realizando cualquier combinación de parámetros válidos o no, lanza la excepción de tipo "TodasMesasInhabilitadas_Exception".

3. testAbreComandaIncorrecto3() → Se intenta abrir una comanda con mesa existente y en estado libre, con mozo existente y en estado AUSENTE y se lanza una excepción de tipo "TodasMesasInhabilitadas_Exception". Se repite.

test.FuncionalidadOperarioTest2

Presenta siete fallas en total en los siguientes métodos:

1. testAbreComandaCorrecto() → Se intenta abrir una comanda con mesa existente y en estado libre y se lanza una excepción de tipo "TodasMesasInhabilitadas_Exception". No debería lanzarla.
2. testAbreComandaIncorrecto2() → Se intenta abrir una comanda con mesa existente y en estado libre, con mozo existente y en estado activo y se lanza una excepción de tipo "TodasMesasInhabilitadas_Exception". Mal lanzada
3. testAbreComandaIncorrecto3() → Se intenta abrir una comanda con mesa existente y en estado libre, con mozo existente y en estado ausente y se lanza una excepción de tipo "TodasMesasInhabilitadas_Exception". Mal lanzada.
4. testConsumoPromedioPorMesaConMesasAtendidas() → el Promedio es mal calculado. No se devuelve un número en res.
5. testConsumoPromedioPorMesaSinMesasAtendidas() → el Promedio es mal calculado. No se devuelve un número en res.
6. testAgregaPromocionProdIncorrecta() → Se intenta agregar una promo con los dos parámetros de aplica2x1 y aplicaDtoPorCant en falso y no se lanza excepcion. Por regla no debería suceder.
7. testEstadisticasEmpleadoSinMesasAtendidas() → el Promedio es mal calculado. No se devuelve un número en res.

test.SistemaTest_ColeccionVacía

Se presentan seis fallas en los siguientes métodos:

1. testLoginIncorrectoAdCorto() → Se inserta como parámetro una nuevaPassword que no cumple con el requisito de tener una longitud mayor a 6.
2. testLoginIncorrectoAdLargo() → Se inserta como parámetro una nuevaPassword que no cumple con el requisito de tener una longitud menor a 12.
3. testLoginIncorrectoAdSinMayus() → Se inserta como parámetro una nuevaPassword que tiene entre 6 y 12 caracteres pero no contiene mayúscula.

4. testLoginIncorrectoAdSinDigito() → Se inserta como parámetro una nuevaPassword que tiene entre 6 y 12 caracteres pero no contiene ningún dígito.
5. testLoginIncorrectoAdNulo() → Se inserta como parámetro una nuevaPassword null
6. testLoginIncorrectoAdVacio() → Se inserta como parámetro una nuevaPassword == "".

test.SistemaTest2_ColeccionNoVacia

Se presentan siete fallas en los siguientes métodos:

1. testLoginCorrectoOp() → Debería devolver una instancia de tipo "FuncionalidadOperario" y devuelve null. Algo no se realiza bien.
2. testLoginIncorrectoAdCorto() → Se inserta como parámetro una nuevaPassword que no cumple con el requisito de tener una longitud mayor a 6.
3. testLoginIncorrectoAdLargo() → Se inserta como parámetro una nuevaPassword que no cumple con el requisito de tener una longitud menor a 12.
4. testLoginIncorrectoAdSinMayus() → Se inserta como parámetro una nuevaPassword que tiene entre 6 y 12 caracteres pero no contiene mayúscula.
5. testLoginIncorrectoAdSinDigito() → Se inserta como parámetro una nuevaPassword que tiene entre 6 y 12 caracteres pero no contiene ningún dígito.
6. testLoginIncorrectoAdNulo() → Se inserta como parámetro una nuevaPassword null
7. testLoginIncorrectoAdVacio() → Se inserta como parámetro una nuevaPassword == "".

Errores resultantes del test de caja negra (Clases de equivalencia)

Los errores resultantes de las baterías de pruebas coinciden en su mayoría con los vistos en el test de Unidad, a diferencia del método abreComanda(int nroMesa) que se ve en el test de Cobertura. En el mismo, sea cual sea el caso de prueba y sea cual sea el escenario, siempre lanza la excepción de tipo "TodasMesasInhabilitadas_Exception" (solo en el caso en que el nroMesa no existe no la lanza, y lanza la correspondiente). Ya sea un caso de prueba válido o uno inválido pero que debería lanzar una excepción de otro tipo, sucede lo mismo. Lo tratamos con más detenimiento en el test de Caja Blanca a continuación.

Test de caja blanca

https://miro.com/app/board/uXjVPHkl0t0=

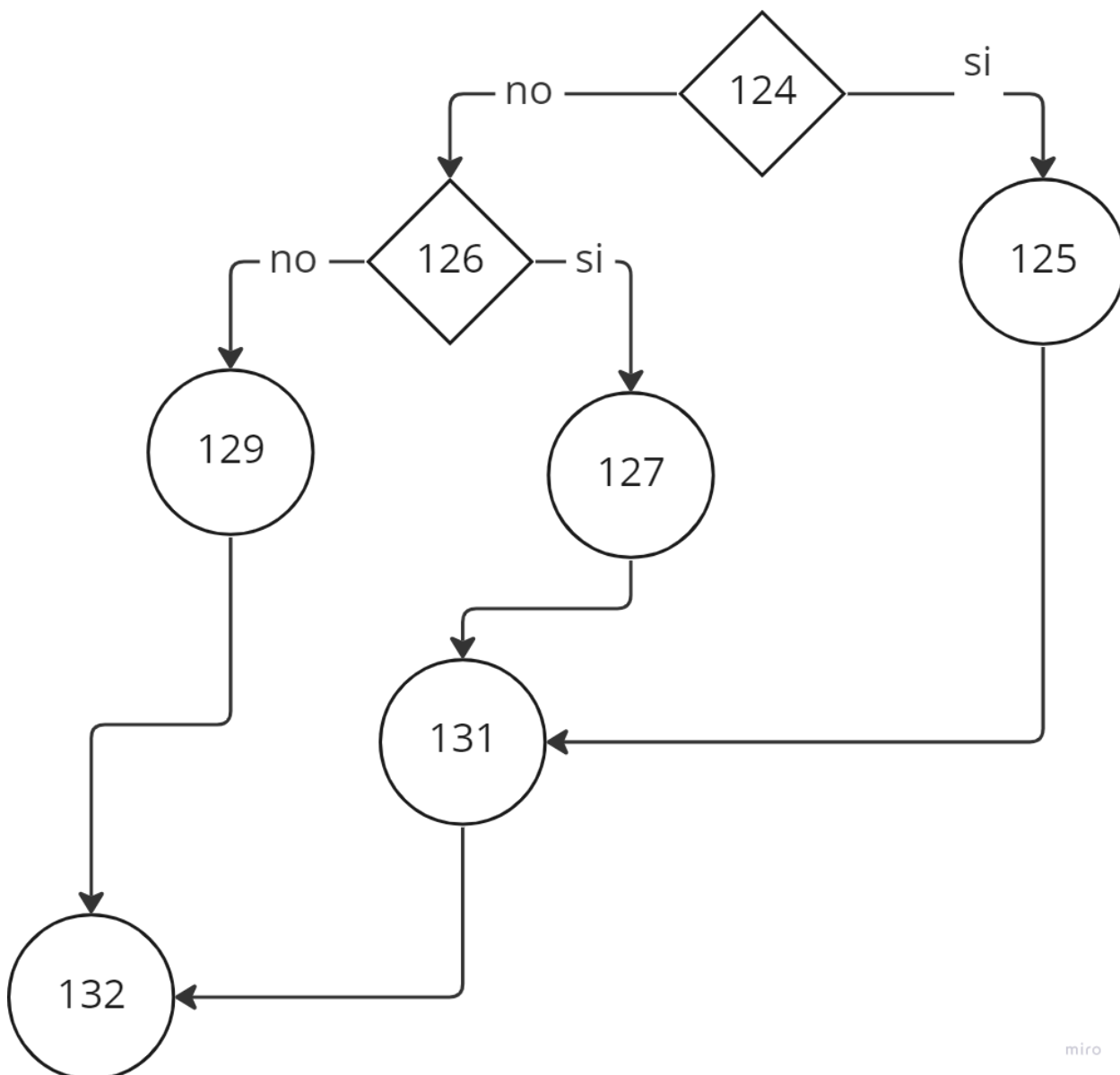
Aplicamos el test de caja blanca a los métodos para los cuales el test de cobertura nos arrojó un porcentaje de cobertura no tan alto.

Clase Sistema

```

121 private FuncionalidadAdmin loginAdmin(String password, String NyAAdmin, String nuevaPasswordAdmin) throw
122 {
123     FuncionalidadAdmin fA = null;
124     if (password.equals(this.passwordADMIN))
125         fA = new FuncionalidadAdmin(new Administrador(NyAAdmin, nuevaPasswordAdmin));
126     else if (this.operarios.get("ADMIN").getPassword().equals(password))
127         fA = new FuncionalidadAdmin(this.operarios.get("ADMIN"));
128     else
129         throw new ContraseñaIncorrecta_Exception();
130
131     return fA;
132 }

```



$$v(g) = \text{arcos} - \text{nodos} + 2 = 8 - 7 + 2 = 3$$

$$v(g) = n^{\circ}\text{regiones} =$$

$$v(g) = n^{\circ}\text{ nodos condición} + 1 = 2 + 1 = 3$$

C1 = 124-126-127-131-132

C2 = 124-126-129-132

C3 = 124-125-131-132

Los caminos básicos coinciden con la complejidad ciclomática del grafo.

Escenario 1 = Colección de operarios vacía.

Escenario 2 = Colección de operarios con el ADMIN, password del admin -> "Juan123".

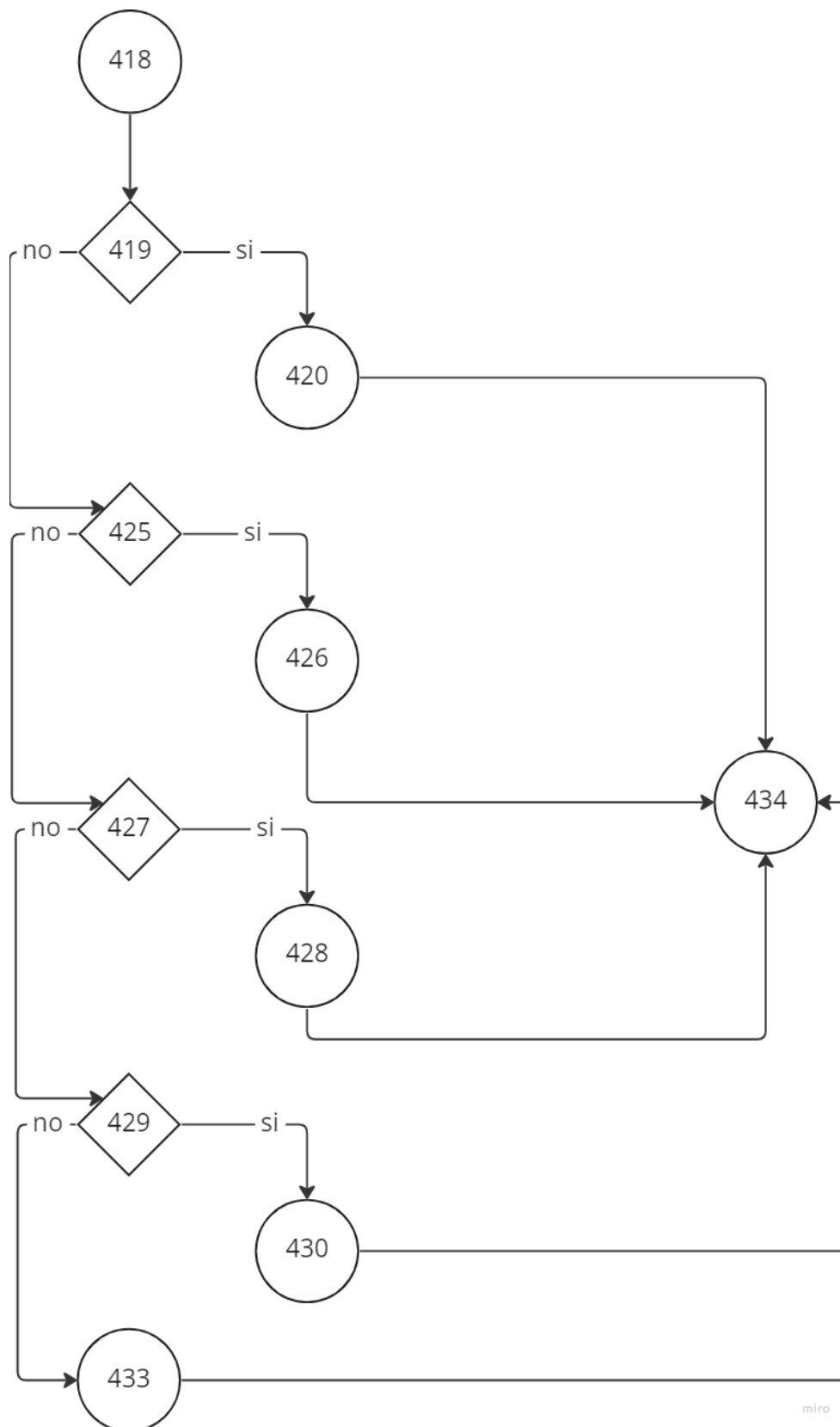
Camino	Escenario	Datos de entrada	Salida esperada
C1	2	Password = "Juan123", nuevaPassword = "Carlos1", NyAAdmin = "Juan"	Hace el new funcionalidadAdmin con el admin que fue sacado de la colección. Retorna la funcionalidad.
C2	2	Password = "Carlos1", nuevaPassword = "Carlos2", NyAAdmin = "Juan"	Lanza ContraseñaIncorrecta_Exception()
C3	3	Password = "ADMIN123", nuevaPassword = "Carlos1", NyAAdmin = "Juan"	Hace el new de funcionalidadAdmin, y el new Administrador. Retorna la funcionalidad.

Error: El C1 no se da nunca como se ve en el test de cobertura ya que el Admin no es agregado a la colección de operarios. Es decir que el escenario 2 no es posible nunca.

```

115 public void abreComanda(int nroMesa)
116     throws TodasMesasInhabilitadas_Exception, TodosMozosInactivos_Exception, MenosDe2ProdsEnPromocion_Exception,
117     MozoNoActivo_Exception, NoHayProductos_Exception, MesaOcupada_Exception, NoExisteEnLaColeccion_Exception
118 {
119     if (!Sistema.getInstance().getMesas().containsKey(nroMesa))
120         throw new NoExisteEnLaColeccion_Exception(Integer.toString(nroMesa));
121     this.TodasMesasInhabilitadas();
122     this.TodosMozosInactivos();
123     this.MenosDe2ProdsEnPromocion();
124     Mesa mesa = Sistema.getInstance().getMesas().get(nroMesa);
125     if (!mesa.getMozo().getEstado().equals(Enumerados.estadoMozo.ACTIVO))
126         throw new MozoNoActivo_Exception(mesa.getMozo().getNyA());
127     if (Sistema.getInstance().getProductos().size() == 0)
128         throw new NoHayProductos_Exception();
129     if (!mesa.getEstado().equals(Enumerados.estadoMesa.LIBRE))
130         throw new MesaOcupada_Exception(nroMesa);
131
132     Sistema.getInstance().getComandas().put(nroMesa, new Comanda(mesa));
133     mesa.setEstado(Enumerados.estadoMesa.OCUPADA);
134 }

```



$$v(g) = \text{arcos} - \text{nodos} + 2 = 14 - 11 + 2 = 5$$

$$v(g) = n^{\circ}\text{regiones} =$$

$$v(g) = n^{\circ}\text{ nodos condición} + 1 = 4 + 1 = 5$$

C1 = 418-419-425-427-429-433-434

C2 = 418-419-420-424

C3 = 418-419-425-426-434

C4 = 418-419-425-427-428-434

C5 = 418-419-425-427-429-430-434

Los caminos básicos coinciden con la complejidad ciclomática del grafo.

Escenario 1 = Colección de mesas con la mesa nroMesa = 1 con estado LIBRE, y con mozo con estadoMozo == ACTIVO.

Escenario 2 = Colección de mesas vacía.

Escenario 3 = Colección de mesas con la mesa nroMesa = 1 con mozo con estadoMozo != ACTIVO.

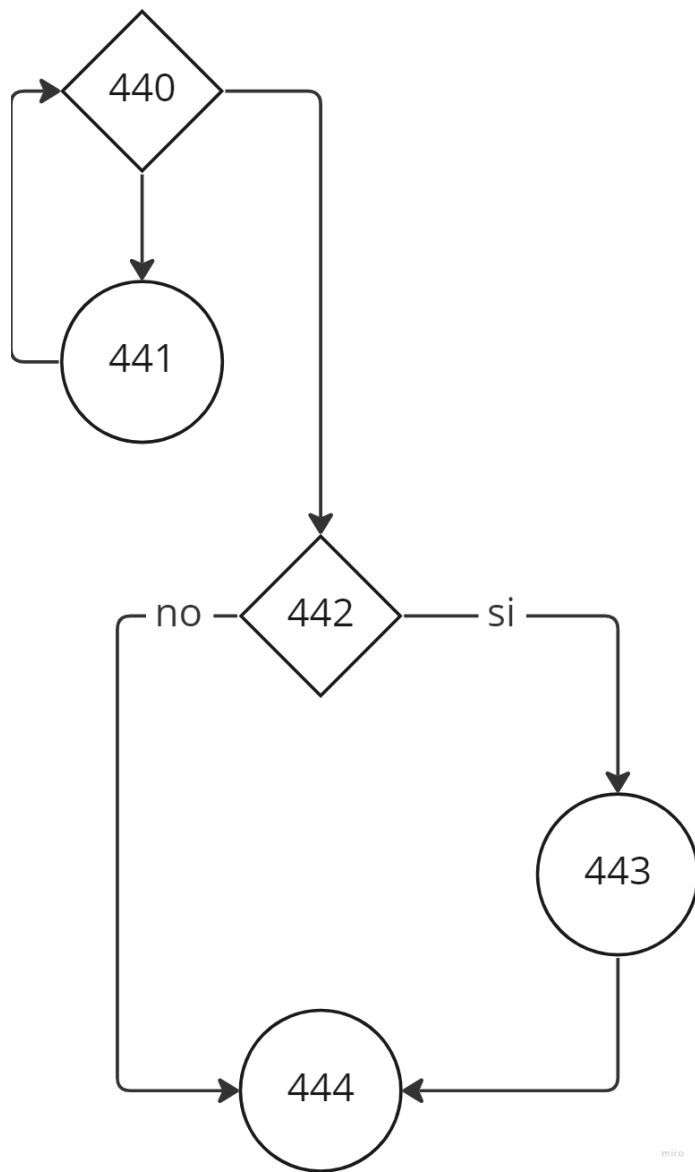
Escenario 4 = Colección de productos vacía.

Escenario 5 = Colección de mesas con la mesa nroMesa = 1 con estado OCUPADA.

Camino	Escenario	Datos de entrada	Salida esperada
C1	1	nroMesa = 1	Setea el estado de la mesa como OCUPADA, agrega a la colección de comandas
C2	2	nroMesa = 1	throw new NoExisteEnLaColeccion_Exception(Integer.toString(nroMesa))

C3	3	nroMesa = 1	throw new MozoNoActivo_Exception(mesa.getMozo().getNyA())
C4	4	nroMesa = 1	throw new NoHayProductos_Exception()
C5	5	nroMesa = 1	throw new MesaOcupada_Exception(nroMesa)

Error : Si el nroMesa existe en la colección, en cualquier otro caso se llama el metodo todasMesasInhabilitadas, que lanza una exception interrumpiendo la ejecución.



$$v(g) = \text{arcos} - \text{nodos} + 2 = 6 - 5 + 2 = 3$$

$$v(g) = \text{n}^\circ \text{regiones} =$$

$$v(g) = \text{n}^\circ \text{ nodos condición} + 1 = 2 + 1 = 3$$

$$C1 = 440-441-440-442-444$$

$$C2 = 440-441-440-442-443$$

$$C3 = 440-442-444$$

$$C4 = 440-442-443-444$$

La complejidad ciclomática del grafo es 3, pero con los caminos C2 y C3 alcanza para pasar por todas las sentencias.

Escenario 1: Colección de mesas con mesas ocupadas.

Escenario 2: Colección de mesas con una mesa libre.

Camino	Escenario	Datos de entrada	Salida esperada
C2	1		Lanza TodasMesasInhabilitadas_Exception()
C3	2		No lanza TodasMesasInhabilitadas_Exception()

Observación 2: Si no existiera ninguna mesa en la colección, lanza de igual manera TodasMesasInhabilitadas_Exception().

Observación 1: En el testing de cobertura la exception es lanzada en todos los casos, por lo que sospechamos que la condición del while tiene un error.

Test de Integración

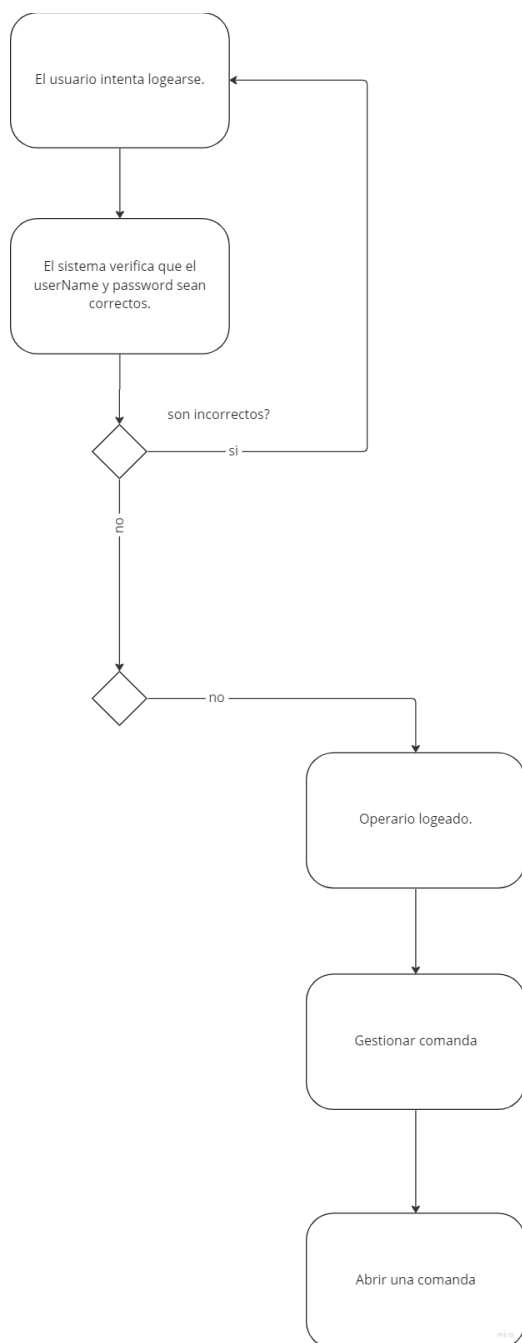
Vamos a realizar el test de integración descendente en profundidad.

En base a nuestro diagrama de actividad, elegimos algunos casos de uso para realizarles un test de integración.

Diagrama de actividad: <https://miro.com/app/board/uXjVPHkl0t0=/>

Entrada	Escenario	Salida esperada
nroMesa= 0	<ul style="list-style-type: none"> Colección de operarios con un operario con Nya = "Juan", username = "Juan10", 	Comanda creada y agregada a la colección de comandas.

	<p>password = "Juan123".</p> <ul style="list-style-type: none">• Colección de mesas con mesa con nroMesa = 0.• Colección de productos con 2 productos.	
--	---	--




```

@Test
public void testProfundidadAbriComandaCorrecto() {

    try {
        this.funcOp = sistema.login(username, password, Nya, "Contra12");
        Assert.assertNotNull("Se deberia haber retornado una instancia de funcionalidad operario", funcOp);
    } catch (UserNameIncorrecto_Exception e1) {

        Assert.fail("No deberia lanzarse la UserNameIncorrecto_Exception");

    } catch (ContrasenaIncorrecta_Exception e1) {
        Assert.fail("No deberia lanzarse la ContrasenaIncorrecta_Exception");
    } catch (OperarioNoActivo_Exception e1) {
        Assert.fail("No deberia lanzarse la OperarioNoActivo_Exception");
    }

    try {
        this.funcOp.abreComanda(mesa.getNroMesa());
        Assert.fail("Deberia lanzarse la MenosDe2ProdsEnPromocion_Exception");
    } catch (TodosMozosInactivos_Exception e) {
        Assert.fail("No deberia lanzarse la TodosMozosInactivos_Exception");
    } catch (MozoNoActivo_Exception e) {
        Assert.fail("No deberia lanzarse la MozoNoActivo_Exception");
    } catch (NoHayProductos_Exception e) {
        Assert.fail("No deberia lanzarse la NoHayProductos_Exception");
    } catch (MesaOcupada_Exception e) {
        Assert.fail("No deberia lanzarse la MesaOcupada_Exception");
    } catch (NoExisteEnLaColeccion_Exception e) {
        Assert.fail("No deberia lanzarse la NoExisteEnLaColeccion_Exception");
    } catch (TodasMesasInhabilitadas_Exception e) {
        Assert.fail("No deberia lanzarse la TodasMesasInhabilitadas_Exception");
    } catch (MenosDe2ProdsEnPromocion_Exception e) {

    }

}

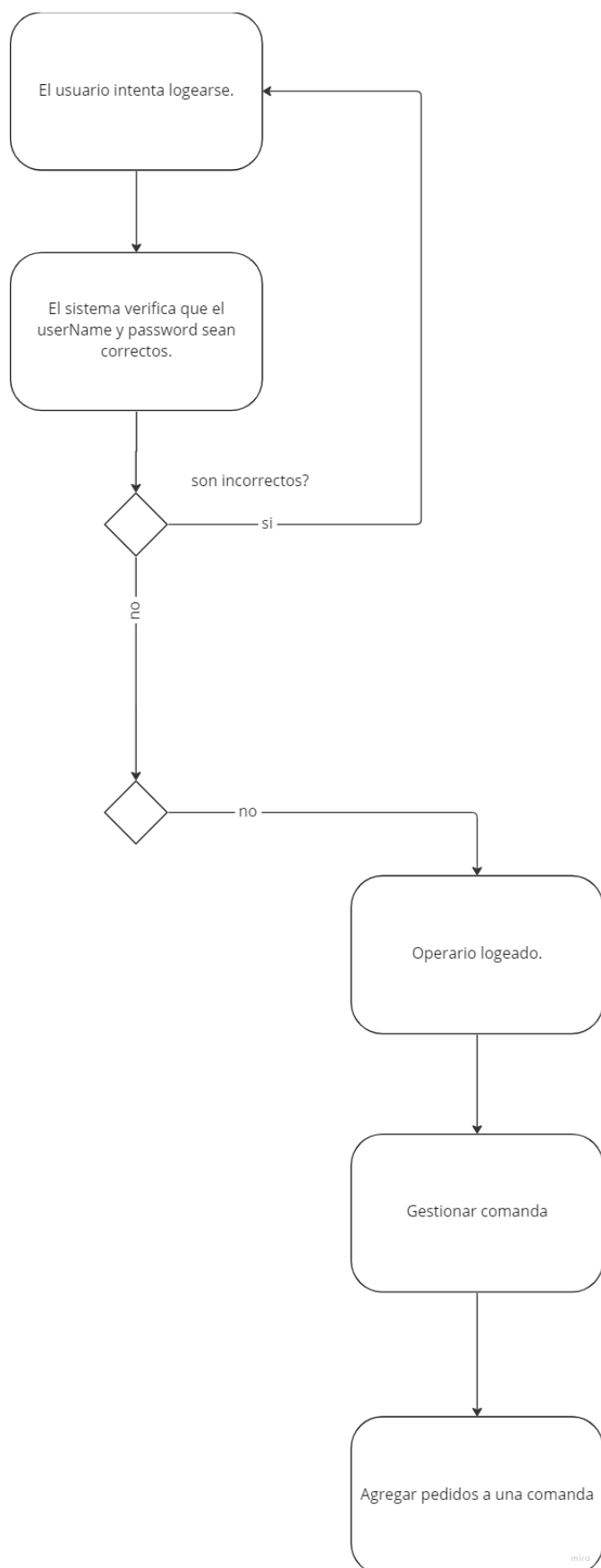
```

En este caso de prueba, corroboramos el buen funcionamiento del login para el caso de un operario.

Error: La creación de la comanda lanza TodasMesasInhabilitadas_Exception, a pesar de que hay una mesa libre debido al método TodasMesasInhabilitadas.

Observación: El método login debería tener nombres más claros en los parámetros.

NyaAdmin corresponde al Nya de Operario que se quiera logear, sea el administrador o no. Y también el campo nuevoPassword no es necesario cuando se quiere logear un operario.



```

@Test
public void testProfundidadAgregarPedidoComandaCorrecto() {

    sistema.getComandas().put(mesa.getNroMesa(), new Comanda(mesa));
    mesa.setEstado(Enumerados.estadoMesa.OCUPADA);

    try {
        this.funcOp = sistema.login(username, password, Nya, "Contra12");
        Assert.assertNotNull("Se deberia haber retornado una instancia de funcionalidad operario", funcOp);
    } catch (UserNameIncorrecto_Exception e1) {

        Assert.fail("No deberia lanzarse la UserNameIncorrecto_Exception");

    } catch (ContraseñaIncorrecta_Exception e1) {
        Assert.fail("No deberia lanzarse la ContraseñaIncorrecta_Exception");
    } catch (OperarioNoActivo_Exception e1) {
        Assert.fail("No deberia lanzarse la OperarioNoActivo_Exception");
    }

    try {
        this.funcOp.AgregaPedidoAComanda(mesa.getNroMesa(), prod1.getIdProd(), 2);
    } catch (MesaNoTieneComanda_Exception e1) {
        Assert.fail("No deberia lanzarse la MesaNoTieneComanda_Exception");
    } catch (StockInsuficiente_Exception e1) {
        Assert.fail("No deberia lanzarse la StockInsuficiente_Exception");
    } catch (NoExisteEnLaColeccion_Exception e1) {
        Assert.fail("No deberia lanzarse la NoExisteEnLaColeccion_Exception");
    }

}

```

Entrada	Escenario	Salida esperada
nroMesa= 0, idProd = 0, cantidad = 2	<p>Colección de operarios con un operario con Nya = "Juan", username = "Juan10", password = "Juan123".</p> <p>Colección de mesas con mesa con nroMesa = 0.</p> <p>Colección de productos con 2 productos.</p> <p>Colección de comandas con una comanda asociada a la mesa nroMesa=0.</p> <p>Producto con idProd = 0 con stock =50</p>	Pedido agregado a la comanda.

El método para este escenario funciona correctamente, sin lanzar ninguna exception.

Observación: Por los testeos previos, pudimos aislar el error del abreComanda y TodasMesasInhabilitadas. Por lo que para los casos de prueba del método AgregaPedidoComanda se crea la comanda en el mismo método testProfundidadAgregarPedidoComandaCorrecto.

Esto podría hacerse también implementando un mock, que creara la comanda y la agregara a la colección.

```
@Test
public void testProfundidadAgregarPedidoComandaIncorrecto() {

    sistema.getComandas().put(mesa.getNroMesa(), new Comanda(mesa));
    mesa.setEstado(Enumerados.estadoMesa.OCUPADA);

    try {
        this.funcOp = sistema.login(username, password, Nya, "Contra12");
        Assert.assertNotNull("Se deberia haber retornado una instancia de funcionalidad operario", funcOp);
    } catch (UserNameIncorrecto_Exception el) {

        Assert.fail("No deberia lanzarse la UserNameIncorrecto_Exception");

    } catch (ContrasenaIncorrecta_Exception el) {
        Assert.fail("No deberia lanzarse la ContrasenaIncorrecta_Exception");
    } catch (OperarioNoActivo_Exception el) {
        Assert.fail("No deberia lanzarse la OperarioNoActivo_Exception");
    }

    try {
        this.funcOp.AgregaPedidoAComanda(mesa.getNroMesa(), prod1.getIdProd(), 100);
        Assert.fail("Deberia lanzarse la StockInsuficiente_Exception");
    } catch (MesaNoTieneComanda_Exception el) {
        Assert.fail("No deberia lanzarse la MesaNoTieneComanda_Exception");
    } catch (NoExisteEnLaColeccion_Exception el) {
        Assert.fail("No deberia lanzarse la NoExisteEnLaColeccion_Exception");
    } catch (StockInsuficiente_Exception el) {

    }

}
```

Entrada	Escenario	Salida esperada
nroMesa= 0, idProd = 0, cantidad = 100	Colección de operarios con un operario con Nya = "Juan",username = "Juan10", password = "Juan123". Colección de mesas con mesa con nroMesa = 0.	StockInsuficiente_Exception

	<p>Colección de productos con 2 productos.</p> <p>Colección de comandas con una comanda asociada a la mesa nroMesa=0.</p> <p>Producto con idProd = 0 con stock =50</p>	
--	--	--

El método cumple con el resultado esperado.

Test de GUI

Decidimos hacer el test de la interfaz que maneja la modificación de un mozo para cualquier tipo de operario.

Cabe aclarar que es una interfaz de nuestra implementación debido a que el código que testeamos no cuenta con ninguna interfaz gráfica.

El diseño perteneciente a la misma es el siguiente:



Modifica Mozo

Seleccione Mozo Valen ▼

Nombre y apellido Valen

Cantidad de hijos 0

Estado ACTIVO ▼

Confirmar

Volver

Luego de unas varias pruebas a mano de la de funcionalidad de la misma, proseguimos a hacer el test de GUI. Definimos como escenario una colección de 4 mozos con nombres: "Valen", "Marti", "Pau", y "Penelope".

Se testean los siguientes casos:


1. public void testCantHijosPos() → Se intenta modificar la cantidad de hijos de un mozo existente por un nro de hijos positivo o igual a cero. El botón de confirmar debería estar habilitado, al igual que el de volver. Se modifica el atributo. Todo esto se cumple ✓
2. public void testCantHijosNeg() → Se intenta modificar la cantidad de hijos de un mozo existente por un nro de hijos negativo. El botón de confirmar debería estar inhabilitado, pero no el de volver volver. Por lo tanto resulta imposible modificarlo. Todo esto se cumple ✓
3. public void testMuestraDatosCorrectos() → Se ingresa cualquier mozo existente en el comboBox. La información de los campos pertenecientes al nombre, la cantidad de hijos y el comboBox de estado debería corresponder al mozo ingresado. Todo esto se cumple ✓
4. public void testMuestraDatosCorrectosMozoNoExiste() → Se ingresa un nombre no existente en la colección de Mozos. El programa crashea ✗

Debería capturar la excepción de alguna manera e informarle al usuario, tal vez con una ventana emergente.

Observación: Aprovechando la situación de que testeamos una ventana propia, implementamos el falsoOptionPane para poder verificar que la ventana emergente correspondiente a la confirmación de la modificación, tire el mensaje esperado. Creemos que por una selección no esperada del mozo en el JComboBox (por la complicación que presenta la clase Robot) no se termina de desarrollar correctamente. Por lo tanto, en la mayoría de los casos, el testing devuelve fallas.

```
public void seleccionaItemMozo(String nombreMozo) {
    JComboBox<String> comboMozos = (JComboBox<String>) TestUtils.getComponentForName(vista, "jComboBoxMozos");
    TestUtils.clickComponent(comboMozos, robot);
    TestUtils.tipeaTexto(nombreMozo, robot);
    robot.keyPress(KeyEvent.VK_ENTER);
    TestUtils.getDelay();
    robot.keyRelease(KeyEvent.VK_ENTER);
    //comboMozos.setSelectedItem(nombreMozo);
    TestUtils.getDelay();
}
```

Teóricamente si el robot suelta la tecla "Enter" el comboBox no queda con el Item seleccionado si es que antes se encontraba vacío. Por eso decidimos implementar esa sentencia que está comentada pero tampoco sería lo más correcto.



Se intentó llevar el mouse a la flecha que despliega la lista, para luego seleccionar con el mismo mouse o teclado alguna de las opciones y también se presentaron algunas complicaciones o inconsistencias. Fuera de eso, lo único que le faltaría implementar sería el catcheo de la excepción que se tira si no existe el mozo en la colección.