

Taller de programación I

Trabajo práctico especial 2022

Testing: Alejandro Olave, Mikel; Giacri, Tobias

Integrantes: Alejandro Olave, Mikel; Giacri, Tobias; Nievas, Nahuel;
Vazquez, Imanol

Repositorio Github: <https://github.com/mikexeneize/taller-de-progra1>

Grupo: 7

Denominación de la materia: Taller de Programación.

Profesor a cargo: Guccione Leonel

Introducción	2
Caja Negra	2
Clase: TestConfiguracionDeSistema	2
Clase: TestGestionDePersonal	4
Clase: TestPedidosFacturacionYPromociones	6
Errores encontrados en el test:	9
Observación:	13
Test de cobertura	14
Caja Blanca	15
Cálculo de complejidad ciclomática	17
Test de Integración	19
Test de Persistencia	20
PersistirSistema()	20
DespersistirSistema()	20
Test de Gui	22

Introducción

Para la realización de este trabajo práctico se presentará un grupo con 4 integrantes, divididos en parejas, los cuales deberán:

- Evaluar una SRS, verificarla, validarla y en caso de ser necesario, modificarla.
- Codificar el sistema de a pares, basados en la SRS validada por la totalidad del grupo.
- Una vez finalizada la codificación y documentación, se hará un intercambio de código entre las parejas. Cada una de ellas deberá, a partir de los documentos, armar su tests.

De esta manera se abarcan todos los conceptos aprendidos a lo largo de la asignatura para un mayor entendimiento de la misma y de lo que conlleva realizar este tipo de tareas.

Caja Negra

Las pruebas de caja negra se realizaron utilizando la herramienta JUnit.

Todos los test se encuentran contenidos dentro del package "test". En el repositorio de github.

A partir del Javadoc del programa y en los cuales se probaron diferentes casos de prueba para cada uno de los módulos del sistema. Se encuentran debajo las tablas de particiones y batería de pruebas para cada método que se consideró de importante funcionalidad.

Aclaraciones:

- Usaremos "x" como referencia al parámetro utilizado
- Fuera del Sistema significa que el atributo no pertenece a su vector en el Singleton Sistema

Clase: TestConfiguracionDeSistema

altaMesa		
Atributo	Clases válidas	Clases inválidas
cantComensales	<ul style="list-style-type: none">- $x \geq 2$- $x \neq \text{null}$	<ul style="list-style-type: none">- $x < 2$- $x = \text{null}$
estado	<ul style="list-style-type: none">- $x \neq \text{null}$	<ul style="list-style-type: none">- $x = \text{null}$

ModificarMesa		
Atributo	Clases válidas	Clases inválidas
mesa	- x != null	- x != null (fuera del Sistema) or (x = null)
cantComensales	- x >= 2 - x != null	- x < 2 - x = null
estado	- x != null	- x = null

eliminarMesa		
Atributo	Clases válidas	Clases inválidas
mesa	- x != null	- x = null

altaProducto		
Atributo	Clases válidas	Clases inválidas
nombre	- x != null	- x = null
precioCosto	- x > 0 - x >= precioVenta	- x < 0 - x < precioVenta
precioVenta	- x > 0 - x >= precioCosto	- x < 0 - x < precioCosto
stockInicial	- x > 0	- x < 0

actualizarProducto		
Atributo	Clases válidas	Clases inválidas
producto	- x != null	- x != null (fuera del Sistema) or (x = null)
cantStock	- x >= 0	- x < 0

modificarProducto		
Atributo	Clases válidas	Clases inválidas
id	- x = cualquier entero	-
nombre	- x = cualquier String	-
precioCosto	- x > 0 - x <= precioVenta	- x <= 0 - x > precioVenta
precioVenta	- x > 0 - x >= precioCosto	- x <= 0 - x < precioCosto
cantStock	- x = cualquier entero	-

eliminarProducto		
Atributo	Clases válidas	Clases inválidas
producto	- x != null	- x = null

promedio		
Atributo	Clases válidas	Clases inválidas
mozo	(x != null) and (x en sistema)	x != null (fuera del Sistema) or (x = null)

Clase: TestGestionDePersonal

altaOperario		
Atributo	Clases válidas	Clases inválidas
operario	- x!=null	- x=null
nombre	- x!=null	- x=null
contraseña	- x>5, 12>x, - -x=ContieneUnaMay uscula, - x>=ContieneUnNum ero, - x!=null	- x<=5, x>=13, - x!=ContieneUnaMay uscula, - x!=ContieneUnNum ero, - x=null

modificarOperario		
Atributo	Clases válidas	Clases inválidas
Operario	<ul style="list-style-type: none"> - x!=null - x en sistema 	<ul style="list-style-type: none"> - x=null
nombreUsuario	<ul style="list-style-type: none"> - x!=null, - -x=NoEstaEnSistema 	<ul style="list-style-type: none"> - x=null, - -x!=NoEstaEnSistema
contraseña	<ul style="list-style-type: none"> - x!=null,- - passwordNoCoincideConUsuario 	<ul style="list-style-type: none"> - x=null,
estado	<ul style="list-style-type: none"> - x=true, x!=null 	<ul style="list-style-type: none"> - x=null, x=false

altaOperarioAdministrador		
Atributo	Clases válidas	Clases inválidas
nombre	<ul style="list-style-type: none"> - x!=vacío 	<ul style="list-style-type: none"> - x=vacío
contraseña	<ul style="list-style-type: none"> - x!=vacío 	<ul style="list-style-type: none"> - x=vacío
estado	<ul style="list-style-type: none"> - x=true 	<ul style="list-style-type: none"> - x=null, x=false

modificarMozo		
Atributo	Clases válidas	Clases inválidas
Mozo	<ul style="list-style-type: none"> - x!=null - x en sistema 	<ul style="list-style-type: none"> - x=null
estado	<ul style="list-style-type: none"> - x=true, x!=null 	<ul style="list-style-type: none"> - x=null, x=false

AltaMozo		
Atributo	Clases válidas	Clases inválidas
Mozo	<ul style="list-style-type: none"> - x!=null - x en sistema 	<ul style="list-style-type: none"> - x=null
estado	<ul style="list-style-type: none"> - x=true, x!=null 	<ul style="list-style-type: none"> - x=null, x=false
nombre	<ul style="list-style-type: none"> - x!=null 	<ul style="list-style-type: none"> - x=null
cantHijos	<ul style="list-style-type: none"> - x>=0 	<ul style="list-style-type: none"> - x<0
edad	<ul style="list-style-type: none"> - x>18 	<ul style="list-style-type: none"> - x >= 18

BajaMozo		
Atributo	Clases válidas	Clases inválidas
Mozo	<ul style="list-style-type: none"> - x!=null - x en sistema 	<ul style="list-style-type: none"> - x=null
estado	<ul style="list-style-type: none"> - x=true, x!=null 	<ul style="list-style-type: none"> - x=null, x=false
nombre	<ul style="list-style-type: none"> - x!=null 	<ul style="list-style-type: none"> - x=null
cantHijos	<ul style="list-style-type: none"> - x>=0 	<ul style="list-style-type: none"> - x<0
edad	<ul style="list-style-type: none"> - x>18 	<ul style="list-style-type: none"> - x >= 18

Clase: TestPedidosFacturacionYPromociones

altaComanda		
Atributo	Clases válidas	Clases inválidas
comanda	<ul style="list-style-type: none"> - x!=null - -x en sistema - x.mesa.estado = "libre" - mesa.estado = "libre" para alguna mesa del sistema - Sistema con 2 o mas promociones activas 	<ul style="list-style-type: none"> - x=null - x.mesa.estado != "activo" - mesa.estado = "ocupada" para todas las mesas del sistema - Sistema con menos de 2 promociones activas

bajaComanda		
Atributo	Clases válidas	Clases inválidas
comanda	- x!=null	- x=null

altaPromocion		
Atributo	Clases válidas	Clases inválidas
producto	x!=null	- x=null
diasDePromo	- x!=null - x != vacío	- x=null - x = vacío
aplicaDosporuno	-	-
aplicaDescCantidadMinima	-	-
cantidadMinima	-	-
precioUnitario	-	-

bajaPromocion		
Atributo	Clases válidas	Clases inválidas
promoción	- x!=null	- x=null

altaPedido		
Atributo	Clases válidas	Clases inválidas
comanda	- x!=null - comanda en sistema	- x=null - (x != null and comanda fuera de sistema)
producto	- x!=null	- x=null
cantidad	- x > 0 - x < producto.stock	- x < 0 - x > producto.stock

altaPromocionTemporal		
Atributo	Clases válidas	Clases inválidas
nombre	- x!=null	- x=null
formaDePago	- x!=null	- x=null
porcentajeDesc	- x!=null	- x = null
diasDePromo	- x != vacío	- x = vacío
esAcumulable	-	-

bajaPromocionTemporal		
Atributo	Clases válidas	Clases inválidas
promocionTemporal	- x!=null	- x=null

cerrarComanda		
Atributo	Clases válidas	Clases inválidas
comanda	- x!=null - x.estado = "abierta"	- x=null - x.estado = "cerrada"
metodoDePago	-	-

Errores encontrados en el test:

Se encuentran detallados el código de los test que encontraron errores:
En celeste está remarcado el “Assert” que indico el Failure.

TestModificarProducto_Exitoso

```
@Test
public void TestModificarProducto_Exitoso() {

    Producto producto = Sistema.getInstance().getProductos().get(0);

    int cantStock = 17;
    String nombre = "Fideos de las 2 am";
    float precioCosto = 230;
    float precioVenta = 230;

    try {
        Sistema.getInstance().getZonaConfiguracionDeSistema().modificarProducto(producto.getId(), nombre, precioCosto, precioVenta, cantStock);
    } catch (DatosIncorrectosException e) {
        Assert.fail("No debia lanzarse la excepcion");
    };

    Assert.assertEquals("Error en stock", cantStock, producto.getStock());
    Assert.assertEquals("Error en nombre", nombre, producto.getNombre());
    Assert.assertEquals("Error en precioCosto", precioCosto, producto.getPrecioCosto());
    Assert.assertEquals("Error en precioVenta", precioVenta, producto.getPrecioVenta());
}
```

TestModificarProducto_idNegativo

```
@Test
public void TestModificarProducto_idNegativo() {

    int cantStock = 17;
    String nombre = "Fideos";
    float precioCosto = 230;
    float precioVenta = 240;

    try {
        Sistema.getInstance().getZonaConfiguracionDeSistema().modificarProducto(-999, nombre, precioCosto, precioVenta, cantStock);
    } catch (DatosIncorrectosException e) {
        Assert.fail("Debia lanzarse la excepcion");
    };
}
```

testLocal

```
@Test
void testLocal() {
    Local local=new Local("Pepe's pizza",(float) 542);
    Assert.assertEquals("Error Nombre","Pepe's pizza",local.getNombre());
    Assert.assertEquals("Error Sueldo", (float)542, (float)local.getSueldo());
}
```

testSetSueldo

```
@Test
void testSetSueldo() {
    Local local=new Local("Pepe's pizza",(float) 542);
    local.setSueldo((float) 200);
    Assert.assertEquals("Error Sueldo", (float)200, (float)local.getSueldo());
}
```

testMozo

```
@Test
void testMozo() {
    Mozo mozo=new Mozo(2,"activo",10.5,"Pepe Lopez",new Date(1/12/1945));
    Assert.assertEquals("Error cantidad de hijos",2,mozo.getCantHijos());
    Assert.assertEquals("Error en estado del mozo", "activo",mozo.getEstado());
    Assert.assertEquals("error en sueldo", 10.5, mozo.getSueldo());
    Assert.assertEquals("error en nombre y apellido", "Pepe Lopez",mozo.getNombreYApellido());
    Assert.assertEquals("Error en la fecha", new Date(1/12/1945),mozo.getNacimiento());
}
```

testSetAcumulado

```
@Test
void testSetAcumulado() {
    Mozo mozo=new Mozo(2,"activo",10.5,"Pepe Lopez",new Date(1/12/1945));
    mozo.setAcumulado(5);
    Assert.assertEquals("Error cantidad de hijos",5,mozo.getAcumulado());
}
```

testSetSueldo

```
@Test
void testSetSueldo() {
    Mozo mozo=new Mozo(2,"activo",10.5,"Pepe Lopez",new Date(1/12/1945));
    mozo.setSueldo(5000);
    Assert.assertEquals("Error sueldo",5000,mozo.getSueldo());
}
```

TestAltaComanda_Exitoso

```
@Test
public void TestAltaComanda_Exitoso() {

    ArrayList<Pedido> pedidos1 = new ArrayList<Pedido>();
    pedidos1.add(new Pedido());
    Comanda comanda1 = new Comanda(new Date(), Sistema.getInstance().getMesas().get(0), pedidos1);
    try {
        Sistema.getInstance().getZonaPedidosYFacturacion().altaComanda(comanda1);
    } catch (DatosIncorrectosException e) {
        Assert.fail("No debio lanzarse excepcion");
    }

    ArrayList<Comanda> comandas = Sistema.getInstance().getComandas();

    Assert.assertEquals("Error en comanda", true, comandas.contains(comanda1));
}
```

testProducto

```
@Test
void testProducto() {
    Producto producto=new Producto("Papas Fritas",1,6,200);
    Assert.assertEquals("Error nombre producto","Papas Fritas",producto.getNombre());
    Assert.assertEquals("Error en precio costo",1,producto.getPrecioCosto());
    Assert.assertEquals("error en precio venta",6,producto.getPrecioVenta());
    Assert.assertEquals("error en Stock",200,producto.getStock());
}
```

testSetPrecioCosto

```
@Test
void testSetPrecioCosto() {
    Producto producto=new Producto("Papas Fritas",1,6,200);
    producto.setPrecioCosto(4);
    Assert.assertEquals("Error en el precio costo",4,producto.getPrecioCosto());
}
```

testSetPrecioVenta

```
@Test
void testSetPrecioVenta() {
    Producto producto=new Producto("Papas Fritas",1,6,200);
    producto.setPrecioVenta(8);
    Assert.assertEquals("Error en el precio venta",8,producto.getPrecioVenta());
}
```

testPromocionTemporal

```
@Test
public void testPromocionTemporal() {
    int id = 0;
    String nombre = "nombre";
    String formaDePago = "Forma";
    int porcentajeDescuento = 20;
    ArrayList<String> diasPromocion = new ArrayList<String>();
    boolean activo = true;
    boolean acumulable = true;

    PromocionTemporal promocionTemporal = new PromocionTemporal(id, nombre, formaDePago, porcentajeDescuento,
diasPromocion, activo, acumulable);

    Assert.assertEquals("Error en id", id, promocionTemporal.getId());
    Assert.assertEquals("Error en nombre", nombre, promocionTemporal.getNombre());
    Assert.assertEquals("Error en formaDePago", formaDePago, promocionTemporal.getFormaDePago());
    Assert.assertEquals("Error en porcentajeDescuento", porcentajeDescuento, promocionTemporal.getPorcentajeDescuento());
    Assert.assertEquals("Error en diasPromocion", diasPromocion, promocionTemporal.getDiasPromocion());
    Assert.assertEquals("Error en activo", activo, promocionTemporal.isActive());
    Assert.assertEquals("Error en acumulable", acumulable, promocionTemporal.isAcumulable());
}
```

testPedido

```
@Test
void testPedido() {
    Producto producto= new Producto("papas",2,23,100);
    Pedido pedido=new Pedido(2,new Date(14/11/2022),4,producto);
    Assert.assertEquals("Error nro id",2,pedido.getId());
    Assert.assertEquals("Error en fecha", new Date(14/11/2022),pedido.getFecha());
    Assert.assertEquals("error en cantidad",4,pedido.getCantidad());
    Assert.assertEquals("error en producto",producto,pedido.getProducto());
}
```

testPromocion

```
@Test
public void testPromocion() {
    Producto producto = new Producto("nombre", 0, 0, 0);
    ArrayList<String> diasPromocion = new ArrayList<String>();
    boolean aplicaDosXUno = true;
    boolean aplicaDescXCantMin = true;
    int descXCantMin = 0;
    float descXCantXPrecioUnitario = 0;
    boolean activa = true;

    Promocion promocion = new Promocion(producto, diasPromocion, aplicaDosXUno,
    aplicaDescXCantMin, descXCantMin, descXCantXPrecioUnitario, activa);

    Assert.assertEquals("Error en producto", producto, promocion.getProducto());
    Assert.assertEquals("Error en diasPromocion", diasPromocion, promocion.getDiasPromocion());
    Assert.assertEquals("Error en aplicaDosXUno", aplicaDosXUno, promocion.isAplicaDosXUno());
    Assert.assertEquals("Error en aplicaDescXCantMin", aplicaDescXCantMin, promocion.isAplicaDescXCantMin());
    Assert.assertEquals("Error en descXCantMin", descXCantMin, promocion.getDescXCantMin());
    Assert.assertEquals("Error en descXCantXPrecioUnitario", descXCantXPrecioUnitario, promocion.getDescXCantXPrecioUnitario());
    Assert.assertEquals("Error en activa", activa, promocion.isActive());
}
```

TestSetDescXCantXPrecioUnitario

```
@Test
public void TestSetDescXCantXPrecioUnitario() {
    Promocion promocion = new Promocion();
    promocion.setDescXCantXPrecioUnitario(5);
    Assert.assertEquals("Error en descXCantXPrecioUnitario", 5, promocion.getDescXCantXPrecioUnitario());
}
```

testFactura

```
@Test
void testFactura() {
    Pedido pedido=new Pedido(5,new Date(14/03/1965),3,new Producto("empanada",5,10,600));
    Mesa mesa =new Mesa(5,"activo");
    ArrayList<Pedido> pedidos=new ArrayList<Pedido>();
    pedidos.add(pedido);
    Comanda comanda=new Comanda(new Date(28/07/1914),new Mesa(5,"activo"),pedidos);
    Factura factura=new Factura(1,new Date(28/07/1914),mesa,pedidos,(float)5000,"Efectivo",new ArrayList<PromocionPadre>());
    Assert.assertEquals("Error en id Factura",1,factura.getId());
    Assert.assertEquals("Error en la fecha",new Date(28/07/1914),factura.getFecha());
    Assert.assertEquals("Error en la mesa",mesa,factura.getMesa());
    Assert.assertEquals("Error en la pedidos",pedidos,factura.getProductos());
    Assert.assertEquals("Error en el total",(float)5000,factura.getTotal());
    Assert.assertEquals("Error en el metodo de pago","Efectivo",factura.getMetodoDePago());
    Assert.assertEquals("Error en la promociones",new ArrayList<PromocionPadre>(),factura.getPromociones());
}
```

testSetTotal

```
@Test
void testSetTotal() {
    Pedido pedido=new Pedido(5,new Date(14/03/1965),3,new Producto("empanada",5,10,600));
    Pedido pedido1=new Pedido(2,new Date(14/03/1965),3,new Producto("empanada de carne",5,10,600));
    Mesa mesa =new Mesa(5,"activo");
    Mesa mesa1 =new Mesa(9,"activo");
    ArrayList<Pedido> pedidos=new ArrayList<Pedido>();
    pedidos.add(pedido);
    Comanda comanda=new Comanda(new Date(28/07/1914),new Mesa(5,"activo"),pedidos);
    Factura factura=new Factura(1,new Date(28/07/1914),mesa,pedidos,(float)5000,"Efectivo",new ArrayList<PromocionPadre>());
    factura.setTotal((float)666);
    Assert.assertEquals("Error en el total", (float)666, factura.getTotal());
}
```

TestModificarOperario_nombreUsuarioInexistente

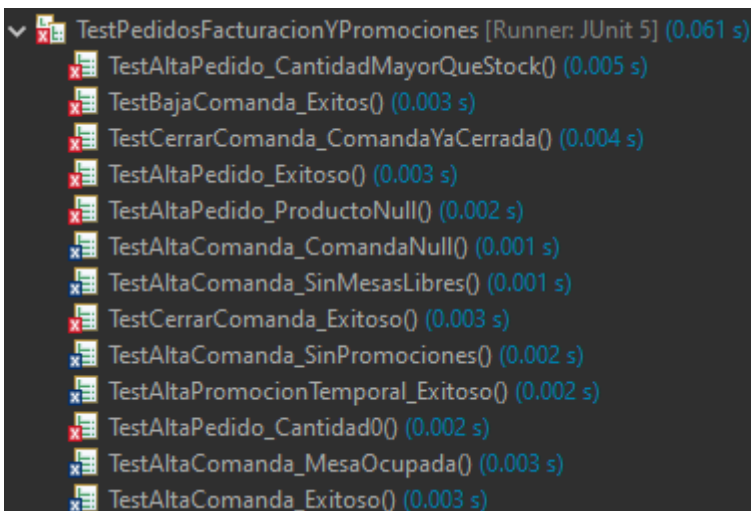
```
@Test
public void TestModificarOperario_nombreUsuarioInexistente() {
    String nombreUsuarioAnterior = "noexiste";
    String nombreUsuario = "nuevo";
    String password = "adaDS89";
    boolean activo = false;

    try {
        Sistema.getInstance().getZonaGestionDePersonal().modificacionOperario(nombreUsuarioAnterior, nombreUsuario,
password, activo);
        Assert.fail("Debio lanzarse excepcion");
    } catch (DatosIncorrectosException e) {
    }
}
```

TestBajaMozo_Exitoso

```
@Test
public void TestBajaMozo_Exitoso() {
    Mozo mozo = Sistema.getInstance().getMozos().get(0);
    try {
        Sistema.getInstance().getZonaGestionDePersonal().bajaMozo(mozo);
    } catch (DatosIncorrectosException e) {
        Assert.fail("No Debio lanzarse excepcion");
    }
    Assert.assertEquals("Error en nombre", true, Sistema.getInstance().getMozos().contains(mozo));
}
```

Observación:



```
TestPedidosFacturacionYPromociones [Runner: JUnit 5] (0.061 s)
  TestAltaPedido_CantidadMayorQueStock() (0.005 s)
  TestBajaComanda_Exitoso() (0.003 s)
  TestCerrarComanda_ComandaYaCerrada() (0.004 s)
  TestAltaPedido_Exitoso() (0.003 s)
  TestAltaPedido_ProductoNull() (0.002 s)
  TestAltaComanda_ComandaNull() (0.001 s)
  TestAltaComanda_SinMesasLibres() (0.001 s)
  TestCerrarComanda_Exitoso() (0.003 s)
  TestAltaComanda_SinPromociones() (0.002 s)
  TestAltaPromocionTemporal_Exitoso() (0.002 s)
  TestAltaPedido_Cantidad0() (0.002 s)
  TestAltaComanda_MesaOcupada() (0.003 s)
  TestAltaComanda_Exitoso() (0.003 s)
```

Al ejecutar un test de todo el package entero, 7 métodos de la TestPedidosFacturacionYPromociones clase dan error, pero al ejecutarlos individualmente pasan el test de unidad.

Se sospecha que se debe a un error con configuración, que impide ejecutar el método con el decorator @Before o @BeforeAfter.

Test de cobertura

Con respecto al test de cobertura, este nos dio un 35% que es muy bajo. Pero analizando el porqué, nos dimos cuenta que tanto los controladores como las ventanas casi no eran recorridas durante la caja negra. Esto tiene sentido, ya que para testear las mismas se debe hacer mediante un test de interfaces (gui), que se verá más adelante durante este informe. Volviendo a lo que a nosotros nos interesaba testear en esta etapa que era la capa de negocios, este nos dio un 51% que nos seguía pareciendo bajo. Al ver el código de por donde pasaba el test de cobertura, vimos que en PedidosFacturacionYPromociones.java, existían errores de codificación en determinados métodos de la clase, por esto el test unitario no pudo cumplir su cometido. Ya que la ejecución de dicho método era necesario para el cumplimiento de las precondiciones de otros métodos, decide hacerle a uno de estos métodos un test de caja blanca. En Sistema.java, ConfiguracionDeSistema.java y GestionDePersonal.java la cobertura no llega al 100% o cerca porque se agregan setters, getters y toString que son utilizados solamente por los controladores de las vistas para poder mostrar cierta información y no son testeadas en esta etapa.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼ tallerDeProgra	35.2 %	7,204	13,247	20,451
▼ src/main/java	20.0 %	2,985	11,976	14,961
> com.tallerDeProgra.tallerDeProgra	0.0 %	0	219	219
> com.tallerDeProgra.capaDePresentacion.Controladores	3.6 %	93	2,461	2,554
> com.tallerDeProgra.capaDePresentacion.Vistas	10.7 %	910	7,560	8,470
> persistencia	24.3 %	91	284	375
> com.tallerDeProgra.enums	39.4 %	122	188	310
▼ com.tallerDeProgra.capaDeNegocios	51.1 %	1,080	1,034	2,114
> PedidosFacturacionYPromociones.java	32.0 %	254	539	793
> Sistema.java	54.9 %	263	216	479
> ConfiguracionDeSistema.java	63.8 %	285	162	447
> GestionDePersonal.java	70.4 %	278	117	395
> excepciones	65.2 %	15	8	23
> com.tallerDeProgra.capaDeDatos	75.2 %	674	222	896
> src/test/java	76.8 %	4,219	1,271	5,490

Caja Blanca

Para el test de caja blanca se decidió utilizar el método `altaComanda()` de la clase `PedidosFacturacionYPromociones.java`. Ya que si este se hacía bien, muchos de los que no son alcanzados por la cobertura podrían ser alcanzados. A continuación se muestra el código.

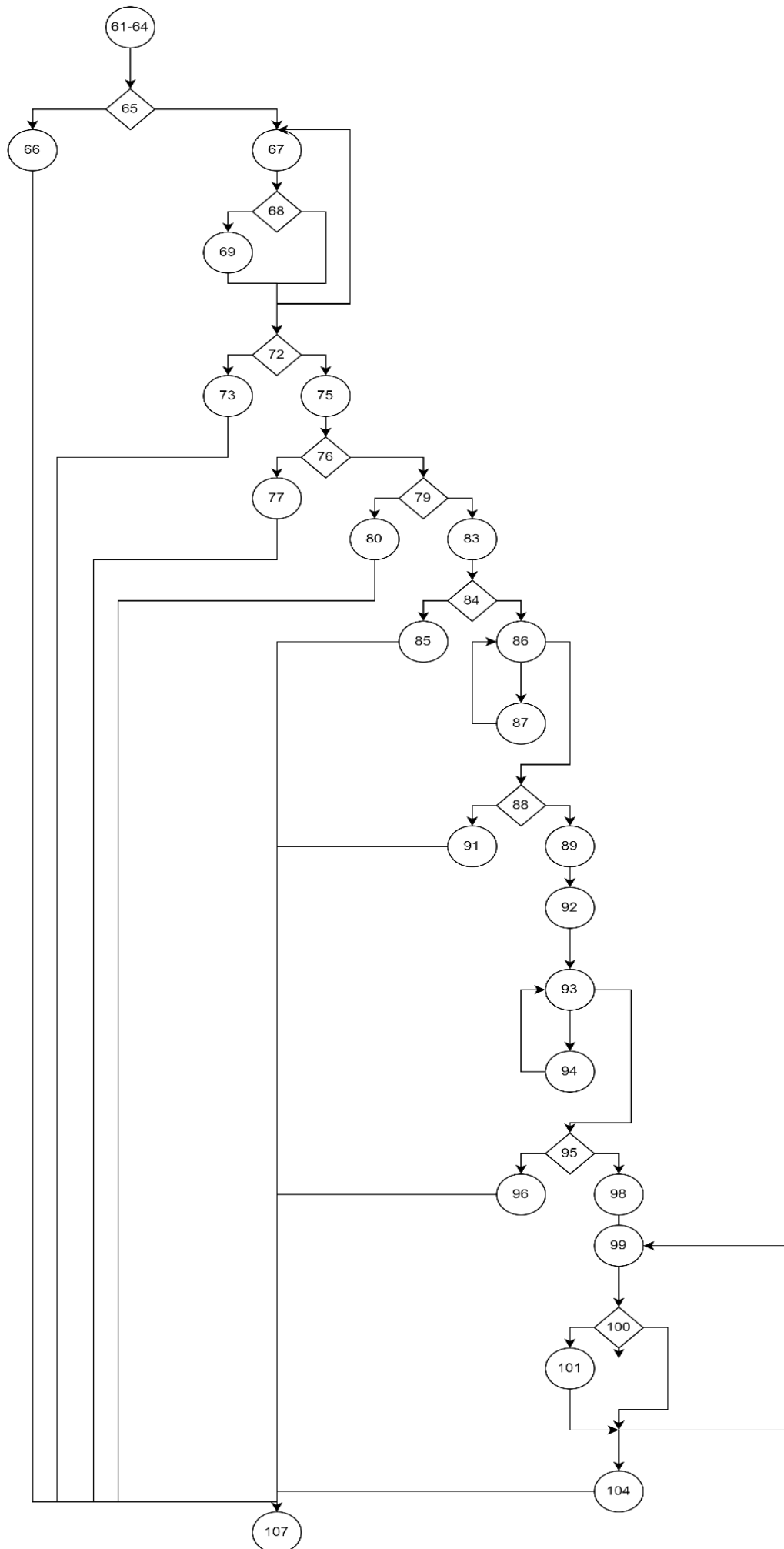
```
/**
 * ADESEA UNA COMANDA AL SISTEMA
 * @param comanda :Parametro que contiene la comanda a ingresar al sistema
 * @throws DatosIncorrectosException :Se lanza si la comanda es nula
 * post: <br>
 * - Se ADESEA UNA NUEVA COMANDA<br>
 * - LA MESA CORRESPONDIENTE SE OCUPA
 */
public void altaComanda(Comanda comanda) throws DatosIncorrectosException {
    Mozo mozoAsociado;
    int i=0;
    boolean hayMesaLibre=false;
    Producto producto;
    if(comanda==null)
        throw new DatosIncorrectosException("Parametros invalidos.");
    for(Mesa mesa: Sistema.getInstance().getMesas()) {
        if(mesa.getEstado().equals(EstadosMesa.LIBRE.getEstado())){
            hayMesaLibre=true;
        }
    }
    if(!hayMesaLibre)
        throw new DatosIncorrectosException("No hay mesas libres en el restaurante");

    mozoAsociado =Sistema.getInstance().getAsignacionesDesdeMesa().get(comanda.getMesa().getId()).getMozo();
    if(comanda.getMesa().getEstado().equals(EstadosMesa.OCUPADA.getEstado()) )
        throw new DatosIncorrectosException("La mesa esta ocupada, debe elegir una mesa activa.");

    if( mozoAsociado==null || !mozoAsociado.getEstado().equals(Estado.ACTIVO.getEstado()))
        throw new DatosIncorrectosException("El mozo asociado no existe o no se encuentra disponible.");

    /*Se usa para validar que haya al menos dos promociones activas*/
    ArrayList<Promocion> promociones=Sistema.getInstance().getPromociones();
    if(promociones==null)
        throw new DatosIncorrectosException("No hay promociones activas.");
    while(i<promociones.size() && promociones.get(i).isActiva())
        i++;
    if(i<promociones.size())
        producto=promociones.get(i).getProducto();
    else
        throw new DatosIncorrectosException("No hay promociones activas.");
    i=0;
    while(i<promociones.size() && promociones.get(i).isActiva() && producto.getId()!=promociones.get(i).getProducto().getId())
        i++;
    if(!i<promociones.size())
        throw new DatosIncorrectosException("No hay suficientes promociones activas. Se necesitan al menos dos promociones de dos productos distintos");

    ArrayList<Mesa> mesas= Sistema.getInstance().getMesas();
    for(Mesa mesa: mesas)
        if(comanda.getMesa().getId()== mesa.getId())
            mesa.setEstado(EstadosMesa.OCUPADA.getEstado());
    Sistema.getInstance().getComandas().add(comanda);
}
```

Cálculo de complejidad ciclomática

$V(G) = \text{nodos} - 1 = 13 - 1 = 12$

C1= 61...64-65-66-107

C2= 61...64-65-67-68-69-72-73-107

C3= 61...64-65-67-68-69-67-68-72-73-107

C4= 61...64-65-67-68-72-75-76-77-107

C5= 61...64-65-67-68-72-75-76-79-80-107

C6= 61...64-65-67-68-72-75-76-79-83-84-85-107

C7= 61...64-65-67-68-72-75-76-79-83-84-86-87-86-88-91-107

C8= 61...64-65-67-68-72-75-76-79-83-84-86-88-91-107

C9= 61...64-65-67-68-72-75-76-79-83-84-86-88-89-92-93-95-96-107

C10= 61...64-65-67-68-72-75-76-79-83-84-86-88-89-92-93-95-98-99-100-101-99-104-107

C11= 61...64-65-67-68-72-75-76-79-83-84-86-88-89-92-93-95-98-99-100-101-104-107

C12= 61...64-65-67-68-72-75-76-79-83-84-86-88-89-92-93-95-98-99-100-104-107

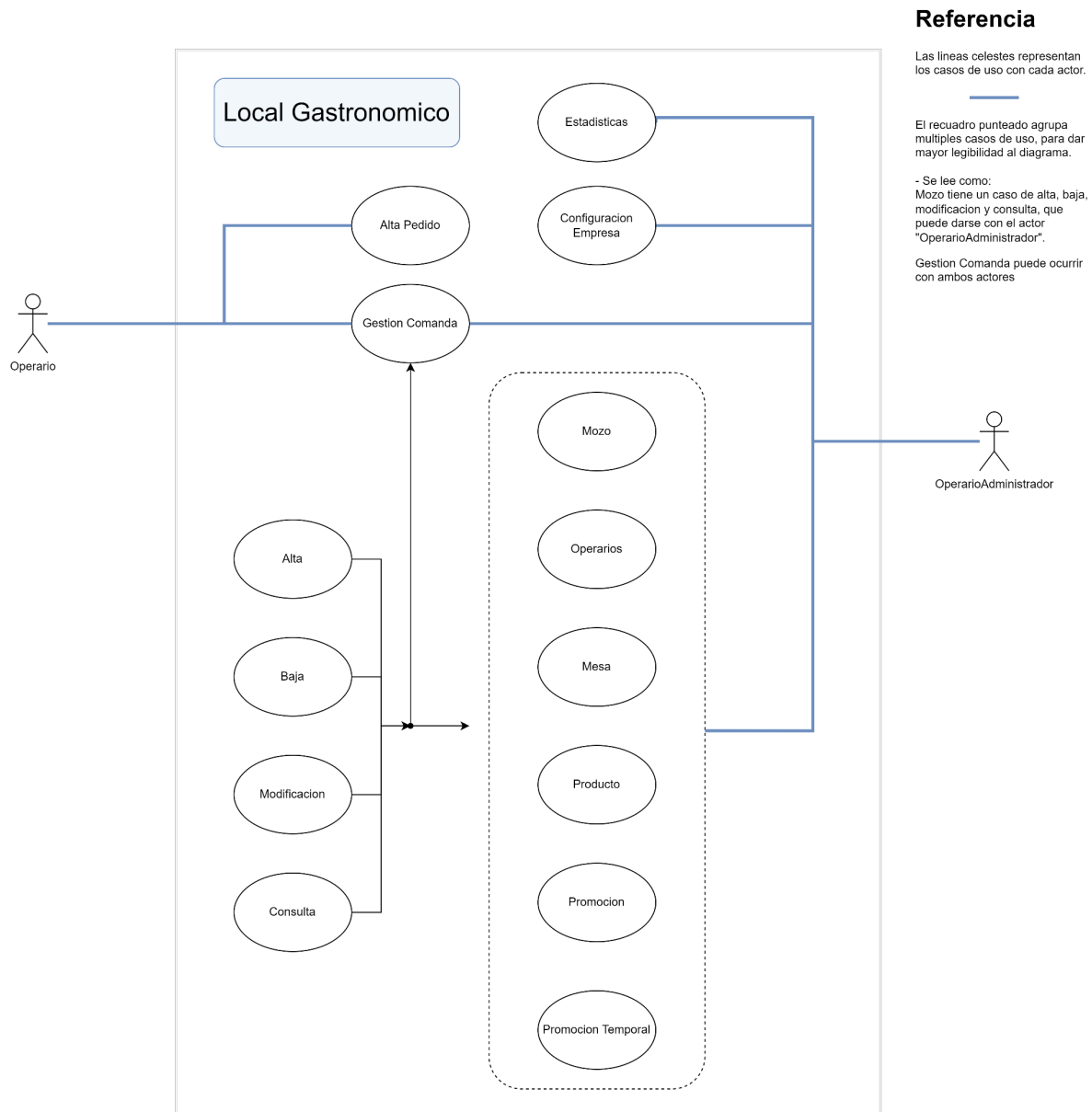
Casos:

Camino	Parámetro de Entrada	Salida Esperada
C1	Comanda==null	DatosIncorrectosException("Parametros inválidos.")
C2	Comanda!=null mesas.estado==ocupado	DatosIncorrectosException("No hay mesas libres en el restaurante");
C3	Comanda!=null mesas.estado==ocupado	DatosIncorrectosException("No hay mesas libres en el restaurante");
C4	Comanda != null mesa.estado == ocupado	DatosIncorrectosException("La mesa está ocupada, debe elegir una mesa activa.");
C5	Comanda != null mozo.estado == ocupado	DatosIncorrectosException("El mozo asociado no existe o no se encuentra disponible.");
C6	Comanda != null mozo.estado == libre promoción==null	DatosIncorrectosException("No hay promociones activas.");
C7	Comanda != null mozo.estado == libre promoción==null	DatosIncorrectosException("No hay promociones activas.");
C8	Comanda != null mozo.estado == libre promoción==null	DatosIncorrectosException("No hay promociones activas.");

C9	Comanda != null mozo.estado == libre promoción!=null	DatosIncorrectosException("No hay suficientes promociones activas. Se necesitan al menos dos promociones de dos productos distintos");
C10	Comanda != null mozo.estado == libre promoción!=null	alta comanda éxito
C11	Comanda != null mozo.estado == libre promoción!=null	alta comanda éxito
C12	Comanda != null mozo.estado == libre promoción!=null	alta comanda éxito

Test de Integración

Realizamos una integración ascendente, testeando las clases más atómicas primero y avanzando hacia las más abstractas. Se consideraron los procesos o funcionalidad del sistema en el siguiente grafo a modo de análisis e interpretación:



En el grafo se muestran las funcionalidades principales del sistema, las cuales fueron testeadas con escenarios con y sin datos. Como se testeo de forma ascendente, no fue necesario la utilización de mocks, ya que los métodos de subclases llamados en la clase Sistema ya fueron probados y se pudieron utilizar para testearla. Para lograr esto, se crearon distintas suitcase donde primero se testean los métodos más atómicos, y luego los más abstractos.

Test de Persistencia

Para este test se realizaron las pruebas en base al local en general, en este caso llamado Sistema, probando los casos de persistir, despersistir, etc. del objeto Sistema. El sistema es único y por lo tanto se aplicó el patrón Singleton, lo cual nos indujo a tener que utilizar un objeto de tipo DTO para lograr persistirlo. Se realizaron los métodos test correspondientes para verificar si el funcionamiento es el correcto. A continuación se visualizan las tablas de datos de prueba.

PersistirSistema()

Escenario

Nro Escenario	Descripción
1	No existe el archivo o se pisa el archivo ya escrito

Tabla de particiones

Condición	Clase válidas	Clase Inválidas
SistemaDTO	Sistema DTO != null (1)	-

Batería de pruebas

Tipo de Clase	Escenario	Valores de entrada	Salida esperada	Clases de prueba correcta
Correcta	1	SistemaDTO != null	Sistema persistido	1

DespersistirSistema()

Escenario

Nro Escenario	Descripción
1	Archivo existente
2	Archivo inexistente

Tabla de Particiones

Condición	Clases válidas	Clases inválidas
Estado del archivo	Existente(1), Vacío(3)	Inexistente(2)

Batería de pruebas

Tipo de clase	Escenario	Valor de entrada	Salida Esperada	Clases Cubiertas
Correcta	1	-	Retorna un Data Transfer Object al sistema, que puede estar vacío o no	1,3
Incorrecta	2	-	IOException	2

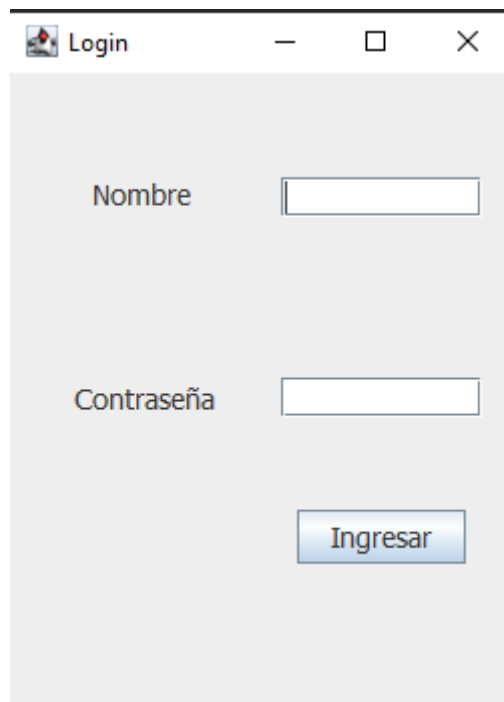
Test de Gui

Con respecto al testeo de la ventana, se realizaron las pruebas sobre la Ventana de ingreso. Se eligió esta ventana, ya que era una de las principales por ser la primera que aparece en el sistema. Además el uso de componentes como los textField complejizan su testeo. El proceso a probar es el de ingresar un usuario operario. Para esto, se procedió a probar diferentes casos tales como:

- Nombre de usuario y contraseña correctos.
- Nombre de usuario correcto y contraseña incorrecta.
- Nombre de usuario correcto y contraseña vacía.
- Nombre de usuario incorrecto
- Nombre de usuario vacío
- Disponibilidad del botón Ingresar al estar los campos en vacíos.

La mayoría de las pruebas fueron exitosas, al ejecutarlas se pudo observar que todo caso (Mensajes de error emergentes) se resuelve de la manera esperada. La única que no fue exitosa fue la de la habilitación de botón. Por lo visto en los test, el botón siempre se encuentra habilitado. Se utilizó la clase Robot de JAVA, y una clase TestUtilVentana que se basa de la clase TestUtil entregada por la cátedra para definir las funciones del robot y así poder utilizarlas repetidas veces.

Se podrá observar al robot en funcionamiento en el repositorio de github.



Vista que fue testeada.