PLAN DE GESTIÓN DE PRUEBAS

Javier Santana Delgado
Jose Antonio Santacruz Gallego
Tamara Redondo Soto
Julio Sánchez de las Heras Martín Consuegra
Alejandro Riquelme Castaño
José Antonio Oliver González-Ortega

PLAN DE GESTIÓN DE PRUEBAS

PLAN DE GESTIÓN DE PRUEBAS

DOCUMENTACIÓN DE PRUEBAS

En este documento se mostrarán algunas **pruebas realizadas de las distintas iteraciones** que encontramos en nuestro proyecto. Estas han sido realizadas en cada una de las capas definidas en el **patrón MVC (Presentación, Dominio y Persistencia).** En este documento nos centraremos en la **capa de dominio** en la que se encuentran clases junto con sus DTO y detallaremos solo estos últimos para no extendernos demasiado ya que en general las pruebas son muy parecidas para todos los DTO. Al final también mostramos una **pequeña prueba** realizada sobre el agente.

DTOCamarero

Se crearán un **DTOCamarero** y **un agente** los cuales serán inicializados en el apartado **BeforeClass** que nos serán útiles para utilizar los métodos del DTO y realizar las consultas a la base de datos con el agente. También se insertan datos de prueba en la **base de datos,** específicamente en la tabla Camarero y Aviso.

```
@BeforeClass
public static void SetUpBeforeClass() {
        dtoCamarero = new DTOCamarero();
        agente = new Agente();
        try {
            agente.Insert("INSERT INTO Camarero
(idCamarero, nombre) VALUES (100,'prueba')");
            agente.Insert("INSERT INTO Aviso
(idAviso, descripcion, idCamarero, turno) "
                    + "VALUES (900, 'test',100,
'"+DTOReserva.obtenerTurno()+"')");
        } catch (SQLException e) {
            e.printStackTrace();
        }
}
```

Luego comprobamos el método **LeerCamareros del DTOCamarero** en el que al leer los camareros de la base de datos estos se guardan en la lista **listaAux** y con **assertNotEquals** comprobamos que la lista que contiene los camareros no tenga un tamaño igual a o puesto que en el **BeforeClass** hemos añadido un camarero por lo que al menos debe haber uno.

```
@Test
public void testLeerCamareros() {
        LinkedList<Camarero>listaAux=new
LinkedList<Camarero>();
        dtoCamarero.leerCamareros(listaAux);
        int actual=listaAux.size();
        int expected=0;
        assertNotEquals(expected, actual);
}
```

DTOAlimento

Creamos un **agente y un objeto DTOAlimento** y en el apartado **BeforeClass** introducimos datos de prueba como hemos comentado anteriormente.

```
@BeforeClass
public static void SetUpBeforeClass() throws SQLException
{
        dtoAl = new DTOAlimento();
        ag.Insert("INSERT INTO Carta (codigo, nombre,
tipo, precio) VALUES (100, 'Croqueta', 'Entrante', 5);");
        ag.Insert("INSERT INTO Carta (codigo, nombre,
tipo, precio) VALUES (101, 'Jamon', 'Entrante', 5);");
}
```

A continuación comprobaremos los **alimentos según su tipo**, en esta ocasión buscaremos el tipo de alimento que sea un "Entrante" y con **assertNotEquals** comprobaremos que la lista que guarda estos alimentos buscados no tiene un tamaño igual a o al haber introducido dos entrantes en **el BeforeClass.**

```
@Test
public void testLeerNombreAlimentosPorTipo() {
        LinkedList<String> al = new
LinkedList<String>();
        dtoAl.leerNombreAlimentosPorTipo(al,
"Entrante");
        assertNotEquals(al.size(), 0);
}
```

DTOPrevisionTest

En este DTO crearemos los objetos de tipo **DateTimeFormatter, DTOPrevision, Agente, LocalDateTime y ExpectecException.** Especificamos el **BeforeClass** con los datos de prueba.

```
@BeforeClass
public static void SetUpBeforeClass() {
       pruebaTime= LocalDateTime.parse("2020:12:15
14:30", df);
        dtoPrev=new DTOPrevision();
        try {
            agente.Insert("INSERT INTO Ingrediente
(nombre, stock) VALUES('test',5)");
            agente.Insert("INSERT INTO Ingrediente
(nombre, stock) VALUES('test2',5)");
            agente.Insert("INSERT INTO Ingrediente
(nombre, stock) VALUES('test3',5)");
            agente.Insert("INSERT INTO Prevision
(ingrediente, cantidad, fecha) VALUES('test3'"
                    + ",5, '2018-12-12')");
        } catch (SQLException e) {
            e.printStackTrace();
        }
}
```

Para probarlo realizaremos una prueba para tratar de que falle y con esto comprobaremos que lo realiza correctamente. El fallo se produce al **intentar leer de la base de datos un ingrediente con el nombre fail ya que este no existe.**

```
@test
public void testLeerStockException() {
        expectedException.expect(SQLException.class);
        dtoPrev.leerStock("fail");
}
```

DTOCuentaTest

Crearemos los objetos **DateTimeFormatter, ExpectedException, LocalDateTime, DTOCuenta, Agente.** Luego las asignaremos en el **BeforeClass** y luego con esto realizaremos las pruebas correspondientes.

```
    @BeforeClass
    public static void SetUpBeforeClass() {
        pruebaTime=DTOReserva.obtenerTurno();
        dtoCuenta=new DTOCuenta();
        agente=new Agente();
        try {
            agente.Insert("INSERT INTO Mesa (idMesa,
estado) VALUES (900, 'ocupada')");
            agente.Insert("INSERT INTO Comanda (idComanda,
idMesa, turno) VALUES(900, 900, '"+pruebaTime+"')");
            agente.Insert("INSERT INTO Carta (codigo,
nombre, tipo, precio) VALUES (900, 'test', 'Postre', 9)");
            agente.Insert("INSERT INTO Pedido (comanda,
codigo, cantidad) VALUES (900, 900, 2)");
        } catch (SQLException e) {
            e.printStackTrace();
        }
}
```

Una prueba de todas las que hemos realizado sería buscar el nombre de un **postre** en una cuenta que tenga un **id de mesa 900** y cuyo nombre es **"test"** para que no haya problemas con la base de datos.

```
@Test
public void testDevolverCuenta() {
        String
actual=dtoCuenta.devolverCuenta("900").getPostre().ge
t(0).getNombre();
        String expected="test";
        assertEquals(expected, actual);
}
```

DTOEstadisticaTest

En este **último DTO** que se encuentra en **la iteración 5,** solo crearemos el objeto **Agente** para utilizar la **sentencia SQL** definida en el **BeforeClass** de la siguiente forma:

```
    @BeforeClass
    public static void setUpBeforeClass() throws SQLException {
        String consulta = "INSERT into Reserva (idReserva,
num_comensales, nombre, tiempoLibre,"
                + " tiempoReservada, tiempoOcupada,
tiempoPidiendo,"
                + " tiempoEnEsperaComida, tiempoServido,
tiempoEsperandoCuenta,"
                + " tiempoPagando, tiempoEnPreparacion,
restaurante) VALUES (250, 6, 'Carlos',"
                + " '2020-12-18 15:41:00', '2020-12-17
15:00:00', '2020-12-18 14:30:00', '2020-12-18 14:33:00',"
                + " '2020-12-18 14:35:00', '2020-12-18
14:45:00', '2020-12-18 15:30:00', '2020-12-18 15:37:00',"
                + " '2020-12-18 15:40:00', 2)";
        ag.Insert(consulta);
}
```

A continuación se calculará el tiempo que tarda en la toma de comandas, por ejemplo comprobaremos el tiempo que tarda en promedio una mesa con 6 comensales en un restaurante de Ciudad Real que tiene el id 2 en la base de datos.

```
        @Test
        public void calcularTiempoTomaComandasRestauranteTest(){
             assertNotEquals(0, (int)
DTOEstadistica.calcularTiempoTomaComandasRestaurante("6", "2"));
}
```

AgenteTest

Por último se comentará las pruebas realizadas al **agente,** para esto crearemos un **objeto agente** y directamente realizaremos los **test** como por ejemplo insertar una bebida con unos valores dados.

```
        @Test
        public void CreateTest() throws SQLException {
            ag.Insert("INSERT INTO Bebida (nombre, codigo,
    stock) VALUES ('Nestea', 7, 14);");
        }
```

Al igual que en anteriores tests, también se han realizado **casos de prueba negativos** para comprobar cómo se comporta el agente cuando realiza **algunas sentencias SQL** con sintaxis errónea o con columnas que no existen en la base de datos. Así, tendremos por ejemplo:

```
    @Test
    public void UpdateExceptionTest() {
        expectedException.expect(SQLException.class);
        ag.Update("UPDATE Ingrediente SET stock=8 WHERE fail =
'Fanta';");
    }
```

Donde se comprueba el método **Update del Agente** introduciendo una **consulta SQL** la cual hace referencia a una columna de la tabla Ingrediente que no existe, capturando consecuentemente la **excepción SQLException.**