Pruebas Realizadas Con JUnit 4 Al Proyecto de Panadería

Rodríguez Correa Juan Sebastián

Universidad de San Buenaventura, Sede Bogotá.

Facultad de Ingeniería.

Programa de Sistemas

Bogotá, Colombia

2020

Pruebas unitarias realizadas al proyecto propuesto sobre la panadería

Este documento tiene como fin mostrar de forma ordenada y precisa el proceso que se llevó a cabo sobre el proyecto de la panadería, el cual fue previamente establecido para solución de otro trabajo asignado en la materia de Análisis y Diseño de Algoritmos en la universidad de San Buenaventura de la sede de Bogotá D.C. Para ello se utilizó un Framework llamado JUnit, este se especializa en la realización de pruebas unitarias sobre softwares, en relación con este proyecto se utilizó la versión del JUnit 4, puesto que, el desarrollo del proyecto se encuentra en el IDE de NetBeans 8.2, en consecuencia, a ello, se decidió usar la herramienta de JUnit en su versión 4, evitando la descarga de plugins y de esa forma usar el ya proporcionado por el IDE previamente nombrado.

Posteriormente se dará comienzo al desarrollo de las pruebas y a su correspondiente análisis.

Testeo de conexión a la base de datos de PostgreSQL – 12

Como primera prueba unitaria que se desarrollará sobre el proyecto, se tiene en consideración la más importante para que el programa funcione de forma correcta, esta corresponde a la conexión y comunicación que tendrá el servidor cliente de PostgreSQL en su versión 12 con el proyecto diseñado en lenguaje Java, para comprobar que dicha relación se construye de forma correcta, se realizó el siguiente testeo con la ayuda de la herramienta de JUnit 4 suministrada por IDE de NetBeans 8.2. Una demostración de los anterior se evidenciará a continuación.

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Tests;

import co.edu.usbbog.ADA.ProjectPanaderia.bo.ProductoBO;
import co.edu.usbbog.ADA.ProjectPanaderia.db.Conexion;
import java.sql.Connection;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Juan Sebastian
 */
public class TestAfterEach {

    Conexion co;

    public TestAfterEach() {
        this.co = new Conexion();
    }
```

```
25
26      public TestAfterEach() {
27          this.co = new Conexion();
28      }
29
30      @BeforeClass
31      public static void setUpClass() {
32          System.out.println("----------------- Se van a ejecutar los Testeos ---------------------");
33      }
34
35      @AfterClass
36      public static void tearDownClass() {
37          System.out.println("----------------- Los testeos se han realizado ---------------------");
38      }
39
40      @Before
41      public void setUp() {
42      }
43
44      @After
45      public void tearDown() {
46          co.desconexion();
47      }
```

```
48
49          // TODO add test methods here.
50          // The methods must be annotated with annotation @Test. For example:
51          //
52          // @Test
53          // public void hello() {}
54
55          @Test
56          public void TestObtenerConexion(){
58              Connection co = Conexion.getConnection();
59          }
60
61          @Test
62          public void TestConexion(){
63              Conexion.getConnection();
64          }
65      }
```

Análisis del Testeo

En los métodos previamente implementados en la prueba, se tiene la intención de probar que la conexión a la base de datos que en este caso es PostgreSQL sea correcta, adicionalmente se realiza la prueba sobre una variable de tipo Connection para validar que la obtención de dicha conexión sea correcta. En la imagen anterior también se puede ver la utilización de los métodos **BeforeClass**, **AfterClass** y **After**, los cuales serán tratados en la siguiente tabla junto al análisis correspondientes de las pruebas.

Prueba	Condición	Parámetros	Objeto	Resultado
1	**`Conexion.getConnection()`** Este método se encarga de comprobar que la conexión con la base de datos sea correcta, es decir se asegura que la comunicación con la base de datos PostgreSQL y la aplicación sea acertada y que esta no tenga problemas que impidan la implementación de esta en el proyecto.	**Class Conexion:** Clase que contiene al método correspondiente con la respuesta de conexión a la base de datos, en este caso **getConnection()**		Conexión con la base de datos PostgreSQL - 12
2	**`Connection co = Conexión.getConnection()`** Este método de prueba se rectifica que el método de obtener conexión con la base de datos PostgreSQL sea correcto.	**getConnection():** Método que obtiene la respuesta a la conexión con la base de datos PostgreSQL - 12	co: objeto que se usará para almacenar la respuesta al método de obtener conexión.	Variable con la respuesta de conexión a la base de datos de PostgreSQL-12

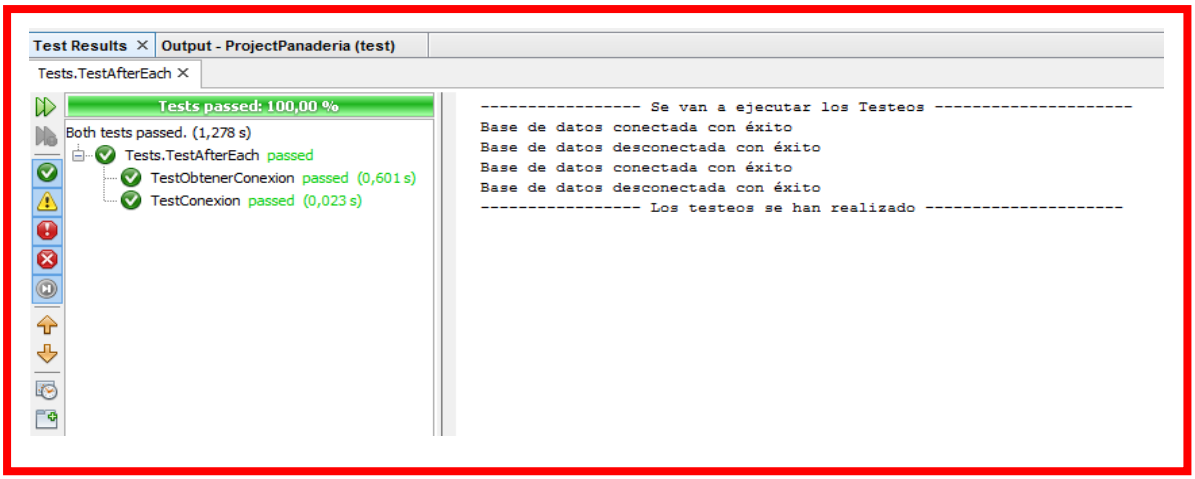
Análisis Resultado

El método para la conexión de la aplicación con la base de datos fue exitoso, esto quiere decir que el testeo fue desarrollado de forma correcta y sin errores, de igual forma de evidencia que el testeo para el método de obtención de la conexión en respuesta al servidor de PostgreSQL fue realizado de forma exitosa.

Consideraciones

No se requiere modificaciones en método de verificación y obtención de la conexión con la base de PostgreSQL – 12, ya que los resultados de la prueba fueron acertados.

Comprobación Testeo



Testeo sobre la gestión de un producto en la aplicación de la panadería

En esta sección se tratará el desarrollo e implementación de las pruebas realizadas sobre la gestión de un producto, es decir, se evaluará los métodos ubicados en la clase **ProductoBO**, los cuales se vinculan con el crear, eliminar y actualizar un producto ingresado en la aplicación y almacenado en la base de datos de PostgreSQL. Esto se irá desarrollando en el análisis correspondiente, junto a sus consideraciones y resultados dados. Para comprender de mejor manera esto, se dará a continuación una serie de imágenes como comprobación.

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package Tests;

import co.edu.usbbog.ADA.ProjectPanaderia.bo.ProductoBO;
import co.edu.usbbog.ADA.ProjectPanaderia.db.Conexion;
import co.edu.usbbog.ADA.ProjectPanaderia.model.Producto;
import java.sql.Date;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Juan Sebastian
 */
public class ProductoBOTest {

    ProductoBO pbo;
    Conexion co;

```

```
27
28      public ProductoBOTest() {
29          this.pbo = new ProductoBO();
30          this.co = new Conexion();
31      }
32
33      @BeforeClass
34      public static void setUpClass() {
35          System.out.println("--------------- Inicio del Testeo -----------------");
36      }
37
38      @AfterClass
39      public static void tearDownClass() {
40          System.out.println("--------------- Fin del Testeo -------------------");
41      }
42
43      @Before
44      public void setUp() {
45          Conexion.getConnection();
46      }
47
48      @After
49      public void tearDown() {
50          co.desconexion();
51      }
```

```
52
53      // TODO add test methods here.
54      // The methods must be annotated with annotation @Test. For example:
55      //
56      // @Test
57      // public void hello() {}
58      @Test
59      public void testCrear() {
60          String mensaje;
61          String mensajeEsperado;
62          try {
63              mensaje = pbo.crear(new Producto(1, "Chocolate", "Sol", Date.valueOf("2021-05-20"), 2500,
64                      4, "24 Kilogramos", Date.valueOf("2020-05-08")));
65              pbo.crear(new Producto(2, "Leche", "VivaMilk", Date.valueOf("2021-08-27"), 3800,
66                      2, "1.5 Litros", Date.valueOf("2020-05-08")));
67              pbo.crear(new Producto(3, "Pan", "Bimbox2", Date.valueOf("2021-03-25"), 2800,
68                      6, "8 Kilogramos", Date.valueOf("2020-05-08")));
69              mensajeEsperado = "GUARDADO CORRECTAMENTE";
70
71              assertEquals(mensajeEsperado, mensaje);
72          } catch (Exception e) {
73              System.out.println("ERROR. " + e.getMessage());
74              fail("Ocurrio un error al crear el producto");
75          }
76      }
77
```

```java
    @Test
    public void testEditar() {
        String mensaje;
        String mensajeEsperado;
        try {
            if (pbo.validacionProducto(1) && pbo.validacionProducto(2) && pbo.validacionProducto(3)) {
                mensaje = pbo.editar(new Producto(1, "Huevos", "Buen Sol", Date.valueOf("2021-05-20"), 2500,
                        4, "24 Kilogramos", Date.valueOf("2020-05-08")));
                pbo.editar(new Producto(2, "Cafe", "Sello Rojo", Date.valueOf("2021-08-27"), 3800,
                        2, "1.5 Litros", Date.valueOf("2020-05-08")));
                pbo.editar(new Producto(3, "Queso", "Don Queso", Date.valueOf("2021-03-25"), 2800,
                        6, "8 Kilogramos", Date.valueOf("2020-05-08")));
                mensajeEsperado = "ACTUALIZADO CORRECTAMENTE";
            } else {
                mensaje = "No ha sido posible actualizar los datos. ID incorrecto.";
                mensajeEsperado = "No ha sido posible actualizar los datos. ID incorrecto.";
            }

            assertEquals(mensajeEsperado, mensaje);
        } catch (Exception e) {
            System.out.println("ERROR. " + e.getMessage());
            fail("Ocurrio un error al actualizar el producto");
        }
    }
```

```java
    @Test
    public void testEliminar() {
        String mensaje;
        String mensajeEsperado;
        try {
            if (pbo.validacionProducto(1) && pbo.validacionProducto(2) && pbo.validacionProducto(3)) {
                mensaje = pbo.eliminar(1);
                pbo.eliminar(2);
                pbo.eliminar(3);
                mensajeEsperado = "BORRADO CORRECTAMENTE";
            } else {
                mensaje = "El producto no ha sido eliminado. El ID del producto no existe";
                mensajeEsperado = "El producto no ha sido eliminado. El ID del producto no existe";
            }

            assertEquals(mensajeEsperado, mensaje);
        } catch (Exception e) {
            fail("Ocurrio un error al eliminar el producto");
        }
    }
}
```

Análisis del Testeo

En los métodos previamente mostrados, se quiere validar que el CRUD de la base de datos funciones de manera adecuada, es decir, se probarán los métodos correspondiente a la clase ProductoBO, puesto que esta contiene los métodos manejados para la correcta gestión de un elemento a la base de datos PostgreSQL – 12, además se tiene la intención de comprobar que la clase ProductosDAOImpl funcione de forma correcta, ya que dichas clases mantiene una relación para el funcionamiento de la aplicación y la interacción con la base de datos de PostgreSQL, sin embargo, posteriormente se demostrará que de igual forma los métodos que contiene la lógica de la gestión de un producto que en su defecto es la clase ProductosDAOImpl funciona y tiene una correcta comunicación con la base de datos previamente designada. Para obtener un análisis más profundo de los métodos anteriormente nombrados, se propone la siguiente tabla.

Prueba	Condición	Parámetros	Objeto	Resultado
1	**`pbo.crear(new Producto())`** Este método hace que un nuevo producto previamente establecido se cree en la base de datos, de esa forma se otorgar como parámetro un nuevo producto y este método retornará un mensaje de confirmación de que el producto fue creado de forma correcta.	**pbo:** Instanciación de la clase ProductoBO, la cual tendrá lo métodos correspondiente al DAO de la base de datos de PostgreSQL. **crear():** Método que realizar el llamado a los procesos necesarios para la creación de un nuevo producto en la base de datos. **new Producto():** Utilización del constructor de la clase **Producto** para la creación de nuevo producto y de esta forma darlo como parámetro al método de Crear previamente explicado.	**`new Producto(1, "Chocolate", "Sol",Date.valueOf("2021-05-20"),2500, 4,"24Kilogramos", Date.valueOf("2020-05-08"))`**	Creación de un nuevo producto en la base de datos de PostgreSQL - 12
2	**`pbo.editar(new Producto())`** Este método hace que un producto sea reemplazado por otro, de esa forma se puede alterar los datos de un producto y almacenar dichos cambios en la base de datos con el producto que previamente fue guardado.	**pbo:** Instanciación de la clase ProductoBO, la cual tendrá lo métodos correspondiente al DAO de la base de datos de PostgreSQL. **editar():** Método para realizar el llamado a los procesos necesarios para la modificación de un producto que previamente fue guardado en la base de datos. **new Producto():** Utilización del constructor de la clase **Producto** para la modificación de un producto y de esta forma darlo como parámetro al método de editar previamente explicado.	**`new Producto(2, "Cafe", "Sello Rojo", Date.valueOf("2021-08-27"), 3800, 2, "1.5 Litros", Date.valueOf("2020-05-08"))`**	Modificación de un producto previamente alamacenado en la base de datos de PostgreSQL - 12
3	**`pbo.eliminar(id)`**	**pbo:** Instanciación de la clase ProductoBO, la cual	**`int id`**	Eliminación de un

	Este método realiza la eliminación de un elemento que se encuentra previamente registrado en la base datos, de esa forma se solicita como parámetro el id de dicho producto, con el fin de ser encontrado en la base de datos, ya que, el id corresponde a la llave primaria de la tabla producto creada en PostgreaSQL-12.	tendrá lo métodos correspondiente al DAO de la base de datos de PostgreSQL. **eliminar():** Método para realizar el llamado a los procesos necesarios para la eliminación de un producto que previamente fue guardado en la base de datos. **id:** Elemento de tipo entero que se dará como parámetros al método de **eliminar** un producto, con el fin de buscar el producto específico que será eliminado de la base de datos.		producto en específico que previamente fue almacenado en la base de datos.

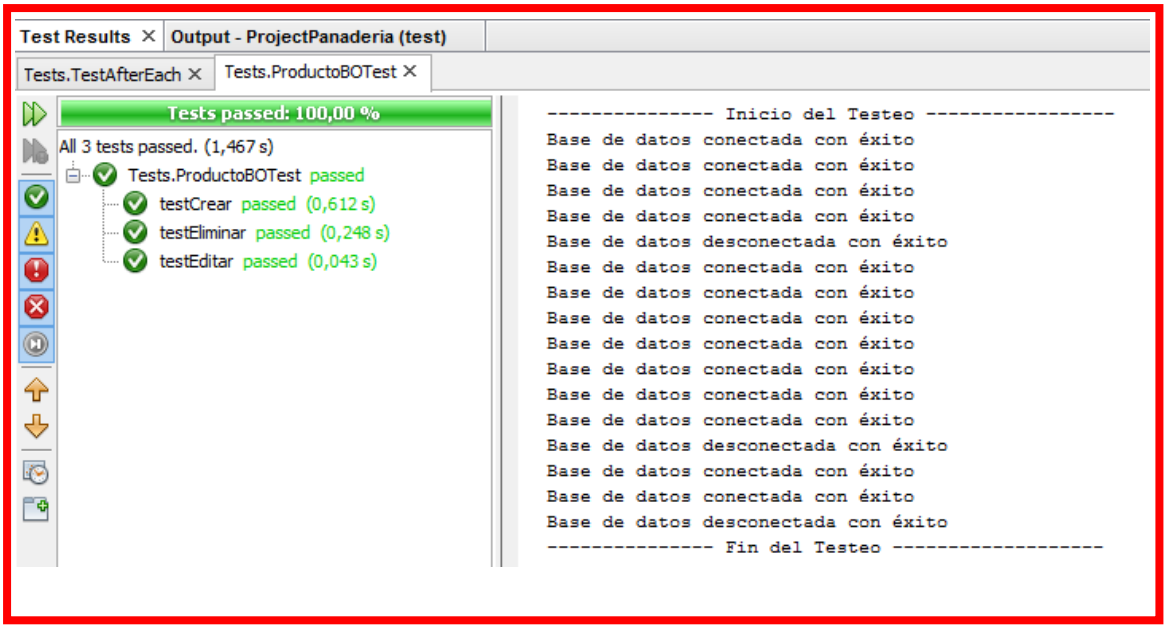
Análisis Resultado

Los métodos relacionados al CRUD de la base de datos en relación con la aplicación de la panadería, obtuvieron resultados buenos, puesto que al realizar el testeo no hubo ningún inconveniente, esto quiere decir que lo métodos fueron exitosos y lograron alcanzar el 100 porciento de la prueba realizada en el Framework de JUnit 4.

Consideraciones

No se requiere que los métodos se han cambiados, puesto que los resultados obtenidos en la prueba fueron exitosos. Esto se podrá evidenciar en la siguiente imagen tomada de la ejecución de la clase correspondiente para la prueba.

Comprobación Testeo



Testeo sobre la clase producto en el proyecto de la panadería

En esta sección se trataran las pruebas relacionadas al testeo de la clase Producto, la cual es nuestra base para agregar o crear un producto, el objetivo de estas pruebas puedes ser sencillo, pero, es esencial para comprobar que un método, en especial, los getters y setters de esta clase, se encuentre aplicados de forma correcta, así se ejecutaran la pruebas que corresponden a la clase que contiene dichos métodos que en este caso es llamada como Producto, adicionalmente se quiso utilizar una anotación extra de JUnit 4, esta puede ser el remplazo de la anotación ***@ParameterizedTest***, la cual consiste en denotar que un método puede ser los parámetros de un testeo en específicos, dicha anotación nombrada pertenece a la versión más reciente de JUnit que en este caso será la 5, pero esto no impidió para que la anotación fuera vista de otra forma e implementada en JUnit4.

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package Tests;
7
8  import co.edu.usbbog.ADA.ProjectPanaderia.model.Producto;
9  import java.sql.Date;
10 import org.junit.After;
11 import org.junit.AfterClass;
12 import org.junit.Before;
13 import org.junit.BeforeClass;
14 import org.junit.Test;
15 import static org.junit.Assert.*;
   import org.junit.runner.RunWith;
   import org.junit.runners.Parameterized;
18 import org.junit.runners.Parameterized.Parameters;
19
20 /**
21  *
22  * @author Juan Sebastian
23  */
24 //@RunWith(Parameterized.class)
25 public class ProductoTest {
26
27     public static Producto pr;
28
```

```
28
29      public ProductoTest() {
30          pr = new Producto();
31      }
32
33      @BeforeClass
34      public static void setUpClass() {
35      }
36
37      @AfterClass
38      public static void tearDownClass() {
39      }
40
41      @Before
42      public void setUp() {
43          pr.setId(0);
44          pr.setNombre("Leche");
45          pr.setMarca("Alpina");
46          pr.setFecha_vencimiento(Date.valueOf("2021-08-18"));
47          pr.setCosto(4500);
48          pr.setCantidad(3);
49          pr.setPeso("2.3 Litros");
50          pr.setFecha_venta(Date.valueOf("2020-05-14"));
51      }
52
```

```
52
53      @After
54      public void tearDown() {
55      }
56
57      // TODO add test methods here.
58      // The methods must be annotated with annotation @Test. For example:
59      //
60      // @Test
61      // public void hello() {}
62
63      @Parameters
64      public String getNombre() {
65          return pr.getNombre();
66      }
67
68      @Test
69      public void testGetProducto() {
70          try {
71              assertEquals(getNombre(), "Leche");
72          } catch (Exception e) {
73              fail("Ocurrio un error al comprobar el producto");
74          }
75      }
76  }
```

Análisis del Testeo

En estas pruebas unitarias se quiere saber si los métodos correspondientes a la clase de Producto, realizan de forma correcta sus funciones, que en este caso es guardar de forma temporal los datos que el usuario ingrese, de esa forma también se espera que los datos sean enviados de forma exitosa cuando estos se soliciten, por lo cual, será primordial la implementación de esto métodos, es conocido que esto métodos son muy sencillos y pueden que en su mayoría no estén mal planteados, pero puede existir la posibilidad que el tipo de dato que se asigne a un atributo de una clase específica sea errónea a lo que realmente queremos guardar y una forma de comprobar eso es con la utilización de pruebas unitarias en nuestro Software. A continuación, se explicará el análisis profundo de esta prueba unitaria realizada.

Prueba	**Condición**	**Parámetros**	**Objeto**	**Resultado**
1	**pr.getNombre()** Este método retorna un valor de tipo String que corresponderá al nombre del producto que fue almacenado en la base de datos. De esa forma se comprobará que el elemento realmente se está guardado temporalmente en los atributos correspondientes a la clase Producto.	**pr:** Instanciación que representa un objeto de la clase Producto, de esa forma se podrán obtener todos lo métodos relacionados a dicha clase. **getNombre():** Método donde se retorna el nombre del producto previamente asignado al atributo del producto.		Retorna el nombre del producto previamente asignado, de esta forma se podrá almacenar de forma más rápida en la base de datos, además de poder operaciones que relacionen al nombre del producto en cuestión.
2	**getNombre** Este método corresponde a la anotación de **@Parameters**, que en JUnit 4 se entiende como el método que dará como resultado	**getNombre():** Método de la clase Test que corresponde al resultado especificado por la anotación **@Parameters** que da a		Retorna el valor de la operación realizada con la marca o anotación de **@Parameters**, que en este caso es la de

	los parámetros para ciertos métodos de testeo o asignados con el **@Test**	conocer los parámetros de un método testeo.		dar el nombre del producto.

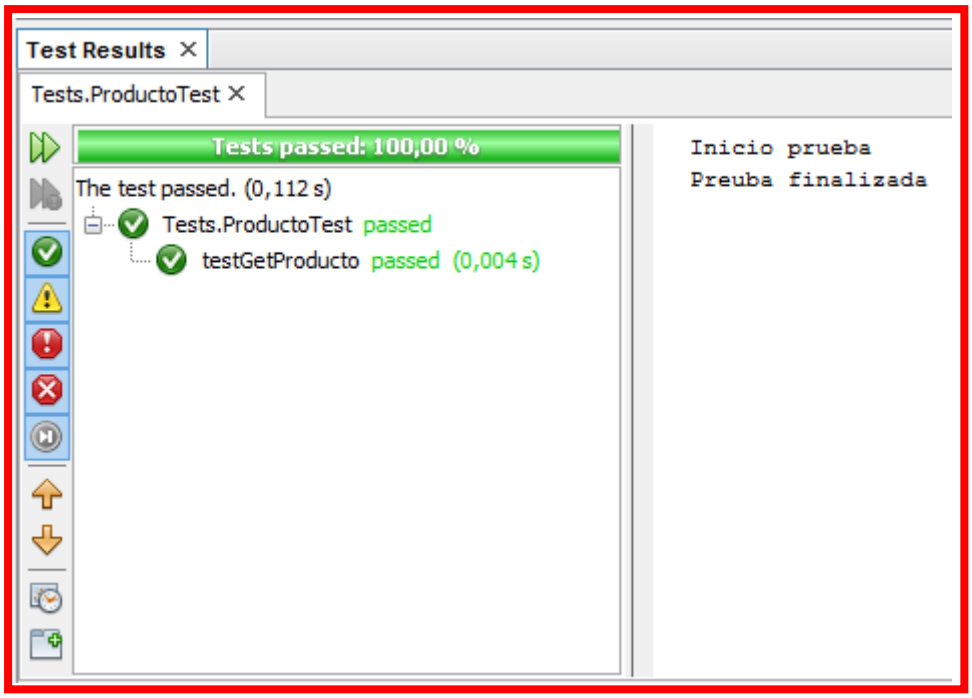
Análisis Resultado

El resultado de la prueba fue exitoso, ya que se obtuvo el resultado de forma correcta al realizar la comprobación con el método de **assert,** de esa forma podemos concluir que la forma de desarrollar la anotación @ParametizedTest fue correcta y se pudo comprobar con un método relacionado al proyecto de panadería.

Consideraciones

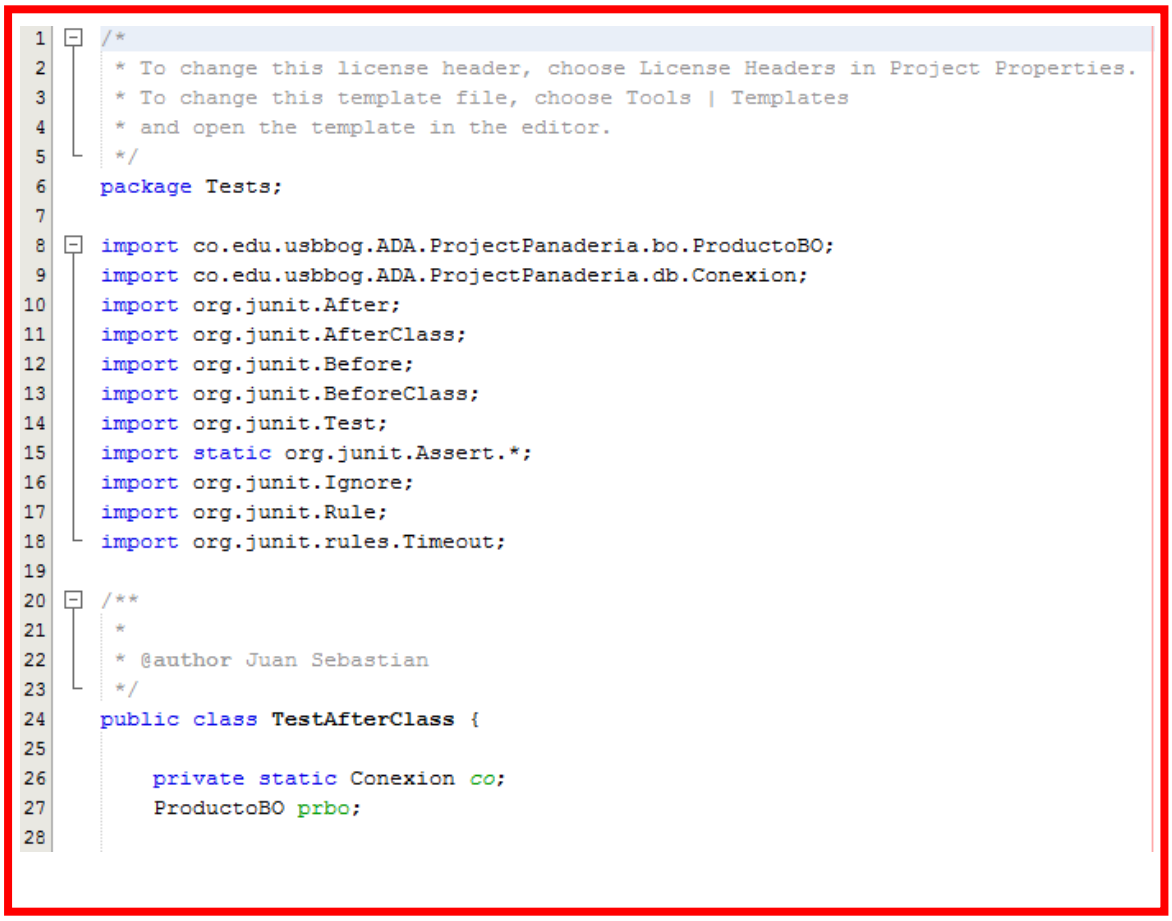
No ha necesidad de cambiar el método, puesto que este dio resultados positivos en la realización de la prueba unitaria, lo cual quiere decir que el método está bien diseñado.

Comprobación Testeo



Testeo sobre el tiempo de ejecución de la prueba en el proyecto de la panadería

En este apartado se quiere es comprobar el tiempo que tardará una prueba es ser realizada, de esa forma tenemos control de cuanto nos puede llevar hacer una prueba en un método específico sobre la anotación **@Test**, así comprobamos que dicha clase con los métodos a testear tarda poco o demasiado tiempo en llevar sus operaciones en el funcionamiento del software en general, para ello se usó la anotación **@Rule** para asignar esta nueva regla en la prueba sobre el tiempo de ejecución que tarda cada método en dicha clase. Lo anterior será analizado en las siguientes tablas.



```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package Tests;
7
8  import co.edu.usbbog.ADA.ProjectPanaderia.bo.ProductoBO;
9  import co.edu.usbbog.ADA.ProjectPanaderia.db.Conexion;
10 import org.junit.After;
11 import org.junit.AfterClass;
12 import org.junit.Before;
13 import org.junit.BeforeClass;
14 import org.junit.Test;
15 import static org.junit.Assert.*;
16 import org.junit.Ignore;
17 import org.junit.Rule;
18 import org.junit.rules.Timeout;
19
20 /**
21  *
22  * @author Juan Sebastian
23  */
24 public class TestAfterClass {
25
26     private static Conexion co;
27     ProductoBO prbo;
28
```

```
28
29      public TestAfterClass() {
30          co = new Conexion();
31          prbo = new ProductoBO();
32      }
33
34      @BeforeClass
35      public static void setUpClass() {
36      }
37
38      @AfterClass
39      public static void tearDownClass() {
40          co.desconexion();
41      }
42
43      @Before
44      public void setUp() {
45      }
46
47      @After
48      public void tearDown() {
49      }
```

```
50
51          // TODO add test methods here.
52          // The methods must be annotated with annotation @Test. For example:
53          //
54          // @Test
55          // public void hello() {}
56          @Test
57          public void TestEliminar() {
58              try {
59                  String mensaje = prbo.eliminar(1);
60                  String mensajeEsperado = "BORRADO CORRECTAMENTE";
61
62                  assertEquals(mensajeEsperado, mensaje);
63              } catch (Exception e) {
64                  fail("Ha ocurrido un error al borrar el producto");
65              }
66          }
67
68          //Con la anotación @Ignore no se tiene en cuenta el testeo de un método en particular.
69          @Test
70          @Ignore
71          public void TestPrueba() {
72              System.out.println(5 * 2);
73          }
74
75          @Rule
76          public Timeout timeout = Timeout.millis(5000);
77      }
78
```

Análisis de Testeo

A continuación, se encontrará el análisis respectivos de los métodos representadas en unas tablas que se mostrarán posteriormente.

Prueba	Condición	Parámetros	Objeto	Resultado
1	**Timeout.millis(long millis)** Este es un método que nos permite establecer el tiempo que se tomará llevar a cabo una prueba, se puede decir, que dicha anotación de **@Rule** sobre el tiempo que lleva a cabo una prueba reemplaza a la anotación de JUnit 5 llamada **@Timeout,** que cumple de igual forma el tiempo de ejecución de cada prueba.	**Timeout:** Instanciación de la clase Time para realizar la cuenta sorbre el tiempo de ejecución del proyecto en cuestión. **Millis(long millis):** Significa el tiempo dado en millasegundos para realizar la cuenta del tiempo en ejecución	**Long millis:** Realiza la cuenta del tiempo de ejecución en milisegundo y lo almacena en una variable de tipo long.	Se realiza prueba del tiempo de ejecución de las pruebas en cuestión.
2	**prbo.eliminar(int id)** Se realiza la prueba del método eliminar de la clase ProductoBO, además se realiza la prueba del tiempo de ejecución	**prbo:** Instanciación de la clase ProductoBO, la cual tendrá lo métodos correspondiente al DAO de la base de datos de PostgreSQL. **eliminar():** Método que elimina un producto de la base de datos por medio de la aplicación	**int id:** Realiza una búsqueda del elemento por medio de la llave primaria que en su representación es el id del producto, esta se transfiere como parámetro al método de eliminar.	Se realiza la eliminación del producto en cuestión en la base de datos, adicionalmente se realiza el testeo del tiempo lleva realizar la prueba unitaria sobre el método de eliminación.
3	**@Ignore** Esta anotación ignora la ejecución de un método de tipo testeo.			