



PECL1

Calidad, pruebas y mantenimiento Software –
2020/2021

Adina Murg



Resumen Documentación Ejercicio 1

Objetivo	Usar un programa OpenSource Java + herramienta de testeo para pruebas unitarias + medida de la cobertura.
Código	https://github.com/whoisadn/paradigmas-de-programacion-pecl
Entorno necesario	Eclipse.
Conclusiones	Eclipse nos proporciona buenas herramientas tanto para medir la cobertura como para realizar pruebas unitarias, además de un fácil uso de las mismas, (aunque haya otros entornos como NetBeans que nos permite realizar pruebas de integración). Lo único complejo es la realización de las pruebas unitarias de forma correcta, sabiendo elegir correctamente los test (además, sumando la complejidad del paradigma Orientado a Objetos, un código con un acoplamiento muy alto, ya que los módulos dependen mucho los unos de los otros, y el hecho de que el código sea concurrente).

Conceptos teóricos:

Pruebas unitarias: técnica que consiste en comprobar si un módulo de nuestro código funciona de manera correcta o incorrecta.

Fases del ciclo completo de pruebas:

1. Planificación pruebas
2. Diseño de pruebas
3. Ejecución de pruebas
4. Evaluación de pruebas, con su posterior depuración o análisis de errores.

Cobertura: criterio que nos indica el porcentaje de nuestro código que ha sido cubierto por las pruebas unitarias.

Ambos conceptos forman parte del diseño de pruebas de caja blanca/transparente/funcional.

Explicación del código:

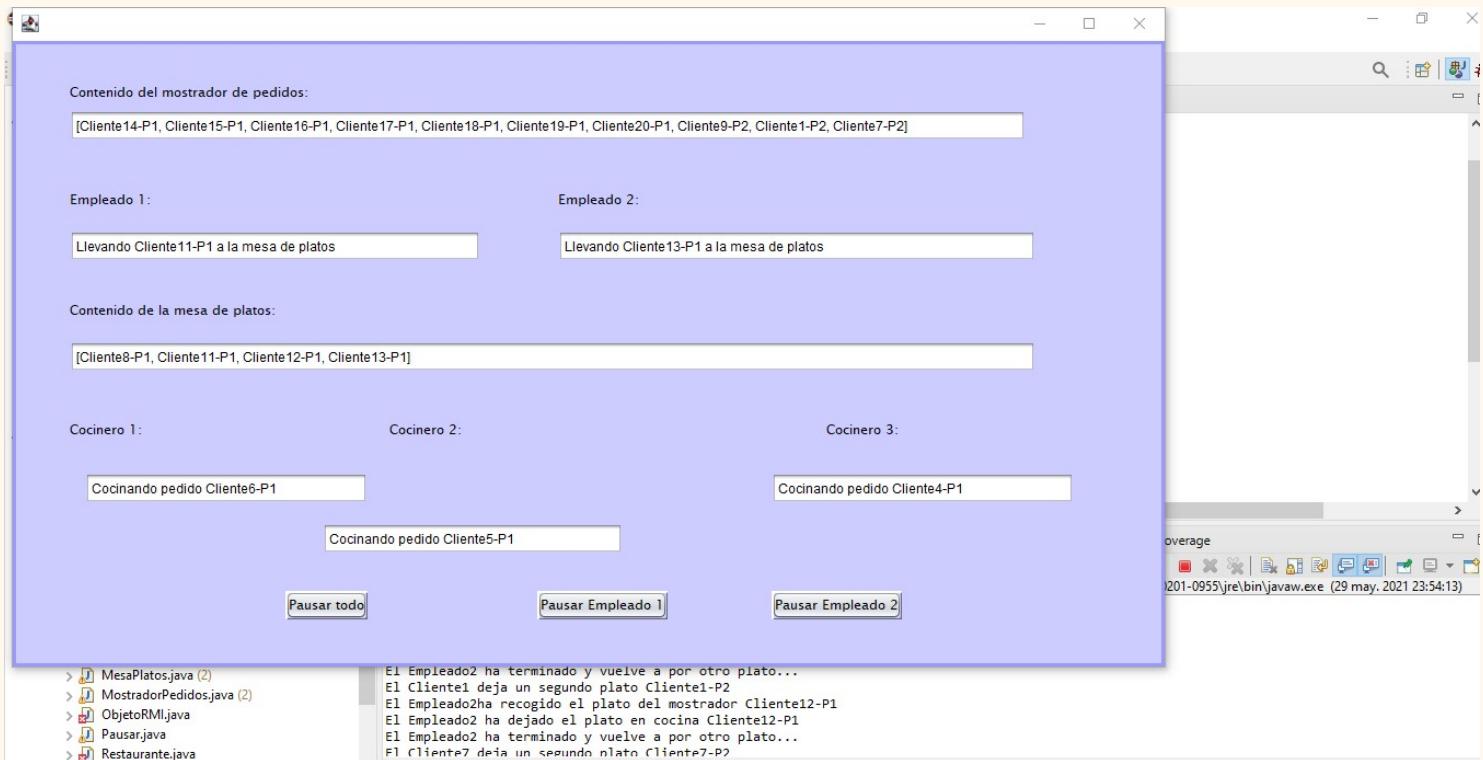
Para esta práctica se usará el código de una práctica de otra asignatura (Paradigmas de Programación).

Considero interesante el analizar y poner a prueba un código que yo misma desarrollé con la ventaja adicional de conocer cómo funciona.

Hay que aclarar que el código fue compilado en NetBeans, y al cambiar y ejecutarlo en otro IDE como Eclipse, algunas clases generaron algunos errores. Por eso, utilicé el mismo código pero ‘simplificado’ (eliminando las clases que generaban problemas en Eclipse).

```
37     texto1.setFont(new java.awt.Font("Lucida Sans Unicode", 1, 14)); // NOI18N
38     texto1.setText("Contenido del mostrador de pedidos:");
39     getContentPane().add(texto1, new org.netbeans.lib.awtextra.AbsoluteConstraints(30, 20, -1, -1));
40 
```

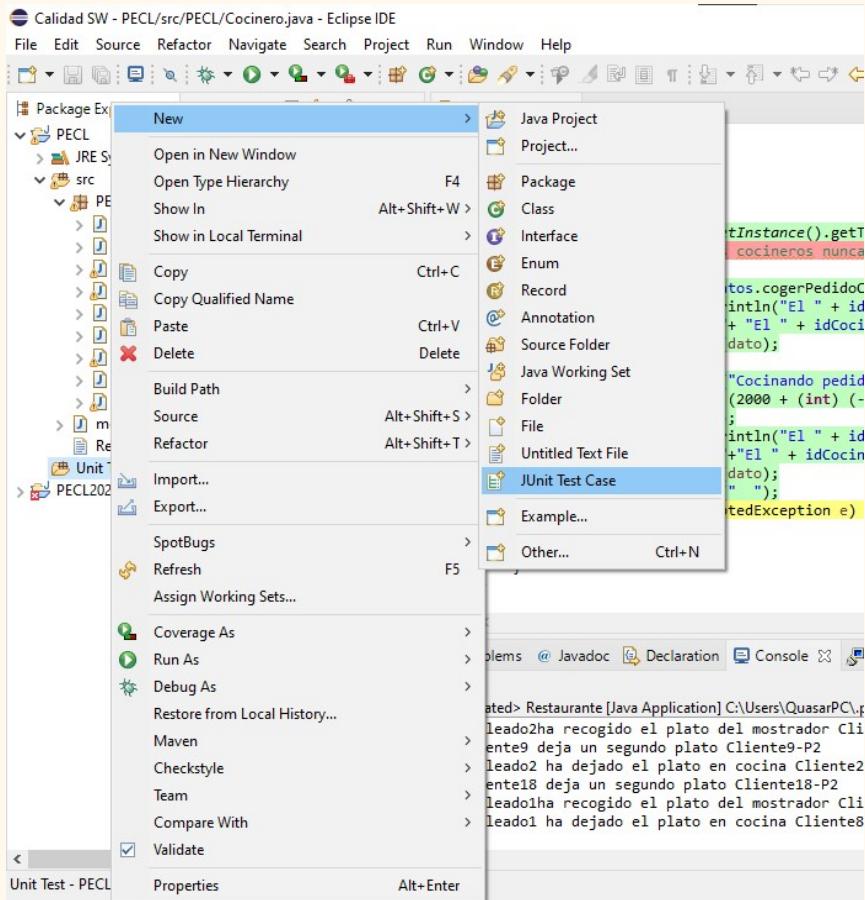
El código simplificado cuenta con las clases Cliente, Cocinero, Empleado, Log, MesaPlatos, MostradorPedidos, Pausar y Restaurante (clase Main). El código hace uso de hilos y herramientas para la protección de la memoria compartida (cerros + cerros implícitos) además de contar con una interfaz gráfica. Nada más ejecutar la clase main Restaurante, el programa empieza a ejecutarse, sin necesidad de introducir ningún dato en ningún momento.

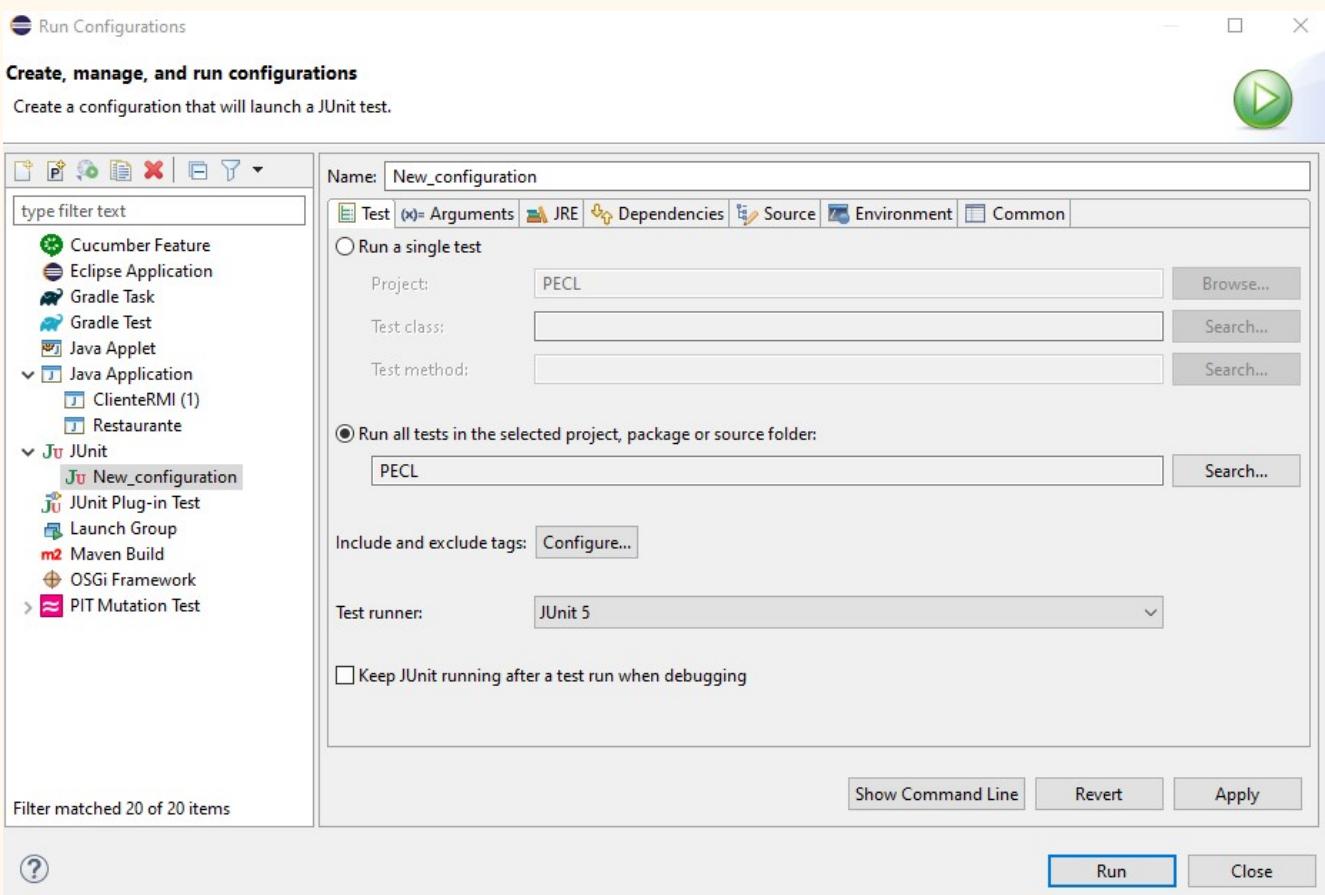
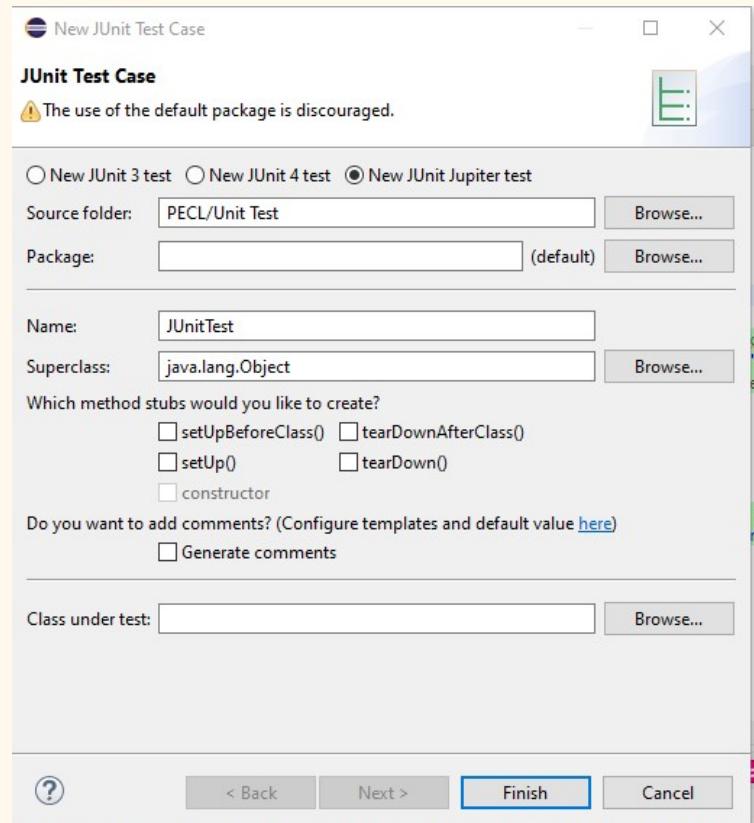


Pruebas unitarias:

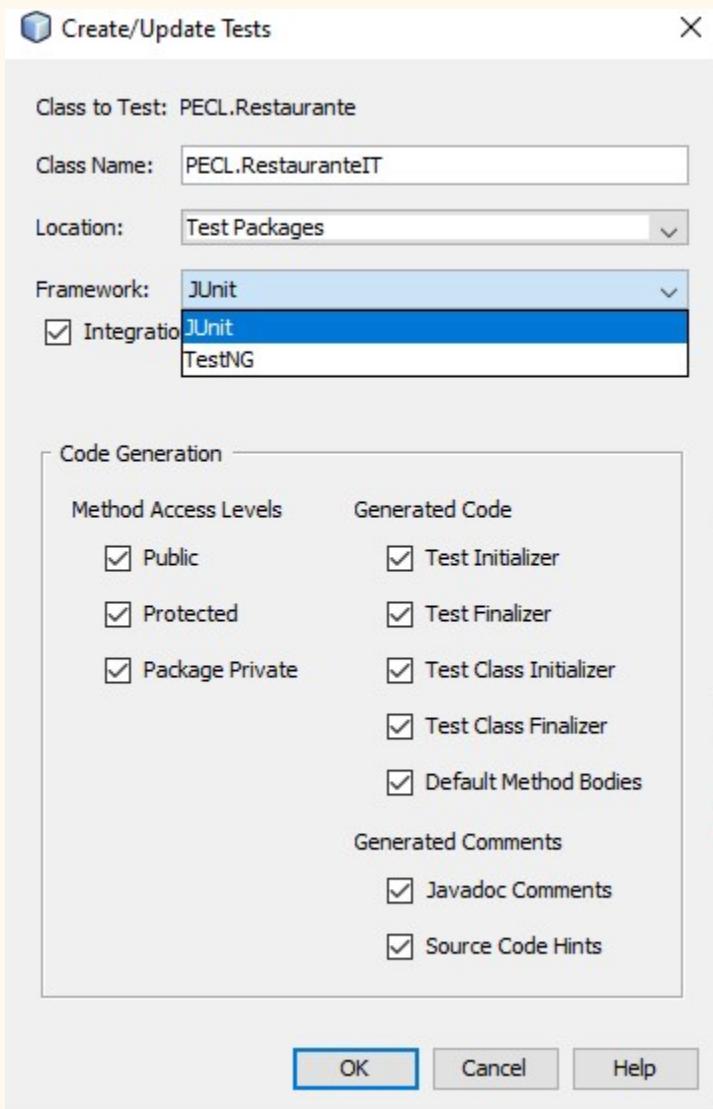
Para realizar los *unit test* usaremos el entorno de Eclipse.

Desde New > JUnit Test Case configuraremos la prueba. Por defecto tenemos la opción de usar JUnit 3, 4 o el Jupiter Test (que es el que seleccionaremos).





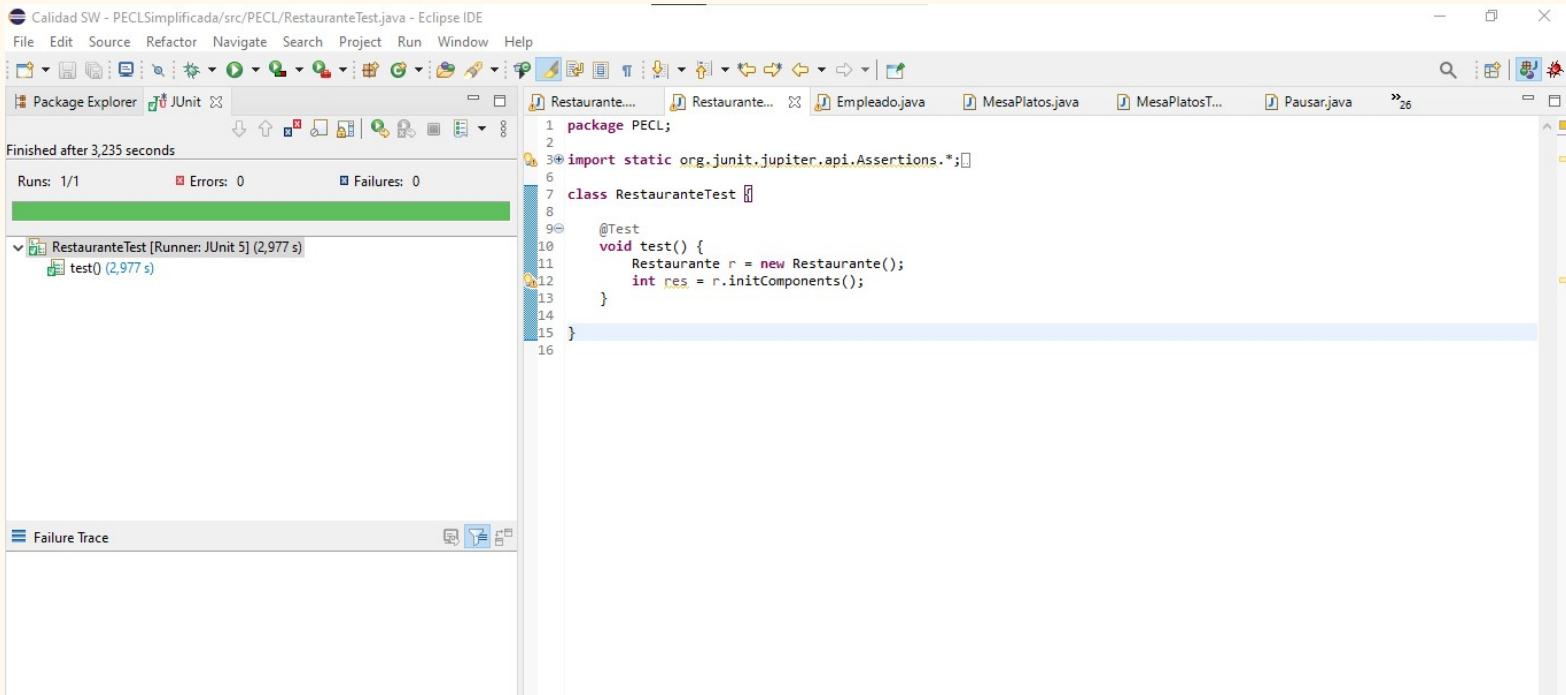
Hay que destacar que Eclipse solo permite las pruebas unitarias, pero, por ejemplo, NetBeans incluye la opción de realizar pruebas de integración.



Vamos a empezar a realizar las pruebas unitarias. Debido a la dificultad del código, (además de hilos, arrays y otros atributos como logs) solo se ha realizado la prueba sobre dos clases, pero son suficientes como para poder representar o, al menos, empezar a entender cómo realizar las pruebas unitarias.

Dentro de la clase que hemos generado y tras importar las librerías necesarias, tenemos que escribir las pruebas en el método void test().

Para la clase Restaurante (main), si ejecutamos la prueba recibimos un resultado de test ejecutado y código correcto. Esto se debe, también, a que la clase Restaurante lanza hilos y se encarga de la parte gráfica, no realiza ninguna tarea que podamos estudiar y probar a fondo.



```
Calidad SW - PECLSimplificada/src/PECL/RestauranteTest.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer JUnit
Finished after 3,235 seconds
Runs: 1/1 Errors: 0 Failures: 0
RestauranteTest [Runner: JUnit 5] (2,977 s)
  test() (2,977 s)

1 package PECL;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class RestauranteTest {
6
7     @Test
8     void test() {
9         Restaurante r = new Restaurante();
10        int res = r.initComponents();
11    }
12
13 }
14
15 }
16
```

Para la siguiente clase, Cliente, realizaremos lo siguiente:

1. Escribiremos el código en void test()
2. Tenemos que declarar un objeto de la clase a analizar, pasándole los 3 parámetros necesarios. Uno de los parámetros más sencillos que podremos analizar es el idCliente, de tipo String.
3. Pasaremos a analizar el método getIdCliente().
4. Usaremos el método assertEquals, el cual evaluará si el test cumple con el objetivo indicado.

Calidad SW - PECLSimplificada/src/PECL/ClienteTest.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit

Finished after 0,399 seconds

Runs: 1/1 Errors: 1 Failures: 0

ClientTest [Runner: JUnit 5] (0,085 s)

testString() (0,085 s)

```

1 package PECL;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import java.util.ArrayList;
6
7 import javax.swing.JTextField;
8
9 import org.junit.jupiter.api.Test;
10
11 class ClienteTest {
12
13     @Test
14     void testString() {
15         String idCliente = 1234;
16         MostradorPedidos mp = new MostradorPedidos(3, null);
17         Log l = new Log();
18         Cliente cli = new Cliente( idCliente, mp, l );
19
20         String getid = cli.getIdCliente();
21
22     }
23
24 }
25
26

```

Failure Trace

java.lang.Error: Unresolved compilation problem:
Type mismatch: cannot convert from int to String

at PECLSimplificada/PECL.ClienteTest.testString(ClienteTest.java:16)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

Problems Declaration Console Terminal Bug Explorer Bug Info Coverage

<terminated> RestauranteTest [JUnit] C:\Users\QuasarPC\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (3)

Si en String idCliente escribimos solamente enteros, el Test nos devolverá el error *Type mismatch cannot convert from int to string*. Si en vez de introducir un entero, introducimos un valor String correcto podemos ver como el test nos devuelve 0 errores.

Calidad SW - PECLSimplificada/src/PECL/ClienteTest.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer JUnit

Finished after 0,367 seconds

Runs: 1/1 Errors: 0 Failures: 0

ClientTest [Runner: JUnit 5] (0,035 s)

testString() (0,035 s)

```

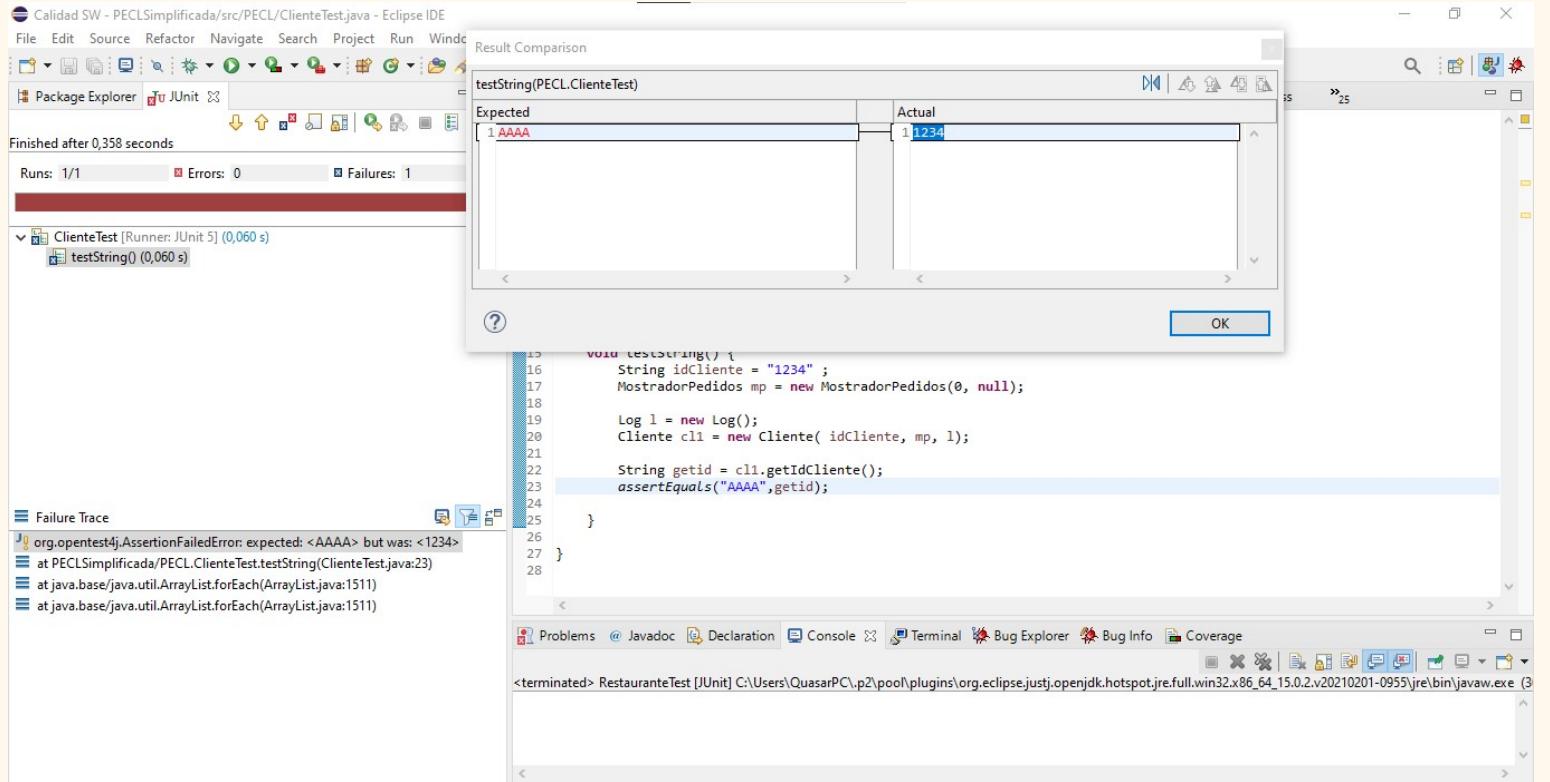
1 package PECL;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import java.util.ArrayList;
6
7 import javax.swing.JTextField;
8
9 import org.junit.jupiter.api.Test;
10
11 class ClienteTest {
12
13     @Test
14     void testString() {
15         String idCliente = "Cliente-01";
16         MostradorPedidos mp = new MostradorPedidos(3, null);
17         Log l = new Log();
18         Cliente cli = new Cliente( idCliente, mp, l );
19
20         String getid = cli.getIdCliente();
21
22     }
23
24 }
25

```

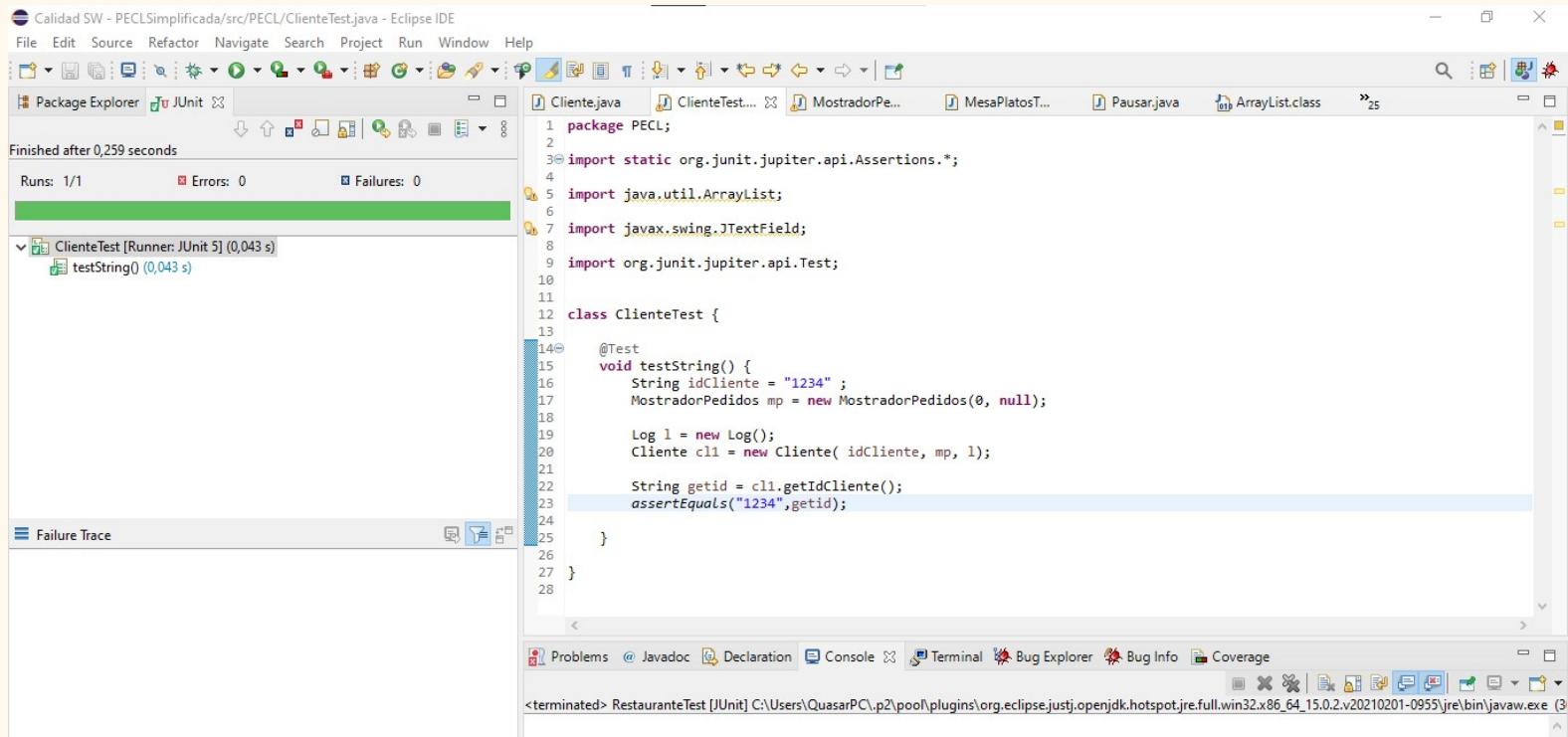
Failure Trace

Problems Declaration Console Terminal Bug Explorer Bug Info Coverage

<terminated> RestauranteTest [JUnit] C:\Users\QuasarPC\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-0955\jre\bin\javaw.exe (3)



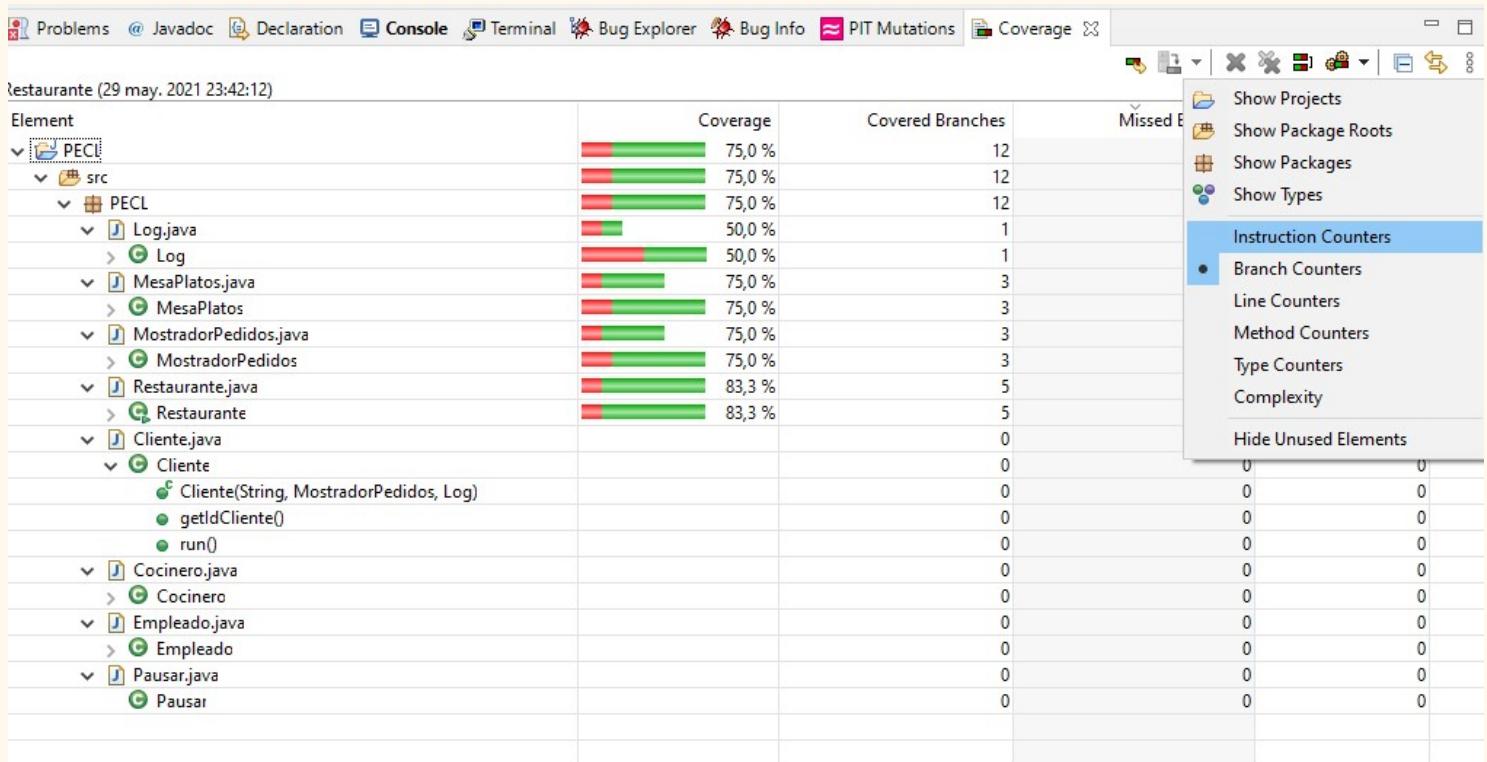
Usando `assetEquals` podemos ver que el test nos devuelve un error si el valor indicado con `idCliente` no coincide (FailedError: expected <valorX> but was <valorY>). Si ambos valores coinciden, significa que el test va a cumplir con lo esperado.



Pruebas de cobertura:

Para las pruebas de cobertura volveremos a usar Eclipse. Para ello solo tenemos que hacer click derecho sobre la clase y marcar Coverage As.

El resultado que obtenemos lo podemos observar por instrucciones, ramas, líneas de código y métodos, entre otros. El resultado de las pruebas de cobertura se puede exportar en formato CSV.



Tal y como podemos observar, la gran mayoría de instrucciones son cubiertas, pero destaca la clase Log y Pausar por la falta de cobertura que presentan.

A continuación analizaremos los motivos del porqué esta falta de cobertura.

Las pruebas de cobertura por instrucciones nos devuelven los siguientes resultados:

MostradorPedidos.java

```

10 public class MostradorPedidos {
11
12     private ArrayList<String> pedidosMostrador; //Array de pedidos
13     private int maximo = 0, numElem = 0;
14     Lock cerrojo = new ReentrantLock();
15     private Condition lleno = cerrojo.newCondition();

```

Restaurante (29 may, 2021 23:42:12)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
PECL	92,0 %	1.609	139	1.748
src	92,0 %	1.609	139	1.748
PECL	92,0 %	1.609	139	1.748
Restaurante.java	91,1 %	925	90	1.015
> Restaurante	95,0 %	862	45	907
Log.java	67,9 %	55	26	81
> Log	67,9 %	55	26	81
MesaPlatos.java	95,0 %	115	6	121
> MesaPlatos	95,0 %	115	6	121
MostradorPedidos.java	94,9 %	112	6	118
> MostradorPedidos	94,9 %	112	6	118
Cliente.java	96,9 %	127	4	131
> Cliente	96,9 %	127	4	131
Pausar.java	0,0 %	0	3	3
> Pausar	0,0 %	0	3	3
Cocinero.java	98,3 %	115	2	117
> Cocinero	98,3 %	115	2	117
Empleado.java	98,8 %	160	2	162
> Empleado	98,8 %	160	2	162

Restaurante (29 may, 2021 23:42:12)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Restaurante()	100,0 %	122	0	122
> initComponents()	100,0 %	710	0	710
Log.java	67,9 %	55	26	81
> Log	67,9 %	55	26	81
crearArchivo()	45,5 %	15	18	33
escribir(String)	80,0 %	32	8	40
MesaPlatos.java	95,0 %	115	6	121
> MesaPlatos	95,0 %	115	6	121
dejarPedidoCocina(String)	91,9 %	34	3	37
getPedidosMesa()	0,0 %	0	3	3
MesaPlatos(int, JTextField)	100,0 %	35	0	35
cogerPedidoCocina()	100,0 %	43	0	43
getMesaPlatos()	100,0 %	3	0	3
MostradorPedidos.java	94,9 %	112	6	118
> MostradorPedidos	94,9 %	112	6	118
cogerPedido()	93,0 %	40	3	43
getPedidosMostrador()	0,0 %	0	3	3
MostradorPedidos(int, JTextField)	100,0 %	35	0	35
dejarPedido(String)	100,0 %	37	0	37
Cliente.java	96,9 %	127	4	131
> Cliente	96,9 %	127	4	131
getIdCliente()	0,0 %	0	3	3
run()	99,1 %	115	1	116
Cliente(String, MostradorPedidos, Log)	100,0 %	12	0	12
Pausar.java	0,0 %	0	3	3
> Pausar	0,0 %	0	3	3

La clase Pausar está vacía (por lo que deducimos que la cobertura la marca como 0 por no tener nada que analizar) y la clase Log cuenta con el método crearArchivo() cuenta con una falta de cobertura posiblemente a consecuencia de la excepción y del uso de archivo de tipo File.

```
1 package PECL;
2
3
4
5 public class Pausar {
6
7 }
8
```

```
9
10 public class Log {
11     private FileWriter escritor;
12     private File archivo;
13     private Lock cerrojo = new ReentrantLock();
14
15     public void crearArchivo(){
16         try {
17             archivo = new File("evolucionRestaurante.txt");
18             if (archivo.createNewFile()){
19                 System.out.println("El archivo se ha creado");
20             }else{
21                 System.out.println("El archivo ya existe y se va a sobreescibir ");
22                 archivo.delete();
23                 archivo.createNewFile();
24             }
25         } catch (IOException e) {
26             System.out.println("No se ha podido ejecutar");
27             e.printStackTrace();
28         }
29     }
30
31     public void escribir(String texto){
32         cerrojo.lock();
33         try {
34             escritor=new FileWriter("evolucionRestaurante.txt",true); //El escritor se inicializa con la misma ruta de antes para esta
35             escritor.write(texto+"\n");
36             escritor.flush();
37             escritor.close();
38         } catch (IOException ex) {
39             Logger.getLogger(Log.class.getName()).log(Level.SEVERE, null, ex);
40         }
41         finally {
42             cerrojo.unlock();
43         }
44     }
45 }
```

Ejemplo de Restaurante, una clase con una alta cobertura.

Empleado.java Cocinero.java Log.java MostradorPedidos.java MesaPlatos.java Restaurante.java

```

70     Texto5 = new javax.swing.JLabel();
71     Texto6 = new javax.swing.JLabel();
72     jLabel3 = new javax.swing.JLabel();
73     MuestraTexto1 = new javax.swing.JTextField();
74     MuestraTexto2 = new javax.swing.JTextField();
75     MuestraTexto3 = new javax.swing.JTextField();
76     MuestraTexto4 = new javax.swing.JTextField();
77     MuestraTexto5 = new javax.swing.JTextField();
78     MuestraTexto6 = new javax.swing.JTextField();
79     MuestraTexto7 = new javax.swing.JTextField();
80     BotonPausarTodo = new javax.swing.JButton();
81     BotonPausarEmpleado1 = new javax.swing.JButton();
82     BotonPausarEmpleado2 = new javax.swing.JButton();
83
84     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
85
86     Fondo.setBackground(new java.awt.Color(204, 204, 255));
87     Fondo.setBorder(javax.swing.BorderFactory.createMatteBorder(3, 3, 3, 3, new java.awt.Color(153, 153, 255)));
88     Fondo.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
89
90     Texto1.setFont(new java.awt.Font("Lucida Sans Unicode", 0, 12)); // NOI18N
91     Texto1.setText("Contenido del mostrador de pedidos:");
92
93     Texto2.setFont(new java.awt.Font("Lucida Sans Unicode", 0, 12)); // NOI18N
94     Texto2.setText("Empleado 1:");
95
96     Texto3.setFont(new java.awt.Font("Lucida Sans Unicode", 0, 12)); // NOI18N
97     Texto3.setText("Empleado 2:");
98
99     Texto4.setFont(new java.awt.Font("Lucida Sans Unicode", 0, 12)); // NOI18N
100    Texto4.setText("Contenido de la mesa de platos:");
101
102    Texto5.setFont(new java.awt.Font("Lucida Sans Unicode", 0, 12)); // NOI18N
103    Texto5.setText("Cocinero 1:");
104

```

Podemos ver, también, que Eclipse nos marca el código con 3 colores en función de la cobertura. El verde nos indica que todas las ramas se han cubierto, el amarillo que faltan por cubrir y el rojo que el código no ha sido cubierto.

```

24 }
25
26⊕  public void dejarPedido(String pedido) throws InterruptedException {
27     cerrojo.lock();
28     All 2 branches covered.Elem == maximo) { //Si el ArrayList estĂ; lleno... Espera
29         lleno.await();
30     }
31     try {

```

```

30         cerrojo.unlock();
31     }
32 }
33
34⊕  public String cogerPedido() throws InterruptedException {
35     cerrojo.lock();
36     1 of 2 branches missed.Elem == 0) { //Si el ArrayList estĂ; vacio... Espera
37         vacio.await();
38     }
39     try {
40         String pedido = pedidosMostrador.get(0);
41         pedidosMostrador.remove(0);

```

Código Clase Cliente:

```
package PECL;

import java.util.Calendar;
import java.util.Date;

public class Cliente extends Thread {

    private String idCliente;
    private MostradorPedidos mpedidos;
    private Log log;

    public Cliente(String idCliente, MostradorPedidos mpedidos, Log log) {
        this.idCliente = idCliente;
        this.mpedidos = mpedidos;
        this.log=log;
    }

    public String getIdCliente() {
        return idCliente;
    }

    public void run() {
        String pedido;
        Date date=Calendar.getInstance().getTime();
        // Hay 200 clientes, ejecutarÃ;n su tarea y finalizarÃ;n
        try {
            String pedidol = idCliente + "-P1";
            mpedidos.dejarPedido(pedidol);
            int tiempo = (1000 + (int) (-500 * Math.random()));
            System.out.println("El " + idCliente + " deja un primer plato " + pedidol);
            pedido=date+": "+ El " + idCliente + " deja un primer plato " + pedidol;
            log.escribir(pedido);
            sleep(tiempo);

            String pedido2 = idCliente + "-P2";
            mpedidos.dejarPedido(pedido2);
            System.out.println("El " + idCliente + " deja un segundo plato " + pedido2);
            pedido=date+": "+ El " + idCliente + " deja un segundo plato " + pedido2;
            log.escribir(pedido);

        } catch (InterruptedException e) {
        }
    }
}
```

Código Clase ClienteTest:

```
package PECL;

import static org.junit.jupiter.api.Assertions.*;
import java.util.ArrayList;
import javax.swing.JTextField;
import org.junit.jupiter.api.Test;

class ClienteTest {

    @Test
    void testString() {
        String idCliente = "1234" ;
        MostradorPedidos mp = new MostradorPedidos(0, null);

        Log l = new Log();
        Cliente cl1 = new Cliente( idCliente, mp, l);

        String getid = cl1.getIdCliente();
        assertEquals("1234",getid);

    }
}
```

Resumen Documentación Ejercicio 2

Objetivo	Usar un programa Opensource Java + herramienta OpenSource de testeo de la lista.
Código	El mismo que en el ejercicio 1.
Entorno necesario	Eclipse + plugin SpotBugs (adicionalmente CheckStyle)
Conclusiones	Herramienta muy sencilla de instalar y probar, en cuestión de minutos podemos obtener resultados sin necesidad de ejecutar el código. Podemos complementar su uso con otros plugin u otro tipo de pruebas.

Usaremos la herramienta SpotBugs: <https://spotbugs.github.io/#using-spotbugs>, que se define a sí misma como un programa que usa análisis estáticos para buscar ‘bugs’ o errores en el código Java. Es el sucesor de Findbugs.

Su funcionamiento se resume en buscar e identificar instancias de patrones de errores/bugs mediante un análisis estático del código. SpotBugs es capaz de buscar alrededor de 400 patrones de errores, como por ejemplo, errores de null pointer, bucles recursivos infinitos, mal uso de las librerías de java, deadlocks, entre muchos otros que podemos encontrar definidos aquí: <https://spotbugs.readthedocs.io/en/latest/bugDescriptions.html>

A continuación resumiremos algunos puntos interesantes de SpotBugs:

- Herramienta muy rápida, que nos ayudará si tenemos muchos módulos, ya que ejecutar un análisis nos llevará muy pocos minutos.
- Aumentamos la calidad, nos permite tener una orientación sobre el estado de nuestro código en cuestión de minutos, permitiéndonos, por ejemplo, pasar a analizar más en profundidad determinados módulos que presentan una mayor concentración de errores.
- El coste es muy bajo, por lo que tendremos reducciones en QA.
- Fácil uso (gracias a la integración con el IDE Eclipse mediante el plugin).
- Requiere mucha memoria y una CPU rápida.
- Open Source con una gran comunidad que apoya la herramienta.
- Su uso es sencillo, cualquier programador con mayor o menor experiencia puede hacer uso de la herramienta.
- Ayuda a mantener un código limpio.
- Se puede complementar con otras herramientas, como los JUnit Test.

¿Qué es un análisis estático del código?

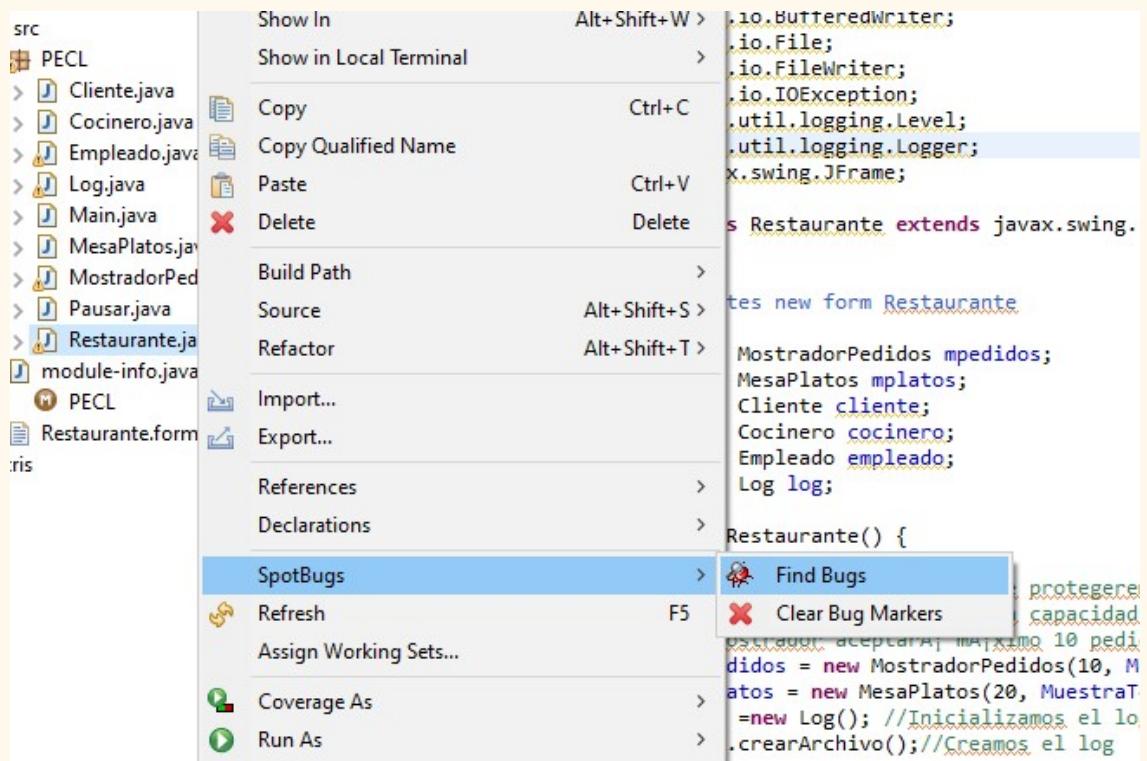
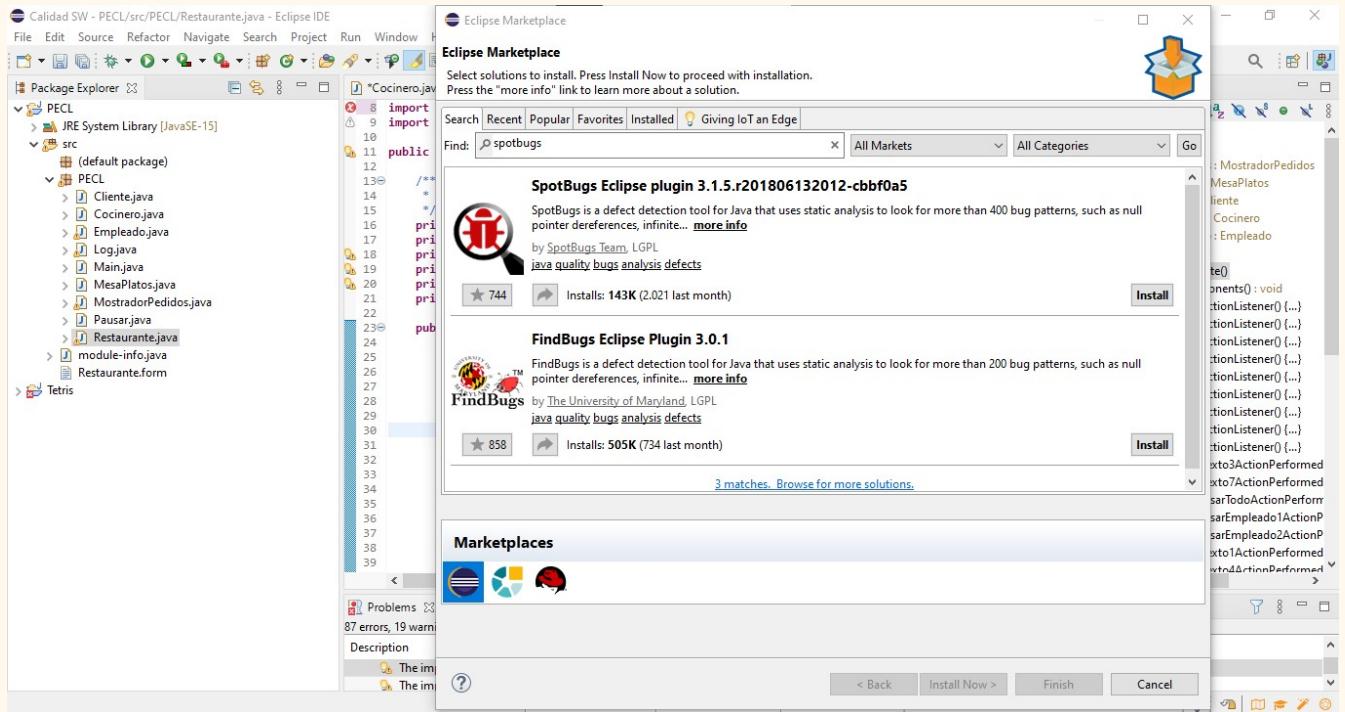
Evaluamos el código sin ejecutarlo.

Es un análisis que se realiza sobre un software sin necesidad de ejecutarlo, de modo puede revisar el diseño y el código, con el fin de garantizar la calidad antes de la ejecución. Esto es fundamental, ya que el proceso de pruebas del software debería de empezar junto a la fase del análisis de requisitos (lo antes posible) y no debería dejarse para el final. Por tanto, el uso de esta herramienta de análisis estática nos permitirá adelantarnos a otras pruebas.

Para empezar, vamos a destacar las 10 clasificaciones de errores/bugs que SpotBugs considera:

- ❖ Bad practice (BAD_PRACTICE)
- ❖ Correctness (CORRECTNESS)
- ❖ Experimental (EXPERIMENTAL)
- ❖ Internationalization (I18N)
- ❖ Malicious code vulnerability (MALICIOUS_CODE)
- ❖ Multithreaded correctness (MT_CORRECTNESS)
- ❖ Bogus random noise (NOISE)
- ❖ Performance (PERFORMANCE)
- ❖ Security (SECURITY)
- ❖ Dodgy code (STYLE)

Empezamos instalando el plugin de Eclipse desde el Marketplace, seguido de seleccionar la clase objetivo a analizar con click derecho > SpotBugs > FindBugs



Observamos que en el código, aparece un icono de 'bug' en las líneas que ha detectado y considera el plugin interesantes a analizar. A la izquierda tenemos una vista general de todos los bugs del proyecto y a la derecha, el comentario de la línea de código.

```

13 /**
14      * Creates new form Restaurante
15      */
16 private MostradorPedidos mpedidos;
17 private MesaPlatos mplatos;
18 private Cliente cliente;
19 private Cocinero cocinero;
20 private Empleado empleado;
21 private Log log;
22
23 public Restaurante() {
24     initComponents();
25     //Variables compartidas que protegeremos con cerrojos y con
26     //Un ArrayList limitarÃ¡ la capacidad mÃaxima
27     //Mostrador aceptarÃ¡ mÃaximo 10 pedidos y la mesa 20 pedidos
28     mpedidos = new MostradorPedidos(10, MuestraTexto1);
29     mplatos = new MesaPlatos(20, MuestraTexto4);
30     log = new Log(); //Inicializamos el log
31     log.crearArchivo(); //Creamos el log
32
33     // CREAMOS LOS HILOS
34     //200 Clientes + su identificador + variable compartida Mo
35     for (int i = 1; i <= 20; i++) {
36         Cliente cli = new Cliente("Cliente" + i, mpedidos, log);
37     }
38 }
39
40 //2 Empleados + variables compartidas Mesa de Platos y Most
41 Empleado emp1 = new Empleado("Empleado1", mpedidos, mplatos);
42 Empleado emp2 = new Empleado("Empleado2", mpedidos, mplatos);
43
44 //3 Cocineros + variable compartida Mesa de platos
45 Cocinero co1 = new Cocinero("Cocinero1", mplatos, MuestraTe;
46 Cocinero co2 = new Cocinero("Cocinero2", mplatos, MuestraTe;
47 Cocinero co3 = new Cocinero("Cocinero3", mplatos, MuestraTe;
48
49 //LANZAMOS LOS HILOS
50 emp1.start();
51 emp2.start();
52 co1.start();
53 co2.start();

```

Más detalle de los bugs encontrados. Podemos observar que 3 de ellos corresponden a hilos y 4 a los cerrojos.

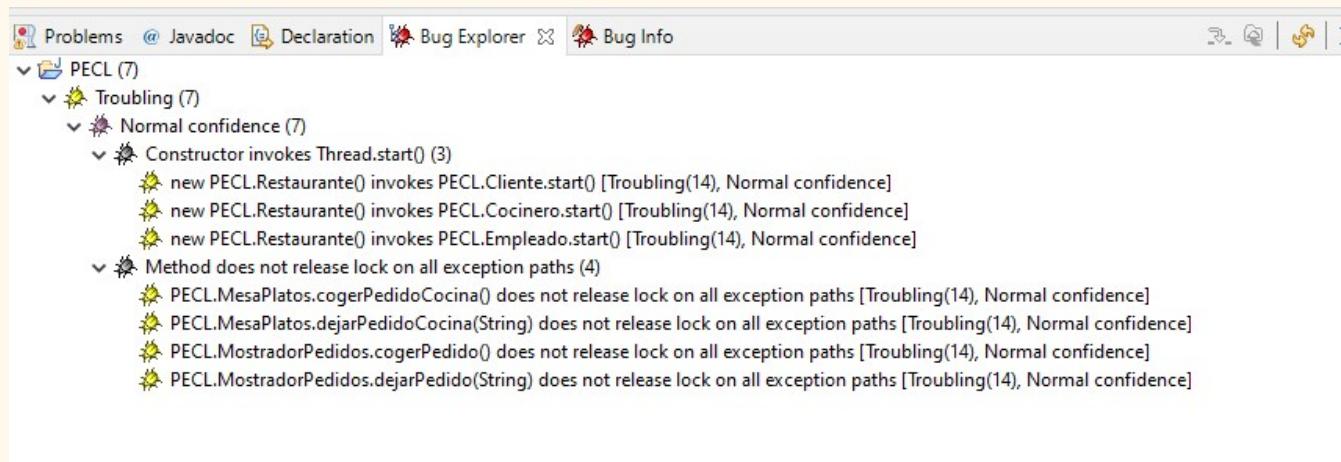


Imagen más en detalle de un bug que afecta al lock para el método cogerPedido().

```
39     }
40 }
41
42 public String cogerPedido() throws InterruptedException {
43     cerrojo.lock();
44     while (numElem == 0) { //Si el ArrayList estÃ¡ vacio... Espera
45         vacio.await();
46     }
47     try {
48         String pedido = pedidosMostrador.get(0);
49         pedidosMostrador.remove(0);
50         numElem--;
51         lleno.signal(); //El ArrayList ya no estÃ¡ lleno
52         text.setText(pedidosMostrador.toString());
53         return (pedido);
54     } finally {
55         cerrojo.unlock();
56     }
57 }
58
59 public ArrayList<String> getPedidosMostrador() {
60     return pedidosMostrador;
61 }
62
63
64
58             cerrojo.unlock();
59     }
60 }
61
62
63
64
58     cerrojo.unlock();
59 }
60 }
61
62
63
64
58
59 public String cogerPedido() throws InterruptedException {
60     PECLMostradorPedidos.cogerPedido() does not release lock on all exception paths [Troubling(14), Normal confidence]
61     while (numElem == 0) { //Si el ArrayList estÃ¡ vacio... Espera
62         vacio.await();
63     }
64     try {
65         String pedido = pedidosMostrador.get(0);
66         pedidosMostrador.remove(0);
67         numElem--;
68         lleno.signal(); //El ArrayList ya no estÃ¡ lleno
69         text.setText(pedidosMostrador.toString());
70         return (pedido);
71     } finally {
72         cerrojo.unlock();
73     }
74 }
75
76 public ArrayList<String> getPedidosMostrador() {
77     return pedidosMostrador;
78 }
79
80
81
82
83
84 }
```

Nota: adicionalmente también decidí hacer uso del plugin Checkstyle, que se centra en el diseño del código (detectando que casi el 99% del código no cumplía con los estándares que CheckStyle sigue).

The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The left sidebar is the Package Explorer, displaying the project structure under the 'src' folder. The main area is the code editor, showing Java code for the 'Cliente' class. A red underline is present under the constructor definition, indicating a warning or error. The code editor tabs show 'MostradorPedidos.java' and 'Cliente.java'. Below the code editor are several toolbars and a status bar with tabs for Problems, Javadoc, Declaration, Bug Explorer, and Bug Info.

```

package PECL;
import java.util.Calendar;
public class Cliente extends Thread {
    private String idCliente;
    private MostradorPedidos mpedidos;
    private Log log;
    public Cliente(String idCliente, MostradorPedidos mpedidos, Log log) {
        this.idCliente = idCliente;
        this.mpedidos = mpedidos;
        this.log = log;
    }
    public String getIdCliente() {
        return idCliente;
    }
    public void run() {
        String pedido;
        Date date=Calendar.getInstance().getTime();
        // Hay 200 clientes, ejecutarán su tarea y finalizarán
        try {
            String pedido1 = idCliente + "-P1";
            mpedidos.dejarPedido(pedido1);
            int tiempo = (1000 + (int) (-500 * Math.random()));
            System.out.println("El " + idCliente + " deja un primer plato " + pedido1);
            pedido=date+": "+ El "+ idCliente + " deja un primer plato " + pedido1;
            log.escribir(pedido);
            sleep(tiempo);
            String pedido2 = idCliente + "-P2";
            mpedidos.dejarPedido(pedido2);
        }
    }
}

```

This screenshot is identical to the one above, showing the same Eclipse IDE interface, project structure in the Package Explorer, and Java code in the code editor. The red underline under the constructor definition remains, indicating the same issue.

```

package PECL;
import java.util.Calendar;
public class Cliente extends Thread {
    private String idCliente;
    private MostradorPedidos mpedidos;
    private Log log;
    public Cliente(String idCliente, MostradorPedidos mpedidos, Log log) {
        this.idCliente = idCliente;
        this.mpedidos = mpedidos;
        this.log = log;
    }
    public String getIdCliente() {
        return idCliente;
    }
    public void run() {
        String pedido;
        Date date=Calendar.getInstance().getTime();
        // Hay 200 clientes, ejecutarán su tarea y finalizarán
        try {
            String pedido1 = idCliente + "-P1";
            mpedidos.dejarPedido(pedido1);
            int tiempo = (1000 + (int) (-500 * Math.random()));
            System.out.println("El " + idCliente + " deja un primer plato " + pedido1);
            pedido=date+": "+ El "+ idCliente + " deja un primer plato " + pedido1;
            log.escribir(pedido);
            sleep(tiempo);
            String pedido2 = idCliente + "-P2";
            mpedidos.dejarPedido(pedido2);
        }
    }
}

```

Bibliografía consultada:

Diapositivas de teoría de la asignatura.

<https://marketplace.eclipse.org/content/spotbugs-eclipse-plugin>

<https://spotbugs.github.io/#logo-license>

<https://spotbugs.readthedocs.io/en/latest/>

<https://www.educative.io/courses/java-unit-testing-with-junit-5/B892KY261z2>