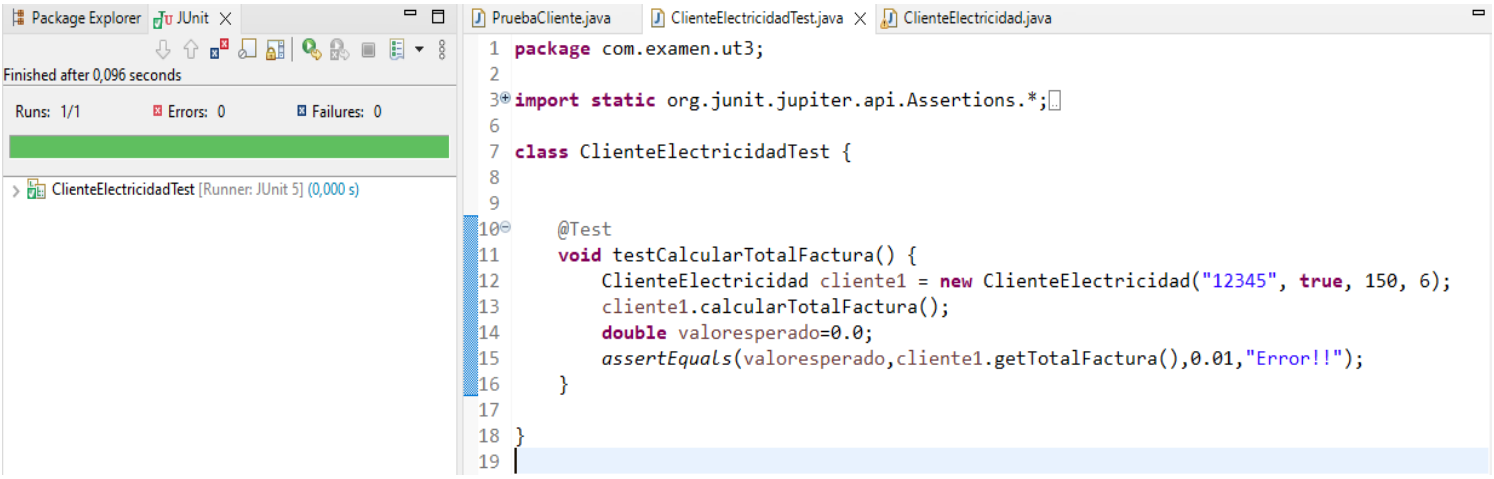
Nombre: Álvaro
Apellidos: Serradilla Otero
Entornos de desarrollo – 1ºDAM

EXAMEN UT3

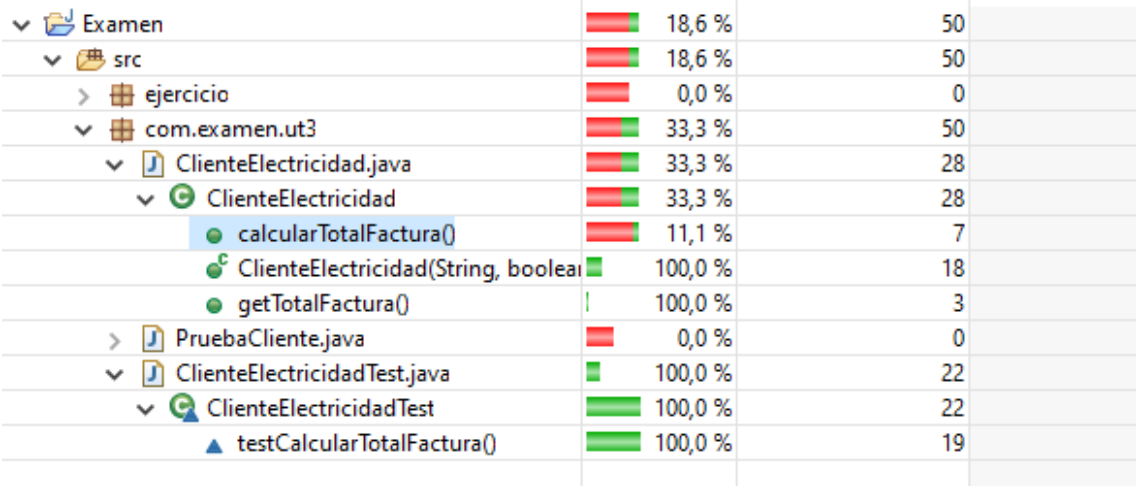
Ejercicio 3.

a) Diseña la prueba de cubrimiento para el método “calcularTotalFactura()”. Hazlo utilizando pruebas automatizadas.

NOTA: Se deberá configurar el parámetro “delta” como 0.01



b) Utilizar la herramienta "coverage" disponible en Eclipse para analizar y determinar la cobertura de código obtenida por las pruebas implementadas. Proporciona una captura de pantalla que muestre los resultados de la cobertura específicamente para el método “calcularTotalFactura()”, verificando así la efectividad de las pruebas diseñadas.



Ejercicio 4.

Dicho ejercicio nos pide realizar unas pruebas de caja negra ya que no tenemos en nuestra posesión el código de el programa, lo que nos lleva a realizar diversas pruebas de clases de equivalencia en las cuales se nos piden varios requisitos a cumplir:

a) Definir las clases de equivalencia,

b) 2 ejemplos de casos de prueba válidos

c) 2 casos de prueba no válidos que cubran dos clases no válidas para la inscripción en el evento académico.

Para ello crearemos dos tabla con dos casos para cada punto necesario:

Pruebas válidas

Numero de caso	Nombre Completo	Correo Electrónico	Tipo de Participante	Selección de Taller	Salida esperada	Salida obtenida
Caso 1	Lucas	lucas20@gmail.com	Estudiante	Taller A	“A sido inscrito”	“A sido inscrito”
Caso 2	Paula	paula40@gmail.com	Profesor		“A sido inscrito”	“A sido inscrito”

Pruebas no válidas

Numero de caso	Nombre Completo	Correo Electrónico	Tipo de Participante	Selección de Taller	Salida esperada	Salida obtenida
Caso 1	Jorge	jorge27@gmail.com	Padre	Taller N	“A sido inscrito”	“No ha sido inscrtipo”
Caso 2	Lucia	Lucia_gmail.com	Prefesional	Taller J	“A sido inscrito”	“No ha sido inscrito”

Ejercicio 5.

Para este ejercicio se nos proporciona un código con diversos fallos de compilación y otros cuantos fallos lógicos.

Los dos errores lógicos encontrados se tratan de dos operaciones echas en el código de CuentasBancarias el cual realizan acciones que no deben:

```
public CuentaBancaria(double saldoInicial) {
    this.saldo = saldoInicial+100000;
}
```

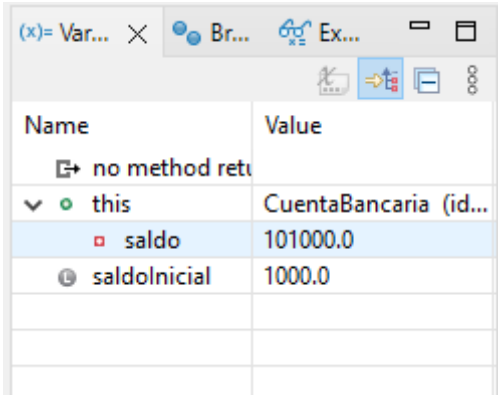
En esta imagen podemos ver como al saldo inicial se le añade de base 100000, para poder detectar este bug hemos idos directamente a utilizar el debug poniendo unos puntos de rotura a la hora de crear miCuenta:

```
9         // Creación de una cuenta bancaria con un saldo inicial de 1000
10        CuentaBancaria miCuenta = new CuentaBancaria(1000.0);
11        System.out.println("Saldo inicial: " + miCuenta.obtenerSaldo());
```

Tras esto nos introduciremos dentro de dicho paso para poder ver como las variables cambian detectando el problema de nuestro código:

```
14    public CuentaBancaria(double saldoInicial) {
15        this.saldo = saldoInicial+100000;
16    }
```

Gracias ha la ejecución de la linea de código vista con anterioridad veremos como las variables no tienen el resultado que nosotros esperamos:



Para solucionar dicho problema simplemente eliminaremos la suma que se le realiza al saldo inicial:

```
public CuentaBancaria(double saldoInicial) {
    this.saldo = saldoInicial;
}
```

Con esto el primer problema quedara solucionado.

El segundo problema lo podemos encontrar a la hora de ingresar dinero en miCuenta:

```
public void depositar(double cantidad) {
    if (cantidad < 0) {
        saldo += cantidad;
    }
}
```

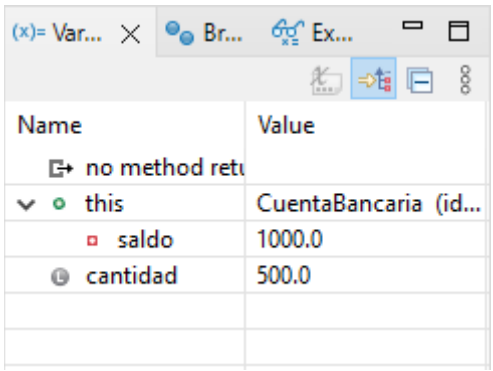
Para detectar este problema hemos vuelto a utilizar el debug colocando los puntos de rotura en la linea de código que utilizamos para depositar el dinero en miCuenta:

```
14        miCuenta.depositar(500.0);
15        System.out.println("Después de depositar 500: " + miCuenta.obtenerSaldo());
```

Tras esto nos introduciremos en nuestro método de depositar en el cual veremos que el fallo se encuentra en la condición:

```
22    public void depositar(double cantidad) {
23        if (cantidad < 0) {
24            saldo += cantidad;
25        }
26    }
```

Como podemos ver en la variable, al pedir que la cantidad sea menor que 0 para realizar el ingreso, nuestra variable saldo no aumentar con la cantidad, si no que se quedara igual como estaba.



Para solucionar esto simplemente tendríamos que cambiar la simbología de menor que a mayor que.

```
22    public void depositar(double cantidad) {
23        if (cantidad > 0) {
24            saldo += cantidad;
25        }
26    }
```

La diferencia entre los errores lógicos y los de compilación, son principal mente que si tienes un error lógico el programa va seguir a dar un resultado y el entorno de desarrollo no lo va a reconocer como un error en cambia los de compilación si te saldrá como un error que no te permitirá compilar el código.

Tras esto podremos comprobar si los resultados obtenidos son los esperados: