



### PRUEBAS UNITARIAS AUTOMATIZADAS CON JUNIT.

#### ACTIVIDADES POR DESARROLLAR:

1. Desarrollar prueba unitarias con JUNIT en el caso de estudio

#### EVIDENCIA(S) A ENTREGAR:

1. Realizar los casos de prueba del caso de estudio.

**DURACIÓN: 24 HORAS**

#### CONTROL DEL DOCUMENTO

|            | Nombre                       | Cargo      | Dependencia | Fecha      |
|------------|------------------------------|------------|-------------|------------|
| Autor (es) | JOSE FERNANDO GALINDO SUAREZ | INSTRUCTOR | CGMLTI      | 08/11/2020 |

#### CONTROL DE CAMBIOS (diligenciar únicamente si realizan ajustes al taller)

|            | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
|------------|--------|-------|-------------|-------|------------------|
| Autor (es) |        |       |             |       |                  |

## PRESENTACIÓN

JUnit es un framework Java para implementar test en Java. Se basa en anotaciones:






- **@Test**: indica que el método que la contiene es un test: expected y timeout.
- **@Before**: ejecuta el método que la contiene justo antes de cada test.
- **@After**: ejecuta el método que la contiene justo después de cada test.
- **@BeforeClass**: ejecuta el método (estático) que la contiene justo antes del primer test.
- **@AfterClass**: ejecuta el método (estático) que la contiene justo después del último test.
- **@Ignore**: evita la ejecución del tests. No es muy recomendable su uso porque puede ocultar test fallidos.

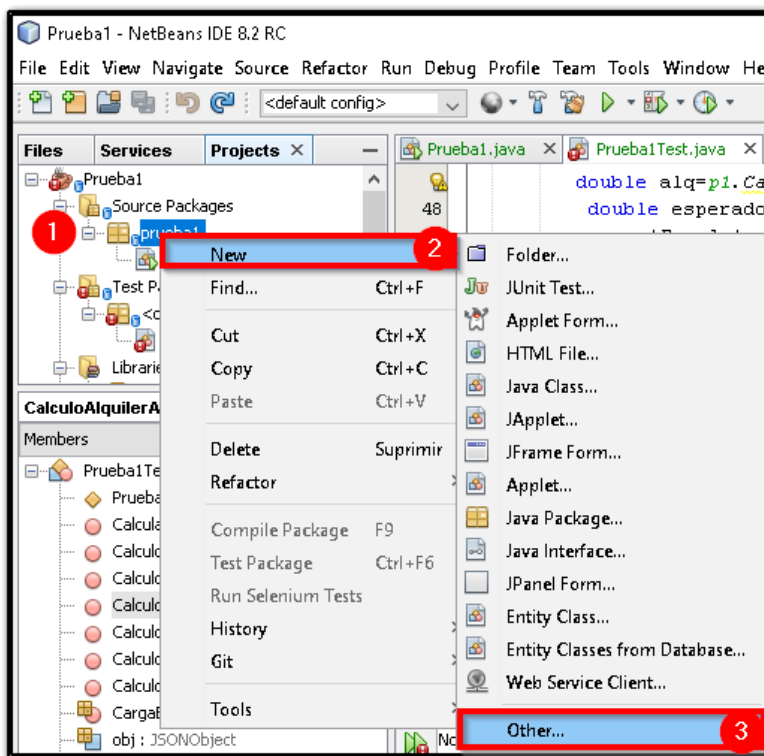
Las condiciones de aceptación de la prueba se implementan con los asserts. Los más comunes son los siguientes:

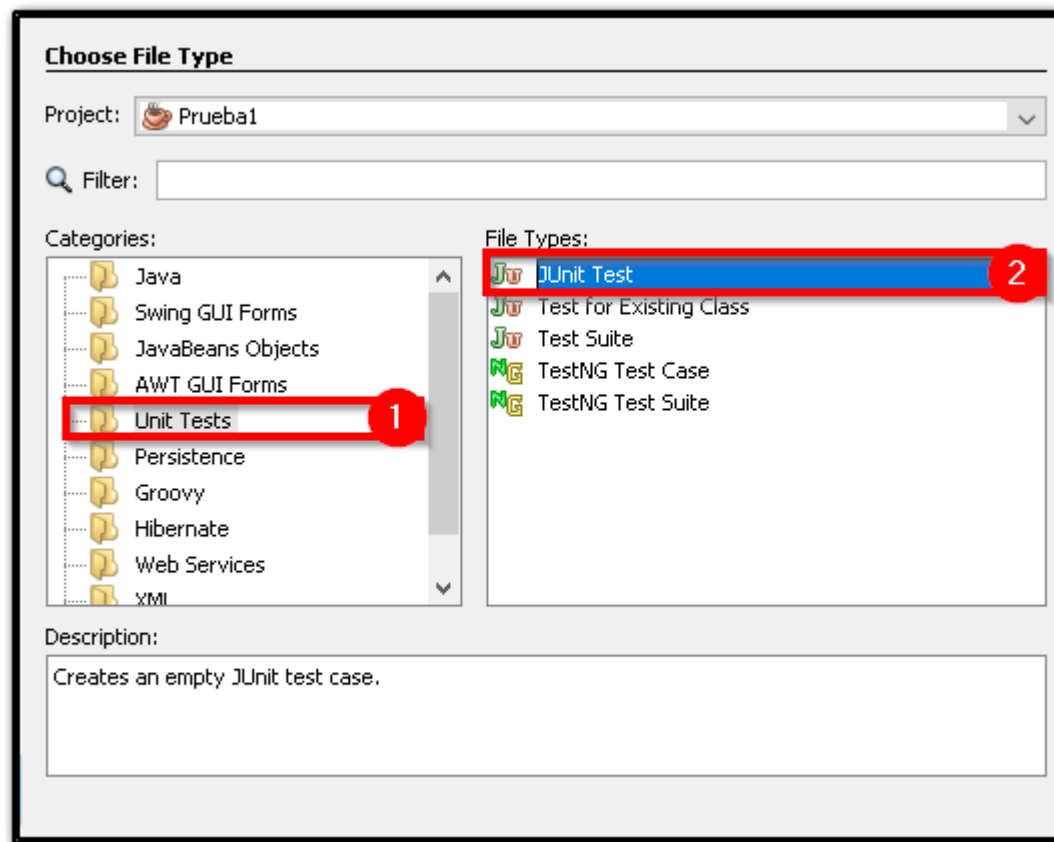
- **assertTrue/assertFalse** (condición a testear): Comprueba que la condición es cierta o falsa.
- **assertEquals/assertNotEquals** (valor esperado, valor obtenido). Es importante el orden de los valores esperado y obtenido.
- **assertNull/assertNotNull** (object): Comprueba que el objeto obtenido es nulo o no.
- **assertSame/assertNotSame**(object1, object2): Comprueba si dos objetos son iguales o no.
- **fail()**: Fuerza que la prueba termine con fallo. Se puede indicar un mensaje.

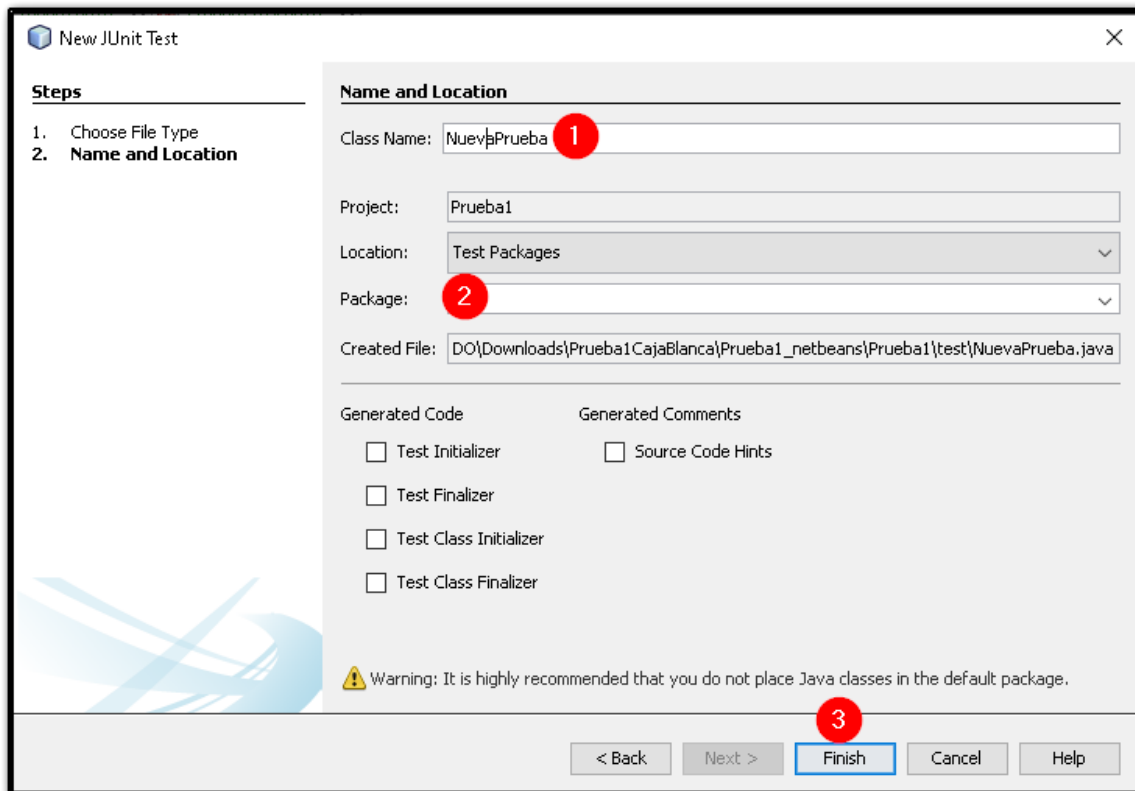
Además existen otras posibilidades más avanzadas:

- **Suites:** es una colección de un conjunto de test que se ejecutan de forma independiente (@RunWith(Suite.class),@SuiteClasses({}))
- **Runners:** Definen cómo ejecutar los test (@RunWith)
- **Test parametrizados:** test genéricos que se ejecutan con juegos de datos distintos (@Parameters)
- **Rules:** Pueden ejecutar código antes, después o dentro de los métodos (@Rule)

|   |                                       |
|---|---------------------------------------|
|  | Recomendaciones                       |
|  | El Aprendiz hace                      |
|  | El instructor explica                 |
|  | Evidencia por entregar                |
|  | Portafolio de evidencias del aprendiz |







**New JUnit Test**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name: NuevaPrueba **1**

Project: Prueba1

Location: Test Packages

Package: **2**

Created File: DO\Downloads\Prueba1CajaBlanca\Prueba1\_netbeans\Prueba1\test\NuevaPrueba.java

**Generated Code**

☐ Test initializer

☐ Test Finalizer

☐ Test Class Initializer

☐ Test Class Finalizer

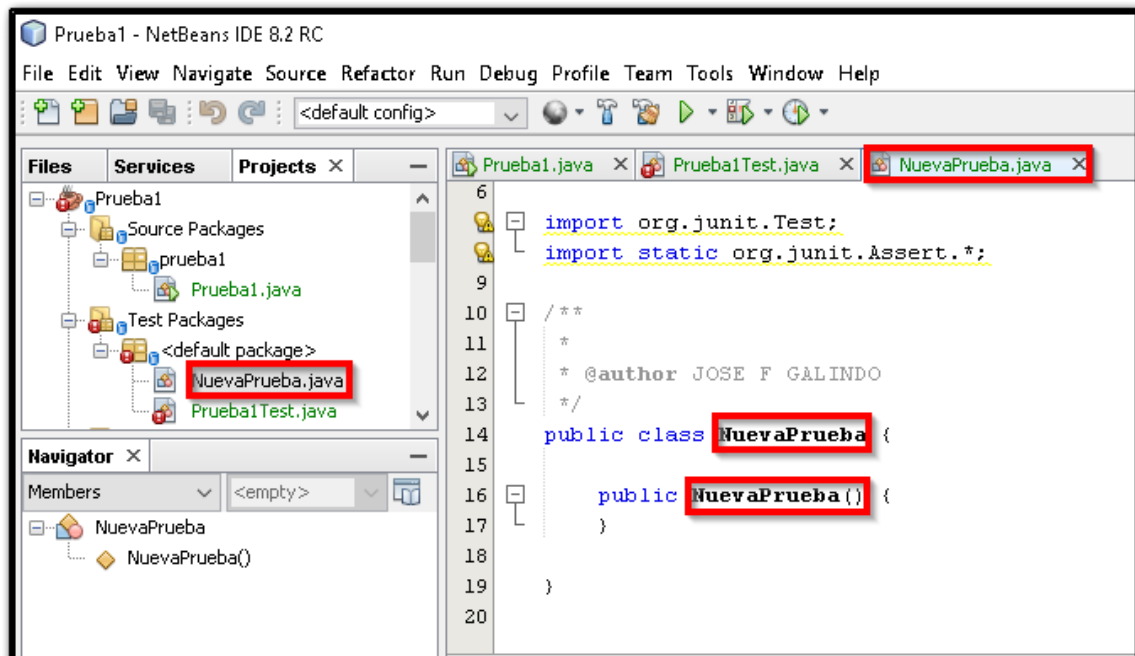
**Generated Comments**

☐ Source Code Hints

**3**

Warning: It is highly recommended that you do not place Java classes in the default package.

< Back   Next >   **Finish**   Cancel   Help



**Prueba1 - NetBeans IDE 8.2 RC**

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config>

**Files**   **Services**   **Projects** ×

Prueba1

- Source Packages
  - prueba1
    - Prueba1.java
  - Test Packages
    - <default package>
      - NuevaPrueba.java**
      - Prueba1Test.java

**Navigator** ×

Members   <empty>

- NuevaPrueba
  - NuevaPrueba()

**Prueba1Test.java** ×   **NuevaPrueba.java** ×

```

6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author JOSE F GALINDO
 */
public class NuevaPrueba {

    public NuevaPrueba() {

    }

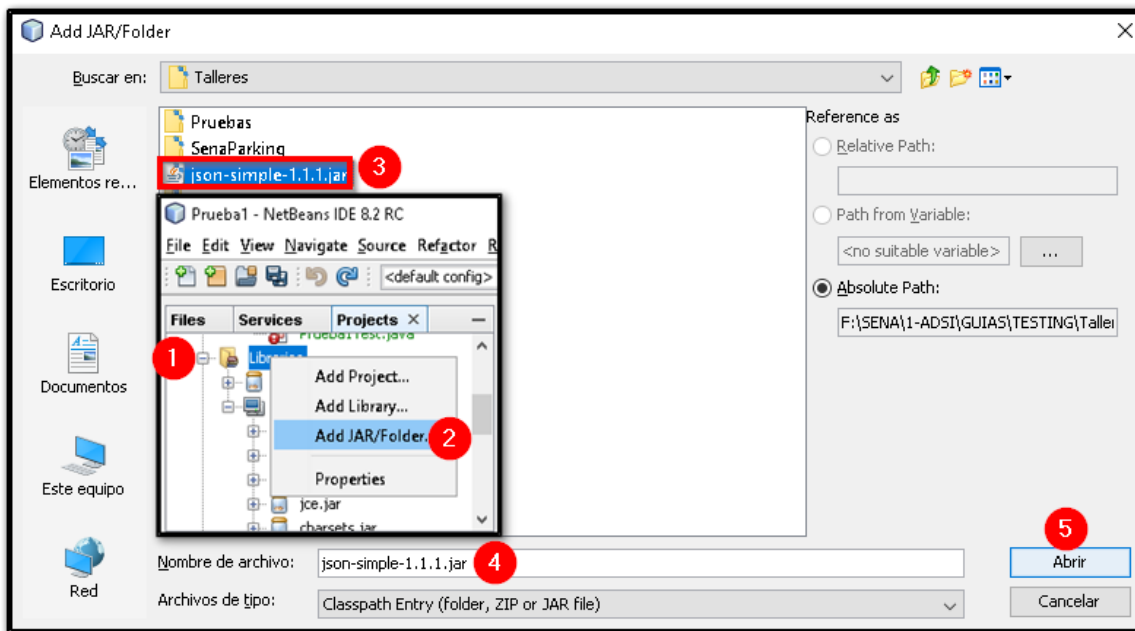
}

```

Descargar el JAR para trabajar con JSON:

<https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/json-simple/json-simple-1.1.1.jar>

Instalar el JAR descargado de esta manera:



Importar la librería para JSON

```
import org.json.simple.JSONArray;  
import org.json.simple.JSONObject;
```

Y declarar el objeto JSON

```
JSONObject obj=new JSONObject();
```

```
import org.junit.Test;
import static org.junit.Assert.*;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;

/**
 *
 * @author JOSE F GALINDO
 */
public class NuevaPrueba {

    public NuevaPrueba() {

    }

}
```

```
import org.junit.Test;
import static org.junit.Assert.*;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;

/**
 *
 * @author JOSE F GALINDO
 */
public class NuevaPrueba {

    static prueba1.Prueba1 p1=new prueba1.Prueba1(); //Cargamos la clase a probar
    JSONObject obj = new JSONObject(); //Creamo un objeto tipo JSON

    public NuevaPrueba() {

    }


}
```

```

public class Prueba1Test {
    1 static prueba1.Prueba1 pi=new prueba1.Prueba1(); //Cargamos la clase a probar
    JSONObject obj = new JSONObject(); //Creamo un objeto tipo JSON
    public Prueba1Test() {
        // Resultados esperados en el caso de prueba
        2 obj.put("equipos", 5);
        obj.put("diasal", 12);
        obj.put("diasad", 11);
        obj.put("opcion", 1);
        obj.put("alqui", 700000.0);
        obj.put("adicional", 525000.0);
        obj.put("domicilio", 24500.0);
        obj.put("descuentos", 21000.0);
        obj.put("establecimiento", 61250.0);
        obj.put("total", 1228500.0);
        CargaBasicos(); 3
    }
}

```

Se Construyen los casos de prueba de acuerdo con la siguiente tabla de resultados:

| MODALIDAD              | EQUIPOS | DIASAL                 | DIASAD | VLR ALQ                | VLRADIC | %   | OTO DIAS ADI | DESCUENTOS | SUBTOTAL | DOMICLIO | DENTROISTA | TOTAL A PAGAR |
|------------------------|---------|------------------------|--------|------------------------|---------|---|--------------|------------|----------|----------|------------|---------------|
| EN LA CIUDAD           | 5       | 4                      | 3      | 700000                 | 525000  | 4   | 21000        | 21000      | 1204000  | 24500    | 0          | 1228500       |
| FUERA DE LA CIUDAD     | 5       | 4                      | 3      | 700000                 | 525000  | 4   | 21000        | 21000      | 1204000  | 61250    | 0          | 1265250       |
| EN EL ESTABLECIMIENTO  | 5       | 4                      | 3      | 700000                 | 525000  | 4   | 21000        | 82250      | 1225000  | 0        | 61250      | 1142750       |
| EN LA CIUDAD           |         | FUERA DE LA CIUDAD     |        | EN EL ESTABLECIMIENTO  |         |  |              |            |          |          |            |               |
| EQUIPOS                | 5       | EQUIPOS                | 5      | EQUIPOS                | 5       |   |              |            |          |          |            |               |
| DIAS DE ALQUILER       | 4       | DIAS DE ALQUILER       | 4      | DIAS DE ALQUILER       | 4       |   |              |            |          |          |            |               |
| DIAS ADICIONALES       | 3       | DIAS ADICIONALES       | 3      | DIAS ADICIONALES       | 3       |   |              |            |          |          |            |               |
| VALOR ALQUILER         | 700000  | VALOR ALQUILER         | 700000 | VALOR ALQUILER         | 700000  |   |              |            |          |          |            |               |
| VALOR DIAS ADICIONALES | 525000  | VALOR DIAS ADICIONALES | 525000 | VALOR DIAS ADICIONALES | 525000  |   |              |            |          |          |            |               |
| DESCUENTOS             | 21000   | DESCUENTOS             | 21000  | DESCUENTOS             | 82250   |   |              |            |          |          |            |               |
| DOMICLIO               | 24500   | DOMICLIO               | 61250  | DOMICLIO               | 0       |   |              |            |          |          |            |               |
| TOTAL A PAGAR          | 0       | TOTAL A PAGAR          | 1E+06  | TOTAL A PAGAR          | 1142750 |   |              |            |          |          |            |               |



Construir un JSON para los alores esperados a comparar con los valores obtenidos al ejecutar los casos de prueba.

```
public class Prueba1Test {
    static prueba1.Prueba1 p1=new prueba1.Prueba1(); //Cargamos la clase a probar
    JSONObject obj = new JSONObject(); //Creamo un objeto tipo JSON
    public Prueba1Test() {
        // Resultados esperados en el caso de prueba
        obj.put("equipos", 5);
        obj.put("diasal", 12);
        obj.put("diasad", 11);
        obj.put("opcion", 1);
        obj.put("alqui", 700000.0);
        obj.put("adicional", 525000.0);
        obj.put("domicilio", 24500.0);
        obj.put("descuentos", 21000.0);
        obj.put("establecimiento", 250.0);
        obj.put("total", 1225000.0);
        CargaBasicos();
    }

    void CargaBasicos(){
        p1.equipos=(int)obj.get("equipos");
        p1.diasal=(int)obj.get("diasal");
        p1.diasad=(int)obj.get("diasad");
        p1.opcion=(int)obj.get("opcion");
    }
}
```

Las pruebas unitarias valida el funcionamiento de una unidad de código generada a partir de un requerimiento.

Utilizar la metodología Red-Green-Refactor que consiste en evaluar los casos de pruebas fallidos(Red), que se deberán ajustar para que sean validos(Green), mediante acciones sobre el código(Refactor), de acuerdo con los requisitos planteados.

Se procede a utilizar el patron de prueba de las res A:<sup>1</sup>

- Arrange (Organizar/Inicializa) => inicializa los objetos y establece los valores de los datos que vamos a utilizar en la prueba que lo contiene.
- Act (Actuar) => realiza la llamada al método a probar con los parámetros preparados para tal fin.
- Assert (Confirmar/Comprobar) => comprueba que el método de pruebas ejecutado se comporta tal y como teníamos previsto que lo hiciera.

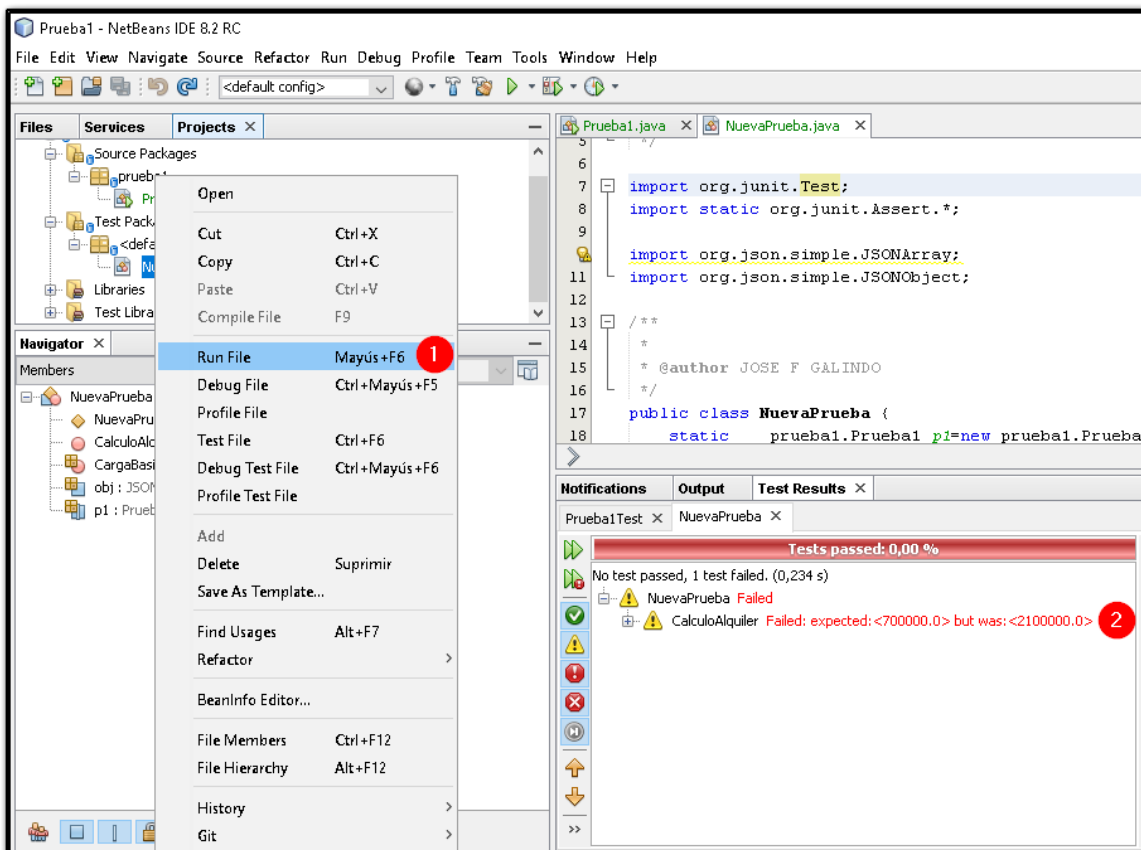
<sup>1</sup> <https://geeks.ms/jorge/2018/08/25/unit-testing-y-el-patron-aaa/>



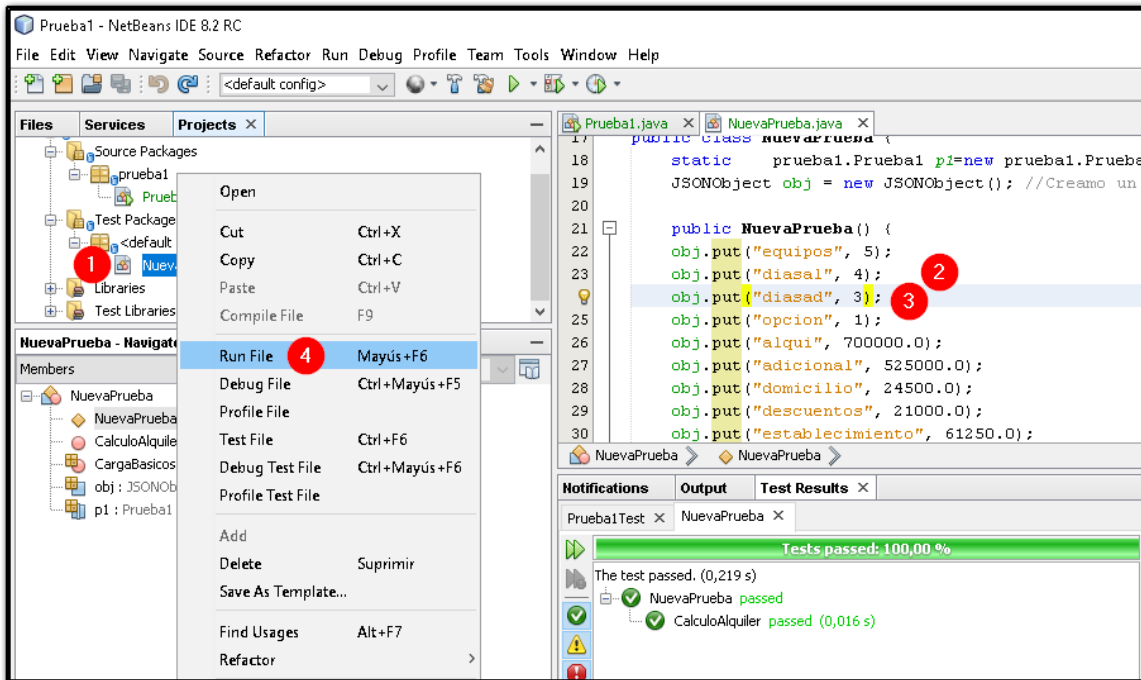
```

@Test
public void CalculoAlquiler() {
    // prueba1.Prueba1 p1=new prueba1.Prueba1();
    double alq=p1.CalculoAlquiler();//7000000 1
    double esperado=(double) obj.get("alqui"); 2
    assertEquals(esperado, alq, 0.0); 3
    System.out.println("Valor alquiler="+alq+"/"+esperado);
}
    
```

Se ejecuta el caso de prueba y se obtiene el resultado fallido como se muestra en la gráfica.



Se procede a cambiar equipos a 5, días de alquiler a 4, días adicionales a 3 y ejecutar la clase de prueba.



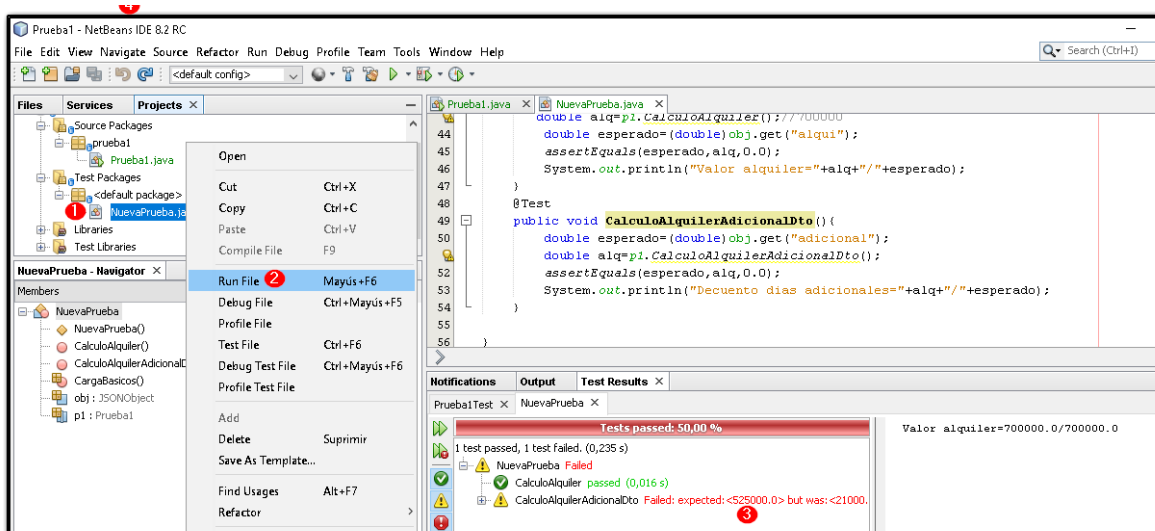
Crear el caso de prueba “CalculoAlquilerAdicionalDto”

```
@Test
public void CalculoAlquilerAdicionalDto() {
    double esperado=(double) obj.get("adicional");
    double alq=pi.CalculoAlquilerAdicionalDto();
    assertEquals(esperado,alq,0.0);
    System.out.println("Decuento dias adicionales="+alq+"/"+esperado);
}
```

Ejecutamos.



Servicio Nacional de Aprendizaje  
Formato Taller  
Centro de Gestión de Mercados, Logística y Tecnologías de la Información.



Adicionar el parámetro “dtoadicional” a 21000.

```
obj.put("equipos", 5);  
obj.put("diasal", 4);  
obj.put("diasad", 3);  
obj.put("opcion", 1);  
obj.put("alqui", 700000.0);  
obj.put("adicional", 525000.0);  
obj.put("domicilio", 24500.0);  
obj.put("dtoadicional", 21000.0); 1  
obj.put("descuentos", 21000.0);  
obj.put("establecimiento", 61250.0);  
obj.put("total", 1228500.0);
```

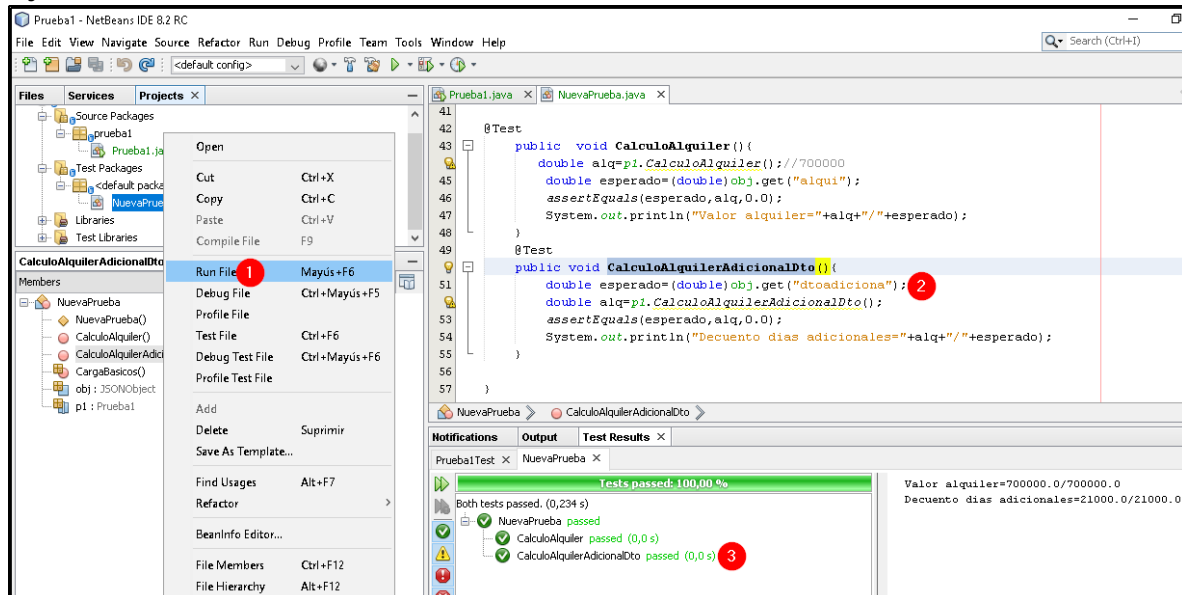
Modificar a “CalculoAlquilerAdicionalDto”

```
@Test  
public void CalculoAlquilerAdicionalDto() {  
    double esperado=(double)obj.get("dtoadicional"); 1  
    double alq=pi.CalculoAlquilerAdicionalDto();  
    assertEquals(esperado,alq,0.0);  
    System.out.println("Decuento dias adicionales="+alq+"/"+esperado);  
}
```



Servicio Nacional de Aprendizaje  
Formato Taller  
Centro de Gestión de Mercados, Logística y Tecnologías de la Información.

### Ejecutamos de nuevo.



Realizado por el instructor José Fernando Galindo Suárez  
[jgalindos@sena.edu.co](mailto:jgalindos@sena.edu.co) 2020

