



**Universidad Nacional de Lanús**  
**Departamento de Desarrollo Productivo y**  
**Tecnológico**

**Trabajo Practico N° 1**

**Carrera:** Licenciatura en Sistemas.

**Materia:** Prueba de Software.

**Año:** 2024.

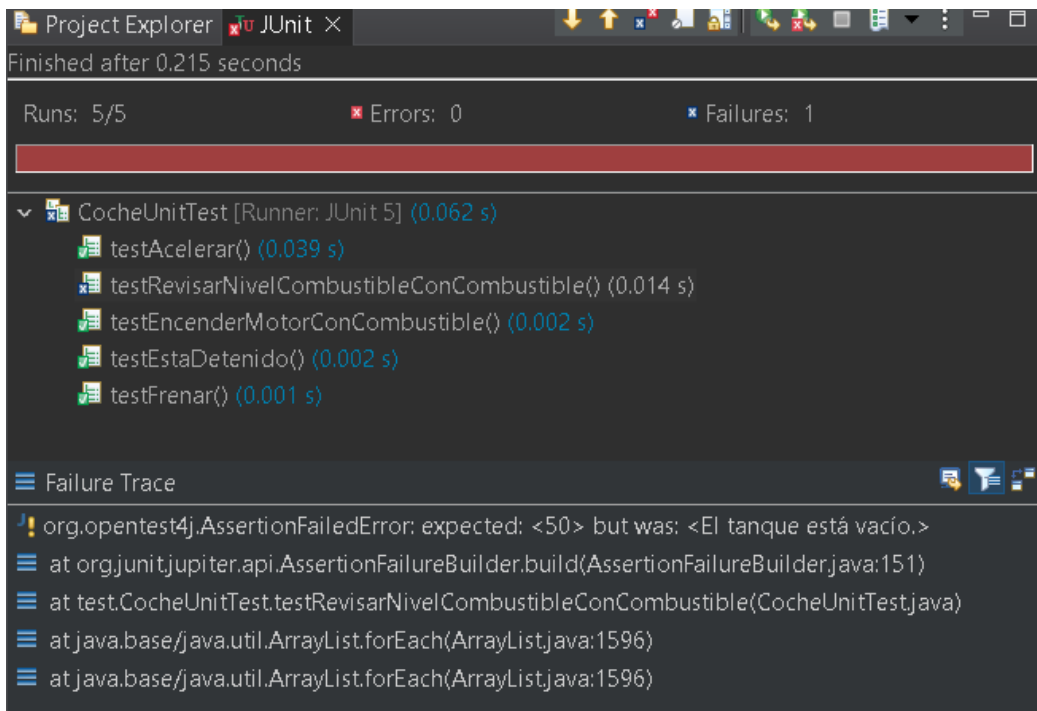
**Docente:** M. Ing. Eduardo Diez.  
Lic. Pablo San Román.

**Integrantes:** Alex Cadima. DNI: 42434535  
Ignacio Mansilla. DNI: 41557864  
Di alessio Martin. DNI: 38841167

**Repositorio:** <https://github.com/Alexcadima/Prueba-Software.git>

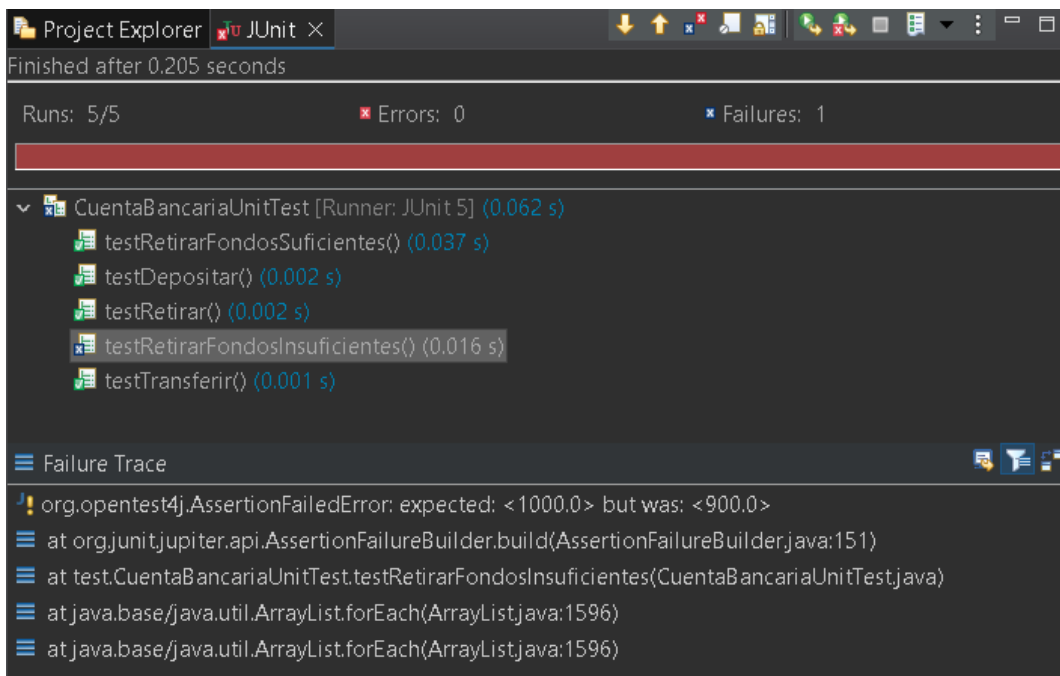
# Ejercicio N°1

## Clase Coche:



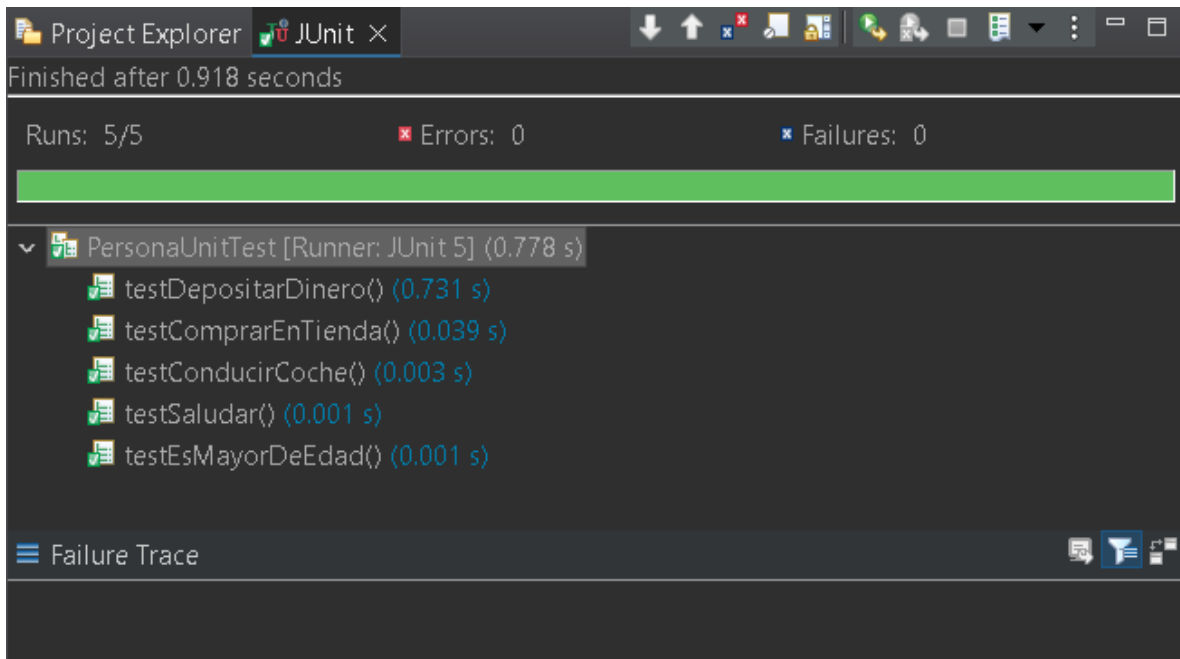
El método arroja un error, dicho error se debe por la espera de un nivel de combustible igual a 50 pero dicho coche arranca con un nivel de combustible en 0.

## Clase CuentaBancaria:



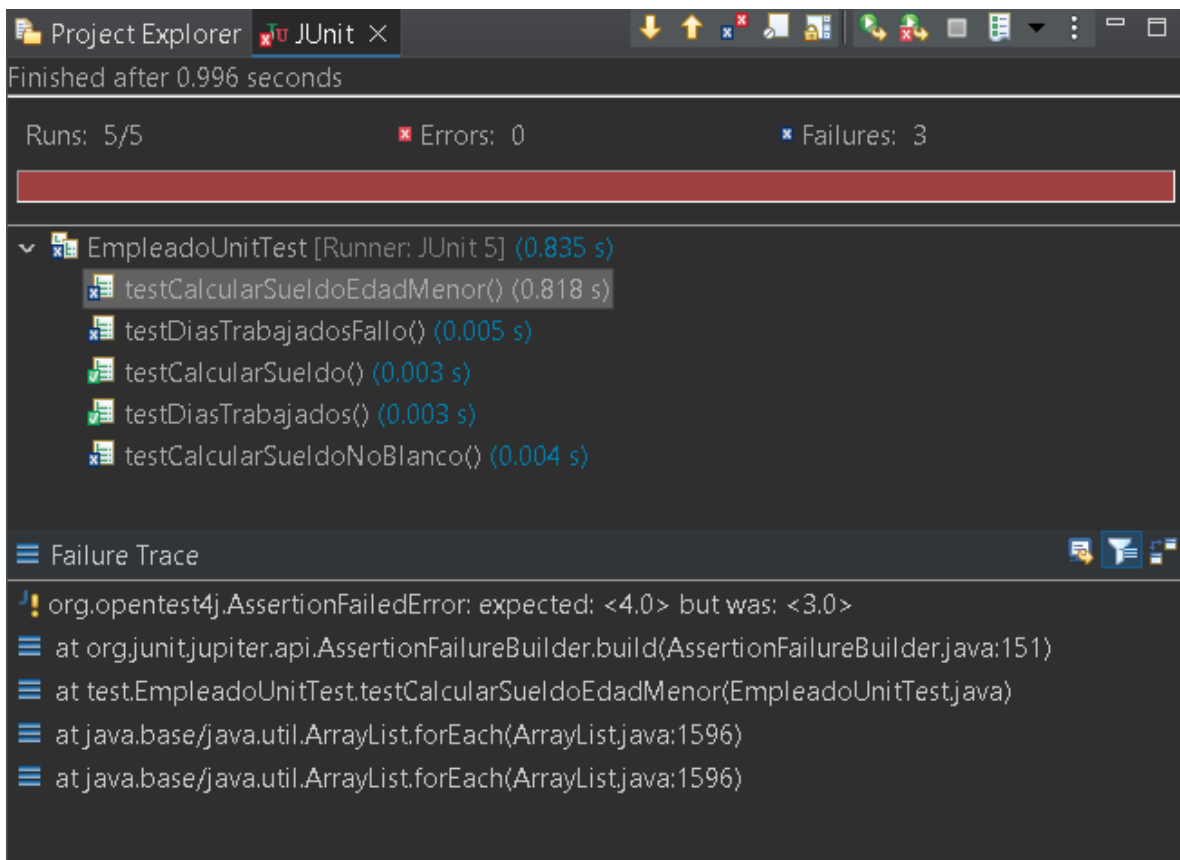
El error se debe al esperar un resultado que sabemos que no obtendremos.

## Clase Persona:



Sin problemas en esta clase

## Clase Empleado:

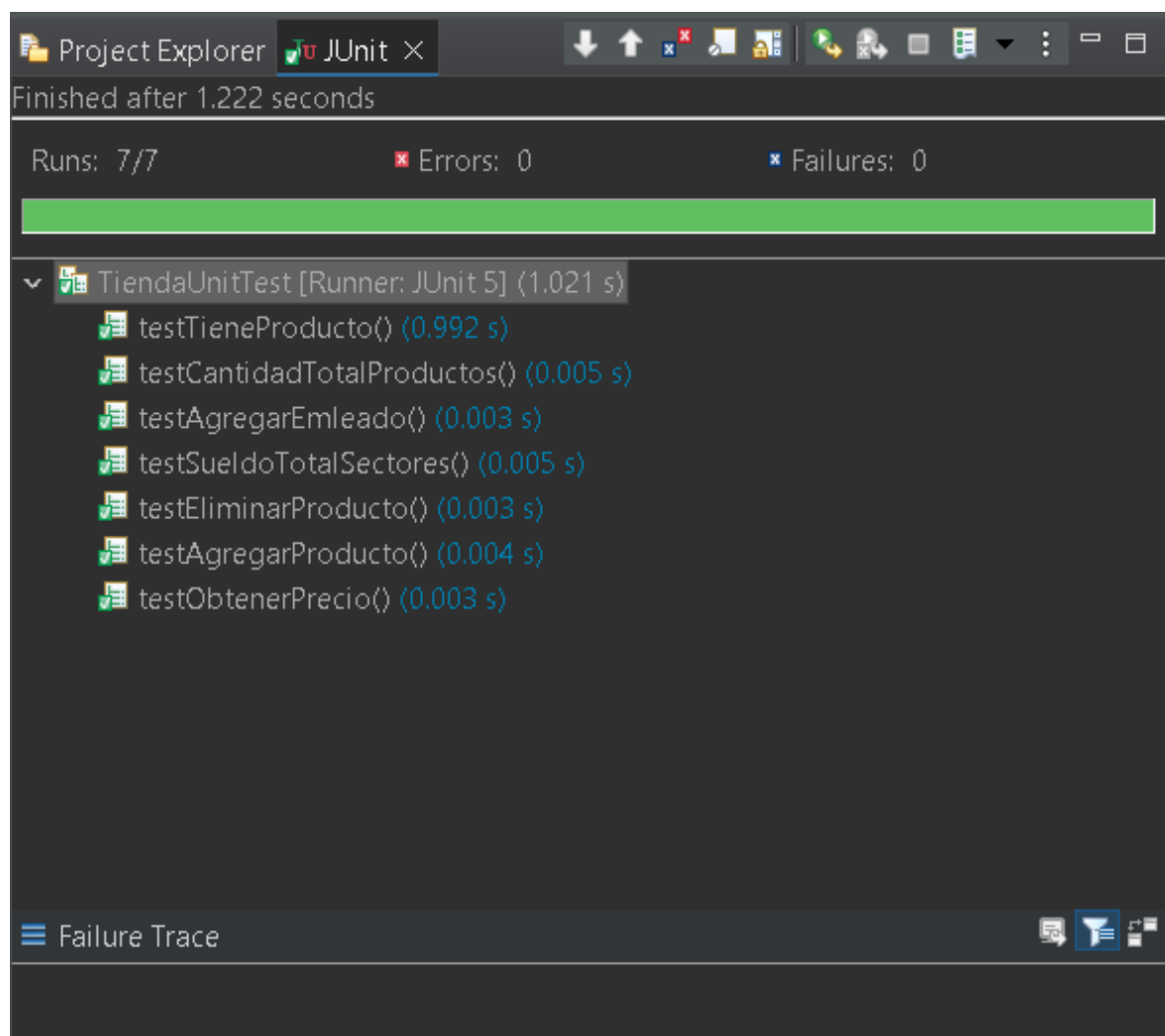


Los tres métodos dan error al esperar resultados erróneos, como por ejemplo esperar 10 días trabajados cuando solo fue 1.

Otro método es por no cumplir con los requisitos de la edad para en este caso tener 19 cuando piden 30 o mayor.

El ultimo método es esperar un sueldo que se multiplique por 2 pero al no estar en blanco no nos dará el resultado esperado.

### Clase Tienda:

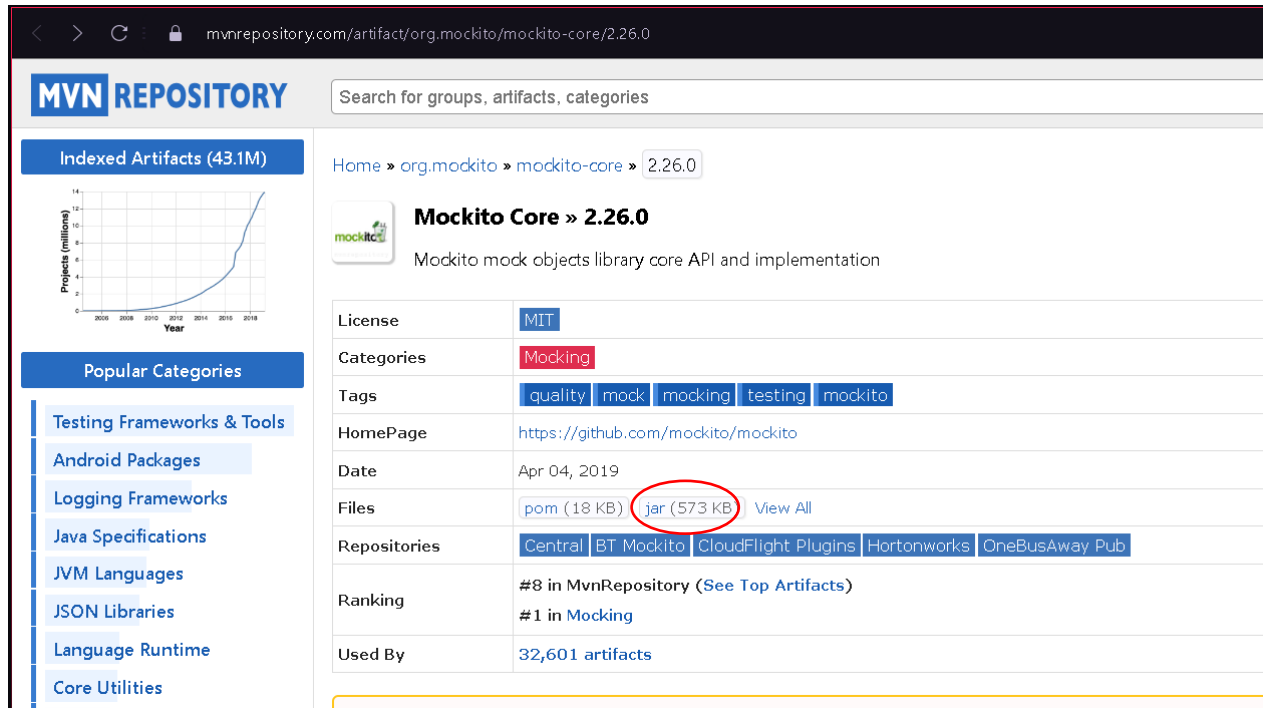


Sin ningún problema.

## Ejercicio N°2

### Instalación de Mockito

Primero descargamos la librería mockito y sus dependencias. Ingresamos a la página Maven Repository (<https://mvnrepository.com/artifact/org.mockito/mockito-core>) y descargamos la versión de mockito core que nos guste:



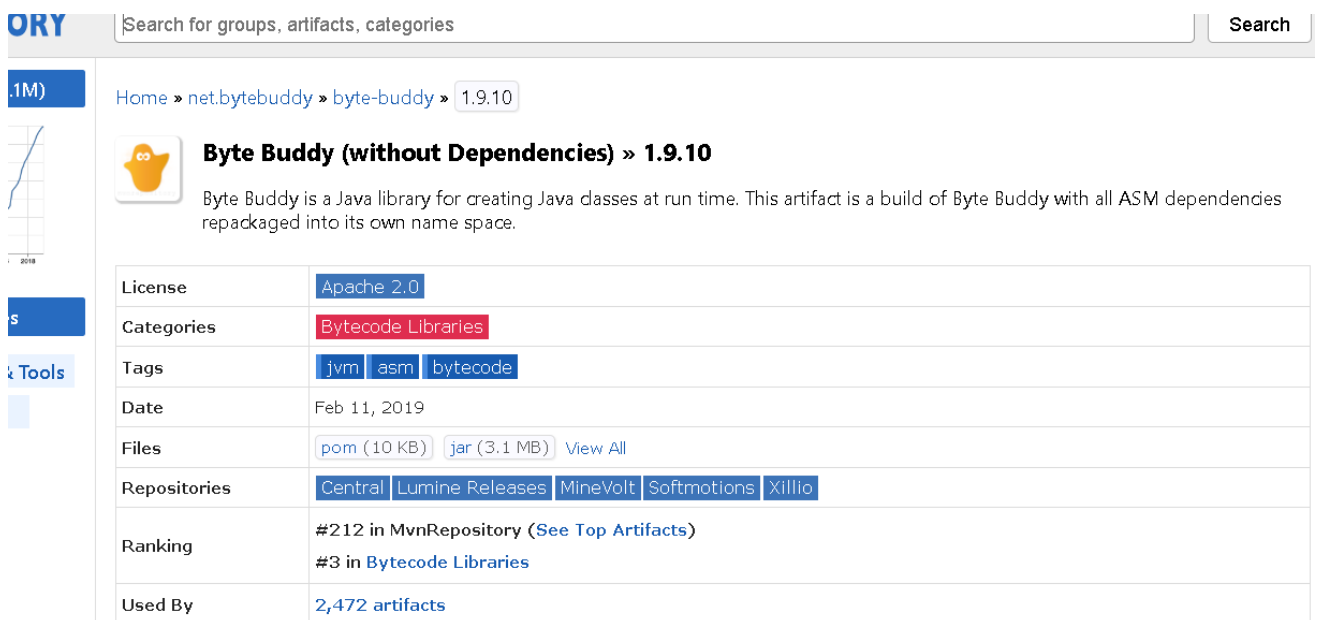
mvnrepository.com/artifact/org.mockito/mockito-core/2.26.0

**Mockito Core » 2.26.0**

Mockito mock objects library core API and implementation

License	MIT
Categories	Mocking
Tags	quality mock mocking testing mockito
HomePage	<a href="https://github.com/mockito/mockito">https://github.com/mockito/mockito</a>
Date	Apr 04, 2019
Files	<a href="#">pom (18 KB)</a> <a href="#">jar (573 KB)</a> <a href="#">View All</a>
Repositories	Central BT Mockito CloudFlight Plugins Hortonworks OneBusAway Pub
Ranking	#8 in MvnRepository (See Top Artifacts) #1 in Mocking
Used By	32,601 artifacts

Le damos a la opción Jar la cual nos servirá para eclipse y en la misma página buscamos y descargamos sus dependencias “Byte Buddy”, “Byte Buddy Java Agent” y “Objenesis”.



Search for groups, artifacts, categories

Home » net.bytebuddy » byte-buddy » 1.9.10

**Byte Buddy (without Dependencies) » 1.9.10**

Byte Buddy is a Java library for creating Java classes at run time. This artifact is a build of Byte Buddy with all ASM dependencies repackaged into its own name space.


License	Apache 2.0
Categories	Bytecode Libraries
Tags	jvm asm bytecode
Date	Feb 11, 2019
Files	<a href="#">pom (10 KB)</a> <a href="#">jar (3.1 MB)</a> <a href="#">View All</a>
Repositories	Central Lumine Releases MineVolt Softmotions Xillio
Ranking	#212 in MvnRepository (See Top Artifacts) #3 in Bytecode Libraries
Used By	2,472 artifacts

RY

Search for groups, artifacts, categories

Search

Home » net.bytebuddy » byte-buddy-agent » 1.9.10



Byte Buddy Agent » 1.9.10


The Byte Buddy agent offers convenience for attaching an agent to the local or a remote VM.

License	Apache 2.0
Categories	Bytecode Libraries
Tags	jvm agent asm bytecode
Date	Feb 11, 2019
Files	<a href="#">pom (5 KB)</a> <a href="#">jar (42 KB)</a> <a href="#">View All</a>
Repositories	<a href="#">Central</a> <a href="#">CubaWork</a> <a href="#">EEA SK</a> <a href="#">GroovyPlugins</a>
Ranking	#745 in MvnRepository (See Top Artifacts) #9 in Bytecode Libraries
Used By	705 artifacts
Vulnerabilities	Vulnerabilities from dependencies: <a href="#">CVE-2020-15250</a>

OSITORY

Search for groups, artifacts, categories

acts (43.1M)



categories

works & Tools

ges

eworks


ions

ps

;

itime

Home » org.objenesis » objenesis » 2.6



Objenesis » 2.6

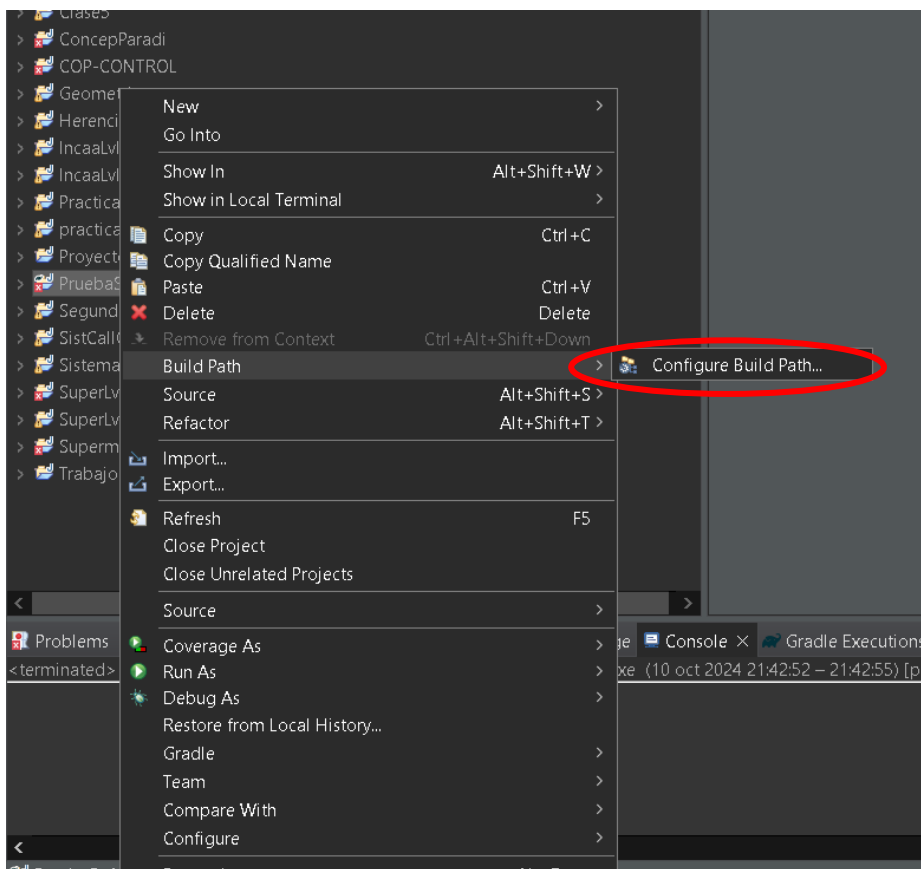
A library for instantiating Java objects

License	Apache 2.0
Categories	Reflection Libraries
Tags	reflection
HomePage	<a href="http://objenesis.org">http://objenesis.org</a>
Date	Jun 20, 2017
Files	<a href="#">pom (2 KB)</a> <a href="#">jar (54 KB)</a> <a href="#">View All</a>
Repositories	<a href="#">Central</a> <a href="#">CubaWork</a> <a href="#">Fit2Cloud</a> <a href="#">Redhat GA</a>
Ranking	#208 in MvnRepository (See Top Artifacts) #4 in Reflection Libraries
Used By	2,525 artifacts

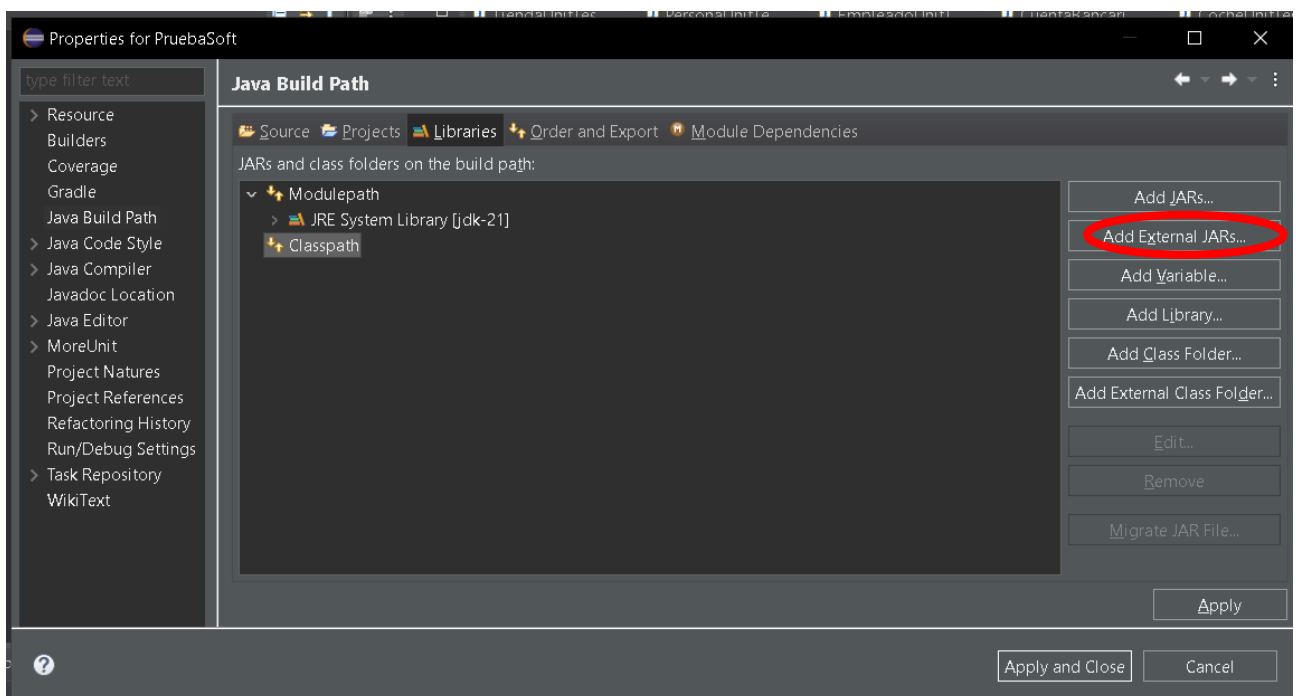
Una vez hecho todo esto nos quedara los siguientes archivos:

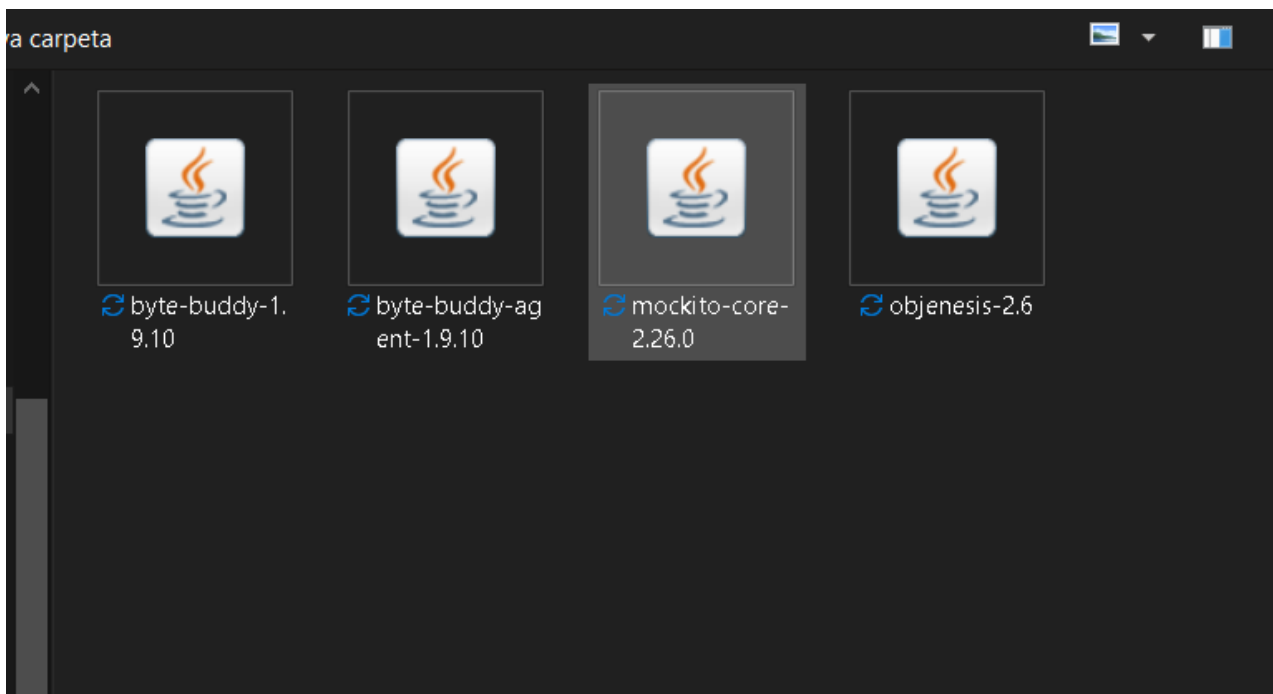


Ahora abrimos eclipse, hacemos click derecho a nuestro proyecto, vamos a la opción “Build Path” y entramos a su configuración

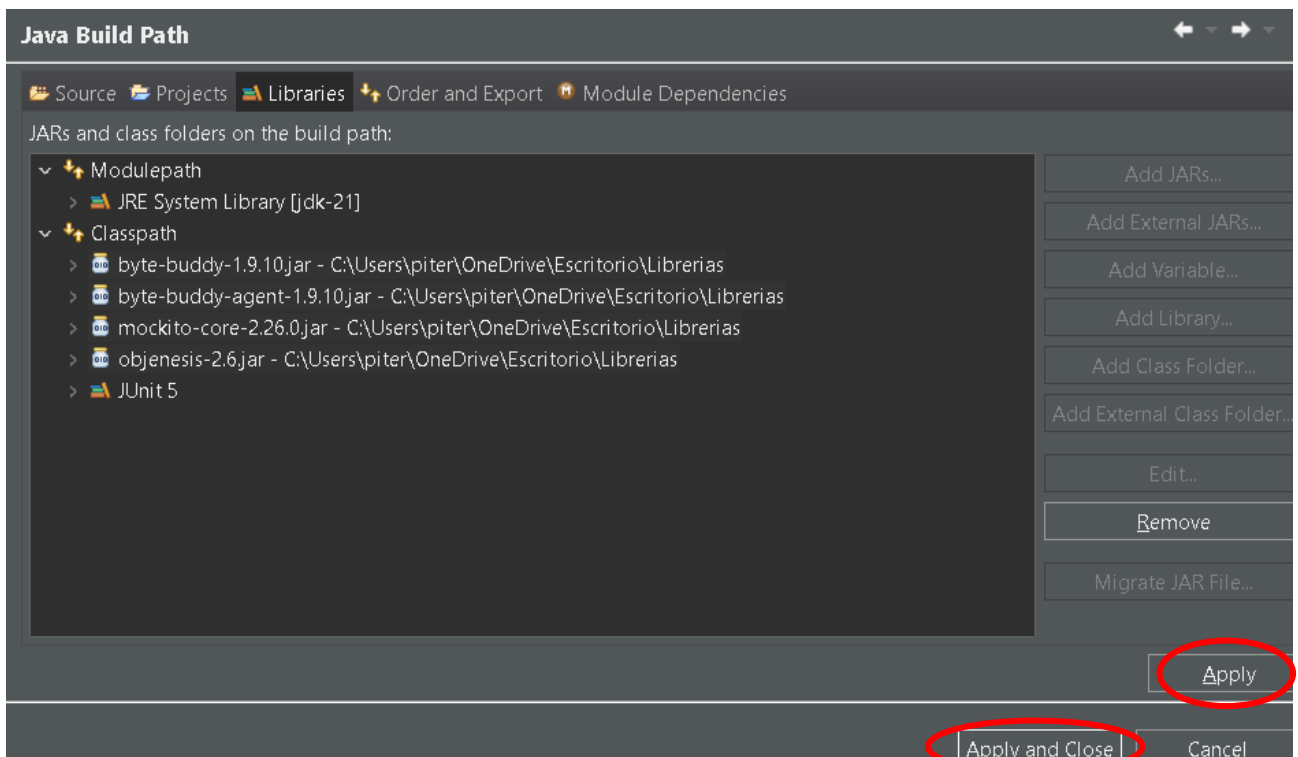


Se nos abrirá la siguiente ventana y le daremos a la opción que dice “add external JARs”





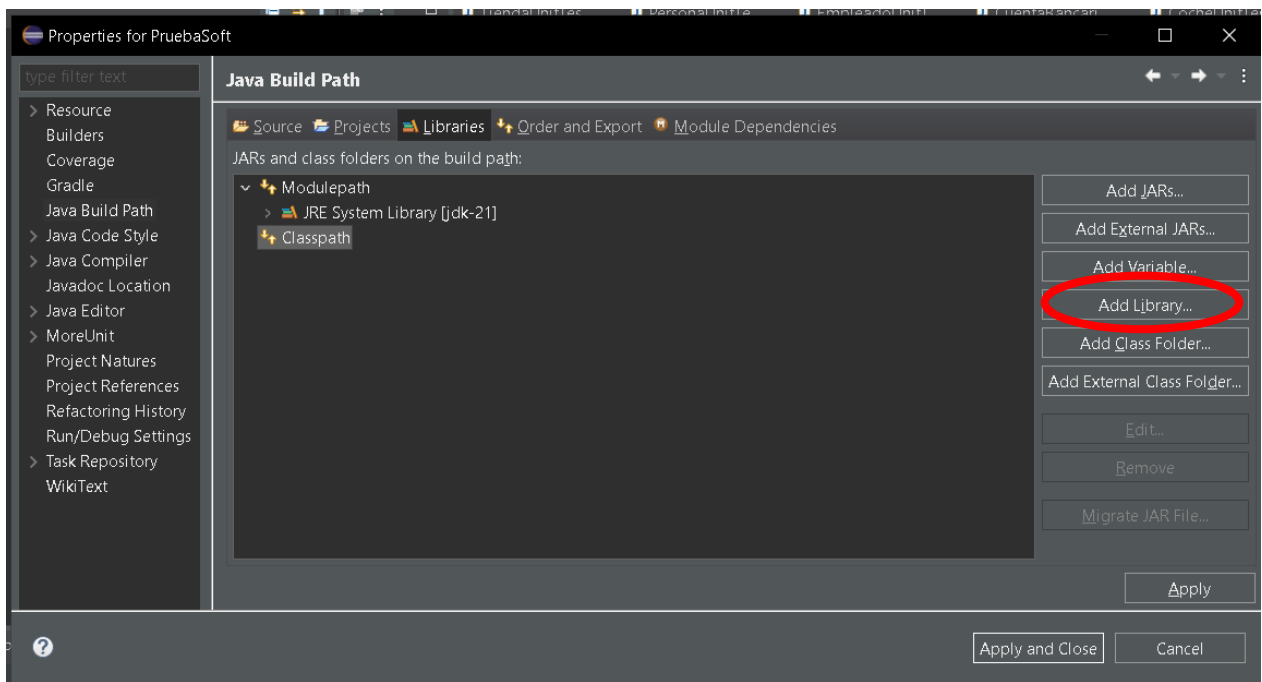
Agregamos las 4 librerías y le damos a Apply y Apply and Close. Con esto terminamos de configurar mockito para poder ahora usarlo sin ningún problema.



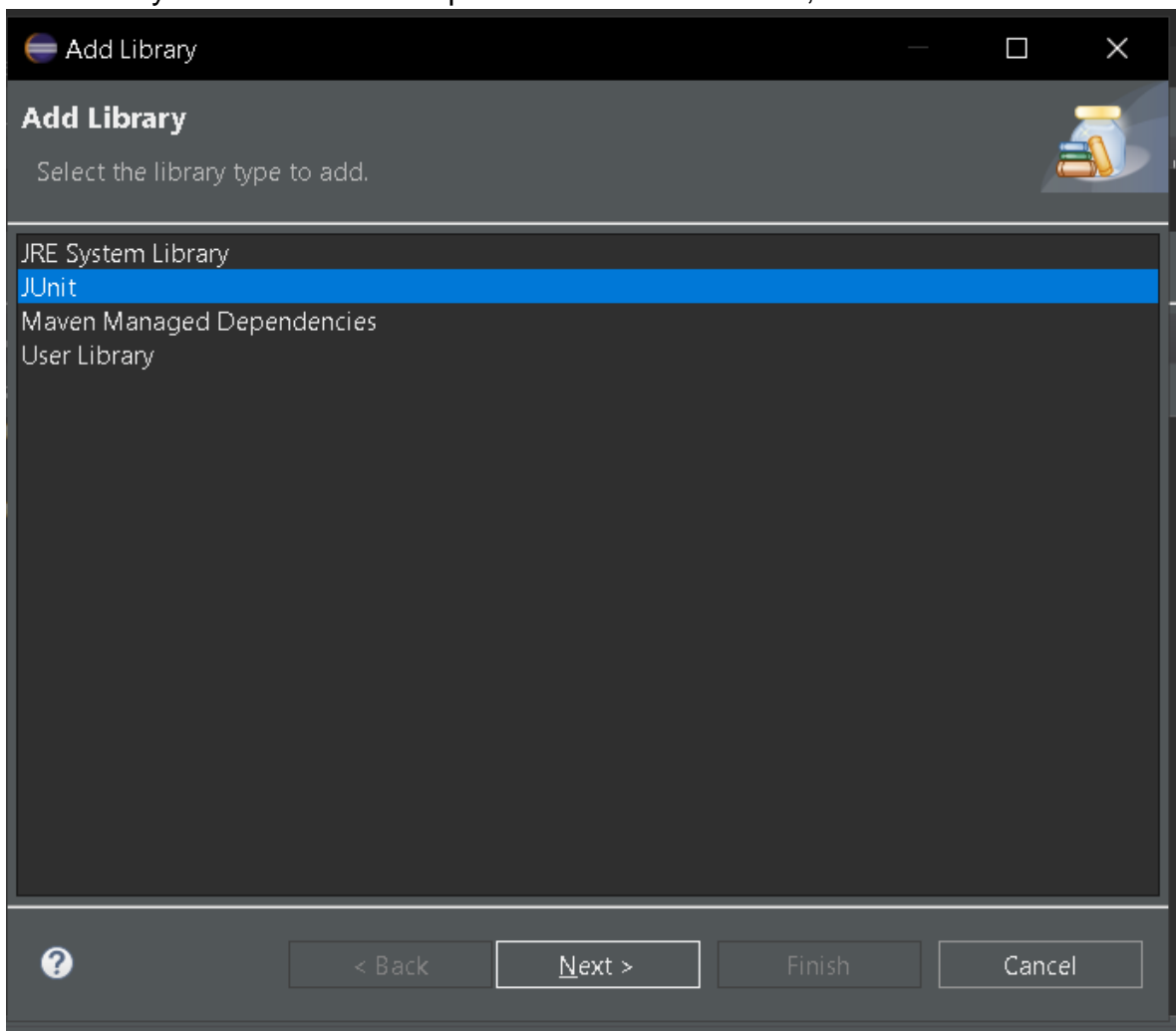
### Importante

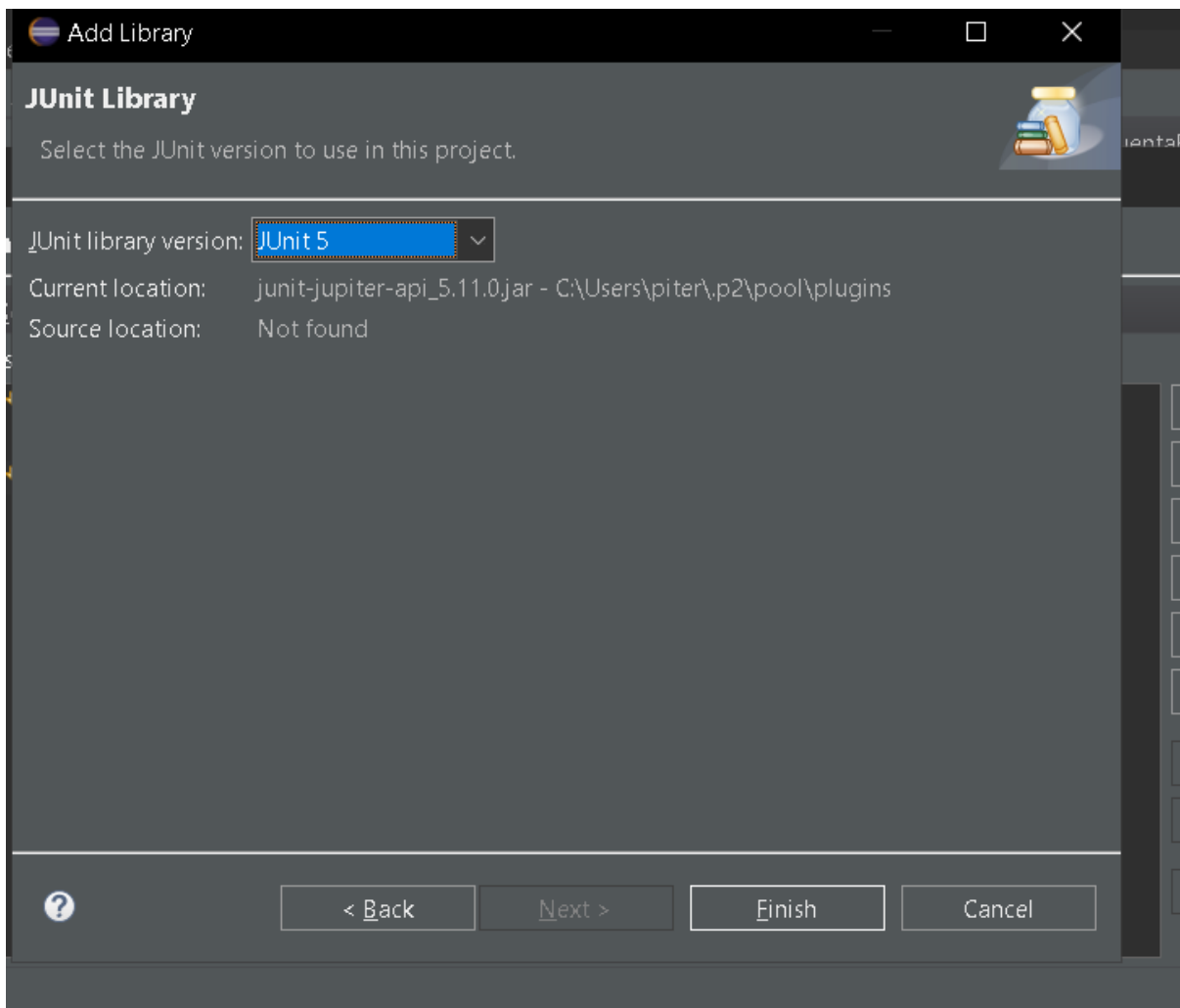
Si no tiene la librería de Junit 5 agregada lo hacemos desde la misma ventana pero entramos desde la opción Add Library



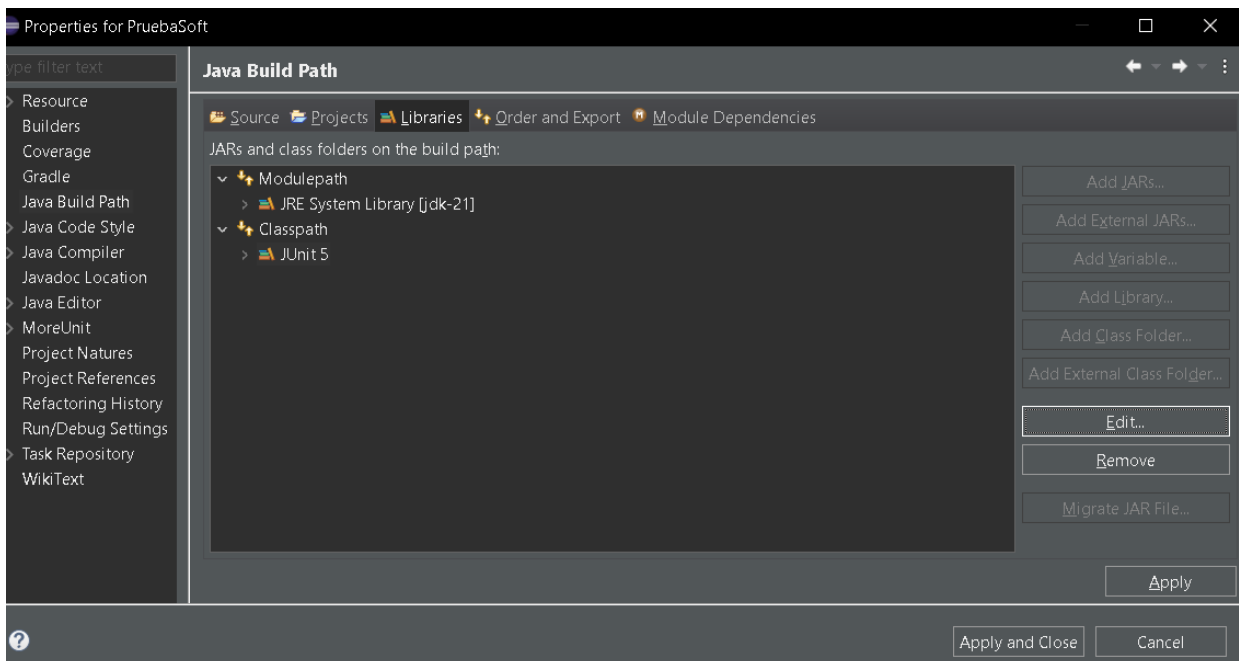


Entramos y seleccionamos la opción Junit con la versión, en este caso 5





Le damos a finish y ya tenemos la librería Junit.



## Uso de Mockito

Tenemos que aislar completamente cada test, para ello vamos a usar el concepto **de stub (o talon)** . Un stub hace referencia a una clase que simula ser otra pero que solo tiene implementada una pequeña parte de su funcionalidad. Lo suficiente para gestionar el método al que invocamos.

```
3
4 import static org.junit.jupiter.api.Assertions.*;
5
6 import org.junit.jupiter.api.BeforeEach;
7 import org.junit.jupiter.api.Test;
8 import org.mockito.Mockito;
9 import paquete.Coche;
10 import paquete.CuentaBancaria;
11 import paquete.Persona;
12 import paquete.Producto;
13 import paquete.Tienda;
14
15 class PersonaUnitTest {
16
17     private Persona persona;
18     private CuentaBancaria cuentaMock;
19     private Coche cocheMock;
20     private Tienda tiendaMock;
21     private Producto productoMock;
22
23     @BeforeEach
24     void setUp() throws Exception {
25         cuentaMock = Mockito.mock(CuentaBancaria.class);
26         cocheMock = Mockito.mock(Coche.class);
27         tiendaMock = Mockito.mock(Tienda.class);
28         persona = new Persona("Juan", 30, cuentaMock, cocheMock);
29     }
30 }
```

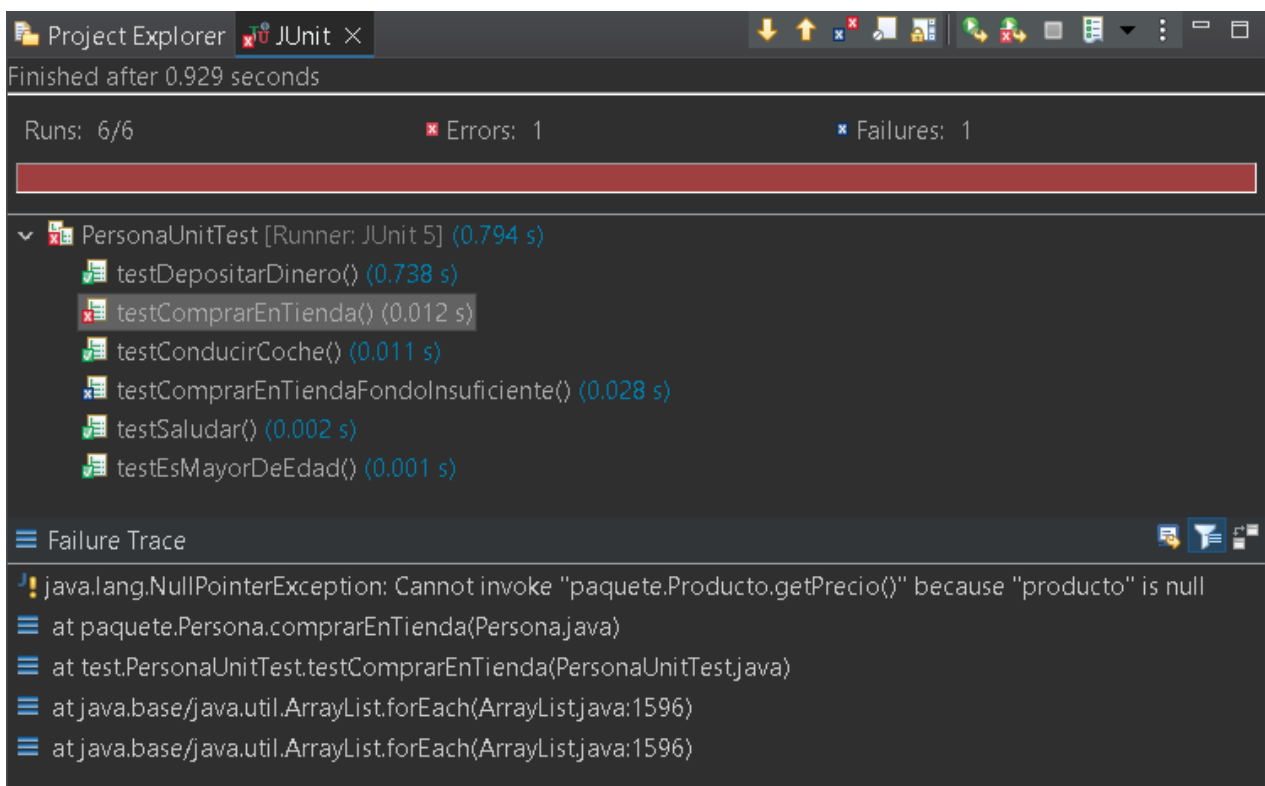
declaramos las variables y usamos el método mock para generar un **Mockito Stub** para coche, tienda y cuenta que seran objetos vacios que simulan ser objetos reales. Esto ya que la clase persona trabaja con dichos objetos.

```
3 public class Persona {
4     private String nombre;
5     private int edad;
6     private CuentaBancaria cuentaBancaria; // Relación con CuentaBancaria
7     private Coche coche; // Relación con Coche
8
9     public Persona() {
10
11     }
12
13     public Persona(String nombre, int edad, CuentaBancaria cuentaBancaria, Coche coche) {
14         this.nombre = nombre;
15         this.edad = edad;
16         this.cuentaBancaria = cuentaBancaria;
17         this.coche = coche;
18     }
19 }
```

Vamos a intentar usar el método comprarEnTienda, pero este necesita de 2 clase, producto y tienda, a continuación, mostramos el método que verifica que el monto que esta en la cuenta bancaria sea mayor al precio del producto, de lo contrario no nos dejara comprarlo.

```
52 public boolean comprarEnTienda(Tienda tienda, Producto producto) {
53     double precio = producto.getPrecio();
54     if (precio <= getCuentaBancaria().getSaldo()) {
55         cuentaBancaria.retirar(precio);
56         tienda.eliminarProducto(producto);
57         System.out.println(nombre + " ha comprado " + producto + " por " + precio + " euros.");
58         return true;
59     } else {
60         System.out.println(nombre + " no tiene suficiente dinero para comprar " + producto + ".");
61         return false;
62     }
63 }
```

```
43 @Test
44 void testComprarEnTienda() {
45
46     assertTrue(persona.comprarEnTienda(tiendaMock, productoMock));
47 }
48
```

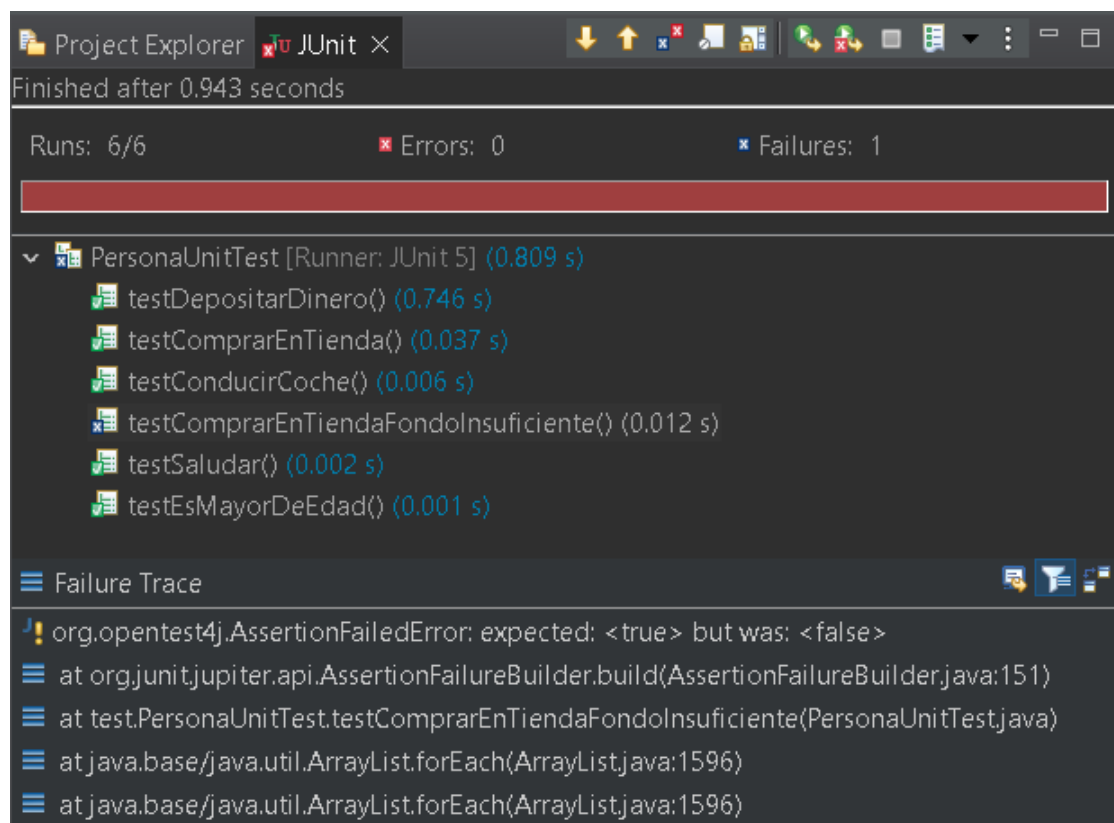


Si ejecutamos el testeo nos dará error diciendo que no pudo hacer el getPrecio del producto porque el producto es nulo, esto ya que el producto es simulado no es el objeto real.

Con Mockito vamos a configurar a través de **thenReturn** que cuando usemos algunos gets nos devuelva un valor fijo en este caso el precio del producto, el saldo de la cuenta bancaria y el nombre del producto. Esto ya que como vimos estos datos son necesario para el método que vamos a testear que es “**comprarEnTienda**” y con el **assertTrue** verificamos que el método nos dará true

```
43 • @Test
44     void testComprarEnTienda() {
45         productoMock = Mockito.mock(Producto.class);
46         tiendaMock.agregarProducto(productoMock);
47         Mockito.when(productoMock.getPrecio()).thenReturn(1200.0);
48         Mockito.when(persona.getCuentaBancaria().getSaldo()).thenReturn(1300.0);
49         Mockito.when(productoMock.getNombre()).thenReturn("Caldo");
50         assertTrue(persona.comprarEnTienda(tiendaMock, productoMock));
51     }
52
53 • @Test
54     void testComprarEnTiendaFondoInsuficiente() {
55         productoMock = Mockito.mock(Producto.class);
56         tiendaMock.agregarProducto(productoMock);
57         Mockito.when(productoMock.getPrecio()).thenReturn(1200.0);
58         Mockito.when(persona.getCuentaBancaria().getSaldo()).thenReturn(100.0);
59         Mockito.when(productoMock.getNombre()).thenReturn("Caldo");
60         assertTrue(persona.comprarEnTienda(tiendaMock, productoMock));
61     }
62
```

Ejecutamos esto y nos ya nos dará un resultado sin errores, de igual manera creamos otro test en donde el saldo de la cuenta bancaria sea menor al precio del producto.



```
Project Explorer x JUnit x
Finished after 0.943 seconds

Runs: 6/6      x Errors: 0      x Failures: 1

v PersonaUnitTest [Runner: JUnit 5] (0.809 s)
  testDepositarDinero() (0.746 s)
  testComprarEnTienda() (0.037 s)
  testConducirCoche() (0.006 s)
  x testComprarEnTiendaFondoInsuficiente() (0.012 s)
  testSaludar() (0.002 s)
  testEsMayorDeEdad() (0.001 s)

Failure Trace
org.opentest4j.AssertionFailedError: expected: <true> but was: <false>
at org.junit.jupiter.api.AssertionFailureBuilder.build(AssertionFailureBuilder.java:151)
at test.PersonaUnitTest.testComprarEnTiendaFondoInsuficiente(PersonaUnitTest.java)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)
```

Hacemos lo mismo con el test unitario de Tienda, declaramos las variables que usaremos y simulamos las clases con Mockito.

```
15 class TiendaUnitTest {
16     private Tienda tienda;
17     private Empleado empleadomock;
18     private Producto productomock;
19
20     @BeforeEach
21     void setUp() throws Exception {
22         empleadomock = Mockito.mock(Empleado.class);
23         productomock = Mockito.mock(Producto.class);
24         tienda = new Tienda("Super");
25     }
26 }
```

Los siguientes tests utilizan las funciones ya antes vistas como son Mockito.mock(clase en particular) para simularla y Mockito.when para definir el resultado de un método. Estos métodos son los que usaremos en todo los test cuando haga falta. Además de usar los assertEquals para saber si el resultado esperado es el resultado obtenido del método y el assertTrue para verificar que el resultado es de tipo booleano true.

En los dos últimos test usamos mockito para los gets y le pasamos el nombre del producto que queramos que devuelva.

```
39 @Test
40 public void testCantidadTotalProductos() {
41     Producto producto2 = Mockito.mock(Producto.class);
42     tienda.agregarProducto(productomock);
43     tienda.agregarProducto(producto2);
44
45     assertEquals(2, tienda.CantidadTotalProductos());
46 }
47
48
49 @Test
50 void testEliminarProducto() {
51     tienda.agregarProducto(productomock);
52     Mockito.when(productomock.getNombre()).thenReturn("leche");
53     assertEquals("leche", productomock.getNombre());
54     assertTrue(tienda.eliminarProducto(productomock));
55 }
56
57
58 @Test
59 void testTieneProducto() {
60     tienda.agregarProducto(productomock);
61     Mockito.when(productomock.getNombre()).thenReturn("Caldo");
62     assertEquals("Caldo", productomock.getNombre());
63 }
```

Y en estos dos test para obtener el precio, nombre y nos devuelva el sector al que pertenece cada empleado.

```
66● @Test
67 void testObtenerPrecio() {
68     tienda.agregarProducto(productomock);
69     Mockito.when(productomock.getPrecio()).thenReturn(220.0);
70     Mockito.when(productomock.getNombre()).thenReturn("Caldo");
71     assertEquals(220.0, tienda.obtenerPrecio("Caldo"));
72 }
73
74● @Test
75 void testSueldoTotalSectores() {
76     tienda.agregarEmpleado(empleadomock);
77     empleadomock.setSectorDeTrabajo("Repositor");
78     Mockito.when(empleadomock.getSectorDeTrabajo()).thenReturn("Repositor");
79
80     assertEquals(1, tienda.EmpleadosPorSector("Repositor"));
81 }
82
```

### Ejercicio 3

Ahora veremos los resultados de la prueba de integracion, recordemos que en la prueba de integracion no usaremos Mockito ya que nos interesa saber el funcionamiento de todas las clases.

Project Explorer JUnit X

Finished after 0.272 seconds

Runs: 10/10 Errors: 0 Failures: 2

IntegracionUnitTest [Runner: JUnit 5] (0.133 s)

- testCompraConTransferenciaEntreEmpleados() (0.039 s)
- testEmpleadoCompraDosProductos() (0.011 s)
- testEmpleadoDepositoYSulInteraccionConTienda() (0.004 s)
- testEmpleadoCompraProductoInexistente() (0.014 s)
- testEmpleadoRetiraFondoInsuficienteYCocheSinCombustible() (0.001 s)
- testEmpleadoCompraProductoYDiasTrabajados() (0.026 s)
- testEmpleadoAgregaYCompraProducto() (0.004 s)
- testEmpleadoConducirYCargarCombustible() (0.006 s)
- testEmpleadoCompraUnProductoYRevisaSuSaldo() (0.009 s)
- testEmpleadoCompraProductoSaldoInsuficienteYDeposito() (0.009 s)

Failure Trace

org.opentest4j.AssertionFailedError: expected: <true> but was: <false>

at org.junit.jupiter.api.AssertionFailureBuilder.build(AssertionFailureBuilder.java:151)

at testIntegracion.IntegracionUnitTest.testEmpleadoCompraUnProductoYRevisaSuSaldo(IntegracionUnitTest.java)

at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)

at java.base/java.util.ArrayList.forEach(ArrayList.java:1596)

Explicaremos los errores que nos tira la ejecución, en este caso solo dos métodos nos dan errores.

```
5  @Test
6  void testEmpleadoCompraProductoInexistente() {
7      tienda.agregarEmpleado(empleado);
8      tienda.agregarProducto(pan);
9      assertFalse(tienda.tieneProducto("Jugo"));
10     empleado.comprarEnTienda(tienda, leche);
11     assertTrue(tienda.tieneProducto("Pan"));
12     assertEquals(100.0, cuentaBancaria.getSaldo());
13 }
```

Vemos que agregamos dos productos a la tienda y tratamos de comprar un producto que no existe, dándonos ya un error de esa línea. Luego compramos un producto que, si existe, pero el valor de este es de 30.0, que se descuenta del saldo de la cuenta bancaria, por ende, esperamos un saldo de 100.0 cuando es menos debido a la compra del producto.

```
116 @Test
117 void testEmpleadoCompraUnProductoYRevisaSuSaldo() {
118     empleado.setContrato(false);
119     tienda.agregarProducto(leche);
120     empleado.comprarEnTienda(tienda, leche);
121     assertEquals(70.0, cuentaBancaria.getSaldo());
122     assertTrue(tienda.tieneProducto("Leche"));
123 }
```

En este segundo test es parecido al primer, pero en este el saldo esperado es el correcto, pero nos dará error en la línea **assertTrue** nos dará un false ya que una vez comprado el producto "leche" dejo de existir en la tienda.

## Ejercicio 4

Ahora veremos la cobertura de código que logramos mediante todos los test.

### Tests Unitarios.

Coche.

Coverage					
CocheUnitTest (12 oct 2024 16:17:55)					
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions	
PruebaSoft	<div><div></div></div> 8.6 %	134	1,432	1,566	
src	<div><div></div></div> 8.6 %	134	1,432	1,566	
paquete	<div><div></div></div> 14.2 %	82	497	579	
test	<div><div></div></div> 8.0 %	52	573	625	
CocheUnitTest.java	<div><div></div></div> 82.5 %	52	11	63	
CuentaBancariaUnitTest.java	<div><div></div></div> 0.0 %	0	70	70	



## Cuenta Bancaria.

Problems Javadoc Declaration Search Progress Coverage X Console Gradle Executions					
CuentaBancariaUnitTest (12 oct 2024 16:18:43)					
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions	
PruebaSoft		109	1,457	1,566	
src		109	1,457	1,566	
paquete		50	529	579	
Coche.java		0	111	111	
CuentaBancaria.java		50	23	73	
Empleado.java		0	82	82	

## Persona.

Problems Javadoc Declaration Search Progress Coverage X Console Gradle Executions					
PersonaUnitTest (12 oct 2024 16:19:17)					
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions	
PruebaSoft		217	1,349	1,566	
src		217	1,349	1,566	
paquete		95	484	579	
test		122	463	585	
CocheUnitTest.java		0	63	63	
CuentaBancariaUnitTest.java		0	70	70	
EmpleadoUnitTest.java		0	114	114	
PersonaUnitTest.java		122	46	168	
Test.java		0	4	4	

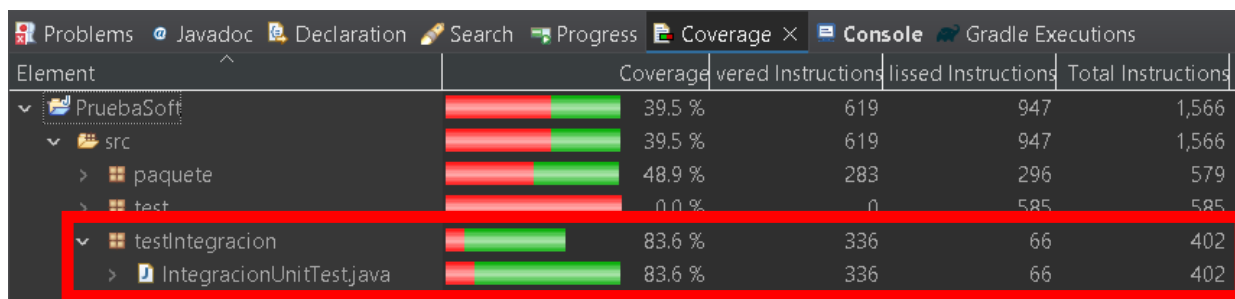
## Empleado.







Problems Javadoc Declaration Search Progress Coverage X Console Gradle Executions					
EmpleadoUnitTest (12 oct 2024 16:19:51)					
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions	
PruebaSoft		140	1,426	1,566	
src		140	1,426	1,566	
paquete		74	505	579	
test		66	519	585	
CocheUnitTest.java		0	63	63	
CuentaBancariaUnitTest.java		0	70	70	
EmpleadoUnitTest.java		66	48	114	
PersonaUnitTest.java		0	168	168	
Test.java		0	4	4	

## Tienda.

Problems Javadoc Declaration Search Progress Coverage X Console Gradle Executions					
TiendaUnitTest (12 oct 2024 16:20:25)					
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions	
PruebaSoft		250	1,316	1,566	
src		250	1,316	1,566	
paquete		84	495	579	
test		166	419	585	
CocheUnitTest.java		0	63	63	
CuentaBancariaUnitTest.java		0	70	70	
EmpleadoUnitTest.java		0	114	114	
PersonaUnitTest.java		0	168	168	
Test.java		0	4	4	
TiendaUnitTest.java		166	0	166	

Por último, tenemos el test de integracion nos dará una cobertura del 83.6% la cual en nuestra opinión es mas que aceptable para este modelo de proyecto.



Element	Coverage	covered Instructions	Uncovered Instructions	Total Instructions
PruebaSoft		619	947	1,566
src		619	947	1,566
paquete		283	296	579
test		0	585	585
testIntegracion		336	66	402
IntegracionUnitTest.java		336	66	402