spring by Pivotal

Java

PRUEBAS UNITARIAS

Avanzado

Bancolombia

JUnit

Java

Registro

Bancolombia

¿Qué sabemos hasta este punto de Pruebas Unitarias?

Hagamos un resumen rápido:

- Definición y características
- Patrón AAA (Arrange, Act, Assert)
- JUnit
- Aserciones básicas
- Anotaciones básicas

¿Qué vamos a ver?

- Mock y spy
- Mockito
  - @Mock, mock() y @RunWith
  - @Spy, spy()
  - when – then
  - verify()
  - @InjectMocks
- Powermockito
  - @PrepareForTest y @RunWith
  - mockStatic() y verifyStatic()
  - whenNew
  - verifyPrivate()

Simuladores (Mock)
Espías (Spy)

Mock

¿Qué es un mock?

Es un objeto que imita o simula el comportamiento de un objeto real de una forma controlada. Para lograr esto, un mock es una copia de la clase real con solo su “esqueleto”.

Ventajas:

- Puede verificar si un método fue llamado.
- Puede verificar con qué parámetros fue llamado un método.
- Puede verificar cuántas veces fue llamado un método.
- Puede retornar una respuesta con un estado determinado.

Cuándo usar mocks?

- En caso de metodos bloqueantes (BD, requests, etc).
- Métodos que aún no existen.
- Métodos fuera del contexto de la prueba unitaria.

Spy

¿Qué es un spy?

Al igual que un mock, permite verificar comportamientos de un objeto, con la diferencia de que conserva los comportamientos y atributos reales del objeto.

Ventajas:

- Puede verificar si un método fue llamado.
- Puede verificar con qué parámetros fue llamado un método.
- Puede verificar cuántas veces fue llamado un método.
- Puede retornar una respuesta con un estado determinado.

¿Cuándo usar un spy?

- Cuando sea necesario conservar los comportamientos reales del objeto.
- En caso de querer “mockear” solo algunos métodos.

Mockito

Mockito

Definición:

Framework para la creación de mocks, spies, etc. en pruebas unitarias para Java.

Ventajas:

- Legibilidad
- API limpia y simple
- Produce errores de verificación limpios

Gradle:

```
testCompile group: 'org.mockito', name: 'mockito-core', version: '2.8.9'
```

@Mock, mock()

Ejemplo @Mock

```
import org.mockito.Mock;


public class MockAnnotationUnitTest {

        ...

        @Mock
        UserRepository mockRepository;

        ...

}
```

Ejemplo mock()

```
import static org.mockito.Mockito.*;

@RunWith(MockitoJUnitRunner.class)
public class MockAnnotationUnitTest {

        ...

        @Test
        public void myFirstMockUnitTest () {
                UserRepository mockRepository =
                mock(UserRepository.class);
        }

        ...

}
```

@Spy, spy()

Ejemplo @Spy

```
import org.mockito.Spy;


public class MockAnnotationUnitTest {

        ...

        @Spy
        UserRepository spyRepository;

        ...

}
```

Ejemplo spy()

```
import static org.mockito.Mockito.*;

@RunWith(MockitoJUnitRunner.class)
public class MockAnnotationUnitTest {

        ...

        @Test
        public void myFirstMockUnitTest () {
                UserRepository spyRepository =
                spy(UserRepository.class);
        }

        ...

}
```

Patrón when - then

Este patrón define cómo se comporta un mock o un spy al momento de llamar a un método en específico.

Algunos métodos que podemos utilizar:

- when(...).thenReturn(returnValue)
- when(...).thenThrow(exception)
- when(...).thenCallRealMethod()
- when(...).thenAnswer()

Ejemplos:

- when(daoMock.save(customer)).thenReturn(true);
- when(daoSpy.save(customer)).thenThrow(RuntimeException.class);

verify()

Las pruebas unitarias no solo se validan a través de aserciones, también pueden hacer uso de verificaciones en ciertos escenarios gracias al uso de mocks. Estas verificaciones se hacen a través del metodo verify().

Qué puedo verificar?

- La invocación simple de un mock
- El número de interacciones con un mock
- Si no hubo interacciones con un mock
- Si no hay interacciones inesperadas
- Orden de interacciones
- Un número mínimo o máximo de interacciones
- La interacción con un argumento exacto

Ejemplos verify()

```
List<String> mockedList = mock(MyList.class);
...
verify(mockedList).size();
verify(mockedList, times(1)).size();
verifyZeroInteractions(mockedList);
...
inOrder.verify(mockedList).size();
inOrder.verify(mockedList).add("a parameter");
inOrder.verify(mockedList).clear();
...
verify(mockedList, never()).clear();
verify(mockedList, atLeast(1)).clear();
verify(mockedList, atMost(10)).clear();
```

Inyección de mocks @InjectMocks

@InjectMocks nos permite inyectar dependencias mockeadas en una clase mockeada. Esto es bastante útil cuando tenemos dependencias externas en la clase que queremos mockear.

Cómo ocurre esta inyección?

Mockito intenta inyectar dependencias mockeadas usando uno de tres posibles enfoques, en el orden especificado:

- **Inyección basada en el constructor**: Cuando existe un constructor, mockito intenta inyectar dependencias usando el constructor más grande.
- **Basada en métodos setter:** Cuando no existe un constructor, mockito intenta inyectar usando métodos setter.
- **Basada en campos:** Si no hay un constructor o métodos setter, mockito intenta inyectar.

Uso de @InjectMocks

```
import org.mockito.InjectMocks;
import org.mockito.Mock;

public class MockitoInjectMocksExample {

    @Mock
    EmailService emailService;

    @Mock
    SMSService smsService;

    @InjectMocks
    AppServices appServices;

    ...
}
```

http://gitlab.com/ingsw-bancolombia/dojo-injectmocks-exercise.git

Ejemplo práctico con @Mock y @InjectMocks

- Hacer que el código sea testeable
- Verificar injección de mocks usando constructores y setters
- Terminar construcción de pruebas

Powermockito

Powermockito

Definición:

Es una extensión de Powermock que permite soportar Mockito.

Ventajas:

- Permite crear mocks sobre métodos final, private y static.
- Permite mocker sin inyección de dependencias.

Gradle:

testCompile group: 'org.powermock', name: 'powermock-api-mockito2', version: '1.7.4'

@PrepareForTest, @RunWith()

@PrepareForTest(…) especifica las clases que deben ser manipuladas por Powermock para que sean mockeadas correctamente.

@RunWith(PowerMockRunner.class) se debe añadir siempre para poder usar las funcionalidades de Powermock.

```
import org.mockito.Mock;

@RunWith(PowerMockRunner.class)
@PrepareForTest( { ClassAbc.class, ClassDef.class })
public class PowermockitoUnitTestExample {
        ...
}
```

@mockStatic() y verifyStatic()

```
import static org.mockito.Mockito;
import org.powermock.api.mockito.PowerMockito;


@RunWith(PowerMockRunner.class)
@PrepareForTest(StaticExample.class)
public class StaticPowermockitoExample {

        public void firstUnitTest () {
                PowerMockito.mockStatic(StaticExample.class);
                Mockito.when(StaticExample.firstStaticMethod(param)).thenReturn(value);

                classCallStaticMethodObj.execute();

                PowerMockito.verifyStatic(Static.class, Mockito.times(2));
                StaticExample.firstStaticMethod(param);
        }
}
```

whenNew(...).then

whenNew() nos permite tomar control de la instaciación de un objeto

```
@RunWith(PowerMockRunner.class)
@PrepareForTest(X.class)
public class XTest {

    @Test
    public void test() {
        whenNew(MyClass.class).withNoArguments().thenReturn(new MyClass());

        X x = new X();
        x.y(); // 'y' es el método que crea la instancia "new MyClass()"
        ...
    }
}
```

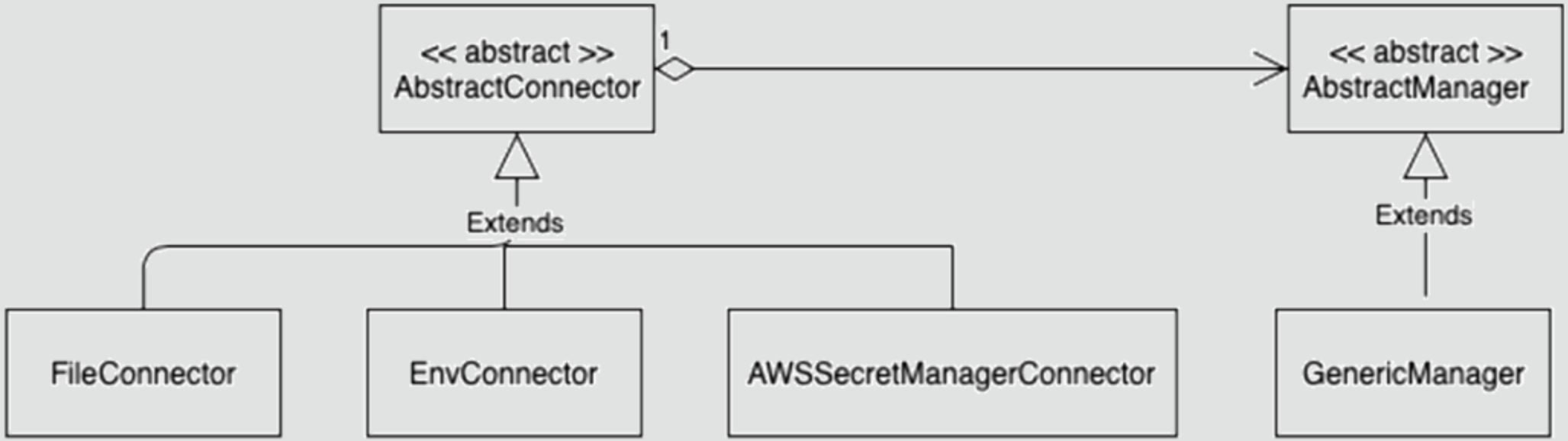
Mockear métodos privados

```
@RunWith(PowerMockRunner.class)
@PrepareForTest(PartialMockClass.class)
public class YourTestCase {
        @Test
        public void privatePartialMockingWithPowerMock() {
                PartialMockClass classUnderTest = PowerMockito.spy(new PartialMockClass());

                // use PowerMockito to set up your expectation
                PowerMockito.doReturn(value).when(classUnderTest, "methodToMock", "param1");

                // execute your test
                classUnderTest.execute();

                // Use PowerMockito.verify() to verify result
                PowerMockito.verifyPrivate(classUnderTest, times(2)).invoke("methodToMock", ”param1");
        }
}
```

Demo
Powermock + mockito

https://grupobancolombia.visualstudio.com/Vicepresidencia%20Servicios%20de%20Tecnolog%C3%ADa/_git/IngSW_UnitTestsJavaSecretsManager

Secrets Manager



spring by Pivotal

Java

GRACIAS!

Bancolombia

JUnit

Java

Calificación

Bancolombia