TP N°2 : Mockito

SOMMAIRE

I- Objectifs :

- ✓ Apprendre comment réaliser les tests unitaires avec Mockito.

II- Outils utilisés :

- ✓ JDK 1.8 ;
- ✓ Eclipse avec le plugin Maven ;
- ✓ Connection Internet pour télécharger les dépendances (Mockito, JUNIT et HAMCREST).

II- Développement des tests unitaires

a. Création d'un projet Maven

- ✓ Créer un projet Maven (exemple : testmockito).
- ✓ Modifier le fichier pom.xml comme suit :

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>ma.cigma</groupId>
        <artifactId>testmockito</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <properties>
                <java.version>1.8</java.version>
        </properties>

        <dependencies>
                <!-- https://mvnrepository.com/artifact/org.mockito/mockito-inline -->
                <dependency>
                        <groupId>org.mockito</groupId>
                        <artifactId>mockito-inline</artifactId>
                        <version>3.9.0</version>
                        <scope>test</scope>
                </dependency>


                <!-- https://mvnrepository.com/artifact/org.mockito/mockito-junit-jupiter -->
                <dependency>
                        <groupId>org.mockito</groupId>
                        <artifactId>mockito-junit-jupiter</artifactId>
                        <version>3.9.0</version>
                        <scope>test</scope>
                </dependency>

        <dependency>
                        <groupId>org.junit.jupiter</groupId>
                        <artifactId>junit-jupiter-api</artifactId>
                        <version>5.7.2</version>
```

```xml
			<scope>test</scope>
		</dependency>

		<dependency>
			<groupId>org.junit.jupiter</groupId>
			<artifactId>junit-jupiter-params</artifactId>
			<version>5.7.2</version>
			<scope>test</scope>
		</dependency>

		<dependency>
			<groupId>org.junit.jupiter</groupId>
			<artifactId>junit-jupiter-engine</artifactId>
			<version>5.7.2</version>
			<scope>test</scope>
		</dependency>

		<dependency>
			<groupId>org.junit.platform</groupId>
			<artifactId>junit-platform-commons</artifactId>
			<version>1.7.2</version>
		</dependency>

		<dependency>
			<groupId>org.junit.platform</groupId>
			<artifactId>junit-platform-console</artifactId>
			<version>1.7.2</version>
			<scope>test</scope>
		</dependency>

		<dependency>
			<groupId>org.junit.platform</groupId>
			<artifactId>junit-platform-console-standalone</artifactId>
			<version>1.7.2</version>
			<scope>test</scope>
		</dependency>

		<dependency>
			<groupId>org.junit.platform</groupId>
			<artifactId>junit-platform-runner</artifactId>
			<version>1.7.2</version>
			<scope>test</scope>
		</dependency>

		<dependency>
			<groupId>org.junit.platform</groupId>
			<artifactId>junit-platform-engine</artifactId>
			<version>1.7.2</version>
			<scope>test</scope>
		</dependency>

		<dependency>
```

```xml
				<groupId>org.junit.platform</groupId>
				<artifactId>junit-platform-launcher</artifactId>
				<version>1.7.2</version>
				<scope>test</scope>
			</dependency>

			<dependency>
				<groupId>org.junit.platform</groupId>
				<artifactId>junit-platform-suite-api</artifactId>
				<version>1.7.2</version>
				<scope>test</scope>
			</dependency>

			<dependency>
				<groupId>org.hamcrest</groupId>
				<artifactId>hamcrest</artifactId>
				<version>2.2</version>
				<scope>test</scope>
			</dependency>

		</dependencies>
</project>
```

- ✓ Dans ce qui suit, nous allons créer les classes suivantes qui vont nous servir pour réaliser les différents tests :

  - ✓ La classe DaoImp :

```java
package ma.cigma.dao;

public class DaoImp {
	private Integer uniqueId;

	public boolean isAvailable() {
		System.out.println("isAvailable est exécutée");
		return false;
	}

	public int getUniqueId() {
		System.out.println("getUniqueId est exécutée");
		return 42;
	}

	public void setUniqueId(Integer uniqueId) {
		System.out.println("setUniqueId est exécutée");
		this.uniqueId = uniqueId;
	}

}
```

- ✓ La classe ArticleDatabase :

```
package ma.cigma.dao;

import ma.cigma.service.ArticleListener;

public class ArticleDatabase {
           public void addListener(ArticleListener articleListener) {
      // TODO Auto-generated method stub
   }
}
```

- ✓ La classe User :

```
package ma.cigma.service;

public class User {
}
```

- ✓ La classe Utility :

```
package ma.cigma.service;

public class Utility {
            public static String getDatabaseConnection(String url) {
                 return "http:///production/" + url;
              }

}
```

- ✓ La classe ServiceImpl :

```
package ma.cigma.service;

import ma.cigma.dao.DaoImp;

public class ServiceImpl {

           private DaoImp dao;

   public ServiceImpl(DaoImp dao) {
      this.dao = dao;
   }

   public boolean query(String query) {
      return dao.isAvailable();
   }
```

```java

    @Override
    public String toString() {
       return "Using database with id: " + String.valueOf(dao.getUniqueId());
    }

}
```

✓ La classe FinalClass :

```java
package ma.cigma.service;

public class FinalClass {
            public final String finalMethod() { return "something"; }
}
```

✓ La classe ArticleManager :

```java
package ma.cigma.service;

import ma.cigma.dao.ArticleDatabase;

public class ArticleManager {
          private User user;
    private ArticleDatabase database;

    public ArticleManager(User user, ArticleDatabase database) {
       super();
       this.user = user;
       this.database = database;
    }

    public void initialize() {
       database.addListener(new ArticleListener());
    }

}
```

✓ La classe ArticleListener :

```java
package ma.cigma.service;

public class ArticleListener {}
```

a. Test 1 :

```java
package ma.cigma.service;

import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.mockito.Mockito.when;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import ma.cigma.dao.DaoImp;

@ExtendWith(MockitoExtension.class)
public class Exemple1 {

	@Mock
	DaoImp dao;

	@Test
	public void testQuery() {
		assertNotNull(dao);
		when(dao.isAvailable()).thenReturn(true);
		ServiceImpl service = new ServiceImpl(dao);
		boolean check = service.query("* from t");
		assertTrue(check);
	}
}
```

b. Test 2 :

```java
package ma.cigma.service;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.when;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import ma.cigma.dao.DaoImp;

@ExtendWith(MockitoExtension.class)
public class Exemple2 {

	@Mock
	DaoImp dao;

	@Test
```

```
        public void ensureMockitoReturnsTheConfiguredValue() {

                // define return value for method getUniqueId()
                when(dao.getUniqueId()).thenReturn(42);

                ServiceImpl service = new ServiceImpl(dao);
                // use mock in test....
                assertEquals(service.toString(), "Using database with id: 42");
        }

}
```

c. Test 3 :

```
package ma.cigma.service;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.ArgumentMatchers.isA;
import static org.mockito.Mockito.when;

import java.util.Iterator;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;


@ExtendWith(MockitoExtension.class)
public class Exemple3 {

        @Mock
    Iterator<String> i;

    Comparable<String> c;

    // demonstrates the return of multiple values
    @Test
    public void testMoreThanOneReturnValue() {
        when(i.next()).thenReturn("Mockito").thenReturn("rocks");
        String result = i.next() + " " + i.next();
        // assert
        assertEquals("Mockito rocks", result);
    }

    // this test demonstrates how to return values based on the input
    // and that @Mock can also be used for a method parameter
    @Test
    public void testReturnValueDependentOnMethodParameter(@Mock Comparable<String> c)  {
            when(c.compareTo("Mockito")).thenReturn(1);
```

```java
        when(c.compareTo("Eclipse")).thenReturn(2);
        //assert
        assertEquals(1, c.compareTo("Mockito"));
        assertEquals(2, c.compareTo("Eclipse"));
    }

    // return a value based on the type of the provide parameter

    @Test
    public void testReturnValueInDependentOnMethodParameter2(@Mock Comparable<Integer> c )  {
        when(c.compareTo(isA(Integer.class))).thenReturn(0);
        //assert
        assertEquals(0, c.compareTo(Integer.valueOf(40)));
    }

}
```

d. Test 4 :

```java
package ma.cigma.service;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.mockito.Mockito.when;

import java.util.Properties;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
public class Exemple4 {
        // demonstrates the configuration of a throws with Mockito
    // not a read test, just "testing" Mockito behavior

        @Mock
        Properties properties;

    @Test
    public void testMockitoThrows() {

      when(properties.get(Mockito.anyString())).thenThrow(new IllegalArgumentException("Stuff"));

      Throwable exception = assertThrows(IllegalArgumentException.class, () -> properties.get("A"));

      assertEquals("Stuff", exception.getMessage());
    }

}
```

e. Test 5 :

```
package ma.cigma.service;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.doReturn;

import java.util.Properties;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Spy;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
public class Exemple5 {

	@Spy
	Properties spyProperties;
	// demonstrates of the spy function
	@Test
	public void testMockitoThrows() {

		doReturn("42").when(spyProperties).get("shoeSize");

		String value = (String) spyProperties.get("shoeSize");

		assertEquals("42", value);
	}

}
```

f. Test 6 :

```
package ma.cigma.service;

import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.mockito.Mockito.doThrow;

import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
public class Exemple6 {
	@Mock
	OutputStream mockStream;
	@Test
	public void testForIOException() throws IOException {
		// create an configure mock
		doThrow(new IOException()).when(mockStream).close();

		// use mock
		OutputStreamWriter streamWriter = new OutputStreamWriter(mockStream);

		assertThrows(IOException.class, () -> streamWriter.close());
	}

}
```

g. Test 7 :

```
package ma.cigma.service;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.doReturn;

import java.util.Arrays;
import java.util.List;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Spy;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
public class Exemple7 {

	@Spy
	List<String> list =Arrays.asList("spy","mock");

	@Test
	public void testLinkedListSpyCorrect() {

		//when(list.get(0)).thenReturn("spy");
		assertEquals("spyll",list.get(0));
```

```
                doReturn("foo").when(list).get(0);
                assertEquals("foo", list.get(0));


        }

}
```

h. Test 8 :

```
package ma.cigma.service;

import static org.mockito.Mockito.atLeast;
import static org.mockito.Mockito.atLeastOnce;
import static org.mockito.Mockito.never;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.verifyNoMoreInteractions;
import static org.mockito.Mockito.when;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.ArgumentMatchers;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

import ma.cigma.dao.DaoImp;

@ExtendWith(MockitoExtension.class)
public class Exemple8 {

        @Test
        public void testVerify(@Mock DaoImp dao) {
                // create and configure mock
                when(dao.getUniqueId()).thenReturn(43);

                // call method testing on the mock with parameter 12
                dao.setUniqueId(12);
                dao.getUniqueId();
                dao.getUniqueId();

                // now check if method testing was called with the parameter 12
                verify(dao).setUniqueId(ArgumentMatchers.eq(12));

                // was the method called twice?
                verify(dao, times(2)).getUniqueId();

                // other alternatives for verifiying the number of method calls for a method
                verify(dao, never()).isAvailable();
                verify(dao, never()).setUniqueId(13);
                verify(dao, atLeastOnce()).setUniqueId(12);
                verify(dao, atLeast(2)).getUniqueId();

                // more options are
                // times(numberOfTimes)
                // atMost(numberOfTimes)
                // This let's you check that no other methods where called on this object.
                // You call it after you have verified the expected method calls.
                verifyNoMoreInteractions(dao);
        }
}
```

i. Test 9 :

```
package ma.cigma.service;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;

import ma.cigma.dao.ArticleDatabase;

@ExtendWith(MockitoExtension.class)
public class Exemple9 {

        @Mock
        ArticleDatabase database;
        @Mock
        User user;

        @InjectMocks
        private ArticleManager manager;

        @Test
        public void shouldDoSomething() {
                // calls addListener with an instance of ArticleListener
                manager.initialize();

                // validate that addListener was called
                Mockito.verify(database).addListener(Mockito.any(ArticleListener.class));
        }

}
```

j. Test 10 :

```java
package ma.cigma.service;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.hasItem;
import static org.mockito.Mockito.verify;

import java.util.Arrays;
import java.util.List;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.ArgumentCaptor;
import org.mockito.Captor;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
public class Exemple10 {
	@Captor
	private ArgumentCaptor<List<String>> captor;

	@Test
	public final void shouldContainCertainListItem(@Mock List<String> mockedList) {
		List asList = Arrays.asList("someElement_test", "someElement");
		mockedList.addAll(asList);

		verify(mockedList).addAll(captor.capture());
		List<String> capturedArgument = captor.getValue();
		assertThat(capturedArgument, hasItem("someElement"));
	}

}
```

k. Test 11 :

```java
package ma.cigma.service;

import static org.junit.jupiter.api.Assertions.assertNotNull;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.MockedStatic;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
```

```
public class Exemple11 {
	@Test
	public void testMockFinal(@Mock FinalClass finalMocked) {
		assertNotNull(finalMocked);
	}

	@Test
	public void testMockFinalViaMockStatic() {
		MockedStatic<FinalClass> mockStatic = Mockito.mockStatic(FinalClass.class);
		assertNotNull(mockStatic);
	}
}
```

I. Test 12 :

```
package ma.cigma.service;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.*;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.MockedStatic;
import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
public class Exemple12 {
	@Test
  void shouldMockStaticMethod() {
    try (MockedStatic<Utility> mockedStatic = Mockito.mockStatic(Utility.class)) {

      mockedStatic.when(() -> Utility.getDatabaseConnection(eq("test"))).thenReturn("testing");
      mockedStatic.when(() -> Utility.getDatabaseConnection(eq("prod"))).thenReturn("production");

      String result1 = Utility.getDatabaseConnection("test");

      assertEquals("testing", result1);
      String result2 = Utility.getDatabaseConnection("prod");
      assertEquals("production", result2);

    }

  }
}
```