TP N°1 : JUnit

SOMMAIRE

I- Objectifs :

- ✓ Apprendre comment réaliser les tests unitaires avec JUnit.

II- Outils utilisés :

- ✓ Eclipse avec le plugin Maven
- ✓ Connection Internet pour télécharger les dépendances (Junit).

II- Développement des tests unitaires

a. Création d'un projet Maven

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

```xml
<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>demojunit</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <properties>
                <java.version>1.8</java.version>
                <maven.compiler.source>1.8</maven.compiler.source>
                <maven.compiler.target>1.8</maven.compiler.target>
                        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.junit.jupiter</groupId>
                        <artifactId>junit-jupiter-api</artifactId>
                        <version>5.7.2</version>
                        <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>
				<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>

		 <reporting>
			<plugins>
				<plugin>
					<groupId>org.apache.maven.plugins</groupId>
					<artifactId>maven-surefire-report-plugin</artifactId>
					<version>2.19.1</version>
```

```
                                        </plugin>
                                </plugins>
                        </reporting>
</project>
```

a. Test 1 :

```java
package demojunit;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

@DisplayName("Ma classe de test JUnit5")
public class TestExample {
        @Test
        @DisplayName("Mon cas de test")
        void test1() {
                System.out.println("simpleTest");
          Assertions.assertTrue(true);
        }
}
```

b. Test 2 :

```java
package demojunit;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class TestExample2 {
        @BeforeAll
         static void initAll() {
          System.out.println("beforeAll");
         }

         @BeforeEach
         void init() {
          System.out.println("beforeEach");
         }

         @AfterEach
         void tearDown() {
          System.out.println("afterEach");
         }

         @AfterAll
         static void tearDownAll() {
          System.out.println("afterAll");
         }

         @Test
         void simpleTest() {
          System.out.println("simpleTest");
          Assertions.assertTrue(true);
         }

         @Test
         void secondTest() {
          System.out.println("secondTest");
          Assertions.assertTrue(true);
         }
}
```

c. Test 3 :

```
package demojunit;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotSame;
import static org.junit.jupiter.api.Assertions.assertTrue;

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

import org.junit.jupiter.api.Test;

public class TestExample3 {

        @Test
        void monPremierTest() {
                assertTrue(true);
                assertTrue(this::isValide);
                assertTrue(true, () -> "Description " + "du cas " + "de test");
                List<String> attendu = Arrays.asList("e1", "e2", "e2");
                List<String> actual = new LinkedList<>(attendu);
                assertEquals(attendu, actual);
                assertEquals(attendu, actual, "Les listes ne sont pas égales");
                assertEquals(attendu, actual, () -> "Les listes " + "ne sont " + "pas égales");
                assertNotSame(attendu, actual, "Les instances sont les memes");
        }

        boolean isValide() {
```

```
                    return true;
            }
}
```

d. Test 4 :

```
package demojunit;

import java.awt.Dimension;
import java.util.ArrayList;
import java.util.Arrays;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class TestExample4 {
        @Test
        void verifierAttributs() {
          Dimension sut = new Dimension(800, 600);
          Assertions.assertAll("Dimensions non conformes",
            () -> Assertions.assertTrue(sut.getWidth() == 801, "Valeur de width erronee"),
            () -> Assertions.assertTrue(sut.getHeight() == 601, "Valeur de height erronee"));
        }

        @Test
        void verifierEgaliteTableaux() {
          Assertions.assertArrayEquals(new int[] { 1, 2, 3 }, new int[] { 1, 2, 3 },
            "Egalite des tableaux");
        }

        @Test
        void verifierNonEgaliteTableaux() {
          Assertions.assertArrayEquals(new int[] { 1, 2, 3 }, new int[] { 3, 2, 1 },
            "Egalite des tableaux");
        }

        @Test
        void verifierEgalite() {
          Dimension sut = new Dimension(801, 601);
          Assertions.assertEquals(new Dimension(800, 600), sut, "Dimensions non egales");
        }

        @Test
        void verifierTrue() {
          boolean bool = true;
          Assertions.assertTrue(bool);
          Assertions.assertTrue(TestExample4::getBooleen, "Booleen different de true");
        }

        static boolean getBooleen() {
```

```
		return false;
	}

	@Test
	void verifierIterableEquals() {
	  Iterable<Integer> attendu = new ArrayList<>(Arrays.asList(1, 2, 3));
	  Iterable<Integer> actuel = new ArrayList<>(Arrays.asList(1, 2, 3));
	  Assertions.assertIterableEquals(attendu, actuel);
	}
}
```

e. Test 5 :

```
package demojunit;

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

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class TestExample5 {
	@Test
	void verifierLinesMatch() {
		List<String> expectedLines = Arrays.asList("A1", "A2", "A3", "A4");
		List<String> emails = Arrays.asList("A1", "A2", "A3", "A4");
		Assertions.assertLinesMatch(expectedLines, emails);
	}

	@Test
	void verifierLinesMatch2() {
		List<String> expectedLines = Arrays.asList("(.*)@(.*)", "(.*)@(.*)");
		List<String> emails = Arrays.asList("test@gmail.com", "jm@test.fr");
		Assertions.assertLinesMatch(expectedLines, emails);
	}

	/*
	 * Il est aussi possible d'ignorer un ou plusieurs éléments durant la
	 * comparaison grâce à un marqueur d'avance rapide : ils peuvent par exemple
	 * permettre d'ignorer des éléments dont la valeur change à chaque exécution.
	 *
	 * Un marqueur d'avance rapide commence et termine par «>>» et doit posséder au
	 * moins quatre caractères.
	 */
	@Test
	void verifierLinesMatch3() {
		List<String> expectedLines = Arrays.asList("(.*)@(.*)", ">>>>", "(.*)@(.*)");
		List<String> emails = Arrays.asList("test@gmail.com", "test", "email", "jm@test.fr");
		Assertions.assertLinesMatch(expectedLines, emails);
	}
	/**
	 * Il est possible de mettre une description entre les doubles chevrons : cette
	 * description sera ignorée.
	 */

	@Test
	void verifierLinesMatch4() {
		List<String> expectedLines = Arrays.asList("(.*)@(.*)", ">> aller au dernier >>", "(.*)@(.*)");
		List<String> emails = Arrays.asList("test@gmail.com", "test", "email", "jm@test.fr");
		Assertions.assertLinesMatch(expectedLines, emails);
	}

	/**
	 * Il est possible de préciser un nombre exact d'éléments à ignorer.
	 *
	 *
	 */

	@Test
	void verifierLinesMatch5() {
		List<String> expectedLines = Arrays.asList("A1", ">> 2 >>", "A4");
		List<String> emails = Arrays.asList("A1", "A2", "A3", "A4");
		Assertions.assertLinesMatch(expectedLines, emails);
	}

	/**
	 * Si le nombre d'éléments à ignorer ne peut être atteint ou est insuffisant
	 * alors la méthode lève une exception.
	 */

	@Test
	void verifierLinesMatch6() {

		List<String> expectedLines = Arrays.asList("A1", ">> 1 >>", "A4");
		List<String> emails = Arrays.asList("A1", "A2", "A3", "A4");
		Assertions.assertLinesMatch(expectedLines, emails);
	}
}
```

f. Test 6 :

```
package demojunit;

import java.awt.Dimension;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class TestExample6 {
	@Test
	void verifierNull() {
	  Object sut = new Dimension(800, 600);
	  Assertions.assertNull(sut);
	}

	@Test
	void verifierNotNull() {
	  Object sut = null;
	  Assertions.assertNotNull(sut);
	}
}
```

g. Test 7 :

```
package demojunit;

import java.awt.Dimension;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class TestExample7 {
	@Test
	void verifierSame() {
		Object sut = new Dimension(800, 600);
		Object expected = new Dimension(800, 600);
		Assertions.assertSame(sut, expected);
	}

	@Test
	void verifierNotSame() {
		Object sut = new Dimension(800, 600);
		Object expected = sut;
		Assertions.assertNotSame(sut, expected);
	}
}
```

h. Test 8 :

```
package demojunit;

import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNull;

import org.junit.jupiter.api.Test;
```

```
public class TestExample8 {
	@Test
	void verifierException() {

		String valeur = null;
		assertThrows(NumberFormatException.class, () -> {
			Integer.valueOf(valeur);
		});
	}

	@Test
	void verifierException2() {
		String valeur = "1";
		assertThrows(NumberFormatException.class, () -> {
			Integer.valueOf(valeur);
		});
	}

	public void maMethode() {
	   throw new RuntimeException("mon message d'erreur");
	 }

	@Test
	 void verifierException3() {
	   TestExample8 sut = new TestExample8();
	   RuntimeException excep = assertThrows(RuntimeException.class, sut::maMethode);
	   assertAll(() -> assertEquals("message erreur", excep.getMessage()),
		() -> assertNull(excep.getCause()));
	 }
}
```

i. Test 9 :

```
package demojunit;

import java.time.Duration;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class TestExample9 {

	/**
	 * Les assertions assertTimeout et assertTimeoutPreemptively vérifie que les
	 * traitements fournis en paramètre s'exécutent avant le délai précisé. La
	 * différence entre les deux est que assertTimeoutPreemptively interrompt
	 * l'exécution des traitements si le délai est dépassé.
	 */
	@Test
	void verifierTimeout() {
		Assertions.assertTimeout(Duration.ofMillis(200), () -> {
			return "";
		});
		Assertions.assertTimeout(Duration.ofSeconds(1), TestExample9::traiter);
	}

	private static String traiter() throws InterruptedException {
		Thread.sleep(2000);
		return "";
	}

	@Test
	void verifierTimeoutPreemptively() {
		Assertions.assertTimeoutPreemptively(Duration.ofMillis(200), () -> {
			return "";
		});

		Assertions.assertTimeoutPreemptively(Duration.ofSeconds(1), TestExample9::traiter);
	}
}
```

j. Test 10 :

```
package demojunit;

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

import org.junit.jupiter.api.Test;

public class TestExample10 {

	@Test
	 void monTest() {
	  fail("la raison de l'échec du test");
	 }
}
```

k. Test 11 :

```
package demojunit;

import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.CoreMatchers.~~is~~;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.Test;

public class TestExample11 {
```

```java
        @Test
        void testAvecHamcrest() {
                assertThat(1 + 2, is(equalTo(4)));
        }
}
```

l. Test 12 :

```java
package demojunit;

import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.junit.jupiter.api.Assumptions.assumeTrue;
import static org.junit.jupiter.api.Assumptions.assumingThat;

import java.io.File;

import org.junit.jupiter.api.Test;

public class TestExample12 {

        @Test
         void testSousWindows() {
           System.out.println(System.getenv("OS"));
           assumeTrue(System.getenv("OS").startsWith("Windows"));
           assertTrue(false);
         }

         @Test
         void testAvecSupposition() {
           assumingThat(System.getenv("OS").startsWith("Windows"), () -> {
             assertTrue(new File("C:/Windows").exists(), "Repertoire Windows inexistant");
           });
           assertTrue(true);
         }

        public static void main(String[] args) {
                System.out.println(System.getenv("OS"));
        }
}
```

m. Test 13 :

```java
package demojunit;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;

public class TestExample13 {
	@Test
	@Disabled("A écrire plus tard")
	void monTest() {
		Assertions.fail("Non implémenté");
	}
}
```

n. **Test 14 :**

```
package demojunit;

import static org.junit.Assert.assertTrue;

import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;

@Tag("principal")
public class TestExample14 {
	@Test
	@Tag("general")
	void testCas1() {
		assertTrue(true);
	}

	@Test
	@Tag("specifique")
	void testCas2() {
		assertTrue(true);
	}
}
```

o. **Test 15 :**

```
package demojunit;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvFileSource;
import org.junit.jupiter.params.provider.CsvSource;

public class TestCsvSource {
	@DisplayName("Addition")
	@ParameterizedTest()
	@CsvSource({ "1, 1", "1, 2", "2, 3" })
```

```
        void testAdditionner(int a, int b) {
                int attendu = a + b;
                Assertions.assertEquals(attendu, a + b);
        }

         @ParameterizedTest()
          @CsvFileSource(resources = "additionner_source.csv")
          void testAdditionner(int a, int b) {
           int attendu = a + b;
           Assertions.assertEquals(attendu, a + b);
          }
}
```

Le fichier additionner_source.csv est :

```
1,1
1,2
2,3
6,9
10,45
15,63
```

p. Test 16 :

```
package demojunit;

import java.util.ArrayList;
import java.util.Collection;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DynamicTest;
import org.junit.jupiter.api.TestFactory;

public class TestdynamicAvecCollection {
        @TestFactory
        Collection<DynamicTest> dynamicTestsAvecCollection() {
          Collection<DynamicTest> resultat = new ArrayList<>();
          for (int i = 1; i <= 5; i++) {
            int val = i;
            resultat.add(DynamicTest.dynamicTest("Ajout " + val + "+" + val,
                () -> Assertions.assertEquals(val * 2, val + val)));
          }
          return resultat;
```

```
            }
}
```

q. Test 17 :

```
package demojunit;

import java.util.stream.Stream;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;

public class TestMethodSource {
        @ParameterizedTest
        @MethodSource("fournirDonnees")
        void testExecuter(String element) {
                Assertions.assertTrue(element.startsWith("elem"));
        }

        static Stream<String> fournirDonnees() {
                return Stream.of("elem1", "elem2","hh");
        }
}
```

r. Test 18 :

```
package demojunit;

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

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;

public class TestMethodSource2 {
         @ParameterizedTest
          @MethodSource("fournirDonnees")
          void testTraiter(int index, String element) {
            Assertions.assertTrue(index > 0);
            Assertions.assertTrue(element.startsWith("elem"));
          }
           static List<Object[]> fournirDonnees() {
            return Arrays.asList(new Object[][] { { 1, "elem1" }, { 2, "elem2" } });
          }
}
```

s. Test 19 :

```
package demojunit;

import java.time.Month;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.EnumSource;
import org.junit.jupiter.params.provider.ValueSource;

public class TestParametre {
	@ParameterizedTest
	@ValueSource(ints = { 1, 2, 3 })
	void testParametreAvecValueSource(int valeur) {
		Assertions.assertEquals(valeur + valeur, valeur * 2);
	}

	 @ParameterizedTest
	 @EnumSource(Month.class)
	 void testParametreAvecEnumSource(Month mois) {
	  System.out.println(mois);
	  Assertions.assertNotNull(mois);
	 }
}
```

t. Test 20 :

```
package demojunit;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;

public class TestParametre2 {
	@DisplayName("Addition")
	 @ParameterizedTest(name = "{index} : l''addition de {0} et {1}")
	 @CsvSource({ "1, 1", "1, 2", "2, 3" })
	 void testAdditioner(int a, int b) {
	  int attendu = a + b;
	  Assertions.assertEquals(attendu, a + b);
	 }
}
```

u. Test 21 :

```
package demojunit;

import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.SelectClasses;
import org.junit.runner.RunWith;

@RunWith(JUnitPlatform.class)
@SelectClasses(TestExample12.class)
public class TestSuitTest {
}
```

v. Test 22 :

```
package demojunit;

import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.SelectPackages;
import org.junit.runner.RunWith;

@RunWith(JUnitPlatform.class)
@SelectPackages("demojunit")
public class TestSuiteTest2 {

}
```

w. Test 23 :

```
package demojunit;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.RepeatedTest;

public class TesttestRepete {
	@DisplayName("test addition repété")
	@RepeatedTest(3)
	void testRepete() {
		Assertions.assertEquals(2, 1 + 1, "Valeur obtenue erronée");
	}
	@DisplayName("test addition repété")
	@RepeatedTest(value = 3, name = RepeatedTest.LONG_DISPLAY_NAME)
	void testRepete2() {
		Assertions.assertEquals(2, 1 + 1, "Valeur obtenue erronée");
	}
}
```