Centre Universitaire de Mila
Institut des sciences et de la technologie

Informatique - Semestre 02
Test logiciel - TP N° 03

TP 3

JUint 5

1. Les annotations (suite)

Le package **org.junit.jupiter.api** de **JUnit Jupiter (JUnit)** contient plusieurs annotations pour la définition et la configuration des tests. Le tableau suivant illustre quelques annotations.

Annotation	Rôle
@Order	Est une annotation utilisée pour configurer l'ordre dans lequel l'élément annoté (c'est-à-dire le champ, la méthode ou la classe) doit être évalué ou exécuté par rapport aux autres éléments de la même catégorie.
@Nested	Est utilisé pour signaler que l'élément annoté est une classe de test imbriquée et non statique (c'est-à-dire une classe interne) qui peut partager la configuration et l'état avec une instance de sa classe englobante. La classe englobante peut être une classe de test de niveau supérieur ou une autre classe de test @Nested, et l'imbrication peut être arbitrairement profonde.
@RepeatedTest	La méthode annotée est un cas de test répété plusieurs fois. Elle ne retourne pas de valeur.
@ParameterizedTest	La méthode annotée est un cas de test paramétré. Elle ne retourne pas de valeur.

Exemple1 :

- Créez un nouveau projet JAVA
- Ecrivez la classe JAVA suivante :

```java
public class PGCD {
	public int calculePGCD(int a, int b) {
	    if (a==0 || b==0) {
	        if (a==0) return b; else return a;
	        }else {
	              while(a-b != 0) {
	                   if (a>b) {
	                     a=a-b;
	                   }else {
	                        b=b-a;
	                   }
	               }
	        }
	        if (a==0) {
	              return b;
	        }else {
	              return a;
	        }

	}
}
```

- Ecrivez la classe de **Junit test** suivante en utilisant l'annotation @Nested:

```java
class NestedPGCDTest {
	private PGCD pgcd;
	@BeforeEach
	public void init() {
		pgcd = new PGCD();
	}
	@Nested
	class NombrePositifs
	{
		@Test
		public void test4() {
			System.out.println("Cas de test 4");
			assertEquals(2,pgcd.calculePGCD(14,32));
		}
		@Test
		public void test5() {
			System.out.println("Cas de test 5");
			assertEquals(1,pgcd.calculePGCD(21,13));
		}
		@Test
		public void test6() {
			System.out.println("Cas de test 6");
			assertEquals(5,pgcd.calculePGCD(30,5));
		}
		@Test
		public void test7() {
			System.out.println("Cas de test 7");
			assertEquals(3,pgcd.calculePGCD(12,15));
		}
		@Test
		public void test8() {
			System.out.println("Cas de test 8");
			assertEquals(1,pgcd.calculePGCD(12,7));
		}
	}
	@Nested
```

```
	class NombrePositifsEtNull
	{
		@Test
		public void test1() {
			System.out.println("Cas de test 1");
			assertEquals(0,pgcd.calculePGCD(0,0));
		}
		@Test
		public void test2() {
			System.out.println("Cas de test 2");
			assertEquals(5,pgcd.calculePGCD(0,5));
		}
		@Test
		public void test3() {
			System.out.println("Cas de test 3");
			assertEquals(7,pgcd.calculePGCD(7,0));
		}
	}
}
```

Exemple2 :

- Ecrivez une nouvelle classe de **Junit test** de la spécification suivante en utilisant l'annotation @order:

```
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
//@TestMethodOrder(MethodOrderer.Random.class)
class TestExecusionOrder {
	@Test
	@Order(4)
	public void test1() {
		System.out.println("Executer le test 1");;
	}
	@Test
	@Order(3)
	public void test2() {
		System.out.println("Executer le test 2");;
	}
	@Test
	@Order(2)
	public void test3() {
		System.out.println("Executer le test 3");;
	}
	@Test
	@Order(1)
	public void test4() {
		System.out.println("Executer le test 4");;
	}
}
```

- Désactivez l'annotation `@TestMethodOrder(MethodOrderer.OrderAnnotation.class)` et activez l'annotation `@TestMethodOrder(MethodOrderer.Random.class)`

- Exécutez à nouveau la classe de test. Quel est votre remarque ?

Exemple3 :

- Ecrivez une nouvelle classe de **Junit test** de la spécification suivante en utilisant l'annotation @RepeatedTest:

```java
class RepeatedClassTest {
	@RepeatedTest(3)
	void repeatedTest1() {
		System.out.println("J'ai exécuté le repeatedTest1, 3 fois");
	}
	@RepeatedTest(value=4, name= RepeatedTest.LONG_DISPLAY_NAME)
	void repeatedTest2() {
		System.out.println("J'ai exécuté ce repeatedTest2, 4 fois");
	}
	@RepeatedTest(value=4, name= RepeatedTest.SHORT_DISPLAY_NAME)
	void repeatedTest3() {
		System.out.println("J'ai exécuté ce repeatedTest3, 4 fois");
	}
	@RepeatedTest(5)
	void repeatedTest4(RepetitionInfo repetitionInofo) {
	System.out.println(repetitionInofo.getCurrentRepetition());
	}
}
```

Exemple4 :

- Ecrivez une nouvelle classe de **Junit test** de la spécification suivante en utilisant l'annotation @ParameterizedTest:

```java
class ParameterizedClassTest {

	private PGCD pgcd;
	@BeforeEach
	public void init() {
		 pgcd = new PGCD();
	}
	//-------------ValueSource----------------
	@ParameterizedTest(name="{index} Run test with args = {0}")
	@ValueSource(ints = {1,5,3,7})
	void ValueSourceTest1(int args) {
		System.out.println( args);
	}

	@ParameterizedTest
	@ValueSource(strings = {"apple","banana","orange","peach"})
	void ValueSourceTest2(String fruit) {
		System.out.println(fruit);
	}

	//-------------EnumSource----------------
	enum Fruits{
		apple,banana,orange,peach;}
	@ParameterizedTest
	@EnumSource(Fruits.class)
	void EnumSourceTest1(Fruits fruit) {
		System.out.println(fruit);
	}

	//-------------MethodSource----------------
	private static String[] fruits(){
		return new String[] {"apple","banana","orange","peach"};
	}
```

```
	@ParameterizedTest
	@MethodSource("fruits")
	void MethodSourceTest1(String fruit) {
		System.out.println(fruit);
	}

	private static Object[]values(){
		return new Object[][] {
				{"Java","Programing langage"},
				{"JUnit5","Unit testing framwork"},
				{"selenium","Automation api"},
				{"Github","Repository"}};
	}
	@ParameterizedTest
	@MethodSource("values")
	void MethodSourceTest2(String technology, String description) {
		System.out.println(technology+" : "+description);
	}

	//-------------CsvSource----------------
	@ParameterizedTest
	@CsvSource({
		"Java, Programing langage",
		"JUnit5,Unit testing framwork",
		"selenium,Automation api",
		"Github,Repository"
	})
	void CsvSourceTest(String technology, String description) {
		System.out.println(technology+" : "+description);
	}
}
```

Exercice :

- En utilisant le test paramétré et l'annotation @ParameterizedTest on veut tester la méthode calculePGCD avec les entrées suivantes : (a=0 ,b=5 → 5), (a=7,b=0 → 7), (a=14,b=32 → 2), (a=21,b=13 → 1), (a=30,b=5 → 5), (a=12,b=15→ 3), (a=12, b=7 → 1).

a) Testez la méthode calculePGCD en utilisant la source de données @CsvSource.

b) Testez la méthode calculePGCD en utilisant la source de données @MethodSource.