1 Rappels sur les tests

1.1 Méthodologie de test

Une classe de test par classe à tester. Une méthode de test par méthode ou cas à tester.
Le code d'une classe de test testant une classe `NameTestedClass` a le format suivant :

```
import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.*;

public class NameTestedClassTest {
  @Test
  void testNameTestedMethod(){
    // code containing assertions to test nameTestedMethod
  }
}
```

1.2 Assertions (liste non-exhaustive)

Pour tester, on utilise des assertions qui doivent être vraies. Vous trouverez ci-dessous une listes des assertions les plus utiles.

- `assertThat(object).isNotNull()` : vérifie que la référence **n'**est **pas** `null`
- `assertThat(actual).isSameAs(expected)` : vérifie que les deux objets sont les mêmes (même référence : utilisation de `==`).
- `assertThat(condition).isTrue()` : vérifie que `condition` est vraie.
- `assertThat(condition).isFalse()` : vérifie que `condition` est faux.
- `assertThat(actual).isEqualTo(expected)` : vérifie que `expected` est égal à `actual` (en appelant `equals` sur `actual`).
- `assertThat(actual).isNotEqualTo(expected)` : vérifie que `expected` **n'**est égal **pas** à `actual` (en appelant `equals` sur `actual`).
- `assertThat(iterable).contains(element)` : vérifie que `iterable` (qui peut être une `List`, un tableau, ...) contient `element`.
- `assertThat(iterable).containsOnly(element)` : vérifie que `iterable` (qui peut être une `List`, un tableau, ...) contient seulement `element`.
- `assertThat(actual).isCloseTo(expected, within(delta))` : vérifie que $|expected - actual| \leq delta$ (comparaison de double).

2 Test d'une classe d'entier

On nous fournit une classe `BigInteger` permettant de manipuler des entiers relatifs en précision arbitraire. Les `BigInteger` sont immutables : aucune méthode ne modifie `this`, ni ses arguments.

La classe `BigInteger` contient :

- des constantes :
  - `static BigInteger ONE` : The `BigInteger` constant one.
  - `static BigInteger TEN` : The `BigInteger` constant ten.

  - `static BigInteger ZERO` : The `BigInteger` constant zero.
- un constructeur :
  - `public BigInteger(String val)` : Translates the decimal `String` representation of a `BigInteger` into a `BigInteger`. The `String` representation consists of an optional minus or plus sign followed by a sequence of one or more decimal digits.
- des méthodes :
  - `boolean equals(Object o)` : Compares this `BigInteger` with the specified `Object` for equality.
  - `BigInteger add(BigInteger val)` : Returns a `BigInteger` whose value is `(this + val)`.
  - `BigInteger negate()` : Returns a `BigInteger` whose value is `(-this)`.
  - `BigInteger subtract(BigInteger val)` : Returns a `BigInteger` whose value is `(this - val)`.
  - `BigInteger multiply(BigInteger val)` : Returns a `BigInteger` whose value is `(this * val)`.
  - `BigInteger divide(BigInt val)` : Returns a `BigInteger` whose value is `(this / val)`.
  - `double doubleValue()` : Converts this `BigInteger` to a `double`.
  - `BigInteger abs()` : Returns a `BigInteger` whose value is the absolute value of this `BigInteger`.
  - `BigInteger gcd(BigInteger val)` : Returns a `BigInteger` whose value is the greatest common divisor of `abs(this)` and `abs(val)`.
  - `String toString()` : Returns the decimal `String` representation of this `BigInteger`.

Question 1 : Quels sont les éléments du code qu'il est important de tester en premier ?

Question 2 : Quel devrait-être le nom de la classe de test qui va tester la classe `BigInteger` ?

Question 3 : Donner le code de tests unitaires pour les différents éléments de la classe (constantes, constructeurs et méthodes).

3 Test d'une classe de file (bonus)

Les files (ou *queue*) implémentent des séquences linéaires d'objets. On peut insérer (`offer`) ou retirer (`poll`) des éléments de la file. L'élément retiré est toujours le plus ancien élément de la file toujours présent dans la file (appelé *head of the queue*), c'est-à-dire celui qu'on a inséré en premier parmi tous les éléments de la file. La classe `Queue` contient :

- des constructeurs :
  - `Queue()` Creates an empty `Queue`
  - `Queue(int[] array)` : Creates a `Queue` initially containing the elements of the given `array`, added in traversal order of the array.
- des méthodes :
  - `boolean isEmpty()` : Returns `true` if this queue contains no elements.
  - `void offer(int element)` : Inserts the specified `element` into this queue.
  - `int poll()` : Retrieves and removes the head of this queue, or returns `-1` if this queue is empty.
  - `int length()` : Returns the number of elements in this queue.

Question 4 : Écrire des tests unitaires pour chacun des constructeurs et des méthodes.