- Question1.1

  Imaginons que l'on exécute avec JUnit la classe TestFooBar. Quelles seraient les méthodes appelées et dans quel ordre ?

- Reponse 1.1:

  methode setUp, puis l'une des 6 méthodes de test puis setUp puis une autre des 6 méthodes de test puis setup puis ...

- Question1.2

  Etudiez les verdicts émis par la méthode testLouche. Expliquez.

- Reponse 1.2:

  - On obtient : java.lang.AssertionError: expected:<1> but was:<2>
  - On voit que la première assertion "saute" mais que la seconde n'est pas évaluée (puisqu'elle devrait sauter aussi). Autant que possible on essaie de ne pas mettre trop d'assertions dans une même méthode test, on cherche à vérifier un objectif à la fois. Il y a en JUnit5 le assertAll qui peut éventuellement être utilisé ici.
  - Egalement toujours bien penser quand on détecte une erreur avec JUnit que l'erreur peut en fait être dans le test !

- Question 2 :
  Etudiez les méthodes testFooInitParDefaut (de 1 à 4), ainsi que les verdicts qu'elles émettent.
  - on obtient des violations d'assertions

1. pour la 1 < java.lang.AssertionError >: ce n'est pas très parlant, c'est parce qu'on a utilisé un assertTrue alors qu'en fait on voulait comparer 2 valeurs.
2. pour la 2: java.lang.AssertionError: expected:<3> but was:<5>
   c'est donc ici la deuxième assertion qui saute. Le message est plus lisible. On a utilisé un assertEquals.

3. pour la 3 : java.lang.AssertionError: expected:<5> but was:<3>
   le message d'erreur est à l'envers, car la méthode assertEquals prend en premier paramètre la valeur attendue et pas la valeur obtenue, il faut y être vigilant.
4. pour la 4 : java.lang.AssertionError: Expected: is <5> but: was <3>
   on a utilisé hamcrest, le message est différent mais ici pas plus explicite.
   → on en déduit : de bien faire attention à la façon d'écrire les assertions, et on en déduit également qu'il y a une erreur dans le code car les tests ont l'air justes vis à vis des spécifications.

- Question 3 :

1. Que pensez-vous de la méthode de testBar ?
   elle ne teste pas grand chose, elle appelle juste la méthode bar et ne vérifie rien.
2. Essayez de concevoir une méthode de test plus pertinente. Quel problème rencontrez-vous ? Que pensez-vous de la testabilité de la classe SUT ?
   C'est très compliqué ! bar manipule des attributs privés pour lesquels on n'a pas d'accesseur. Donc cette méthode n'a pas un comportement directement observable. La classe n'est donc pas très testable en l'état.

- Question 4

1. Imaginons que l'on exécute cette classe avec JUnit. Quelles méthodes seraient exécutées ?

   Cela n'est pas évident à savoir, cela nécessite de comprendre finement comment fonctionnent els tests paramétrés en JUnit 5. De fait, d'abord la méthode static qui fournit les paramètres est appelée. Ensuite le constructeur de la classe est appelé (celui par défaut ici) puis une première fois la méthode testFoo avec le premier jeu de paramètres, puis encore le constrtcuetr, puis une deuxième fois la méthode de test puis ...
2. Avez-vous ainsi réalisé un test exhaustif de la méthode foo ?
   on ne fait jamais de test exhaustif ! On cherche juste des données de test pertinents. Ici comme ce qui

compte est juste la compararaison des valeurs de x z t entre elles, on choisit des données de test raisonnables au vu de la spec.