

Commencé le	mardi 13 décembre 2022, 08:06
État	Terminé
Terminé le	mardi 13 décembre 2022, 09:16
Temps mis	1 heure 10 min
Note	17,21 sur 21,00 (81,97%)

Question 1

Correct

Note de 2,00 sur 2,00

Nous voulons enregistrer l'historique d'évènements (*logs*). Nous choisissons de représenter chaque évènement par un **LogItem** qui enregistre l'action réalisée, et la valeur associée à cette action, sous la forme d'une string.

Par exemples :

- un évènement = *avancer de 10* sera transformé en un LogItem d'action "avancer" et de valeur "10".
- un appel : *pile.push(4)* sera transformé en un LogItem d'action "push" et de valeur "4".

👉 Implémenter la classe **LogItemImplV0** qui

1. implémente l'interface **LogItemV0** donnée ci-dessous et
2. définit un constructeur qui initialise l'action et la valeur.
3. permet de vérifier l'égalité de deux logItem par l'égalité de leur action et de leur valeur

```
interface LogItemV0 {
    String getAction();
    String getValue();
}
```

----- English version

We want to record the history of events (*logs*). We choose to represent each event by a **LogItem** that records the action performed and the value associated with this action in the form of a string.

For Example :

- an event = *advance by 10* will be transformed into a LogItem with action "advance" and value "10".
- a call : *stack.push(4)* will be transformed into a LogItem of action "push" and value "4".

👉 Implement the **LogItemImplV0** class which

1. implements the **LogItemV0 interface** given below and
2. defines a constructor that initializes the action and the value.
3. allows checking the equality of two logItem by the equality of their action and their value

```
interface LogItemV0 {
    String getAction();
    String getValue();
}
```

Par exemple:

Test	Résultat
LogItemV0 log = new LogItemImplV0("Push","One"); assertEquals("Push",log.getAction());	Push equals Push? true
LogItemV0 log = new LogItemImplV0("Push","One"); assertEquals("One",log.getValue());	One equals One? true
LogItemV0 log1 = new LogItemImplV0("Push","One"); LogItemV0 log11 = new LogItemImplV0("Push","One"); LogItemV0 log2 = new LogItemImplV0("Push","Two"); assertFalse(log1.equals(log2)); assertTrue(log1.equals(log11) ((LogItemImplV0)log1).equals((LogItemImplV0)log11)); assertFalse(log1 == log11);	false true false

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1 public class LogItemImplV0 implements LogItemV0 {
2     private String action;
3     private String value;
4
5     public LogItemImplV0( String action, String value) {
6         this.action = action;
7         this.value = value;
8     }
9
10
11     @Override
12     public String getAction() {
13         return action;
14     }
```

```
15
16     @Override
17     public String getValue() {
18         return value;
19     }
20
21     @Override
22     public boolean equals(Object o) {
23         if (o == null)
24             return false;
25         if (o == this)
26             return true;
27         if (!(o instanceof LogItemV0))
28             return false;
29         LogItemV0 other = (LogItemV0) o;
30         return action.equals(other.getAction()) && value.equals(other.getValue());
31     }
32 }
33
```

	Test	Résultat attendu	Résultat obtenu	
✓	LogItemV0 log = new LogItemImplV0("Push","One"); assertEquals("Push",log.getAction());	Push equals Push? true	Push equals Push? true	✓
✓	LogItemV0 log = new LogItemImplV0("Push","One"); assertEquals("One",log.getValue());	One equals One? true	One equals One? true	✓
✓	LogItemV0 log1 = new LogItemImplV0("Push","One"); LogItemV0 log11 = new LogItemImplV0("Push","One"); LogItemV0 log2 = new LogItemImplV0("Push","Two"); assertFalse(log1.equals(log2)); assertTrue(log1.equals(log11) ((LogItemImplV0)log1).equals((LogItemImplV0)log11));); assertFalse(log1 == log11);	false true false	false true false	✓

Tous les tests ont été réussis ! ✓

► Solution de l'auteur de la question (Java)

Correct

Note pour cet envoi : 2,00/2,00.

Question 2

Correct

Note de 3,00 sur 3,00

L'implémentation des LogItem utilisée dans la question 0 n'est pas satisfaisante, parce que l'on veut pouvoir travailler sur différents types de Log. On choisit donc de rendre générique l'interface **LogItem**. Elle devient générique sur la valeur :

```
interface LogItem <V> {
    String getAction();
    V getValue();
}
```

👉 Implémenter la classe **LogItemImpl<T>** qui implemente **LogItem<T>** et définit un constructeur comme précédemment.

Attention à partir de maintenant nous travaillons avec **LogItem** et **LogItemImpl** et pas la version V0.

----- English version

The implementation of the LogItem used in question 0 is not satisfactory because we want to be able to work on different types of Log. So we choose to make the **LogItem** interface generic, and it becomes generic on the value :

```
interface LogItem <V> {
    String getAction();
    V getValue();
}
```

👉 Implement the **LogItemImpl<T>** class qui implemente **LogItem<T>** and defines a constructor as before.

Attention from now on, we will work with **LogItem** and **LogItemImpl** and not the V0 version.

Par exemple:

Test	Résultat
LogItem<String> log = new LogItemImpl<>("Push","One"); String action = log.getAction(); assertEquals("Push",action);	Push equals Push? true
LogItem<Integer> log = new LogItemImpl<>("Push",100); int value = log.getValue(); assertEquals(100,value);	100 equals 100? true

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1 public class LogItemImpl<T> implements LogItem<T> {
2     private String action;
3     private T value;
4
5     public LogItemImpl(String action, T value) {
6         this.action = action;
7         this.value = value;
8     }
9
10    @Override
11    public String getAction() {
12        return action;
13    }
14
15    @Override
16    public T getValue() {
17        return value;
18    }
19
20    //ne pas oublier la méthode equals
21    @Override
22    @SuppressWarnings("unchecked")
23    public boolean equals(Object o) {
24        if (o == null)
25            return false;
```

```

26         if (o == this)
27             return true;
28         if (!(o instanceof LogItem))
29             return false;
30         LogItem other = (LogItem) o;
31         return action.equals(other.getAction()) && value.equals(other.getValue());
32     }
33
34
35
36 }
37

```

	Test	Résultat attendu	Résultat obtenu	
✓	LogItem<String> log = new LogItemImpl<>("Push", "One"); String action = log.getAction(); assertEquals("Push", action);	Push equals Push? true	Push equals Push? true	✓
✓	LogItem<String> log = new LogItemImpl<>("Push", "One"); assertEquals("One", log.getValue());	One equals One? true	One equals One? true	✓
✓	LogItem<Integer> log = new LogItemImpl<>("Push", 1); int value = log.getValue(); assertEquals(1, value);	1 equals 1? true	1 equals 1? true	✓
✓	LogItem<Integer> log = new LogItemImpl<>("Push", 100); int value = log.getValue(); assertEquals(100, value);	100 equals 100? true	100 equals 100? true	✓
✓	LogItem<Integer> log1 = new LogItemImpl<>("Push", 1); LogItem<LogItem<Integer>> log = new LogItemImpl<>("Push", log1); assertTrue(log1.equals(log.getValue()) ((LogItemImpl)log1).equals((LogItemImpl)log.getValue()));	true	true	✓
✓	LogItem<Integer> log1 = new LogItemImpl<>("Push", 1); LogItem<LogItem<Integer>> log = new LogItemImpl<>("Push", log1); assertTrue(new LogItemImpl<>("Push", 1).equals((LogItemImpl) log.getValue()));	true	true	✓

Tous les tests ont été réussis ! ✓

► **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 3,00/3,00.

Etant donnée une liste de **LogItem<Integer>** calculer la somme des valeurs des logItem.

👉 Implémenter la **méthode** `public int sum(List<LogItem<Integer>> logs)`.

Rappels :

```
public interface LogItem <V> {
    String getAction();
    V getValue();
}
```

-- English version

Given a list of LogItem<Integer>, calculate the sum of the logItem values.

👉 Implement the method `public int sum(List<LogItem<Integer>> logs)`.

Recall:

```
public interface LogItem <V> {
    String getAction();
    V getValue();
}
```

Par exemple:

Test	Résultat
<pre>List<LogItem<Integer>> gameTrace = new ArrayList<>(); gameTrace.add(new LogItemImpl<>("win" , 100)); gameTrace.add(new LogItemImpl<>("lost" , -10)); gameTrace.add(new LogItemImpl<>("win" , 50)); gameTrace.add(new LogItemImpl<>("win" , 60)); int sum = manager.sum(gameTrace); assertEquals(200,sum);</pre>	200 equals 200? true

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1 int sum( List<LogItem<Integer>> logs) {
2     int sum = 0;
3     for (LogItem<Integer> log : logs) {
4         sum += log.getValue();
5     }
6     return sum;
7 }
```

	Test	Résultat attendu	Résultat obtenu	
✓	<pre>List<LogItem<Integer>> gameTrace = new ArrayList<>(); gameTrace.add(new LogItemImpl<>("win" , 100)); gameTrace.add(new LogItemImpl<>("lost" , -10)); gameTrace.add(new LogItemImpl<>("win" , 50)); gameTrace.add(new LogItemImpl<>("win" , 60)); int sum = manager.sum(gameTrace); assertEquals(200,sum);</pre>	200 equals 200? true	200 equals 200? true	✓
✓	<pre>List<LogItem<Integer>> gameTrace = new ArrayList<>(); int sum = manager.sum(gameTrace); assertEquals(0,sum);</pre>	0 equals 0? true	0 equals 0? true	✓
✓	<pre>List<LogItem<Integer>> gameTrace = new ArrayList<>(); gameTrace.add(new LogItemImpl<>("win" , 100)); gameTrace.add(new LogItemImpl<>("win" , 50)); gameTrace.add(new LogItemImpl<>("win" , 60)); int sum = manager.sum(gameTrace); assertEquals(210,sum);</pre>	210 equals 210? true	210 equals 210? true	✓

Tous les tests ont été réussis ! ✓

► **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 2,00/2,00.

Question 4

Correct

Note de 3,00 sur 3,00

Etant donnée une liste de LogItem, nous vous demandons de les trier sur leurs actions (comparaison de String simple).

👉 Implémenter la méthode :

```
public List<LogItem> sortByAction(List<LogItem> logs)
```

Vous pouvez utiliser un comparator.

Rappel :

```
public interface LogItem <V> {
    String getAction();
    V getValue();
}
```

--- English version

Given a list of LogItems, we ask you to sort them on their actions (simple String comparison).

👉 Implement the method:

```
public List<LogItem> sortByAction(List<LogItem> logs)
```

You can use a comparator.

Recall :

```
public interface LogItem <V> {
    String getAction();
    V getValue();
}
```







Par exemple:

Test	Résultat
<pre>List<LogItem> gameTrace = new ArrayList<> (); gameTrace.add(new LogItemImpl<>("forward" , 100)); gameTrace.add(new LogItemImpl<>("right" , 10)); gameTrace.add(new LogItemImpl<>("left" , 20)); gameTrace.add(new LogItemImpl<>("stop" , 60)); System.out.println("Before : " + gameTrace); List<LogItem> res = manager.sortByAction(gameTrace); System.out.println("Sorted list : " + gameTrace); String actionOne = gameTrace.get(0).getAction(); String actionTwo = gameTrace.get(1).getAction(); assertEquals("forward",actionOne); assertEquals("left",actionTwo);</pre>	<pre>Before :[Log{value=100, action='forward'}, Log{value=10, action='right'}, Log{value=20, action='left'}, Log{value=60, action='stop'}] Sorted list : [Log{value=100, action='forward'}, Log{value=20, action='left'}, Log{value=10, action='right'}, Log{value=60, action='stop'}] forward equals forward? true left equals left? true</pre>

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1 //sort the logItem on their action
2 public List<LogItem> sortByAction(List<LogItem> logs){
3     Collections.sort(logs, new Comparator<LogItem>() {
4         public int compare(LogItem o1, LogItem o2) {
5             return o1.getAction().compareTo(o2.getAction());
6         }
7     });
8     return logs;
9 }
```


Test	Résultat attendu	Résultat obtenu	
 <pre> List<LogItem> gameTrace = new ArrayList<>(); gameTrace.add(new LogItemImpl<> ("win" , 100)); gameTrace.add(new LogItemImpl<> ("lost" , -10)); gameTrace.add(new LogItemImpl<> ("equals" , 20)); gameTrace.add(new LogItemImpl<> ("win" , 60)); List<LogItem> res = manager.sortByAction(gameTrace); String actionOne = gameTrace.get(0).getAction(); String actionTwo = gameTrace.get(1).getAction(); assertEquals("equals",actionOne); assertEquals("lost",actionTwo); </pre>	<pre> equals equals equals? true lost equals lost? true </pre>	<pre> equals equals equals? true lost equals lost? true </pre>	
 <pre> List<LogItem> gameTrace = new ArrayList<>(); gameTrace.add(new LogItemImpl<> ("forward" , 100)); gameTrace.add(new LogItemImpl<> ("right" , 10)); gameTrace.add(new LogItemImpl<> ("left" , 20)); gameTrace.add(new LogItemImpl<> ("stop" , 60)); System.out.println("Before :" + gameTrace); List<LogItem> res = manager.sortByAction(gameTrace); System.out.println("Sorted list : " + gameTrace); String actionOne = gameTrace.get(0).getAction(); String actionTwo = gameTrace.get(1).getAction(); assertEquals("forward",actionOne); assertEquals("left",actionTwo); </pre>	<pre> Before :[Log{value=100, action='forward'}, Log{value=10, action='right'}, Log{value=20, action='left'}, Log{value=60, action='stop'}] Sorted list : [Log{value=100, action='forward'}, Log{value=20, action='left'}, Log{value=10, action='right'}, Log{value=60, action='stop'}] forward equals forward? true left equals left? true </pre>	<pre> Before :[Log{value=100, action='forward'}, Log{value=10, action='right'}, Log{value=20, action='left'}, Log{value=60, action='stop'}] Sorted list : [Log{value=100, action='forward'}, Log{value=20, action='left'}, Log{value=10, action='right'}, Log{value=60, action='stop'}] forward equals forward? true left equals left? true </pre>	
 <pre> List<LogItem> gameTrace = new ArrayList<>(); gameTrace.add(new LogItemImpl<> ("push" , 100)); gameTrace.add(new LogItemImpl<> ("push" , 10)); gameTrace.add(new LogItemImpl<> ("pop" , 20)); List<LogItem> res = manager.sortByAction(gameTrace); String actionOne = gameTrace.get(0).getAction(); String actionTwo = gameTrace.get(1).getAction(); assertEquals("pop",actionOne); assertEquals("push",actionTwo); </pre>	<pre> pop equals pop? true push equals push? true </pre>	<pre> pop equals pop? true push equals push? true </pre>	

Tous les tests ont été réussis ! 

► **Solution de l'auteur de la question (Java)**

Correct

Note pour cet envoi : 3,00/3,00.

Question 5

Correct

Note de 3,00 sur 3,00

- 👉 Implémenter la classe **LogItemStack** qui permet de manipuler des piles de **LogItem<V>** telle que :
- 1. elle offre les méthodes de l'interface **StackInterface** qui définit les méthodes standards d'une pile (voir ci-dessous)
 - 2. elle implémente une méthode **boolean lastAction(String currentAction)** qui renvoie vraie si le dernier *LogItem* empilé correspond à une action égale à *currentAction* et faux sinon.

Rappels, les classes et interfaces suivantes vous sont données :

```
public interface StackInterface<T> {
    boolean isEmpty();
    int size();
    T peek() throws EmptyStackException;
    void push(T x);
    T pop() throws EmptyStackException;
}

public class ArrayStack<T> implements StackInterface<T> {
```

-- english version

- 👉 I Implement the **LogItemStack** class that allows to manipulate stacks of **LogItem<V>** such as:
- 1. it offers the methods of the **StackInterface** interface which defines the standard methods of a stack (see below)
 - 2. it implements **boolean lastAction(String currentAction)** which returns true if the last stacked LogItem corresponds to an action equal to currentAction and false otherwise.

Recall, the following classes and interfaces are given :

```
public interface StackInterface<T> {
    boolean isEmpty();
    int size();
    T peek() throws EmptyStackException;
    void push(T x);
    T pop() throws EmptyStackException;
}

public class ArrayStack<T> implements StackInterface<T> {
```

Par exemple:

Test	Résultat
LogItemStack<Integer> lgs = new LogItemStack<>(); lgs.push(new LogItemImpl<>(PUSH,2)); lgs.push(new LogItemImpl<>(PUSH,3)); lgs.push(new LogItemImpl<>(POP,3)); assertTrue(lgs.lastAction(POP)); //Do not modify the stack assertTrue(lgs.lastAction(POP)); assertFalse(lgs.lastAction(PUSH));	true true false

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1 //To Implement
2 public class LogItemStack<V>
3     extends ArrayStack<LogItem<V>>{
4
5     boolean lastAction(String action) {
6         try {
7             return peek().getAction().equals(action);
8         } catch (EmptyStackException e) {
9             return false;
10        }
11    }
12 }
```

	Test	Résultat attendu	Résultat obtenu	
✓	<pre>LogItemStack<Integer> lgs = new LogItemStack<>(); lgs.push(new LogItemImpl<>(PUSH,2)); lgs.push(new LogItemImpl<>(PUSH,3)); lgs.push(new LogItemImpl<>(POP,3)); assertEquals(3,lgs.size());</pre>	<pre>3 equals 3 ? true</pre>	<pre>3 equals 3 ? true</pre>	✓
✓	<pre>LogItemStack<Integer> lgs = new LogItemStack<>(); lgs.push(new LogItemImpl<>(PUSH,2)); lgs.push(new LogItemImpl<>(PUSH,3)); lgs.push(new LogItemImpl<>(POP,3)); assertTrue(lgs.lastAction(POP)); //Do not modify the stack assertTrue(lgs.lastAction(POP)); assertFalse(lgs.lastAction(PUSH));</pre>	<pre>true true false</pre>	<pre>true true false</pre>	✓
✓	<pre>LogItemStack<Integer> lgs = new LogItemStack<>(); assertFalse(lgs.lastAction(PUSH));</pre>	<pre>false</pre>	<pre>false</pre>	✓

Tous les tests ont été réussis ! ✓

► Solution de l'auteur de la question (Java)

Correct

Note pour cet envoi : 3,00/3,00.

Question 6

Incorrect

Note de 2,50 sur 5,00

👉 Implémenter une classe **UndoableStack<T>** :

- 1) Elle implémente l'interface **StackInterface<T>**
- 2) Elle implémente l'interface **UndoInterface**.

A chaque appel à undo la dernière action push ou pop est défaite.

Ainsi soit s1 une pile vide, après la séquence : s1.push(1), s1.push(2), s1.undo() -> s1 = [1 ->

- 3) Elle implémente la méthode **public LogItem<T> lastAction()** qui renvoie la dernière action réalisée *push* ou *pop*.

Aides :

- A chaque appel à push ou pop enregistrer dans une **LogItemStack** l'action réalisée.

Les classes et interfaces suivantes sont données :

```
interface UndoInterface {
    boolean undo();
}
```

```
interface StackInterface<T> {
    boolean isEmpty();
    int size();
    T peek() throws EmptyStackException;
    void push(T x);
    T pop() throws EmptyStackException;
}
```

```
class ArrayStack<T> implements StackInterface<T>.
```

```
interface LogItem <V> {
    String getAction();
    V getValue();
}
```

```
class LogItemImpl<V> implements LogItem<V>
```

```
class LogItemStack<V> une pile de logItem<V> (voir question précédente).
```

--- English version

👉 Implément a class **UndoableStack<T>** :

- 1) It implements **StackInterface<T>** interface.
- 2) It implements interface **UndoInterface**.

At each call to undo the last push or pop action is undone.

So let s1 be an empty stack, after the sequence: s1.push(1), s1.push(2), s1.undo() -> s1 = [1 ->

- 3) It implements the method **public LogItem<T> lastAction()** which returns the last action performed *push* ou *pop*.

Helps :

- At each call to push or pop record in a **LogItemStack** the action performed.

The following classes and interfaces are given:

```
interface UndoInterface {
    boolean undo();
}
```

```
interface StackInterface<T> {
    boolean isEmpty();
    int size();
    T peek() throws EmptyStackException;
    void push(T x);
    T pop() throws EmptyStackException;
}
```

```
class ArrayStack<T> implements StackInterface<T>.
```

```
interface LogItem <V> {
    String getAction();
    V getValue();
}
```

```
class LogItemImpl<V> implements LogItem<V>
```

```
class LogItemStack<V> une pile de logItem<V> (voir question précédente).
```

Par exemple:

Test	Résultat
<pre>//To check undo on push UndoableStack<Integer> stack = new UndoableStack<>(); stack.push(100); stack.push(2); //undo the last push, the size is 1, the top is 100 boolean b = stack.undo(); assertEquals(1,stack.size()); int peekValue = stack.peek(); assertEquals(100,peekValue);</pre>	<pre>1 equals 1 ? true 100 equals 100 ? true</pre>
<pre>//To check lastAction UndoableStack<Integer> stack = new UndoableStack<>(); stack.push(1); stack.push(2); LogItem<Integer> action = stack.lastAction(); assertEquals("push",action.getAction()); int value = action.getValue(); assertEquals(2,value);</pre>	<pre>push equals push ? true 2 equals 2 ? true</pre>
<pre>UndoableStack<Integer> stack = new UndoableStack<>(); stack.push(1); stack.push(2); stack.pop(); stack.undo(); LogItem<Integer> action = stack.lastAction(); assertEquals("push",action.getAction()); int value = action.getValue(); assertEquals(2,value);</pre>	<pre>push equals push ? true 2 equals 2 ? true</pre>

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1 public class UndoableStack<T>
2     extends ArrayStack<T>
3     implements UndoInterface {
4
5         //A Implémenter
6
7         public LogItem<T> lastAction() throws EmptyStackException {
8             try {
9                 return peek();
10            } catch (EmptyStackException e) {
11                return null;
12            }
13        }
14
15        public boolean undo() {
16            try {
17                pop();
18                return true;
19            } catch (EmptyStackException e) {
20                return false;
21            }
22        }
23    }
```

Erreur(s) de syntaxe

__tester__.java:19: error: incompatible types: T cannot be converted to LogItem<T>
return peek();
^
where T is a type-variable:
T extends Object declared in class UndoableStack
1 error

▼ Solution de l'auteur de la question (Java)

```
1 class UndoableStack<T> extends ArrayStack<T> implements UndoInterface {
2     private static final String PUSH_ACTION = "push";
3     private static final String POP_ACTION = "pop";
4
5     LogItemStack<T> actions ;
6 }
```

```
7 // Build an undoable stack
8 public UndoableStack() {
9     super();
10    actions = new LogItemStack<>();
11 }
12
13
14 // Push the value x onto the stack.
15 @Override
16 public void push(T x) {
17     super.push(x);
18     actions.push(new LogItemImpl<T>(PUSH_ACTION, x));
19 }
20
21 // Pop the stack and return the value popped
22 @Override
23 public T pop() throws EmptyStackException {
24     T value = super.pop();
25     actions.push(new LogItemImpl<T>(POP_ACTION, value));
26     return value;
27 }
28
29 // undo le dernier effectif push ou pop
30 public boolean undo() {
31     try {
32         if ((actions.isEmpty())) {
33             return false;
34         }
35         LogItem<T> action;
36         action = actions.pop();
37         if (action.getAction().equals(PUSH_ACTION)) {
38             super.pop();
39         } else {
40             super.push(action.getValue());
41         }
42     } catch (EmptyStackException e) {
43         assert false : "unexpected exception in undo ";
44     }
45     return true;
46 }
47
48 public LogItem<T> lastAction() throws EmptyStackException {
49     return actions.peek();
50 }
51
52 }
```

Incorrect

Note pour cet envoi : 0,00/5,00.

//

- 🔗 Implémenter une classe **ReversibleQueueImpl** telle que :
- elle offre toutes les méthodes d'une file (Queue) telle que définie dans **EnhancedQueueInterface** (rappel : la classe EnhancedQueueImpl implémente cette interface et vous ai donnée);
 - elle implémente une méthode **reverse()** qui renvoie une nouvelle file correspondant à la file courante mais renversée .

public EnhancedQueueInterface<T> reverse() throws EmptyQueueException

Soit une file : <- 1 2 3 <- où 1 est la tête et 3 la queue.
Renverser cette file renverra une file : <- 3 2 1 <-

Given :

```
- class ArrayStack<T> implements StackInterface<T>
- interface EnhancedQueueInterface<T> extends QueueInterface<T>{
    EnhancedQueueInterface<T> copy(); }

- public class EnhancedQueueImpl<T> implements EnhancedQueueInterface<T> {...
- interface QueueInterface<T>
```

----- English version -----

- 🔗 implement a **ReversibleQueueImpl** class such that :
- it offers all the methods of a queue (Queue) as defined in EnhancedQueueInterface (reminder: the EnhancedQueueImpl class implements this interface and is given);
 - it implements a reverse() method which returns a new queue corresponding to the current queue but reversed.

Let's have a queue: <- 1 2 3 <- where 1 is the head and 3 is the tail. Reversing this queue will return a queue: <- 3 2 1 <-

Par exemple:

Test	Résultat
<pre>ReversibleQueueImpl<Integer> q = new ReversibleQueueImpl<Integer>(); q.enqueue(1); q.enqueue(2); q.enqueue(3); QueueInterface<Integer> res = q.reverse(); System.out.println("Reversed queue : " + res); System.out.println("q is not modified : " + q); int value = q.peek(); assertEquals(1,value); assertEquals(3,q.size); q.dequeue();q.dequeue(); value = q.peek(); assertEquals(3,value); value = res.peek(); assertEquals(3,value); res.dequeue(); value = res.peek(); assertEquals(2,value); res.dequeue(); value = res.peek(); assertEquals(1,value);</pre>	<pre>Reversed queue : <- 3 2 1 <- q is not modified : <- 1 2 3 <- 1 equals 1 ? true 3 equals 3 ? true 3 equals 3 ? true 3 equals 3 ? true 2 equals 2 ? true 1 equals 1 ? true</pre>

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
1 public class ReversibleQueueImpl<T> extends EnhancedQueueImpl<T> {
2
3     public QueueInterface<T> reverse() throws EmptyQueueException {
4         EnhancedQueueInterface<T> reversed = this.copy();
5         QueueInterface<T> result = new EnhancedQueueImpl<T>();
6         while (!reversed.isEmpty()) {
7             result.enqueue(reversed.dequeue());
8         }
9         return result;
10    }
11 }
12
```

	Test	Résultat attendu	Résultat obtenu	
✓	<pre>ReversibleQueueImpl<Integer> q = new ReversibleQueueImpl<Integer>(); QueueInterface<Integer> res = q.reverse(); System.out.println("Reversed queue : " + res); System.out.println("q is not modified : " + q); assertTrue(res.isEmpty());</pre>	<pre>Reversed queue : <- <- q is not modified : <- <- true</pre>	<pre>Reversed queue : <- <- q is not modified : <- <- true</pre>	✓
✓	<pre>ReversibleQueueImpl<Integer> q = new ReversibleQueueImpl<Integer>(); q.enqueue(10); QueueInterface<Integer> res = q.reverse(); System.out.println("Reversed queue : " + res); System.out.println("q is not modified : " + q); assertEquals(1,res.size());</pre>	<pre>Reversed queue : <- 10 <- q is not modified : <- 10 <- 1 equals 1 ? true</pre>	<pre>Reversed queue : <- 10 <- q is not modified : <- 10 <- 1 equals 1 ? true</pre>	✓
✗	<pre>ReversibleQueueImpl<Integer> q = new ReversibleQueueImpl<Integer>(); q.enqueue(1); q.enqueue(2); q.enqueue(3); QueueInterface<Integer> res = q.reverse(); System.out.println("Reversed queue : " + res); System.out.println("q is not modified : " + q); int value = q.peek(); assertEquals(1,value); assertEquals(3,q.size()); q.dequeue();q.dequeue(); value = q.peek(); assertEquals(3,value); value = res.peek(); assertEquals(3,value); res.dequeue(); value = res.peek(); assertEquals(2,value); res.dequeue(); value = res.peek(); assertEquals(1,value);</pre>	<pre>Reversed queue : <- 3 2 1 <- q is not modified : <- 1 2 3 <- 1 equals 1 ? true 3 equals 3 ? true 3 equals 3 ? true 3 equals 3 ? true 2 equals 2 ? true 1 equals 1 ? true</pre>	<pre>Reversed queue : <- 1 2 3 <- q is not modified : <- 1 2 3 <- 1 equals 1 ? true 3 equals 3 ? true 3 equals 3 ? true 3 equals 1 ? false 2 equals 2 ? true 1 equals 3 ? false</pre>	✗

Montrer les différences

▼ Solution de l'auteur de la question (Java)

```

1 public class ReversibleQueueImpl<T> extends EnhancedQueueImpl<T> {
2
3     public EnhancedQueueInterface<T> reverse() throws EmptyQueueException {
4         EnhancedQueueInterface<T> tmp = this.copy();
5
6         ArrayStack<T> stackToReverse = new ArrayStack<>();
7         while (! tmp.isEmpty()){
8             T value = tmp.dequeue();
9             stackToReverse.push(value);
10        }
11
12        EnhancedQueueInterface<T> reversed = new EnhancedQueueImpl<>();
13        while (! stackToReverse.isEmpty()){
14            try {
15                reversed.enqueue(stackToReverse.pop());
16            } catch (EmptyStackException e) {
17                assert false : "unexpected situation";
18            }
19        }
20
21        return reversed;
22    }
23 }
```

Partiellement correct

Note pour cet envoi : 1,71/3,00.

