Formation : INFO2

Rapport du TD n°05
Java Avancé

Guillaume CAU

21/10/2019

Exercice 01 :

Question 01 :

Dans cette File, nous ne pouvons pas dire que la file est vide si *tail* est strictement inférieure à *head*. En effet, la liste est circulaire, *tail* peut donc être inférieure à *head*.

La méthode restante la plus simple est donc de compter le nombre d'éléments dans le tableau. Il faudra donc stocker un champ de type **int** dans notre classe FIFO.
Lorsque le champ de type **int** contiendra la valeur 0, c'est que la file sera vide.

Question 02 :

Pour créer la classe générique représentant une Fifo, j'ai utilisé différentes variables :

- size : De type **int**, cette variable représente la taille des éléments du tableau.
- maxSize : De type **int,** cette variable représente la taille maximale du tableau.
- tail : De type **int**, cette variable représente l'indice de la queue dans le tableau.
- head : De type **int**, cette variable représente l'indice de la tête dans le tableau
- tab : De type **E[]**, ce tableau représente notre file.

Pour initialiser ces variables, j'ai créé un constructeur. Ce constructeur prend en paramètre la taille maximale du tableau. Cette valeur doit être vérifiée. En effet, nous ne souhaitons pas que l'utilisateur nous envoie une taille négative. Le

constructeur lèvera donc une ***IllegalArgumentException*** si une valeur négative est envoyée.

Toutes les autres variables **int** sont initialisée à 0.
Pour la variable *tab,* il est plus difficile de l'initialiser. En effet, en Java, il est impossible de créer un tableau de type paramétré lors de l'exécution. Je suis donc dans l'obligation de créer un tableau d'objets qui sera ensuite casté en **E[]**.
Le cast précédent provoque un warning. Pour s'en débarrasser, on peut utiliser l'annotation *@SuppressWarnings(''unchecked'')*.

Question 03 :

Pour détecter que la file est pleine, il faut vérifier que la taille actuelle est égale à la taille maximale du tableau.
Si la file est plaine, nous pouvons :

- Écraser l'élément le plus vieux.
- Augmenter la taille du tableau
- Lever une exception `IllegalStateException`

Dans notre situation, il est préférable d'utiliser la 3eme option.

Question 04 :

Si la file est vide, lors d'un poll, il faut lever une `IllegalStateException` `également.`

Question 05 :

La fonction StringJoiner a été utilisé pour réaliser la fonction d'affichage plus facilement.

Question 06 :

Un *memory leak* est lorsque un objet/type primitif qui n'est plus utilisé dans l'application est toujours conservée en mémoire et ne sera pas *free* par le garbage collector.

Dans notre application, un memory leak peut survenir. En effet, si, lors d'un poll, nous ne faisons qu'incrémenter la variable *head*, le tableau contient encore la référence vers l'objet, le carbage collector ne la libérera donc pas.

Pour ce faire, lors du poll, il faut mettre la case du tableau a **null** après l'avoir reécupérée.

Question 07 :

Les méthodes size et isEmpty sont pertinentes pour les utilisateurs souhaitant utiliser ton code et pour suivre certaine norme en quelque sorte.

Question 08 :

Un itérateur est une interface permettant de parcourir des ensembles/listes de façon uniforme.
Le retour de la méthode iterator() doit renvoyer un objet du type ***Iterator***.

Question 09 :

Pour éviter de provoquer des comportement non-voulu, l'itérateur ne doit pas modifier notre file. Il doit être en « read-only ».

Question 10 :

L'interface ***Iterable*** permet de rendre uniforme l'utilisation d'objets souhaitant être iterable. En effet, les interfaces obliges les sous-types a implémenter certaines fonctions obligatoirement.

Exercice 02 :

Question 01 :

Si la fille est pleine, il faut désormais augmenter sa taille par deux.

Pour ce faire, il faut créer un tableau dit *buffer* qui nous permet de faire étape intermédiaire.

Ensuite, il faut copier la partie de head à la fin du tableau dans le nouveau puis la partiede tail à head.

Conclusion :

La partie algorithmique de ce TP était plus difficile que la partie technique. En effet, les notions utilisées ici avaient déjà été utilisées dans les Tps précédents. J'ai donc réutiliser mes compétences pour utiliser les types paramétrés et les classes internes.
J'ai également compris qu'il faut faire attention au memory leaks. Ils peuvent survenir si nous ne faisons pas attention.
J'avais également la possibilité d'utiliser les méthodes de AbstractQueue pour éviter de réinventer du code.

J'ai donc améliorés mes compétences.