Projet Programmation Fonctionnelle
Une classe liste FIFO

Simon Camille 20/01/2017

Ce compte-rendu complète le code présent dans le fichier *Queue.scala*.

1 Rappel du sujet

Dans les listes chaînées que nous avons rencontré jusqu'à présent, nous pouvons avoir accès au premier élément de la liste en temps constant (O(1)). En revanche avoir accès au dernier élément (celui entré en premier dans la liste) prend un temps proportionnel à la taille de la liste (O(n)).

Nous souhaiterions mettre en place une liste spécifique permettant un accès FIFO (first-in, first-out) c'est-à-dire une liste dans laquelle on peut retirer le dernier élément en temps raisonnable. Une telle liste est souvent appelée « queue ». Nous souhaiterions en outre conserver l'accès au premier élément en temps constant.

Une façon de faire cela consiste à utiliser en interne deux listes nommées in et out, entrée, et sortie. La liste d'entrée reçoit les éléments que l'on ajoute au fil du temps. La seconde reste vide tant que l'on n'a pas besoin de retirer d'élément à la fin. Dès que cette occasion se présente, on inverse la liste d'entrée et on l'assigne à la sortie, la liste d'entrée devient alors vide.

L'opération d'inversion ne se produit en retirant les éléments à la fin que si la liste de sortie est vide. C'est pourquoi on dit que la structure fonctionne en temps constant « amorti ».

2 Présentation des méthodes

- *enqueue* : elle prend en paramètre l'élément à ajouter à la liste et le place au début de la liste *in*.
- *dequeue* : cette méthode ne prend pas de paramètre et retourne un couple constitué d'une part par l'élément le plus ancien de la liste, et d'autre part par le nouvel état de la file d'attente.

  Dans cette méthode il faut traiter différemment les situations en fonction des états de *in* et de *out*.

  - Si *in* est vide et *out* est vide, on lève une exception.
  - Si *in* est vide et *out* est non vide, on affiche le première élément de *out* et on le retire de la file.
  - Si *in* est non vide et *out* est vide, on place dans *out* l'inverse de la liste *in* puis on retourne le première élément de *out* que l'on retire de la liste.
  - Si *in* est non vide et *out* est non vide, on inverse la liste *in* que l'on vient placer à la suite de la liste *out* déjà existante. On retourne le première élément de *out* ainsi que la file moins cet élément.

- *head* : permet d'accéder au premier élément de la liste sans l'enlever. On entend par élément le plus ancien, celui qui a été ajouté en premier à la liste.

  Dans cette méthode il faut traiter différemment les situations en fonction des états de *in* et de *out*.

  - Si *in* est vide et *out* est vide, on lève une exception.
  - Si *in* est non vide et *out* est vide, on retourne le premier élément de l'inverse de *in*.
  - Si *out* est non vide et quelque soit l'état de *in*, on affiche le premier élément de *out*.

- *rear* : permet d'accéder au dernier élément de la liste, c'est à dire, celui qui a été ajouté en dernier.

  Dans cette méthode il faut traiter différemment les situations en fonction des états de *in* et de *out*.

  - Si *in* est vide et *out* est vide, on lève une exception.
  - Si *in* est vide et *out* est non vide, on retourne le premier élément de l'inverse de *out*.
  - Si *in* est non vide et quelque soit l'état de *out*, on affiche le premier élément de *in*.

- *isEmpty* : retourne vrai si la file est vide, faux sinon.

- *length* : retourne la taille de la file, c'est à dire, le nombre d'élément dans *in* additionné au nombre d'élément dans *out*.