Département de génie informatique et de génie logiciel

LOG3430

Méthodes de tests et validations

TP3

Test OO - MaDUM

Soumis par :

Jean-Frédéric Fontaine (1856632)

Sébastien Cadorette (1734603)

29 octobre 2018

Soumis à : Hiba Bagane

Polytechnique Montréal

Table des matières

Matrice MaDUM

Matrice MaDUM						
	Queue	isEmpty	size	peek	enqueue	dequeue
first	C	O		R	T	T
last	C				T	T
n	C		R		T	T

Tableau 1 : Matrice MaDUM de la classe Queue

C : Constructeur

R : Rapporteurs

T : Transformateurs

O : Autres

Tranches

La classe Queue a 3 attributs, donc elle possède 3 tranches.

1. $Slice_{first}$(Queue) = < first, { Queue(), isEmpty(), peek(), enqueue(), dequeue() } >
2. $Slice_{last}$(Queue) = < last, { Queue(), isEmpty(), enqueue(), dequeue() } >
3. $Slice_{n}$(Queue) = < n, { Queue(), size(), enqueue(), dequeue() } >

Séquences

- $Slice_{first}$:
    - Queue() -> isEmpty() -> peek() -> enqueue() -> dequeue()
    - Queue() -> isEmpty() -> peek() -> dequeue() -> enqueue()
    - Queue() -> isEmpty() -> enqueue() -> dequeue() -> peek()
    - Queue() -> isEmpty() -> enqueue() -> peek() -> dequeue()
    - Queue() -> isEmpty() -> dequeue() -> enqueue() -> peek()
    - Queue() -> isEmpty() -> dequeue() -> peek() -> enqueue()
    - Queue() -> peek() -> isEmpty() -> enqueue() -> dequeue()
    - Queue() -> peek() -> isEmpty() -> dequeue() -> enqueue()
    - Queue() -> peek() -> enqueue() -> isEmpty() -> dequeue()
    - Queue() -> peek() -> enqueue() -> dequeue() -> isEmpty()
    - Queue() -> peek() -> dequeue() -> isEmpty() -> enqueue()
    - Queue() -> peek() -> dequeue() -> enqueue() -> isEmpty()
    - Queue() -> enqueue() -> isEmpty() -> peek() -> dequeue()
    - Queue() -> enqueue() -> isEmpty() -> dequeue() -> peek()
    - Queue() -> enqueue() -> peek() -> isEmpty() -> dequeue()
    - Queue() -> enqueue() -> peek() -> dequeue() -> isEmpty()
    - Queue() -> enqueue() -> dequeue() -> isEmpty() -> peek()
    - Queue() -> enqueue() -> dequeue() -> peek() -> isEmpty()
    - Queue() -> dequeue() -> isEmpty() -> peek() -> enqueue()
    - Queue() -> dequeue() -> isEmpty() -> enqueue() -> peek()
    - Queue() -> dequeue() -> peek() -> isEmpty() -> enqueue()
    - Queue() -> dequeue() -> peek() -> enqueue() -> isEmpty()
    - Queue() -> dequeue() -> enqueue() -> isEmpty() -> peek()

    - Queue() -> dequeue() -> enqueue() -> peek() -> isEmpty()
- $Slice_{last}$:
    - Queue() -> isEmpty() -> enqueue() -> dequeue()
    - Queue() -> isEmpty() -> dequeue() -> enqueue()
    - Queue() -> enqueue() -> isEmpty() -> dequeue()
    - Queue() -> enqueue() -> dequeue() -> isEmpty()
    - Queue() -> dequeue() -> isEmpty() -> enqueue()
    - Queue() -> dequeue() -> enqueue() -> isEmpty()
- $Slice_{n}$:
    - Queue() -> size() -> enqueue() -> dequeue()
    - Queue() -> size() -> dequeue() -> enqueue()
    - Queue() -> enqueue() -> size() -> dequeue()
    - Queue() -> enqueue() -> dequeue() -> size()
    - Queue() -> dequeue() -> size() -> enqueue()
    - Queue() -> dequeue() -> enqueue() -> size()

Tests

Tester les rapporteurs

Pour les tests rapporteurs, il faut utiliser les méthodes « setters » et vérifier que la valeur est bonne avec les « getters ».

Attribut n

Il y a une méthode « getter » pour l'attribut *n*. Il s'agit de la méthode *size()*.

d1 = < { Queue() -> size() } , { size() == 0 } >

Attribut first

Il y a une méthode « getter » pour l'attribut *first*. Il s'agit de la méthode *peek()*. Il y a deux cas possibles, soit lorsque la variable vient tout juste d'être initialisé et lorsqu'elle contient des éléments. Voici donc les deux tests :

d2 = < { Queue() -> peek() } , { peek() == Error Queue underflow } >

d3 = < { Queue() -> enqueue(1) -> peek() } , { peek() == 1 } >

Tester les constructeurs

Pour les tests constructeurs, il faut utiliser le constructeur et les « getters » pour vérifier que les attributs sont correctement initialisés.

Avec le test de rapporteur, on a déjà testé l'attribut *n*. Il ne reste qu'à vérifier que les attributs *first* et *last* sont également bien créés. Toutefois, l'attribut *last* n'a pas de « getter ». Nous ne pouvons donc pas vérifier qu'il est correctement initialisé.

Tester les transformateurs

Pour les tests transformateurs, il faut, pour chaque attribut, instancier l'objet avec le constructeur et utiliser les séquences pour une couverture complète.

Attribut n

Nous avons suivi les séquences de l'attribut *n* présentées ci-haut. Voici les tests :

d4 = < { Queue() -> enqueue(1) -> dequeue() -> size() } , { size() == 0 } >

d5 = < { Queue() -> dequeue() -> enqueue(1) -> size() } , { Error Queue underflow } >

d6 = < { Queue() -> dequeue() -> size() -> enqueue(1) } , { Error Queue underflow } >

d7 = < { Queue() -> enqueue(1) -> size() -> dequeue() } , { size() == 1 } >

d8 = < { Queue() -> size() -> enqueue(1) -> dequeue() } , { size() == 0 } >

d9 = < { Queue() -> size() -> dequeue() -> enqueue(1) } , { Error Queue underflow } >

Attribut first

Nous avons suivi les séquences de l'attribut *first* présentées ci-haut. Dans les séquences sélectionnées, il faut qu'il y ait un enqueue(1) avant un dequeue() ou un peek(). Dans tous les cas où c'est le contraire, il y aura une erreur. Alors pour simplifier les choses, seul un test contentant un dequeue avant un enqueue sera présent, ainsi qu'un seul test contenant un peek avant un enqueue. De cette façon, nous économisons quelques tests qui donnent des erreurs. On obtient quand même une couverture complète du code.

d10 = < { Queue() -> isEmpty() -> peek() -> enqueue(1) -> dequeue()) } , { Error Queue underflow } >

d11 = < { Queue() -> isEmpty() -> enqueue(1) -> peek() -> dequeue() } , { isEmpty = true, peek() == 1 } >

d12 = < { Queue() -> isEmpty() -> dequeue() -> enqueue(1) -> peek() } , { Error Queue underflow } >

d13 = < { Queue() -> enqueue(1) -> isEmpty() -> peek() -> dequeue() } , { isEmpty() == false, peek() == 1 } >

d14 = < { Queue() -> enqueue(1) -> peek() -> isEmpty() -> dequeue() } , { isEmpty() == false, peek() == 1 } >

d15 = < { Queue() -> enqueue(1) -> peek() -> dequeue() -> isEmpty() } , { isEmpty() == true, peek() == 1 } >

Attribut last

L'attribut *last* ne contient aucun rapporteur, il est donc impossible de vérifier cet attribut.

Si nous avions eu un rapporteur, nous aurions pu prendre les mêmes tests que pour l'attribut *first* en y remplaçant la méthode *peek()* par le rapporteur de *last*.

Tester les autres

Les tests ci-haut couvre presque l'entièreté du code. Il ne manque que deux situations particulières pour s'assurer d'une couverture complète : lorsque nous faisons au moins deux *enqueue(int val)* d'affilés et lorsque nous faisons deux *dequeue()* d'affilés. Voici ainsi un test nous permettant de couvrir cette situation.

d16 = < { Queue() -> enqueue(1) -> enqueue(2) -> size() } , { size() == 2 } >

Avec l'ajout de ce test, nous couvrons à 100% le code de la classe *Queue*.