Domaine Sciences et Technologies

Licence informatique : parcours MI

Aix*Marseille université
Socialement engagée

Année 2023-24

Approfondissements en informatique (COO) : Devoir 1

Code UE : SIN6U28T

Vector et Deque

Consignes pour le rendu

TP noté

Ce devoir est à faire pour le 15 avril 2024. Il faudra que le dépôt *git* soit à jour pour le 15 avril 2024 à 23h59 et que je (Arnaud Labourel, identifiant `@alaboure`) sois membre du projet avec des droits au moins égal à `reporter`.

Consignes pour démarrer le TP

Vous allez utiliser git pour la gestion de versions. Il vous faut donc vous reporter aux consignes du premier exercice. Vous allez devoir forker le dépôt de base afin d'avoir votre propre dépôt. Le nom de votre dépôt devra être `vector-deque-` suivi de votre nom de famille. Par exemple, pour moi, le nom du dépôt devrait être `vector-deque-labourel`.

Lien vers le projet gitlab à forker pour le TP : https://etulab.univ-amu.fr/alaboure/vector-deque-template

Pour exécuter les tests, il faut passer par l'onglet *gradle* à droite et cliquer deux fois sur `vector-deque` -> `Tasks` -> `verification` -> `test`. N'oubliez pas de supprimer les tags `@Disabled` devant tous les tests afin de les exécuter.

Respect de la propriété intellectuelle

Comme pour tout devoir, nous vous demandons de ne pas partager votre programme, complet ou partiel, avec des membres d'autres étudiants. Le non-respect de cette consigne vous expose à recevoir une **note nulle**. Tout emprunt que vous effectuez doit être proprement documenté en indiquant quelle partie de votre programme est concerné et de quelle source elle provient (nom d'un autre étudiant, site internet, ...).

Critères d'évaluation

Vous serez évalué sur :

— **La propreté du code :** comme indiqué dans le cours, il est important de programmer proprement. Des répétitions de code trop visibles, des noms mal choisis ou des fonctions ayant beaucoup de lignes de code (plus de dix) vous pénaliseront. Le sujet vous donne les méthodes que vous devez absolument écrire mais il est tout à fait autoriser d'écrire des méthodes supplémentaires, de créer des constantes, ... pour augmenter la lisibilité du code. On rappelle que vous devez écrire le code en anglais.

— **La correction du code :** on s'attend à ce que votre code soit correct, c'est-à-dire respecte les spécifications dans le sujet. Comme indiqué dans le sujet, vous devez tester votre code pour vérifier son comportement.

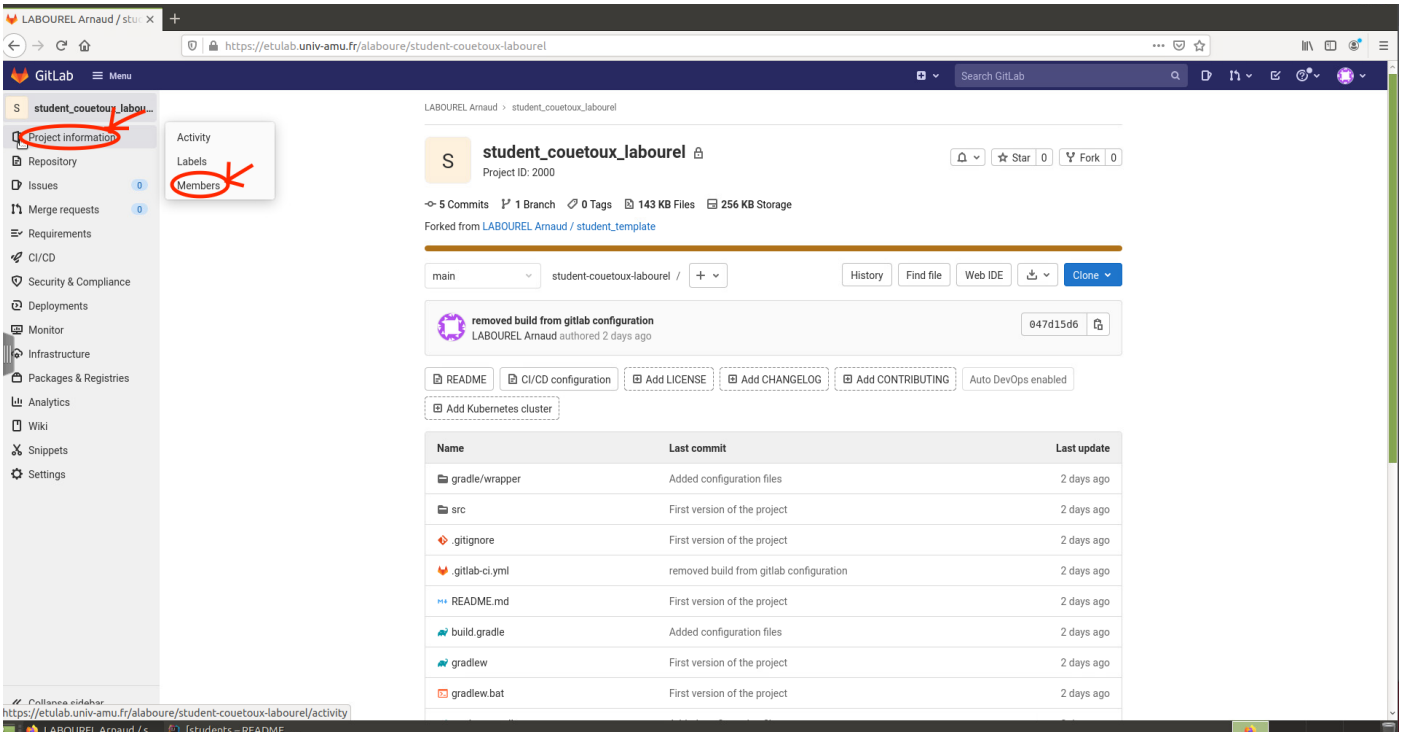
— **Les commit/push effectués :** il vous faudra travailler en continu avec `git` et faire des *push/commit* le plus régulièrement possible. Un projet ayant très peu de *push/commit* effectués juste avant la date limite sera considéré comme suspicieux et noté en conséquence. Un minimum accepté pour ce devoir sera d'au moins **2 pushs sur deux jours différents** et d'au moins **10 commits** au total. Si ces minimums ne sont pas respectés, le devoir recevra une note de zéro. Vous devez faire un commit par méthode que vous codez et un push après chaque tâche.

Première modification de votre dépôt

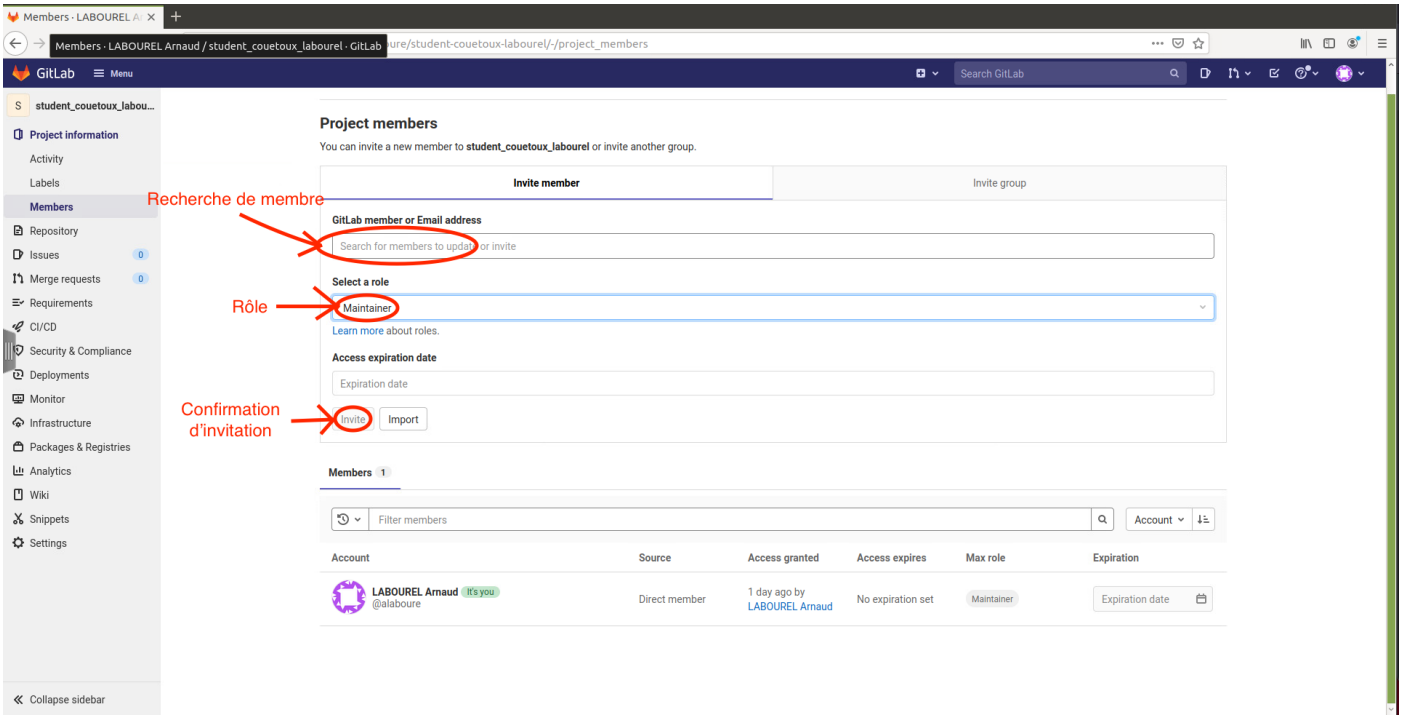
Modifier le fichier `README.md` à la racine du projet. Ce fichier devra contenir votre nom de famille et votre prénom.

Ajout chargé de cours en tant que membre

Une fois le projet créé, vous devez me (Arnaud Labourel, identifiant `@alaboure`) rajouter en cliquant sur `project information` dans le menu de gauche puis `members`.



Ensuite vous pouvez me rechercher (Arnaud Labourel, identifiant `@alaboure`) dans la barre dédiée. Une fois que vous m'avez trouvé, vous devez me donner le rôle de `reporter` puis confirmer son invitation en cliquant sur le bouton `invite`.



Vecteur d'objets

Le but de cet exercice sera de compléter la classe `ArrayVector` du *package* `vector` du dépôt et sa classe de test `TestArrayVector`.

La classe `ArrayVector<T>` permet de gérer un tableau d'éléments de type `T` dont la capacité augmente automatiquement si celui-ci est plein. Cette classe contient deux attributs : un tableau d'éléments nommé `elements` et un entier `size`. La longueur du tableau `array` peut être supérieure à `size`. Néanmoins, les éléments réellement présents dans le vecteur sont stockés dans les `size` premières cases du tableau `array`. Le constructeur prend en paramètre la longueur initiale du tableau `array`, c'est-à-dire la capacité initiale du vecteur. Si la longueur du tableau `array` ne permet plus de conserver tous les éléments du vecteur, elle est automatiquement augmentée à l'aide de la méthode `ensureCapacity`. La classe fournit les méthodes suivantes :

- `void ensureCapacity(int capacity)` fait en sorte que le tableau `elements` puisse contenir `capacity` éléments. Si la capacité actuelle du vecteur est inférieure à `capacity`, la capacité est augmentée. La nouvelle capacité doit être égale à max(capacity, 2 × capacité actuelle). Les nouvelles cases du tableau `elements` sont initialisées à `null`. Le nombre d'éléments (c'est-à-dire la valeur de `size`) n'est pas modifié.
- `void resize(int size)` modifie la taille du vecteur. Si la capacité est inférieure à `size`, elle est augmentée. Dans tous les cas les nouvelles cases sont initialisées à `null`.
- `int size()` retourne la taille actuelle du vecteur, c'est-à-dire son nombre d'éléments.
- `boolean isEmpty()` retourne `true` si le vecteur est vide, `false` sinon.
- `void add(T element)` ajoute l'élément `element` à la fin du vecteur.
- `void set(int index, T element)` affecte l'élément `element` à la position `index` dans le tableau. Si le tableau contient moins de `index + 1` éléments, la méthode ne fait rien.

— `T get(int index)` retourne l'élément à la position `index` dans le vecteur. Si le vecteur contient moins de `index + 1` éléments, la méthode retourne `null`.

Tâche 1 : compléter la classe `ArrayVector` afin que toutes les méthodes décrites ci-dessus fonctionnent correctement.

Tâche 2 : Tester (en complétant le code des méthodes existantes et en rajoutant des méthodes de tests) toutes les méthodes décrites ci-dessus dans la classe `TestArrayVector`. N'oubliez de réactiver les tests désactivés en remplaçant les `@Disabled` par `@Test`.

Implémentations de l'interface `Deque`

Le but de cet exercice sera de coder des files à double extrémité ou *deque* (abréviation de l'anglais *double-ended queue*). Une file à double extrémité est un type abstrait permettant d'ajouter et de supprimer des données à la fin (queue) ou au début (tête), réunissant ainsi les avantages des files et des piles. En Java, ce type est défini par l'interface `Deque`. Pour cet exercice, vous allez coder et tester une classe implémentant l'interface `DoubleEndedQueue` qui définit les 14 méthodes suivantes :

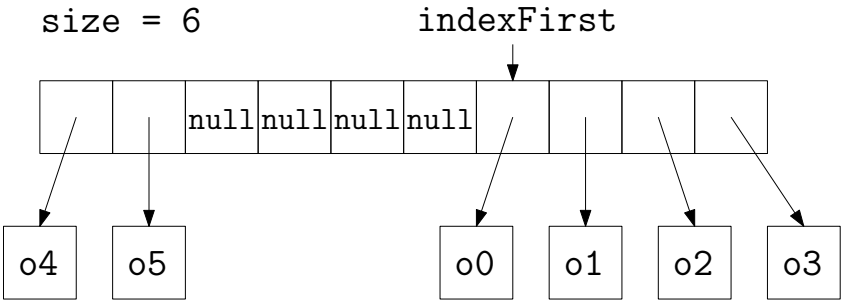
— `void addFirst(E e)` qui ajoute un élément en début de file, levant une exception de type `IllegalStateException` si la file a déjà atteint sa capacité maximale ;
— `void addLast(E e)` qui ajoute un élément en fin de file, levant une exception de type `IllegalStateException` si la file a déjà atteint sa capacité maximale ;
— `E removeFirst()` qui renvoie et supprime l'élément en début de file, levant une exception de type `NoSuchElementException` si la file n'a pas d'élément ;
— `E removeLast()` qui renvoie et supprime l'élément en fin de file, levant une exception de type `NoSuchElementException` si la file n'a pas d'élément ;
— `E getFirst()` qui renvoie l'élément en début de file, levant une exception de type `NoSuchElementException` si la file n'a pas d'élément ;
— `E getLast()` qui renvoie l'élément en fin de file, levant une exception de type `NoSuchElementException` si la file n'a pas d'élément ;
— `boolean offerFirst(E e)` qui ajoute si possible un élément en début de file et retournant `false` si l'ajout s'est fait (file n'ayant pas encore atteint sa capacité maximale) et `false` sinon ;
— `boolean offerLast(E e)` qui ajoute si possible un élément en début de file et retournant `false` si l'ajout s'est fait (file n'ayant pas encore atteint sa capacité maximale) et `false` sinon ;
— `E pollFirst()` qui renvoie et supprime l'élément en début de file (renvoie `null` si la file est vide) ;
— `E pollLast()` qui renvoie et supprime l'élément en début de file (renvoie `null` si la file est vide) ;
— `E peekFirst()` qui renvoie l'élément en début de file (renvoie `null` si la file est vide) ;
— `E getLast()` qui renvoie l'élément en fin de file (renvoie `null` si la file est vide) ;
— `int size()` qui renvoie le nombre d'éléments de la file ;

— `boolean contains(Object o)` qui renvoie `true` si la file contient un élément égal (par `equals`) à `o` et `false` sinon, la méthode levant une `NullPointerException` si l'objet `o` est `null`.

En résumé, en plus des deux méthodes `size` et `contains`, l'interface définit des méthodes pour accéder aux éléments aux deux extrémités de la file. Des méthodes sont fournies pour insérer, supprimer et examiner ces éléments. Chacune de ces méthodes existe sous deux formes : l'une lève une exception si l'opération échoue, l'autre renvoie une valeur spéciale (`null` ou `false`, selon l'opération). On peut résumer cela dans le tableau suivant :

Opération	Premier élément (exception)	Premier élément (`null`/`false`)	Dernier élément (exception)	Dernier élément (`null`/`false`)
Insérer	`addFirst(e)`	`offerFirst(e)`	`addLast(e)`	`offerLast(e)`
Supprimer	`removeFirst()`	`pollFirst()`	`removeLast()`	`pollLast()`
Examiner	`getFirst()`	`peekFirst()`	`getLast()`	`peekLast()`

Pour implémenter l'interface `DoubleEndedQueue` vous allez définir une classe `ArrayDoubleEndedQueue`. Cette classe utilisera un tableau d'`Object` pour stocker les éléments de la file (il faudra donc faire un *cast* pour obtenir le bon type d'objet). La taille du tableau sera égale à la capacité (*capacity*) de la file (déterminée à la construction de la file avec le constructeur `ArrayDoubleEndedQueue(int capacity)`) et ne changera pas par la suite. La file se comptera comme un buffer circulaire avec comme attribut l'indice du premier élément et le nombre d'éléments de la file. Les éléments de la file seront donc compris entre `indexFirst` inclus et `(indexFirst + size)% capacity` exclus (voir figure ci-dessous). Toutes les cases ne correspondant pas à des éléments de la file devront être à `null`. Des exceptions devront être levées dans certains cas conformément à la Javadoc de `DoubleEndedQueue`.



Tâche 3 : Créez une classe `ArrayDoubleEndedQueue` implémentant l'interface `DoubleEndedQueue`. Pour le moment, contentez-vous de définir des méthodes par défaut (ne retournant `null`, `0` ou `false` et sans autre code). Vous pouvez réaliser cela rapidement sous IntelliJ en faisant clic droit puis *generate* puis *implements method* une fois que vous avez défini que la classe implémente l'interface.

Tâche 4 : Créez et codez la classe `TestArrayDoubleEndedQueue` testant que la classe `ArrayDoubleEndedQueue` respecte les contraintes décrites dans la documentation de `DoubleEndedQueue`. Pour le moment, ne modifiez pas la classe `ArrayDoubleEndedQueue`.

Tâche 5 : Complétez la classe `ArrayDoubleEndedQueue` afin qu'elle passe les tests de `TestArrayDoubleEndedQueue`.