TP2

Package cible : tp2.exo1 - tp2.exo2 - tp2.exo3

Exercice 1 : Manipuler des chaînes de caractères

L'objectif de cet exercice est d'appliquer quelques algorithmes sur la manipulation de chaînes de caractères. Pour chacune des classes, vous trouverez des tests unitaires pour vérifier la validité de ce que vous faites, dans le dossier "test" du projet Eclipse. Les méthodes à remplir se trouvent dans les classes Casse, Palindrome et StringReverse, dans le dossier "src".

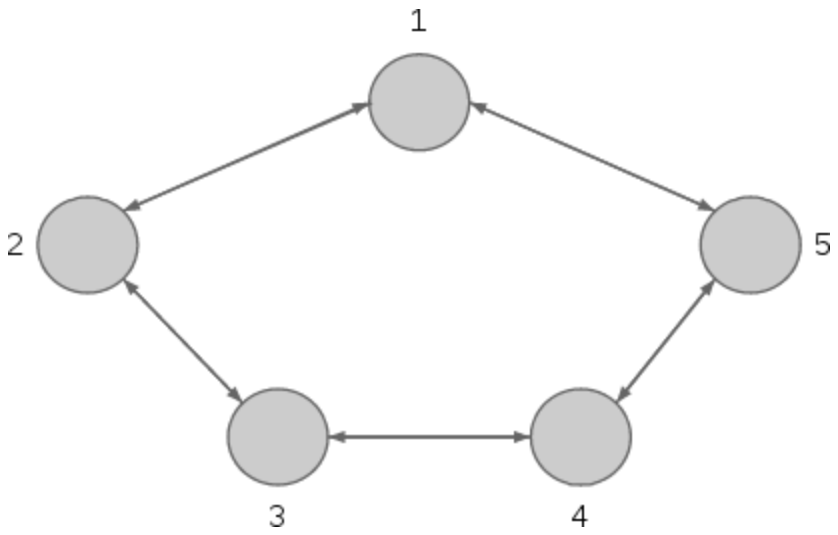
- Casse : Remplir les méthodes "countUpperCase" et "countLowerCase" qui renvoient respectivement le nombre de majuscules et de minuscules depuis le tableau de caractères donné en paramètres. Pour déterminer si un caractère est une majuscule ou une minuscule, vous pouvez utiliser les méthodes static de la classe Character dont la documentation se trouve ici et ici.

- Palindrome : Un palindrome est un mot qui, s'il est lu de gauche à droite ou de droite à gauche est toujours le même. Par exemple : "kayak". Remplir la méthode "isPalindrome" qui renvoie vrai si le tableau de caractères donné en paramètre est un palindrome.

- StringReverse : Remplir la méthode "reverse" qui inverse le tableau de caractères donné en paramètre. Exemple : Si l'on a passé "salut", cette méthode donnera "tulas". **Attention, cette méthode ne renvoie rien car elle modifie directement le tableau. Vous ne devez pas créer de nouveau tableau, mais modifier directement le paramètre.**

Exercice 2 : Ring

L'objectif de cet exercice est de créer une classe permettant de représenter un anneau. Chaque élément de l'anneau possède un suivant et un précédent de tel sorte que l'ensemble des noeuds de l'anneau se joignent, comme dans l'image suivante :



Une autre façon de percevoir un anneau est de dire qu'il s'agit simplement d'une liste doublement chaînée circulaire, soit que chaque élément pointe vers son suivant et son précédent et que la tête et la queue pointent aussi, comme n'importe quel autre élément, sur leurs suivants et précédents.

Dans la suite de l'exercice, les appellations noeuds et éléments sont identiques.

1. Créer la classe Ring, qui est la représentation du concept de l'anneau.

2. Ajoutez 3 variables d'instances représentant un identifiant (qui s'avère être un entier, et représenté sur l'image précédente par les nombres) et deux références vers le précédent et le suivant. Cela implique que ces deux variables sont du type de la classe. En effet, l'élément 3 a un suivant, l'élément 4 et un précédent l'élément 2, donc des objets de la même classe que l'élément.

3. Ajoutez une variable static servant de compteur dont **la valeur initiale est 0**. Cette variable servira à générer des identifiants.

4. Créer un constructeur vide, qui ne fait qu'attribuer un identifiant au nouvel objet. Créer un constructeur prenant en paramètre 2 objets de type Ring, représentant le précédent et le suivant de l'anneau que l'on veut créer. Affectez aussi un nouvel identifiant.

Attention, lors de l'affectation d'un identifiant, il ne doit y avoir aucun élément dont l'identifiant est 0.

5. Ajoutez les getters / setters sur tous les attributs **SAUF** l'identifiant, qui n'aura pas de setter.

6. Maintenant que la classe est globalement créée, l'objectif va être de passer des messages dans l'anneau. Un noeud peut envoyer un message à son successeur / prédécesseur, qui l'envoie dans le même sens, jusqu'à ce que l'initiateur reçoive son propre message et arrête de le faire circuler.

Créez la méthode "*public void receiveMessage(Ring sender, String message, boolean toNext)*" sans lui affecter de corps. Cette méthode sera remplie plus tard dans le sujet.

7. Créez une méthode "*private void send(Ring sender, String message, boolean toNext)*" qui a pour but

d'envoyer un message. Si le booléen est mis à true, cette méthode envoie le message à son suivant, sinon à son prédécesseur. Le premier argument correspond au noeud qui a envoyé le message, le noeud initiateur. Donc si le noeud "1" envoie un message, le noeud "4" verra que l'émetteur est "1".
Pour se faire, envoyer un message correspond à le faire recevoir par le voisin.

8. Créez la méthode "*public void sendMessage(String message, boolean toNext)*" qui initie un message. Le noeud appelant cette méthode est donc initiateur. Pensez à utiliser le code déjà utilisé !

9. Remplir le code de la méthode "*public void receiveMessage(Ring sender, String message, boolean toNext)*". Si le noeud courant est l'initiateur du message, cette méthode arrête la circulation du message. Sinon, elle affiche le message et l'identifiant du noeud courant et fait passer le message à son voisin.

Exercice 3 : Gestion de processus

Un peu de théorie...

Cet exercice a pour objectif de créer un ordonnanceur de processus. Page wikipedia sur l'ordonnancement (*scheduling* en anglais) : http://en.wikipedia.org/wiki/Scheduling_(computing)

Les systèmes d'exploitation comme Linux et Windows sont des systèmes multi-tâches, c'est à dire que vous pouvez utiliser plusieurs processus en même temps (Mozilla Firefox et Word par exemple). Ces tâches sont gérées par le système grâce à un ordonnanceur qui choisit quelles tâches peut s'exécuter à quel moment. Il existe un nombre très importants de stratégies d'ordonnancement qui ne seront pas détaillées ici, mais nous utiliserons la plus simple : FIFO.

Notre système se basera sur le système Linux.

1. Un processus est identifié par un numéro : le Process IDentifier (ou PID).
2. Tout processus possède un processus père, qui est son créateur. Lorsque vous lancez votre navigateur depuis le Bureau, le processus navigateur est fils du processus qui gère le Bureau. Tout processus possède l'identifiant de son père appelé : Parent Process IDentifier (ou ppid).
3. Le père de tout les processus s'appelent init et possède un PID de 1 et un PPID de 0.
4. Mis à part init, tous les processus ont un père. Si un processus père se termine avant un processus fils, alors le père adoptif est toujours init.

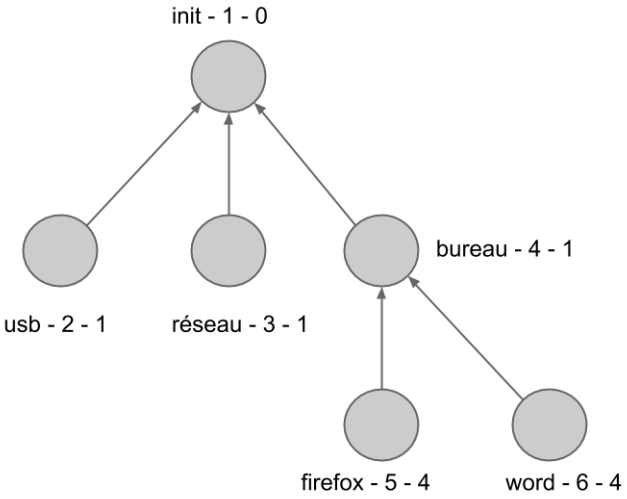
Un processus correspond à l'exécution d'un programme. Lorsque vous lancez 3 fois le programme correspondant à votre navigateur, vous avez créé 3 processus du même programme.
L'image ci-dessous montre les informations suivantes :

1. Deux processus s'exécutent nommés "ps -f" et "/bin/bash". Le premier correspond à la commande tapée dans un terminal Linux permettant de visualiser les processus en cours, le second est le processus associé à l'exécution du terminal dans lequel je peux écrire des commandes.
2. PID : Le pid du processus. 2881 et 3165 respectivement
3. PPID : Le pid du père. Ici on remarque que le processus "ps -f" a un PPID identique au PID de "/bin/bash". Cela s'explique par le fait que j'ai exécuté une commande dans un terminal (/bin/bash), donc que ce processus terminal est le père du processus de la commande.

```
pitton@pitton ~ $ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
pitton    2881  2879  0 16:38 pts/1    00:00:00 /bin/bash
pitton    3165  2881  0 16:52 pts/1    00:00:00 ps -f
pitton@pitton ~ $
```

Retenez donc que : Un processus a un identifiant (PID) et l'identifiant de son père (PPID).

Un exemple plus concret montrant un arbre de processus. Les textes décrivent respectivement "nom du processus" - "pid" - "ppid". On voit donc que 3 processus ont pour père "init", et deux processus ont pour père "bureau". Tous les processus ont des PID distincts, mais les PPID et de "firefox" et "word" est le PID de "bureau".



Enfin, tous les processus ont une durée de vie (dans notre programme en tout cas, la réalité est autre...). Pour cela, chaque processus possède un Time To Live, correspondant à sa durée de vie. Dès lors que l'ordonnanceur choisit un processus, le TTL de celui-ci est décrémenté. Si son TTL atteint 0, le processus est mort et il est supprimé de l'ordonnanceur, sinon il est remis à la fin de la file d'attente.

En pratique...

1) La classe Processus

Reprenez la classe Processus dans le dossier "src".

- Créer les accesseurs et modifieurs de toutes les variables d'instance.
- Ecrire la méthode "*boolean canDie()*" qui renvoie si le processus courant peut mourir. Seul le processus "init" ne peut pas mourir, tous les autres le peuvent.
- Ecrire la méthode "*boolean isDead()*" qui renvoie vrai si le processus est mort. Un processus est mort s'il

peut mourir (donc que ce n'est pas init) et si son TTL est égal à 0.
- Ecrire la méthode "*Processus fork()*" qui crée un processus fils du processus courant. Cette méthode crée un nouveau Processus, lui affecte un nouveau PID grâce au compteur (la variable static PID), place le PPID du nouveau processus, et met le TTL du nouveau processus à la valeur de son PID. Donc si le processus courant a un pid de 5, le nouveau processus a un pid de 6, un ppid de 5 et un TTL de 6.
- La méthode "*void exec()*" correspond à l'exécution du processus courant. Dès lors que l'ordonnanceur choisit une tâche, il exécute cette méthode. Modifiez le code de cette méthode pour qu'elle décrémente le TTL du processus courant.

2) Liste doublement chaînée circulaire

Une liste doublement chaînée circulaire est une liste chaînée où chaque élément de la liste connaît son prédecesseur et son successeur. De plus, la queue a pour suivant la tête et la tête a pour précédent la queue.

Documentation de ce type de structure ici.

Cette liste contiendra l'ensemble de nos processus et notre futur ordonnanceur s'appuiera sur cette liste pour récupérer les processus à exécuter.

2.1°) La classe Element

Créer une classe Element.

- Cette classe possède 3 variables d'instance : Un processus, la valeur associée à l'élément de la liste, un Element suivant et un Element precedent.
- Créer les accesseurs et modifieurs de toutes les variables d'instance.

2.2°) La classe LinkedList

Créer une classe LinkedList.

- Cette classe possède une unique variable d'instance de type Element, la tête. Cette variable représente la tête de la liste.
- Créer une méthode "*boolean isEmpty()*" qui renvoie vrai si la liste est vide. Si la liste ne contient pas de tête, alors elle est vide.

Attention lorsque vous itérez dans la liste, puisque vous ne possédez pas la taille de la liste, vous devez utiliser un while. Vous devrez donc vous arrêter dès que vous arrivez à la fin de la liste (la queue). Pensez aussi à tester / utiliser sa valeur. Exemple si je veux calculer la taille de la liste :

```
Element p = head;
int size = 0;
while (p != head.getPrevious()) {
  size++;
  p = p.getNext();
}
// On est sorti du while car p == queue. Donc on n'a pas incrémenté pour compter la queue
// On incrémente maintenant.
```

```
size++;
return size;
```

- Créer une méthode "*int size()*" qui renvoie la taille de la liste.
- Créer une méthode "*void addLast (Processus p)*" qui ajoute le processus spécifié à la fin de la liste.
- Créer une méthode "*void addFirst(Processus p)*" qui ajoute le processus spécifié au début de la liste.
- Créer une méthode "*Processus removeFirst()*" qui supprime la tête de la liste et renvoie le processus associé.
- Créer une méthode "*Processus removeLast()*" qui supprime la queue de la liste et renvoie le processus associé.
- Créer une méthode "*Processus gettFirst()*" qui renvoie le processus associé à la tête de la liste.
- Créer une méthode "*Processus getLast()*" qui renvoie le processus associé à la queue de la liste.
- Créer une méthode "*Processus search(long pid)*" qui renvoie le processus dont le pid est spécifié.
- Créer une méthode "*Processus get(int pos)*" qui renvoie le processus situé à la position spécifiée de la liste.

Optionnel : La méthode suivante n'est pas obligatoire et ne sera pas utilisée.

- Créer une méthode "*boolean remove(Processus p)*" qui supprime le processus spécifié de la liste.

3) La classe Scheduler

Créer une classe Scheduler correspondant à notre ordonnanceur.

- Ajoutez une variable d'instance de type LinkedList, la classe créée précédemment.
- Créer un constructeur. Ce constructeur initialise la variable d'instance et crée le premier processus init. C'est le seul processus créé sans passer par la méthode "fork" créée précédemment. Ce processus a donc un pid de 1 et un ppid de 0 et est ajouté à la liste.
- Créer une méthode "*void addProcessus(Processus p)*" qui ajoute le processus spécifié à la fin de la liste.
- Créer une méthode "*Processus addProcessus()*" qui crée un nouveau processus dont le père est init. C'est à dire qu'il faut d'abord récupérer init, et créer un nouveau processus fils à ce-dernier. Le nouveau processus est ajouté dans la liste et renvoyer.
- Créer une méthode "*void executeInit()*" qui va simuler l'exécution du processus init. L'algorithme est le suivant : On parcourt tous les processus et pour chacun d'entre eux, on cherche si le père est présent dans la liste. S'il n'est pas présent dans la liste, alors le processus est orphelin, et on passe son ppid à 1, signifiant que son père est init.
PS : Pensez à ajouter un message dans la console lors de l'exécution de cette méthode et si un processus possède un nouveau père.
- Créer une méthode "*boolean executeNext()*" qui simule l'exécution d'un processus. L'algorithme est le suivant : On supprime le premier processus de la liste. Si la liste est vide, on renvoie faux. Si le processus ne peut pas mourir (comme init), on le remet à la fin. Si le processus n'est pas init, on l'exécute. Après l'exécution, si le processus est mort, on lance init (car le processus mort peut être père d'autres processus), sinon on le remet à la fin de la liste car le processus n'a pas fini de s'exécuter. On renvoie vrai, il nous reste quelque chose à exécuter.
- Créer une méthode "*void start()*" qui lance l'ordonnanceur. Cette méthode exécute l'ensemble des processus au fur et à mesure (grâce à executeNext()) jusqu'à ce qu'il ne reste plus que le processus init.

Vous pouvez vous inspirer du main fourni pour tester votre programme dès lors que vous l'avez terminé.