TP noté : Test « boite noire » de classes Java

On considère la spécification informelle ci-dessous :

La classe `Système` gère une (unique) ressource, par exemple un processeur, qu'il doit partager entre des processus. Un processus ne termine jamais et ne libère jamais spontanément la ressource mais uniquement sur la requête du système via l'opération `swap` qui permet d'attribuer la ressource à l'un des processus en attente. Le système n'est pas forcément équitable et se contente simplement, quand il a le choix, de ne pas redonner la ressource au processus qui le possédait déjà lors de l'échange précédent. Il dispose pour cela des attributs

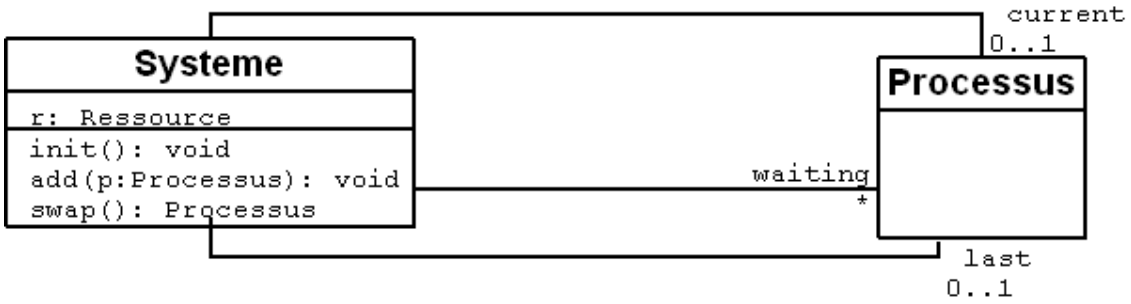
- `current`*, l'éventuel processus qui possède actuellement la ressource,*
- `last`*, l'éventuel processus qui possédait la ressource juste avant* `current`
- `waiting`*, l'ensemble des processus demandeurs (qui inclut* `last`*).*

On peut associer au diagramme les invariants suivants :

- si `current` est `null`, alors `last` est `null` et `waiting` est vide.
- `current` n'appartient jamais à `waiting`
- *Si* `last` *est non* `null`*, il fait partie de la collection* `waiting` *et est différent de* `current`.

Un processus est caractérisé par son nom. On peut créer un processus en fournissant un nom en paramètre, avoir accès à son nom, tester l'égalité entre deux processus. Deux instances de Processus de même nom désignent le même processus.

Ceci peut être résumé par le diagramme UML suivant :



L'opération `init()` met le système dans un état initial où il n'existe aucun processus.

L'opération `add(p:Processus)` ajoute un processus au système. L'opération est indéfinie si `p` vaut `null`. Le processus `p` ne doit pas avoir été déjà ajouté au système. S'il n'existe aucun processus, `p` devient le processus courant qui détient la ressource, sinon il est ajouté à `waiting`

L'opération `swap()` change le processus actif qui détient la ressource. Elle est décrite informellement par :

- `swap` lève l'exception `ErreurSystème` (sous-classe de `Exception`) s'il existe n'existe aucun processus en attente de la ressource et laisse l'état inchangé.
- Sinon : À la sortie de `swap` ce n'est plus le processus actuel qui dispose de la ressource, le possesseur à l'entrée a rejoint la liste des demandeurs et on se souvient que c'est lui qui détenait la ressource au moment de l'échange. Si on a le choix on attribue la ressource à un

processus autre que celui qui l'avait lors de l'échange précédent. Le système peut choisir n'importe quel processus demandeur parmi ceux qui respectent les contraintes précédentes.

La classe `Systeme` comprend en plus les observateurs suivants :

- `Processus current()` : renvoie le processus qui détient actuellement la ressource ou lève l'exception `ErreurSysteme` s'il n'existe aucun tel processus.
- `Collection<Processus> getWaiting()` : renvoie une collection des processus en attente de la ressource.
- `boolean isCurrent(Processus p)` : teste si `p` est le processus qui détient actuellement la ressource. Si on passe `null` en paramètre la méthode renvoie `true` *ssi* la ressource est actuellement libre.
- `boolean isLast(Processus p)` : similaire à la méthode précédente pour tester la valeur de `last`.

La classe `Processus` dispose des observateurs `String name()` et de `equals`.

Vous ne disposez pas d'autres moyens d'observer le comportement de ces classes.

1. Écrire un pilote de test JUNIT4 pour tester une future implémentation des classes du système. Pour chaque méthode de test, fournir sous forme de commentaire une brève explication de l'objectif de ce test.

2. Vous avez à disposition divers fichiers `.jar` dont chacun contient des réalisation des classes `Systeme`, `Processus` et `ErreurSysteme`. À l'aide de votre pilote de test, indiquez quelles sont les réalisations correctes et quelles sont celles qui sont incorrectes. Pour chaque réalisation incorrecte décrivez au moins un des tests qui permet de montrer le défaut repéré : contexte (instances mises en jeu, dans quel état, …) appel de méthode à réaliser (avec les arguments), observation qui permet de montrer que le résultat obtenu n'est pas celui attendu (on précisera donc le résultat qui était attendu).