Aucun document. Aucune machine. Le barème est indicatif.
Une question peut toujours être traitée en utilisant les précédentes (traitées ou non).
Les morceaux de code Java devront être clairement présentés, indentés et commentés.
L'utilisation de collections de l'API Java est interdite (entre autres `LinkedList` et `ArrayList`).

On veut modéliser le partage d'un processeur par plusieurs processus par une version très simplifiée de l'algorithme de *round robin*. Le principe de ce partage est le suivant : le processeur possède une liste de processus auxquels il accorde, à tour de rôle et dans l'ordre, un temps de calcul qu'on fixe à 2 millisecondes ; si le processus a terminé son travail dans le temps imparti, il disparaît de la liste des processus existants.

1 Processus

Un processus est une tâche en train de s'exécuter. Pour simplifier, on suppose qu'il correspond au lancement d'un exécutable (ex : **`cd`**, **`emacs`**, **`firefox`**, etc.). Il est identifié par un entier (son *identifiant*). La classe **`Processus`** contiendra donc au moins les attributs privés suivants : **`String nomExec`**, **`int id`** et **`int ms`** ; ce dernier attribut correspondant au temps d'exécution restant, en millisecondes (ce qui est un abus de modélisation puisque dans la réalité ce temps n'est pas connu).

On modélise un processus comme une instance de la classe **`Processus`** d'interface :

```
/** renvoie l'identifiant du processus
 * (identifiant unique: deux processus ne peuvent avoir le meme) */
int getId();
/** renvoie le nom de l'executable associe */
String getNomExec();
/** teste si le calcul est termine (<=> le temps d'execution restant est nul) */
boolean calculFini();
/** simule le calcul pendant millis millisecondes
 * (et met a jour le temps d'execution restant) */
void faisTonCalcul(int millis);
```

Cette classe possède en outre un constructeur. Celui-ci :
- prend en argument une chaîne de caractères (le nom de l'exécutable associé) et un booléen destiné à spécifier si le processus s'exécute en temps limité (ex : **`ls`**, **`date`**) ou en temps illimité jusqu'à ce que l'utilisateur l'interrompe (ex : **`emacs`**, **`firefox`**) —par convention **true** correspond au temps limité,
- attribue un temps d'exécution pour les processus s'exécutant en temps limité, en tirant aléatoirement un temps de calcul entier qu'on supposera compris entre 1 et 100 millisecondes,
- attribue au processus un identifiant unique de type entier (on suppose qu'on ne dépasse pas $2^{31} - 1$ processus, mais on ne vous demande pas de faire de test à ce sujet).

Exercice 1. (5 points) Écrire la classe **`Processus`**.

On pourra utiliser les fonctions suivantes :
- **`static void java.lang.Thread.sleep(int t);`** qui met en arrêt le programme pendant **`t`** millisecondes.
- **`static double java.lang.Math.random();`** qui renvoie un **double** aléatoire dans l'intervalle $[0, 1[$.

2 Processeur

Dans cette partie, on modélise le processeur et la liste des processus en cours par deux classes **`Processeur`** et **`CellProc`**.

Les processus en cours sont représentés par une liste doublement chaînée circulaire (voir Fig. 1). Ainsi à chaque processus correspond une cellule, instance de la classe **`CellProc`**. Une telle cellule contient, en plus de la référence vers le **`Processus`**, une référence vers la cellule précédente et une autre vers la cellule suivante. Une instance de la classe **`Processeur`** ne contient alors qu'une référence **`pC`** vers la cellule du processus courant.

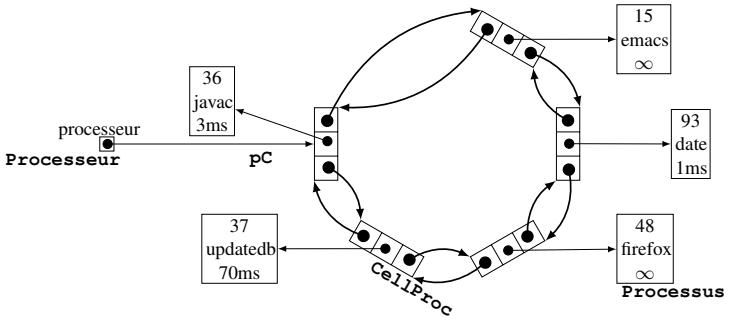


Fig.1

Les cellules contenant une référence à un processus et les références aux cellules précédentes et suivantes sont des instances de la classe **`CellProc`**.

La classe **`Processeur`** contient une référence vers la première cellule (ou cellule courante) : son attribut **`pC`** référence la cellule du processus courant, c'est-à-dire celui qui est en train ou qui est sur le point de bénéficier du processeur pour faire ses calculs.

Dans une liste circulaire doublement chaînée, la première cellule possède une référence vers la dernière cellule (c'est sa cellule précédente) et la dernière cellule possède une référence vers la première (c'est sa cellule suivante).

La classe **`Processeur`** possède l'interface suivante :

```
/** affiche la liste des processus (id+nom pour chaque processus) */
void ps();
/** ajoute un nouveau processus dans la liste
 * @param p processus a ajouter */
void addNewProcess(Processus p);
/** stoppe le processus d'identifiant id (le retire de la liste des processus)
 * @param id identifiant du processus a stopper
 * @return true si le processus existait, false sinon */
boolean kill(int id);
/** stoppe les processus dont le nom est donne en parametre
 * (les retire de la liste des processus)
 * @param nom d'executable associe aux processus a stopper
 * @return true s'il existait de tels processus, false sinon */
boolean killAll(String nom);
/** donne 2 millisecondes de temps de calcul au processus courant,
 * le supprime de la liste des processus s'il y a lieu et
 * met a jour le processus courant */
void execCurrent();
```

Exercice 2. (3 points) Donner les attributs et les constructeurs des classes **`CellProc`** et **`Processeur`**.

Exercice 3. (3 points) Écrire les méthodes **`ps`** et **`addNewProcess`**.

Dans la situation de la Fig. 1 :

un appel à **`ps`** provoque l'affichage :

```
PID   CMD
36    javac
15    emacs
93    date
48    firefox
37    updatedb
```

un appel à **`addNewProcess(new Processus("jgrasp",false))`** mène à :

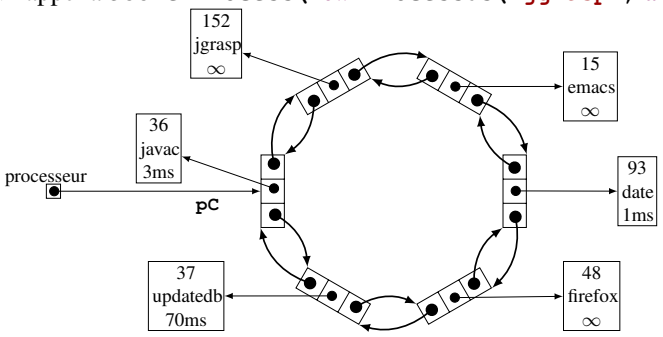
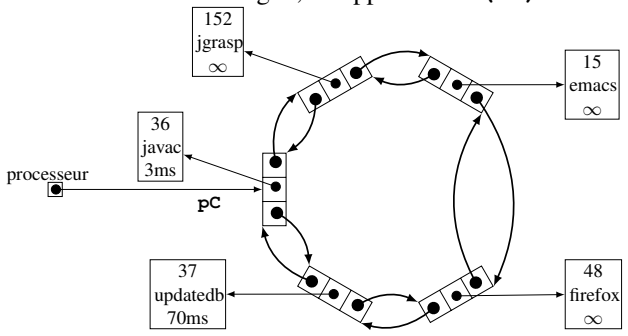


Fig. 2

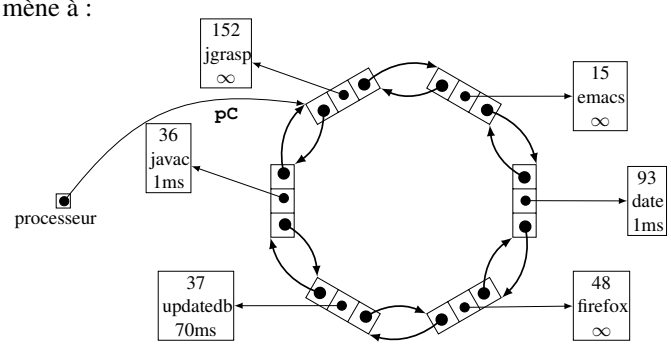
Exercice 4. (4 points) Écrire les méthodes **`kill`** et **`killAll`**.

Dans la situation de la Fig. 2, un appel à **`kill(93)`** mène à :



Exercice 5. (3 points) Écrire la méthode **`execCurrent`**.

Dans la situation de la Fig. 2, un appel à **`execCurrent()`** mène à :



3 Algorithme Round Robin

Exercice 6. (2 points) Dans une classe **`RoundRobin`**, écrire une fonction **`roundRobin`** prenant en paramètre une instance de la classe **`Processeur`** et qui fait tourner ce processeur tant qu'il y a des processus.