FSA11BA **Examen FSAB1401** 20 juin 2011	**Remplissez ce cadre en MAJUSCULES et LISIBLEMENT, svp** NOM: . . . . . . PRÉNOM: . . . . . . NOMA: ☐☐☐☐☐☐**0 0**	**Réservé**

INFORMATIQUE 1 (durée totale : 3h00 maximum)

Consignes (à lire attentivement avant de répondre aux questions)

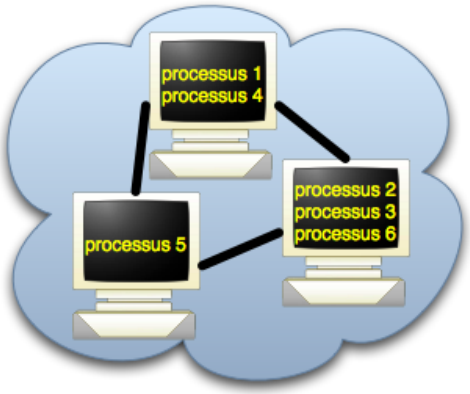
1. Veuillez déposer votre **carte d'étudiant** sur la table.
2. Ecrivez vos réponses lisiblement **au bic ou au stylo** (pas au crayon, svp !).
3. **Remplissez l'en-tête** ci-dessus et en haut de chaque feuille de réponse **avant de commencer votre examen**.
4. **Ne dégrafez pas** les feuilles du fascicule.
5. **Utilisez exclusivement les feuilles qui vous sont distribuées**. Des feuilles de brouillon supplémentaires peuvent être obtenues sur demande. **Remettez toutes les feuilles** utilisées à la fin de l'examen, y compris les feuilles de brouillon.
6. **Répondez exclusivement dans les cadres prévus à cet effet**. Si vous manquez de place, continuez au verso et indiquez clairement dans le cadre où se trouve la suite de votre réponse. Aucune autre réponse ne sera prise en compte. Utilisez les emplacements vides et le revers des feuilles comme brouillon.
7. Seul **un livre de référence sur Java** (à l'exclusion de toutes photocopies ou feuilles supplémentaires) peut être utilisé en plus du présent fascicule pendant cet examen.
8. Il n'est **pas** demandé (et donc pas nécessaire) d'appliquer les principes de la **programmation défensive**.
9. Dans vos réponses, vous pouvez utiliser tous les constructeurs et toutes les méthodes qui figurent dans les (fragments de) classes ou interfaces fournies, même si leur implémentation n'est pas donnée, pour autant que vous disposiez des informations nécessaires pour les utiliser correctement. Vous pouvez également utiliser les classes et méthodes de l'API standard Java.

Remarque: les points attribués à chaque question (sur 20) sont donnés à titre indicatif et pourront fluctuer légèrement lors de la pondération finale.

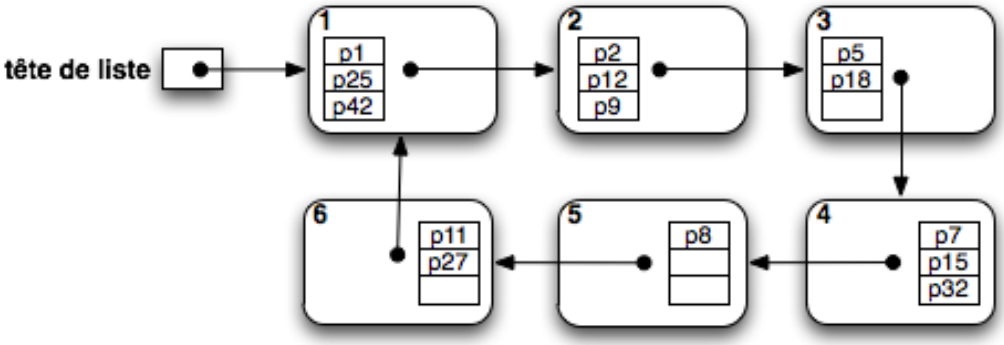
Bon travail !

Contexte

Une *grappe d'ordinateurs* (computer cluster) est un ensemble d'ordinateurs connectés entre eux et organisés de manière à apparaître comme une large ressource de calcul sur laquelle peuvent s'exécuter simultanément de nombreuses tâches, la répartition des tâches entre les différents ordinateurs étant gérée par le système de manière transparente pour l'utilisateur. On appelle *processus* (process) une tâche individuelle dont un utilisateur peut demander l'exécution sur la grappe.



Le programme que vous devez compléter permet de représenter la gestion des processus sur une grappe d'ordinateurs. La grappe est décrite par la classe **`Cluster`**. Une grappe contient la liste des ordinateurs de la grappe, organisée sous forme de liste chaînée circulaire : les noeuds de la liste sont connectés en boucle, de sorte que la liste n'a pas de début ni de fin. Cette organisation est utilisée pour ajouter les nouveaux processus à chaque ordinateur à tour de rôle, dans la mesure des ressources disponibles.



Le fonctionnement de la liste est documenté dans la classe **`Cluster`**. La tête de liste se décale à chaque ajout de processus pour assurer une répartition plus équitable. Par exemple, dans la situation illustrée ci-dessus, l'ordinateur 1 est en tête de liste. Si on désire ajouter un processus, et que ni l'ordinateur 1 ni le 2 ne disposent des ressources suffisantes, le processus sera ajouté à l'ordinateur 3, et l'ordinateur 4 deviendra la nouvelle tête de liste.

Une grappe peut contenir plusieurs types d'ordinateurs, dont les fonctionnalités communes sont décrites dans l'interface **`ComputerIF`**. Deux implémentations de cette interface sont fournies :

- **`BasicComputer`** qui supporte un seul processus à la fois, et
- **`FullComputer`** qui supporte un nombre maximum déterminé de processus, gérés dans un tableau.

Un processus est décrit par la classe **`Process`**, que vous devez écrire à la question 4. Le programme utilise aussi sa propre classe d'exceptions, définie dans **`UnavailableException`**.

Dans tout le programme, on suppose que chaque processus et chaque ordinateur est représenté par un et un seul objet, de sorte qu'il suffit de comparer les références pour vérifier l'égalité entre processus ou entre ordinateurs.

La suite de ce fascicule contient les questions auxquelles vous devez répondre, suivies des sources (incomplètes) du programme auquel se rapportent ces questions.

Question 1 (1[5] pts) Ecrivez le corps de la méthode **`removeProcess`** de la classe **`BasicComputer`.**

```
/**
 * @pre p!=null
 * @post le processus p a été retiré de cet ordinateur, s'il s'agit bien du
 *       processus présent. Retourne true si le processus a été supprimé,
 *       false sinon.
 */
public boolean removeProcess(Process p)
```

Q1

Question 2 (2 pts) Écrivez complètement le constructeur de la classe **`FullComputer`**.

```
/**
 * @pre n > 0, name != null, storage >= 0
 * @post Construit un FullComputer de nom name, supportant n processus
 *       et avec une capacité de stockage égale à storage
 */
```

Q2

Question 3 (4^5 pts) Ecrivez complètement (y compris les spécifications) la classe **`Process`** qui représente un processus. Chaque processus a un nom (**`String`**), une capacité de stockage requise (**`int`**) et un identifiant de processus ou PID (**`int`**). Les PIDs sont attribués séquentiellement à la création de chaque nouveau processus (1 pour le premier, 2 pour le deuxième, etc.) La classe doit définir :

- Un constructeur avec le nom et la capacité comme paramètres,
- Des méthodes **`getName`**, **`getRequiredStorage`** et **`getPid`** retournant respectivement le nom, la capacité et le PID,
- Une méthode **`getDescr`** retournant une chaîne de caractères comprenant le nom du processus et la capacité de stockage nécessaire, séparés par un espace,
- Toutes les variables et méthodes complémentaires nécessaires à votre implémentation.

Q3

Q3 (suite)

Question 4 (2 pts) Ecrivez le corps de la méthode **`addProcess`** de la classe **`FullComputer`**. Pensez à utiliser les méthodes de la classe **`Process`** définies à la question 3.

```
/**
 * @pre  p != null, p ne se trouve pas déjà sur cet ordinateur
 * @post le processus p a été ajouté à cet ordinateur, si (1) le nombre de
 *       processus maximal n'est pas atteint et (2) la capacité de stockage
 *       nécessaire pour p est disponible. Retourne true si le processus a
 *       été ajouté, false sinon.
 */
public boolean addProcess(Process p)
```

Q4

Question 5 (2^5 pts) Ecrivez complètement la méthode **`addProcess`** de la classe **`Cluster`**. Aidez-vous de l'exemple présenté en page 2. Indice: vous pouvez utiliser la valeur de **`count`**.

```
/**
 * @pre p != null, p ne se trouve pas déjà sur un ordinateur du cluster.
 * @post Le processus p a été ajouté au premier ordinateur, à partir de la
 *       tête de la liste, disposant des ressources nécessaires. La nouvelle
 *       tête de liste est le noeud qui suit celui de l'ordinateur où p a
 *       été ajouté. Si aucun ordinateur ne dispose de ressources
 *       suffisantes, la tête de liste est inchangée et une
 *       UnavailableException est lancée.
 */
```

Q5

Question 6 (3 pts) Ecrivez complètement la méthode **`removeComputer`** de la classe **`Cluster`**. Indice: vous pouvez utiliser la valeur de **`count`**.

```
/**
 * @pre comp != null
 * @post L'ordinateur comp a été retiré du cluster, s'il s'y trouvait. Si
 *       comp est en tête de liste, la tête de liste passe au noeud suivant,
 *       sinon elle est inchangée. Retourne true si comp a été retiré, false
 *       sinon.
 */
```

Q6

Question 7 (4 pts) Ecrivez le corps de la méthode **`loadState`** de la classe **`Cluster`**. Notez que la pré-condition garantit le format du contenu du fichier; pas besoin donc de traiter les erreurs de format du texte.

```
/**
 * @pre filename le nom d'un fichier qui respecte le format produit par saveState
 * @post Retire tous les processus présents dans le cluster, puis ajoute au
 *       cluster les processus dont les noms et capacités sont donnés dans
 *       le fichier, selon le format généré par saveState.  Arrête le
 *       programme si une erreur d'I/O se produit ou si la capacité du
 *       cluster est insuffisante.
 */
public void loadState(String fileName)
```

Q7

Annexes:

Fichier ComputerIF.java

```java
/**
 * Un ordinateur, sur lequel on peut ajouter et retirer des processus.
 * L'ordinateur dispose de ressources (nombre de processus, volume de stockage)
 * éventuellement limitées, et donc peut refuser l'ajout de processus supplémentaires.
 * Un ordinateur a un nom.
 *
 * @author O. Bonaventure, Ch. Pecheur
 * @version Dec 2007
 */

public interface ComputerIF
{
    /**
     * @pre  p != null, p ne se trouve pas déjà sur cet ordinateur
     * @post le processus p a été ajouté à cet ordinateur, si les ressources
     *       nécessaires sont disponibles.  Retourne true si le processus
     *       a été ajouté, false sinon.
     */
    public boolean addProcess(Process p);

    /**
     * @pre p != null
     * @post le processus p a été retiré de cet ordinateur, si ce processus
     *       se trouve sur cet ordinateur.  Retourne true si le processus
     *       a été supprimé, false sinon.
     */
    public boolean removeProcess(Process p);

    /**
     * @pre -
     * @post Tous les processus de cet ordinateur ont été retirés.
     */
    public void removeAllProcesses();

    /**
     * @pre -
     * @post retourne le nom de l'ordinateur.
     */
    public String getName();

    /**
     * @pre -
     * @post Retourne la liste des processus de cet ordinateur sous forme de texte,
     *       avec une ligne par processus, chaque ligne comprenant le nom du processus
     *       et sa taille de stockage, séparés par un espace, et se termine par
     *       un passage à la ligne.  Par exemple:
     *
     *       process1 0
     *       bigprocess 200
     *       smallprocess 20
     */
    public String getState();
}
```

Fichier BasicComputer.java

```
/**
 * Un ordinateur de base, supportant un seul processus et sans capacité de stockage.
 *
 * @author O. Bonaventure, Ch. Pecheur
 * @version Dec. 2007
 */
public class BasicComputer implements ComputerIF
{
    private String name; // Nom de l'ordinateur
    private Process proc; // processus éventuel, null si absent

    /**
     * @pre name != null
     * @post Construit un BasicComputer de nom name.
     */
    public BasicComputer(String name)
    {
        this.name = name;
    }

    /**
     * @pre -
     * @post retourne le nom de l'ordinateur.
     */
    public String getName()
    {
        return name;
    }

    /**
     * @pre  p != null, p ne se trouve pas déjà sur cet ordinateur
     * @post le processus p a été ajouté à cet ordinateur, si aucun processus
     *       n'est présent et si p ne demande pas de stockage. Retourne true si
     *       le processus a été ajouté, false sinon.
     */
    public boolean addProcess(Process p)
    {
      if (proc == null && p.getRequiredStorage() == 0) {
            proc = p;
            return true;
      } else {
            return false;
      }
    }

    /**
     * @pre p != null
     * @post le processus p a été retiré de cet ordinateur, s'il s'agit bien du
     *       processus présent. Retourne true si le processus a été supprimé,
     *       false sinon.
     */
    public boolean removeProcess(Process p)
```

QUESTION 1

```
    /**
     * @pre -
     * @post Tous les processus de cet ordinateur ont été retirés. Retire proc
     *       s'il est actif.
     */
    public void removeAllProcesses()
    {
      proc = null;
    }

    /**
     * @pre -
     * @post Retourne la liste des processus de cet ordinateur sous forme de texte,
     *       avec une ligne par processus, chaque ligne comprenant le nom du processus
     *       et sa taille de stockage, séparés par un espace, et se termine par
     *       un passage à la ligne.  Par exemple:
     *
     *       process1 0
     *       bigprocess 200
     *       smallprocess 20
     */
    public String getState() {
      if (proc != null) {
            return proc.getDescr() + "\n";
      } else {
            return "";
      }
    }
}
```

Fichier Process.java

QUESTION 3

Fichier UnavailableException.java

```
/**
 * Une exception utilisée lorsqu'une opération ne peut être effectuée
 * par manque de ressources.
 *
 * @author O. Bonaventure, Ch. Pecheur
 * @version Dec. 2007
 */
public class UnavailableException extends Exception
{
      public UnavailableException()
      {
            super();
      }
}
```

Fichier FullComputer.java

```
/**
 * Un ordinateur avec capacité de stockage limitée et nombre de processus limité.
 *
 * @author O. Bonaventure, Ch. Pecheur
 * @version Dec. 2007
 */
public class FullComputer extends BasicComputer
{
      /**
       * Les processus présents sur cet ordinateur.  Les processus sont dans
       * procs[0] .. procs[count-1], et procs[i] == null pour i >= count.
       */
    private Process[] procs;
    private int count;         // nombre de processus présents
    private int storage;       // capacité de stockage totale
    private int availStorage;  // capacit?é de stockage restante

    /**
     * @pre n > 0, name != null, storage >= 0
     * @post Construit un FullComputer de nom name, supportant n processus
     *       et avec une capacité de stockage égale à storage
     */
```

QUESTION 2

```
    /**
     * @pre  p != null, p ne se trouve pas déjà sur cet ordinateur
     * @post le processus p a été ajouté à cet ordinateur, si (1) le nombre de
     *       processus maximal n'est pas atteint et (2) la capacité de stockage
     *       nécessaire pour p est disponible. Retourne true si le processus a
     *       été ajouté, false sinon.
     */
    public boolean addProcess(Process p)
```

QUESTION 4

RESTE DU CODE NON FOURNI

```
}
```

Fichier Cluster.java

```
/**
 * Une grappe (cluster) d'ordinateurs formant une ressource commune pour
 * l'exécution de processus.  Les ordinateurs du cluster sont gérés comme
 * une liste circulaire, de telle manière que les processus soient distribués
 * à tour de rôle à chaque ordinateur, dans la limite de leurs ressources disponibles.
 * La tête de la liste correspond prochain ordinateur à recevoir un nouveau processus,
 * pour autant qu'il ait les ressources nécessaires.
 *
 * @author O. Bonaventure, Ch. Pecheur
 * @version Dec. 2007
 */

import java.io.*;

public class Cluster
{
    // classe interne: un noeud de la liste circulaire des ordinateurs du cluster
    private class ListNode {
        ListNode next;
        ComputerIF elem;
    }

    /**
     * La tête courante de la liste des ordinateurs. Les noeuds suivants sont
     * chaînés de manière circulaire: la chaîne finit toujours par revenir à
     * current.
     */
    private ListNode current;
    private int count; // nombre d'ordinateurs dans le cluster

    /**
       * Constructeur
       */
    public Cluster()
    {
        count = 0;
        current = null;
    }

    /**
     * @pre p != null, p ne se trouve pas déjà sur un ordinateur du cluster.
     * @post Le processus p a été ajouté au premier ordinateur, à partir de la
     *       tête de la liste, disposant des ressources nécessaires. La nouvelle
     *       tête de liste est le noeud qui suit celui de l'ordinateur où p a
     *       été ajouté. Si aucun ordinateur ne dispose de ressources
     *       suffisantes, la tête de liste est inchangée et une
     *       UnavailableException est lancée.
       */
```

QUESTION 5

```java
/**
 * @pre p != null
 * @post Le processus p a été retiré du premier ordinateur du cluster
 *       sur lequel il se trouvait, à partir de la tête de la liste.
 *       Si p n'est pas trouvé, lance UnavailableException.
 */
public void removeProcess(Process p) throws UnavailableException
```

CODE NON FOURNI

```java
/**
 * @pre -
 * @post Tous les processus de tous les ordinateurs ont été retirés.
 */
public void removeAllProcesses()
```

CODE NON FOURNI

```java
/**
 * @pre  comp != null, comp ne fait pas déjà partie du cluster.
 * @post L'ordinateur comp est ajouté à la liste des ordinateurs.
 */
public void addComputer(ComputerIF comp)
{
    ListNode l = new ListNode();
    l.elem = comp;
   if(count == 0)
    {
     l.next = l;
     current = l;
    }
    else
    {
        l.next = current.next;
        current.next = l;
    }
    count++;
}

/**
 * @pre comp != null
 * @post L'ordinateur comp a été retiré du cluster, s'il s'y trouvait. Si
 *       comp est en tête de liste, la tête de liste passe au noeud suivant,
 *       sinon elle est inchangée. Retourne true si comp a été retiré, false
 *       sinon.
 */
```

QUESTION 6

```
    /**
     * @pre filename est un nom de fichier
     * @post Le fichier filename contient l'état du cluster sous forme de texte.
     *       Pour chaque processus de chaque ordinateur du cluster, le fichier
     *       contient une ligne composée du nom et de la capacité demandée,
     *       séparés par un espace. Par exemple :
     *           BlueJ 456
     *           Firefox 297
     *           Emacs 1276
     *           Terminal 27
     *       Le nom des ordinateurs sur lesquels se trouvent les processus n'est
     *       pas sauvegardé. Arrête le programme si une erreur d'I/O se produit.
     */
    public void saveState(String filename)
```

CODE NON FOURNI

```
    /**
     * @pre filename le nom d'un fichier sauvegardé par saveState
     * @post Retire tous les processus présents dans le cluster, puis ajoute au
     *       cluster les processus dont les noms et capacités sont donnés dans
     *       le fichier, selon le format généré par saveState.  Arrête le
     *       programme si une erreur d'I/O se produit ou si la capacité du
     *       cluster est insuffisante.  Les processus sont répartis équitablement
     *       entre les différents ordinateurs du cluster.
     */
    public void loadState(String fileName)
```

QUESTION 7

```
}
```