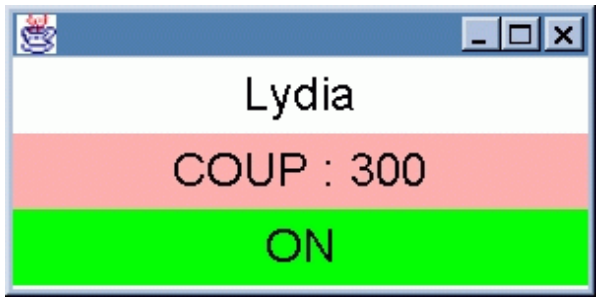
IUT Bordeaux 1 licence SIL-ACPI
Département d'Informatique

TP : Java - Semaphore

Le site http://java.sun.com/javase/6/docs/api/ contient la description complétée des classes (avec leurs méthodes) de Java SE 1.6.
Vos programmes doivent être stockés `$HOME/ProgConc/TP` (répertoire à créer, si nécessaire).

Exercice 1
Vous trouverez dans le répertoire `/net/Bibliotheque/ProgConc` le fichier `Alt.java`. Recopiez-le.

Normalement, `Alt` simule un jeu dans lequel deux joueurs (Alice et Bob) jouent alternativement. (un seul des 2 joueurs est actif à un instant donné) . Le joueur actif tire un nombre entier « aléatoirement », tps, et il reste actif durant tps secondes ; puis il passe la main à l'autre joueur. Chaque joueur (Alice et Bob) est visualisé par une fenêtre indiquant son nom, le nombre de coup joué et son état (On ou OFF), voir la figure ci-dessous. Alice commence les parties.



1. Complétez le code de `Alt.java` pour qu'il affiche le nombre de processus légers (thread) créés durant l'exécution du programme. Pour chaque processus léger, indiquez son nom, son groupe, sa priorité, et s'il s'agit d'un processus démon ou non.
2. Compilez puis exécuter `Alt.java`.
3. Réalisez le diagramme de Classes correspondant au code de `Alt.java`
4. Expliquez le problème qui se pose.
5. Modifiez le code (utilisez un seul Semaphore) pour que Bob attente la fin du premier coup d'Alice pour commencer à jouer.
indice : Solution de l'exercice 1 de la feuille de TD sur les sémaphores.
6. Utilisez un deuxième sémaphore pour bloquer Alice pendant le premier coup de Bob.
indice : Solution de l'exercice 2 de la feuille de TD sur les sémaphores.

Code de Alt.java (sauf classe Vue)

```
public class Alt {

    public static void main(String args[]) {
       try  {
           Jeu j = new Jeu();
           Joueur alice = new Joueur (j, "ALICE");
           Joueur bob = new Joueur (j, "BOB");
           alice.start();   bob.start(); alice.join(); bob.join();
           System.exit(0);
       } catch (Exception e) {System.err.println("Exception "+e);} }
}
```

```
class Jeu {
    private static int MAX_JOUEURS=2;
    private Joueur _joueurs[];
    private int _nbJoueurs;
    private Joueur _joueurCourant;

    public Jeu () {
        _joueurs = new Joueur[MAX_JOUEURS];
        _nbJoueurs=0; _joueurCourant =null;}

    public boolean addJoueur(Joueur jr) {
        if (_nbJoueurs >=  MAX_JOUEURS) return (false);
        if (_nbJoueurs>0) _joueurs[_nbJoueurs-1].setSuivant(jr);
         _joueurs[_nbJoueurs]=jr; jr.setSuivant(_joueurs[0]); _nbJoueurs++;
        If (_nbJoueurs == 1) setJoueurCourant();
         return(true); }

    public Joueur getJoueurCourant(){
        if (_joueurCourant==null) return(null);
        return _joueurCourant ;}

    // A completer et modifier
    private void setJoueurCourant() {_joueurCourant = _joueurs[0] ; }

    //A completer et modifier
    //donne la main au joueur suivant
    public boolean next(){
        if (_joueurCourant==null) return(false);
        _joueurCourant=_joueurCourant.getSuivant();
        return true; }
}

class Joueur extends Thread{
    private Random _rd;
    private Vue _maVue;
    static int MAX=20;
    private Jeu _jeu;
    private String _nom;
    private Joueur _suivant;

    //A Completer et modifier
     Joueur(Jeu j, String n){
        _jeu = j; rd = new Random(System.currentTimeMillis());
        _maVue = new Vue(n);_nom = n;_suivant = null; j.addJoueur(this);}

     public void setSuivant(Joueur jr) {_suivant = jr; }
     public Joueur getSuivant() {return _suivant; }

     //A Completer et modifier
     public void jouer(int nbCoups) {
        int tps; _maVue.setStatus(true);
        try {
         tps = _rd.nextInt(MAX)+1; _maVue.setValue(nbCoups,tps); sleep(tps*500);
        } catch(Exception e){System.err.println("Exception "+e); }
        _maVue.setStatus(false);}

    public void run(){
        int nbCoups=0;
        try {
            while(true) {     nbCoups++;jouer(nbCoups);_jeu.next(); }
        } catch(Exception e){ System.err.println("Exception "+e);} }
}
```

Exercice 2

Section critique – Une section critique est une portion de code qui doit être exécutée par au plus un processus à un instant donné. Une section critique peut être protégée par un sémaphore (généralement nommé Mutex).

Code C1 :

```
Debut
      Sémaphore Mutex ;
      Parbegin ProgA ; ProgA ; Parend
Fin
```

ProgA:

```
Debut
        T1;
        T_SC;
        T2;
Fin
```

1. Utilisez le sémaphore `Mutex` pour protéger la section critique `T_SC` dans le code C1.

Vous trouverez dans le répertoire `/net/Bibliotheque/ProgConc` le fichier `Compte.java`. Recopiez-le.

2. Compilez puis exécutez (plusieurs fois) `Compte`.
3. Expliquez le problème qui se pose.
4. Trouvez la section critique dans le code de la classe Cpt.java.
5. Protégez la section critique (utilisez un Semaphore).

```
import java.util.concurrent.Semaphore;
```

Code de compte.java:

```
class Compte extends Thread {
    private  static int _cpt = 1;
    private String _name;

    Compte(String name) { _name = name;   }
    static int getValeur() {return _cpt;}

    public void run() {
      int _c;
      for (int i = 1; i <= 100000; i++) {
          _c = _cpt; _cpt = _c+1; } }

     public static void main(String args[]) {
       System.out.println("VALEUR "+Cpt.getValeur());
       Cpt thr1 = new Cpt("Processus1");
       Cpt thr2 = new Cpt("Processus2");
       thr1.start(); thr2.start();
       try { thr1.join();thr2.join();} catch (Exception e) {System.out.println(e);}
       System.out.println("VALEUR "+Cpt.getValeur()); }
}
```

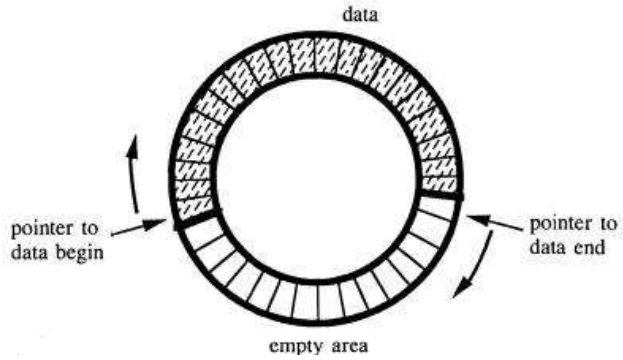
Exercice 3
Modifiez une dernière fois le code Alt.java pour répondre au spécification du problème
indice : Solution de l'exercice 5 de la feuille de TD sur les sémaphores.

Exercice 4
Recopiez le fichier `Alt.java` dans `Alt3.java`. Modifiez le code pour que le jeu se joue à trois joueurs Alice, Bob et Cathy ; Alice commence toutes les parties et à un instant donné un seul joueur est actif (joue). Les executions seront du type : Alice,Bob,Cathy, Alice, Bob, Cathy, ...

Exercice 5
Recopiez le fichier `Alt.java` dans `Alt2.java`. Modifiez le code pour qu'Alice commence toutes les parties et executent deux coups successifs avant de donnez la main à Bob :
Les executions seront du type : Alice,Alice,Bob,Alice, Alice, Bob, Alice, ...

Exercice 6 : Problème du tampon circulaire.
Des processus légers « *Producteur* » produisent des objets qui sont consommées par des processus légers « *Consommateur* ».

- Les objets doivent être consommés dans l'ordre de la production (ordre FIFO – First In First out).
- Les processus doivent pouvoir s'exécuter en parallèle en ayant des rythmes différents. C'est pourquoi les objets produits sont stockés dans un tampon circulaire de taille bornée. Le tampon peut contenir au plus TAILLE_TAMPON objets.



Vous trouverez dans le répertoire `/net/Bibliotheque/ProgConc` le fichier `ProdConc.java` qui contient le code à l'exception de la classe « Tampon ». Recopiez le fichier et lisez-le

1. Construisez le diagramme de classes en UML. Donner la signature des méthodes.
2. Programmer les méthodes la classe « Tampon » qui réalise le dépôt et le retrait d'objet dans le tampon circulaire à l'aide de sémaphores.

Contenu de `ProdConc.java` (sauf classe VueTampon et Vue) :

```
public class ProdConc {
  public static void main(String args[]) {
     try  {
       Tampon tp  = new Tampon();
       Producteur pr1 = new Producteur (tp, "Producteur1");
       Consommateur cs1 = new Consommateur (tp, "Consommateur1");
       Producteur pr2 = new Producteur (tp, "Producteur2");
       Consommateur cs2 = new Consommateur (tp, "Consommateur2");
       Producteur pr3 = new Producteur (tp, "Producteur3");
       pr1.start();   cs1.start(); pr2.start(); cs2.start() pr3.start();
   } catch (Exception e) {System.err.println("Exception "+e);} }
}
```

```
class Tampon {
    private static int TAILLE_TAMPON = 10;
    private int _ptrDebut;
    private char[] _tab;
    private int _espaceLibre ;
    private int _ptrFin;
    private VueTampon _maVue;
    private int _nbDepots, _nbRetraits;

    public Tampon() {  }
    public void deposer(char a) { }
    public char retirer() {char c='b'; return c; }
}

class Producteur extends Thread{
    private Random _rd;
    private Tampon _tp;
    private String _nom;
    private static int MAX=4;
    private Vue _maVue;

    public Producteur (Tampon tp, String n){
      _tp = tp; _rd = new Random(System.currentTimeMillis());
      _nom = n;_maVue  = new Vue(_nom);  }

    private char produire(int nbCoups) {
       int tps;
       try { tps = _rd.nextInt(MAX); sleep(tps*1000); }
       catch(Exception e){System.err.println("Exception "+e); }
       return 'a'; }

   public void run(){
       char c; int nbCoups =0;
       while(true) {
         c = produire(nbCoups);_maVue.setStatus(false);
         _tp.deposer(c);_maVue.setStatus(true); nbCoups++; }}
}

class Consommateur extends Thread{
    private Random _rd;
    private Tampon _tp;
    private String _nom;
    private static int MAX=4;
    private Vue _maVue;

    public Consommateur (Tampon tp, String n){
      _tp = tp; _rd = new Random(System.currentTimeMillis());
       _nom = n; _maVue = new Vue(_nom); }

   char consommer(int nbCoups) {
      int tps;
      try {    tps = _rd.nextInt(MAX); sleep(tps*1000);
      } catch(Exception e){System.err.println("Exception "+e); }
      return 'a';}

  public void run(){
   char a; int nbCoups =0;
   while(true) {
         a = consommer(nbCoups);_maVue.setStatus(false);
         a = _tp.retirer();_maVue.setStatus(true);nbCoups++;} }
}
```