Récupérez l'archive compressée `TP_B.tgz` disponible sur le site de l'UE et décompressez-la avec

```
tar -xvzf TP_B.tgz
```

Vous obtiendrez entre autres choses les fichiers suivants, issus des TD précédents :

```
BlancheNeige/SeptNains.java
Diner/Diner.java
```

Exercice B.1 Blanche-Neige équitable Le moniteur de Blanche-Neige de la figure 8 constitue une correction de l'exercice I.4. Ce code est disponible dans le fichier `SeptNains.java` de l'archive.

Question 1. Cette solution n'est pas *équitable* sur la plupart des machines. Testez-le sur votre machine.

Question 2. Corrigez le code de ce moniteur en plaçant les nains dans une *file d'attente* lorsqu'ils exécutent la méthode **`requerir()`** afin que chaque nain accède à Blanche-Neige à tour de rôle.

Question 3. Testez votre solution en affichant le contenu de la file d'attente.

Exercice B.2 La corde au dessus du canyon Il y a un canyon très profond quelque part dans le parc national Kruger, en Afrique du Sud, et une corde unique qui enjambe le canyon. Les babouins qui vivent là peuvent traverser le canyon à l'aide de la corde en se balançant d'une main sur l'autre. Cependant, lorsque deux babouins avancent dans des directions opposées, ils finissent par se rencontrer au dessus du canyon, car un babouin ne recule jamais : ils se battent alors et chutent mortellement tous les deux. Par ailleurs, la corde est juste assez solide pour supporter 5 babouins. S'il y a plus de babouins sur la corde en même temps, elle se rompt et provoque la chute mortelle de tous les babouins en train de traverser.

Nous supposerons dans cet exercice qu'il est possible d'enseigner aux babouins à utiliser la corde sans risque afin que toute traversée entâmée par un babouin se termine sans combat, ni surcharge. Pour cela, les babouins doivent simplement attendre pour s'engager sur la corde que les conditions soient favorables. Ainsi, pour éviter les chutes, l'utilisation de la corde doit satisfaire deux propriétés :

① Il n'y a jamais 2 babouins sur la corde qui avancent en sens opposés : un babouin qui observe sur la corde un congénère avançant vers lui devra donc attendre avant de s'engager.

② Il n'y a jamais plus de 5 babouins sur la corde : un babouin qui observe 5 de ses congénères en train de se balancer sur la corde devra donc attendre avant de s'engager.

Le respect de ces deux conditions garantit que toutes les traversées entamées se terminent avec succès.

Le code des babouins, donné sur la figure 9 et disponible dans l'archive, comporte un programme principal qui lance une vingtaine de babouins répartis sur les deux côtés du canyon. Ce programme s'appuie sur une énumération qui contient deux objets : EST et OUEST, qui représentent les deux côtés du canyon.

Question 1. Écrire le code de la classe **`Corde`** *sous la forme d'un moniteur*, de sorte que la corde puisse être utilisée par les babouins sans risque de chute *(sans modifier le programme principal, ni le code des babouins)*.

Question 2. L'éducateur des babouins décide à présent d'instaurer une nouvelle règle. Si un groupe de babouins circule sur la corde dans un sens alors qu'un babouin patiente sur l'autre côté (pour traverser dans l'autre sens), alors aucun autre babouin ne pourra s'engager dans le même sens avant qu'un babouin n'ait effectué une traversée dans l'autre sens. Ainsi, le babouin qui patiente aura la possibilité de s'engager lorsque le groupe aura terminé sa traversée et l'aura rejoint. Modifier le code de la corde afin d'implémenter cette nouvelle règle puis tester le code obtenu.

Exercice B.3 Apprentissage du lâcher-prise par des philosophes À partir du code des philosophes du fichier `Diner.java` de l'archive `TP_B.tgz` et de la figure 10, programmez des philosophes qui suivent le principe suivant : quand une fourchette est acquise, ils doivent tenter d'acquérir la seconde pour manger, mais en cas d'échec *prennent soin de relâcher la première* (pour ne pas être bloqué) puis recommencent à essayer de saisir les deux fourchettes (dans le même ordre).

Question 1. Munir les fourchettes d'une nouvelle méthode **`boolean tenter()`** qui permet de prendre une fourchette et qui renvoie **`true`** si la tentative a réussie, **`false`** sinon.

Question 2. Modifiez le code des philosophes afin qu'ils lâchent la première fourchette s'ils ne parviennent pas à saisir la seconde.

Question 3. Testez votre programme en affectant tous les philosophes d'une patience nulle et en signalant à l'écran les lâcher-prise lorsqu'ils se produisent.

Question 4. Selon vous, vos nouveaux philosophes peuvent-ils mourir de faim dans le cas particulier où chacune de leur patience est nulle ?

```
  class BlancheNeige{
    private volatile boolean libre = true; // Initialement, Blanche-Neige est libre

    public synchronized void requerir(){
      System.out.println(Thread.currentThread().getName() + "␣veut␣la␣ressource");
    }
    public synchronized void acceder(){
      while( ! libre ) {
        // Le nain s'endort sur le moniteur Blanche-Neige tant qu'elle n'est pas disponible.
        try { wait(); }
        catch (InterruptedException e) {e.printStackTrace();}
      }
      libre = false;
      System.out.println("\t" + Thread.currentThread().getName() + "␣accède␣à␣la␣ressource.");
    }
    public synchronized void relacher(){
      System.out.println("\t\t" + Thread.currentThread().getName() + "␣relâche␣la␣ressource.");
      libre = true;
      notifyAll();
    }
  }
```

FIGURE 8: Les sept nains

```
class Corde { ... }

enum Cote { EST, OUEST }                    // Le canyon possède un côté EST et un côté OUEST

class Babouin extends Thread{
  private static int numeroSuivant = 0;     // Compteur partagé par tous les babouins
  private int numero;                       // Numéro du babouin
  private Corde corde;                      // Corde utilisée par le babouin
  private Cote origine;                     // Côté du canyon où apparaît le babouin: EST ou OUEST
  Babouin(Corde corde, Cote origine){       // Constructeur de la classe Babouin
    this.corde = corde;                     // Chaque babouin peut utiliser la corde
    this.origine = origine;                 // Chaque babouin apparaît d'un côté précis du canyon
    numero = ++numeroSuivant;               // Chaque babouin possède un numéro distinct
  }
  public void run(){
    System.out.println("Le␣babouin␣" + numero + "␣arrive␣sur␣le␣côté␣" + origine + "␣du␣canyon.");
    corde.saisir(origine);                  // Pour traverser, le babouin saisit la corde
    System.out.println("Le␣babouin␣" + numero +
                       "␣commence␣à␣traverser␣sur␣la␣corde␣en␣partant␣de␣l'" + origine + ".");
    try { sleep(5000); } catch(InterruptedException e){} // La traversée ne dure que 5 secondes
    System.out.println("Le␣babouin␣" + numero + "␣a␣terminé␣sa␣traversée.");
    corde.lacher(origine);                  // Arrivé de l'autre côté, le babouin lâche la corde
    System.out.println("Le␣babouin␣" + numero + "␣a␣lâché␣la␣corde␣et␣s'en␣va.");
  }
  public static void main(String[] args){
    Corde corde = new Corde();              // La corde relie les deux côtés du canyon
    for (int i = 1; i < 20; i++){
      try { Thread.sleep(500); } catch(InterruptedException e){}
      if (Math.random() >= 0.5){
        new Babouin(corde, Cote.EST).start();    // Création d'un babouin à l'est du canyon
      } else {
        new Babouin(corde, Cote.OUEST).start();  // Création d'un babouin à l'ouest du canyon
      }
    } // Une vingtaine de babouins sont répartis sur les deux côtés du canyon
  }
}
```

FIGURE 9: Le code d'une corde enjambant un canyon, à compléter

```
import java.util.Random;
class Fourchette {
  boolean volatile libre;
  Thread possesseur;
  public Fourchette(){ libre = true; }
  synchronized void prendre(){
      while ( !libre ) {
          try { wait(); }
          catch (InterruptedException e) { e.printStackTrace(); }
      }
      possesseur = Thread.currentThread();
      libre = false;
  }
  synchronized void lacher(){
    if ( possesseur == Thread.currentThread() ){
      libre = true;
      notifyAll();
    }
  }
}
class Philosophe extends Thread {
  Fourchette fg;       // La fourchette à gauche
  Fourchette fd;       // La fourchette à droite
  String nom;          // Le nom du philosophe
  int patience;        // Le temps de la reflexion
  public Philosophe(Fourchette fg, Fourchette fd, String nom){
    this.fg = fg;
    this.fd = fd;
    this.nom = nom;
    Random alea = new Random();
    this.patience = alea.nextInt(5000);
  }
  public void run(){
    while(true){
      System.out.println("[Phil] ("+this.nom+") Je pense "+patience+" ms.");
      try { Thread.sleep(patience); }
      catch (InterruptedException e) { e.printStackTrace(); }
      System.out.println("[Phil] ("+this.nom+") Je suis affamé.");
      fg.prendre();  // Le philosophe prend d'abord la fourchette à gauche
      fd.prendre();  // Puis la fourchette à droite
      System.out.println("[Phil] ("+this.nom+") Je mange pendant 3s.");
      try { Thread.sleep(3000); }
      catch (InterruptedException e) { e.printStackTrace(); }
      System.out.println("[Phil] ("+this.nom+") J'ai bien mangé.");
      fg.lacher();
      fd.lacher();
    }
  }
  public class Diner {
    public static void main(String[] argv){
      Fourchette f1 = new Fourchette();
      Fourchette f2 = new Fourchette();
      Fourchette f3 = new Fourchette();
      Fourchette f4 = new Fourchette();
      Fourchette f5 = new Fourchette();
      Philosophe p1 = new Philosophe(f5,f1,"Socrate");
      Philosophe p2 = new Philosophe(f1,f2,"Aristote");
      Philosophe p3 = new Philosophe(f2,f3,"Epicure");
      Philosophe p4 = new Philosophe(f3,f4,"Descartes");
      Philosophe p5 = new Philosophe(f4,f5,"Nietzsche");
      p1.start(); p2.start(); p3.start(); p4.start(); p5.start();
    }
  }
}
```

FIGURE 10: La classe des philosophes