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

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 espèce de *file d'attente* lorsqu'ils exécutent la méthode **`requerir()`** de sorte que chaque nain accède à Blanche-Neige à tour de rôle.

Question 3. Testez votre solution, en affichant si nécessaire le contenu de la file d'attente à chaque étape.

```
class BlancheNeige {
  private volatile boolean libre = true;       // Initialement, l'objet partagé bn 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 bn tant qu'il n'est pas libre.
      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 4 – Blanche-Neige non équitable

Exercice B.2 Interruption des nains Indépendamment de son état courant, chaque thread possède un *statut d'interruption*, levé ou non, codé sous la forme d'un booléen. Initialement faux (c'est-à-dire baissé), le statut d'interruption d'un thread **`t`** peut être levé en appelant la méthode **`t.interrupt()`**. La méthode **`t.isInterrupted()`** renvoie le statut d'interruption du thread **`t`** sans le modifier. En revanche, la méthode **`t.interrupted()`** renvoie le statut d'interruption du thread **`t`** et le replace à faux s'il est levé.

Lorsqu'un thread exécute une instruction bloquante telle que **`sleep()`**, **`join()`** ou **`wait()`**, une exception **`InterruptedException`** est levée si ce thread est interrompu lors de l'appel ou au cours de son exécution. Point important, le statut d'interruption du thread est rabaissé lorsque cette exception est levée.

Nous reprenons dans cet exercice la correction de l'exercice I.4 disponible dans le fichier `SeptNains.java` de l'archive `TP_B.tgz`. Il s'agit à présent de provoquer la fin de l'exécution de ce programme au bout de 5 secondes.

Question 1. Modifiez le code du **`main`** afin que les sept nains soient interrompus au bout de 5 secondes.

Question 2. Assurez-vous que chaque nain termine en affichant un message d'adieu dès qu'il est interrompu.

Question 3. Modifiez le code du **`main`** afin que le programme affiche un message final après que chaque nain a affiché son message d'adieu.

Exercice B.3 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 le poids de 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 5 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. Ajoutez au programme un affichage permettant de visualiser à chaque étape la liste des babouins en attente de chaque côté du canyon et en transit sur la corde. Puis testez votre programme.

```
class Corde { ... }
enum Cote { EST, OUEST }                  // Le canyon possède un côté EST et un côté OUEST

class Babouin extends Thread{
  private final 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 c, Cote o, int i){        // 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 = i;                           // 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 +
                        "␣a␣commencé␣à␣traverser␣sur␣la␣corde␣en␣partant␣de␣l'" +origine+ ".");
    try { sleep(5000); } catch(InterruptedException ignoree){}  // La traversée dure 5 secondes
    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(2000); } catch(InterruptedException ignoree){}
      if (Math.random() >= 0.5){
        new Babouin(corde, Cote.EST, i).start();    // Création d'un babouin à l'est du canyon
      } else {
        new Babouin(corde, Cote.OUEST, i).start();// Création d'un babouin à l'ouest du canyon
      }
    }                  // Une vingtaine de babouins sont répartis sur les deux côtés du canyon
  }
}
```

FIGURE 5 – Le code d'une corde enjambant un canyon, à compléter