Exercice B.1 Blanche-Neige incorrecte Nous poursuivons l'exercice I.4 des sept nains à l'aide du code du fichier `SeptNains.java` disponible dans l'archive `TP_B.zip`. La classe **`SeptNains`** de ce programme contient deux classes internes : **`BlancheNeige`** et **`Nain`** décrites sur les figures 7 et 9. Ce programme est erroné puisque plusieurs nains peuvent accéder simultanément à Blanche-Neige, comme le montre l'exécution décrite sur la figure 8.

Question 1. Vérifiez ce scenario catastrophique sur votre machine [5]. Pouvez-vous l'expliquer ?

Question 2. Remplacez **`if`** par **`while`** dans le code de la méthode **`accéder()`**. Le code est alors correct, puisque l'exclusion mutuelle est assurée, mais cependant l'objet Blanche-Neige n'est pas *équitable*. Vérifiez-le sur votre machine [6].

Question 3. Corrigez le code de Blanche-Neige en plaçant intuitivement les nains dans une *file d'attente* lorsqu'ils exécutent la méthode **`requérir()`**, qui réalise ainsi une sorte d'inscription, afin que chaque nain accède à Blanche-Neige à tour de rôle *dans l'ordre où ils sont inscrits*.

Question 4. Testez votre solution, en affichant le contenu de la file d'attente à chaque étape.

```
static class BlancheNeige {
  private volatile boolean libre = true;   // Initialement, Blanche-Neige est libre
  public synchronized void requérir() {
    affiche("veut un accès exclusif à la ressource");
  }
  public synchronized void accéder() throws InterruptedException {
    if ( ! libre ) wait() ;                // Le nain attend un signal sur this
    libre = false;
    affiche("accède à la ressource");
  }
  public synchronized void relâcher() {
    affiche("relâche la ressource");
    notifyAll();                           // Le nain envoie un signal sur this
    libre = true;
  }
}
```

FIGURE 7 – Blanche-Neige erronée

```
$ java SeptNains
[09h 02mn 15,562s] [Simplet] veut un accès exclusif à la ressource.
[09h 02mn 15,563s] [Simplet] accède à la ressource.
[09h 02mn 15,563s] [Simplet] a un accès (exclusif) à Blanche-Neige.
[09h 02mn 15,563s] [Joyeux] veut un accès exclusif à la ressource.
[09h 02mn 15,563s] [Atchoum] veut un accès exclusif à la ressource.
[09h 02mn 15,563s] [Dormeur] veut un accès exclusif à la ressource.
[09h 02mn 15,563s] [Timide] veut un accès exclusif à la ressource.
[09h 02mn 15,563s] [Prof] veut un accès exclusif à la ressource.
[09h 02mn 15,564s] [Grincheux] veut un accès exclusif à la ressource.
[09h 02mn 17,568s] [Simplet] s'apprête à quitter Blanche-Neige.
[09h 02mn 17,569s] [Simplet] relâche la ressource.
[09h 02mn 17,569s] [Simplet] veut un accès exclusif à la ressource.
[09h 02mn 17,570s] [Simplet] accède à la ressource.
[09h 02mn 15,570s] [Simplet] a un accès (exclusif) à Blanche-Neige.
[09h 02mn 17,571s] [Joyeux] accède à la ressource.
[09h 02mn 17,571s] [Joyeux] a un accès (exclusif) à Blanche-Neige.
^C
```

FIGURE 8 – Une exécution erronée du programme

5. L'exécution observée sur votre machine devrait être analogue à celle de la figure 8.
6. À nouveau, l'inéquité de Blanche-Neige devrait apparaître immédiatement quelle que soit la machine utilisée.

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 poursuivons l'exercice B.1 et souhaitons à présent provoquer la fin de l'exécution de ce programme au bout de 5 secondes. Ceci implique que le `main()` parvienne au bout de son code, de même que chaque nain. Au cours de cette terminaison du programme, l'exclusion mutuelle de l'accès à Blanche-Neige doit évidemment être conservée.

Question 1. Modifiez le `main()` afin que les sept nains soient interrompus au bout de 5 secondes. Observez que les nains ne s'arrêtent pas, mais qu'il y a des exceptions levées.

Question 2. Modifiez le `run()` des nains afin que chaque nain termine l'exécution de son code en affichant un message d'adieu dès qu'il est interrompu. Ceci implique de traiter toutes les exceptions.

Question 3. Modifiez le `main()` afin que le programme affiche un message final après que chaque nain a affiché son message d'adieu et terminé son exécution.

Question 4. Modifiez le `run()` des nains afin que le nain interrompu pendant qu'il accède à la ressource conserve son privilège pendant les 2 secondes allouées avant de libérer Blanche-Neige et de sortir de la boucle d'itération.

```
static class Nain extends Thread {
  public Nain(String nom) {
    this.setName(nom);
  }
  public void run() {
    while(true) {
      bn.requérir();
      try { bn.accéder(); } catch (InterruptedException e) {e.printStackTrace();}
      affiche("a un accès (exclusif) à Blanche-Neige");
      try { sleep(2000); } catch (InterruptedException e) {e.printStackTrace();}
      affiche("s'apprête à quitter Blanche-Neige");
      bn.relâcher();
    }
  }
}
```

FIGURE 9 – Les sept nains

Exercice B.3 Grincheux impatient Nous poursuivons l'exercice B.2 et souhaitons modifier le comportement du nain Grincheux de la manière suivante : lorsqu'il fait appel à la méthode `accéder()` et qu'il patiente parce que Blanche-Neige est indisponible, il appeler la méthode `affiche()`) toutes les secondes (mais pas plus) pour exprimer un message d'impatience, du genre « *Et alors ?* »

Question 1. Corrigez le code de la méthode `accéder()` afin de mettre en oeuvre ces messages supplémentaires de la part du nain Grincheux. Vous pourrez utiliser la méthode `currentTimeMillis()` de la classe `System` qui renvoie un entier long correspondant à la date courante (en millisecondes), et la méthode `wait(d)` de la classe `Object` qui attend un signal pendant une durée limitée à `d` millisecondes.

Question 2. Testez votre programme.

Question 3. Vérifiez que votre code est bien robuste vis-à-vis des réveils intempestifs (les « *spurious wake-up* ») en lançant à partir du `main` un signal sur l'objet Blanche-Neige chaque dixième de seconde.