Exercice D.1 Problème de vue Le programme `Visible.java` ci-dessous est disponible sur le site de l'UE dans l'archive `CM_5.zip`.

```
public class Visible {

  public static void main(String[] args) throws Exception {
    A a = new A() ;          // Création d'un objet "a" de la classe A
    a.start() ;              // Lancement du thread "a"
    Thread.sleep(1000) ;
    a.valeur = 1 ;           // Modification de l'attribut valeur de "a"
    a.fin = true ;           // Modification de l'attribut fin de "a"
    System.out.println("Le main a terminé.") ;
  }
}

class A extends Thread {
  public int valeur = 0 ;
  public boolean fin = false ;

  public void run() {
    while(! fin) {} ;        // Boucle d'attente active
    System.out.println(valeur) ;
  }
}
```

Question 1. Vérifiez tout d'abord que ce programme ne termine pas sur votre machine. Sinon, changez de machine et prenez par exemple un ordinateur des salles de TP.

Question 2. Testez à nouveau ce programme

(a) ou bien en supprimant l'instruction **`Thread.sleep(1000);`** dans le **`main`**

(b) ou bien en ajoutant une instruction **`System.out.print();`** dans la boucle d'attente active.

Le programme termine-t-il après l'une ou l'autre de ces modifications ?

Question 3. Que faut-il ajouter, en principe, au code fourni dans l'archive pour que le programme termine à coup sûr ? Testez cette troisième alternative.

Exercice D.2 Ajout d'une louche au pôt des sauvages Nous poursuivons l'exercice III.2 à l'aide du codage du pôt donné sur la figure 20 et disponible dans l'archive `TP_D.zip`.

Question 1. Observez sur une exécution que plusieurs sauvages peuvent attendre simultanément que le pôt soit plein. Observez également que le premier sauvage ayant réveillé le cuisinier n'est pas toujours le premier à se servir et qu'il n'est même pas certain de pouvoir se servir une fois le pôt rempli.

Question 2. Il s'agit dans cet exercice de garantir que le sauvage qui réveille le cuisinier en premier soit toujours le premier à se servir lorsque le pôt est plein. Pour celà, il suffit d'ajouter une louche au pôt et d'exiger qu'un sauvage prenne la louche pour se servir et la garde lorqu'il réveille le cuisinier. Décommentez les lignes appropriées dans le code donné dans l'archive afin de modifier le comportement des sauvages.

Question 3. Observez que le système peut bloquer. Pouvez-vous expliquer cette situation ? Dans quel état sont le cuisinier et le sauvage qui tient la louche ?

Question 4. Corrigez le programme faisant de l'attribut **`louche`** un simple booléen et en adaptant le reste du code du pôt afin d'utiliser correctemant cette louche. Testez votre programme pour vérifier que le sauvage qui réveille le cuisinier est bien le premier à se servir ensuite.

Question 5. Assurez-vous que le cuisinier ne remplit jamais le pôt sans qu'un sauvage ne le réveille.

Exercice D.3 Les nains impatients La figure 16 décrit une Blanche-Neige équitable sous la forme d'un moniteur. Ce code est disponible dans l'archive `TP_D.zip`. Il s'agit dans cet exercice de modifier le comportement des nains de la manière suivante : lorsqu'un nain fait appel à la méthode **`accéder()`** et qu'il patiente parce que Blanche-Neige est indisponible, il doit afficher un message d'impatience « Et alors ? » toutes les secondes (mais pas plus).

Question 1. Corrigez le code de la méthode **`accéder()`** afin de mettre en oeuvre ces messages supplémentaires. 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(délai)`** de la classe **`Object`** qui attend un signal pendant une durée limitée à **`délai`** millisecondes.

```
class Pot {
  private volatile int nbPortions;
  private final int volume;

  public Pot(int nbPortions) {
    this.volume = nbPortions;
    this.nbPortions = nbPortions;
  }

  synchronized public void remplir() throws InterruptedException {
    while( nbPortions != 0 ){ wait(); } // Tant que le pôt n'est pas vide, je ne le remplis pas.
    System.out.println("Cuisinier: Je suis réveillé et je cuisine...");
    Thread.sleep(2000);
    nbPortions = volume;
    System.out.println("Cuisinier: Le pôt est plein!");
    notifyAll();
  }

  synchronized public void seServir() throws InterruptedException {
    if ( nbPortions != 0 ) {
      System.out.println(Thread.currentThread().getName()+": Il y a une part disponible ! ");
    } else { // Le pot est vide: on réveille le cuisinier
      System.out.println(Thread.currentThread().getName()+": Le pôt est vide!");
      System.out.println(Thread.currentThread().getName()+": Je réveille le cuisinier.");
      notifyAll();
      System.out.println(Thread.currentThread().getName()+": J'attends que le pôt soit plein!");
      while ( nbPortions == 0 ) { wait(); } // Tant que le pôt est vide, je ne me sers pas.
      System.out.println(Thread.currentThread().getName()+": Je me réveille! Je me sers.");
    }
    nbPortions--;
  }
}
```

FIGURE 20 – Pôt des sauvages sous la forme d'un moniteur

Question 2. Testez votre programme en veillant à dater lisiblement les nouveaux messages.

Question 3. Vérifiez que votre code est bien robuste vis-à-vis des réveils intempestifs (les « *spurious wake-up* ») en lançant, via le **main**, des signaux superflus suffisamment nombreux vers l'objet Blanche-Neige.

Exercice D.4 Fabrique d'un GIF animé Pour conclure, il s'agit de fabriquer une image GIF animée, telle que celle donnée dans le répertoire `Mandelbrot` de l'archive `TP_D.zip`, afin d'illustrer l'évolution du programme `Mandelbrot.java` donné dans l'archive `TP_A.zip`. Pour celà, il faut au préalable construire une série de fichiers qui décrivent l'évolution de l'état du dessin au cours de l'exécution du programme.

Question 1. Modifiez la valeur de **max** pour que l'exécution dure entre 20 et 60 s. sur votre machine.

Question 2. L'instruction **image.save("toto.png")** sauvegarde dans le fichier `toto.png` une image au format PNG qui correspond à l'état courant de l'objet **image**. Modifiez le programme pour sauvegarder, dans une série de fichiers numérotés sur trois chiffres et nommés sous la forme `picXXX.png`, les états intermédiaires du dessin, *à raison d'un fichier toutes les 100 millisecondes*, c'est-à-dire dix images par seconde.

Ça ne devrait donc pas faire plus que 600 fichiers, approximativement. Vous prendrez soin néanmoins d'empêcher votre programme de produire plus d'un millier de fichiers PNG quelque soit la durée réelle de l'exécution du programme.

Question 3. Lancez la commande Linux : `convert pic*.png m.gif` puis affichez le fichier `m.gif` obtenu.