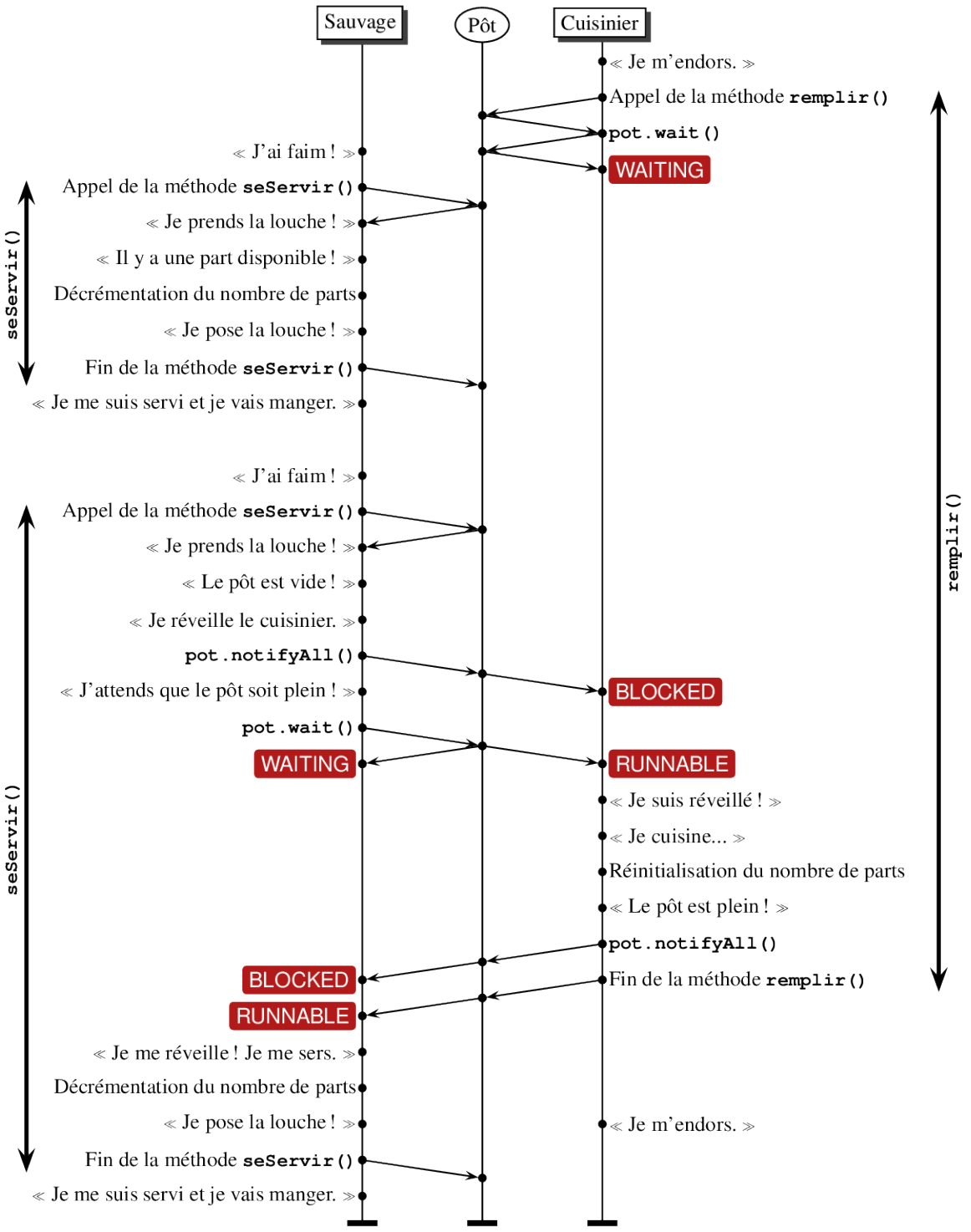
Exercice C.1 Le pôt des sauvages Nous poursuivons l'exercice III.1 en choisissant de munir le pôt d'une louche utilisée pour se servir, selon le diagramme de séquence ci-dessous :



Complétez le code du programme `Diner.java` disponible dans l'archive `TP_C.tgz` (fig. 9) *en conservant une structure de moniteur pour le pôt*, sans modifier le programme principal ni les classes **`Sauvage`** et **`Cuisinier`**. Il vous faut donc dans cet exercice ajouter la saisie et la pose de la louche, puis vérifier sur quelques exécutions que le sauvage qui réveille le cuisinier est bien toujours le premier à se servir. Vérifiez ensuite que le cuisinier ne remplit jamais le pôt sans qu'un sauvage ne le réveille parce qu'il a constaté que le pôt est vide.

Exercice C.2 Le passeur de Nains et de Hobbits Quelque part en Terre du Milieu, un passeur utilise une barque pour aider des hobbits et des nains à traverser une rivière. La barque peut contenir quatre passagers en plus du passeur ; celui-ci ne quitte une berge que lorsque la barque est complètement remplie, afin d'optimiser son effort. Pour garantir la tranquilité de la traversée et éviter les chamailleries habituelles, le passeur s'interdit de mettre un nain dans sa barque avec trois hobbits, ou de mettre un hobbit avec trois nains. Toute autre combinaison de passagers dans la barque ne pose aucune difficulté. Dès lors que quatre passagers sont montés à bord, le passeur se saisit des rames afin d'effectuer la traversée.

Cet exercice consiste à compléter la classe **`Barque`** dans le programme de la figure 11 afin de modéliser ce système. Il y a initialement **`nbNainsAuNord`** nains et **`nbHobbitsAuNord`** hobbits au Nord de la rivière, et **`nbNainsAuSud`** nains et **`nbHobbitsAuSud`** hobbits au Sud. Tous veulent traverser la rivière sur la barque du passeur. Le **`main`** de ce programme lance un thread pour le passeur, puis un thread pour chaque nain et pour chaque hobbit. Chacun de ces threads connaît la barque du passeur.

Question 1. Le squelette de code indiqué sur la figure 11 est disponible dans l'archive `TP_C.tgz`. Codez la classe **`Barque`** sous la forme d'un moniteur *en supposant tout d'abord qu'il n'y a aucun hobbit sur les berges.*
- Veillez à ce qu'aucun passager ne monte sur la barque avant que le passeur ne dise de monter. Veillez aussi à ce qu'aucun passager ne descende de la barque avant que le passeur ne dise de descendre.
- Hormis *les variables indiquant la répartition des nains et des hobbits sur les berges*, le programme principal et les classes **`Passeur`**, **`Nain`** et **`Hobbit`** ne doivent pas être modifiés.
- L'attente active est naturellement formellement proscrite.

Question 2. Ajoutez, dans le code de la classe **`Barque`**, des messages sur la sortie standard afin de visualiser pour chaque nain l'embarquement d'un côté de la rivière et le débarquement de l'autre côté. Testez et éventuellement corrigez votre programme. Puis sauvegardez une sortie écran d'une exécution *complète* de votre programme dans un fichier nommé `test1.txt` avec initialement 8 nains au Nord et 4 au Sud.

Question 3. Complétez la classe **`Barque`** afin de gérer également la présence de hobbits sur les berges. Vérifiez qu'à aucun moment la barque ne supporte une configuration de passagers interdite.

Question 4. Testez votre programme avec : au Nord, 6 nains et 2 hobbits ; au Sud, 4 nains et 2 hobbits. Sauvegardez ensuite dans un fichier `test2.txt` une sortie écran d'une exécution complète de votre programme qui illustre le fait que lorsque deux nains et un hobbit sont déjà montés sur la barque, un troisième nain ne peut monter pour compléter le chargement.

```
public class Barque{
  public enum Berge { NORD, SUD }
  final static int nbNainsAuNord = 6;
  final static int nbHobbitsAuNord = 2;
  final static int nbNainsAuSud = 4;
  final static int nbHobbitsAuSud = 2;
  public static void main(String[] args){
    Barque laBarque = new Barque();
    new Passeur(laBarque);
    for(int i=0; i<nbNainsAuNord; i++) new Nain(laBarque,Berge.NORD).setName("NN-"+i);
    for(int i=0; i<nbHobbitsAuNord; i++) new Hobbit(laBarque,Berge.NORD).setName("HN-"+i);
    for(int i=0; i<nbNainsAuSud; i++) new Nain(laBarque,Berge.SUD).setName("NS-"+i);
    for(int i=0; i<nbHobbitsAuSud; i++) new Hobbit(laBarque,Berge.SUD).setName("HS-"+i);
  }
  ... // COMPLÉTEZ LA BARQUE EN NE MODIFIANT NI LE PASSEUR, NI LES NAINS, NI LES HOBBITS
}
class Passeur extends Thread{
  public Barque laBarque;
  public Passeur(Barque b){ this.laBarque = b; start(); }
  public void run(){
    laBarque.accosterLaBerge(Barque.Berge.NORD);
    System.out.println("PASSEUR> La barque est au NORD: montez!");
    while(true){
      laBarque.charger();
      System.out.println("PASSEUR> La barque est correctement chargée.");
      System.out.println("PASSEUR> Je rame vers le SUD.");
      try { sleep(1000); } catch (InterruptedException e) {e.printStackTrace();}
      laBarque.accosterLaBerge(Barque.Berge.SUD);
      System.out.println("PASSEUR> La barque est au SUD: descendez!");
      laBarque.decharger();
      System.out.println("PASSEUR> La barque est vide: montez!");
      laBarque.charger();
      System.out.println("PASSEUR> La barque est correctement chargée.");
      System.out.println("PASSEUR> Je rame vers le NORD.");
      try { sleep(1000); } catch (InterruptedException e) {e.printStackTrace();}
      laBarque.accosterLaBerge(Barque.Berge.NORD);
      System.out.println("PASSEUR> La barque est au NORD: descendez!");
      laBarque.decharger();
      System.out.println("PASSEUR> La barque est vide: montez!");
    }
  }
}
class Nain extends Thread{
  public Barque laBarque;
  public Barque.Berge origine;
  public Nain(Barque b, Barque.Berge l){ this.laBarque = b; this.origine = l; start(); }
  public void run(){
    Random alea = new Random();
    try {sleep(500+alea.nextInt(100)*50);} catch (InterruptedException e) {e.printStackTrace();}
    System.out.println(getName() + "> Je souhaite traverser.");
    laBarque.embarquerUnNain(origine);
    laBarque.debarquerUnNain(origine);
  }
}
class Hobbit extends Thread{
  public Barque laBarque;
  public Barque.Berge origine;
  public Hobbit(Barque b, Barque.Berge l){ this.laBarque = b; this.origine = l; start(); }
  public void run(){
    Random alea = new Random();
    try {sleep(500+alea.nextInt(100)*50);} catch (InterruptedException e) {e.printStackTrace();}
    System.out.println(getName() + "> Je souhaite traverser.");
    laBarque.embarquerUnHobbit(origine);
    laBarque.debarquerUnHobbit(origine);
  }
}
```

FIGURE 11 – Code du système à compléter