Programmation Répartie **Votre nom:** _____ **2020 – 2021**

DS n°1 *Devoir*

Répondez à ces questions dans un document que vous déposerez sur Célène. Vous disposez de 40 minutes.

Exercice 1. Question de cours

1.1 Quel problème entraîne un interblocage (ou deadlock), et en quoi est-ce difficile de le détecter ?

1.2 Quels sont les avantages et les inconvénients des verrous par rapport aux blocs `synchronized` ?

1.3 Qu'est-ce qu'un réveil parasite d'un thread, et comment l'éviter ?

1.4 Quel est l'avantage du framewok Executor par rapport à la création classique de thread ?

1.5 A quoi sert la classe `CountDownLatch` ? Quelle est la principale différence avec la classe `CyclicBarrier` ?

1.6 Vous avez développé un objet accessible en RMI. Que devez vous donner au client pour qu'il puisse l'utiliser (informations, adresse, classe/interface ...)

Exercice 2. Code

Soit les deux codes suivant (disponible également dans une archive), quels sont les potentiels problèmes pour chacun (conception, compilation, exécution ...). Vous n'avez pas à fournir de code corrigé. Les deux questions sont indépendantes.

2.1

```
import java.util.concurrent.locks.*;

/**
Classe utile dans le cadre d'un Producteur/consommateur
 **/
class Donnee{


    Object valeur;
    Lock verrou;
    Condition attente;


    public synchronized void ajout(Object nouvelleValeur) throws
       Exception{
  verrou.lock();
  if (valeur!=null)
      attente.wait();

  valeur=nouvelleValeur;
  verrou.unlock();
    }

    public synchronized  Object recuperer() throws Exception {
  verrou.lock();
  if (valeur==null)
      attente.wait();

      Object retour = valeur;
      valeur = null;
      return retour;
    }



}
```

2.2

```
import java.util.concurrent.locks.*;
import java.util.*;

class Messages{

    List<String> messages;

    public Messages(){
  messages= new ArrayList<String>();
    }

    public String lireLePlusRecent(){
      Lock lk = new ReentrantLock();
      lk.lock();
      String dernier="";
      if (messages!=null)
   dernier =messages.get(0);
      lk.unlock();
      return dernier;
    }

    public void ajouterMessage(String message){
  messages.add(message);
    }

    public static void main(String args[]){
      Messages messages = new Messages();
      T1 t1 = new T1(messages); t1.start();
      T1 t2 = new T1(messages); t2.start();
    }
}

class T1 extends Thread{
    Messages messages;
    T1(Messages messages){
      this.messages = messages;
    }

    public void run(){
      Lock lk = new ReentrantLock();
      lk.lock();
        messages.ajouterMessage("Bonjour");
      lk.unlock();
      System.out.println(messages.lireLePlusRecent());
    }

}
```