TP n°5 bis

Les threads

Exercice 1 : Sémaphore

Un sémaphore correspond à un compteur qui ne peut être accédé que par deux opérations atomiques: **acquire** et **release**. L'opération **acquire** décrémente le compteur alors que l'opération **release** l'incrémente. Les sémaphores sont souvent utilisés pour créer des sections critiques accessibles à n threads (n correspondant à la valeur d'initialisation du compteur **permits**). L'opération acquire doit donc mettre en attente le (n+1)ième thread souhaitant entrer en section critique. L'opération **release** réveille un thread en attente. Lorsque la variable **permits** est négative, sa valeur absolue représente le nombre de threads en attente. La classe **Semaphore** est fournie ci-dessous:

```
class Semaphore {
    private int permits;
    public Semaphore(int initialPermits) {
        permits = initialPermits;
    }
    public synchronized void acquire() {
        permits = permits 1;
        while (permits < 0) {
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }
    }
    public synchronized void release() {
        permits = permits + 1;
        if (permits <= 0)
        notify();
    }
}
```

Utiliser la classe ***Semaphore*** pour compléter les classes ***Main***, ***ThreadA***, ***ThreadB*** et ***ThreadC*** fournies. Les classes ***ThreadA***, ***ThreadB*** et ***ThreadC*** affichent respectivement cinq A, cinq B et cinq C. Synchronisez vos threads à l'aide de sémaphores de sorte que l'affichage effectué par l'application soit : ACBACBACBACBACB.

```
public class Main {
    public static void main(String args[]) {
        Semaphore mutexA = new Semaphore(...);
        ...
        new ThreadA(mutexA, ...).start();
        new ThreadB(...).start();
        new ThreadC(...).start();
    }
}

public class ThreadA extends Thread {
    Semaphore mutexA;
    ...
    public ThreadA(Semaphore mutexA, ...) {
        this.mutexA = mutexA;
        ...
    }
```

```
      public void run() {
            for (int i = 0; i < 5; i++) {
            ...
            System.out.print("A");
            ...
            }
      }
}

public class ThreadB extends Thread {
      ...
      public ThreadB(...) {
            ...
      }
      public void run() {
            for (int i = 0; i < 5; i++) {
                  ...
            }
      }
}

public class ThreadC extends Thread {
      ...
      public ThreadC(...) {
            ...
      }
      public void run() {
            for (int i = 0; i < 5; i++) {
                  ...
            }
      }
}
```

Exercice 2 : Simulation d'un parking

On souhaite modéliser un parking (classe ***CarPark***). Ce parking possède un nombre limité de places défini dans la variable ***capacity***. Chaque voiture est modélisée par un thread (classe ***Cars***). Chaque voiture essaye d'entrer dans le parking en appelant la méthode ***arrive*** qui la met en attente si aucune place n'est disponible. Elle s'en va en appelant la méthode ***departs***. La méthode principale est définie dans la classe ***MainCarPark***. Elle crée un parking (c'est à dire une instance de la classe ***CarPark***) puis 20 voitures (en espaçant les créations de 0 à 5 secondes).

1) Compléter les méthodes ***arrive*** et ***depart*** de la classe ***CarPark*** sans utiliser les sémaphores. La classe ***Cars*** est fournie, ainsi que la classe principale.

```
public class Cars implements Runnable {
      private CarPark carpark;
      private Random r;
      private String nom;
      public Cars(String nom, CarPark carpark) {
            this.nom = nom;
            this.carpark = carpark;
            r = new Random();
      }
      public void run() {
            Thread.currentThread().setName(nom);
            carpark.arrive();
```

```
            try {
                  Thread.sleep(r.nextInt(10) * 1000);
            } catch (InterruptedException e) {
                  return;
            }
            carpark.depart();
      }
}

public class CarPark {
      private int capacity;
      public CarPark(int capacity) {
            this.capacity = capacity;
      }

      public synchronized void arrive() {
            System.out.println(Thread.currentThread().getName() + " arrive");
            // completer...
            System.out.println(Thread.currentThread().getName() + " attend...");
            // completer...
            System.out.println(Thread.currentThread().getName()  +  "  entre  dans
le parking [il reste " + capacity + " place(s)");
      }

      public synchronized void depart() {
            System.out.println(Thread.currentThread().getName() + " repart");
            // completer...
      }
}

public class MainCarPark {

    public static void main(String args[]) {
        CarPark carpark = new CarPark(5);
        Random r = new Random();
        for (int i = 0; i < 20; i++) {
            try {
                Thread.sleep(r.nextInt(5) * 1000);
            } catch (InterruptedException e) {
            }
            new Thread(new Cars("voiture" + i, carpark)).start();
        }
    }
}
```

2) Refaire la même chose en utilisant les sémaphores.