TD: threads
Correction

Exercice 1. Compteurs

Un compteur a un nom (une chaîne de caractère) et compte de 0 à un entier n en marquant une pause aléatoire (Math.random() renvoit un flottant entre 0 et 1) entre chaque itération.

a. Écrire la classe compteur qui affiche chaque nombre compté, et un message final sous la forme `Le compteur Toto a fini de compter jusqu'à 12`. Écrire aussi un programme lançant une instance illustrant l'utilisation de plusieurs compteurs simultanément.

Correction

```
class Compteur extends Thread {

    public Compteur(String nom, int n) {
        this.nom = nom;
        this.n = n;
    }

    public void run() {
        for (int i = 0; i <= n; ++i) {
            System.out.println(nom + ": " + i);
            try {
                // Le thread se met en pause pour une durée aléatoire
                Thread.sleep((long)(Math.random()*1000));
            } catch (InterruptedException e) {}
        }
        System.out.println(String.format("%s: fini de compter jusqu'à %d", nom, n));
    }

    private String nom;
    private int n;
}

public class Ex1 {
        public static void main(String args[]) {
        // Création de deux compteurs
        Compteur Compteur1 = new Compteur("Compteur1", 10);
        Compteur Compteur2 = new Compteur("Compteur2", 10);
        // On lance les deux compteurs via la méthode start héritée de Thread
        Compteur1.start();
        Compteur2.start();

        return;
        }
}
```

b. Modifier cette classe pour que chaque compteur affiche son ordre d'arrivée au bout du décompte. `Le compteur CompteurX a fini de compter jusqu'à 12 en position 3`.

Correction
On ajoute un attribut statique `pos` qui compte s'incrémente à chaque fois qu'un compteur a terminé de compter.

```
class Compteur extends Thread {

    public Compteur(String nom, int n) {
        this.nom = nom;
        this.n = n;
    }

    public void run() {
        for (int i = 0; i <= n; i++) {
            System.out.println(nom + ": " + i);
            try {
                Thread.sleep((long)(Math.random()*1000));
            } catch (InterruptedException e) {}
        }
        pos += 1;
        System.out.println(String.format("%s: fini de compter jusqu'à %d
        en position %d", nom, n, pos));
    }

    private String nom;
    private int n;
    private static int pos = 0;
}
```

c. Est-ce que vous pouvez garantir que votre programme s'exécutera proprement ? Donner un exemple de scénario où le comportement est non conforme.

d. Proposer une modification de votre code permettant d'éviter ce problème en utilisant les attributs `synchronized`

Correction

```
class Compteur extends Thread {

    public Compteur(String nom, int n) {
        this.nom = nom;
        this.n = n;
    }

    public void run() {
        for (int i = 0; i <= n; i++) {
            System.out.println(nom + ": " + i);
            try {
                Thread.sleep((long)(Math.random()*1000));
            } catch (InterruptedException e) {}
        }
        printPos(this);
    }

    public static synchronized void printPos(Compteur compteur) {
        System.out.println(String.format("%s: fini de compter", compteur.nom));
        pos += 1;
        try {
            Thread.sleep((long)(5000));
        } catch (InterruptedException e){}
        System.out.println(String.format("%s: fini de compter jusqu'à %d
        en position %d", compteur.nom, compteur.n, compteur.pos));
    }
```

```
    private String nom;
    private int n;
    private static int pos = 0;
}
```

Exercice 2. Recherche de nombres premiers

On souhaite écrire un programme recherchant un nombre premier ayant au moins un certain nombre b de bits. Pour cela, on va énumérer les nombres entiers impairs plus grands que 2^b et tester leur primalité en testant s'il sont divisible par tous les entiers inférieurs à leur racine carré (algorithme peu efficace).

On souhaite par ailleurs exploiter le parallélisme de la machine hôte en utilisant des threads : on lancera ainsi un nombre fixe de threads testant chacun un intervalle de nombre.

Enfin, il faut que dès qu'un thread a trouvé un nombre premier, il le signale aux autres et les fasse s'arrêter.

Écrire ce programme, en le structurant en classes utilisant les Threads Java.

Correction

```
class PrimR extends Thread {
        public PrimR(long lo, long step, int numT) {
        this.lo = lo; this.step = step; this.numT = numT;
        tA = new PrimT[numT];
        doneT = new boolean[numT];
        }

        public void run() {
                for (int i = 0; i < numT; i++) {
                        doneT[i] = false;
                        tA[i] = new PrimT(lo, lo + step, i, this);
                        lo += step;
                        tA[i].start();
                }
                while (!found) {
                        try {
                                sleep(1000000); // should be infinite
                        } catch (InterruptedException e) {
                                if (!found) {
                                        for (int i = 0; i < numT; i++) {
                                                if (CheckUnset(i)) {
                                                        tA[i] = new PrimT(lo, lo + step, i, this);
                                                        lo += step;
                                                        tA[i].start();
                                                }
                                        }
                                }
                        }
                }
        }

        public synchronized void foundPrime(long i) {
                if (!found) {
                        found = true;
                        System.out.println("Found a prime "+ i);
                        for (int j = 0; j < numT; j++) {
                                tA[j].interrupt();
                        }
                }
        }

        public synchronized void done(int i) {
                doneT[i] = true;
```

```
        }

        public synchronized boolean CheckUnset(int i) {
                if (doneT[i]) {
                        doneT[i] = false;
                        return true;
                }
                return false;
        }

        private long lo;
        private long step;
        private int numT;
        private boolean found;
        private PrimT tA[];
        private boolean doneT[];
}

class PrimT extends Thread {
        public PrimT(long lo, long high, int tid, PrimR objBack) {
                if (lo % 2 == 0)
                        lo += 1;
                this.lo = lo; this.high = high; this.tid = tid; this.objBack = objBack;
        }

        private static boolean isPrime(long num) {
                long sqrt = (long)Math.ceil(Math.sqrt(num));
                for (long i = 3; i <= sqrt; i++) {
                        if (num % i == 0)
                        return false;
                }
                return true;
        }

        public void run() {
                for (long i = lo; i < high; i+=2) {
                        if (isPrime(i)) {
                                objBack.foundPrime(i);
                                objBack.interrupt();
                                return;
                        }
                        if (Thread.interrupted()) {
                                return;
                        }
                }
                objBack.done(tid);
                objBack.interrupt();
        }

        private long lo;
        private long high;
        private int tid;
        PrimR objBack;
}


public class Ex2 {
        public static void main(String args[]) {
                PrimR ps = new PrimR(1000000000051, 1, 2);
                ps.start();

                return;
        }
}
```