TP2 : Thread, Runnable, join, synchronized

Exercice 1 - Hello Thread

1. Un **Runnable** sert à executer un code sur des threads.

2. ```
   public class HelloThread {
       private static Runnable code() {
           return () -> {
               for (var i = 0; i < 5000; i++) {
                   System.out.println("thread " + i);
               }
           };
       }

       private static void addThreads(ArrayList<Thread> threads, int nbThread) {
           for (var i = 0; i < nbThread; i++) {
               threads.add(new Thread(code()));
               threads.get(i).start();
           }
       }

       public static void main(String[] args) {
           final int nbThread = 4;

           var threads = new ArrayList<Thread>();

           addThreads(threads, nbThread);
       }
   }
   ```

3. On remarque que les nombres sont bien ordonnée. Ce n'est pas normal car ça devrait être non ordonnée.

4. ```
   public class HelloThread {
       private static Runnable code(int thread) {
           return () -> {
               for (var i = 0; i < 5000; i++) {
                   System.out.println("thread " + thread + " " + i);
               }
           };
       }

       private static void addThreads(ArrayList<Thread> threads, int nbThread) {
           for (var i = 0; i < nbThread; i++) {
               threads.add(new Thread(code(i)));
               threads.get(i).start();
           }
       }

       public static void main(String[] args) {
           final int nbThread = 4;
   ```

```
        var threads = new ArrayList<Thread>();

        addThreads(threads, nbThread);
    }
}
```

Exercice 2 - This is the end, my friend ...

1.
```
public class HelloThreadJoin {
    private static Runnable code(int threadNum) {
        return () -> {
            for (var i = 0; i < 5000; i++) {
                System.out.println("thread " + threadNum + " " + i);
            }
        };
    }

    private static void addThreads(ArrayList<Thread> threads, int nbThread) {
        for (var i = 0; i < nbThread; i++) {
            threads.add(new Thread(code(i)));
            threads.get(i).start();
        }
    }

    private static void deathThreads(ArrayList<Thread> threads) throws
InterruptedException {
        for (Thread thread: threads) {
            thread.join();
        }
    }

    public static void main(String[] args) throws InterruptedException {
        final int nbThread = 4;

        var threads = new ArrayList<Thread>();

        addThreads(threads, nbThread);
        deathThreads(threads);

        System.out.println("Le programme est fini");
    }
}
```

Exercice 3 - When things add up

1.
```
public class HelloListBug {
    private static Runnable code(ArrayList<Integer> list) {
```

```
        return () -> {
            for (var i = 0; i < 5000; i++) {
                list.add(i);
            }
        };
    }

    private static void addThreads(ArrayList<Integer> list, ArrayList<Thread>
threads, int nbThread) {
        for (var i = 0; i < nbThread; i++) {
            threads.add(new Thread(code(list)));
            threads.get(i).start();
        }
    }

    private static void deathThreads(ArrayList<Thread> threads) throws
InterruptedException {
        for (Thread thread: threads) {
            thread.join();
        }
    }

    public static void main(String[] args) throws InterruptedException {
        final int nbThread = 4;

        var threads = new ArrayList<Thread>();
        var list = new ArrayList<Integer>(5000 * nbThread);

        addThreads(list, threads, nbThread);
        deathThreads(threads);

        System.out.println(list.size());
    }
}
```

2. L'incrémentation de la taille n'est pas synchronisé.

3. On obtient une exception **"index ouf of range"** car étant donnée que le compteur qui défini la taille de la liste n'est incrémenté de 1 par appel à la méthode **add** même lorsque plusieurs **threads** tentent d'y accéder en même temps.

4.
```
private static final Object monitor = new Object();

private static Runnable code(ArrayList<Integer> list) {
    return () -> {
        for (var i = 0; i < 5000; i++) {
            synchronized(monitor) {
                list.add(i);
            }
        }
    };
}
```

5.
```
 public class ThreadSafeList {
     private final List<Integer> list = new ArrayList<>();

     public void add(int thread) {
         synchronized(list) {
             list.add(thread);
         }
     }

     public int size() {
         synchronized(list) {
             return list.size();
         }
     }
 }

 public class HelloListBug {
     private static Runnable code(ThreadSafeList threads) {
         return () -> {
             for (var i = 0; i < 5000; i++) {
                 threads.add(i);
             }
         };
     }

     private static void addThreads(ThreadSafeList threadSafeList,
ArrayList<Thread> threads, int nbThread) {
         for (var i = 0; i < nbThread; i++) {
             threads.add(new Thread(code(threadSafeList)));
             threads.get(i).start();
         }
     }

     private static void deathThreads(ArrayList<Thread> threads) throws
InterruptedException {
         for (Thread thread: threads) {
             thread.join();
         }
     }

     public static void main(String[] args) throws InterruptedException {
         final int nbThread = 4;

         var threads = new ArrayList<Thread>();
         var threadSafeList = new ThreadSafeList();

         addThreads(threadSafeList, threads, nbThread);
         deathThreads(threads);

         System.out.println(threadSafeList.size());
     }
```

```
}
```

6. Une classe **threadsafe** est une classe qui peut être utiliser par plusieurs threads sans avoir d'états incohérents.

Steve Chen M1 Info 16/10/2021