S14 - Optimisation de codes

0. PRÉPARATION. Créer un projet Eclipse et un **package `tsp`**, importer `TSP0.java`, `TSP.java` `TSPPoint.java` et `TSPTest.java`. Importer `tspobf.jar` puis l'ajouter dans les ressources du projet : Project→Properties→BuildPath→Add Jar...

1. Etudier le programme TSP mis à disposition. Essayer d'optimiser les performances de ce programme avec les moyens vus au cours (ou d'autres !). *Ne jamais modifier la spécification de la méthode* `salesman()`. Expliquer chaque optimisation envisagée. Sauver les versions successives dans des classes TSP11, TSP12, TSP13, ...

2. Tester sur machine l'exercice 1. Un programme principal est fourni. Mesurer les gains de performance de chacune des modifications **sans JIT** (`java -Xint ...`), et exprimer les *gains relatifs d'une version à la suivante*. Essayer de faire mieux que les versions (TSP113, 120, 170, 177) données sous forme compilée ! Assurez-vous que les mesures sont fiables (travailler si possible sur une machine "inactive", mesurer des durées suffisamment longues (> 2[s]), lancer plusieurs fois le même algorithme, ...).
   Finalement, par curiosité, mesurer/discuter les performances cette fois **avec JIT**.

   FACULTATIF : Mesurer les temps de calcul ***sérieusement*** avec JMH (cf. TspViaJmh.java).

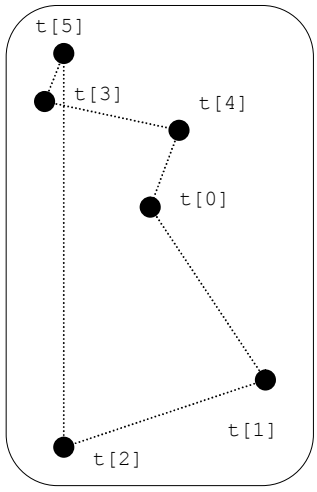
3. a) FACULTATIF : Ecrire une variante multithread et mesurer le gain.
   b) FACULTATIF : Analyser avec un profileur la version de base et la version optimisée.
   c) FACULTATIF : Lire un article sur le *Java benchmarking* (cf. lien sur Moodle).
   d) FACULTATIF : Outre le code, qu'est-ce qui influence les temps d'exécution mesurés ?
   e) FACULTATIF : Ecrire une variante en C (via JNI) et mesurer le gain.
   f) FACULTATIF : Expliquer pourquoi les résultats d'un profileur doivent parfois être considérés avec prudence, et ce qui caractérise le "Honest Java Profiler" de R. Warburton.

COMPLÉMENT JIT : les curieux trouveront peut-être de l'intérêt aux options suivantes de la JVM :
`-XX:+PrintCompilation -XX:+PrintInlining` (avec `-XX:+UnlockDiagnosticVMOptions`)

```java
package tsp;
public class TSP0 implements TSP {
  //------------------------------------------------------------
  public void salesman(TSPPoint[] t, int[] path) {
    int i, j;
    int n = t.length;
    boolean[] visited = new boolean[n];
    int thisPt, closestPt = 0;
    double shortestDist;

    thisPt = n-1;
    visited[thisPt] = true;
    path[0] = n-1;  // choose the starting city
    for(i=1; i<n; i++) {
      shortestDist = Double.MAX_VALUE;
      for(j=0; j<n; j++) {
        if (visited[j])
          continue;
        if (distance(t[thisPt], t[j]) < shortestDist ) {
          shortestDist = distance(t[thisPt], t[j]);
          closestPt = j;
        }
      }
      path[i] = closestPt;
      visited[closestPt] = true;
      thisPt = closestPt;
    }
  }
  //------------------------------------------------------------
  static private double sqr(double a) {
    return a*a;
  }
  //------------------------------------------------------------
  static private double distance(TSPPoint p1, TSPPoint p2) {
    return Math.sqrt(sqr(p1.x-p2.x) + sqr(p1.y-p2.y));
  }
}
```

```java
package tsp;
public class TSPPoint {public double x; public double y;}
```




```
path[0] == 5
path[1] == 3
path[2] == 4
path[3] == 0
path[4] == 1
path[5] == 2
```

```java
package tsp;
public interface TSP {
  void salesman(TSPPoint[] t,
                int[] path);
}
```