```java
import java.util.Random;

public class Etat {

    public int dimEtat;
    public int[] vecteur;

    private static Random generateur = new Random();
    private int oldIndexI;
    private int oldIndexJ;
    private double oldPoids;

    /*****************************************************************************************/
    /* Methodes locales */
    /*****************************************************************************************/
    public void exchange(int I, int J) {
        int buffer;
        buffer = vecteur[I];
        vecteur[I] = vecteur[J];
        vecteur[J] = buffer;
    }

    /*****************************************************************************************/
    /* Constructeur */
    /*****************************************************************************************/
    public Etat(int dimEtat) {
        this.dimEtat = dimEtat;
        vecteur = new int[dimEtat];
    }

    /*****************************************************************************************/
    /* Initialisation aleatoire de l'etat */
    /*****************************************************************************************/
    public void initAleatEtat() {
        int ranIndex;
        int temp;
        for (int i = 0; i < dimEtat; ++i) {
            // vecteur[i] = generateur.nextInt(1); ex1
            vecteur[i] = i;
        }
        /*
```

```
42          * for (int i = 0; i < dimEtat; ++i) { ranIndex =
  generateur.nextInt(dimEtat-1);
43          * exchange(ranIndex, i); }
44          */
45      }
46
47      /
  ************************************************************************
  ***************************************/
48      /* Affichage */
49      /
  ************************************************************************
  ***************************************/
50
51      public String afficherEtat() {
52          return ""; // "Poids= " + oldPoids;
53      }
54
55      /
  ************************************************************************
  ***************************************/
56      /* GenererVoisin */
57      /
  ************************************************************************
  ***************************************/
58
59      public void genererVoisin() {
60          /*
61           * ex1 int indexI; indexI = generateur.nextInt(dimEtat);
  vecteur[indexI] =
62           * (vecteur[indexI] + 1)%2; oldIndexI = indexI;
63           */
64          int tempMin, tempMax;
65          int indexI = generateur.nextInt(dimEtat);
66          int indexJ = generateur.nextInt(dimEtat);
67          // exchange(indexI, indexJ); ***1er methode pour changer la
  position
68          // aleatoirement
69          for (int i = 0; i < Math.ceil(Math.abs(indexI - indexJ) / 2); ++i)
  {
70              exchange(Math.min(indexI, indexJ) + i, Math.max(indexI,
  indexJ) - i);
71              // System.out.println(i+ " " + (Math.min(indexI, indexJ)+i) +
  " " +
72              // (Math.max(indexI, indexJ)-i) );
73          }
74
75          oldIndexI = indexI;
76          oldIndexJ = indexJ;
77      }
78
79      /
  ************************************************************************
```

```
   ****************************************/
80     /* Retour a la solution précédente */
81     /
   ****************************************************************************
   ****************************************/
82     public void comeBack() {
83         // vecteur[oldIndexI] = (vecteur[oldIndexI] + 1)%2; ***ex1
84         // exchange(oldIndexI, oldIndexJ); **1er methode pour changer la
   position
85         // aleatoirement
86         for (int i = 0; i < Math.ceil(Math.abs(oldIndexI - oldIndexJ) /
   2); ++i) {
87             exchange(Math.min(oldIndexI, oldIndexJ) + i,
   Math.max(oldIndexI, oldIndexJ) - i);
88         }
89     }
90
91     /
   ****************************************************************************
   ****************************************/
92     /* Evaluation des objectifs */
93     /
   ****************************************************************************
   ****************************************/
94     public double calculCritere() {
95
96         double cost = 0;
97         double poids = 0;
98         double dx, dy;
99
100        /*
101         * ex1 for (int i=0; i<dimEtat; ++i) cost += vecteur[i]; oldPoids
   = cost; return
102         * cost;
103         */
104
105        /*
106         * ex2 for (int i=0; i<dimEtat; ++i){ cost +=
107         * vecteur[i]*Data.tabValeurs[vecteur[i]]; poids +=
108         * vecteur[i]*Data.tabPoids[vecteur[i]]; } if (poids <= 2000)
   { oldPoids = poids;
109         * return cost; } else{ return 0; }
110         */
111
112        for (int i = 0; i < dimEtat - 1; ++i) {
113            dx = Data.tabVilles[vecteur[i + 1]][0] -
   Data.tabVilles[vecteur[i]][0];
114            dy = Data.tabVilles[vecteur[i + 1]][1] -
   Data.tabVilles[vecteur[i]][1];
115            cost += Math.sqrt(dx * dx + dy * dy);
116        }
117        dx = Data.tabVilles[vecteur[dimEtat - 1]][0] -
   Data.tabVilles[vecteur[0]][0];
```

```
118         dy = Data.tabVilles[vecteur[dimEtat - 1]][1] -
    Data.tabVilles[vecteur[0]][1];
119         cost += Math.sqrt(dx * dx + dy * dy);
120 
121         return cost;
122 
123         /*
124          *
125          *
126          *
127          *
128          */
129     }
130 
131 }
132 
```