```java
package Serveur.maths.vectors;

/**
 * Aucune fonction ne change l'état du vecteur. Les fonctions de transformations
 * renvoient un nouveau vecteur. Cette classe travaille avec des double. Cette
 * classe agit à la fois comme un vecteur et comme un point.
 */
public class Vector3d implements Cloneable {
    public double x;
    public double y;
    public double z;

    /**
     * Créé un vecteur nul en 3D.
     */
    public Vector3d() {
        x = 0;
        y = 0;
        z = 0;
    }

    /**
     * Créé un vecteur en 3D.
     */
    public Vector3d(double x, double y, double z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    /**
     * Créé un vecteur aléatoire de norme 1
     */
    public static Vector3d random() {
        Vector3d result = new Vector3d();
        double norm;

        do {
            result.x = Math.random() * 2 - 1;
            result.y = Math.random() * 2 - 1;
            result.z = Math.random() * 2 - 1;
            norm = result.norm();
        } while (norm > 1);

        return new Vector3d(result.x / norm, result.y / norm, result.z / norm);
    }

    /**
     * Créé un vecteur avec des coordonnées sphériques.
     */
    public static Vector3d spherique(double norm, double theta, double phi) {
        double sinPhi = Math.sin(phi);

        return new Vector3d(norm * sinPhi * Math.cos(theta),
                norm * sinPhi * Math.sin(theta), norm * Math.cos(phi));
    }

    /**
     * @return La norme du vecteur.
     */
    public double norm() {
        return Math.sqrt(x * x + y * y + z * z);
```

```
63      }
64
65      /**
66       * @return La norme du vecteur au carré.
67       */
68      public double normSquared() {
69          return x * x + y * y + z * z;
70      }
71
72      /**
73       * @return La distance par rapport à un point.
74       */
75      public double distance(double x, double y) {
76          return Math.sqrt((this.x - x) * (this.x - x)
77                  + (this.y - y) * (this.y - y) + (this.z - z) * (this.z - z));
78      }
79
80      /**
81       * @return La distance par rapport à un point.
82       */
83      public double distance(final Vector3d v) {
84          return Math.sqrt((x - v.x) * (x - v.x) + (y - v.y) * (y - v.y)
85                  + (z - v.z) * (z - v.z));
86      }
87
88      /**
89       * @return La distance au carré par rapport à un point.
90       */
91      public double distanceSquared(double x, double y) {
92          return (this.x - x) * (this.x - x) + (this.y - y) * (this.y - y)
93                  + (this.z - z) * (this.z - z);
94      }
95
96      /**
97       * @return La distance au carré par rapport à un point.
98       */
99      public double distanceSquared(final Vector3d v) {
100         return (x - v.x) * (x - v.x) + (y - v.y) * (y - v.y)
101                 + (z - v.z) * (z - v.z);
102     }
103
104     /**
105      * @return Le produit scalaire avec le vecteur (x, y, z).
106      */
107     public double dotProduct(double x, double y, double z) {
108         return this.x * x + this.y * y + this.z * z;
109     }
110
111     /**
112      * @return Le produit scalaire avec le vecteur v.
113      */
114     public double dotProduct(final Vector3d v) {
115         return x * v.x + y * v.y + z * v.z;
116     }
117
118     /**
119      * Ajoute un autre vecteur.
120      */
121     public Vector3d add(double x, double y) {
122         return new Vector3d(this.x + x, this.y + y, this.z + z);
123     }
124
```

```java
    /**
     * Ajoute un autre vecteur.
     */
    public Vector3d add(final Vector3d v) {
        return new Vector3d(x + v.x, y + v.y, z + v.z);
    }

    /**
     * Soustrait un autre vecteur.
     */
    public Vector3d sub(double x, double y, double z) {
        return new Vector3d(this.x - x, this.y - y, this.z - z);
    }

    /**
     * Soustrait un autre vecteur.
     */
    public Vector3d sub(final Vector3d v) {
        return new Vector3d(x - v.x, y - v.y, z - v.z);
    }

    /**
     * Multiplie par un scalaire.
     */
    public Vector3d mult(double k) {
        return new Vector3d(x * k, y * k, z * k);
    }

    /**
     * Divise par un scalaire.
     */
    public Vector3d div(double k) {
        return new Vector3d(x / k, y / k, z / k);
    }

    /**
     * Applique une homothétie.
     *
     * @param k
     *            : Le rapport de l'homothétie.
     * @param origine
     *            : Le point de l'origine de l'homothétie.
     * @return Le nouveau vecteur auquel on a appliqué l'homothétie.
     */
    public Vector3d homothetie(double k, final Vector3d origine) {
        return new Vector3d((x - origine.x) * k + origine.x,
                (y - origine.y) * k + origine.y,
                (z - origine.z) * k + origine.z);
    }

    /**
     * @return Un nouveau vecteur de même sens mais de norme 1.
     */
    public Vector3d normalize() {
        double norm = norm();
        return new Vector3d(x / norm, y / norm, z / norm);
    }

    /**
     * @return Un nouveau vecteur identique.
     */
    public Vector3d clone() {
```

```
187         return new Vector3d(x, y, z);
188     }
189 
190     /**
191      * @return Une chaine indiquant les composantes du vecteur en ligne.
192      */
193     public String toString() {
194         return "(" + x + ", " + y + ", " + z + ")";
195     }
196 
197     /**
198      * @return Une chaine indiquant les composantes du vecteur en colonne plutot
199      *         qu'en ligne.
200      */
201     public String toStringVertical() {
202         return "[" + x + "]\n[" + y + "]\n[" + z + "]";
203     }
204 }
```