```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 Vector2d implements Cloneable {
    public double x;
    public double y;

    /**
     * Créé un vecteur nul en 2D.
     */
    public Vector2d() {
        x = 0;
        y = 0;
    }

    /**
     * Créé un vecteur en 2D.
     */
    public Vector2d(double x, double y) {
        this.x = x;
        this.y = y;
    }

    /**
     * Créé un vecteur avec des coordonnées polaire.
     */
    public static Vector2d polaire(double norm, double angle) {
        return new Vector2d(norm * Math.cos(angle), norm * Math.sin(angle));
    }

    /**
     * Créé un vecteur aléatoire de norme 1
     *
     * @return Le vecteur créé.
     */
    public static Vector2d random() {
        double angle = Math.random() * 2 * Math.PI;

        return new Vector2d(Math.cos(angle), Math.sin(angle));
    }

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

    /**
     * @return La norme du vecteur au carré.
     */
    public double normSquared() {
        return x * x + y * y;
    }

    /**
     * @return La distance par rapport à un point.
     */
```

```java
    public double distance(double x, double y) {
        return Math.sqrt(
                (this.x - x) * (this.x - x) + (this.y - y) * (this.y - y));
    }

    /**
     * @return La distance par rapport à un point.
     */
    public double distance(final Vector2d v) {
        return Math.sqrt((x - v.x) * (x - v.x) + (y - v.y) * (y - v.y));
    }

    /**
     * @return La distance au carré par rapport à un point.
     */
    public double distanceSquared(double x, double y) {
        return (this.x - x) * (this.x - x) + (this.y - y) * (this.y - y);
    }

    /**
     * @return La distance au carré par rapport à un point.
     */
    public double distanceSquared(final Vector2d v) {
        return (x - v.x) * (x - v.x) + (y - v.y) * (y - v.y);
    }

    /**
     * @return Le nouveau vecteur de même sens mais de norme 1.
     */
    public Vector2d normalize() {
        double norm = norm();
        return new Vector2d(x / norm, y / norm);
    }

    /**
     * @return Le produit scalaire avec le vecteur (x, y).
     */
    public double dotProduct(double x, double y) {
        return this.x * x + this.y * y;
    }

    /**
     * @return Le produit scalaire avec le vecteur v.
     */
    public double dotProduct(final Vector2d v) {
        return x * v.x + y * v.y;
    }

    /**
     * Ajoute un autre vecteur.
     */
    public Vector2d add(double x, double y) {
        return new Vector2d(this.x + x, this.y + y);
    }

    /**
     * Ajoute un autre vecteur.
     */
    public Vector2d add(final Vector2d v) {
        return new Vector2d(x + v.x, y + v.y);
    }

```

```java
    /**
     * Soustrait un autre vecteur.
     */
    public Vector2d sub(double x, double y) {
        return new Vector2d(this.x - x, this.y - y);
    }

    /**
     * Soustrait un autre vecteur.
     */
    public Vector2d sub(final Vector2d v) {
        return new Vector2d(x - v.x, y - v.y);
    }

    /**
     * Multiplie par un scalaire.
     *
     * @param k
     *            : Le coefficient multiplicateur.
     * @return Le nouveau vecteur.
     */
    public Vector2d mult(double k) {
        return new Vector2d(x * k, y * k);
    }

    /**
     * Divise par un scalaire.
     */
    public Vector2d div(double k) {
        return new Vector2d(x / k, y / 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 Vector2d homothetie(double k, final Vector2d origine) {
        return new Vector2d((x - origine.x) * k + origine.x,
                (y - origine.y) * k + origine.y);
    }

    /**
     * Applique une rotation, d'origine (0, 0).
     *
     * @param angle
     *            : L'angle de rotation en radians.
     * @return Le nouveau vecteur auquel on a appliqué la rotation.
     */
    public Vector2d rotate(double angle) {
        double cos = Math.cos(angle), sin = Math.sin(angle);
        return new Vector2d(x * cos - y * sin, x * sin + y * cos);
    }

    /**
     * Applique une rotation.
     *
     * @param angle
```

```
187      *            : L'angle de rotation en radians.
188      * @param origine
189      *            : Le point de l'origine de la rotation.
190      * @return Le nouveau vecteur auquel on a appliqué la rotation.
191      */
192     public Vector2d rotate(double angle, final Vector2d origine) {
193         double cos = Math.cos(angle), sin = Math.sin(angle);
194         return new Vector2d(
195                 (x - origine.x) * cos - (y - origine.y) * sin + origine.x,
196                 (x - origine.x) * sin + (y - origine.y) * cos + origine.y);
197     }
198 
199     /**
200      * @return Un nouveau vecteur identique.
201      */
202     public Vector2d clone() {
203         return new Vector2d(x, y);
204     }
205 
206     /**
207      * @return Une chaine indiquant les composantes du vecteur en ligne.
208      */
209     public String toString() {
210         return "(" + x + ", " + y + ")";
211     }
212 
213     /**
214      * @return Une chaine indiquant les composantes du vecteur en colonne plutot
215      *         qu'en ligne.
216      */
217     public String toStringVertical() {
218         return "[" + x + "]\n[" + y + "]";
219     }
220 }
```