

Vector2d.java

```
1 package Serveur.maths.vectors;
2
3 /**
4  * Aucune fonction ne change l'état du vecteur. Les fonctions de transformations
5  * renvoient un nouveau vecteur. Cette classe travaille avec des double. Cette
6  * classe agit à la fois comme un vecteur et comme un point.
7  */
8 public class Vector2d implements Cloneable {
9     public double x;
10    public double y;
11
12    /**
13     * Créé un vecteur nul en 2D.
14     */
15    public Vector2d() {
16        x = 0;
17        y = 0;
18    }
19
20    /**
21     * Créé un vecteur en 2D.
22     */
23    public Vector2d(double x, double y) {
24        this.x = x;
25        this.y = y;
26    }
27
28    /**
29     * Créé un vecteur avec des coordonnées polaire.
30     */
31    public static Vector2d polaire(double norm, double angle) {
32        return new Vector2d(norm * Math.cos(angle), norm * Math.sin(angle));
33    }
34
35    /**
36     * Créé un vecteur aléatoire de norme 1
37     *
38     * @return Le vecteur créé.
39     */
40    public static Vector2d random() {
41        double angle = Math.random() * 2 * Math.PI;
42
43        return new Vector2d(Math.cos(angle), Math.sin(angle));
44    }
45
46    /**
47     * @return La norme du vecteur.
48     */
49    public double norm() {
50        return Math.sqrt(x * x + y * y);
51    }
52
53    /**
54     * @return La norme du vecteur au carré.
55     */
56    public double normSquared() {
57        return x * x + y * y;
58    }
59
60    /**
61     * @return La distance par rapport à un point.
62     */
63 }
```

Vector2d.java

```
63 public double distance(double x, double y) {
64     return Math.sqrt(
65         (this.x - x) * (this.x - x) + (this.y - y) * (this.y - y));
66 }
67
68 /**
69  * @return La distance par rapport à un point.
70  */
71 public double distance(final Vector2d v) {
72     return Math.sqrt((x - v.x) * (x - v.x) + (y - v.y) * (y - v.y));
73 }
74
75 /**
76  * @return La distance au carré par rapport à un point.
77  */
78 public double distanceSquared(double x, double y) {
79     return (this.x - x) * (this.x - x) + (this.y - y) * (this.y - y);
80 }
81
82 /**
83  * @return La distance au carré par rapport à un point.
84  */
85 public double distanceSquared(final Vector2d v) {
86     return (x - v.x) * (x - v.x) + (y - v.y) * (y - v.y);
87 }
88
89 /**
90  * @return Le nouveau vecteur de même sens mais de norme 1.
91  */
92 public Vector2d normalize() {
93     double norm = norm();
94     return new Vector2d(x / norm, y / norm);
95 }
96
97 /**
98  * @return Le produit scalaire avec le vecteur (x, y).
99  */
100 public double dotProduct(double x, double y) {
101     return this.x * x + this.y * y;
102 }
103
104 /**
105  * @return Le produit scalaire avec le vecteur v.
106  */
107 public double dotProduct(final Vector2d v) {
108     return x * v.x + y * v.y;
109 }
110
111 /**
112  * Ajoute un autre vecteur.
113  */
114 public Vector2d add(double x, double y) {
115     return new Vector2d(this.x + x, this.y + y);
116 }
117
118 /**
119  * Ajoute un autre vecteur.
120  */
121 public Vector2d add(final Vector2d v) {
122     return new Vector2d(x + v.x, y + v.y);
123 }
124
```

Vector2d.java

```

125  /**
126   * Soustrait un autre vecteur.
127   */
128  public Vector2d sub(double x, double y) {
129      return new Vector2d(this.x - x, this.y - y);
130  }
131
132  /**
133   * Soustrait un autre vecteur.
134   */
135  public Vector2d sub(final Vector2d v) {
136      return new Vector2d(x - v.x, y - v.y);
137  }
138
139  /**
140   * Multiplie par un scalaire.
141   *
142   * @param k
143   *       : Le coefficient multiplicateur.
144   * @return Le nouveau vecteur.
145   */
146  public Vector2d mult(double k) {
147      return new Vector2d(x * k, y * k);
148  }
149
150  /**
151   * Divise par un scalaire.
152   */
153  public Vector2d div(double k) {
154      return new Vector2d(x / k, y / k);
155  }
156
157  /**
158   * Applique une homothétie.
159   *
160   * @param k
161   *       : Le rapport de l'homothétie.
162   * @param origine
163   *       : Le point de l'origine de l'homothétie.
164   * @return Le nouveau vecteur auquel on a appliqué l'homothétie.
165   */
166  public Vector2d homothetie(double k, final Vector2d origine) {
167      return new Vector2d((x - origine.x) * k + origine.x,
168                          (y - origine.y) * k + origine.y);
169  }
170
171  /**
172   * Applique une rotation, d'origine (0, 0).
173   *
174   * @param angle
175   *       : L'angle de rotation en radians.
176   * @return Le nouveau vecteur auquel on a appliqué la rotation.
177   */
178  public Vector2d rotate(double angle) {
179      double cos = Math.cos(angle), sin = Math.sin(angle);
180      return new Vector2d(x * cos - y * sin, x * sin + y * cos);
181  }
182
183  /**
184   * Applique une rotation.
185   *
186   * @param angle

```

Vector2d.java

```
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 }
```