Design Pattern
TP nº 3 : De la POO classique

Le but de ce TP est de vous montrer que les principes de la POO peuvent se transposer en JavaScript. Le but principal est de détecter la collision de deux triangles. Chaque classe devra se situer dans son propre fichier. Il est conseillé de tester régulièrement l'implémentation.

GeometrieErreur

Définissez une classe `GeometrieErreur` qui représente les erreurs que pourra produire la bibliothèque. Il s'agit simplement d'une sous-classe de `Error` sans nouveau champ ou méthode.

Point

Définissez une classe `Point` qui représente un point dans le plan. Elle disposera d'un constructeur à deux paramètres pour les coordonnées. Les coordonnées devront être immuables.

```
let point = new Point(0, 1);
alert(`Point de coordonnées (${point.x}, ${point.y})`);
point.x = 2; // ignoré ou erreur, les coordonnées sont immuables
```

Vecteur

1. Définissez une classe `Vecteur` qui représente un vecteur en deux dimensions. On peut construire un vecteur en lui passant directement les coordonnées ou à partir de deux points. [1]

   ```
   let a = new Point(0, 1);
   let b = new Point(1, 0);
   let v1 = new Vecteur(1, -1); // le vecteur de coordonnées (1, -1)
   let v2 = new Vecteur(a, b); // aussi le vecteur de coordonnées (1, -1)
   // let v3 = new Vecteur(1, b); // erreur
   ```

2. Ajoutez une méthode `determinant` qui renvoie le déterminant du vecteur courant avec le vecteur passé en paramètre. [2]

   ```
   let v1 = new Vecteur(1, 2);
   let v2 = new Vecteur(1, 1);
   alert(v1.determinant(v2)); // affiche -1
   ```

3. Ajoutez une méthode statique `position` qui prend trois points A, B et C en paramètres. La méthode indique la position de C par rapport à la droite (AB) dans le sens de $\overrightarrow{AB}$. Plus précisément, la méthode renvoie :
   — une valeur strictement positive si C est dans le demi-plan gauche,
   — une valeur strictement négative si C est dans le demi-plan droit,
   — `0` si les points A, B et C sont alignés.
   Pour implémenter cette méthode, il suffit de renvoyer le déterminant de $\overrightarrow{AB}$ avec $\overrightarrow{AC}$.

   ```
   let a = new Point(0, 0);
   let b = new Point(1, 0);
   let c = new Point(0, 1);
   alert(Vecteur.position(a, b, c) > 0); // true
   alert(Vecteur.position(a, c, b) < 0); // true
   alert(Vecteur.position(a, b, b) == 0); // true
   ```

1. Si les coordonnées des points A et B sont (x_A, y_A) et (x_B, y_B), les coordonnées du vecteur $\overrightarrow{AB}$ sont $(x_B - x_A, y_B - y_A)$.
2. Si les coordonnées des vecteurs $\vec{u}$ et $\vec{v}$ sont (x, y) et (x', y'), alors le déterminant de $\vec{u}$ avec $\vec{v}$ est le nombre $xy' - x'y$.

Segment

1. Définissez une classe `Segment` qui représente un segment. Un segment est défini par ses deux extrémités.
2. Ajoutez une méthode `intersecte` qui indique si le segment courant et le segment passé en paramètre se croisent (on ignore les extrémités). Les segments $[AB]$ et $[CD]$ se croisent si les points A et B sont de part et d'autres de la droite (CD) et si les points C et D sont de part et d'autres de la droite (AB).

```
let a = new Point(1, 0);
let b = new Point(-1, 0);
let c = new Point(0, 1);
let d = new Point(0, -1);

let s1 = new Segment(a, b);
let s2 = new Segment(c, d);
let s3 = new Segment(a, c);
let s4 = new Segment(b, d);

alert(s1.intersecte(s2)); // true
alert(s3.intersecte(s4)); // false
```

Triangle

1. Définissez une classe `Triangle` qui représente un triangle. Un triangle est caractérisé par ses trois sommets.
2. Ajoutez une méthode `estInterieur` qui prend un point en paramètre et indique s'il est à l'intérieur du triangle. Un point D est à l'intérieur d'un triangle ABC s'il est toujours dans le même demi-plan (gauche ou droit) par rapport aux droites (AB), (BC) et (CA) orientées respectivement par les vecteurs $\overrightarrow{AB}$, $\overrightarrow{BC}$ et $\overrightarrow{CA}$. Autrement dit, les valeurs `Vecteur.position(A, B, D)`, `Vecteur.position(B, C, D)` et `Vecteur.position(C, A, D)` doivent être non nulles et de même signe.

```
let a = new Point(1, -1);
let b = new Point(-1, -1);
let c = new Point(0, 2);
let d = new Point(0, 0);
let e = new Point(1, 1);
let t = new Triangle(a, b, c);

alert(t.estInterieur(d)); // true
alert(t.estInterieur(e)); // false
```

3. Ajoutez une méthode `intersecte` qui indique si le triangle courant intersecte le triangle passé en paramètre. C'est le cas si l'une des conditions suivantes est vérifiée :
   — Un sommet d'un triangle est contenu dans l'autre triangle.
   — Un côté d'un triangle intersecte un côté de l'autre triangle.

```
let t1 = new Triangle(new Point(-2, -2), new Point(2, -2), new Point(0, 2));
let t2 = new Triangle(new Point(-2, 2), new Point(2, 2), new Point(0, -2));
let t3 = new Triangle(new Point(-3, 3), new Point(3, 3), new Point(0, -3));
let t4 = new Triangle(new Point(1000, 0), new Point(1001, 0), new Point(1000, 1));

alert(t1.intersecte(t2)); // true
alert(t1.intersecte(t3)); // true
alert(t1.intersecte(t4)); // false
```

Forme

1. Que pensez-vous de la classe `Forme` définie ci-dessous ? Vous rappelle-t-elle un concept connu ?

```
export class Forme {
  constructor() {
    if (new.target == Forme) {
      throw new GeometrieErreur("Forme ne peut être instanciée directement.");
    }
  }

  // Renvoie les triangles composant la forme.
  // Une forme s'approxime toujours par une décomposition en triangles.
  triangles() {
    throw new GeometrieErreur("Pas encore implémentée.");
  }

  // Renvoie true si this et that rentrent en collision ; faux sinon.
  rentreEnCollision(that) {
    let trianglesThis = this.triangles();
    let trianglesThat = that.triangles();
    return trianglesThis.some((t1) =>
      trianglesThat.some((t2) => t1.intersecte(t2))
    );
  }
}
```

2. Modifiez la classe `Triangle` pour qu'elle hérite de `Forme`.

```
let t1 = new Triangle(new Point(-2, -2), new Point(2, -2), new Point(0, 2));
let t2 = new Triangle(new Point(-2, 2), new Point(2, 2), new Point(0, -2));
let t3 = new Triangle(new Point(-3, 3), new Point(3, 3), new Point(0, -3));
let t4 = new Triangle(new Point(1000, 0), new Point(1001, 0), new Point(1000, 1));

alert(t1.rentreEnCollision(t2)); // true
alert(t1.rentreEnCollision(t3)); // true
alert(t1.rentreEnCollision(t4)); // false
```

Pour aller plus loin

La bibliothèque est extensible de plusieurs façons. À vous de voir si vous avez le temps et l'envie de l'améliorer. Suggestions :

— Ajoutez une classe `FormeComposee` qui permet de combiner des formes arbitraires.
— Ajoutez des formes prédéfinies (rectangles, cercles, ...).
— Améliorations des performances en utilisant des boîtes englobantes.
— ...