Développement Web
TP n° 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. On cherche à détecter la collision de deux triangles (pour un moteur graphique par exemple). Pour cela, on développe une petite bibliothèque de géométrie. Chaque classe devra se situer dans son propre fichier et constituera ainsi son propre module. Il est conseillé de tester régulièrement l'implémentation. On rappelle que la page de test et les scripts doivent être servis par le même serveur.

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 immutables.

```
const point = new Point(0, 1);
console.log(`Point de coordonnées (${point.x}, ${point.y})`);
// point.x = 2; // erreur, les coordonnées sont immutables
```

Vecteur

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

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

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

   ```
   const v1 = new Vecteur(1, 2);
   const v2 = new Vecteur(1, 1);
   console.log(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 $\overrightarrow{AC}$.

   ```
   const a = new Point(1, 1);
   const b = new Point(0, 0);
   const c = new Point(1, 0);
   console.log(Vecteur.position(a, b, c) > 0); // true
   console.log(Vecteur.position(b, a, c) < 0); // true
   console.log(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 $xy' - x'y$.

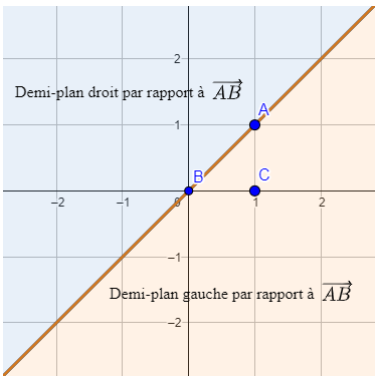


FIGURE 1 – Positions par rapport au vecteur $\overrightarrow{AB}$.

Segment

1. Définissez une classe `Segment` qui représente un segment. Un segment est défini par ses deux extrémités (des points).
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).

```
const a = new Point(0, 0);
const b = new Point(3, 0);
const c = new Point(2, -1);
const d = new Point(2, 1);
const e = new Point(1, 0);

const s1 = new Segment(a, b);
const s2 = new Segment(c, d);
const s3 = new Segment(a, e);

console.log("s1 intersecte s2 :", s1.intersecte(s2));  // true
console.log("s1 intersecte s3 :", s1.intersecte(s3)); // false
```

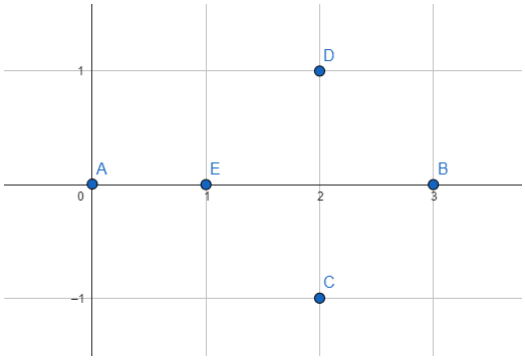


FIGURE 2 – $[CD]$ a une intersection avec $[AB]$, mais pas avec $[AE]$.

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.

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

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

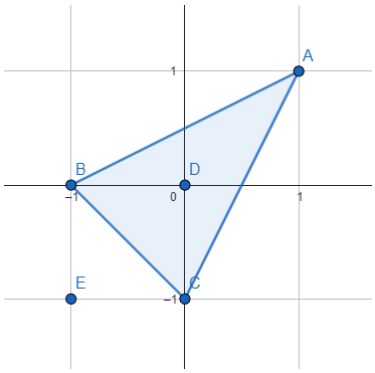
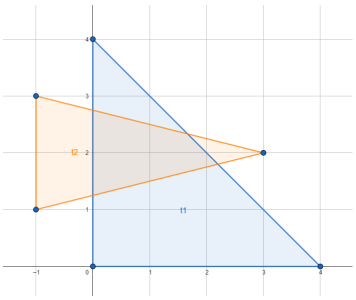


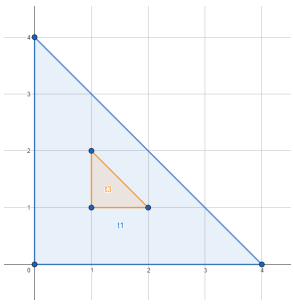
FIGURE 3 – D est à l'intérieur de ABC, car il est toujours à gauche de $\overrightarrow{AB}$, $\overrightarrow{BC}$ et $\overrightarrow{CA}$. E est à l'extérieur de ABC, car il est à gauche de $\overrightarrow{AB}$ et $\overrightarrow{CA}$, mais à droite de $\overrightarrow{BC}$.

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.

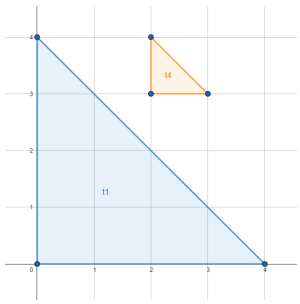
```
const t1 = new Triangle(new Point(0, 0), new Point(4, 0), new Point(0, 4));
const t2 = new Triangle(new Point(-1, 1), new Point(3, 2), new Point(-1, 3));
const t3 = new Triangle(new Point(1, 1), new Point(2, 1), new Point(1, 2));
const t4 = new Triangle(new Point(2, 3), new Point(3, 3), new Point(2, 4));

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



(a) $t1$ et $t2$ s'intersectent.



(b) $t1$ et $t3$ s'intersectent.



(c) $t1$ et $t4$ ne s'intersectent pas.

Forme

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

```
export default class Forme {
  constructor() {
    if (this.constructor === Forme) {
      throw new Error("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 Error("Doit être implémentée dans les classes filles.");
  }

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

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

```
const t1 = new Triangle(new Point(0, 0), new Point(4, 0), new Point(0, 4));
const t2 = new Triangle(new Point(-1, 1), new Point(3, 2), new Point(-1, 3));
const t3 = new Triangle(new Point(1, 1), new Point(2, 1), new Point(1, 2));
const t4 = new Triangle(new Point(2, 3), new Point(3, 3), new Point(2, 4));

console.log(t1.rentreEnCollision(t2)); // true
console.log(t1.rentreEnCollision(t3)); // true
console.log(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.
— ...