Développement orienté objet

TD6 : Héritage

Exercice 1

Indication : quand cela s'applique pensez à utiliser la fonction `super`. Il ne faut pas qu'il y ait de redondance de code.

On dispose des classes suivantes.

```
class Rectangle {
  private _largeur: number;
  private _longueur: number;

  constructor(largeur: number, longueur: number) {
    this._largeur = largeur;
    this._longueur = longueur;
  }

  surface(): number {
    return this._largeur * this._longueur;
  }

  get longueur(): number {
    return this._longueur;
  }

  get largeur(): number {
    return this._largeur;
  }

  set largeur(largeur: number) {
    this.largeur = largeur;
  }

  set longueur(longueur: number) {
    this._longueur = longueur;
  }
}

class Cercle {
  // abscisse du centre
  private _x: number;
  // ordonnée du centre
  private _y: number;
  private _rayon: number;

  constructor(x: number, y: number, rayon: number) {
```

```
    this._x = x;
    this._y = y;
    this._rayon = rayon;
  }

  affiche() {
    console.log("centre = (" + this._x + ", " + this._y + ")");
  }

  get x(): number {
    return this._x;
  }

  get y(): number {
    return this._y;
  }

  get rayon(): number {
    return this._rayon;
  }

  set rayon(rayon: number) {
    this._rayon = rayon;
  }

  fixeCentre(x: number, y: number) {
    this._x = x;
    this._y = y;
  }

  surface(): number {
    return Math.PI * this._rayon * this._rayon;
  }

  estIntérieur(x: number, y: number): boolean {
    return (
      (x - this._x) * (x - this._x) + (y - this._y) * (y - this._y) <= this._rayon *
this._rayon
    );
  }
}
```

1. Donner les diagrammes de classes correspondant.
2. Définissez une classe `RectangleColore` qui hérite de la classe `Rectangle`. Cette classe possède un attribut supplémentaire, `couleur`, qui est représenté par un entier. Le constructeur doit permettre de fournir la longueur, la largeur et la couleur. Testez le code suivant :

```
let r = new RectangleColore(10, 20, 4);
console.log(r.surface());
```

3. Définissez une classe `Figure`. Celle-ci contient :
    - deux attributs `x` et `y` qui représentent le centre de la figure (pour les rectangles, le *centre* représentera le coin supérieur gauche)
    - une méthode `affiche` qui affiche simplement les coordonnées du centre
    - un constructeur prenant les coordonnées du centre en paramètre
4. Faites hériter les classes `Cercle` et `Rectangle` de la classe `Figure`.
5. Testez vos classes avec les instructions suivantes :

```
let r = new RectangleColore(10, 20, 4);
r.affiche();

let c = new Cercle(10, 20, 5);
c.affiche();
```

6. Donnez le diagramme de classes UML final.

Exercice 2

Cet exercice est une extension de l'exercice précédent. On souhaite pouvoir travailler sur un tableau de `Figure` pouvant contenir à la fois des objets de type `Cercle`, des objets de type `Rectangle` et des objets de type `RectangleColore`.

1. Sans le coder dans un premier temps, le programme suivant fonctionne-t'il ?

```
let f = new Array<Figure>();
f.push(new Rectangle(0, 0, 10, 20));
f.push(new Cercle(10, 20, 5));
f.push(new RectangleColore(0, 0, 10, 20, 4));

for (let i = 0; i < f.length; i++) {
  f[i].affiche();
  console.log("La surface est de " + f[i].surface());
}
```

Si non, pourquoi ?

2. Quelle solution proposez-vous ? Implémentez-là et testez-là.
3. Donnez le diagramme de classes UML final.