TP2

Intégration de code et évolutions

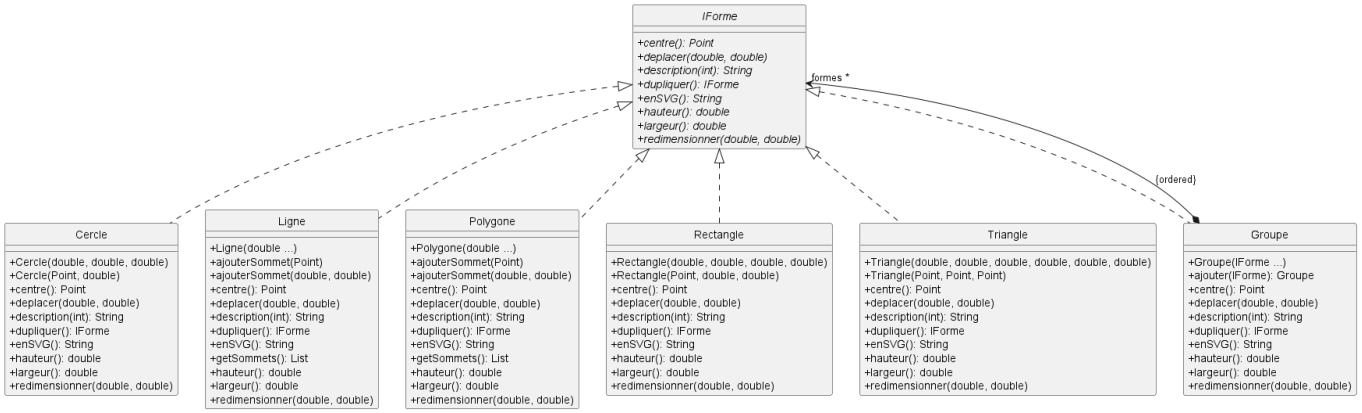
Objectif

L'objectif de ce second TP est de mettre en commun ce que vous avez réalisé au TP1, puis de le faire évoluer en équipe.

Le groupe de TP est divisé par votre encadrant de TP à sa convenance en 2 équipes nommées Alpha et Bravo. Comme en situation réelle, le fait que vous ne puissiez choisir vos co-équipiers fait partie de l'expérience. La suite de l'ensemble des TP de GEN se fera avec cette même structure par équipe. On attend que les membres d'une même équipe s'entre-aident, ce qui ne signifie pas de faire le travail d'un autre à sa place.

Question 1 *Partager le code de votre package* `fr.univrennes.istic.l2gen.geometrie` *avec les autres membres de votre équipe afin que chacun ait sur son poste de travail l'ensemble des classes développées au TP1.*

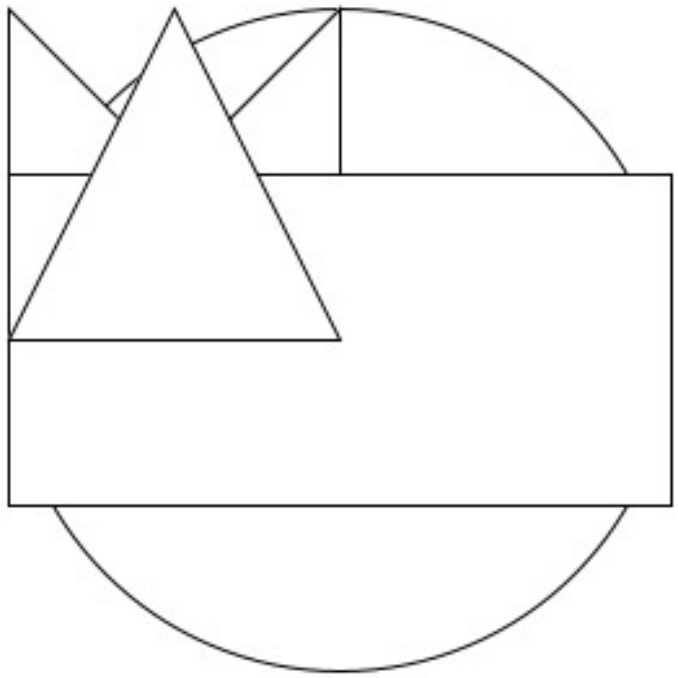
Chacun devra donc avoir un ensemble de classes qui correspond au diagramme ci-dessous :



Ceci doit permettre d'écrire le programme ci-dessous :

```
Groupe tableau = new Groupe();
tableau.ajouter(new Cercle(256, 256, 128));
tableau.ajouter(new Ligne(128, 128, 128, 256, 256, 128, 256, 256));
tableau.ajouter(new Polygone(128, 128, 128, 256, 256, 128, 256, 256));
tableau.ajouter(new Rectangle(256, 256, 256, 128));
tableau.ajouter(new Triangle(192, 128, 256, 128, 256, 256));
System.out.println(tableau.enSVG())
```

Ce programme génère le dessin suivant :



Question 2 *Nous allons maintenant faire évoluer notre logiciel en lui ajoutant les classes et les fonctionnalités ci-dessous. Organisez-vous de manière à vous répartir le travail au sein de votre équipe.*

1. Ajouter une nouvelle classe Texte implémentant IForme telle que

   ```
   IForme f = new Texte(192, 128, 64, "Istic␣L2GEN")
   System.out.println(f.enSVG());
   ```

   affiche le texte :

   ```
   <text x="192.0" y="128.0" font-size="64" text-anchor="middle"
       fill="black" stroke="black">Istic L2GEN</text>
   ```

2. Ajouter dans IForme et toutes ses sous-classes la méthode `colorier(String ... couleurs)` et modifier partout les méthodes `dupliquer()`, `description()` et `enSVG()` pour tenir compte de cette nouvelle fonctionalité. **Rappel :** le paramètre `String ... couleurs` permet d'appeler `colorier` avec un nombre variable d'arguments. Dans le cas des figures simples, on n'utilisera que le premier élément de `couleurs[]` pour réaliser le coloriage, mais pour Groupe on utilisera à tour de rôle chacune des couleurs passées en paramètre pour colorier les éléments du groupe (le 1er élément prend la 1ère couleur, le second la seconde et ainsi de suite jusqu'à épuisement des couleurs, auquel cas on repart de la première couleur).
3. Refaites le programme de la question 1 en ajoutant des couleurs artistiquement choisies et regarder le résultat SVG sur un navigateur.
4. Composez un tableau de style art moderne en combinant esthétiquement les formes et les couleurs disponibles.

Question 3 *L'intégration des codes des différents membres de votre équipe a certainnement conduit à ce qu'il y ait beaucoup de code dupliqué. Nous allons maintenant réorganiser cette base de code en utilisant l'héritage pour factoriser le code commun.*

1. Discutez d'une architecture d'héritage mettant le maximum de code en commun, tout en préservant le principe de substituabilité de Liskov. En particulier évitez l'héritage entre classes concrètes.
2. Dessiner cette nouvelle architecture de votre package geometrie à l'aide d'un diagramme de classes d'UML.
3. Répartissez vous la tâche de modifier votre code Java en conséquence.

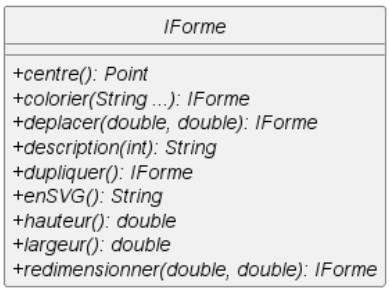
Question 4 Obligatoire pour le parcours défi, optionnelle pour les autres

Une API fluide, ou API fluente (en anglais, "fluent API"), est un style de conception d'interface de programmation qui vise à rendre le code plus lisible et expressif, en permettant une syntaxe proche du langage naturel. L'idée est de créer des chaînes de méthodes (ou d'appels de fonctions) qui peuvent être lues comme une phrase, facilitant ainsi la compréhension du code.

Par exemple on voudrait pouvoir écrire :

```
IForme r = new Rectangle(50,50,100,100).colorier("blue").deplacer(20,20);
```

Pour permettre cela, modifiez l'interface IForme de manière à ce que chaque méthode renvoie `IForme` à la place de `void`.



Répartissez-vous le travail pour mettre à jour l'ensemble des autres classes de manière à ce que chaque méthode "fluente" renvoie l'objet sur lequel elle a été appelée.

Si vous êtes arrivé à terminer ce TP dans les temps, félicitations à votre équipe ! Sinon le prochain CM vous donnera des pistes pour faire face aux difficultés que vous avez probablement rencontrées.