# Afficher graphiquement les points et les segments
# Corrigé

**Objectifs :**

— Utiliser la documentation javadoc
— Utiliser des paquetages
— Comprendre l'intérêt des interfaces comme outil de spécification
— Manipuler les interfaces

**Exercice 1 : Utiliser l'afficheur graphique Écran**
Le paquetage `afficheur`[1] propose une classe `Écran` qui permet de dessiner sur un écran graphique des points, des lignes, des cercles et des textes. Notons que l'`Écran` est une réalisation de l'interface `Afficheur` qui spécifie les méthodes que l'on doit trouver sur tout afficheur.

Écrire un programme (classe `ExempleEcran`) qui définit un écran de dimension 400 pixels en largeur et 250 en hauteur avec une unité de 15 pixels. Le titre de la fenêtre contenant l'écran sera `"ExempleEcran"`. Sur cet écran seront dessinés :

— les axes,
— un point de couleur verte en position (1,2),
— une ligne entre les points (6,2) et (11,9) et de couleur rouge,
— un cercle jaune de centre (4,4) et de rayon 2,5,
— le texte « Premier dessin » en bleu à la position (1, -2).

**Attention :** Dans cet exercice, nous n'utilisons pas les classes `Point` et `Segment`.
**Conseil :** Il faut, comme toujours, compiler et exécuter régulièrement.
**Solution :** Il suffit de lire la documentation de la classe `Ecran` pour savoir quelles méthodes utiliser et connaître les paramètres qu'elles attendent.

**Exercice 2 : Afficher graphiquement le schéma**
Compléter la classe `ExempleSchema1` pour que le schéma soit dessiné sur un écran de dimension 600x400 avec l'unité fixée à 20 pixels. On ajoutera une méthode `dessiner` dans la classe `Point` et dans la classe `Segment`.

1. Pour que ce paquetage soit accessible depuis le projet Java, il faut télécharger afficheur.jar depuis la page du module (rubrique « ressources »), le sauver dans un dossier, faire un clic droit sur la racine du projet Java pour choisir *Build Path / Add External Archives...* et sélectionner afficheur.jar. On peut aussi faire *Build Path / Configure Build Path...*, sélectionner l'onglet *Librairies* et faire *Add External JARs*.
En ligne de commande, il faut utiliser l'option `-classpath` ou la variable d'environnement `CLASSPATH`.

**Conseil :** Avant de modifier la classe, il faut commencer par la compiler et l'exécuter.
**Solution :** On veut pouvoir dessiner un point sur un écran. On a donc naturellement un « sous-programme » (`dessiner`) avec deux paramètres de type `Point` et `Ecran`. En Java, il faut mettre ce sous-programme (on l'appellera méthode) dans une classe. Ici on pourrait le mettre dans la classe `Ecran` mais c'est une classe d'une bibliothèque et nous n'avons pas accès à son code source. De plus, il ne serait pas logique d'avoir à modifier la classe `Ecran` à chaque fois que l'on veut dessiner un nouvel objet.

On peut définir la méthode `dessiner` dans la classe `Point`. C'est la solution que nous allons retenir ici.

Une troisième possibilité aurait été de définir la méthode `dessiner` dans une troisième classe. C'est l'approche retenue dans le cours pour la méthode `trier`.

Dans la classe `Point`, nous pouvons définir cette méthode comme suit.

```
public void dessiner(Ecran ecran) {
   ecran.dessinerPoint(this.getX(), this.getY(), this.getCouleur());
}
```

Le même raisonnement nous conduit à définir la méthode `dessiner` dans `Segment` avec comme paramètre un `Ecran`.

```
public void dessiner(Ecran ecran) {
   ecran.dessinerLigne(this.extremite1.getX(), this.extremite1.getY(),
         this.extremite2.getX(), this.extremite2.getY(),
         this.getCouleur());

   // Dessiner les deux extrémités
   // ...on est pas obligé mais ce sera utile pour la suite du cours
   this.extremite1.dessiner(ecran);
   this.extremite2.dessiner(ecran);
}
```

Ainsi pour dessiner un schéma composé de points et segments, il suffira d'utiliser leur méthode dessiner avec comme l'écran à utiliser.

### Exercice 3 : Translater le schéma

Dans le programme de l'exercice 2, après avoir affiché et dessiné le schéma, ajouter les instructions pour :

1. le translater de 4 suivant l'axe des X et -3 suivant l'axe des Y (translater le schéma, c'est translater les trois segments et le barycentre) ;
2. l'afficher et le dessiner de nouveau.

Expliquer les résultats de l'exécution. On ne cherchera pas à corriger le programme.
**Solution :** Le barycentre n'est plus au centre du triangle. C'est en fait lui qui est à la position attendue. Le triangle a été trop translaté, deux fois trop. C'est parce que chaque point (p1, p2 et p3) est extrémité de deux segments (par exemple p1 est extrémité de s12 et s31). Translater une fois chaque segment, c'est donc translater deux fois chaque point.

On constate que les points sont partagés entre les segments. La relation entre la classe Segment et la classe Point (via les points extrémités) est donc une relation d'agrégation. Cet exemple laisse penser qu'ici ce serait plutôt une relation de composition qui est souhaitée. En TD nous verrons comment réaliser une telle relation.

### Exercice 4 : Afficher le schéma en SVG

Le paquetage `afficheur` définit une deuxième réalisation de l'interface `Afficheur` appelée `AfficheurSVG`. Elle dessine le schéma en SVG [2], spécification définie pour représenter les dessins vectoriels.

Compléter le programme de l'exercice 2 pour afficher le schéma en SVG en utilisant la classe `AfficheurSVG`. On affichera le schéma sur la sortie standard et dans un fichier, par exemple `schema.svg`. Ce fichier peut alors être ouvert avec inkscape ou firefox.

**Solution :** Si on applique le même raisonnement que pour l'exercice 2, on va donc ajouter une méthode `dessiner` dans la classe `Point` qui prend en paramètre un `AfficheurSVG`.

```
public void dessiner(AfficheurSVG svg) {
   svg.dessinerPoint(this.getX(), this.getY(), this.getCouleur());
}
```

Est-ce bien logique de faire ainsi ? Cette approche à deux défauts principaux. Les avez-vous identifiées ? Non, faisons donc l'exercice suivant (exercice 5)...

**Remarque :** Bien sûr, on procède de même pour la classe `Segment`. Ceci peut vous aider à répondre aux questions précédentes... Au moins en partie.

### Exercice 5 : Afficher le schéma sous forme d'un texte explicite

L'objectif de cet exercice est de réaliser un nouvel afficheur pour dessiner un schéma sous une forme textuelle qui permet de facilement comprendre la nature des objets du schéma et leurs caractéristiques. Nous allons donc définir une nouvelle réalisation [3] de l'interface `Afficheur` du paquetage `afficheur`. Nous l'appellerons `AfficheurTexte`. Elle sera définie dans le même paquetage que les classes `Point` et `Segment`. La manière dont doivent être affichés les points, segments, cercles et textes est donnée ci-dessous en reprenant l'exemple écrit à l'exercice 1.

```
Point {
      x = 1.0
      y = 2.0
      couleur = java.awt.Color[r=0,g=255,b=0]
}
Ligne {
      x1 = 6.0
      y1 = 2.0
      x2 = 11.0
      y2 = 9.0
      couleur = java.awt.Color[r=255,g=0,b=0]
}
Cercle {
      centre_x = 4.0
      centre_y = 4.0
      rayon = 2.5
      couleur = java.awt.Color[r=255,g=255,b=0]
}
```

2. SVG, Scalable Vector Graphics, est une spécification définie par le W3C qui s'appuie sur XML. On peut trouver plus d'informations à l'URL http ://www.w3.org/TR/SVG/.

3. Avec Eclipse, on peut faire *New / Class*, renseigner son nom, puis sélectionner l'interface qu'elle réalise en faisant *Add...*. Si l'option *Inherited abstract methods* est cochée (c'est le cas par défaut), le squelette des méthodes de l'interface est inclu dans le texte de la classe engendrée. Il ne reste plus qu'à compléter... Si on ne l'a pas fait à la création, on peut ensuite choisir dans le menu contextuel *Source / Override/implement Methods...* ou utiliser les propositions de correction des erreurs.

```
Texte {
      x = 1.0
      y = -2.0
      valeur = "Premier dessin"
      couleur = java.awt.Color[r=0,g=0,b=255]
}
```

Écrire la classe `AfficheurTexte` correspondant à cet afficheur et compléter le programme de l'exercice 2 pour l'utiliser.

**Indication :** L'instruction Java « `System.out.println(java.awt.Color.GREEN)` » affiche :

```
java.awt.Color[r=0,g=255,b=0]
```

**Solution :** Le but ici est d'abord d'écrire la réalisation d'une interface. Pas de difficulté particulière, juste plein de `System.out.println` !

On constate qu'un outil tel que Eclipse peut faciliter la vie du programmeur en ajoutant automatiquement le squelette des méthodes abstraites de l'interface.

Est-ce qu'on ajoute une méthode `dessiner(AfficheurTexte)` dans la classe `Point` (et dans la classe `Segment`) ?

Ceci a deux défauts principaux :

1. En suivant cette approche, on doit modifier les classes `Point` et `Segment` à chaque fois que l'on voudra dessiner notre schéma sur un nouvel afficheur. Ce n'est donc pas très pratique ! On aimerait bien « fermer » les classe `Point` et `Segment` et ne plus y toucher.
2. Le code qu'on écrit pour les différentes méthodes `dessiner` (dans `Point` ou dans `Segement`, on le constate tout particulièrement dans `Segment`) est exactement le même (au nom et au type du paramètre près). Éventuellement le nom de la méthode pourrait changé : dessiner, dessinerSVG... Mais est-ce logique de changer le nom de la méthode ? On veut dessiner sur un écran, un afficheur SVG ou afficheur texte. Le nom qui va bien est donc `dessiner`. Le type de l'afficheur est en paramètre de la méthode donc inutile de changer son nom. C'est tout l'intérêt de la surcharge.

   On a donc du code redondant, redondance qu'il serait bon d'éviter.

   En fait, on constate que l'on n'utilise aucune méthode particulière de l'afficheur considéré mais seulement les méthodes spécifiées dans l'interface `Afficheur`.

Ces deux défauts (l'un ou l'autre en fait) nous incitent à n'écrire qu'une seule méthode `dessiner` dans les classes `Point` et `Segment` (extensibilité et suppression des redondances). Par exemple, dans `Point`, on écrira :

```
public void dessiner(Afficheur afficheur) {
   afficheur.dessinerPoint(this.getX(), this.getY(), this.getCouleur());
}
```

On peut maintenant dessiner un point (ou un segment) sur n'importe quel afficheur (`Ecran`, `AfficheurSVG`, `AfficheurTexte` ou tout autre afficheur qui pourrait être écrit plus tard et par d'autres). **Nous n'avons plus à toucher à la classe Point si de nouveaux afficheurs devaient être ajoutés.** Nous pouvons « fermer » la classe `Point` (et la classe `Segment`).

Le diagramme de classe de l'application est donné à la figure 1.

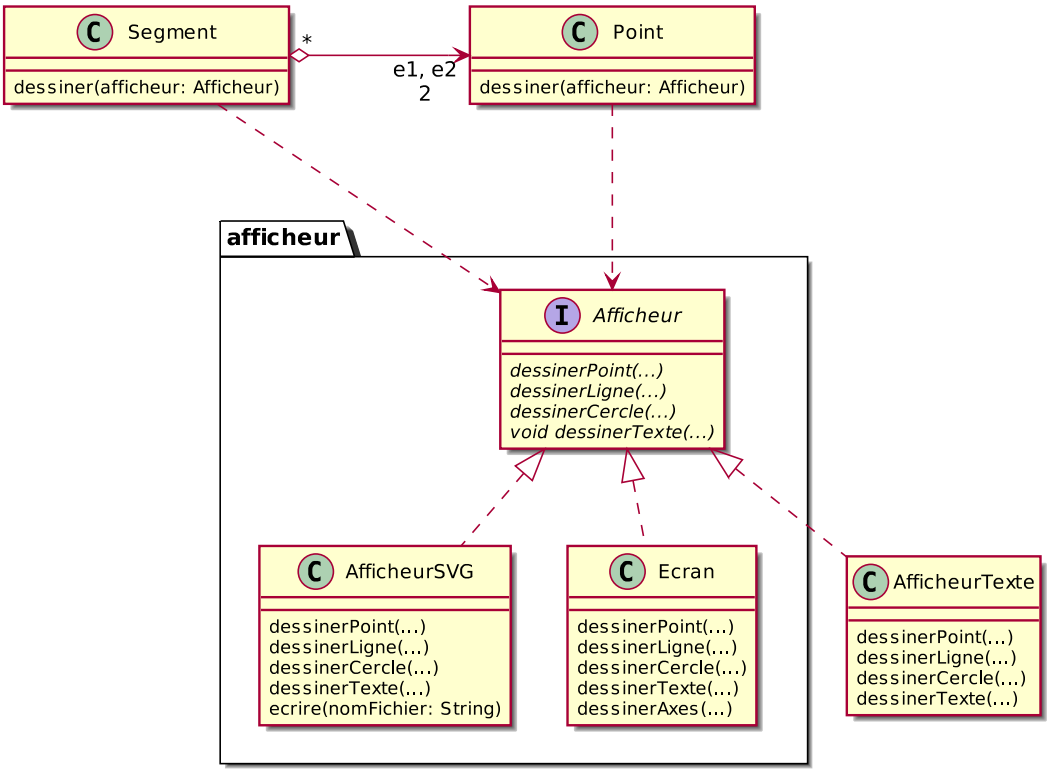


FIGURE 1 – Diagramme de classe

Ainsi, cet exercice n'aurait dû consister qu'en l'écriture de la réalisation `AfficheurTexte` de l'interface `Afficheur` et en la modification de la classe `ExempleSchema1` pour dessiner le schéma sur ce nouvel afficheur.

Écrire la classe `ExempleSchema1` est fastidieux car à chaque fois qu'on veut faire quelque chose avec le schéma (afficher, translater, dessiner sur un écran graphique, dessiner sur un afficheur SVG, dessiner sur un afficheur texte, etc.), il faut appliquer ce quelque chose aux trois segments et au barycentre. Ceci est lourd et fastidieux ! Il doit être possible d'améliorer ceci. On le verra dans les prochains TP.