TP : Héritage en Smalltalk

N. Bouraqadi

1 Gestion d'un établissement scolaire

Dans cette partie du TP, vous allez définir quelques classes fondamentales dans un logiciel de gestion d'un établissement scolaire. Il s'agit dans un premier temps de réviser la création de classes et la manipulation d'objets. Puis, dans un deuxième temps de vous initier à l'héritage.

1.1 Personne

Définissez la classe **Personne** qui dispose de deux variables d'instances : **nom** (chaîne de caractères) et **age** (entier). Dotez la classe des accesseurs en lecture et écriture à ces deux variables d'instances, ainsi que de la méthode **printOn: unStream** afin que la représentation textuelle d'une personne soit son nom.

1.2 Matiere

Définissez la classe **Matiere** qui représente une matière enseignée. Dotez là de deux variables d'instances : **nom** (chaîne de caractères) et **volumeHoraire** (nombre). Définissez les méthodes d'accès en lecture/écriture à ces deux variables d'instances.

1.3 Prof

Définissez la classe **Prof** comme une sous-classe de **Personne** qui dispose d'une variable d'instances **matieresEnseignees** (collection de type **Set** qui contient des instances de **Matiere**). Dotez la classe **Prof** des deux des accesseurs à **matieresEnseignees** ainsi que des méthodes suivantes :

- **ajouteMatiere: uneMatiere** – Le paramètre étant une instance de la classe **Matiere**.
- **enseigneMatiere: uneMatiere** – Retourne **true** si l'instance de **Matiere** passée en paramètre fait partie des matières enseignée par le prof.
- **printOn: unStream –** étend la représentation textuelle définie dans la superclasse **Personne** en ajoutant le pré-fixe '**Prof.** '. Ainsi, pour un prof nommé **Tournesol**, la représentation textuelle sera: **'Prof. Tournesol'.**

1.4 Eleve

Définissez la classe **Eleve**, sous-classe de **Personne** qui dispose de la variable d'instance **notes**. Cette dernière est un **Dictionary** dont les clés sont des matièrs (instances de **Matiere**) et dont les valeurs sont des nombres réels correspondant aux notes obtenues. Dotez la classe **Eleve** des méthodes suivantes :

- **noteObtenueA: uneMatiere –** Retourne la note obtenue à la matière passée en paramètre.
- **note: uneNote obtenueA: uneMatiere** – mémorise la note **uneNote** en l'associant à **uneMatière**.
- **moyenne –** retourne la moyenne des notes obtenues par l'élève.
- **printOn: unStream –** étend la représentation textuelle définie dans la superclasse **Personne** en ajoutant comme post-fixe la moyenne de l'élève. Ainsi, un élève nommé *Tintin* qui a *13,25* de moyenne aura pour représentation textuelle **'Tintin – 13.25'**.

2 Figures Géométriques

2.1 Eléments fournis

Nous manipulerons des figures géométriques. Nous utiliserons le repère suivant :

- origine = coin supérieur gauche de l'écran
- axe des abscisses orienté de gauche à droite
- axe des ordonnées orienté de haut en bas

Dans le code que vous allez produire, vous utiliserez la classe **FenetreDessin** fournie (image Pharo fournie) et la classe **Point** qui fait partie de la bibliothèque de base de Pharo. Vous pouvez accéder aux définitions de ces classes simplement avec le "Browser". Rappelons que :

- l'expression **10@20** crée un point dont l'abscisse est **10** et dont l'ordonnée est **20**;
- que différentes opérations (notamment l'addition, la soustraction et la distance) sont possibles sur les points (instances de **Point**);
- la classe **FenetreDessin** dispose d'un ensemble de méthodes pour dessiner des figures géométriques simples

2.2 La classe Figure

Définissez la classe **Figure** qui ne déclare aucune variable d'instance et qui défini, dans la catégorie **affichage,** les deux méthodes suivantes :

- **dessinerSur: uneFenetre** méthode abstraite. Les sous-classes sont censées la redéfinir pour dessiner la figure l'instance de la classe **FenetreDessin** donnée en paramétre.
- **dessiner** utilise la méthode **dessinerSur:** pour dessiner la figure courante sur une nouvelle fenêtre.
- **translater: delta** méthode abstraite. Les sous-classes sont censées la redéfinir de manière à déplacer le receveur autour de sa position courante.

2.3 La classe FigureSimple

Une figure simple est caractérisée par une couleur. Définissez la classe abstraite **FigureSimple** qui déclare la variable d'instances **couleur** (instance de **Color**). Dotez là des méthodes suivantes :

- Accesseurs en lecture/écriture pour la variable d'instance **couleur**

- **position** méthode abstraite qui doit être redéfinie dans les sous-classe pour retourne le point correspondant à la position de la figure simple.
- **deplacer: unPoint** c'est une méthode abstraite. Elle est censée déplacer la figure de sorte que sa nouvelle position corresponde au point passé en argument.
- **translater: delta** déplace le receveur autour de sa position courante. L'argument **delta** est un point dont l'abscisse et l'ordonnée représentent, respectivement, le déplacement suivant l'axe des abscisses et l'axe des ordonnées. Pensez à réutiliser la méthode **deplacer:**
- **printOn: unStream** définit la représentation textuelle d'une figure simple. Cette représentation consiste à donner le nom de la classe, la position du receveur et sa couleur. Précisons que le message **className** retourne le nom de la classe de l'objet destinataire.

2.4 La classe Segment

Définissez la classe **Segment** sachant qu'un segment est une figure simple qui est caractérisée par deux point (instances de **Point**) : l'origine et l'extrémité. La position d'un segment correspond à son origine. La classe **Segment** définit les méthodes suivantes :

- **origine: unPoint extremite: unAutrePoint** change l'origine et l'extrémité du segment.
- **origine** retourne l'origine du segment.
- **extremite** retourne l'extrémité du segment.
- **position** retourne l'origine du segment.
- **deplacer: nouvelOrigine** redéfinition qui remplace l'origine et actualise l'extrémité en conséquence.
- **longueur** retourne la longueur du segment = la distance entre l'origine et l'extrémité (trouvez dans la classe **Point** la méthode appropriée pour ce calcul)
- **dessinerSur: uneFenetre** redéfinition qui dessine le segment.
- **printOn: unStream** enrichie la représentation textuelle avec la longueur du segment.

2.5 La classe Polygone

Un polygone est une figure simple caractérisée par un ensemble de sommets. Tous les sommets sont reliés par des segments. Le dernier sommet est relié au premier.

Définissez la classe **Polygone** qui la variable d'instance **sommets** qui correspond à une collection ordonnée de points. Puis, dotez la classe d'accesseurs en lectures/écriture pour cette variable. Définissez également les méthodes suivantes :

- **deplacer: unPoint** déplace la figure de sorte que le premier sommet soit remplacer par le paramètre. Les autres sommets doivent être modifiés de manière à opérer la translation du polygone sans déformation.
- **segments** retourne un ensemble (instance de **Set**) regroupant les segments (instances de la classe **Segment** définie ci-dessus) qui constituent le polygone (c'est à dire qui relient ses différents sommets).
- **perimetre** retourne le périmètre du receveur = somme des longueurs des segments qui constituent la figure.

- **dessinerSur: uneFenetre** dessine le polygone sur la fenêtre passée en paramètre, sous la forme de segments.
- **printOn: unStream** enrichie la représentation textuelle avec le périmètre du polygone.

Définissez les méthodes de classe suivantes :

- **sommets: desPoints** retourne un nouveau polygone dont les sommets sont donnés en paramètre
- **rectangleLargeur: longueurEnX hauteur: longueurEnY** retourne un nouveau polygone qui correspond à un rectangle dont le premier sommet est 0@0
- **carreDeCote: longueur** retourne un nouveau polygone qui correspond à un carré dont le premier sommet est 0@0

2.6 La classe FigureComplexe

Une figure complexe est une figure géométrique qui regroupe un ensemble de "sous-figures" géométriques elles mêmes simples ou complexes. Nous utilisons ainsi le schéma de conception (en anglais *design pattern*) **Composite.**

Dotez cette classe d'une variable d'instance **sousFigures** (qui représente l'ensemble des sous-figures), ainsi que des méthodes d'instance suivantes :

- **sousFigures** retourne l'ensemble des sous-figures. Effectue une initialisation paresseuse.
- **sousFigures: unSet** remplace l'ensemble des sous-figures par l'ensemble passé en argument.
- **ajouterFigure: uneFigure** ajoute une sous-figure.
- **enleverFigure: uneFigure** retire une sous-figure. Si la figure passée en argument ne fait pas partie des sous-figures, rien ne se produit (pas d'erreur…).
- **translater: delta** translate l'ensemble des sous-figures.
- **dessinerSur: uneFenetre** dessine l'ensemble des sous-figures.

Définissez également les méthodes de classe suivantes :

- **maison** retourne une figure géométrique complexe qui représente une maison
- **village** retourne une figure géométrique complexe qui est constituée d'un ensemble de maisons