

# SAE S2.01

## L'ardoise magique de Farida

H. AZZAG

B. LEMAIRE - EINFO-LEARNING.FR

### Objectifs :

Le but de cette SAE est d'écrire une application capable de stocker des formes qui composent une ardoise ainsi que de pouvoir restituer celles-ci pour leur dessin<sup>1</sup> par une application tierce.

### Les compétences mises en oeuvre :

- Être capable de lire un cahier des besoins (sujet de la SAE) et de réaliser une spécification UML en utilisant les diagramme de classes.  
*Mots clefs : classe, classe abstraite, association, agrégation, composition, héritage...*
- Être capable de proposer une implémentation java à partir de diagrammes de classes.  
*Mots clefs : héritage, redéfinition, polymorphisme...*

### Modalités de rendu :

La SAE est à faire en binomes. Vous devez rendre vos fichiers (sources Java, exécutables, un fichier help) sur un lien Github.

Il faudra voir à vos chargés de TDs selon leurs directives.

- Date limite du rendu de la SAE le 14 JUIN 2023
- Contrôle SAE prévu le 15 JUIN 2023

### Ressources nécessaires à la réalisation de la SAE :

Vous devez vous connecter à la plateforme `einfo-learning.fr`

- Type de connexion : `anonyme`
- Mot de passe : `SaE;2023;S201#Fr`

---

1. Il ne s'agit pas ici des les dessiner graphiquement.

Accéder à la plateforme

Nom d'utilisateur

Mot de passe

Connexion





Nom d'utilisateur ou mot de passe oublié ?

Connexion anonyme

Catégories de cours:

Informatique / Base de la programmation objet / Base de la programmation objet

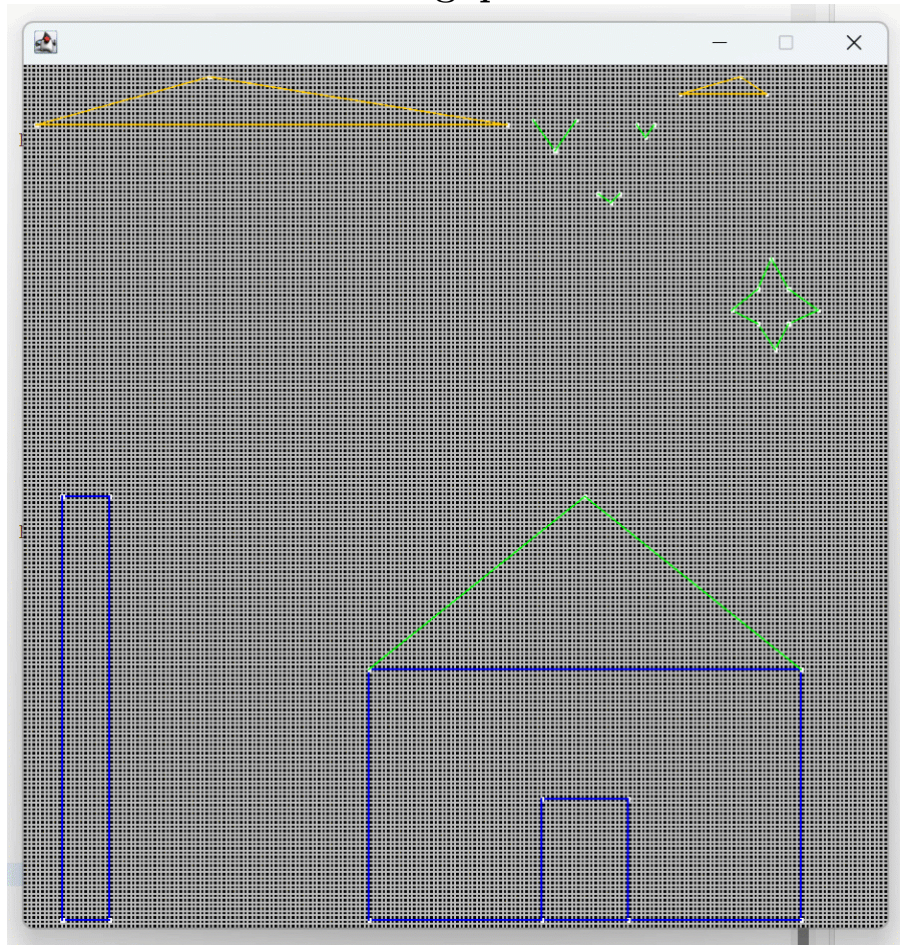
Rechercher des cours

 Base de la programmation objet Base la programmation - contrôle long de 2021 (Corrigé) Correction du contrôle long 2021 Accéder	 Base de la programmation objet Base de la programmation objet - cours 9 Ce cours aborde les classes abstraites et les interfaces en Java Accéder	 Base de la programmation objet Base de la programmation objet - contrôle 1 (Correction) Correction du contrôle 1 Accéder	 Base de la programmation objet Base de la programmation objet - SAE SAE Accéder
--	---	--	--

## ▼ Base de la programmation objet

▸ Base de la programmation objet <sup>(13)</sup>▸ Base de la programmation objet [2020-2021] <sup>(13)</sup>

## L'ardoise magique de Farida



### Ardoise magique ? kézaco ?

L'ardoise magique de Farida peut être considérée comme un tableau noir avec une grille de points de 200 x 200 points sur laquelle viennent se placer des formes. Le point de coordonnées (0,0) est le coin supérieur gauche.

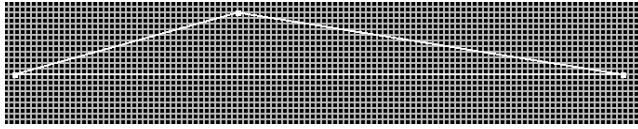
### Les formes

L'ardoise est composée de différentes formes ; ces dernières :

- possèdent :
  - \* un *nom* qui caractérise le rôle de la figure dans l'ardoise,
  - \* un *type de figure* (voir description plus loin).
- doivent-être capable :
  - de se dessiner, c'est à dire renvoyer la liste de tous les segments qui permettent de dessiner la forme.
  - de se déplacer.

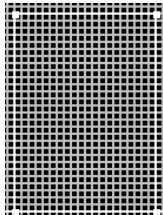
Voici les différentes formes :

- les **triangles** qui sont composées de 3 points (instance de `PointPlan`<sup>2</sup>)



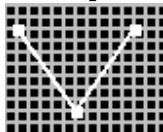
- \* ils sont construits à partir de trois points ;
- \* leur type de figure est "T" (Chaine de caractères "T" en majuscule).

- \* les **quadrilatères** qui sont composées de 4 points



- \* ils sont construits à partir de 2 points (le point supérieur gauche et le point inférieur droit) ;
- \* leur type de figure est "Q" (Chaine de caractères "Q" en majuscule).

- \* les **chapeaux** qui sont composées de 3 points



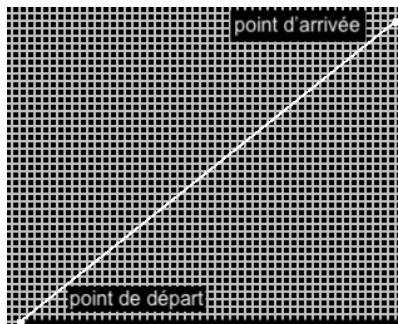
- \* ils sont construits à partir de 3 points ;
- \* leur type de figure est "C" (Chaine de caractères "C" en majuscule).

## Tout n'est que pas que formes

Il existe d'autres éléments qui ne sont pas des formes.

**Les segments :**

Lorsqu'une forme se dessine, elle retourne la liste des segments qui la composent.



<sup>2</sup>. classe fournie dans la librairie `ardoise.jar`

- \* un segment n'est pas une forme,
- \* un segment est composé d'un point de départ et un point d'arrivée.

**Exemple :**

Dans l'ardoise se trouve trois oiseaux. Considérons l'oiseau qui s'appelle "*oiseau 1*".

Lorsque *oiseau 1* se dessine, il renvoie une `ArrayList<Segment>` contenant les segments suivants<sup>3 4</sup> :

```
1
2 (118,13)-(123,20)    Premier segment (PointPlan de dpart) - (PointPlan d'arrive)
3 (123,20)-(128,13)    Deuxime segment (PointPlan de dpart) - (PointPlan d'arrive)
```

On en déduit que les points qui représentent *oiseau 1* sont ;

```
1 PointPlan p1 = new PointPlan(118,13);
2 PointPlan p2 = new PointPlan(123,20);
3 PointPlan p3 = new PointPlan(128,13);
```

— l'*oiseau 1* a été instancié comme suit :

```
1 Forme oiseau1 = new Chapeau("oiseau 1", p1, p2, p3);
```

— lorsque l'ardoise demande à l'*oiseau 1* de se **dessiner**, elle lui envoie le message `dessiner` et récupère la liste des segments qui le décrivent :

```
1 ArrayList<Segment> lSegOiseau1 = oiseau1.dessiner();
```

Vous disposez sur la plateforme `einfo-learning.fr` :

- de la librairie `ardoise.jar` qui devra être intégrée à votre projet<sup>5</sup>. Dans cette bibliothèque se trouvent les classes :
  - `PointPlan` qui implémente les points qui composent les formes et les segments,
  - `Segment` qui implémente les segments qui permettront aux formes de se dessiner,
  - `Ardoise` qui correspond à l'ardoise magique de Farida qui permettra de dessiner les formes<sup>6</sup>.
- `liste_segments.txt` : fichier contenant tous les segments que les formes doivent renvoyer lorsqu'elles se dessinent<sup>7</sup> ;
- `javadocs.zip` archive ZIP cotenant les javadocs des classes fournies<sup>8</sup>.

---

3. La liste de tous les segments que les formes communiquent à l'ardoise est disponibles dans les ressources pour la SAE.

4. Les coordonnées proviennent du fichier `liste_segments.txt` fournis dans les ressources.

5. Il est vous est fortement conseillé d'utiliser un environnement intégré de développement comme *Eclipse*

6. Cette classe vous permettra de tester votre implémentation Java.

7. Le but de ce fichier est de vous permettre de construire vos différentes formes.

8. A utiliser obligatoirement.

## Travail préliminaire

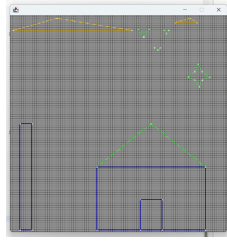
### Question 1 :

Réalisez le diagramme de classes qui correspond aux javadocs des classes fournies.

### Question 2 : Intégrer la librairie ardoise.jar à votre projet Eclipse

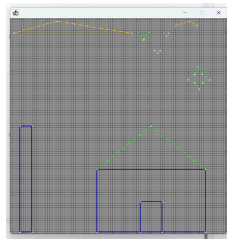
Configurer votre projet sous *Eclipse* et testez le source java suivant pour valider votre installation :

```
1  /**
2   * Comment intégrer la librairie ardoise.jar dans votre projet Eclipse
3   *
4   * 1/ Créer un dossier libs dans votre projet Eclipse
5   *     Clic droit sur votre projet -> New -> Folder
6   *
7   * 2/ Copier ardoise.jar dans le dossier libs
8   *
9   * 3/ Déclarer ardoise.jar comme librairie externe pour votre projet
10  *     Clic droit sur ardoise.jar (dossier libs) -> Build Path -> Add to Build Path
11  */
12
13
14
15  /**
16   * Importation des classes
17   * PointPlan
18   * Segment
19   * Ardoise
20   *
21   * qui sont fournis dans la librairie ardoise.jar
22   */
23  import ardoise.*;
24
25
26
27  /**
28   * Classe TestArdoise
29   * @author Bouchaib
30   * @version 2013
31   *
32   * Validation de l'intégration de la librairie ardoise.jar au projet Eclipse
33   */
34  public class TestArdoise {
35
36      public static void main(String []args ){
37
38          Ardoise ardoise = new Ardoise();
39
40          ardoise.test();
41
42      }
43  }
44 }
```



## Phase 1

1. Compléter le diagramme de classes pour modéliser toutes les formes qui apparaissent dans l'ardoise.
2. Implémenter le code Java correspondant à votre spécification UML.
3. Testez votre solution.
4. Après affichage des formes, attendre une seconde<sup>9</sup> et déplacer tous les oiseaux de 10 points en abscisse et 20 points en ordonnée.



## Phase 2

On constate que certaines formes sont composées elle-même d'autres formes comme la maison qui est constituée d'un chapeau (toit) et de deux rectangles (corps et porte de la maison). Le type de figure qui sera attribué à ces formes composées est "GF" (Chaine de caractères "GF" en majuscule).

- Faites évoluer vos diagrammes de classes et proposez une implémentation Java de cette évolution. Testez.

## Phase 3

Dans cette phase, nous allons nous focaliser sur la robustesse du code que vous avez développé :

- Représentez toutes les situations pouvant provoquer des erreurs
- traitez ces erreurs en utilisant les notions que vous avez abordé dans la ressource R203 (les exceptions)

---

9. Un exemple est fourni sur la plateforme