SAE 3.01 – Développement d’une application

Table des matières :

Fonctionnalités :

- Importation et lecture d’un fichier .class (pour créer un squelette de class)
- Génération automatique de diagrammes de classes à partir de packages Java.
- Exportation des diagrammes au format image, PlantUML ou résultat compilé.
- Générer l’interface graphique
- Modification des diagrammes : ajout de classes, méthodes, etc.
- Générer les squelettes des classes
- Afficher ou masquer des éléments (classe et méthodes)
- Afficher ou masquer l’héritage et les implémentations

Cas d’utilisation :

1. **Importer un package Java**
   L'utilisateur sélectionne un package sur son ordinateur par l'intermédiaire de l’application pour analyse via introspection.
2. **Générer un diagramme de classes**
   Le système crée une représentation des classes et des relations.
3. **Modifier un diagramme**
   L'utilisateur peut :
   - Ajouter de nouvelles classes.
   - Ajouter des méthodes à des classes existantes.
4. **Déplacer les classes sur l'écran**
   Permet de réorganiser les éléments pour une meilleure lisibilité.
5. **Afficher/masquer certains éléments**
   L'utilisateur peut :
   - Afficher/masquer des classes.
   - Afficher/masquer des méthodes.
   - Afficher/masquer la classe parente ou les interfaces d'une classe.
6. **Exporter un diagramme**
   Le diagramme peut être exporté sous :
   - Différents formats d'images.
   - Fichier source PlantUML.
   - Résultat compilé PlantUML.
7. **Générer les squelettes de classes**
   Créer les fichiers Java correspondants aux classes définies dans l'application.

Diagrammes de cas d’utilisation :

Cas importer un package java

Déroulement normal:

(1) L'utilisateur choisit un package java présent sur son ordinateur qu'il veut utiliser dans l'application. (2)l'application charge le package (3), l'application fait l'introspection du package, l'application génère les noms, attributs, méthode de chaque classe, (4) puis il génère les différentes relations.(5) Enfin, l'application génère l'interface une interface graphique de ce qu'il a généré. (6) l'utilisateur peut s'il le souhaite modifier le diagramme.

Variante:

(A) importation d'un package non java : étape (1)
(B) erreur dans l'importation du package : étape (1)
(C) erreur lors de l'introspection : étape (3)

Cas modifier le diagramme des classes

précondition:

l'utilisateur doit importer un package java qui ne contient pas d'erreur.

Postcondition:

A chaque modification et à la fin des modifications l'application doit régénérer : les noms, attributs, méthode de chaque classe, etc…

Déroulement normal:

(1) L'utilisateur peut réorganiser graphique chaque éléments de l'interface de l'application.(2) L'utilisateur peut afficher ou masquer n'importe quel élément.(3) l'utilisateur peut ajouter ou modifier les éléments déjà présents: ajouter des classes, attributs méthodes ou bien des relations entre les classes.

Variante:

modification invalide qui ne respecte pas la POO:

Héritage

1. Pas de double héritage de classes :
   Une classe ne peut hériter que d'une seule classe. Toutefois, une classe peut implémenter plusieurs interfaces.

2. Finalité des classes :
   - Une classe déclarée avec le mot-clé `final` ne peut pas être étendue.

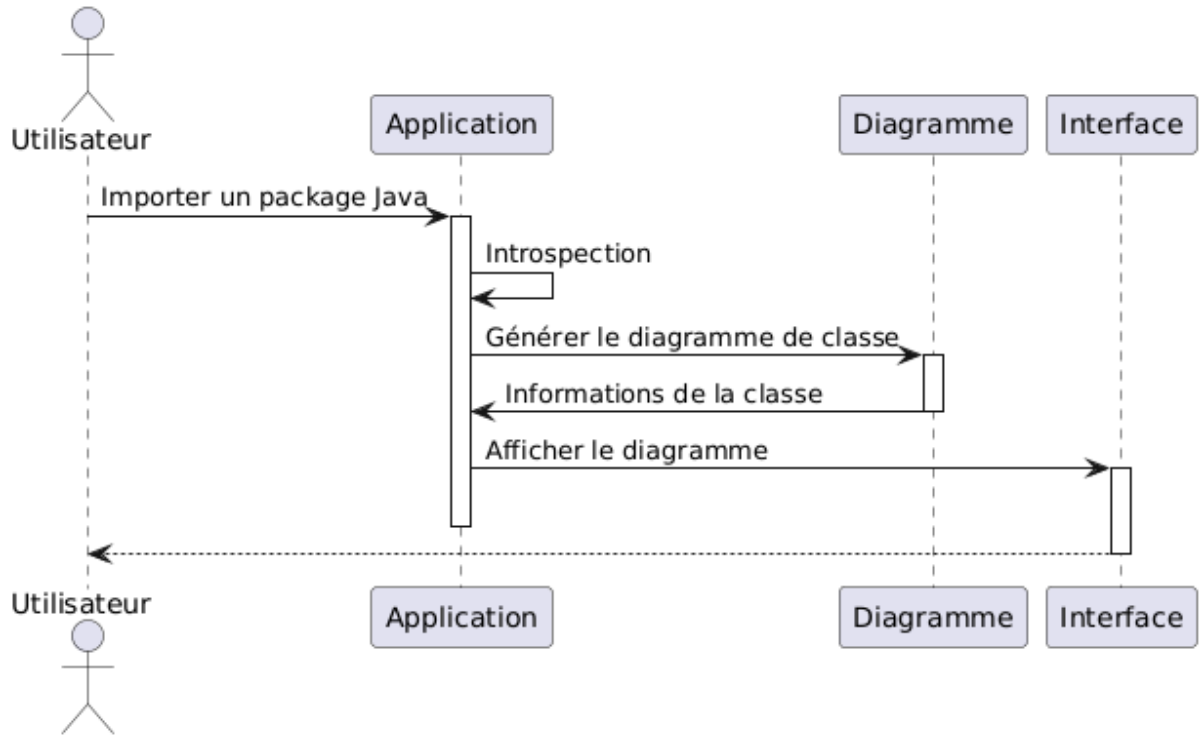
Interfaces et Abstractions

1. Implémentation stricte des interfaces :
   Une classe qui implémente une interface doit fournir une implémentation pour toutes les méthodes abstraites de l'interface, sauf si la classe est déclarée `abstract`.

Si une de ces règles n'est pas respectée alors l'application mettra en évidence l'erreur de l'utilisateur et refusera l'exportation tant que l'erreur sera présente.

précondition:
l'utilisateur doit importer un package java qui ne contient pas d'erreur.
Déroulement normal:
(1) L'utilisateur choisit le format qu'il souhaite (png, pdf, etc..), (2) l'application génère le résultat selon le format choisi par l'utilisateur. (3) téléchargement du fichier sur l'ordinateur de l'utilisateur.

DDS importation d'un package java:



DDS Création d'un diagramme de classe :

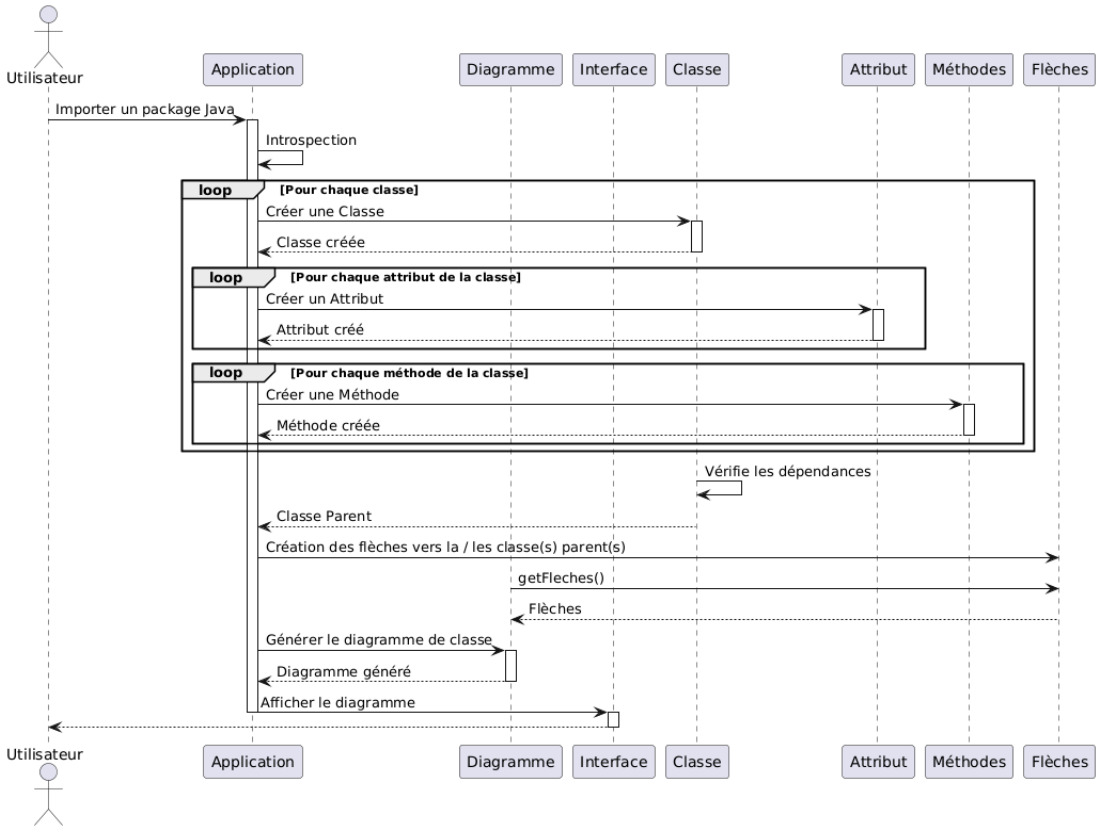


Diagramme d'activité de l'application :

L’utilisateur importe un package java, l’application vérifie s' il y a une erreur lors de l’importation. Si aucune erreur n’est trouvée, elle effectue l’introspection et vérifie de nouveau s' il y a une erreur. Si encore une fois aucune erreur n’est trouvée, elle génère le diagramme de classe et l’affiche dans l’interface graphique. Il est possible d’effectuer des modifications (Déplacer, ajouter ou masquer un élément). La modification doit évidemment respecter les règles de java (Par exemple : Une classe ne peut pas hériter de plusieurs classes). Il génère le diagramme avec les modifications effectuées. Finalement, il offre la possibilité d’exporter le diagramme dans différents formats et génère les squelettes des classes.

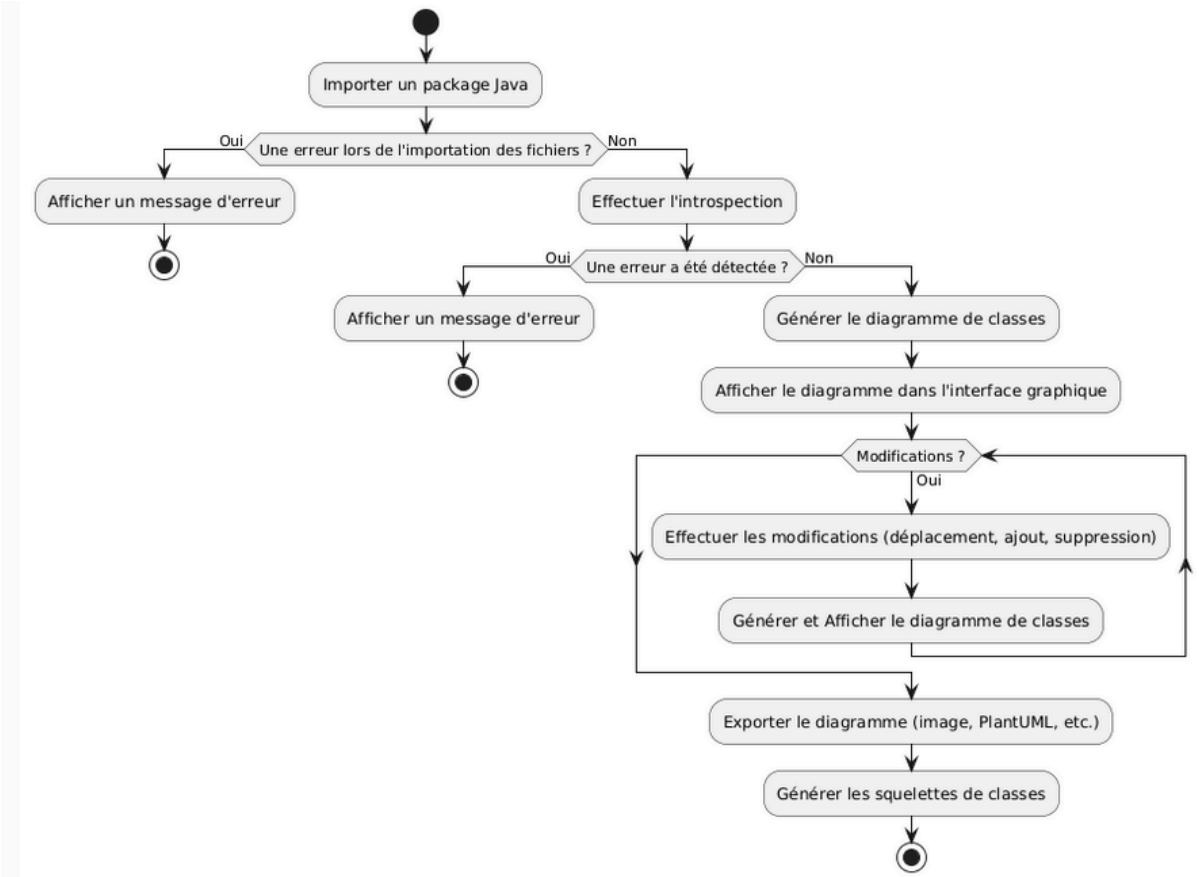


Diagramme de classe :

diagrammedeclasse.png

Singleton+Composite.png

Fabrique.png

MVC.png

Patrons conception :

Dans ce projet réalisé par nos soins, nous prévoyons d'utiliser les patrons singleton et fabrique pour la classe FabriqueBasicGraphicAttributes qui nous permettra de créer des Objet BasicGraphicAttributes avec des paramètres pré remplis pour facilement distinguer visuellement les attributs, les classes et les méthodes et une seule instance sera nécessaire, car les objets produits ne dépendent pas de caractéristiques de cette instance.

Nous prévoyons également d'utiliser le patron Observateur à travers le modèle MVC, où l'observateur est la classe abstraite Vue, qui se contentera de se mettre à jour lorsque le modèle lui ordonne pour changer l'affichage, le modèle lui effectuera des opérations lorsque qu'un Contrôleur lui ordonnera de le faire avec éventuellement des informations transmises sur l'opération (comme par exemple les nouvelles coordonnées de la vue et la vue).

Nous utiliserons aussi le patron composite pour gérer la composition des classes, où les composants seront les attributs primaires et les classes seront les composites, cela permettra de gérer plus efficacement les attributs d'une classe qui peuvent être une classe ou un attribut primaire.

Le patron stratégie lui servira à éviter les copier coller dans les classes en regroupant des éléments communs et à utiliser une méthode sans avoir à se préoccuper de quelle classe est l'instance comme pour les contrôleurs ou les vues, et cela, permet d'alléger les classes Interface, Attribut et Classe via une interface CompositionClasse qui va posséder les attributs et méthodes communes, puis via un héritage entre Classe et Jonction où Jonction posséderas les attributs en communs, la même chose est faite avec une interface Visibilite pour gérer les éléments masquables. Les instances d'Interface n'étant pas un attribut de classe, elle implémente l'interface CompositionClasse pour éviter de posséder des instances de cette classe en guise d'attribut.

Maquette balsamiq :

GenDiagClasseBalsamiq.pdf