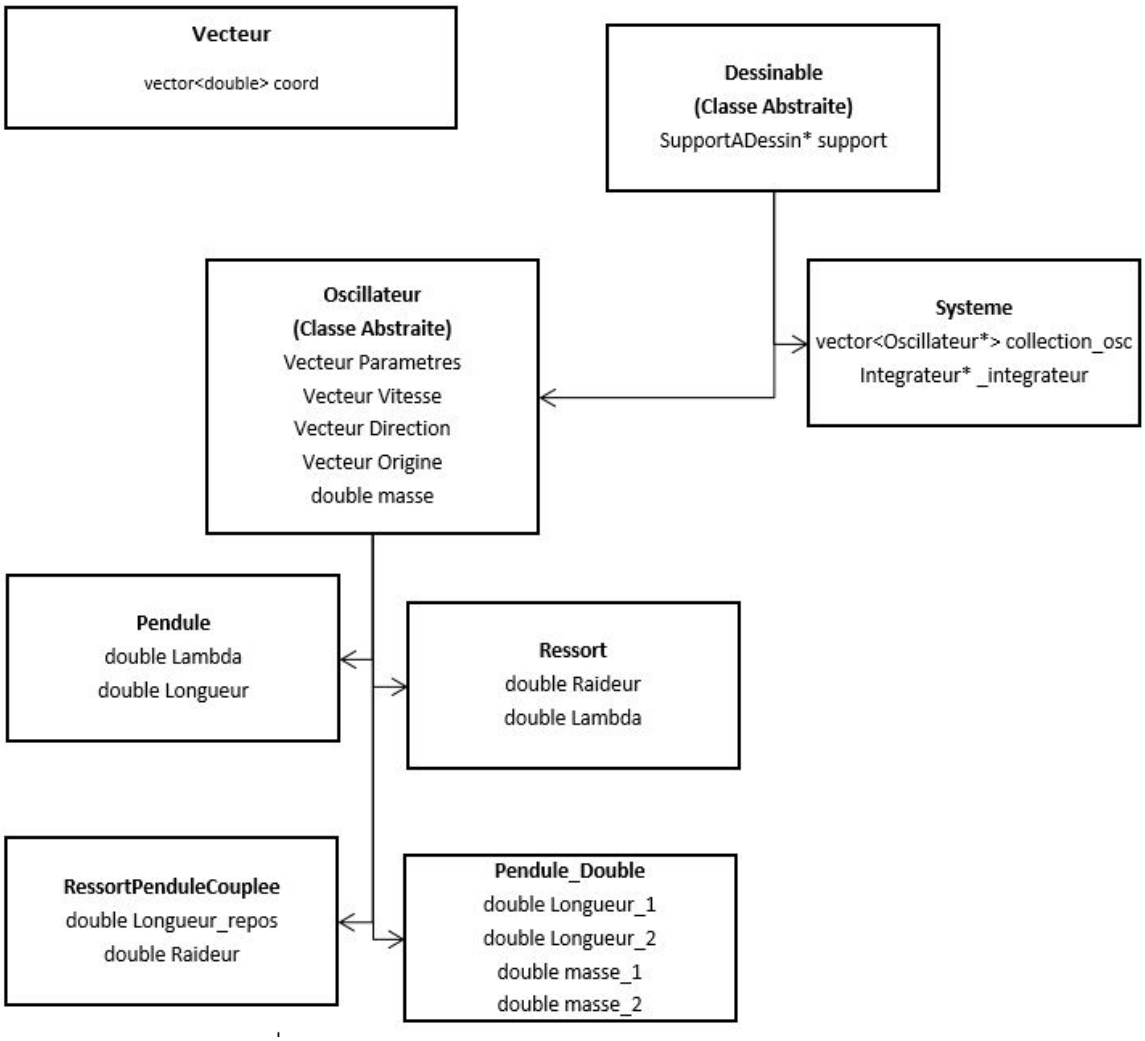
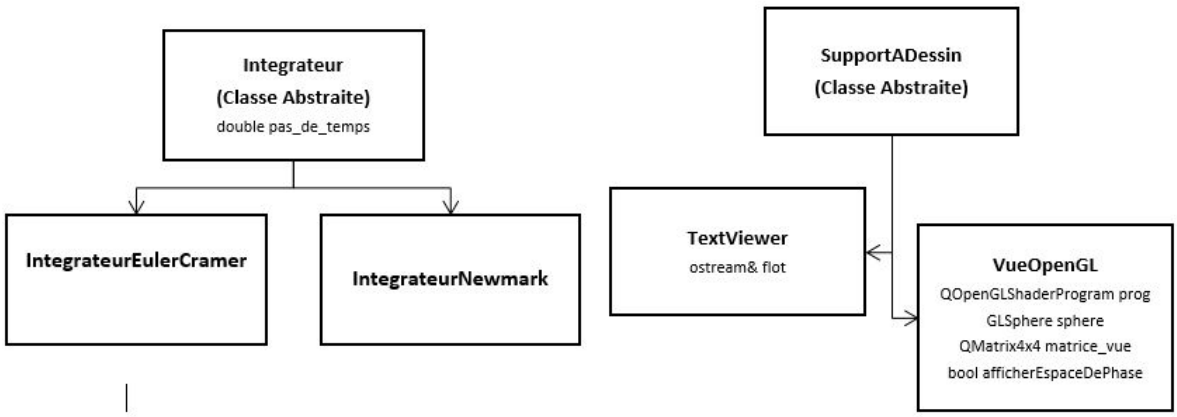
Notre projet est structuré en trois grandes parties:

1. Le noyau qui est formé par les classes de base: Vecteur, Oscillateur et ses sous-classes, Systeme, Dessinable, SupportADessin et Integrateur et ses sous classes.
2. Les classes additionnelles pour implémenter la simulation en mode texte.
3. Les classes additionnelles pour implémenter la simulation en mode graphique.

En ce qui concerne le noyau du projet, l'image ci-dessous illustre les liens entre les classes et leurs attributs:





En effet, les classes Ressort, Pendule, Pendule_Double et RessortPenduleCouplees sont des sous-classes de la super classe abstraite Oscillateur qui est elle-même une sous-classe de la classe abstraite Dessinable. Puis celle-ci possède un pointeur sur une instance de SupportADessin.
Ensuite, si nous considérons la classe système, elle est une sous classe de Dessinable qui possède une collection hétérogène d'oscillateurs et un pointeur sur un Integrateur (qui eventuellement pourrait être partagé par plusieurs systèmes).
La classe Vecteur encapsule la notion d'un vecteur en plusieurs dimension et pourra être manipulé avec les opérateurs usuelles entre vecteurs. Elle est nécessaire pour manipuler les attributs de la classe Oscillateur et ses sous classes.

Les simulations en mode texte et en mode graphique ne possèdent aucune particularités en ce qui concerne la structure. C'est-à-dire, pour la simulation en mode texte nous avons simplement ajouté une classe TextViewer qui est une sous classe de SupportADessin et pour la simulation en mode graphique nous avons repris l'exemple 05 du tutoriel et nous l'avons modifié pour qu'il satisfasse les besoins de notre projet.

De plus, voyons quelques particularités de notre projet.

1. Nous avons implémenté un système de gestion d'exceptions tel que s'il y a un soucis avec l'initialisation des oscillateurs ou avec les calculs à réaliser, en mode texte le programme s'arrête et affiche où se trouve l'erreur et, en mode graphique, il s'arrête, affiche l'erreur sur le terminal et, en plus, le programme ouvre une fenêtre de type "QMessageBox" qui précise aussi de quelle erreur il s'agit. Cependant, en mode graphique, il faut préciser que s'il y a une erreur dans l'initialisation d'un oscillateur, les oscillateurs initialisée avant seront affichés correctement tandis que ceux initialisés plus tard ne seront même pas construits.

2. L'implémentation de la possibilité de tracer les trajectoires dans l'espace des phases se base sur le fait que chaque oscillateur possède un tableau comme attribut où l'on stocke à chaque pas de temps le point de sa trajectoire dans l'espace des phases. Mais au cas où cette partie de notre projet ne soit pas claire, voilà une **note** contenant une description détaillée des changements que nous avons dû effectuer.
   [Note]
   D'abord, on a ajouté un attribut à la classe Oscillateur: un tableau dynamique de vecteur appelé espaceDePhase (c'est ici où l'on stockera les points de la trajectoire à chaque pas de temps). Ensuite, on a aussi ajouté à la classe Oscillateur une méthode "ajouterAEspacePhase" dont le rôle est d'ajouter un point (i.e. Un Vecteur de dimension 2) au tableau décrit précédemment.
   Alors, on a modifié la méthode "integre" de tous les intégrateurs de façon à qu'elle appelle à la fonction "ajouterAEspacePhase" à chaque pas de temps afin qu'à toutes les itérations le point formé par la première composante du vecteur Parametres et la première composante du vecteur Vitesse soit stocké dans un tel tableau.
   De plus, on a ajouté une autre méthode "dessine_phase" dans la classe Oscillateur qui appel à une méthode (aussi ajouté expressément) de la classe SupportADessin nommée "dessine_phase" qui prend un tableau dynamique de vecteur comme paramètre et dont le rôle varie selon on veut faire la simulation en mode text ou en mode graphique. (Ce tableau dynamique passé en paramètre sera le tableau espaceDePhase)
   Dans tous les cas, cette méthode "dessine_phase" de la classe Oscillateur sera appelée depuis la méthode "dessine" de système, laquelle fait un boucle pour dessiner tous les oscillateurs qui le composent et leurs espaces des phases.
   Ceci forme le coeur de notre conception pour l'affichage de l'espace des phases de chaque oscillateur.

   On distingue maintenant la conception que nous avons développé pour la simulation en mode texte et celle en mode graphique.

   En mode texte, nous avons décidé d'utiliser les flots d'entrée/sorties que l'on a vu au premier semestre pour stocker ces points qui après serviront pour tracer la trajectoire dans l'espace des phases avec gnuplot.
   Or, ici on a dû faire face à un problème de conception… Dans nos systèmes on aura surement plusieurs oscillateurs et donc plusieurs espaces de phases à dessiner, mais comment pourrions

nous implémenté un programme qui puisse distinguer à quel oscillateur appartiennent les différents points stockés? Malheureusement, nous n'avons pas encore trouvé une réponse générale à cette question et c'est pour cela que nous avons choisi que ça soit la trajectoire dans l'espace des phases du dernier oscillateur ajouté au système le seul qui va être afficher.
En effet, la méthode "dessine_phase" hérité de SupportADessin écrit dans un fichier qu'elle nomme "data.txt" tous les points stockés dans le tableau qu'elle reçoit comme paramètre (**rappel:** c'est le tableau espaceDePhase de la classe Oscillateur) mais avant de ceci elle le nettoie complètement. De cette façon, à la fin de l'exécution du programme les seul points qui figurent sur "data.txt" sont ceux qui appartiennent au dernier oscillateur du système.

En mode graphique, la méthode "dessine_phase" de la classe vue_opengl utilise GL_LINE_STRIP pour relier et dessiner à tous les coups les points de l'espace de phase de l'oscillateur en question.
[FIN DE NOTE]

3. Finalement, nous aimerions préciser que dû au fait que la boucle for pour afficher les oscillateurs d'un système se réalise dans la méthode "dessine" de système mais qu'on a besoin de surcharger l'opérateur << pour définir la méthode qui dessine un système de la classe TextViewer, nous n'avons pas trouvé une meilleur façon de faire que d'implémenter une méthode affiche de système dont le rôle est simplement d'afficher: "Le système est constitué des oscillateurs suivants: ". De cette façon, l'utilisateur pourrait essayer d'afficher un système avec l'opérateur << mais la seule chose qu'il arriverait à afficher c'et le message précédent.
Cependant, cette façon d'afficher un système n'a pas de sens. En effet, le fait que nous devons créer une instance de TextViewer pour pouvoir créer un oscillateur implique cette nécessité d'appeler la méthode dessine de l'objet à dessiner au lieu de simplement l'afficher en utilisant la surcharge de l'opérateur <<.