# Fiche d’exercices – Complexité

Pour chaque exercice ci-dessous, fournir le code **Java**, ainsi que la **modélisation UML** (projet *NetBeans* & *Visual Paradigm for NetBeans* compacté sur *Nom-FicheRobust-ExerciceX.zip*).

1. Modéliser et implémenter en Java le mécanisme de gestion d’inscriptions d’une école à partir des notions ci-dessous.
	1. Dans une école de musique, les étudiants sont organisés en groupe. Chacun aura une ou plusieurs notes ;
	2. Lorsque le nombre d’inscrits sur un groupe dépasse un seuil prédéfini par l’école, plusieurs groupes sont alors créés ;
	3. Un groupe est affecté à un enseignant responsable.
2. Construire une application capable de lire les notes des étudiants à partir d’un fichier. Chaque fichier doit contenir les notes d’un seul groupe d’étudiants. L’application doit alors calculer la moyenne et le résultat de chaque étudiant par groupe.
3. Faire évoluer l’application précédente de manière à ce qu’elle puisse dessiner un histogramme avec les moyennes du groupe.
4. Considérer le scénario suivant :

Une compagnie propose à ces clients de coffrets cadeaux pour les enfants. Dans un coffret, on peut avoir des jouets, voir d’autres coffrets plus petits. Chaque produit (coffret ou jouet) possède une description et un prix, tandis que les coffrets possèdent également leurs dimensions (largeur, hauteur, profondeur). La compagnie souhaite pouvoir calculer le prix d’un coffret (qui évidemment varie en fonction de son contenu) et valider sa composition (par exemple, si les dimensions des coffrets à l’intérieur sont inférieures à celles du coffret extérieur).

1. Modéliser la ligne de coffrets proposée par la compagnie de manière à pouvoir calculer leur prix et valider les coffrets.
2. Considérer maintenant l’usage du design pattern Composite. Son usage est-il viable dans ce contexte ?
3. Implémenter en Java la modélisation choisie.
4. Considérer le scénario suivant :

Vous devez créer un nouvel éditeur de figures géométriques. Plusieurs formes de figures sont possibles (des carrés, des triangles, etc.), ainsi que des figures regroupant toutes ces formes. Une figure va avoir des points qui la définissent (un point étant des coordonnées x et y), ainsi que des attributs tels que la couleur de la ligne et celle du remplissage. On souhaite que l’outil puisse réaliser des transformations (des rotations, des translations, etc.) sur les figures, quelque soit la figure. L’outil doit être extensible : on doit pouvoir ajouter facilement des nouvelles transformations.

1. Modéliser les figures et les transformations.
2. Avez-vous pu utiliser un design pattern ? Si oui, lequel et pourquoi ?
3. Implémenter en Java la modélisation choisie.
4. Considérer le scénario suivant :

Une compagnie d’assurance possède plusieurs méthodes d’analyse des risques pour leurs assurés. Ces méthodes analysent diverses informations liées à un assuré afin de lui établir un perceptuel de risque (entre 0 et 100). Par exemple, une méthode A étudie l’âge et le sexe de l’assuré, la méthode B analyse surtout les sinistres (accidents) que l’assuré a déjà eus. On souhaite avoir un traitement le plus homogène possible entre les différentes méthodes d’analyse et surtout, on souhaite être capable d’ajouter facilement des nouvelles méthodes au système. Chaque méthode est identifiée par un nom et doit être capable de calculer le perceptuel de risque à partir des informations sur l’assuré, ses polices d’assurance et les sinistres associées aux polices. Chaque assuré (dont on connaît le nom, l’âge et le sexe) possède une ou plusieurs polices d’assurances. A chaque police d’assurance (dont on connaît le type et la période de validité), on doit pouvoir associer un ou plusieurs sinistres (accidents). Chaque sinistre possède une date et une description.

1. Modéliser le scénario ci-dessus en UML.
2. proposer une implémentation Java pour les méthodes d’analyse (il n’est pas nécessaire de remplir tous les corps des méthodes).
3. Expliquez les points forts et les éventuelles faiblesses de votre proposition (modules, éléments réutilisables, patterns utilisés, API utilisées, etc.)