# Programmation Orientée Objet

# TD 6 – Exceptions, Map, ...

**Contexte :** on partira d'une implémentation simple des `Pierres` et du `Coffre`, avant inclusion de la généricité. On testera et commentera au fur et à mesure.

**Exercice 1 : Exception : Valeur non initialisée...**

La méthode `getValue()` de Gemstone n'a pas de sens si la `Pierre` n'a pas été évaluée au préalable. Au sein de cette exercice, on ne fera pas d'expertise au sein de `getValue()`

**a :** Ajouter un attribut booléen `evaluated` qui indique si on a déjà expertisé une pierre. Ajuster le code en conséquence (gestion de `évalué` dans le constructeur et expertize).

**b :** Définir une exception spécifique `NotExpertisedException` qui se déclenche quand on demande la valeur d'une pierre non expertisée.

**c :** Adapter l'évaluation d'un coffre pour tenir compte de cette exception. Réfléchir au comportement souhaité dans ce cas (traiter ou faire suivre).

**Exercice 2 : Exception : Ouvrir un coffre ouvert ?**

Créez des exceptions pertinentes pour les cas suivants :
— Tentative d'ouverture d'un coffre ouvert, de fermeture d'un coffre fermé.
— Ajout d'une pierre dans un coffre fermé.
— Code erroné.

Dans chacun des cas, réfléchir au comportement souhaité (traiter ou faire suivre).

**Exercice 3 : Comparaison de 2 pierres**

Dans l'exercice suivant, on souhaite classifier et comparer des `Gemstones`.

**a :** Donner un nom à chaque classe concrète dérivant `Gemstone` ("topaze" pour `Topaze`, etc).

**b :** Pour manipuler pleinement des gemmes, il nous faudra surcharger les méthodes `equals` et `hashCode`.

On dira que deux pierres sont identiques si elles ont le même nom (type) et la même valeur.

**c :** Pour comparer deux pierres quelconques, il faut avoir une méthode `CompareTo` déclarée dans l'interface `Comparable<T>`.

On utilisera l'ordre alphanumérique de la concaténation nom+valeur.

**d :** Mettre en place ces modes de comparaison et tester.

**Exercice 4 : Stockage plus compliqué qu'une `ArrayList`**

**Note :** on pourra repartir d'un coffre ou créer une nouvelle classe qui met en œuvre simplement les fonctionnalités souhaitées.

Afin de pouvoir faire des recherches efficaces de Pierres, on souhaite les *ranger* de manière à retrouver facilement les pierres d'une certaine valeur.

Pour cela, on définit une `Map` qui associe une valeur à l'ensemble des pierres stockées dans le coffre ayant cette valeur.

**a :** Qu'à-t-on fait nous permettant d'utiliser un `SortedSet` pour cet ensemble ? Un `HashSet` ?

**b :** Donner la nouvelle version de l'attribut `MyGems` (qui permet de faire ce stockage). On utilisera la collection la plus pertinente.

**c :** Mettre à jour les méthodes `addGem` et `removeGem`.

**d :** Créer une méthode permettant de dire si une `GemStone` passée en paramètre (et expertisée au préalable) est présente dans le coffre.

**Exercice 5 : Une belle GUI**

À l'aide du modèle MVC développé en cours, créez une GUI permettant de manipuler des `Gemstone`s en utilisant Swing.