INSA Centre Val de Loire
Département STI

Année 2016–2017

Programmation Orientée Objet

TD 6 – Exceptions, Map, ...

Contexte : on partira d'une implémentation simple des `Pierres` et du `Coffre`, avant inclusion de la généricité.

Exercice 1 : Exception

La méthode `getValeur()` n'a pas de sens si la `Pierre` n'a pas été évaluée au préalable.

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

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 : Comparaison de 2 pierres

En préalable de l'exercice suivant, il faut pouvoir comparer deux pierres quelconques. Pour cela, il faut avoir une méthode `CompareTo` définie dans l'interface `Comparable<T>`.

On dira que deux pierres sont identiques si elles ont la même valeur.

a : Mettre en place ce mode de comparaison et tester.

Exercice 3 : 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 : Donner la nouvelle version de l'attribut `MesPierres` (qui permet de faire ce stockage).

b : Mettre à jour les méthodes `ajouterPierre`, `retirerPierre`.

c : Créer une méthode permettant de dire si une Pierre passée en paramètre (et expertisée au préalable) est présente dans le coffre.

d : Bien sûr, tester et commenter au fur et à mesure.