
Exercice 1. *En cours (live coding).* Création et utilisation d'une classe simple

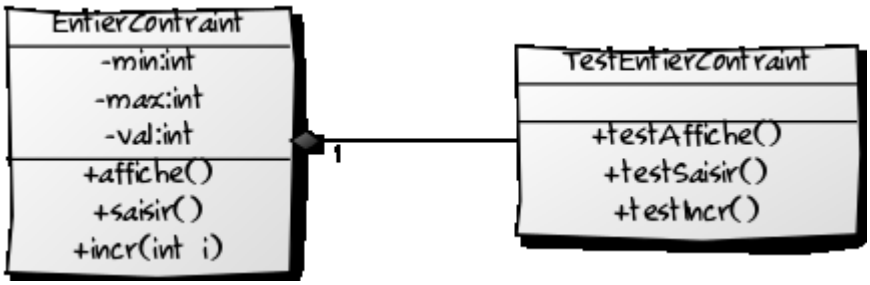
Ecrire une classe EntierContraint qui permet de représenter un entier compris dans un intervalle [min; max] donné. Cette classe doit au minimum contenir :

- Un constructeur qui fixera l'intervalle [min; max] des valeurs autorisées pour l'entier, en s'assurant que l'intervalle est valide (la borne inférieure est inférieure ou égale à la borne supérieure)
- Une méthode d'affichage qui affiche l'entier.
- Une méthode de saisie qui permet de saisir une nouvelle valeur pour l'entier. La saisie doit se terminer seulement si l'entier saisi se trouve bien dans l'intervalle, et doit recommencer sinon.
- Proposer une méthode d'incrémentation (la valeur de l'incrément étant un argument) pour incrémenter un entier contraint. Attention à bien gérer les problèmes de débordement d'intervalle.

On veillera à bien respecter les principes de l'encapsulation des données.

- Ecrire une classe de test TestEntierContraint pour tester la classe précédente.
- Rappeler les étapes de l'initialisation des champs d'un objet.

Diagramme de classes :



Exercice 2. Champ d'instance avec l'attribut *final* (champ constant)

En supposant que la valeur des bornes inférieure et supérieur d'un entier contraint restent les mêmes au cours de la vie de cet objet, quel est l'intérêt de les déclarer avec l'attribut final ? Faites-le et vérifier que vos tests précédents fonctionnent toujours.

Exercice 3. Méthodes d'accès aux champs privés : les *getters* et les *setters*

Proposer les modifications adéquates à la classe EntierContraint afin de pouvoir accéder depuis l'extérieur de la classe aux valeurs des bornes et de l'entier. Même chose cette fois pour modifier la valeur de l'entier sans passer par une saisie (n'autoriser le changement de valeur que si la nouvelle valeur est bien dans l'intervalle).

Tester ces nouvelles fonctionnalités.

Diagramme de classes proposé (à regarder uniquement après avoir écrit votre proposition de diagramme de classes) : http://yuml.me/edit/59c8ff8e

Exercice 4. 1 contrat initial et 2 implémentations possibles pour la classe

Dans cet exercice, nous allons insister sur ce qui doit rester inchangé au cours de la vie d'une classe, à savoir le *contrat* (= l'ensemble des services qu'elle doit réaliser au minimum) et ce qui peut évoluer/changer au fil du temps, à savoir les *implémentations* de la classe.

On vous demande une classe permettant de convertir les nombres sexagésimaux (heures, minutes et secondes) en nombres flottants (en considérant l'heure comme unité) et vice versa. Cette classe aura au minimum :

- 2 constructeurs, 1 prenant en entrée un nombre sexagésimal, un autre un nombre flottant.
- Une méthode getDec fournissant la représentation décimale de l'objet.
- Trois autres méthodes : getH, getM et getS fournissant les trois composantes d'un nombre sexagésimal.

Ecrire 2 solutions/implémentations (donc 2 classes différentes, mais implémentant les mêmes services : 2 constructeurs et 4 accesseurs dans les 2 classes):

- Une solution représentant les données sous la forme d'un nombre flottant.
- Une autre solution représentant les données sous la forme d'un nombre sexagésimal.

On fera bien attention à ce que les services demandés aient exactement les mêmes prototypes dans les 2 solutions proposées.