

Programmation orientée objet – Java**Travaux Pratiques***Maxime Guériau*

On souhaite modéliser et implémenter une montre multifonctions à affichage digital (heures, minutes, secondes). Pour cela, ce TP a pour but, étape par étape de :

- Définir une classe *Compteur* avec les opérations de base d'un compteur (pas de l'incréméntation, borne min et borne max).
- Définir une classe *ChaineDeCompteurs*, qui est une succession de compteurs qui fonctionnent en chaine. Incrémenter une chaine de compteurs consiste à incrémenter son premier compteur ; si celui-ci déborde, cela incrémente le suivant, et ainsi de suite.
- Définir la classe *Montre* en utilisant les classes précédentes. On utilisera la fonction *attendre1s()* pour simuler l'écoulement d'une seconde.
- Définir une classe *Réveil*, à partir des classes précédentes, qui ajoute une fonctionnalité permettant d'activer à une certaine date (heure, min) un *Accessoire* (sonnerie, une radio ou une cafetière).

Exercice 1 : Implémentation de la classe *Compteur*

Proposer une implémentation en java de la classe *Compteur* à partir des informations complémentaires suivantes. On définira ainsi :

- les attributs en privé: *int pas*, *int borneSup*, *int borneInf*, *int valeur*
- les méthodes en public, dont :
 - les constructeurs :
 - *Compteur()* : qui initialise un compteur avec une borne inférieure à 0, une borne supérieure à la valeur maximal pour un int (Integer.MAX_VALUE) et un pas à 1,
 - *Compteur(int valeur)* : qui reprend le constructeur précédent et le spécifie en définissant l'attribut valeur à la valeur passée en paramètre. *Indication* : un constructeur peut appeler un constructeur plus spécifique avec *this(...)*, en passant les bons paramètres entre parenthèses,
 - *Compteur(int borneInf, int BorneSup, int pas)* : qui initialise les attributs avec les paramètres et l'attribut valeur à la borne inférieure.
 - les accesseurs en lecture seulement pour chacun des attributs.
- les autres méthodes :
 - void *incrémenter()* : qui incrémente la valeur du compteur de la valeur du pas,
 - void *remiseAZero()*: qui affecte à la valeur du compteur la borne inférieure,
 - void *afficher()* : qui affiche la valeur du compteur (dans la console).

Prévoir un programme *Test* permettant de tester la classe *Compteur*.

Exercice 2 : Implémentation de la classe *ChaineDeCompteurs*

Proposer une implémentation en java de la classe *ChaineDeCompteurs* en utilisant un tableau de n *Compteur*. On définira dans cette classe :

- les attributs en privé : *int nbCompteurs* (*optionnel*) et *Compteur[] tabCompteurs*;
- les méthodes en public dont :
 - les constructeurs :

- *ChaineDeCompteurs(int n)* : qui initialise une chaine de n objets *Compteur* en utilisant *Compteur()* comme constructeur pour chacun des compteurs.
 - *ChaineDeCompteurs(int n, int [] valeur)* : qui initialise une chaine de n objets *Compteur* en utilisant *Compteur(int valeur)* comme constructeur pour chacun des compteurs.
 - *ChaineDeCompteurs(int n, int[] borneInf, int[] borneSup, int[] pas)* : qui initialise une chaine de n objets *Compteur* en utilisant *Compteur (int borneInf, int borneSup, int pas)* comme constructeur pour chacun des compteurs.
 - les accesseurs en lecture seulement pour chacun des attributs.
 - les autres méthodes:
 - void *incrementer()* : qui incrémente la valeur de la chaine de compteurs en utilisant la méthode *incrémenter* de la classe *Compteur*,
 - void *remiseAZero()*: qui appelle *remiseAZero()* de la classe *Compteur* sur chacun des compteurs,
 - void *afficher()*: qui appelle *afficher()* de la classe *Compteur* sur chacun des compteurs et affiche les valeurs sous la forme *val1:val2:val3:...*, etc.
- Remarque : on peut éviter d'écrire *afficher* en surchargeant la méthode *toString()* héritée de *Object*.

Modifiez votre programme *Test* (ou en créer un nouveau) afin de tester la classe *ChaineDeCompteur*.

Exercice 3 : Implémentation de la classe fille Montre

Programmer et tester la classe *Montre*, qui hérite de la classe *ChaineDeCompteur*.

Exercice 4 : Extension Réveil

Dans le cas où l'on choisit un réveil avec une sonnerie ou avec une radio, on a la possibilité d'interrompre le signal émis pendant une durée de temps prédéfinie, à l'aide d'une fonction *pause()*, au bout de laquelle le signal est émis à nouveau et ce jusqu'au prochain *pause()*, *arrêt()* ou jusqu'à atteindre la durée préprogrammée pour le signal. Par exemple, la sonnerie d'un réveil, préprogrammée pour une 1 minute, sonne pendant une minute jusqu'à ce qu'on l'arrête définitivement - *arrêt()* - ou momentanément en exécutant *pause()*. Après l'écoulement de la durée de pause, le réveil se remet à sonner tant que la minute ne s'est pas écoulée et qu'il n'ait pas été à nouveau arrêté ou mis en pause.

Proposez une implémentation (et un programme de test) de cette nouvelle fonctionnalité à partir de votre code précédent.

Exercice 5 : Gestion des erreurs

Les accessoires spéciaux comme la cafetière ne peuvent pas être arrêtés ou mis en pause une fois lancés. Proposez un système de gestion des erreurs dans votre programme à l'aide d'exceptions.

Exercice 6 : Système de log

On souhaite ajouter au réveil une fonctionnalité lui permettant de décrire son activité dans un fichier texte. Le réveil doit tenir un journal mis à jour dynamiquement, dans lequel figure tous les événements horodatés survenus (actions de l'utilisateur, exceptions, activation d'accessoires, etc.).