SAL3PROO – POO

TP n°2

Exercice 1. Créer un projet Java nommé `POOTP2` à partir d'un template (création d'une classe `Main` avec le `main`) et définir le package `exo1`.

1. Créer une classe `Client` caractérisée par les attributs suivants sans préciser de visibilité (visibilité par défaut) : `int idClient`, `String nom`, `String prenom`, `String societe`, `boolean actif`.
2. Dans le `main` de la classe `Main`, créer un client et afficher toutes ses caractéristiques.Consulter les valeurs par défaut affectées aux attributs.
3. Dans la classe `Client`, générer un constructeur ayant deux paramètres `idClient` et `nom` avec IntelliJ : Generate->Constructor-> cocher `idClient` et `nom`.
4. Dans le `main` de la classe `Main`, peut-on encore appeler le constructeur sans paramètre ? Pourquoi ?
   Modifier la ligne pour créer un client `(1, "Dupont")` et afficher toutes ses caractéristiques.
5. Définir la visibilité `private` pour tous les attributs. Que faut-il ajouter dans la classe pour consulter les informations du client dans le `main` ? Générez-le avec intelliJ (Generate). Modifier le `main`.
6. Surcharger le constructeur avec un autre constructeur prenant cinq paramètres `idClient`, `nom`, `prenom`, `societe`, `actif`. Pour éviter la redondance de code, on appellera le code du constructeur avec deux paramètres déjà défini dans le nouveau constructeur.
   Générer avec IntelliJ tous les getters et seulement les setters dont on a besoin sachant qu'on ne veut pas pouvoir mettre à jour l'identifiant du client.
   Créer un client dans le `main` avec ce nouveau constructeur et afficher ses informations.
7. On souhaite maintenant affecter un identifiant qui s'incrémente de 1 à chaque nouveau client. Le premier client recevra l'identifiant 1. On pourra créer une nouvelle classe `ClientBis` et copier/coller le code de la classe `Client`.
   Proposer une solution en gérant l'incrémentation de l'identifiant dans le constructeur.
   Créer deux clients de la classe `ClientBis` dans le `main` et afficher leurs informations.
   Proposer une seconde solution en utilisant un bloc d'initialisation.

Exercice 2. Dans le projet POOTP2, créer un package nommé exo2.
Créer une classe `Temps` permettant d'enregistrer un temps exprimé en heures, minutes et secondes, les minutes et les secondes étant comprises entre 0 et 59.
La classe `Temps` sera caractérisée par un attribut `heures` de type `long` et 2 attributs de types `int` : `minutes` et `secondes`.
Elle fournira 3 constructeurs :

- un constructeur sans paramètre,

- un constructeur ayant 3 paramètres `heures`, `minutes`, `secondes`,
- un constructeur ayant un paramètre de type `long` : `t` qui permettra d'initialiser un temps exprimé en secondes.

On souhaite pouvoir consulter et modifier les attributs heures, minutes et secondes d'un objet de type `Temps`.
Choisissez une visibilité adaptée pour les trois attributs et ajoutez des accesseurs si besoin sachant qu'on devra toujours garantir que les valeurs des attributs minutes et secondes soient bien comprises entre 0 et 59. On pourra définir une méthode `void normaliser()` qui permet de convertir un temps en heures, minutes et secondes de manière à ce que les minutes et les secondes soient bien comprises entre 0 et 59. Quelle visibilité doit-on donner à cette méthode sachant qu'elle ne doit être appelée que dans la classe `Temps`?

Ajouter les méthodes suivantes :
- `public long conversion ()` qui retourne l'entier long obtenu en convertissant les heures, minutes, secondes en secondes.
- `public void ajouterTemps (Temps t)` qui ajoute au temps le temps `t` passé en paramètre.

Exercice 3. Dans le projet POOTP2, créer un package nommé exo3. Écrire une classe `Main` dont la méthode `main` additionne les arguments de type entier donnés au programme comme arguments du `main`.

Exercice 4. Dans le projet POOTP2, créer un package nommé exo4. Écrire une classe `Main` dont la méthode `main` permet de saisir un entier positif `n` et affichant la somme des `n` premiers entiers jusqu'à ce que l'utilisateur tape `0`. On pourra utiliser la classe `java.util.Scanner` pour la lecture des entiers. On suppose que la saisie est correcte, i.e. qu'on saisit bien un entier.

Exercice 5. Dans le projet POOTP2, créer un package nommé exo5.

On considère la classe `Maillon` suivante :

```
class Maillon {
     private String valeur;
     private Maillon suivant;

     public Maillon(String s, Maillon m) {
          valeur = s;
          suivant = m;
     }
}
```

1. Définir la classe `Pile` (LIFO) dont le seul attribut privé nommé `sommet` est de type `Maillon`. La classe `Pile` fournira un constructeur sans paramètres et les méthodes suivantes :

- `public void empiler(String s)`
- `public String depiler ()`
- `public Maillon getSommet()`
- `public boolean estVide()`

•

Ajouter les getters et setters dans la classe `Maillon` si besoin.

2. Définir la classe `Main1` en écrivant un `main` qui empile les 5 premières lettres de l'alphabet, affiche le contenu de la pile, puis dépile deux fois de suite et enfin affiche le contenu de la pile.

3. Définir la classe `Main2` en écrivant un `main` qui empile à cinq reprises un entier généré aléatoirement dans l'intervalle [1..100], affiche le contenu de la pile, puis dépile deux fois de suite et enfin affiche le contenu de la pile.

Pourquoi le principe d'encapsulation n'est-il pas respecté ? Que pourrait-on vouloir changer dans l'implémentation de la classe `Pile` qui obligerait à modifier les classes `Main1` et `Main2` ? Modifier la classe `Pile` pour respecter le principe d'encapsulation et permettre, dans les classes `Main1` et `Main2`, de consulter le contenu de la pile, les éléments de la pile apparaissant entre crochets séparés par des virgules.