TP 2 partie 1 (POO)

Création de classes

Le but de ce TP est de définir et de manipuler quelques classes élémentaires. Il se compose de deux exercices, le premier a pour but de définir une classe proposant des méthodes de classe pour les saisies. Le deuxième consiste à définir des classes pour une application.

Une classe est définie par le mot clé `class` :

```
class <nom de la classe> {
  <définitions de variables d'instance>     // utilisez le mot clé private pour forcer l'encapsulation
  <définitions de variables de classe>      // utilisez les mot clés static et private
  <définitions des méthodes d'instance>  // vous pouvez utilisez public
  <définitions de méthodes de classe>       // utilisez le mot clé static
}
```

Une méthode est définie de la manière suivante :

```
static | public <type retourné> <nom méthode> ( <liste d'arguments>) {
            <déclarations et instructions>
}
```

Le mot clé `return` délivre la valeur de la méthode. Si la méthode ne délivre rien (pas d'utilisation du mot clé `return`) le type retourné par la méthode est `void` (ce qui signifie vide). L'annexe donne un exemple de définition de classe.

Exercice 1/

Question 1/ Définissez la classe `Saisir`. Cette classe propose les méthodes de classe `entier()`, `reeld()`, `reelf()`, `chaine()` et `car()` pour saisir respectivement un type primitif `int`, `double`, `float`, une instance de `String` et un type primitif `char`. Cette classe sera à rajouter dans tout vos projets que vous allez créer par la suite (ce qui vous permettra de saisir des types primitifs).

Pour saisir au clavier un type primitif il est nécessaire de mettre la saisie dans une instance de la classe `String`. Les déclarations sont :

```
String ligne;
BufferedReader entree=new BufferedReader
                          (new InputStreamReader (System.in));
```

Le code pour une saisie est :

```
// saisie d'une ligne
System.out.println("Tapez une ligne");
ligne = entree.readLine();
```

Il faut alors convertir la saisie en type primitif. Voici quelques méthodes nécessaires aux conversions :

- la méthode de classe `parseInt(String str)` de la classe `Integer` a comme argument une instance de la classe `String` (elle délivre la conversion de cette instance en type primitif `int`),

- la méthode de classe `valueOf(String str)` de la classe `Double` a comme argument une instance de classe `String` (elle délivre la conversion de la chaîne en instance de `Double`),

- la méthode d'instance `doubleValue()` délivre la conversion d'une instance de `Double` en type primitif `double`,

- la méthode de classe `valueOf(String str)` de la classe `Float` a comme argument une instance de classe `String` (elle délivre la conversion de la chaîne en instance de `Float`),

- la méthode d'instance `floatValue()` délivre la conversion d'une instance de `Float` en type primitif `float`.

Question 2/ Les méthodes que vous avez écrites dans la question précédente sont déjà implémentées en Java. Etudiez et testez la class `Scanner` de la documentation JAVA en ligne.

Exercice 2/ Simulation de la gestion d'un stock

Dans cet exercice, on souhaite gérer un stock de produits frais.

Chaque produit est décrit par sa référence (de la classe `String`) et sa date d'entrée dans le stock (de type `int`). La date d'entrée dans le stock est un entier qui représente la date Julienne (le numéro d'ordre du jour de l'année). La date du jour est une variable (int `dateJour`) déclarée et initialisée dans le *main()*.

On proposera à l'utilisateur (*cf.* question 4) un menu :

- e : entrée d'un produit dans le stock ;
- s : sortie d'un produit du stock ;
- i : incrémenter la date du jour ;
- q : quitter.

Question 1/ Proposez et testez une classe `Produit`.

Cette classe comporte notamment :

- une méthode pour afficher un produit,
- un constructeur ayant en paramètre la date du jour et lisant au clavier la référence du produit à l'aide de la classe `Scanner`. La date d'entrée en stock est la date du jour.

Question 2/ Gestion du stock de produit à l'aide d'une classe `Pile`.

A chaque entrée d'un produit dans le stock, ce produit est saisi au clavier et la date d'entrée en stock est la valeur de la date du jour (`dateJour`). Le temps passant, la fraîcheur des produits en stock diminue.

Dans cet exercice, nous partirons du principe que la gestion du stock s'effectue selon la doctrine : "le produit le plus frais, donc le plus récemment entré en stock, est sorti le premier du stock". Nous allons donc utiliser une structure de pile pour modéliser ce type de gestion de stock.

Proposez et testez une classe `Pile` ayant en variables d'instance un tableau de `Produit` et son indice, et comportant les méthodes suivantes :

- le constructeur `Pile(int max)` qui réservera la place pour *max* produits et initialisera l'indice du tableau ;
- `boolean pilevide()`, délivrant `Vrai` si la pile est vide, `Faux` sinon ;
- `boolean pilepleine()`, délivrant `Vrai` si la pile est pleine, `Faux` sinon ;
- `void empiler(Produit p)`, permettant de stocker un nouveau produit `p` au sommet de la pile :

- `void depiler()`,permettant de sortir le produit situé au sommet de la pile ;
- `Produit sommet()`, délivrant le produit au sommet de la pile.

Question 3/ Proposez et testez une classe `Stock`.

La classe `Stock` a en variable d'instance une `Pile` (encapsulée et inaccessible à l'utilisateur) et comporte les méthodes suivantes :

- le constructeur `Stock(int taille)` qui réserve un stock de `taille` produits.
- `void entrer(produit p)`, qui permet d'entrer un produit `p` dans le stock ;
- `void sortir(int dateJ)`, qui sort le produit le plus frais du stock. A la sortie d'un produit du stock, il ne peut être vendu si sa date d'entrée dans le stock est supérieure de 5 jours à celle de la date du jour `dateJ`. Dans un tel cas, on sortira successivement de la pile tous les produits ne pouvant être vendus.

Question 4/ Proposez et testez une classe `GestionStock`.

La classe `GestionStock` comporte :

- une méthode statique `static void afficheMenu()`, permettant d'afficher le menu ;
- la méthode principale `public static void main(String [] args) throws IOException` qui permet de simuler la gestion du stock (on utilisera la classe `Scanner`).

Annexe

Voici une classe définissant des personnes :

```
class Personne
{
	// variable d'instance
	// donc privees

	private String nom;
	private int age;

	// variable de classe
	// donc static

	static private int nb=0;

	// constructeur : de meme nom que la classe
	// eventuellement plusieurs constructeurs

	public Personne(String lenom, int lage){
		nom=lenom;
		age=lage;
		 nb++;
	}

	public Personne(){
		nom="essai";
		age=10;
		nb++;
	}
	// Methodes d'instance

	public int obtenirAge(){
		return this.age;
	}

	public String obtenirNom(){
		return this.nom;
	}

	public void modifierNom(String nouveauNom){
		this.nom=nouveauNom;
	}

	public void modifierAge(int nouvelAge){
		this.age = nouvelAge;
	}

	// Methode de classe

	static int obtenirNb(){
		return nb;
	}

}
```