M2103-- TD2

Dans ce TD nous aurons les classes suivantes : Etudiant, Console, JeuVideo, ainsi qu'une classe principale TD2. L'idée sera de donner la possibilité à un étudiant d'avoir une console, sur laquelle il pourra installer et désinstaller des jeux, jouer aux jeux (seul ou avec d'autres étudiants), etc.

Exercice I

Nous allons partir sur une classe `JeuVideo`. Les objets de cette classe auront les attributs suivants : un attribut `nom` de type `String`, un attribut `nombreJoueurs` de type `int`, un attribut `prix` de type `double`, un attribut `typeConsole` de type `String`, ainsi qu'un attribut `tailleInstallation` de type `double`.

1. Ecrivez un constructeur pour cette classe, qui aura la signature `JeuVideo(String, int, double, String, double)`. Ce constructeur instancie les valeurs des attributs de la classe `JeuVideo` aux valeurs données en entrée du constructeur (en ordre).

   Attention : cela veut dire qu'on suppose qu'on joue chaque jeu avec un certain nombre de joueurs à chaque fois : un joueur OU deux joueurs (mais pas les deux).
   De plus, on suppose que chaque objet de type `JeuVideo` sera utilisable pour seulement un type de console !

2. Nous aurons besoin de cinq méthodes (attention, on ne les écrira pas !) qui retourneront les valeurs de chaque attribut. Quels seront les signatures de ces cinq méthodes ?

3. Supposons l'existence d'une méthode `String toString()` qui pour chaque jeu retourne le texte suivant :

   `Jeu[<nom>, <nombreJoueurs> joueurs, <prix> euros, <typeConsole>, <tailleInstallation> Go]`

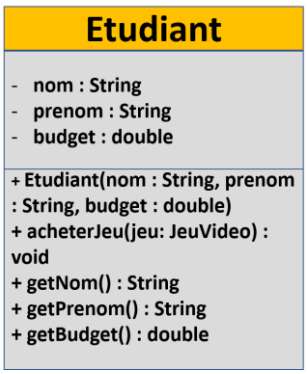
   Dans une autre classe TD2, dans une méthode `main` nous avons le code suivant :

```
	final JeuVideo marioKart = new JeuVideo("Mario Kart", 2, 19.99, "Wii U", 25.50);
	System.out.println(marioKart);
```

   Décrivez dans vos propres mots ce que fait chaque ligne de ce code. Est-ce que le code compile ? Est-ce qu'il s'exécute (quel est le résultat de l'exécution) ?

Exercice II

Nous allons avancer à une deuxième classe, `Etudiant`. Un etudiant (un objet de type `Etudiant`) aura la possibilité d'acheter des jeux à partir d'un budget qu'il a. Plus tard, on lui donnera la possibilité d'acheter des consoles, de faire installer ses jeux, et puis de jouer également. Mais, pour cet exercice notre but sera de réaliser la classe décrite par le diagramme ci-dessous.



1. Nous allons considérer qu'on a déjà (correctement) écrit le code de toutes les méthodes de la classe étudiant sauf la méthode `void acheterJeu(JeuVideo jeu)`. Cette dernière doit vérifier si le paramètre en entrée est `null` (référence à un objet non-existant). Si c'est le cas, on affiche le texte suivant : `Veuillez indiquer un jeu valide.` Sinon, on vérifie si le budget restant à l'etudiant suffit pour acheter le jeu (par rapport à la valeur stockée par l'attribut prix de l'objet `jeu`). Si c'est le cas, on enlève le montant du prix du budget et on affiche le message : `Felicitations. Vous avez bien achete le jeu <nom du jeu> !` Si le budget ne suffit pas, affichez le message : `Le prix de ce jeu depasse votre budget !`

   NB : Je n'utilise pas d'accents dans Java -- on ne sait jamais ce que va se passer.

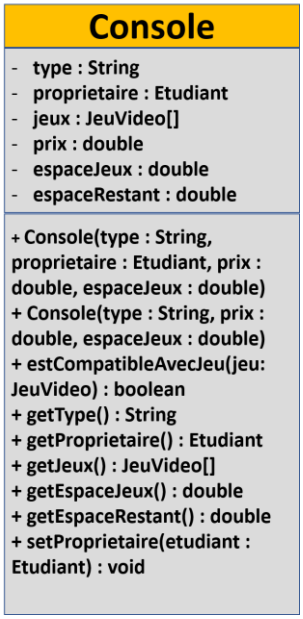
2. Pourquoi affiche-t-on des messages d'erreur variés dans la méthode `acheterJeu` ?
3. Donnez le code (à mettre dans la classe TD2 dans la méthode `main`) qui permet de créer deux etudiants avec les caractéristiques suivantes :
   - Un premier etudiant Jean Dupont (prenom nom), avec un budget de 400 euros
   - Un deuxième etudiant Marie Lagrange (prenom nom), avec un budget de 550 euros

Exercice III

Pour cet exercice nous allons créer une classe `Console`, dont le diagramme est donnée ci-dessous.

Les objets de type Console auront notamment six attributs :

- Un attribut `type` de type `String`
- Un attribut `proprietaire` de type `Etudiant` (celui-ci peut être mis à `null` si la console n'a pas encore un propriétaire)
- Un attribut `jeux` de type `JeuVideo[]` : **on va supposer qu'une console ne peut pas stocker plus que 50 jeux vidéo en même temps**.
- Un attribut `prix` de type `double` : celui-ci sera le prix, en euros, de la console
- Deux attributs de type `double` : `espaceJeux` et `espaceRestant`. Le premier stocke l'espace total (en Go) que vous avez pour installer un jeu sur la console, le deuxième stocke l'espace restant (après un certain nombre d'installations). Le deuxième de ces deux attributs est susceptible de changer dans le cours de l'exécution, tandis que le premier restera constant.



1. Trouvez le(s) constructeur(s) sur le diagramme de classe.
   Pour le premier constructeur, nous voulons initialiser les valeurs des attributs `type`, `proprietaire`, `prix` et `espaceJeux` aux valeurs données en entrée (en ordre). De plus, on va initialiser le tableau `jeux` comme étant un tableau de type `JeuVideo[]` de taille 50 (pourquoi cette taille ?). Finalement, l'attribut `espaceRestant` sera initialisé à la même valeur que `espaceJeux`.

   Ecrivez ce constructeur.

2. Supposons que dans une méthode main d'une classe TD2 on voulait déclarer et instancier une variable playstation4 de type `"PS4 Slim"`, sans propriétaire (la valeur de `proprietaire` est mise à `null`), avec un espace de jeux de 500 Go. Ecrivez le code dont on a besoin pour déclarer et instancier cette variable.

3. Dans l'exercice précédent, nous avons appelé le premier constructeur de la classe Etudiant en mettant l'attribut `proprietaire` à `null`. Cette situation (devoir instancier une console qui n'a pas encore de propriétaire) se répète souvent en réalité ; c'est pourquoi c'est intéressant d'écrire un deuxième constructeur qui mettra cet attribut par défaut à `null`.

   Sans copier aucune ligne du code de votre premier constructeur, écrivez ce deuxième constructeur.

4. La méthode `public boolean estCompatibleAvecJeu(JeuVideo jeu)` retourne `true` si le type de cette console est le même que le type de console indiquée parmi les attributs de l'objet `jeu`, et `false` autrement. Ecrivez cette méthode.

5. Supposons maintenant que toutes les méthodes indiquées dans le diagramme de classe sont déjà (correctement) écrites. Dans la méthode `main` de notre classe TD2 on a déjà mis le code suivant :

```
final Etudiant jeanDupont = new Etudiant("Dupont", "Jean", 400);
final Etudiant marieLagrange = new Etudiant("Lagrange", "Marie", 550);
```

   Ecrivez le code qui nous permet maintenant de créer ces deux consoles :
   - Un objet `wiiU` de type `Console` qui a le `typeConsole` "`Wii U`" sans propriétaire, qui coûte 150 euros et a 350 Go de place.
   - Un objet `ps4` de type `Console` qui a le `typeConsole` "PS4" qui appartient à l'etudiant Marie Lagrange, qui coûte 250 euros et a 450 Go de place.

6. On ajoute le code suivant au code écrit dans la question précédente :

```
final JeuVideo marioKart = new JeuVideo("Mario Kart", 2, 19.99, "WiiU",
25.50);
```

   Si on appelle ensuite la méthode `estCompatibleAvecJeu(marioKart)` pour les objets `wiiU` et `ps4` créés dans la question précédente, quels seront les deux résultats ?

7. Dans la classe `Etudiant` on veut maintenant écrire une méthode qui permet à un objet de ce type de devenir le propriétaire d'un objet de type Console. Cette méthode aura la signature `void acheterConsole(Console console)`. Nous voulons que cette méthode fonctionne ainsi : si le paramètre en entrée est valide (non-`null`) et si l'étudiant en question a assez d'argent, alors il deviendra le propriétaire de la console en question (et il paiera le prix indiqué pour la `console`). Si le paramètre en entrée n'est pas valide (il est `null`) alors on affiche un message d'erreur indiquant qu'il faut bien saisir un paramètre valide en entrée. Finalement, si le paramètre donné est valide, mais l'etudiant n'a pas assez d'argent, alors on retourne un autre message d'erreur indiquant cela.
   Complétez le code suivant pour cette méthode :

```
public void acheterConsole(Console console) {
	if (null != console && this.budget >= console.getPrix()) {
```

```
                // A completer

        }
        else {
                if (// A completer) {
                        System.out.println("Pas assez d'argent !");
                }
                else {
                        System.out.println("Il faut bien specifier une console !");
                }
        }
}
```

Exercice IV

Nous avons plusieurs choix lorsqu'on veut programmer les interactions entre les trois classes, notamment l'installation/la désinstallation des jeux sur la console, jouer un jeu, etc. Par exemple, un choix pourrait-être de mettre ce code dans la classe `Etudiant` (car on dit que c'est un `etudiant` qui fait installer le jeu sur la console). Nous allons choisir autrement : notamment, on va inclure la partie principale du code d'installation et de jeu dans la classe `Console`, car en fait installer le jeu a plus à faire avec cette classe (l'espace restant, le type de console, etc.).

1. Dans la classe `Console`, nous aurons une méthode `void installerJeu(JeuVideo jeu)`. On va commencer par se demander si la console et le jeu sont compatibles (sinon, on envoie un message d'erreur décrivant cela). Puis, on va chercher si on a encore de la place par rapport au nombre de jeux déjà existants sur la console -- sinon, on envoie encore un message d'erreur qui indique cela. Si on a assez de place, on fait installer le jeu : le jeu est ajouté au tableau jeux de l'objet de type `Console`, on diminue l'espace restant sur la console par la place prise par le jeu (voir l'attribut `tailleInstallation` dans la classe `JeuVideo`), et on affiche un message qui indique le succès de cette méthode.

   Ecrivez le code de cette méthode.

2. Dans la classe `Console` nous aurons également une méthode `void jouer(Etudiant[] joueurs, JeuVideo jeu)`, qui devrait simuler l'action des étudiants dans le tableau `joueurs` de jouer au jeu indiqué dans le deuxième paramètre. Dans cette méthode il faut vérifier si le jeu indiqué est valide ; dans ce cas on vérifie si le nombre de joueurs indiqué par le paramètre joueurs est bien le même que le nombre de joueurs indiqué par l'attribut correspondant dans la classe `JeuVideo`. Si tout va bien on affiche un message de succès, sinon on envoie un message d'erreur.

   Complétez le fragment de code ci-dessous avec le bon code :

```
	public void jouer(Etudiant[] joueurs, JeuVideo jeu) {
		if (null == jeu) {
			System.out.println("On ne peut pas jouer a un jeu non-valide.");
		}
		else {
			if (// A COMPLETER) {
				// A COMPLETER


			}
			else {
				System.out.println("On commence le jeu. Amusez-vous bien
!");
			}
		}
	}
```

3. Dans la classe TD2, dans la méthode main nous voulons écrire du code qui crée un nouveau tableau de deux étudiants, `jeanDupont` et `marieLagrange`.

4. Dans la classe TD2, dans la méthode main nous supposons qu'on a rajouté le code suivant :

   ```
   final JeuVideo fortnite = new JeuVideo("Fortnite", 1, 39.99, "PS4", 25.50);
   ```

   Ecrivez du code qui permet à jeanDupont de jouer à Fortnite tout seul. Puis écrivez du code qui permet à jeanDupont et marieLagrange de jouer ensemble à Mario Kart.