Contrôle 4 - Créer un jeu Tamagotchi en Java

Ce Travail Pratique intégre les principaux concepts de la **Programmation Orientée Objet (POO)** : héritage, encapsulation, polymorphisme, abstraction et interface. Il inclue aussi la construction de **Diagrammes UML** (Diagramme de Cas d'Utilisation, Diagramme de Classes (exemple ci-dessous), Diagramme Séquentiel et Diagramme de Relation d'Entités).

Objectifs du TP :

- Créer un jeu Tamagotchi avec des animaux virtuels.
- Utiliser des méthodes et classes statiques pour affiner le fonctionnement de l'application.
- Implémenter l'héritage pour créer différentes sortes de Tamagotchis.
- Utiliser l'encapsulation pour protéger les données sensibles des Tamagotchis.
- Appliquer le polymorphisme pour gérer différentes actions possibles de manière générique.
- Utiliser l'abstraction pour définir des classes de base et des comportements génériques.
- Créer des interfaces pour définir des comportements spécifiques.
- Créer les diagrammes par rétro-conception dans Umbrello.

Exemple de Diagramme de Classes décrivant l'application:

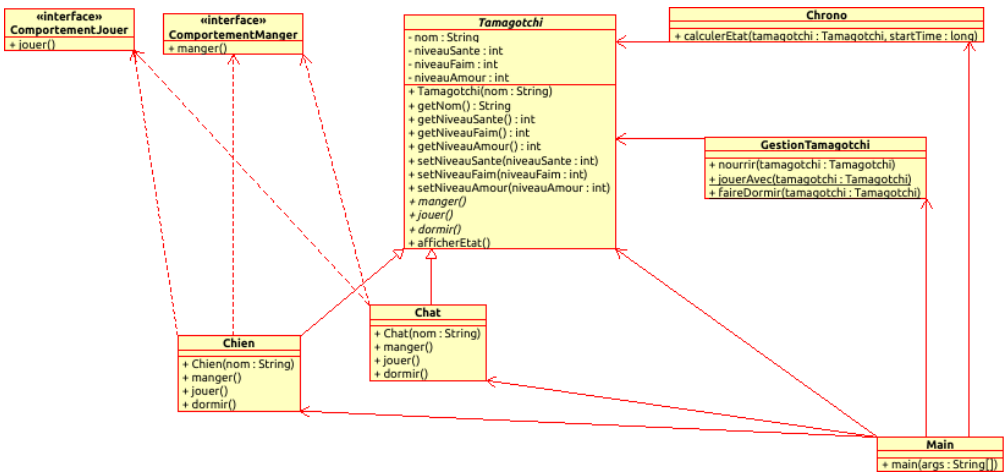

Figure 1: "Diagramme de Classes"

Structure du projet :

1. **Classe `Tamagotchi` (classe de base abstraite)** :
   - Représente un Tamagotchi de manière générale.
   - Définit des méthodes abstraites comme `manger()`, `dormir()`, `jouer()`.
   - Définit les méthodes getter/setter.
2. **Interfaces** :
   - `ComportementManger` : définira une méthode `manger()`.
   - `ComportementJouer` : définira une méthode `jouer()`.
3. **Classes dérivées** :
   - `Chat` : représente un Tamagotchi de type Chat.
   - `Chien` : représente un Tamagotchi de type Chien.
4. **Classe `GestionTamagotchi`** : pour gérer l'état du Tamagotchi (santé, faim, etc.).
5. **Classe `Chrono`** : pour gérer l'état temporel du Tamagotchi.
6. **Classe `Main`** : classe de test pour exécuter le jeu.

Explication du Code à implémenter :

1. **Méthodes et Classes Statiques** : Les classes `GestionTamagotchi` et `Chrono` permettent d'intervenir sur l'état de chaque instance (`monChat` et `monChien`) durant toute la session de l'application.
2. **Héritage** : Les classes `Chat` et `Chien` héritent de la classe abstraite `Tamagotchi` et implémentent des comportements spécifiques via les interfaces `ComportementManger` et `ComportementJouer`.
3. **Encapsulation** : Les variables privées (`nom`, `niveauSante`, `niveauFaim`, etc.) sont protégées et accessibles uniquement via des méthodes `get` et `set`.
4. **Polymorphisme** : Les méthodes `manger()`, `jouer()`, et `dormir()` sont définies de manière polymorphique dans les classes filles. Chaque type de Tamagotchi peut avoir une implémentation différente de ces actions.

5. **Abstraction** : La classe `Tamagotchi` définit des méthodes abstraites que les classes filles doivent implémenter, ce qui permet de créer des objets Tamagotchi génériques sans connaître le type exact à l'avance.
6. **Interfaces** : Les interfaces `ComportementManger` et `ComportementJouer` permettent d'ajouter des comportements spécifiques à chaque Tamagotchi.

Remarque :

Je vous laisse libre de déterminer les niveaux pour chaque instance ainsi que pour `calculerEtat()` en terme d'écoulement de temps et sa conséquence sur chaque niveau. Pour ma part j'ai utilisé les valeurs suivantes:

Pour `Chien`, `niveauAmour` a pour défaut **100**, `jouer()` correspond à une valeur de **+15**, `calculerEtat()` conduit à un `niveauAmour` de **-1** par minute écoulée.

Pour `Chat`, `niveauAmour` a pour défaut **100**, `jouer()` correspond à une valeur de **+10**, `calculerEtat()` conduit à un `niveauAmour` de **-1** par minute écoulée.

Pour `Chien`, `niveauFaim` a pour défaut **0**, `manger()` correspond à une valeur de **-15**, `calculerEtat()` conduit à un `niveauFaim` de **+1** par minute écoulée.

Pour `Chat`, `niveauFaim` a pour défaut **0**, `manger()` correspond à une valeur de **-20**, `calculerEtat()` conduit à un `niveauFaim` de **+1** par minute écoulée.

Pour `Chien`, `niveauSante` a pour défaut **100**, `dormir()` correspond à une valeur de **+20**, `calculerEtat()` conduit à un `niveauSante` de **-1** par minute écoulée.

Pour `Chat`, `niveauSante` a pour défaut **100**, `dormir()` correspond à une valeur de **+30**, `calculerEtat()` conduit à un `niveauSante` de **-1** par minute écoulée.