	Université de Corse - Pasquale PAOLI
	Diplôme : MASTER 1 Informatique DFS-DE ***2024-2025***
	Cours DESIGN PATTERNS
	TD N°2 : Design Patterns Gof Créationnels
	Enseignant : Evelyne VITTORI

Exercice 1 - Jeu d'aventure – Factory method

On dispose du diagramme de classe suivant

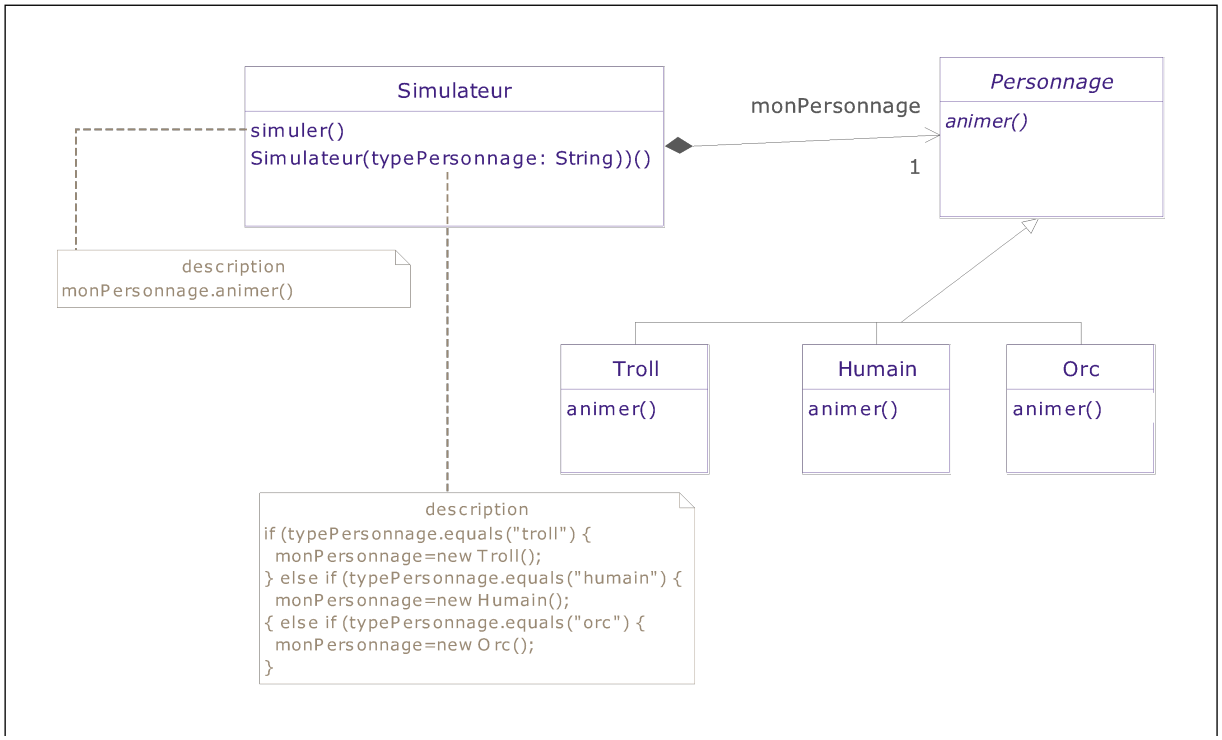

Figure 1 : Diagramme de classe Jeux Aventures

Remarque : *La relation de composition entre la classe Simulateur et la classe personnage oblige la réalisation de l'instanciation du personnage dans le constructeur de la classe Simulateur.*

Question 1 : Coder en Java une version simplifiée des classes de la figure 1 :

1. Ajoutez un attribut nom dans la classe Personnage et modifiez en conséquence le constructeur de la classe Personnage et de la classe Simulateur.
2. Les méthodes animer() des classes concrètes Troll, Humain et Orc afficheront simplement un message dans la console « Le Troll de nom xxx s'anime », « L'humain de nom xxx s'anime » ou « L'orc de nom xxx s'anime ».
3. Ajoutez une classe Test contenant la méthode main() permettant de créer un simulateur comportant un personnage de type troll nommé Diablo et de lancer la simulation.

Question 2 : Quels sont les inconvénients de la conception de la classe Simulateur présentée dans diagramme de classe de la figure 1 dans l'hypothèse où l'on souhaite pouvoir faire évoluer le logiciel en ajoutant de nouvelles catégories de personnages?

Question 3 : Proposez une solution utilisant le pattern « Factory method » pour améliorer la conception de la figure1 :

1 - Dessinez le diagramme de classes.
2 - Coder en Java votre solution.
3 - Expliquez comment votre solution permet d'éviter les inconvénients évoqués en question2.

Exercice 2 - Jeu d'aventure (suite) – Abstract Factory

On suppose à présent que les personnages possèdent des accessoires qu'ils utilisent pour s'animer.
Les accessoires sont de différents types. On distingue les armes, les sacs et les costumes.
Les armes peuvent être des épées, des battons ou des dagues. Les sacs sont des sacs à dos ou des besaces. Les costumes sont des robes ou des uniformes.
Comme le montre le diagramme de classe de la figure 2, chaque personnage possède exactement trois accessoires : une arme, un sac et un costume.

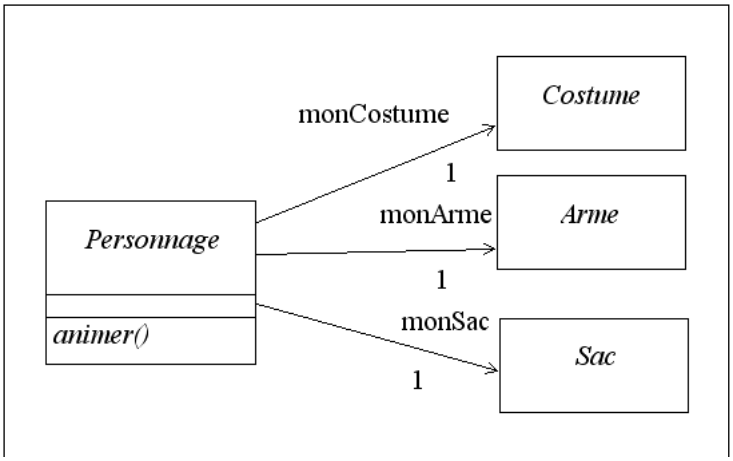

Figure2 : Personnages et Accessoires

Les combinaisons possibles des accessoires sont limitées. Pour l'instant, on distingue deux combinaisons : l'équipement de base (épée+besace+robe) et l'équipement de luxe (baton+sac à dos +uniforme).
Dans le cadre du simulateur, on supposera que les Trolls et les Orcs sont créés avec un équipement de base et les humains avec un équipement de luxe. Cette option n'est pas définitive. On devra offrir la possibilité de changer cette initialisation par défaut sans avoir à modifier les classes concrètes Troll, Orc et Humain.

Question 1 : Proposez un diagramme de classe prenant en compte ces nouveaux besoins en utilisant le pattern « Abstract Factory » au niveau de la création des combinaisons d'accessoires.

Indications

- La classe Simulateur et la classe Test de la question3 ne devront pas être modifiées.

- Le constructeur de la classe Personnage doit permettre l'initialisation des trois accessoires.
- La conception doit permettre d'envisager facilement l'ajout d'un type d'équipement supplémentaire correspondant à une nouvelle combinaison d'accessoires sans modifier la classe Personnage.

Question 2 : Coder en Java votre solution en insérant des affichages consoles permettant de visualiser les équipements affectés aux personnages lors de leur création.

Question 3 : Définissez un diagramme de séquence représentant l'exécution du programme de test localisé dans la classe Test. Ce programme doit créer une instance de la classe Simulateur comportant un personnage de type troll et lancer la simulation. Vous mentionnerez précisément les différentes les différentes instances impliquées (personnages, accessoires,…).

Exercice 3 - Jeu d'aventure (suite) – Builder

Question 1 : On doit enrichir la classe Simulateur en lui associant un décor (cf. figure 3).

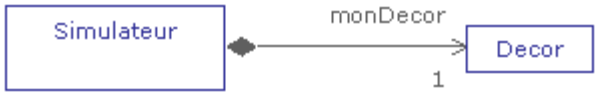

Figure3 : Simulateur et Decor

Un décor est un objet complexe qui a été défini et implémenté par une équipe de concepteurs spécialisés. D'après les informations qu'ils ont bien voulu nous communiquer, nous savons qu'un décor est composé de plusieurs éléments (des édifices, des chemins et des végétaux). Dans un premier temps, ils doivent nous fournir une version simple des éléments mais d'autres versions correspondant à différents styles (fantastique, design, …) devraient par la suite être disponibles pour construire notre décor.
Le problème est que nous ignorons comment ces éléments sont assemblés pour construire un objet décor. Les concepteurs veulent en effet se réserver la possibilité de changer l'implémentation de l'assemblage sans avoir à nous consulter. Ils ne veulent donc pas nous donner accès aux instances des objets composant le décor (Edifices, ect…).
Cependant, lors de la création d'une instance de Simulateur, nous devons pouvoir créer un objet Décor en précisant ces éléments constitutifs.
Nous avons donc demandé aux concepteurs du décor de nous aider à trouver une solution nous permettant de construire notre décor. Ils nous ont dit : « Il faut utiliser le pattern **Builder** !!» et sans plus d'explication, ils nous ont communiqué l'interface IDecor (cf. figure4).

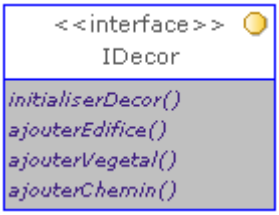

Figure 4 : Interface IDecor

Question 1 : Reportez vous au diagramme général définissant le pattern Builder vu en cours.

1. Identifiez à quel élément correspond l'interface IDecor. Modifiez son nom afin que son rôle soit plus explicite relativement au pattern Builder.
2. Complétez l'interface IDecor par une méthode importante qui a été omise.

Question 2 : Définissez un diagramme de classe montrant comment appliquer le pattern Builder à la construction du décor associé au simulateur.
Remarque: vous pourrez remplacer l'interface IDecor par une classe abstraite si vous le jugez opportun.

Question 3 : Pour pouvoir tester le fonctionnement du simulateur, nous devons demander aux concepteurs du décor de nous fournir au moins une classe supplémentaire. Laquelle ?
Par la suite, ils ont promis de nous en fournir d'autres, les quelles ?
Ajoutez ces classes dans votre diagramme de la question 5.2.

Question 4 : Coder en Java votre solution en insérant simplement des affichages console à chaque étape de création du décor.

Question 5 : Définissez un diagramme de séquence représentant l'exécution du programme de test permettant de créer une instance de la classe Simulateur et de lancer la simulation. Vous mentionnerez notamment les différentes étapes de la création du décor et les différentes instances impliquées.
NB : Ne mentionnez pas les instances relatives aux personnages évoquées dans les questions précédentes.

Exercice 4 - Singleton

Dans le cadre d'une application bancaire, on souhaite pouvoir afficher toutes les opérations (dépôt et retrait) effectuées sur les différents comptes. Pour cela, on vous demande d'implémenter une classe Journal ayant pour attribut une chaine de caractère contenant les informations de toutes les opérations sur l'ensemble des comptes.
Cette classe devra implémenter le pattern Singleton afin de garantir que le programme de test ne pourra utiliser qu'une seule et même instance de la classe Journal.

- Récupérer les classes Compte, CompteCourant, CompteLivret et Prog sur l'ENT.

- Créez la classe Journal et complétez les classes existantes afin que le programme de test de la classe Prog provoque l'affichage console ci-dessous.

```
                    JOURNAL UNIQUE POUR TOUS LES COMPTES

19/10/2024 INFO Dépôt de 100.0 Euros sur le compte 123
19/10/2024 INFO Echec Retrait de -80.0 Euros sur le compte 123 : Somme
négative
19/10/2024 INFO Tentative Retrait de 2000.0 Euros sur le compte  123
19/10/2024 INFO Echec Tentative Retrait de 2000.0 Euros sur le compte 123 :
Decouvert dépassé
19/10/2024 INFO Tentative Retrait de 1000.0 Euros sur le compte  456
19/10/2024 INFO Succés Retrait de 1000.0 Euros sur le compte 456
```