Université de Corse - Pasquale PAOLI

Master DFS-DE 1ère année

Cours Patterns

Eléments de correction TD N°2 : Design Patterns Gof Créationnels

Enseignant : Evelyne VITTORI

Exercice 1 - Jeu d'aventure

Question 1 : Coder en Java une version simplifiée des classes de la figure 1 : (cf. code)

- Ajoutez un attribut nom dans la classe Personnage et modifiez en conséquence le constructeur de la classe Personnage et de la classe Simulateur.
- 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 ».
- 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?

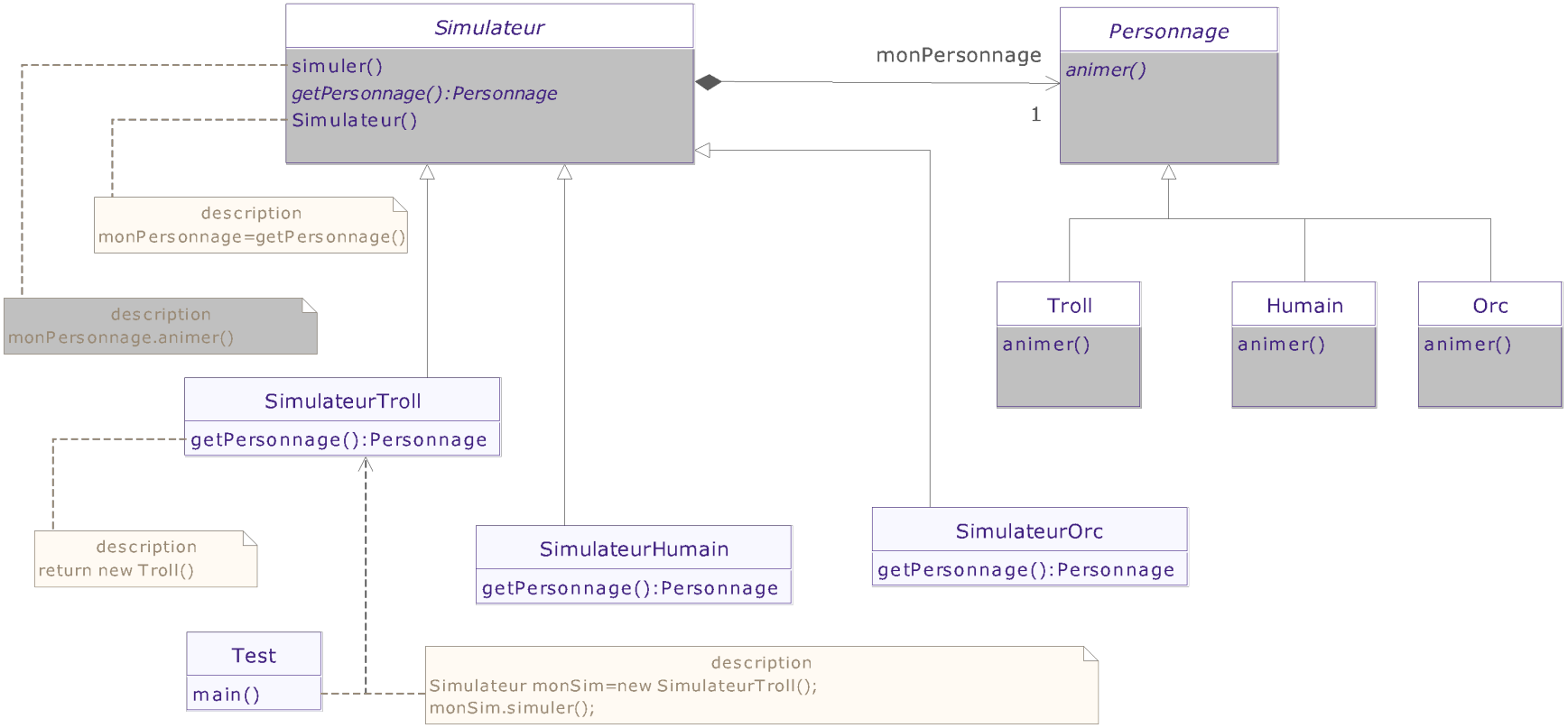
Le diagramme ne permet pas facilement de faire évoluer la conception. Si l'on ajoute une nouvelle catégorie de personnages, cela oblige à remanier le code. Le problème vient du fait que la classe simulateur est couplée aux classes concrètes Troll, ... La classe Simulateur ne doit pas dépendre des classes de personnages concrets.

Elle doit respecter le principe Ouvert/Fermé : pour étendre ses fonctionnalités, on ne doit pas avoir à modifier son code (« elle est fermée ») mais la conception doit permettre de le faire néanmoins en ajoutant des sous-classes par exemple (« elle est ouverte ».

Une solution pourrait consister à instancier le personnage dans la classe Test mais cela ne traduirait pas le lien de composition qui oblige à intégrer l'instanciation dans le simulateur.

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

1 - Dessinez le diagramme de classes en détaillant sous forme de notes les portions de codes vous paraissant significatives.



2 - Coder en Java votre solution.

3 - Expliquez comment votre solution permet d'éviter les inconvénients évoqués en question2.

La classe Simulateur n'est plus couplée qu'à la classe abstraite Personnage. La classe Test en revanche est couplée aux classes concrètes de simulateurs.

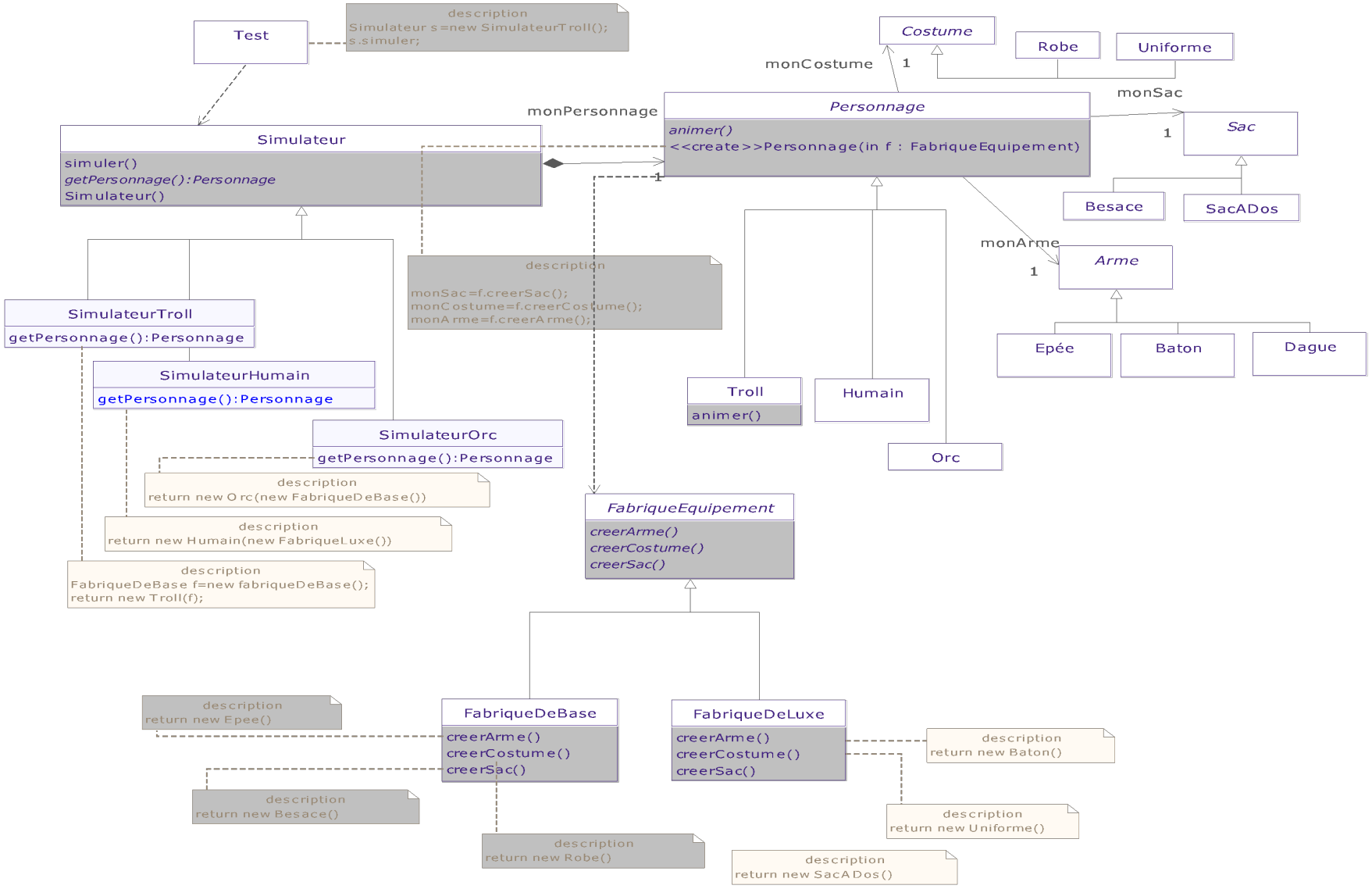
Si un nouveau type de personnage est envisagé, les classes existantes (personnage et simulateur) n'ont pas à être modifiées, elles respectent donc le principe Ouvert/Fermé. Il faut juste définir une nouvelle sous-classe de personnage et un nouveau simulateur concret.

Question 4 : On suppose à présent que les personnages possèdent des accessoires qu'ils utilisent pour s'animer.

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.

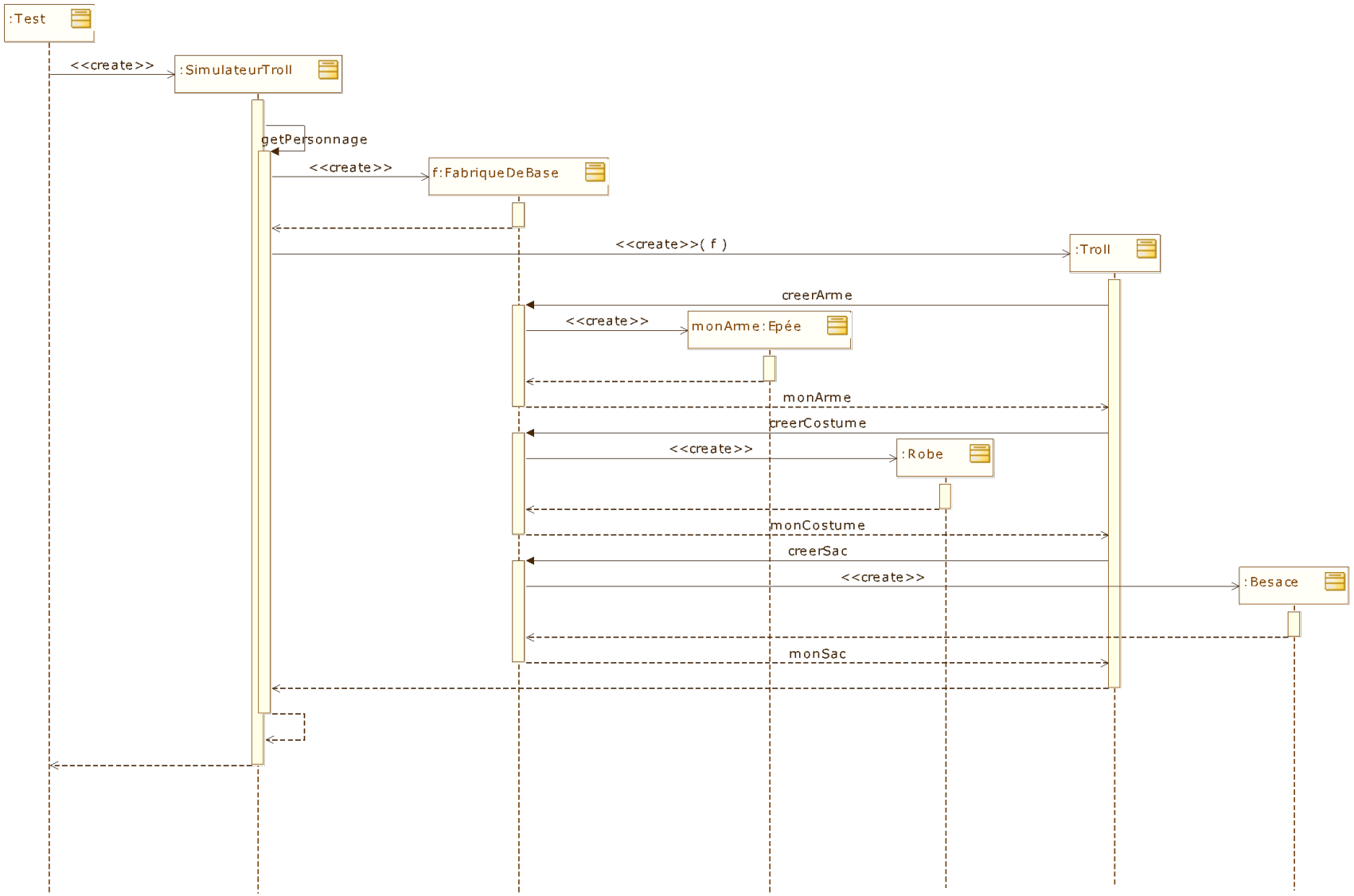
Indications

- 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.


Test
description
Simulateur s=new SimulateurTroll();
s.simuler;
Costume
Robe
Uniforme
monCostume
1
monSac
monPersonnage
Personnage
animer()
<<create>>Personnage(in f : FabriqueEquipement)
Sac
1
Simulateur
simuler()
getPersonnage():Personnage
Simulateur()
1
Besace
SacADos
monArme
1
Arme
description
monSac=f.creerSac();
monCostume=f.creerCostume();
monArme=f.creerArme();
SimulateurTroll
getPersonnage():Personnage
SimulateurHumain
getPersonnage():Personnage
SimulateurOrc
getPersonnage():Personnage
Troll
animer()
Humain
Orc
Epée
Baton
Dague
description
return new Orc(new FabriqueDeBase())
description
return new Humain(new FabriqueLuxe())
description
FabriqueDeBase f=new fabriqueDeBase();
return new Troll(f);
FabriqueEquipement
creerArme()
creerCostume()
creerSac()
description
return new Epee()
FabriqueDeBase
creerArme()
creerCostume()
creerSac()
FabriqueDeLuxe
creerArme()
creerCostume()
creerSac()
description
return new Baton()
description
return new Besace()
description
return new Uniforme()
description
return new Robe()
description
return new SacADos()

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

2 – 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,…).

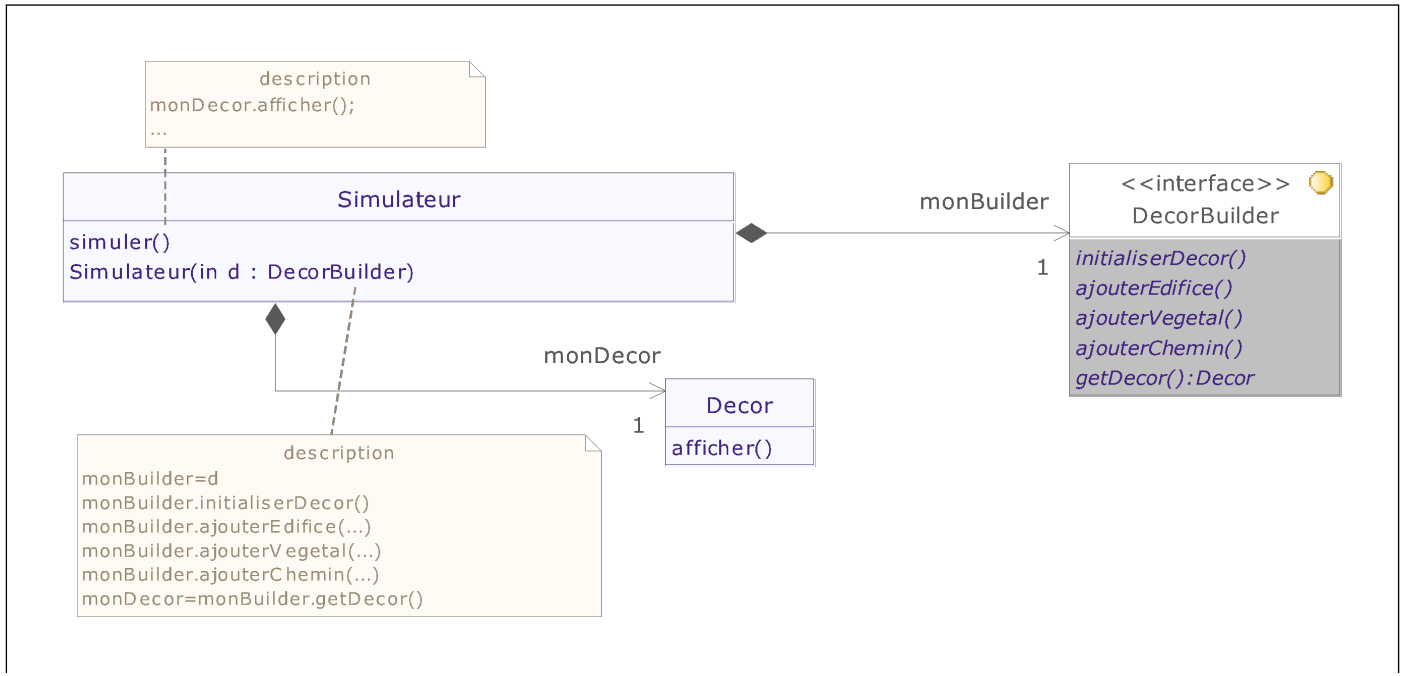

:Test
<<create>>
:SimulateurTroll
getPersonnage
<<create>>
f:FabriqueDeBase
<<create>>(f)
:Troll
creerArme
<<create>>
monArme:Epée
monArme
creerCostume
<<create>>
:Robe
monCostume
creerSac
<<create>>
:Besace
monSac

Question 5 : On doit enrichir la classe Simulateur en lui associant un décor (cf. figure 3).

Question 5.1 – Reportez vous au diagramme général définissant le pattern Builder vu en cours.

1. Identifiez à quel élément correspond l'interface IDecor : *IDecor correspond à l'élément (classe abstraite ou interface) Builder (ou Monteur).*
   Modifiez son nom afin que son rôle soit plus explicite relativement au pattern Builder : *son nom devient DecorBuilder*
2. Complétez l'interface IDecor par une méthode importante qui a été omise*: la méthode omise est celle qui renvoie le décor à la fin de la construction soit la méthode getDecor : Decor.*

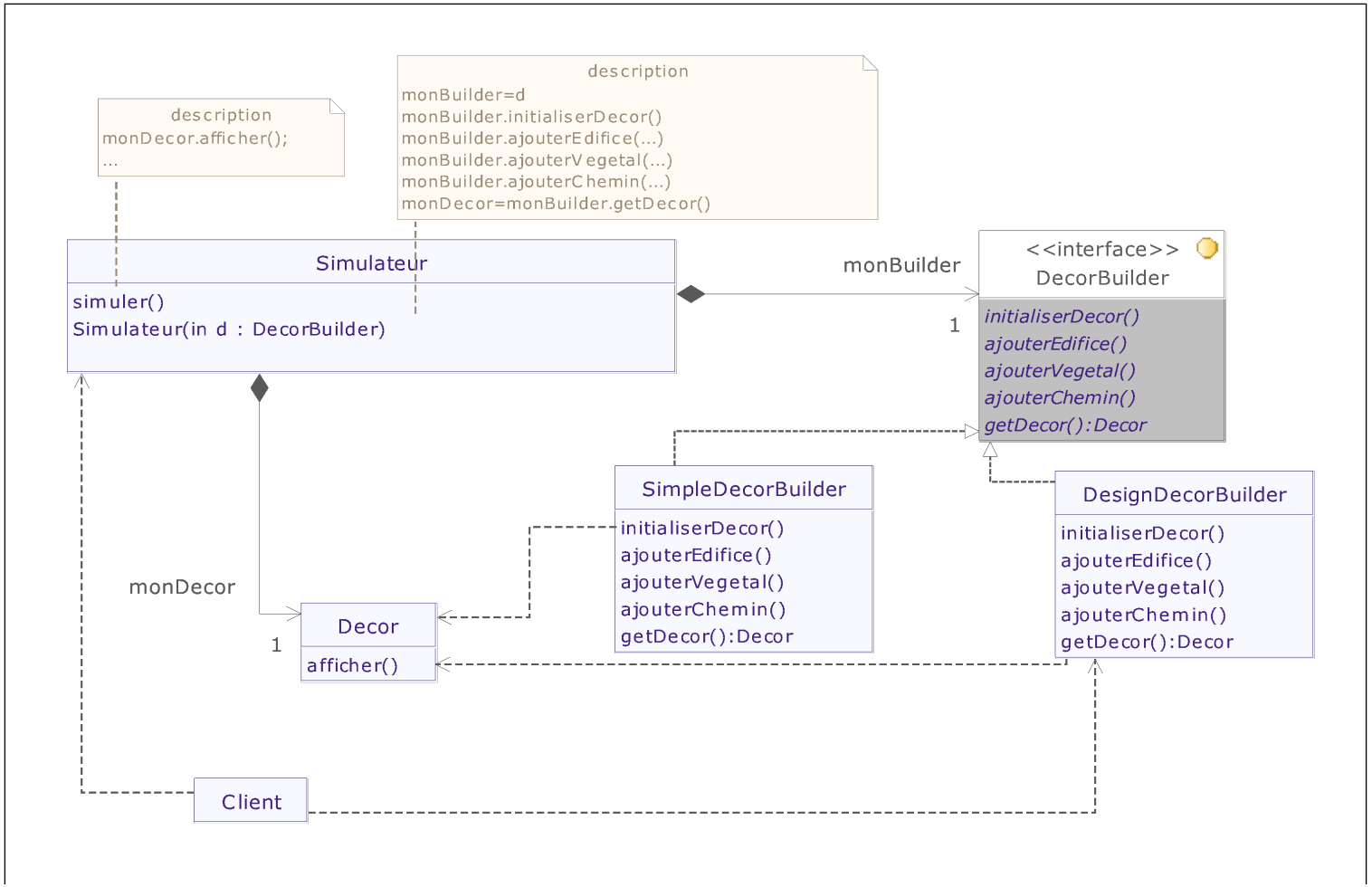
Question 5.2 – Définissez un diagramme de classe montrant comment appliquer le pattern Builder à la construction du décor associé au simulateur.



Question 5.3 – Pour pouvoir tester le fonctionnement du simulateur, nous devons demander aux concepteurs du décor de nous fournir au moins une classe supplémentaire. Laquelle ? *Une classe builder concret correspondant à un décor simple : la classe SimpleDecorBuilder.*

Par la suite, ils ont promis de nous en fournir d'autres, les quelles ? *D'autres classes builder concrets : DesignDecorBuilder et FantatisqueDecorBuilder par exemple.*

Ajoutez ces classes dans votre diagramme de la question 5.2.



Question 5.4 – Coder en Java votre solution en insérant simplement des affichages console à chaque étape de création du décor. (cf. correction code)

Question 5.5 – 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.