M1 : Ingénierie du Logiciel - UNIVERSITE PIERRE & MARIE CURIE (PARIS VI)

Examen 2eme Session

7 Juin 2017 (2 heures avec documents : tous SAUF ANNALES CORRIGEES).

Barème indicatif sur 21,5 points (donne le poids relatif des questions) (max 20/20).

Questions de cours [4 Pts]

Répondez de façon précise et concise aux questions.

Barème : VALABLE sur toutes les questions de cours : -25 à -50% si la réponse inclut la bonne idée, mais qu'elle est noyée dans des infos ou autres réponses fausses/inappropriées.

Question Cours (QC):

QC1) Expliquez la fonction et le rôle d'un serveur d'intégration continue.

* 70% fonction : surveille un serveur de versions, reconstruit les artefacts + tests à chaque modification, déploie
* 30% role : assure la non regression, facilite et accélère le cycle de dév (livraison continue)

QC2) Pourquoi préconise-t-on de n'utiliser que des types simples (Bool, Int, String) dans les signatures d'opérations d'interface ?

50% portabilité/indépendance implémentations

50% pas de dépendances structurelles

QC3) Expliquez la notion de raffinement qui existe entre les diagrammes de séquence d'analyse et les diagrammes de séquence inter-composants réalisés à l'étape conception architecturale dans l'approche en V de l'UE.

* 70% la ligne de vie du système est décomposée en parties
* 30% le comportement extérieur vis à vis des acteurs est préservée

QC4) Quel est le rôle d'un composant bouchon dans un test d'intégration ? Est-on toujours obligé d'en définir ?

* 70% : satisfaire les interfaces requises d'un composant pour le tester en isolation
* 30% : non, si le composant est bout de chaine (pas de dépendances) ce n'est pas la peine.

2. Problème: Analyse StoneHearth [Barème sur 8 Pts]

Votre compagnie a décidé d'investir sur le segment des jeux pour téléphone mobiles "free to play/pay to win", et veut se lancer en force avec un nouveau jeu de cartes ultra-original et inédit. L'équipe chargée du design graphique promet d'innover avec des dragons, des robots, des lasers, des mages... De quoi plaire à tous les publics.

Le Jeu "StoneHearth" est un jeu à deux joueurs, où les joueurs s'affrontent en jouant chacun leur tour des cartes. La particularité du jeu par rapport aux jeux de cartes classiques (tarot, belote...) est que les cartes sont toutes différentes et dotées d'effets particuliers. Chaque carte a un nom et une description qui explique informellement ses effets.

Chaque joueur a donc une collection de cartes, à partir de laquelle il devra composer son "deck", c'est-à-dire choisir les 30 cartes (sans doublon) qu'il utilisera pour affronter son adversaire. Il dispose d'une interface lui permettant de mémoriser 3 decks qu'il peut utiliser en partie, mais il peut débloquer d'autres

emplacements de decks pour 2eu ou 3$ chacun. Les cartes sont rangées en quatre catégories de "rareté" croissante : "basique", "commune" puis "rare" puis "légendaire". Les cartes les plus puissantes sont aussi les plus rares.

Le modèle commercial est que les cartes ne peuvent pas être échangées entre les joueurs. Pour ajouter des cartes à sa collection, le joueur doit ouvrir des "packs", contenant 5 cartes aléatoires dont au moins une "rare". Les nouveaux joueurs ont droit automatiquement à toutes les cartes "basique" ce qui leur permet de composer un deck, plus 5 packs gratuits pour se lancer.

Les packs peuvent être achetés dans le jeu pour 1,39eu ou $1.99. Les joueurs assidus se voient également attribuer un pack toutes les 10 parties jouées, dans la limite de deux packs par 24h.

Comme les cartes issues des packs sont aléatoires, un joueur ayant des cartes en double peut détruire une carte ce qui lui donne des "joyaux". Les joyaux sont une monnaie dans le jeu, qui ne sert qu'à acheter des cartes, mais au choix de l'utilisateur cette fois.

La seule façon d'en obtenir est de détruire des cartes de sa collection (typiquement les cartes obtenues en double, mais pas forcément). Détruire une carte commune rapporte 2 joyaux, une carte rare 5 joyaux, une carte légendaire rapporte 20 joyaux.

Les cartes "basique" ne peuvent pas être détruites ni obtenues dans les packs. Pour acheter une carte commune il faut 20 joyaux, une carte rare 50 joyaux, une carte légendaire coûte 200 joyaux.

Les joueurs doivent créer un compte pour se connecter au jeu et accéder à leur collection de cartes et leurs decks mémorisés. Ils doivent y renseigner leurs coordonnées bancaires pour pouvoir acheter des cartes, mais ce n'est pas obligatoire de le faire immédiatement. Ils peuvent accéder aux options du compte par la suite, et les saisir quand ils le souhaitent.

Les joueurs connectés peuvent alors directement décider de jouer contre un joueur de même rang. Ils choisissent un deck dans la liste des decks qu'ils ont mémorisé, puis le jeu leur trouve un adversaire de rang similaire (écart <= 3 rangs).

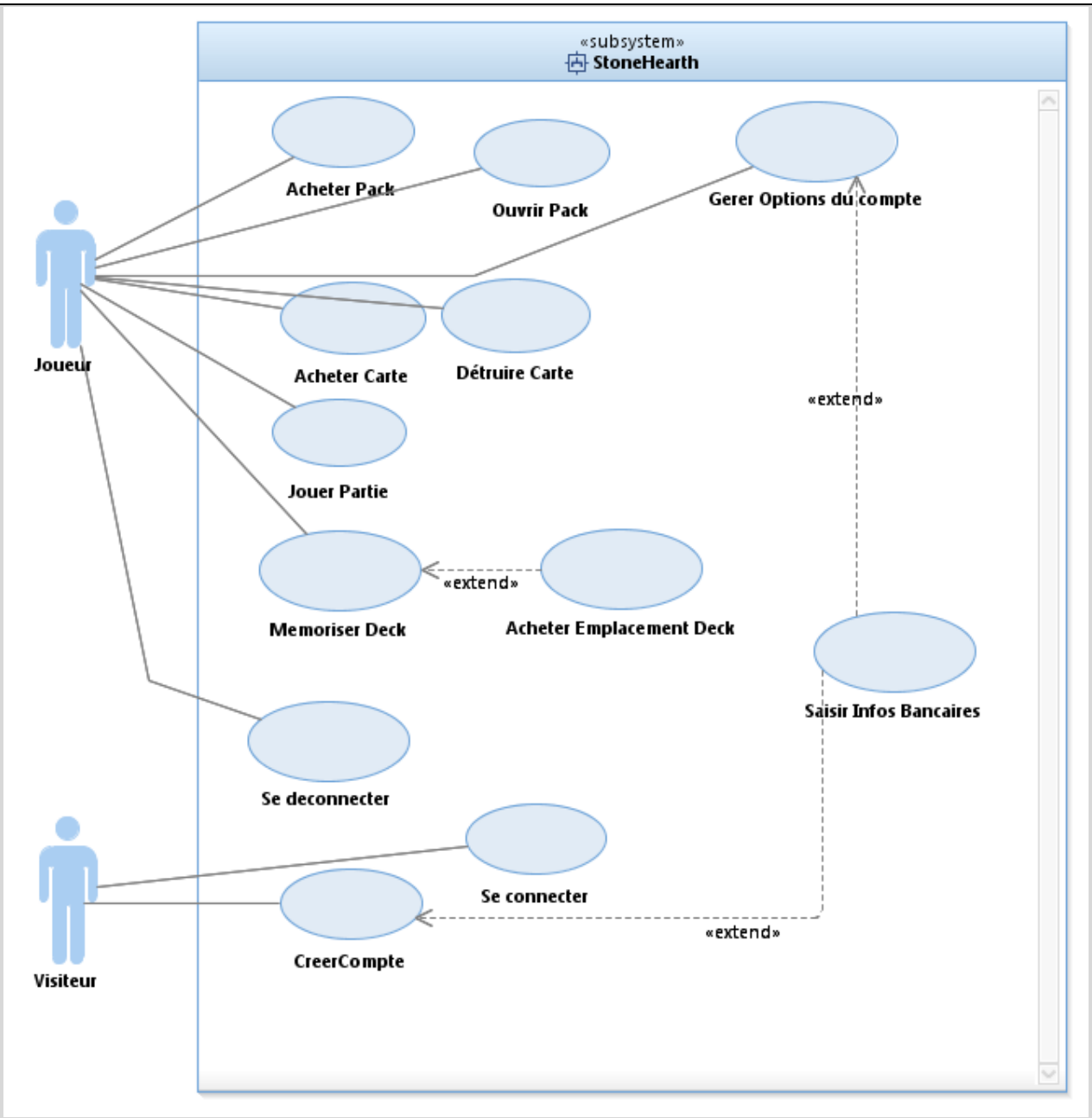
Le rang des nouveaux joueurs est initialement 0. Chaque partie gagnée incrémente le rang (sauf s'il est 100), chaque partie perdue le décrémente de 1 (sauf s'il est à 0).

Les joueurs jouent alors chacun leur tour des cartes jusqu'à ce que l'issue soit décidée par la victoire d'un des joueurs; il n'y a pas de match nul possible.

Dans cet énoncé on ne détaillera pas les règles du jeu lui-même, ni le déroulement de la partie.

Question 2.1 : (3 pts) Réalisez le diagramme de cas d'utilisation de la phase d'analyse. Vous justifierez tous vos choix, par un texte ou des annotations sur le diagramme.

Acteur : joueur



Barème :

Use case du Joueur : (10% chacun * 8) :

Acheter Pack, Ouvrir Pack

Acheter Carte, Détruire Carte (on accepte gérer cartes si on a un commentaire)

Jouer partie

Mémoriser Deck, Acheter Emplacement

Options du compte (CB…)

Gestion correcte de l'authentification : 20%

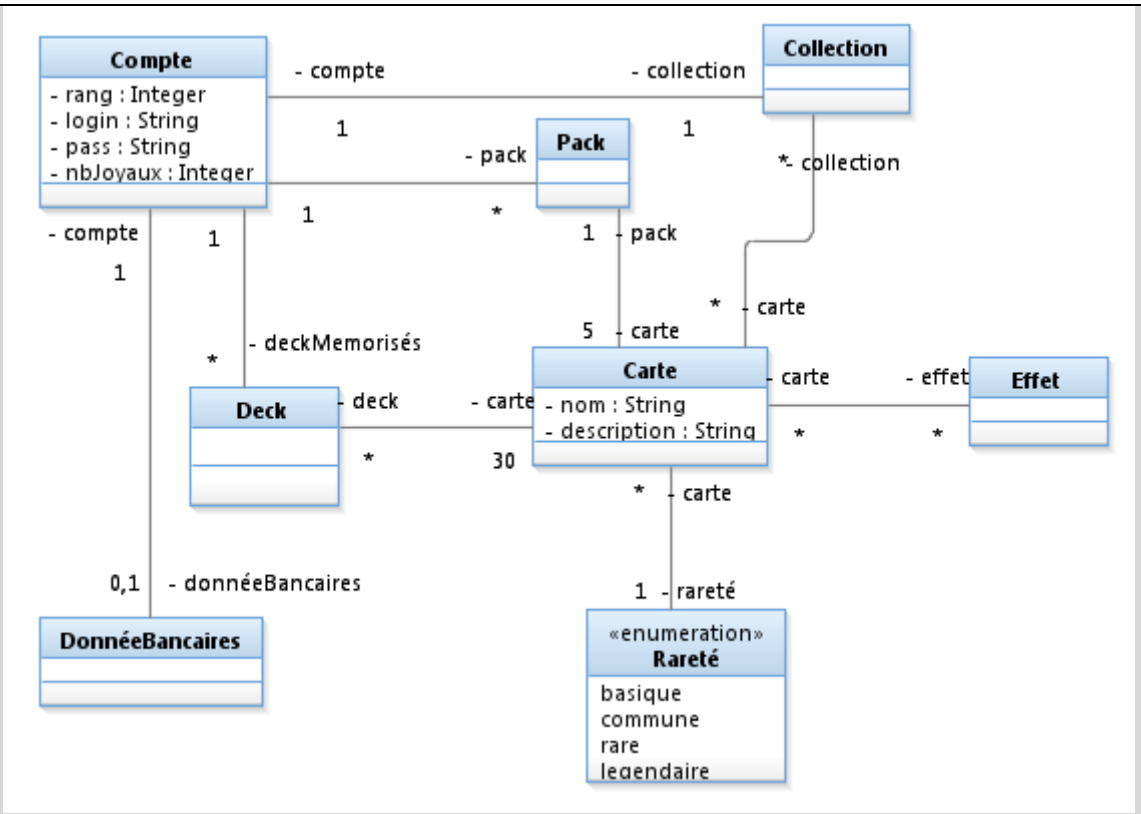
Inclut Créer Compte et Se Connecter (Déconnecter est optionnel)

10% seulement si on a lié ces use case à Joueur (donc un seul acteur)

Bonus : +5% pour le saisir Carte clairement optionnel

-10% par include ou extend injustifiable

Question 2.2 : (3 points) Réalisez le diagramme de classes métier de la phase d'analyse. Vous justifierez tous vos choix, par un texte ou des annotations sur le diagramme. On ne représentera pas la classe représentant le « Système », introduite dans l'approche en V du module.



Barème : (sur 110%)

Compte : porte nbJoyaux (10%) et rang (10%)

Compte asocié à Données bancaires : 10% si on voit un 0,1, 5% si c'est un attribut du compte

Compte associé aux decks mémorisés : 10%

Compte associé aux * Pack (pas encore ouverts) : 10%

Compte associé à * Carte : 10% (dans le corrigé via Collection mais ce n'est pas obligé)

Deck : associé à 30 cartes : 10% (on accepte * plutôt que 30)

Pack : associé à 5 cartes (* OK)

Carte : avec un nom et une description : 10%

Rareté de la carte : géré via un enum 10% . On sera indulgent sur la syntaxe de l'enum + les cardinalités du lien entre cet enum et Carte. 5% si c'est juste un attribut de Carte.

Effet : lié * * à Carte : 10% (c'est du bonus).

-10% par faute grossière, méthodes, modélisation d'acteurs…

Ce n'est pas super riche, mais modéliser un peu les règles c'est trop vite compliqué.

Question 2.3 : (2 pts) Ecrivez un test de validation couvrant l'achat d'une carte avec des joyaux.

* contexte : on est connecté sur "testeur", le compte dispose de fonds suffisants en joyaux (25 joyaux)

* entrée : la carte commune "Dragon vert"

* Scenario :

1. selection de l'option acheter carte

2. saisie de dragon vert dans la boite de recherche

3. selection du dragon vert

4. valider

R.A. : la carte est ajoutée à la collection, la quantité de joyaux du compte est diminuée de 20.

MV: dans la construction de deck, s'assurer que la carte est disponible.

La quantité de joyaux affichée en permanence dans le coin en bas à droite est passée à 5 joyaux.

Barème :

Contexte **20%** : « on est connecté » 10%, on a les sous 10%

Entrée **10%**: champ utilisé correctement, devrait citer une carte, + éventuellement ce qu'il faut pour le scenario. : 10%. Si on met des choses qui n'habitent pas là (e.g. le prix de la carte) 0%.

Scenario **20%** : 10% cohérent avec l'objectif, étapes du testeur seulement…, 10% précision suffisante pour la reproductibilité (e.g. le testeur choisit une carte = imprécis)

R.A **20%** : carte ajoutée aux cartes du joueur 10%, joyaux débités 10%

M.V. **20% :** on vérifie la présence de la carte (10%) et le débit de joyaux (10%)

Précision numérique : **10%** le scenario cite explicitement un prix chiffré pour la carte et donc le débit, qui soit cohérent avec le CdC.

-10% à -20% aberrations énorme (alternatives dans le scenario, …)

3. Problème: Conception ChatRoom [Barème sur 8,5 Pts]

On considère un système de Chat ou discussion instantanée en ligne, on donne les interfaces :

Permet de trouver les disscussions en cours, ou d'en créer de nouvelles.

«interface»
IForum

+ listerSujets ( ) : String [*]
+ trouverChatRoom (sujet : String) : IChatRoom
+ creerChatRoom (sujet : String) : Boolean

listerSujets rend l'ensemble des sujets des chatroom actuellement ouvertes.

trouver ChatRoom rend la discussion associée au sujet donné (ou null si pas trouvé)

creerChatRoom permet de créer une nouvelle discussion sur le sujet donné

Une discussion : un ensemble de Chatter y participent, on y poste des messages que tous liront.

«interface»
IChatRoom

+ getSujet ( ) : String
+ rejoindre (c : IChatter)
+ quitter (c : IChatter)
+ posterMessage (auteur : String, texte : String)

Les chatters peuvent rejoindre (s'abonner) ou quitter (se désabonner) d'une ChatRoom.

Les chatters abonnés sont des participants, si l'on poste un message sur cette ChatRoom, ils le recevront

Fonctionnement global du composant : Sujet dans DP Observer

Une abstraction pour un participant à une discussion

«interface»
IChatter

+ getNom ( ) : String
+ recevoirMessage (sujet : String, auteur : String, texte : String)

Le nom du chatter peut être utilisé comme nom d'auteur dans les post.

Recevoir message est invoqué si le chatter participe à des discussions et que l'on a posté un message sur l'une d'elles.

Fonctionnement global : Observateur du DP du même nom.

Un forum est un point central de connection qui permet de savoir quelles sont les discussions en cours.

IChatRoom fonctionne sur le modèle du DP Observer : chaque IChatRoom est associée à un ensemble de participants, c'est à dire des IChatter ayant *rejoint* la IChatRoom. Un IChatter participant à une IChatRoom va recevoir des notifications à chaque fois que l'on y poste un message.

Donc posterMessage("toto","Hello") sur une IChatRoom nommée "UML" doit provoquer l'invocation sur tous les participants de recevoirMessage("UML","toto","Hello").

Question 3.1 : (3 pts)

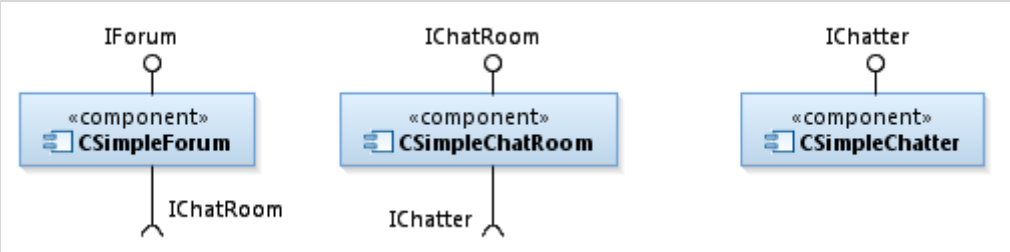
On considère trois composant : CSimpleChatter, CSimpleChatRoom, CSimpleForum qui réalisent chacun de façon *minimale* mais fonctionnellement correcte une des interfaces introduites (par exemple *recevoirMessage* peut se contenter d'écrire sur stdout).

a) (1,5) Représentez sur un diagramme de composant ces trois composants (interfaces requises et offertes).
b) (1,5) Proposez à l'aide d'un diagramme de classes une conception détaillée possible du composant CSimpleForum.

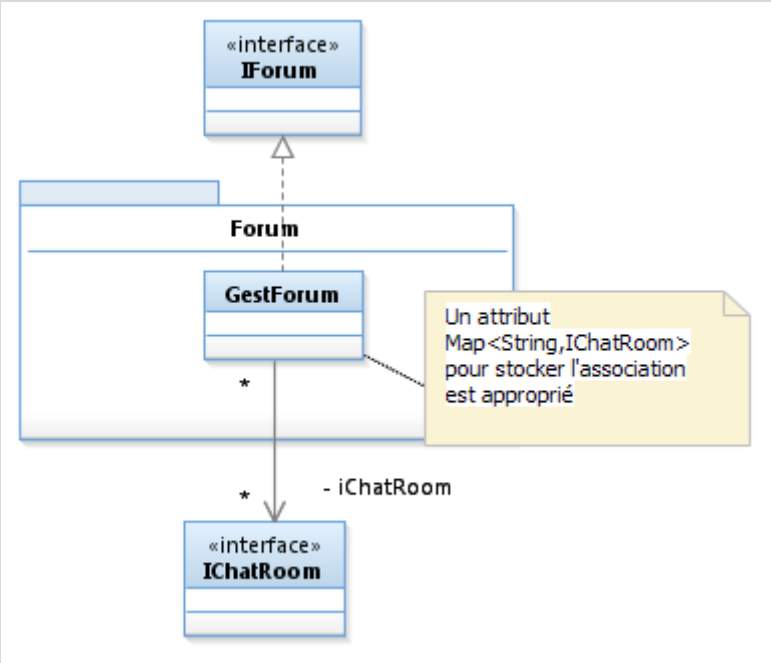
a)

35% par composant, binaire (rien en trop, rien en moins)

max 100%



b)



20% classe centrale

30% réalise itf

40% associe * IChatRoom ; dont 10% pour le fait qu'on lise bien *, 20% pour l'association elle-même, 10% pour le fait que ce soit ICR et pas SimpleCR.

10% utilisation correcte du package : interfaces dehors, classes dedans.

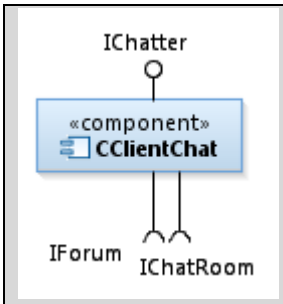
10% on suggère un Map (c'est du bonus)

On ne sanctionne pas s'il y a trop de classes (e.g. ChatRoom…)

Question 3.2 : (1 pts)

Le composant CClientChat est une application complète qui tourne sur les machines des chatters. Il permet de se connecter à un IForum, d'y trouver ou d'y créer une IChatRoom, de la rejoindre, puis de poster et de recevoir des messages.

Sur un diagramme de composants représentez ce composant CClientChat.

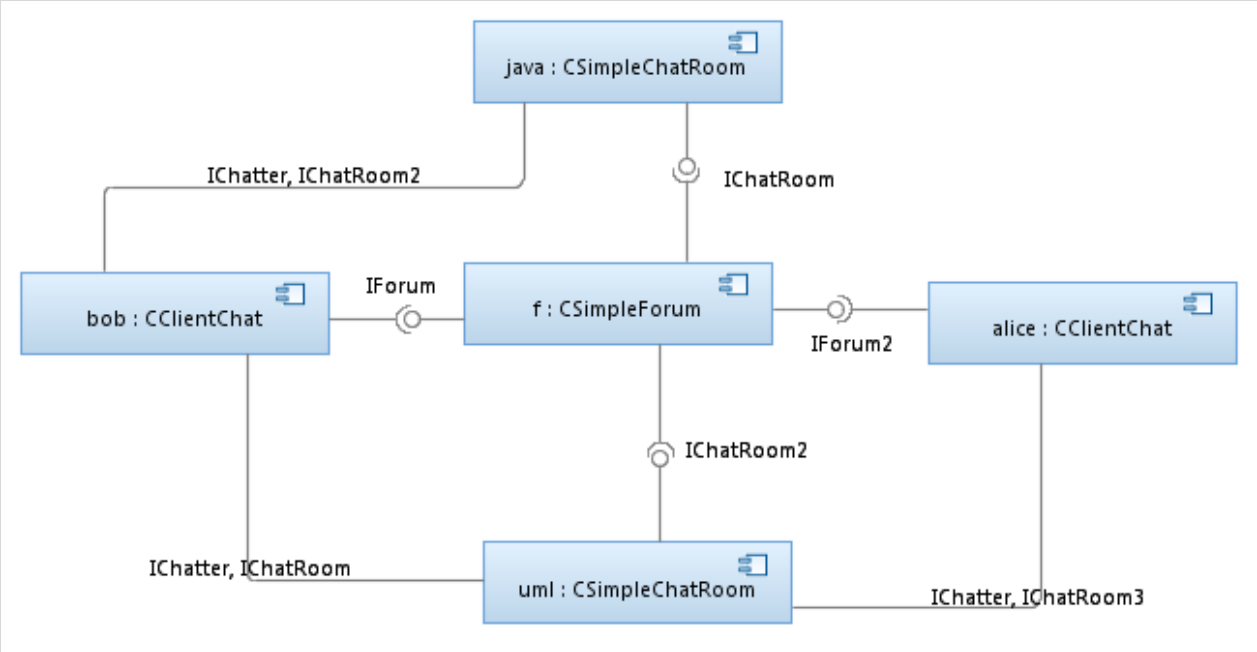


-50% par erreur : interface manquante, en trop, mal orientée.

Question 3.3 : (3,5 pts)

On considère la situation suivante : un forum qui contient deux chatroom de sujets respectifs « UML » et « Java ». « Bob » et « Alice » sont deux utilisateurs utilisant un client de chat. Bob a rejoint à la fois « UML » et « Java », mais Alice n'a rejoint que « UML ».

a) (1,5 pts) Modélisez cette situation sur un diagramme de structure interne.
b) (2 pts) Dans un diagramme de séquence de niveau intégration (une ligne de vie par composant) mettant en jeu les mêmes instances de composant, modélisez le scenario :
Alice crée et rejoint une chatroom "UML". Bob cherche et trouve UML puis rejoint la discussion. Alice envoie un message "bonjour" dans la chatroom « UML » (qui doit donc être reçu par Bob et Alice).



Instances : 50% = 5 instances * 10%

Liens : (nommés et orientés)

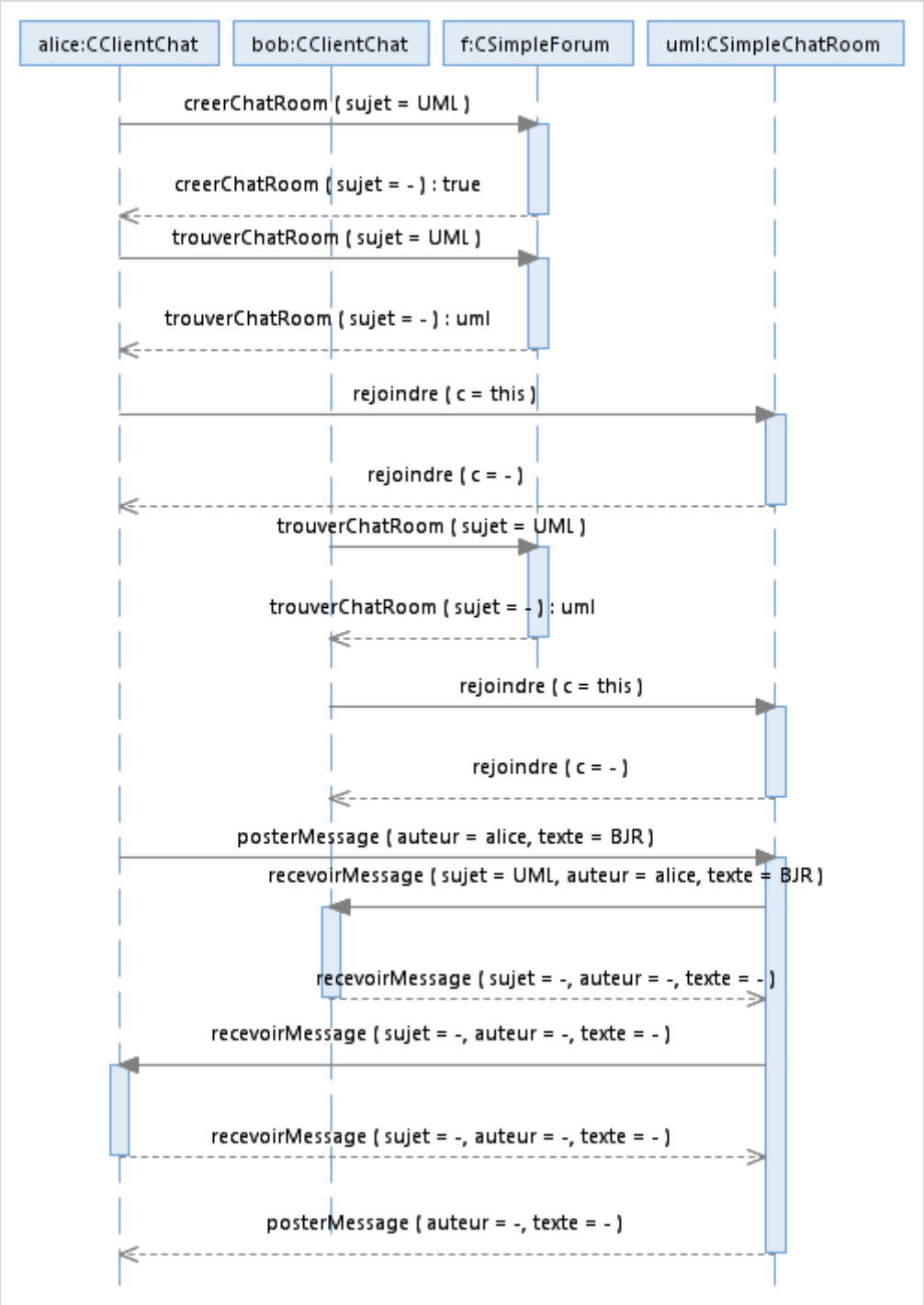
10% le forum connait les deux CR

10% les ChatClient connaissent le Forum

10% * 3 : Bob connait UML et Java, Alice connait UML. On sera indulgent sur l'orientation et le nommage des connecteurs pour ces trois connections.

-10% soucis de syntaxe, ce ne sont pas clairement des instances, les types sont absents…

b)



Mesages recherchés, il faut que ligne de vie source et destination soient correctes, + données raisonnables.

20% alice cree

10% alice rejoint

20% bob cherche

10% bob rejoint

10% alice poste

10% * 2 alice et bob recoivent

10% les deux recevoir message sont clairement dans le contexte du posterMessage.

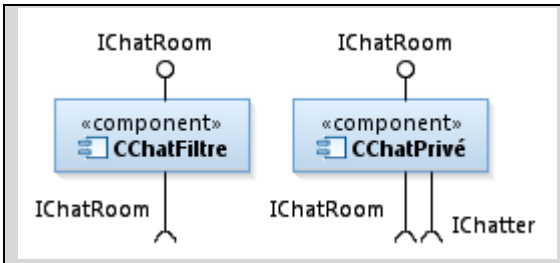
Question 3.4 : (2 pts)

Dans l'esprit du DP Decorator (cf Annexe), on souhaite permettre plus de flexibilité au niveau de la définition des IChatRoom, c'est-à-dire pouvoir doter a posteriori (après son instanciation) une IChatRoom de propriétés supplémentaires.

On considère un composant CChatRoomFiltre, qui empêche de poster des messages avec des gros mots sur un IChatRoom donné (poster ne fait rien si on détecte des gros mots dans le texte du message).

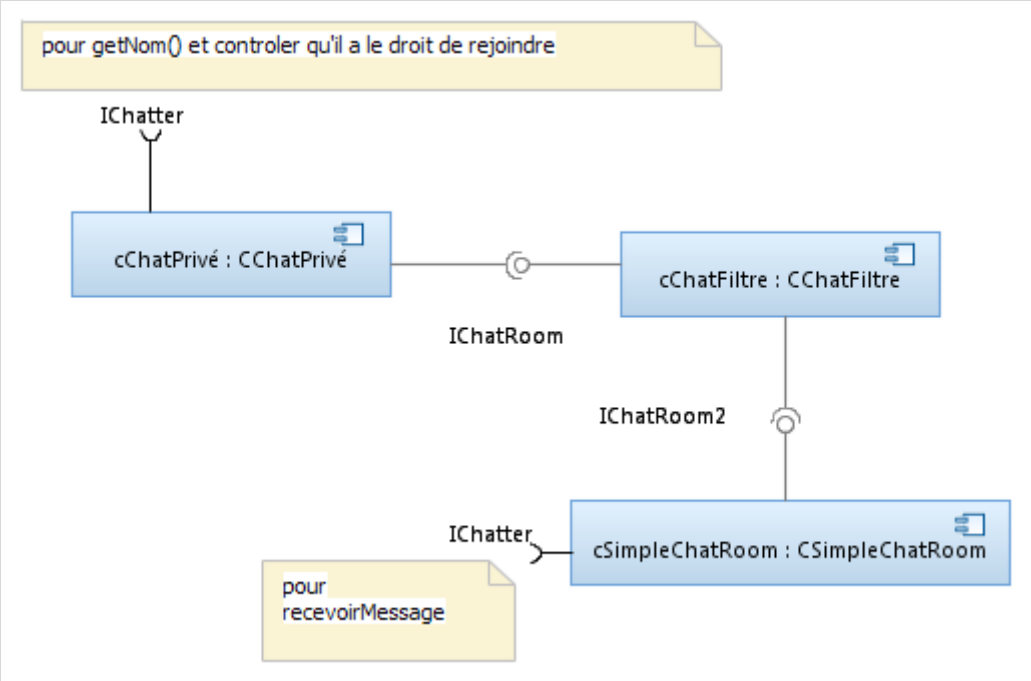
On considère également un composant CChatRoomPrivée, qu'on initialise avec une liste de noms d'utilisateurs, et qui seront les seuls à pouvoir rejoindre la ChatRoom concernée.

a) (1 pt) Sur un diagramme de composants, modélisez ces deux nouveaux composants.
b) (1 pt) Représentez sur un diagramme de structure interne une instanciation d'une ChatRoom protégée par un filtre sur les utilisateurs **et** sur les gros mots.



Barème : +**20% par itf attendue, -10% par itf en trop**

b)



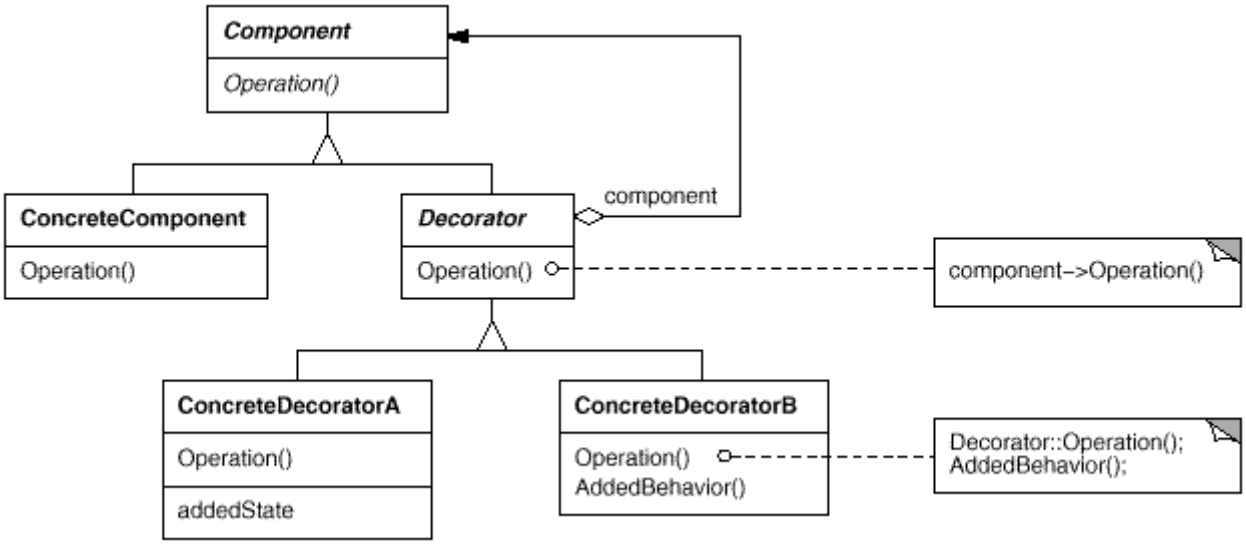
70% on voit une chaine de composants

30% instances * 3

Annexe : DP Decorator (extrait du GOF)

Intention : Attacher dynamiquement de nouvelles responsabilités aux objets. Les décorateurs offrent une alternative flexible à l'héritage pour étendre des fonctionnalités.

Structure :



Participants :

- Component : une interface qu'implantent les objets auxquels il faut attacher de nouvelles responsabilités
- ConcreteComponent : un objet (simple) auquel on veut attacher des responsabilités
- Decorator (abstrait): détient une référence à un Component, et implante cette interface par délégation sur cette instance. Permet de factoriser le code des décorateurs concrets.
- DecorateurConcret : redéfinit les opérations souhaitées, pour ajouter du comportement à l'objet décoré.