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

Examen Réparti 2eme partie

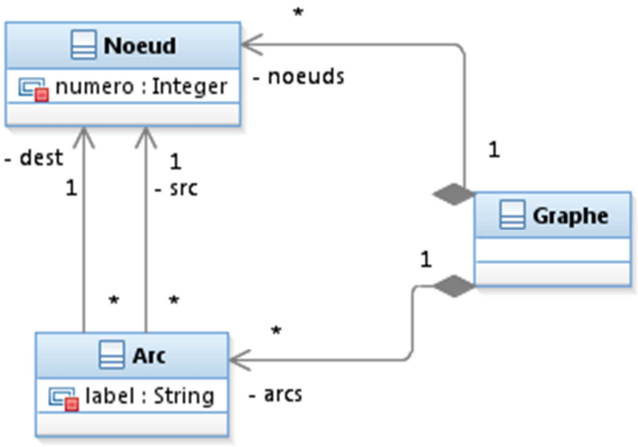
9 Janvier 2015 (2 heures avec documents : tous SAUF ANNALES CORRIGEES).
Barème indicatif sur 21 points (donne le poids relatif des questions) (max 20/20).

Questions de cours [5 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.

Soit le méta-modèle suivant permettant de représenter des instances de graphes composés de nœuds portant un numéro et d'arcs orientés d'un nœud source vers un nœud destination et portant une étiquette qui est un string.



Question Cours (QC):

a) Représentez sous une forme textuelle arborescente (proche de la structure d'un fichier XMI) les objets d'un modèle instance représentant un graphe constitué de deux nœuds (indicés 0 et 1) et de deux arcs de 0 à 1 (étiquette « a ») et de 1 à 0 (étiquette « b »).

b) Avec ce méta-modèle peut-on construire un modèle instance où plusieurs arcs portant la même étiquette « a » lient le nœud 0 au nœud 1 ? Justifiez pourquoi.

c) Proposez une modification de ce méta-modèle permettant de représenter qu'un nœud du graphe est le nœud initial.

d) Proposez une modification de ce méta-modèle permettant de représenter qu'un modèle peut contenir un ensemble de graphes et pas un seul.

e) Les arcs pondérés sont des arcs qui portent en plus de leur étiquette un poids représenté par un entier. Modifiez le méta-modèle pour permettre de gérer ce cas.

Les questions de cours sont notées sur 100% une seule note à saisir, 20% par Q de cours

a)

```
<graphe>
  <noeuds>
    <nœud numero=0 id=#n0 />
    <nœud numero=1 id=#n1 />
  </nœuds>
```

```
    <arcs>
      <arc label= « a » src=#n0 dest=#n1 id=#a1 />
      <arc label= « b » src=#n1 dest=#n0 id=#a2 />
    </arcs>
  </graphe>
```

Barème :

On accepte pas mal de variablité dans la réponse.

5% on a une tentative, mais ça finit mal, arc fils de nœud ou l'inverse, réponses avec duplications des nœuds en fils de graphe et de arc….

10% on a 4 objets clairement, 2 arcs 2 nœuds

10% on a des refs aux nœuds pour src et cible de l'arc

Donc on a 20% pour les réponses qui font src=0 dest=1 et qui ont clairement 4 objets pas fils les uns des autres

25% réponse parfaite : les src et dest utilisent clairement des id d'objets définis, pas les numéros des états (donc 5% bonus)

-5% une réponse bizarre avec des private ou autres éléments pris au TD sur les MM. On peut quand même trouver des éléments à points dans ces réponses (4 instances, refs de arcs sur nœuds)

a) Oui, il n'y pas de raison particulière d'interdire ça. Chaque arc est un objet distinct, il a son id. De plus les cardinalités le permettent.

Barème :

10% réponse, 10% explication

b) 20% On ajoute une **aggregation** 0,1 ou 1 orientée de graphe sur nœud, nom de role initial

La solution avec une composition plutôt que aggregation donne 20% aussi.

10% : un bool dans Nœud initial. Cette réponse ne place pas cette information correctement dans le contexte du graphe, e.g. on peut du coup avoir plusieurs états initiaux.

Les solutions avec NoeudInitial hérite de Noeud, s'il est lié au graphe par un lien correct est ok. Mais c'est inutile ici, la différence entre un nœud initial et un autre nœud est contextuelle au graphe, pas intrinsèque au noeud.

c) 20% Nouvelle métaclasse racine Modele qui compose * Graphe. 10% si on a modélisé une aggregation.

Barème :

15% association reflexive de graphe sur lui-même. Cette solution modélise en fait une notion de sous-graphe, donc pas exactement ce qu'on demande. 5% si c'est une aggregation pas une composition.

0% solution où l'on change la cardinalité à * pour l'extrémité graphe des associations graphe-nœud et graphe-arc. Déjà ça ne peut pas être cohérent avec le lien de composition qui interdit ce cas de figure, et ça ne répond pas à la problématique « plusieurs graphes ».

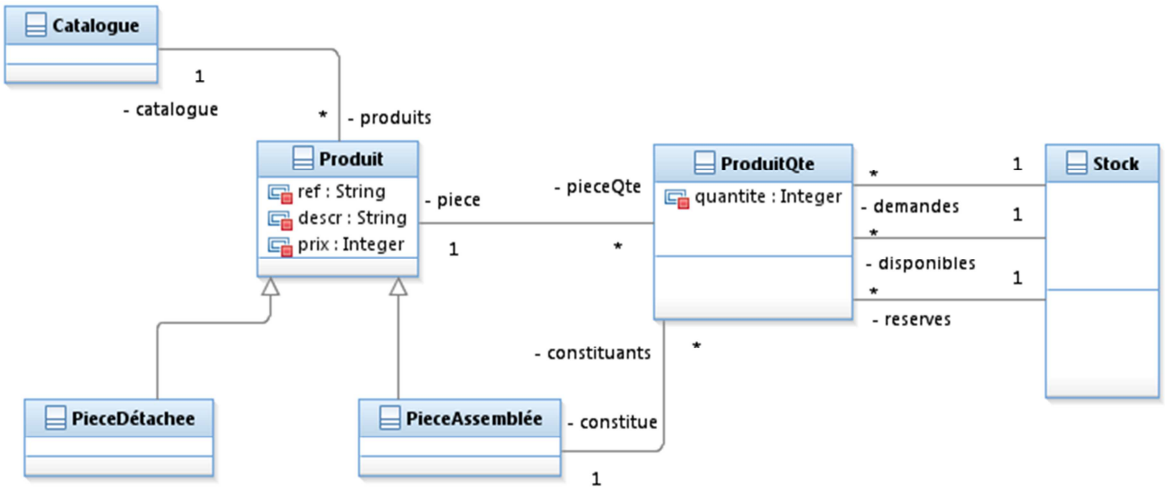
d) 20% Nouvelle métaclasse ArcPondéré qui extends Arc et a un attribut entier.

10% on ajoute un poids directement à Arc. On ne doit pas forcer tous les arcs à être pondérés.

2. Problème: Gestion de production [Barème sur 16 Pts]

Rappels d'analyse : Nous considérons une application permettant la gestion de stocks et de production d'une usine de cycles. Un produit est muni d'une référence unique, d'une description et d'un prix. Certains produits sont des pièces détachées, achetées par l'usine auprès de fournisseurs. D'autres produits sont des pièces assemblées dans l'usine, en utilisant d'autres produits comme constituants.

On rappelle ces éléments (simplifiés) du diagramme de classe d'analyse.



On s'intéresse ici à la gestion du stock, qui va s'appuyer sur trois composants, gérant respectivement un ensemble de produits associés à des quantités (un composant utilitaire simple, largement réutilisé), un catalogue (avec les descriptions détaillées des produits et de leurs constituants éventuels), et un composant gérant la mise à jour de l'état des stocks suite aux achats de pièces détachées auprès des fournisseurs, aux opérations d'assemblage réalisées dans l'usine et à l'évolution des commandes.

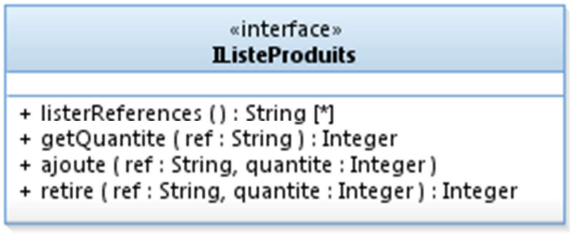
Les classes Catalogue, Produit, et PièceXX iront dans le composant **CCatalogue**, et le stock dans **CStock**. La découpe en composants proposée va donc isoler dans le composant **CListeProduit** la classe **ProduitQte**. Ceci impose de casser toutes les associations sur cette classe.

Pour casser le lien structurel entre **ProduitQte** et **Produit**, on passera à un attribut **ref** typé String dans la classe **ProduitQte** (donc on a orienté de **ProduitQte** sur **Produit**).

On retrouvera en conception les divers liens de ce diagramme sur **ProduitQte** comme des dépendances des composants **CStock** et **CCatalogue** sur **CListeProduit** (donc tous les autres liens à découper sont orientés vers **ProduitQte**).

I. Liste de produits (5 points)

Soit l'interface suivante permettant d'associer des références de produits à des quantités. Par exemple, une liste peut contenir 50 occurrences du produit de référence « rayon », et 1 occurrence de « jante ».



- `listerReferences` : rend la liste des références associées à une quantité non nulle
- `getQuantite` : rend la quantité associée à une référence, 0 si la référence est inconnue

- `ajoute` : ajoute *quantité* occurrences de la référence à la liste
- `retire` : essaie de retirer *quantité* occurrences de la référence de la liste. Rend le nombre de références effectivement obtenues (plus petit ou égal à *quantité*), la quantité dans la liste ne peut pas descendre en négatif.

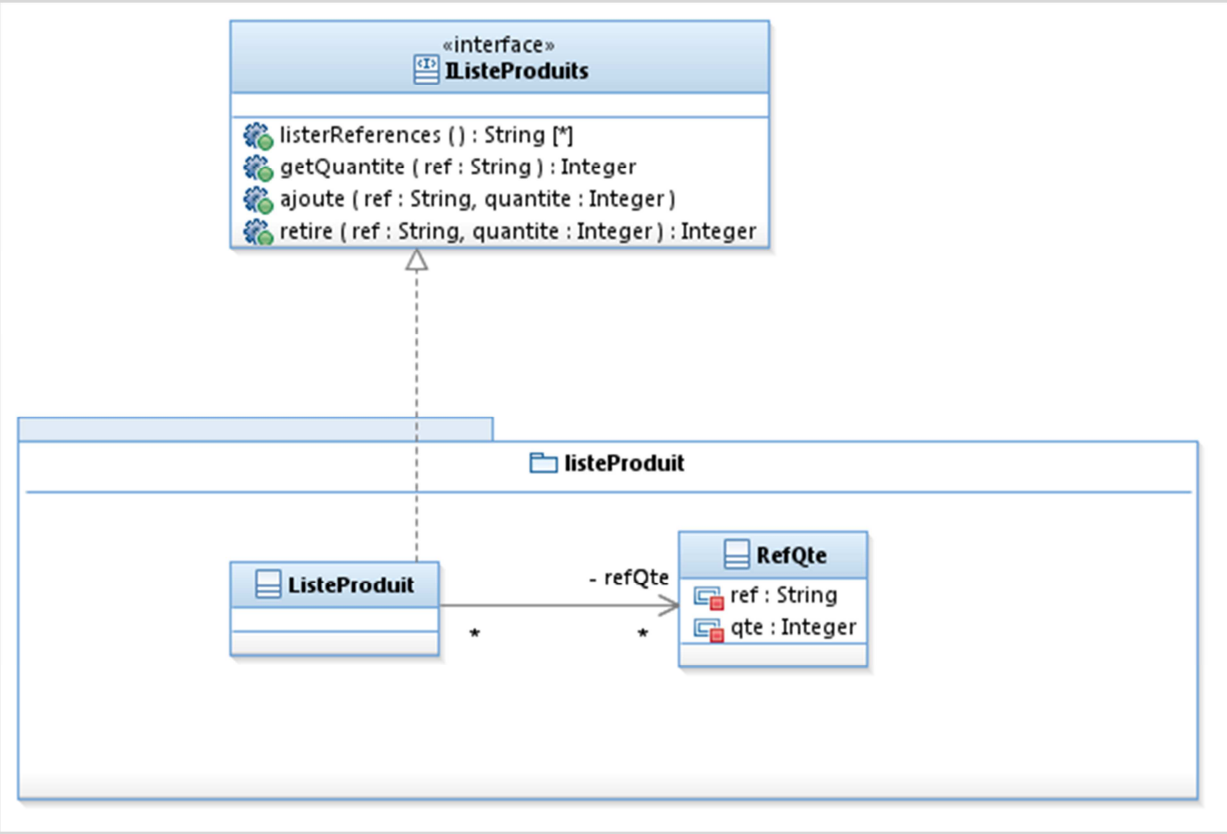
Question I.1 : (2 pts)

a) Représentez sur un diagramme de composant un composant CListeProduit qui réalise cette interface.
b) Proposez à l'aide d'un diagramme de classes une conception détaillée possible de ce composant.

a)

20% : composant, une interface offerte. Notation binaire 0 ou 20%.

b) Sur 80%. A posteriori l'énoncé aurait du nommer IListeProduitQte et ClisteProduitQte ce composant, lui attribuant plus explicitement la classe métier ProduitQte et pas la classe Produit. Le nommage ou la formulation a poussé pas mal d'étudiants à stocker les produits (classe métier avec un prix et une description) dans ce composant, ce qui pose des problèmes au catalogue plus tard (il n'a plus de rôle à tenir). On peut quand même avoir des points ici malgré des infos stockées inutiles (prix description).



30% une nouvelle classe Liste de facade réalise l'interface

30% elle associe * objets porteurs d'une référence (ça pourrait être des Produit)

20% l'objet aggregé porte une quantité sous la forme d'un entier. On ne donne pas ces 20% si c'est via une indirection * (e.g. dans * ProduitQte si on a ramené Produit et ProduitQte dans ce composant)

+10% on suggère un map de clé les références.

La réponse avec un simple Map de String sur int est considérée réponse parfaite 80%

0% Produit ou ProduitQte réalise l'interface, i.e. on ne peut pas stocker * quantités dans le composant.

-20% la classe produit liée en * qui implémente une hypothétique interface IProduit, si c'est cohérent avec en a) une interface offerte en plus (donc on n'a pas eu les points sur a)) on ne met pas le malus.

-20% on a * ProduitQte lié au produit, typiquement on a ramené tout sauf Stock dans ce composant.

Question I.2 : (3 pts)

On souhaite mettre en place des tests d'intégration de ce composant validant (entre autres) le scénario suivant : partant d'une liste vide, on ajoute 5 vélos, on en retire 3, la quantité de vélos dans la liste doit être de 2.

a) Représentez un ou plusieurs composants (testeur, bouchon(s)) permettant de mettre en place ces tests.
b) Sur un diagramme de structure interne, représentez une configuration des composants pour le test.
c) Rédigez (avec une syntaxe exécutable en pseudocode style JUnit) un test d'intégration couvrant le scénario précité.

Une seule note à saisir, questions sur 30/30/40.

a) sur 30%

30% binaire

-10% interface requise n'est pas nommée

On donne 10% si on trouve autre chose, mais que c'est cohérent avec I.1 a).

0% si on définit un bouchon pour IListeProduit…

b)

30% : 10% par instance, 10% le lien nommé

10% si cohérence avec le a) juste au-dessus, même si c'est faux.

-10% ce ne sont pas clairement des instances.

c)

```
IListeProduits sut = ComposantFactory.createListeProduits() ;
Sut.ajout(« velo »,5) ;
Sut.retire(« velo »,3) ;
assertTrue(sut.getQuantite(« velo »)==2) ;
```

Barème : sur 40%, Syntaxe libre,

10% l'objet SUT utilisé est clairement typé CListeProduit ou IListeProduit dans le pseudo-code. On donne aussi 10% si la ref sut est typée par la classe concrète qui réalise l'interface de 1.b). La Factory

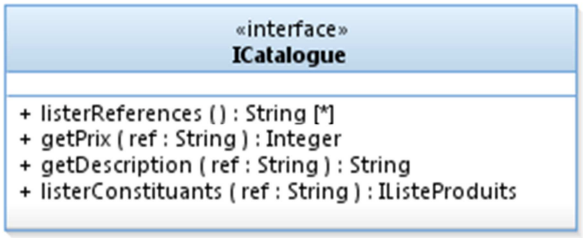
n'est pas demandée, la création pourrait se faire ailleurs. On accepte meme un « new IListeProduit » assez incorrect.

10% on doit voir qu'on invoque les opérations sur l'objet (ajout, retire),

20% placement d'un assert pour le résultat attendu.On donne 10% si on teste que retire rend 2 (ce qui est faux, mauvaise lecture de l'API).

II. Catalogue (6 points)

Le composant **CCatalogue** stocke la description des produits. Il réalise l'interface suivante :



- `listerReferences` : donne la liste des références de produits du catalogue
- `getPrix` : rend le prix d'un produit particulier du catalogue (0 si la référence est inconnue)
- `getDescription` : rend la description associée à un produit du catalogue (**null** si la référence est inconnue)
- `listerConstituants` : rend les constituants d'un produit du catalogue (**null** si la référence est inconnue ou si la référence correspond à une pièce détachée)

Question II : (6 pts)

a) Représentez **CCatalogue** sur un diagramme de composant.
b) Proposez à l'aide d'un diagramme de classes une conception détaillée possible de ce composant.
c) Un composant **CCatalogueIHM** permet d'afficher l'information contenue dans le catalogue. Représentez-le sur un diagramme de composant.
d) (2 pts) Faites un diagramme de séquence de niveau intégration (où les lignes de vies sont des instances de composants) qui permette à une instance de CCatalogueIHM d'afficher (en cohérence avec l'exemple donné en III):

Une « roue » est constituée de 50 « rayon » et de 1 « jante ». « jante » est une pièce détachée. « rayon » est une pièce détachée.

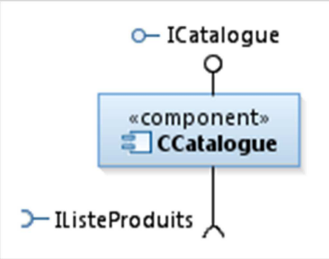
On ne demande pas que cette IHM affiche les descriptions ou les prix des produits. Cependant l'IHM doit tenter de récursivement afficher les constituants. On prendra soin de spécifier arguments et valeurs de retours sur les messages.

e) Représentez sur un diagramme de structure interne les instances de composants qui participent à l'interaction décrite en d).

A) et B) : une seule note sur 100%

A)

Sur 20 % : +10% par itf correcte, -10% par incorrect



B) Sur 80%

Classe Gest :

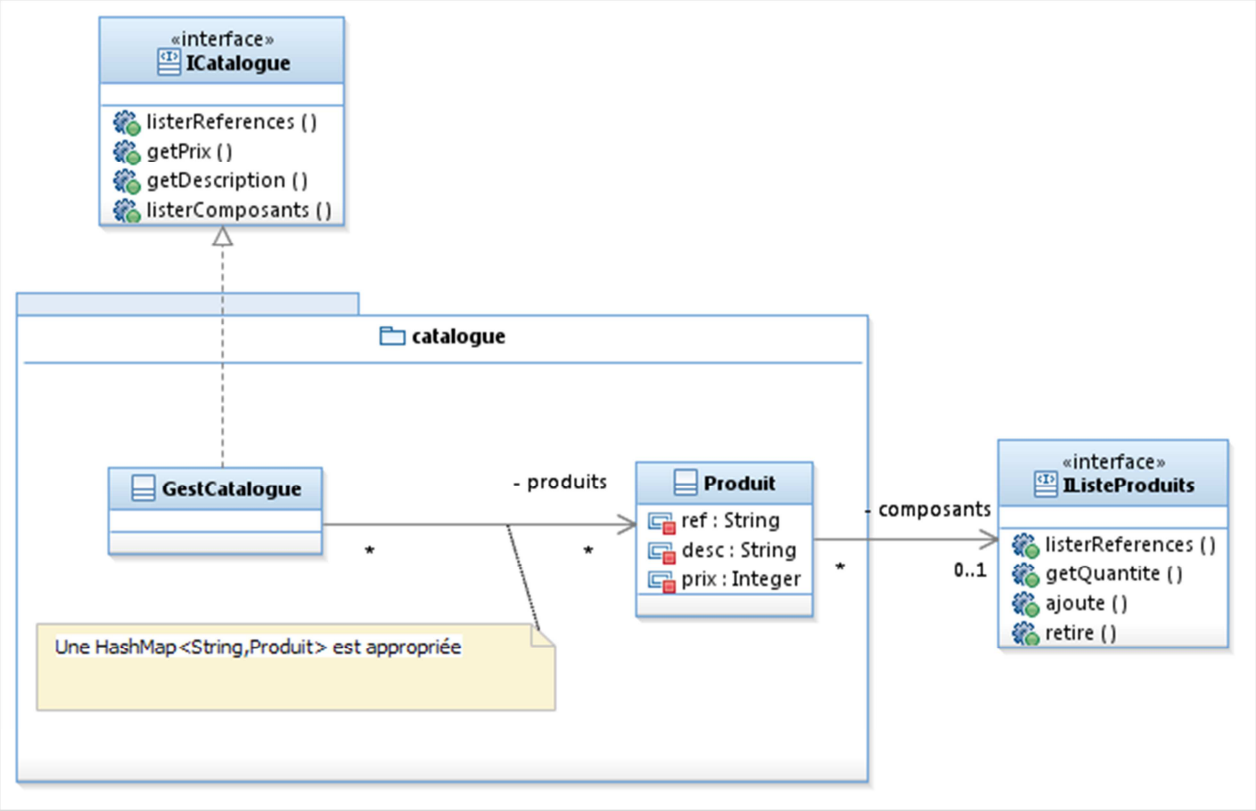
20% implements Icatalogue

20% associe * Produit, +10% si on suggère une Map.

Classe Produit

20% desc, prix

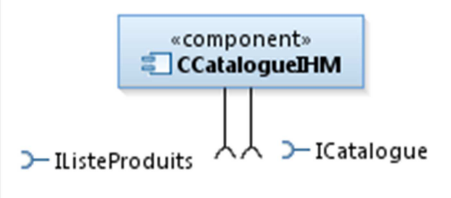
20% associe une IListeProduit



C) et D) une seule note sur 100%

C) sur 20%

10% par itf



D) Sur 80%

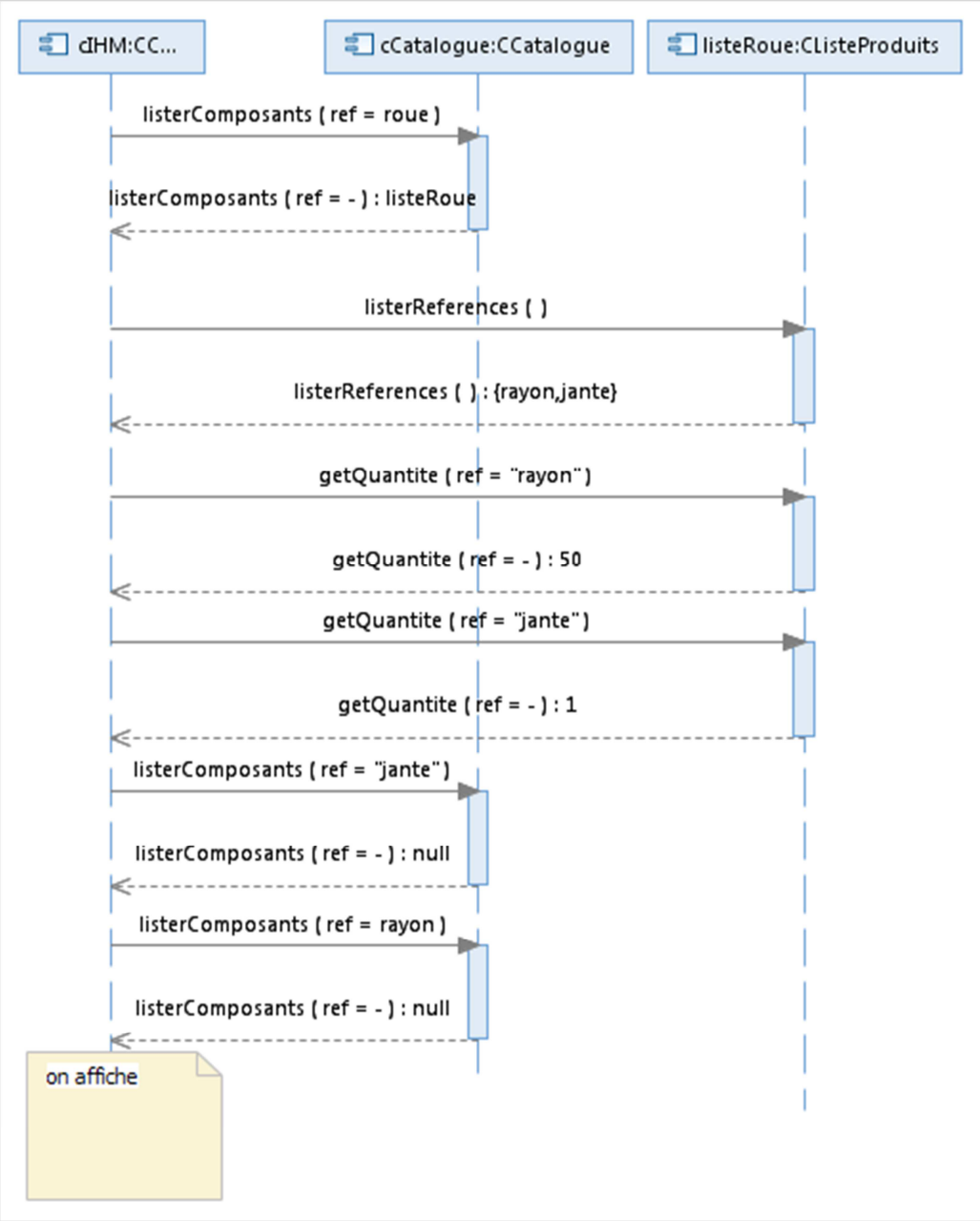
Ne donner que la moitié des points si les arguments//valeurs sont mal spécifiés.

20% listerComposants, rend bien une ref sur l'autre ligne de vie du diagramme.

20% listerRefs

20% les deux getQte

20% les deux listerComposants qui donnent null

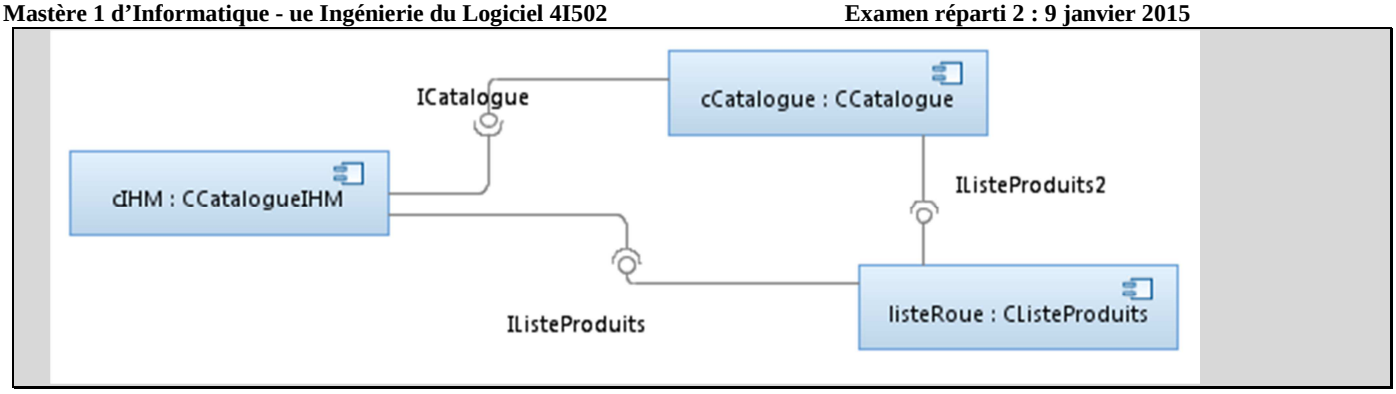


E) Sur 100

10% le nom des instances colle avec les noms des lifeline du diagramme de séquence

10% par instance : *3 = 30%

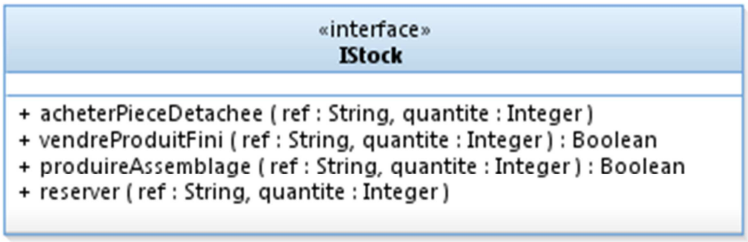
20% par lien correctement orienté (0 sinon) :*3 = 60%



III. Stock (5 points)

Le composant **CStock** gère et maintient à jour trois listes de produits :

- Les produits **disponibles** sont immédiatement en stock et n'ont pas été réservés.
- Les produits **réservés** sont présents dans le stock, mais font l'objet d'une réservation pour honorer une commande
- Les produits **demandés** ne sont pas présents dans le stock, il faudra les acheter auprès d'un fournisseur (produit de type pièce détachée) ou les produire dans l'usine (produit de type pièce assemblée).



Il offre l'interface suivante :

- `acheterPieceDetachee` : permet d'ajouter *quantité* occurrences de la pièce détachée portant la référence *ref* au stock. Cette opération est invoquée quand le fournisseur livre des pièces à l'usine. Si le stock contient 70 **demandes** pour *ref*, et si on en achète 100 unités, le stock contiendra 70 produits **réservés** et 30 produits **disponibles**.
- `vendreProduitFini` : permet d'enregistrer la sortie d'usine (et donc du stock) de *quantité* occurrences d'un produit assemblé fini, pour honorer les commandes. Les produits à vendre doivent impérativement être **réservés** dans le stock, ou l'opération échoue et rend « false ».
- `produireAssemblage` : permet d'enregistrer la fin d'une opération de production, c'est-à-dire l'assemblage de *quantité* occurrences d'une pièce assemblée *ref* dans l'usine. Les pièces nécessaires à l'assemblage sont retirées du stock, en prenant d'abord dans les pièces **réservées**, puis dans les pièces **disponibles**. L'opération échoue et rend « false » si les constituants ne sont pas présents dans le stock. L'ajout des produits nouvellement assemblés suit la même règle que l'achat de pièces détachées : on bascule de **demandé** à **réservé**, l'excédent éventuel devient **disponible**.
- `reserver` : permet de préparer la satisfaction des commandes, tous les produits figurant sur une commande sont réservés quand la commande est enregistrée. Les éléments passent de **disponible** à **réservés**, s'il n'y a pas assez de produits en stock, le reste est ajouté sous la forme de demandes. Les nouvelles demandes de produits assemblés nécessitent récursivement de réserver les constituants de l'assemblage.

Exemple :

Catalogue : Une « roue » est constituée de 50 « rayon » et 1 « jante ». « rayon » et « jante » sont des pièces détachées.

1. Initialement le stock contient :

- **Disponible** : 10 jantes, 3 roues
- **Réservé** et **Demandé** sont vides

2. Une nouvelle commande contenant 4 roues est saisie. On réserve donc 4 « roue » ; le stock devient :

- **Disponible** : 9 jantes
- **Réservé** : 1 jante, 3 roues
- **Demandé** : 1 roue, 50 rayon

3. On achète 1000 rayon au fournisseur ; le stock devient

- **Disponible** : 950 rayon, 9 jantes
- **Réservé** : 50 rayon, 1 jante, 3 roues
- **Demandé** : 1 roue

4. On produit l'assemblage d'une roue ; le stock devient :

- **Disponible** : 950 rayons, 9 jantes
- **Réservé** : 4 roue
- **Demandé** : vide

5. On vend les 4 roues ; le stock devient :

- **Disponible** : 950 rayons, 9 jantes
- **Réservé** et **Demandé** sont vides

Question III. : (5 pts)

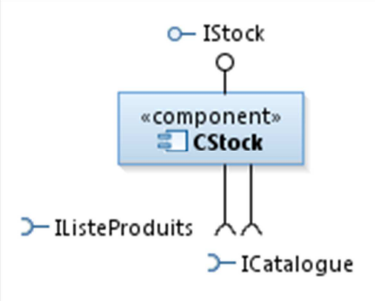
a) (1 pt) Modélisez **CStock** sur un diagramme de composants.
b) (1,5 pts) Proposez une conception détaillée de **CStock** à l'aide d'un diagramme de classe. Les listes de produits disponibles, réservés et demandés seront représentées par des occurrences de **CListeProduit.**
c) (2,5 pts) Modélisez sur un diagramme de séquence de niveau interaction l'étape 2 du scénario exemple donné ci-dessus (on suppose un composant **CIHM** qui initie l'interaction). Faites figurer le stock et les occurrences de listes de produits qu'il gère. On fera attention à bien préciser les arguments et les valeurs de retours sur les messages. Comme ce scénario nécessite certaines interactions déjà modélisées, on pourra placer un simple commentaire expliquant ce qu'on réutilise, ne recopiez pas ici la réponse à II.d).

A)

30% offre IStock

30% requiert IListeProduit

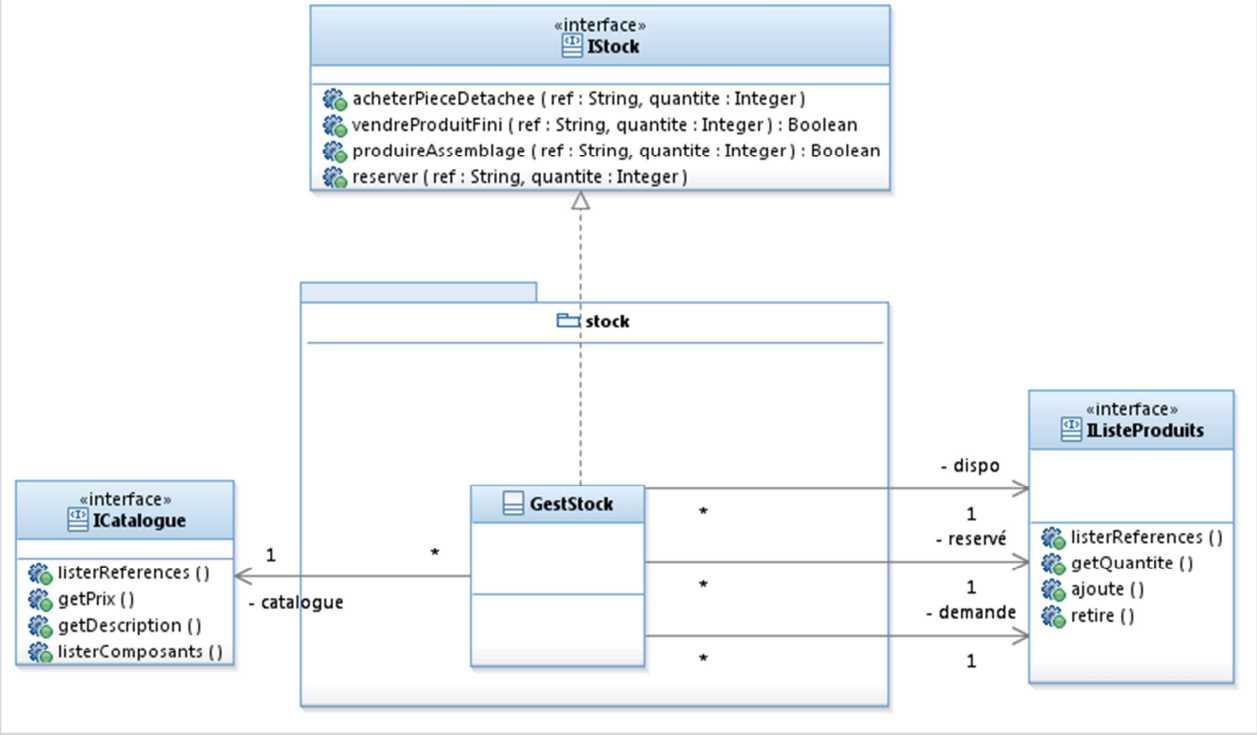
40% utilise Icatalogue



B)

20% par lien

La solution avec un seul lien (cardinalité *) de la classe facade vers IListeProduit donne 20% au lieu de 60% sur ces 3 liens. Le typage ListProduit ou CListeProduit donne une pénalité -10, si les 3 liens sont bien on a donc quand même 50%.

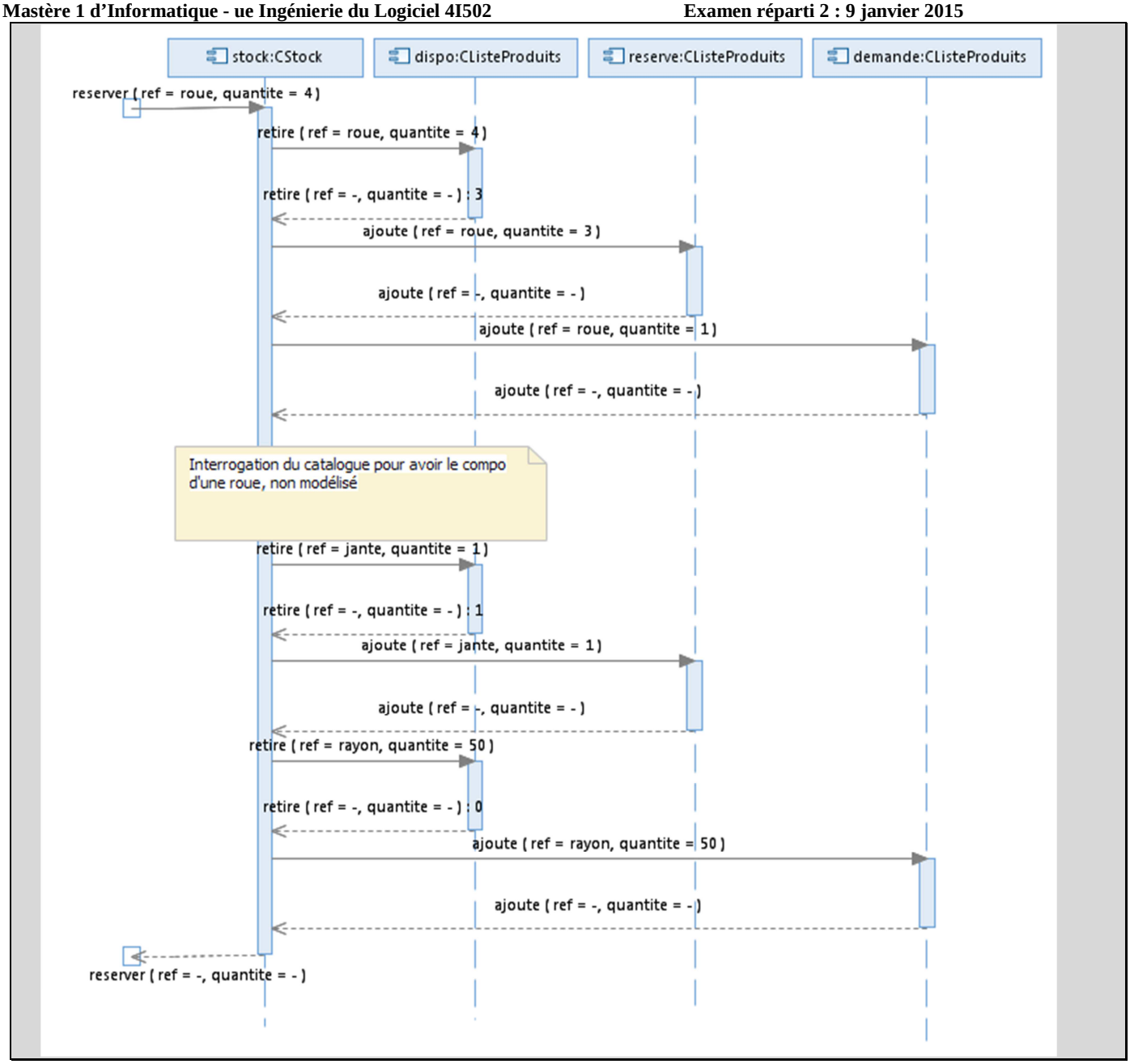


C)

10% invocation initiale reserve « 4 roue »

20% commentaire pertinent de réutilisation bien placé

10% par message sur le corrigé : *7 = 70%


stock:CStock
dispo:CListeProduits
reserve:CListeProduits
demande:CListeProduits
reserver (ref = roue, quantite = 4)
retire (ref = roue, quantite = 4)
retire (ref = -, quantite = -) : 3
ajoute (ref = roue, quantite = 3)
ajoute (ref = -, quantite = -)
ajoute (ref = roue, quantite = 1)
ajoute (ref = -, quantite = -)
Interrogation du catalogue pour avoir le compo d'une roue, non modélisé
retire (ref = jante, quantite = 1)
retire (ref = -, quantite = -) : 1
ajoute (ref = jante, quantite = 1)
ajoute (ref = -, quantite = -)
retire (ref = rayon, quantite = 50)
retire (ref = -, quantite = -) : 0
ajoute (ref = rayon, quantite = 50)
ajoute (ref = -, quantite = -)
reserver (ref = -, quantite = -)