IPST-CNAM
Intranet et Designs patterns
NSY 102
Vendredi 27 Avril 2016

Durée : **2 h 45**
Enseignants : LAFORGUE Jacques

1ère Session NSY 102

CORRECTION

1ère PARTIE – SANS DOCUMENT (durée: 1h15)

1. QCM (35 points)

Mode d'emploi :

Ce sujet est un QCM dont les questions sont de 3 natures :

- **les questions à 2 propositions**: dans ce cas une seule des 2 propositions est bonne.
  - +1 pour la réponse bonne
  - –1 pour la réponse fausse
- **les questions à 3 propositions** dont 1 seule proposition est bonne
  - + 1 pour la réponse bonne
  - –½ pour chaque réponse fausse
- **les questions à 3 propositions** dont 1 seule proposition est fausse
  - + ½ pour chaque réponse bonne
  - –1 pour la réponse fausse

Il s'agit de faire une croix dans les cases de droite en face des propositions.

On peut remarquer que cocher toutes les propositions d'une question revient à ne rien cocher du tout (égal à 0).

Si vous devez raturer une croix, faites-le correctement afin qu'il n'y ait aucune ambiguïté.

N'oubliez pas d'inscrire en en-tête du QCM, votre nom et prénom.

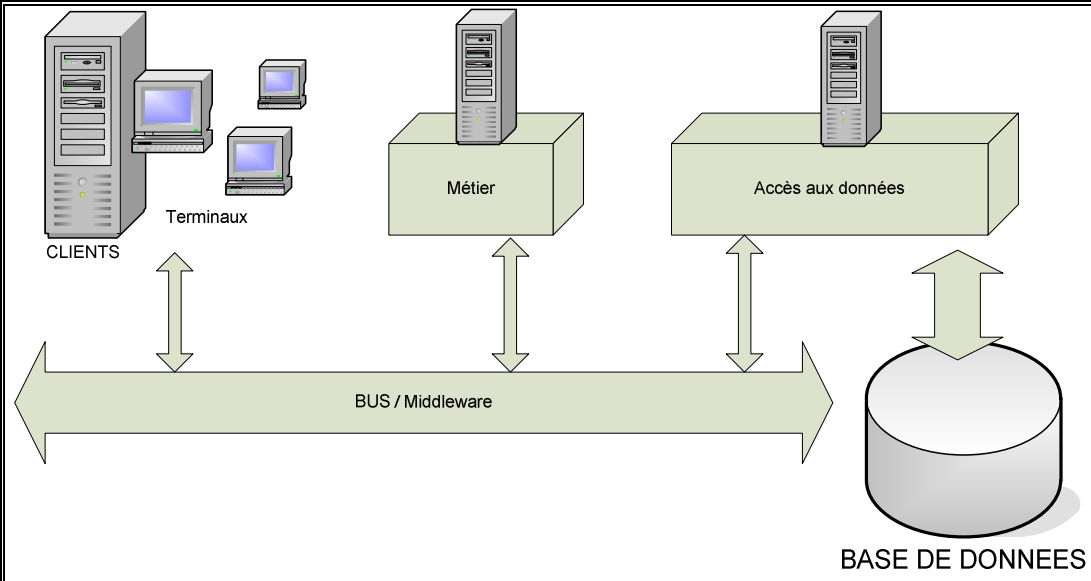
Vous avez droit à **4 points** négatifs sans pénalité.

NOM:	PRENOM:

Un Middleware est un framework, comme eclipse, qui assiste un développeur à développer les composants de son architecture logicielle		Q 1.
1	OUI	
2	NON	X

Une application dite "distribuée" est une application logicielle dans lequel les données informatiques sont réparties sur le réseau et accessibles par tout logiciel qui utiliserait un ORB		Q 2.
1	OUI	X
2	NON	

Une application dite "distribuée" est une application logicielle dans lequel les données informatiques sont		Q 3.
1	centralisées dans un singleton crée dans un programme accessible par tous les composants du réseau	
2	réparties dans des Factory répartis sur le réseau	X



Ce schéma représente une architecture 3-Tiers		Q 4.
1	OUI	X
2	NON	

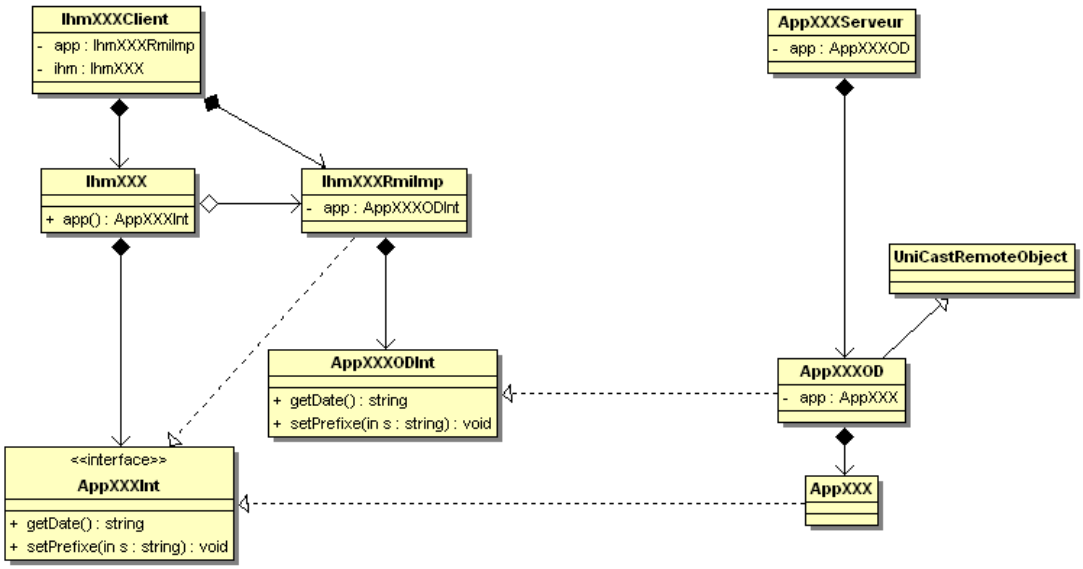
L'IDL (Interface Definition Language) est un langage informatique utilisé par les ORB pour :		Q 5.
1	générer le code permettant de développer les servants (ou Objets distants)	X
2	compiler les servants	

Un ORB (Object Request broker) est composé de, au moins : - un annuaire pour enregistrer les objets distribués, - un compilateur idl pour la génération des amorces er des squelettes - une API de classes prédéfinis pour programmer son application distribuée		Q 6.
1	OUI	X
2	NON	

Dans un ORB (Object Request broker) le rôle d'un annuaire est de servir d'intermédiaire dans l'envoi et la réception des messages échangés entre les objets distribués		Q 7.
1	OUI	
2	NON	X

Un Objet Distribué (ou Objet Distant) est un objet dont les méthodes sont accessibles depuis une autre machine.		Q 8.
1	OUI	X
2	NON	

Soit un objet quelconque Obj qui est une instance de la classe A qui n'hérite pas d'une autre classe et qui implémente l'interface AInt. En Java RMI, il est très facile de transformer cet objet en un objet distribué. Pour cela il suffit de :		Q 9.
1	faire que la classe A implémente aussi l'interface Remote	
2	faire que la classe A implémente l'interface Serializable, puis écrire cet objet dans un annuaire RMI	
3	créer un proxy de A . Ce proxy hérite de UnicastRemoteObject et implémente l'interface AInt qui hérite de l'interface Remote	X

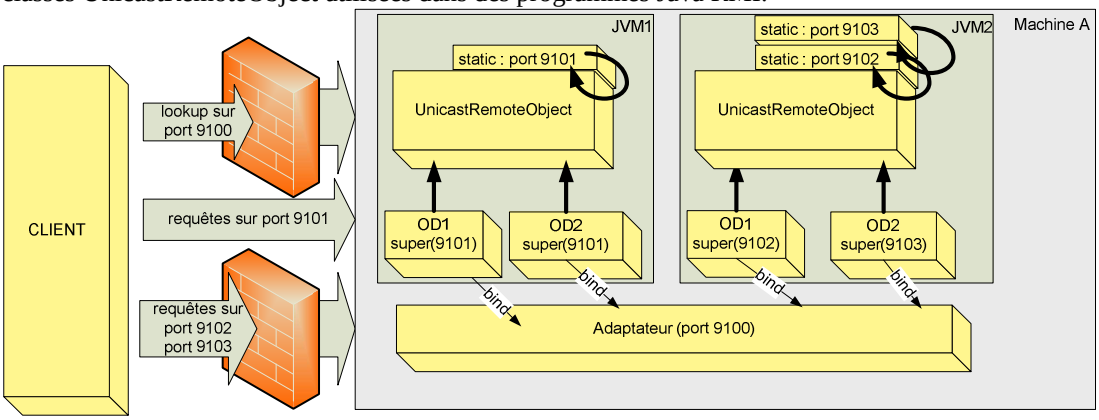


Ceci est le diagramme de classe d'un système composé d'un client IHM (classe IhmXXX) et de son applicatif (AppXXX) que l'on veut rendre distant.

La classe IhmXXXRmiImp est un Adaptateur de AppXXX :

Q 10.

1	OUI	
2	NON	X

Soit le schéma suivant qui représente un fonctionnement possible de plusieurs serveurs de socket des classes UnicastRemoteObject utilisées dans des programmes Java RMI.		Q 11.
		
1	On peut créer un nouvel OD dans la JVM1 qui s'exécute sur le port 9101	**X**
2	On peut créer un nouvel OD dans la JVM1 qui s'exécute sur le port 9102	
3	On peut créer un nouvel OD dans la JVM2 qui s'exécute sur le port 9101	

En RMI, l'appel d'une méthode distante, entre un client et un objet distribué RMI se fait de la manière suivante : 1/ le client récupère l'amorce (ou stub) de l'objet distribué 2/ le client utilise les méthodes de l'amorce		Q 12.
1	OUI	**X**
2	NON	

Un Design Pattern (DP) ou Patron est une norme de description des interfaces entre les composants d'une architecture logicielle orientée objet		Q 13.
1	OUI	
2	NON	**X**

Soit le code suivant d'implémentation d'un singleton :

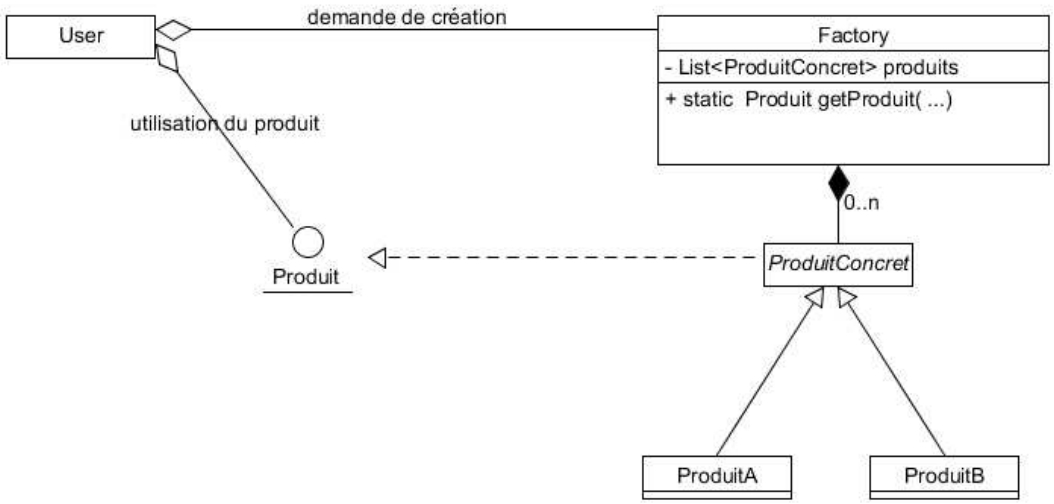
```
public class SingletonXXX
{
    static private SingletonXXX sg = new SingletonXXX ();

    private SingletonXXX () { }

    static public SingletonXXX getSingletonXXX()
    {
        return sg;
    }
}
```

Ce code est correct. — Q 14.

1	OUI	**X**
2	NON	

Le Singleton est le Design Pattern qui décrit comment il est possible de créer un objet unique parmi l'ensemble des objets répartis sur un réseau		Q 15.
1	OUI	
2	NON	**X**

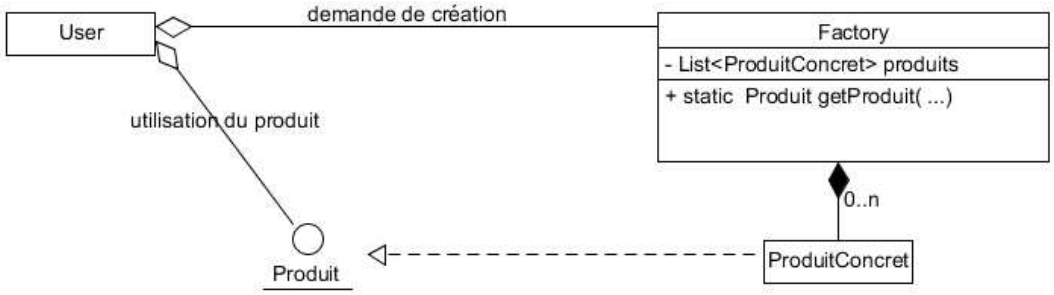
Q 16.



Ce DP est celui du Factory.
ProduiConcret est une classe abstraite dont héritent les classes ProduitA et ProduitB
Le rôle de la méthode getProduit du Factory est de créer des produits en faisant l'instanciation des classes ProduitA ou ProduitB.

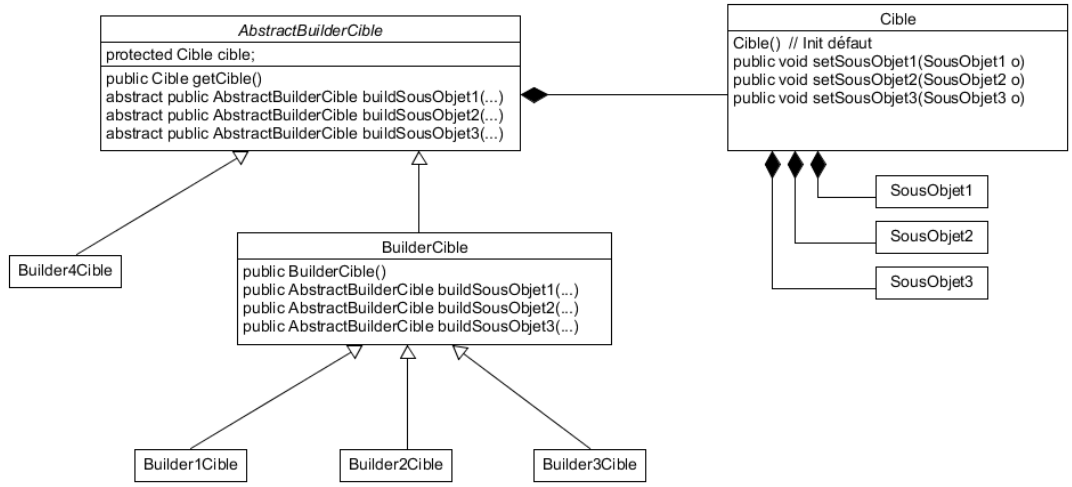
1	OUI	**X**
2	NON	

Q 17.



Ce DP est celui d'un Factory.

1	OUI	**X**
2	NON	



Q 18.

Ce DP est celui du Builder.
Le rôle de la classe BuilderCible est de construire totalement ou partiellement une instance de Cible

1	OUI	**X**
2	NON	

Le rôle du DP "Délégation" est de déléguer à une autre classe de réaliser des traitements qu'une classe aurait dû implémenter.		Q 19.
1	OUI	**X**
2	NON	

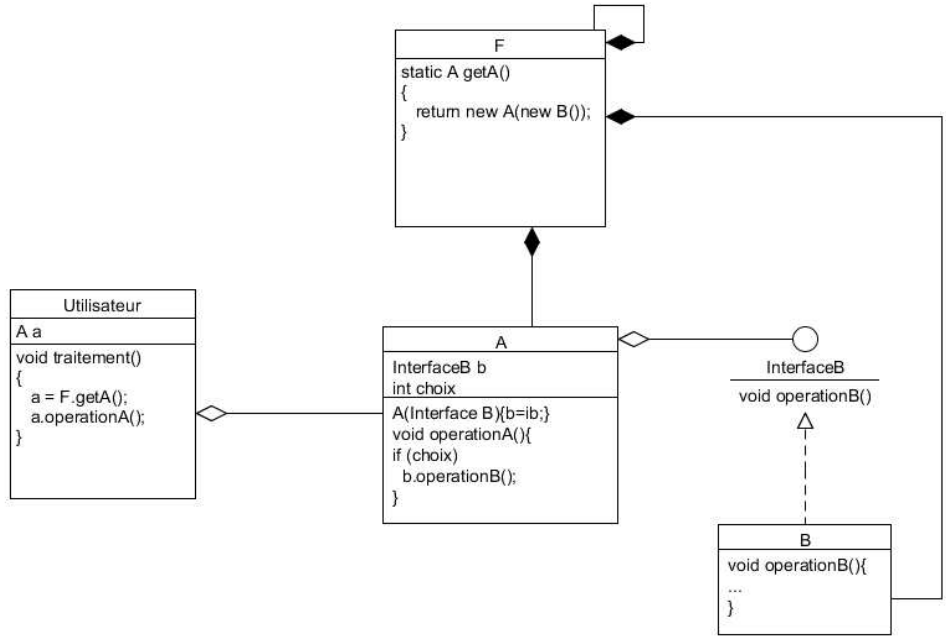
Le DP "Délégation" est utiliser dans le DP "injection de dépendance"		Q 20.
1	OUI	**X**
2	NON	

Le "Décorateur" est un Design Pattern qui permet d'étendre les méthodes d'une classe sans utiliser l'héritage		Q 21.
1	OUI	**X**
2	NON	

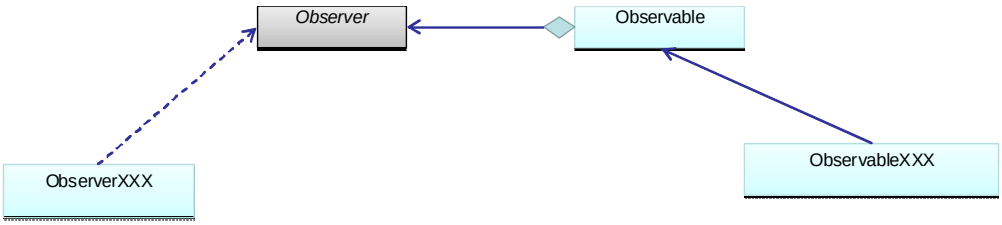
Si la classe A est un décorateur de la classe B alors les classes A et B héritent toutes deux d'une même classe abstraite.		Q 22.
1	OUI	**X**
2	NON	

L' "inversion de contrôle" est un principe de conception qui:.		Q 23.
1	permet à son application logicielle de contrôler dynamiquement les appels à une couche logicielle dont il utilise les fonctions	
2	permet de déléguer à un framework les appels aux fonctions de son application logicielle	**X**

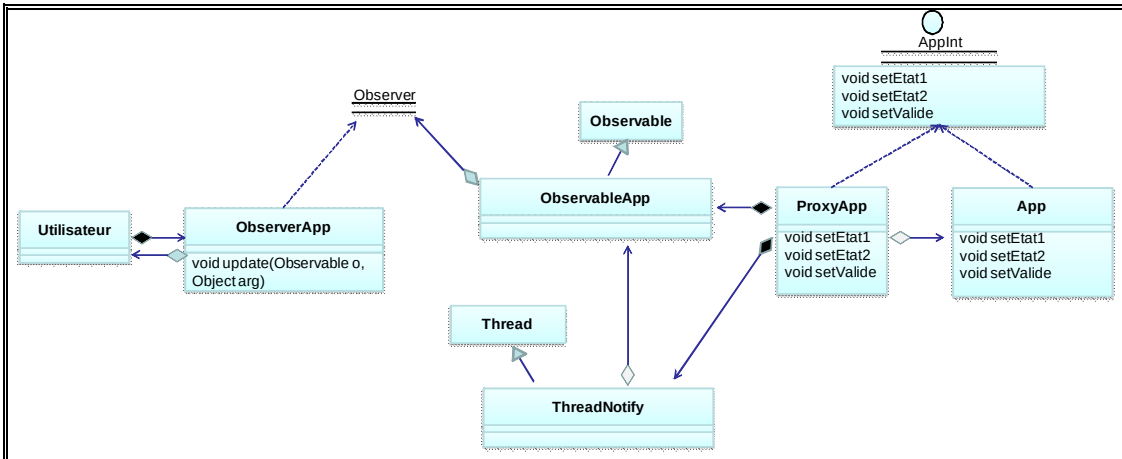
L'injection de dépendance utilise le principe de l'inversion de contrôle (IoC) appliqué au contrôle de la dépendance entre deux classes.		Q 24.
1	OUI	**X**
2	NON	

Le diagramme suivant :		Q 25.
		
représente		
1	une injection de dépendance par l'utilisation d'un setteur	
2	une injection de dépendance par l'utilisation d'un constructeur	X
3	une injection de dépendance par l'utilisation d'un proxy	

Dans le DP Observeur, l'Observable réalise la notification de tous ses observers toujours de manière synchrone		Q 26.
1	OUI	
2	NON	X

Soit le Design Pattern Observateur suivant :		Q 27.
		
La classe ObserverXXX implémente la méthode update de l'interface Observer qui est appelée par Observable		
1	OUI	X
2	NON	

Le DP Observateur est constitué d'une classe (Observable) et d'une interface (Observer).		Q 28.
1	Une fonction de la classe Observable est d'ajouter u nouvel Observer dans sa collection d'Observer	X
2	L'interface Observer est une interface qui doit être implémentée par la classe Observable	



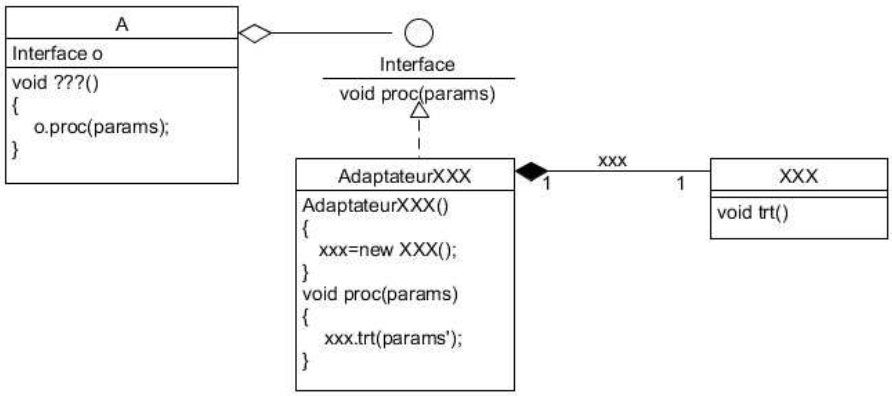
Q 29.

Ce schéma de Design Pattern est :

1	Le DP Observer/Observable de type "pull"	
2	Le DP Observer/Observable de type "push synchrone"	
3	Le DP Observer/Observable de type "push asynchrone"	X

Soit le diagramme de classe suivant :

Q 30.



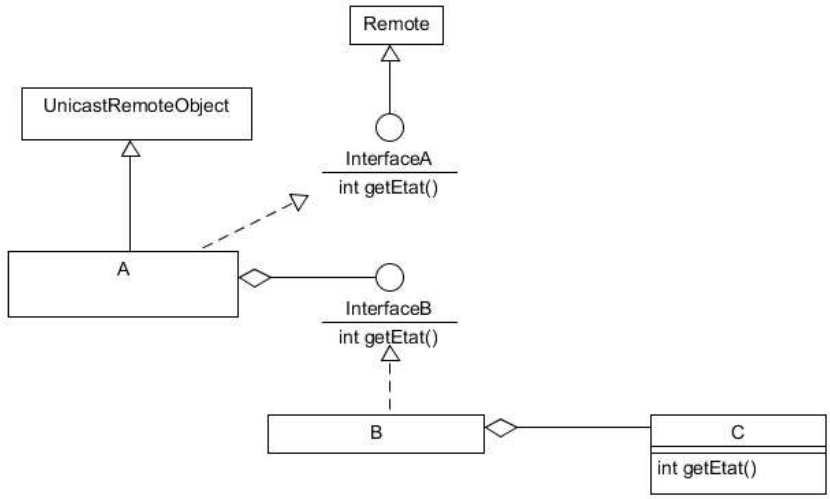
Ce diagramme de classe représente celui d'un DP Adaptateur car la classe XXX ne pouvant pas implémenter l'interface Interface, on crée une classe AdaptateurXXX qui le fait pour elle

1	OUI	X
2	NON	

Un proxy est une classe se substituant à une autre classe. Par convention et simplicité, le proxy implémente la même interface que la classe à laquelle il se substitue.

Q 31.

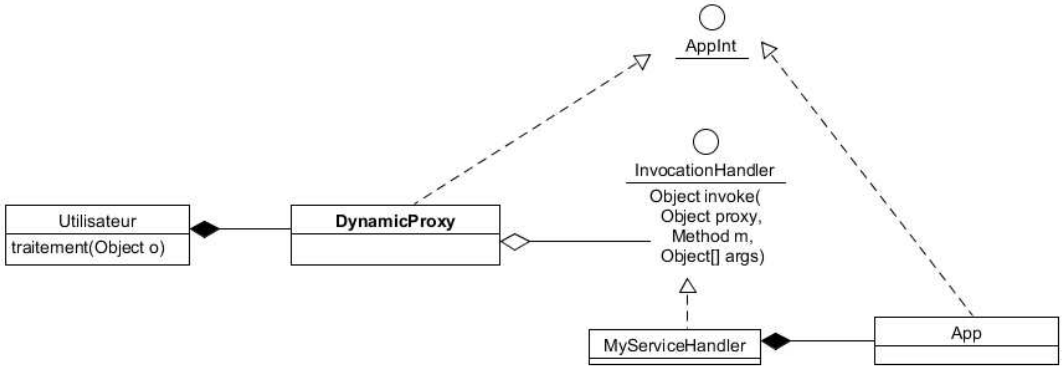
1	OUI	X
2	NON	



Q 32.

Ce diagramme de classe est la conception d'un Objet Distribué dans lequel :

1	B est un Proxy de C	
2	B est un Adaptateur de C	**X**



Q 33.

Ce schéma est celui du DP Dynamic proxy.
Le rôle de la classe MyServiceHandler est ici de :

1	créer une instance d'une classe qui implémente l'interface AppInt, dont le rôle (l'instance) est de servir de proxy à l'appel des méthodes de App	**X**
2	d'implémenter toutes les méthodes de l'interface AppInt	
3	d'appeler les méthodes de App décritent dans l'interface AppInt	**X**

La définition de l'envoi d'un message **synchrone** entre un producteur et plusieurs consommateurs est que, avant d'envoyer un nouveau message, le producteur attend que le message envoyé ait été consommé par tous les consommateurs	Q 34.
1 OUI	**X**
2 NON	

Dans la communication synchrone via un "canal d'évènement" entre un producteur et un consommateur, le producteur utilise un proxy de consommateur afin de lui pousser un évènement	Q 35.
1 OUI	**X**
2 NON	

Fin du QCM

Suite (Tournez la page)

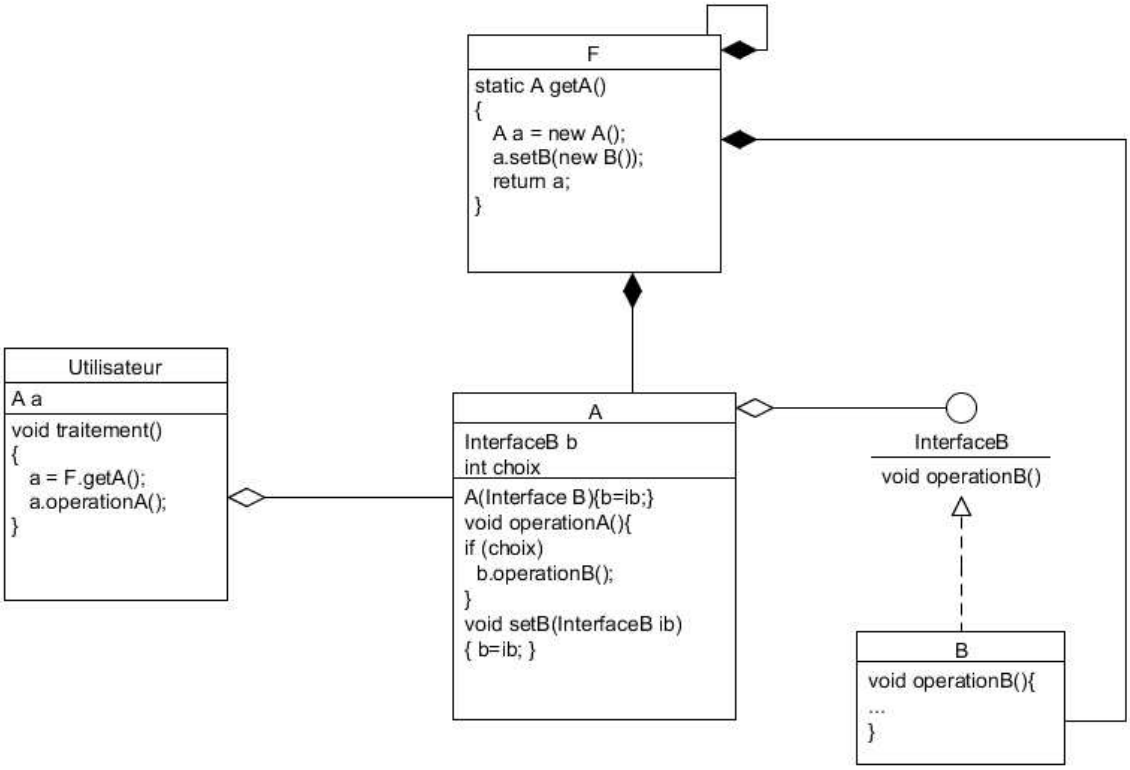
2. Questions libres (15 points)

Chaque question est notée sur 5 points.
*Vous répondez à ces questions sur une **copie vierge double** en mettant bien le numéro de la question, sans oublier votre nom et prénom.*
Vous mettez le QCM dans la copie vierge double.

QUESTION NUMERO 1

Ecrivez le diagramme UML du DP de **l'injection de dépendance** par setteur et expliquez le rôle de ce DP.

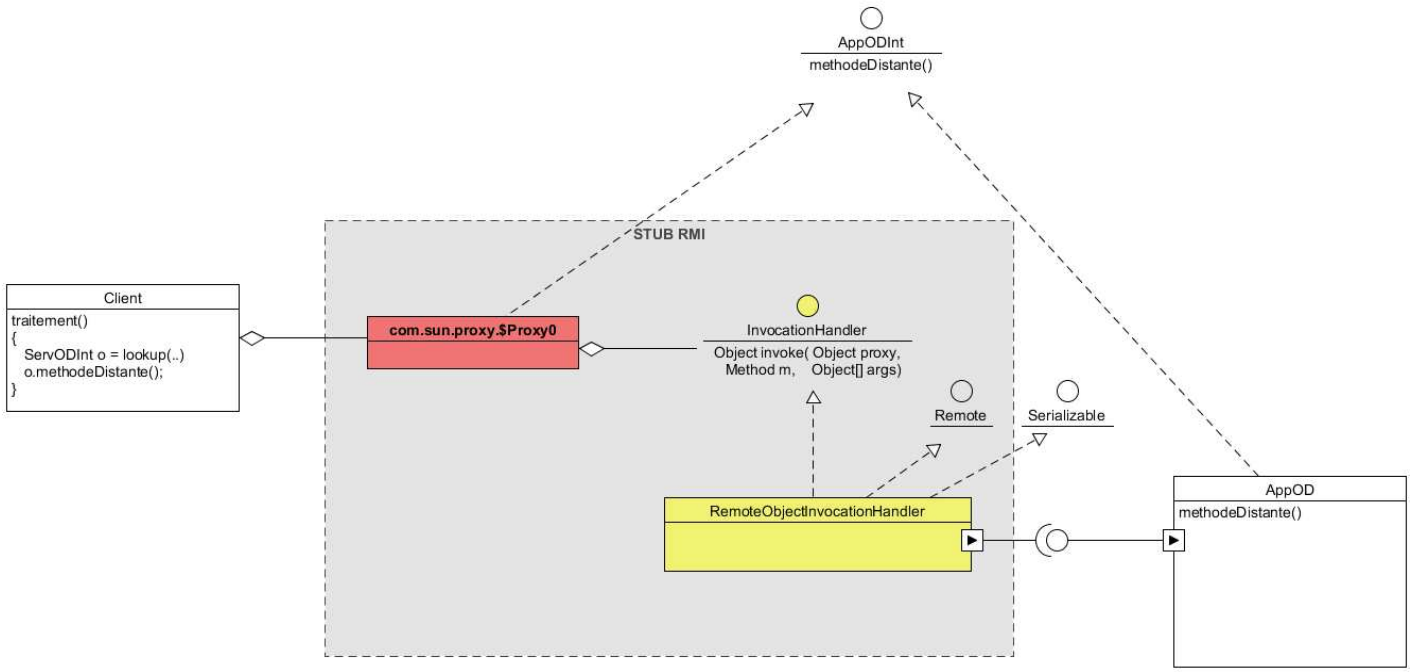
Correction :
Le diagramme est le suivant :



Le rôle de ce DP est d'injecter une dépendance entre deux classes. Ici, F (ex: Framework) fait l'injection de B dans A en utilisant un setteur de A. Ce setteur accepte en paramètre toute classe qui implémente l'interface InterfaceB. L'utilisateur délègue à F le soin de réaliser cette injection. Il ne connait pas ainsi la classe B.

L'injection de dépendance utilise le principe de l'inversion de contrôle (IoC) appliqué au contrôle de la dépendance entre deux classes.
La classe F utilise souvent un fichier de configuration pour déterminer quelle injection doit être réalisée.

QUESTION NUMERO 2



Ce diagramme UML est le diagramme de conception d'un stub RMI qui utilise le DP DynamicProxy.

Expliquez le fonctionnement de ce diagramme dans le cadre du protocole RMI.

Correction :
L'implémentation de la méthode invoke de l'interface InvocationHandler dans RemoteObjectInvocationHandler consiste à écrire et lire sur le socket (avec serialization des paramètres).

Ainsi, le lookup construit un DynamicProxy. Le mandataire réalise les accès et reste transparent pour l'utilisateur.

Le Stub du protocole RMI est un DynamicProxy.

QUESTION NUMERO 3

Citez 3 exemples de l'utilisation du DP Proxy. Précisez pour chacun de ces exemples le fonctionnement du proxy.

Correction :

Exemple 1 : Un proxy qui permet de réaliser une notification à un utilisateur client pour chaque utilisation des setteurs dans la couche serveur par exemple.
Dans ce cas le proxy implémente les méthodes de setteur en réalisant successivement les deux actions suivantes :

- la notification à travers un observable de l'appel au setteur (nom de l'attribut + la valeur de l'attribut)
- l'appel au setteur de la classe applicative dont il est le proxy.

Exemple 2 : Un proxy pour vérifier que à chaque appel d'un traitement métier, méthode d'un contrôleur d'un MVC, que l'utilisateur est bien connecté : vérification du user et mot de passe.
Dans ce cas le proxy implémente les méthodes du contrôleur en réalisant les deux actions suivantes :

- vérifier que un utilisateur est bien connecté (attribut du controleur)
- l'appel au traitement du contrôleur si correct sinon erreur.

Exemple 3 : Un proxy pour utiliser de manière distante les méthodes d'une classe (exemple RMI).
Dans ce cas le proxy est créé par le lookup dont le rôle est d'implémenter chaque méthode de l'interface distante en réalisant les actions suivantes :

- connexion au serveur de socket de UnicatRemoteObject
- écriture d'une requête sur le socket (encodage du nom de la méthode et des paramètres de la méthode)

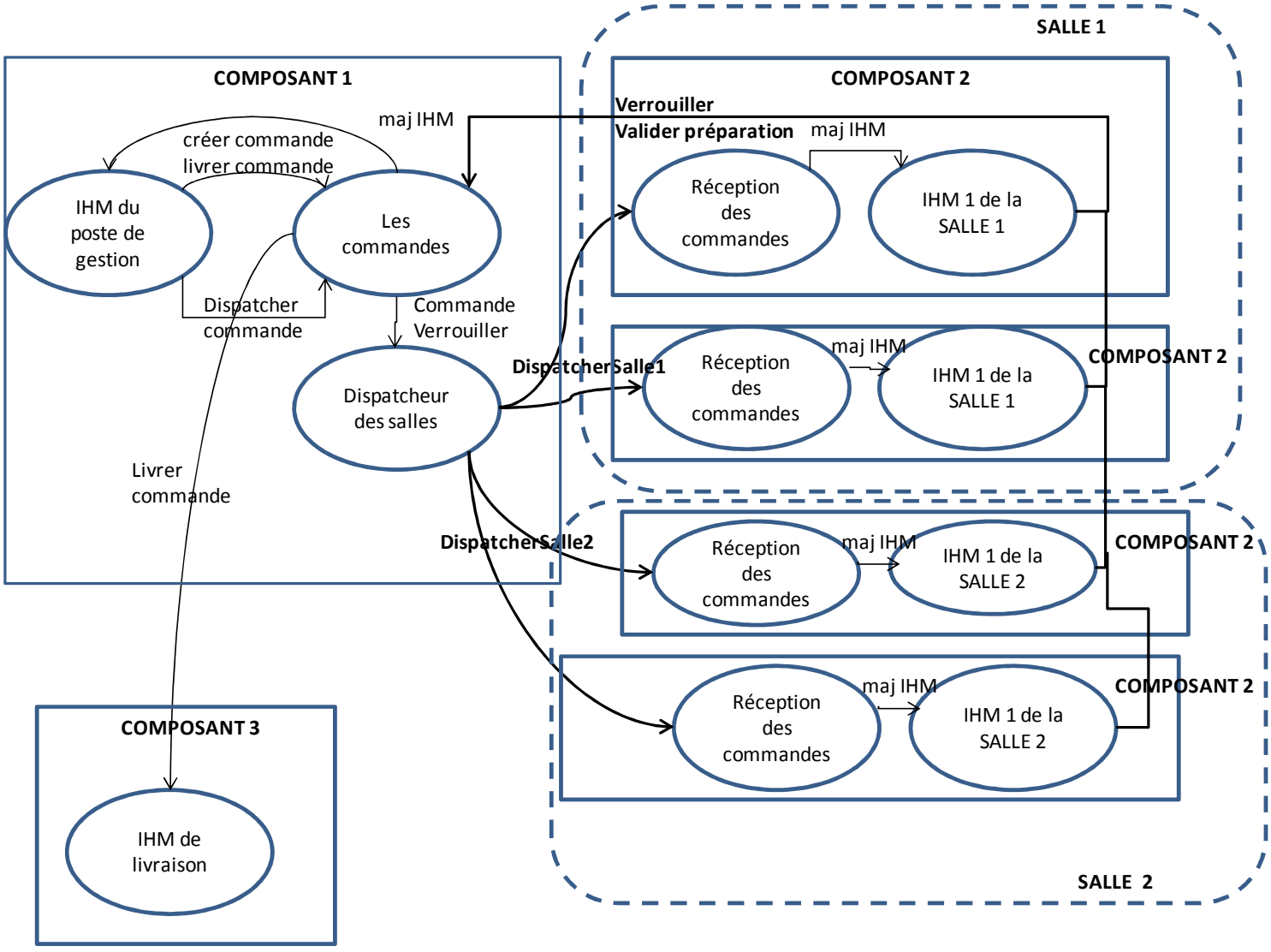
- attente de réponse du serveur de socket
- lecture sur le socket de la valeur de retour et décodage
- retourne la valeur de retour.

Fin de la 1ère partie sans document

2ème PARTIE – AVEC DOCUMENT (durée: 1h30)

3. PROBLEME (50 points)

1/
[10 POINTS]

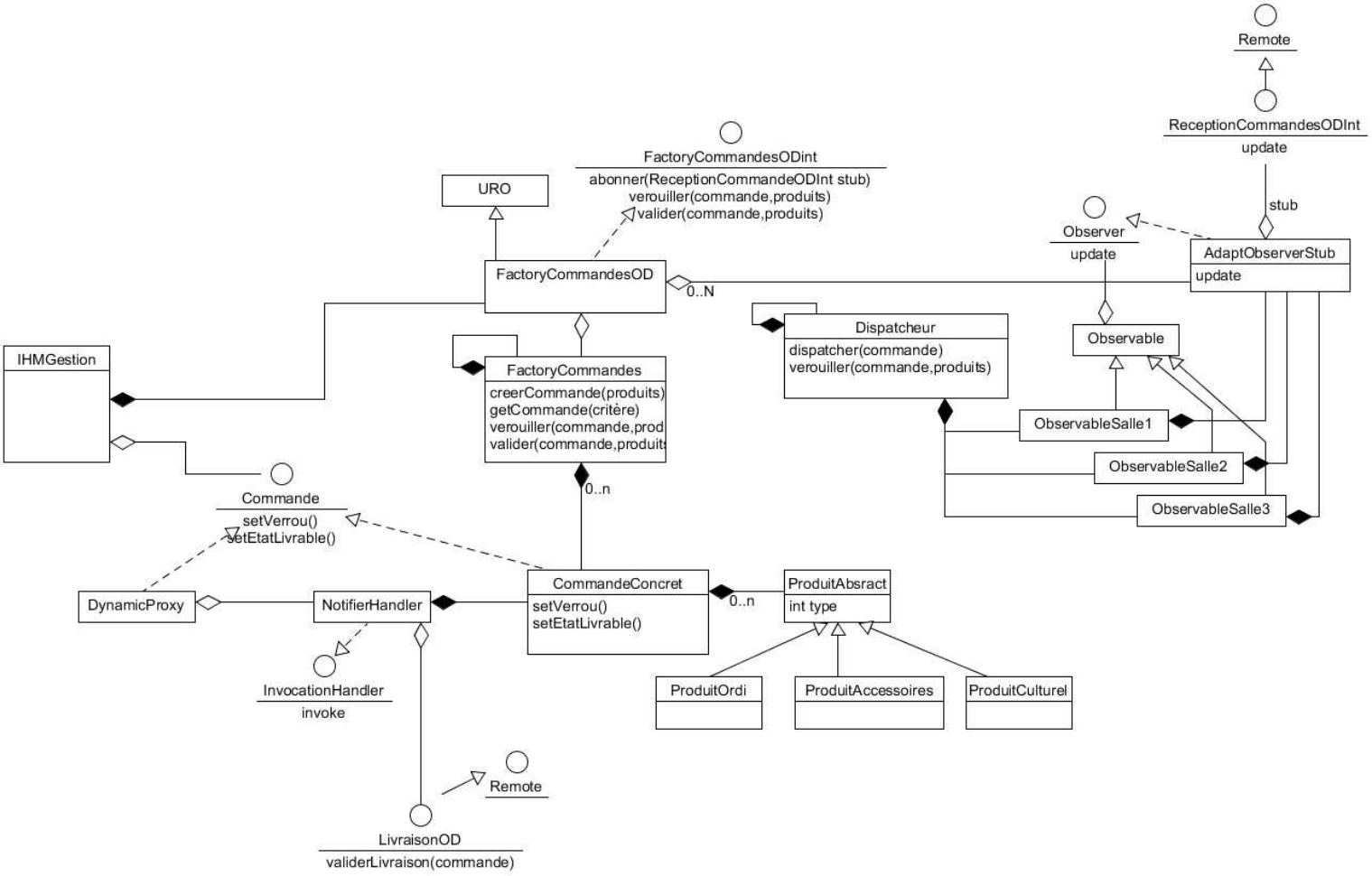


Le composant 1 :
Ce composant est le poste de gestion des commandes.
L'IHM du poste de gestion demandes à un factory de commandes de créer une commande, dispatcher les produits d'une commande aux 3 salles ou de livrer une commande terminée au poste de livraison.
Le dispatcheur de commandes permet d'envoyer les commandes sur chacun des postes d'IHM des 3 salles. Il met en évidence dans la commande les produits en fonction de la salle destinatrice.
Le factory de commande est prévenu des produits des commandes en cours de préparaton (vérouillés) et des commandes partiellement terminées de chacune des salles. Il dispatche l'état de vérouillé vers les postes d'IHM de la salle concerné. C'est donc le composant 1 qui assure la notification à toutes les IHM d'une salle quand un opérateur de cette salle verrouille une des commandes.

L'IHM du poste de gestion permet de consulter les commandes terminées par les différentes salles. Quand la commande est complète, il met à jour la commande qui prévient le poste de livraison.

Le composant 2 :
Ce composant est le poste de préparation des commandes. Il existe plusieurs de ce poste dans une même salle. Il réceptionne les commandes à préparer et verrouillés envoyés par le dispatcheur du composant 1. Il met à jour l'IHM en conséquence. Par l'IHM, l'opérateur verrouille une commande qui prévient les commandes du composant 1 qui via le dispatcheur prévient toutes les IHM de la salle que la commande est verrouillé. Les autres IHM de la salle sont donc prévenues.
L'IHM permet de valider la commande partiellement terminée qui met à jour les commandes du composant 1.

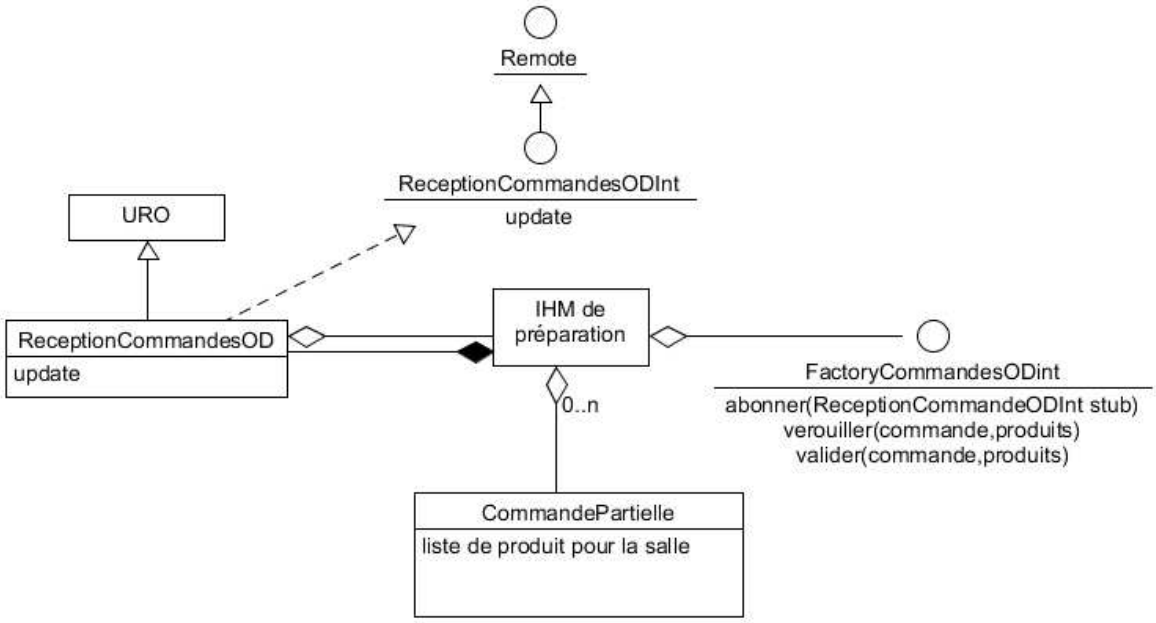
2/
[COMPOSANT 1]
[30 POINTS] (5 points pour explications + 25 points pour le schéma)



Les commandes sont gérées dans un factory (FactoryCommandes) composé de commandes. Chaque commande (CommandeConcret) est composée des produits (1 sous-classe par héritage que de type de produit). Les commandes sont vues par une interface par le client (principe du DP factory et ce qui permettra de mettre en place un Dynamic Proxy).
Ce Factory est un singleton dans le composant 1.
Pour que la factory soit accessible depuis le composant 2 (pour réaliser les actions de validation d'une commande partielle ou pour verrouiller une commande partielle) on crée par agrégation un factory distant (FactoryCommandesOD) dont les méthodes distantes sont utilisées par le composant 2.
Pour dispatcher les commandes, on crée le Dispatcheur (traitement métier) qui analyse la commande. Il notifie à tous les postes de préparation (composant 2) les commandes qui les concernent. Pour réaliser cette notification, on décide d'utiliser le DP Observable/Observer distant. On crée 3 Observables, un pour chaque salle. Chaque poste de préparation d'une même salle s'abonne au même observable.

On crée un DynamicProxy qui surcharge la méthode setLivrable utilisée par l'IHM pour marquer la commande comme livrable. Cette surcharge utilise une interface distante de l'IHM de livraison pour réaliser la livraison.

[COMPOSANT 2]
[10 POINTS] (3 points pour les explications, 7 points pour le schéma)



L'IHM de préparation utilise l'interface distante du factory des commandes du composant 1 pour s'abonner en lui passant le stub de ReceptionCommandeOD. Elle utilise cette interface distante pour verrouiller une commande et pour valider la préparation partielle d'une commande.
L'IHM gère une liste des commandes partielles à préparer.
ReceptionCommandeOD implémente la méthode update à travers laquelle est passé en paramètre une CommandePartielle (crée dans le Dispatcheur du composant 1) dans laquelle est transmis le verrou sur la commande partielle.

REMARQUE : une variante qui a été réalisé par quelques auditeurs a été de :

- décider que toutes les commandes sont des OD contenant l'état de verrou
- c'est l'IHM de préparation qui interroge le factory OD des commandes pour choisir une commande à préparer et la verrouille. Ainsi, une autre IHM de préparation ne peut pas choisir une commande verrouillée
- cela simplifie la conception, cela est correct sur le principe mais est un peu contraire au sujet qui précise : "le poste de gestion permet de dispatcher les produits des commandes vers les salles".
- j'ai considéré comme juste une telle solution.