IPST-CNAM
Intranet et Designs patterns
NSY 102
Jeudi 16 Mai 2024

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.

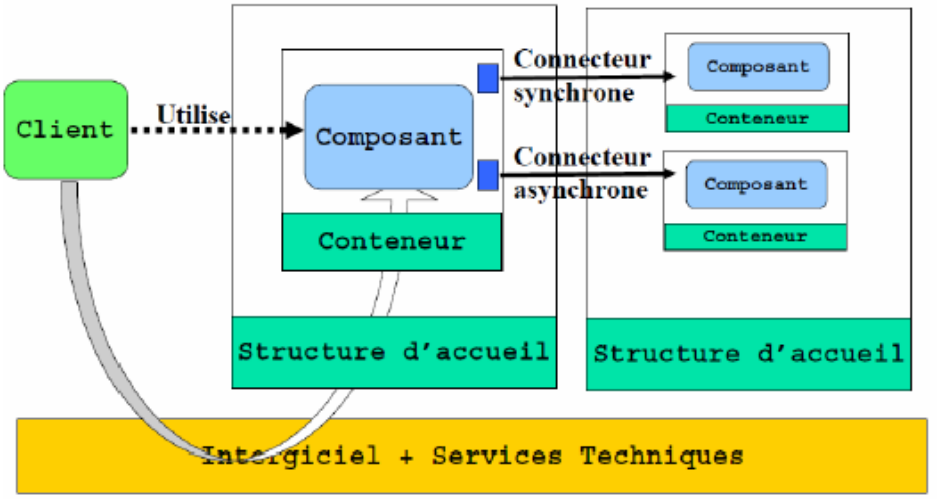
Vous avez droit à **4 points** négatifs sans pénalité.

NOM:	PRENOM:

La "Démarche d'Architecture" est un guide méthodologique pour créer le dossier d'architecture d'un Système d'Information.		Q 1.
1	OUI	X
2	NON	

A l'issu de la Configuration Architecturale, on obtient la description d'un "réseau" de description du Système d'Information dont les nœuds sont les Composants et les liens les Connecteurs.		Q 2.
1	OUI	X
2	NON	

Soit le schéma suivant de description d'une architecture type à base de composants :



Ce schéma montre qu'un Client du SI basé sur ce type d'architecture utilise directement les composants du SI.

Q 3.

1	OUI	
2	NON	X

Il existe deux façons pour utiliser les méthodes distantes d'un objet distant : 1/ demander le stub de connexion à un annuaire, 2/ demander le stub de connexion à celui qui a créé l'objet distant (exemple un factory); puis d'utiliser ce stub pour appeler les méthodes distantes		Q 4.
1	OUI	X
2	NON	

Dans la démarche de conception logicielle utilisée dans ce cours, le diagramme de communication correspond à la Configuration Architecturale de la démarche d'architecture.		Q 5.
1	OUI	X
2	NON	

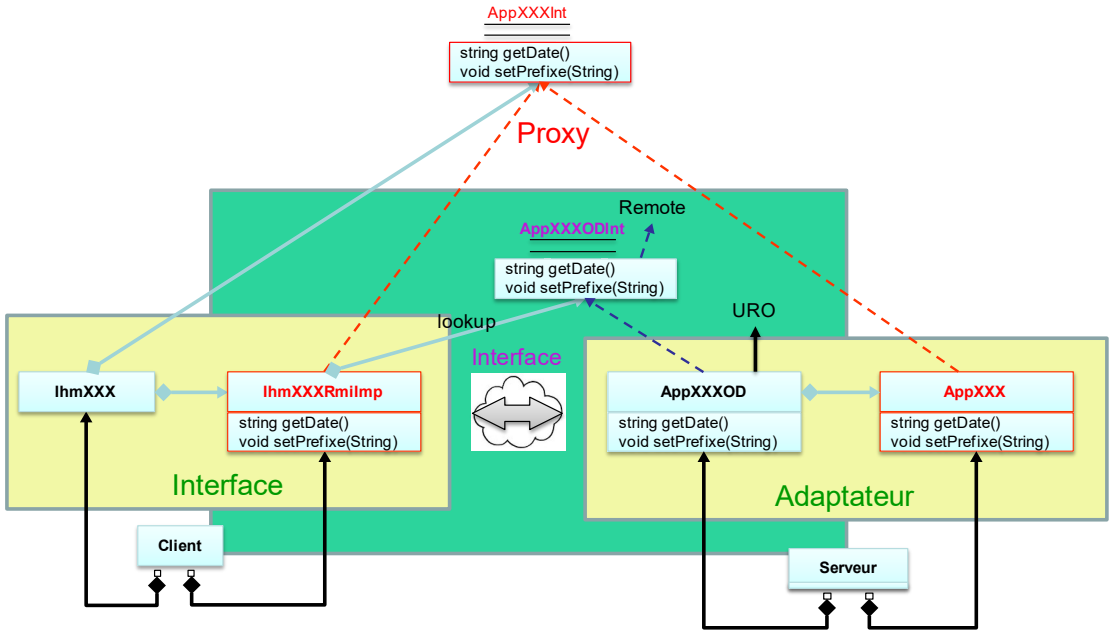
Dans la démarche de conception logicielle utilisée dans ce cours, le diagramme de communication décrit des liens de communication entre chacun des composants et sous-composants. Ces liens sont :		Q 6.
1	toujours des communications distantes.	
2	des liens non orientés. Leurs orientations sera résolues à l'étape suivante des diagrammes de classes.	
3	Des liens orientés, distants entre les composants logiciels, ou locaux entre les sous-composants.	X

La validation d'un diagramme de communication consiste à vérifier que tous les Cas d'Utilisation sont réalisables à travers le cheminement des liens entre les composants et les sous-composants.		Q 7.
1	OUI	X
2	NON	

Dans la description de l'architecture technique, un connecteur est un lien de dépendance entre deux composants qui peut être réalisé par le principe du design pattern de l'injection de dépendance.		Q 8.
1	OUI	X
2	NON	

Soit un objet quelconque Obj (instance de la classe A qui implémente l'interface IntA). En Java RMI, il est très facile de transformer cet objet en un objet distant. Pour cela il suffit de :		Q 9.
1	créer une Adaptateur de la classe A. Cet adaptateur hérite de UnicastRemoteObject et implémente l'interface distante AdaptIntA qui hérite de Remote.	X
2	créer un proxy de A . Ce proxy hérite de UnicastRemoteObject et implémente l'interface de A	

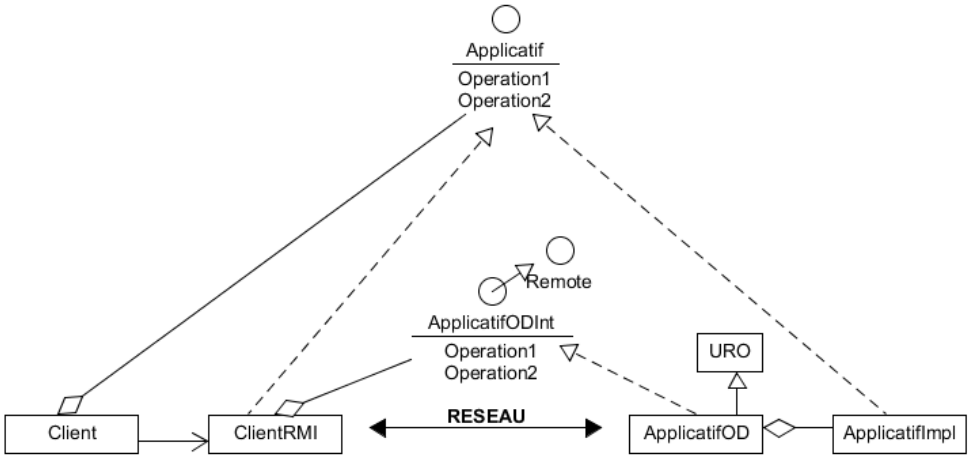
Q 10.

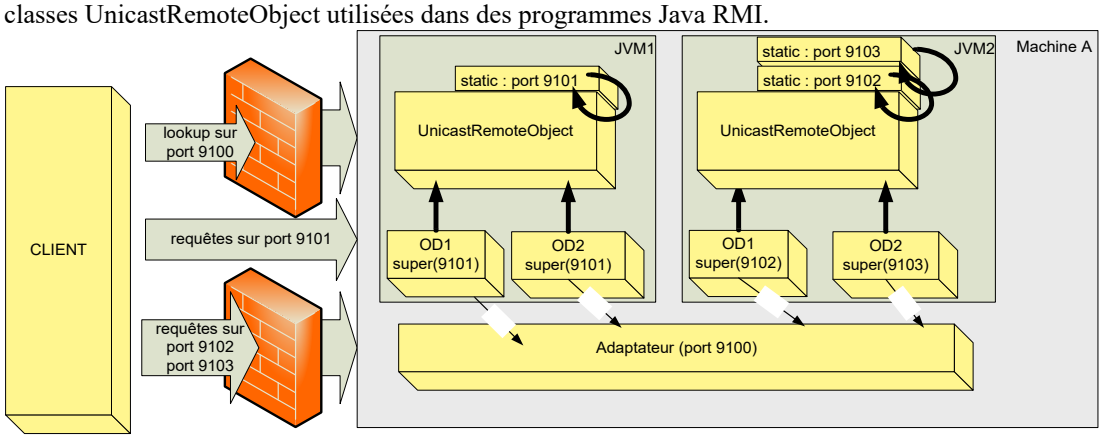


Ceci est le diagramme de classe d'un système composé d'un client IHM (classe IhmXXX) et de son applicatif (AppXXX) que l'on veut rendre distant.

Le rôle de l'interface AppXXXODInt est essentiel car :

1	Elle décrit les méthodes que la classe IhmXXXRmiImp doit implémenter	
2	Elle est une interface RMI, une description des méthodes distantes de AppXXXOD	X
3	Elle décrit les méthodes distantes que l'objet distant AppXXXOD doit implémenter	X

Ceci est le diagramme de classe d'un système composé d'un client et de son applicatif que l'on veut rendre distant.		Q 11.
		
1	Client est un proxy de ApplicatifImpl	
2	ApplicatifOD est un proxy de ApplicatifODInt	
3	ClientRmi est un proxy de ApplicatifImpl	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 12.
		
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 9103	X

En Java, la méthode **public static Remote toStub(Remote obj)** permet de convertir un objet distant (obj) en un stub qui est passé en paramètre d'une méthode distante afin que le client puisse se connecter à l'objet distant.		Q 13.
1	OUI	X
2	NON	

Les "patrons de conception" ou Design Patterns" sont des modèles standard et réutilisables de conception de la solution à la problématique de la réalisation d'une partie d'un logiciel informatique.		Q 14.
1	OUI	X
2	NON	

Le rôle du DP Singleton est de :		Q 15.
1	créer un objet distant unique sur le réseau (Singleton d'un Objet distant)	
2	limiter le nombre d'instance d'une classe qui dans le cas d'un singleton est toujours égal à 1.	X
3	pouvoir accéder à un objet principal et unique de n'importe où dans le code sans avoir besoin de le passer en paramètre ou en attribut d'un objet.	X

Il existe une façon unique de créer un singleton de classe Singleton : instancier le singleton dans la définition de la classe.		Q 16.
1	OUI	
2	NON	X

Il est interdit de créer des setteurs sur un singleton permettant de modifier les attributs du singleton.		Q 17.
1	OUI	
2	NON	X

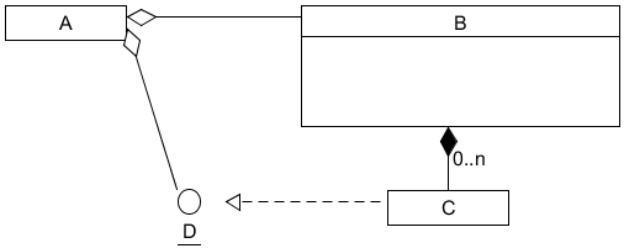
Dans une application distribuée, un **factory** peut être un objet distant, enregistré dans un annuaire, et dont le rôle est de créer des objets distants qui sont utilisés par le client à travers un proxy client (appelé "stub")		Q 18.
1	OUI	X
2	NON	

Le rôle de la classe abstraite, dans un Factory, est de servir de proxy entre les classes concrètes d'implémentation des produits du factory et la classe qui utilise le Factory.		Q 19.
1	OUI	
2	NON	X

Dans le DP Factory, les produits du Factory sont toutes des instances de classes qui héritent d'une même classe abstraite et/ou implémentent la même interface		Q 20.
1	OUI	X
2	NON	

Dans le DP Factory, les produits du Factory sont des objets uniques. Ces objets sont donc vus par le reste du programme comme des singletons.		Q 21.
1	OUI	
2	NON	X

Ce DP est celui du Factory. Q 22.

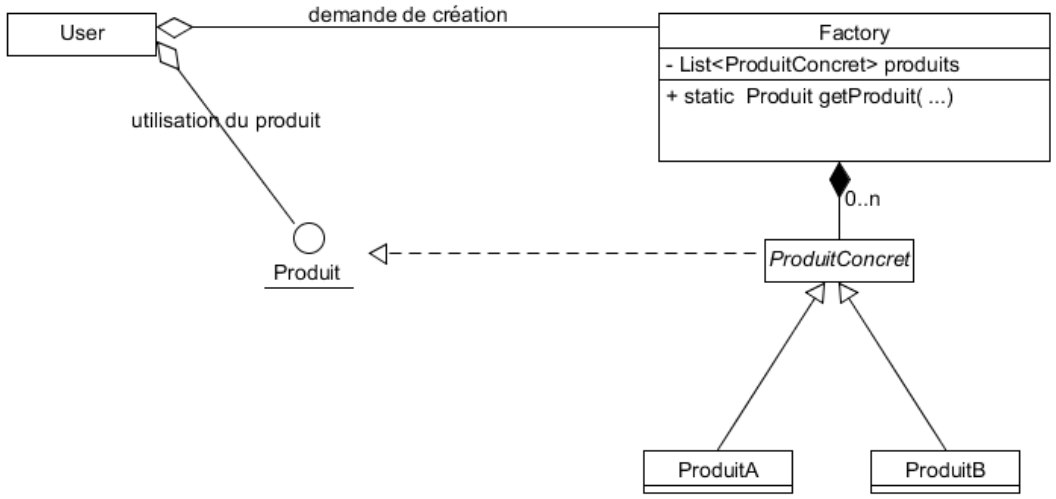


La signification des lettres A, B, C et D est :

1	A = Client; B=Product (interface); C=Concrete Product; D = Factory	
2	A=Client; B=Factory; C=Concrete Product; D=Product (interface);	X
3	A=Factory; B = Concrete Product; C=Product (Interface); D=Client	

Dans le DP Factory, les produits du Factory sont nécessairement stockés en mémoire du Factory.		Q 23.
1	OUI	
2	NON	X

Q 24.



Ce DP est celui du Factory.
Le rôle de l'interface **Produit** est d'abstraire, à la classe User, la classe ProduitA ou ProduitB utilisé pour créer l'objet

1	OUI	X
1	NON	

Dans un DP Factory, le Client doit connaître les différents types de produits créés par le factory		Q 25.
1	OUI	
2	NON	X

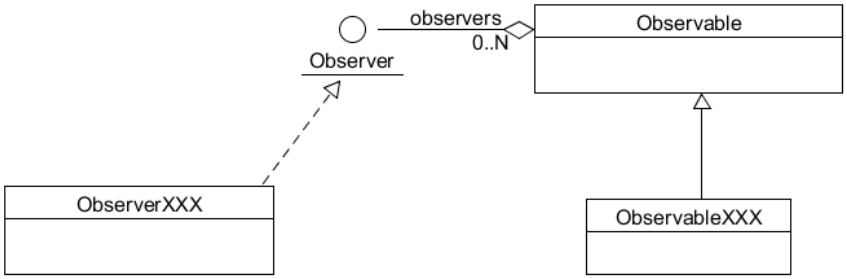
Le rôle du DP "Délégation" est de déléguer à une autre classe de réaliser des traitements qu'une classe aurait dû implémenter.		Q 26.
1	OUI	X
2	NON	

Le rôle du DP **Itérateur** est d'abstraire la façon de parcourir des éléments contenus dans une collection sans en connaître la structuration et/ou la séquentialité de ces éléments.		Q 27.
1	OUI	X
2	NON	

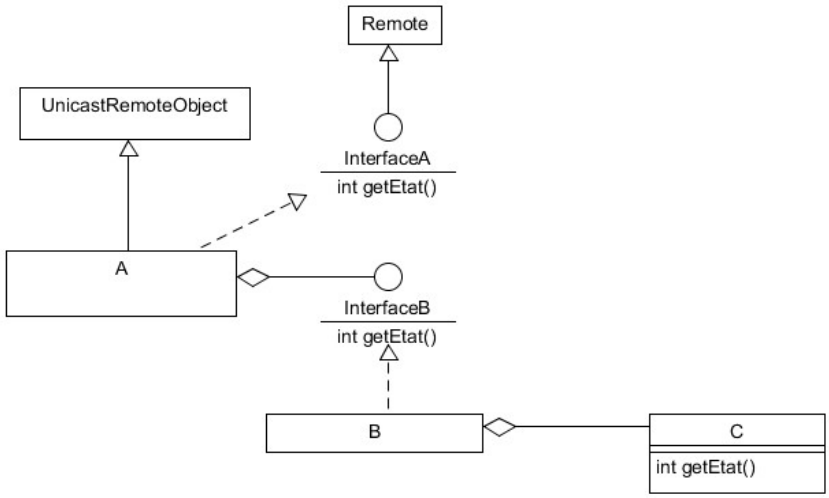
Le DP Visiteur (ou Visitor) permet d'ajouter de nouveaux comportements à une hiérarchie de classes sans modifier l'existant.		Q 28.
1	OUI	X
2	NON	

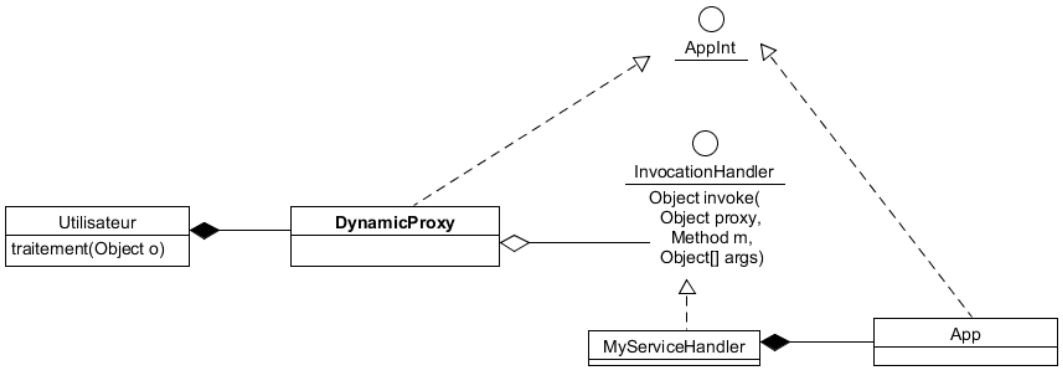
Si la classe A est un décorateur de la classe B alors les classes A et B héritent toutes deux d'une même classe abstraite.		Q 29.
1	OUI	X
2	NON	

Toute classe qui implémente l'interface Observer peut devenir un observateur (Observer) de n'importe quel observé (Observable)		Q 30.
1	OUI	**X**
2	NON	

Soit le Design Pattern Observateur décrit (volontairement simplifié) de la manière suivante :		Q 31.
		
1	La classe ObservableXXX notifie les évènements à une instance de Observable	
2	La classe ObserverXXX implémente la méthode update de l'interface Observer qui est appelée par Observable	**X**
3	La classe Observable pousse (modèle du "push") les évènements à ObserverXXX	**X**

Les descriptions suivantes sont des modèles de communication asynchrones :		Q 32.
1	un serveur pousse ses évènements dans une file d'attente par client connecté (intermédiaire), et les clients tirent ses évènements à leur rythme	**X**
2	un serveur pousse son évènement dans un proxy de consommateur, et à son tour, le proxy de consommateur pousse l'évènement au consommateur	
3	un serveur appelle la méthode distante d'un client afin de lui pousser l'évènement	

		Q 33.
		
Ce diagramme de classe est la conception d'un Objet Distant suivant le modèle :		
1	d'un DP Proxy	
2	d'un DP Adaptateur	**X**



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

Q 34.

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	
2	d'implémenter toutes les méthodes de l'interface AppInt	
3	d'appeler les méthodes de App décritent dans l'interface AppInt	X

Le principe d'un MOM (Model Orienté Message) est d'utiliser un composant logiciel qui sert d'intermédiaire entre les producteurs et les consommateurs. Ce composant logiciel utilise :

Q 35.

1	un DP Factory de canaux d'évènement pour créer les canaux d'évènement	X
2	Un DP Observer/Observable pour notifier les Consommateurs des évènements produits par les Producteurs	X
3	Un DP ModelVueControleur pour produire les évènements des Producteurs	

Fin du QCM

Suite (Tournez la page)

2. Questions libres (15 points)

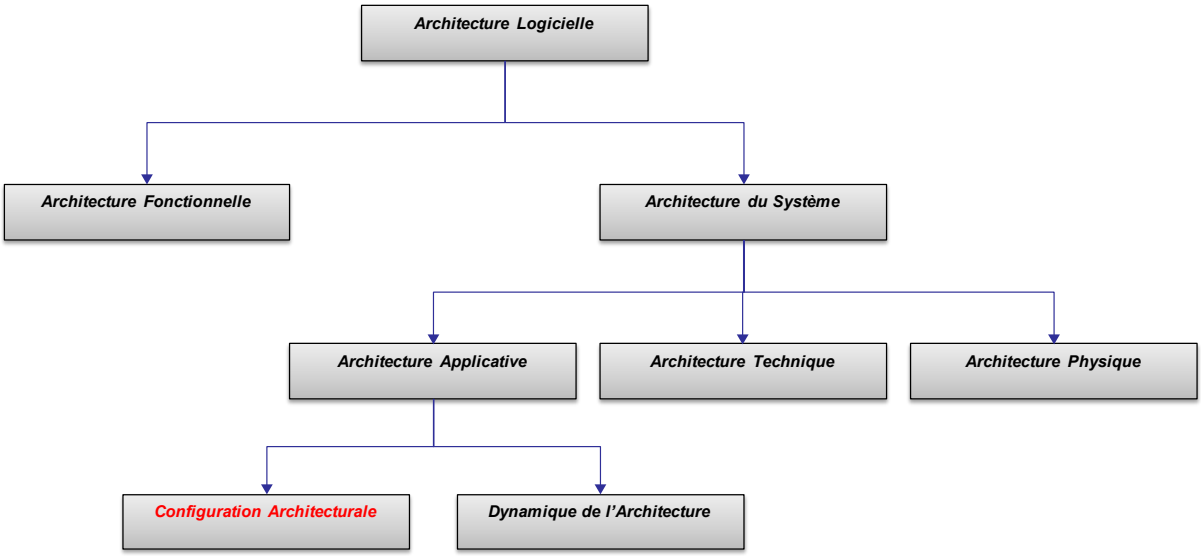
Chaque question est notée sur 5 points.
*Vous répondez à ces questions sur une **copie vierge double** en mettant bien le numéro de la question, sans oublier votre nom et prénom.*
Vous mettez le QCM dans la copie vierge double.

QUESTION NUMERO 1

Le diagramme de **Communication** UML permet de mettre en évidence des aspects importants dans la conception d'un Système d'Information. Quels sont ces aspects ? (En citer au moins 5)
- Identifier les composants logiciels (JVM) du Système d'Information
- Identifier les liens de communication entre les composants et les sous-composants
- Identifier les sous-composants qui sont pour certains des Designs Patterns
- Mettre en évidences les points de communications distants
- Caractériser le sens de communication entre les composants et les sous-composants
- Permettre l' « exécution » des cas d'utilisation à travers les composants et sous-composants

QUESTION NUMERO 2

La démarche d'architecture se décompose en différentes parties :



Quel est le rôle de :
- la Configuration Architecturale
- la Dynamique de l'Architecture
- l' Architecture Technique
- l'Architecture Physique
- ~~l'Architecture Fonctionnelle~~

(Pas plus de une à deux phrases pour chacun. Soyez précis, pas de paraphrase.)
La **Configuration Architecturale** est l'architecture logicielle des composants et sous-composants liés, entre eux par des connecteurs qui représentent l'échange des informations entre ces derniers.
La **Dynamique de l'Architecture** est le comportement dynamique entre les composants de l'architecture et la dynamique interne des composants et sous-composants.
L'**Architecture Technique** définit avec quels technologies vont être développés les composants, les sous-composants et les connecteurs.
L'**Architecture Physique** définit sur quels machines et matériels sont exécutes les composants et les sous-composants.

QUESTION NUMERO 3

a) Quel est le rôle du Design Pattern de l'Injection de Dépendance ?
b) Décrire son fonctionnement (vous pouvez vous aider d'un diagramme mais cela n'est pas obligatoire).

a) Le rôle du DP « Injection de dépendance » est, pour un composant utilisateur (U) de déléguer, à un composant particulier (F), de réaliser la dépendance entre deux autres composants (A et B).

b) Le composant U demande à F de lui donner une instance de A. F crée cette instance A en décidant avec quel composant B, A sera lié. La liaison entre A et B est réalisé soit par le constructeur de A soit par un setteur de A.

Fin de la 1ère partie sans document

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

3. PROBLEME [50 points]

On se propose de réaliser la conception d'un Système d'Information qui permet à un organisme bancaire de valider des ordres de transactions boursières réalisés par leurs clients.

Nous ne traitons pas ici la façon dont les clients créent leurs ordres de transactions boursières (site internet). Les ordres de transaction boursières proviennent de l'extérieur et arrivent sur un serveur de la banque [COMPOSANT 1].
Le serveur gère le stocke de ces ordres sous forme d'objets. Il crée en mémoire autant d'objet qu'il existe d'ordre.
Il existe plusieurs types d'ordre (ex : ordre de vente et d'achat d'action, regroupement, différé, ….).

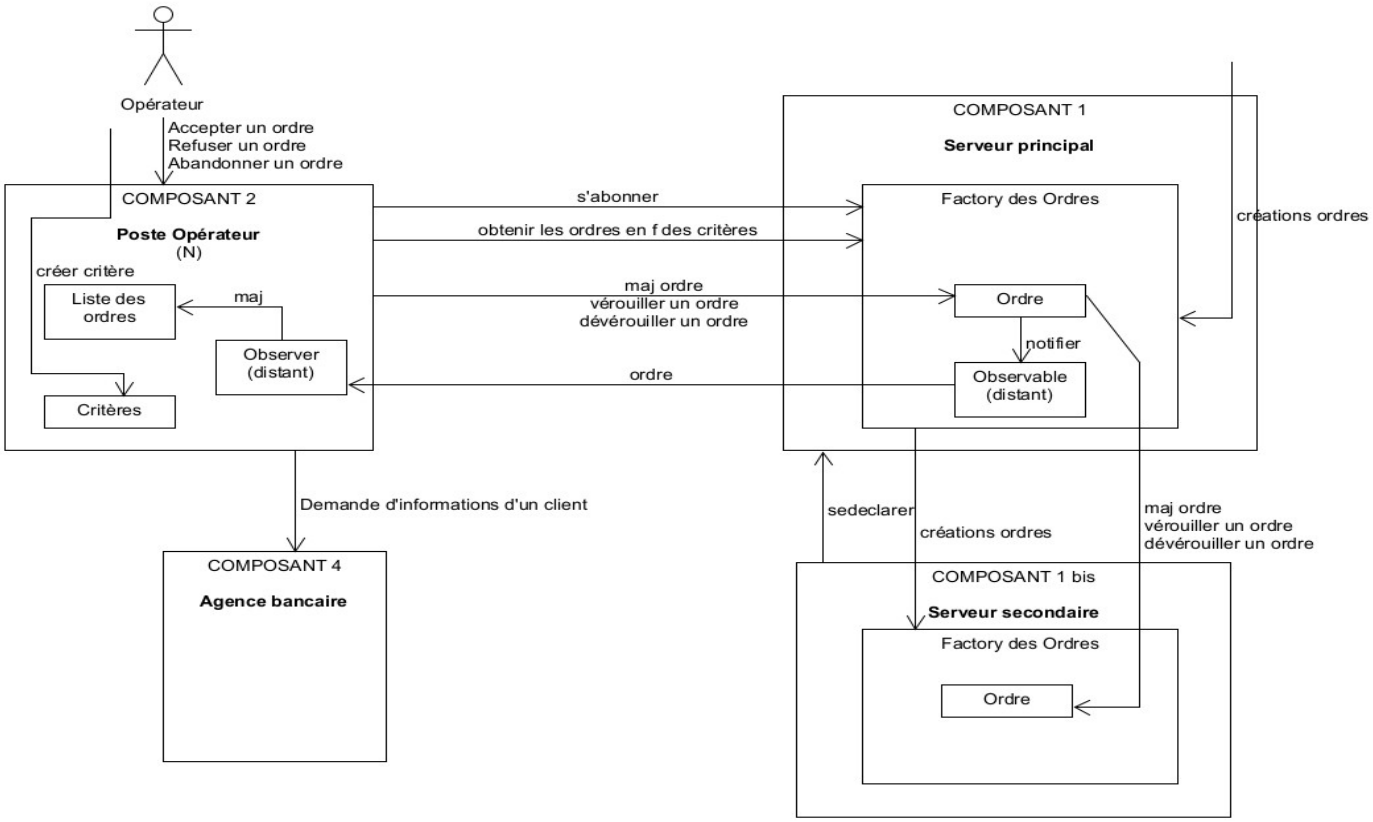
Des postes d'IHM [COMPOSANT 2] (nombre indéterminé) sont sur le réseau intranet (VPN) de la banque et communiquent en RMI avec le serveur.
Le rôle de ces postes est de visualiser des ordres en fonction de certains critères (intervalle de date, nombre, type, montant, donneur d'ordre, ...) afin de valider manuellement ces ordres.
La validation consiste à accepter ou refuser l'ordre. Dans le cas d'un refus, l'opérateur renseigne le motif du refus.
Toutes ces informations (acceptation, refus, motif) sont renseignées sur le serveur dans l'objet de l'ordre correspondant.
C'est l'opérateur qui demande explicitement de s'occuper de certains ordres en fonction de certains critères.
La liste de ces ordres est alors affichée et l'opérateur verrouille sur le serveur autant d'ordre qu'il souhaite (s'ils ne sont pas déjà verrouillés par un autre opérateur). Il a alors tout le temps pour analyser les ordres verrouillés par lui.
Pour faire cette analyse, l'opérateur peut consulter toutes les informations des clients qui sont gérées par chaque Agence Bancaire [COMPOSANT 4] répartis sur le territoire français.

Les ordres sont déverrouillés : par l'opérateur s'il annule ou abandonne son analyse; ou automatiquement lors de l'acceptation ou du refus.

Chaque liste des ordres de chaque IHM est rafraîchie en temps réel (arrivé d'un nouvel ordre, ordre modifié par un autre opérateur, ...).

Pour des raisons de sécurité, il existe un deuxième serveur, appelé "serveur redondant" [COMPOSANT 1 (bis)] qui est une duplication du serveur principal [COMPOSANT1]. A chaque création ou modification d'un ordre de transaction boursière par le serveur principal, automatiquement, la même création ou modification est réalisée sur le serveur secondaire.
Cela permet en cas de panne du serveur principal, de basculer, à chaud, sur le serveur secondaire, les IHM se reconnectant alors automatiquement sur le serveur secondaire.

1/ **[15 points]** **commentaire : 5 points diagramme 10 points**
Faites le diagramme de communication de ce Système d'Information. Il y a donc 4 composants logiciels à décrire qui communiquent entre eux de manière distant.
Commentez votre schéma (rôles des composants et sous-composants, comportement dynamique général, échanges des informations, localisation des données).
Nous rappelons que ce schéma doit permettre de connaître vos choix d'organisation des composants, le sens des communications, et les designs patterns envisagés.

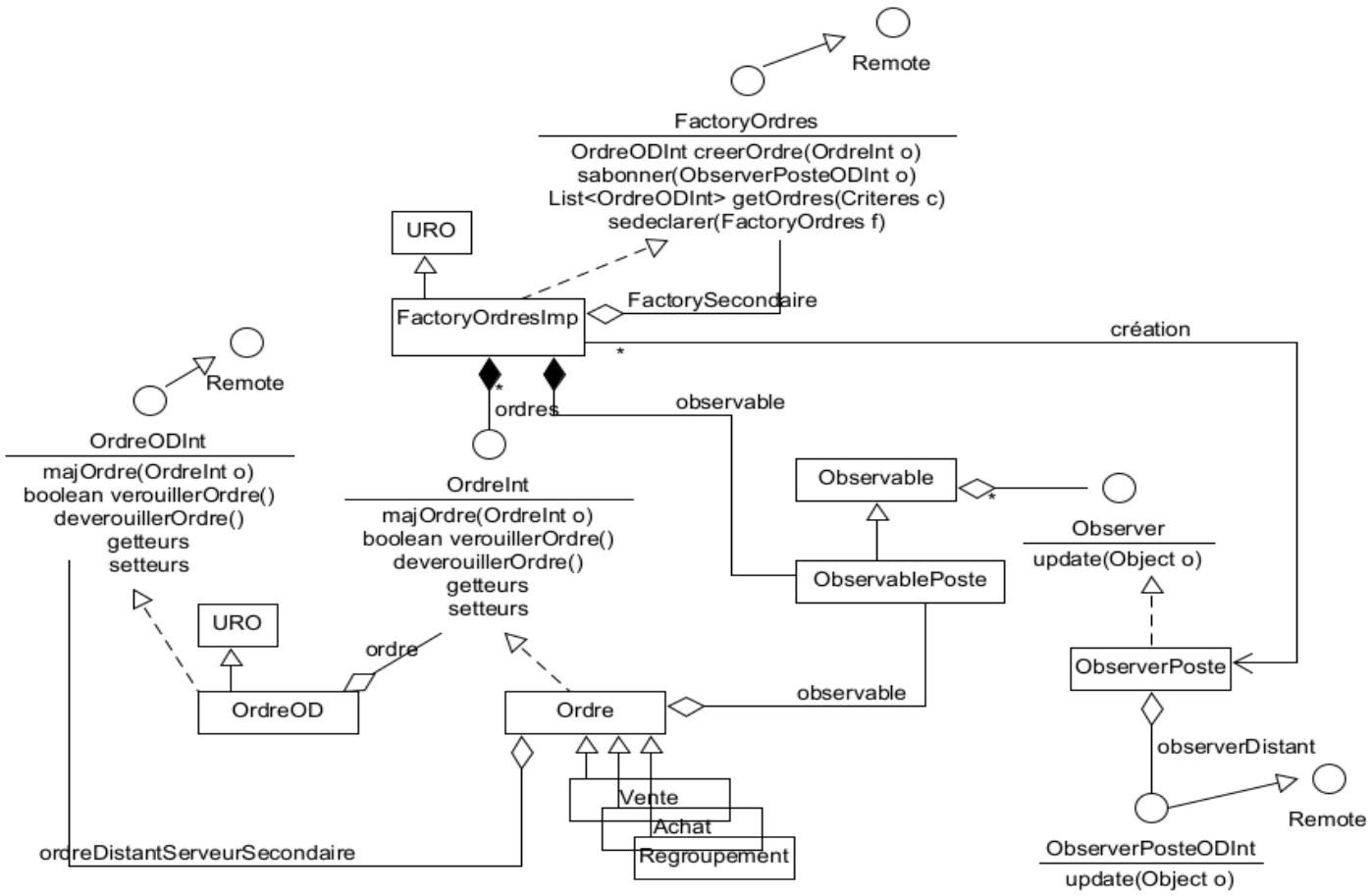


Tous les postes opérateurs s'abonnent au serveur principal afin de recevoir par notification les ordres crées ou modifiés sur le serveur principal afin de mettre à jour la liste des ordres affichés sur le poste.
Le serveur principal (et secondaire) contient un factory qui gère tous les ordres en mémoire.
Dès qu'un ordre est créé ou modifié sur le serveur principal, l'ordre est créé ou modifié sur le serveur secondaire qui est une duplication du serveur principal.
L'opérateur demande explicitement au factory des ordres du serveur principal, les ordres en fonction de ses critères qui sont créés sur chacun des postes opérateurs. En retour, le serveur renvoie les stubs de chacun des ordres car les ordres sont des objets distants. On a fait le choix que les ordres sont des objets distants qui restent sur le serveur.
Ainsi, suite à l'acceptation, le refus ou l'abandon d'un ordre, la maj de l'ordre est faite directement sur l'objet distant de l'ordre.
Le serveur secondaire se déclare au près du serveur principal. Ainsi, seul le serveur principal sera enregistré dans un annuaire.
De même, le verrouillage ou le déverrouillage d'un ordre se fait directement sur l'objet distant de l'ordre.
L'opérateur peut demander des informations aux différentes agence bancaires qui contient les informations de chacun des clients de l'agence. La référence du client contient l'identification de son agence.

Pour la duplication, une autre solution serait que le serveur secondaire soient notifié des ordres créés et modifiés, comme pour une IHM. Cette solution a été le choix de nombreux auditeurs.

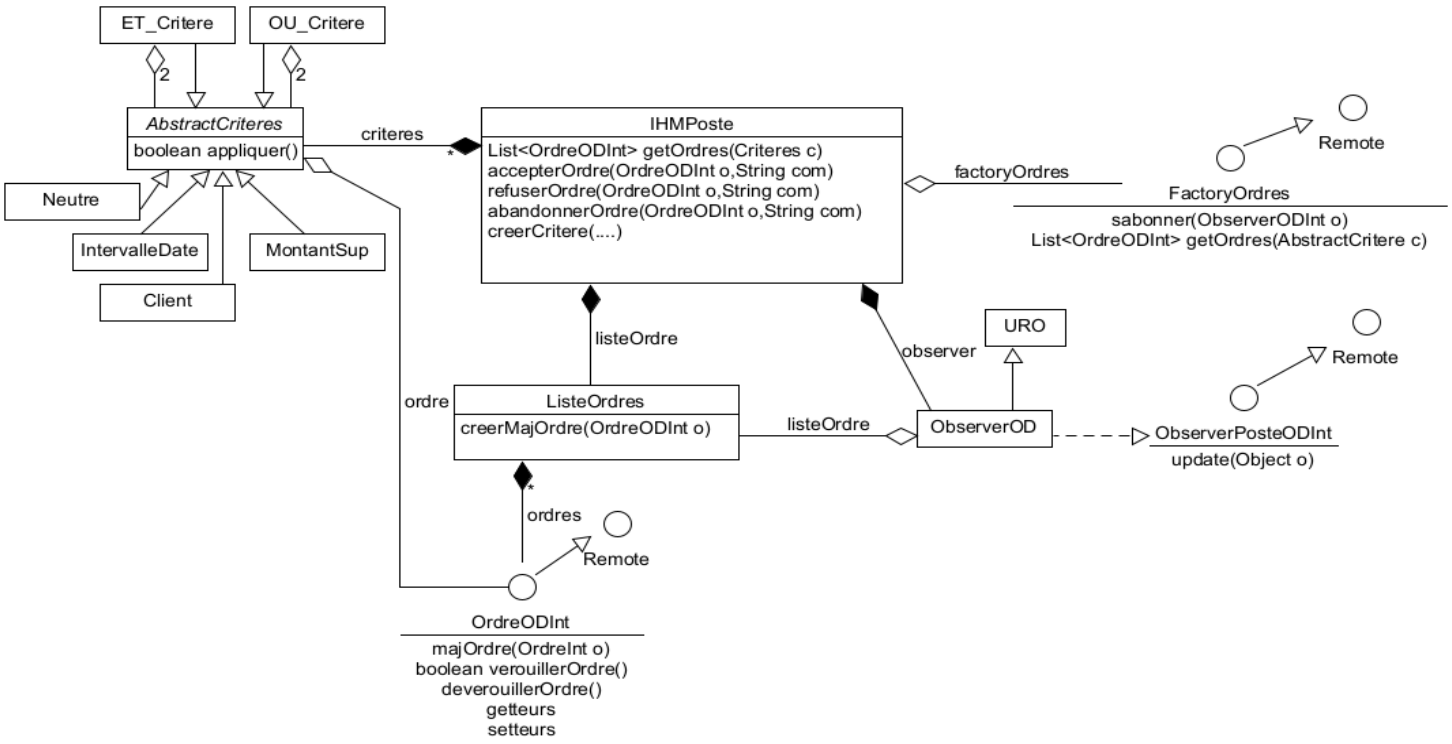
2/ **[35 points]**
Faites le(s) diagramme(s)e de classe UML des [COMPOSANT 1] et [COMPOSANT 2] en mettant en évidence les Designs Patterns utilisés.
Commentez chacun de(s) diagramme(s).

COMPOSANT 1 **commentaire : 8 points diagramme 15 points**



Le factory des ordres est un factory distant par héritage.
Il utilise un Observable pour notifier chacun des postes IHM qui envoie, lors de son abonnement, le stub de son interface distante. A chaque abonnement un ObserverPoste est créé qui s'abonne à l'observable et utilise l'interface distante ObserverPosteODInt pour notifier les ordres.
Pour mettre à jour le serveur secondaire, le factory utilise naturellement l'interface distante du factory secondaire pour la création des ordres (attribut factorySecondaire), et chaque Ordre utilise l'interface distante de l'ordre équivalente situé sur le serveur secondaire (attribut ordreDistantServeurSecondaire) pour la mise à jour d'un ordre. La méthode creerOrdre du factory principal retourne le stub de l'ordre créé sur le serveur secondaire.
Ordre est une classe abstraite dont hérite tous les types d'ordre.
Ordre est transformé en objet distant par l'adaptateur OrdreOD.

COMPOSANT 2 commentaire : 4 points diagramme 8 points



La classe IHMPoste contient toutes les méthodes correspondant aux actions de l'opérateur.
Elle utilise l'interface distante du factory principal (FactoryOrdres) pour s'abonner en lui donnant le stub de ObserverOD pour recevoir les notification, et pour demander les ordres en fonction d'un critère.
Elle gère une liste des critères créés par l'opérateur en créant et combinant les critères de base prédéfinis. La méthode appliquer de Critere est utilisé par le serveur principal pour déterminer les ordres à retourner.
Elle gère la liste des ordres reçus par le serveur principal. Ces ordres sont les stubs (OrdreODInt) des ordres qui restent sur le serveur principal.
Elle crée un observer distant (ObserverOD) afin de recevoir les ordres créés et mises à jour le serveur principal.

Précisions :
Un composant applicatif [COMPOSANT X] correspond à une JVM ou process. Cela signifie que les COMPOSANTS X communiquent sur le réseau à travers des interfaces distantes.
Ainsi, pour une description précise de vos diagrammes de classe, on fait le choix que toutes les communications distantes entre les composants sont réalisées en RMI (utilisation de la classe URO = UnicastRemoteObject et de l'interface Remote).

Fin du sujet