IPST-CNAM
Intranet et Designs patterns
NSY 102 – Cours du soir
27 Juin 2019

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

1ère Session .

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:

Dans la démarche d'architecture d'un SI, l' **Architecture Applicative** réunit :		Q 1.
1	la **Configuration Architecturale**, l' **Architecture Technique** et l' **Architecture Physique.**	
2	l' **Architecture Fonctionnelle** et l' **Architecture du Système.**	
3	la **Configuration Architecturale** et la **Dynamique de l' Architecture.**	**X**

A l'issu de la **Configuration Architecturale**, on obtient la description d'un graphe de description du Système d'Information dont les nœuds sont les Composants et les liens les Connecteurs entre ces composants.		Q 2.
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 3.
1	OUI	**X**
2	NON	

En RMI de Java, la classe d'appartenance d'un objet distribué (ou distant)		Q 4.
1	hérite de UnicastRemoteObject et implémente une interface qui décrit les méthodes distantes et qui hérite de l'interface prédéfinie Remote.	**X**
2	hérite de RemoteObject et implémente l'interface Remote	
3	n'hérite d'aucune classe particulière et doit être envoyé, par sérialisation; à l'annuaire RMI pour y être enregistré.	

Soit un objet quelconque Obj (instance de la classe A qui n'hérite pas d'une autre classe et qui n'implémente aucune interface). Pour transformer cet objet en un objet distant, il suffit de :		Q 5.
1	créer un proxy B de la classe A qui implémente les méthodes devant être appelées de manière distante	
2	créer un adaptateur de la classe A qui implémente les méthodes devant être appelées de manière distante	
3	créer une classe B qui est composé de A et qui implémente les méthodes devant être appelées de manière distante	**X**

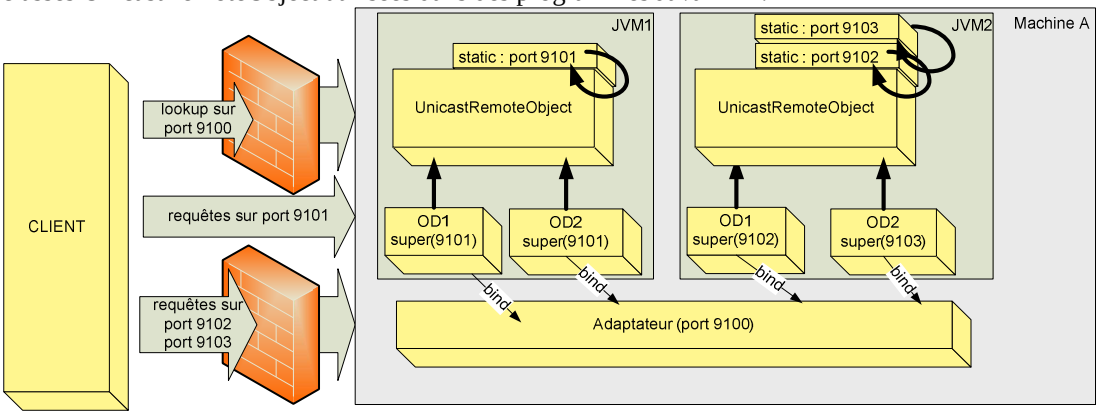
En RMI Java, l'amorce ou stub d'un Objet Distant est un proxy client de l'Objet Distant.		Q 6.
1	OUI	**X**
2	NON	

Un Design Pattern définit des principes de conception, et non des implémentations spécifiques de ces principes.		Q 7.
1	OUI	**X**
2	NON	

Dans la conception UML d'une application informatique, le "Canevas" est un Design Pattern particulier qui répond à la problématique de création d'objets graphiques génériques qui doivent dessiner dans un dessin (ou canvas).		Q 8.
1	OUI	
2	NON	**X**

Le rôle du Design Pattern **Singleton** est la création unique d'une instance d'une classe.		Q 9.
1	OUI	**X**
2	NON	

Soit le schéma suivant qui représente un fonctionnement possible de plusieurs serveurs de socket des classes UnicastRemoteObject utilisées dans des programmes Java RMI.

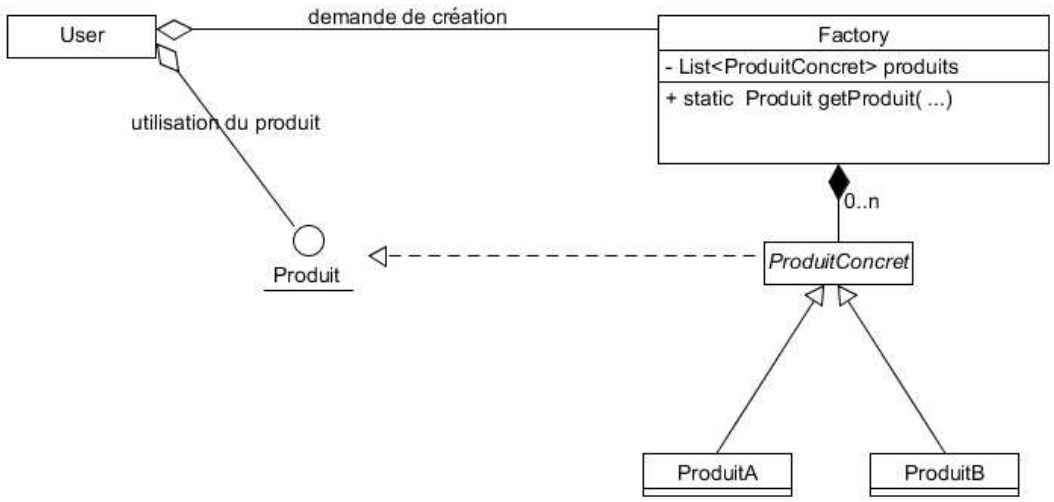


		Q 10.
1	On peut créer un nouvel OD dans la JVM1 qui s'exécute sur le port 9102	
2	Sur la machine A, on peut créer une nouvelle JVM3 dans laquelle, on crée un nouvel OD qui s'exécute sur le port 9104	**X**
3	Dans la JVM2, on peut créer un nouvel objet distribué RMI sur le port 9102	**X**

Le rôle du DP Singleton est de :		Q 11.
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**

Le rôle du DP Factory qui est un Singleton est de rendre global à tout le programme, la création et l'utilisation de certains objets.		Q 12.
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.		Q 13.
1	OUI	
2	NON	**X**



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.		Q 14.
1	OUI	X
1	NON	

Le rôle du DP "Délégation" est de déléguer à une autre classe la réalisation des traitements qu'une classe aurait dû implémenter.		Q 15.
1	OUI	X
2	NON	

Le rôle du Design Pattern **Observateur** est de créer, dynamiquement des objets dont la classe d'appartenance implémente l'interface **Observer.**		Q 16.
1	OUI	
2	NON	X

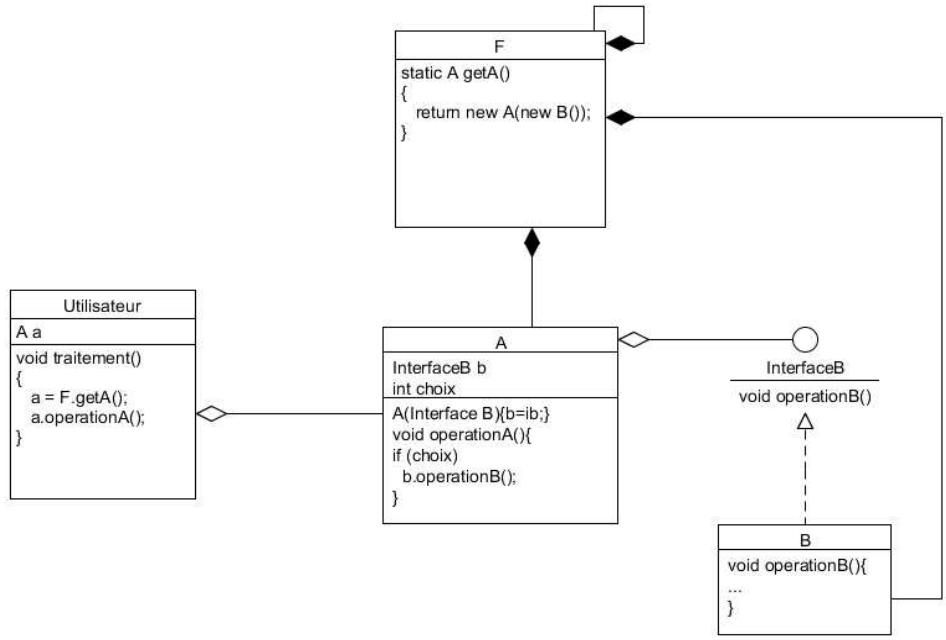
Le ClientProxy est un DP Proxy utilisé par un client, dans lequel les méthodes réelles ont été remplacées par un appel distant à ces méthodes.		Q 17.
1	OUI	X
2	NON	

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 18.
1	OUI	X
2	NON	

Dans une architecture MOM, le mode "Queue" assure que tous les consommateurs connectés au canal d'évènement de la queue d'évènement, reçoivent l'évènement publié par le Producteur.		Q 19.
1	OUI	
2	NON	X

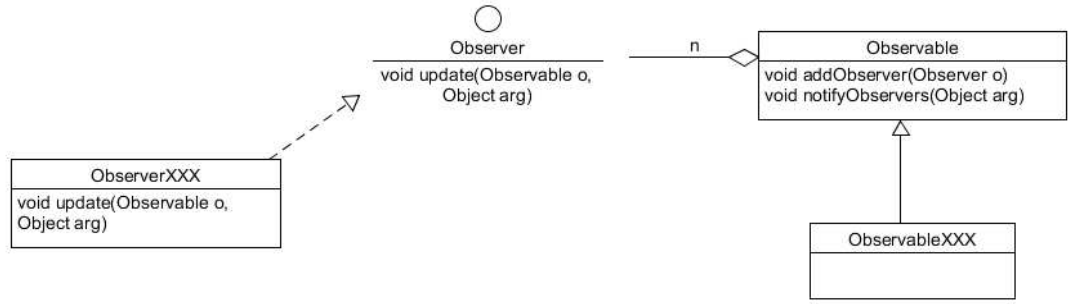
La communication synchrone entre un producteur et un consommateur par un "Canal d'évènement" se fait :		Q 20.
1	via le modèle du "invoke" en passant par un intermédiaire	
2	via le modèle du "Push" en passant par un intermédiaire	X
3	via le modèle du "Pull" en passant par un intermédiaire	

Le diagramme suivant :		Q 21.



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	

Soit le Design Pattern Observateur :		Q 22.

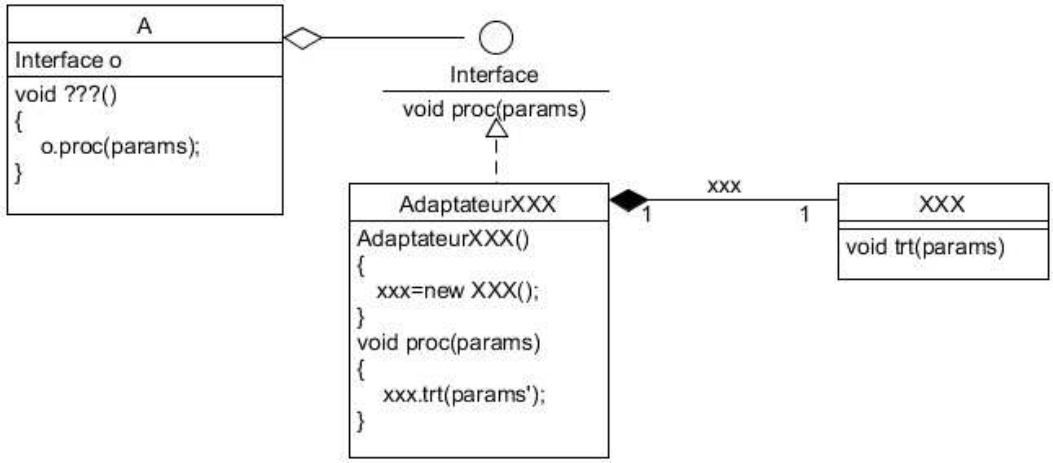


Ce DP est		
1	de type "pull"	
2	de type "push synchrone"	**X**
3	de type "push asynchrone"	

Le DP proxy est un diagramme de classe dans lequel :		Q 23.
1	deux classes A et B héritent d'une même classe C non abstraite.	
2	deux classes A et B implémentent la même interface, et A a un lien d'agrégation avec B	**X**

Comme dans l'Injection de Dépendance, le DP Stratégie permet d'injecter dynamiquement un traitement générique dans un utilisateur.		Q 24.
1	OUI	**X**
2	NON	

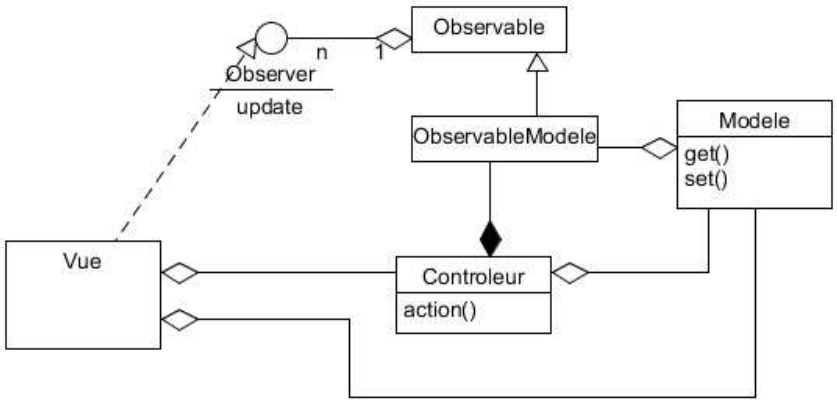
Soit le diagramme de classe du DP Adaptateur suivant : Q 25.



Dans ce diagramme, le rôle AdaptateurXXX est de :

1	servir de proxy entre la classe A et la classe XXX	
2	rendre compatible la classe XXX pour l'appel de la méthode proc utilisé dans la classe A	**X**
3	déléguer à la classe XXX le traitement **trt**	

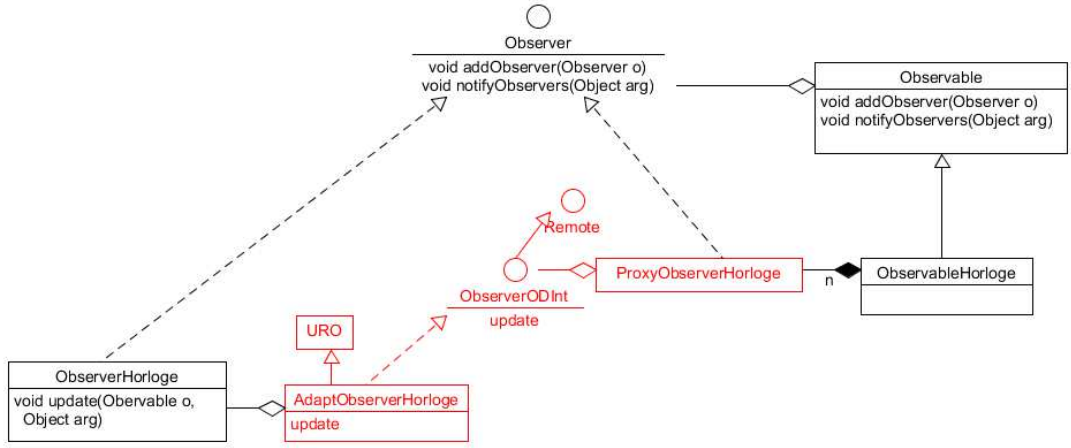
Soit le diagramme de classe suivant : Q 26.



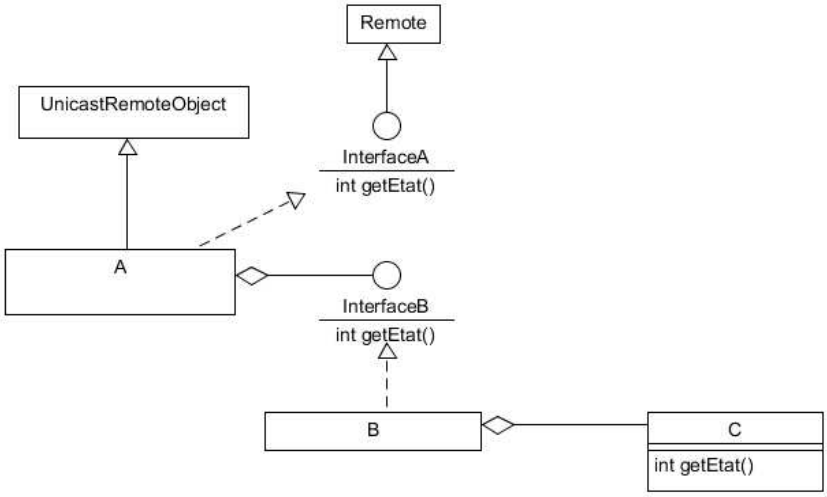
Ce diagramme représente le Design Pattern Model-Vue-Contrôleur et il est correct.

1	OUI	**X**
2	NON	

Les descriptions suivantes sont des modèles de communication asynchrone :		Q 27.
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 transmettre l'évènement	

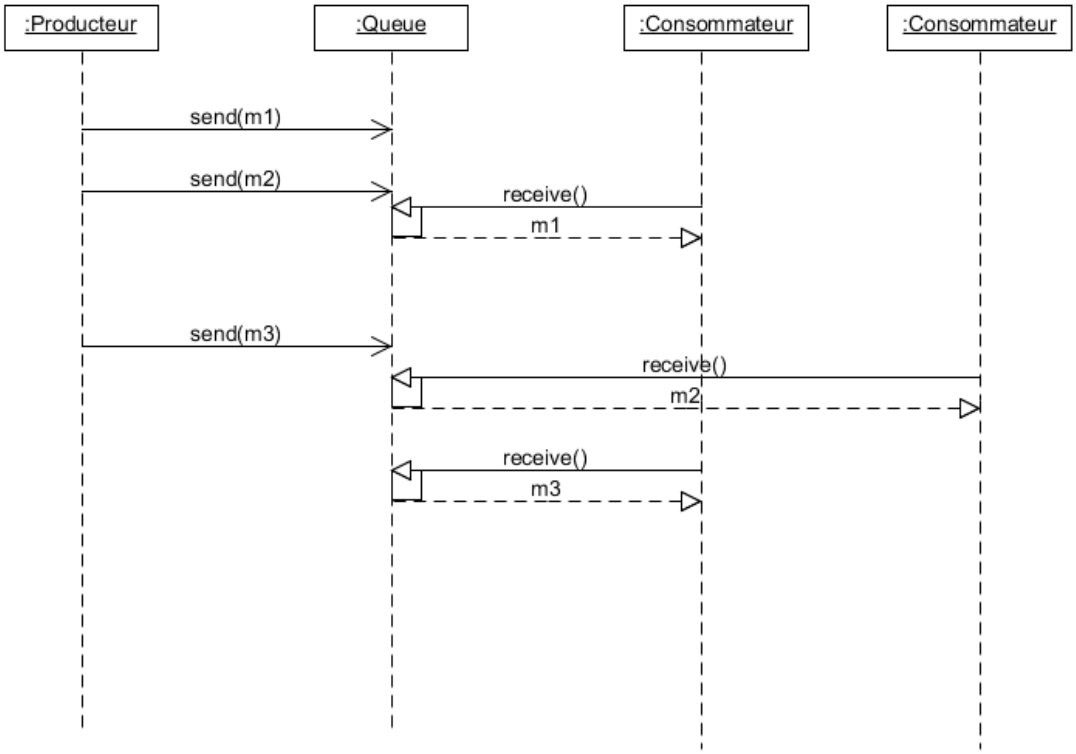


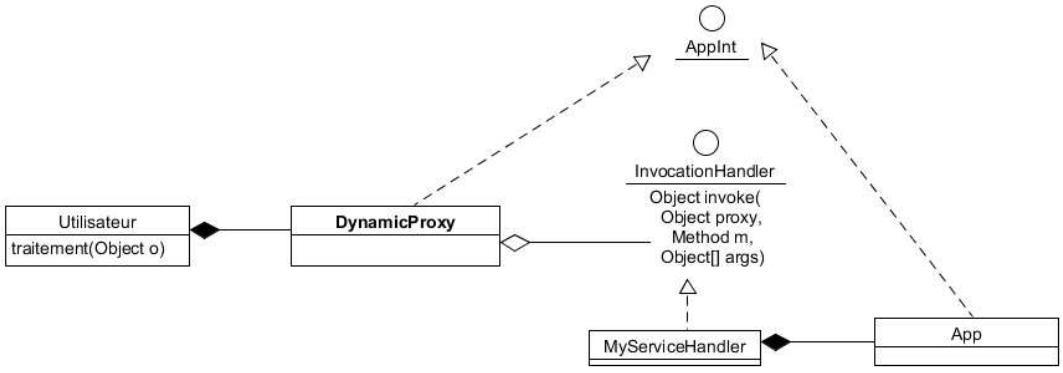
Ce diagramme de classe est celui du Design Pattern "Observateur Distant"		Q 28.
1	OUI	**X**
2	NON	



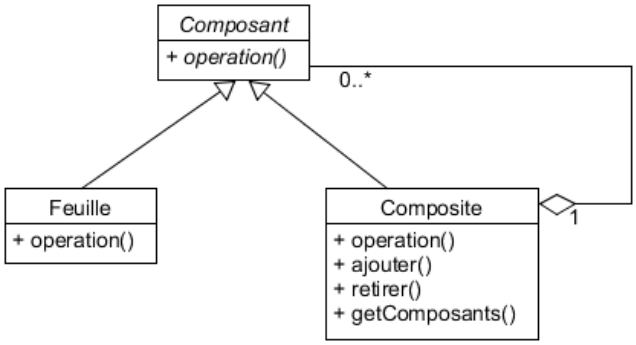
Ce diagramme de classe est la conception d'un Objet Distant suivant le modèle :		Q 29.
1	d'un DP Proxy	
2	d'un DP Adaptateur	**X**

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 30.
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 ModelVueControler pour produire les évènements des Producteurs	

En JMS (Java Messaging System), il existe (notamment) deux modes de communication : Queue et Topic.  Ce diagramme de transition correspond au mode Queue		Q 31.
1	Ici, le producteur envoie les messages de manière synchrone aux consommateurs	
2	Ici, le producteur envoie les messages de manière asynchrone aux consommateurs	X
3	Ici, chaque consommateur consomme les messages de manière asynchrone	X

 Ce schéma est celui du DP Dynamic proxy. Le rôle de la classe MyServiceHandler est d'implémenter toutes les méthodes de l'interface AppInt.		Q 32.
1	OUI	
2	NON	X

Le DP DynamicProxy est utilisé en RMI pour créer dynamiquement le Proxy client utilisé dans le stub d'un objet distant pour communiquer avec l'objet distant.		Q 33.
1	OUI	X
2	NON	

Soit le DP Composite décrit comme suit :		Q 34.
		
Le rôle de ce DP est de décrire la conception d'objets qui sont composés d'objets similaires.		
1	OUI	**X**
2	NON	

L' "inversion de contrôle" est un principe de conception qui:.		Q 35.
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 l'application logicielle.	**X**

Fin du QCM

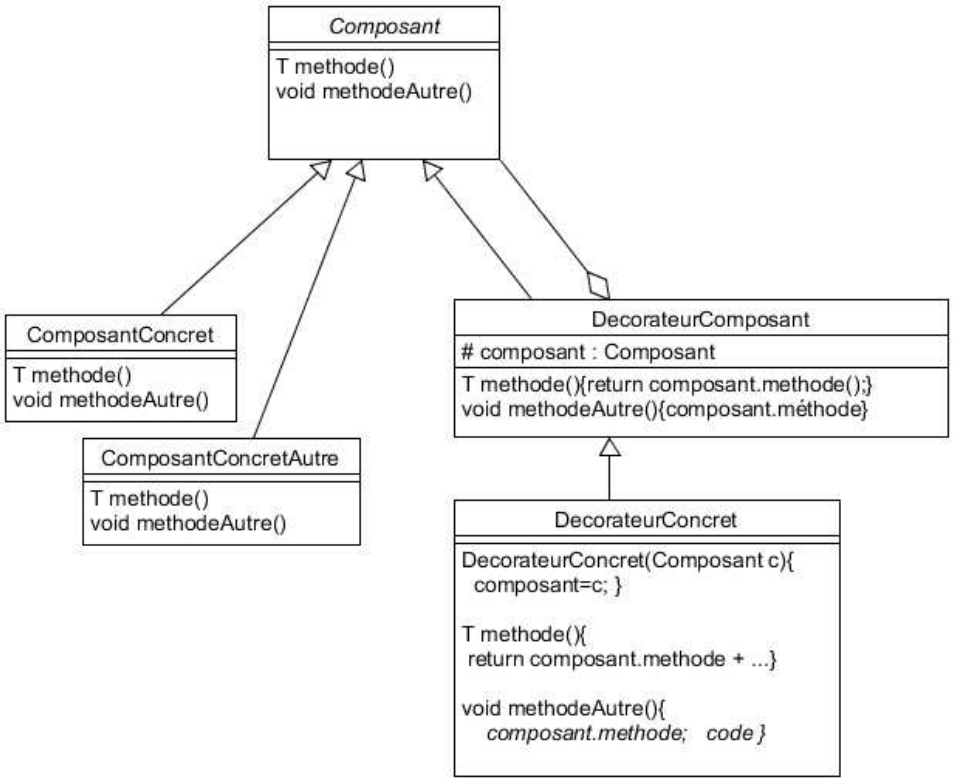
Suite (Tournez la page)

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

Faites le diagramme de classe du design pattern "**Decorateur**".
Expliquez le comportement de ce design pattern.
Correction :



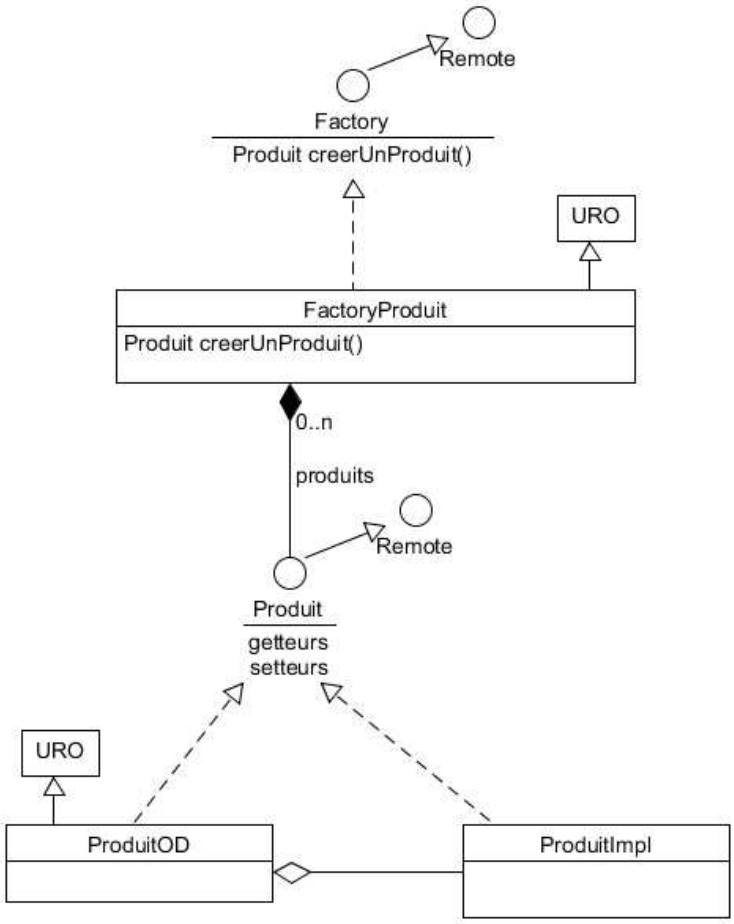
Le rôle de ce DP est de surcharger (étendre ou remplacer) les méthodes d'une classe **Composant** sans utiliser le mécanisme de surcharge (ou redéfinition) de l'héritage qui a ses limites.
Un principe fort est qu'il est possible d'empiler plusieurs décorateurs les uns sur les autres : le décorateur est de même type que la classe qu'il décore et retourne l'objet surchargé.

QUESTION NUMERO 2

Expliquez le principe du DP DynamicProxy. Vous pouvez aider votre explication par un diagramme (non obligatoire).
Correction :
Soit un Proxy A de la classe B, A et B implémente toutes les deux l'interface I.
Le principe du DynamicProxy est de pouvoir créer le proxy, instance de la classe A, dynamiquement (pendant l'exécution) dont la classe A (créée dynamiquement) implémente toutes les méthodes de l'interface I. Cette implémentation consiste à appeler pour chaque méthode de l'interface I une méthode unique (invoke) qui doit être implémentée par une classe Z (qui implémente l'interface InvocationHandler) dont l'instance est passée en paramètre de la méthode de création du Proxy.
La classe B est une agrégation ou une composition de la classe Z.

QUESTION NUMERO 3

Soit un Factory implémenté dans un serveur qui crée des produits à la demande d'un client. Ces produits restent locaux au serveur et sont utilisés de manière distante par le client qui les a créés.
Faites le diagramme de classe de ce Factory.
Correction :



Fin de la 1ère partie sans document

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

3. PROBLEME [50 points]

Nous envisageons de réaliser un Système d'Information (SI) qui permet de jouer en réseau à un jeu de carte de table (comme par exemple le poker, la belote, le bridge, …). Chacun des joueurs qui veulent faire une partie ensemble, rejoint une même Table de Jeu. Le SI permet de gérer plusieurs Tables de jeu en même temps.
Le SI ne sait jouer qu'à un seul jeu en particulier. Il n'est pas demandé que le SI sache s'adapter dynamiquement à plusieurs types de jeu de carte mais votre conception doit permettre de faciliter la programmation à différents types de jeu de carte.

Pour jouer, chacun des joueurs exécute un programme client [COMPOSANT 1] qui prend en entrée le nom du joueur (on considère ici que le nom est unique pour chaque joueur). Ce composant se connecte à un serveur [COMPOSANT 2] qui va prendre en charge la communication avec tous les joueurs de la table de jeu et gérer le fonctionnement du jeu.

Quand un joueur exécute le [COMPOSANT 1], une fenêtre principale apparait permettant de faire 2 choix :

- créer une Table de Jeu en saisissant le nom de la table de jeu et un mot clef d'accès. Puis retour à la fenêtre principale ;
- rejoindre une Table de Jeu en sélectionnant le nom d'une Table de Jeu parmi la liste de toutes les Tables de Jeu gérées par le serveur, et en saisissant le mot clef d'accès (on considère ici que les joueurs ont pris connaissance de ce mot clef par un moyen extérieur qui n'est pas traité ici). Ensuite, le [COMPOSANT 1] affiche la Table de Jeu.

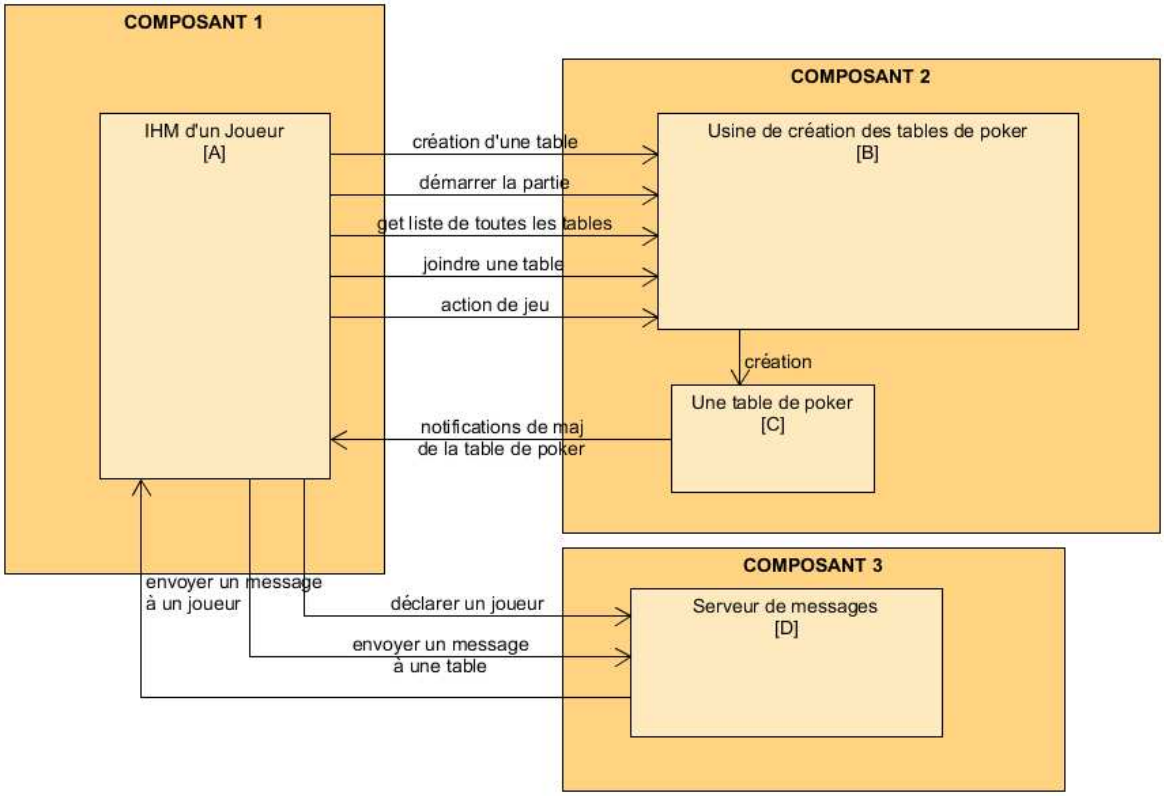
Seul le joueur qui a créé la Table de Jeu peut démarrer la partie.
Une fois la partie démarrée, chaque joueur peut réaliser ses actions de jeu. Le [COMPOSANT 2] qui gère le déroulement d'une partie, met à jour l'état globale de la partie, détermine qui doit jouer et notifie tous les joueurs afin de rendre actif ou inactif les actions de chacun des joueurs en conséquence, et met à jour l'affichage de la Table de Jeu de chacun des joueurs.

Dans le [COMPOSANT 1], une zone verticale affichée à côté de la Table de Jeu permet à un joueur d'envoyer un message à tous les joueurs de la même Table de Jeu. Cette zone affiche chronologiquement tous les messages envoyés et reçus préfixés par le nom du joueur. C'est un troisième composant [COMPOSANT 3] qui permet de gérer la communication de ces messages entre tous les joueurs.

1/ **[15 points]**
Faites le diagramme de communication (ou comportement) de ce Système d'Information. [10 points]
Commentez votre schéma [5 points] (rôles des composants et sous-composants, comportement dynamique général, échanges des informations, localisation des données).
Nous rappelons que ce schéma doit permettre de connaitre vos choix d'organisation des sous-composants de chacun des COMPOSANTS de ce SI.

Correction :

Le diagramme de communication est le suivant :



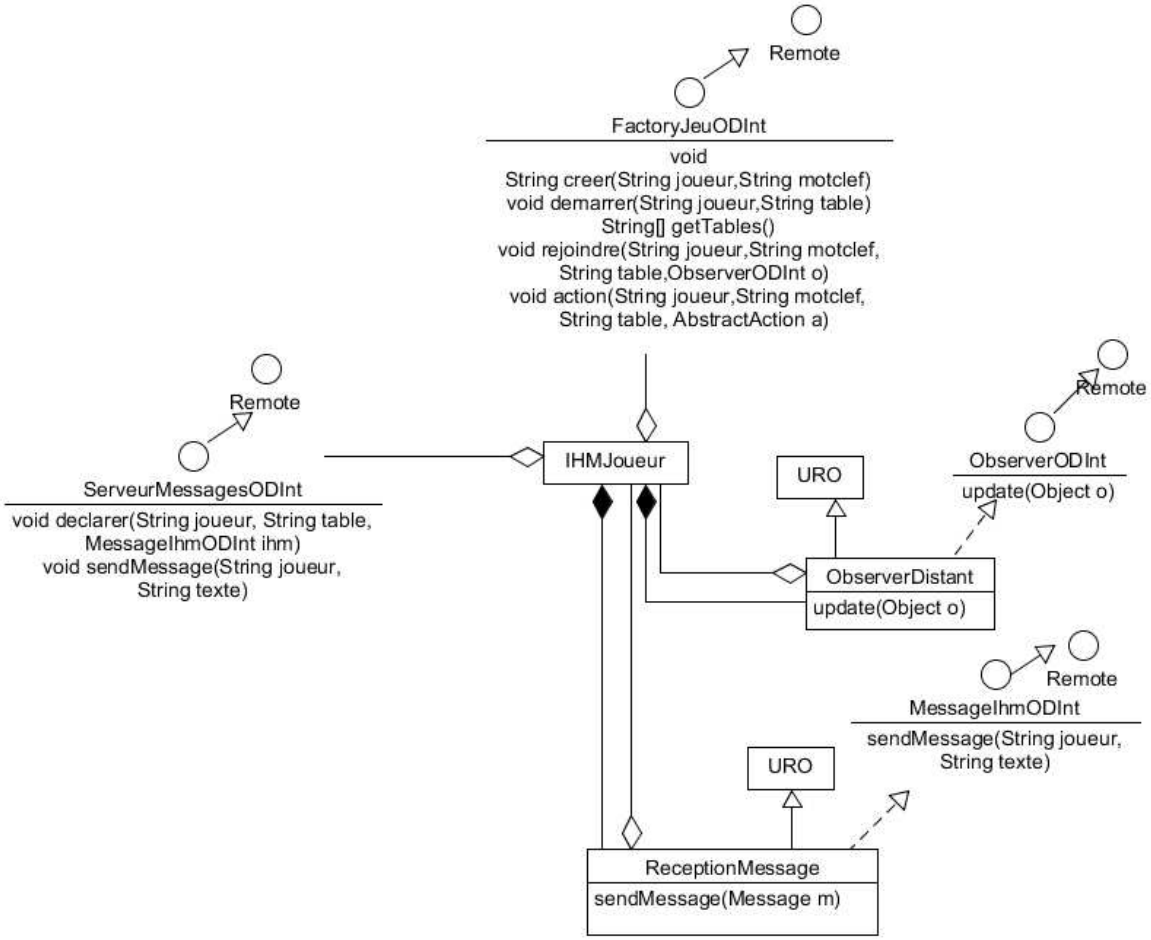
Chaque IHM d'un joueur [A] utilise à distance une usine [B] de création de tables de poker [C], mais aussi pour démarrer une partie, connaitre la liste de toutes les tables, et joindre une table de poker.
Le joueur de [A] envoie son action de jeu à [B] avec l'action, son nom et le nom de sa table. [B] donne cette action à la bonne table [C] qui à son tour va notifier toutes les IHM des joueurs de la même table du résultat.

Pour gérer les messages envoyés entre les joueurs, l'ihm [A] se déclare au serveur de messages [D]. Ainsi quand [A] envoie un message à [D], [D] peut relayer ce message à tous les autres joueurs de la même table.

2/ **[35 points]**
Faites le(s) diagramme(s)e de classe UML des [COMPOSANT 1], [COMPOSANT 2] et [COMPOSANT 3] en mettant en évidence les Designs Patterns utilisés.
Commentez chacun de(s) diagramme(s).

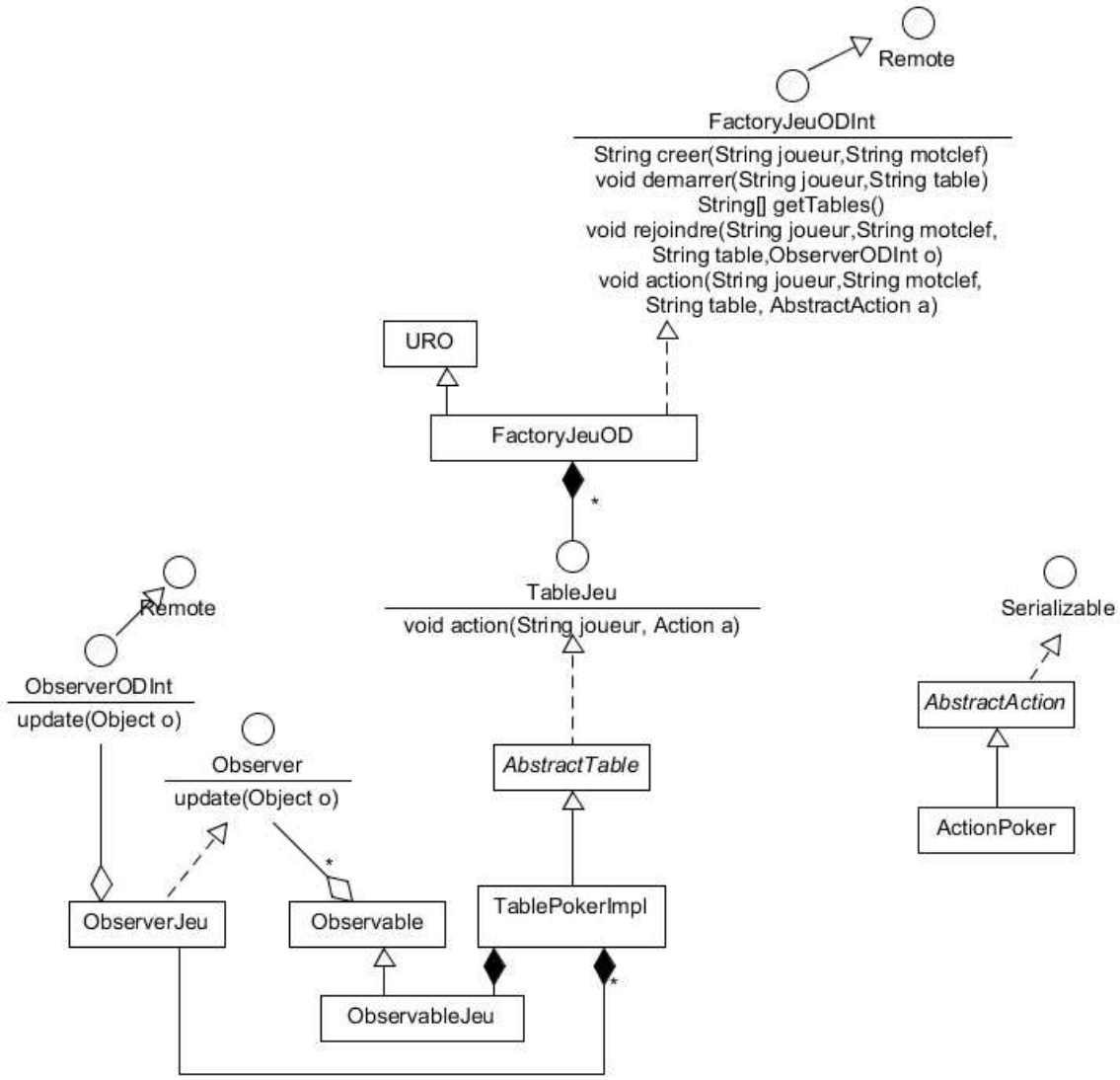
COMPOSANT 1	schéma [10 points]	commentaire [4 points]
COMPOSANT 2	schéma [10 points]	commentaire [4 points]
COMPOSANT 3	schéma [5 points]	commentaire [2 points]

COMPOSANT 1 :



Le [COMPOSANT 1] est constitué de la classe IHMJoueur qui gère la page principal et l'affichage de la table de Poker. Elle utilise l' interface distante du Factory distant, interface FactoryPokerODInt, du serveur de poker pour créer une table, démarrer une partie, connaitre les tables, joindre une table, faire une action de jeu.
La classe IHMJoueur crée un observer distant, classe ObserverDistant, (DP Observateur/Observer Distant) dont l'interface distante est envoyée au serveur de poker quand un joueur rejoint une table afin de pouvoir être notifié de l'évolution de la table de poker. Celui qui crée la table de poker doit également rejoindre la table de poker.
La classe IHMJoueur crée un objet distant ReceptionMessage, afin de recevoir les messages du serveur de message.

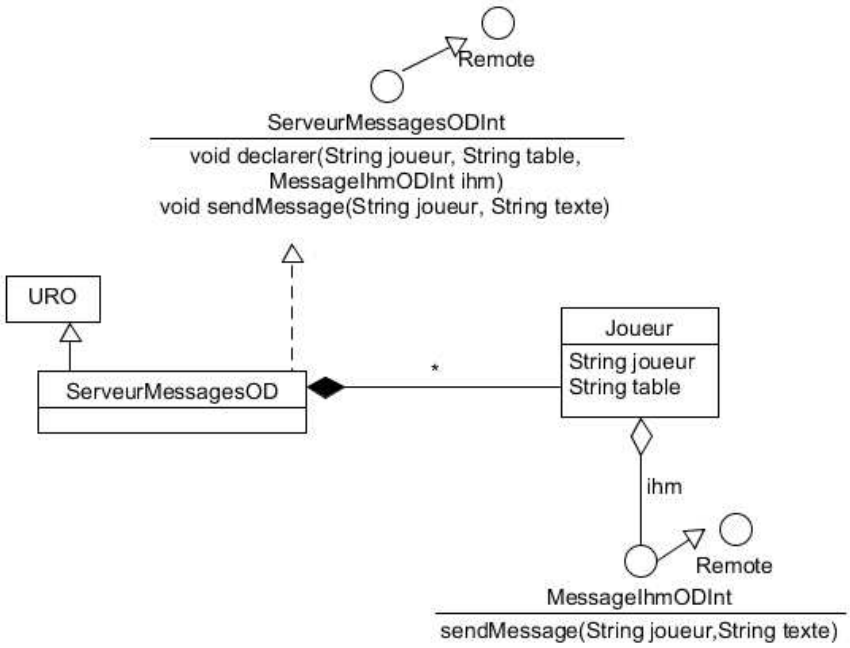
COMPOSANT 2 :



Le [COMPOSANT 2] crée un factory distant (DP Factory), classe FactoryJeuOD (DP OD par héritage) afin de créer à la demande des tables de poker, classe TablePokerImpl qui hérite d'une classe abstraite AbstractTable. Chaque table créée, crée un observable, classe ObservableJeu afin de notifier toutes les ihm des joueurs de la table de poker à travers un observer local ObserverJeu qui utilise l'observer distant ObserverODInt (DP Observateur/observable Distant).

Les classes abstraites AbstractAction et AbstractTable permettent de pouvoir étendre le SI pour la programmation d'autres types de jeu.

COMPOSANT 3:



La classe ServeurMessageOD est un Objet Distant (OD par héritage) dont l'interface distante ServeurMessagesODInt permet à un joueur de se déclarer et d'envoyer un message au serveur de messages. La classe gère la table de Joueur qui permet de savoir à quelle table appartient chaque joueur,et à quelle interface distante correspond chaque joueur. Ainsi, la classe ServeurMessagesOD détermine à quels joueurs, il faut envoyer le message reçu.

Cette interface distante permet à un joueur d'envoyer un message au serveur de messages.

Précisions :
Un composant logiciel [COMPOSANT X] correspond à une JVM ou process. Cela signifie que les COMPOSANTS X communique 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