IPST-CNAM
Intranet et Designs patterns
NSY 102
Mercredi 7 Mai 2014

Durée : **2 h 30**
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 sert d'interface de communication entre un client et un serveur		Q 1.
1	OUI	X
2	NON	

Une application dite "distribuée" est une application logicielle dans lequel les données informatiques sont		Q 2.
1	centralisées dans un singleton crée dans un programme	
2	réparties sur le réseau et accessibles par tout logiciel qui utiliserait un ORB	X
3	toutes créées dans un Factory unique	

L'IDL (Interface Definition Language) est un langage informatique utilisé par les ORB pour écrire le code ~~des POA et~~ des servants.		Q 3.
1	OUI	X
2	NON	X

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 4.
1	OUI	X
2	NON	

On appelle un objet "distribué" tout objet qui est sérialisé et qui circule sur le réseau		Q 5.
1	OUI	
2	NON	X

Pour qu'un client puisse accéder à un objet distribué distant (OD1), il est indispensable que cet OD soit enregistré dans un annuaire afin que le client puisse se connecter à l'OD.		Q 6.
1	OUI	
2	NON	X

Soit 2 clients (A et B) qui appellent la méthode distante m1 de l'objet distribué OD1.		Q 7.
1	A et B peuvent appeler en même temps la méthode m1 de OD1 à condition que la méthode m1 soit "synchronized".	
2	A et B peuvent appeler en même temps la méthode m1 de OD1.	X

	Cette représentation de modèle de classes en UML est celle vu en cours permettant une communication RMI entre une IHM (IhmXXX) et son Applicatif (AppXXX). IhmXXXRmiImp est un DP Proxy de AppXXX :	Q 8.
1	OUI	X
2	NON	

En RMI de Java, les paramètres des méthodes distantes peuvent être :		Q 9.
1	des objets de n'importe quelle classe	
2	des objets dont la classe d'appartenance est une classe qui implémente l'interface Serializable	X
3	des éléments de type primitif (int, char, long, byte, ...)	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 10.
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 RMI Java, l'amorce ou stub d'un Objet Distribué est un proxy de l'Objet Distribué.		Q 11.
1	OUI	X
2	NON	

Soit le code suivant d'implémentation d'un singleton : — Q 12.

```
public class SingletonXXX
{
    private SingletonXXX () { }

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

Ce code est correct.

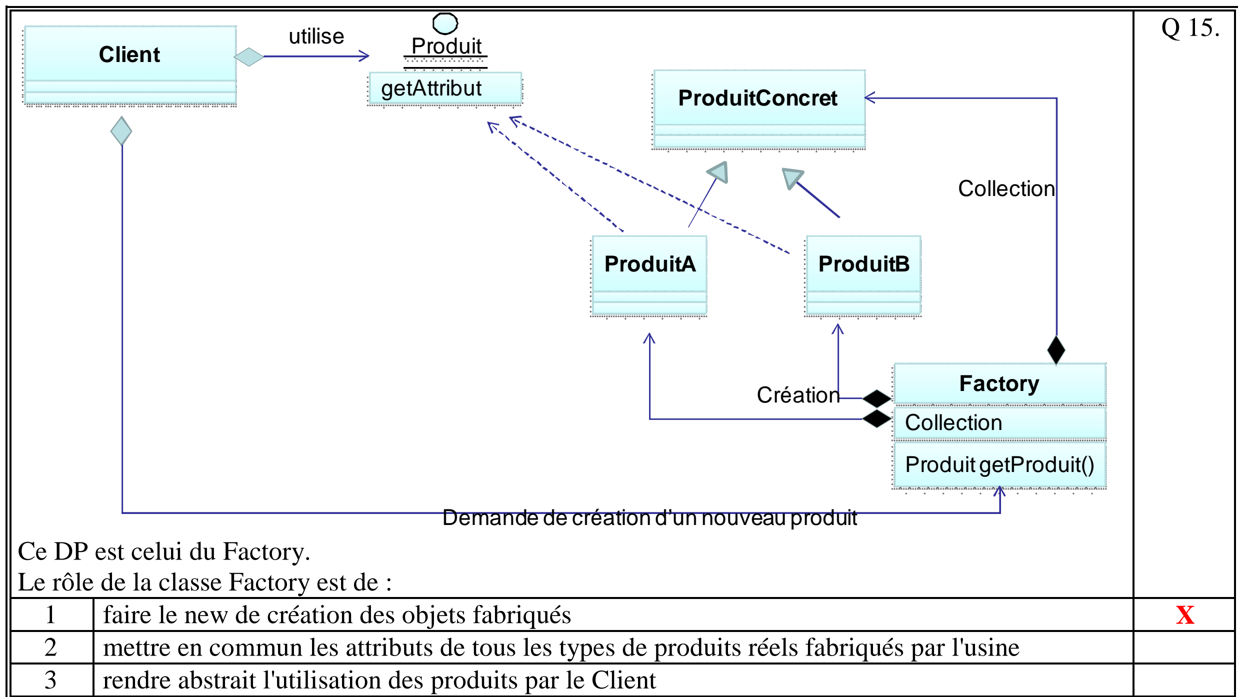
1	OUI	
2	NON	X

Dans un système réparti, le DP Singleton est utilisé pour créer un objet distribué unique sur le réseau		Q 13.
1	OUI	
2	NON	X

Q 14.

Ce DP est celui du Factory.
La signification des lettres A, B, C et D est :

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



Ce DP est celui du Factory. Le rôle de la classe Factory est de :		Q 15.
1	faire le new de création des objets fabriqués	X
2	mettre en commun les attributs de tous les types de produits réels fabriqués par l'usine	
3	rendre abstrait l'utilisation des produits par le Client	

Dans le DP Observeur, la communication entre l'Observer (consommateur d'évènement) et l'Observable (producteur d'évènement) est nécessairement asynchrone car la communication se fait toujours par l'envoi d'un message sans valeur de retour.		Q 16.
1	OUI	
2	NON	X

Dans le DP Observeur, plusieurs Observer (consommateurs d'évènement) peuvent être connectés au même Observable (producteur d'évènement)		Q 17.
1	OUI	X
2	NON	

Dans le DP Observeur, un DP Proxy peut être utilisé car :		Q 18.
1	il sert d'adaptateur entre l'observer et l'observable	
2	il permet que tous les changements d'état nécessitant une notification soient formalisés dans une interface	X
3	il assure que la mécanique Observer/Observable est implémentée dans le proxy et ainsi le codage de l'objet cible devient indépendant de cette mécanique.	

Il existe des Factory qui créent des objets concrets de différentes natures dont les classes d'appartenance héritent d'une classe abstraite et implémentent la même interface.		Q 19.
1	OUI	X
2	NON	

Soit le schéma suivant :		Q 20.
ObserverInt: update; ObserverODInt: update; Observer: update; ObserverHorloge: update; ObserverHorlogeOD: update; Stub: update; AdaptObserver HorlogeStub: update; ObservableHorloge: addObserver		
Les classes et interfaces en jaune représentent :		
1	un proxy client de communication entre ObservableHorloge et ObserverHorloge	
2	un pont de communication permettant à un Observable (ObservableHorloge) de notifier les évènements à un Observer (ObserverHorloge) se trouvant dans une autre JVM.	X

Un canal d'évènement est constitué de deux DP : un DP Factory permettant de créer des évènements et un DP Iterator permettant de parcourir ces évènements.		Q 21.
1	OUI	
2	NON	X

Le modèle de communication "Push asynchrone" est un DP dans lequel la classe qui implémente l'interface Observer implémente aussi l'interface Runnable afin de créer un thread qui réalise la notification.		Q 22.
1	OUI	
2	NON	X

La communication synchrone entre un producteur et un consommateur par "Canal d'évènement" se fait :		Q 23.
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	X

Dans le DP MVC, les Vues utilisent le DP Observer/Observable pour :		Q 24.
1	tirer les évènements du Modèle	
2	recevoir les notifications de changement des états du Modèle.	X
3	envoyer les commandes opérateurs au Contrôleur	

Le DynamicProxy est un DP qui hérite de ClassLoader et dont l'objectif est de permettre à une JVM de charger dynamiquement les classes à travers un proxy qui est passé en paramètre de la JVM		Q 25.
1	OUI	
2	NON	X

Un Proxy est un DP dans lequel deux classes A et B implémentent la même interface, et A utilise B		Q 26.
1	OUI	X
2	NON	

L'adaptateur et l'adapté implémente la même interface		Q 27.
1	OUI	
2	NON	X

Un Adaptateur est un DP constitué d'une classe A qui implémente une interface I à la place d'une autre classe B qui ne peut pas implémenter cette interface		Q 28.
1	OUI	X
2	NON	

Le DP Observateur est constitué d'une classe (Observable) et d'une interface (Observer).		Q 29.
1	Une fonction de la classe Observable est de stocker des objets qui implémentent l'interface Observer.	X
2	L'interface Observer contient une méthode (par exemple update) qui est appelée par l'Observable	X
3	Une fonction de la classe Observable est de servir d'adaptateur à tout modèle qui ne peut pas implémenter l'interface Observer	

Le DP Builder est un DP utilisé dans celui du Factory afin de construire le produit par assemblage d'autres classes		Q 30.
1	OUI	X
2	NON	

Le rôle d'un factory est, entre autre, de :		Q 31.
1	de créer à la demande de nouveaux objets	X
2	de créer à la demande des singletons qui sont ainsi utilisées dans toutes les classes du Factory	

Le modèle de communication "Pull synchrone" est réalisé avec le DP Observateur (Observer/Observable)		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 distribué pour communiquer avec l'objet distribué		Q 33.
1	OUI	X
2	NON	

Le DP Observateur est un modèle de communication synchrone suivant le principe de :		Q 34.
1	pull	
2	push	X

Une classe d'Adaptateur et une classe de Proxy ont en commun le faite		Q 35.
1	qu'elles héritent toutes deux d'une classe abstraite	
2	qu'elles implémentent toutes deux une interface	X
3	qu'elles utilisent toutes deux une autre classe	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

Nous avons vu qu'il existe 5 conceptions différentes pour créer un objet distribué :
- par héritage
- par composition
- par interface
- par adaptateur
- par proxy

En une ou deux phrases, pas plus, précisez la raison ou l'avantage d'utiliser chacune de ces conceptions.

Par héritage :
C'est la plus simple et la plus concise: une seule classe. Tout est dans la classe. L'objet distribué contient les données métier.

Par composition :
On encapsule l'objet métier dans un objet distribué. Il y a séparation entre l'objet distribué et l'objet métier. L'objet métier est créé par l'objet distribué.

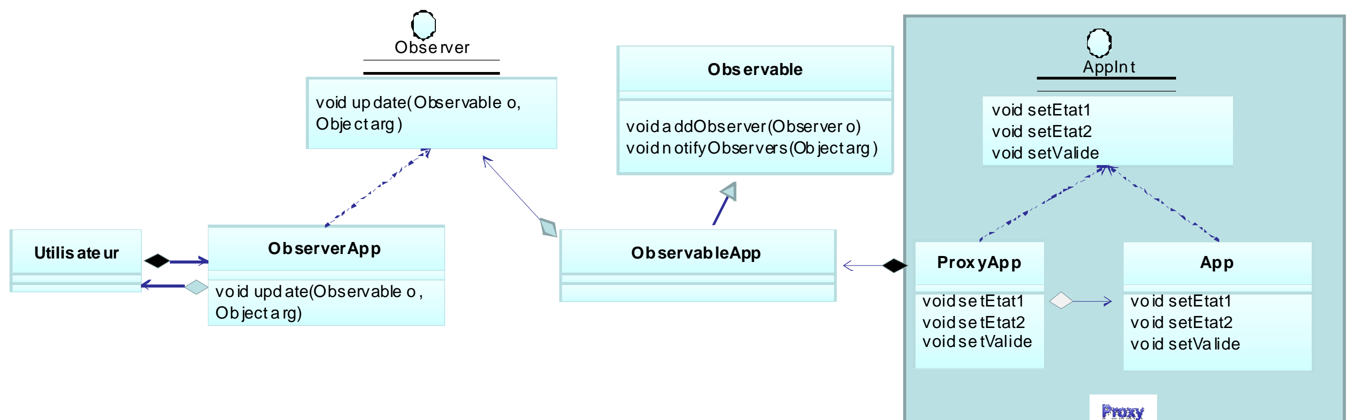
Par interface :
L'objet métier est vu comme une interface par l'objet distribué. Ainsi l'objet distribué peut rendre distant n'importe quel objet métier qui implémente cette interface. Les méthodes distantes ne sont pas identiques aux méthodes de l'objet métier.

Par adaptateur :
Comme précédemment mais dans le cas où l'objet métier ne peut pas implémenter l'interface.

Par proxy :
L'objet métier et l'objet distribué implémente la même interface qui est une interface distante. Les méthodes distantes sont identiques aux méthodes de l'objet métier. L'objet métier peut aussi être utilisé en local.

QUESTION NUMERO 2

Le schéma suivant est une des architectures possibles de conception du DP Observer/Observable :



Commentez ce schéma.

Un DP Proxy (ProxyApp) est utilisé afin de faire la notification par changement d'état de tous les setteurs du modèle App. Ainsi, ProxyApp et App implémentent la même interface AppInt.

La notification de ces changements d'état se fait en utilisant un DP Observer/Observable.
L'obervableApp hérite de Observable et fait la notification à tous ceux qui se sont abonnés (addObserver).
L'observable est encapsulé dans le ProxyApp ce qui rend la conception de App indépendant de la notification.

L'Utilisateur crée un ObserverApp qui implémente l'interface Observer et qui s'est abonné à ObservableApp afin d'être notifié.
Sur notification (exécution de la méthode update), ObserverApp appelle un traitement de Utilisateur.

En synthèse : Utilisateur s'abonne aux changements des états de App.

QUESTION NUMERO 3

Expliquez le principe de base du DP Proxy.
Citez 3 cas d'utilisation de ce DP. Expliquez.

Le principe de base du DP Proxy est de surcharger par encapsulation les méthodes d'une classe qui sont décrites dans une interface. Le Proxy se fait passer pour la classe. Tous ceux qui utilisaient la classe à travers son interface, utilise le Proxy sans toujours le savoir.

1er cas :
Un proxy sur les setteurs des attributs d'un modèle afin de notifier les mises à jour de ces attributs. Il faut que le proxy soit utilisé par ceux qui mettent à jour ces attributs. Le proxy et le modèle implémente la même interface contenant les setteurs.

2ème cas:
Un proxy client qui permet d'utiliser les méthodes distantes d'un objet distribué. Le rôle de ce proxy est de traduire chaque méthode de l'interface en écriture et lecture sur un socket. On appelle ce proxy le stub d'un objet distribué. Il peut être créé par un DynamicProxy.

3ème cas :
Un proxy entre une classe d'Objet Distribué et la classe Métier. Toutes deux implémentent la même interface qui contient les méthodes qui sont à la fois distantes et locales. Celui qui utilise cette interface ne sait pas à priori s'il utilise l'objet de manière distante ou locale.

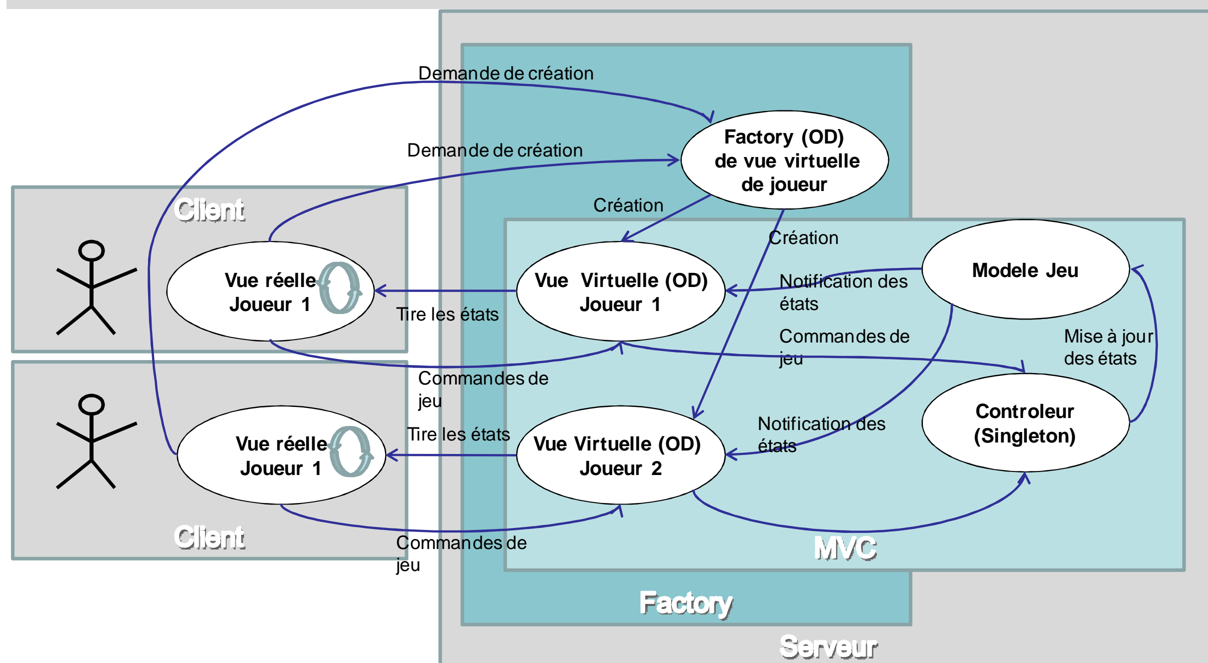
Fin de la 1ère partie sans document

2ème PARTIE – AVEC DOCUMENT (durée: 1h15)

3. PROBLEME (50 points)

1/ Faite le schéma d'architecture logiciel de votre solution (composants, acteurs, fonctions)
Précisez le rôle de chacun des composants.

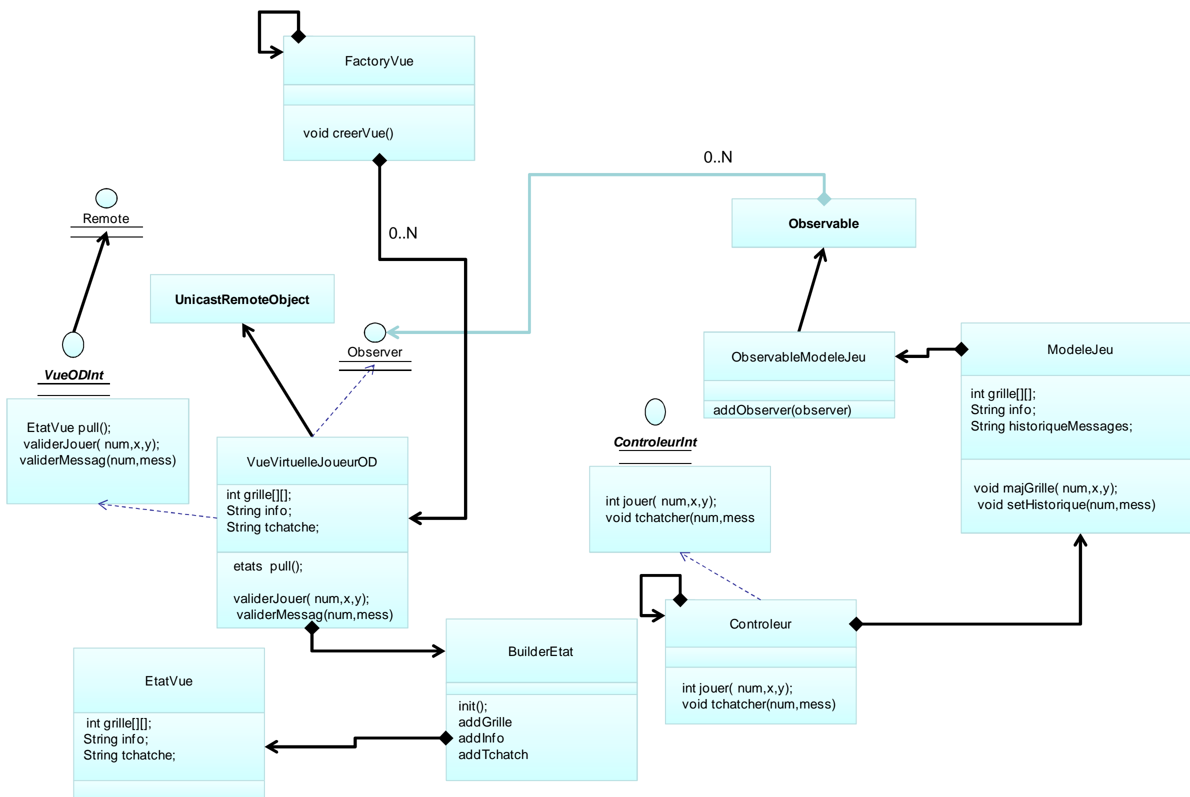
Voici le schéma d'architecture :



Il y a 2 composants :
- le client qui est une IHM (Vue réelle) pour chaque joueur qui tire les états du jeu depuis le serveur de jeu. Son rôle est d'afficher la grille , la zone d'info et la zone de tchatche avec les boutons d'action.
- le serveur de jeu qui contient un modele MVC et un factory. Son trôle est de modéliser le jeu qui est centralisé pour tous les joueurs.

2/ Faire le diagramme de classe UML de chacun des composants. Commentez.
Mettez en évidence les méthodes et les attributs importants.

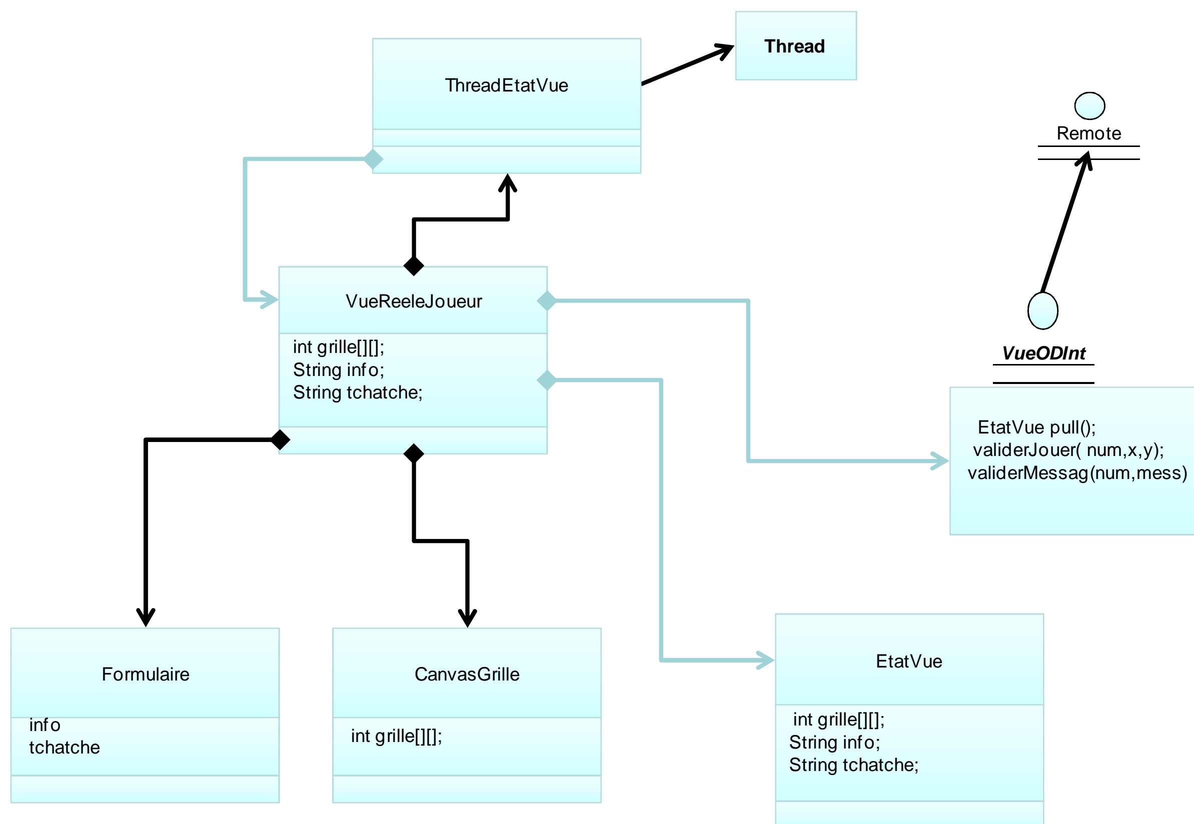
Pour le serveur :



A la demande du client (l'Ihm du joueur), le factory crée une VueVirtuelleJoueurOD qui contient tous les attributs de l'ihm du joueur. Le modèle de jeu est créé au lancement du serveur (controleur et factory).
Chaque vue virtuelle créée est un Observer.
Le modèle notifie aux vues virtuelles les changements des attributs (ou états) du modèle de jeu (les coups joués par les joueurs, les messages d'information de chaque joueur, les messages échangés entre les joueurs).
La vue virtuelle construit un EtatVue qui rassemble les états que le client devra tirer via la méthode pull().
La vue virtuelle est un OD utilisé par le client pour tirer les états et pour réaliser les actions du joueur qui sont autant de méthodes distantes.

Ainsi le modèle MVC fonctionne indépendamment des Ihm qui sont distantes et peuvent être sur des postes banalisés à travers Internet.
Les vues virtuelles sont des "simulations" des vues réelles.

Pour le client :



Le client est une vue (IHM) constitué de Formulaire (message d'info et historique des messages) et d'une grille d'IHM CanvasGrille qui affiche une grille et qui permet au joueur de cliquer dans une case.
Cette vue crée un thread ThreadEtatVue qui tire régulièrement les états depuis la vue virtuelle distante via l'interface distante VueODInt. Ces états sont rassemblés dans un seul objet sérialisé EtatVue.
L'interface distante VueODInt contient aussi les méthodes distantes permettant au joueur de jouer.

3/ Mettez en évidence les Designs Patterns que nous avons vu en cours qui se retrouvent dans vos diagrammes de classe.

Dans le client : le DP Interface pour utiliser l'OD.

Dans le serveur :

Le DP Builder dans le serveur pour créer les états dans un objet unique par assemblage de toutes les informations notifiées par le modèle.

Le DP Factory qui crée des OD VueVirtuelleJoueurOD qui implémentent tous l'interface VueODInt. A la cération le factory retourne le stub de l'OD.

Le DP MVC pour le cœur du serveur de jeu.

Le DP Singleton pour le controleur. Toutes les vues virtuelles créées sur le serveur utilise le controleur comme un singleton.

Le DP Observeur dans le MVC pour faire la notification des changements d'attribut du modèle à toutes les vues virtuelles.