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

Durée : **2 h 30**
Enseignants : LAFORGUE Jacques

1ère Session NSY 102

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	
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	
3	toutes créées dans un Factory unique	

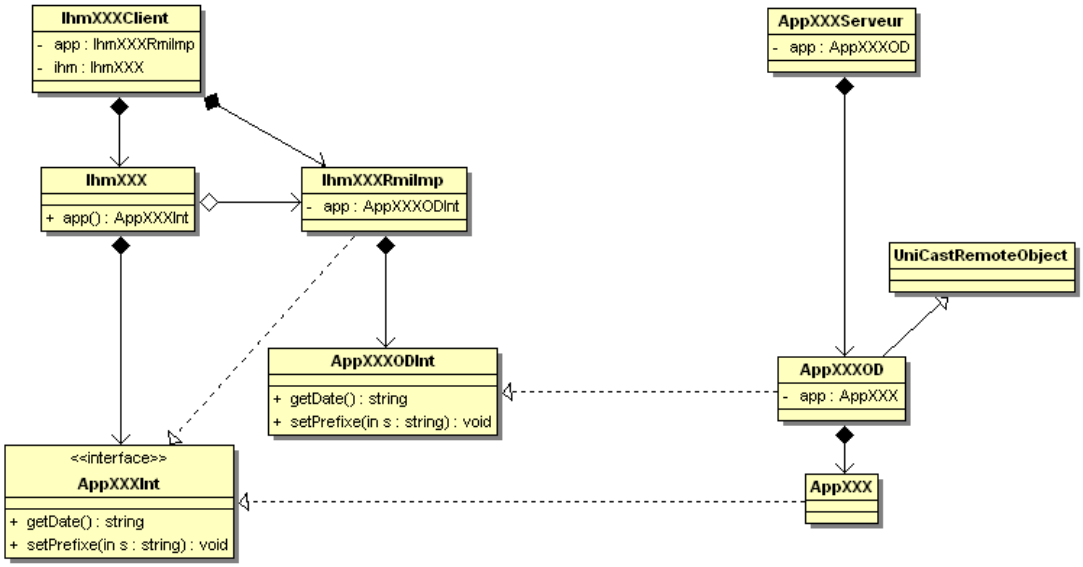
L'IDL (Interface Definition Language) est un langage informatique utilisé par les ORB pour écrire le code des POA et des servants.		Q 3.
1	OUI	
2	NON	

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	
2	NON	

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

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	

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.	



Q 8.

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).

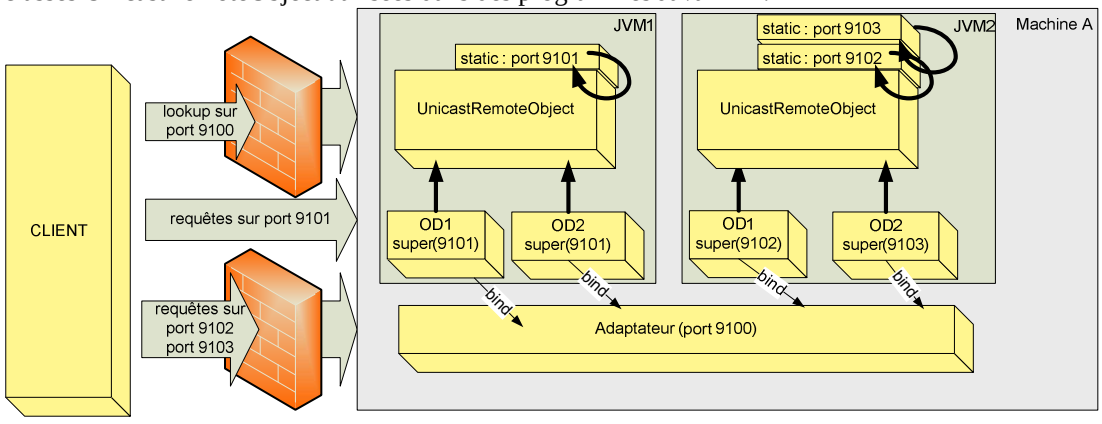
IhmXXXRmiImp est un DP Proxy de AppXXX :

1	OUI	
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	
3	des éléments de type primitif (int, char, long, byte, ...)	

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	
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	

En RMI Java, l'amorce ou stub d'un Objet Distribué est un proxy de l'Objet Distribué.		Q 11.
1	OUI	
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 ();
    }
}
```

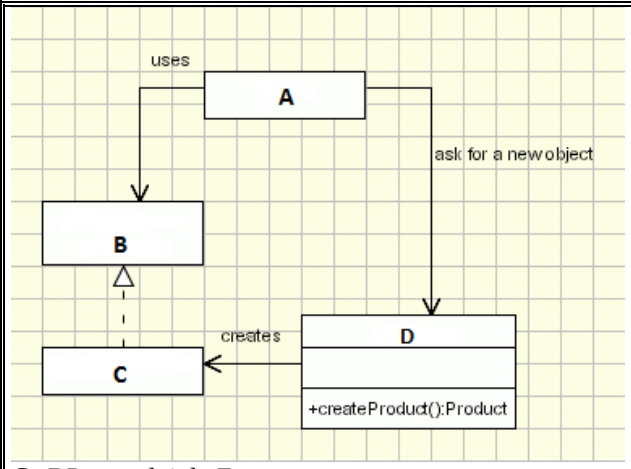
Ce code est correct.

1	OUI
2	NON

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

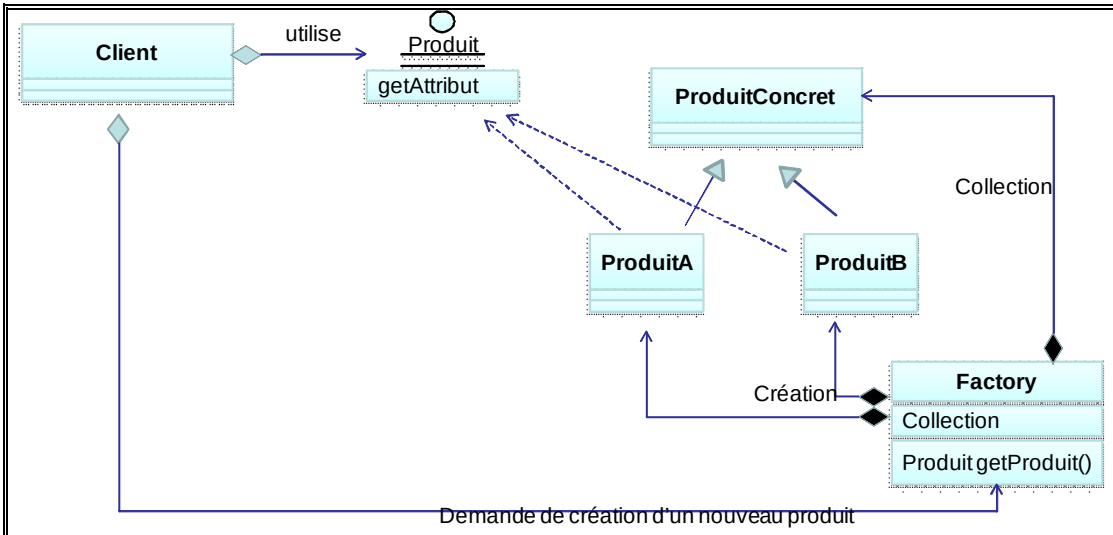
1	OUI
2	NON

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



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	
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	

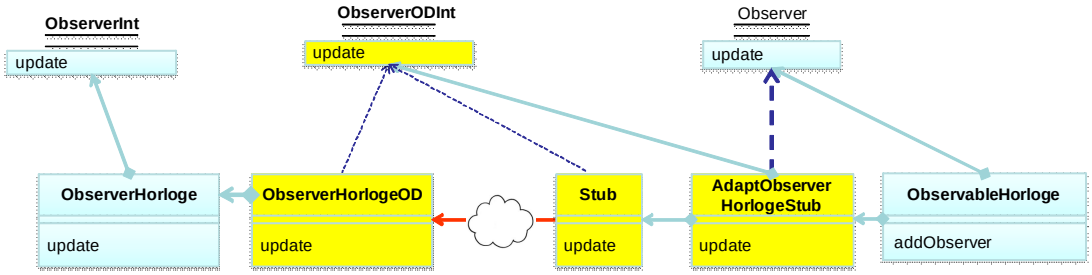
Dans le DP Observeur, la communication entre l'Observer (consommateur d'évènement) et l'Observable (producteur d'évènement) est nécessairement asynchrone car la communication se fait toujours par l'envoi d'un message sans valeur de retour.		Q 16.
1	OUI	
2	NON	

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

Soit le schéma suivant :	Q 20.



Les classes et interfaces en gris 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.	

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	

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	

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	
3	via le modèle du "Pull" en passant par un intermédiaire	

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.	
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	

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

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

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	
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.	
2	L'interface Observer contient une méthode (par exemple update) qui est appelée par l'Observable	
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	
2	NON	

Le rôle d'un factory est, entre autre, de :		Q 31.
1	de créer à la demande de nouveaux objets	
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	

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	
2	NON	

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

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	
3	qu'elles utilisent toutes deux une autre classe	

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.

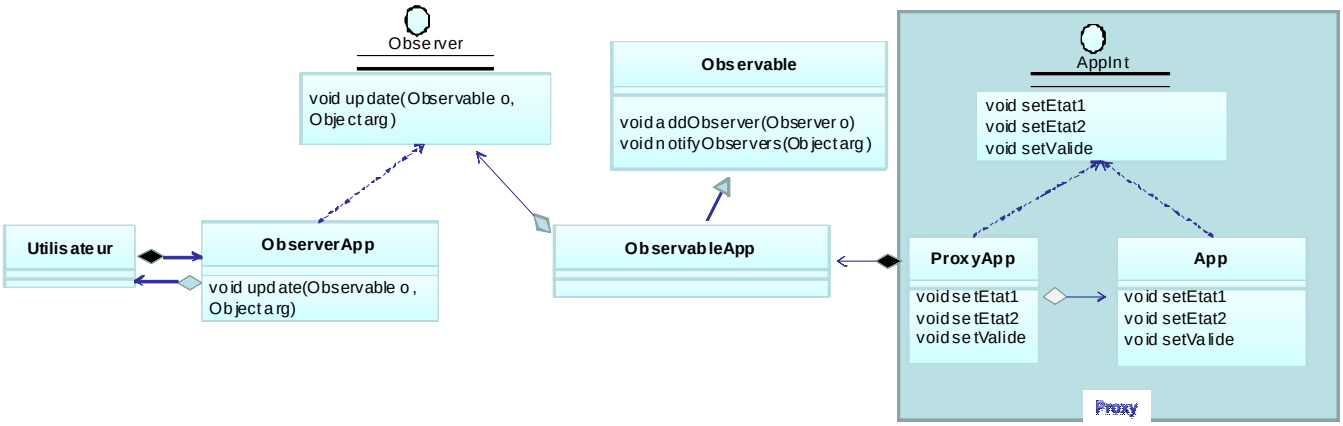
QUESTION NUMERO 1

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.

QUESTION NUMERO 2

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



Commentez ce schéma.

QUESTION NUMERO 3

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

Fin de la 1ère partie sans document

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

3. PROBLEME (50 points)

Nous envisageons de réaliser un système d'information Intranet de jeu de go, de morpion, ... de tout jeu qui consiste à sélectionner la case d'une grille afin d'y déposer son pion de couleur.

Le serveur de jeu gère de 2 à N joueurs.

Chaque joueur utilise une Ihm (ou vue "réelle") sur sa machine qui est en communication avec le serveur de Jeu.

Le serveur de jeu est développé suivant un modèle MVC où les vues sont des vues **virtuelles** créées par un factory distant. Il y a autant de vue virtuelle que de joueur.

Chaque vue réelle de chacun des joueurs tire cycliquement les états de la vues virtuelles. On parle de synchronisation des états entre la vue réelle et la vue virtuelle de chacun des joueurs.
Chaque action d'un joueur se fait par l'envoi d'une requête à la vue virtuelle qui à son tour envoie la commande au contrôleur du modèle MVC.

Chaque mise à jour du modèle entraine une notification locale (Observer/Observalble) aux vues virtuelles.

Une vue (virtuelle et réelle) est caractérisée par :
- une grille de taille 19x19 dont chaque case a la valeur de 0 à 10 (0 = case libre, 1 à 10 = couleur du joueur)
- une zone d'information permettant au modèle de jeu d'afficher un texte à un joueur (Exemples : "A vous de jouer", "Attendez", "Votre coup est invalide", ...)
- une zone de tchatche permettant d'envoyer un message à tous les joueurs et une zone d'historique de tous les messages échangés.

Les actions d'un joueur sont :
- jouer un coup en désignant une case x,y et précisant son numéro de joueur
- saisir et envoyer un message de tchatche à tous les joueurs.

Le modèle est constitué :
- d'une grille de taille 19x19 dont chaque case a la valeur de 0 à 10 (0 = case libre, 1 à 10 = couleur du joueur)
- de la liste des joueurs (chaque joueur est caractérisé par le texte à lui afficher)
- l'historique de tous les messages échangés entre les joueurs.

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

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

3/ Mettez en évidence certains des Designs Patterns vu en cours qui se retrouvent dans vos diagrammes de classe.