

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

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

1ère Session NSY 102

1^{ère} 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
 - -1/2 pour chaque réponse fausse
- **les questions à 3 propositions** dont 1 seule proposition est fausse
 - + 1/2 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	

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	
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	

<p>Ce schéma représente une architecture 3-Tiers</p>		Q 4.
1	OUI	
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)	
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 et des squelettes - une API de classes prédéfinis pour programmer son application distribuée		Q 6.
1	OUI	
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	

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

<pre> classDiagram class IhmXXXClient { - app : IhmXXXRmiImp - ihm : IhmXXX } class IhmXXX { + app() : AppXXXInt } class IhmXXXRmiImp { - app : AppXXXODInt } class AppXXXODInt { + getDate() : string + setPrefix(in s : string) : void } class AppXXXOD { - app : AppXXX } class AppXXX { } class AppXXXServer { - app : AppXXXOD } class UniCastRemoteObject class AppXXXInt { <<interface>> + getDate() : string + setPrefix(in s : string) : void } IhmXXXClient o-- IhmXXX IhmXXXClient o-- IhmXXXRmiImp IhmXXX o-- IhmXXXRmiImp IhmXXXRmiImp o-- AppXXXODInt AppXXXODInt .. > AppXXXInt AppXXXODInt .. > AppXXXInt AppXXXODInt .. > AppXXXInt AppXXXOD o-- AppXXX AppXXXServer o-- AppXXXOD AppXXXServer o-- UniCastRemoteObject </pre>		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.		
La classe IhmXXXRmiImp est un Adaptateur de AppXXX :		
1	OUI	
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.

The diagram illustrates the RMI architecture. On the left, a CLIENT sends requests to an orange brick-shaped component. The top request is labeled 'lookup sur port 9100'. The middle request is 'requêtes sur port 9101'. The bottom request is 'requêtes sur port 9102 port 9103'. This component connects to a yellow 'Adaptateur (port 9100)' at the bottom. Above the adapter, two JVMs are shown on 'Machine A'. JVM1 contains 'UnicastRemoteObject' (static: port 9101) and two ODs: 'OD1 super(9101)' and 'OD2 super(9101)'. JVM2 contains 'UnicastRemoteObject' (static: port 9102) and two ODs: 'OD1 super(9102)' and 'OD2 super(9103)'. Arrows labeled 'bind' point from the adapter to each OD in both JVMs.

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 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

1	OUI	
2	NON	

Un Design Pattern (DP) ou Patron est une norme de description des interfaces entre les composants d'une architecture logicielle orientée objet

1	OUI	
2	NON	

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.

1	OUI	
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

1	OUI	
2	NON	

Q 16.

Ce DP est celui du Factory.
ProduitConcret 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
2	NON

Q 17.

Ce DP est celui d'un Factory.

1	OUI
2	NON

<pre> classDiagram class AbstractBuilderCible { +protected Cible cible +public Cible getCible() +abstract public AbstractBuilderCible buildSousObjet1(...) +abstract public AbstractBuilderCible buildSousObjet2(...) +abstract public AbstractBuilderCible buildSousObjet3(...) } class BuilderCible { +public BuilderCible() +public AbstractBuilderCible buildSousObjet1(...) +public AbstractBuilderCible buildSousObjet2(...) +public AbstractBuilderCible buildSousObjet3(...) } class Builder4Cible class Builder1Cible class Builder2Cible class Builder3Cible class Cible { +Cible() // Init défaut +public void setSousObjet1(SousObjet1 o) +public void setSousObjet2(SousObjet2 o) +public void setSousObjet3(SousObjet3 o) } class SousObjet1 class SousObjet2 class SousObjet3 AbstractBuilderCible < -- BuilderCible AbstractBuilderCible < -- Builder4Cible BuilderCible < -- Builder1Cible BuilderCible < -- Builder2Cible BuilderCible < -- Builder3Cible AbstractBuilderCible o-- BuilderCible Cible *-- SousObjet1 Cible *-- SousObjet2 Cible *-- SousObjet3 </pre>	<p>Q 18.</p>	
<p>Ce DP est celui du Builder.</p> <p>Le rôle de la classe BuilderCible est de construire totalement ou partiellement une instance de Cible</p>		
1	OUI	
2	NON	

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

<p>Le DP "Délégation" est utiliser dans le DP "injection de dépendance"</p>	<p>Q 20.</p>	
1	OUI	
2	NON	

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

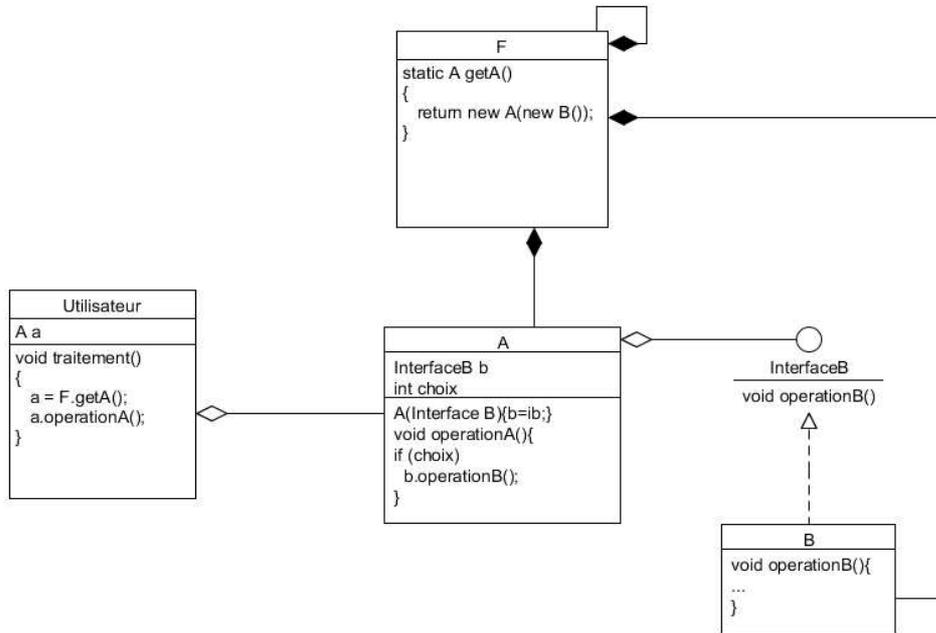
<p>Si la classe A est un décorateur de la classe B alors les classes A et B héritent toutes deux d'une même classe abstraite.</p>	<p>Q 22.</p>	
1	OUI	
2	NON	

<p>L' "inversion de contrôle" est un principe de conception qui:</p>	<p>Q 23.</p>	
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	

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

Le diagramme suivant :

Q 25.



représente

1	une injection de dépendance par l'utilisation d'un setter	
2	une injection de dépendance par l'utilisation d'un constructeur	
3	une injection de dépendance par l'utilisation d'un proxy	

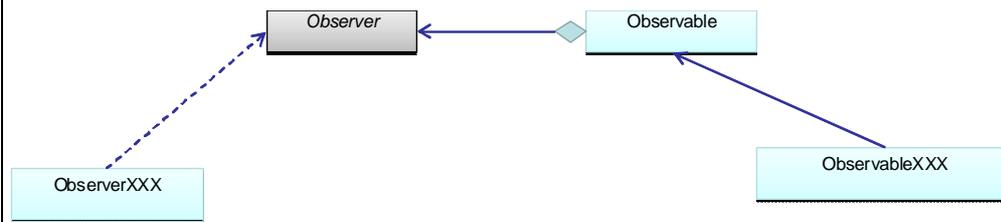
Dans le DP Observateur, l'Observable réalise la notification des tous ses observateurs toujours de manière synchrone

Q 26.

1	OUI	
2	NON	

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	
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	
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"	

Soit le diagramme de classe suivant :

Q 30.

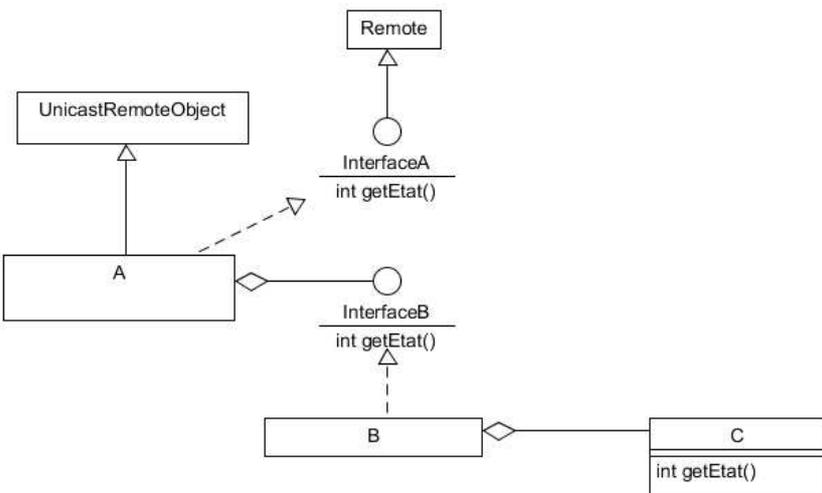
Ce diagramme de classe représente celui d'un DP Adaptateur car la classe XXX ne pouvant pas implémenter l'interface Interface, on crée une classe AdaptateurXXX qui le fait pour elle

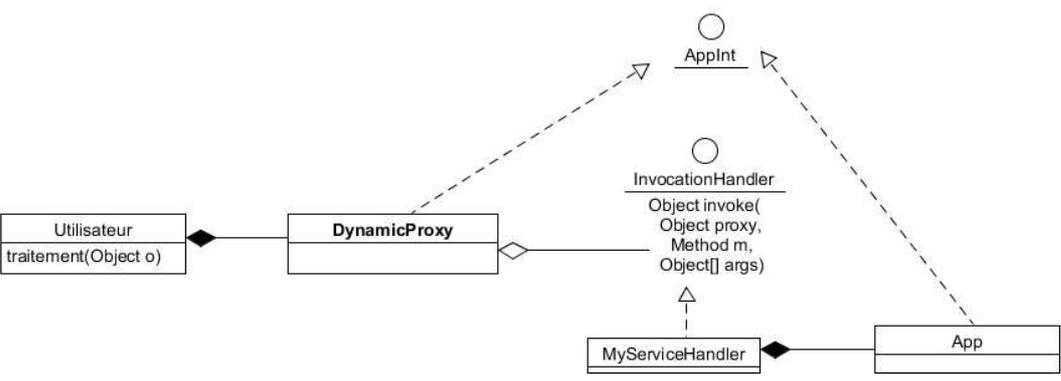
1	OUI	
2	NON	

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	
2	NON	

 <p>The diagram shows a class hierarchy and composition. InterfaceA has a method <code>int getEtat()</code>. InterfaceB also has <code>int getEtat()</code>. Remote inherits from InterfaceA. UnicastRemoteObject inherits from A. A has a dashed dependency on InterfaceA and a composition relationship with B. B has a dashed dependency on InterfaceB and a composition relationship with C. C has a method <code>int getEtat()</code>.</p>	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	

 <p>The diagram shows a Dynamic Proxy pattern. AppInt is an interface with a method <code>Object invoke(Object proxy, Method m, Object[] args)</code>. InvocationHandler is an abstract class that implements AppInt. MyServiceHandler inherits from InvocationHandler. App is a class that inherits from AppInt and has a composition relationship with MyServiceHandler. DynamicProxy is a class that has a composition relationship with App and a composition relationship with InvocationHandler. Utilisateur is a class that has a composition relationship with DynamicProxy and a method <code>traitement(Object o)</code>.</p>	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	
2	d'implémenter toutes les méthodes de l'interface AppInt	
3	d'appeler les méthodes de App décrites dans l'interface AppInt	

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	
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	
2	NON	

Fin du QCM

Suite (Tournez la page)

2. Questions libres (15 points)

Chaque question est notée sur 5 points.

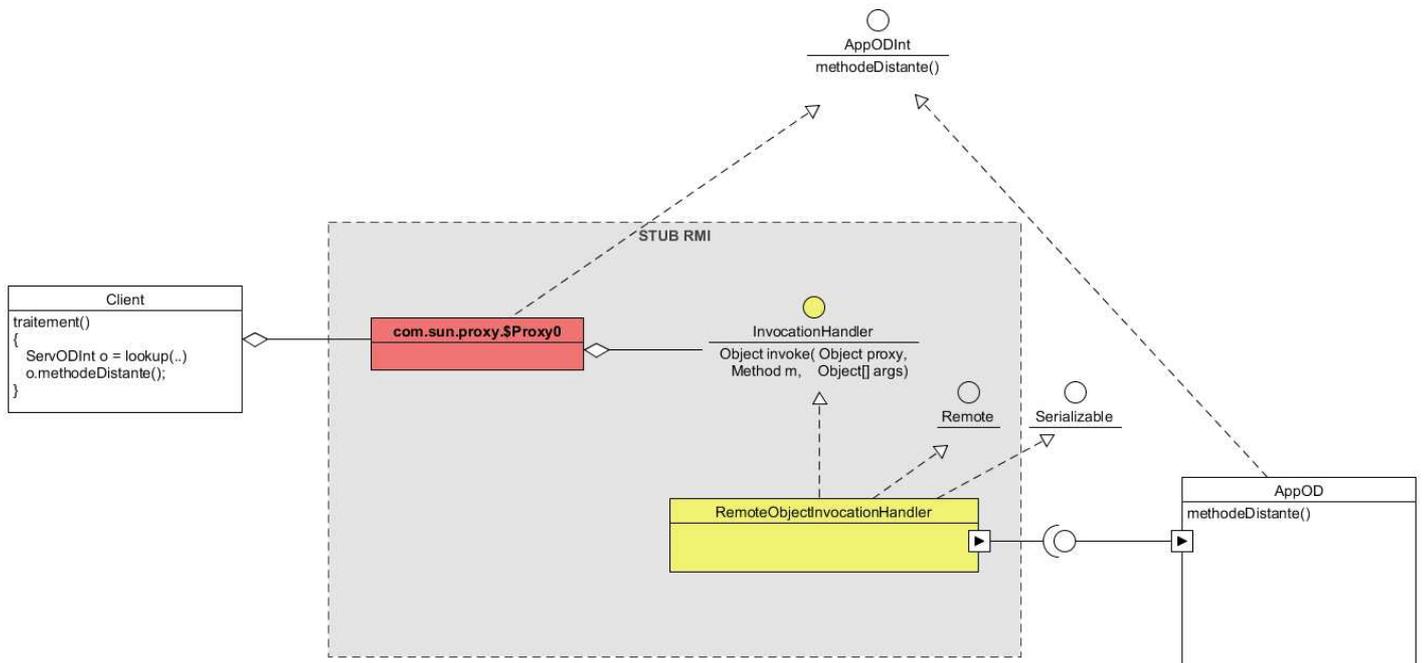
Vous répondez à ces questions sur une **copie vierge double** en mettant bien le numéro de la question, sans oublier votre nom et prénom.

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.

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.

QUESTION NUMERO 3

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

Fin de la 1^{ère} partie sans document

2ème PARTIE – AVEC DOCUMENT (durée: 1h30)**3. PROBLEME (50 points)**

Soit un site de vente en ligne de produits informatiques de 3 types :

- des ordinateurs portables ou fixes, prêts à l'emploi (souris, écran, clavier, uc);
- des accessoires informatiques (clef usb, dd externes, cables, ...);
- des livres d'informatique, des DVD de jeux, ...

La société de vente en ligne possède 3 grandes salles de stock car chacun de ces types de produit nécessite des manipulations et une expertise différente (configuration des ordinateurs, emballages différents, ...).

Chaque salle dispose de 1 à N poste de préparation (en fonction de l'activité) [COMPOSANT 2] (pour 1 poste) permettant de préparer les produits de la commande concernant la salle.

Un poste unique de gestion [COMPOSANT 1] permet de dispatcher les produits des commandes vers les salles de préparation (les commandes sont chargées en mémoire de ce poste unique issues depuis une base de données. L'aspect base de données n'est pas à traiter ici).

Quand un des préparateurs d'une salle décide de préparer les produits (il ne traite que les produits du type le concernant) d'une commande, il verrouille sa préparation en cours. Une fois le colis terminé contenant les produits, il valide la terminaison de sa préparation. Cette validation prévient le poste de gestion qu'une partie de la commande a été réalisée.

Une fois la commande complète, le poste de gestion prévient un poste de livraison [COMPOSANT 3], afin qu'un opérateur rassemble au plus les 3 colis dans un seul. Vous devrez utiliser un Dynamic Proxy pour réaliser cette notification.

1/ Faites le schéma d'architecture logicielle de votre solution (composants, acteurs, fonctions). Commentez votre schéma (fonctionnement, rôle, fonctions).

2/ Faire le diagramme de classe UML des composants : [COMPOSANT 1] et [COMPOSANT 2] en mettant en évidence certains des Design Patterns vus en cours.

Pour une description précise de vos diagrammes de classe, on peut faire le choix que toutes les communications distantes entre les composants sont réalisées en RMI, mais vous pouvez faire le choix d'un autre standard de communication (si cela est le cas précisez le standard utilisé).