IPST-CNAM
Intranet et Designs patterns
NSY 102
Mercredi 27 Mars 2019

Durée : **2 h 45**
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.

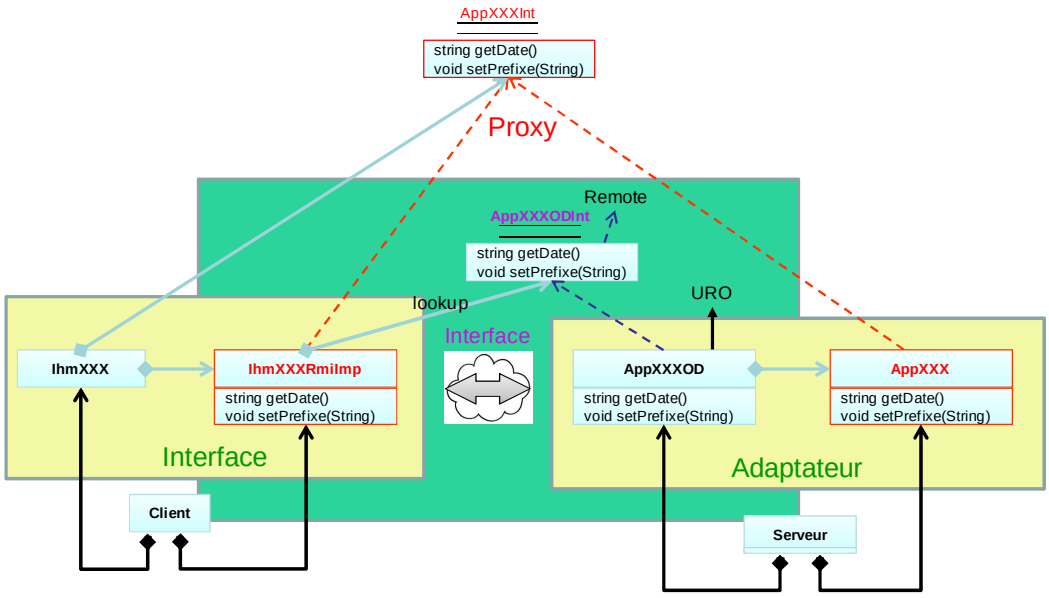
Vous avez droit à **4 points** négatifs sans pénalité.

NOM:	PRENOM:

Soit un objet, instance de la classe A. Pour transformer cet objet en un objet distant, il suffit que ::		Q 1.
1	A hérite de la classe UnicastRemoteObject et implémente une interface publique (I) qui hérite de Remote	
2	A soit un proxy de la classe UnicastRemoteObject	
3	A soit une agrégation d'une classe B qui hérite de la classe UnicastRemoteObject et implémente une interface publique (I) qui hérite de Remote	

Soit un objet quelconque Obj (instance de la classe A qui n'hérite pas d'une autre classe). En Java RMI, il est très facile de transformer cet objet en un objet distant. Pour cela il suffit de :		Q 2.
1	faire que la classe A implémente 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 de A qui hérite de Remote	

Q 3.

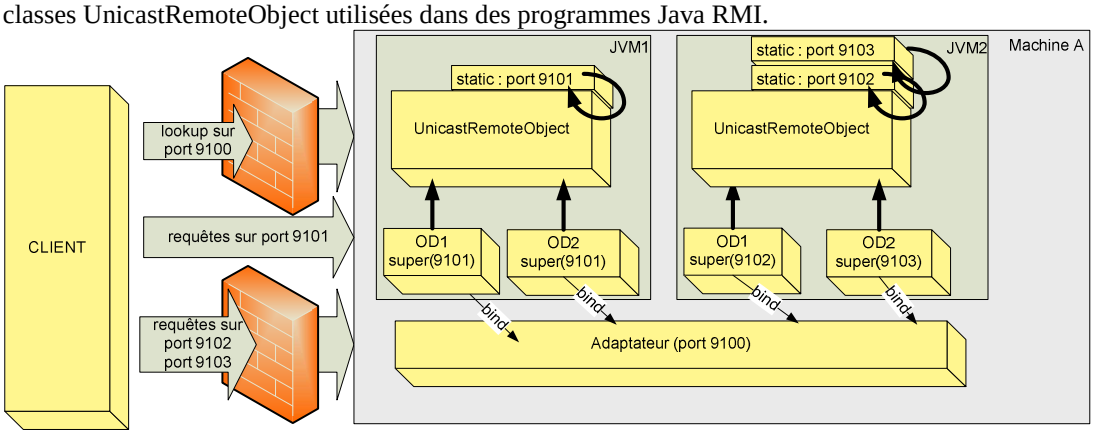


Ceci est le diagramme de classe d'un système composé d'un client IHM (classe IhmXXX) et de son applicatif (AppXXX) que l'on veut rendre distant.

IhmXXXRmiImp est un DP Proxy Client 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. — Q 4.



1	On peut créer un nouvel OD (Objet Distant) 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 9103	
3	Dans la JVM2, on peut créer un nouvel OD sur le port 9102	

Un Design Pattern (DP) est une implémentation spécifique d'un principe général de conception décrit sous la forme d'un diagramme de classe		Q 5.
1	OUI	
2	NON	

Le DP Singleton permet de rendre transparent pour le programmeur la création unique d'un objet		Q 6.
1	OUI	
2	NON	

Soit le code suivant d'implémentation d'un singleton : — Q 7.

```
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	

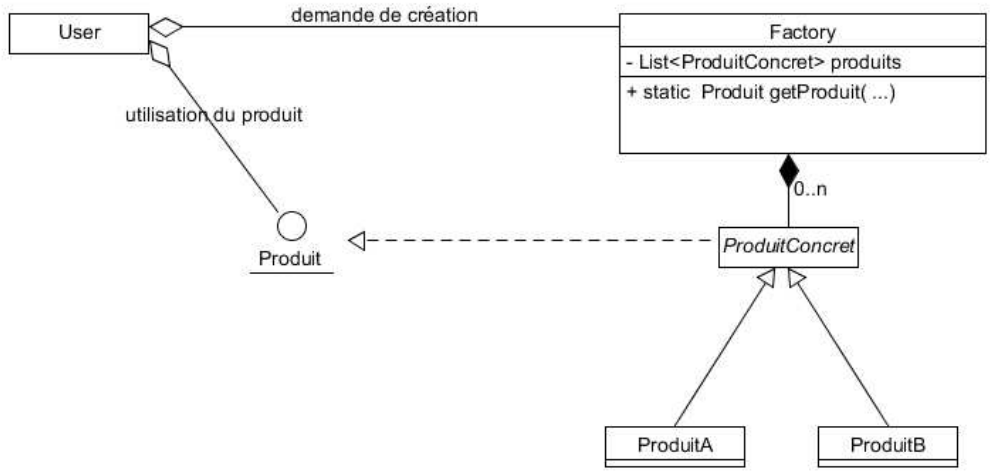
Dans un système réparti, le DP Singleton permet de créer un objet distant unique sur le réseau.		Q 8.
1	OUI	
2	NON	

Le DP Factory est aussi un DP Singleton		Q 9.
1	OUI	
2	NON	

Un DP Singleton est un Factory qui crée un produit unique.		Q 10.
1	OUI	
2	NON	

Le DP Factory a pour fonction la création d'objet dont les classes héritent d'une même classe abstraite ou implémentent la même interface.		Q 11.
1	OUI	
2	NON	

Q 12.

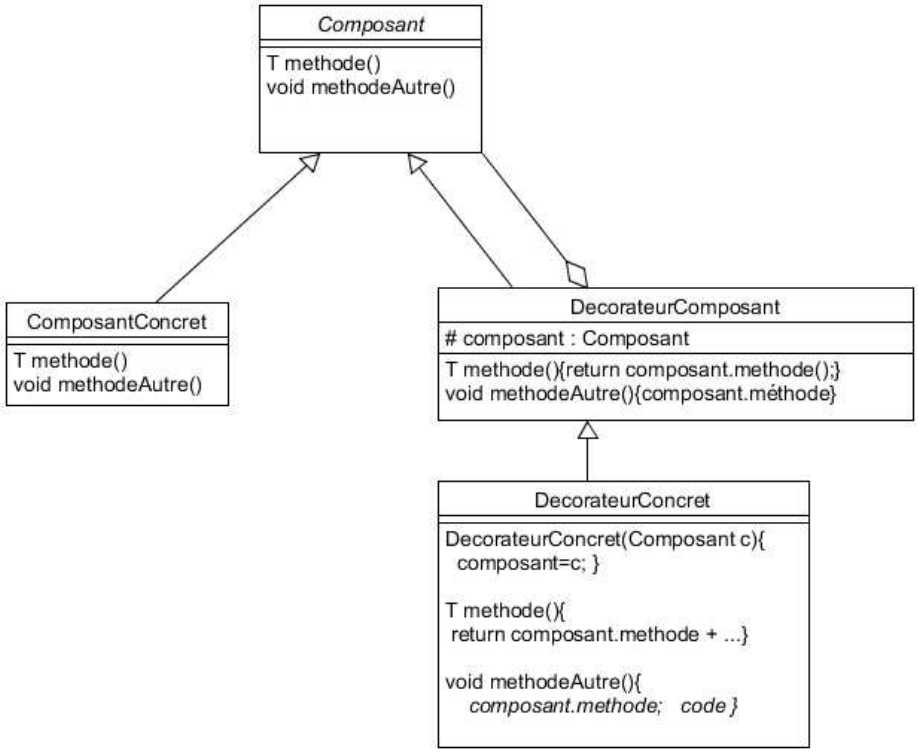


Ce DP est celui du Factory.
ProduiConcret est une classe abstraite dont héritent les classes ProduitA et ProduitB
Le rôle de la méthode getProduit du Factory est :

1	de demander à la classe ProduitConcret de créer (new) des objets de type Produit	
2	de créer des produits en faisant l'instanciation des classes ProduitA ou ProduitB.	

Le DP Builder peut être utilisé dans le DP Factory afin de faciliter la production d'un objet complexe.		Q 13.
1	OUI	
2	NON	

Si le rôle d'un Factory est de créer des objets distants alors l'interface de tous ces produits est une interface qui hérite de Remote.		Q 14.
1	OUI	
2	NON	



		Q 15.
Ce schéma est celui du DP Décorateur. Ce schéma est correct.		
1	OUI	
2	NON	

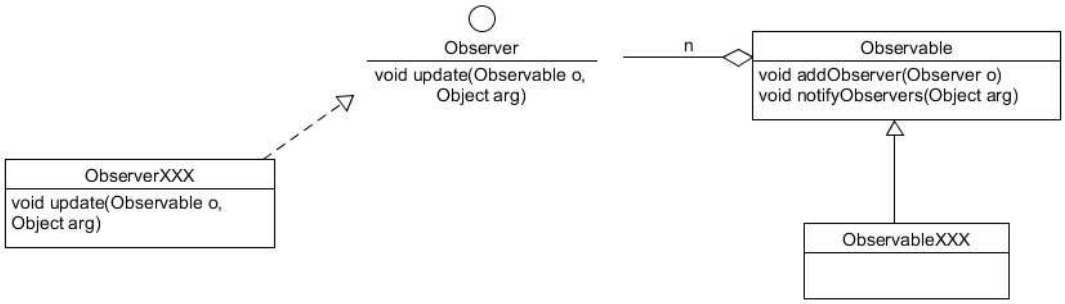
L'injection de dépendance utilise le principe de l'inversion de contrôle (IoC) appliqué au contrôle de la dépendance entre deux classes.		Q 16.
1	OUI	
2	NON	

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 17.
1	OUI	
2	NON	

Le Design Pattern Observateur est utilisé dans :		Q 18.
1	le Design Pattern Factory	
2	le Design Pattern MVC (Model-Vue-Controleur)	
3	le Design Pattern IoC (Inversion de contrôle)	

Le DP Observeur/Observable, peut être utilisé pour réaliser un connecteur Producteur/Consommateur		Q 19.
1	OUI	
2	NON	

Soit le Design Pattern Observateur suivant :	Q 20.

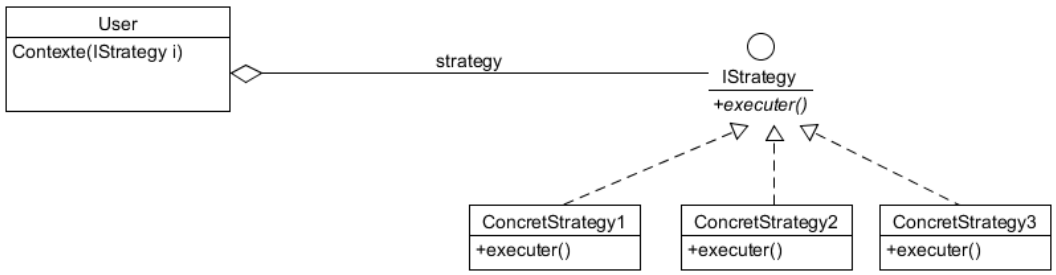


La classe ObserverXXX implémente la méthode update de l'interface Observer qui est appelée par ObservableXXX

1	OUI	
2	NON	

Comme dans l'Injection de Dépendance, le DP Stratégie permet d'injecter dynamiquement un traitement générique dans un utilisateur.	Q 21.
1	OUI
2	NON

Q 22.

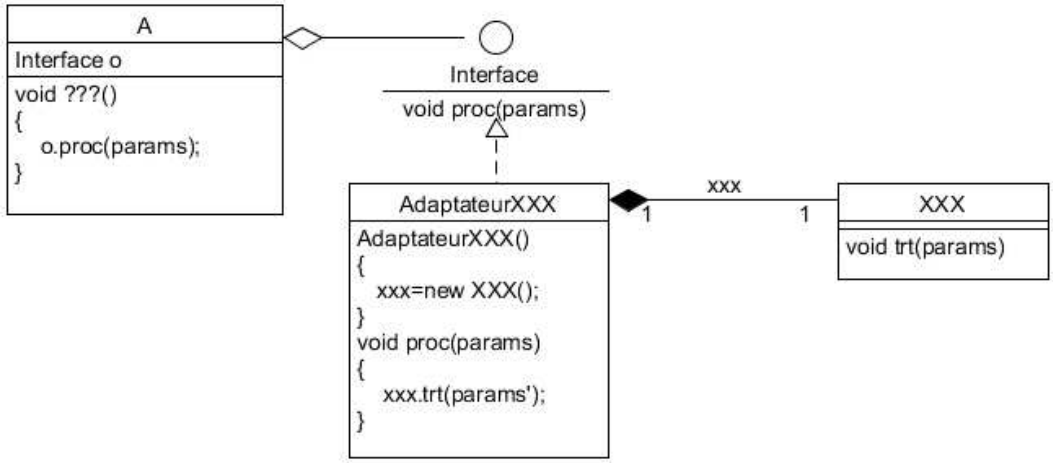


Ce schéma est le diagramme du DP Stratégie.

1	OUI	
2	NON	

Dans le DP Adaptateur, l'adaptateur et l'adapté implémente la même interface.	Q 23.
1	OUI
2	NON

Soit le diagramme de classe suivant :	Q 24.



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	

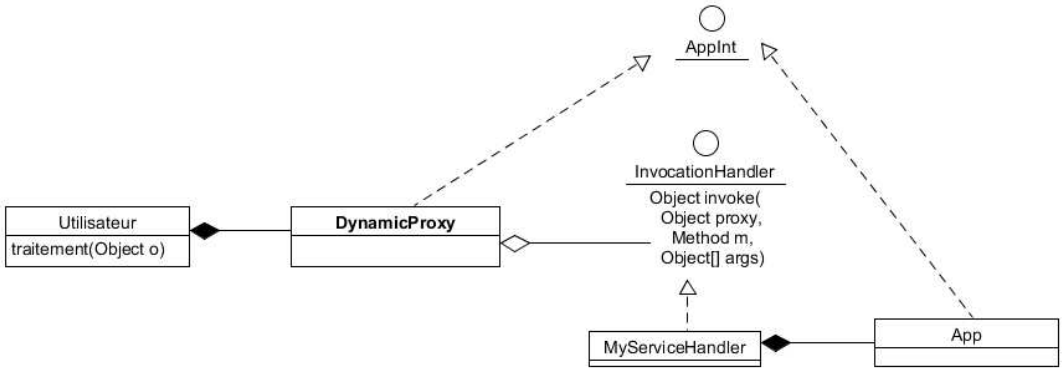
Dans le DP MVC, le Modèle utilise le DP Observer/Observable pour :		Q 25.
1	recevoir les actions du contrôleur de mise à jour du Modèle	
2	envoyer aux vues les notifications de changement des états du Modèle.	

On peut utiliser le Design Pattern Proxy pour rendre distant		Q 26.
1	une classe quelconque	
2	une classe qui implémente l'interface du Proxy	

En RMI, le "stub" est un Proxy sur l'interface qui hérite de Remote.		Q 27.
1	OUI	
2	NON	

Dans la communication synchrone via un "canal d'évènement" entre un producteur et des consommateurs, le producteur utilise un proxy de consommateur (et non les consommateurs directement), afin de leurs pousser un évènement.		Q 28.
1	OUI	
2	NON	

Laquelle des descriptions suivantes est un principe de communication synchrone ?		Q 29.
1	le producteur dépose à son rythme ses évènements dans une file. Le ou les consommateurs peuvent alors récupérer ces évènements	
2	le producteur pousse ("push") chaque évènement vers chacun des consommateurs via une méthode distante qui retourne un état de consommation	



Ce schéma est celui du DP Dynamic proxy.
Le rôle de la classe MyServiceHandler est ici de : Q 30.

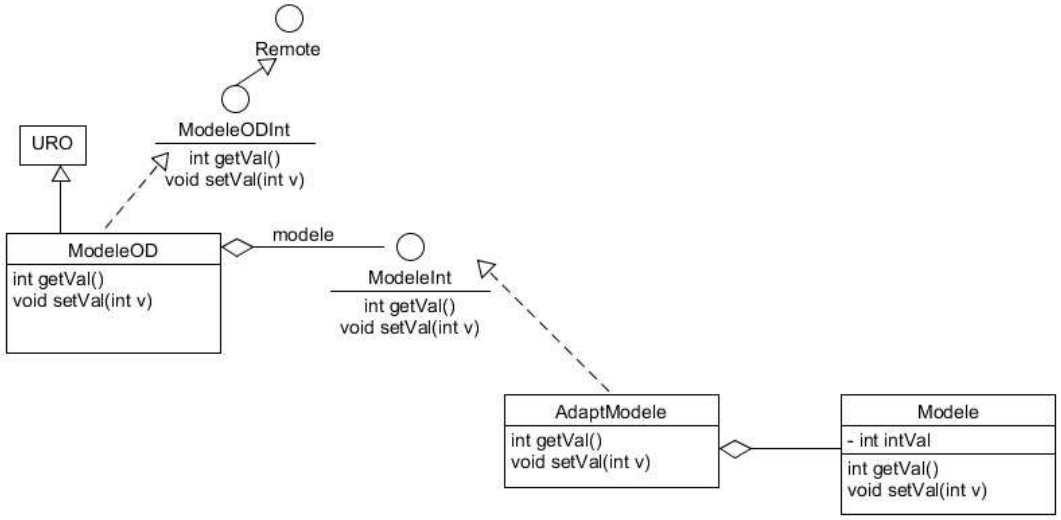
1	créer une instance d'une classe qui implémente l'interface AppInt, dont le rôle (l'instance) est de servir de proxy à l'appel des méthodes de App	
2	d'implémenter toutes les méthodes de l'interface AppInt	
3	d'appeler les méthodes de App décritent dans l'interface AppInt	

Le Dynamic Proxy est utilisé dans la technologie RMI de Java, pour :		Q 31.
1	créer dynamiquement le skelton permettant de traiter les requêtes des clients reçues par l'objet distant	
2	créer dynamiquement le stub permettant d'envoyer les requêtes des clients à l'objet distant.	

En Java, dans un Dynamic Proxy, le chargement de classe permet de réaliser de l'injection de dépendance entre la classe utilisatrice et la classe de service utilisé.		Q 32.
1	OUI	
2	NON	

Dans le DP Observer/Observable de base (de Java), le mode de communication entre les Observers et l'Observable est :		Q 33.
1	synchrone	
2	asynchrone	

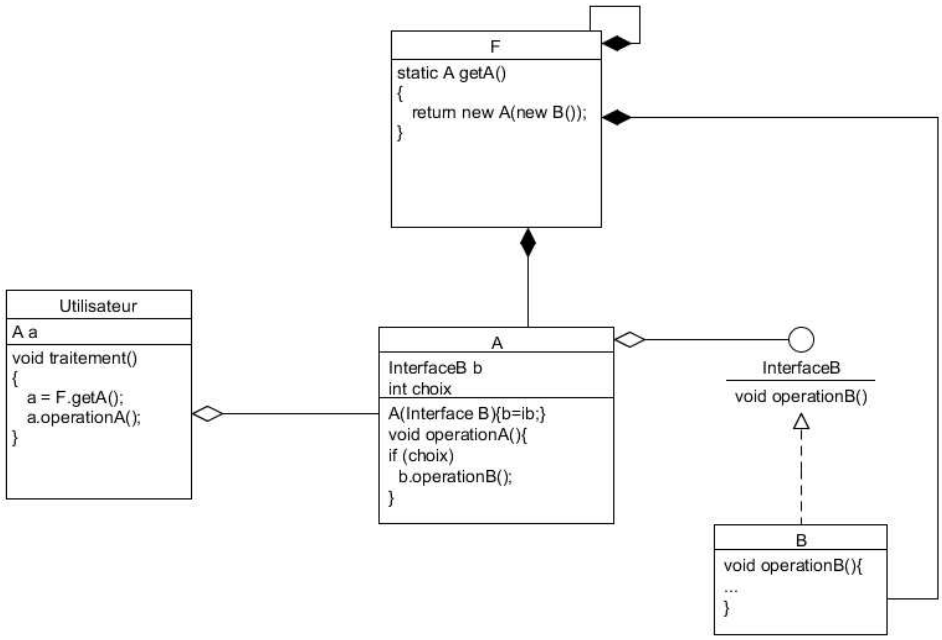
Q 34.



Ce DP est une des formes de conception d'un Objet Distant.
Ce DP est composé de :

1	1 Adaptateur	
2	2 Adaptateurs	
3	1 Proxy et 1 Adaptateur	

Le diagramme suivant : Q 35.



représente

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

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 **Builder**.
Expliquez le comportement de ce design pattern.

QUESTION NUMERO 2

Expliquer le rôle du DP "Injection de dépendance".
Citez, avec précision, 2 exemples pratiques de l'utilisation de ce DP.

QUESTION NUMERO 3

Expliquez, en terme de Design Pattern, la différence entre une les deux modes de communication "push" et "pull", réalisé entre un Producteur et ses Consommateurs.

Fin de la 1ère partie sans document

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

3. EXERCICE [15 points]

Faites le diagramme de classe d'une usine de fabrique dont chaque objet produit peut être utilisé, soit de manière locale, soit de manière distante. Le choix de création des objets, locaux ou distants, est un choix réalisé dans un fichier de configuration unique. Vous devez utiliser au moins les DP suivants : Factory, Singleton, Stratégie, Objet Distant, Proxy, Injection de Dépendance.
Commentez votre schéma. (Le commentaire est noté).

4. PROBLEME [35 points]

On se propose de créer un Système d'Information (SI) de gestion de portefeuilles d'actions cotées en bourse.

Le Serveur [COMPOSANT 1] permet de créer des portefeuilles d'action pour les utilisateurs du SI, et d'ajouter à un portefeuille en particulier, une nouvelle action boursière dont l'utilisateur a acheté des titres.
Ce Serveur se connecte à la bourse de Paris, de Londres et de Tokyo à travers trois interfaces distantes. Ces interfaces lui permettent de consulter, à la demande, les valeurs boursières de certaines actions. Il doit être possible de configurer le serveur avec d'autres bourses de par le monde. Il existe différents types de support financier d'action gérés par un portefeuille (Action, Obligation, .Bon au trésor, …).

Une IHM [COMPOSANT 2] (distante du Serveur et propre à chaque utilisateur) permet à l'utilisateur de se connecter à son portefeuille.
Cette IHM lui permet de :

- surveiller en temps réel l'évolution de ses actions favorites.
- de créer, pour certaines actions, des seuils en-dessous desquels, une alerte est levée sur le serveur et qui est remontée au niveau de l'IHM.

Cette IHM est une IHM lourde standalone mais pensez à alléger sa complexité grâce à des choix d'architecture adéquate.

Nous faisons l'hypothèse que toutes les données sont gérées en mémoire des différents composants (l'utilisation d'une base de données est accessoire).

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

2/ **[25 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 le(s) diagramme(s).

Conseils :
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