PST-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)

CORRECTION

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.

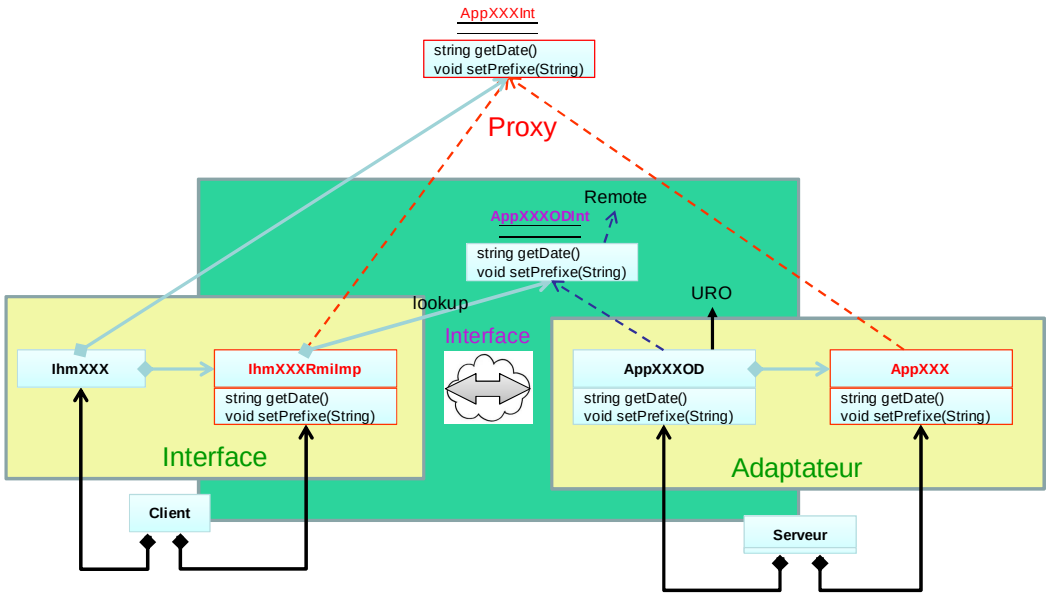
N'oubliez pas d'inscrire en en-tête du QCM, votre nom et prénom.

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	X
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	X

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	X

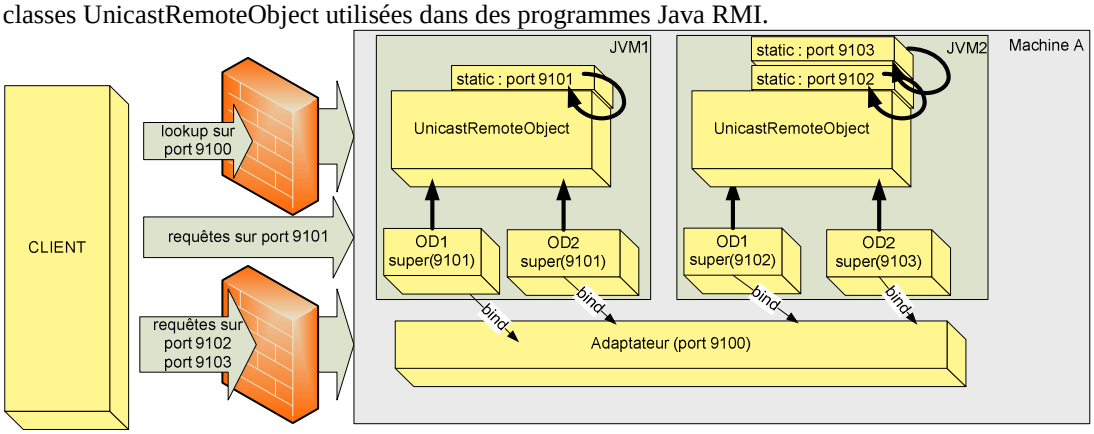


Q 3.

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	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 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	**X**

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	**X**
2	NON	**X**

Le DP Singleton permet de rendre transparent pour le programmeur la création unique d'un objet		Q 6.
1	OUI	**X**
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.
		Q 7.

1	OUI	**X**
2	NON	

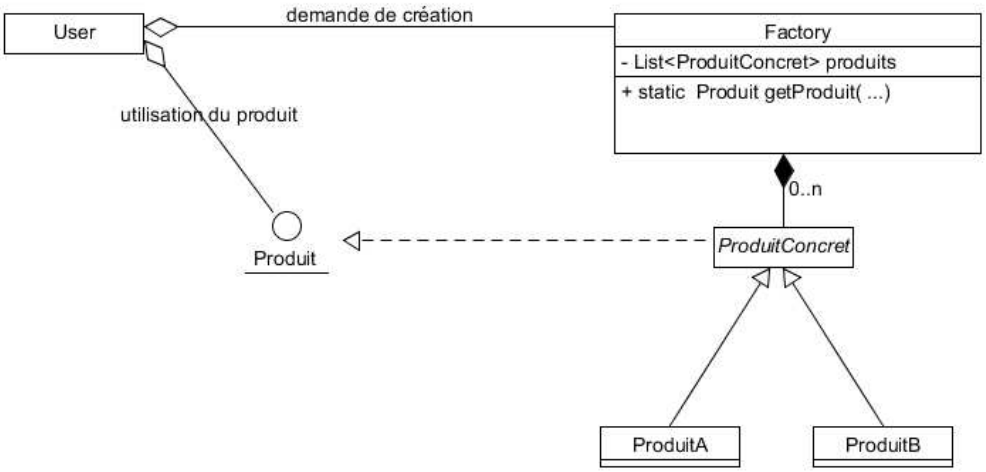
Dans un système réparti, le DP Singleton permet de créer un objet distant unique sur le réseau.		Q 8.
1	OUI	
2	NON	**X**

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

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

Q 12.

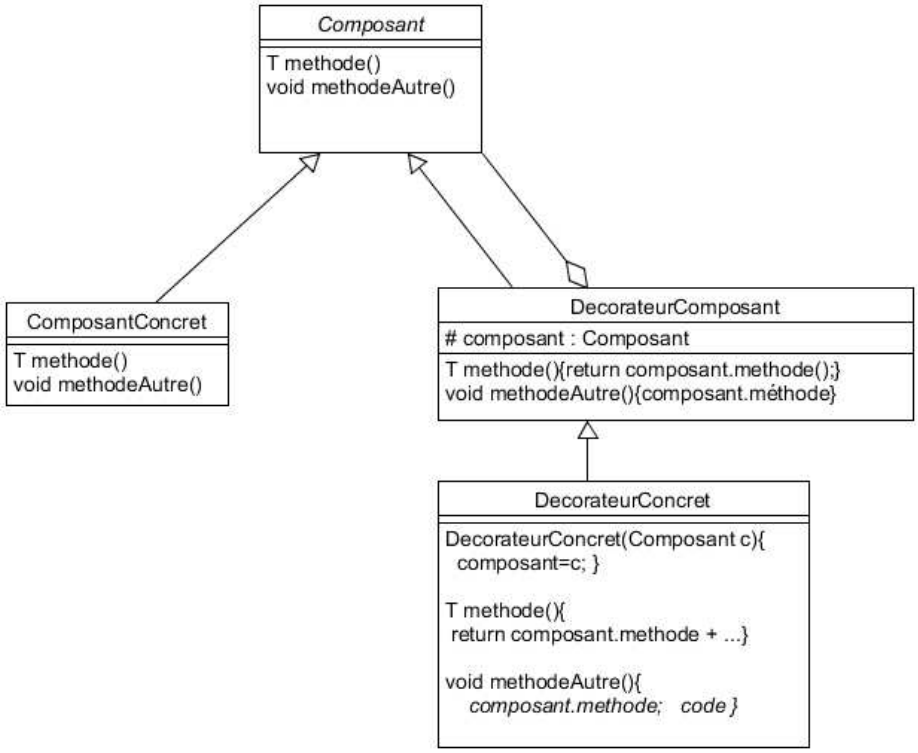


Ce DP est celui du Factory.
ProduiConcret est une classe abstraite dont héritent les classes ProduitA et ProduitB
Le rôle de la méthode getProduit du Factory est :

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.	**X**

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



Q 15.

Ce schéma est celui du DP Décorateur.
Ce schéma est correct.

1	OUI	**X**
2	NON	

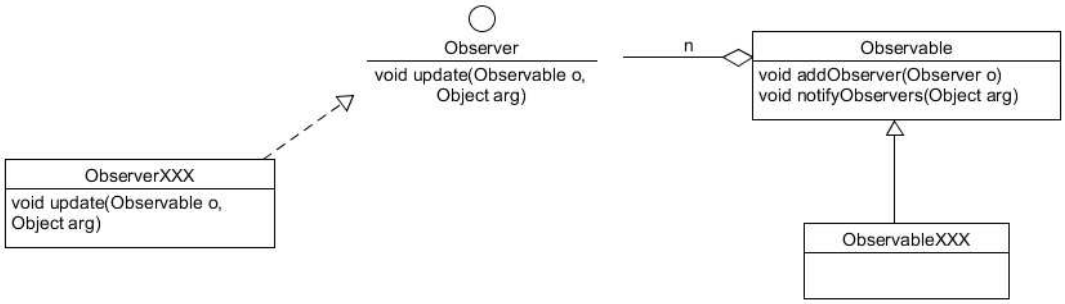
L'injection de dépendance utilise le principe de l'inversion de contrôle (IoC) appliqué au contrôle de la dépendance entre deux classes.		Q 16.
1	OUI	**X**
2	NON	

Dans le DP Observateur, 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	**X**

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

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

Soit le Design Pattern Observateur suivant :	Q 20.

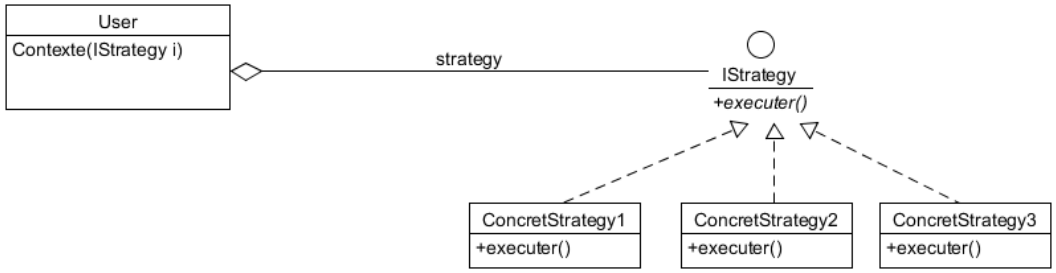


La classe ObserverXXX implémente la méthode update de l'interface Observer qui est appelée par ObservableXXX

1	OUI	
2	NON	**X**

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

Q 22.

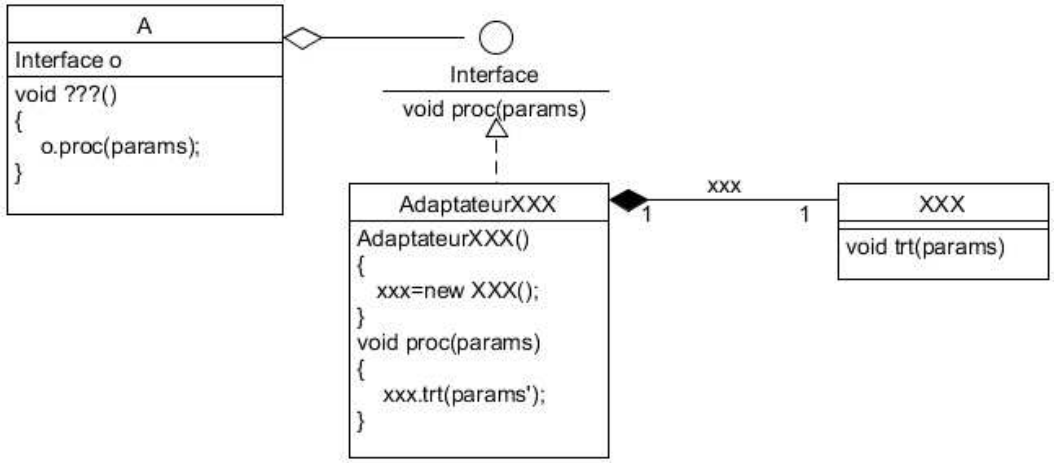


Ce schéma est le diagramme du DP Stratégie.

1	OUI	**X**
2	NON	

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

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	**X**
2	NON	

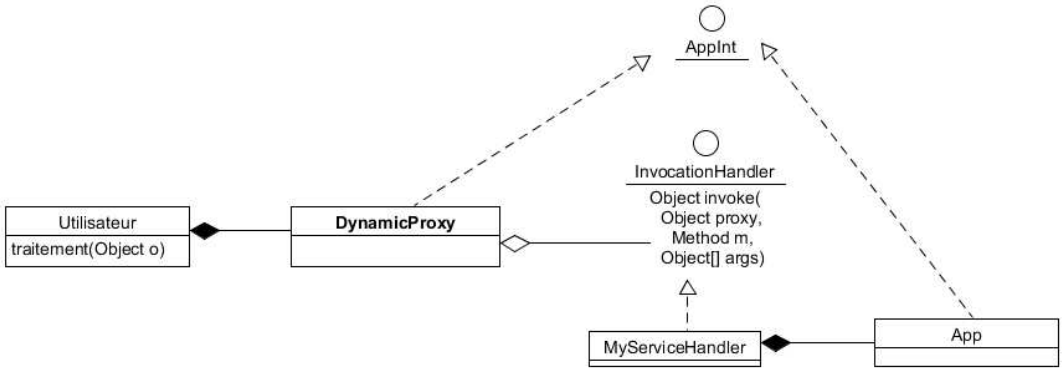
Dans le DP MVC, le Modèle utilise le DP Observateur pour :		Q 25.
1	recevoir les actions du contrôleur de mise à jour du Modèle	
2	envoyer aux vues les notifications de changement des états du Modèle.	**X**

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	**X**

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



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	**X**

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.	**X**

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.

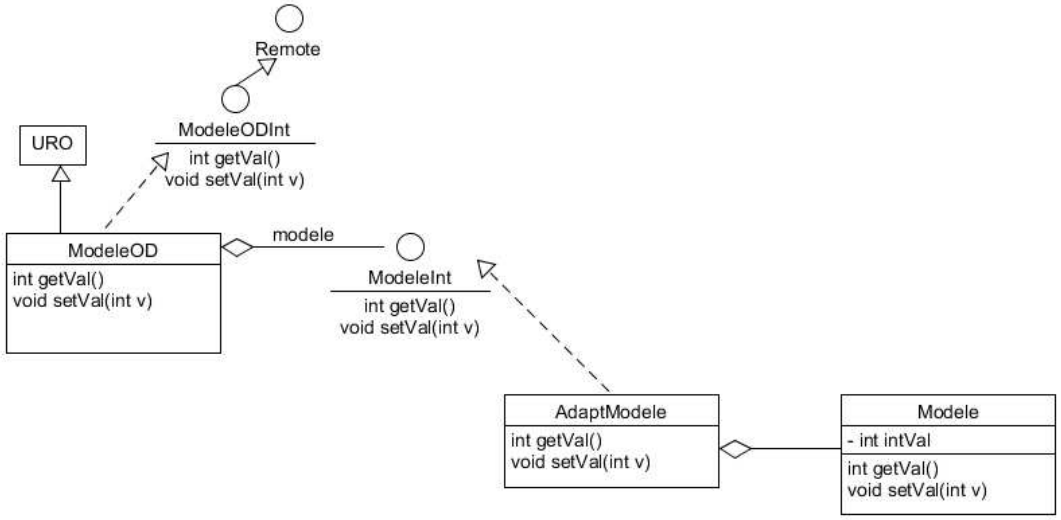
1	OUI	**X**
2	NON	

Dans le DP Observateur de base (de Java), le mode de communication entre les Observers et l'Observable est :

Q 33.

1	synchrone	**X**
2	asynchrone	

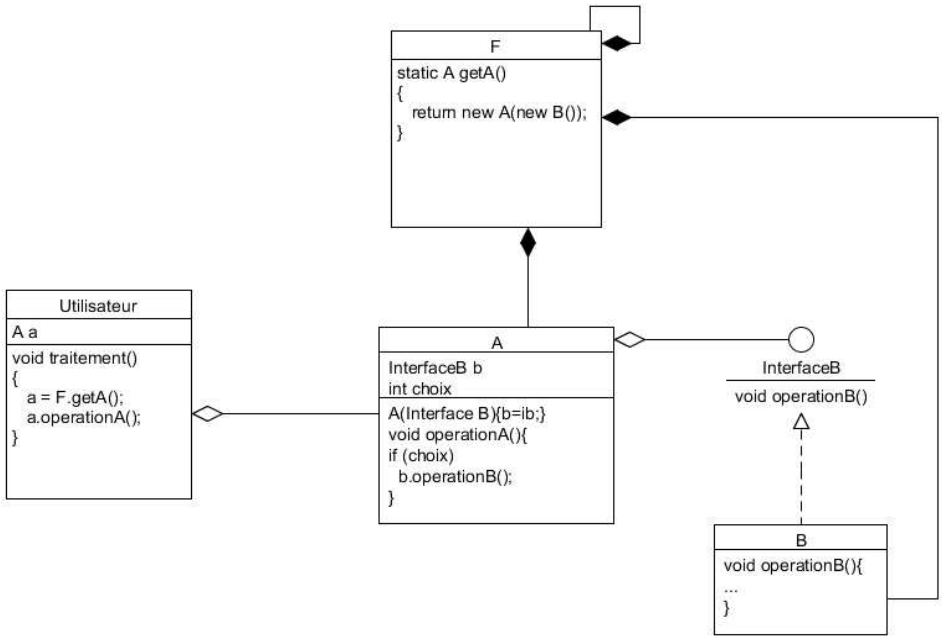
Q 34.



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

1	1 Adaptateur	
2	2 Adaptateurs	**X**
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	**X**
3	une injection de dépendance par l'utilisation d'un proxy	

Fin du QCM

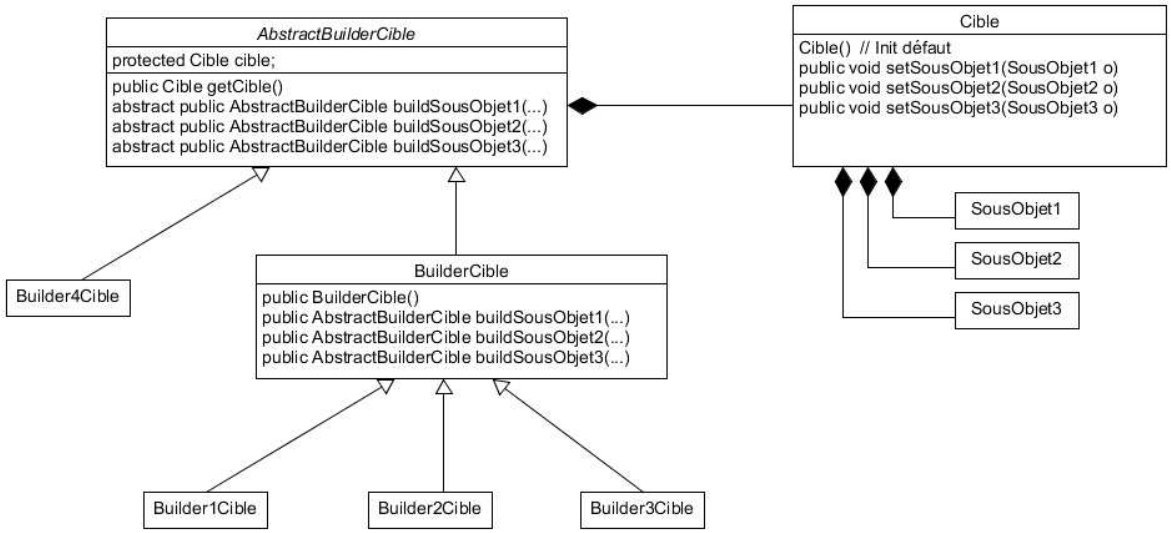
Suite (Tournez la page)

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**. **[3 points]**
Expliquez le comportement de ce design pattern. **[2 points]**
Correction :
Le schéma est le suivant :



Les classes concrètes de builder (Builder1Cible, Builder2Cible, …) permettent de créer un objet cible par héritage de la classe AbstractBuilderCible sans connaître pour celui qui utilise ces builders la façon dont l'objet a été créé par défaut. Ensuite, ces builders concrets permettent d'initialiser certains attributs de l'objet cible.
A tout moment ces builders permettent de retourner l'objet cible (méthode getCible).

QUESTION NUMERO 2

Expliquer le rôle du DP "Injection de dépendance". **[3 points]**
Citez, avec précision, 2 exemples pratiques de l'utilisation de ce DP. **[2 x 1 point]**
Correction :
Le rôle du DP "Injection de dépendance" est de déléguer à une autre classe (l'injecteur) de réaliser l'Inversion de Contrôle (Ioc) pour faire la création de la dépendance entre deux objets.
Premier exemple : On crée un injecteur qui va lier une classe métier avec la classe DAO correspondante qui va réaliser l'écriture sur un support persistant des attributs de la classe métier. En fonction d'une certaine configuration, l'injecteur choisira d'utiliser la classe DAO qui écrit dans un fichier plat ou qui écrit dans une base de données, ou une classe DA0 spécifique en fonction du type de base de données utilisé.
Deuxième exemple : Soit une classe A qui doit utiliser une autre classe B. La classe B n'est pas finie d'être programmée. On utilise un injecteur de dépendance pour lier la classe A avec une classe B_Sim qui va simuler à minima le fonctionnement de la classe B afin de pouvoir quand même avancer dans le test d'intégration du programme.

QUESTION NUMERO 3

Expliquez, en terme de Design Pattern (rare l'on fait (+2points si c'est juste ou +1point partiellement juste)=>l'explication suffit pour avoir 5 points) , la différence entre les deux modes de communication "push" et "pull", réalisé entre un Producteur et ses Consommateurs.

Correction :
En mode "push", le producteur va utiliser un DP Observateur synchrone ou asynchrone pour pousse son produit vers les consommateurs qui se sont au préalablement déclarés auprès de l'Observable du DP Observateur. Les consommateurs ont implémenté l'interface Observer utilisé par l'Observable.
En mode "pull", chaque consommateur va utiliser un Proxy du Producteur afin de récupérer un élément qui a été produit par le Producteur.

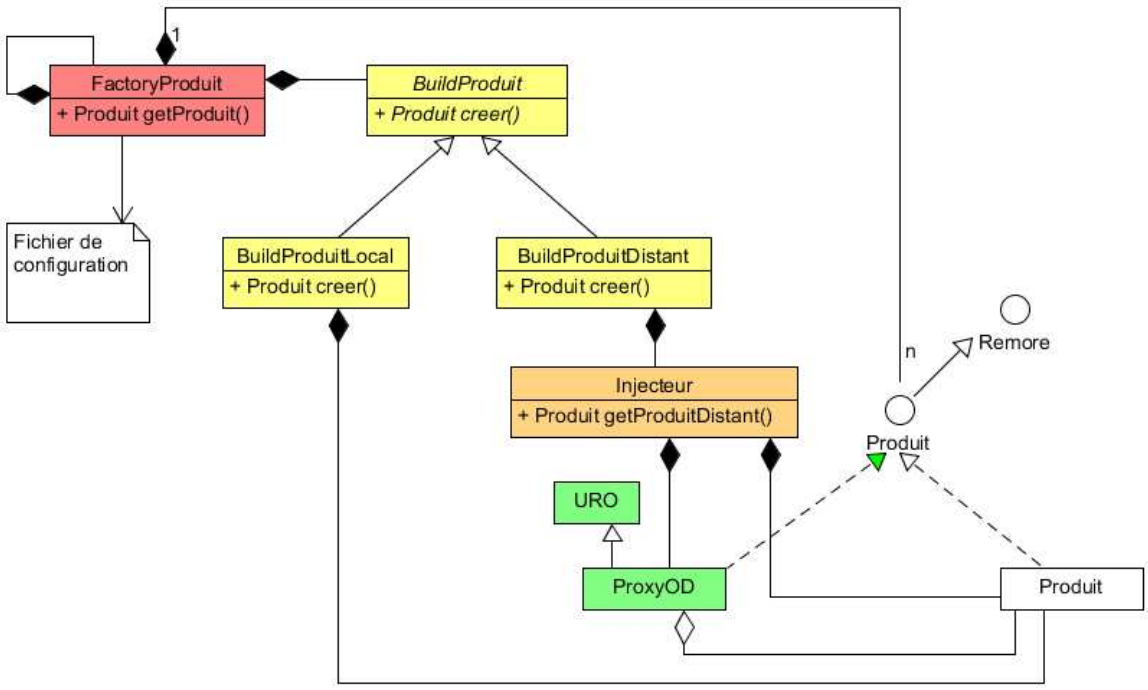
Dans les deux cas, les Consommateurs et les Producteurs ne sont pas liés directement.

Fin de la 1ère partie sans document

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

3. EXERCICE [15 points]

Le diagramme est le suivant : **[10 points]**



[5 points]

En fonction du fichier de configuration le FactoryProduit (DP Factory+Singleton en rouge) crée un BuildProduitLocal ou un BuildProduitDistant (DP de Stratégie en jaune).
Le BuildProduitLocal crée un Produit (cas d'une configuration locale)
Le BuildProduitDistant (cas d'une configuration distante) utilise un Injecteur qui va créer et lier entre eux un ProxyOD et le Produit créé (DP Proxy OD en vert).

L'avantage d'utiliser un Injecteur est que l'on pourra facilement changer de type de Proxy de communication en fonction de différents protocoles de communication.
L'avantage d'utiliser un Stratégie est que l'on pourra facilement créer de nouvelles façons différentes de créer un produit (produit persistant en base de données, produit appartenant à un MVC, …).

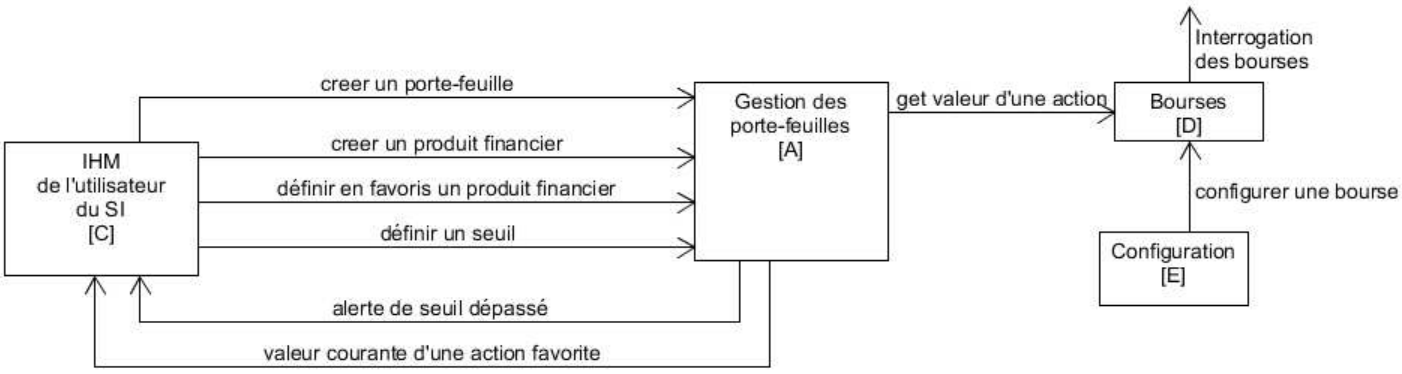
Le produit local et le produit distant sont vu sous la même interface qui hérite de Remote.

Remarque : Il est possible d'avoir deux interfaces différentes. Dans ce cas quand la méthode getProduit du factory est utilisé sur un client distant, il faut que le factory crée un adaptateur pour convertir l'interface distante en une interface locale.

Remarque : Certains on fait le choix de 2 factorys (un pour la création des objets en local, l'autre pour la création des objets en distants). Cela est tout à fait correct, à condition de les faire hériter d'une classe abstraite commune.

4. PROBLEME [35 points]

1/ Le diagramme de communication (ou comportement) de ce Système d'Information : **[10 points]**

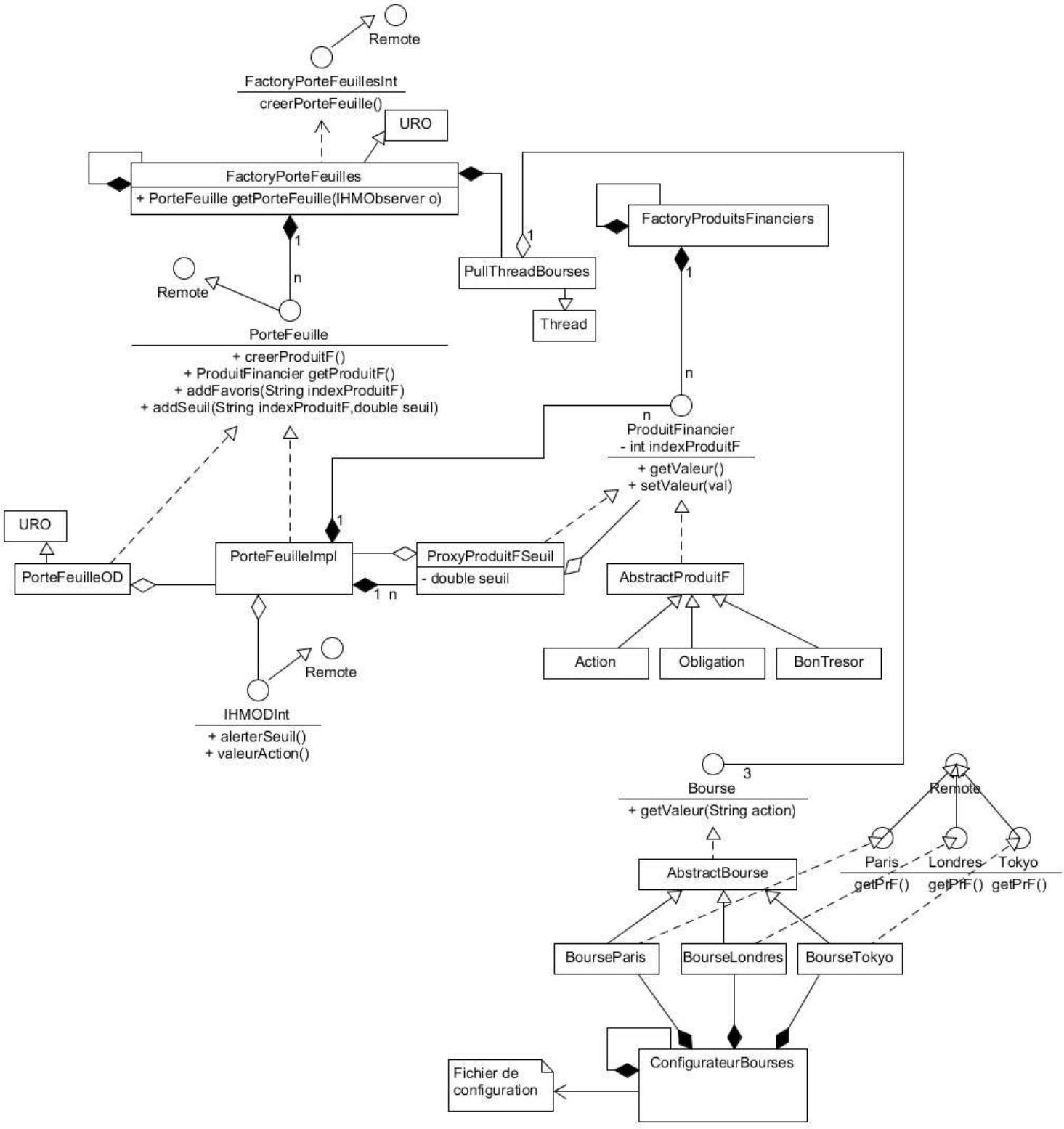


[4 points]

Commentaire :

Le composant [A] gère en mémoire les portefeuilles de tous les utilisateurs. Il tire régulièrement les nouvelles valeurs boursières du composant [D] afin de mettre à jour les actions des portefeuilles. Si un utilisateur est connecté à travers son IHM [C], il notifie l'IHM avec cette nouvelle valeur si l'action a été définie en favoris.
L'IHM d'un utilisateur [C] permet de faire toutes les actions sur le portefeuille de l'utilisateur. Si l'utilisateur définit un seuil sur une action alors le composant [A] notifiera une alerte à l'IHM quand la valeur de cette action dépassera le seuil.
Le composant [D] interroge toutes les bourses qui ont été configurées par [E] pour connaitre la valeur courante d'une action en particulier.

2/ Le diagramme de classe UML du [COMPOSANT 1] : **[13 points]**



[4 points]

Commentaire :

On crée 2 factorys. Un pour créer et gérer les portefeuilles **FactoryPorteFeuilles** et un autre pour créer et gérer tous les produits financiers **ProduitFinanciers.**

Un portefeuille d'un utilisateur **PorteFeuilleImpl** a un lien vers tous ses produits financiers.

Etant donné qu'il existe plusieurs types de produits financiers, on créer une classe abstraite **AbstractProduitF** dont héritent tous les types **Action Obligation** et **BonTresor**.

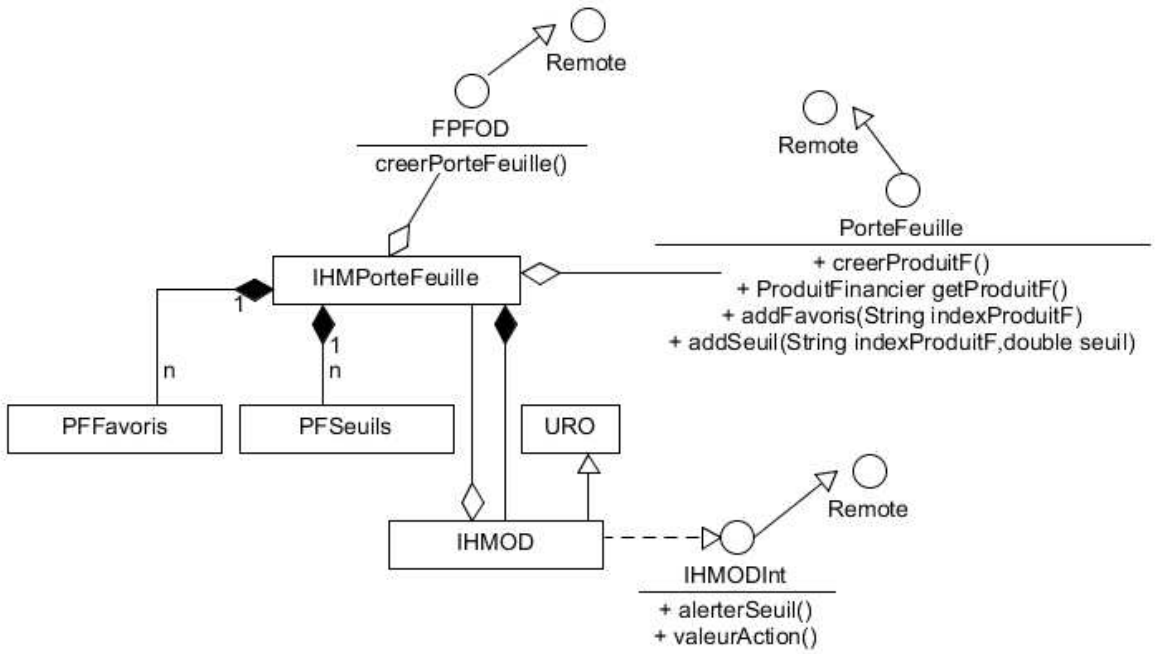
Le **FactoryPorteFeuilles** est un objet distant par héritage permettant de créer à distance un nouveau portefeuille. L'interface **FactoryPorteFeuillesInt** distante est utilisé par l'IHM.

Etant donné que chaque portefeuille peut être utilisé à distance par l'IHM d'un utilisateur, on crée un objet distant par proxy **PorteFeuilleOD** pour chaque portefeuille.
Le portefeuille d'un utilisateur communique avec son IHM à travers une interface distante **IHMODInt**. Cette interface permet d'envoyer à l'IHM une alerte de dépassement de seuil d'un produit financier et d'envoyer la nouvelle valeur d'une action favorite (*).
Pour détecter un seuil sur un produit financier, on crée un proxy ProxyProduitFSeuil sur le produit financier afin de redéfinir le setteur de la valeur qui préviendra l'ihm lors du dépassement du seuil.

Pour la gestion des Bourses, **PullThreadBourses** est un thread qui utilise 3 instances de bourse pour interroger régulièrement les valeurs boursières. La classe **ConfigurateurBourses** permet de créer les connexions distantes à chaque bourse. A la charge de chacune des classes concrètes de s'adapter aux connexions de chacune des bourses.

(*) On ne crée pas un model Observer/Observable Distant entre le portefeuille et l' IHM car l'IHM est unique.

Le diagramme de classe UML du [COMPOSANT 2] : **[3 points]**



[1 point]
Commentaire :
L'IHM d'un portefeuille **IHMPorteFeuille** d'un utilisateur utilise le portefeuille situé sur le serveur avec l'interface distante **PorteFeuille**. Elle définit des produits financiers favoris PFFavoris et des valeurs de seuil PFSeuils.
L'interface distante **FactoryPorteFeuillesInt** permet de créer un portefeuille sur le serveur.
La classe **IHMOD** est l'objet distant utilisé par le portefeuille situé sur le serveur pour recevoir les alertes et les valeurs d'action favorites.

Fin du sujet