IPST-CNAM
Intranet et Designs patterns
NSY 102
Mercredi 4 Avril 2018

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:

La "Démarche d'Architecture" est un guide méthodologique pour créer le dossier d'architecture d'un Système d'Information.		Q 1.
1	OUI	**X**
2	NON	

A l'issu de la Configuration Architecturale, on obtient la description d'un "réseau" de description du Système d'Information dont les nœuds sont les Composants et les liens les Connecteurs.		Q 2.
1	OUI	**X**
2	NON	

Les rôles fonctionnels d'un composant sont toujours des fonctionnalités décrites dans les exigences du besoin initial.		Q 3.
1	OUI	
2	NON	**X**

Une application dite "distribuée" est une application logicielle dans lequel les données informatiques sont toutes centralisées dans un composant unique appelé Singleton		Q 4.
1	OUI	
2	NON	**X**

En RMI de Java, la classe d'appartenance d'un objet distribué		Q 5.
1	hérite de UnicastRemoteObject et implémente une interface qui décrit les méthodes distantes et qui hérite de l'interface prédéfinie Remote.	**X**
2	hérite de RemoteObject et implémente l'interface Remote	
3	n'hérite d'aucune classe particulière et doit être envoyé, par sérialisation; à l'annuaire pour y être enregistré.	

Un DP définit des principes de conception, et non des implémentations spécifiques de ces principes		Q 6.
1	OUI	**X**
2	NON	

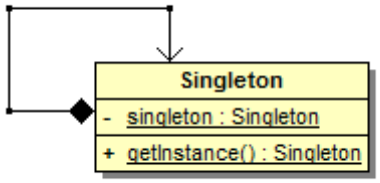
Un Design Pattern (DP) ou Patron de Conception est une norme de description des interfaces entre les composants d'une architecture logicielle orientée objet.		Q 7.
1	OUI	
2	NON	**X**

Le rôle du Design Pattern **Singleton** est la création unique d'une instance d'une classe.		Q 8.
1	OUI	**X**
2	NON	

Dans un système réparti, le DP Singleton est utilisé pour créer un objet distant unique sur le réseau.		Q 9.
1	OUI	
2	NON	**X**

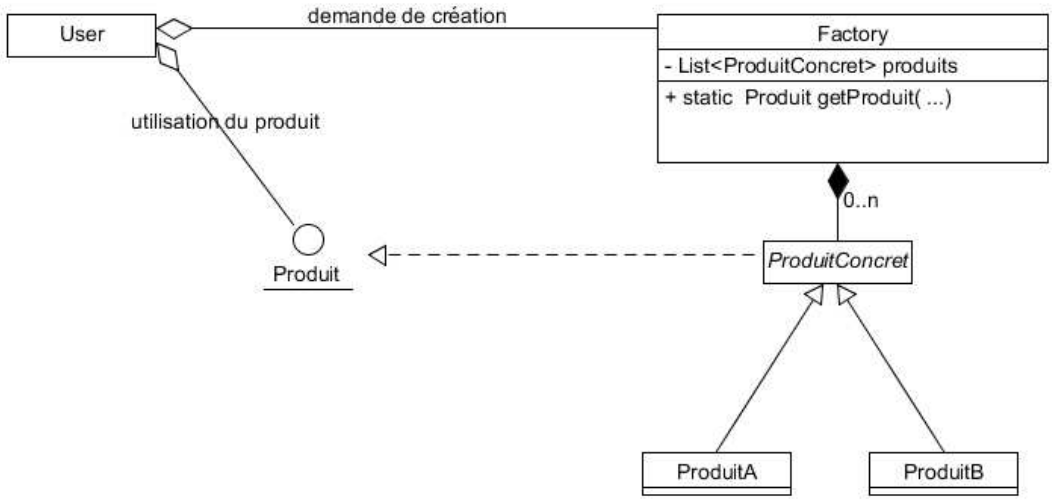
Le rôle du DP Factory qui est un Singleton est de rendre global à tout le programme, la création et l'utilisation de certains objets.		Q 10.
1	OUI	**X**
2	NON	

Un DP Singleton définit une classe particulière dont le rôle est de créer l'objet unique lors de sa première utilisation.		Q 11.
1	OUI	X
2	NON	

Le schéma suivant :  est celui du DesignPattern **Singleton**		Q 12.
1	OUI	X
2	NON	

Dans une application distribuée, un **factory** est un objet distant, enregistré dans un annuaire, et dont le rôle est de créer des objets distants qui sont utilisés par le client à travers un proxy client (appelé "stub")		Q 13.
1	OUI	X
2	NON	

Le rôle de la classe abstraite, dans un Factory, est de servir de proxy entre les classes concrètes d'implémentation des produits du factory.		Q 14.
1	OUI	
2	NON	X

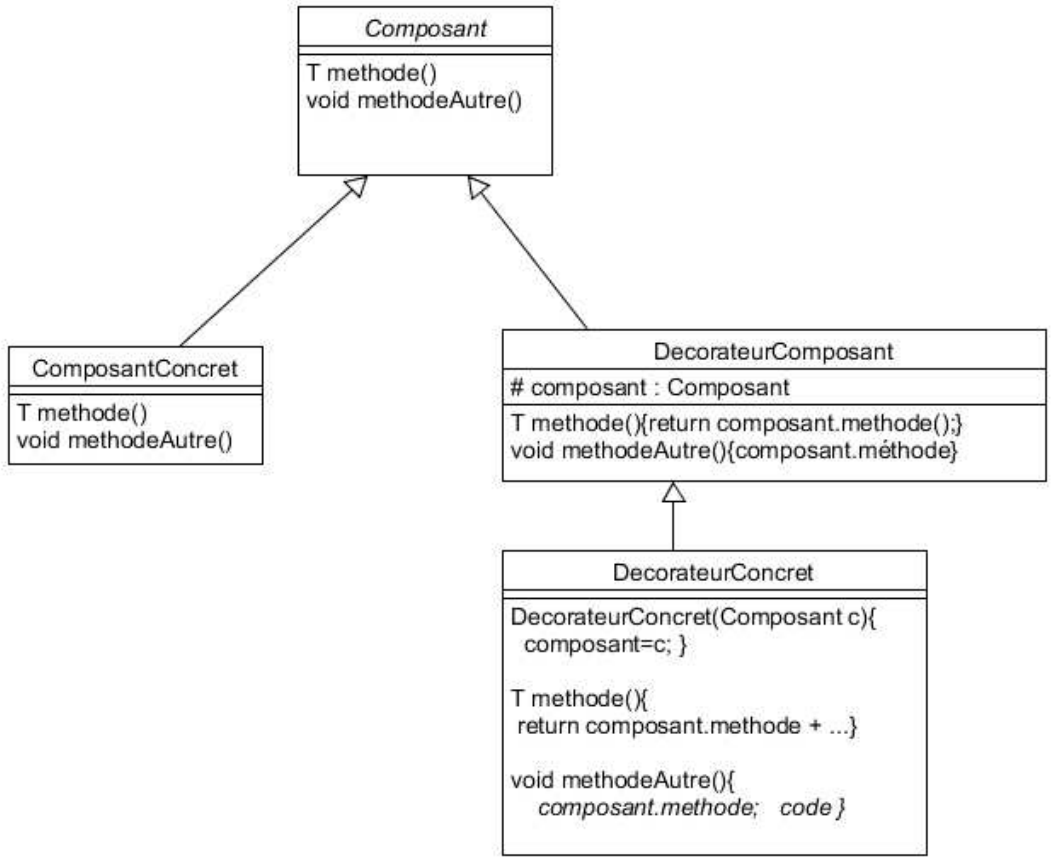
 Ce DP est celui du Factory. Le rôle de l'interface **Produit** est d'abstraire, à la classe User, la classe ProduitA ou ProduitB utilisé pour créer l'objet		Q 15.
1	OUI	X
1	NON	

Dans un DP Factory, le Client doit connaître les différents types de produits créés par le factory		Q 16.
1	OUI	
2	NON	X

Le DP Builder est un DP utilisé dans celui du Factory afin de construire le produit par assemblage d'autres classes.		Q 17.
1	OUI	X
2	NON	

Le DP "Délégation" est un Design Pattern qui appartient à la famille des Proxys.		Q 18.
1	OUI	
2	NON	X

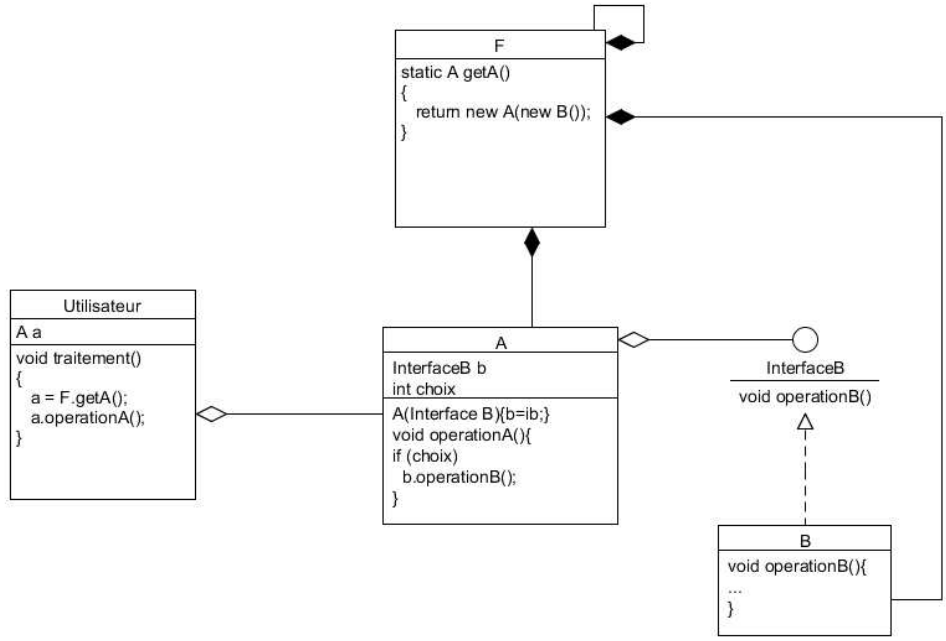
Q 19.



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

1	OUI	
2	NON	X

Le diagramme suivant :		Q 20.



représente une injection de dépendance par l'utilisation d'un proxy		
1	OUI	
2	NON	**X**

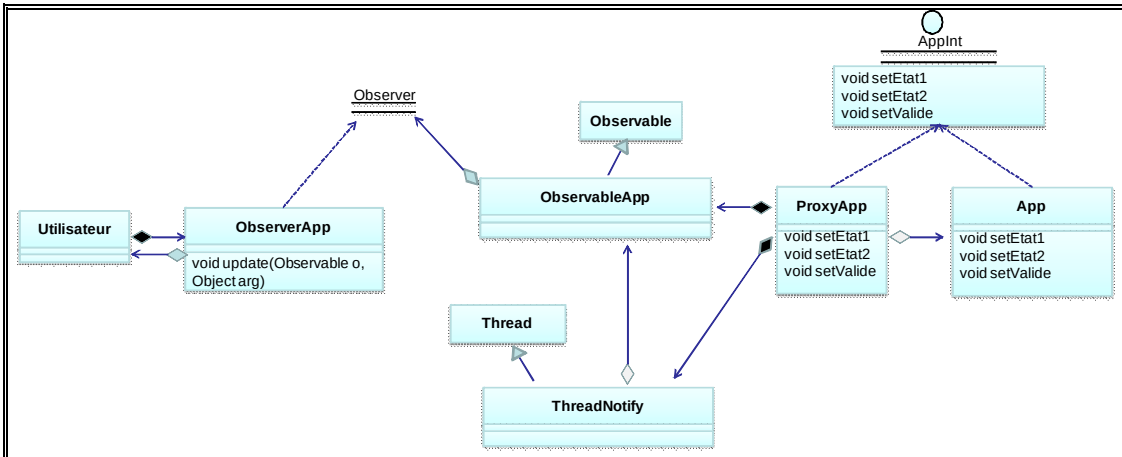
Le rôle du Design Pattern **Observateur** est de créer, dynamiquement des objets dont la classe d'appartenance implémente l'interface **Observer**		Q 21.
1	OUI	
2	NON	**X**

Le DP Observateur contient		Q 22.
1	une classe qui hérite de la classe Observer	
2	une classe qui hérite de la classe Observable	**X**
3	une classe Factory qui crée les évènements à notifier	

Toute classe qui implémente l'interface Observer peut devenir un observateur (Observer) de n'importe quel observé (Observable)		Q 23.
1	OUI	**X**
2	NON	

Le Design Pattern Observateur est utilisé dans :		Q 24.
1	le Design Pattern Factory	
2	le Design Pattern MVC (Model-Vue-Controleur)	**X**

Le DP Observeur/Observable, peut être utilisé, dans une architecture MOM, pour réaliser		Q 25.
1	le connecteur entre le Provider et les Consommateurs	**X**
2	le connecteur entre le Producteur et le Provider	



Ce schéma de Design Pattern est :		Q 26.
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	**X**

Le ClientProxy est un DP Proxy utilisé par un client, dans lequel les méthodes réelles ont été remplacées par un appel distant à ces méthodes.		Q 27.
1	OUI	**X**
2	NON	

On peut utiliser le Design Pattern Proxy pour rendre distant :		Q 28.
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 29.
1	OUI	**X**
2	NON	

A l'opposé de la communication synchrone, la communication asynchrone est un type de communication notamment basé sur le modèle du pull, comme par exemple un thread d'un client qui tire régulièrement les événements d'un serveur.		Q 30.
1	OUI	**X**
2	NON	

Dans la communication synchrone via un "canal d'évènement" entre un producteur et un consommateur, le producteur utilise un proxy de consommateur (et non les consommateurs directement), afin de lui pousser un évènement.		Q 31.
1	OUI	**X**
2	NON	

Laquelle des descriptions suivantes est un principe de communication synchrone :		Q 32.
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**

Le principe général d'un MOM (Model Orienté Message) est que tous les composants connectés au bus du MOM reçoivent tous les évènements publiés dans le fournisseur de service MOM (Provider) quels que soit les canaux d'évènement utilisés.		Q 33.
1	OUI	
2	NON	**X**

Dans une architecture MOM, le mode "Queue" assure que tous les consommateurs connectés au canal d'évènement de la queue d'évènement, reçoivent l'évènement publié par le Producteur.		Q 34.
1	OUI	
2	NON	**X**

Le Design:Pattern **DynamicProxy** est composé d'une classe qui implémente l'interface du proxy. Cette classe est :		Q 35.
1	une classe que l'on crée et qui implémente aussi l'interface InvocationHandler	
2	une classe créée dynamiquement par le framework ou le langage utilisé (ex Java)	**X**

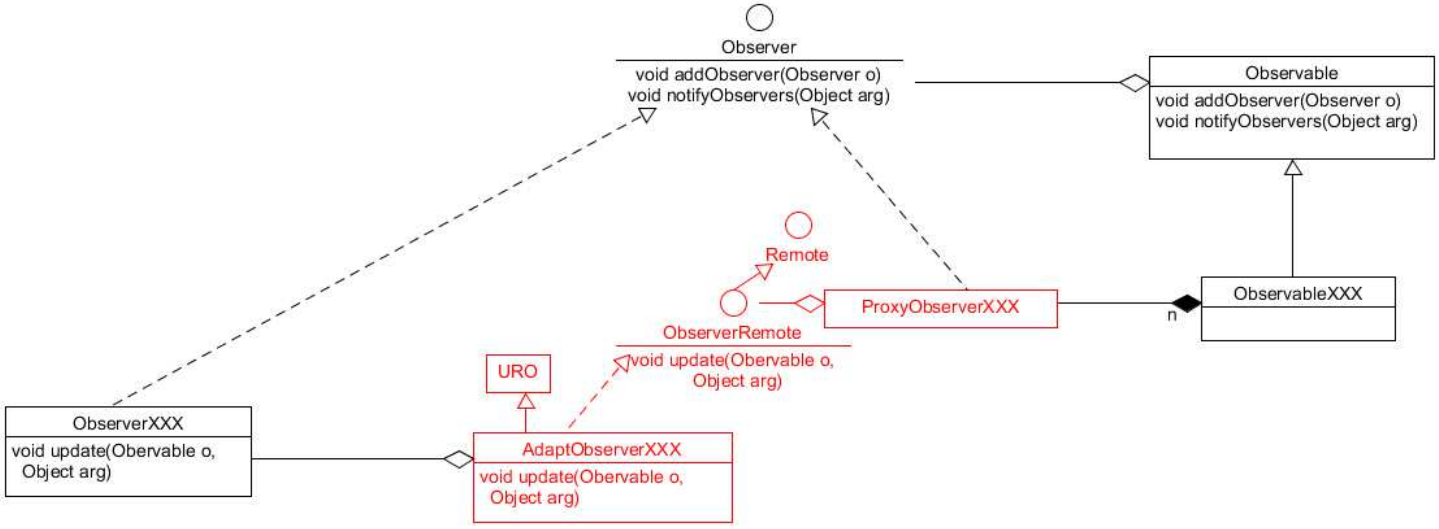
Fin du QCM

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



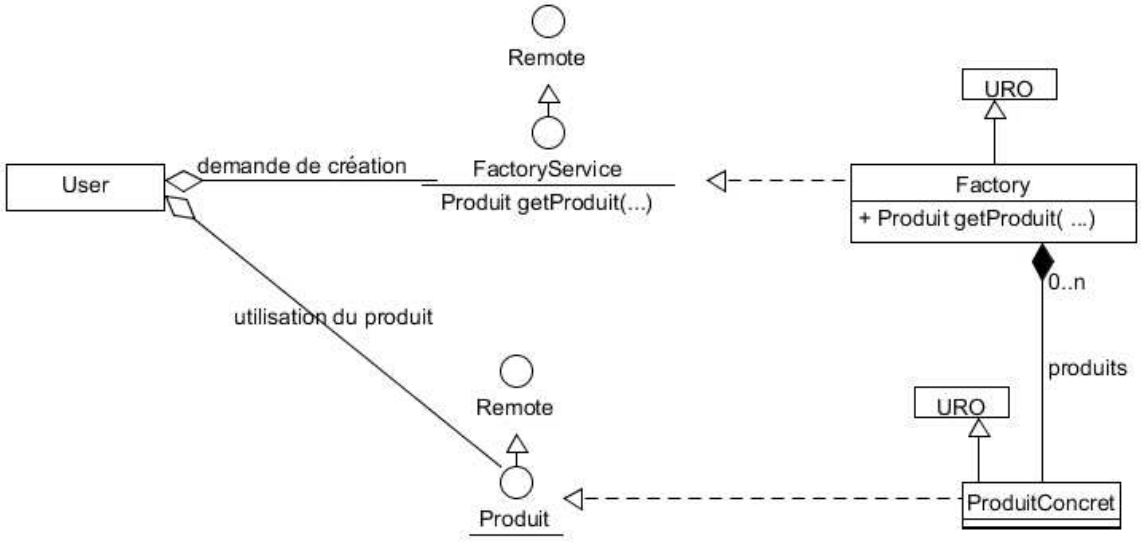
Le principe est de remplacer l'observer par un proxy d'un Observer (ProxyObserverXXX) qui aura pour but d'appeler la méthode **update** d'un objet distant (AdaptObserverXXX) qui sert d'adaptateur à l'observer (adaptateur de conversion de l'interface Observer en ObserverRemote).

QUESTION NUMERO 2

Comme son nom l'indique, le principe du Design Pattern **DynamicProxy** est de créer dynamiquement un Proxy d'interfaces données.
Le **Proxy** implémente, de lui-même, toutes les méthodes de toutes les interfaces dont le code de chaque méthode consiste d'appeler la méthode **invoke** décrite par l'interface **InvocationHandler**. A la charge du programmeur de créer la classe qui implémente cette interface.

La méthode **invoke** prend en entrée le nom de la méthode concernée et ses paramètres. A la charge du programmeur de créer le code en fonction de ces paramètres.

QUESTION NUMERO 3

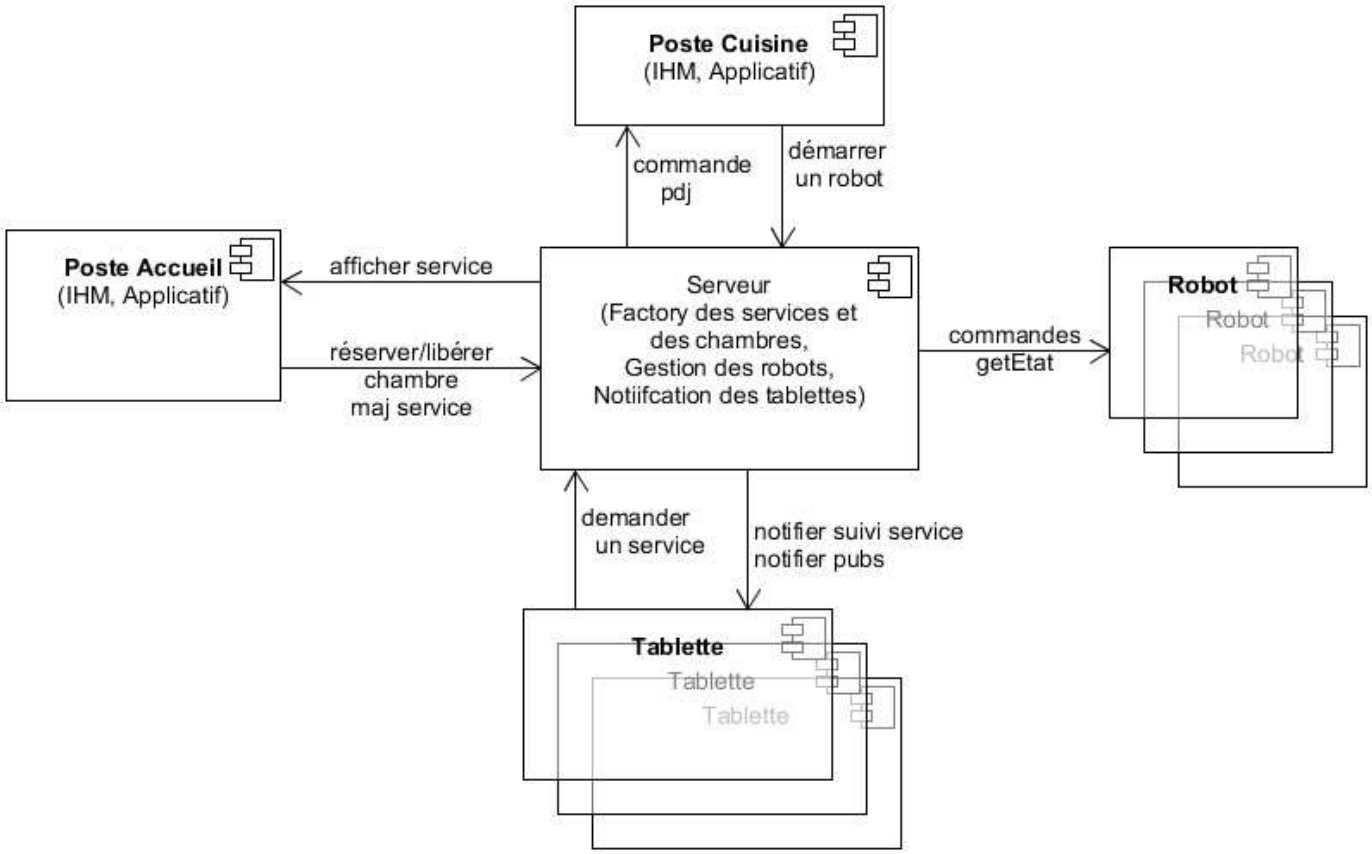


Fin de la 1ère partie sans document

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

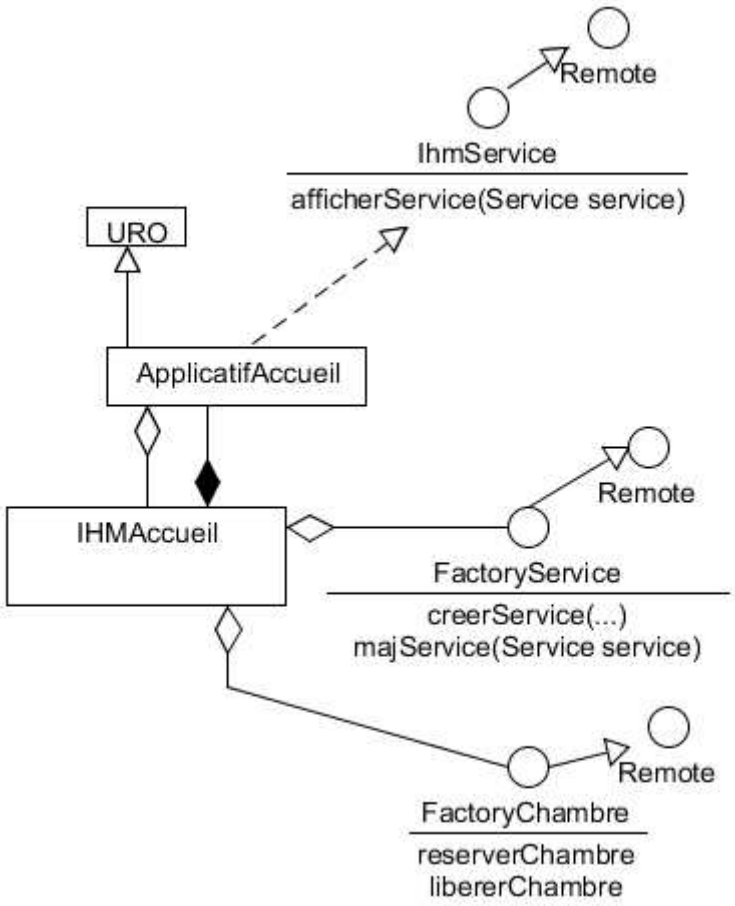
3. PROBLEME (50 points)

1/ Le schéma d'architecture logicielle



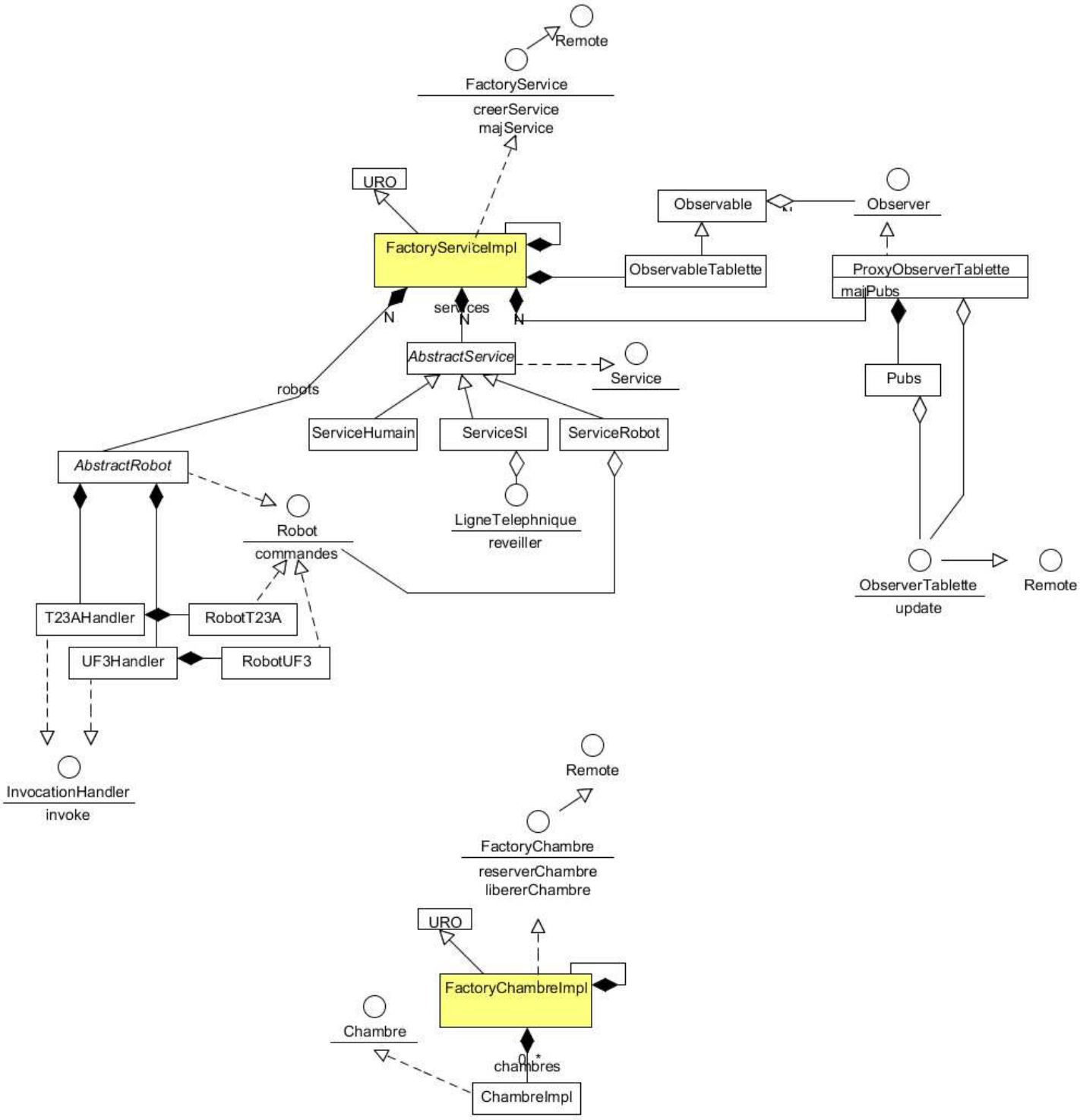
Le SI est constitué d'un **Serveur** qui centralise toutes les données dans des factorys et contient les traitements métiers. Il communique avec 5 composants : le **Poste d'Accueil**, le **Poste de Cuisine**, les **Tablettes** utilisées dans les chambres et les **Robots**.
Une tablette demande un service au serveur (interface distante). Le serveur crée un service. Il l'affiche sur l'IHM de l'accueil. Si le service est une commande d'un petit déjeuner, il envoie cette commande (interface distante) à l'IHM de la cuisine. Une fois le robot chargé, l'ihm de cuisine demande au serveur de démarrer le robot. C'est le serveur qui pilote le robot et connait son état.
Le serveur notifie (Observeur distant) l'ensemble des tablettes connectées du suivi d'une service et de publicités.
Le poste d'accueil met à jour la réservation des chambres (interface distante) et met à jour les services manuels (nettoyage).
N'est pas visible ici l'utilisation du réseau téléphonique par le serveur pour réaliser le service de réveil.

2/
Le diagramme de classe UML du [COMPOSANT 1]



Le poste d'accueil est composé d'une IHM et de son applicatif qui exporte en RMI une interface permettant d'afficher les services (utilisé par le serveur). Elle utilise les interfaces RMI distantes des factorys se trouvant sur le serveur : les services et les chambres.

Le diagramme de classe UML du [COMPOSANT 2]



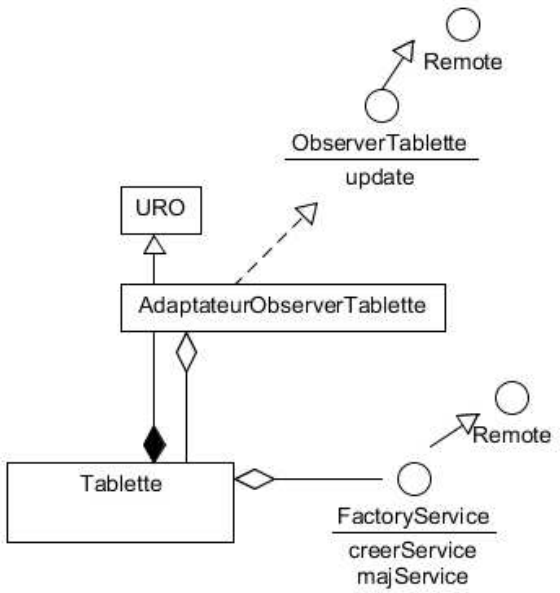
Un factory de gestion des services, 3 types de service différents (un service réalisé par un humain, un service réalisé par le serveur (informations), un service réalisé par un robot. Ce factory est un DP Factory Distant par héritage avec plusieurs produits vus toutes à travers une interface Service (non distante). Il permet de créer tous les services et de notifier les tablettes
Un factory de gestion des chambres : un DP Factory distant par héritage; pour gérer la réservation des chambres.

Un DP Observer/Observable distant (constitué d'un Adaptateur et d'un Proxy) afin de gérer un système de notification, par le serveur, de toutes les tablettes connectées. Les publicités à notifier sont gérées par le proxy dédié à chaque tablette (publicités ciblées).

Chaque robot est vu par le serveur à travers un proxy dynamique, permettant de s'adapter dynamiquement, à différents modèles de robot dont le InvocationHandler implémente les commandes à un modèle de robot particulier.

Un service du SI utilise la ligne téléphonique pour réaliser le réveil automatiquement.

Le diagramme de classe UML du [COMPOSANT 3]



La tablette est un Observer Distant du serveur afin d'être notifié par le serveur du suivi de la commande et des publicités.
La tablette utilise le factory de service à travers son interface distante pour faire la demande d'un service.

Fin du sujet