IPST-CNAM
Intranet et Designs patterns
NSY 102
Jeudi 29 Juin 2023

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

1ère Session NSY 102 **CORRECTION**

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).

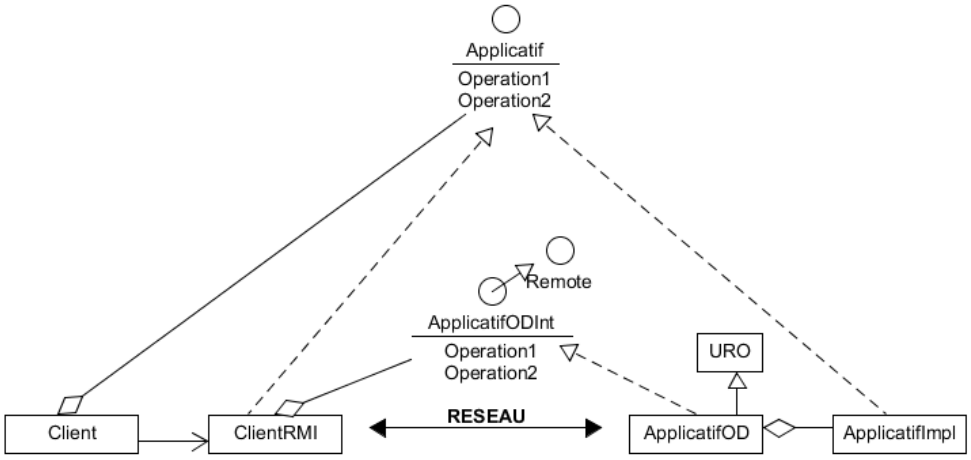
Si vous devez raturer une croix, faites-le correctement afin qu'il n'y ait aucune ambiguïté.

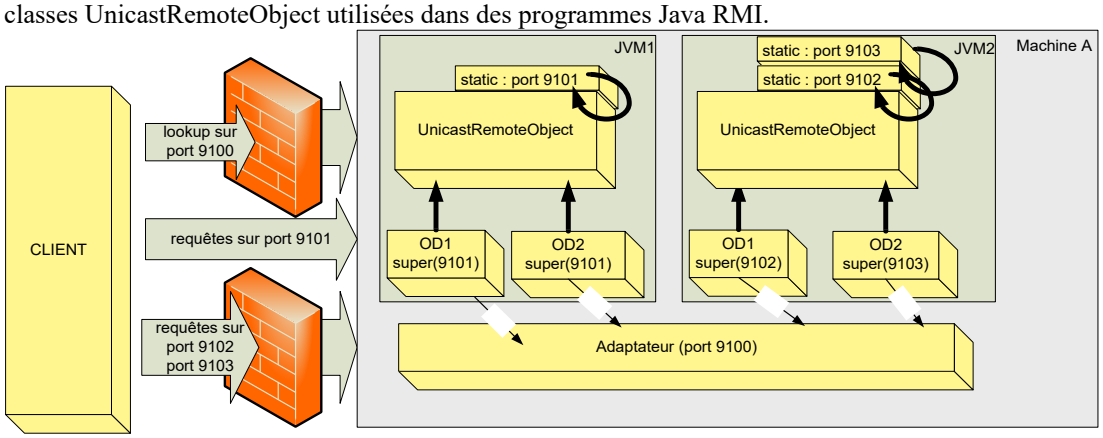
N'oubliez pas d'inscrire en en-tête du QCM, votre nom et prénom.

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

NOM:	PRENOM:

Soit un objet quelconque Obj (instance de la classe A qui hérite déjà d'une autre classe quelconque). En Java RMI, pour transformer cet objet en un objet distant, il suffit de :		Q 1.
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**

Ceci est le diagramme de classe d'un système composé d'un client et de son applicatif que l'on veut rendre distant.		Q 2.
		
1	Client est un proxy de ApplicatifImpl	
2	ApplicatifOD est un Adaptateur de ApplicatifImpl à ApplicatifODInt	**X**
3	ClientRmi est un proxy de ApplicatifImpl	**X**

Soit le schéma suivant qui représente un fonctionnement possible de plusieurs serveurs de socket des classes UnicastRemoteObject utilisées dans des programmes Java RMI.		Q 3.
		
1	On peut créer un nouvel OD 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 créée un nouvel objet distribué RMI sur le port 9102	X

En Java, la méthode **public static Remote toStub(Remote obj)** permet :		Q 4.
1	d'enregistrer l'objet distant 'obj' dans l'annuaire RMI	
2	de retourner un proxy de communication qui implémente l'interface de l'objet distant obj	X

Soit un objet, instance de la classe A. Pour transformer cet objet en un objet distant, il suffit que :		Q 5.
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

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 6.
1	OUI	
2	NON	X

Dans un système réparti, le DP Singleton est utilisé pour créer un objet distribué unique sur le réseau		Q 7.
1	OUI	
2	NON	X

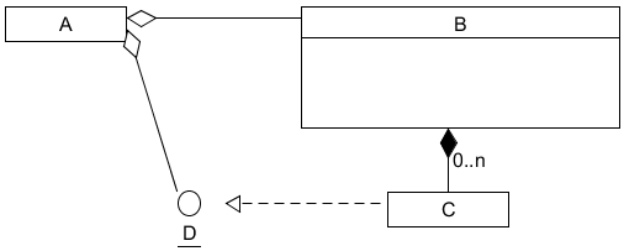
Il existe deux façons de créer un singleton de classe Singleton : - soit d'instancier le singleton dans la définition de la classe - soit d'instancier le singleton dans l'appel de la méthode *Singleton getInstance()*		Q 8.
1	OUI	X
2	NON	

Il est toujours possible de créer des setteurs sur un singleton permettant de modifier des attributs du singleton.		Q 9.
1	OUI	X
2	NON	

Le DP Factory a pour fonction la création d'objet dont les classes héritent d'une même classe abstraite et/ou implémentent la même interface		Q 10.
1	OUI	**X**
2	NON	

Le rôle du DP Factory est de créer des objets qui sont vus par le reste du programme comme des singletons :		Q 11.
1	OUI	
2	NON	**X**

Ce DP est celui du Factory. Q 12.



La signification des lettres A, B, C et D est :		
1	A=Factory; B = Concrete Product; C=Product (Interface); D=Client	
2	A=Client; B=Factory; C=Concrete Product; D=Product (interface);	**X**
3	A = Client; B=Product (interface); C=Concrete Product; D = Factory	

Le DP Factory mémorise toujours les produits créés dans une collection.		Q 13.
1	OUI	
2	NON	**X**

Le DP "Délégation" est utiliser dans le DP "injection de dépendance"		Q 14.
1	OUI	**X**
2	NON	

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.		Q 15.
1	OUI	**X**
2	NON	

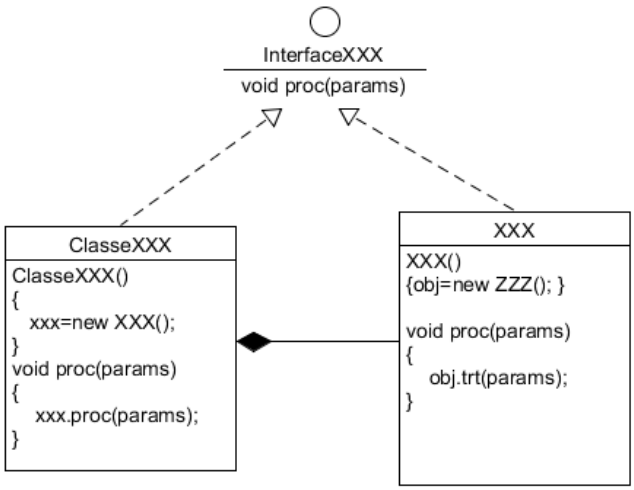
Dans le DP Observateur, la communication entre l'Observer (consommateur d'évènement) et l'Observable (producteur d'évènement) est nécessairement asynchrone car la communication se fait toujours par l'envoi d'un objet sans valeur de retour.		Q 16.
1	OUI	
2	NON	**X**

Dans le DP Observateur, la communication entre l'objet observé (producteur d'évènement) et l'observateur (consommateur d'évènement) est :		Q 17.
1	synchrone ou asynchrone (choix de conception)	**X**
2	toujours asynchrone	
3	toujours synchrone	

Le DP Observateur , peut être utilisé, dans une architecture MOM, pour réaliser		Q 18.
1	le connecteur entre le Provider et les Consommateurs	**X**
2	le connecteur entre le Producteur et le Provider	

Dans le DP Adaptateur, l'adaptateur et l'adapté implémente la même interface		Q 19.
1	OUI	
2	NON	**X**

Soit le diagramme de classe suivant : Q 20.

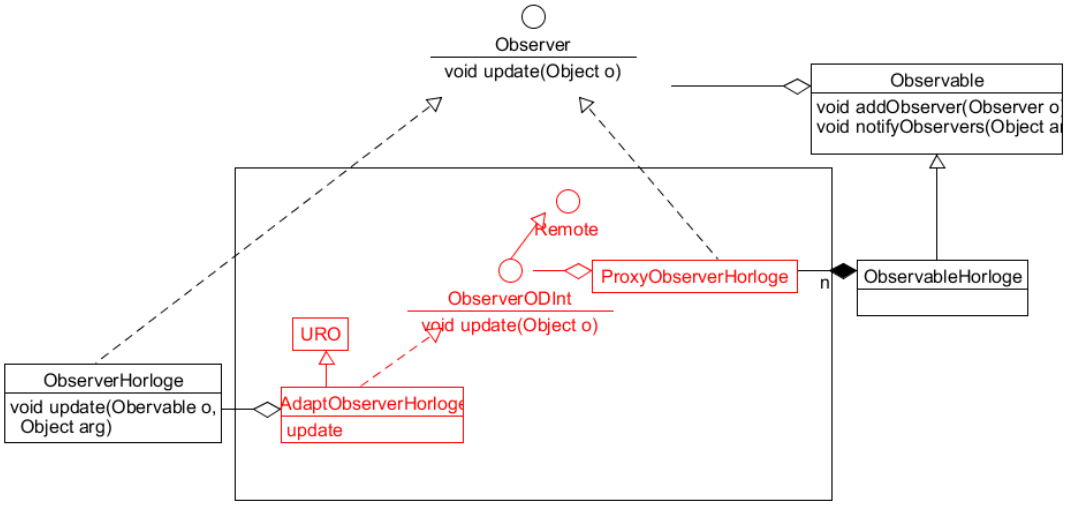


1	Ce diagramme de classe représente le DP Adaptateur	
2	Ce diagramme de classe représente le DP Proxy	**X**

Dans la communication synchrone, en mode push, 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 21.
1	OUI	**X**
2	NON	

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 22.
1	OUI	
2	NON	**X**

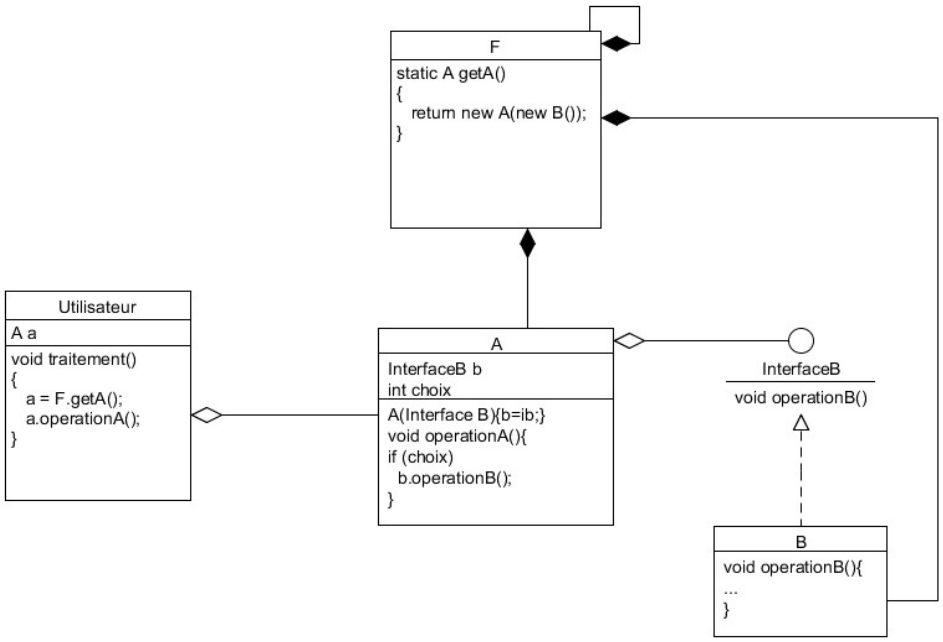
Soit le schéma suivant : Q 23.



Les classes et interfaces en rouge encadrées représentent :

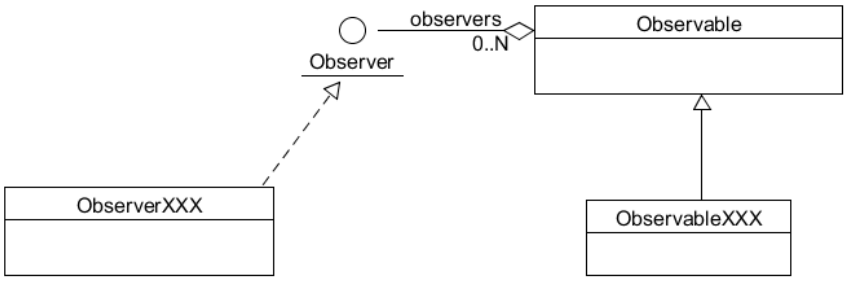
1	un proxy client de communication de ObserverHorloge vers ObservableHorloge	
2	un pont de communication permettant à un Observable (ObservableHorloge) de notifier les évènements à un Observer (ObserverHorloge) se trouvant dans une autre JVM.	X

Le diagramme suivant : Q 24.



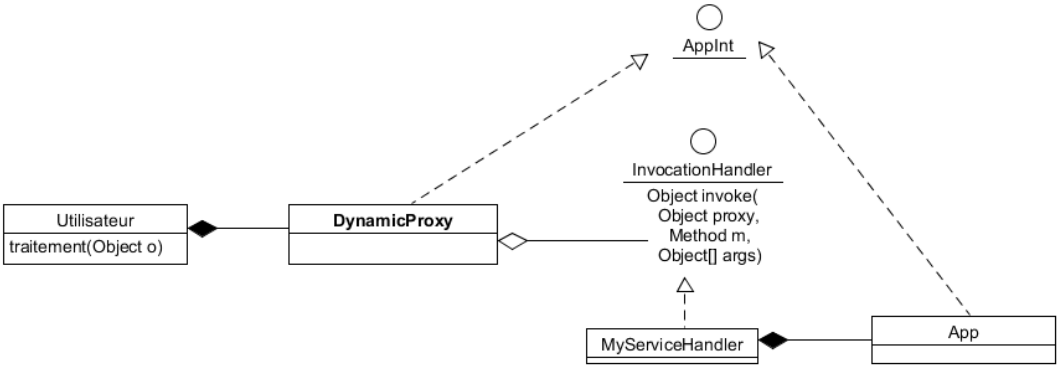
représente

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	

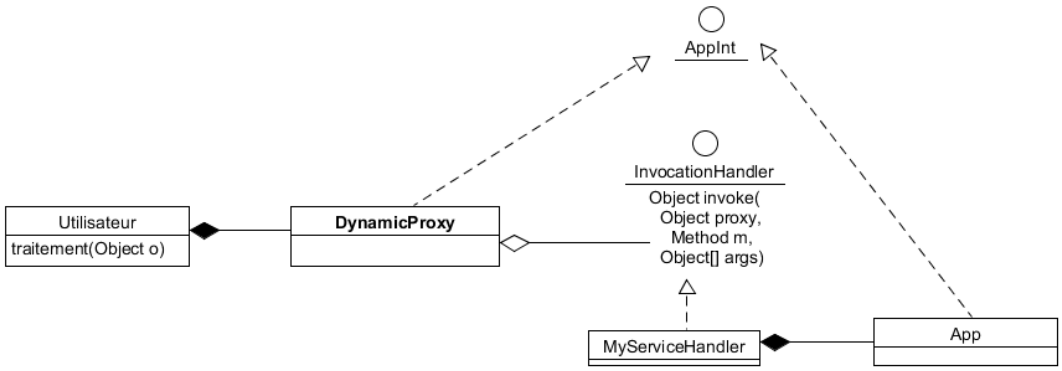
Soit le Design Pattern Observateur décrit (volontairement simplifié) de la manière suivante :		Q 25.
		
1	La classe ObservableXXX notifie les évènements à une instance de Observable	
2	La classe ObserverXXX implémente la méthode update de l'interface Observer qui est appelée par Observable	X
3	La classe Observable pousse (modèle du "push") les évènements à ObserverXXX	X

Laquelle des descriptions suivantes est un principe de communication synchrone :		Q 26.
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, à un même canal d'évènement du bus du MOM, en mode Topic, reçoivent tous les évènements publiés dans le fournisseur de service MOM (Provider)		Q 27.
1	OUI	X
2	NON	

		Q 28.
 Ce schéma est celui du DP Dynamic proxy. Le rôle de la classe MyServiceHandler est d'implémenter toutes les méthodes de l'interface AppInt.		
1	OUI	
2	NON	X

Q 29.

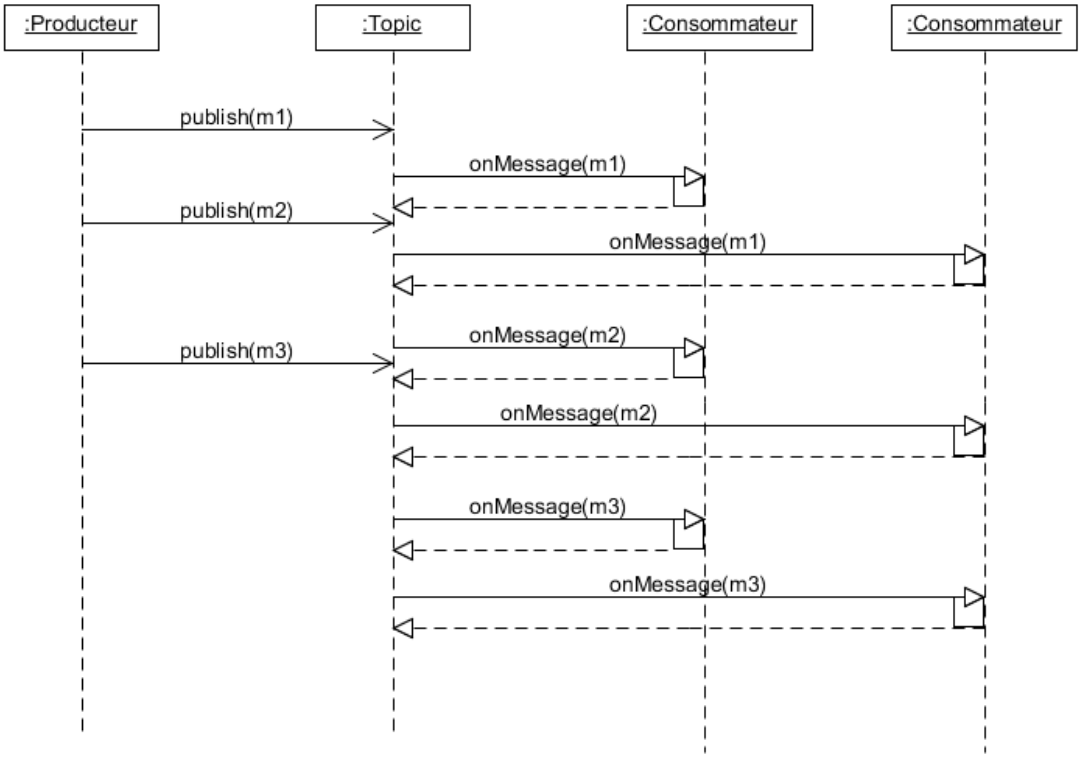


L'API RMI de Java utilise le DP DynamicProxy pour :

1	créer par l'Injection de Dépendance, la dépendance entre la classe MyServiceHandler et App.	
2	créer le proxy client dans la méthode loookup, permettant la communication avec l'objet.	X

En JMS (Java Messaging System), il existe (notamment) deux modes de communication : Queue et Topic.

Q 30.



Ce diagramme de transition correspond au mode :

1	Queue	
2	Topic	X

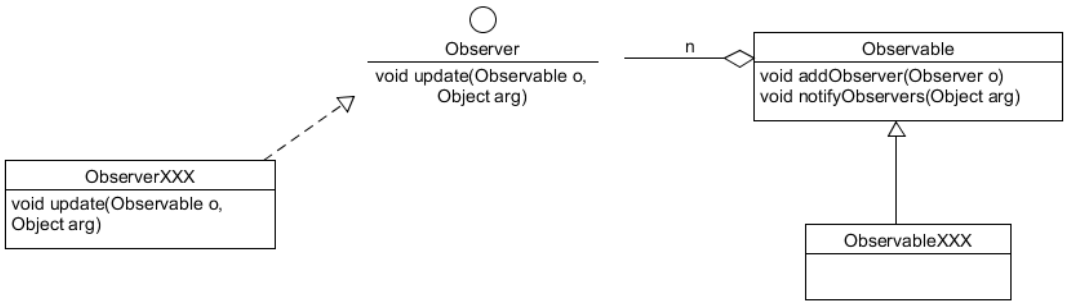
En JMS (Java Messaging System), les producteurs et les consommateurs sont tous des clients d'un composant logiciel appelé JMS Provider		Q 31.
1	OUI	X
2	NON	

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 32.
1	OUI	
2	NON	X

Les descriptions suivantes sont des modèles de communication asynchrones :		Q 33.
1	un serveur pousse ses évènements dans une file d'attente par client connecté (intermédiaire), et les clients tirent ses évènements à leur rythme	**X**
2	un serveur pousse son évènement dans un proxy de consommateur, et à son tour, le proxy de consommateur pousse l'évènement au consommateur. Chaque proxy est en attente du retour.	
3	un serveur appelle la méthode distante d'un client afin de lui pousser l'évènement, et n'attends pas le retour.	**X**

Le modèle de communication "Pull synchrone" est réalisé avec le DP Observateur (Observer/Observable)		Q 34.
1	OUI	
2	NON	**X**

Soit le Design Pattern Observateur : Q 35.



Ce DP est

1	de type "pull"	
2	de type "push synchrone"	**X**
3	de type "push asynchrone"	

Fin du QCM

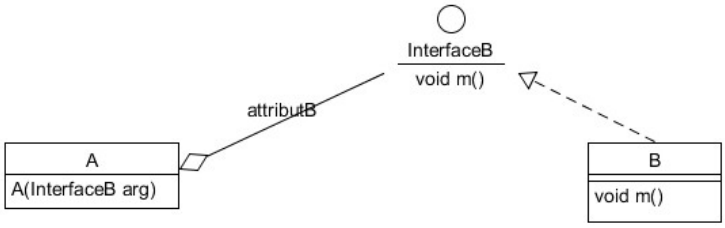
Suite (Tournez la page)

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.

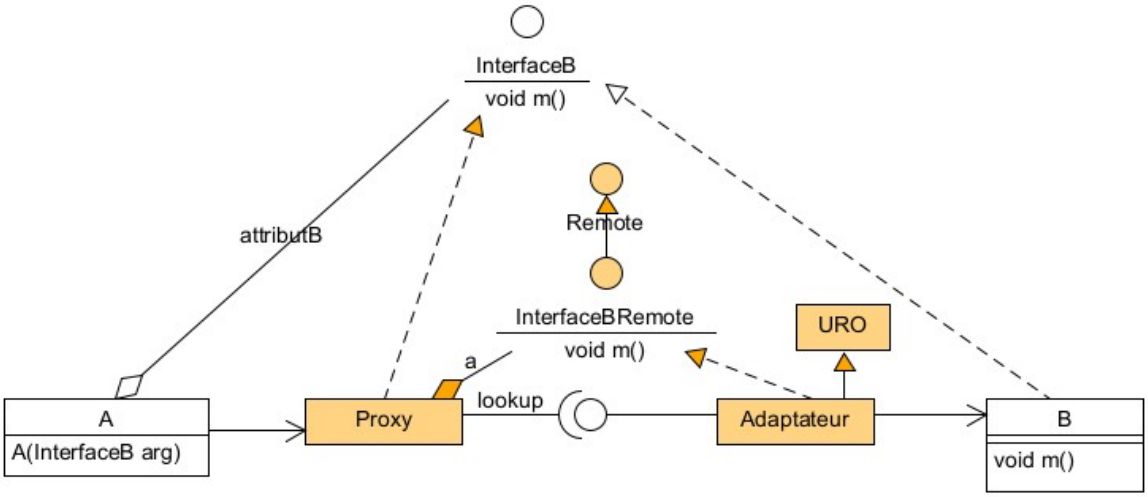
QUESTION NUMERO 1

Soit 2 classes A et B dont A utilise B à travers une interface :



Écrire le Diagramme de classe qui permet de rendre distant la dépendance entre A et B en utilisant la technologie RMI de Java, sans modifier les classes A et B.

Le diagramme est le suivant :

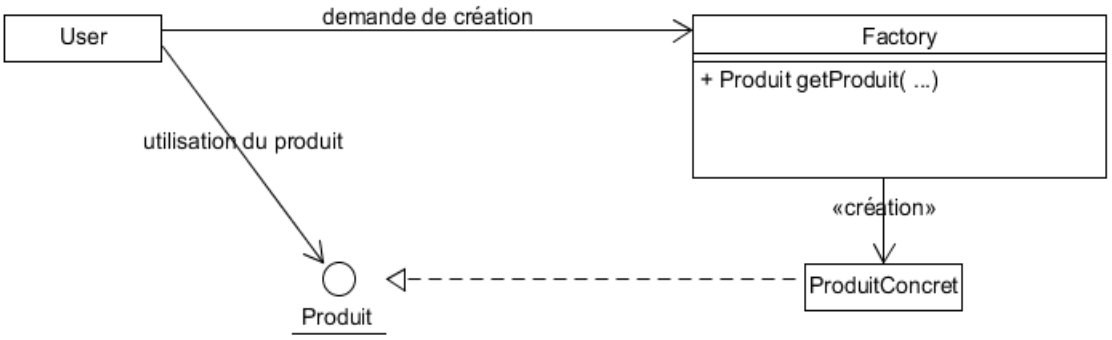


QUESTION NUMERO 2

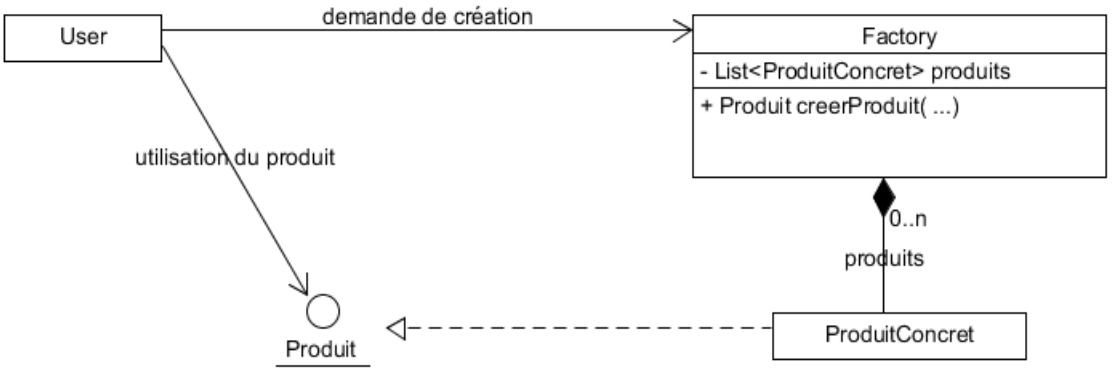
Quel est le principe fondamental du Design Pattern : Factory ?
Citez au moins 2 variantes de DP de ce DP, en faisant un schéma UML pour chacun d'eux. Expliquez succinctement.

Le principe fondamental d'un Factory est de rendre abstrait le processus de création d'un objet pour une classe utilisatrice. L'objet créé est vu par l'utilisateur du factory sous la forme d'une interface ou d'une classe abstraite.

Variante 1 : un factory qui crée un objet à chaque fois

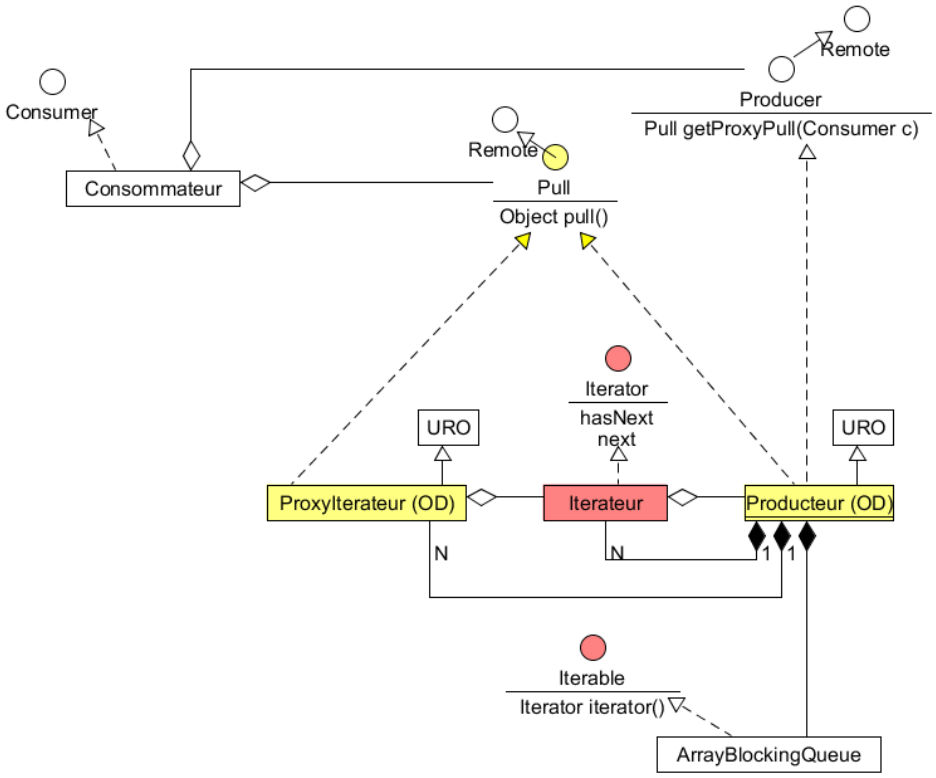


Variante 2 : un factory dans lequel les objets créés sont stockés dans une collection connue que du factory.



QUESTION NUMERO 3

Soit le Design Patern Pull Asynhcrone décrit comme ceci (vu en cours) :



Expliquez le fonctionnement de ce DP.

Ce DP répond à la problématique suivante :

On veut que la consommation des évènements par les consommateurs soit asynchrone par rapport à leurs productions et nous voulons également que les consommateurs tirent, indépendamment de leurs rythmes, tous les évènements produits. Sachant que le producteur gère 1 seul groupe physique d'évènements.

Chaque consommateur utilise la méthode Pull getProxyPull(Consumer c) pour obtenir le stub d'un nouveau ProxyIterateur qui implémente la méthode pull. Le consommateur peut ensuite appeler, à son rythme, la méthode next de l'itérateur pour tire un évènement.

Cette méthode crée à chaque appel l'instance de ProxyIterateur qui crée un nouvel itérateur (en appelant la méthode iterator de ArrayBlockingQueue).

Chaque consommateur a donc son propre itérateur.

De son côté, à son prpre rythme, le Producteur crée les évènements dans ArrayBlockingQueue.

Fin de la 1ère partie sans document

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

3. PROBLEME [50 points]

Nous envisageons de réaliser un Système d'Information (SI) qui réalise des chaînes de traitements d'images satellites.

Ce SI est composé de 4 COMPOSANTS :

- COMPOSANT 1, appelle Récepteur, qui reçoit les images brutes provenant de différentes Stations de Réceptions d'Image, vérifie leurs qualités brutes, et les envoie au Serveur (COMPOSANT 2) dans le format TIFF (Tagged Image File Format).
- COMPOSANT 2, appelé Serveur, qui gère les images brutes reçues et réalise les traitements d'images demandées par un opérateur.
- COMPOSANT 3, appelé Poste Opérateur, en nombre indéterminé, qui permet, notamment, à un opérateur, de demander au Serveur de réaliser une chaîne de traitements d'image sur une image brute.

- COMPOSANT 4, appelé Archivage qui permet d'archiver dans une base de données les images traités, et de rechercher ces images pour pouvoir les retraitées.

Quand le Serveur reçoit une image brute, il crée un **Produit d'Image** qui contient :
- le numéro d'identification (généré par le serveur)
- l'image brute (celle envoyée par le Récepteur)
- la liste ordonnées des différentes images résultats de l'application de chacun des traitements d'image successifs (vide par défaut).
- l'état courant du Produit d'Image : « non traité » (par défaut), « en cours », « traité ».

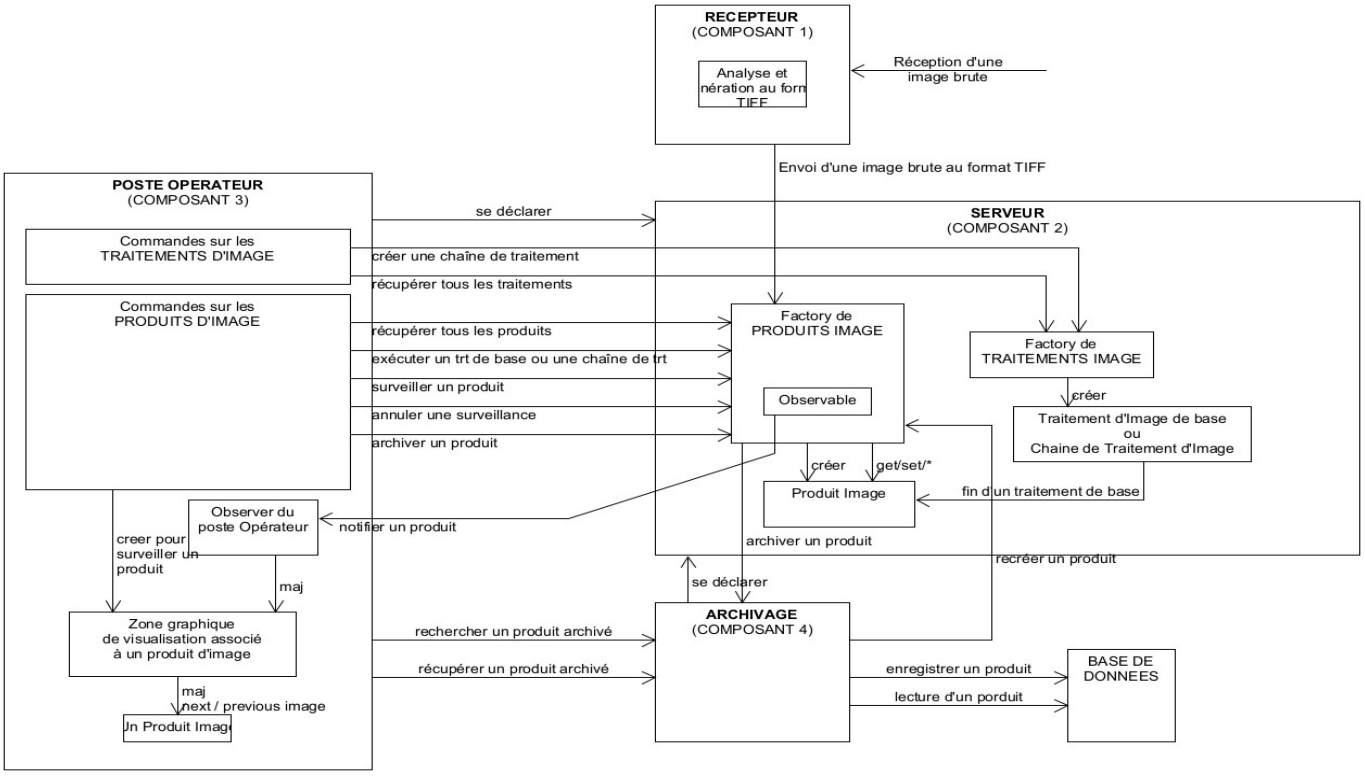
Par défaut, le Serveur contient tout un ensemble de traitements d'image de base qu'il est possible d'appliquer sur une image (corrections géométriques, corrections de luminosité, de contraste, effets de seuils, filtres radiométriques, fusion d'images, classifications d'image, …).
Le serveur permet de créer des chaînes de traitements constitués de traitements de base, ou d'autres chaînes de traitement.

Le Poste Opérateur permet de:
- créer une chaîne de traitement, sur le serveur, en précisant la valeur des paramètres de chacun des traitements de base.
- afficher la liste de tous les Produits d'Image connus du Serveur
- sélectionner un Produit d'Image « non traité » ou « traité » pour demander au serveur de réaliser l'exécution d'un traitement de base ou d'une chaîne de traitement.
- sélectionner un Produit d'Image :
  - si « en cours », pour surveiller le déroulement des traitements sur ce produit dans une zone graphique : à chaque fois que le Serveur a fini d'exécuter un traitement de base, il notifie tous les Postes Opérateur qui ont fait la demande de surveillance du Produit d'Image.
  - Si « traité » pour afficher les résultats du Produit d'Image dans une zone graphique
- fermer une zone graphique qui annule la surveillance ou l'affichage du produit d'image associé.
- sélectionner un Produit d'Image « traité » pour en demander l'archivage, ce qui entraîne sa suppression sur le serveur.
- rechercher et choisir un Produit d'Image archivé qui entraîne sa création sur le Serveur à l'état « non traité ». Il est alors possible d'exécuter, à nouveau, une chaîne de traitement sur ce produit.

Sur un Poste Opérateur, chaque surveillance d'un Produit d'Image est réalisée dans une zone graphique qui affiche le résultat de la dernière exécution d'un traitement de base. Quand le Produit d'Image est dans l'état « traité », deux boutons, flèche gauche et flèche droite, permettent d'afficher les différentes images résultats.

Pour démarrer le SI, les composants suivants sont exécutés dans cet ordre : Serveur, Récepteur, Archivage. Ensuite les Postes Opérateur démarrent au grès de leurs utilisations.

1/ **[15 points]**
Faites le diagramme de communication de ce Système d'Information. Il y a donc 4 composants à décrire qui communiquent entre eux de manière distant. 10 points
Commentez votre schéma (rôles des composants et sous-composants, comportement dynamique général, échanges des informations, localisation des données).
Nous rappelons que ce schéma doit permettre de connaître vos choix d'organisation des composants, le sens des communications, et les designs patterns envisagés. 5 points



Le Serveur est composé de deux factory distants : le factory de gestion des Produits d'Image et le factory de gestion des Traitements d'Image.
Le factory de gestion des Produits d'Image crée les produits au fur et à mesure que le Récepteur lui envoie de nouvelles images à traiter.
Le factory de gestion des Traitements d'Image contient les traitements d'image de base et les chaînes de traitement créés par un opérateur.
Le factory de gestion des Produits d'Image contient un Observable dédié afin de notifier les Postes Opérateurs qui ont demandé la surveillance, au moins, d'un Produit d'Image. A la fin de l'exécution de chaque traitement de base, le produit image concerné est prévenu afin de demander au factory de notifier les postes opérateurs.

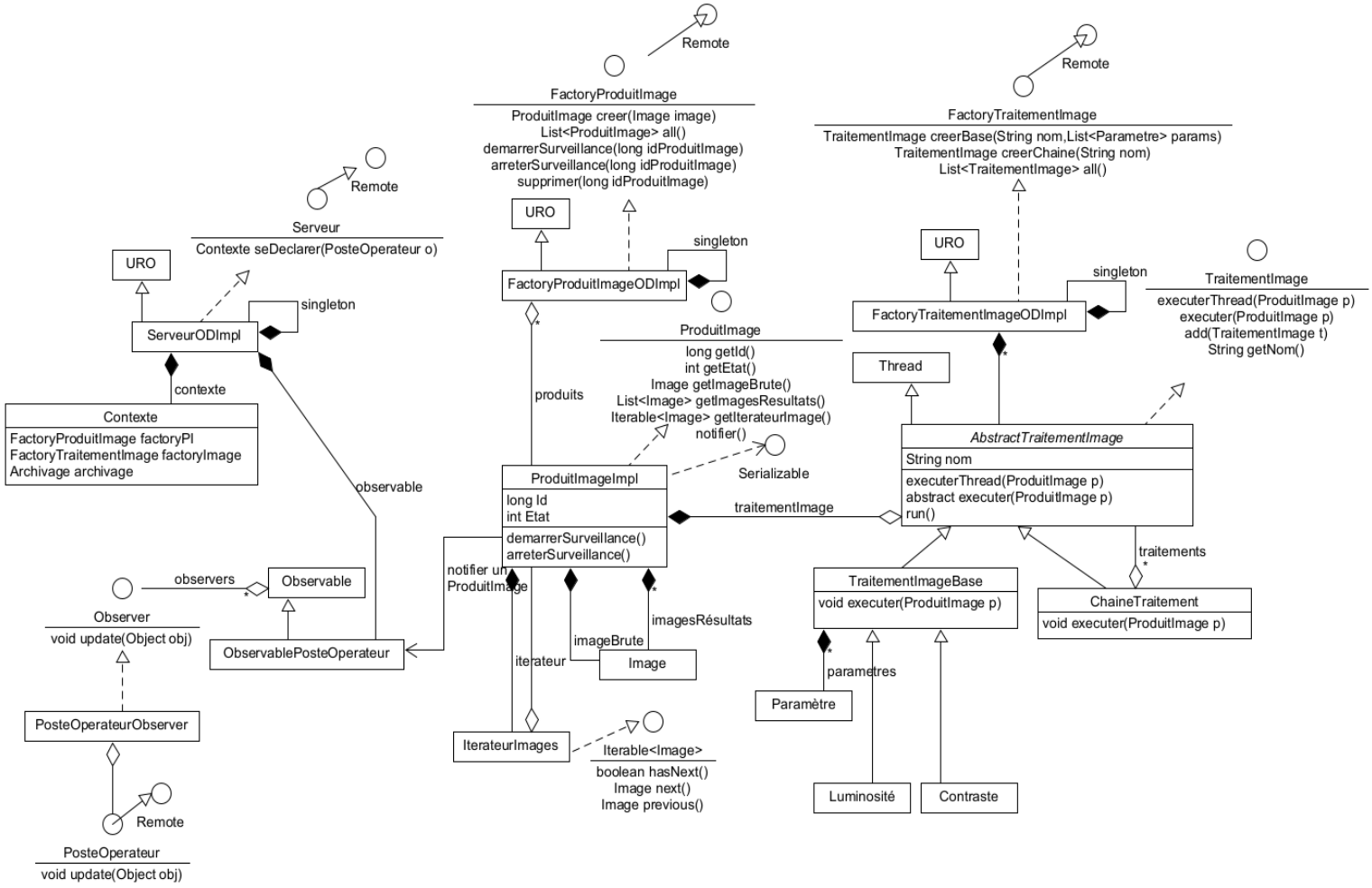
Le Serveur est un composant distant qui s'enregistre dans un annuaire.
Chaque poste opérateur se déclare au Serveur qui retourne les points d'accès aux deux factory et retourne aussi le point d'accés du composant Archivage qui s'est, au préalable, lui aussi, déclaré au Serveur.
Le Poste opérateur communique avec ces 3 points d'accès pour réaliser toutes les actions réalisées par un opérateur. Il crée un Observer afin de recevoir les notifications envoyées par le factory de Produit d'Image.
L'élément notifié est tout un Produit d'Image qui contient donc toutes les images résultats de l'exécution des traitements d'une chaîne de traitement.

2/ **[35 points]**
Faites le(s) diagramme(s)e de classe UML des
[COMPOSANT 2] Diag 20 points / Commentaire 5 points
et [COMPOSANT 3] Diag 7 points / Commentaire 3 points
en mettant en évidence les Designs Patterns utilisés.

Commentez chacun de(s) diagramme(s).

Précisions :
Un composant applicatif [COMPOSANT X] correspond à une JVM ou process. Cela signifie que les COMPOSANTS X communiquent 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).

COMPOSANT 2 :



Chaque factory est un Objet Distant afin de traiter les actions d'un Opérateur (DP Objet Distant par héritage)
Le DP Factory de Traitement d'Image gère les Traitements d'Image créés par un Opérateur qui permet de :
- créer un traitement de base avec ses paramètres
- créer une chaîne de traitement.

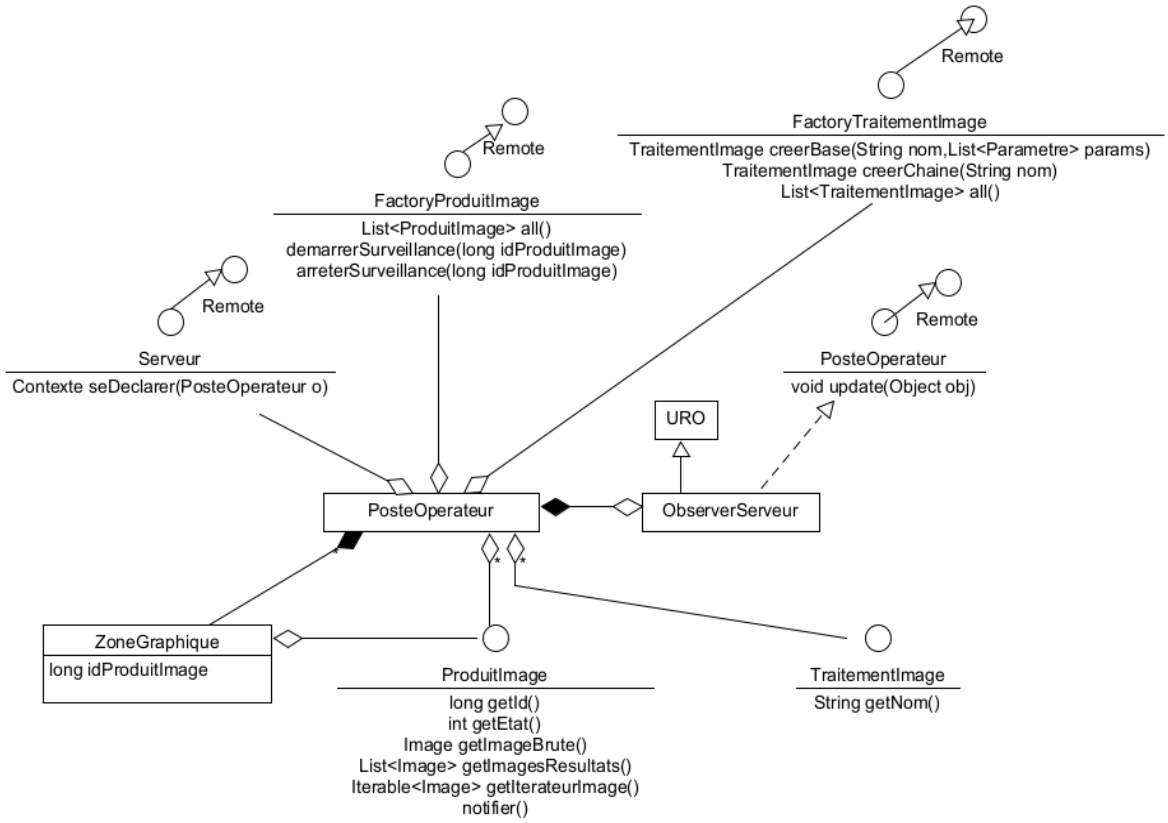
Le DP Composite implémente un traitement d'image qui peut être soit un TraitementImageBase, soit une ChaineTraitement qui contient des TraitementImageBase ou des ChaineTraitement.
Si le traitement d'image est une chaîne de traitement, la méthode add permet d'ajouter un traitement d'image à une chaîne de traitement. La méthode executerThread permet d'exécuter un traitement d'image sur un Produit Image dans un Thread qui appelle successivement les méthodes d'exécution des traitements d'image.
Le DP Factory de Produit d'Image gère les Produits d'Image. Chaque Produit d'Image contient un DP Itérateur pour parcourir les images résultats de l'exécution des traitements d'image.

Un DP Observateur distant est créé entre le serveur et tous les Postes Opérateur pour notifier tout un Produit Image. Cette notification est déclenchée par la méthode d'exécution d'un traitement de base qui appelle la méthode notifier du Produit Image (passé en paramètre) qui utilise l'ObservablePosteOperateur du Serveur.

L'OD ServeurODImpl permet à chaque Poste Opérateur de se déclarer (via un lookup). En retour les stubs des Factory distant son retournés afin que le Poste Opérateur utilisent les méthodes distantes des factorys.
Quand un Poste Opérteur se déclare un observer PosteOperateurObserver est créé qui lui est dédié.

Les 3 composants principaux du serveur sont des Singletons.

COMPOSANT 3 :



Un Poste Operateur utilise toutes les interfaces distantes pour réaliser les actions. Ces interfaces sont dans Contexte, retourné lors de la déclaration du poste au serveur via la méthode seDeclarer qui prend en entrée le stub de ObserverServeur.
Ainsi, chaque Poste Opérateur contient un observer distant ObserverServeur afin d'être notifier d'un Produit Image. La notification d'un Produit Image consiste à rafraîchir la zone graphique associé à ce produit.
L'opérateur utilise l'itérateur du Produit Image pour parcourir les résultats.

Le Poste Opérateur gère la liste des Produits Image et des Traitements d'Image retournées par le serveur pour réaliser toutes les actions.

Pour exécuter un traitement d'image, l'opérateur sélectionne un Produit Image et sélectionne un TraitementImage, et demande au serveur d'exécuter le traitement d'image sur le produit d'image concerné. Le serveur crée une instance du traitement d'image puis exécute la méthode executerThread de ce TraitementImage en lui passant en paramètre le ProduitImage.

Fin du sujet