IPST-CNAM
Intranet et Designs patterns
NSY 102
Mercredi 15 Avril 2020

Durée : **2 h 15**
Enseignants : LAFORGUE Jacques

1ère Session NSY 102

NOM : ……..
PRENOM : ……
NUMERO D'INSCRIPTION : ……

CONSIGNES :

1/ Vous venez de télécharger le sujet d'examen de NSY 102 1ère session 2019-2020.
Ce fichier est un fichier word. Il doit rester dans ce format.

Vous renommez, tout de suite, ce fichier avec votre nom et prénom suivant le format "NOM PRENOM.doc".

2/ Vous écrivez, tout de suite, ci-dessus, votre nom, prénom et numéro d'inscription.

3/ Vous répondez à tous les énoncés, en insérant vos textes et vos diagrammes en-dessous de chaque énoncé. Vous ne modifiez pas les énoncés.

4/ Au bout de 2 heures et 15 minutes, c'est à dire à **16h00**, vous déposez votre copie dans :
"**DEPOT DES COPIES**".

Attention : Après avoir remis votre copie, vous restez en ligne afin que je vous dise que la séance d'examen est terminée car il me faut vérifier que tout le monde ait bien remis sa copie.

En cas de panne de courant, pensez à enregistrer régulièrement votre travail (surtout si vous n'utilisez pas un ordinateur portable ou n'utilisez pas la batterie).

Si vous avez perdu votre connexion internet et que vous ne pouvez pas déposer votre copie alors la procédure à suivre est la suivante :
- vous me téléphonez au afin de me prévenir de cette difficulté
- vous m'envoyez par téléphone une photo de chaque page de votre copie (précisez votre nom dans le SMS)
- dès que possible, vous déposerez votre copie comme prévu, et vous m'enverez un message par SMS pour me prévenir du dépôt (précisez votre nom dans le SMS).

Au total, ce document est constitué de 4 pages. Veuillez vérifier que vous avez bien 4 pages et que la dernière page se termine bien par la phrase « Fin du sujet ».

J'ai volontairement mis les pages suivantes en mode paysage afin que les copiés/collés de vos diagrammes soient plus visibles.

1. EXERCICE 1 [15 Points] (15mn)

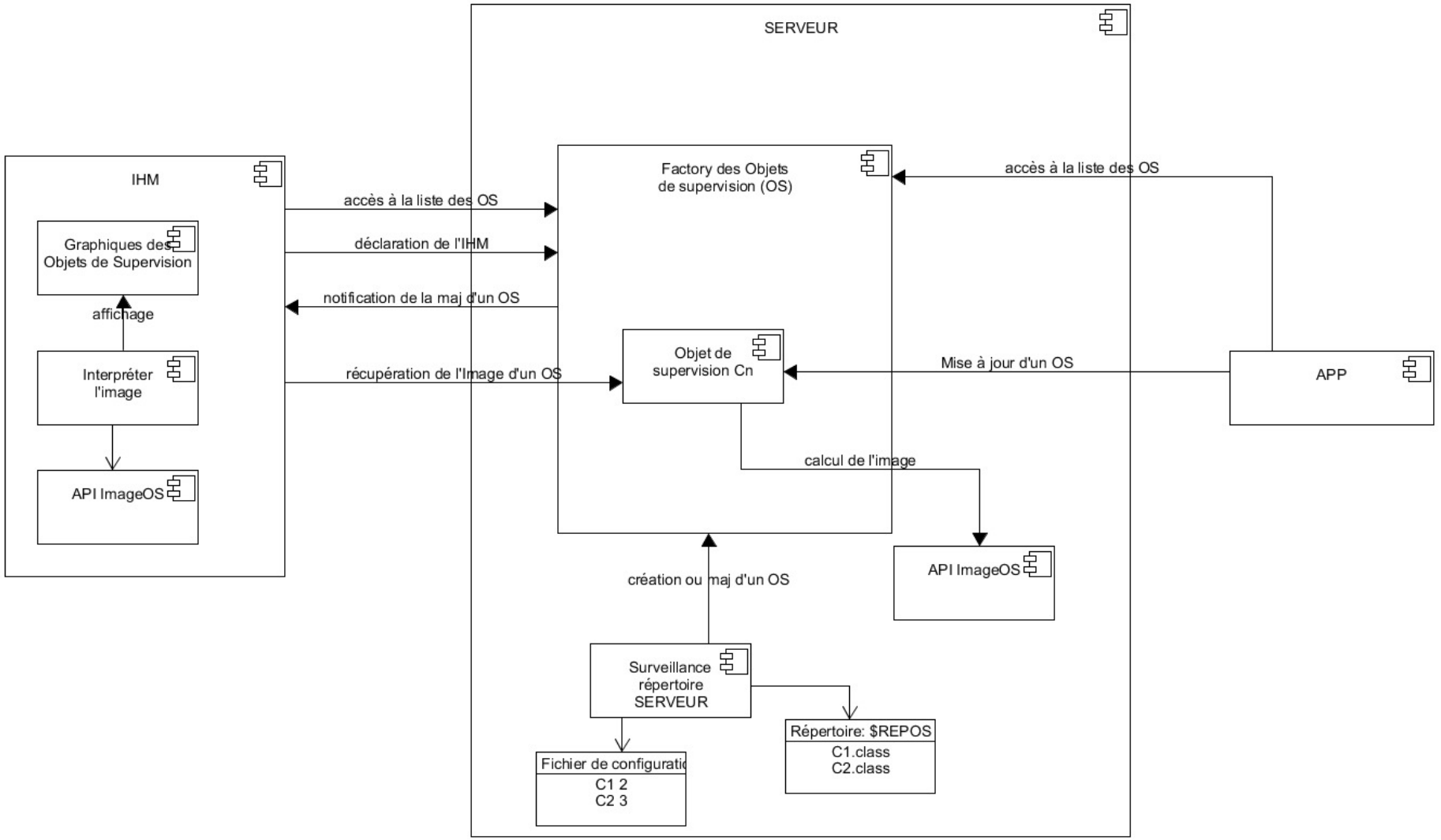
Expliquez avec vos propres mots (ne pas faire de copié/collé avec le cours) pourquoi il est souhaitable d'utiliser les « Designs Patterns » dans la conception d'un Système d'Information et quelles sont les difficultés que l'on peut rencontrer. Soyez convaincant, explicatif et démonstratif dans vos arguments.

Il est souhaitable d'utiliser les Designs Patterns dans la conception d'un Système d'Information car ces derniers représentent une **conception générique** de certains principes informatiques que l'on retrouve d'une manière **récurrente** dans la conception de beaucoup de SI. Autant faire avec ce qui a été éprouvé par des "pères" de l'informatique. Cela amène une **robustesse** intrinsèque dans la conception de son SI.
Leurs réalisations assurent une meilleure **lisibilité, réutilisabilité, évolutivité et maintenance** des SI.
Les DP sont un guide de "**bonnes pratiques**" éprouvées. C'est donc un gage de **qualité**.
Cela permet aussi d'utiliser un **"langage" de conception communs** à tous ceux qui connaissent les designs patterns. Cela est une preuve de compétence.
Cela permet aussi de **faciliter la validation des choix de conception**.
De plus, une fois le choix du design pattern pris, cela permet de se concentrer sur l'aspect plus "métier" de son logiciel.
Il est à noter la forte utilisation des DP dans la réalisation des **Frameworks** utilisés par tous. Ce qui est une garantie de robustesse et d'évolutivité de ces derniers.

Une des difficultés est d'utiliser ces Designs Patterns de manière **adéquate**. C'est-à-dire de discerner dans le besoin de son SI, la possibilité d'utiliser tel Design Pattern plutôt qu'un autre. C'est un équilibre difficile à avoir et seul **l'expérience** permet de l'atteindre.
Une autre grosse difficulté est dans la **volonté** d'utiliser les DP, d'être persuadé de leurs utilités. La **complexité** de ces dernier, la difficulté à les comprendre n'aident pas à la généralisation des DP dans une démarche de conception d'un SI.
Cela nécessite un engagement fort et une implication importante dans la démarche de conception.

2. PROBLEME [50 points] (1h30mn)

Partie 1 **[15 points]** :


SERVEUR
IHM
Graphiques des Objets de Supervision
affichage
Interpréter l'image
API ImageOS
accès à la liste des OS
déclaration de l'IHM
notification de la maj d'un OS
récupération de l'Image d'un OS
Factory des Objets de supervision (OS)
accès à la liste des OS
Objet de supervision Cn
Mise à jour d'un OS
APP
calcul de l'image
API ImageOS
création ou maj d'un OS
Surveillance répertoire SERVEUR
Répertoire: $REPOS
C1.class
C2.class
Fichier de configurati
C1 2
C2 3

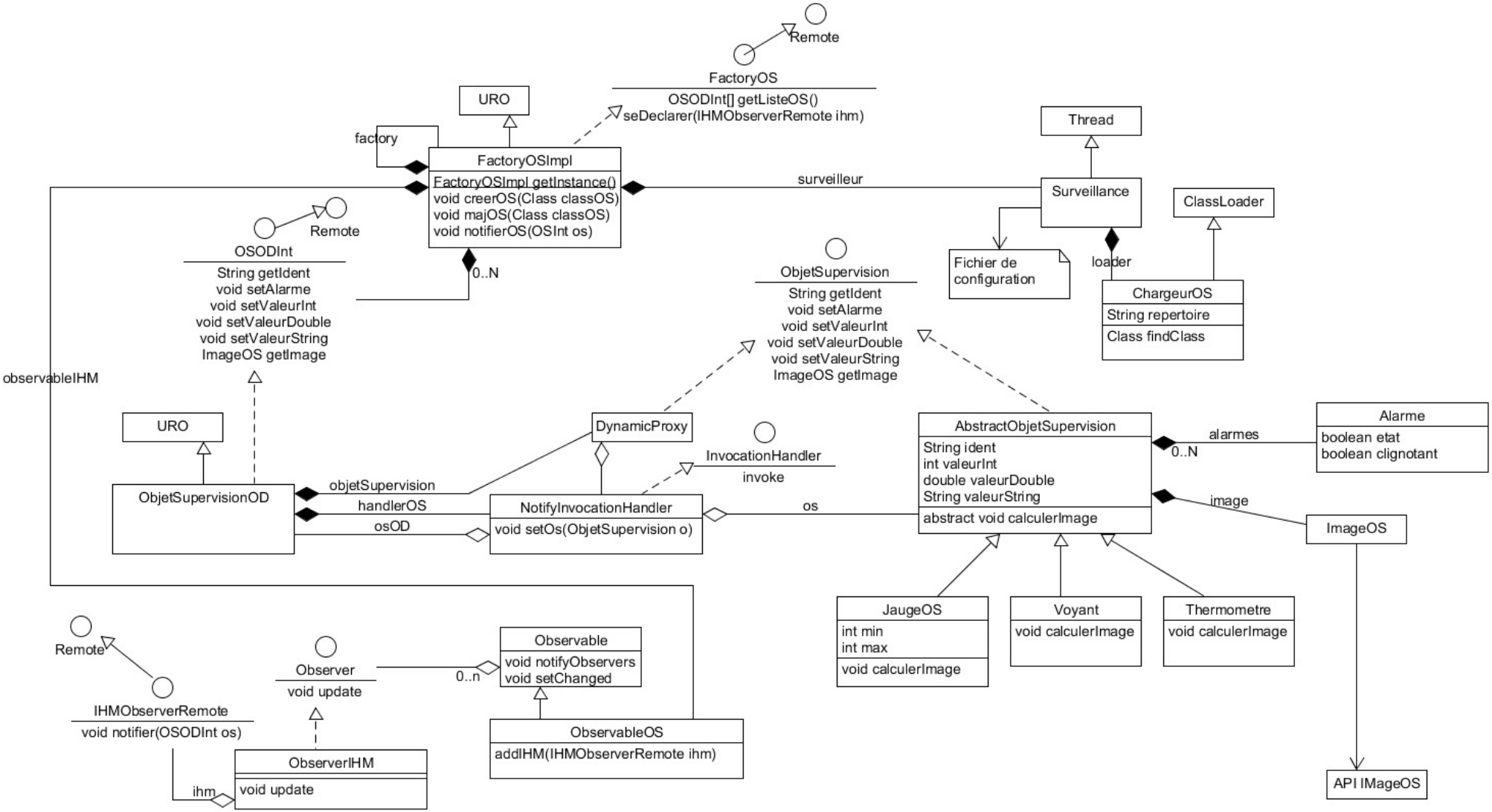
Le sous-composant du SERVEUR, "Surveillance répertoire SERVEUR", s'exécute en tâche de fond. Dès qu'un fichier Cn.class est déposé dans le répertoire $REPOS, ce sous composant demande au "Factory des Objets de supervision" de créer de nouveaux objets de supervision ou de remplacer ceux qui existent déjà.

Le composant APP, demande la liste des objets de supervision se trouvant dans le factory. Il peut donc mettre à jour à distance cet objet de supervision.

A chaque fois qu'un objet de supervision est mis à jour, le SERVEUR notifie, à toutes les IHM qui se sont déclarées, l'information comme quoi un objet de supervision donné a été mis à jour. A la charge du composant IHM de déterminer s'il est intéressé par la maj de cet objet de supervision. S'il est intéressé, le composant IHM demande à cet objet de supervision son image afin de l'interpréter et l'afficher.
Les objets de supervision utilisent une API, "ImagesOS", qui leur permettent de calculer leur représentation graphique en XML. Cette image est calculée à chaque fois qu'une information de l'objet de supervision a été mise à jour.
L'IHM est prévenu de ce changement et peut alors demander l'image de l'objet de supervision.

Partie 2 **[35 points]** :

COMPOSANT 1 :


Remote
FactoryOS
OSODInt[] getListeOS()
seDeclarer(IHMObserverRemote ihm)
URO
factory
FactoryOSImpl
FactoryOSImpl getInstance()
void creerOS(Class classOS)
void majOS(Class classOS)
void notifierOS(OSInt os)
surveilleur
Thread
Surveillance
ClassLoader
Remote
OSODInt
String getIdent
void setAlarme
void setValeurInt
void setValeurDouble
void setValeurString
ImageOS getImage
0..N
Fichier de configuration
loader
ChargeurOS
String repertoire
Class findClass
ObjetSupervision
String getIdent
void setAlarme
void setValeurInt
void setValeurDouble
void setValeurString
ImageOS getImage
observableIHM
URO
DynamicProxy
InvocationHandler
invoke
AbstractObjetSupervision
String ident
int valeurInt
double valeurDouble
String valeurString
abstract void calculerImage
alarmes
0..N
Alarme
boolean etat
boolean clignotant
image
ImageOS
ObjetSupervisionOD
objetSupervision
handlerOS
osOD
NotifyInvocationHandler
void setOs(ObjetSupervision o)
os
JaugeOS
int min
int max
void calculerImage
Voyant
void calculerImage
Thermometre
void calculerImage
Remote
IHMObserverRemote
void notifier(OSODInt os)
Observer
void update
0..n
Observable
void notifyObservers
void setChanged
ObservableOS
addIHM(IHMObserverRemote ihm)
ihm
ObserverIHM
void update
API IMageOS

Il y avait 4 points à adresser :
- la mémorisation des objets de supervision (DP FACTORY + SINGLETON)
- la surveillance du répertoire et le chargement dynamique des classes d'objet de supervision (Thread, Class loader)
- la prise en compte des requêtes provenant de APP (DP OD par composition pour le factory distant) (DP OD par composition pour chacun des OS)
- la notification aux IHM de la notification des modifications des objets de supervision (DP PROXY sur setteurs puis notification par un DP Observateur distant)

Le **factory** (FactoryOSImpl) est un Singleton et est un Objet Distant RMI utilisé à distance par APP via une interface Remote. Ce factory permet de créer un objet de supervision (en locla) qui est constitué de 3 éléments : l'objet distant de l'OS, un dynamic proxy de notification sur les setteurs, et l'objet de supervison : ObjetSupervisionOD + DynamicProxy/NotifyInvocationHandler + un objet dont la classe hérite de AbstractObjetSupervision.
Le **Dynamic Proxy** permet que, automatiquement, dès qu'un des setteurs de l'interface de l'OS est utilisé, il demande au factory de faire la notification aux IHM en lui donnant le stub de l'OS (afin que l'IHM puisse appeler sa méthode getImage).

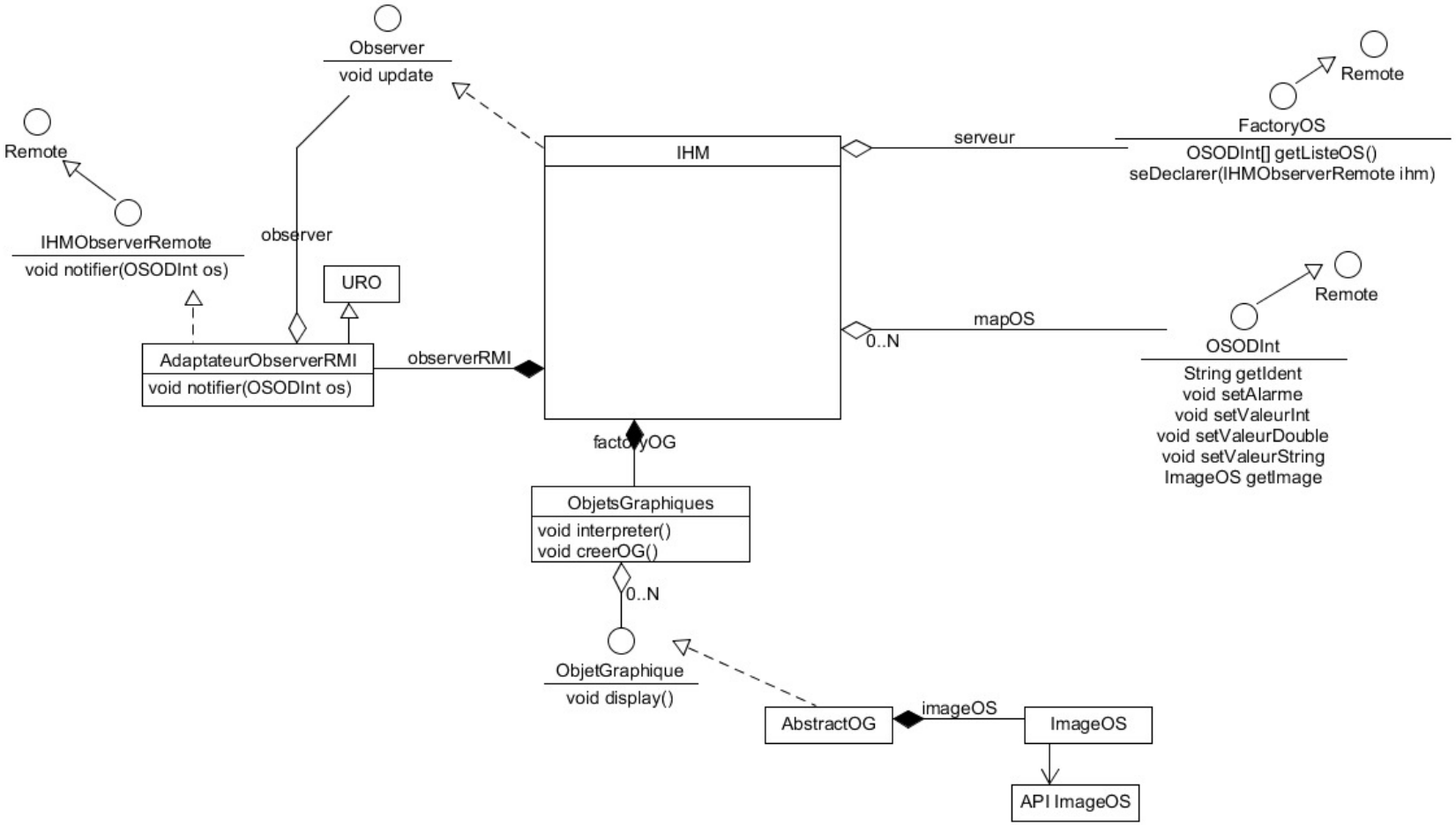
Pour notifier les IHM, le factory utilise un **Observable** (ObservableOS) dont les observers sont des ObserverIHM dont le rôle est d'appeler la méthode distante de l'IHM, notifier, de l'interface distante IHMObserverRemote.

Les objets de supervision sont toutes des classes spécifiques (JaugeOS, Voyant, Thermometre, …) qui héritent de AbstractObjetSupervision. Chacun de ces classes définissent la méthode calculerImage qui calcule la représentation graphique en XML de l'objet de supervision de type ImageOS. Pour cela, on utilise une API permettant de créer cette description au format XML. Cette API sera utilisée par l'IHM pour afficher l'objet de supervision en interprétant ce XML.

La classe **Surveillance** est un Thread qui surveille un changement dans le répertoire. Dès qu'il détecte un nouveau fichier ou la mise d'un fichier, qui est un fichier .class, il charge la classe grâce au chargeur de classe ChargeurOS, puis demande au factory de créer ou mettre à jour cet OS via la méthode creerOS ou majOS qui prend en entrée la classe Java de l'OS.

Dans le cas de la mise à jour d'un OS, le factory utilise le setteur setOS de NotifyInvocationHandler pour remplacer l'OS sans changer l'Objet Distant de cet OS ce qui permet de garder les connexions des OS utilisées par ailleurs.

COMPOSANT 2 :



La classe IHM est un Observer qui utilise un adaptateur distant AdaptateurObserverRMI pour se faire notifier par le serveur.
L'IHM utilise l'interface distante FactoryOS du factory situé sur le serveur. La méthode, seDeclarer, lui permet de se déclarer au près du serveur en lui passant en paramètre le stub de l'adaptateur RMI qui est utilisé par le serveur pour lui notifier un changement d'un objet de supervision (via IHMObserverRemote).

En paramètre de cette notification, l'IHM reçoit le stub de l'objet de supervision, il peut ainsi demander l'image de l'objet de supervision via la méthode getImage de l'interface distante ODSODInt de ce stub et afficher l'objet de supervision.
L'IHM gère un factory d'objets graphiques, classe ObjetsGraphiques, pour chaque objet de supervision qu'elle doit afficher.

Lors de son initialisation, l'IHM utilise la méthode getListe de l'interface distante FactoryOS pour obtenir la liste de tous les objets de supervision afin d'afficher leurs images.

Chaque objet graphique sont des instances de classes qui héritent de la classe abstraite AbstractOG (jauge, thermomettre, courbe, synoptique, …). La classe ImageOS permet d'interpréter l'image qui est une description XML, en une forme graphique. Pour cela la classe ImageOS utilise l'API ImageOS..