

	<p align="center">Programmation Java : Programmation Objet</p> <p align="center">Projet de NFA 032</p> <p align="center">2016 - 2017</p>	<p align="right">14/05/2017</p> <p align="right">Par J. LAFORGUE</p>
-----------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------

CONSERVATOIRE
NATIONAL
DES ARTS
ET METIERS

cnam

CENTRE REGIONAL
MIDI - PYRENEES

NFA 032 – Travaux pratiques

*IPST-
CNAM
Licence*

PROJET

Années 2016-2017
TOULOUSE
J. LAFORGUE

1. LE BESOIN

On se propose de continuer le projet qui a été réalisé dans le cadre du cours NFA031.

Le sujet et les sources de ce projet se trouvent sur le site :
http://jacques.laforque.free.fr/SITE_NFA031/Site/Projets.phtml

Pour ceux qui n'ont pas participé à ce projet, il est important qu'ils prennent connaissance de ce projet dans le détail.

Les évolutions que l'on propose de réaliser dans ce projet de NFA 032 sont :

- [FONC 1] au lieu d'utiliser une seule classe de définition des bons de commande, on va utiliser une arborescence **d'héritage** de classe (autant de classe que de **type de bons de commande**) et la gestion d'une **collection polymorphe**, et ceci à iso-fonctionnalités avec le projet de NFA031.


On identifie deux types de bons de commande :

- ✦ ceux réalisés par un client du site (string : nom du client, ...)
 - ✦ ceux réalisés par un rayon du magasin (int : numéro du rayon, ...).
- [FONC 2] trier, dans la collection, les bons de commande par type de bon de commande puis par date croissante.
- [FONC 3] la sauvegarde et la restauration des données du programme dans un **fichier binaire** (Data) afin de ne pas perdre les modifications.
- [FONC 4] réaliser une IHM **distante** permettant de saisir une commande via une communication par **socket**.

De plus, dans tout le projet, il faut utiliser les **exceptions**, pour faire en sorte qu'une mauvaise utilisation de l'opérateur ne déclenche pas d'erreur. L'opérateur est alors prévenu de son erreur.


Toutes ces évolutions seront réalisées en 3 étapes :

- Etape1 : [FONC 1] et [FONC 2]

	<p style="text-align: center;">Programmation Java : Programmation Objet</p> <p style="text-align: center;">Projet de NFA 032</p> <p style="text-align: center;">2016 - 2017</p>	<p style="text-align: right;">14/05/2017</p> <p style="text-align: right;">Par J. LAFORGUE</p>
----------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------

- Etape 2 : [FONC 3]
- Etape 3 : [FONC 4]

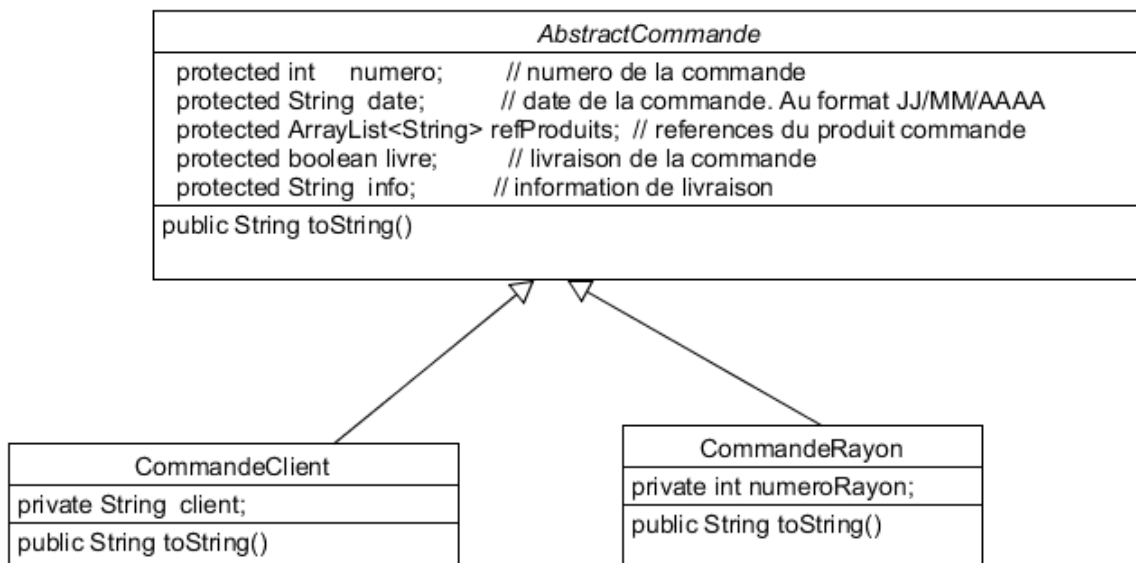
Il est important de respecter la chronologie de la réalisation de ces étapes.

	Programmation Java : Programmation Objet Projet de NFA 032 2016 - 2017	14/05/2017 Par J. LAFORGUE
----------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------	--------------------------------------

2. ETAPE 1

Cette étape consiste à faire les fonctionnalités: [FONC 1] et [FONC 2].

Il s'agit de modifier le projet de NFA 031 à iso-fonctionnalités en envisageant une nouvelle implémentation de la classe Commande (qui disparaît) et est remplacée par l'architecture de classes suivante :



La classe Site n'utilise plus la classe **Commande** qui est remplacée par la classe **AbstractCommande**.

Il faut donc :


- modifier la classe Site :
 - ➔ afin qu'elle utilise la classe AbstractCommande
 - ➔ modifier la méthode initialiserCommandes :
 - si le 3^{ème} champ de la commande n'est pas un entier alors créer une **CommandeClient**
 - sinon créer une **CommandeRayon**
- créer les classes AbstractCommande, CommandeClient, CommandeRayon

Les attributs de la classe AbstractCommande sont protected.

Modifier le fichier Commandes.txt afin de mettre des commandes dont le 3^{ème} champ est un numéro de rayon.

Une fois que le programme fonctionne comme avant mais avec cette nouvelle implémentation de classes, il faut trier les commandes de la collection. Ainsi les commandes s'afficheront par type de commande puis par ordre chronologique croissante sur la date.

NB: Pour l'ordre de la date, utilisez les méthodes de la classe fournie : fr.cnam.util.**DateString**.

	Programmation Java : Programmation Objet Projet de NFA 032 2016 - 2017	14/05/2017 Par J. LAFORGUE
----------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------	--------------------------------------

3. ETAPE 2

Cette étape consiste à faire la fonctionnalité suivante : [FONC 3].

Avant de réaliser cette fonction, il faut faire une modification de la classe **IHMSite** et de la classe **Site** afin que l'initialisation des bons de commande avec un fichier texte soit un choix dans l'IHM : on ajoute une zone de saisie pour le nom du fichier et un bouton "Importer" qui réalise l'import dans le Site des bons de commande contenues dans un fichier texte se trouvant dans le répertoire data. Cela veut dire que l'on ne fait plus la lecture du fichier texte dans le constructeur de la classe Site.

Pour [FONC 3], il faut impacter les classes Site, AbstractCommande, CommandeXXX et Produit pour que :

- l'on puisse faire la sauvegarde dans le fichier "data/SITE.bin" de toutes les données du site. Ce fichier est un fichier binaire. On réalise la sauvegarde en utilisant DataOutputStream
- cette sauvegarde se fait quand on clique dans un bouton "Sauvegarder"
- dans le constructeur de la classe Site, automatiquement, le fichier binaire est chargé afin d'initialiser le site avec le contenu du fichier. On réalise le chargement en utilisant DataInputStream . Le stock lu par le fichier texte est écrasé par le contenu du fichier binaire.

On sauvegarde dans le fichier binaire :

- le stock
- les commandes

	<p align="center">Programmation Java : Programmation Objet</p> <p align="center">Projet de NFA 032</p> <p align="center">2016 - 2017</p>	<p align="right">14/05/2017</p> <p align="right">Par J. LAFORGUE</p>
-----------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------

4. ETAPE 3

Cette étape consiste à faire la fonctionnalité suivante : [FONC 4].

Il s'agit de faire un deuxième programme Java (un client) qui est une nouvelle IHM permettant de saisir un bon de commande **à distance**.

Pour cela, il faut modifier notre projet pour créer un **serveur de socket** qui accepte des requêtes réseau.

Sur le serveur et le client, on utilise les classes **DataOutputStream** et **DataInputStream** pour écrire et lire les données sur le socket. On décide d'utiliser ce format (Data) afin que les projets entre les auditeurs soient compatibles à travers le réseau.

Cette nouvelle IHM doit permettre de :

- à l'initialisation, l'IHM demande au serveur (Site) la liste des références des produits du stock
- saisir un bon de commande client ou rayon en sélectionnant les références des produits du stock. Le numéro de bon de commande n'est pas saisi car il est déterminé automatiquement par le serveur (numéro unique (ex: max numéro + 1)) (*). Quand le site traite la requête et ajoute la nouvelle commande, il demande à son ihm d'afficher toutes les commandes du site.

(*) Remarque : dans l'étape2, on fait l'import du fichier de commande. Il faut que le numéro de bon de commande du fichier soit ignoré et remplacé par celui déterminé par le site (numéro unique).

Nous avons ainsi deux requêtes à implémenter qui doivent respecter la syntaxe suivante afin que vous puissiez communiquer avec un autre projet d'un autre auditeur.

Requête 1 :

Pour l'envoi :

"GET_REFERENCES_STOCK"

Pour la réponse:

<nb références> [<référence>]*

avec <nb références> ::= *entier*
<référence> ::= *chaîne*

Requête 2 :

Pour l'envoi :


"AJOUTER_COMMANDE" <date> <origine> <nb ref produit> [<ref produit>]*

Pour la réponse:

<réponse>

avec <date> ::= *chaîne*
<origine> ::= *chaîne*
<nb ref produit> ::= *entier*
<ref produit> ::= *chaîne* de la forme <reference>=<quantité>
<réponse> ::= *chaîne* (texte quelconque)

Les *chaîne* sont lues et écrites avec readUTF et writeUTF
Les *entier* sont lus et écrits avec readInt et writeInt.

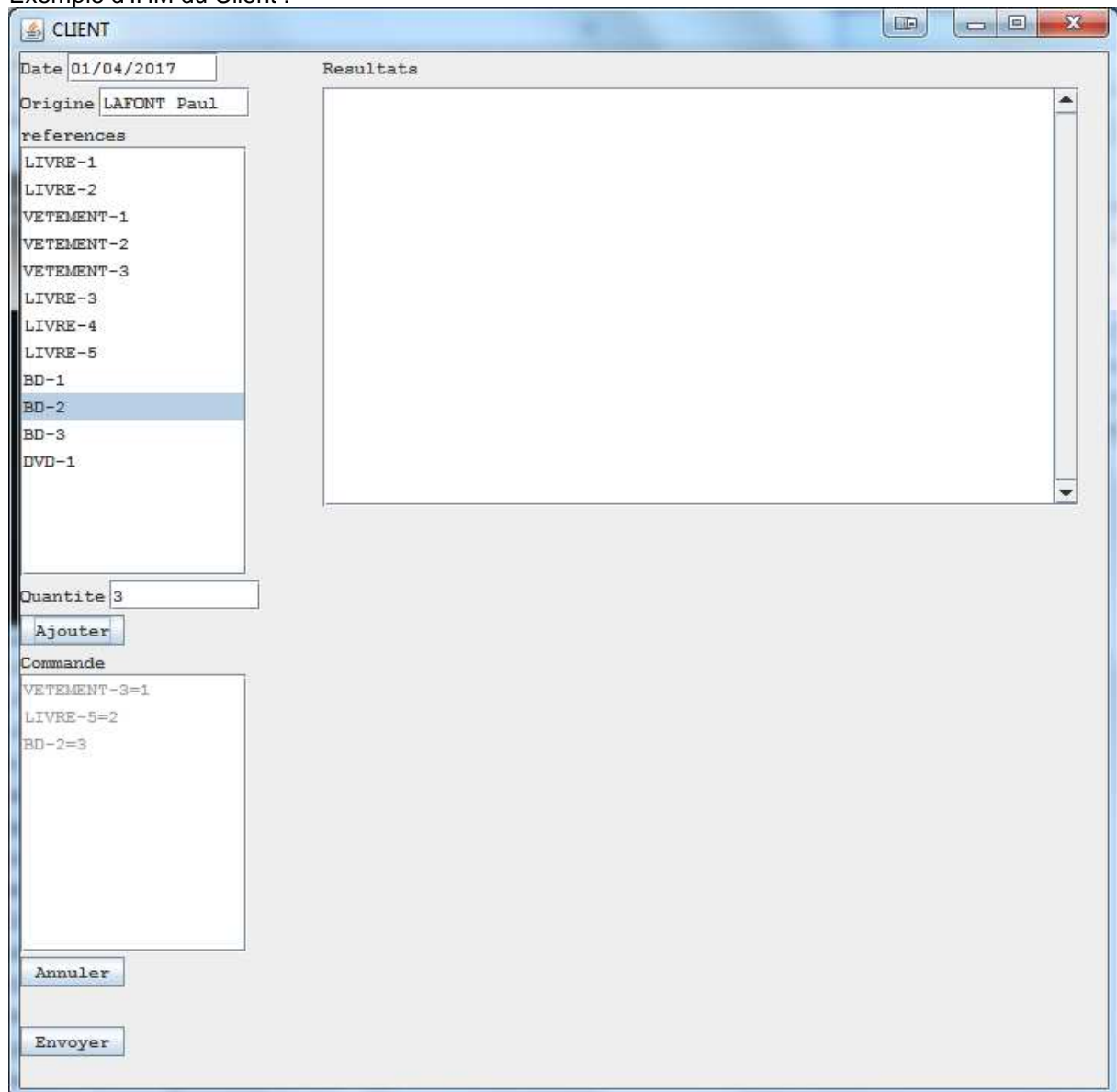
	Programmation Java : Programmation Objet Projet de NFA 032 2016 - 2017	14/05/2017 Par J. LAFORGUE
----------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------	--------------------------------------

Afin que le client puisse se connecter sur une machine distante, on passe en paramètre du programme **l'adresse ip** (ou le host) de la machine distante du serveur et le **port** de connexion du serveur de socket.

Pour changer, éventuellement de **port**, on le passe en paramètre de lancement du programme **Projet**.

Dans la salle de TP du CNAM, il faut utiliser le port **9100** pour exécuter le serveur de socket. Le firewall du CNAM nous autorise d'utiliser les ports de 9100 à 9110.

Exemple d'IHM du Client :



	Programmation Java : Programmation Objet Projet de NFA 032 2016 - 2017	14/05/2017 Par J. LAFORGUE
----------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------	--------------------------------------

5. QUELQUES PRECISIONS


Les règles de codage que vous devez suivre sont :

- sauf pour les constantes, les attributs de toutes les classes doivent être privés
- toutes les classes commencent par une majuscule
- tous les attributs, paramètres et variables commencent par une minuscule
- les noms de tous les répertoires de package sont en minuscule
- les packages créés commencent tous par "fr.cnam"
- une indentation stricte est respectée dans tout le code
- un minimum de commentaire est nécessaire afin de comprendre ce que fait le code. Surtout quand ce que fait le code n'est pas décrit précisément dans le sujet.

Conseils :

- Ecrivez peu de code à chaque fois afin d'avoir toujours un code qui se compile correctement
- Après chaque étape ou sous-étape, le programme s'exécutant correctement, faites une sauvegarde des sources. Cela permet de pouvoir revenir en arrière si besoin et permet aussi de pouvoir donner à l'enseignant une version du programme qui s'exécute correctement et qui remplit correctement une partie des fonctionnalités demandées dans le sujet.
- Dans un premier temps ne faites pas plus que le sujet ne le demande. Une fois que le programme respecte le sujet, vous faites une sauvegarde et vous pouvez apporter des améliorations qui sont toujours appréciées par le correcteur, à condition qu'elles soient bien commentées.

Dans le fichier `Projet.java`, vous renseigner les noms et prénoms du groupe.

	Programmation Java : Programmation Objet Projet de NFA 032 2016 - 2017	14/05/2017 Par J. LAFORGUE
----------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------	--------------------------------------

6. ARCHITECTURE DES REPERTOIRES

L'architecture des répertoires est la suivante, dans le répertoire racine :

- le script **compil.bat** qui permet de compiler les deux programmes
- le script **run.bat** qui exécute le programme de gestion du site (projet)
- le script **runClient.bat** qui exécute le programme client qui communique avec le programme de gestion du site.

Ces 3 scripts permettent de compiler et d'exécuter les programmes indépendamment d'éclipse.

- le répertoire **src** qui contient les sources de tous les packages
- le répertoire **bin** dans lequel sont créés les .class issus de la compilation
- le répertoire **data** qui contient le fichier texte d'import et le fichier binaire de sauvegarde du compte bancaire.

7. PLANNING

Vous avez 3 séances de TP pour réaliser ce projet

Le projet sera ramassé 1/4 heure avant la fin de la dernière séance de TP (le temps de les ramasser tous).

Il est donc **impératif** que, au moins un membre du groupe de TP, soit présent lors de la dernière séance afin de donner son projet.