CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL MIDI - PYRENEES

NFA 032 – Travaux pratiques

IPST-CNAM Licence

PROJET

Années 2015-2016
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.laforgue.free.fr/SITE_NFA031/Site/Projets.phtml

Pour ceux qui n'ont pas participés 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 le projet de NFA 032 sont :

- [FONC 1] au lieu d'utiliser une seule classe de définition des opérations bancaire, on va utiliser une arborescence **d'héritage** de classe (autant de classe que de type d'opération bancaire) et la gestion d'une **collection polymorphe,** et ceci à iso-fonctionnalités avec le projet de NFA031.
- [FONC 2] trier les opérations de la collection polymorphe à chaque modification.
- [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** de consultation du compte bancaire via une communication par **socket**.

De plus, dans tout ce projet, il faut utiliser les **exceptions**, pour faire en sorte que, les erreurs d'exécution (mauvaise saisie, mauvais choix, le fichier de compte d'import n'existe pas,…) ne déclenchent pas d'erreur d'exécution.

Toutes ces évolutions seront réalisées en 3 étapes :

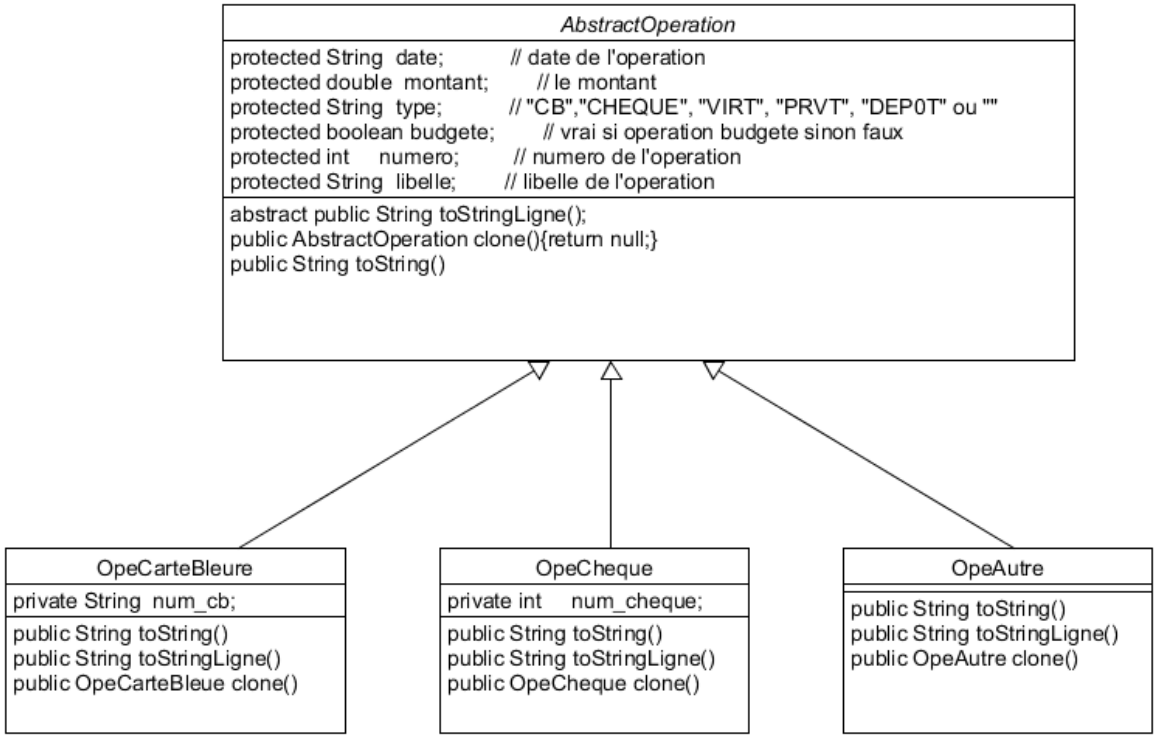
- Etape1 : [FONC 1] et [FONC 2]
- Etape 2 : [FONC 3]
- Etape 3 : [FONC 4]

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

2. ETAPE 1

Cette étape consiste à faire les fonctionnalités suivantes : [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 Operation (qui disparait) et est remplacée par l'architecture de classes suivante :



La classe Compte n'utilise plus la classe **Operation** qui est remplacée par la classe **AbstractCollection**.

Il faut donc :

- modifier la classe Compte :
  - afin qu'elle utilise la classe AbstractOperation
  - modifier la méthode ajouterOperation :
    - si le type de l'opération est "CB" alors créer une OpeCarteBleue
    - si le type de l'opération est "CHEQUE" alors créer une OpeCheque
    - si le type de l'opération est autre que "CB" et "CHEQUE" alors créer une OpeAutre
- créer les classes OpeCarteBleue, OpeCheque, OpeAutre

Les attributs de la classe AbstractOperation sont protected.

Remarque : la méthode clone de la classe AbstractOperation ne peut pas être abstraite car elle existe dans la classe Object. Il faut donc implémenter cette méthode dans la classe AbstractOperation avec

un code par défaut qui ne sera jamais utilisé, et la surcharger dans les classes OpeXXX qui réalise le clonage.

Une fois que le programme fonctionne comme avant mais avec cette nouvelle implémentation de classes, il faut trier les opérations de la collection à chaque modification. Ainsi les opérations s'afficheront par ordre chronologique croissante sur la date.
Pour les opérations budgétées qui n'ont pas de date complète, elles sont considérées comme toujours inférieures à toutes les opérations non budgétées.

Remarque : il est inutile de modifier l'IHM d'ajout d'une opération car elle permet de saisir tous les champs de toutes les opérations possibles.
Dans l'absolu on devrait d'abord demander quel type d'opération on veut puis afficher une IHM différente adaptée à chacune des classes d'opérations.

3. ETAPE 2

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

Avant de réaliser cette fonction, il faut faire une modification de l'IHM et de la classe Compte afin que l'initialisation du Compte 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 Compte, des opérations contenues dans un fichier texte. Cela veut dire que l'on ne fait plus la lecture du fichier texte dans le constructeur de la classe Compte.

Pour [FONC 3], il faut impacter les classes Comptes, AbstractOperation et OpeXXX pour que :

- l'on puisse faire la sauvegarde dans le fichier "DATA/COMPTE.bin" de toutes les données du compte. 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 Compte, automatiquement, le fichier binaire est chargé afin d'initialiser le compte avec le contenu du fichier. On réalise le chargement en utilisant DataIntputStream

On sauvegarde dans le fichier binaire :

- le mois courant
- toutes les opérations

4. ETAPE 3

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

Il s'agit de faire un deuxième programme Java qui affiche une nouvelle IHM permettant de consulter **à distance** le compte qui est géré par notre projet.

Pour cela, il faut modifier notre projet pour créer un **serveur de socket** qui accepte des requêtes réseau.
On utilise les classes **DataOutputStream** et **DataInputStream** pour écrire et lire les données sur le socket.

Cette nouvelle IHM doit permettre de :

- rechercher une opération en fonction d'une sous-chaine du libellé et affiche au format long les opérations trouvées
- afficher toutes les opérations non budgétées au format court
- afficher toutes les opérations budgétées au format court

Pour réaliser ces 3 besoins, on définit les **deux** requêtes suivantes :

- [REQUETE 1] rechercher une chaine, indépendamment de la case, dans les libellés de toutes les opérations. Elle retourne toutes les opérations trouvées.
- [REQUETE 2] demande les opérations du compte budgétées ou non budgétées

Afin que tout le monde puisse interroger le compte des autres, il faut respecter une même interface des requêtes utilisées par tous.

[**REQUETE 1**] :

- Envoi :
  - (String UTF) "RECHERCHER_OPERATIONS_LIBELLE"
  - (String UTF) chaine à rechercher
- Reception :
  - (String UTF) résultat : le serveur construit la concaténation de toutes les opérations trouvées

[**REQUETE 2**] :

- Envoi :
  - (String UTF) "OPERATIONS"
  - (boolean) budgétées/non budgétées
- Reception :
  - (String UTF) résultat : le serveur construit la concaténation de toutes les opérations budgétées ou non budgétées.

Afin que le client puisse se connecter sur une machine distante, on passe en paramètre de lancement 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**.

Côté client : si la communication avec le serveur ne peut pas être établie alors l'erreur est affiché dans l'IHM cliente.

Côté serveur: chaque requête est traitée dans un **thread** afin de traiter toutes les requêtes des clients soient traitées en parallèle. Ainsi un client n'est pas en attente de la fin de traitement du client précédent.

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.

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.

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 compte bancaire (projet)
- le script **runClient.bat** qui exécute le programme client qui communique avec le programme de gestion du compte bancaire.
  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.