# NFA019 : Série 5.1- Morane Gruenpeter

a. La série 5.1 consiste à créer des fichiers pour garder les informations traitées et ajoutées pendant le travail avec le logiciel.

b. Les classes qui ont été modifiées :

- Articles51, ArticlePromo51 et AbstraitArticle : implements serializable
- Client51 : ajout de deux objets de gestion et appel au fichier avec ces objets de gestion pour mettre les données dans des structures sur lesquelles on peut travailler.
- GestionTableDesArticles51 :
    1. Ajout de la méthode lire(), fait appelle à connectionsFichier
    2. Ajout de la méthode écrire(structure), fait appelle à connectionsFichier
    3. Ajout d'un objet connectionsFichier

- GestionTableDesCommandes51 :
    1. Ajout de la méthode lire(), fait appelle à connectionsFichier
    2. Ajout de la méthode écrire(structure), fait appelle à connectionsFichier
    3. Ajout d'un objet connectionsFichier
    4. Changement de la méthode ajouter, car il faut s'occuper de la génération d'idCommande.

- GestionDuneCommande51 :
    Rien à changer
- UneCommande51 :
    1. Changement du numeroCommande en idCommande =>un String
    2. Changement du constructeur> UneCommande51(int num, DateUser d) qui construit l'idCommande par rapport à ces deux param.
    3. Ajout d'une dateFacturisation et d'un booléen facturée pour la série 5.2
    4. : implements serializable
- LigneDeCommande51 : implements serializable
- Tools : ajout d'une nouvelle méthode valider(String, Hashtable<>)
- iPane.ES : rien à changer.

# Class ConnectionFichiers<TypeStructure>

```
java.lang.Object

      ConnectionFichiers<TypeStructure>
```

---

```
public class ConnectionFichiers<TypeStructure>extends java.lang.Object
```

---

| Constructor Summary |
| --- |
| **ConnectionFichiers**()<br>constructeur par défaut |
| **ConnectionFichiers**(java.lang.String nomPhysique)<br>constructeur avec param |

| Method Summary | |
| --- | --- |
| void | **ecrire**(TypeStructure tab)<br>méthode d'écriture de la structure qui a été travailler dans le fichier consiste à sauvegarder une nouvelle version de cette structure et à effacer la plus ancienne |
| java.lang.String | **getNomPhysique**()<br>getter du nom physique qui lie le logiciel avec le fichier |
| TypeStructure | **lire**()<br>méthode de lecture du fichier qui sauvegarde la structure sur laquelle ont veut travailler |
| void | **setNomPhysique**(java.lang.String nomPhysique)<br>setter du nomPhysique qui lie le logiciel avec le fichier |

c. **Le scénario de la classe Client :**
Le client à les mêmes choix mais maintenant à chaque création de fichier ou de sauvegarde il y a un message qui est affiché. En plus, à chaque ouverture du programme le fichier qui a été modifié est accessible, donc le client peut travailler avec des données ultérieures.

d**. Les apports du cours :**
**Serializable :** concept que j'ai rencontré là pour la première fois. Chaque classe qui est sauvegardé dans le fichier doit être sérialisé.
**Nom physique et nom logique :** association de ces deux noms pour travailler sur le fichier avec le logiciel.
**FileInputStream et FileOutputStream :** FileInputStream est conçu pour les courants de lecture des octets. Pour la lecture des flux de caractères, il faut utiliser FileReader.
FileOutputStream est conçu pour les mêmes courants en écriture.
**ObjectInputStream et ObjectInputStream:** ObjectOutputStream et ObjectInputStream peuvent fournir une application avec un stockage persistant pour les graphes d'objets lorsqu'il est utilisé avec un FileOutputStream et FileInputStream respectivement. ObjectInputStream est utilisé pour récupérer les objets précédemment sérialisés.

**Exceptions :** pour l'utilisation des fichiers, les exceptions sont très importantes à gérer. Je ne suis pas sûre que l'affichage à chaque traitement d'exception soit nécessaire.

e. **Conclusion :**
L'utilisation d'un fichier était un concept difficile pour moi, ne comprenant pas les inputSream ou les bufferReader. Cette série m'a permis de travailler avec les fichiers d'une manière plus directe et agréable sans ce cassé trop la tête. C'est vraiment magique de voir que ça marche sans écrire trop de ligne de code. Maintenant l'utilisation de fichier me parait logique et même facile.

# NFA019 : Série 5.2- Morane Gruenpeter

a. La série 5.2 consiste à créer l'entité Facture avec sa table de factures et la gestion des factures. Il faut faire bien attention d'implémenter Serializable, car on voudrait mettre les factures aussi dans des fichiers.

b. Les classes qui ont été modifiées :

J'ai reconstruit le projet dans des nouveaux packages pour ranger les classes par thème et en enlevant le suffixe 51 ou 52. Les packages :

- Metiers
- Structures
- Gestionnaires
- Interfaces
- Tools : DateUser et Tools .
- ipane

Les changements :

- Articles, ArticlePromo et AbstraitArticle : rien à changer
- Client52 :
    1. ajout de l'objet gestion des factures et le cas 3 dans le menu principal qui nous envoie vers le menu gestion des factures.
    2. Ajout des méthodes lire() et écrire() pour le fichier les Factures.
- GestionTableDesArticles :
    Changement de Objet o avec TableDesFactures ;
    //je pense que cela n'est pas nécessaire, or si on voudra plus tard un accès
    //direct de la gestion des articles au factures c'est la seule option.
- GestionTableDesCommandes51 :
    1. Changement de Objet o avec TableDesFactures ;
    2. Changement de la méthode supprimer(), qui vérifie si la commande a été facturée
    3. Changement de la méthode ajouter(), qui vérifie si la commande qui est ajoutée à la table des commandes a été facturée par le menu gestion d'une commande.
    4. Changement de la méthode facturer, qui fait appel à la méthode `GestionTableFacture.transferComVersFact(lesFactures, commandeCourante, lesArticles);`

- GestionDuneCommande :
    1. Changement de Objet o avec TableDesFactures ;
    2. Changement de la méthode facturer, qui fait appel à la méthode `GestionTableFacture.transferComVersFact(lesFactures, commandeCourante, lesArticles);`

- UneCommande :

5. Ajout d'un booléen facturée, qui désigne si la commande a été facturée
6. Ajout des méthodes set et get pour le booléen facturée.

- LigneDeCommande : rien à changer.
- Tools : rien à changer.
- iPane.ES : rien à changer.
- DateUser : ajout de la méthode `public DateUser ajouterNombreJours(int nbj)` qui retourne une nouvelle date qui est « nbj » plus tard.

c. **Le scénario de la classe Client :**
Le client à un nouveau choix dans le menu principale, qui est « gestion des factures » ; dans ce menu il peut :

1. Editer une facture => une commande dans la table des commandes devient une facture dans la table des factures.
2. Supprimer une facture => suppression définitive d'une facture de la table des factures après un délai choisi. (question de réflexion : est-ce que la commande aussi doit être supprimée au même moment ?)
3. Visualiser une facture=> affichage de la facture en question
4. Editer toutes les facture=> après confirmation, toutes les commandes de la table des commandes sont facturées.

---

# Class GestionTableFacture

`java.lang.Object`

**`GestionTableFacture`**

**All Implemented Interfaces:**

InterfaceGestion<TableDesFactures,TableDesCommandes,TableArticle>, java.io.Serializable

## Constructor Summary

| |
|---|
| `GestionTableFacture(java.lang.String nomFichierFacture)`<br>constructeur avec param |

## Method Summary

| | |
|---|---|
| void | `ajouter(TableDesFactures lesFactures, TableDesCommandes lesCommandes, TableArticle lesArticles)`<br>ajout d'une facture à la table des facture, après vérification qu'il y a au moins une commande à facturer et vérification si la commande choisi n'a pas déjà été facturé |

| | |
|---|---|
| void | **ecrire**(TableDesFactures tabFact)<br>ecriture de la table des factures en param dans le fichier des factures |
| void | **editer**(TableDesFactures lesFactures, TableDesCommandes lesCommandes, TableArticle lesArticles)<br>visualisation d'une facture choix de la facture et affichage |
| void | **facturerToutes**(TableDesFactures lesFactures, TableDesCommandes lesCommandes, TableArticle lesArticles)<br>facturation de toutes les factures qui n'ont pas été facturées |
| TableDesFactures | **lire**()<br>lecture du fichier qui sauvegarde la table des factures |
| int | **menuChoix**(TableDesFactures tab1)<br>affichage plus choix |
| void | **menuGeneral**(TableDesFactures lesFactures, TableDesCommandes lesCommandes, TableArticle lesArticles)<br>menu général qui renvoie vers la méthode choisi par l'utilisateur |
| void | **supprimer**(TableDesFactures lesFactures, TableDesCommandes lesCommandes, TableArticle lesArticles)<br>suppression d'une facture de la table des factures après test supprimable() |
| static void | **transferComVersFact**(TableDesFactures lesFactures, UneCommande commande, TableArticle lesArticles)<br>procédure de création de facture à partir d'une commande et son ajout à la table des factures il n'y a aucun test dans cette procédure, il faut l'appeler seulement après tests elle est en statique pour laisser la possibilité de l'appeler de gestion des commandes |

---

# Class TableDesFactures

java.lang.Object

**TableDesFactures**

**All Implemented Interfaces:**

InterfaceStructure<java.lang.String,Facture>, java.io.Serializable

---

| Constructor Summary |
| --- |
| **TableDesFactures**()<br>constructeur par défaut |
| **TableDesFactures**(java.util.Hashtable<java.lang.String,Facture> tabFacture)<br>constructeur avec param |

| Method Summary | |
| --- | --- |
| void | **ajouter**(Facture actuelle)<br>ajoute la facture en param à la table des factures |
| java.lang.String | **cle**()<br>retourne toutes les clés de la structure |
| Hashtable<String,Facture> | **getLesFactures**()<br>getteur de la table des factures |
| Facture | **retourner**(java.lang.String cle)<br>retourne la facture avec la clé en param |
| void | **supprimer**(java.lang.String cle)<br>supprime la facture avec la clé en param (il faut faire les tests en dehors de cette méthode) |
| int | **taille**()<br>renvoie la taille de la table des factures |

# Class Facture

java.lang.Object

**Facture**

**All Implemented Interfaces:**

java.io.Serializable

---

## Constructor Summary

| |
|---|
| **Facture**()<br>constructeur par défaut |
| **Facture**(DateUser dateFacture, java.lang.String numero, java.lang.String facture)<br>constructeur avec param |

## Method Summary

| | |
|---|---|
| DateUser | **getDateFacture**()<br>renvoie la date à laquelle cette facture a été créer |
| java.lang.String | **getFacture**()<br>renvoie le String facture |
| java.lang.String | **getNumero**()<br>renvoie le numéro de la facture |
| void | **setDateFacture**(DateUser dateFacture)<br>setter de la date de la facturation |
| void | **setFacture**(java.lang.String facture)<br>setter du String facture |
| void | **setNumero**(java.lang.String numero)<br>setter du numéro de la facture |
| boolean | **supprimable**()<br>méthode qui renvoie si l'on peut supprimer la facture |
| java.lang.String | **toString**()<br>chaîne de caractères qui désigne la facture |

d**. Les apports du cours et Conclusion :**

Pour cette série nous n'avons pas abordé en cours de nouveaux concepts. Donc, j'écris mes apports du cours et mes conclusions sous le même titre.

Cette série m'a permis de synthétiser le travail que j'ai réalisé pendant le projet de la superette et de m'approprié des manières de procédure pour réaliser un projet en Java.

La classe de gestion est une classe qui fait appel aux méthodes de la classe structure qui contient la ou les classes métiers.

Pour cette série on ajoute un nouveau gestionnaire « gestionTableDesFactures » qui manipule la structure « tableDesFactures » qui contient des objets métiers « facture ».

La manière que j'ai choisi de procédé est de faire les tests seulement dans la classe gestion avant chaque action sur la classe structure.

Par ailleurs, j'ai remarqué que maintenant en utilisant les fichiers, il fallait faire un test si la structure ou l'objet n'est pas null, car sinon une exception NullPointerException était levé.

Les deux interfaces : interfaceGestion et interfaceStructure, m'ont facilité la construction des deux nouvelles classes qui implémente une interface, car la signature des méthodes essentielles était déjà écrite. On aurait pu ajouter une interface métier, mais peut-être ce n'était pas indispensable.

Maintenant, l'application est prête pour la prochaine phase : l'interface graphique.