Travail pratique #3

Ce travail PEUT être fait en équipe de deux.

Aucun retard permis

Notions mises en pratique : Respect d'un ensemble de spécifications, les classes et les objets, les tableaux, et les exceptions.

1. Spécifications

Ce travail consiste à concevoir une classe `Contact` servant à modéliser un contact personnel, et une application de gestion d'un carnet de contacts qui utilisera la classe `Contact`.

Les **fichiers fournis** pour faire le TP (doivent se trouver dans votre projet) :

Fichier	Classe
Telephone.java	Classes `Telephone` et `TelephoneInvalideException` (À NE PAS MODIFIER)
Adresse.java	Classes `Adresse` et `AdresseInvalideException` (À NE PAS MODIFIER)
ContactInvalideException.java	Classe `ContactInvalideException` (À NE PAS MODIFIER)
CarnetContacts.java	Classe CarnetContacts (À COMPLÉTER)

1.1 Implémentation de la classe `Contact`

DANS UN PREMIER TEMPS, vous devez implémenter la classe `Contact` ayant les attributs et méthodes suivantes.

Attributs de classe

Nom de l'attribut	Type	Description
nbrContactsFavoris	`int`	Contient le nombre de contacts personnels qui ont été signalés en tant que contacts favoris.

Attributs d'instance

Nom de l'attribut	Type	Description
nom	`String`	Le nom de ce contact.
prenom	`String`	Le prénom de ce contact.
telephones	`Telephone []`	Les différents téléphones de ce contact. Ce tableau NE DOIT JAMAIS ÊTRE NULL.
nbrTelephones	`int`	Le nombre de téléphones associé à ce contact. À tout moment, cet attribut doit contenir le nombre de téléphones de ce contact, et doit donc être ajusté à mesure qu'on ajoute ou supprime des téléphones.
adresse	`Adresse`	L'adresse de ce contact.
courriel	`String`	Le courriel de ce contact.
favori	`boolean`	`true` si ce contact est un favori, `false` sinon.

***Note** : Évidemment, pour une modélisation plus fine, un contact devrait pouvoir posséder plusieurs adresses, et plusieurs courriels (comme pour les téléphones), mais dans le cadre de ce TP, les attributs énumérés ci-dessus sont suffisants.*

ATTENTION, les paramètres dans l'entête des constructeurs et méthodes doivent respecter l'ordre d'énumération dans les tableaux décrivant leurs paramètres.

Constructeurs

Constructeur 1 (5 paramètres) :

Paramètre	Type	Description
nom	`String`	Le nom du contact à construire. Ne peut être `null` ou vide.
prenom	`String`	Le prénom du contact à construire. Ne peut être `null` ou vide.
tel	`Telephone`	Un premier téléphone à ajouter au tableau de téléphones du contact à construire (Si `null`, n'est pas ajouté).
adresse	`Adresse`	L'adresse du contact à construire (peut être `null`).
courriel	`String`	Le courriel du contact à construire (peut être vide ou `null`)

Ce constructeur construit un objet contact en initialisant ses attributs avec les valeurs passées en paramètre. De plus :

- L'attribut `favori` doit être initialisé à false.
- Le tableau de téléphones est initialisé avec un tableau de DEUX cases, chacune contenant la valeur `null`.
- Si le paramètre `tel` est non `null`, il est ajouté au tableau de téléphones de ce contact. Sinon, aucun téléphone n'est ajouté. N'OUBLIEZ PAS de mettre à jour l'attribut `nbrTelephones`, s'il y a lieu.
- Si le paramètre `courriel` est vide, le courriel de ce contact est initialisé à `null`.
- Si le paramètre `nom` ou le paramètre `prenom` est `null` ou vide, le constructeur lève une exception `ContactInvalideException`.

Constructeur 2 (2 paramètres) :

Paramètre	Type	Description
nom	`String`	Le nom du contact à construire. Ne peut être `null` ou vide.
prenom	`String`	Le prénom du contact à construire. Ne peut être `null` ou vide.

Ce constructeur construit un objet contact en initialisant ses attributs `nom` et `prenom` avec les valeurs passées en paramètre. De plus :

- L'attribut `favori` est initialisé à `false`.
- Le tableau de téléphones est initialisé avec un tableau de DEUX cases, chacune contenant la valeur `null`.
- Aucun téléphone n'est ajouté au tableau de téléphones du contact à construire.
- Les attributs `adresse` et `courriel` sont initialisés à `null`.

Méthodes d'instance publiques – observateurs (*getters*)

Les 6 *getters* pour tous les attributs d'instance (sauf `telephones`) qui ne prennent aucun paramètre, et retournent la valeur de l'attribut d'instance concerné dont voici les noms (que vous devez respecter) :

`getNom, getPrenom, getAdresse, getCourriel, getNbrTelephones,` **`isFavori.`**

***Note** : lorsqu'un getter retourne un booléen, le style Java veut qu'on utilise le mot `is` au lieu de `get` : `isFavori`.*

Méthodes d'instance publiques – modificateurs (*setters*)

Les 5 *setters* suivants qui prennent en paramètre la valeur de modification de l'attribut, et ne retournent rien (`void`) :

Méthode	Précision
`setNom`	Si le paramètre est `null` ou vide, la modification n'est pas effectuée, et la méthode lève une exception `ContactInvalideException`.
`setPrenom`	Si le paramètre est `null` ou vide, la modification n'est pas effectuée, et la méthode lève une exception `ContactInvalideException`.
`setAdresse`	Aucune précision.
`setCourriel`	Si le paramètre est une chaine vide, l'attribut `courriel` de ce contact est plutôt modifié avec la valeur `null`.
`setFavori`	N'oubliez pas d'ajuster l'attribut de classe `nbrContactsFavoris` s'il y a lieu

Autres méthodes d'instance publique

Nom méthode : `ajouterTelephone`
Type retour : `void`

Paramètre	Type	Description
tel	`Telephone`	Le téléphone à ajouter au tableau de téléphones de ce contact.

Permet d'ajouter le `tel` donné en paramètre au tableau `telephones` de ce contact. N'oubliez pas d'ajuster l'attribut `nbrTelephones`, s'il y a lieu. De plus :

- Si `tel` est `null`, aucun téléphone n'est ajouté.
- Si `tel` n'est pas `null`, celui-ci est ajouté dans LA PREMIÈRE CASE LIBRE du tableau `telephones` de ce contact. Une case libre est une case contenant `null`.

  Par exemple, si `telephones = {tel1, tel2, **null**, tel3, null, null}`, le téléphone sera ajouté dans la case 2.
- S'il n'y a plus de place dans le tableau `telephones`, la méthode doit agrandir le tableau `telephones` de DEUX CASES supplémentaires avant d'y ajouter le `tel` donné en paramètre dans la première case libre.

Nom méthode : `obtenirIemeTelephone`
Type retour : `Telephone`

Paramètre	Type	Description
ieme	`int`	Spécifie le téléphone du tableau de téléphones de ce contact à retourner.

Permet d'obtenir le i^{ème} téléphone du tableau de téléphones de ce contact. S'il n'y a pas de i^{ème} téléphone dans le tableau de téléphones, la méthode retourne `null`.

Dans le tableau `telephones`, on détermine le i^{ème} comme suit :
- 1er = le premier téléphone non `null` dans le tableau de téléphones.
- 2^{ième} = le deuxième téléphone non `null` dans le tableau de téléphones.
- n^{ième} = le n^{ième} téléphone non `null` dans le tableau de téléphones.

Par exemple, si `telephones = {tel1, tel2, null, tel3, null, null}` :
`ieme` = 1 correspond à `tel1` dans le tableau.

`ieme` = 2 correspond à `tel2` dans le tableau.
`ieme` = 3 correspond à `tel3` dans le tableau.
`ieme` = 0 ou -1 ou 4 ne correspond à aucun téléphone dans le tableau.

Nom méthode : `supprimerTelephone`
Type retour : `boolean`

Paramètre	Type	Description
ieme	`int`	Spécifie le téléphone à supprimer du tableau de téléphones de ce contact.

Permet de supprimer le i^{ème} téléphone du tableau de téléphones de ce contact. S'il n'y a pas de i^{ème} téléphone dans le tableau de téléphones, aucun téléphone n'est supprimé. La méthode retourne `true` si la suppression a eu lieu, `false` sinon. N'oubliez pas d'ajuster l'attribut `nbrTelephones`, s'il y a lieu.

***Note** : voir la description de la méthode `obtenirIemeTelephone` pour savoir comment déterminer le i^{ème} téléphone.*

Nom méthode : `modifierTelephone`
Type retour : `void`

Paramètre	Type	Description
ieme	`int`	Spécifie le téléphone à modifier dans le tableau de téléphones de ce contact.
type	`String`	La nouvelle valeur pour le type du téléphone à supprimer.
numero	`String`	La nouvelle valeur pour le numéro du téléphone à supprimer.
poste	`String`	La nouvelle valeur pour le poste du téléphone à supprimer.

Permet de modifier le type, le numero, et le poste du i^{ème} téléphone de ce contact. De plus :

- Si le paramètre `ieme` ne correspond à aucun téléphone dans le tableau de téléphones de ce contact, aucune modification n'est effectuée.
- Si le paramètre `type` est `null`, le type du i^{ème} téléphone n'est pas modifié.
- Si le paramètre `poste` est `null`, le poste du i^{ème} téléphone n'est pas modifié.
- Si le paramètre `numero` est `null`, le poste du i^{ème} téléphone n'est pas modifié.
- Si le paramètre numero n'est pas `null`, et n'est pas valide, le numéro du i^{ème} téléphone n'est pas modifié, et la méthode lève une exception `ContactInvalideException`. Un numéro valide est une chaine ne contenant que des caractères numériques de longueur Telephone.LNG_NUM_1 ou Telephone.LNG_NUM_2.

***Note** : voir la description de la méthode `obtenirIemeTelephone` pour savoir comment déterminer le i^{ème} téléphone.*

Nom méthode : `toString`
Type retour : `String`
Paramètres : aucun

Retourne une représentation sous forme de chaine de caractères de ce contact. La chaine retournée doit respecter le format montré dans les exemples suivants.

1) Exemple avec un contact n'ayant aucun téléphone, pas d'adresse, pas de courriel, et qui n'est pas un favori :

```
DOE, Jane

TELEPHONE(S) : Aucun.

ADRESSE : Aucune.

COURRIEL : Aucun.
```

2) Exemple avec un contact ayant un téléphone, une adresse, un courriel, et qui est un favori :

```
DOE, Jane [FAVORI]

TELEPHONE(S) :
1. TEL. CELLULAIRE : (123) 456-7892, poste 23

ADRESSE :
34 Sherbrooke, apt. 3
Montreal, Que, CANADA
J3R 4H8

COURRIEL : janedoe@gmail.com
```

3) Exemple avec un contact ayant 3 téléphones, une adresse, un courriel, et qui est un favori :

```
DOE, Jane [FAVORI]

TELEPHONE(S) :
1. TEL. CELLULAIRE : (123) 456-7892, poste 23
2. TEL. RESIDENCE : (476) 287-6449
3. TEL. BUREAU : (476) 287-6449

ADRESSE :
34 Sherbrooke, apt. 3
Montreal, Que, CANADA
J3R 4H8

COURRIEL : janedoe@gmail.com
```

Précisions :

- Le nom du contact doit être en majuscules.
- Les étiquettes `[FAVORI]`, `TELEPHONE(S)`, `ADRESSE`, et `COURRIEL` sont en majuscules.
- Lorsqu'il y a des téléphones, les numéroter (dans l'ordre) avec leur $i^{ème}$ position dans le tableau de téléphones (voir la méthode `obtenirIemeTelephone` pour savoir comment déterminer le $i^{ème}$ téléphone).
- L'adresse est représentée sous le format retourné par la méthode `toString` de la classe `Adresse`.
- Chaque téléphone (à l'exception de la numérotation) est représenté sous le format retourné par la méthode `toString` de la classe `Telephone`.

Méthodes de classe publiques

Nom méthode : `modifierNbrContactsFavoris`
Type retour : `void`

Paramètre	Type	Description
nbr	`int`	La valeur de modification de l'attribut de classe `nbrContactsFavoris`.

Permet de modifier la valeur de l'attribut de classe `nbrContactsFavoris` par la valeur passée en paramètre.

Nom méthode : `obtenirNbrContactsFavoris`
Type retour : `int`
Paramètre : aucun.

Permet d'obtenir la valeur de l'attribut de classe `nbrContactsFavoris`.

Précisions sur l'implémentation de la classe `Contact`

- Vous devez respecter le principe d'encapsulation des données.
- Les méthodes énumérées dans cette section sont les seules méthodes publiques dans la classe `Contact`. Vous pouvez, ET DEVRIEZ ajouter des méthodes privées pour bien découper vos méthodes publiques et éviter la répétition du code.
- Les seules variables globales permises sont les attributs (de classe et d'instance) énumérés au début de cette section. Vous pouvez cependant ajouter des constantes de classe publiques (`public static final`) si vous le jugez pertinent.

***Note** : assurez-vous de bien tester votre classe `Contact` AVANT DE COMMENCER la classe `CarnetContacts`. Des erreurs dans les méthodes de la classe `Contact` peuvent provoquer des erreurs dans les méthodes de la classe `CarnetContacts` qui les utilisent.*

1.2 Implémentation de la classe d'application `CarnetContacts`

Vous devez compléter la classe `CarnetContact` qui est une application minimale servant à gérer un carnet de contacts personnels. Cette classe contient déjà quelques méthodes utilitaires que vous devriez utiliser pour coder les méthodes à compléter dans cette classe. Elle contient aussi des méthodes de sauvegarde et de lecture des contacts dans un fichier texte, que vous NE DEVEZ PAS UTILISER. Celles-ci sont déjà appelées dans la méthode `main`. Cette dernière est entièrement codée et ne doit pas être modifiée. ATTENTION, toutes les méthodes déjà implémentées dans la classe `CarnetContacts` NE DOIVENT PAS ÊTRE MODIFIÉES.

***Note** : Les entêtes des méthodes à compléter sont déjà écrits dans la classe `CarnetContact` (juste au-dessus de la méthode `main` qui se trouve à la fin de la classe).*

Le carnet de contacts est représenté en mémoire par un tableau d'objets de type `Contact`. De plus, une variable `nbrContacts` de type `int` conserve le nombre de contacts effectivement dans le tableau. Cette variable est donc mise à jour à chaque ajout ou suppression d'un contact dans le tableau (tout ceci se fait dans la méthode `main`).

Sauvegarde et lecture des contacts sauvegardés

Lorsque l'utilisateur quitte le programme, la méthode `sauvegarderContacts` (déjà codée et appelée dans la méthode `main`) s'occupe de sauvegarder tous les contacts du carnet, à la racine du projet, dans un fichier nommé *contacts.txt*. Aussi, au lancement du programme, la méthode `lireFichierContacts` (déjà codée et appelée dans la méthode `main`) vérifie s'il existe un fichier nommé *contacts.txt* à la racine du projet. Si oui, la méthode lit les contacts dans ce fichier, et recrée le tableau de contacts en mémoire pour que celui-ci contienne les mêmes contacts, dans le même ordre, que lors de la dernière fermeture (menu quitter) du programme. La variable contenant le nombre de contacts dans le tableau est aussi ajustée en conséquence. Sinon (ou si le fichier est vide), le tableau de contacts est initialisé avec une longueur égale à 2 où chacune des cases est initialisée à `null` (cases vides). La variable contenant le nombre de contacts est alors initialisée à 0. De cette manière, les données entrées ne sont pas perdues d'une utilisation à l'autre de l'application.

Ajout et suppression de contacts

Au premier démarrage de l'application, le tableau de contacts ne contient aucun contact, et est de longueur 2 (les deux cases étant initialisées à `null`). Les cases `null` représentent des places libres dans le tableau. L'**ajout** d'un contact au carnet doit se faire dans la première place libre (case `null`) trouvée dans le tableau de contacts. Si le tableau est plein (n'a plus de cases `null`), vous devez doubler la taille du tableau de contacts avant d'ajouter le contact dans la première case libre trouvée. Pour doubler la taille du tableau, disons `tab1`, vous devez d'abord créer un tableau 2 fois

plus grand, disons `tab2`. vous devez ensuite copier chacune des cases de `tab1` dans `tab2` : `tab2[i] = tab1[i]`, pour tout `i` allant de 0 à (longueur `tab1` – 1).

La **suppression** d'un contact du carnet consiste à affecter la valeur `null` à la case du tableau dans laquelle se trouve le contact à supprimer. Par exemple, voici une série d'ajouts/suppressions de contacts (`c1, c2, ...`) qui montrent le tableau (`contacts`) et la variable `nbrContacts` en mémoire :

```
Tab contacts = {null, null} //tableau initial

Ajout de c1         => Tab contacts = {c1, null}                                  nbrContacts = 1
Ajout de c2         => Tab contacts = {c1, c2}                                    nbrContacts = 2
Ajout de c3         => Tab contacts = {c1, c2, c3, null}                          nbrContacts = 3
Suppression de c1   => Tab contacts = {null, c2, c3, null}                       nbrContacts = 2
Ajout de c4         => Tab contacts = {c4, c2, c3, null}                          nbrContacts = 3
Ajout de c5         => Tab contacts = {c4, c2, c3, c5}                            nbrContacts = 4
Ajout de c6         => Tab contacts = {c4, c2, c3, c5, c6, null, null, null}      nbrContacts = 5
Suppression de c2   => Tab contacts = {c4, null, c3, c5, c6, null, null, null}    nbrContacts = 4
Suppression de c3   => Tab contacts = {c4, null, null, c5, c6, null, null, null}  nbrContacts = 3
Ajout de c7         => Tab contacts = {c4, c7, null, c5, c6, null, null, null}    nbrContacts = 4
```

Dans la classe `CarnetContacts` à compléter, la méthode `main` est déjà implémentée. Vous avez à compléter les méthodes qui traitent les 4 premières options du menu principal. Aussi, certaines méthodes utilitaires sont déjà implémentées, utilisez-les autant que possible.

Lancement du programme

Le programme débute avec une brève présentation du logiciel et offre un menu de 5 options :

```
Ce programme permet de gerer un carnet de contacts.


-----
MENU
-----
1. AJOUTER UN CONTACT
2. SUPPRIMER UN CONTACT
3. VIDER LE CARNET DE CONTACTS
4. AFFICHER LES CONTACTS
5. QUITTER

Entrez votre choix au menu :
```

Le choix au menu doit être validé. Les seules entrées valides sont 1, 2, 3, 4 ou 5 (déjà codé dans la classe `CarnetContacts`).

***Note** : Pour une application plus utilisable, il faudrait ajouter des options au menu pour permettre la recherche de contacts dans le carnet (par nom, prénom, etc.) pour modifier les contacts existants, etc. Dans le cadre de ce TP, nous nous en tiendrons au menu ci-dessus.*

Option 1 : ajouter un contact

1. Lorsque cette option est choisie, le programme demande d'abord les informations suivantes (dans cet ordre) :
   - Le nom du contact. Il doit être validé : chaine dont la longueur est entre 1 et 25 inclusivement.
   - Le prénom du contact. Il doit être validé : chaine dont la longueur est entre 1 et 25 inclusivement.
2. Le programme demande ensuite à l'utilisateur s'il veut entrer un numéro de téléphone (la réponse doit être validée : les 4 seules réponses valides sont 'o' ou 'O' pour oui, et 'n' ou 'N' pour non).

   Si la réponse est oui, le programme demande alors d'entrer les informations suivantes :
   - Le type de téléphone. Toutes les chaines sont valides. Note : si l'utilisateur entre la chaine vide, le type du téléphone = Telephone. TYPE_PAR_DEFAUT.

- Le numéro de téléphone. Il doit être validé : chaine exclusivement composée de caractères numériques, et de longueur 7 ou 10 uniquement). Utilisez la méthode `numeroTelValide` de la classe Telephone.
- Le poste téléphonique. Toutes les chaines sont valides. Une chaine vide signifie qu'il n'y a aucun poste.

***Note** : le programme doit permettre à l'utilisateur d'entrer autant de téléphones qu'il le désire, l'un à la suite de l'autre (voir exemples d'exécution).*

Si la réponse est non, le programme passe à l'étape suivante.

3. Le programme demande ensuite à l'utilisateur s'il veut entrer une adresse (la réponse doit être validée : les 4 seules réponses valides sont 'o' ou 'O' pour oui, et 'n' ou 'N' pour non).

   Si la réponse est oui, le programme demande alors d'entrer les informations suivantes :
   - Le numéro de porte. Il doit être validé : chaine de longueur comprise entre 1 et 8 inclusivement.
   - La rue. Elle doit être validée : chaine de longueur comprise entre 1 et 50 inclusivement.
   - Le numéro d'appartement. Toutes les chaines sont valides. Une chaine vide signifie qu'il n'y a pas d'appartement.
   - La ville. Elle doit être validée : chaine de longueur comprise entre 1 et 50 inclusivement.
   - La province (ou état…). Elle doit être validée : chaine de longueur comprise entre 1 et 50 inclusivement.
   - Le pays. Il doit être validé : chaine de longueur comprise entre 1 et 50 inclusivement.
   - Le code postal. Toutes les chaines sont valides. Une chaine vide signifie qu'on ne connaît pas le code postal pour cette adresse.

   Puis le programme passe à l'étape suivante.

   Si la réponse est non, le programme passe à l'étape suivante.

4. Le programme demande ensuite à l'utilisateur s'il veut entrer un courriel. La réponse doit être validée : les 4 seules réponses valides sont 'o' ou 'O' pour oui, et 'n' ou 'N' pour non). **Si la réponse est oui**, le programme demande alors d'entrer une adresse de courriel. Le courriel doit être validé : chaine de longueur comprise entre 5 et 100 inclusivement. Puis le programme passe à l'étape suivante. **Si la réponse est non**, le programme passe à l'étape suivante.

5. Le programme demande finalement à l'utilisateur s'il veut ajouter ce nouveau contact à ses favoris. La réponse doit être validée : les 4 seules réponses valides sont 'o' ou 'O' pour oui, et 'n' ou 'N' pour non. **Si la réponse est oui**, le contact est alors marqué comme étant un favori.

6. Finalement, le programme ajoute le nouveau contact dans le carnet (dans la première place libre du tableau de contacts), et affiche de nouveau le menu principal.

Vous devez compléter la méthode `ajouterContact` dans la classe `CarnetContacts`. Cette méthode doit traiter l'option 1 du menu principal. Elle prend en paramètre le tableau de contacts représentant le carnet de contacts (dans lequel on veut ajouter le nouveau contact), et retourne le tableau de contacts dans lequel on a ajouté le nouveau contact. Veuillez vous référer aux exemples d'exécution de cette option dans le fichier *exemplesAjouterContactEtAfficherContacts.txt* donné.

Option 2 : supprimer un contact

1. Lorsque cette option est choisie, le programme demande d'abord les informations suivantes (dans cet ordre) :
   - Le nom du contact à supprimer. Il doit être validé : chaine dont la longueur est entre 1 et 25 inclusivement.
   - Le prénom du contact à supprimer. Il doit être validé : chaine dont la longueur est entre 1 et 25 inclusivement.

2. Le programme trouve ensuite tous les contacts ayant le nom ET le prénom entrés. Si aucun contact n'est trouvé, le programme affiche un message à cet effet, et attend que l'utilisateur appuie sur la touche ENTER avant d'afficher de nouveau le menu principal. Si un ou plusieurs contacts sont trouvés, le programme les présente un à un à l'utilisateur en lui demandant, pour chaque contact, s'il veut vraiment le supprimer. La réponse oui/non pour confirmer chaque suppression doit être validée : les 4 seules réponses valides sont 'o' ou 'O' pour oui, et 'n' ou 'N' pour non). **Si la réponse est oui**, le contact est supprimé du carnet de contacts. **Si la réponse est non**, le contact n'est pas supprimé du carnet.

3. Le programme affiche de nouveau le menu principal.

Vous devez compléter la méthode `supprimerContact` dans la classe `CarnetContacts`. Cette méthode doit traiter l'option 2 au menu principal. Elle prend en paramètre le tableau de contacts représentant le carnet de contacts, et retourne le nombre de contact(s) effectivement supprimés. Veuillez vous référer aux exemples d'exécution de cette option dans le fichier *exemplesSupprimerContactEtAfficherContacts*.txt donné.

***Note** : Chaque fois que vous supprimez un contact FAVORI, n'oubliez pas d'ajuster la variable de classe `nbrContactsFavoris` dans la classe `Contact`. Vous aurez besoin de cette valeur pour l'affichage des favoris.*

Option 3 : vider le carnet de contacts

1. Lorsque cette option est choisie, le programme demande une confirmation à l'utilisateur. La réponse oui/non pour confirmer la suppression de tous les contacts doit être validée : les 4 seules réponses valides sont 'o' ou 'O' pour oui, et 'n' ou 'N' pour non).

   Si la réponse est oui :
   - Tous les contacts sont supprimés du carnet.
   - Le tableau de contacts représentant le carnet en mémoire est réinitialisé à une longueur = 2 (et ses cases sont initialisées à `null`).

   Si la réponse est non, aucun contact n'est supprimé.

2. Le programme affiche de nouveau le menu principal.

Vous devez compléter la méthode `viderCarnet` dans la classe `CarnetContacts`. Cette méthode doit traiter l'option 3 du menu principal. Elle prend en paramètre le tableau de contacts représentant le carnet de contacts, et retourne `true` si l'utilisateur a confirmé la suppression de tous les contacts, `false` sinon. Veuillez vous référer aux exemples d'exécution de cette option dans le fichier *exemplesViderCarnetEtAfficherContacts.txt* donné.

***Note** : N'oubliez pas d'ajuster la variable de classe `nbrContactsFavoris` dans la classe `Contact`.*

Option 4 : Afficher les contacts

1. Lorsque cette option est choisie, le programme affiche d'abord le sous-menu suivant :

```
************************
* AFFICHER LES CONTACTS *
************************

1. AFFICHER TOUS LES CONTACTS
2. AFFICHER LES FAVORIS

Entrez votre choix :
```

2. Si l'utilisateur choisit l'option 1, tous les contacts seront affichés. S'il choisit l'option 2, seuls les contacts favoris seront affichés. Le choix au sous-menu doit être validé. Les deux seules réponses valides sont 1 et 2.

   S'il y a des contacts à afficher, le programme affiche d'abord « CARNET DE CONTACTS » puis le nombre de contacts qui seront affichés, entre parenthèses (nombre de tous les contacts si option 1 choisie ou nombre de favoris si option 2 choisie). Puis, il affiche le premier contact, et ensuite tous les autres contacts un à la fois, en demandant à l'utilisateur d'appuyer sur ENTRÉE avant d'afficher le contact suivant. Lorsque le dernier contact est affiché, le programme affiche finalement « FIN DE LA LISTE DE CONTACTS. », et demande une dernière fois à l'utilisateur de saisir ENTRER pour continuer (et revenir au menu principal).

   S'il n'y a aucun contact à afficher (carnet vide ou aucun favori, par exemple), le programme affiche simplement :

```
CARNET DE CONTACTS (0)


FIN DE LA LISTE DE CONTACTS.

Appuyez sur "ENTREE" pour continuer...
```

3. Finalement le programme affiche de nouveau le menu principal.

Vous devez compléter la méthode `afficherContacts` dans la classe `CarnetContacts` qui traite l'option 4 du menu principal. Cette méthode prend en paramètre le tableau de contacts représentant le carnet de contacts, ainsi que le nombre total de contacts présents dans le carnet. Veuillez vous référer aux exemples d'exécution dans les divers fichiers d'exemples d'exécution donnés.

***Note** : le format d'affichage de chaque contact est celui retourné par la méthode `toString` de la classe `Contact` (donc utilisez-la !), avec seulement une petite modification à faire au niveau de la première ligne.*

Précisions sur l'implémentation de la classe `CarnetContacts`

- Lorsqu'on vous demande de valider une valeur saisie par l'utilisateur, cela sous-entend une BOUCLE de validation qui sollicite la valeur, lit la valeur, et lorsque la valeur saisie est invalide, elle affiche un message d'erreur, elle sollicite et lit de nouveau la valeur, et ce tant que la valeur saisie n'est pas valide.
- Vous DEVEZ respecter à la lettre les **informations, les messages**, et le **format** de l'affichage qui sont montrés dans les exemples d'exécution fournis avec l'énoncé du TP.
- Toute **VARIABLE GLOBALE** est **INTERDITE : toutes les variables doivent être déclarées à l'intérieur d'une méthode** (sauf pour les boucles for)**.**
- Vous devez découper les méthodes à compléter en appelant d'autres méthodes. Vous pouvez utiliser les méthodes « utilitaires » déjà codées dans la classe `CarnetContacts`, et devez/pouvez aussi en ajouter d'autres.
- Toutes les méthodes de cette classe doivent être `static`.
- Prenez soin de bien modulariser votre code à l'aide de méthodes cohésives et robustes. Voici quelques lignes directrices pour vous guider dans la conception de vos méthodes :
  - Chaque fois que vous vous apercevez que différentes parties de votre code se ressemblent énormément, essayer d'en faire une méthode bien paramétrée (si possible).
  - Faites une méthode pour chaque petite tâche spécifique à accomplir dans la résolution du problème.
  - Si une méthode semble trop longue, posez-vous la question à savoir si elle pourrait être subdivisée en plusieurs autres méthodes plus spécialisées (cohésives).
- Vous DEVEZ utiliser la classe `Clavier.java` pour effectuer toutes les saisies.
- Votre programme NE DOIT JAMAIS PLANTER lors d'une saisie faite par l'utilisateur.
- Utilisez des constantes (`final`) autant que possible.
- Les constantes doivent être déclarées et initialisées au niveau de la classe : `public static final`…
- L'affichage des résultats doit se faire à la console.
- Utilisez des variables réelles uniquement lorsque requis. Par exemple, un compteur ne doit pas être `float` ou `double`.
- Votre code doit compiler et s'exécuter avec le JDK 7.
- Vous DEVEZ respecter le style Java vu en classe.

2. Précisions supplémentaires

- **Le non-respect de toute spécification ou consigne se trouvant dans l'énoncé du TP (et les exemples d'exécution donnés avec l'énoncé du TP) est susceptible d'engendrer une perte de points.**
- Vous ne DEVEZ PAS modifier ce qui est déjà implémenté dans la classe `CarnetContacts`.
- Dans vos deux classes à remettre, toutes les méthodes (sauf `main`) doivent être documentées correctement à l'aide des commentaires de documentation, en respectant le style Java.

- Les classes `Contact` et `CarnetContacts` (et les autres classes fournies) doivent se trouver dans le paquetage par défaut.
- Votre code doit compiler et s'exécuter avec le JDK 7.
- Vous DEVEZ respecter le style Java vu en classe.
- **N'oubliez pas d'écrire vos noms complets et codes permanents dans l'entête des deux classes à remettre.**

Note : *Si quelque chose est ambigu, obscure, s'il manque de l'information, si vous ne comprenez pas les spécifications, si vous avez des doutes… vous avez la responsabilité de vous informer auprès de votre enseignante.*

3. Détails sur la correction

3.1 La qualité du code (40 points)

Concernant les critères de correction du code, lisez attentivement le document "**Critères généraux de correction du code Java**" et le document **Conventions sytle Java**, dans la section CONTENU DU COURS / Travaux, sur Moodle.

Note : *Votre code sera corrigé sur la totalité ou une partie des critères de correction indiqués ci-dessus, ainsi que sur le respect des spécifications/consignes mentionnées dans ce document.*

3.2 L'exécution (60 points)

Un travail qui ne compile pas se verra attribuer la note 0 pour l'exécution.

Note : *Votre code sera testé en tout ou en partie. Les points seront calculés sur les tests effectués. Ceci signifie que si votre code fonctionne dans un cas x et que ce cas x n'est pas testé, vous n'obtiendrez pas de points pour ce cas. Il est donc dans votre intérêt de vous assurer du bon fonctionnement de votre programme dans tous les cas possible.*

4. Date et modalité de remise

4.1 Remise

Date de remise : 12 décembre 2016 avant minuit (ou le 13 décembre À MINUIT).

Les 2 fichiers à remettre : `Contact.java` et `CarnetContacts.java` (complété) – REMETTEZ LES DEUX FICHIERS SÉPARÉMENT (PAS DANS UNE ARCHIVE ZIP, RAR…)

Remise via Moodle uniquement.

Vous devez remettre (téléverser) vos deux fichiers sur le site du cours (Moodle). Vous trouverez la boîte de remise dans la section **BOÎTES DE REMISE - TRAVAUX PRATIQUES / REMISE DU TP3**.

POUR LA REMISE D'UN TRAVAIL FAIT EN ÉQUIPE :

AVANT la date de remise du TP3, assurez-vous que votre équipe soit bien inscrite dans la liste des équipes présente sur Moodle, dans la section CONTENU DU COUS / Travaux / TP3. Sinon, écrivez-moi un courriel me spécifiant les membres de votre équipe (**en mettant l'autre coéquipier en copie**). Le courriel doit être formé comme suit :

SUJET du message : INF1120-**GROUPE** : Équipe TP3
CORPS du message : vos 2 codes permanents doivent être dans le corps du message.

Vous devez remettre vos fichiers sur UN SEUL compte Moodle, celui du membre de l'équipe dont le code permanent est le plus petit (alphabétiquement parlant).

4.2 Politique concernant les retards

AUCUN RETARD NE SERA ACCEPTÉ.

4.3 Remarques générales

- Un programme ne compilant pas se verra attribuer la note 0 pour l'exécution du programme.
- Aucun programme reçu par courriel ne sera accepté.
- Vous avez la responsabilité de conserver des copies de sauvegarde de votre travail (sur disque externe, Dropbox, Google Drive, etc.). La perte d'un travail due à un vol, un accident, un bris… n'est pas une raison valable pour obtenir une extension pour la remise de votre travail.
- **N'oubliez pas d'écrire votre nom complet et votre code permanent (ou vos noms complets et codes permanents si vous êtes en équipe de deux) dans l'entête des deux classes à remettre.**
- N'attendez pas à la dernière minute pour commencer le travail, vous allez fort probablement rencontrer des problèmes inattendus!

Le règlement sur le plagiat sera appliqué sans exception. Vous devez ainsi vous assurer de ne pas échanger du code avec des collègues ni de laisser sans surveillance votre travail au laboratoire. Vous devez également récupérer rapidement toutes vos impressions de programme au laboratoire.

BON TRAVAIL !