Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2018 – 2019
DEV1

DEV1 – Laboratoires Java I

TD 13 – Mise en pratique
La gestion des clients

1 Introduction

Dans ce TD, on vous propose de mettre en pratique les notions vues auparavant pour créer un programme gérant l'enregistrement de nouveaux clients d'une entreprise (imaginaire). De nombreuses données différentes peuvent être associées à un client d'une entreprise donnée : son identifiant (numéro d'identification au sein de l'entreprise), son nom, son numéro de compte courant, son adresse postale, son adresse e-mail, sa date de naissance, sa nationalité, ... Pour simplifier les choses dans ce TD, nous considérerons uniquement les identifiants, noms, prénoms et l'adresse e-mail. Tous ces champs sont enregistrés sous forme de chaînes de caractères (des `String`). Nous considérerons qu'un identifiant client est simplement une suite de 10 chiffres décimaux. Le programme que nous vous demandons de concevoir devra maintenir à jour un tableau de clients et proposer à l'utilisateur (qui sera un employé de l'entreprise) le menu suivant :

```
**** Que souhaitez-vous faire ? ****

1 -- Ajouter un nouveau client (tapez 1),
2 -- Supprimer un client (tapez 2),
3 -- Afficher les clients déjà encodés (tapez 3),
4 -- Quitter le programme (tapez 4),

Votre choix:
```

La section suivante vous guide étape par étape dans la réalisation d'un tel programme.

2 Développement

Exercice 1

Créer la structure Client

Déclarer une nouvelle structure `Client` comportant les champs repris dans l'introduction : identifiant, nom, prénom, adresse-e-mail.

Vous prendrez soin de déclarer un constructeur approprié ainsi qu'une méthode `toString()` renvoyant une description du client (il suffit pour ce faire de concaténer les différents champs les uns à la suite des autres).

Version du document
30 novembre 2018

Exercice 2 — Créer un tableau de clients

Créer une classe `GestionClients` qui contiendra la fonction `main` de notre programme. Nos clients seront enregistrés dans un tableau de structures `Client`. Mais comme notre but est de proposer à l'utilisateur d'ajouter/supprimer des clients comme il le souhaite, il n'est pas possible de savoir à l'avance le nombre de clients que le tableau contiendra à un moment donné de l'exécution du programme. Nous allons donc supposer que le nombre de clients ne dépassera pas une certaine borne (disons 200) qui sera codée comme une constante du programme (et qu'on peut donc facilement modifier au besoin) :

```
public class GestionClients
{
   public static void main (String args[])
   {
         final int MAX_CLIENTS = 200;

         Client [] clients = new Client[MAX_CLIENTS];
   }
}
```

Mais alors, comment savoir combien de clients contient effectivement le tableau ? Comme on souhaite permettre à l'utilisateur de supprimer et d'ajouer des clients quand il veut, ce ne sera certainement pas `clients.length` qui nous fournira la réponse (ceci donnera toujours le résultat `MAX_CLIENTS`).

En réalité, tant que la case i du tableau (pour i entre 0 et `MAX_CLIENTS-1`) n'a pas été initialisée avec un nouveau client, cette case contiendra la valeur `null`. Les cases de notre tableau qui contiennent effectivement des clients seront donc toutes celles dont la valeur n'est pas égale à `null`. On convient de rassembler tous les clients en début de tableau.

Exercice 3 — Afficher le menu

Compléter la méthode `main` pour qu'elle affiche le menu décrit dans l'introduction. Le menu s'affichera à nouveau tant que l'utilisateur n'a pas choisi l'option quitter (choix numéro 4). Pour ce faire, vous pouvez créer une méthode :

```
static int afficherMenu ()
```

qui affiche le menu, demande à l'utilisateur de rentrer un choix, et renvoie le résultat.

Exercice 4 — Ajouter un nouveau client

Écrire une méthode :

```
static void ajoutClient (Client [] clients)
```

qui demande un identifant, un nom, un prénom et une adresse e-mail à l'utilisateur et ajoute un nouveau client ayant exactement ces données après la dernière case utilisée du tableau `clients`.

Si toutes les cases du tableau sont déja utilisées, la méthode renverra une `UnsupportedOperationException`. Compléter la méthode `main` (à vous de prendre garde a ne pas permettre à l'utilisateur de rentrer plus de `MAX_CLIENTS` clients, grâce à un message d'erreur.)

Exercice 5 — Afficher les clients encodés

Écrire une méthode

```
static void afficher(Client clients[])
```

qui affiche à l'écran, pour chaque client du tableau `clients[]` les donnés du client. Cette méthode fera appel à la méthode `toString()` de la classe `Client`. Remarque : si le tableau ne contient aucun client (toutes les cases contiennent `null`), la méthode affichera "Aucun client n'est encore encodé".

Exercice 6 Vérifier le format de l'adresse e-mail

L'utilisateur peut se tromper en entrant les données du client. Dans notre exemple imaginaire on va simplement vérifier que l'adresse contient exactement un symbole `@` suivi d'au moins un point, c'est-à-dire a la forme suivante :

`s1@s2.s3`

où s1,s2,s3 sont des chaînes de caractère ne contenant pas de `@`. Écrire une méthode

```
boolean verifierEmail(String email)
```

qui renvoie `true` si email a le format d'une adresse e-mail et `false` sinon. Mettre la méthode `ajoutClient` à jour : si l'utilisateur entre une mauvaise adresse pour un certain client, le programme refuse de créer le nouveau client et lui demande de corriger l'adresse entrée.

Exercice 7 Vérifier l'absence de doublons

Bien que ce soit rare, il est possible que deux clients aient le même nom et le même prénom. Par contre, ils ne peuvent avoir le même identifiant au sein de la société. Écrire une méthode

```
boolean verifierID(Client[] clients, String id)
```

qui renvoie `true` si l'identifiant `id` est déja utilisé dans le tableau `clients`, et `false` sinon. Adapter la méthode `ajoutClient` de sorte que l'utilisateur ne puisse jamais encoder deux clients avec le même identifiant.

Exercice 8 Supprimer un client

Écrire une méthode :

```
void supprimer (Client [] clients, String id)
```

qui supprime le client dont l'identifiant est donné par `id`. Si aucun client ayant un identifiant égal à `id` n'est trouvé, la méthode déclenchera une `NoSuchFieldException`.

Attention : pour supprimer un client, mettre la case correspondante du tableau à `null` ne suffit pas. Tous les clients doivent être regroupés en début de tableau (sans "trou").

Mettre la méthode `main` a jour. A vous de prendre garde à ce que l'utilisateur n'essaie pas de supprimer un client inexistant.