Cahier des charges : Javacard

Projet de développement d'application Desktop

Contents

I. Introduction

L'objectif de ce projet est de développer une application "desktop" Java permettant de gérer une liste de contacts et d'exporter les informations sous différents formats.

Choix technologiques :

- l'application doit être codée en Java (version 17 ou ultérieure)
- l'interface graphique devra être construite avec JavaFX
- les vues de l'application devront être codées en FXML
- pas de contrainte sur les bibliothèques utilisables

Modalités de travail :

- date de début du projet : 26/07/2024
- présentation du projet : 07/08/2024
- équipes de 2
- gestion du projet inspiré par la méthode **Scrum**
- utilisation d'un dépôt Git partagé

Livrables attendus :

- code source de l'application
- diagrammes UML éventuels
- "zoning" et "wireframe" de l'interface graphique

II. Fonctionnalités de l'application

Vous trouverez dans cette partie du cahier des charges une liste des fonctionnalités attendues pour le logiciel, ci-dessous un tableau récapitulatif. Chaque fonctionnalité est accompagné de son indicateur MoSCoW:

Fonctionnalité	MoSCoW
Interface graphique ergonomique proposant une gestion des contacts (CRUD sur les contacts)	**M**
Sauvegarde des informations de contact pour les recharger lorsque le logiciel se lance	**M**
Export d'un ou plusieurs contacts en **JSON**	**M**
Export d'un ou plusieurs contacts en **vCard**	**M**
Rechercher dans la liste des contact	**S**
Générer un QRCode contenant les informations de la vCard et l'afficher à l'écran	**C**
Export d'un ou plusieurs contacts en **CSV**	**C**

Dans la suite de ce document, vous trouverez des informations détaillées sur ces différentes fonctionnalités.

II.1. Gestion de contacts

L'application devra proposer une **interface graphique ergonomique** permettant de gérer une liste de contacts.

Les opérations attendues sont les opérations **CRUD** :

- consultations d'une liste de contact(s)
- création de contact(s)
- modification de contact(s)
- suppression de contact(s)

Une contact est défini par les caractéristiques suivantes :

- **nom** (obligatoire)
- **prénom** (obligatoire)
- **genre** : homme/femme/non-binaire (obligatoire)
- **date de naissance** (optionnel)
- **un pseudonyme** (optionnel)
- **adresse** (obligatoire)
- **numéro de téléphone personnel** (obligatoire)
- **numéro de téléphone professionnel** (optionnel)
- **une adresse email** (obligatoire)
- **une adresse postale** (obligatoire)
- **un lien vers la page Github ou Gitlab du contact** (optionnel)

II.1.1. Affichage des contacts et création/modification

Un **tableau** qui liste tous les contacts gérés par l'application est à intégrer à l'interface principale.

Sur la même vue, un formulaire de création/modification devra être intégrer. Le clic sur une ligne du tableau devra pré-remplir les champ du formulaire pour permettre la modification de l'élément.

Veillez à bien vérifier la validité des informations saisies par l'utilisateur pour les champs :

- adresse email
- lien vers la page Github ou Gitlab

Si l'utilisateur saisit des données incorrectes il faudra une **indication visuelle** explicite indiquant le champ est en erreur.

La vérification de ces informations pourra se faire en utilisant des **expressions régulières**.

Une expression régulière est une chaîne de caractères type qui correspond à un **motif** (ou "pattern") de caractères possibles. Ce motif permet de trouver une **correpondance dans une autre chaîne de caractère** (c'est le "matching").

Afin de découvrire le fonctionnement des expressions régulières il vous est conseillé de suivre le tutoriel en ligne ReagexLearn (les 30 premiers exercices vous permettront de bien cerner le fonctionnement).

Une fois le fonctionnement des expressions régulières compris, vous pourrez vous référer à l'article disponible ici pour apprendre à les implémenter en Java.

II.1.2. Chargement de la liste des contacts

Il devra être possible de sauvegarder la liste des utilisateurs dans un fichier binaire afin de pouvoir les recharger lors du lancement de l'application.

Pour se faire vous pouvez mettre en place un système de **sérialisation binaire** de la liste de tous les contacts.

II.2. Sauvegarde des contacts dans des fichiers

II.2.1. Export d'un fichier vCard

II.2.1.1. Qu'est-ce qu'une vCard ?

vCard est un format standard ouvert d'échange de données personnelles ("Visit Card"). Une vCard est un fichier de contact qu'il est possible d'utiliser afin de partager les coordonnées d'une personne.

La structuration d'un vCard en version 4 est définie dans la RFC6350 développée par l'IETF.

Vous trouverez, en ligne, de nombreux exemples de vCard. Voici un exemple fonctionnel de vCard4 :

```
BEGIN:VCARD
VERSION:4.0
N:Gump;Forrest;;Mr.;
FN:Sheri Nom
ORG:Sheri Nom Co.
TITLE:Ultimate Warrior
PHOTO;MEDIATYPE=image/gif:http://www.sherinnom.com/dir_photos/my_photo.gif
TEL;TYPE=work,voice;VALUE=uri:tel:+1-111-555-1212
TEL;TYPE=home,voice;VALUE=uri:tel:+1-404-555-1212
ADR;TYPE=WORK;PREF=1;LABEL="Normality\nBaytown\, LA 50514\nUnited States of
America":;;100 Waters Edge;Baytown;LA;50514;United States of America
ADR;TYPE=HOME;LABEL="42 Plantation St.\nBaytown\, LA 30314\nUnited States of
America":;;42 Plantation St.;Baytown;LA;30314;United States of America
EMAIL:sherinnom@example.com
REV:20080424T195243Z
x-qq:21588891
END:VCARD
```

Il vous faudra prévoir une fonctionnalité permettant de sauvegarder dans un fichier ayant pour extension **".vcf"** les contacts gérés via votre interface graphique.

La section suivante vous guidera sur la façon de mener l'analyse de la structure des vCard 4 afin de comprendre quoi inscrire dans le fichier.

II.2.1.2. Ecriture dans un fichier de la vCard

La première étape est d'arriver à constuire une chaîne de caractères contenant les informations de la vCard.

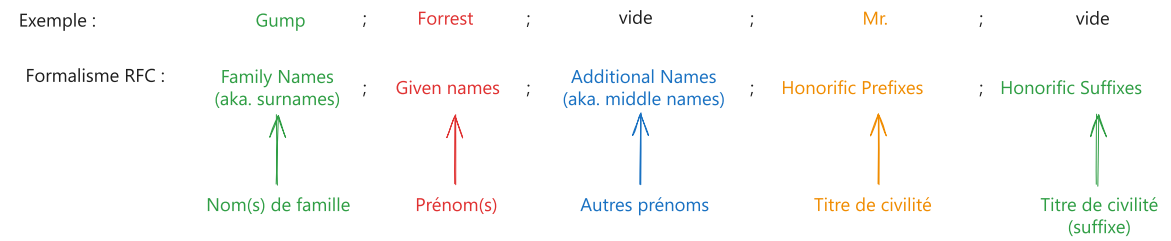
Avant de pouvoir implémenter la fonctionnalité, vous allez devoir analyser le contenu d'une vCard (par exemple celle fournie en exemple) afin d'en comprendre la structuration.
Cette structuration est détaillée dans la RFC6350.

Voici un premier exemple, concernant le nom du contact :

```
N:Gump;Forrest;;Mr.;
```

`N:` indique que l'information qui suit concerne le nom du contact.

Ci-dessous une description de cette chaîne de caractères :

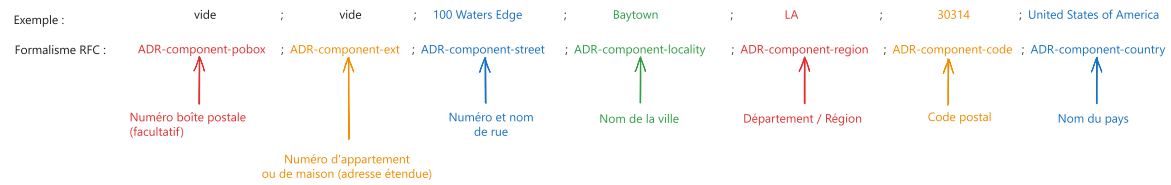


Nous observons que de nombreux champs sont facultatifs et peuvent rester vides.

Voici un autre exemple, il s'agit ici de la ligne comprenant les informations d'adresse personnelle :

```
ADR;TYPE=HOME;LABEL="42 Plantation St.\nBaytown\, LA 30314\nUnited States of
America":;;42 Plantation St.;Baytown;LA;30314;United States of America
```

Ci-dessous un descriptif de la partie située après les ":" suite à la chaîne de caractères définissant le "LABEL".



Une fois que ceci est fait il faut écrire cette chaîne dans un fichier.

Pour écrire dans un fichier vous pouvez vous inspirer des solutions présentées par cet article (la **"méthode 1"** est, par exemple, exploitable).

II.2.1.3. Tester ses vCard

Une fois vos vCards créées, il vous faudra les tester. Vous pouvez utiliser un outil d'affichage de contenu de vCard tel que vCard Editor sous Windows.

II.2.2. Sauvegarde dans un fichier JSON

II.2.2.1. Détails de la fonctionnalité

En plus de la sauvegarde en vCard, l'application devra permettre sauvegarder les contacts en utilisant le **format JSON**.

JSON est un format de données textuel sous forme de tableau associatif (stockage "clé-valeur").

Voici, par exemple, un extrait de JSON représentant les données d'une partie de la classe "Contact" :

```
{
  "firstName": "Ada",
  "lastName": "Lovelace",
  "nickname": "Mothergoddess"
  "birthDate": "10-12-1815",

  ...
}
```

Vous pourrez découvrir le format JSON à l'aide de cette vidéo de Grafikart ou en consultant cet article.

II.2.2.2. Conseils d'implémentation

Votre objectif est de récupérer les **informations de contacts issues d'objets** et de sauvegarder ces informations dans un fichier ayant pour extension **".json"**.

Afin d'écrire des objets dans un JSON, vous pourrez utiliser une des nombreuses bibliothèques permettant de le faire, par exemple :

- json-simple : tutoriel disponible ici
- jackson : tutoriel disponible ici
- moshi : tutoriel disponible ici

`json-simple` semble être une solution à la mise en place aisée qui permet de mener à bien la tâche demandée.

III. Conception graphique

III.1. Contraintes graphiques

Il vous est demandé de concenoir une application compatible avec les tailles de fenêtre allant de **1024px × 768px** à **1920px x 1080px**.

L'organisation des composants graphiques devra rester harmonieuse pour toute les tailles intermédiaires.

III.2. Conception de "wireframe"

Pour ce projet, il ne vous est pas demandé de mener une recherche graphique poussée (le design importe peu, vous allez vous concentrer sur les fonctionnalités) mais il vous faudra faire en sorte de développer une interface la plus simplet et ergonomique possible.

Habituellement, pour construire une maquette graphique, vous devriez suivre les phases suivants :

1. **zoning** : schématisation grossière de ce sera que la vue de l'application. On y indique uniquement des rectangles qui correspondent à des blocs de composants graphiques logiquement liés.
2. **wireframe** : version plus précise du zoning sur lequel on ajoute du texte et des détails sur les composants graphiques utilisés. Attention cependant, il n'y a pas encore de notion de "design" dans cette phase
3. **maquette haute fidelité / mockup** : développement graphique détaillé se voulant au plus proche du design final (avec l'intégration des couleurs, des images...)

Vous trouverez des exemples de ces 3 phases dans l'article disponible ici.

Pour ce projet il vous est fortement conseillé de constuire une "zoning" et un "wireframe" pour 2 résolutions spécifiques :

- 1024px × 768px
- 1920px x 1080px

La maquette haute fidelité importe moins dans notre cas du fait qu'il n'est pas demandé de mettre en place un style complexe.

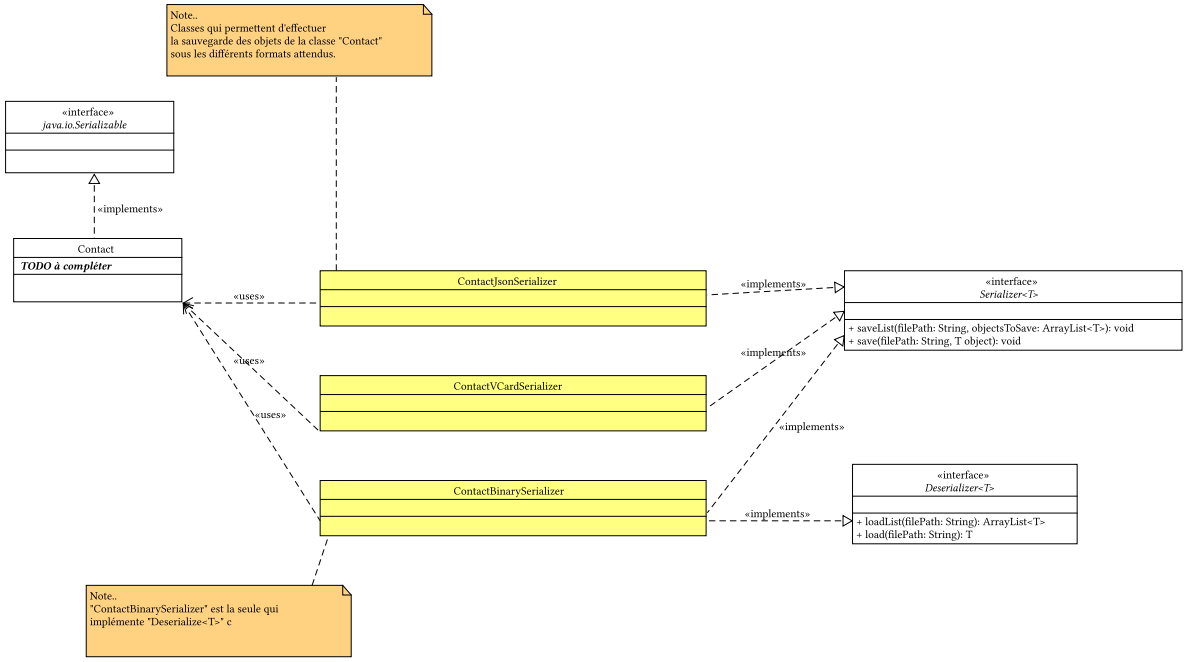
Pour dessiner ces "zoning" et "wireframe" deux choix s'offrent à vous :

- dessiner sur une feuille
- dessiner en utilisant un outil collaboratif en ligne tel que "Figma"

Attention, veillez à faire valider votre maquette par le client avant de commencer à coder.

IV. Proposition d'achitecture logicielle

Ci-dessous une proposition d'architecture logicielle utilisant le langage UML (hors fonctionnalités optionelles) :



V. Phases de travail

Votre équipe pourra suivre les phases de travail détaillées suivantes :

1. Analyse fonctionnelle :
    - lecture du cahier des charges
    - définition des "persona"
    - construction d'un "backlog" avec les "user stories"
2. Maquettage de l'application :
    - création du "zoning" et du "wireframe"
    - validation de l'interface avec le "product owner" (autrement dit : votre formateur)
3. Conception de l'application (réflexion architecturale + conception de diagrammes)
4. Préparation du "sprint" et début des développements
5. Tests fonctionnels au cours des développements
6. Livraison de l'application : **07/07/2024**

VI. Fonctionnalités ‘C’ (“Could have this…”)

Si le temps vous le permet, vous pourrez implémenter les fonctionnalités non prioritaires de l’application :

Fonctionnalité	MoSCoW
Générer un QRCode contenant les informations de la vCard et l’afficher à l’écran	C
Export d’un ou plusieurs contacts en CSV	C

Pour la génération d’un QRCode pour pourrez utiliser une bibliothèque telle que ZXing.

En ce qui concerne l’export des contacts CSV il faudra permettre à l’utilisateur de choisir le séparateur qu’il souhaite utiliser (par défaut la ‘,’ sera proposée).