Ceri - Université d'Avignon
Master Informatique 2017-2018

M1: Génie Logiciel Avancé
Prototypage d'interface utilisateur

TP 2 / 2 : Prototypage fonctionnel et avancé avec Netbeans Platform (énoncé)

Objectif du TP 2

La seconde partie de ce TP a pour but de vous faire manipuler le framework complet de **NetBeans Platform** comme support pour le prototypage de l'application, en intégrant des composants **JavaFX**. Ici il est question de réaliser un prototypage avancé en vue d'un développement d'une application riche, complexe et évolutive.

Pour ce TP, vous allez travailler dans un projet NetBeans différent du TP précédent afin de ne pas mélanger les sources.

Évaluation du TP

Il vous est demandé de rendre le projet NetBeans du TP 1 avec les sources, et tous les projets Netbeans du TP 2 avec les sources (platform + module).

Adresse courriel Pro : tehdy.draoui@kware.fr

Date limite de rendu: Lundi 09 avril 2018 inclus.
Dépôt sur E-UAPV uniquement.

NetBeans Platform 8.x est une solution pure Java/Swing, avec un système intégré de positionnement des fenêtres, une structure basée sur des modules/plugins, une gestion avancée du cycle de vie des objets par lookup, des mises à jours par le réseau ... et tout un ensemble d'API qui fournissent des fonctionnalités de plus haut niveau que les Swing/JavaFX.

1. Création du projet

→ Créer un nouveau projet « Netbeans Module » de type « NetBeans Platform Application » que vous nommerez « ContactManagerApplication». Le squelette de l'application est automatiquement généré. Ce projet permet de définir l'ensemble des modules qui vont enrichir l'application.

→ Ajouter un nouveau module nommé «ContactManagerDataModule ». Utiliser le nom de paquetage « m1.piu.data » dans le « Code Name Base ». Ce module va contenir les données de l'application de gestion des contacts (AdressBook, Contact, etc...). Ce module doit exposer ses classes aux autres modules.

→ Ajouter un nouveau module nommé «ContactManagerUIModule ». Utiliser le nom de paquetage « m1.piu » dans le « Code Name Base » pour faciliter la réutilisation des éléments de la maquette conçue au TP précédent. Ce module va contenir l'interface utilisateur de l'application de gestion des contacts. Ce module doit avoir accès aux classes de données du module «ContactManagerDataModule ».

→ Ajouter dans le module « ContactManagerUIModule » le fichier « m1/piu/layer.xml », depuis le menu "*File > New File… > Module Development > XML Layer*".

→ Exécuter votre application et observer les éléments « par défaut » affichés dans l'interface utilisateur.

Définition des dépendances

Avant de porter l'application vers NetBeans Platform, il est nécessaire de s'assurer que toutes les librairies nécessaires sont bien présentes dans le class path:

Propriété du module→Libraries→
- Base Utilities API
- Lookup API
- Settings API
- UI Utilities API
- Utilities API
- Window System API

Pour utiliser des fonctionnalités avancées offertes par Netbeans Platform, vous pouvez regarder la documentation des librairies suivantes (non traité dans ce TP):

- File System API
- JavaHelp Integration
- Look & Feel Customization Library
- Module System API
- Options Dialog and SPI
- Tab Control
- Text API

2. Personnalisation du menu et de la barre d'état

→ Élaguer la « Menu Bar » et les « Toolsbar » en supprimant les items qui sont présents par défaut. Pour cela, il suffit de modifier la structure définie dans le fichier « layer.xml » du module. Ce fichier se manipule graphiquement, en dépliant l'élément « layer.xml-><this layer in context>->Menu Bar » dans la vue « Projects ». Les éléments supprimés seront marqués « instance_hidden ».

→ Ajouter dans la barre de menu une action « Clear » qui vide un carnet d'adresse (ne pas implémenter l'action!). Pour créer une action, NetBeans propose un assistant disponible dans la création d'un nouveau fichier, catégorie «Module Development», type « Action ». Des règles de positionnement et de déclenchement peuvent être appliquées.

→ Ajouter dans la barre d'outils un composant JavaFX « ToolBar » correspondant à la barre d'outil FXML conçue au TP précédent.

→ Définir une barre d'état personnalisée contenant un label en utilisant le mécanisme de déclaration de services par annotation:

- créer une classe « MyStatusBar » de type « javax.swing.JLabel » (ou reprenez la barre d'état conçue au TP 1 en FXML)
- créer une classe « MyStatusLineElementProvider » qui implémente l'interface « org.openide.awt.StatusLineElementProvider » et qui mettra à disposition une instance de votre barre d'état
- annoter votre fournisseur de barre d'état « MyStatusLineElementProvider » pour déclarer le service de contribution à la barre d'état de NetBean.

3. Branding de l'application

→ Changer le titre de l'application dans l'onglet « Basic » du menu « Branding ».

→ Personnaliser l'image de démarrage de l'application dans l'onglet « Splash Screen » du menu « Branding »

4. Intégration NetBeans de l'interface:

NetBeans Platform propose deux types de conteneurs : les éditeurs et les vues. Tous deux héritent de la classe «org.openide.windows.TopComponent». C'est l'équivalent Swing du composant « JPanel ». La différence entre ces deux types de conteneur se fait sur leur utilisation:

- l'éditeur permet d'afficher le contenu d'un document, et peut en afficher plusieurs dans des onglets différents. Tous les éditeurs sont concentrés dans la zone centrale d'édition.

- la vue permet d'afficher des informations, des outils, des palettes, ..., et à une vocation unique (1 type de vue à la fois). Une vue peut être placée tout autour de la zone d'édition, et peut être regroupée dans un sous ensemble de vues sous forme d'onglets.

→ Décomposer la maquette en conteneurs NetBeans, en y intégrant les composants JavaFX conçus au TP précédent (« FXMLMainPanel »).

Pour créer un conteneur, créer un nouveau fichier de type « Window Component » dans la catégorie « Module Development ». Vous pouvez alors choisir la position de la fenêtre, « explorer » étant à gauche, « properties » à droite, « output » en bas et « editor » au milieu. L'option « Open on Application Start » permet de garantir l'affichage du conteneur lors de l'ouverture de l'application.

5. Boites de dialogue

Pour pouvoir afficher des boites de dialogues évoluées, votre module doit déclarer une dépendance vers la bibliothèque « Dialogs API ».

Pour afficher un message dans une boite de dialogue, il suffit d'utiliser la classe « DialogDisplayer »:

```
DialogDisplayer.getDefault().notify(new NotifyDescriptor.Message("Message.", NotifyDescriptor.WARNING_MESSAGE));
```

→ Ajouter une demande de confirmation sur l'action « Clear » (depuis la barre de menu).

→ Ajouter un message de notification lorsque l'utilisateur **confirme** l'action « Clear ».

6. Progression

Pour pouvoir notifier l'utilisateur de la progression, votre module doit dépendre de « Progress API ».

NetBeans Platform intègre un mécanisme qui permet de lancer des taches avec une progression sur l'état d'avancement. Voici un exemple:

```
Runnable myRunnable = new Runnable() {
   public void run() {
      ProgressHandle myProgressHandle = ProgressHandleFactory.createHandle("Please wait....");
      myProgressHandle.start();
      //DO TASK HERE
      myProgressHandle.progress("Doing task...");
      myProgressHandle.finish();
   }
};
RequestProcessor.Task myTask = RequestProcessor.getDefault().post(myRunnable);
```

→ Appliquer ce mécanisme pour simuler la progression de la sauvegarde et du chargement d'un carnet d'adresses depuis les actions « Save » et « Open... » (depuis le menu).

7. Wizard et assistants

Le formulaire de saisie d'un contact peut être intégré dans un Wizard qui gère le processus de validation, d'enregistrement de plusieurs pages du formulaire, et d'aide contextuelle.

→ Créer un nouveau fichier de type « Wizard » dans la catégorie « Module Development ». Nommer ce wizard « ContactForm ». L'action associée au lancement du Wizard est automatiquement créée.

→ Intégrer le formulaire de saisie de contact en JavaFX dans ce Wizard. L'ouverture du formulaire doit se faire par le bouton « Nouvelle Fiche ».

→ Contraindre la validation du Wizard sur le renseignement du champs « nom ». Prendre en compte le champs manquant et notifier l'utilisateur de la nature du problème qui empêche la création du contact.

8. Properties et Binding JavaFX

→ Dans le module «ContactManagerDataModule », créer une classe statique « DB » et ajouter des modèles de données factices de type « ObservableList » , accessibles en statique:
- Liste des carnets d'adresse : DB.getAdressBooks(),
- Liste de tous les contacts : DB.getAllContacts()
- Liste de tous les contacts pour un carnet d'adresse donnée : DB.getContacts(AddressBook))

→ Utiliser les données de la classe « DB » pour initialiser la liste des carnets d'adresses et le tableau des contacts (en utilisant le premier carnet d'adresse affiché dans la liste).

→ Ajouter 2 labels en entête du formulaire de saisie d'un contact pour afficher le genre et le nom du contact (cf. Annexe 1 du TP1). Ajouter ensuite un binding entre le RadioButton de sélection du genre du contact et le premier label, puis entre le TextField de saisie du nom d'un contact et le second label, en utilisant le binding JavaFX. Il vous faudra gérer la conversion du genre «Boolean» en « String » de façon à afficher « Mr » ou «Mme » devant le nom du contact. Le binding sur le prénom n'est pas demandé.

→ Ajouter un listener sur la modification de l'adresse email saisie dans le formulaire de saisie d'un contact et faire apparaître à l'utilisateur un retour sur l'erreur de saisie (ou aucun message si la saisie est valide). Vous pouvez utiliser l'api RegEx du JDK pour valider l'adresse email:

```
Pattern pattern = Pattern.compile(".+@.+\\.[a-z]+");
Matcher matcher = pattern.matcher(string);
boolean isValid = matcher.find();
```

9. Nodes et explorateur

L'API Nodes permet d'encapsuler les objets métiers dans une structure hiérarchique. La visualisation de ces données se fait par différentes vues proposés par l'API Explorer. Votre module doit donc déclarer une dépendance vers les bibliothèques « Nodes API » et « Explorer & Property Sheet API ». L'objectif ici est de proposer une nouvelle vue « Gender » qui affiche tous les contacts selon le genre.

→ Ajouter un nouveau module nommé «ContactManagerGenderModule ». Ce module va contenir la nouvelle vue « Gender » qui va afficher les contacts du module « ContactManagerDataModule ».

→ Encapsuler l'objet « Contact » dans un nœud « **ContactNode** » qui hérite de « BeanNode<Contact> ». Ce nœud est une feuille qui doit afficher une icône spécifique ainsi que le nom en majuscule et le prénom (ex : DUPONT Jean)

→ Créer une factory « **ContactChildFactory** » qui hérite de « ChildFactory<Contact> » pour la création de nœuds « ContactNode » . Cette factory doit prendre en paramètre du constructeur un « Boolean » pour définir le genre, afin de ne créer que les nœuds pour les contacts du genre associé.

→ Encapsuler l'objet «Boolean» dans un nœud « **GenderNode** » qui hérite de « AbstractNode » et qui

utilise la factory « ContactChildFactory ». Ce nœud aura comme enfants des « ContactNode ». Il doit afficher une icône spécifique et le genre en toute lettre (ex : « Male » si le Boolean est TRUE ou « Female » si le Boolean est FALSE)

→ Créer une factory « **GenderChildFactory** » qui hérite de « ChildFactory<Boolean> » pour la création de nœuds « GenderNode ». Les nœuds enfants a créer sont « Boolean.TRUE » et « Boolean.FALSE ».

→ Créer la classe « **GenderRootNode** » qui hérite de « AbstractNode » et qui utilise la factory « GenderChildFactory ». Ce nœud sera la racine de l'arborescence et devra afficher «All contacts».

→ Créer une nouvelle vue « **GenderTopComponent** » qui hérite de « TopComponent » et qui implémente « ExplorerManager.Provider » pour l'affichage de tous les contacts regroupés par genre dans un visualisateur de type « OutlineView ». Ce visualisateur doit être configuré pour afficher également une colonne « Age ». Utiliser un « ExplorerManager » pour définir le contexte racine de cette vue.

10. Sélection et Properties Editor

La plateforme NetBeans propose une vue pour éditer les propriétés des JavaBeans. Cette vue est connectée au mécanisme de sélection globale et permet d'éditer automatiquement les propriétés de la sélection courante (qui doit être compatible JavaBeans).

→ Associer le lookup de « GenderTopComponent » avec son « ExplorerManager » de façon à détecter la sélection dans l'arbre des contacts par genre.

→ Inclure le code suivant dans le constructeur de « GenderTopComponent » pour forcer l'affichage de la vue « Properties » :

```
TopComponent propertyEditor = WindowManager.getDefault().findTopComponent("properties");
if(propertyEditor!=null)
        propertyEditor.open();
```

→ Vérifier que la vue « Properties » affiche bien les propriétés du contact sélectionné dans la vue « Gender ».