TP3 – Introduction au Framework JavaFX et à l'API Jakarta Json

Avant-propos

Ouvrez un terminal.

Linux

Dans le terminal, lancez :

- `xhost +`

MacOS

Téléchargez et installez **XQuartz** depuis https://www.xquartz.org
Exécutez **XQuartz**, puis allez dans **XQuartz** > **Réglages** > **Sécurité** et cochez [**Autoriser les connexions de clients réseaux**].
Dans le terminal, lancez :

- `xhost +`
- `export DISPLAY=host.docker.internal:0`

Windows

Téléchargez et installez **VcXsrv** depuis https://sourceforge.net/projects/vcxsrv
Exécutez **VcXsrv** et cochez [**Disable access control**].

Docker

Dans le terminal, lancez :

- `docker image pull r401-runtime`
- `docker run -it --name r401-runtime-container --userns=host --net=host -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix:ro r401-runtime`

Si vous n'êtes pas sur une machine fixe de l'IUT, n'oubliez pas de changer le nom de l'**image Docker** en **fripouillon/r401-runtime**.

Vous vous retrouverez dans le **terminal Docker** avec l'invite **`root@r401-runtime>`**.

La compilation et l'exécution se feront par l'intermédiaire d'un script en ligne de commande dans le **terminal de Docker.**

Dans l'intégralité des TP, **vous utiliserez Visual Studio Code** pour éditer le code à partir des extensions permettant d'accéder au **conteneur Docker**.

Afin de ne pas compliquer inutilement, vous définirez **systématiquement** vos classes dans le package **par défaut**. Prenez garde que **les erreurs affichées dans l'éditeur ne sont pas forcément les erreurs réelles** mais peuvent être dues aux défauts de votre personnalisation de l'éditeur.

Exercice 1 : Framework JavaFX
(Java et FXML)

Préambule :

JavaFX est un Framework et une bibliothèque d'interface utilisateur issue du projet **OpenJFX** qui permet aux développeurs Java de créer une interface graphique pour des applications de bureau, des applications internet enrichies et des applications smartphones et tablettes tactiles.

Dans ce Framework, la composition et l'organisation des composants de l'interface graphique sont décrites dans un fichier ***.fxml** (éventuellement associé à une feuille de style ***.css**) tandis qu'une classe de contrôle reçoit les événements d'interaction de l'utilisateur.

La documentation complète de **JavaFX** est disponible sur documentation JavaFX.

Exercice :

Le but de l'exercice est d'implémenter une application GUI qui affiche le message « **Hello World** » lorsqu'on appuie sur un bouton.

Histoire de vous (re)familiariser avec **JavaFX**, vous allez donc compléter les fichiers :

- **myscreen.fxml** (décrivant l'interface de la fenêtre de l'application GUI, aussi bien les composants graphiques que les événements d'interaction associés en FXML, un langage de balise dérivé du XML),
- **FxController.java** (contenant la classe de contrôle qui va recevoir les événements d'interaction, comme l'appui sur un bouton, décrits dans l'interface)
- **MyApplication.java** (contenant la classe d'entrée de l'application qui va notamment charger l'interface de l'application GUI)

qui sont dans le dossier **/workspace/1-JavaFXAPI/src** du **conteneur Docker**.

Afin de vous guider, sachez que **myscreen.fxml** devra contenir :

- une fenêtre **<AnchorPane>** dont les événements seront envoyés à la classe de contrôle **FxController** (fx:controller). La fenêtre contiendra :
  - un **<Label>** avec l'identifiant **m_message** (fx:id)
  - un bouton **<Button>** dont l'action (onAction) sera reliée à la méthode **sayHelloWorld()** de la classe de contrôle **FxController**. La méthode affectera le texte « **Hello World** » à l'élément de type **Label**.

Pour compiler et exécuter votre code, dans le **terminal Docker**, lancez :

- `run.sh 1-JavaFXAPI`

Les fichiers **myscreen.fxml** et **MyApplication.java** étant vides, **l'exécution ne fera absolument rien avant que vous ne commenciez à développer !**

Vous pourrez vous inspirer du tuto fxml-developpez.com :

- fxml-developpez.com: les bases pour **myscreen.fxml** et **MyApplication.java**
- fxml-developpez.com: le contrôleur pour **FxController.java**

Exercice 2 : API Jakarta Json

(Java)

Préambule :

Le format **JavaScript Object Notation (Json)** est un format de données textuel dérivé de la notation des objets du langage JavaScript utilisé couramment pour la représentation et la transmission d'information structurée. Si des bibliothèques pour le format **Json** existent dans de nombreux langages de programmation, nous allons nous baser sur celle de **Jakarta**.

La documentation pour écrire un **Json** est sur documentation JsonWriter.
La documentation pour lire un **Json** est sur documentation JsonReader.

Exercice :

Le but de l'exercice est d'implémenter une application en ligne de commande pour manipuler un objet **Json** en écriture et en lecture qui devra avoir la forme :

```
{
   "montexte" : "Hello World",
```

```
    "montableau" : [
         "valeur1", "valeur2", "valeur3"
    ]
}
```

Vous allez donc compléter le fichier :

- **MyApplication.java** (contenant la classe d'entrée de l'application)

qui est dans le dossier **/workspace/2-JsonAPI** du **conteneur Docker**.

La création du **Json** se fera dans une méthode statique **create()** qui retournera la chaine du **Json** que l'on affichera pour vérification.

Le parcours du **Json** se fera dans une méthode statique **parse()** qui prendra en paramètre la chaine du **Json** précédemment créée et affichera la valeur du champ **montexte** puis **montableau** en bouclant sur ses valeurs.

Pour compiler et exécuter votre code, dans le **terminal Docker**, lancez :

- `run.sh 2-JsonAPI`

Vous pourrez vous inspirer des exemples documentation JsonObject et documentation JsonArray.