Introduction à la programmation en `Java`

Projet

Stéphane Airiau

Université Paris-Dauphine

Objectifs

- développer une application par binôme
- apprendre (par vous-même) à réaliser une interface graphique avec `javaFX`
- les spécifications ne sont pas précises
  - plus de liberté/créativité en tant que développeur
  - un peu moins facile
- proposer l'application au département

Pour l'évaluation, vous serez donc évalué en fonction de ce que vous avez appris et produit.
Il sera plus important d'avoir une application qui fonctionne plutôt qu'une très belle interface graphique qui ne fonctionne pas complètement.

l'Application

Aide à la création des emplois du temps

- Souvent les emplois du temps des formations sont créés **à la main** ! oui, c'est le cas à MIDO !
- on veut fournir un outil qui aide à la création des emplois du temps

Deux parties :

- aide à la création manuelle / changements manuels :
  on assiste un utilisateur dans la création des emplois du temps
  On peut sauvegarder les emplois du temps, les ouvrir, les modifier
  - grace à une interface graphique
  - grace à la detection automatique d'impossibilités
- génération automatique d'emplois du temps

A terme :

- on veut utiliser l'outil automatique pour gérer l'emploi du temps
- on veut pouvoir modifier à la main l'emploi du temps et vérifier que la modification est correcte

Problème d'emplois du temps

- problème classique de programmation par contraintes
- contraintes de salles :
  - une même salle ne peut être utilisée que pour un seul cours
  - une salle a une capacité
- contraintes enseignants
  - un enseignant ne peut donner qu'un seul cours à la fois dans une seule salle
  - un enseignant peut donner cours dans plusieurs formations
  - un enseignant peut aussi avoir d'autres contraintes et ne pas être libre sur certains créneaux
- contraintes des étudiants
  - un étudiant peut avoir au plus un cours pour chaque créneau horaire

Il serait bon que notre application puisse

- dire si une solution existe ou non
- si une solution existe,
  un but minimal est de trouver un emploi du temps qui satisfait toutes ses contraintes

Optimisation

On peu imaginer qu'en fait il y a plusieurs solutions.

- on peut essayer de trouver un emploi du temps qui soit meilleur du point de vue des étudiants. Par exemple :
  - un emploi du temps qui garantisse une pause à midi
  - un emploi du temps qui n'a pas trop de "trous"
  - un emploi du temps qui n'a pas trop de cours "tard"
- on pourrait aussi prendre en compte des voeux d'enseignants

S'il n'y a pas de solutions, on pourrait orienter les propositions par exemple, l'outil peut cibler quelques changements qui rendraient possible une solution

Représentation

- Quelle représentation interne pour représenter les emplois du temps
- Quelle représentation pour l'interface graphique
- On doit pouvoir ajouter/supprimer facilement
  - des salles
  - des enseignants
  - des cours
  - des formations
  - des groupes

⇨ une architecture que l'on peut modifier

javaFX

- moteur graphique appelé `Prism`
- un système de fenêtre appelé `Glass`
- un moteur de media
- un moteur web

On peut donc créer des formulaires, des graphiques, etc...

Interface utilisateur : contrôle

Interface utilisateur : organisation

On peut utiliser différentes classes pour organiser son interface de façon dynamique

layout

- `BorderPane` organise par région (haut, bas, gauche, centre...)
- `HBox` organise le contenu sur une ligne
- `VBox` organise le contenu sur une colonne
- `StackPane` organise le contenu comme un tas
- `GridPane` organise le contenu dans une grille
- `TilePane` organise le contenu selon des cellules de même taille

javaFX :

- transformer chaque élément : rotation, zoom, translation
- on peut utiliser les couleurs, utiliser des dégradés, ajouter des effets (ombre, flou, reflections...)
- on peut modifier l'aspect du curseur
- faire des animations

bref, on peut faire beaucoup !

javaFX : Application

- pour faire une interface graphique, on crée un objet qui va hériter d'une classe `Application`
- ainsi, pour réaliser l'interface, il suffit d'écrire la méthode `public void start(Stage primaryStage)`

La méthode `main` appellera simplement la méthod `launch` de la classe `Application`.

```
public static void main(String[] args){
   Application.launch(MaClass.class, args);
}
```

javaFX : Stage

La méthode `launch` de la classe `Application` va appeler la méthode `start` qui prend en paramètre une **scène**.

La scène (stage en anglais) représente l'interface que l'on bâtit.

```
1 @Override
2 public void start(Stage primaryStage) {
3     primaryStage.setTitle("JavaFX Welcome");
4
5     primaryStage.show();
6 }
```

La fenêtre créée aura pour titre "JavaFX Welcome" et sera visible.
Elle sera à présent complètement vide.

javaFX : Scene

Pas de chance en français, Stage et Scene se traduisent par le même mot : la scène. On va donc essayer de dire la pièce pour Scene.
Tout ce qui apparaîtra dans notre pièce doit être inséré dans un objet `Scene` qui lui même se trouvera dans un objet `Stage`.

```
1 GridPane grid = new GridPane();
2 grid.setAlignment(Pos.CENTER);
3 grid.setHgap(10);
4 grid.setVgap(10);
5 grid.setPadding(new Insets(25, 25, 25, 25));
6
7 Scene scene = new Scene(grid, 300, 275);
8 primaryStage.setScene(scene);
```

- Ici, on prépare un layout sous la forme d'une grille.
- on crée une pièce avec ce layout
- on insert la pièce sur scène.

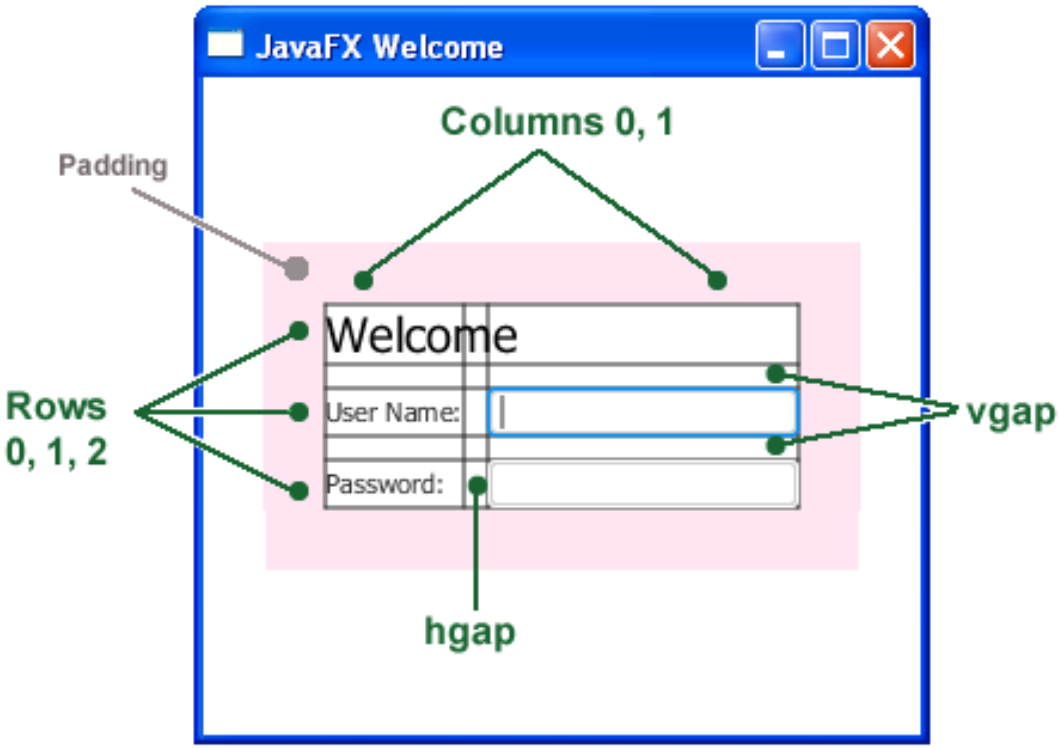
Il ne nous reste plus qu'à insérer les différents composants (noeuds graphiques) de notre interface.

Exemple

On pourrait ajouter des formes (ex un objet de la classe `Circle`), etc... Ici, on va ajouter du texte, un champ pour entrer du texte.

```
1  Text scenetitle = new Text("Welcome");
2  scenetitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));
3  grid.add(scenetitle, 0, 0, 2, 1);
4
5  Label userName = new Label("User Name:");
6  grid.add(userName, 0, 1);
7
8  TextField userTextField = new TextField();
9  grid.add(userTextField, 1, 1);
10
11 Label pw = new Label("Password:");
12 grid.add(pw, 0, 2);
13
14 PasswordField pwBox = new PasswordField();
15 grid.add(pwBox, 1, 2);
```

Exemple


JavaFX Welcome
Columns 0, 1
Padding
Welcome
User Name:
Password:
Rows
0, 1, 2
vgap
hgap

Exemple : ajout d'un bouton et d'un texte

```
1  Button btn = new Button("Sign in");
2  HBox hbBtn = new HBox(10);
3  hbBtn.setAlignment(Pos.BOTTOM_RIGHT);
4  hbBtn.getChildren().add(btn);
5  grid.add(hbBtn, 1, 4);
6
7  final Text actiontarget = new Text();
8  grid.add(actiontarget, 1, 6);
```

Exemple : ajout d'un bouton et d'un texte

Gestion d'évènement

A chaque objet graphique, on peut associer un objet de type `EventHandler` qui exécutera une fonction à chaque fois qu'un certain type d'évènement se produit.
par exemple :

- entrée ou sortie de la souris dans une zone
- clic ou relâchement du clic
- mouvement de la souris
- ....

Dans l'exemple ci-dessous, on fera quelque chose à chaque fois que l'on clique sur ce qui est représenté par `object`.

```
object.setOnMouseClicked(new EventHandler<MouseEvent>(){
    public void handle(MouseEvent me){
        // code à exécuter après un clic
    }
});
```

Qu'est ce le code argument de la méthode `setOnMouseClicked` ?

Exemple : Gestoin d'un évènement

Pour un bouton, si on enfonce le bouton (par un clic ou par le clavier), on "actionne" le bouton, donc la méthode s'appelle `setOnAction`.

```
1 btn.setOnAction(new EventHandler<ActionEvent>() {
2
      @Override
      public void handle(ActionEvent e) {
          actiontarget.setFill(Color.FIREBRICK);
          actiontarget.setText("Sign in button pressed");
      }
  });
```

Quand l'utilisateur va presser le bouton, le text "sign in button pressed" apparaîtra en couleur rouge brique.

Exemple : ajout d'un bouton et d'un texte

Exemple

on a donc des actions

- `setOnMouseEntered`
- `setOnMouseExited`
- `setOnMousePressed`
- `setOnMouseReleased`
- `onKeyPressed`
- `onKeyReleased`

grosse librairie

- on peut utiliser CSS
- faire des animations
- ajouter du contenu multimedia
- ...

Organisation du projet (tentative)

- semaine du 23 janvier
- semaine du 30 janvier
- semaine du 6 février : interface finalisée
- semaine du 22 février : gestion à la main fonctionnelle
- semaine du 27 février : travail sur l'automatisation
- semaine du 13 mars : rendu du travail
  présentation et démo
  rapport contenant une notice d'utilisation, le diagramme uml
  (le code devra être évidemment commenté et avoir des tests)