Interfaces graphiques - JavaFX

responsable : Wiesław Zielonka
zielonka@liafa.univ-paris-diderot.fr
http://liafa.univ-paris-diderot.fr/~zielonka

March 15, 2016

Remarques sur le focus

Les entrées clavier sont envoyées vers l'application qui possède le focus.
A l'intérieur de l'application c'est le noeud qui possède le focus qui reçoit les événements clavier.
Changement de noeud qui possède le focus :

- touche TAB – pour passer le focus au noeud suivant ou
- clicquer sur le noeud qui devra recevoir le focus.

Certains noeuds sont focusable par leur nature (peuvent obtenir le focus) : `TextField`, `Button` etc. Par défaut leur propriété

focusTraversable

est `true`.

Mais `Pane` n'est pas focusable, la propriété `focusTraversable` de Pane a valeur `false` par défaut.

Donc même si un pane possède de handlers (ou filtres) pour recevoir KeyEvents ces événements ne seront pas délivrés dans ces handlers tant que la propriété focusTravesable du pane reste false.

Comment rendre Pane focusTraversable ?

```
pane.setFocusTraversable(true);
```

Pour demander le focus pour un noeud depuis l'application:

```
noued.requestFocus();
```

Cette demande ne sera pas satisfaite si le noeud n'est pas focusTraversable.

Rendre visible que pane obtient ou perd le focus

L'idée : différent Border en fonction de la valeur de la propriété `focus` (`focus` est la propriété Boolean qui indique si le noeud possède le focus) :

```
Pane pane = new Pane().
pane.setFocusTraversable(true);
pane.focusedProperty().addListener( (observable,
                                     oldValue, newValue)
   -> {
       if (newValue) {
         pane.setBorder(new Border(new BorderStroke(
             Color.YELLOWGREEN, /* vert si focus */
             BorderStrokeStyle.SOLID,
             CornerRadii.EMPTY,
           new BorderWidths(4))));
       } else {
         pane.setBorder(new Border(new BorderStroke(
             Color.RED, /* RED si pas de focus */
             BorderStrokeStyle.SOLID,
             CornerRadii.EMPTY,
            new BorderWidths(4))));
       }
});
```

Lambda expressions

Lambda expressions permettent d'implémenter plus facilement les interfaces avec une seule méthode.

```
ColorPicker picker = new ColorPicker(shapeColor);
picker.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
      Color c = picker.getValue();
      if (c != null) {
         shapeColor = c;
      }
    }
});
```

La même chose de façon plus concise avec une lambda expression :

```
ColorPicker picker = new ColorPicker(shapeColor);
picker.setOnAction( e -> {
      Color c = picker.getValue();
      if (c != null) {
           shapeColor = c;
      }
});
```

Lambda expression – plusieurs arguments

`ChangeListener` a une méthode

```
void changed(
    ObservableValue<? extends T> observable,
    T oldValue, T newValue)
```

Implémenter un `ChangeListener` avec une lambda expression :

```
BooleanProperty creerRect = new SimpleBooleanProperty();
creerRect.addListener( (observable, oldValue, newValue) -> {
  if (newValue) {
     System.out.println("rectangle listenr yes");
  } else {
     System.out.println("rectangle listenr no");
  }
});
```

`creerRect` est une propriété booléenne, donc java peut déduire les types de trois paramètres de la méthode `changed`.

javafx.scene.canvas.Canvas

`Canvas` est un noeud sur lequel nous pouvons dessiner.
Constructeur: `Canvas(double width, double height)`
Pour obtenir le contexte graphique de Canvas utliser la méthode :

```
GraphicsContext getGraphicsContext2D()
```

Ensuite toutes les modifications de Canvas passent par le contexte graphique.
Si Canvas n'est pas dans une scène alors n'importe quel thread peut écrire dedans, une fois Canvas mis dans une scène uniquement le thread de l'application peut modifier le contexte graphique associé.

GraphicsContext – dessiner un chemin (path) dans Canvas

Quelques méthodes de GraphicsContext :

```
beginPath() //commencer un chemin
//aller dans le point (x0,y0)
moveTo(double x0, double y0)
bezierCurveTo(double xc1, double yc1,
              double xc2, double yc2,
              double x1, double y1)
quadraticCurve To(double xc, double yc,
                  double x1, double y1)
lineTo(double x1, double y1) //
closePath() //terminer le chemin en le fermant
stroke() //dessiner le chemin
fill() //remplir la forme entouree par le chemin
```

GraphicsContext – méthodes de GraphicsContext (suite)

```
setFill(Paint p) // specifier la couleur de remplissage
setStroke(paint p) //specifier la couleur de lignes
setFont( Font f) // specifier la police de caracteres
setLineWidth(double lw)
```

Canvas est transparent donc on peut utiliser des noeuds Canvas pour faire overlay en mettant un Canvas sur l'autre.

Exemple de l'utilisation de Canvas pour décorer un bouton (un ToggleButton):

```
ToggleGroup toggleGroup = new ToggleGroup();
/* ajouter le bouton avec rectangle */
Canvas rectCanvas = new Canvas(25, 20);
GraphicsContext context = rectCanvas.getGraphicsContext2D();
context.setStroke(Color.BLACK);
context.strokeRect(5, 5, 15, 10);
ToggleButton rectButton = new ToggleButton(null, rectCanvas);
tool.getItems().add(rectButton);
rectButton.setToggleGroup(toggleGroup);
```

Canvas – faire une grille sur Pane

```
ChangeListener<Number> paneListener =
  new ChangeListener<Number>() {
   @Override
   public void changed(ObservableValue<? extends Number> obse
                       Number oldValue, Number newValue) {

     //si la nouvelle valeur plus petite que l'ancienne
     //pas besoin de refaire Canvas
     if (newValue.doubleValue() < oldValue.doubleValue()) {
       return;
     }
     Canvas gridCanvas = new Canvas(pane.getWidth(),
                                    pane.getHeight());
     GraphicsContext gc = gridCanvas.getGraphicsContext2D();
     gc.setLineWidth(0.5);

     gc.setStroke(Color.BLACK);
     double height = gridCanvas.getHeight();
     double width = gridCanvas.getWidth();
     final int dist = 50;
```

Canvas – faire une grille sur Pane

```
        for (int i = dist; i < width; i += dist) {
            gc.strokeLine(0, i, width, i);
        }
        for (int i = dist; i < width; i += dist) {
            gc.strokeLine(i, 0, i, width);
        }
        //remplacer Canvas - l'enfant l'index 0 de pane
        pane.getChildren().remove(0);
        pane.getChildren().add(0, gridCanvas);
    }
};

Canvas gridCanvas = new Canvas();
pane.getChildren().add(0, gridCanvas);
pane.widthProperty().addListener(paneListener);
pane.heightProperty().addListener(paneListener);
```

Snapshot – prendre “photo” d’un noeud

```
Button button = new Button();
button.setText("prendre snapshot");
button.setOnAction(e -> {
    WritableImage image = new WritableImage(400, 300);
    pane.snapshot(null, image);
    File file = new File("image.png");
    try {
        ImageIO.write(SwingFXUtils.fromFXImage(image, null),
                      "png", file);
    } catch (IOException ex) {
        System.err.println(ex.getMessage());
    }
});
```

Transformation de `javafx.scene.image.Image` en `BufferedImage`

La classe SwingFXUtils possède la méthode

static BufferedImage fromFXImage(Image img, BufferedImage bmg)

pour transformer javafx Image en BudfferedImage.

La sauvegarde de `BufferedImage` dans un fichier se fait comme dans swing (et avec la classe ImageIO de swing).

Afficher une image

```
Image image = new Image("rose.png");
ImageView imv = new ImageView();
imv.setView(image);
```

Le paramètre du constructeur de `Image` peut être une URL.
`ImageView` est un noeud qui permet d'afficher une image.

On peut lire les pixels d'une image avec un `PixelReader`.
Par contre les pixels de Image ne sont pas modifiables.
Pour avoir une image avec les pixels modifiables construire `WritableImage` et utiliser `PixelWriter`.
Aussi bien Image que ImageWriter peuvent utiliser plusieurs formats de pixels et le travail sur l'image dépend de ce format.

Complément sur `java.scene.shape`

Construire une courbe (Path) en utilisant :

`MoveTo` `ArcTo` `CubicCurveTo` `LineTo` `QuadCurveTo`
`ClosePath` `HLineTo` `VLineTo`

Toutes ces classes sont dérivées de `PathElement`.
Exemple : un triangle :

```
Path path = new Path();
MoveTo a = new MoveTo(200,200);
LineTo b = new LineTo(300,300);
LineTo c = new LineTo(100,300);
ClosePath d = new ClosePath();
path.getElements().addAll(a,b,c,d);
Pane pane = new Pane();
pane.getChildren().add(path);
```

Complément sur `java.scene.shape`

D'autres formes 2D :

`Circle` `Ellipse` `Polygon` `Polyline` `Rectangle` `SVGPath`