Instituto Politécnico de Beja — Escola Superior de Tecnologia e Gestão

Guia Prático 1 — Interfaces Gráficas

João Paulo Barros

Licenciatura em Engenharia Informática
6 de março de 2018 (versão beta)

Objectivos: elementos constituintes da interface gráfica (`Stage`, `Scene`, *control* `Button` e *pane* `VBox`; *layout* e comportamento; definição de *handlers*.

1 AWT, Swing e JavaFX

A primeira versão da linguagem Java, em 1995, incluiu a biblioteca *Abstract Window Toolkit* (AWT) para a construção de interfaces gráficas. Esta era, e é pois ainda existe, simples mas também muito limitada e muito dependente do suporte existente no sistema operativo em que o programa Java era executado. De forma a possibilitar uma maior uniformidade entre ambientes e também para oferecer uma maior quantidade de componentes mais sofisticados, surgiu, em 1997, a biblioteca Swing. Esta foi desenhada para a construção de interfaces gráficas para aplicações *desktop*. Embora continue disponível e a ser utilizada, existe uma biblioteca que a substitui: a biblioteca JavaFX que vamos aqui designar por "o JavaFX"no sentido de "o código JavaFX". Na sua versão atual, o JavaFX permite o desenvolvimento de *rich Internet applications* (RIA). Estas são aplicações que permitem um tipo de interatividade semelhante ao de uma aplicação *desktop*. Uma aplicação JavaFX pode ser executada no *desktop* (standalone) ou no *browser*. O JavaFX também tem suporte para dispositivos multitoque, animação 2D e 3D, *playback* de som e vídeo e suporta a utilização de css. Nós vamos utilizar apenas uma pequena parte das possibilidades do JavaFX, e vamos utilizá-lo para aplicações `desktop`.

Há duas razões principais para irmos utilizar JavaFX: (1) por ser mais simples e (2) por o Swing estar já descontinuado, pois não se prevêem mais atualizações. Além disso o JavaFX abre mais possibilidades para o futuro devido ao suporte para a *web* e para dispositivos móveis. Em resumo, pode-se dizer que é uma biblioteca melhor e mais completa do que a biblioteca Swing.

2 Os tipos principais — Stage, Scene, Control e Pane

A Listagem 1 exemplifica um programa "hello world"em JavaFX; as palavras reservadas estão a vermelho e as classes utilizadas a azul:

Listagem 1: Um programa "Hello World"

```
1  package pt.ipbeja.hello.gui;

3  import javafx.application.Application;
   import javafx.geometry.Pos;
5  import javafx.scene.Scene;
   import javafx.scene.control.Label;
7  import javafx.stage.Stage;

9  public class Main extends Application {
           @Override
11         public void start(Stage primaryStage) {
                   Label helloLabel = new Label("Hello World!");
13                 helloLabel.setAlignment(Pos.CENTER);
                   Scene scene = new Scene(helloLabel, 200, 100);
15                 primaryStage.setScene(scene);
                   primaryStage.show();
17         } // end start

19         public static void main(String[] args) {
                   Application.launch(args);
21         }
   }
```

Eis o essencial para cria uma interface gráfica em JavaFX:

1. Na linha 20, o método **`static`** `launch` da classe `Application`. (`Application.launch(args)`) provoca a execução do método `start` (`public void start (Stage primaryStage)`).
2. Um `Stage` (palco) corresponde a uma janela na interface gráfica.
3. No método `start`, linha 11, o programa tem disponível um objecto do tipo `Stage` que corresponde a um "palco" (janela) principal (`primaryStage`).
4. Cada cena é um objecto da classe `Scene`.
5. No palco pode estar uma cena (`Scene`): para tal enviamos uma cena para o objeto palco: `primaryStage.setScene(scene);`.
6. Cada cena tem um nó "pai"(*parent*); no exemplo anterior é uma `Label`; o texto na mesma é centrado pelo método na linha 13.
7. É o nó "pai" que nós vemos dentro da janela (`Stage`), a cena é invisível.
8. O palco (janela) deve ser colocado como visível: `primaryStage.show();`

3 Adição de comportamento — um *handler* para um `Button`

Para que um botão (`Button`) faça algo, temos de lhe dizer o que queremos que ele faça. Para tal, informamos o botão sobre qual a função que queremos que seja executada quando o mesmo é premido. O código na Listagem 2 ilustra essa possibilidade.

Listagem 2: Um programa com um botão e a função *handler* passada num objeto de uma classe `ButtonHandler` que implementa a interface `EventHandler<ActionEvent>`

```
2  package pt.ipbeja.hello.gui;

4  import javafx.application.Application;
   import javafx.event.ActionEvent;
6  import javafx.event.EventHandler;
   import javafx.scene.Scene;
8  import javafx.scene.control.Alert;
   import javafx.scene.control.Button;
10 import javafx.stage.Stage;

12 public class Main extends Application {
       @Override
14     public void start(Stage primaryStage) {
               Button button = new Button("Press me");
16             ButtonHandler buttonHandler = new ButtonHandler();
               button.setOnAction(buttonHandler);
18             Scene scene = new Scene(button, 200, 100);
               primaryStage.setScene(scene);
20             primaryStage.show();
       }
22
       class ButtonHandler implements EventHandler<ActionEvent> {
24         @Override
           public void handle(ActionEvent event) {
26             Alert alert = new Alert(Alert.AlertType.INFORMATION, "Hello");
               alert.showAndWait();
28         }
       }
30
       public static void main(String[] args) {
32         Application.launch(args);
       }
34 }
```

Os detalhes são os seguintes:

1. Na verdade, a função que queremos que seja executada está dentro de um objecto e é esse objecto que passamos para o botão;
2. O objecto que passamos, tem de ser de uma classe que é do tipo `EventHandler<ActionEvent>`;
3. Para que uma classe seja desse tipo definimo-la como implementando esse tipo. No nosso exemplo é a classe com o nome `ButtonHandler` (ver linha 23 da Listagem 2);
4. Temos então de criar um objecto dessa classe `new ButtonHandler()` (ver linha 16 da Listagem 2) e passar esse objecto para o botão utilizando o método `button.setOnAction`, tal como é feito na linha 17 da Listagem 2 .

5. O código que queremos executar está na função `handle` (linha 29 da Listagem 2) na classe `ButtonHandler`.

6. As classes como a `ButtonHandler` — que servem para criar objectos que têm um método que queremos que seja executado para tratar (*handle*) de um evento — costumam ser definidas dentro da classe onde é criado o objecto (o nosso `Button`) para o qual queremos definir o comportamento; a estas classes chamamos `inner classes`. Portanto, na Listagem 2, a classe `ButtonHandler` é uma *inner classe* (classe interna) da classe `Main`.

4 Exercícios Propostos

1. Coloque a funcionar o programa da Listagem 2).

2. Faça um novo programa com dois botões; estes dois botões devem fazer o mesmo e para tal devem utilizar o mesmo objecto da classe `ButtonHandler`. Na cena só podemos colocar um nó (Node) que na Listagem é um botão. Como agora necessitamos de dois botões, vamos ter de colocar na cena um novo tipo de nó. Vamos utiliza uma "caixa vertical" (`VBox`) e é nesse objecto que vamos colocar os dois botões. Depois colocamos o objecto `vBox` na cena, como nó principal:

   ```
   VBox vBox = new VBox();
   vBox.getChildren().addAll(button1, button2);
   Scene scene = new Scene(vBox, 200, 100);
   ```

   Em rigor, adicionámos os botões à lista de "filhos" (*children*) do objecto `vBox`.

3. Faça um novo programa com dois botões; agora quando prime um botão a mensagem que surge deve indicar se foi premido o botão 1 ou 2. Para tal, o método `handle` deve perguntar ao objecto `event` qual o objecto que originou esse evento: no nosso exemplo foi de certeza um botão mas não sabemos qual. O objecto `event` que é um parâmetro do método `handle`, tem o método `getSource` que devolve esse objecto. Note que deve fazer um *cast* para o tipo que interessa: `Button` no nosso caso:

   ```
   Button buttonClicked = (Button)(event.getSource());
   ```

4. Faça um novo programa com dois botões; estes dois botões devem mudar o valor de duas `Labels`. Cada `Label` tem o valor zero no início e quando o respetivo botão é premido essa `Label` incrementa o valor. Veja os métodos `getText()` e `setText` da classe `Label`.

5 Informação adicional

O JavaFX tem imensas possibilidades suportadas por centenas de classes e milhares de métodos. As referências seguintes são uma ajuda importante para encontrar exemplos e consultar informação relevante para o que pretendemos fazer. Algum do código nas referências seguintes utiliza FXML, construtores de GUIs (GUI builders) e/ou css. Pelo menos para já, não vamos utilizar nada disso.

- *JavaFX: Working with JavaFX UI Components – Table of Contents*: `http://docs.oracle.com/javase/8/javafx/user-interface-tutorial`

- Exemplos: `http://docs.oracle.com/javase/8/javafx/sample-apps/`
- *Getting Started with JavaFX*: `http://docs.oracle.com/javase/8/javafx/get-started-tutorial/index.html`
- *Using Built-in Layout Panes*: `http://docs.oracle.com/javase/8/javafx/layout-tutorial/builtin_layouts.htm`
- *Sizing and Aligning Components*: `http://docs.oracle.com/javase/8/javafx/layout-tutorial/size_align.htm`

6 Créditos

Algumas partes são baseadas no livro [1] do qual deve já ler as secções 14.1 a 14.3.

Referências

[1] Y. Daniel Liang. *Intro to Java Programming, Comprehensive edition.* Pearson, 10th edition, 2015.

Deve continuar a resolução deste guia fora das aulas. Traga as dúvidas para a próxima aulas ou coloque-as no fórum de dúvidas da disciplina. As sugestões para melhorar este texto também são bem-vindas.