Programação Avançada

JavaFX

JavaFX

- Biblioteca para desenvolvimento de aplicações gráficas (*Rich Internet Applications*) em *Java*
  - A primeira versão foi disponibilizada em Outubro de 2008
  - Framework desenvolvida em Java

- Inicialmente era incluída na distribuição do JDK, mas passou a ser distribuído à parte, a partir do JDK 11
  - A Oracle passou o seu desenvolvimento para o OpenJDK, mais concretamente para um projeto específico designado OpenJFX

OpenJFX

- A documentação mais atualizada pode ser obtida a partir do *website*
  - https://openjfx.io/

- A integração do *JavaFX* num projeto *Java* pode ser realizada fazendo *download* da biblioteca e associando-a ao projeto
  - A biblioteca é constituída por diversos ficheiros *jar*

Instalação

- Fazer *download* a partir do *website*: `http://openjfx.io`
  - *Direct link*: `https://gluonhq.com/products/javafx/`

- Para novos projetos, fazer *download* de uma versão *LTS* ou versão mais recente
  - Sugestão: *JavaFX 17*
  - Fazer *download*
    - *SDK* para o sistema operativo pretendido
    - Documentação (*JavaDoc*)

- Descompactar para um diretório, preferencialmente junto ao *JDK* em uso (a documentação pode ser descompactada para um diretório `doc` dentro do diretório base criado)
  - *Windows*: `C:\Program Files\Java`
  - *MacOS*: `/Library/Java/JavaVirtualMachines`

Configuração do IntelliJ

- Criar um projeto Java como realizado para projetos Java anteriores
- Ir a opção `File → Project Structure`
  - `Global Libraries`
  - `"+" → New Java Library...`
    - Indicar o caminho completo para o diretório `lib` do *JavaFX*
  - Adicionar à biblioteca os *URL* para a documentação
    - Exemplo para ficheiros locais:
      - `file:///Users/ans/Projects/javafx-sdk-17/doc`
      - `file:///Users/ans/Projects/javafx-sdk-17/doc/javafx.base`
      - `file:///Users/ans/Projects/javafx-sdk-17/doc/javafx.controls`
      - `file:///Users/ans/Projects/javafx-sdk-17/doc/javafx.fxml`
      - `file:///Users/ans/Projects/javafx-sdk-17/doc/javafx.graphics`
      - `file:///Users/ans/Projects/javafx-sdk-17/doc/javafx.media`
  - Estes passos são realizados apenas uma vez

Configuração do IntelliJ

- Criar um projeto Java como realizado para projetos Java anteriores
- Nos projectos onde se pretende usar o *JavaFX*
  - `File` → `Project Structure`
  - `Global Libraries`
  - Dar um toque com o botão direito sobre a biblioteca e adiciona-se ao módulo (projeto criado)

Configuração do IntelliJ

- Nas configurações de execução
  - Adicionar as seguintes configurações às opções VM `Options`

    `--module-path /path_to_javafx_sdk/lib`

    `--add-modules javafx.controls`

    - Se for usado xml para a definição da interface:

      `--add-modules javafx.controls,javafx.fxml`

    - Incluir todos os módulos do *JavaFX*

      `--add-modules ALL-MODULE-PATH`

Aplicação *JavaFX*

- Uma aplicação *JavaFX* é encapsulada através de um objeto `javafx.application.Application`

- O desenvolvimento de uma aplicação *JavaFX* inicia-se normalmente pela criação de uma nova classe que deriva da classe `Application`
  - Deve ser definido o método abstrato `void start(Stage);`

- O objeto `Application` é criado pelo sistema quando o método estático `Application.launch(...)` for executado

Application

- Quando a classe que deriva de `Application` é a mesma onde está definida a função `main`

```
public class Main extends Application {

    public static void main(String [] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {
        // TODO
    }
}
```

Application

- Quando a classe que deriva de `Application` é diferente da classe onde está definida a função `main`

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

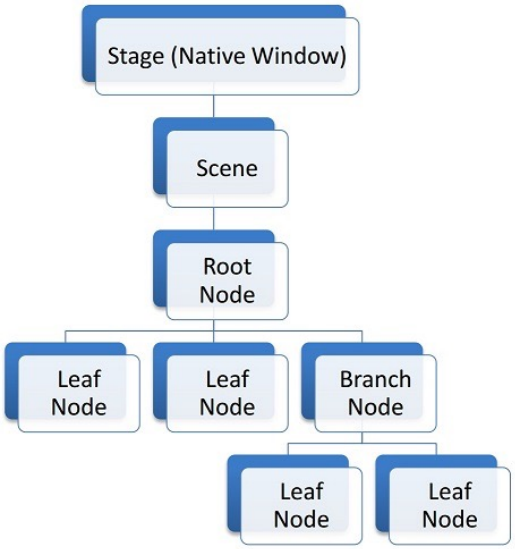
```
public class JavaFXMain extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        // TODO
    }
}
```

Ciclo de vida de uma aplicação

- Quando a instância de um objeto `Application` é criada após o método `launch` ser chamado, são executados os seguintes métodos:
  - `void init()`
    - Realização das iniciações necessárias para a aplicação
  - `void start(Stage)`
    - Ponto de entrada principal da aplicação *JavaFX*, onde se cria todo o interface gráfico
  - `void stop()`
    - Libertação de recursos antes do encerramento da aplicação

Scene Graph: Stage, Scene e root node

- A interface gráfica que caracteriza uma aplicação JavaFX deve ser criada no método `start` do objeto `Application`
- Os objetos que constituem a interface são organizados através de uma árvore de objetos, designada *Scene Graph*, a qual é constituído por:
  - `Stage`
  - `Scene`
  - *Root node* e outros *nodes*



Hierarquia de objetos JavaFX

- javafx.stage.Window
  - PopupWindow
    - Popup
    - PopupControl
      - ContextMenu, Tooltip
  - **Stage**

- javafx.scene.**Scene**

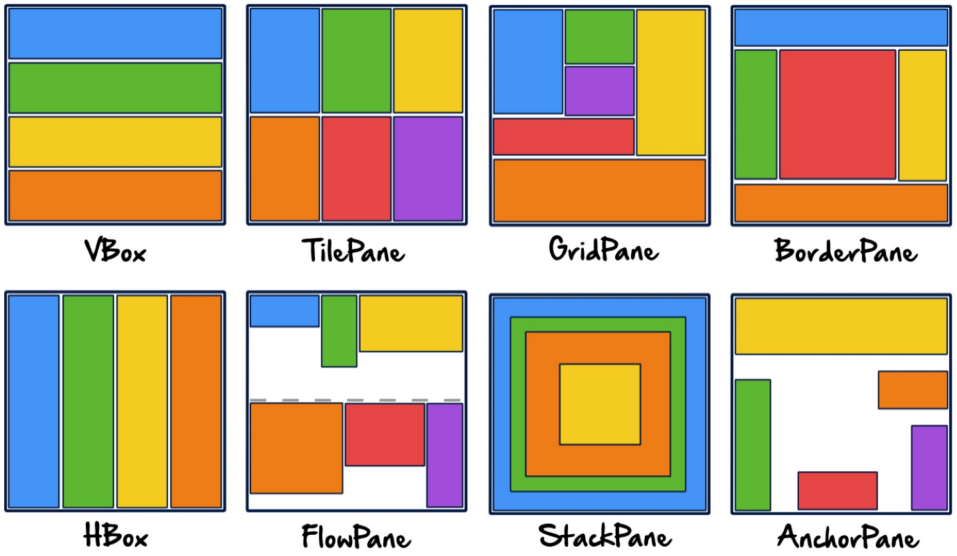
- javafx.scene.Node
  - Parent
    - Group
    - Region
      - *Next slide…*
    - WebView
  - **Shape**
    - Arc, **Circle**, CubicCurve, **Ellipse**, **Line**, **Path**, Polygon, **Polyline**, QuadCurve, **Rectangle**, SVGPath, **Text**
  - Shape3D
    - Box, Cylinder, MeshView, Sphere
  - **Canvas**
  - **ImageView**
  - Camera, LightBase, MediaView, SubScene, SwingNode

Hierarquia de objetos JavaFX

- Region
  - Control
    - TextInputControl
      - **TextArea**, **TextField**
    - ComboBoxBase
      - **ColorPicker**, ComboBox, **DatePicker**
    - Labeled
      - ButtonBase
        - **Button**
        - MenuButton
          - SplitMenuButton
        - ToggleButton
          - **RadioButton**
        - **CheckBox**
        - Hyperlink
      - Cell, **Label**, TitledPane
    - Accordion, ButtonBar, **ChoiceBox**, HTMLEditor, **ListView**, MenuBar, Pagination, ProgressIndicator, ScrollBar, **ScrollPane**, Separator, Slider, Spinner, SplitPane, TableView, TabPane, ToolBar, TreeTableView, **TreeView**
  - **Pane**
    - **AnchorPane**, **BorderPane**, DialogPane, **FlowPane**, **GridPane**, **HBox**, PopupControl.CSSBridge, **StackPane**, TextFlow, **TilePane**, **VBox**
  - Axis, Chart, TableColumnHeader, VirtualFlow

Objetos Pane (*layout*)

- `AnchorPane`
- `BorderPane`
- `FlowPane`
- `GridPane`
- `HBox`
- `StackPane`
- `TilePane`
- `VBox`



Formatação do *layout*

- Dependendo do tipo de objeto de *layout* usado, estão disponíveis diversos parâmetros de formatação e formas de adicionar os componentes que gere (*children*)
  - `getChildren().add, getChildren().addAll`
  - `setPadding`
  - `setAlignment`
  - `setSpacing`
  - `setTopAnchor, setBottomAnchor, setLeftAnchor, setRightAnchor`
  - `setTop, setBottom, setLeft, setRight, setCenter`
  - `setMargin`
  - …

Formatação de Nodes

- As cores e outras configurações de cada componente podem ser configuradas através de métodos específicos
  - `setBorder`
  - `setMaxSize, setMinWidth, setMaxHeight, setPrefHeight, ...`
  - `setText, setTextAlignment, setTextFill`
  - `setStyle("`*CSS style string*`")`
    - `Ex: obj.setStyle("-fx-background-color: #ffffd0;");`
  - …

Programação orientada por eventos

- A partir do momento em que os vários elementos ficam disponíveis, as diversas ações do utilizador são traduzidas através da geração de eventos sobre os elementos
- A programação orientada por eventos corresponde a programar previamente aquilo que deverá ser executado quando o evento ocorre
- Cada evento é encapsulado através de um objeto adequado
  - Por exemplo, o evento relativo ao *click* num botão é representado através de um objeto o `ActionEvent`

ActionEvent

- Como referido, quando um botão (`Button`) é clicado é gerado um evento representado através de um objeto `ActionEvent`

- O processamento dos eventos é realizado através de objetos `EventHandler<T>`
  - No objeto `EventHandler<T>` deve ser redefinido o método `void handle(T event)`, no contexto do qual se deve fazer o processamento pretendido

- Formas de criar um objeto `EventHandler<ActionEvent>`
  - criar uma instância de uma classe que implementa a interface `EventHandler<ActionEvent>`
  - criar um objeto *inline* (classe anónima) que implementa a interface `EventHandler<ActionEvent>`
  - *Lambda expression*

ActionEvent

- Exemplo de uma classe para processar o evento `ActionEvent`

```
class MyHandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent actionEvent) {
        //TODO
    }
}
```

... a qual pode ser associada a um botão da seguinte forma:

```
Button button = new Button("Go");
button.setOnAction(new MyHandler());
```

ActionEvent

- Exemplos com *Lambda Expressions*
  - `setOnAction`

```
Button button = new Button("Increment");
button.setOnAction(actionEvent -> {
    //TODO
});
```

  - `addEventFilter`

```
Button button = new Button("Decrement");
button.addEventFilter(
   ActionEvent.ACTION,
   actionEvent -> {
      //TODO
   }
);
```

    - Com o método `addEventFilter` podem-se tratar diferentes tipos de eventos e podem ser configurados vários processamentos para o mesmo evento