

Programação Orientada por Objetos

JavaFX – Janelas e Formas

Prof. José Cordeiro,

Prof. Cédric Grueau,

Prof. Laercio Júnior

Departamento de Sistemas e Informática

Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal

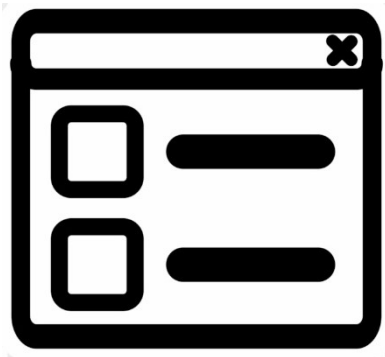
2019/2020

Módulo Janelas e Formas

- ❑ Sessão 1 – Janelas e Diálogos
 - Criação de diálogos

- ❑ Sessão 2 – Diálogo FileChooser
 - **FileChooser**

- ❑ Sessão 3 – Formas e Cores
 - **0 package javafx.scene.shape.*;**
 - ❑ Arc, Circle, CubicCurve, Ellipse,
 - ❑ Line, Path,
 - ❑ Polygon, Polyline,
 - ❑ QuadCurve,
 - ❑ Rectangle,
 - ❑ SVGPath
 - ❑ Text
 - **0 package javafx.scene.paint.*;**
 - **Databinding**



Módulo 14 – JavaFX – Janelas e Formas

SESSÃO 1 – JANELAS E DIÁLOGOS

JavaFX-Novas Janelas

- A classe **Stage** é subclasse de **Window** e permite criar janelas principais, que podem conter objetos gráficos

```
javafx.stage.*;
```

- A outra subclasse de **Window** é a classe, abstrata, **PopupWindow**, que pretende representar janelas auxiliares: tooltips, menus de contexto, etc.
- É possível criar novas janelas, na aplicação, através da criação de novos **Stage**.

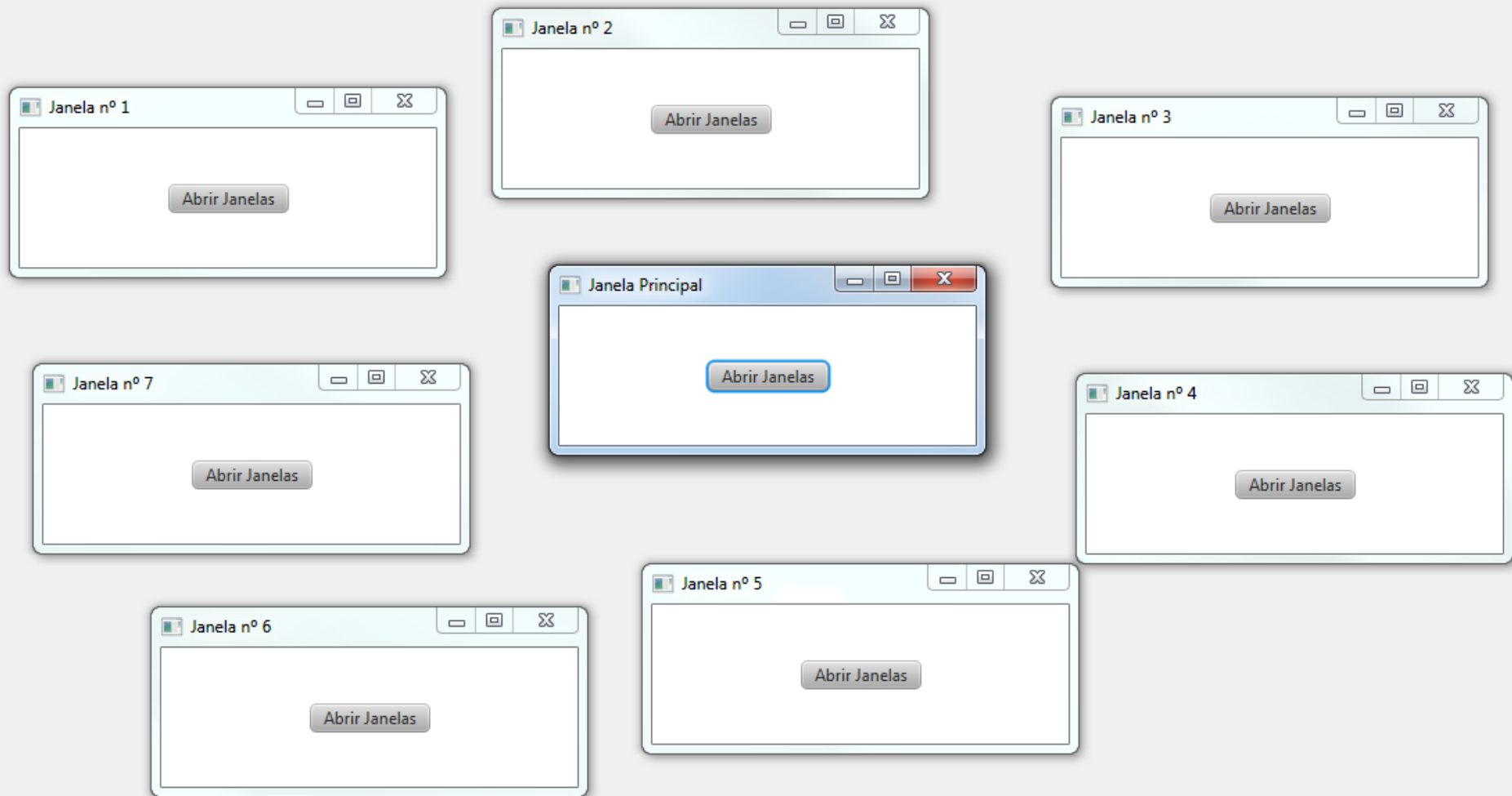
```
@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Janela Principal");
    Scene scene = new Scene(new AbrirJanelas(), 300, 100);
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

```
public class AbrirJanelas extends StackPane {

    private static int quantas = 0;

    public AbrirJanelas() {
        Button botao = new Button("Abrir Janelas");
        botao.setOnAction(e -> {
            Stage janela = new Stage();
            janela.setTitle("Janela nº " + ++quantas);
            janela.setScene(new Scene(new AbrirJanelas(),
                300, 100));
            janela.show();
        });
        this.getChildren().add(botao);
    }
}
```

JavaFX-Novas Janelas

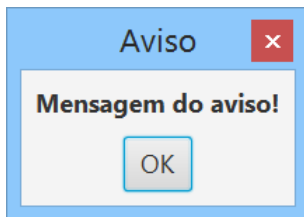


JavaFX - Diálogos

- A classe **Stage** tem um conjunto de métodos que permitem configurar o aspeto e a funcionalidade da janela:
 - **setResizable(boolean)** – indica se é possível modificar o tamanho da Janela
 - **setIconified(boolean)** – indica se é possível minimizar a janela
 - **centerOnScreen()** – apresenta a janela centrada em relação ao ecrã
 - **initStyle(StageStyle)** – indica o estilo da janela:
 - **StageStyle.DECORATED** – janela com fundo branco sólido e com os ícones típicos do S.O.
 - **StageStyle.UNDECORATED** – janela com fundo branco sólido e sem os ícones típicos do S.O.
 - **StageStyle.TRANSPARENT** – janela com fundo transparente e sem os ícones típicos do S.O.
 - **StageStyle.UTILITY** – janela com fundo branco sólido e com o mínimo de ícones típicos do S.O.
 - **initModality(Modality)** – indica o modo de execução da janela:
 - **Modality.NONE** – a janela não bloqueia nada
 - **Modality.WINDOW_MODAL** – a janela impede que os eventos sejam despachados para qualquer uma das janelas da sua ascendência hierárquica.
 - **Modality.APPLICATION_MODAL** – a janela impede que os eventos sejam despachados para qualquer uma das janelas da aplicação.
- Utilizando os métodos anteriores é possível criar pequenas janelas para avisos que funcionam na forma *modal*: obrigam o utilizador a responder, através dos seus botões, pois toda a restante aplicação fica vedada – os diálogos.

JavaFX — Diálogo de Aviso

- É possível criar um diálogo para apresentar avisos, onde existe um botão de **OK** para o utilizador assinalar que leu o aviso.
- No diálogo de aviso é possível indicar o título do diálogo e a mensagem a apresentar (poder-se-ia criar classes mais genéricas para incluir ícones, cores de fundo, etc.)



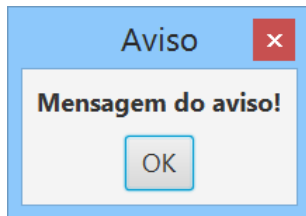
```
public class DialogoAviso extends Stage {  
  
    public DialogoAviso(String titulo, String mensagem) {  
  
        //Criar o painel com a informação  
        VBox painelPrincipal = new VBox(10);  
        painelPrincipal.setPadding(new Insets(10));  
        painelPrincipal.setAlignment(Pos.CENTER);  
  
        //Criar a mensagem:  
        Label apresentacaoMensagem = new Label(mensagem);  
        apresentacaoMensagem.setStyle("-fx-font-weight: bold;");  
  
        //Criar botão:  
        final Button botaoOk = new Button("OK");  
        botaoOk.setOnAction(e ->  
            ((Stage) botaoOk.getScene().getWindow()).close());  
    }  
}
```

JavaFX — Diálogo de Aviso

- Depois de criar todos os elementos do diálogo (mensagem e botão – centrado no painel) é preciso formatar a janela:

setResizable – false
initStyle – UTILITY
initModality – APPLICATION_MODAL
setIconified – false
centerOnScreen

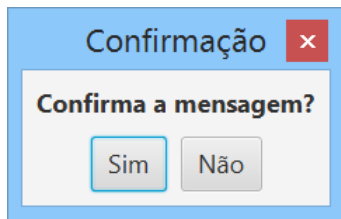
```
//Posicionar os nós:  
painelPrincipal.getChildren()  
    .addAll(apresentacaoMensagem, botaoOk);  
setResizable(false);  
initStyle(StageStyle.UTILITY);  
initModality(Modality.APPLICATION_MODAL);  
setIconified(false);  
centerOnScreen();  
setTitle(titulo);  
setScene(new Scene(painelPrincipal));  
show();
```



```
new DialogoAviso("Aviso", "Mensagem do aviso!");
```


JavaFX — Diálogo de Confirmação

- É possível criar um diálogo para apresentar um pedido de confirmação, onde existem dois botões Sim/Não.
 - Caso o utilizador escolha o "Sim" será executada uma ação.
 - A opção "Não" apenas fecha a janela.
- Neste diálogo é possível indicar o título do diálogo e a mensagem a apresentar, bem como a ação a executar em caso de ser escolhido o "Sim" (poder-se-ia criar classes mais genéricas para incluir ícones, cores de fundo, etc.)

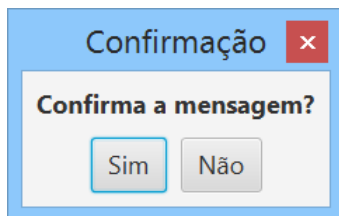


```
public class DialogoConfirmacao extends Stage {  
  
    public DialogoConfirmacao(String titulo,  
                               String mensagem,  
                               final EventHandler<ActionEvent> accaoSim) {  
        //Criar o painel com a informação  
        VBox painelPrincipal = new VBox(10);  
        painelPrincipal.setPadding(new Insets(10));  
        painelPrincipal.setAlignment(Pos.CENTER);  
        //Criar a mensagem:  
        Label apresentacaoMensagem = new Label(mensagem);  
        apresentacaoMensagem  
            .setStyle("-fx-font-weight: bold;");  
        //Criar botões:  
        final HBox painelBotoes = new HBox(10);  
        painelBotoes.setAlignment(Pos.CENTER);  
        Button botaoSim = new Button("Sim");  
        botaoSim.setOnAction(e -> {  
            accaoSim.handle(e);  
            ((Stage) painelBotoes.getScene().getWindow())  
                .close();  
        });  
    }  
}
```

JavaFX – Diálogo de Confirmação

- Depois de criar todos os elementos do diálogo (mensagem e dois botões – centrados no painel) é preciso formatar a janela:

`setResizable – false`
`initStyle – UTILITY`
`initModality – APPLICATION_MODAL`
`setIconified – false`
`centerOnScreen`



```
Button botaoNao = new Button("Não");
botaoNao.setOnAction(e -> ((Stage) painelBotoes
    .get Scene().getWindow()
    ).close());
painelBotoes.getChildren().addAll(botaoSim, botaoNao);

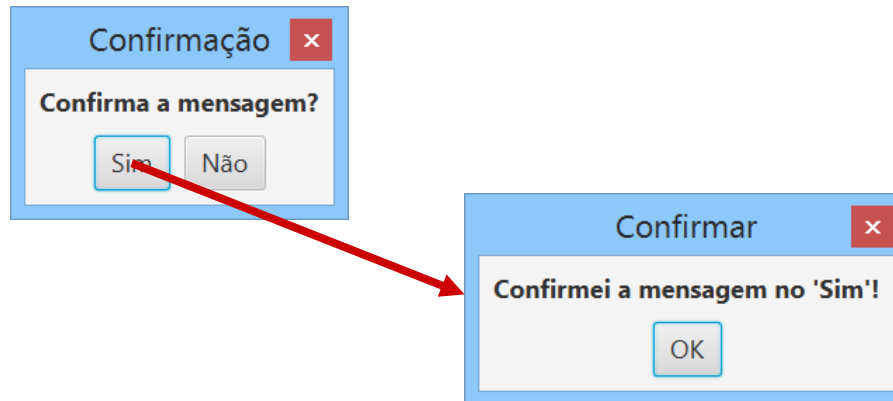
//Posicionar os nós:
painelPrincipal.getChildren()
    .addAll(apresentacaoMensagem, painelBotoes);

setResizable(false);
initStyle(StageStyle.UTILITY);
initModality(Modality.APPLICATION_MODAL);
setIconified(false);
centerOnScreen();
setTitle(titulo);
setScene(new Scene(painelPrincipal));
show();
```

JavaFX — Diálogo de Confirmação

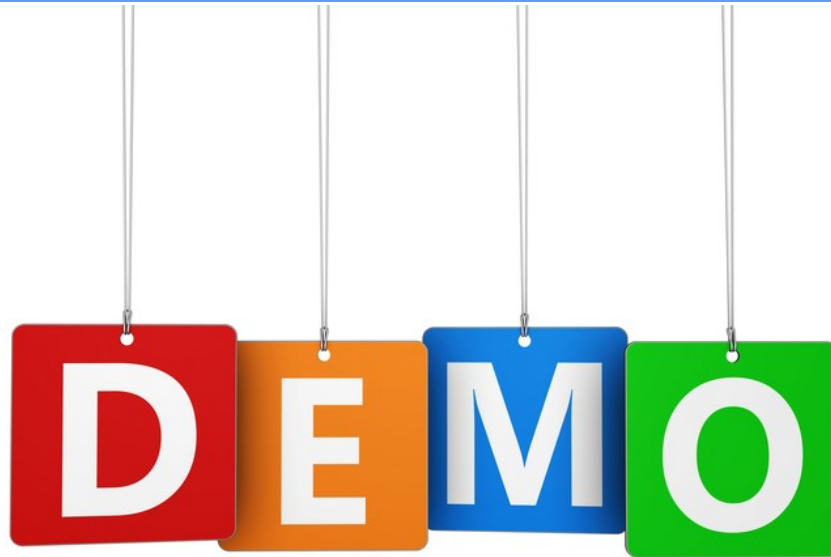
- É possível chamar um Diálogo de Aviso como resposta a um Diálogo de Confirmação:

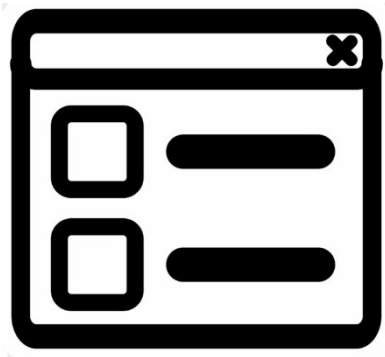
```
new DialogoConfirmacao("Confirmação",  
    "Confirma a mensagem?",  
    e -> new DialogoAviso("Confirmar",  
        "Confirmei a mensagem no 'Sim!'"));
```



- Nota: Seria desejável criar uma super classe **Dialogo** (que herdaria de **Stage**) e que conteria o código repetido (indicado a *bold*) destas duas classes. Estas passariam a subclasses dessa e chamariam o construtor **super()** para "obterem" as configurações comuns.

JavaFX- Eventos : Exemplo





Módulo 14 – JavaFX – Janelas e Formas

SESSÃO 2 – DIÁLOGO FILECHOOSER

JavaFX — Acesso e criação de ficheiros

- ❑ O acesso a ficheiros (obtenção de um ficheiro no *file system* ou criação de um novo ficheiro) é feito através da classe **FileChooser**.
- ❑ A classe **FileChooser** permite indicar os tipos de ficheiros que se podem aceder/criar através de uma lista (`getExtensionFilters().add("...")`) com as diversas extensões permitidas.
- ❑ As extensões permitidas são indicadas através de um **ExtensionFilter**.

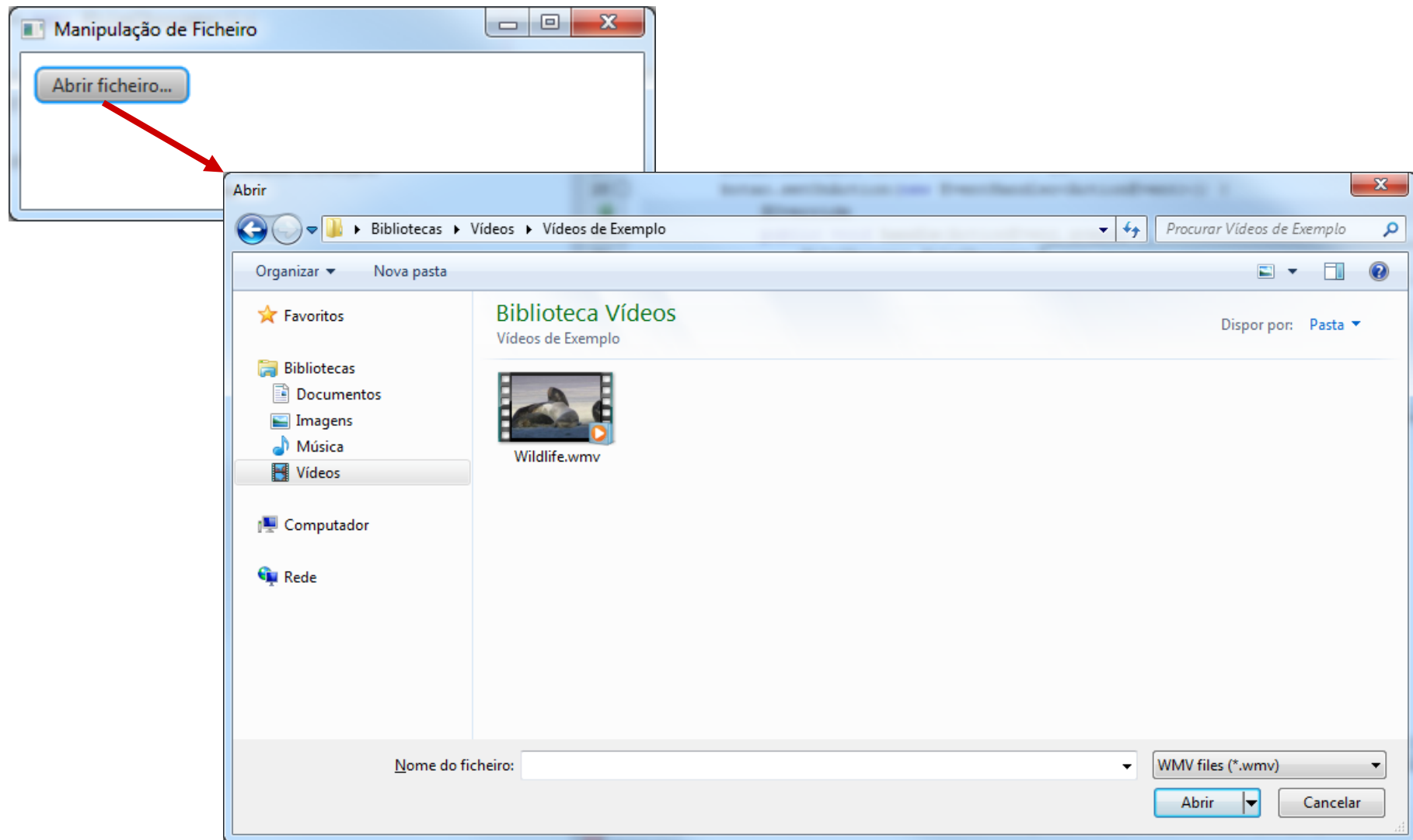
```
@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Manipulação de Ficheiro");
    primaryStage.setScene(new Scene(new ManipularFicheiros(), 400, 100));
    primaryStage.show();
}
```

JavaFX — Acesso e criação de ficheiros

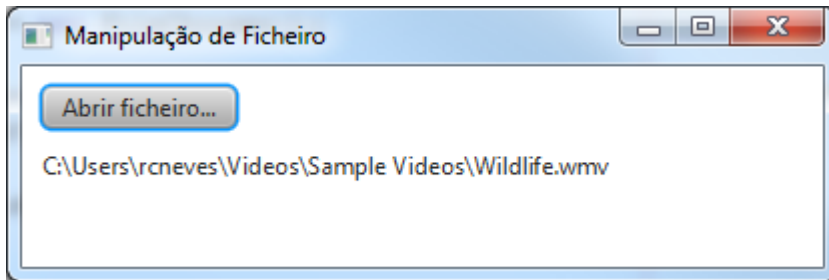
```
public class ManipularFicheiros extends VBox {

    public ManipularFicheiros() {
        final Label nomeDoFicheiro = new Label();
        Button botao = new Button("Abrir ficheiro...");
        botao.setOnAction(e -> {
            FileChooser fileChooser = new FileChooser();
            ExtensionFilter extensao = new ExtensionFilter("WMV files (*.wmv)", "*.wmv");
            fileChooser.getExtensionFilters().add(extensao);
            File file = fileChooser.showOpenDialog(null);
            if (file != null) {
                nomeDoFicheiro.setText(file.getPath());
            } else {
                nomeDoFicheiro.setText("");
            }
        });
        getChildren().addAll(botao, nomeDoFicheiro);
        setPadding(new Insets(10));
        setSpacing(10);
    }
}
```

JavaFX — Acesso e criação de ficheiros

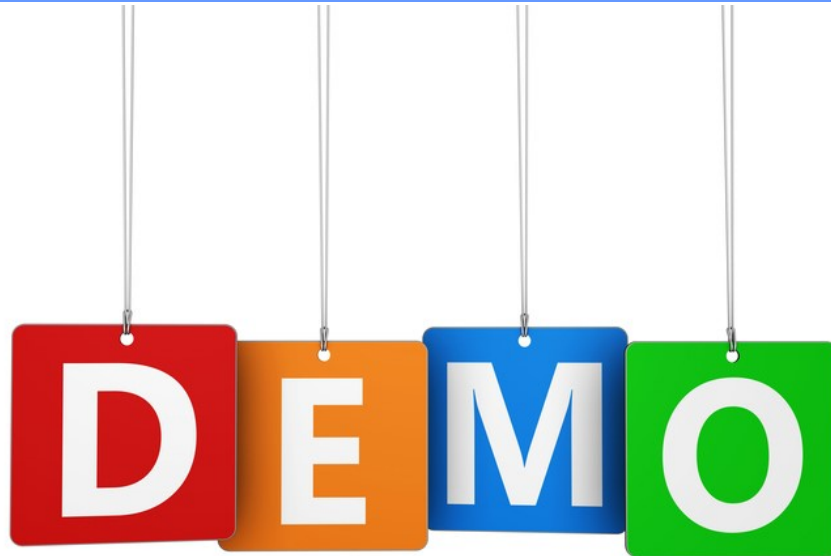


JavaFX — Acesso e criação de ficheiros



- ❑ Com o método **showOpenMultipleDialog** da classe **FileChooser** é possível selecionar mais do que um ficheiro.
- ❑ Com o método **showSaveDialog** da classe **FileChooser** é possível indicar o ficheiro a criar.
- ❑ Os métodos **showOpenDialog**, **showOpenMultipleDialog**, **showSaveDialog** devolvem objetos (ou uma lista de objetos) da classe **File**. O objeto retornado poderá ser utilizado nas classes **FileWriter** e/ou **FileReader**, para permitir a escrita e leitura em ficheiros.
- ❑ A classe **DirectoryChooser** é equivalente ao **FileChooser** mas funciona para diretorias (pastas).

JavaFX- Eventos : Exemplo





Módulo 14 – JavaFX – Janelas e Formas

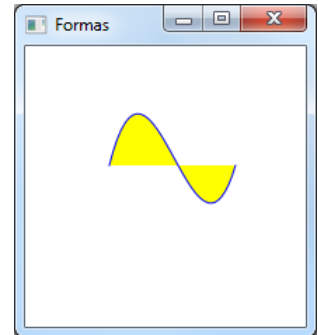
SESSÃO 3 – FORMAS E CORES

JavaFX- Formas

- A classe abstrata **Shape** e as suas subclasses concretas `javafx.scene.shape.*`;

- Para desenhar formas o JavaFX disponibiliza uma série de subclasses de **Shape**: **Arc**, **Circle**, **CubicCurve**, **Ellipse**, **Line**, **Path**, **Polygon**, **Polyline**, **QuadCurve**, **Rectangle**, **SVGPath**, e **Text**

```
import javafx.scene.paint.Color;
import javafx.scene.shape.*;
//outros import omitidos
public class CriarShapes extends Application {
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Formas");
        Group root = new Group();
        Scene scene = new Scene(root, 200, 200);
        CubicCurve curva = new CubicCurve();
        curva.setStartX(60);
        curva.setStartY(85);
        curva.setControlX1(90);
        curva.setControlY1(-35);
        curva.setControlX2(120);
        curva.setControlY2(185);
        curva.setEndX(150);
        curva.setEndY(85);
        curva.setStrokeType(StrokeType.CENTERED);
        curva.setStrokeWidth(1);
        curva.setStroke(Color.BLUE);
        curva.setFill(Color.YELLOW);
        root.getChildren().add(curva);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    //método main() omitido
}
```



JavaFX- Cores

- O package **paint**

`javafx.scene.paint.*;`

- Disponibiliza um conjunto de classes para criar cores e gradientes usados para preencher formas e fundos de janelas ao apresentar o grafo de cena.

- A aplicação de cores a uma **Shape** passa por duas etapas:

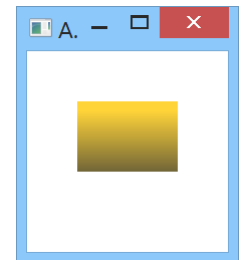
1. Criar a cor ou o gradiente com o construtor apropriado
2. Alterar a propriedade a colorir com o `setXxx` da **Shape**.

```
import javafx.scene.paint.*;
public class AtribuirCores extends Application {
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Atribuir Cores");
        Group root = new Group();
        Scene scene = new Scene(root, 200, 200);
        Rectangle rectangle = new Rectangle(50, 50, 100, 70);

        LinearGradient linearGradiente
            = new LinearGradient(
                50, //startX
                50, //startY
                50, //endX
                50 + rectangle.prefHeight(-1) + 25, //endY
                false, //proportional
                CycleMethod.NO_CYCLE, //cycleMethod
                new Stop(0.1f, Color.rgb(255, 200, 0, 0.784)), //stops
                new Stop(1.0f, Color.rgb(0, 0, 0, 0.784)));

        rectangle.setFill(linearGradiente);

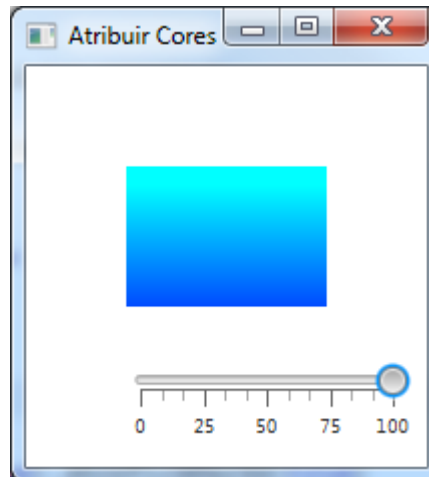
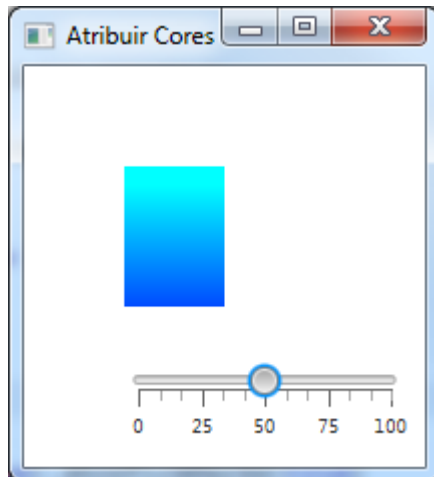
        root.getChildren().add(rectangle);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    //método main() omitido
}
```



Databinding

- ❑ Anteriormente, falou-se de **DataBinding**.
- ❑ Pegando no último exemplo de desenhar um retângulo, vamos mostrar como associar a largura do retângulo ao valor de um **Slider**.
- ❑ Para tal vamos usar o método **bind** (unidirecional) e associar a propriedade do comprimento do retângulo - **widthProperty()** à propriedade valor do **Slider** - **valueProperty()**.

```
rectangle.widthProperty().bind(slider.valueProperty());
```



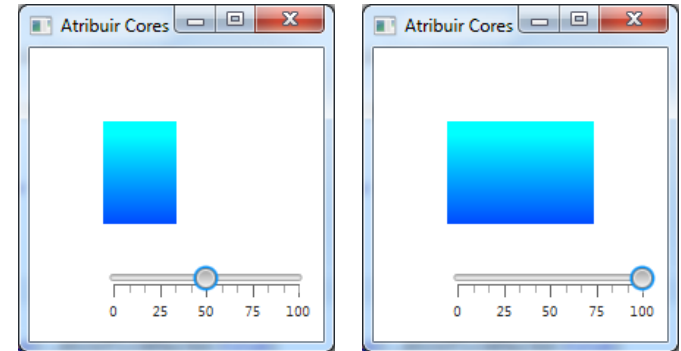
Databinding

□ ○ exemplo completo:

```
public void start(Stage primaryStage) {
    primaryStage.setTitle("Atribuir Cores");
    Group root = new Group();
    Scene scene = new Scene(root, 200, 200);
    Rectangle rectangle = new Rectangle(50, 50, 100, 70);
    LinearGradient linearGradient
        = new LinearGradient(
            50, //startX
            50, //startY
            50, //endX
            50 + rectangle.prefHeight(-1) + 25, //endY
            false, //proportional
            CycleMethod.NO_CYCLE, //cycleMethod
            new Stop(0.1f, Color.AQUA), new Stop(1.0f, Color.BLUE)); //stops
    rectangle.setFill(linearGradient);
    Slider slider = new Slider(0, 100, 50);
    slider.setLayoutX(50);
    slider.setLayoutY(150);
    slider.setMinorTickCount(2);
    slider.setShowTickLabels(true);
    slider.setShowTickMarks(true);

    rectangle.widthProperty().bind(slider.valueProperty());

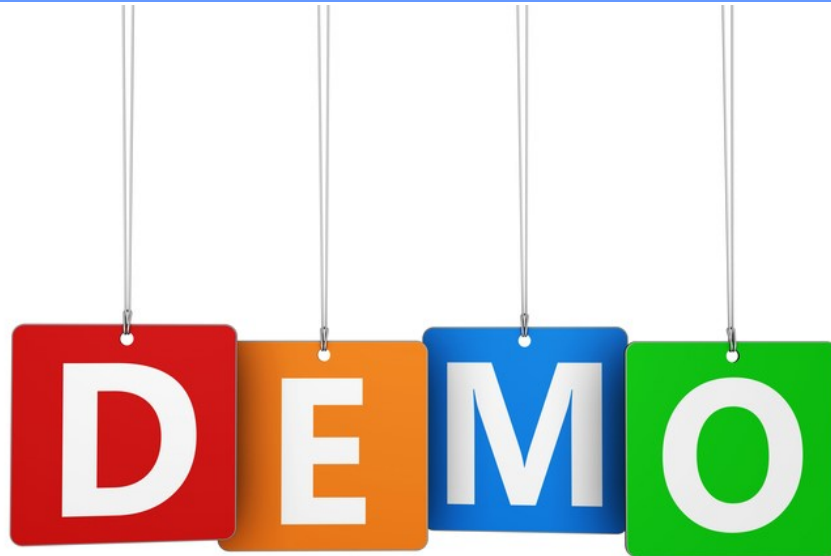
    root.getChildren().addAll(rectangle, slider);
    primaryStage.setScene(scene);
    primaryStage.show();
}
```



Resumindo

- ❑ Criação de novas janelas
 - Diálogos
 - ❑ Criação de Diálogos
 - ❑ **FileChooser** e **DirectoryChooser**
- ❑ Formas
 - O package **javafx.scene.shape.***;
 - A classe abstrata **javafx.scene.shape.Shape**;
- ❑ Cores
 - O package **javafx.scene.paint.***;
 - A classe **javafx.scene.paint.Color**;
 - A aplicação de cores a uma **Shape** passa por duas etapas:
 - ❑ 1. Criar a cor ou o gradiente com o construtor da classe;
 - ❑ 2. Alterar a propriedade a colorir com o **setXxx** da **Shape**.
- ❑ Databinding para modificar os valores das propriedades dos objetos

JavaFX- Eventos : Exemplo



Chapter 2 - JavaFX Fundamentals Pgs 31 a 53

- Sobre a Janelas e Diálogos:
 - <http://docs.oracle.com/javase/8/javafx/api/javafx/stage/package-summary.html>
- Sobre a classe abstrata Shape:
 - <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/shape/Shape.html>
- Sobre o package paint
 - <http://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/package-summary.html>

