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 4 – Diálogo Alert
- Sessão 5 – Diálogos ChoiceDialog e TextInputDialog
- Sessão 6 – Diálogos Personalizados
- Sessão 7 – Exemplo Visualizador Pessoa

Módulo 14 – JavaFX – Diálogos

SESSÃO 4 – DIÁLOGO ALERT

JavaFX – Diálogo de Alerta

- Os diálogos são janelas independentes utilizadas para mostrar ou recolher informação do utilizador. O JavaFX, a partir da versão 8, disponibilizou no pacote dos controlos a classe **Dialog** que funciona como a classe base dos diálogos.

- A classe **Alert** é uma subclasse de **Dialog**, usada para mostrar ou recolher informações simples.

- A informação mostrada pela classe **Alert** pode ser uma mensagem simples, de erro ou de aviso ou ainda, um pedido de confirmação.

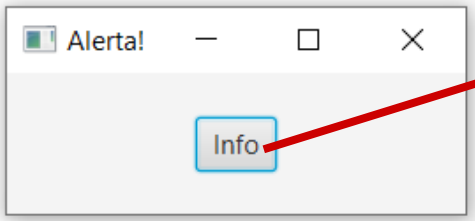
```
@Override
public void start(Stage primaryStage) {
    Button btn = new Button();
    btn.setText("Info");
    btn.setOnAction(e -> mostrarInformacao());
    StackPane root = new StackPane();
    root.getChildren().add(btn);
    Scene scene = new Scene(root, 300, 250);
    primaryStage.setTitle("Alerta!");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

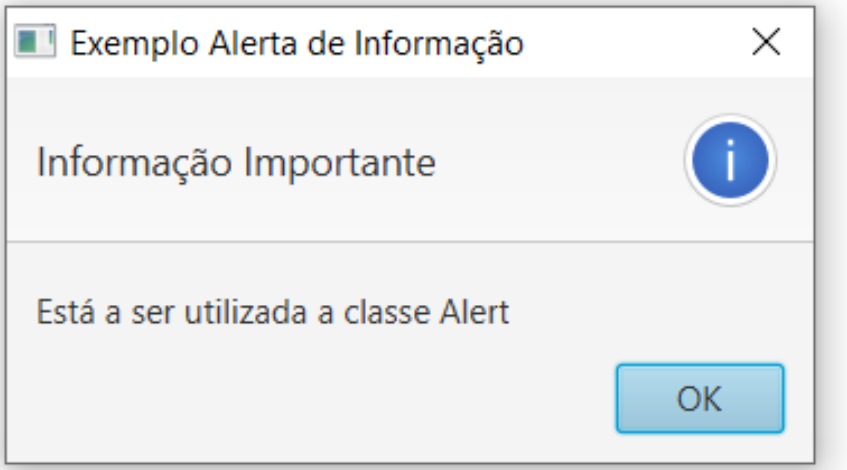
```
private void mostrarInformacao() {
    Alert alert = new Alert(AlertType.INFORMATION);
    alert.setTitle("Exemplo Alerta de Informação");
    alert.setHeaderText("Informação Importante");
    String s = "Está a ser utilizada a classe Alert";
    alert.setContentText(s);
    alert.show();
}
```

JavaFX – Diálogo de Alerta

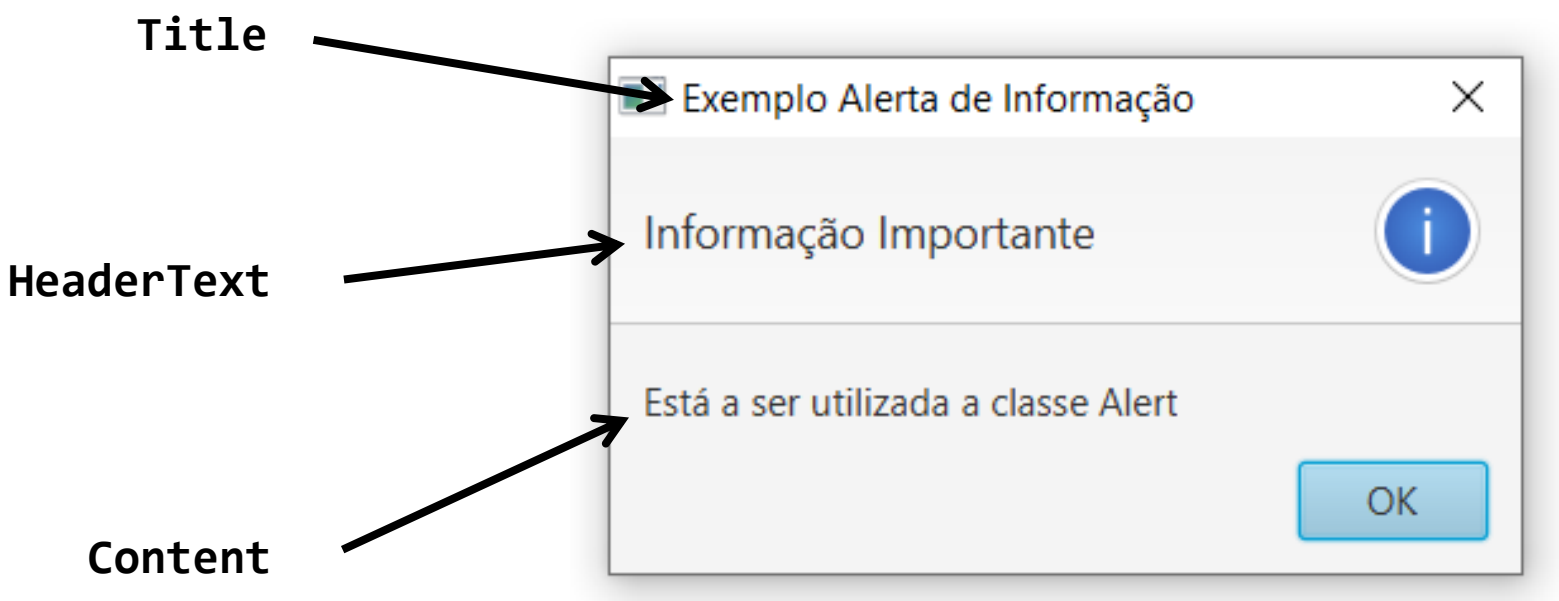
- Para definir qual o tipo de alerta que queremos criar usamos o tipo enumerado **`AlertType`**.
- Para um alerta que mostra simplesmente uma mensagem usamos o valor: **`AlertType.INFORMATION`**
- O método **`setTitle`** permite-nos definir o título da janela, **`setHeaderText`** o cabeçalho da mensagem e **`setContent`** a mensagem. **`show`** mostra o diálogo sem bloquear a janela da aplicação (não modal).

```
private void mostrarInformacao() {
    Alert alert = new Alert(AlertType.INFORMATION);

    alert.setTitle("Exemplo Alerta de Informação");

    alert.setHeaderText("Informação Importante");

    String s = "Está a ser utilizada a classe Alert";
    alert.setContentText(s);

    alert.show();
}
```





JavaFX – Diálogo de Alerta

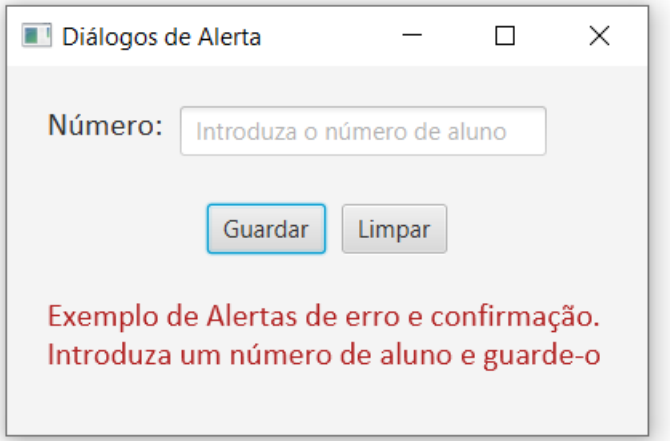


- Além do alerta anterior, `AlertType.INFORMATION` a classe **`Alert`** permite criar alertas dos seguintes tipos:

```
AlertType.ERROR
AlertType.WARNING
AlertType.CONFIRMATION
```

JavaFX – Diálogo de Alerta

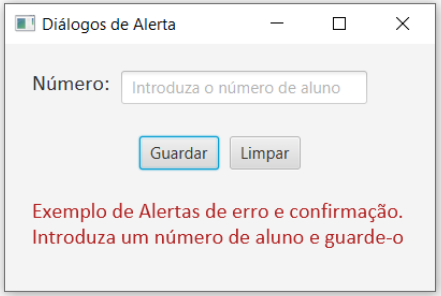
Objetivo: Criar uma aplicação para receber o número de aluno de um estudante.



- Se o número introduzido for inválido deve ser mostrada uma mensagem de erro.
- Se o número introduzido estiver correto deve ser mostrada uma mensagem de confirmação.

JavaFX – Diálogo de Alerta

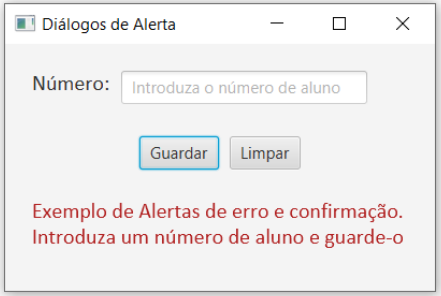
- No método **start** é definido o conteúdo da janela principal e a ação dos botões.




```
public class ExemploAlertaConfirmacao extends Application {
    private Text estado;
    private TextField txtNumero;
    @Override
    public void start(Stage primaryStage) {
        // Receber número
        Label lblNumero = new Label("Número:");
        lblNumero.setFont(Font.font("Calibri", FontWeight.NORMAL, 20));
        txtNumero = new TextField();
        txtNumero.setMinHeight(30.0);
        txtNumero.setPromptText("Introduza o número de aluno");
        txtNumero.setPrefColumnCount(15);
        HBox hbNumero = new HBox();
        hbNumero.setSpacing(10);
        hbNumero.getChildren().addAll(lblNumero, txtNumero);
        // Botões
        Button btnGuardar = new Button("Guardar");
        btnGuardar.setOnAction(e -> {
            if (!verificarNumero()) {  mostrarErro();
            } else {  confirmarGravacao();
            }
        });
        HBox hbBotoes = new HBox(10);
        Button btnLimpar = new Button("Limpar");
        btnLimpar.setOnAction(e -> limpar());
        hbBotoes.setAlignment(Pos.CENTER);
        hbBotoes.getChildren().addAll(btnGuardar, btnLimpar);
        // Continua…
```

JavaFX – Diálogo de Alerta

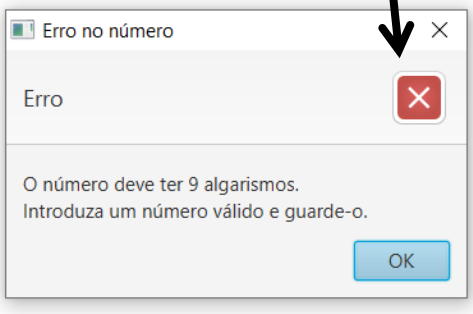
- No método **start** é definido o conteúdo da janela principal e a ação dos botões.




```
        // Continuação da classe de aplicação

        // Mensagem de estado
        estado = new Text();
        estado.setFont(Font.font("Calibri", FontWeight.NORMAL, 20));
        estado.setFill(Color.FIREBRICK);

        // Painel principal
        VBox root = new VBox(30);
        root.setPadding(new Insets(25, 25, 25, 25));;
        root.getChildren().addAll(hbNumero, hbBotoes, estado);

        Scene scene = new Scene(root, 400, 230);

        primaryStage.setTitle("Diálogos de Alerta");
        primaryStage.setScene(scene);
        primaryStage.show();

        // Inicial
        estado.setText("Exemplo de Alertas de erro e confirmação. “ +
                       "\nIntroduza um número de aluno e guarde-o");
        btnGuardar.requestFocus();
    }

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

    // Continua…
```

JavaFX – Diálogo de Alerta

- Um **alerta de erro** é semelhante aos alertas anteriores com a diferença no ícone de erro que é mostrado.

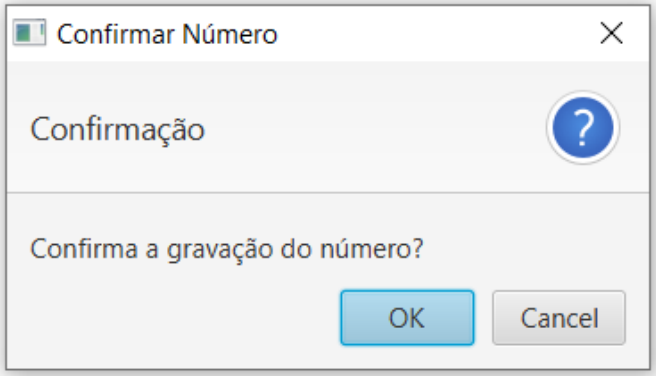



```
    // Continuação da classe de aplicação

private boolean verificarNumero() {
    String numero = txtNumero.getText().trim();
    if (numero.isEmpty() || numero.length() != 9) {
        return false;
    }
    if (numero.chars().anyMatch(c -> !Character.isDigit(c))) {
        return false;
    }
    return true;
}

private void mostrarErro() {
    Alert alert = new Alert(AlertType.ERROR);
    alert.setTitle("Erro no número");
    alert.setHeaderText("Erro");
    String s = "O número deve ter 9 algarismos. "
            + "\nIntroduza um número válido e guarde-o. ";
    alert.setContentText(s);

    alert.showAndWait();
    estado.setText("Foi introduzido um número inválido");
}

private void limpar() {
    txtNumero.setText("");
    estado.setText("Introduza um número de aluno");
}
// Continua…
```

JavaFX – Diálogo de Alerta

- Um **alerta de confirmação** recolhe o tipo de botão que o utilizador selecionou que é um enumerado **ButtonType**
- O método **isPresent()** permite saber se o resultado existe e o método **get** devolve o resultado

```
    // Continuação da classe de aplicação
    private void confirmarGravacao() {
        Alert alert = new Alert(AlertType.CONFIRMATION);
        alert.setTitle("Confirmar Número");
        alert.setHeaderText("Confirmação");
        String s = "Confirma a gravação do número?";
        alert.setContentText(s);

        Optional<ButtonType> resultado = alert.showAndWait();
        if ((resultado.isPresent()) && (resultado.get() == ButtonType.OK))
        {
            estado.setText("Número aceite: " + txtNumero.getText());
            txtNumero.setText("");
        }
        else
            estado.setText("Introduza um número de aluno.");
    }
}
```



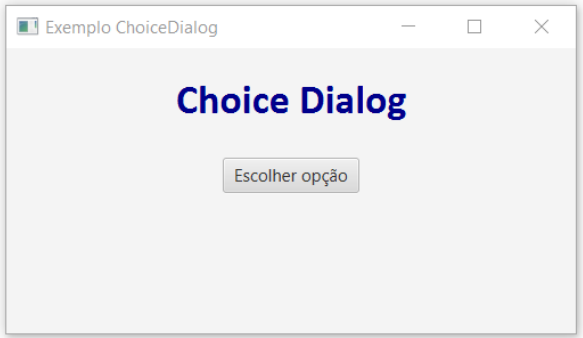
JavaFX- Eventos : Exemplo

Módulo 14 – JavaFX – Janelas e Formas

SESSÃO 5 – CHOICEDIALOG, TEXTINPUTDIALOG

JavaFX – Diálogo de Escolha

- A classe **ChoiceDialog** é uma subclasse de **Dialog**, e é usada para a escolha de uma opção a partir de uma lista de opções.



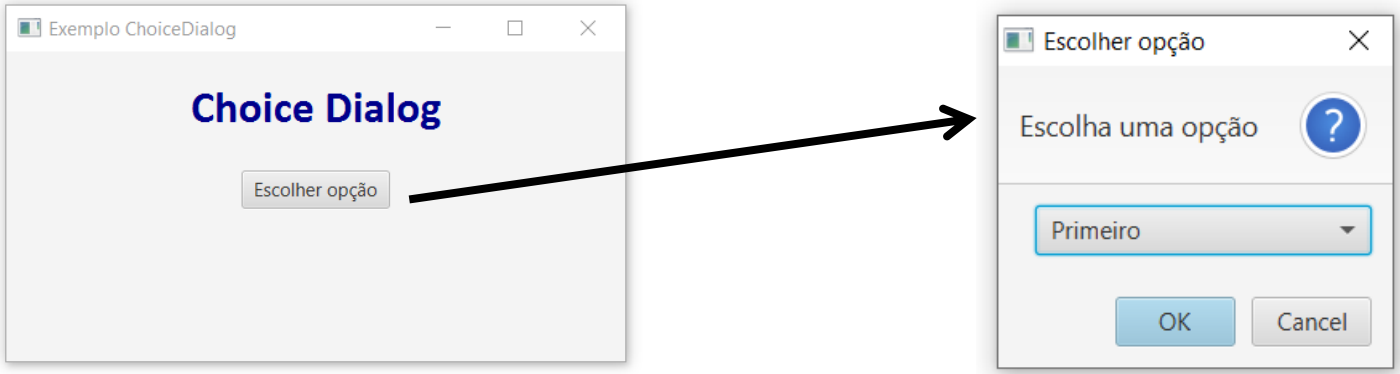
- **Nota:** No método start mostrado foram retiradas as instruções de formatação. O exemplo completo é disponibilizado com estes slides.

```
private ChoiceDialog<String> dialogo;
private final String[] arrayDados = {"Primeiro", "Segundo",
                                     "Terceiro", "Quarto"};
private List<String> dadosDialogo;
private Text estado;
@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Exemplo ChoiceDialog");
    // Título
    Label lblTitulo = new Label("Choice Dialog");
    HBox hbTitulo = new HBox();
    hbTitulo.getChildren().add(lblTitulo);
    // Botão
    Button btnEscolha = new Button("Escolher opção");
    btnEscolha.setOnAction(e -> escolherOpcao());
    HBox hbBotao = new HBox(10);
    hbBotao.getChildren().addAll(btnEscolha);
    // Status message text
    estado = new Text();
    // Painel principal
    VBox root = new VBox(30);
    root.getChildren().addAll(hbTitulo, hbBotao, estado);
    // Scene
    Scene scene = new Scene(root, 500, 250); // w x h
    primaryStage.setScene(scene);
    primaryStage.show();
    // Initial dialog
    dadosDialogo = Arrays.asList(arrayDados);
    escolherOpcao();
}
```

JavaFX – Diálogo de Escolha

- As opções de escolha de um **ChoiceDialog** são mostradas numa **ComboBox**.
- O resultado da escolha é devolvido num objeto da classe **Optional<T>**. Neste caso, usa-se o método **ifPresent()** para saber se existe um objeto devolvido (não **null**) e o método **get()** para obter o objeto.

```
private void escolherOpcao() {
    estado.setText("");
    dialogo = new ChoiceDialog<String>(dadosDialogo.get(0),
                                       dadosDialogo);
    dialogo.setTitle("Escolher opção");
    dialogo.setHeaderText("Escolha uma opção");

    Optional<String> resultado = dialogo.showAndWait();
    String selecionado = "cancelado.";

    if (resultado.isPresent()) {
        selecionado = resultado.get();
    }

    estado.setText("Seleção: " + selecionado);
}
```



JavaFX – Diálogo de Introdução de Texto

- **`TextInputDialog`** é também uma subclasse de **`Dialog`**, usada para a recolha de um texto fornecido pelo utilizador.

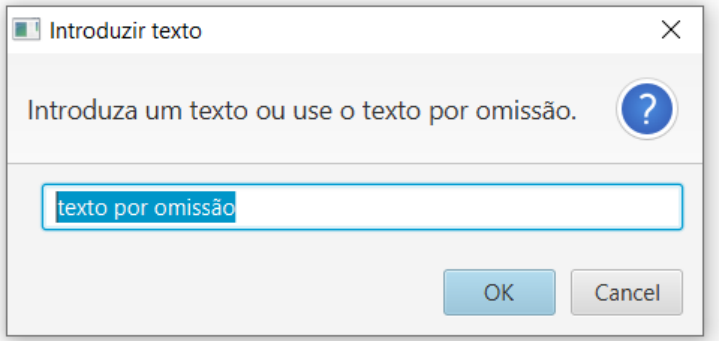
- **Nota:** No método start mostrado foram retiradas as instruções de formatação. O exemplo completo é disponibilizado com estes slides.

```
private TextInputDialog dialogo;
private final String textoOmissao = "texto por omissão";
private Text mensagemEstado;

@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Exemplo TextInputDialog");
    // Título
    Label lblTitulo = new Label("Text Input Dialog");
    HBox hbTitulo = new HBox();
    hbTitulo.getChildren().add(lblTitulo);
    // Botão
    Button btnObterTexto = new Button("Obter texto");
    btnObterTexto.setOnAction(e -> obterTexto());
    HBox hbObterTexto = new HBox(10);
    hbObterTexto.getChildren().addAll(btnObterTexto);
    // Mensagem de estado
    mensagemEstado = new Text();
    mensagemEstado.setFill(Color.FIREBRICK);
    // Painel principal
    VBox root = new VBox(30);
    root.getChildren().addAll(hbTitulo, hbObterTexto,
                              mensagemEstado);
    // Scene
    Scene scene = new Scene(root, 500, 250); // w x h
    primaryStage.setScene(scene);
    primaryStage.show();
    // diálogo inicial
    obterTexto();
}
```

JavaFX – Diálogo de Introdução de texto

- O texto de um **`TextInputDialog`** é recolhido num **`TextField`**.
- O resultado da escolha é devolvido num objeto da classe **`Optional<T>`** como anteriormente.

```
private void obterTexto() {
    mensagemEstado.setText("");

    dialogo = new TextInputDialog(textoOmissao);
    dialogo.setTitle("Introduzir texto");
    dialogo.setHeaderText("Introduza um texto ou use " +
                          "o texto por omissão.");
    Optional<String> resultado = dialogo.showAndWait();
    String introduzido = "nenhum.";
    if (resultado.isPresent()) {
        introduzido = resultado.get();
    }

    mensagemEstado.setText("Texto introduzido: " +
                           introduzido);
}
```



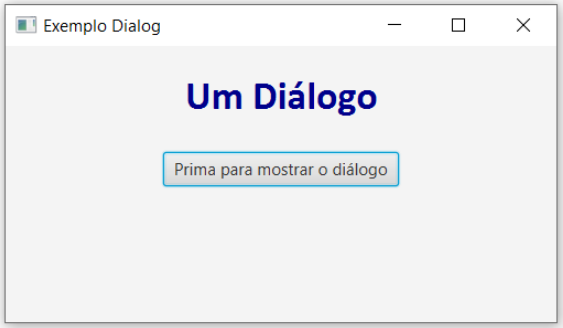
JavaFX- Eventos : Exemplo

Módulo 14 – JavaFX – Diálogos

SESSÃO 6 – DIÁLOGOS PERSONALIZADOS

JavaFX – Diálogos Personalizados

- Para a criação de diálogos em que queremos definir o conteúdo da janela usamos a classe **Dialog**.




```
private Text estado;
@Override
public void start(Stage primaryStage) {
    primaryStage.setTitle("Exemplo Dialog");
    // Título
    Label lblTitulo = new Label("Um Diálogo");
    lblTitulo.setTextFill(Color.DARKBLUE);
    lblTitulo.setFont(Font.font("Calibri", FontWeight.BOLD, 36));
    HBox hbTitulo = new HBox();
    hbTitulo.setAlignment(Pos.CENTER);
    hbTitulo.getChildren().add(lblTitulo);
    // Botão
    Button btnMostrarDialogo = new Button("Mostrar o diálogo");
    btnMostrarDialogo.setOnAction(e -> mostrarDialogo());
    HBox hbMostrarDialogo = new HBox(10);
    hbMostrarDialogo.setAlignment(Pos.CENTER);
    hbMostrarDialogo.getChildren().addAll(btnMostrarDialogo);
    // Mensagem de estado
    estado = new Text();
    estado.setFont(Font.font("Calibri", FontWeight.NORMAL, 20));
    estado.setFill(Color.FIREBRICK);
    // Painel Principal
    VBox root = new VBox(30);
    root.setPadding(new Insets(25, 25, 25, 25));;
    root.getChildren().addAll(hbTitulo, hbMostrarDialogo, estado);
    // Scene
    Scene scene = new Scene(root, 500, 250); // w x h
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

JavaFX – Diálogos Personalizados

Objetivo: Criar um diálogo para receber o nome e telefone de um contacto.

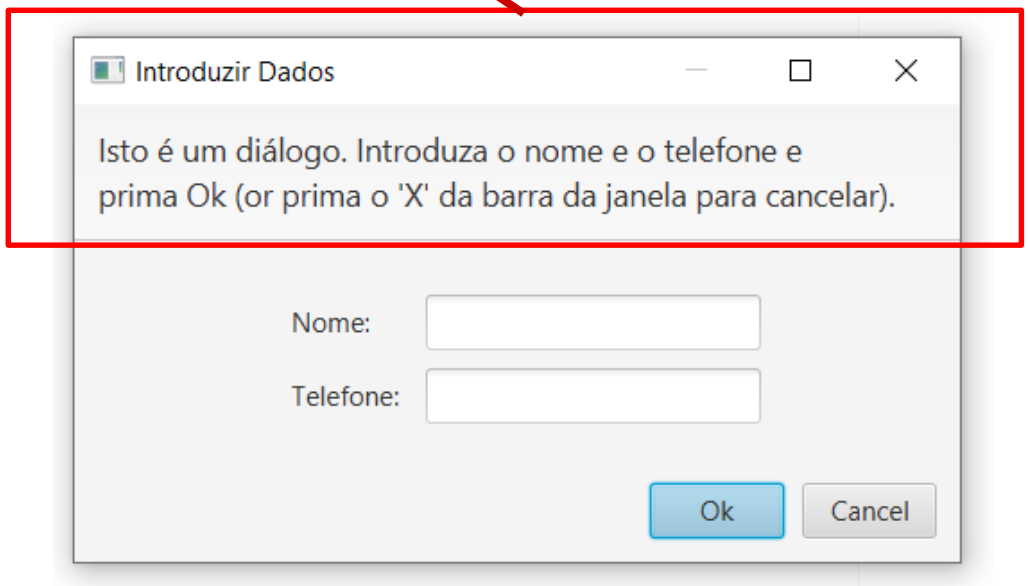
```
public class Contacto {

    private String nome;
    private String telefone;

    Contacto(String nome, String telefone) {

        this.nome = nome;
        this.telefone = telefone;
    }

    @Override
    public String toString() {

        return (nome + ", " + telefone);
    }
}
```

- A classe **`Contacto`** neste exemplo representa o contacto guardando a informação do nome e telefone nos seus atributos.

JavaFX – Diálogos Personalizados

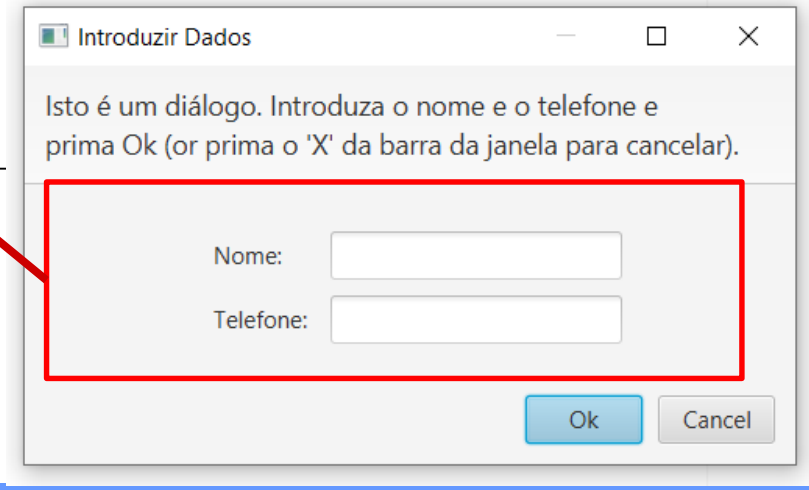
```
private void mostrarDialogo() {
    estado.setText("");
    // Dialog Personalizado
    Dialog<Contacto> dialog = new Dialog<>();
    dialog.setTitle("Introduzir Dados");
    dialog.setHeaderText("Isto é um diálogo. Introduza o nome e o telefone e \n"
            + "prima Ok (or prima o 'X' da barra da janela para cancelar).");
    dialog.setResizable(true);

    // Continua…
```

- Neste exemplo, o método **`mostrarDialogo()`** vai criar o diálogo e estabelecer o aspeto e a informação geral que é apresentada. Mais tarde, é criado o painel usado como conteúdo do diálogo. Finalmente é recolhido o resultado com o contacto introduzido pelo utilizador.



JavaFX – Diálogos Personalizados

```
// Widgets
Label lblNome = new Label("Nome: ");
Label lblTelefone = new Label("Telefone: ");
TextField txtNome = new TextField();
TextField txtTelefone = new TextField();
// Criar o layout e adicioná-lo ao diálogo
GridPane painelDados = new GridPane();
painelDados.setAlignment(Pos.CENTER);
painelDados.setHgap(10);
painelDados.setVgap(10);
painelDados.setPadding(new Insets(20, 35, 20, 35));
painelDados.add(lblNome, 1, 1); // col=1, row=1
painelDados.add(txtNome, 2, 1);
painelDados.add(lblTelefone, 1, 2); // col=1, row=2
painelDados.add(txtTelefone, 2, 2);
dialog.getDialogPane().setContent(painelDados);

// Continua…
```

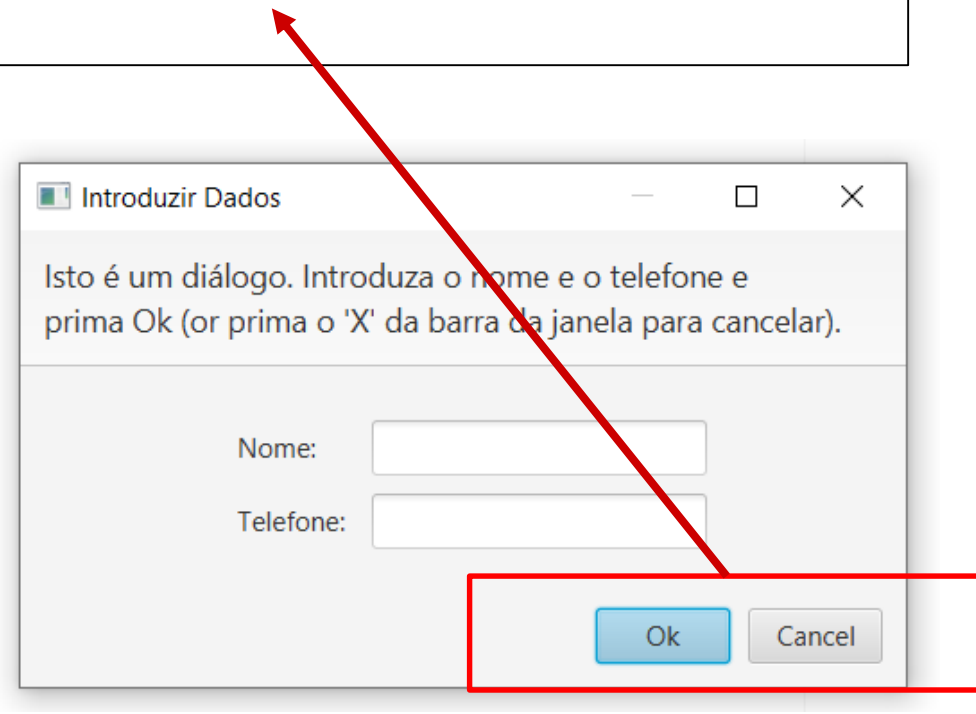


- Criação do painel **`GridPane`** usado para colocar os controlos que recebem a informação do contacto. O método **`setContent`** é usado para colocar este painel como conteúdo do diálogo.

JavaFX – Diálogos Personalizados

```
// Adicionar os botões ao diálogo
ButtonType btnOk = new ButtonType("Ok", ButtonData.OK_DONE);
ButtonType btnCancel = new ButtonType("Cancel", ButtonData.CANCEL_CLOSE);

dialog.getDialogPane().getButtonTypes().addAll(btnOk, btnCancel);

// Continua…
```

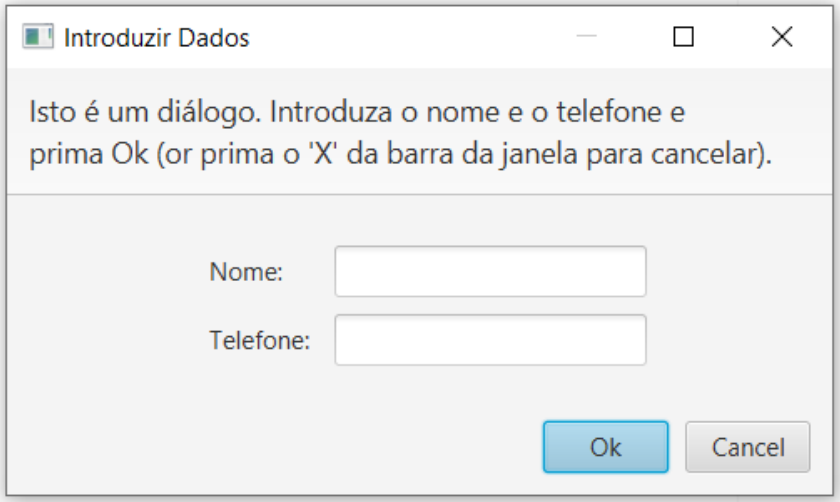
- Para adicionar botões ao diálogo, começa-se por criar os botões que devem ser do tipo **`ButtonType`**. A seguir, recorre-se ao método **`getDialogPane`**. Este método dá acesso ao painel do diálogo. Depois, é só aceder à coleção de botões usando o método **`getButtonTypes`** e adicionar os botões que foram definidos antes.



JavaFX – Diálogos Personalizados

```
// Conversor do resultado para o diálogo
dialog.setResultConverter(new Callback<ButtonType, Contacto>() {
    @Override
    public Contacto call(ButtonType btn) {
        if (btn == btnOk) {
            return new Contacto(txtNome.getText(), txtTelefone.getText());
        }
        return null;
    }
});
// Continua…
```

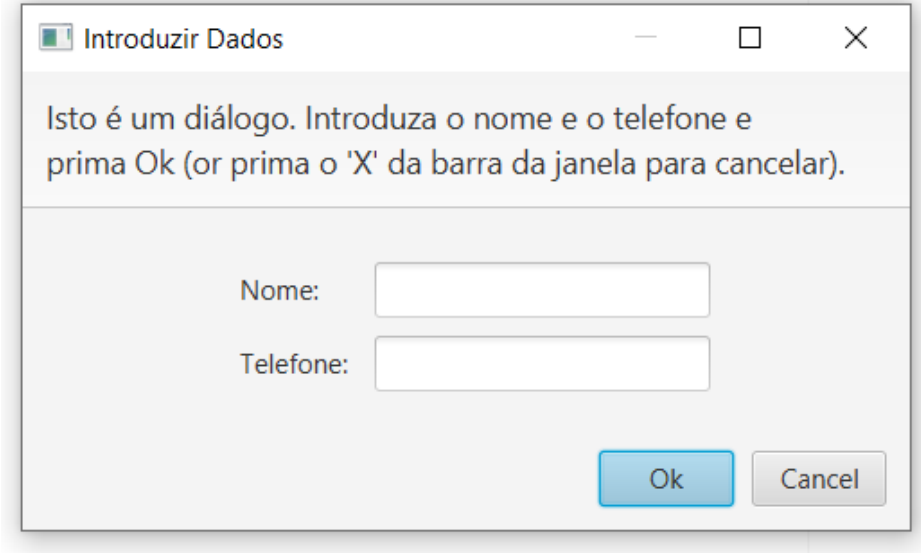
- Para definir o resultado do diálogo, neste caso um objeto da classe **`Contacto`**, usa-se um conversor de resultado (**`ResultConverter`**) que é definido no método **`setResultConverter`**
- O argumento deste método é um objeto de uma classe que implementa a interface **`Callback<P,R>`**. O único método desta interface é o método **`call`** que será chamado quando o diálogo for fechado.
- **P** define o tipo de valor que o método **`call`** recebe e **R** o tipo do resultado.



JavaFX – Diálogos Personalizados

```
        // Mostrar o diálogo
        Optional<Contacto> resultado = dialog.showAndWait();

        if (resultado.isPresent()) {
            estado.setText("Resultado: " + resultado.get());
        }

    } // Fim do método mostrarDialogo
```

- Para concluir o método **`mostrarDialogo`**, falta mostrar o diálogo, o que é feito recorrendo ao método **`showAndWait()`**.
- Depois tem-se acesso ao resultado usando um objeto da classe **`Optional<Contacto>`** explicada neste módulo.



JavaFX- Eventos : Exemplo

Módulo 14 – JavaFX – Diálogos

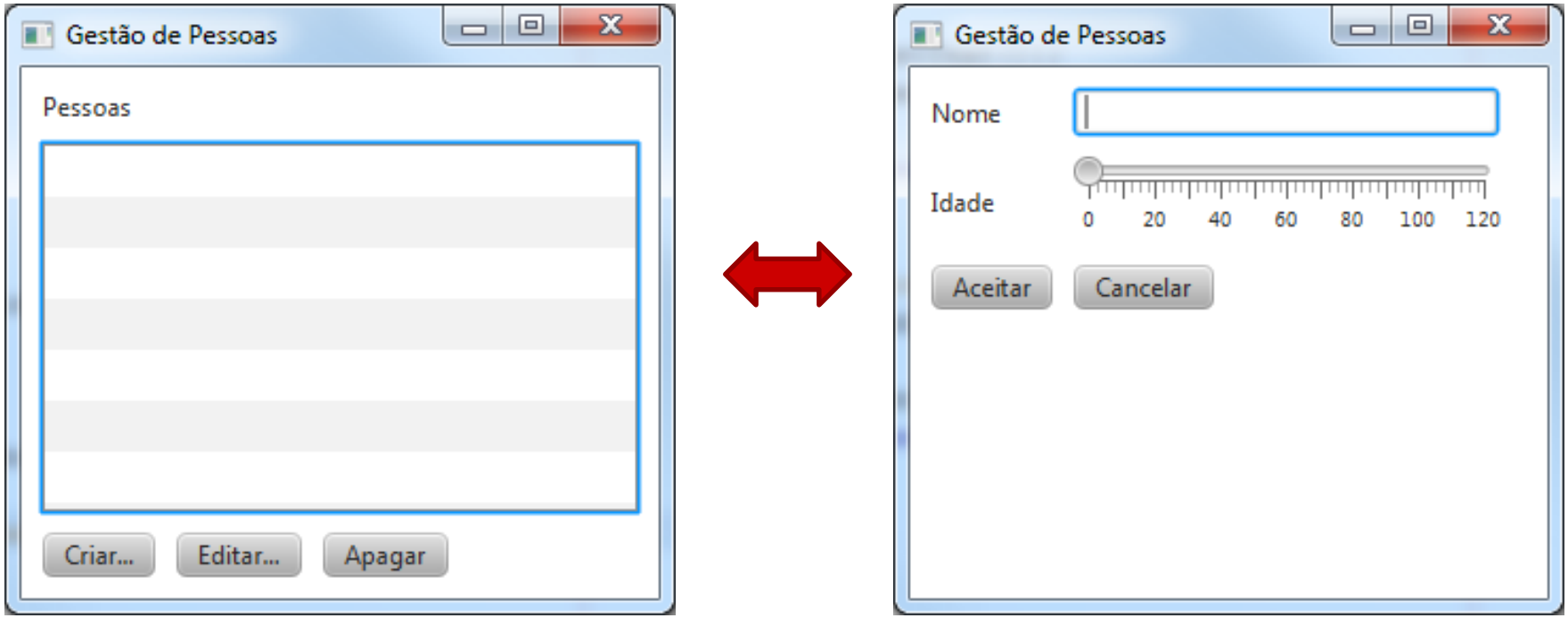
SESSÃO 7 – VISUALIZADOR PESSOA

Exemplo – Visualizador de Pessoas

- Requisitos da aplicação:
  - Criar uma aplicação que permita gerir a informação de um grupo de pessoas.
  - A informação a guardar para cada pessoa é apenas o nome e a idade.
  - Deve ser possível adicionar, alterar, visualizar e apagar uma pessoa.
  - A visualização deve ser feita em JavaFX em duas cenas diferentes, uma que mostra as pessoas e outra que visualiza apenas uma pessoa.

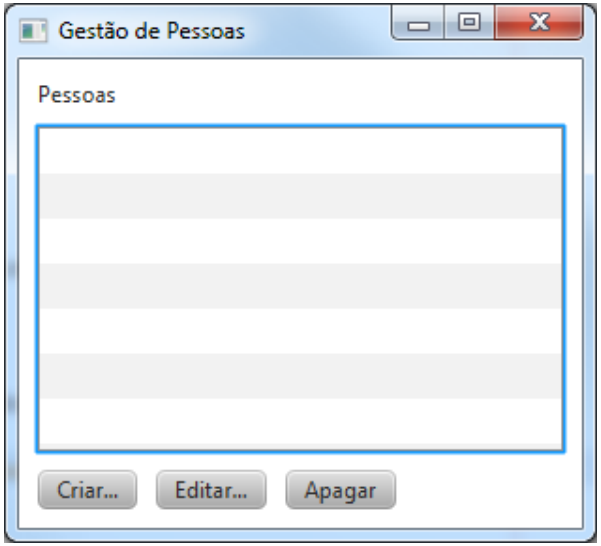
Definir a interface gráfica

- A interface gráfica é feita através de dois ecrãs:
  - Visualiza e mostra as alterações da coleção de pessoas.
  - Visualiza uma pessoa. Usado na criação e edição de uma pessoa.
- O utilizador controla a troca de écrãs através dos botões Criar/Editar e Aceitar/Cancelar



Visualizador da coleção Pessoas

- É criada uma subclasse de **VBox** para apresentar a coleção de pessoas.
  - É usado um objeto desta classe como raiz do grafo de cena.
  - A classe recebe no construtor a coleção de pessoas.

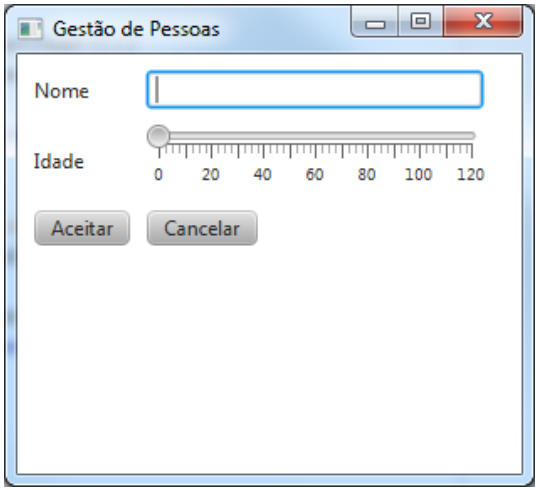



```
public class VisualizadorPessoas extends VBox {

    public VisualizadorPessoas(final Pessoas pessoas) {
        //Criar o rótulo
        Label labelLista = new Label("Pessoas");

        //Criar a lista
        final ObservableList<Pessoa> listaPessoas = FXCollections.observableArrayList(pessoas);
        final ListView<Pessoa> lista = new ListView<>(listaPessoas);

        //Continua...
```

Visualizador de uma Pessoa

- É criada uma subclasse de **GridPane** para apresentar uma **Pessoa**.
  - Esta classe é utilizada na criação e edição de uma pessoa.
  - A classe recebe no construtor a coleção de pessoas.
  - No construtor também é recebido um objeto **Pessoa**. Se for **null** a classe é utilizada na criação de uma pessoa. Se for diferente de **null**, é utilizada na edição da pessoa recebida.

```
public class VisualizadorPessoa extends GridPane {

    public VisualizadorPessoa(final Pessoas pessoas, final Pessoa pessoa) {
        //Criar o rótulo do nome
        Label nomeRotulo = new Label("Nome");
        //Criar a caixa de texto para o nome
        final TextField nomeCampo = new TextField();
        nomeCampo.setPrefWidth(200);
        //Criar o rótulo da idade
        Label idadeRotulo = new Label("Idade");
        //Criar o slider da idade
        final Slider idadeSlider = criarSliderIdade();
        //Inicializar com a informação da pessoa
        if (pessoa != null) {
            nomeCampo.setText(pessoa.getNome());
            idadeSlider.setValue(pessoa.getIdade());
        }
```



Exemplo – Visualizador de Pessoas

- **VisualizadorPessoas**

```
//Botão criar
Button botaoCriar = new Button("Criar...");
botaoCriar.setOnAction (e -> lista.getScene().setRoot(new VisualizadorPessoa(pessoas, null)));
//Botão editar
Button botaoEditar = new Button("Editar...");
botaoEditar.setOnAction (e -> {
    Pessoa pessoa = lista.getSelectionModel().getSelectedItem();
    if (pessoa != null) {
        lista.getScene().setRoot(new VisualizadorPessoa(pessoas, pessoa));
    }
});
```

- **VisualizadorPessoa**

```
//Botão aceitar
Button botaoAceitar = new Button("Aceitar");
botaoAceitar.setOnAction(e -> {
    if (pessoa == null) { //Criar
         pessoas.add(new Pessoa(nomeCampo.getText(), (int) idadeSlider.getValue()));
    } else { //Atualizar
        pessoa.setNome(nomeCampo.getText());
        pessoa.setIdade((int) idadeSlider.getValue());
    }
    nomeCampo.getScene().setRoot(new VisualizadorPessoas(pessoas));
});
//Botão cancelar
        Button botaoCancelar = new Button("Cancelar");
        botaoCancelar.setOnAction(e ->
            nomeCampo.getScene().setRoot(new VisualizadorPessoas(pessoas)));
```

Exemplo – Visualizador de Pessoas

- **VisualizadorPessoas**

```
botaoCriar.setOnAction (
          e -> lista.getScene().setRoot(new VisualizadorPessoa(pessoas, null))
);
botaoEditar.setOnAction (e -> {
          // ...
          lista.getScene().setRoot(new VisualizadorPessoa(pessoas, pessoa));
    }
});
```

- **VisualizadorPessoa**

```
botaoAceitar.setOnAction(e -> {
    // ...
    nomeCampo.getScene().setRoot(new VisualizadorPessoas(pessoas));
});
botaoCancelar.setOnAction(
          e -> nomeCampo.getScene().setRoot(new VisualizadorPessoas(pessoas))
);
```

- A alternância entre ecrãs é feita mudando a cena que é mostrada na janela da aplicação.

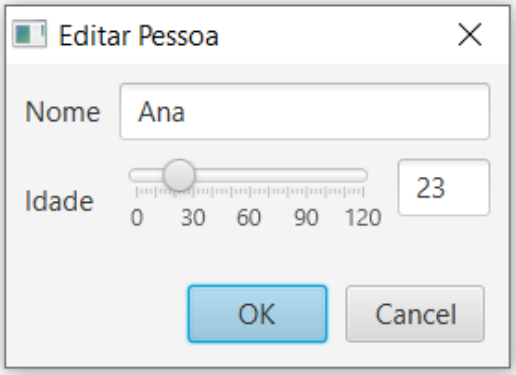
Exemplo – Visualizador de Pessoas

- **Problemas da solução apresentada**
  - Estamos a criar objetos das classes de visualização sempre que se muda a cena.
  - A coleção de pessoas está ser passada no construtor das classes de visualização.
    - Cria-se um dependência destas classes com a coleção de pessoas que apenas é necessária na vizualização das pessoas

- **Solução**
  - Usar um diálogo para editar ou criar uma pessoa
    - Cria-se menos dependência entre as classes.
    - Pode-se reutilizar o diálogo criado noutras partes do programa ou noutros programas.

Novo Diálogo Pessoa

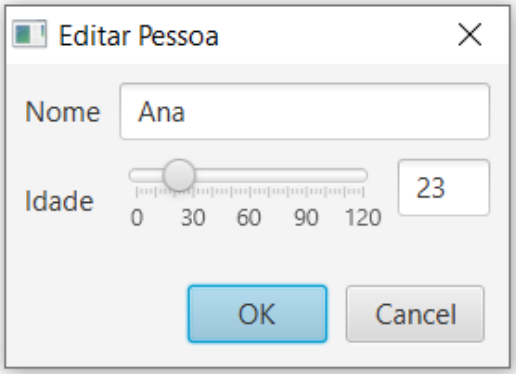
- Cria-se um novo diálogo para a criaçao e edição de uma pessoa.
- O painel **GridPane** usado para a colocação dos controlos passa a ser um atributo da nova classe.
- Este painel passa a ser o conteúdo do diálogo.




```
public class VisualizadorPessoa extends Dialog<Pessoa> {

    public VisualizadorPessoa(final Pessoa pessoa) {
        GridPane gridPessoa = new GridPane();

        //Criar os nós
        Label nomeRotulo = new Label("Nome");
        final TextField nomeCampo = new TextField();
        nomeCampo.setPrefWidth(200);
        Label idadeRotulo = new Label("Idade");
        final VisualizadorIdade idadeSlider = new VisualizadorIdade();

        setTitle("Criar Pessoa");
        if (pessoa != null) {
            setTitle("Editar Pessoa");
            nomeCampo.setText(pessoa.getNome());
            idadeSlider.setValue(pessoa.getIdade());
        }

        //Posicionar os nós
        gridPessoa.setPadding(new Insets(10));
        gridPessoa.setVgap(10);
        gridPessoa.setHgap(10);
        gridPessoa.add(nomeRotulo, 0, 0);
        gridPessoa.add(nomeCampo, 1, 0);
        gridPessoa.add(idadeRotulo, 0, 1);
        gridPessoa.add(idadeSlider, 1, 1);

        getDialogPane().setContent(gridPessoa);
```

Novo Diálogo Pessoa

- Cria-se um conversor para devolver a pessoa criada ou editada.
- Neste exemplo em vez de um objeto que implementa **`Callback<P,R>`** usou-se uma expressão lambda.

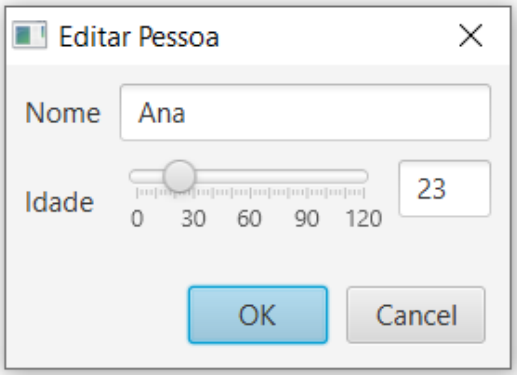



```
        //Botão aceitar
        getDialogPane().getButtonTypes().addAll(ButtonType.OK,
                                                ButtonType.CANCEL);

        setResultConverter(button -> {
            if (button == ButtonType.OK) {
                if (pessoa == null) { //Criar
                    try {
                        return new Pessoa(nomeCampo.getText(),
                                          (int) idadeSlider.getValue());
                    } catch (Exception ex) {
                        return null;
                    }
                } else { //Atualizar
                    pessoa.setNome(nomeCampo.getText());
                    pessoa.setIdade((int) idadeSlider.getValue());
                    return pessoa;
                }
            }
            return null;
        });
    }
}
```

Classe VisualizadorPessoas

- Na classe **VisualizadorPessoas** usa-se o diálogo criado para a criação ou edição de uma pessoa.

```
//Botão criar
Button botaoCriar = new Button("Criar...");
botaoCriar.setOnAction(e -> {
    VisualizadorPessoa dlgPessoa = new VisualizadorPessoa(null);
    dlgPessoa.showAndWait().ifPresent(pessoa -> {
        pessoas.add(pessoa);
        listaPessoas.add(pessoa);
    });
});

//Botão editar
Button botaoEditar = new Button("Editar...");
botaoEditar.setOnAction(e -> {
    Pessoa pessoa = lista.getSelectionModel().getSelectedItem();
    if (pessoa != null) {
        VisualizadorPessoa dlgPessoa = new VisualizadorPessoa(pessoa);
        dlgPessoa.showAndWait();
    }
});
```



JavaFX- Eventos : Exemplo

Leitura Complementar

- Sobre Diálogos foi utilizado e adaptado o seguinte tutorial:
  - https://examples.javacodegeeks.com/desktop-java/javafx/dialog-javafx/javafx-dialog-example/

- Outro tutorial sobre diálogos:
  - https://code.makery.ch/blog/javafx-dialogs-official/

code.makery
Learning how to code.

- Documentação da Oracle
  - https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Dialog.html
  - https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Alert.html
  - https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ChoiceDialog.html
  - https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TextInputDialog.html
  - https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/DialogPane.html
  - https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html