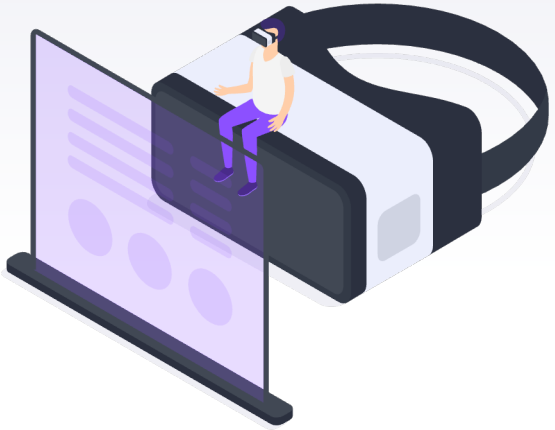
Programação Orientada por Objetos

JavaFX – Eventos e Painéis



Prof. Cédric Grueau
Prof. José Sena Pereira
Departamento de Sistemas e Informática
Escola Superior de Tecnologia de Setúbal
Instituto Politécnico de Setúbal
2022/2023

Sumário

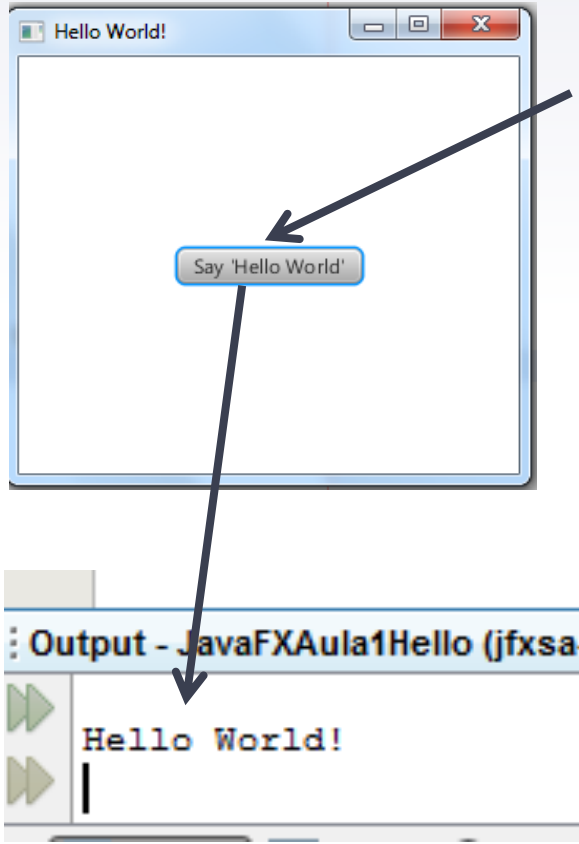
- Programação por eventos
- Classe **`Event`**
  - Ações, Eventos e Resultados das ações
  - Um nó que reaja a diferentes ações
  - Exemplo
- Paineis
  - **`BorderPane`**
  - **`GridPane`**
  - **`HBox`** e **`Vbox`**
  - Painéis compostos por painéis
- Exemplo de Eventos e Paineis

Programação por Eventos

- Eventos e Paineis

JavaFX - Eventos

- Na programação baseada em eventos o código é executado quando ocorre o acontecimento (evento)
  - Por exemplo, podemos querer que o programa execute uma operação em resposta a ações do utilizador tais como:
    - Premir um botão do rato
    - Deslocar o rato
    - Premir um botão
    - Premir uma tecla
  - Ou em resposta a outros acontecimentos como a passagem do tempo (gerida por temporizadores)



1. **Ação**: quando o utilizador prime o botão ...
2. **Evento**: é gerado um evento ...
3. **Executa Código**: se o programador associou código ao evento gerado esse código é executado.

JavaFX – Eventos

- A classe **Event**

```
javafx.event.Event;
```

- Podemos olhar para um evento (acontecimento) como sendo um sinal, representado por um objeto, enviado ao programa, assinalando a ocorrência de uma ação.
  - Geralmente os eventos ocorrem na sequência de ações do utilizador (mas nem sempre)
  - O programa pode reagir ao evento ou ignorá-lo.
  - Um evento é originado por um objeto fonte, que pode ser conhecido através do método:

```
Object getSource()
```

```
public void start(Stage primaryStage) {
  primaryStage.setTitle("Hello World!");
  Button btn = new Button();
  btn.setText("Say 'Hello World'");
  btn.setOnAction(new  EventHandler<ActionEvent>(){
    @Override
    public void handle(ActionEvent event) {
       System.out.println("Hello World!");
    }
  });
  StackPane root = new StackPane();
  root.getChildren().add(btn);
  primaryStage.setScene(new Scene(root, 300, 250));
  primaryStage.show(); }
```

Dentro do método **handle** seria possível aceder ao botão, que gerou o evento, através de:

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

JavaFX - Eventos

- O método **setOnAction**, da classe **Button**, recebe como argumento um objeto **EventHandler<ActionEvent>**:

  public final void setOnAction(EventHandler<ActionEvent> value)

- **EventHandler** é uma interface, genérica, que implementa o método **handle**:

```
public interface EventHandler<T extends Event> extends EventListener {
  public void handle(T event);
}
```

- O método **handle** recebe como argumento um objeto da classe **Event**, ou respetivas subclasses, que contém informação sobre o evento (ex. **getSource()** devolve o objeto onde foi gerado o evento)
- Na abordagem tradicional teríamos que criar uma classe que implementasse a interface **EventHandler<ActionEvent>** (fazendo o override do método **handle**) e passar um objeto dessa classe como argumento para o **setOnAction**.

JavaFX - Eventos

```
import javafx.event.ActionEvent;
import javafx.event.EventHandler;

public class ButtonAction implements EventHandler<ActionEvent> {
  private String message;
  public ButtonAction(String message) {
    this.message = message;
  }
  public String getMessage() {
    return message;
  }
  public void setMessage(String message) {
    this.message = message;
  }
  @Override
  public void handle(ActionEvent event) {
    System.out.println(message);
  }
}
```

```
//...
btn.setOnAction(new ButtonAction("Hello World!"));
//...
```

JavaFX – Eventos (classe anónima)

- Para não se criar uma classe específica para cada ação associada a cada evento, é possível recorrer à criação de classes anónimas (estender uma classe já existente ou implementar uma interface):
  - http://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html
- Com a utilização de uma classe anónima é possível declarar e instanciar uma classe num único passo.
- Na definição de uma classe anónima utiliza-se o operador **new**, seguido do nome da interface (ou da classe que se pretenda estender), seguido de eventuais argumentos do construtor (apenas () quando se trata de uma implementação de interface), seguido do corpo da classe:

```
new EventHandler<ActionEvent>(){
  @Override
  public void handle(ActionEvent event) {
    System.out.println("Hello World!");
  }
}
```

JavaFX - Eventos

- Como associar ações do utilizador a nós do grafo de cena?

1. Definir o evento a capturar na respetiva propriedade do nó em causa. Exemplo para o nó btn:

   `btn.setOnAction(eventHandler)`

2. Associar a essa propriedade o (novo) tratador de eventos (definido em função do tipo de evento que pretendemos capturar):

   `EventHandler<EventoACapturar>()`

3. Redefinir o método **`handle`** do tratador de eventos passado, colocando nesse método o código a ser executado

```
public void handle (EventoACapturar event)
```

```
public void start(Stage primaryStage) {
  primaryStage.setTitle("Hello World!");
  Button btn = new Button();
  btn.setText("Say 'Hello World'");
  btn.setOnAction( new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
       System.out.println("Hello World!");
       }
    });
       StackPane root = new StackPane();
       root.getChildren().add(btn);
       primaryStage.setScene(new Scene(root, 300, 250));
      primaryStage.show();
}
```

JavaFX – Eventos (expressões lambda)

- O uso de classes anónimas veio simplificar bastante a indicação do código a executar (método **`handle`**) quando o evento é gerado.
- No Java 8 é possível simplificar a utilização de implementações de interfaces anónimas que apenas possuem um método (as chamadas interfaces funcionais), através da utilização do conceito de **expressão lambda**: http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
- Uma **expressão lambda** (o seu nome advém do estudo "λ-calculus" realizado pelo matemático Alonzo Church na década de 1930) permite representar um método indicando apenas o(s) seu(s) parâmetro(s) e o seu corpo (omitindo o seu nome).
- Assim, o anterior método handle poderia ser representado por:

```
(ActionEvent event) -> { System.out.println("Hello World!"); }
```

JavaFX – Eventos (expressões lambda)

- Nas expressões lambdas pode ser omitido o tipo do parâmetro (o compilador consegue determiná-lo em função da interface onde o método foi definido), assim `(ActionEvent event)` poderá apenas ser representado por (event).
- Quando o método apenas possui um parâmetro é, ainda, possível eliminar os parêntesis (estes são importantes quando existem diversos parâmetros, que serão separados por virgulas). Nesta situação é comum utilizar apenas uma letra para designar o parâmetro:

```
e -> { System.out.println("Hello World!"); }
```

- Analogamente, se o corpo apenas possuir uma instrução, as chavetas podem ser omitidas, bem como o ponto e vírgula terminador:

```
e -> System.out.println("Hello World!")
```

- Se a instrução for um **`return`**, este poderá ser omitido. Um método da forma

```
int sum (int a, int b)  {
 return a + b;
}
```

poderia ser representado por `(a, b) -> a + b`

JavaFX – Eventos

- Em resumo, em Java 8, a associação do método do evento pode ser feita:
  - Através da criação de uma classe específica que implementa a interface do evento:

```
public class ActionButton implements EventHandler<ActionEvent> {
  private String message;
  public ActionButton(String message) { this.message = message; }
  public String getMessage() {return message; }
  public void setMessage(String message) {this.message = message;}
  @Override
   public void handle(ActionEvent event) { System.out.println(message); }
}
```

```
btn.setOnAction(new ActionButton("Hello World!"));
```

  - Através da criação de uma classe anónima diretamente na associação do evento:

```
btn.setOnAction(new EventHandler<ActionEvent>(){
  @Override
  public void handle(ActionEvent event){
    System.out.println("Hello World!"); } });
```

  - Através da utilização de uma expressão lambda:

```
btn.setOnAction(e -> System.out.println("Hello World!"));
```

JavaFX - Eventos

- Associar múltiplas ações do utilizador a um nó do grafo de cena

1. Definir o evento a capturar nas respetivas propriedades do nó :

```
btn.setOnMouseClicked(
  new EventHandler <MouseEvent>() ... )

btn.setOnMouseExited(
  new EventHandler <MouseEvent>() ... )
```

2. Para cada uma das propriedades proceder com uma das alternativas do slide anterior

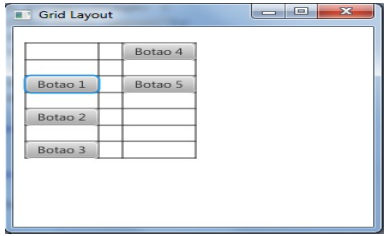
```
public void start(Stage primaryStage) {
  primaryStage.setTitle("Hello World!");
  Button btn = new Button();
  btn.setText("Say 'Hello World'");

  btn.setOnMouseClicked(new EventHandler<MouseEvent>(){
    @Override
    public void handle(MouseEvent event) {
      System.out.println("Premi o botão!");
    }
  });

  btn.setOnMouseExited(new EventHandler<MouseEvent>(){
    @Override
    public void handle(MouseEvent event) {
      System.out.println("Saí da área do botão");
    }
  });

  StackPane root = new StackPane();
  root.getChildren().add(btn);
  primaryStage.setScene(new Scene(root, 300, 250));
  primaryStage.show();
}
```

Paineis

- Eventos e Paineis

JavaFX – Gestores de Painéis

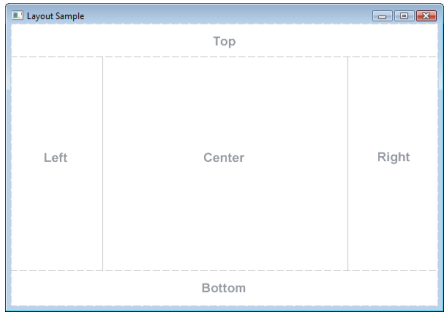
- Até ao momento, temos inserido os nós folha diretamente nas janelas ou no painel **`StackPane`**, sem nos preocuparmos em definir as suas posições (absolutas ou relativas).
- Em aplicações reais isso não é aceitável e temos de especificar as posições dos componentes por forma a produzir uma interface gráfica do utilizador (GUI) facilmente compreensível.
- Duas abordagens para inserir componentes gráficos:
  - Rígida: especificar onde os componentes deverão ser colocados, através de coordenadas absolutas
    - Usada apenas na prototipagem rápida de aplicações
    - A apresentação final fica dependente da plataforma
    - Não suporta adequadamente o redimensionamento do container (janela ou painel)
  - Flexível: utilização de painéis
    - Adapta a posição dos componentes gráficos a possíveis redimensionamentos da janela
    - Apresentação final adapta-se à plataforma utilizada



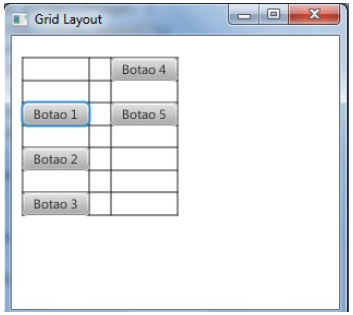
JavaFX – Gestores de Painéis

- O package **`layout`** - **`javafx.scene.layout.*;`**
  - O JavaFX disponibiliza no package **`layout`** vários modelos pré-definidos para disposição de componentes.
  - Os componentes que estabelecem e gerem os diferentes modelos de disposição de elementos são denominados painéis (**`Pane`**).
  - Os elementos gráficos são adicionados a esses painéis e são dispostos de acordo com o modelo definido por esse painel.
  - Vamos ver apenas quatro dos mais comuns.

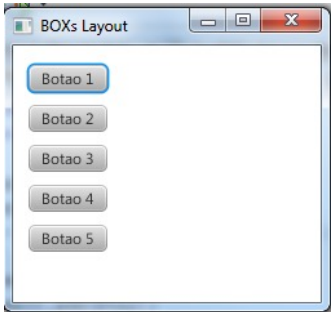
- **BorderPane**



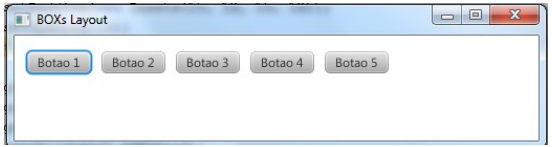
- **GridPane**



- **VBox**



- **HBox**

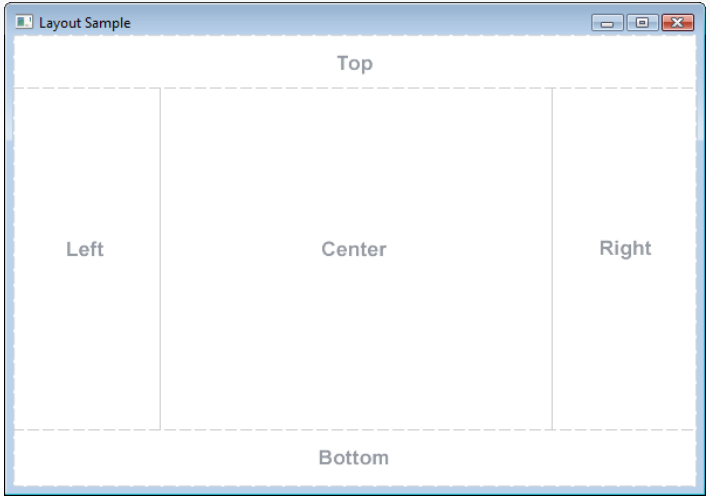


JavaFX – Gestores de Painéis

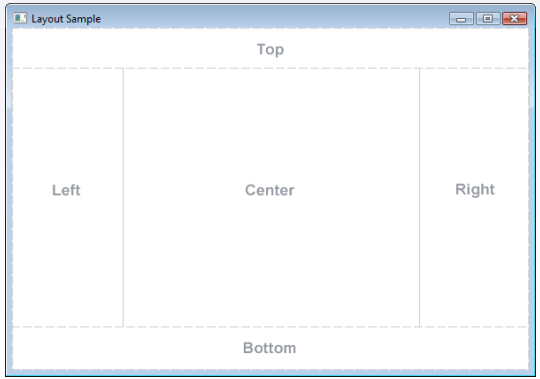
- **BorderPane**

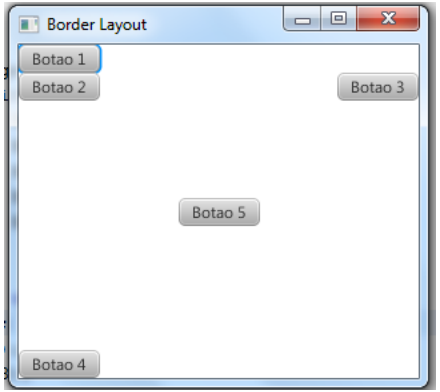
```
javafx.scene.layout.BorderPane;
```

- O **BorderPane** possui cinco zonas para colocação de um elemento gráfico.
  - O elemento é colocado numa posição dentro da zona definida pelo próprio **BorderPane**.
  - Cada zona só pode conter um componente, mas este pode ser outro painel.



JavaFX – Gestores de Painéis





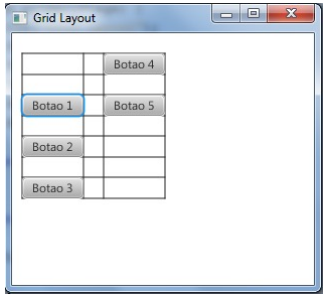

```
import javafx.scene.layout.BorderPane;

public class BorderPaneExample extends Application {

 @Override
 public void start(Stage primaryStage) {

  Button btn1 = new Button("Botao 1");
  Button btn2 = new Button("Botao 2");
  Button btn3 = new Button("Botao 3");
  Button btn4 = new Button("Botao 4");
  Button btn5 = new Button("Botao 5");
  BorderPane root = new BorderPane();
  root.setTop(btn1);
  root.setLeft(btn2);
  root.setRight(btn3);
  root.setBottom(btn4);
  root.setCenter(btn5);
  primaryStage.setTitle("Border Layout");
  primaryStage.setScene(new Scene(root, 300, 250));
  primaryStage.show();
}

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

JavaFX – Gestores de Painéis

- **GridPane**

  javafx.scene.layout.GridPane;

- Um **GridPane** define uma grelha (tabela).
  - Os elementos gráficos são adicionados a uma célula especifica da tabela.
  - É possível definir o espaçamento entre células.
  - É possível determinar se as linhas da grelha são ou não visíveis.



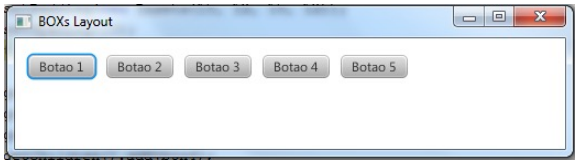

```
public void start(Stage primaryStage) {
  primaryStage.setTitle("Grid Layout");
  Button btn1 = new Button("Botao 1");
  Button btn2 = new Button("Botao 2");
  Button btn3 = new Button("Botao 3");
  Button btn4 = new Button("Botao 4");
  Button btn5 = new Button("Botao 5");
  GridPane root = new GridPane();
  root.setHgap(20);
  root.setVgap(20);
  root.setPadding(new Insets(20, 10, 20, 10));
  root.setGridLinesVisible(true);
  root.add(btn1,0,1);
  root.add(btn2,0,2);
  root.add(btn3,0,3);
  root.add(btn4,1,0);
  root.add(btn5,1,1);
  primaryStage.setScene(new Scene(root,300,250));
  primaryStage.show();
}
```

JavaFX – Gestores de Painéis

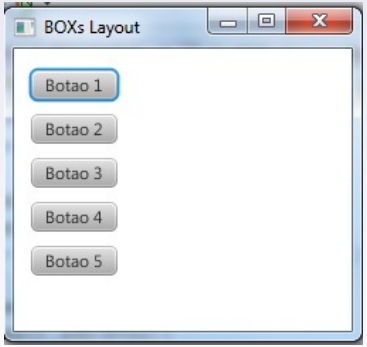
HBox e VBox

```
javafx.scene.layout.HBox;
javafx.scene.layout.VBox;
```

- Um painel **HBox** distribui os componentes na horizontal



Um painel **VBox** distribui os componentes na vertical




```
public void start(Stage primaryStage) {
  primaryStage.setTitle("BOXs Layout");
  Button btn1 = new Button("Botao 1");
  Button btn2 = new Button("Botao 2");
  Button btn3 = new Button("Botao 3");
  Button btn4 = new Button("Botao 4");
  Button btn5 = new Button("Botao 5");
  HBox root = new HBox();
  root.setPadding(new Insets(15,12,15,12));
  root.setSpacing(10);
  root.getChildren().addAll(btn1, btn2, btn3, btn4, btn5);
  primaryStage.setScene(new Scene(root, 500, 250));
  primaryStage.show();
}
```

JavaFX – Gestores de Painéis

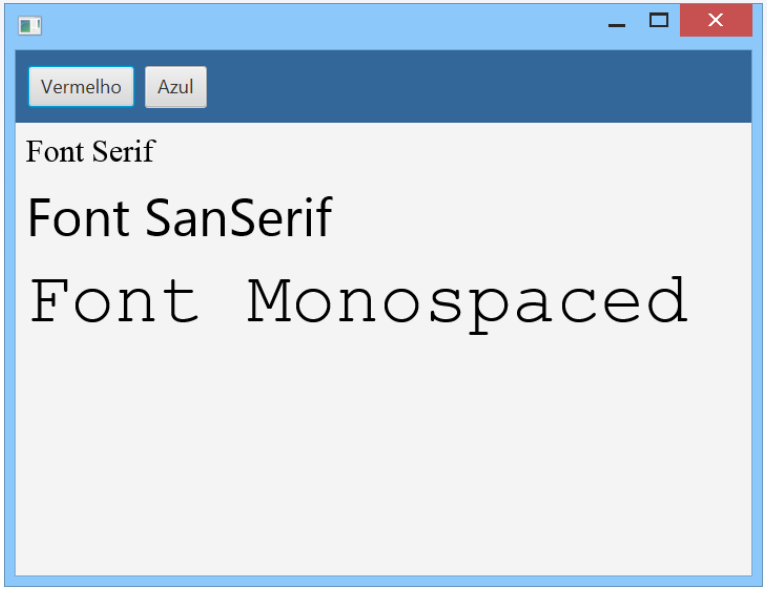
- Painéis Compostos
  - Objetivo: criar um **BorderPane** com um **HBox** no topo e um contentor **VBox** no centro.
  - Criar painel do tipo **BorderPane**
  - Criar e adicionar um painel do tipo **Hbox**
  - Criar e adicionar um painel do tipo **Vbox**
  - Definimos o **HBox** e adicionamos botões ao **Hbox**
  - Definimos um **VBox** e adicionamos texto.

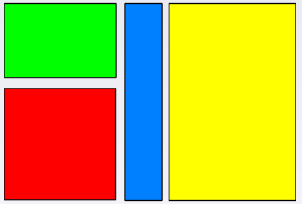
```
public void start(Stage primaryStage) {
  BorderPane root = new BorderPane();
  HBox hbox = addBotoes();
  root.setTop(hbox);
  VBox vbox = addTextos();
  root.setCenter(vbox);
  Scene scene = new Scene(root, 700, 500);
  primaryStage.setScene(scene);
  primaryStage.show();
}
private HBox addBotoes() {
  HBox hbox = new HBox();
  hbox.setPadding(new Insets(15,12,15,12));
  hbox.setSpacing(10);
  hbox.setStyle("-fx-background-color: #336699;");
  addButtons(hbox);
  return hbox;
}
private VBox addTextos() {
  VBox vbox = new VBox();
  vbox.setPadding(new Insets(10));
  vbox.setSpacing(10);
  addTexts(vbox);
  return vbox;
}
```

JavaFX – Gestores de Painéis

- Painéis Compostos
  - Objetivo: criar um **`BorderPane`** com um **`HBox`** no topo e um **`VBox`** no centro.
  - Estrutura de nós do grafo de cena:

1. Root (**`BorderPane`**)
1.1 Top – **`HBox`** – (área de botões)
    1. Botão Vermelho
    2. Botão Azul
1.2 Center - **`VBox`** – (área do texto)
    1. Text ("Font Serif")
    2. Text ("Font SanSerif")
    3. Text ("Font Monospaced")



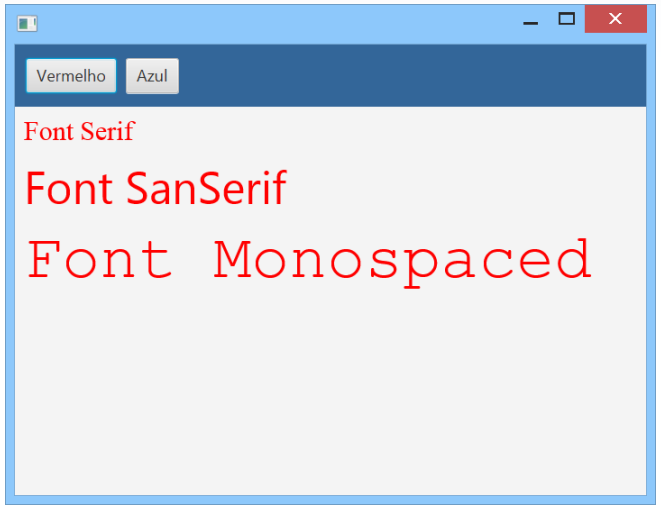


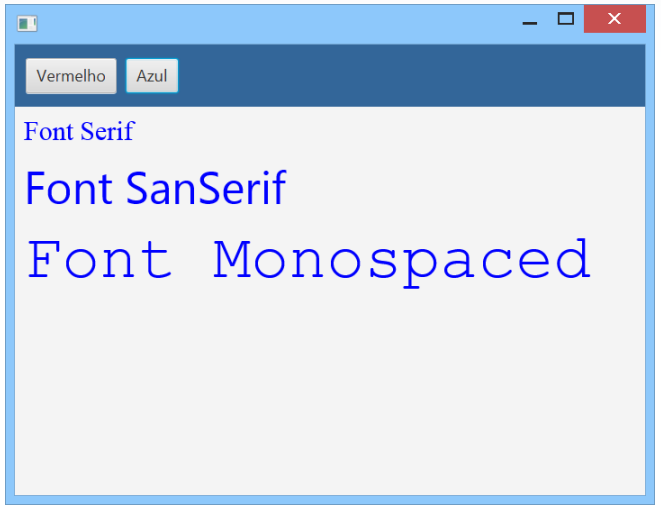
Exemplo de Eventos e Paineis

- Eventos e Paineis

JavaFX- Eventos : Exemplo

- **Objetivo:** Construir uma aplicação com a interface gráfica onde os textos mudam de vermelho para azul e vice-versa em função do botão acionado.





JavaFX- Eventos : Exemplo

Objetivo

Criar a estrutura principal da aplicação.

1. Criar um BorderPane
2. Adicionar os Botões
3. Adicionar os Textos
4. Criar a Cena
5. Associar a Cena ao Stage
6. Mostrar

```
public void start(Stage primaryStage) {
  BorderPane root = new BorderPane();
  HBox hbox = addButtons();
  root.setTop(hbox);
  VBox vbox = addTexts();
  root.setCenter(vbox);
  Scene scene = new Scene(root, 700, 500);
  primaryStage.setScene(scene);
  primaryStage.show();
}
```

JavaFX- Eventos : Exemplo

Objetivo

Criar o painel com os botões (será colocado no topo).

1. Criar um painel **HBox**, definindo as suas características
2. Adicionar os Botões como filhos do painel
3. Na criação do botão é definido o seu rótulo e a ação a executar em caso de seleção (acesso a "variável externa")

```
private HBox addButtons() {
  HBox hbox = new HBox();
  hbox.setPadding(new Insets(15, 12, 15, 12));
  hbox.setSpacing(10);
  hbox.setStyle("-fx-background-color: #336699;");
  addButtons(hbox);
  return hbox;
}

private void addButtons(HBox hbox) {
  hbox.getChildren().add(createButton("Vermelho", Color.RED));
  hbox.getChildren().add(createButton("Azul", Color.BLUE));
}

private Button createButton(String text, final Color color) {
  Button btn = new Button(text);
  btn.setOnAction(e -> changeColor(color, e));
  return btn;
}
```

JavaFX- Eventos : Exemplo

Objetivo

Criar o painel com os textos (será colocado no centro).

1. Criar um painel **VBox**, definindo as suas características
2. Adicionar os Textos como filhos do painel
3. Na criação do texto é indicado o seu conteúdo, o nome da fonte e o respetivo tamanho

```
private VBox addTexts() {
  VBox vbox = new VBox();
  vbox.setPadding(new Insets(10));
  vbox.setSpacing(10);
  addTexts(vbox);
  return vbox;
  }

private void addTexts(VBox vbox) {
  vbox.getChildren().addAll(
     createText("Font Serif","Serif",30),
     createText("Font SanSerif","SanSerif",50),
     createText("Font Monospaced","Monospaced",70)
  );
}

private Text createText(String sentence,
                        String fonteName,
                        int size) {
  Text text = new Text(sentence);
  Font font = Font.font(fonteName, size);
  text.setFont(font);
  return text;
}
```

JavaFX- Eventos : Exemplo

Objetivo: Ação a executar pelos botões.

1. Obter o botão que gerou o evento (**`getSource()`**)
2. Seguir a hierarquia para obter o painel principal
3. Obter o painel (**`VBox`**) que tem os textos
4. Percorrer todos os nós do painel e, caso sejam textos, modificar a sua cor

```
private void changeColor(Color color, ActionEvent event) {
  Button bnt = (Button)event.getSource();
  BorderPane root = (BorderPane)bnt.getParent().getParent();
  VBox vbox = (VBox)root.getCenter();
  Text text;
  for (Node no : vbox.getChildrenUnmodifiable()) {
    if (no instanceof Text) {
      text = (Text)no;
      text.setFill(color);
    }
  }
}
```

JavaFX- Eventos : Exemplo

Resumindo

- Eventos
  - A Programação baseada em eventos permite interagir com uma interface gráfica
  - A Classe **`Event`**
    - Uma ação do utilizador sobre um componente do GUI
    - Faz com que esse objeto gere um evento
    - Evento esse que, ao ser apanhado pelo "handler" apropriado, despoleta a execução do código desse "handler"
    - Para que um qualquer componente do GUI (nó) reaja a diferentes ações do utilizador basta criar um "handler" para cada uma dessas ações no respetivo componente.

- Painéis (**`Pane`**)
  - **`BorderPane`** (cinco zonas: top, bottom, left, right, center)
  - **`GridPane`** (uma grelha)
  - **`HBox e VBox`** (dispõem os componentes horizontal e verticalmente)
  - Painéis compostos por painéis (podemos sempre inserir painéis em painéis)

Leitura Complementar

- Chapter 3 – Lambdas and Properties Pgs 61 a 73
- Chapter 4 – Layouts and UI Controls Pgs 91 a 101

- Documentação:
  - http://docs.oracle.com/javase/8/javafx/api/toc.htm
- Eventos:
  - http://docs.oracle.com/javase/8/javafx/api/javafx/event/package-summary.html
  - http://docs.oracle.com/javase/8/javafx/events-tutorial/events.htm
- Painéis
  - http://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/package-frame.html
  - http://docs.oracle.com/javase/8/javafx/layout-tutorial/index.html