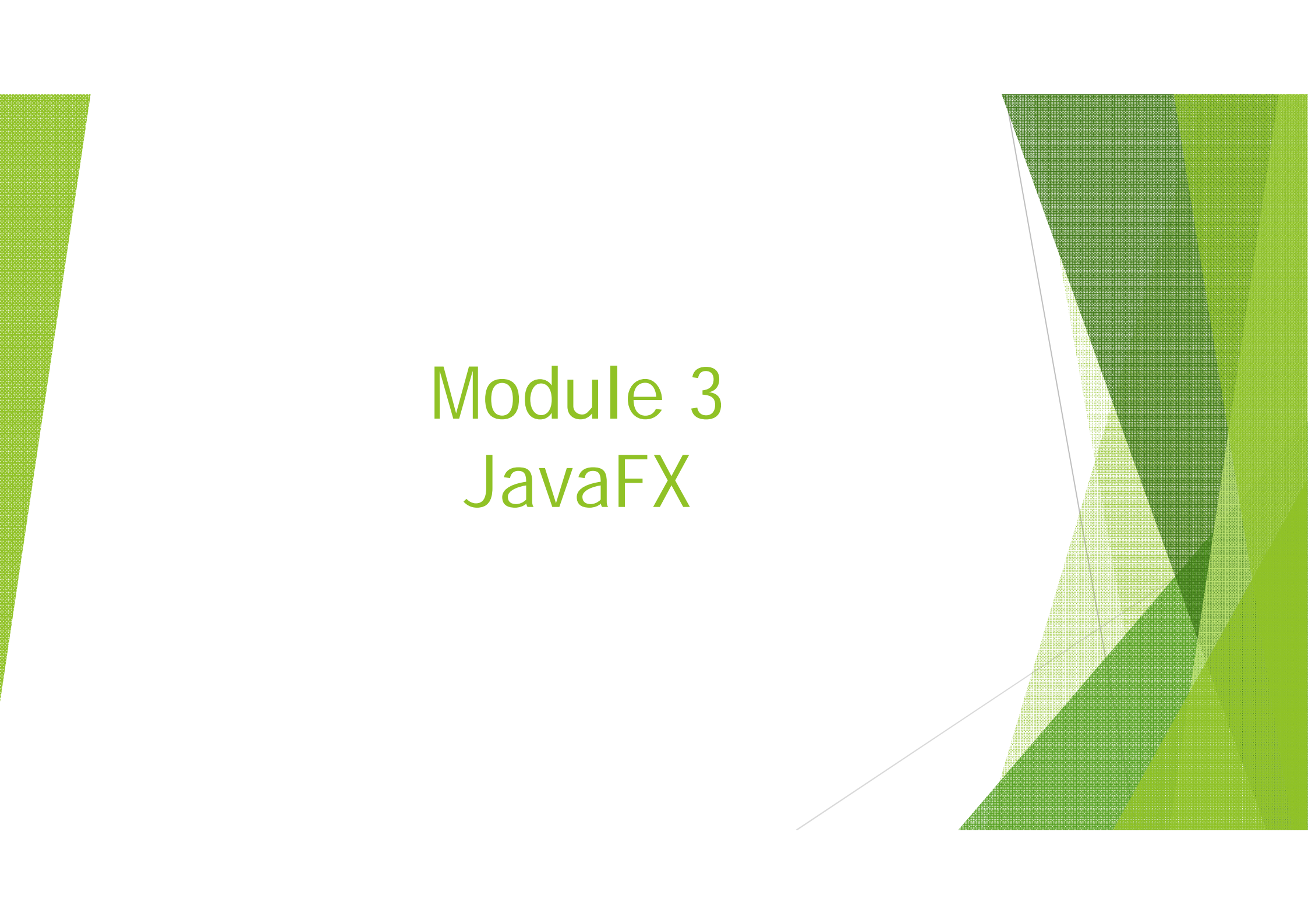


# PMCA502L Java Programming

Thilagavathi M, AP(Sr.), SCORE



# Module 3

## JavaFX

# Event handling in JavaFX

## ► Types of Events

In general, the events are mainly classified into the following two types.

### 1. Foreground Events

Foreground events occur due to the direct interaction of the user with the GUI of the application. Such as clicking the button, pressing a key, selecting an item from the list, scrolling the page, etc.

### 2. Background Events

Background events doesn't require the user's interaction with the application. These events mainly occur because of operating system interrupts, failure, operation completion, etc.

# Events in JAVAFX

- ▶ JavaFX provides a wide variety of events. Some of them are listed below.
  - ▶ **Action Event**
  - ▶ **Mouse Event**
  - ▶ **Key Event**
  - ▶ **Drag Event**
  - ▶ **Window Event**
  - ▶ **Scroll Event**

# Events in JAVAFX

- ▶ JavaFX provides a wide variety of events. Some of them are listed below.
  - ▶ **Action Event** - This is an input event that occurs when a user clicks on a button or enters text in a text field or chooses an item from menu, etc. It is represented by the class named **ActionEvent**.
  - ▶ **Mouse Event** - This is an input event that occurs when a mouse is clicked. It is represented by the class named **MouseEvent**. It includes actions like mouse clicked, mouse pressed, mouse released, mouse moved, mouse entered target, mouse exited target, etc.
  - ▶ **Key Event** - This is an input event that indicates the key stroke occurred on a node. It is represented by the class named **KeyEvent**. This event includes actions like key pressed, key released and key typed.
  - ▶ **Drag Event** - This is an input event which occurs when the mouse is dragged. It is represented by the class named **DragEvent**. It includes actions like drag entered, drag dropped, drag entered target, drag exited target, drag over, etc.
  - ▶ **Window Event** - This is an event related to window showing/hiding actions. It is represented by the class named **WindowEvent**. It includes actions like window hiding, window shown, etc.

# Processing Events in JavaFX

- ▶ The class **javafx.event.Event** contains all the subclasses representing the types of Events that can be generated in JavaFX. Any event is an instance of the class **Event** or any of its subclasses.
- ▶ There are various events in JavaFX i.e. **MouseEvent**, **KeyEvent**, **ScrollEvent**, **DragEvent**, etc.
- ▶ We can also define our own event by inheriting the class **javafx.event.Event**

# The properties of an event

	Property	Description
1	Source	It represents source of the event i.e. the origin which is responsible to generate/trigger the event.
2	Target	It is the node on which the event is generated. It remains unchanged for the generated event. It is the instance of any of the class that implements the EventTarget interface.
3	Event Type	<p>It is the type of the event that is being generated. It is basically the instance of EventType class. It is hierarchical.</p> <p>The instance of EventType class is further classified into various type of events for example KeyEvent class contains KEY_PRESSED, KEY_RELEASED, and KEY_TYPED types.</p>

# Event handlers

## ► Event Handlers and Filters

- Event Handlers and filters contains application logic to process an event.
- A node can be registered to more than one Event Filter.
- The interface **javafx.event.EventHandler** must be implemented by all the event handlers and filters.



# Adding Event Handlers

- ▶ The `setOn_____()` methods can also be used to register an event handler.
- ▶ The `setOn_____()` methods take the appropriate event handler reference as an argument.
- ▶ For example to register
  - Action Event we can use `setOnAction()`
  - Mouse Event we can use `setOnMousePressed()`
  - Key Event we can use `setOnKeyTyped()`

# Adding Event Handlers

- ▶ EventHandler is an interface that has an abstract method  
`public void handle(EventType e);`
- ▶ The `handle()` method takes one argument that indicate the reference of the object of a specified event type (example: `ActionEvent`, `MouseEvent`, etc.,).
- ▶ The statements to handle the event should be placed within the `handle()` method.

# Ways to create listener object

1) Create a class that implements the EventHandler interface and override the handle() method.

- Pass the object of this class as an argument to addEventHandler() or setOn\_\_() methods.

```
class EHandler implements EventHandler<ActionEvent>
{
    public void handle(ActionEvent e)
    {
        System.out.println("Button is pressed");
    }
}
```

Assume an event handler should be registered for a button object.

```
Button b1 = new Button("Click");
b1.setOnAction(new EHandler());
```

## Ways to create listener object

2) Defining the class and creating the object can be done together.

Assume an event handler should be registered for a button object.

```
Button b1 = new Button("Click");
```

```
EventHandler<ActionEvent> event = new EventHandler<ActionEvent>()
```

```
{  
    public void handle(ActionEvent e)  
    {  
        System.out.println("Button is pressed");  
    }  
}; //; is mandatory as the class definition is enclosed in an executable statement.  
b1.setAction(event);
```

## Ways to create listener object

3) The definition of the class can be done within the setOn\_\_\_\_() method (Method local inner class).

Assume an event handler should be registered for a button object.

```
Button b1 = new Button("Click");  
b1.setAction( new EventHandler<ActionEvent>()  
{  
    public void handle(ActionEvent e)  
    {  
        System.out.println("Button is pressed");  
    }  
});
```

# Adding Event Handlers

- ▶ To add an event handler to a node, `addEventHandler()` method of the Node class can also be used to register the handler.
- ▶ `addEventHandler()` method takes two arguments - the event type and the reference of the event handler.

//Creating the mouse event handler

```
EventHandler<MouseEvent> eventHandler = new EventHandler<MouseEvent>()
```

```
{  
    public void handle(MouseEvent e)  
    {  
        System.out.println("Hello World");  
        circle.setFill(Color.BLUE);  
    }  
}
```

```
};
```

//Adding the event handler

```
circle.addEventHandler(MouseEvent.MOUSE_CLICKED, eventHandler);
```

//The `handle()` method in the event handler is called when user clicks on the circle

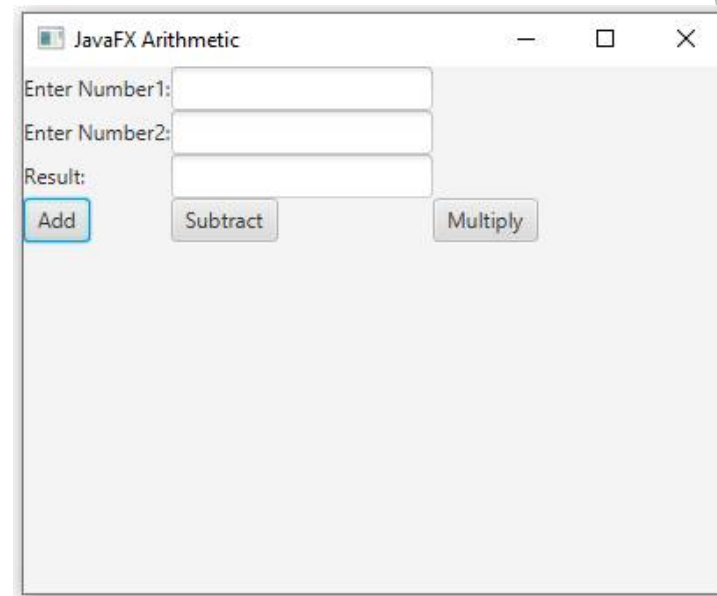
# JavaFX TextField

```
TextField textField = new TextField();  
Button btn = new Button("Click to get text");  
btn.setOnAction(new EventHandler<ActionEvent>() {  
    public void handle(ActionEvent event) {  
        System.out.println(textField.getText());  
    }  
});  
HBox hbox = new HBox(textField, button);
```



# Example - Arithmetic Operation using JavaFX

```
//import the required packages
public class JavaFXArithmetic extends Application {
    @Override
    public void start(Stage primaryStage) {
        Label l1 = new Label("Enter Number1:");
        TextField t1 = new TextField();
        Label l2 = new Label("Enter Number2:");
        TextField t2 = new TextField();
        Label l3 = new Label("Result:");
        TextField t3 = new TextField();
        t3.setEditable(false);
        Button add = new Button("Add");
        Button sub = new Button("Subtract");
        Button mul = new Button("Multiply");
```





```
EventHandler<ActionEvent> event=new EventHandler<ActionEvent>() {  
    @Override  
    public void handle(ActionEvent e) {  
        int n1 = Integer.parseInt(t1.getText());  
        int n2 = Integer.parseInt(t2.getText());  
        int result=0;  
        if(((Button)e.getSource()).getText().equals("Add"))  
        {  
            result=n1+n2;  
            t3.setText(result+"");  
        }  
        else if(((Button)e.getSource()).getText().equals("Subtract"))  
        {  
            result=n1-n2;  
            t3.setText(result+"");  
        }  
        else if(((Button)e.getSource()).getText().equals("Multiply"))  
        {  
            result=n1*n2;  
            t3.setText(result+"");  
        }  
    }  
};
```

```
add.setOnAction(event);
sub.setOnAction(event);
mul.setOnAction(event);
GridPane root = new GridPane();
root.addRow(0,l1,t1);
root.addRow(1,l2,t2);
root.addRow(2,l3,t3);
root.addRow(3,add,sub,mul);
Scene scene = new Scene(root, 600, 600);

primaryStage.setTitle("JavaFX Arithmetic");
primaryStage.setScene(scene);
primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}
}
```

# JavaFx menus

## ► JavaFX Menu

- JavaFX provides a Menu class to implement menus. Menu is the main component of any application. In JavaFX, **javafx.scene.control.Menu** class provides all the methods to deal with menus. This class needs to be instantiated to create a Menu.
- The following sample of code shows the implementation of JavaFX menu.
- `MenuBar menubar = new MenuBar(); //creating MenuBar`
- `Menu MenuName = new Menu("Menu Name"); //creating Menu`
- `MenuItem MenuItem1 = new MenuItem("Menu Item 1 Name"); //creating Menu Item`
- `MenuName.getItems().add(MenuItem1); //adding Menu Item to the Menu`
- `menubar.getMenus().add(MenuName); //adding Menu to the MenuBar`

## Example

```
MenuBar menubar = new MenuBar();  
Menu FileMenu = new Menu("File");  
MenuItem filemenu1=new MenuItem("New      Ctrl+N");  
MenuItem filemenu2=new MenuItem("Open    Ctrl+O");  
MenuItem filemenu3=new MenuItem("Save    Ctrl+S");  
Menu EditMenu=new Menu("Edit");  
MenuItem EditMenu1=new MenuItem("Cut");  
MenuItem EditMenu2=new MenuItem("Copy");  
MenuItem EditMenu3=new MenuItem("Paste");  
EditMenu.getItems().addAll(EditMenu1,EditMenu2,EditMenu3);  
FileMenu.getItems().addAll(filemenu1,filemenu2,filemenu3);  
menubar.getMenus().addAll(FileMenu,EditMenu);
```

## Example

```
//import the required packages
public class JavaFXMenu extends Application {
    public void start(Stage primaryStage) {
        MenuBar menubar = new MenuBar();
        Menu FileMenu = new Menu("File");
        MenuItem filemenu1=new MenuItem("New      Ctrl+N");
        MenuItem filemenu2=new MenuItem("Open      Ctrl+O");
        MenuItem filemenu3=new MenuItem("Save      Ctrl+S");

        Menu EditMenu=new Menu("Edit");
        MenuItem editmenu1=new MenuItem("Cut");
        MenuItem editmenu2=new MenuItem("Copy");
        MenuItem editmenu3=new MenuItem("Paste");

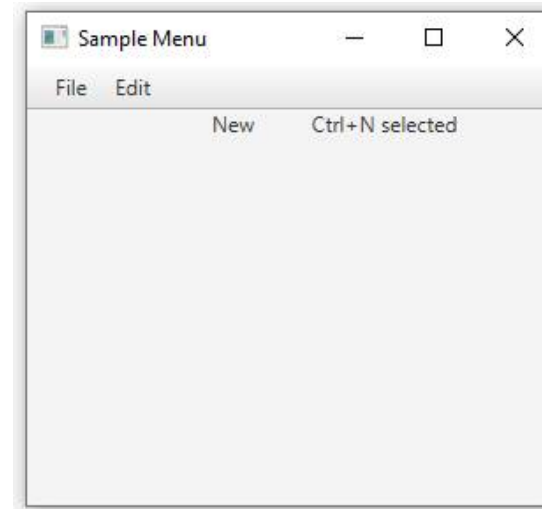
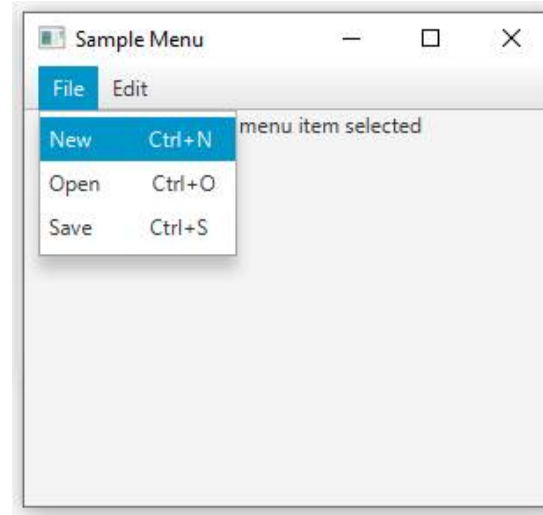
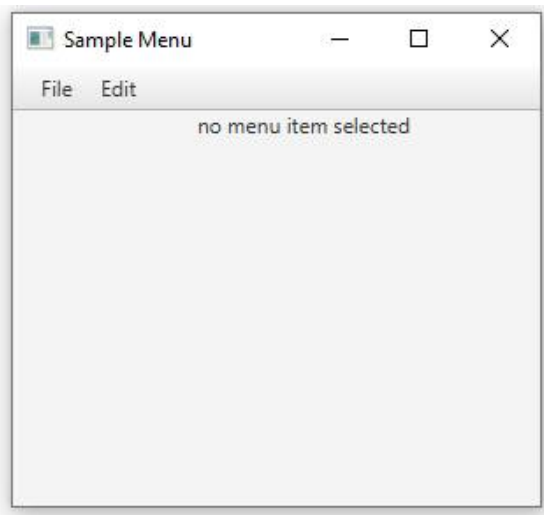
        EditMenu.getItems().addAll(editmenu1,editmenu2,editmenu3);
        FileMenu.getItems().addAll(filemenu1,filemenu2,filemenu3);

        menubar.getMenus().addAll(FileMenu,EditMenu);

        // label to display events
        Label l1 = new Label("\t\t\t\t\t" + "no menu item selected");
```

```
// create events for menu items - action event
EventHandler<ActionEvent> event = new EventHandler<ActionEvent>() {
    public void handle(ActionEvent e)    {
        l1.setText("\t\t\t\t" + ((MenuItem)e.getSource()).getText() + " selected");
    }
};
// add event
filemenu1.setOnAction(event);
filemenu2.setOnAction(event);
filemenu3.setOnAction(event);
editmenu1.setOnAction(event);
editmenu2.setOnAction(event);
editmenu3.setOnAction(event);
VBox vb = new VBox(menuubar,l1);
Scene scene = new Scene(vb, 300, 250);
primaryStage.setTitle("Sample Menu");
primaryStage.setScene(scene);
primaryStage.show();
}
public static void main(String[] args) {
    launch(args);
}
}
```

# Output





# JavaFX PasswordField

- ▶ Package: `javafx.scene.control.PasswordField`
- ▶ Constructors
  - ▶ `PasswordField p1 = new PasswordField();`
- ▶ Methods

```
p1.setPromptText("Your password");
```

```
String str = t1.getText();
```



# JavaFX PasswordField

```
PasswordField passwordField = new PasswordField();

Button button = new Button("Click to get password");

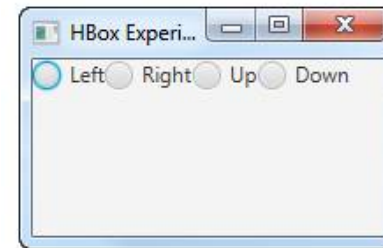
button.setOnAction(action -> {
    System.out.println(passwordField.getText());
});

HBox hbox = new HBox(passwordField, button);

Scene scene = new Scene(hbox, 200, 100);
primaryStage.setScene(scene);
primaryStage.show();
```

# JavaFX RadioButton

```
RadioButton radioButton1 = new RadioButton("Left");  
RadioButton radioButton2 = new RadioButton("Right");  
RadioButton radioButton3 = new RadioButton("Up");  
RadioButton radioButton4 = new RadioButton("Down");  
ToggleGroup toggleGroup = new ToggleGroup();  
radioButton1.setToggleGroup(toggleGroup);  
radioButton2.setToggleGroup(toggleGroup);  
radioButton3.setToggleGroup(toggleGroup);  
radioButton4.setToggleGroup(toggleGroup);  
HBox hbox = new HBox(radioButton1, radioButton2, radioButton3, radioButton4);
```



# JavaFX RadioButton

## Reading Selected State of a ToggleGroup

You can read which RadioButton of a ToggleGroup is selected using the `getSelectedToggle()` method,

```
RadioButton selectedRadioButton =  
    (RadioButton) toggleGroup.getSelectedToggle();
```

If no RadioButton is selected the `getSelectedToggle()` method returns `null`.

# Other Methods of RadioButton

- ▶ **isSelected()**: This method returns whether the radio button is selected or not.
- ▶ **setSelected(boolean x)**: This method sets whether the radio button is selected or not.
- ▶ **setToggleGroup(ToggleGroup g)**: This method sets the toggle group for the radio button.
- ▶ **getText()**: It returns the text label for the radio button.

# JavaFX CheckBox

```
CheckBox checkBox1 = new CheckBox("Green");  
HBox hbox = new HBox(checkBox1);
```

## Reading Selected State

```
boolean isSelected = checkBox1.isSelected();
```

