Application Program Development

Segment : Working with Menus

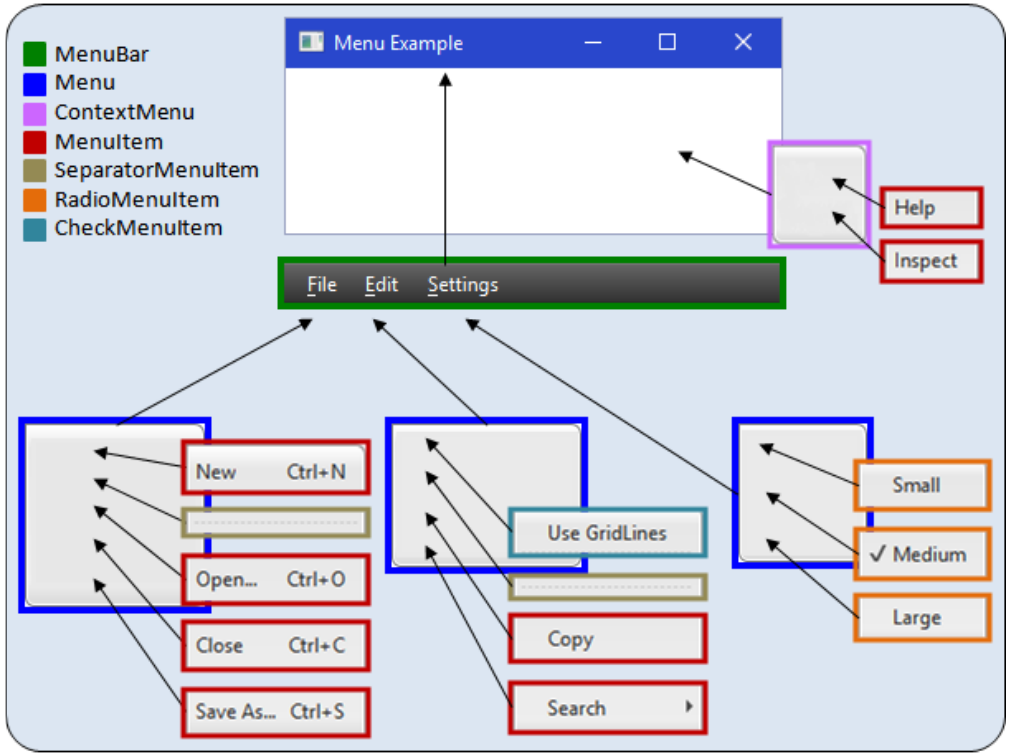
Mahboob Ali

Outcomes

- Understanding Menus in JavaFX
- Different options to work with Menus.

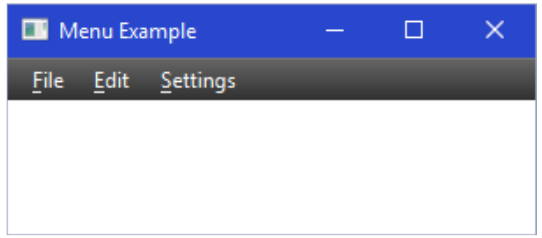
Working with Menus

- A ***menu*** is a list of commands presented to the user at his/her request.
- Menus can be attached to a menu bar at the top of an application or they may be pop-up menus that appear anywhere on the screen.
- In JAVA FX, menus are as easy to use as buttons.
- There are several component classes that may be used including **MenuBar**, **Menu**, **ContextMenu**, **MenuItem**, **CheckMenuItem**, **SeparatorMenuItem** and **RadioMenuItem**.



Example

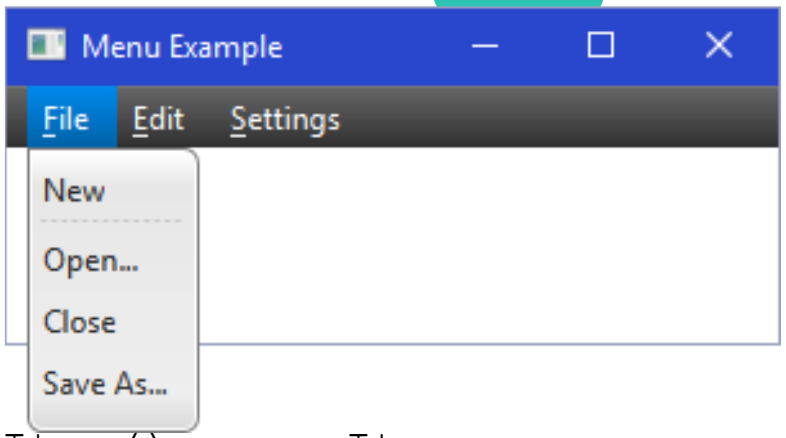
- Notice that the **MenuBar** is attached to the main application **Pane** as well as the **ContextMenu**.
- The individual **Menus** are then added to the **MenuBar**, or to another menu to form a ***cascaded menu*** (e.g., the **Search** menu here).
- The **MenuItems** are simply added to the **Menus**.




```
import javafx.application.Application; import javafx.event.*;
import javafx.scene.Scene; import javafx.scene.control.*;
import javafx.scene.layout.VBox; import javafx.stage.Stage;

public class MenuExample extends Application {
        private VBox aPane;
        public void start(Stage primaryStage) {
                aPane = new VBox(); Scene scene = new Scene(aPane, 300, 100);
                Menu fileMenu = new Menu("_File"); Menu editMenu = new Menu("_Edit");
                Menu settingsMenu = new Menu("_Settings");
                // Add the menus to a menubar and then add the menubar to the pane
                MenuBar menuBar = new MenuBar();
                menuBar.getMenus().addAll(fileMenu, editMenu, settingsMenu);
                aPane.getChildren().add(menuBar);
                primaryStage.setTitle("Menu Example"); primaryStage.setScene(scene);
                primaryStage.show();
        } public static void main(String[] args) { launch(args); }
}
```

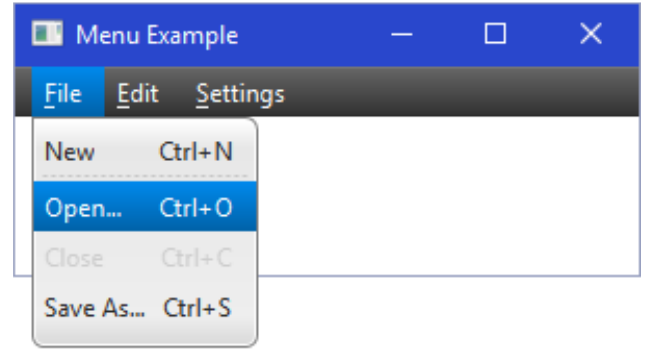
- To create the items in the menus, we simply create and add **MenuItems**.

```
MenuItem newItem = new MenuItem("New");
MenuItem openItem = new MenuItem("Open...");
MenuItem closeItem = new MenuItem("Close");
MenuItem saveAsItem = new MenuItem("Save As...");

fileMenu.getItems().addAll(newItem, new SeparatorMenuItem(), openItem,
                           closeItem, saveAsItem);
```



- A **SeparatorMenuItem** was added between the **New** and **Open...** options in the menu.
- It is good to often add key ***Accelerators***, which are "quick key" presses that allow the user to select the menu item without having to go through the menus. Here is how to add some of these:

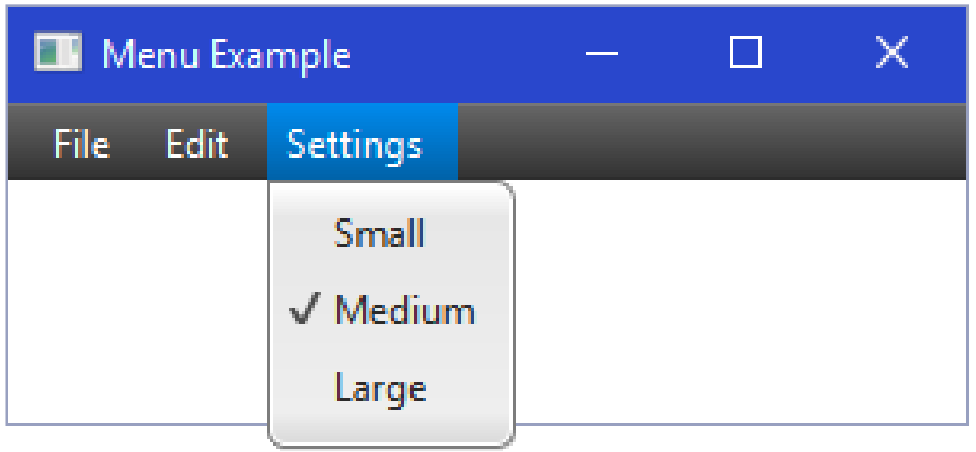
```
newItem.setAccelerator(KeyCombination.keyCombination("Ctrl+N"));
openItem.setAccelerator(KeyCombination.keyCombination("Ctrl+O"));
closeItem.setAccelerator(KeyCombination.keyCombination("Ctrl+C"));
saveAsItem.setAccelerator(KeyCombination.keyCombination("Ctrl+S"));
closeItem.setDisable(true);
```



- The **Settings** menu will have some **RadioMenuItems** on it so that only one will be selected at a time. Here is how to do that:

```
ToggleGroup settingGroup = new ToggleGroup();
RadioMenuItem smallItem = new RadioMenuItem("Small");
smallItem.setToggleGroup(settingGroup);
RadioMenuItem mediumItem = new RadioMenuItem("Medium");
mediumItem.setToggleGroup(settingGroup);
RadioMenuItem largeItem = new RadioMenuItem("Large");
largeItem.setToggleGroup(settingGroup);
settingsMenu.getItems().addAll(smallItem, mediumItem, largeItem);
```

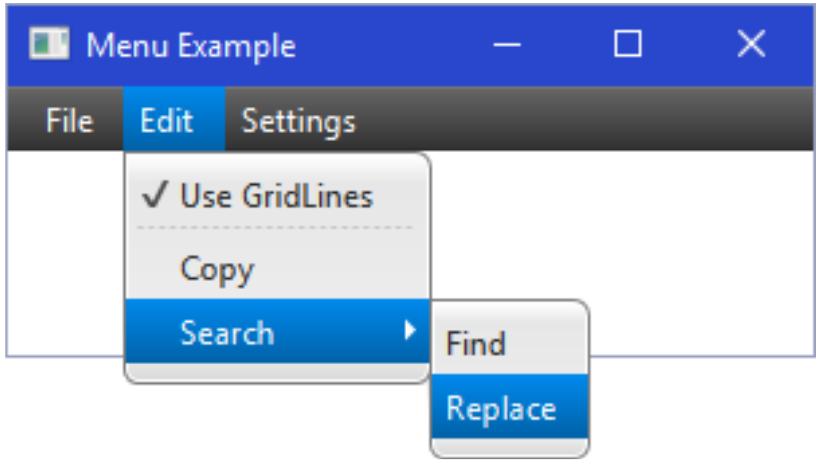
- Each **RadioMenuItem** is added to a **ToggleGroup**, to ensure that only one can be selected at a time, and then all are added to the menu



- **CheckMenuItems** are added in the same manner.
- Also, we can add a ***cascaded menu*** (i.e., a menu within a menu) just by adding a menu to another menu as if it was a **MenuItem**:

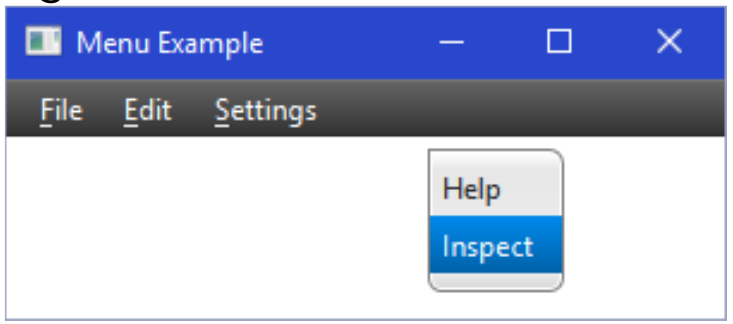
```
Menu searchMenu = new Menu("Search");
MenuItem findItem = new MenuItem("Find");
MenuItem replaceItem = new MenuItem("Replace");
searchMenu.getItems().addAll(findItem, replaceItem);

CheckMenuItem gridItem = new CheckMenuItem("Use GridLines");
MenuItem copyItem = new MenuItem("Copy");
editMenu.getItems().addAll(gridItem, new SeparatorMenuItem(), copyItem,
                                                    searchMenu);
```

- Notice how the **SearchMenu** is created like any other menu, but it is then added to the **Edit** menu.



- A final menu to be added is a **ContextMenu**.
- This is a menu that will pop up wherever we right click on the pane.
- Notice that it is created like any other menu. However, it is not added to anything:

```
ContextMenu popupMenu = new ContextMenu();
MenuItem helpItem = new MenuItem("Help");
MenuItem inspectItem = new MenuItem("Inspect");
popupMenu.getItems().addAll(helpItem, inspectItem);
```



- We create a MOUSE_CLICKED event handler to cause it to appear by using the **show()** method:

```
aPane.setOnMouseClicked(new EventHandler<MouseEvent>() {
        public void handle(MouseEvent e) {
                if (e.getButton() == MouseButton.SECONDARY)
                        popupMenu.show(aPane,e.getScreenX()-50,e.getScreenY()-25);
                }
        });
```

- **show()** method required the **Pane** on which it is to be shown, followed by the location of the top left of the menu.
- This location has been set to **50** pixels left and **25** pixels up from the location that the mouse was clicked.
- The code also ensures that it was the right mouse button that was clicked (i.e., `MouseButton.SECONDARY`).

- In order to get the menus to respond to user selections, we need to add the event handlers.
- This is done in the same manner as setting up event handlers for **Buttons**.
- We simply add it to the **MenuItem** instead of the **Button** as follows:

```
newItem.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
                System.out.println("NEW has been pressed"); }});
```

- When it comes to **CheckMenuItems**, we may want to determine whether or not it was just "checked" or "unchecked". We can do this with the **isSelected()** method:

```
gridItem.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
                if (((CheckMenuItem)e.getSource()).isSelected())
                        System.out.println("USE GRIDLINES has been selected");
                else
                        System.out.println("USE GRIDLINES has been unselected");
                }
        });
```

- Notice here that we can get the source of the **ActionEvent** and ask if it has been selected.
- We need to type-cast to **CheckMenuItem** because the **getSource()** method returns an **Object** type.
- Alternatively, we can store the **CheckMenuItem** as an instance variable and then can access it by means of that variable instead of asking the event for its source.

- Lastly, we can disable and enable various menu items whenever we want.

For example, consider the **File** menu. We may want to disable the **Close** option while nothing has been opened. Once a file is opened, we could re-enable the **Close** option.

Similarly, if we want only one file open at a time, once a file has been opened, we could disable the **New** and **Open** options. We could then re-enable them once the file has been closed.

```
newItem.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
                closeItem.setDisable(false);
                newItem.setDisable(true);
                openItem.setDisable(true);
                System.out.println("NEW has been pressed");
        }
});
openItem.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
                closeItem.setDisable(false);
                openItem.setDisable(true);
                newItem.setDisable(true);
                System.out.println("OPEN has been pressed");
        }
});
closeItem.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent e) {
                openItem.setDisable(false); newItem.setDisable(false);
                closeItem.setDisable(true);
                System.out.println("CLOSE has been pressed");
        }
});
```