



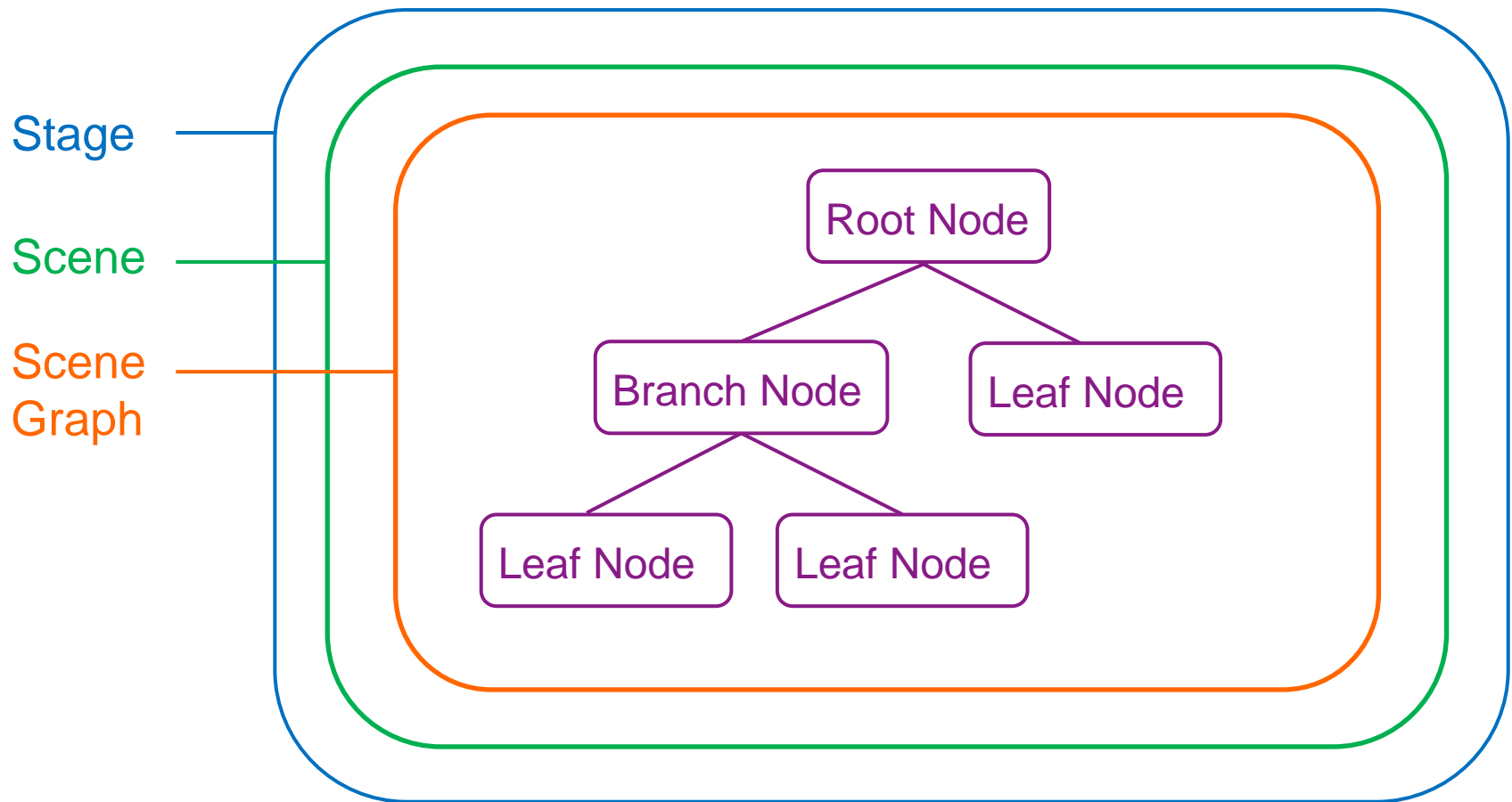
Chapitre 7 bis

Composants JavaFX

Interface utilisateur graphique et gestion des événements

1. Structure de l'application

Structure de l'application




1. Structure de l'application
2. Application JavaFX

Application JavaFX

- Sous-classe de la classe *Application*
 - Méthode *main*
 - Contient : *launch*(args) ⇒ appelle la méthode start
 - Méthode *start*
 - Argument
 - Un objet de type *Stage* créé automatiquement par la plateforme

Application JavaFX

```
public class Principal extends Application {  
  
    public static void main (String[ ] args) {  
        launch(args);  
    }  
  
    @Override  
    public void start (Stage primaryStage) {
```



Appelle

Application JavaFX

- Dans la méthode **start**
 - Préparer un **Scene Graph** avec les nœuds nécessaires
 - Préparer un objet de type **Scene**
 - Avec les bonnes dimensions
 - Y placer la racine (Root Node) du **Scene Graph**
 - Préparer un objet de type **Stage**
 - Y ajouter l'objet de type **Scene**
 - Afficher le contenu de l'objet de type **Stage**

1. Structure de l'application
2. Application JavaFX
3. Noeuds

Noeuds

- Objets graphiques (2D ou 3D)
 - Ex : Circle, Rectangle, Polygon...
- UI Controls
 - Ex : Button, CheckBox, ComboBox, TextArea...
- Conteneurs (Layout Panes)
 - Ex : BorderPane, GridPane, FlowPane...
- Éléments de type Media
 - Ex : Audio, Video, Image...

Noeuds

- Différents types
 - **Root Node** : le premier noeud (racine)
 - **Branch Node / Parent Node** : avec noeuds enfants
 - **Group**
 - Contient une liste de noeuds enfants
 - **Region**
 - Classe de base basée sur les contrôles UI
 - Ex : Chart, Pane, Control
 - **WebView**
 - Pour le Web
 - **Leaf Node** : sans enfant
 - Ex : Rectangle, Ellipse, Box, ImageView, MediaView...

1. Structure de l'application
2. Application JavaFX
3. Noeuds
4. Scene Graph

Scene Graph

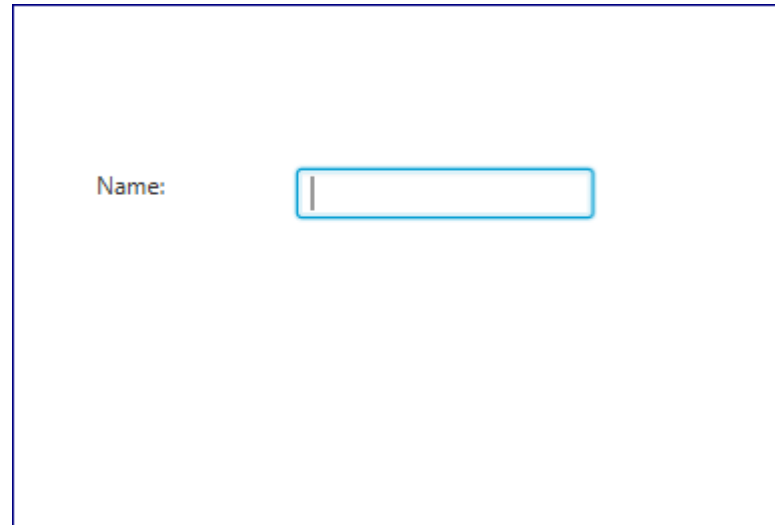
- Structure hiérarchique (arbre)
- Représente le contenu d'une scène
- Le noeud racine (Root Node) peut être
 - **Group**
 - **Region**
 - **WebView**

Root Node de type Group

- **Group**
 - Contient une liste de noeuds enfants
 - Toute action appliquée au groupe l'est sur chacun des enfants
 - Ex: l'affichage
- Méthode **getChildren()**
 - Retourne un objet de type **ObservableList**
 - qui contient la liste des noeuds enfants
 - à laquelle on peut ajouter de nouveaux noeuds
- Les composants sont positionnés au pixel près
 - Via **setLayout(...)** appelés sur les composants

Scene Graph

Exemple :



A JavaFX scene graph example. It consists of a single root node, a `Scene`, which contains a single child node, a `Text` node. The `Text` node is labeled "Name:" and is positioned to the left of a text input field. The input field is a `TextField` node, which is a child of the `Text` node. The input field is empty and has a light blue border.

Scene Graph

Exemple :

```
Label nameLabel = new Label("Name: ");  
TextField nameTextField = new TextField();  
  
// Positionnement des composants  
nameLabel.setLayoutX(50);  
nameLabel.setLayoutY(100);  
nameTextField.setLayoutX(150);  
nameTextField.setLayoutY(100);  
  
Group root = new Group();  
root.getChildren().add(nameLabel);  
root.getChildren().add(nameTextField);
```

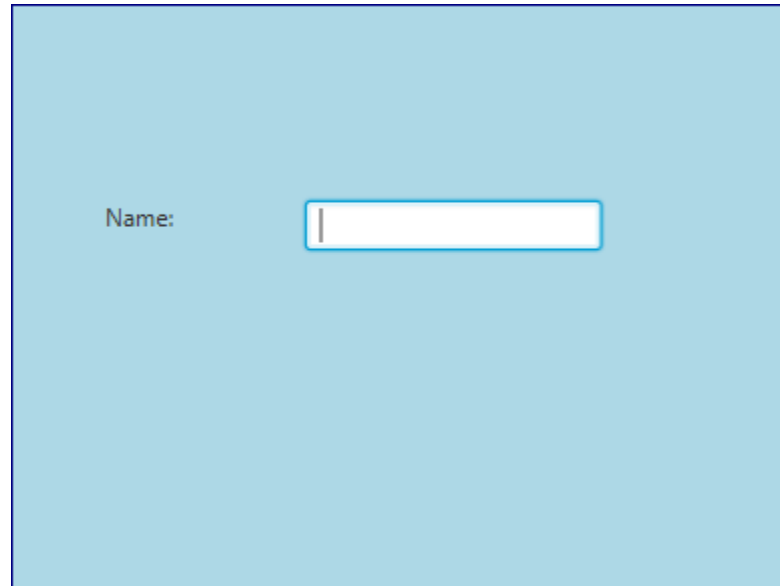
1. Structure de l'application
2. Application JavaFX
3. Noeuds
4. Scene Graph
5. Scene

Scene

- Créer un objet de type Scene
 - Précisez les dimensions
 - Y placer l'objet racine (Root Node)

Scene

Exemple :



A JavaFX Scene represented as a light blue rectangle. Inside the scene, on the left, is the text "Name:". To the right of the text is a white rectangular text input field with a thin blue border and a vertical cursor line on the left side.

Scene

Exemple :

```
Group root = new Group();
```

```
...
```

```
Scene scene = new Scene(root, 500, 550);
```

```
scene.setFill(Color.LIGHTBLUE);
```

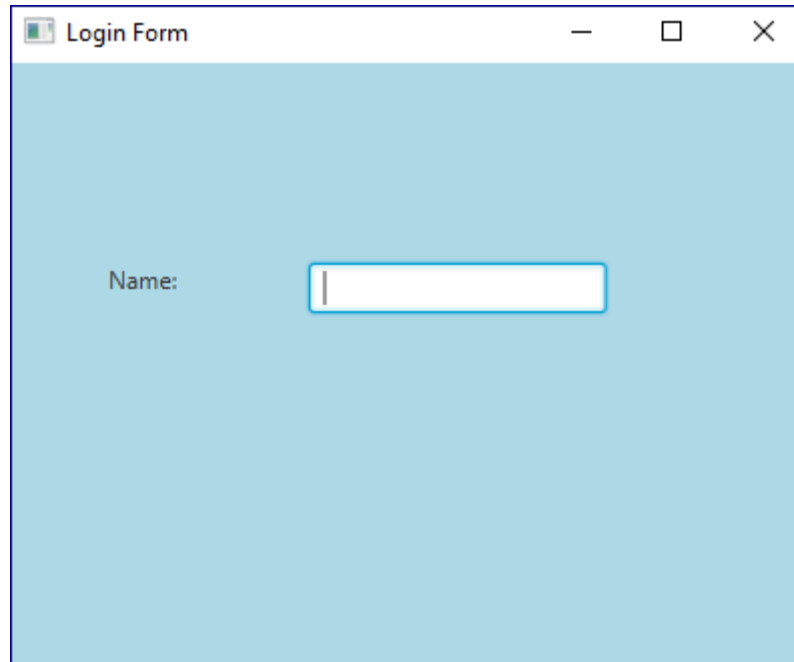
1. Structure de l'application
2. Application JavaFX
3. Noeuds
4. Scene Graph
5. Scene
6. Stage

Stage

- Créez un objet de type Stage
 - Y ajouter l'objet de type Scene
 - Afficher le contenu de l'objet de type Stage

Stage

Exemple :




A screenshot of a JavaFX application window titled "Login Form". The window has a standard title bar with minimize, maximize, and close buttons. The main content area has a light blue background. In the center, there is a label "Name:" followed by a white text input field with a blue border and a vertical cursor.

Stage

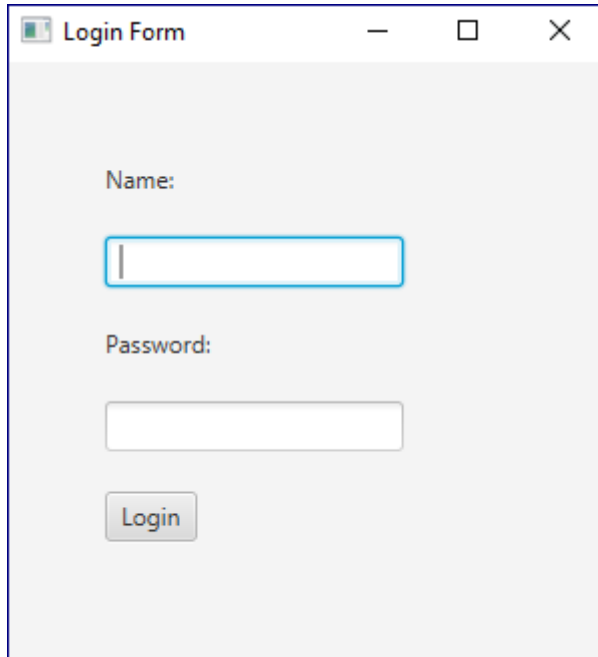
Exemple :

```
public class Principal extends Application {  
    public static void main(String[] args) {  
        launch(args);  
    }  
  
    @Override  
    public void start(Stage primaryStage) {  
        ...  
        Group root = new Group();  
        root.getChildren().add(...);  
        root.getChildren().add(...);  
        ...  
        Scene scene = new Scene(root, 400, 300);  
  
        primaryStage.setTitle("Login Form");  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
}
```



1. Structure de l'application
2. Application JavaFX
3. Noeuds
4. Scene Graph
5. Scene
6. Stage
7. FlowPane

FlowPane

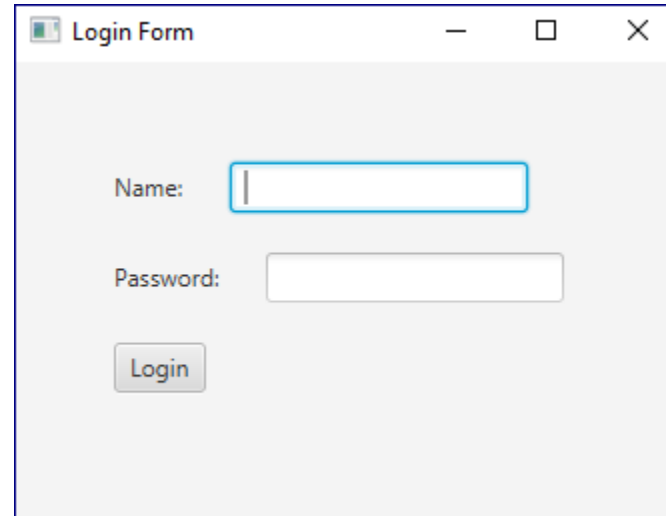


Login Form

Name:

Password:

Login

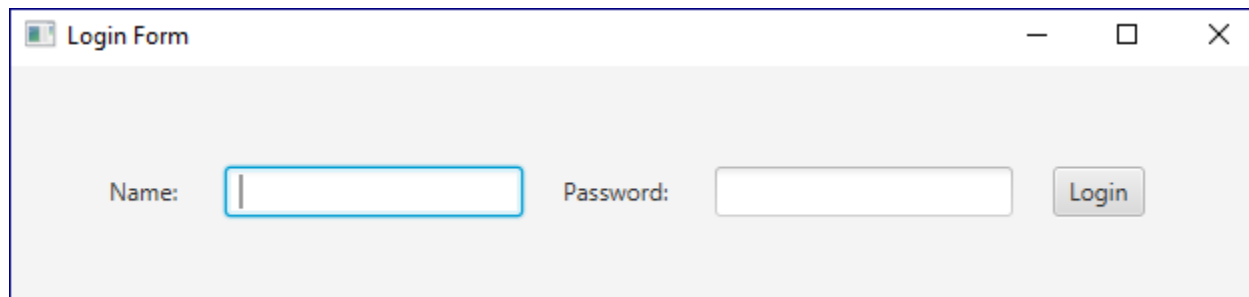


Login Form

Name:

Password:

Login



Login Form

Name:

Password:

Login

FlowPane

```
FlowPane flowPane = new FlowPane();
```

```
// Ajoute des marges autour du panneau
```

```
flowPane.setPadding(new Insets(50, 50, 50, 50));
```

```
// Ajoute un espace entre des composants (horizontalement et verticalement)
```

```
flowPane.setHgap(20);
```

```
flowPane.setVgap(20);
```

```
Label nameLabel = new Label("Name: ");
```

```
flowPane.getChildren().add(nameLabel);
```

→ Ajoute le composant au
flowLayout

```
TextField nameTextField = new TextField();
```

```
flowPane.getChildren().add(nameTextField);
```

```
Label passwordLabel = new Label("Password: ");
```

```
flowPane.getChildren().add(passwordLabel);
```

```
PasswordField passwordField = new PasswordField();
```

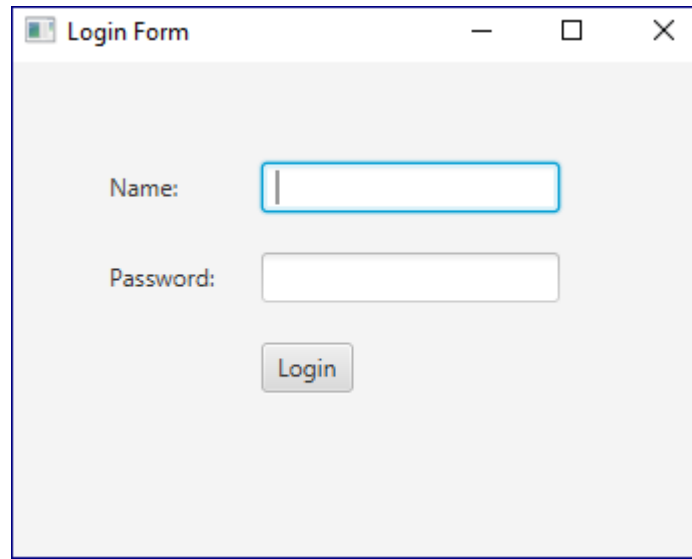
```
flowPane.getChildren().add(passwordField);
```

```
Button button = new Button("Login");
```

```
flowPane.getChildren().add(button);
```

1. Structure de l'application
2. Application JavaFX
3. Noeuds
4. Scene Graph
5. Scene
6. Stage
7. FlowPane
8. GridPane

GridPane



Composants organisés sous forme de 3 lignes X 2 colonnes

GridPane

```
GridPane gridPane = new GridPane();

gridPane.setPadding(new Insets(50, 50, 50, 50));
gridPane.setHgap(20);
gridPane.setVgap(20);

Label nameLabel = new Label("Name: ");
gridPane.add(nameLabel, 0, 0);

TextField nameTextField = new TextField();
gridPane.add(nameTextField, 1, 0);           // 2me colonne – 1re ligne

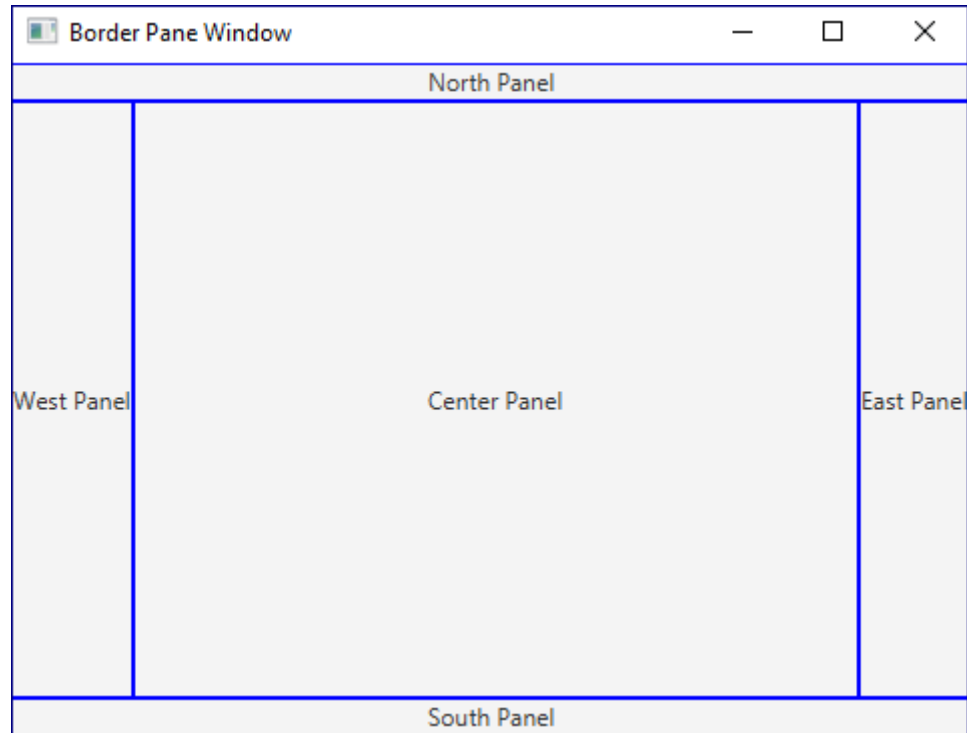
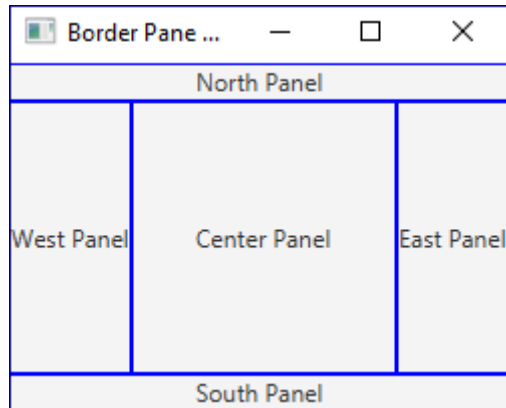
Label passwordLabel = new Label("Password: ");
gridPane.add(passwordLabel, 0, 1);

PasswordField passwordField = new PasswordField();
gridPane.add(passwordField, 1, 1);

Button button = new Button("Login");
gridPane.add(button, 1, 2);
```

1. Structure de l'application
2. Application JavaFX
3. Noeuds
4. Scene Graph
5. Scene
6. Stage
7. FlowPane
8. GridPane
9. BorderPane

BorderPane



Les composants sont placés au nord, sud, centre, est et ouest

Chaque composant étant lui-même un BorderPane contenant un label placé au centre

BorderPane

```
BorderPane borderPane = new BorderPane();
```

```
BorderPane northPanel = new BorderPane(); // Création du panneau à placer au nord  
northPanel.setBorder(...);                // Ajout d'une bordure au panneau  
northPanel.setCenter(new Label("North Panel")); // Ajout d'un label centré  
borderPane.setTop(northPanel);             // Placement du panneau au nord
```

```
BorderPane westPanel = new BorderPane();  
BorderPane centerPanel = new BorderPane();  
BorderPane eastPanel = new BorderPane();  
BorderPane southPanel = new BorderPane();  
...
```

```
borderPane.setLeft(westPanel);             // Placement du panneau à l'ouest  
borderPane.setCenter(centerPanel);         // Placement du panneau au centre  
borderPane.setRight(eastPanel);            // Placement du panneau à l'est  
borderPane.setBottom(southPanel);         // Placement du panneau au sud
```


...

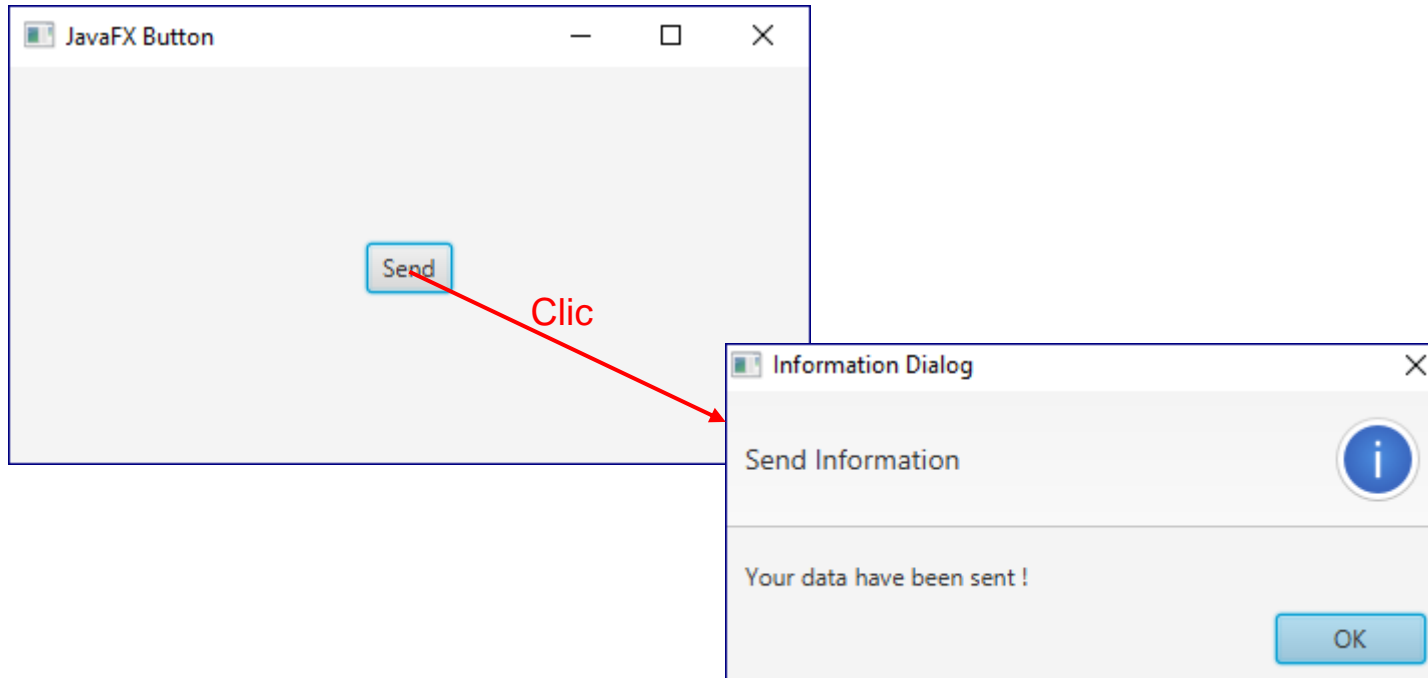
9. BorderPane

10. Button

Button

- Gestion d'événement sur un bouton
 - Implémenter l'interface `EventHandler<ActionEvent>`
 - Redéfinir la méthode
`public void handle(ActionEvent event)`

Button



Button

```
public class Principal extends Application {
```

```
...
```

```
@Override
```

```
public void start(Stage primaryStage) {
```

```
    Button button = new Button("Send");
```

Crée un écouteur d'évènement

```
    ButtonListener buttonListener = new ButtonListener();
```

```
    button.setOnAction(buttonListener);
```

Associe l'écouteur au composant à écouter

```
    ... // Création d'un objet Scene contenant le bouton et affichage de primaryStage
```

```
}
```

```
private class ButtonListener implements EventHandler<ActionEvent> {
```

Interface

```
    public void handle(ActionEvent event) {
```

Appelée si clic sur bouton

```
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Information Dialog");
        alert.setHeaderText("Send Information");
        alert.setContentText("Your data have been sent !");
        alert.showAndWait(); }
```

*Affichage
d'une boîte de
dialogue*

```
}}
```

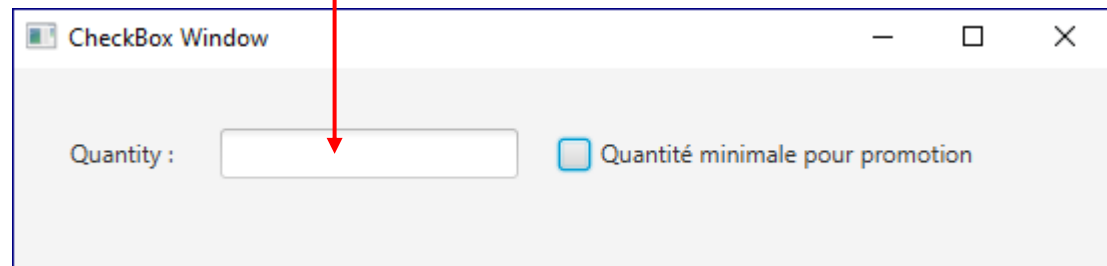
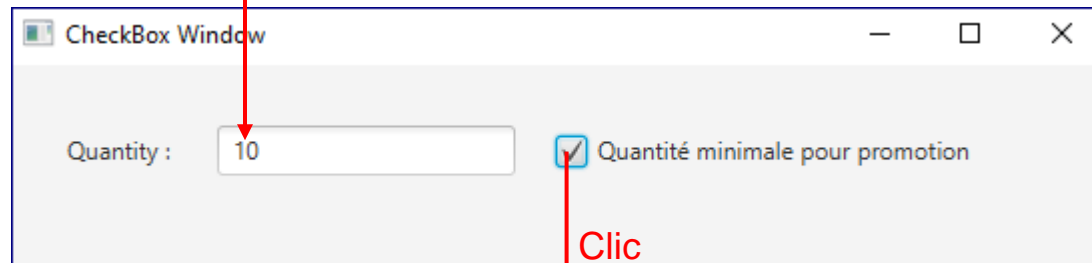
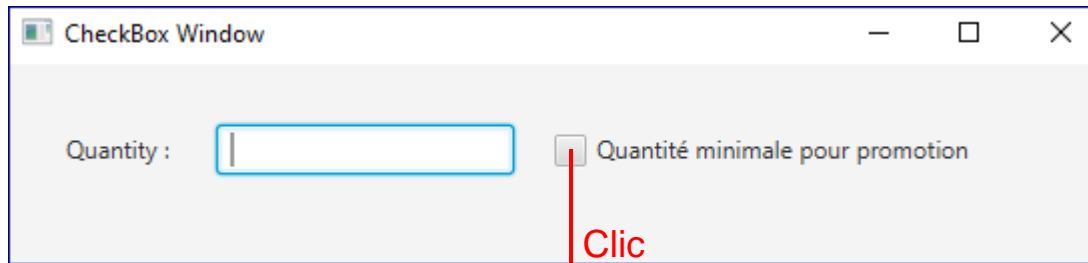
...

9. BorderPane

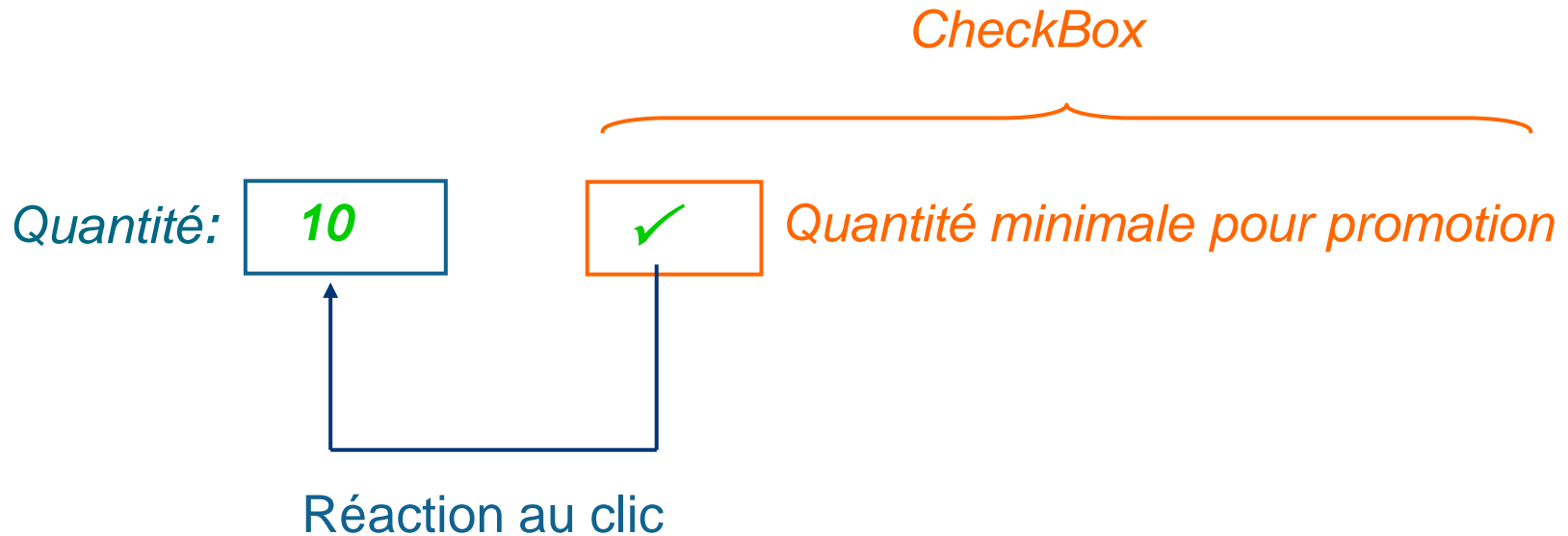
10. Button

11. CheckBox

CheckBox



CheckBox



CheckBox

```
public class Principal extends Application {  
  
    private Label quantityLabel;  
    private TextField quantityText;  
    private CheckBox defaultQuantity;  
  
    @Override  
    public void start(Stage primaryStage) {  
  
        quantityLabel = new Label("Quantité : ");  
        quantityText = new TextField();  
  
        defaultQuantity = new CheckBox("Quantité minimale pour promotion");  
  
        ...  
  
    }  
}
```


Gestion événements – Version 1

```
...
@Override
public void start(Stage primaryStage) {
    ...
    CheckBoxListener checkBoxListener = new CheckBoxListener();
    defaultQuantity.setOnAction(checkBoxListener);
    ...
}

private class CheckBoxListener implements EventHandler<ActionEvent> {
    public void handle(ActionEvent event) {
        if (defaultQuantity.isSelected())
            quantityText.setText("10");
        else quantityText.setText("");
    }
}
```

Crée un écouteur d'évènement

Associe l'écouteur au composant à écouter

Interface

Appelée si clic sur check box

Gestion événements – Version 2

```
...
@Override
public void start(Stage primaryStage) {
    ...
    CheckBoxListener checkBoxListener = new CheckBoxListener();
    defaultQuantity.selectedProperty().addListener(checkBoxListener);
    ...
}

private class CheckBoxListener implements ChangeListener {
    @Override
    public void changed(ObservableValue observable, Object oldValue, Object newValue) {
        if ( defaultQuantity.isSelected( ))
            quantityText.setText("10");
        else    quantityText.setText("");
    }
}
```

Crée un écouteur d'évènement

Associe l'écouteur au composant à écouter

Interface

Appelée si clic sur check box

...

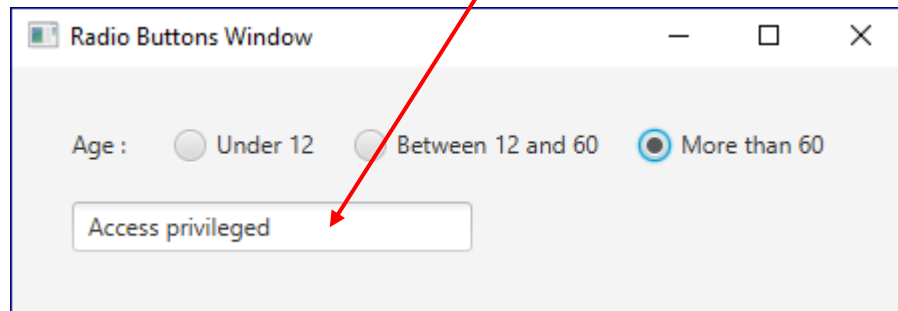
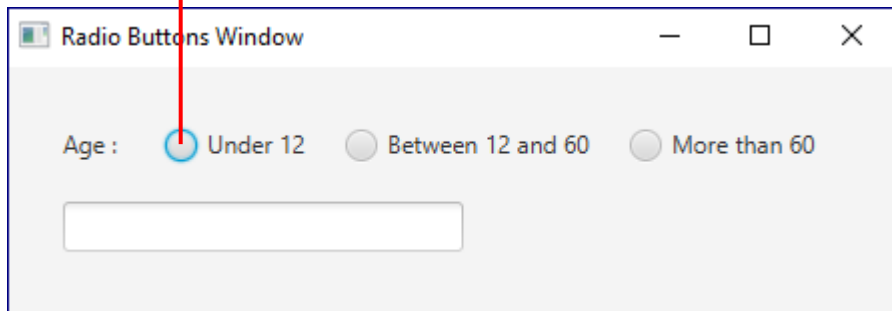
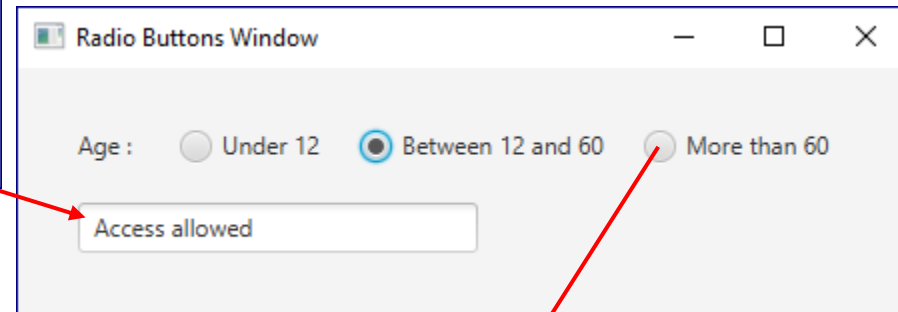
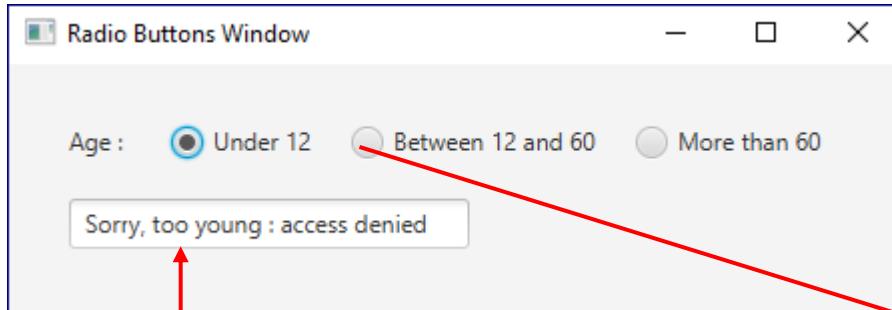
9. BorderPane

10. Button

11. CheckBox

12. RadioButton

RadioButton



RadioButton

```
public class Principal extends Application {  
    ...  
    private TextField zoneText;  
    private ToggleGroup ageGroup ; —————→ Gère le groupe : un seul bouton radio coché à la fois  
    private RadioButton under12RadioButton;  
    private RadioButton between12and60RadioButton;  
    private RadioButton upper60RadioButton;  
  
    @Override  
    public void start(Stage primaryStage) {  
        ...  
        under12RadioButton = new RadioButton("Under 12");  
        between12and60RadioButton = new RadioButton("Between 12 and 60");  
        upper60RadioButton = new RadioButton("More than 60");  
  
        ageGroup = new ToggleGroup();  
        under12RadioButton.setToggleGroup(ageGroup);  
        between12and60RadioButton.setToggleGroup(ageGroup);  
        upper60RadioButton.setToggleGroup(ageGroup);  
    }  
}
```

*Ajouter les boutons
au ToggleGroup*

RadioButton

```
public void start(Stage primaryStage) {  
    ...  
    RadioButtonListener radioButtonListener = new RadioButtonListener();  
    under12RadioButton.selectedProperty().addListener(radioButtonListener);  
    between12and60RadioButton.selectedProperty().addListener(radioButtonListener);  
    upper60RadioButton.selectedProperty().addListener(radioButtonListener);  
    ...  
}  
private class RadioButtonListener implements ChangeListener<Boolean> {  
    @Override  
    public void changed(ObservableValue observable, Boolean oldValue, Boolean newValue) {  
        if ( under12RadioButton.isSelected() && newValue )  
            zoneText.setText("Sorry, too young : access denied ");  
        else  
            if ( between12and60RadioButton.isSelected() && newValue )  
                zoneText.setText("Access allowed");  
            else if ( newValue )  
                zoneText.setText("Access privileged ");  
    }  
}
```

Crée un écouteur d'évènement

Associe l'écouteur aux radio boutons

Appelée si clic sur un bouton radion

*Seulement si coché
// si la nouvelle valeur du
bouton radio est à true*

JavaFX

...

9. BorderPane

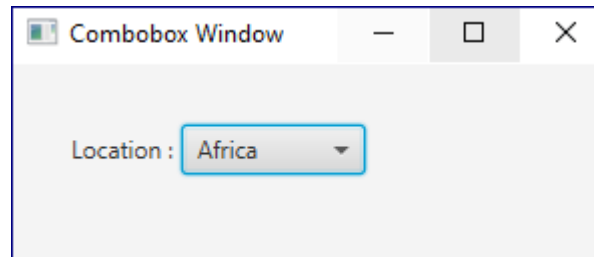
10. Button

11. CheckBox

12. RadioButton

13. Combobox

ComboBox



ComboBox

```
public class Principal extends Application {  
    ...  
    private ComboBox locationComboBox;  
  
    @Override  
    public void start(Stage primaryStage) {  
        ...  
  
        locationComboBox = new ComboBox();  
  
        // Initialisation des valeurs de la liste  
        locationComboBox.getItems().addAll  
            ("Africa", "America", "Asia", "Australia", "Europa");  
  
        // Valeur par défaut : la première de la liste  
        locationComboBox.getSelectionModel().select(0);  
    }  
}
```

ComboBox

```
public void start(Stage primaryStage) {  
    ...  
    ComboboxListener comboboxListener = new ComboboxListener();  
    locationComboBox.valueProperty().addListener(comboboxListener);  
    ...  
}
```

Crée un écouteur d'évènement

Associe l'écouteur aux radio boutons

```
private class ComboboxListener implements ChangeListener<String> {  
  
    @Override  
    public void changed(ObservableValue observable, String oldValue, String newValue) {  
        System.out.println(newValue);  
    }  
}
```

Appelée si clic sur un bouton radion

Nouvelle valeur sélectionnée dans la combobox

...

9. BorderPane

10. Button

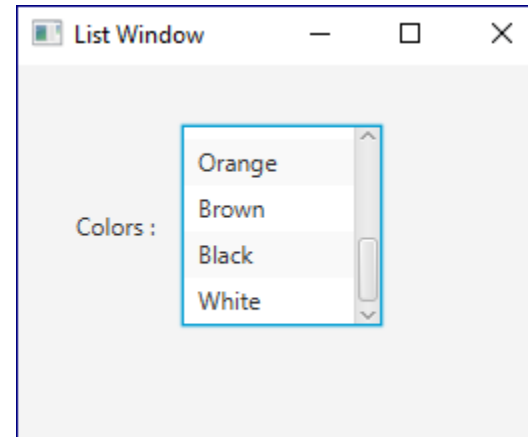
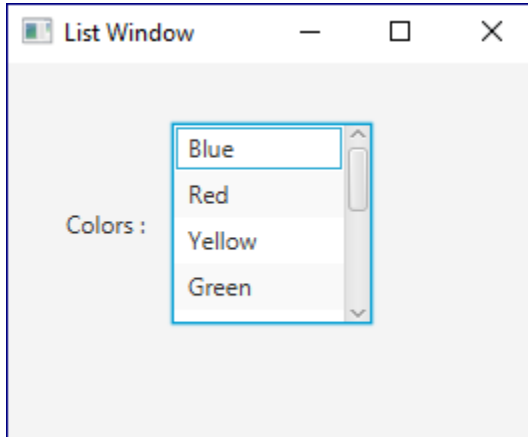
11. CheckBox

12. RadioButton

13. Combobox

14. ListView

ListView



ListView

```
public class Principal extends Application {  
    ...  
    private ListView<String> colorListView;  
  
    @Override  
    public void start(Stage primaryStage) {  
        ...  
        ObservableList<String> colors =  
            FXCollections.observableArrayList(  
                "Blue", "Red", "Yellow", "Green", "Pink", "Orange", "Brown", "Black", "White");  
  
        colorListView = new ListView<String>(colors);  
  
        // Permettre de sélectionner plusieurs valeurs  
        colorListView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);  
  
        // Préciser la largeur et la hauteur de la liste => défilant éventuel  
        colorListView.setPrefSize(100,100);  
    }  
}
```

...

9. BorderPane

10. Button

11. CheckBox

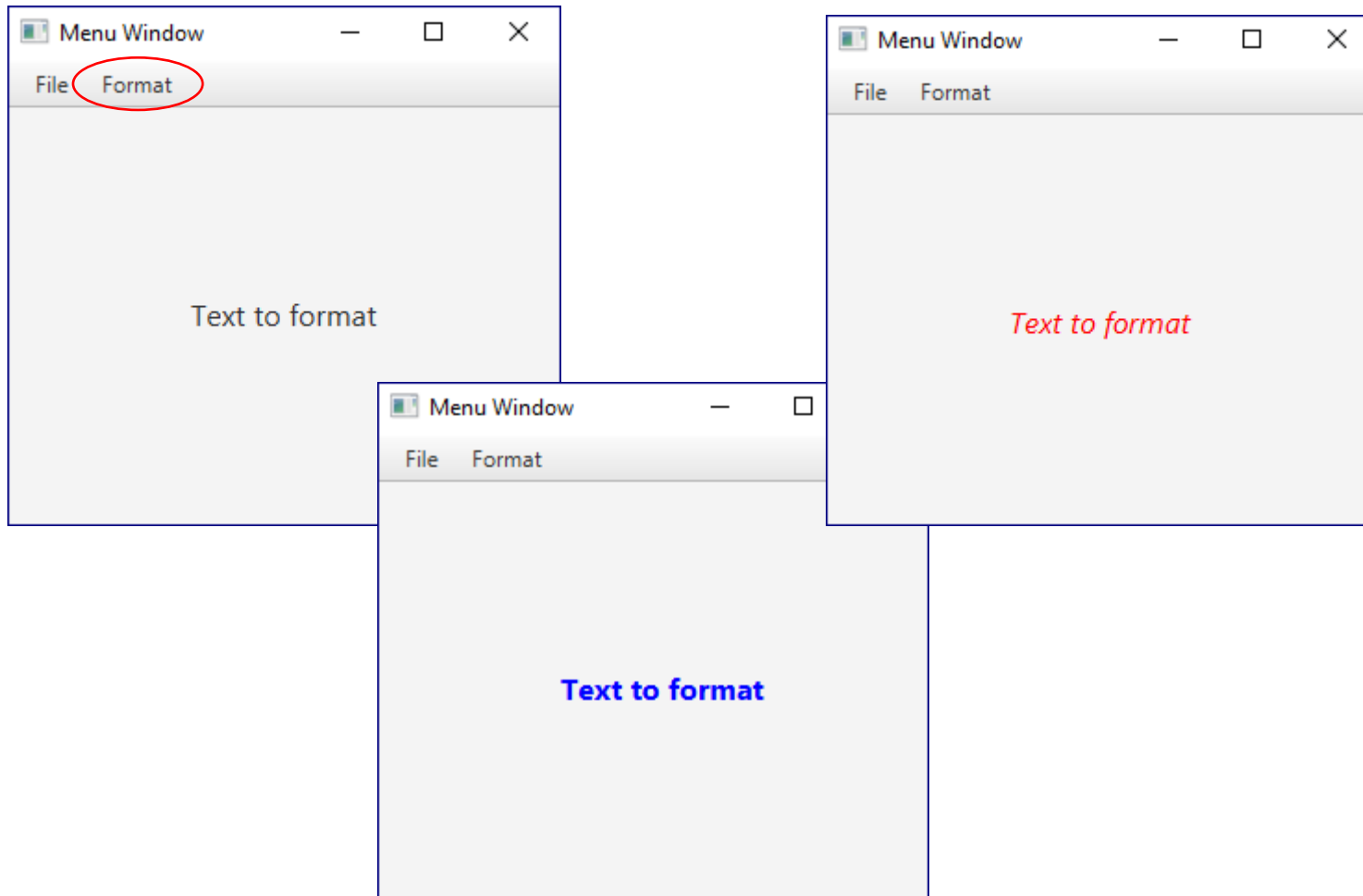
12. RadioButton

13. Combobox

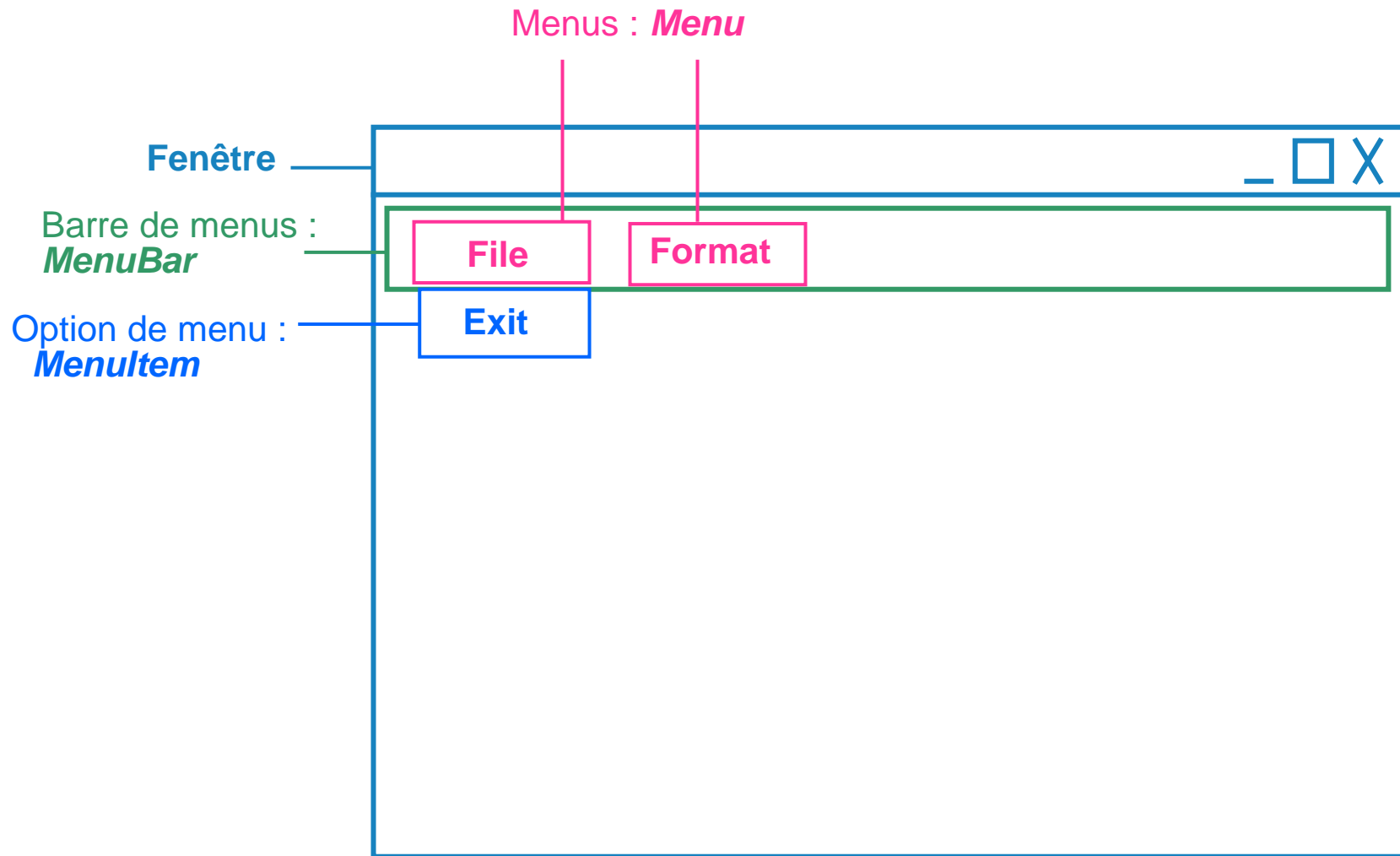
14. ListView

15. Menus

Menus



Menus




```
public class Principal extends Application {
```

```
    private MenuBar menuBar;
```

```
    private Menu fileMenu, formatMenu;
```

```
    private MenuItem exit;
```

```
@Override
```

```
public void start(Stage primaryStage) {
```

```
    ...
```

```
    fileMenu = new Menu("File");
```



Crée un menu

```
    exit = new MenuItem("Exit");
```



Crée une option de menu

```
    fileMenu.getItems().addAll(exit);
```



Ajoute l'option de menu Exit au menu File

```
    formatMenu = new Menu("Format");
```

```
    menuBar = new MenuBar();
```



Crée une barre de menus

```
    menuBar.getMenus().addAll(fileMenu, formatMenu);
```



Ajoute les menus à la barre de menus

```
    ... // Création du panneau contenant les composants. Ex: flowpane
```

```
    Scene scene = new Scene(new VBox(),300, 250);
```

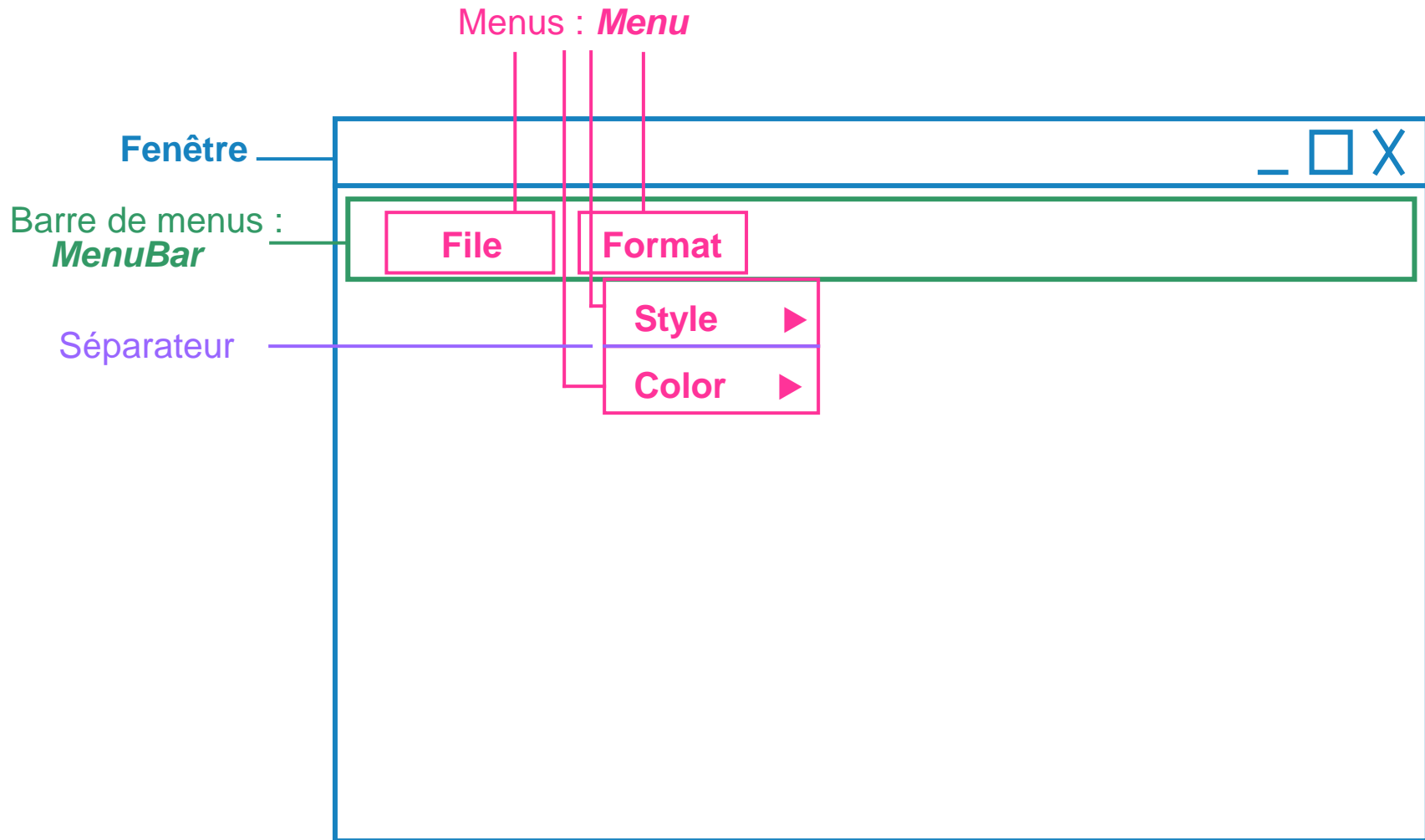
```
// L'objet Scene est une vertical box qui accueillera 2 composants : la barre de menu et le panneau (flowpane)
```

```
    ((VBox) scene.getRoot()).getChildren().addAll(menuBar, flowPane);
```



Ajoute la barre de menus à la fenêtre

Menus



Menus

```
private Menu styleMenu, colorMenu;
```

Variables d'instance

Méthode start

```
styleMenu = new Menu("Style");
```

```
colorMenu = new Menu("Color");
```

Ajoute les sous-menus au menu *Format*

```
formatMenu.getItems().addAll
```

```
(styleMenu, new SeparatorMenuItem(), colorMenu);
```

Ajoute un séparateur

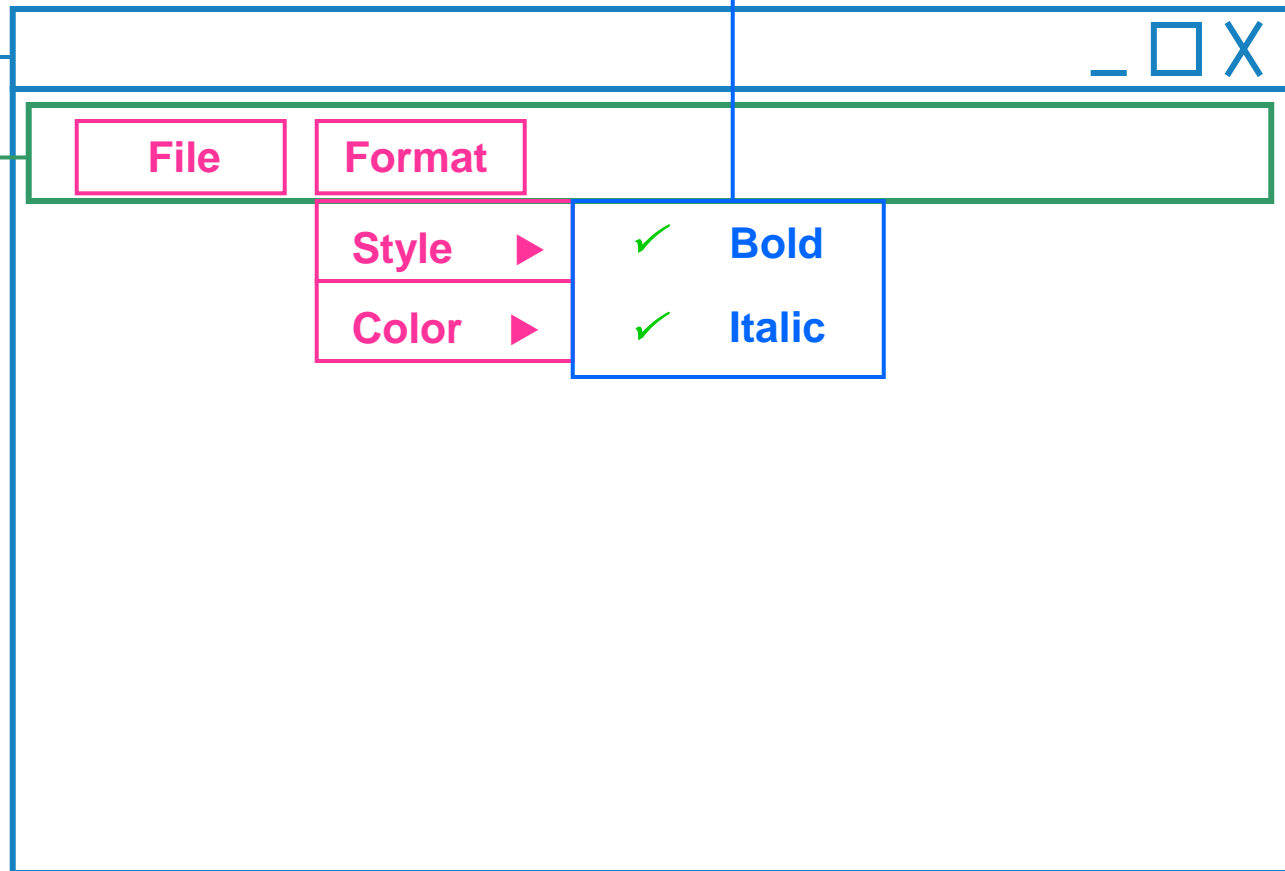
Menus

Menus : *Menu*

Menus de type case à cocher :
CheckMenuItem

Fenêtre

Barre de menus :
MenuBar



Menus

```
private CheckMenuItem bold, italic;
```

Variables d'instance

Méthode start

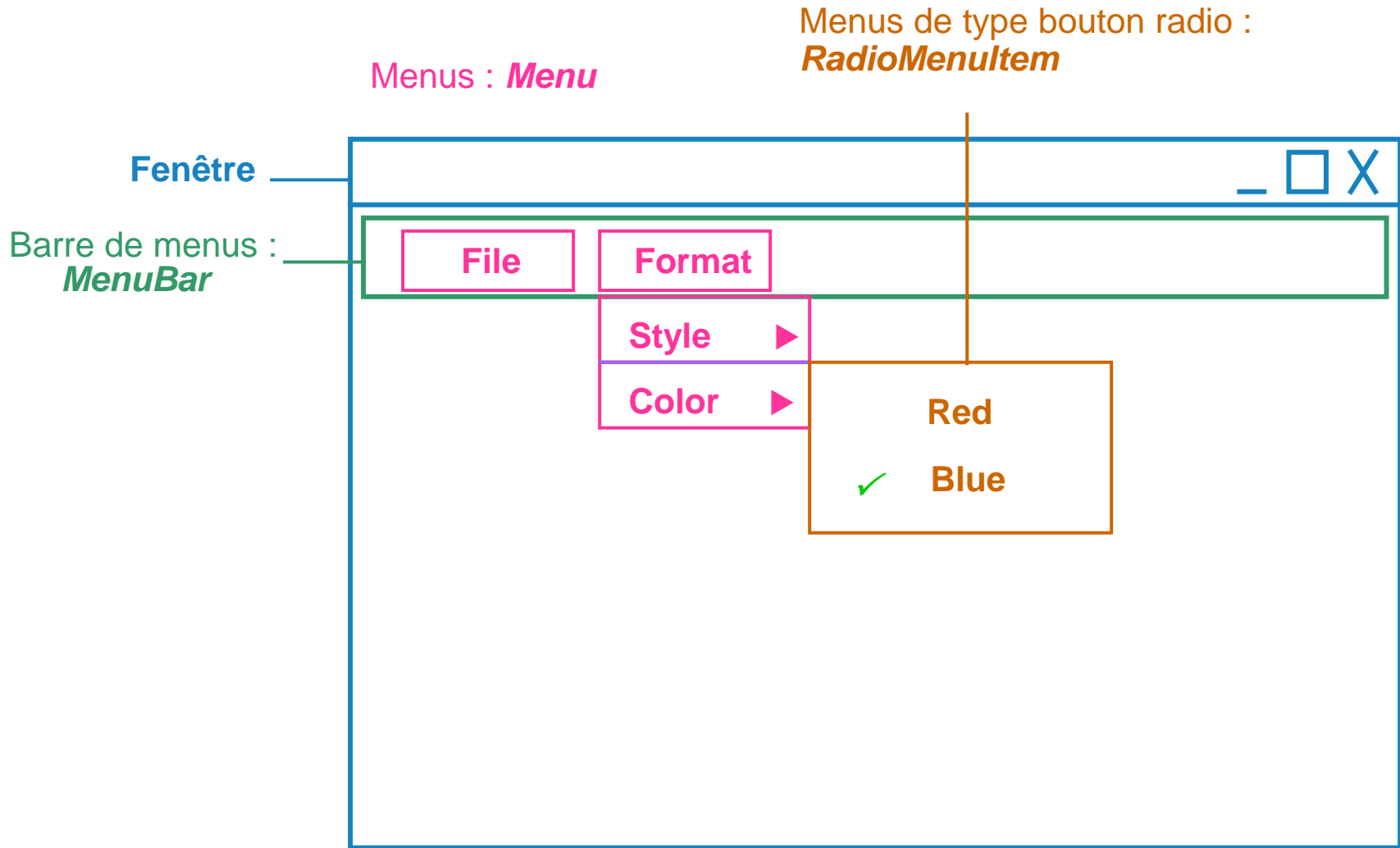
```
bold = new CheckMenuItem ("Bold");
```

```
italic = new CheckMenuItem ("Italic");
```

```
styleMenu.getItems().addAll(bold, italic);
```

→ Ajoute les options de menu au menu *Style*

Menus



Menus

```
private RadioMenuItem blue, red;  
private ToggleGroup colorGroup;
```


Variables d'instance

Méthode start

```
blue = new RadioMenuItem("Blue");
```

```
red = new RadioMenuItem("Red");
```

```
colorMenu.getItems().addAll(blue, red);
```

 Ajoute les options de menu au menu *Color*

```
colorGroup = new ToggleGroup();
```

```
blue.setToggleGroup(colorGroup);
```

```
red.setToggleGroup(colorGroup);
```

Rappel : ToggleGroup :
une seule option bouton radio
cochée à la fois

Menus

```
private Label text;  
private FontWeight currentFontWeight;    // Normal ou en gras  
private FontPosture currentFontPosture;  // Normal ou en italique
```

Variables d'instance

Méthode start

```
text = new Label("Text to format");  
text.setFont(Font.font("Helvetica", FontWeight.NORMAL, FontPosture.REGULAR, 16));
```


Menus – Gestion événements

```
exit = new JMenuItem("Exit");
```

```
ExitMenuItemListener exitMenuItemListener = new ExitMenuItemListener();
```

```
exit.setAction(exitMenuItemListener);
```

```
private class ExitMenuItemListener implements EventHandler<ActionEvent> {  
    public void handle(ActionEvent event) {  
        System.exit(0);  
    }  
}
```

Menus – Gestion événements

```
CheckMenuItemListener checkMenuItemListener = new CheckMenuItemListener();
```

```
bold.selectedProperty().addListener(checkMenuItemListener);
```

```
italic.selectedProperty().addListener(checkMenuItemListener);
```

```
...
```

```
private class CheckMenuItemListener implements ChangeListener {
```

```
    @Override
```

```
    public void changed(ObservableValue observable, Object oldValue, Object newValue) {
```

```
        if (bold.isSelected())
```

```
            currentFontWeight = FontWeight.BOLD;
```

```
        else
```

```
            currentFontWeight = FontWeight.NORMAL;
```

```
        if (italic.isSelected())
```

```
            currentFontPosture = FontPosture.ITALIC;
```

```
        else
```

```
            currentFontPosture = FontPosture.REGULAR;
```

```
        text.setFont(Font.font("Helvetica", currentFontWeight, currentFontPosture, 16));
```


```
    }
```

```
}
```

Menus – Gestion événements

```
RadioMenuItemListener radioMenuItemListener = new RadioMenuItemListener();  
blue.selectedProperty().addListener(radioMenuItemListener);  
red.selectedProperty().addListener(radioMenuItemListener);
```

```
private class RadioMenuItemListener implements ChangeListener<Boolean> {  
    @Override  
    public void changed(ObservableValue observable, Boolean oldValue, Boolean newValue) {  
        if ( blue.isSelected() && newValue )  
            text.setTextFill(Color.BLUE);  
        else  
            if ( newValue )  
                text.setTextFill(Color.RED);  
    }  
}
```

 Si cochée

...

9. BorderPane

10. Button

11. CheckBox

12. RadioButton

13. Combobox

14. ListView

15. Menus

16. Gestion des événements sur une fenêtre

Gestion d'événements sur la fenêtre

On peut écouter l'objet de type Stage

⇒ réagir aux différents événements sur la fenêtre :

- Affichage (show)
- Fermeture
- Minimisation (iconification)
- "Dé-iconification"
- Agrandissement / rétrécissement
- ...



Fermer la fenêtre

```
primaryStage.setOnHiding(new EventHandler<WindowEvent>() {  
    public void handle(WindowEvent we) {  
        ...  
    }  
});
```

Autre gestion d'événements

```
primaryStage.addEventHandler(WindowEvent.WINDOW_HIDDEN,  
    new EventHandler<WindowEvent>() {  
        @Override  
        public void handle(WindowEvent window) {  
            ...  
        }  
    }  
);
```

Autres types d'événement :

WindowEvent.WINDOW_SHOWN

WindowEvent.WINDOW_SHOWING

WindowEvent.WINDOW_HIDDEN

WindowEvent.WINDOW_CLOSE_REQUEST

Iconifier / dé-iconifier la fenêtre

```
primaryStage.iconifiedProperty().addListener(new ChangeListener<Boolean>() {  
    @Override  
    public void changed(ObservableValue observable, Boolean oldValue, Boolean newValue) {  
        ...  
    }  
});
```


Agrandir / rétrécir la fenêtre

```
scene.widthProperty().addListener(new ChangeListener<Number>() {  
    @Override  
    public void changed(ObservableValue observable, Number oldSceneWidth,  
                        Number newSceneWidth) {  
        ...  
    }  
});
```

```
scene.heightProperty().addListener(new ChangeListener<Number>() {  
    @Override  
    public void changed(ObservableValue observable, Number oldSceneHeight,  
                        Number newSceneHeight) {  
        ...  
    }  
});
```