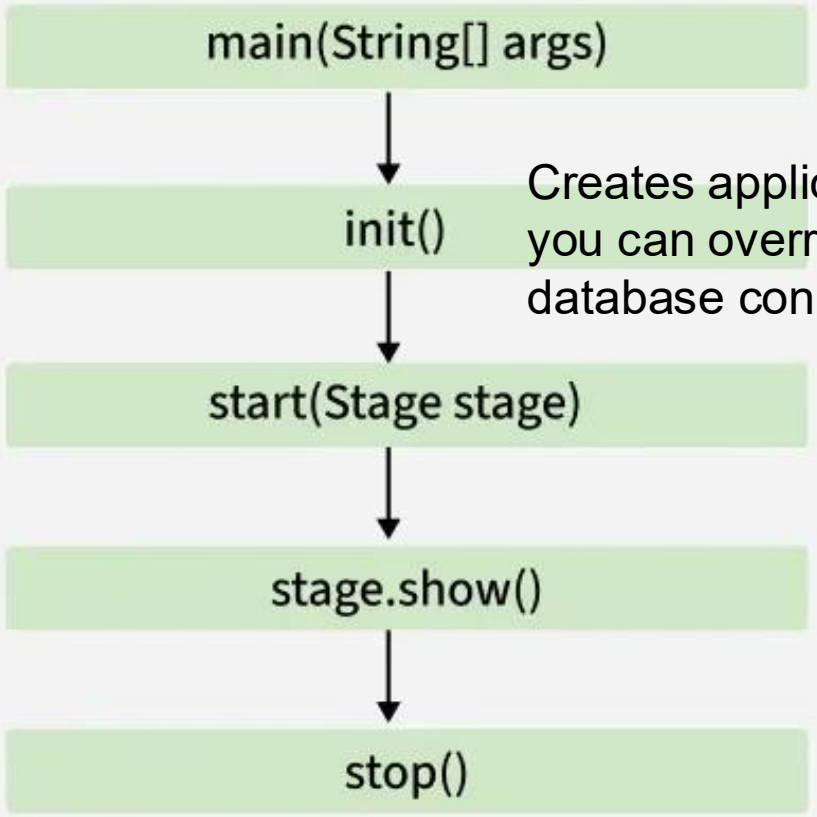
JavaFX

JavaFX overview

- JavaFX is a platform for creating cross-platform desktop applications in Java
- JavaFX provides:
  - *User Interface (UI) Controls* for building interactive applications
  - *Layouts* for managing the arrangement of UI elements
  - 2D and 3D *Shapes* for graphics
  - *Event handling* for responsive applications (e.g., due to mouse clicks)

JavaFX Application phases

Phases managed by the *javafx.application.Application* class



Calls *Application.launch()*

Creates application instance (not called directly but you can override it for important non-UI setup (like database connections)

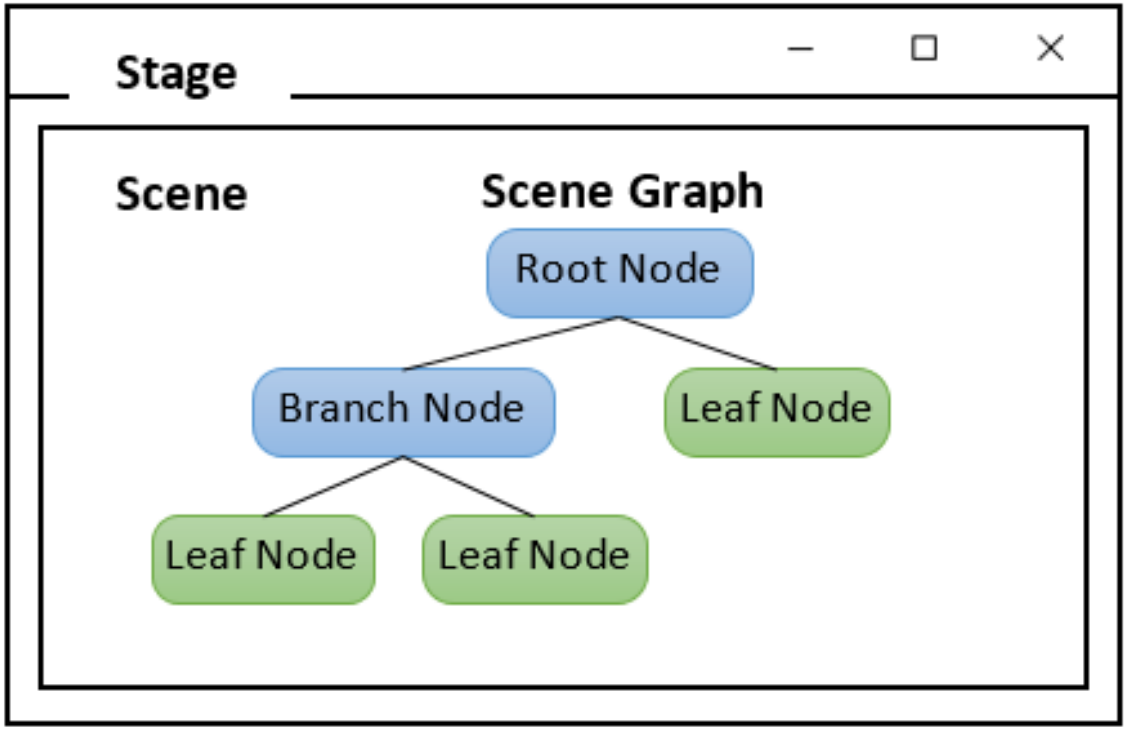
Abstract method that you must override. Where you build the UI.

Makes the window visible. Called in *start*.

Not called directly, but called when app is shutting down. Can be overwritten.

Structure of of a JavaFX application window

- Stage: application window
- Scene: container for everything displayed



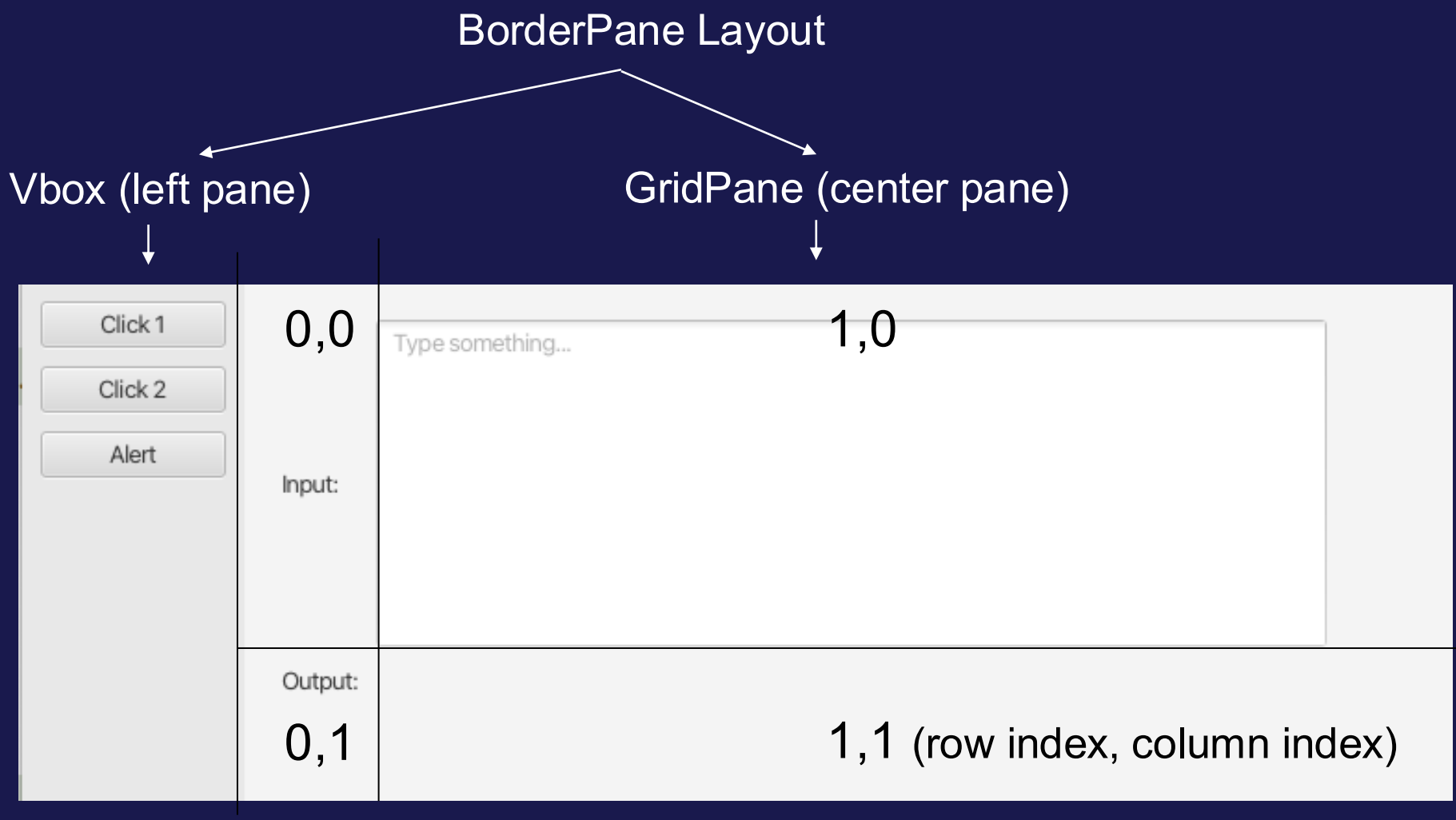
Decoration
(Title bar and border)

Content Area

To create a window:
- Create a stage object
- Assign it a scene
- Call .show()
- You may swap scenes, but it is more common to swap root node

Image from: https://www3.ntu.edu.sg/home/ehchua/programming/java/Javafx1_intro.html

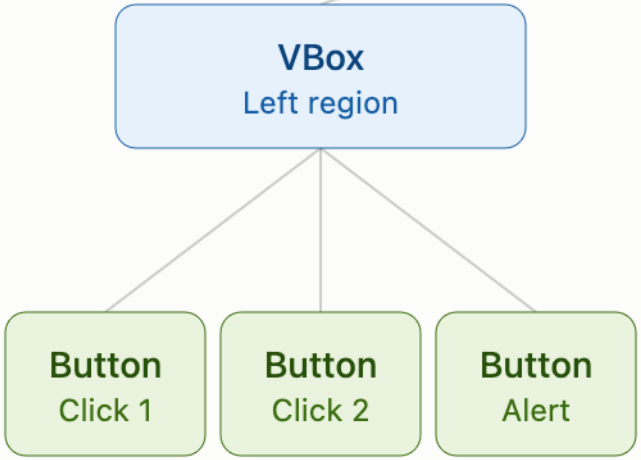
Example application



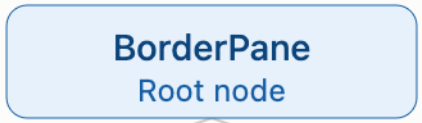
Scene graph for our example application

```
Scene scene =
    new Scene(root);
stage.setScene(scene);
```

```
VBox leftPane = new VBox(10);
```

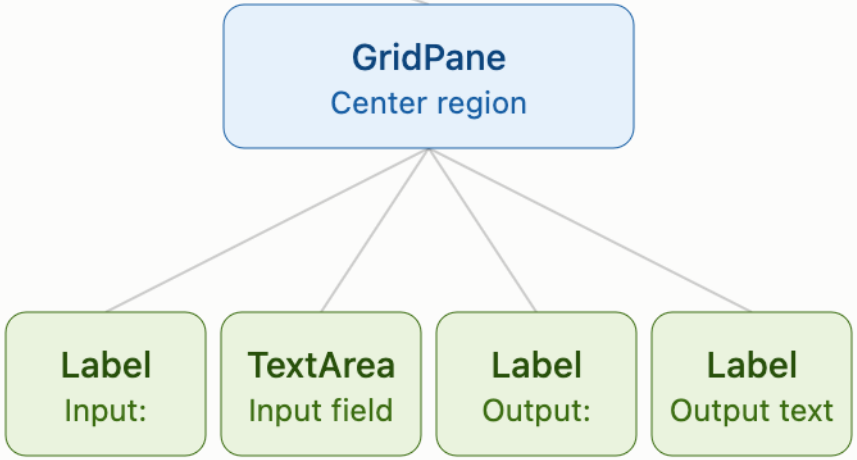



```
// add Click 1 button
Button b = new Button("Click 1");
leftPane.getChildren().add(b);
```




```
BorderPane root = new BorderPane();
root.setLeft(leftPane);
root.setCenter(grid);
```

```
GridPane grid = new GridPane();
```




```
// add Input label and text area
grid.add(new Label("Input:"), 0, 0);

TextArea input = new TextArea();
grid.add(input, 1, 0);
```

Common JavaFX Controls

- **Button:** Clickable button for user actions
- **Label:** Displays text or images
- **TextField:** Single-line text input
- **TextArea:** Multi-line text input
- **CheckBox:** Selectable box
- **ComboBox:** Drop-down list (ComboBox)
- **TableView:** Displays tabular data
- **Slider:** Selects a numeric value from a range (Slider)

Taken from https://www.geeksforgeeks.org/java/javafx-tutorial/

Common JavaFX Layouts

- **HBox:** Arranges nodes horizontally.
- **VBox:** Arranges nodes vertically.
- **BorderPane:** Arranges nodes in top, bottom, left, right, center.
- **GridPane**: Arranges nodes in a grid of rows and columns (ideal for forms).
- **FlowPane:** Arranges nodes in a flow, wrapping horizontally or vertically.
- **StackPane:** Arranges nodes on top of each other like a stack (e.g., text on top of an image).

Taken from https://www.geeksforgeeks.org/java/javafx-tutorial/

Aside: lambda expression

- A shortcut for implementing a functional interface
- Notation:
  - (parameters) -> body
  - (x) -> x + 1 // add1 example
  - x -> x + 1 // parenthesis optional with 1 argument
  - (x,y) -> x + y // …but required with >1 arguments
  - (x,y) -> { // use braces for more complex logic

    int sum = x + y;

    return sum;

    }
  - () -> System.out.println(“Hello”) // no arguments

How to set a button action

final void setOnAction(EventHandler<ActionEvent> value)

EventHandler is a *FunctionalInterface* with a single abstract *handle* method that needs to be implemented.

All functional interfaces have only a single abstract method.

We need an Object that implements the EventHandler interface and therefore implements the* handle *method

Setting a button action using an explicit named class

```
// create a class that implements EventHandler
public class ClickHandler implements
        EventHandler<ActionEvent> {

  @Override
  public void handle(ActionEvent event) {
        System.out.println("Clicked!");
  }
}

EventHandler<ActionEvent> handler = new ClickHandler();
button.setOnAction(handler);
```

Setting a button action with an anonymous inner class

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

Setting a button using a lambda expression

```
// Explicitly create an object using a lambda expression
EventHandler<ActionEvent> handler = e -> System.out.println("Clicked!");
b.setOnAction(handler);

// Lambda inline – this is preferred approach!
b.setOnAction(e -> System.out.println("Clicked!"));

// Example using braces for multiple lines of code
b.setOnAction(e -> {
        System.out.println("Clicked!")
});
```

Since EventHandler is a FunctionalInterface, the lambda expression is automatically mapped to its single handle *method*