### 31. Introduction to FXML

* + FXML - HTML for Java

### 30. Binding Properties Example

//Input and Label

TextField userInput = new TextField();

userInput.setMaxWidth(200);

Label label1 = new Label("Welcome to the site ");

Label label2 = new Label("");

HBox bottomText = new HBox(label1, label2);

bottomText.setAlignment(Pos.CENTER);

VBox vBox = new VBox(10, userInput, bottomText); //spacing of 10

vBox.setAlignment(Pos.CENTER);

label2.textProperty().bind(userInput.textProperty()); //text the user put into the field that will be displayed as a label. When the text field is changed this changes too

### 29. Binding

* + Binding - tying objects together where one is responsive to the other automatically. If x changes y changes automatically.

import javafx.beans.property.IntegerProperty; //new

import javafx.beans.property.SimpleIntegerProperty; //new

IntegerProperty x = new SimpleIntegerProperty(3); //sets the x initial value to 3

IntegerProperty y = new SimpleIntegerProperty(); //y has a null value

y.bind(x.multiply(10)); //the value of y is bound to the value of x \* 10

System.out.println("x: " + x.getValue());

System.out.println("y: " + y.getValue() + "\n");

x.setValue(9); //sets the value to 9

System.out.println("x: " + x.getValue());

System.out.println("y: " + y.getValue() + "\n");

### 28. Properties

* + In JavaFx you can treat the properties as a object, this allows us to have a listener

import javafx.beans.property.SimpleStringProperty; //new, allows us to use new string properties to read and write to

import javafx.beans.property.StringProperty; //new

public class Person {

private StringProperty firstName = new SimpleStringProperty(this, "firstName", ""); //this variable, the variable name, default value

//returns the StringProperty object

public StringProperty firstNameProperty() {

return firstName;

}

//return the firstName of value (ie. "Dakota")

public String getFirstName() {

return firstName.get();

}

//set the firstName value

public void setFirstName(String firstName) {

this.firstName.set(firstName);

}

}

public class PropertiesExample extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

…

Person person1 = new Person();

person1.firstNameProperty().addListener((v, oldValue, newValue) -> {

System.out.println("Name change to " + newValue);

System.out.println("firstNameProperty(): " + person1.firstNameProperty());

System.out.println("getFirstName(): " + person1.getFirstName());

}); //when we give the variable a name the listener will kick off

button = new Button("Click Me");

button.setOnAction(e -> person1.setFirstName("Dakota"));

…

}

}

### 27. CSS Custom Style Classes and Selectors

//CSS1.css file

.button-blue{

-fx-background-color: blue;

-fx-text-fill: white;

-fx-background-radius: 4;

}

#label-bold{

-fx-font-weight: bold;

}

public class CSSCustom extends Application{

…

nameLabel.setId("label-bold"); //sets the variable to the Id

…

//Sign up

Button signUpButton = new Button("Sign Up");

signUpButton.getStyleClass().add("button-blue"); //the sign up button has the style of button-blue from the css file

GridPane.setConstraints(signUpButton, 1, 3);

…

}

### 26. CSS Inline Styles and Selectors

* + Incline example:

nameLabel.setStyle("-fx-text-fill: white"); //sets the variable to a different color

* + Selector example:

//CSS1.css file

.root {

-fx-background-color: #383838;

}

.label{ //all variables with the Label class will have this

-fx-text-fill: white; //how the text is changed

}

.button{

-fx-background-color: linear-gradient(red, pink);

-fx-text-fill: white;

-fx-background-radius: 10;

}

### 25. CSS Themes and Styles

* + It’s a good idea to have a CSS file
  + You can style the individual buttons

public class CSSThemesandStyles extends Application{

…{

loginButton.setOnAction(e -> { //when clicked the GUI will change to a different style

setUserAgentStylesheet(STYLESHEET\_CASPIAN);});

Scene scene = new Scene(grid, 300, 250);

scene.getStylesheets().add("CSS1.css"); //overrides all the current styles to the GUI

window.setScene(scene);

window.show();

}

}

//CSS1.css file.

.root {

-fx-background-color: #383838; //changes the background color

-fx-font-size: 10px; //changes the font size

}

### 24. RadioMenuItem

import javafx.scene.control.RadioMenuItem; //new

import javafx.scene.control.ToggleGroup; //new

public class RadioMenuItemExample extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

…

//RadioMenuItem - a group of check boxes but you can only have one selected at a time

//Difficulty Menu

Menu difMenu = new Menu("\_Difficulty");

ToggleGroup difficultyToggle = new ToggleGroup(); //you need this to group the options together

RadioMenuItem easy = new RadioMenuItem("Easy");

RadioMenuItem medium = new RadioMenuItem("Medium");

RadioMenuItem hard = new RadioMenuItem("Hard");

//this sets the difficulty options in difficultyToggle

easy.setToggleGroup(difficultyToggle);

medium.setToggleGroup(difficultyToggle);

hard.setToggleGroup(difficultyToggle);

easy.setSelected(true);

difMenu.getItems().addAll(easy, medium, hard); //adds the option but can't add just the toggle group

//Main MenuBar

MenuBar menuBar = new MenuBar();

menuBar.getMenus().addAll(fileMenu, editMenu, helpMenu, difMenu);

…

}

}

### 23. CheckMenuItem

import javafx.scene.control.CheckMenuItem; //new

public class CheckMenuItemExample extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

…

//CheckMenuItem - allows you to toggle a check box on and off

//Help Menu

Menu helpMenu = new Menu("\_Help");

CheckMenuItem showLines = new CheckMenuItem("Show Line Numbers");

showLines.setOnAction(e -> {

if (showLines.isSelected()) { //if showLines is selected

System.out.println("Program will display line numbers");

} else { //if showLines is not selected

System.out.println("Hiding line numbers");

}

});

CheckMenuItem autoSave = new CheckMenuItem("Enable Autosave");

autoSave.setSelected(true); //has it so the option is selected by default

helpMenu.getItems().addAll(showLines, autoSave);

//Main MenuBar

MenuBar menuBar = new MenuBar();

menuBar.getMenus().addAll(fileMenu, editMenu, helpMenu);

…

}

}

### 22. Handling Menu Clicks

import javafx.scene.control.SeparatorMenuItem; //new

public class MenuClicks extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

…

//File menu

Menu fileMenu = new Menu("\_File"); //putting an underscore in front of a file, the program will treat it as a short cut

//Menu Items

MenuItem newFile = new MenuItem("New...");

newFile.setOnAction(e -> System.out.println("Create a New File...")); //has the function of sysout when clicked

fileMenu.getItems().add(newFile);

…

fileMenu.getItems().add(new SeparatorMenuItem()); //separates the menuItems into groups

…

//EditMenu

Menu editMenu = new Menu("\_Edit");

editMenu.getItems().add(new MenuItem("Cut"));

editMenu.getItems().add(new MenuItem("Copy"));

MenuItem paste = new MenuItem("Paste");

paste.setOnAction(e -> System.out.println("Pasted"));

paste.setDisable(true); //this will disable the paste option

editMenu.getItems().add(paste);

//Main MenuBar

MenuBar menuBar = new MenuBar();

menuBar.getMenus().addAll(fileMenu, editMenu);

layout = new BorderPane();

layout.setTop(menuBar); //sets the menuBar to the top

Scene scene = new Scene(layout, 300, 250);

window.setScene(scene);

window.show();

}

}

### 21. Making Menus

import javafx.scene.control.Menu; //new

import javafx.scene.control.MenuBar; //new

import javafx.scene.control.MenuItem; //new

public class MakingMenus extends Application{

Stage window;

BorderPane layout;

…

@Override

public void start(Stage primaryStage) throws Exception {

…

//File menu

Menu fileMenu = new Menu("File"); //takes one string parameter as the menu name

//Menu Items - adds items to your menu

fileMenu.getItems().add(new MenuItem("New Project...")); //adds in a new menu item named New Project

fileMenu.getItems().add(new MenuItem("Package..."));

fileMenu.getItems().add(new MenuItem("Class..."));

fileMenu.getItems().add(new MenuItem("Interface..."));

//if a menuItem has ... when you click it a new window/dialogue will open

//if a menuItem has > when you click it there's a new sub-menu

//if a menuItem has nothing there is only one option

//Main MenuBar - this adds the file menu, you can add more menus to it

MenuBar menuBar = new MenuBar(); //no text needed, just a bar to organize the menu

menuBar.getMenus().addAll(fileMenu);

layout = new BorderPane();

layout.setTop(menuBar);

Scene scene = new Scene(layout, 300, 250);

…

}

}

### 20. Adding and Deleting TableView Rows

public class AddDelTables extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

…

//Buttons

Button addButton = new Button("Add");

addButton.setOnAction(e -> addButtonClicked()); //when button is clicked it adds the new row

Button delButton = new Button("Delete");

delButton.setOnAction(e -> delButtonClicked()); //when button is clicked it deletes the selected row

…

}

//Add button

private void addButtonClicked() {

Product product = new Product();

product.setName(nameInput.getText()); //getText gets the value of the property text

product.setPrice(Double.parseDouble(priceInput.getText())); //sets the priceInput text to double

product.setQuantity(Integer.parseInt(quantityInput.getText())); //sets the quantityInput text to int

table.getItems().add(product);

nameInput.clear(); //clears the items in the table

priceInput.clear(); //clears the items in the table

quantityInput.clear(); //clears the items in the table

}

//Delete Button

private void delButtonClicked() {

ObservableList<Product> productSelected, allProducts;

allProducts = table.getItems();

productSelected = table.getSelectionModel().getSelectedItems();

productSelected.forEach(allProducts::remove); //for all products selected remove them from allProducts

}

private ObservableList<Product> getProduct(){...}

}

### 19. Editable Tables

public class IntroToTableView extends Application{

…

TextField nameInput, priceInput, quantityInput;

…

@Override

public void start(Stage primaryStage) throws Exception {

…

//Name Input

nameInput = new TextField();

nameInput.setPromptText("Name");

nameInput.setMinWidth(100);

//Price Input

priceInput = new TextField();

priceInput.setPromptText("Price");

priceInput.setMinWidth(100);

//Quantity Input

quantityInput = new TextField();

quantityInput.setPromptText("Quantity");

quantityInput.setMinWidth(100);

//Button

Button addButton = new Button("Add");

Button delButton = new Button("Delete");

HBox hBox = new HBox();

hBox.setPadding(new Insets(10, 10, 10, 10)); //gives padding around the layout (this hBox)

hBox.setSpacing(10); //spaces out the objects

hBox.getChildren().addAll(nameInput, priceInput, quantityInput, addButton, delButton);

…

VBox vBox = new VBox();

vBox.getChildren().addAll(table, hBox);

…

}

private ObservableList<Product> getProduct(){…}

}

### 18. Simple TableView

* + In a TableView, you need at least one column

import javafx.collections.FXCollections; //new

import javafx.scene.control.TableColumn; //new

import javafx.scene.control.TableView; //new

import javafx.scene.control.cell.PropertyValueFactory; //new

public class IntroToTableView extends Application{

Stage window;

TableView<Product> table;

public static void main(String[] args) {

launch(args);

}

@SuppressWarnings("unchecked")

@Override

public void start(Stage primaryStage) throws Exception {

window = primaryStage;

window.setTitle("Intro To Table View");

//Name column

TableColumn<Product, String> nameColumn = new TableColumn<>("Name");

nameColumn.setMinWidth(200); //minimum width of 200px

nameColumn.setCellValueFactory(new PropertyValueFactory<>("name")); //last thing to do, this need to be the exact name of the object

//Price column

TableColumn<Product, Double> priceColumn = new TableColumn<>("Price");

priceColumn.setMinWidth(100); //minimum width of 100px

priceColumn.setCellValueFactory(new PropertyValueFactory<>("price")); //last thing to do

//Quantity column

TableColumn<Product, Integer> quantityColumn = new TableColumn<>("Quantity");

quantityColumn.setMinWidth(100); //minimum width of 100px

quantityColumn.setCellValueFactory(new PropertyValueFactory<>("quantity")); //last thing to do

table = new TableView<>();

table.setItems(getProduct()); //loads in all of our data

table.getColumns().addAll(nameColumn, priceColumn, quantityColumn);

VBox vBox = new VBox();

vBox.getChildren().addAll(table);

Scene scene = new Scene(vBox); //don't need WxH for this since the table already sets them

window.setScene(scene);

window.show();

}

//get all of the products. when this is called it will return all of the products in the observable list

private ObservableList<Product> getProduct(){

ObservableList<Product> products = FXCollections.observableArrayList(); //a type of list that can store JavaFx objects

products.add(new Product("Laptop", 859.90, 20));

products.add(new Product("Tablet", 200.99, 11));

products.add(new Product("Samrt Phone", 500.76, 45));

products.add(new Product("Desktop", 1000.89, 5));

products.add(new Product("XBox", 550.45, 23));

return products;

}

}

public class Product { //we need name, price, and quantity data for each item

private String name;

private double price;

private int quantity;

public Product() { //default constructor

this.name = "";

this.price = 0.0;

this.quantity = 0;

}

public Product(String name, double price, int quantity) { //override constructor

this.name = name;

this.price = price;

this.quantity = quantity;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

}

### 17. Introduction to TableView

* + TableView - like a spreadsheet
  + When making a spreadsheet its a good idea to take that data and stick it inside a new class

### 16. TreeView

* + TreeView - you can take your list items and arrange them in a hierarchy. Like a file menu that has other files
  + Root - the main overall “tree” or the main file
    - Branch - child of the root file(branches can have other branches)
    - Leaf - a branch that has no children
    - Siblings - 2 items in the same location

import javafx.scene.control.TreeItem; //new

import javafx.scene.control.TreeView; //new

public class TreeViewExample extends Application{

Stage window;

TreeView<String> tree;

…

@Override

public void start(Stage primaryStage) throws Exception {

…

TreeItem<String> root, branch1, branch2; //setting the root and its branches

//Root

root = new TreeItem<>();

root.setExpanded(true); //when ever the program starts everything is expanded or open

//Branch1

branch1 = makeBranch("Dakota", root); //sets its name and the parent of it

makeBranch("Education", branch1); //sets its name and the parent of it

makeBranch("Athletics", branch1); //sets its name and the parent of it

makeBranch("Clubs", branch1); //sets its name and the parent of it

//Branch2

branch2 = makeBranch("Walker", root); //sets its name and the parent of it

makeBranch("Education", branch2); //sets its name and the parent of it

makeBranch("Work", branch2); //sets its name and the parent of it

//Create tree

tree = new TreeView<>(root); //the tree needs a main root

tree.setShowRoot(false); //this makes it so we don't have to expand root to see the others

tree.getSelectionModel().selectedItemProperty().addListener((v, oldValue, newValue) -> {

if(newValue != null) { //prints out everytime a new item is selected

System.out.println(newValue.getValue());

}

});

…

layout.getChildren().add(tree);

…

}

//Create branches

private TreeItem<String> makeBranch(String title, TreeItem<String> parent){

TreeItem<String> item = new TreeItem<>(title);

item.setExpanded(true); //expands the item passed through

parent.getChildren().add(item); //makes sure the item is add to the parent

return item;

}

…

}

### 15. ListView

* + ListsView - a list of items in a fixed spot. You can select multiple items

import javafx.collections.ObservableList; //new

import javafx.scene.control.ListView; //new

public class ListViewExample extends Application{

…

ListView<String> listView;

…

@Override

public void start(Stage primaryStage) throws Exception {

window = primaryStage;

window.setTitle("Listview Demo");

button = new Button("Submit");

listView = new ListView<>();

listView.getItems().addAll("Iron Man", "Captain America", "Thor", "Hulk");

listView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE); //this allows you to select multiple items. control for each individual multiple, shift to get all between to items

button.setOnAction(e -> buttonClicked());

…

}

private void buttonClicked() {

String message = "";

ObservableList<String> movies;

movies = listView.getSelectionModel().getSelectedItems();

for(String m: movies) { //one by one will store the movies in message

message += m + "\n";

}

System.out.println(message);

}

…

}

### 14. ComboBox

* + ComboBox - like a ChoiceBox but you can input your own choice in

import javafx.scene.control.ComboBox; //new

public class ComboBoxExample extends Application{

…

ComboBox<String> comboBox;

…

@Override

public void start(Stage primaryStage) throws Exception {

window = primaryStage;

window.setTitle("ComboBox Demo");

button = new Button("Submit");

comboBox = new ComboBox<>(); //like a choice box

comboBox.getItems().addAll("Spider-Man", "The Princess Bride", "Django");

comboBox.setPromptText("What is your Favorite Movie?");

comboBox.setEditable(true); //gives the user the option to write in their own movie

button.setOnAction(e -> printMovie());

comboBox.setOnAction(e -> System.out.println("User selected: " + comboBox.getValue())); //prints when selected

…

}

//Print out a movie

private void printMovie() {

System.out.println(comboBox.getValue()); //gets value then prints value when the button is selected

}

}

### 13. Listening for Selection Changes

* + Listening - waits for the user to do something

import javafx.scene.control.ChoiceBox; //new

public class ListeningForSelectionChanges extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

window = primaryStage;

window.setTitle("Drop Down List");

//Choice Box

…

//Listen for Selection Changes

choiceBox.getSelectionModel().selectedItemProperty().addListener((v, oldValue, newValue) -> System.out.println(newValue)); //the item the user selected from the list. v is the property of the item, oldValue is the item previously selected, newValue is the new item selected

…

}

}

### 12. ChoiceBox (Drop Down Menu)

* + ChoiceBox - basically a drop down menu

import javafx.scene.control.ChoiceBox; //new

public class ChoiceBoxDropDownMenu extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

window = primaryStage;

window.setTitle("Drop Down List");

button = new Button("Click Me");

//Choice Box

ChoiceBox<String> choiceBox = new ChoiceBox<>();

//getItems returns ObservableList object which you can add items to

choiceBox.getItems().add("Apples"); //adds one item to the list

choiceBox.getItems().addAll("Oranges", "Grapes", "Bananas"); //adds a bunch of items to the list

choiceBox.setValue("Apples"); //Set the default value (set it to one that already exists)

button.setOnAction(e -> getChoice(choiceBox));

…

}

//to get the value of the item selected

private void getChoice(ChoiceBox<String> choiceBox) {

String food = choiceBox.getValue(); //gets the value of the choice selected

System.out.println(food);

}

}

### 11. CheckBox

* + CheckBox - a list of options with check boxes by each item

import javafx.scene.control.CheckBox; //new

public class CheckBoxExample extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

window = primaryStage;

window.setTitle("Subs");

//CheckBox

CheckBox box1 = new CheckBox("Bacon"); //appears right of the check box

CheckBox box2 = new CheckBox("Tuna");

box2.setSelected(true); //automatically selected by default

//Button

button = new Button("Order Me!");

button.setOnAction(e -> handleOptions(box1, box2));

//Layout

VBox layout = new VBox(10);

layout.setPadding(new Insets(20, 20, 20, 20));

layout.getChildren().addAll(box1, box2, button);

scene = new Scene(layout, 300, 250);

window.setScene(scene);

window.show();

}

//handle check box options

private void handleOptions(CheckBox box1, CheckBox box2) {

String message = "Users Order: \n";

if(box1.isSelected()) { //isSelected returns true if selected

message += "Bacon\n";

}

if(box2.isSelected()) { //isSelected returns true if selected

message += "Tune\n";

}

System.out.println(message);

}

}

### 10. Extract and Validate Input

public class ExtractandValidateInput extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

window = primaryStage;

window.setTitle("GridPane");

//Form

label = new Label("UserAge: ");

TextField ageInput = new TextField();

button = new Button("Click Me");

button.setOnAction(e -> isInt(ageInput, ageInput.getText())); //when you click on the button it calls the the isInt method

//Layout

VBox layout = new VBox(10);

layout.setPadding(new Insets(20, 20, 20, 20));

layout.getChildren().addAll(label, ageInput, button);

scene = new Scene(layout, 300, 250);

window.setScene(scene);

window.show();

}

private boolean isInt(TextField ageInput, String message) { //passes in values

try {

int age = Integer.parseInt(ageInput.getText()); //this converts the text into an integer

System.out.println("User is: " + age);

return true; //returns true if ageInput is an integer

}catch(NumberFormatException e){ //finds the error of the try

System.out.println("Error Wrong " + message + " is not a number");

return false; //returns false if ageInput is not an integer

}

}

…

}

### 9. GridPane

* + GridPane - organizes the layout of the app in a grid

import javafx.geometry.Insets; //new

import javafx.scene.layout.GridPane; //new

public class GridPaneExample extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

window = primaryStage;

window.setTitle("GridPane");

GridPane grid = new GridPane();

grid.setPadding(new Insets(10, 10, 10, 10)); //sets the padding around the edges in pixels

grid.setVgap(8); //vertical spacing of individual cells

grid.setHgap(10); //Horizontal spacing of individual cells

//name label

Label nameLabel = new Label("Username: ");

GridPane.setConstraints(nameLabel, 0, 0); //sets the label variable at column 0 and row 0

//name Input

TextField nameInput = new TextField("Userlog"); //the textfield will have text already set

GridPane.setConstraints(nameInput, 1, 0); //sets the textfield variable at column 1 and row 0

//password label

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

GridPane.setConstraints(passwordLabel, 0, 1); //sets the label variable at column 0 and row 1

//name Input

TextField passwordInput = new TextField();

passwordInput.setPromptText("Password"); //sets the password input prompt in light gray

GridPane.setConstraints(passwordInput, 1, 1); //sets the textfield variable at column 1 and row 1

Button loginButton = new Button("Log In");

GridPane.setConstraints(loginButton, 1, 2); //sets the button variable at column 1 and row 2

grid.getChildren().addAll(nameLabel, nameInput, passwordLabel, passwordInput, loginButton);

Scene scene = new Scene(grid, 300, 250);

window.setScene(scene);

window.show();

}

}

### 8. Embedding Layouts

import javafx.scene.layout.BorderPane; //new

import javafx.scene.layout.HBox; //new

public class EmbeddingLayouts extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

window = primaryStage;

window.setTitle("Embedding Layouts");

HBox topMenu = new HBox(); //horizontal button box

Button buttonA = new Button("A");

Button buttonB = new Button("B");

Button buttonC = new Button("C");

topMenu.getChildren().addAll(buttonA, buttonB, buttonC); //adds all of these buttons

VBox leftMenu = new VBox(); //vertical button box

Button buttonD = new Button("D");

Button buttonE = new Button("E");

Button buttonF = new Button("F");

leftMenu.getChildren().addAll(buttonD, buttonE, buttonF); //adds all of these buttons

BorderPane borderPane = new BorderPane();

borderPane.setTop(topMenu); //sets the entire topMenu

borderPane.setLeft(leftMenu); //sets the entire leftMenu

Scene scene = new Scene(borderPane, 300, 250);

window.setScene(scene);

window.show();

}

}

### 7. Closing the Program Properly

public class ClosingtheProgramProperly extends Application{

…

@Override

public void start(Stage primaryStage) throws Exception {

…

window.setOnCloseRequest(e -> {

e.consume(); //this will consume the event instead of letting it close

closeProgram();

}); //whenever you click the X icon

button = new Button("Close Program");

button.setOnAction(e -> closeProgram()); //whenever you click the Close Program Button, it will call the closeProgram method

…

}

private void closeProgram() { //closeProgram method that calls the ConfirmBox class to get the boolean value

Boolean answer = ConfirmBox.display(null, "Are you sure?");

if(answer) { //if answer is true

System.out.println("File is saved");

window.close();

}

}

}

### 6. Communicating between windows

public class CommunicatingBetweenWindows extends Application{

…

button.setOnAction(e -> {

boolean result = ConfirmBox.display("Question:", "Are you studying GUI?"); //Gets the return value of ConfirmBox display method

System.out.println(result);

});

…

}

public class ConfirmBox {

static boolean answer;

public static boolean display(String title, String message) {

…

//create 2 buttons

Button yesButton = new Button("Yes");

Button noButton = new Button("No");

yesButton.setOnAction(e -> {

answer = true;

window.close();

});

noButton.setOnAction(e -> {

answer = false;

window.close();

});

…

return answer; //returns the value of the answer

}

}

### 5. Creating Alert Boxes

public class CreatingAlertBoxes extends Application{

…

button.setText("Click Me");

button.setOnAction(e -> AlertBox.display("The Alert Box", "You found the Alert Box!")); //Calls the AlertBox class to display and new window. Passes over the title and message.

…

}

import javafx.geometry.Pos; //new

import javafx.stage.Modality; //new

public class AlertBox { //only one method

public static void display(String title, String message) {

Stage window = new Stage();

window.initModality(Modality.APPLICATION\_MODAL); //doesn't allow you switch tabs without closing this one

window.setTitle(title);

window.setMinWidth(250); //minimum width of the program

Label label = new Label();

label.setText(message);

Button closeButton = new Button("Close the Window");

closeButton.setOnAction(e -> window.close());

VBox layout = new VBox(10);

layout.getChildren().addAll(label, closeButton);

layout.setAlignment(Pos.CENTER);

Scene scene = new Scene(layout);

window.setScene(scene);

window.showAndWait();

}

}

### 4. Switching Scenes

* + Label - a little chunk of static text

import javafx.scene.control.Label; //new

import javafx.scene.layout.VBox; //new

public class SwitchingScenes extends Application {

Stage window;

Scene scene1, scene2; //there are 2 scenes (or 2 pages)

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage primaryStage) throws Exception {

window = primaryStage;

Label label1 = new Label("Welcome to the first scene!");

Button button1 = new Button("Go to scene 2");

button1.setOnAction(e -> window.setScene(scene2)); //switches to scene2 when the button is pressed

//Layout 1 - children are laid out in a vertical column

VBox layout1 = new VBox(20); //spaced out 20 pixels vertically

layout1.getChildren().addAll(label1, button1);

scene1 = new Scene(layout1, 200, 200);

//button2

Button button2 = new Button("Go to scene 1");

button2.setOnAction(e -> window.setScene(scene1));

//layout 2

StackPane layout2 = new StackPane();

layout2.getChildren().add(button2);

scene2 = new Scene(layout2, 300, 300);

window.setScene(scene1);

window.setTitle("Scenes");

window.show();

}

}

### 3. Anonymous Inner Classes and Lambda Expressions

* + Anonymous Inner class button - A more confined button but it is only one button

= new Button();

button.setText("Click Me");

button.setOnAction(new EventHandler<ActionEvent>(){ //A more compact button method used just one time

@Override

public void handle(ActionEvent event) {

System.out.println("I am Anonymous inner class");

}

});

* + Lambda button - A very confined button that many can be made

button = new Button();

button.setText("Click Me");

button.setOnAction(e -> {System.out.println("This is a lambda expression");});

### 2. Handle User Events

* + Implements EventHandler<ActionEvent> is a simple button click. Afterward, you need the handle method.
    - There are many different events like touch, press, click, enter, etc…
  + Event - Whenever a button is clicked

import javafx.event.ActionEvent; //new

import javafx.event.EventHandler; //new

public class HandleUserEvents extends Application implements EventHandler<ActionEvent>{

Button button;

…

@Override

public void start(Stage primaryStage) throws Exception {

primaryStage.setTitle("Basic\_Window");

button = new Button();

button.setText("Click Me");

button.setOnAction(this); //whenever they click this button that code to handle this is in this class. Basically look in this class for the handle method

StackPane layout = new StackPane();

layout.getChildren().add(button);

…

}

@Override

public void handle(ActionEvent event) {

if(event.getSource() == button) { //if the button is clicked it will sysout “You clicked me!”

System.out.println("You clicked me!");

}

}

}

### 1. Creating a Basic Window (and basic terms)

* + JavaFx imports used in the course

| import javafx.application.Application; |
| --- |
| import javafx.beans.property.IntegerProperty; |
| import javafx.beans.property.SimpleIntegerProperty; |
| import javafx.beans.property.SimpleStringProperty; |
| import javafx.beans.property.StringProperty; |
| import javafx.collections.FXCollections; |
| import javafx.collections.ObservableList; |
| import javafx.event.ActionEvent; |
| import javafx.event.EventHandler; |
| import javafx.geometry.Insets; |
| import javafx.geometry.Pos; |
| import javafx.scene.control.Button; |
| import javafx.scene.control.cell.PropertyValueFactory; |
| import javafx.scene.control.CheckMenuItem; |
| import javafx.scene.control.ChoiceBox; |
| import javafx.scene.control.ComboBox; |
| import javafx.scene.control.Label; |
| import javafx.scene.control.ListView; |
| import javafx.scene.control.Menu; |
| import javafx.scene.control.MenuBar; |
| import javafx.scene.control.MenuItem; |
| import javafx.scene.control.RadioMenuItem; |
| import javafx.scene.control.SeparatorMenuItem; |
| import javafx.scene.control.TableColumn; |
| import javafx.scene.control.TableView; |
| import javafx.scene.control.ToggleGroup; |
| import javafx.scene.control.TreeItem; |
| import javafx.scene.control.TreeView; |
| import javafx.scene.layout.BorderPane; |
| import javafx.scene.layout.GridPane; |
| import javafx.scene.layout.HBox; |
| import javafx.scene.layout.StackPane; |
| import javafx.scene.layout.VBox; |
| import javafx.scene.Scene; |
| import javafx.stage.Modality; |
| import javafx.stage.Stage; |

* + Your class must extend Application. Afterward, you need the start method to launch. You then call launch(args) in main.
  + Stage -the window displayed
  + Scene - the content inside the stage
  + Layout - how you want everything arranged on the screen

import javafx.application.Application; //new

import javafx.scene.Scene; //new

import javafx.scene.control.Button; //new

import javafx.scene.layout.StackPane; //new

import javafx.stage.Stage; //new

public class BasicWindow extends Application{

Button button;

public static void main(String[] args) {

launch(args); //calls the start method

}

@Override

public void start(Stage primaryStage) throws Exception {

primaryStage.setTitle("Basic\_Window"); //sets title of the main window

button = new Button();

button.setText("Click Me"); //sets button text

StackPane layout = new StackPane(); //this will position the button in the middle (default)

layout.getChildren().add(button);

Scene scene = new Scene(layout, 300, 250); //sets the scene size 300 by 250

primaryStage.setScene(scene); //use this scene for the window of the program

primaryStage.show(); //displays it to the user

}

}