17.1 Introduction to graphical user interfaces with JavaFX

This section has been set as optional by your instructor.

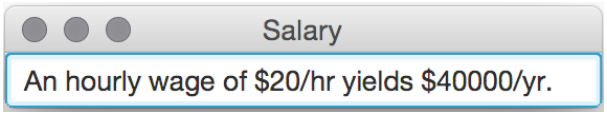
JavaFX is a set of packages and APIs for developing programs with graphical user interfaces, 3D graphics, etc. A ***graphical user interface***, or ***GUI***, enables the user to interface with a program using graphical components, such as windows, buttons, text boxes, etc., as opposed to text-based interfaces like the command line. The following program calculates a yearly salary based on an hourly wage and utilizes JavaFX GUI components to display the program's output.

©zyBooks 12/08/22 21:39 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 17.1.1: Displaying a yearly salary using a GUI.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.control.TextField;

public class SalaryGuiFx extends Application {
   @Override
   public void start(Stage applicationStage) {
      int hourlyWage;
      int yearlySalary;
      Scene scene = null;                  // Scene contains all content
      Pane pane = null;                    // Positions components within
scene
      TextField outputField = null;        // Displays output salary

      pane = new Pane();                   // Create an empty pane
      scene = new Scene(pane);             // Create a scene containing
the pane

      // Calculate yearly salary
      hourlyWage = 20;
      yearlySalary = hourlyWage * 40 * 50;

      // Create text field and display program output using the text
field
      outputField = new TextField();
      outputField.setText("An hourly wage of $" + hourlyWage + "/hr " +
                          "yields $" + yearlySalary + "/yr.");
      outputField.setEditable(false);      // Prevent user from editing
text
      outputField.setPrefColumnCount(22);

      pane.getChildren().add(outputField); // Add text field to pane

      applicationStage.setScene(scene);    // Set window's scene
      applicationStage.setTitle("Salary"); // Set window's title
      applicationStage.show();             // Display window
   }

   public static void main(String [] args) {
      launch(args); // Launch application
   }
}
```


Screenshot:



A JavaFX GUI uses four classes/objects, namely Application, Stage, Scene, and Pane, to display graphical components. The following outlines one approach to create a JavaFX GUI, using the SalaryGuiFx class as the example application.

1. *Extend and launch the application:* An ***Application*** is a JavaFX class that provides the basic functionality for a JavaFX program and is available via the import statement `javafx.application.Application;`. The SalaryGuiFx class is derived from the Application class by appending `extends Application` after SalaryGuiFx in the class definition, as in `class SalaryGuiFx extends Application`. The SalaryGuiFx class inherits the functionality of the Application class, so that SalaryGuiFx can display a GUI. The concept of class inheritance is explained in more detail elsewhere.

   The main() method calls the launch() method using the statement `launch(args);`. The launch() method creates a SalaryGuiFx object and calls the SalaryGuiFx object's start() method.

2. *Override the start() method:* A JavaFX Application starts by executing the start() method, which must be overridden in the derived Application class. The start() method takes a Stage parameter, has a return type of void, as in `public void start(Stage applicationStage) {...}`, and is preceded by the annotation `@Override`. A ***Stage*** is a JavaFX top-level container that contains all content within a window and is available via the import statement `import javafx.stage.Stage;`.

3. *Create a pane and scene:* A ***Scene*** is a JavaFX component that contains all graphical components that are displayed together and is available via the import statement `import javafx.scene.Scene;`. An application can have multiple scenes, but only one scene may be visible at a time. A ***Pane*** is a JavaFX component that controls the layout, i.e., position and size, of graphical components and is available via the import statement `import javafx.scene.layout.Pane;`. The statement `pane = new Pane();` creates an empty Pane object. The statement `scene = new Scene(pane);` creates a new Scene containing the pane object.

4. *Create and add graphical components to a pane:* A ***TextField*** is a JavaFX GUI component that enables a programmer to display a line of text and is available via the import statement `import javafx.scene.control.TextField;`. The statement `outputField = new TextField();` creates a TextField object. A TextField's setText() method specifies the text that will be displayed. Ex: `outputField.setText("An hourly ... ");`. By default, a TextField allows users to modify the text for the purposes of input (discussed elsewhere). A program can use TextField's setEditable() method with an argument of false to prevent users from editing the text, as in `outputField.setEditable(false);`. A TextField's width can be set using the setPrefColumnCount() method. Ex: `outputField.setPrefColumnCount(22)` sets the width to 22 columns.

Graphical components are added to a scene by adding the components to the scene's pane. A pane can contain numerous graphical components, which are called children. The statement `pane.getChildren().add(outputField);` adds a TextField object named outputField to the pane's list of children.

5. *Set and display scene:* Stage's setScene() method sets the scene that will be displayed, as in `applicationStage.setScene(scene);`. The setTitle() method specifies the text that will be displayed as the application's title. Ex: `applicationStage.setTitle("Salary");` displays "Salary" in the application's title bar. Stage's show() method makes the stage visible, which displays the application's window to the user. Ex: `applicationStage.show();` displays the application's window with the title "Salary" and text "An hourly wage of $20/hr yields $40000/yr."

PARTICIPATION ACTIVITY 17.1.1: Using JavaFX GUI components.

1) Write a statement that sets the text of a TextField object nameField to "Mary".

Check Show answer

2) Given a Stage object named appStage, write a statement that sets the stage's title to "Employees".

Check Show answer

3) Given a Pane object appPane and a TextField object nameField, write a statement that adds nameField to the pane.

```
appPane.getChildren().
;
```

Check Show answer

4) Given a Stage object named

appStage, write a statement that makes the stage visible.

Check Show answer

Exploring further:

- JavaFX overview, tutorials, and references from Oracle's Java Documentation
- JavaFX Application class from Oracle's Java Documentation
- JavaFX Stage class from Oracle's Java Documentation
- JavaFX Scene class from Oracle's Java Documentation
- JavaFX Pane class from Oracle's Java Documentation
- JavaFX TextField class from Oracle's Java Documentation

17.2 Positioning GUI components using a GridPane

This section has been set as optional by your instructor.

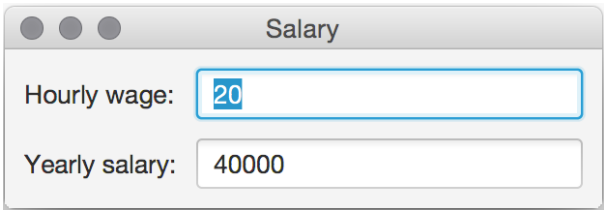
A ***GridPane*** is a JavaFX Pane component that positions graphical components in a two-dimensional grid. The following program demonstrates the use of a GridPane to position graphical components in a GUI that displays an hourly wage and the associated yearly salary.

Figure 17.2.1: Using a GridPane to arrange graphical components.

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;

public class SalaryLabelGuiFx extends Application {
   @Override
   public void start(Stage applicationStage) {
      int hourlyWage;
      int yearlySalary;
      Scene scene = null;          // Scene contains all content
      GridPane gridPane = null;    // Positions components within scene
      Label wageLabel = null;      // Label for hourly salary
      Label salaryLabel = null;    // Label for yearly salary
      TextField salField = null;   // Displays yearly salary
      TextField wageField = null; // Displays hourly wage
      Insets gridPadding = null;

      gridPane = new GridPane();    // Create an empty pane
      scene = new Scene(gridPane); // Create scene containing the grid
pane

      // Calculate yearly salary
      hourlyWage = 20;
      yearlySalary = hourlyWage * 40 * 50;

      // Set hourly and yearly salary
      wageLabel = new Label("Hourly wage:");
      salaryLabel = new Label("Yearly salary:");

      // Create wage and salary text fields
      wageField = new TextField();
      wageField.setPrefColumnCount(15);
      wageField.setEditable(false);
      wageField.setText(Integer.toString(hourlyWage));

      salField = new TextField();
      salField.setPrefColumnCount(15);
      salField.setEditable(false);
      salField.setText(Integer.toString(yearlySalary));

      gridPane.add(wageLabel, 0, 0);    // Add wage label to location (0,
0)
      gridPane.add(wageField, 1, 0);    // Add wage text field to location
(1, 0)
      gridPane.add(salaryLabel, 0, 1); // Add salary label to location
(0, 1)
      gridPane.add(salField, 1, 1);     // Add salary text field to
location (1, 1)
```

```
        gridPadding = new Insets(10, 10, 10, 10); // Padding values for
top, right, bottom, and left
        gridPane.setPadding(gridPadding);         // Set padding around
grid
        gridPane.setHgap(10);                     // Spacing between
columns
        gridPane.setVgap(10);                     // Spacing between rows

        applicationStage.setScene(scene);     // Set window's scene
        applicationStage.setTitle("Salary"); // Set window's title
        applicationStage.show();              // Display window
    }

    public static void main(String [] args) {
        launch(args); // Launch application
    }
}
```


Screenshot:



A ***Label*** is a JavaFX component that displays non-editable text and is available via the import statement `import javafx.scene.control.Label;`. Labels are typically for describing, or labeling, other GUI components. For example, the SalaryLabelGuiFx program uses two Labels, wageLabel and salaryLabel, to describe the contents of the wage and salary text fields, respectively.

The statement `wageLabel = new Label("Hourly wage:");` creates a Label object with the string "Hourly wage:" as the Label's displayed text. A program can also use Label's setText() method to set the label's text. Ex: `wageLabel.setText("Hourly wage:");`

PARTICIPATION ACTIVITY 17.2.1: Using a JavaFX Label component.

1) Write a statement to create a new Label object called nameLabel with the text "Name:".

Check Show answer

2) Given the Label creation statement

`Label passwordLabel = new Label();`, write a statement that sets passwordLabel's text to "Password:".

Check Show answer

A GridPane allows programmers to set the location of graphical components within a two-dimensional grid. Each location of the grid is indexed using one number for the column and another number for the row. The top-left location of the grid has column and row indices of (0, 0). The indices of other locations are specified relative to the top-left location, with increasing column indices going right and increasing row indices going down.

PARTICIPATION ACTIVITY 17.2.2: Specifying layouts for GUI components.

Animation captions:

1. Create a GridPane object to position graphical components in a two-dimensional grid. Add the GridPane to the application's Scene.
2. Create graphical components for the GUI.
3. Add each component to the GridPane at the specified column and row.

PARTICIPATION ACTIVITY 17.2.3: Adding components to a GridPane.

Given the following code that creates a GridPane and several graphical components:

```
GridPane gridPane = new GridPane();
Scene scene = new Scene(gridPane);

Label cityLabel = new Label("City:");
TextField cityField = new TextField();

Label stateLabel = new Label("State:");
TextField stateField = new TextField();
```


1) Write a statement that adds cityLabel to the top-left location of the gridPane.

Check Show answer

2) Write a statement that adds cityField to the grid location just right of cityLabel.

Check Show answer

3) Write a statement that adds stateLabel to the grid location just below cityLabel.

Check Show answer

4) Write a statement that adds stateField to the grid location corresponding to column 1 and row 1.

Check Show answer

A GridPane's setPadding() method specifies the spacing, or padding, between the outer edges of the grid and the window. The statement `gridPadding = new Insets(10, 10, 10, 10);` first creates an Insets object with the four arguments for the top, right, bottom, and left padding, respectively, in pixels. Then, the statement `gridPane.setPadding(gridPadding);` applies the padding to the gridPane. Insets is available via the import statement `import javafx.geometry.Insets;`.

A GridPane's setHgap() and setVgap() methods specify the padding between columns (horizontal gap) and rows (vertical gap) respectively. Ex: `gridPane.setHgap(10);` sets the padding between columns to 10 pixels.

PARTICIPATION ACTIVITY 17.2.4: Applying padding to GridPane.

Given a GridPane Object named gridPane:

1) Write a statement that creates an Insets object named gridPadding such that the subsequent statement `gridPane.setPadding(gridPadding);` applies a padding of 15 pixels above and below the grid and a padding of 5 pixels to the left and right of the grid.

```
Insets gridPadding = new Insets(
                     );
```

Check Show answer

2) Write a statement that sets the grid's horizontal gap to 3 pixels.

Check Show answer

3) Write a statement that sets the grid's vertical gap to 8 pixels.

Check Show answer

Exploring further:

- JavaFX overview, tutorials, and references from Oracle's Java Documentation
- JavaFX GridPane class from Oracle's Java Documentation
- JavaFX Label class from Oracle's Java Documentation
- JavaFX Insets class from Oracle's Java Documentation

©zyBooks 12/08/22 21:39 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

17.3 Input and event handlers

This section has been set as optional by your instructor.

A ***Button*** is a JavaFX GUI component that represents a labeled button that a user can press to interact with a program. A JavaFX GUI component that supports user input generates an ***action event*** to notify the program when a user interacts with the component, such as when pressing a button. An ***event handler*** defines how the program should respond to action events. The following GUI uses a text field to enable the user to enter an hourly wage as an input for the calculation of a yearly salary, which is triggered by a button press.

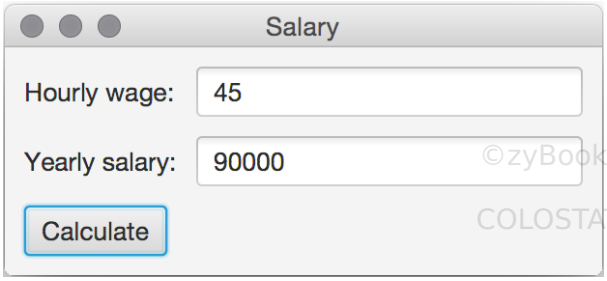
©zyBooks 12/08/22 21:39 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 17.3.1: Using a Button to trigger a yearly salary calculation.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.GridPane;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.geometry.Insets;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;

public class SalaryCalcButtonGuiFx extends Application {
   private Label wageLabel;     // Label for hourly salary
   private Label salLabel;      // Label for yearly salary
   private TextField salField;  // Displays hourly salary
   private TextField wageField; // Displays yearly salary
   private Button calcButton;   // Triggers salary calculation

   @Override
   public void start(Stage applicationStage) {
      Scene scene = null;         // Scene contains all content
      GridPane gridPane = null;   // Positions components within scene

      gridPane = new GridPane();   // Create an empty pane
      scene = new Scene(gridPane); // Create scene containing the grid
pane

      // Set hourly and yearly salary
      wageLabel = new Label("Hourly wage:");
      salLabel = new Label("Yearly salary:");

      wageField = new TextField();
      wageField.setPrefColumnCount(15);
      wageField.setEditable(true);
      wageField.setText("0");

      salField = new TextField();
      salField.setPrefColumnCount(15);
      salField.setEditable(false);

      // Create a "Calculate" button
      calcButton = new Button("Calculate");

      gridPane.setPadding(new Insets(10, 10, 10, 10)); // Padding around
grid
      gridPane.setHgap(10);                            // Spacing between
columns
      gridPane.setVgap(10);                            // Spacing between
rows

      gridPane.add(wageLabel, 0, 0);  // Add wage label to location (0,
```

```
0)
      gridPane.add(wageField, 1, 0);  // Add wage text field to location
(1, 0)
      gridPane.add(salLabel, 0, 1);   // Add salary label to location (0,
1)
      gridPane.add(salField, 1, 1);   // Add salary text field to
location (1, 1)
      gridPane.add(calcButton, 0, 2); // Add calculate button to location
(0, 2)

      // Set an event handler to handle button presses
      calcButton.setOnAction(new EventHandler<ActionEvent>() {
         /* Method is automatically called when an event
            occurs (e.g, button is pressed) */
         @Override
         public void handle(ActionEvent event) {
            String userInput;
            int hourlyWage;
            int yearlySalary;

            // Get user's wage input and calculate yearly salary
            userInput = wageField.getText();
            hourlyWage = Integer.parseInt(userInput);
            yearlySalary = hourlyWage * 40 * 50;

            // Display calculated salary
            salField.setText(Integer.toString(yearlySalary));
         }
      });

      applicationStage.setScene(scene);    // Set window's scene
      applicationStage.setTitle("Salary"); // Set window's title
      applicationStage.show();             // Display window
   }

   public static void main(String [] args) {
      launch(args); // Launch application
   }
}
```


Screenshot:



The GUI enables user input by making the text displayed by the TextField wageField editable. Ex: `wageField.setEditable(true);` allows the user to enter an hourly wage value. TextField's getText() method returns the TextField's text, allowing the program to get the user's input.

The user triggers the yearly salary calculation by pressing the button labeled "Calculate". The following outlines the approach used in the SalaryCalcButtonGuiFx class to create a JavaFX GUI that handles button presses .

1. *Create and add a button*: The statement `calcButton = new Button("Calculate");` creates a Button object with the string "Calculate" as the Button's label. The program then adds the Button to the GridPane, as in `gridPane.add(calcButton, 0, 2);`.The Button class is available via the import statement `import javafx.scene.control.Button;`.
2. *Set and define an event handler*: An ***ActionEvent*** is an object that notifies the program of the occurrence of a component-related event, such as a user pressing a button, and is available via the import statement `import javafx.event.ActionEvent;`. An ***EventHandler*** is an object that defines how a program should respond to specific events, such as an ActionEvent, and is available via the import statement `import javafx.event.EventHandler;`. Ex: The EventHandler defined for calcButton in SalaryCalcButtonGuiFx calculates and displays a yearly salary whenever the user presses the button.

   A program specifies a Button's EventHandler by calling Button's setOnAction() method with an EventHandler object as an argument. SalaryCalcButtonGuiFx defines an EventHandler using an advanced concept known as an anonymous class, which combines both class declaration and instantiation for conciseness. The following code can be used as a template to create and set a Button's EventHandler.

Figure 17.3.2: Template for creating and setting a JavaFX Button's EventHandler.

```
// Set an event handler to handle button presses
buttonObject.setOnAction(new
EventHandler<ActionEvent>() {
   /* Method is automatically called when an event
      occurs (e.g., button is pressed) */
   @Override
   public void handle(ActionEvent event) {

      // Write event handling instructions here
   }
});
```


The highlighted lines can be modified to create a custom EventHandler for any Button as follows:

1. Specifying an EventHandler for a different Button is done by replacing buttonObject with the name of another Button object. Ex: `otherButton.setOnAction(...);`.
2. The instructions for responding to ActionEvents are written inside EventHandler's handle() method. The EventHandler can access the EventHandler's enclosing class' fields and

methods. Ex: calcButton's EventHandler in the SalaryCalcButtonGuiFx program can access the class' field wageField to get the hourly wage. However, an EventHandler cannot access local variables or objects declared in the enclosing method, such as local variables in the start() method.

PARTICIPATION ACTIVITY 17.3.1: User input with TextField and Button.

Complete the code to achieve the stated goal.

1) Create a Button called goButton with the label "Go!".

```
Button goButton =
                  ;
```

Check Show answer

2) Make the TextField voltageField editable by the user.

```
voltageField.
                  ;
```

Check Show answer

3) Set an EventHandler for a Button called startButton.

```
                  
(new
EventHandler<ActionEvent>()
{
   /* Method is
automatically called when
an event
      occurs (e.g, button
is pressed) */
   @Override
   public void
handle(ActionEvent event) {
      // ...
   }
});
```

Check Show answer

4) Get user input from an editable

TextField dateField and assign the text to a String dateStr when scheduleButton pressed.

```
scheduleButton.setOnAction(new
EventHandler<ActionEvent>() {
   @Override
   public void
handle(ActionEvent event) {
      String dateStr = "";
      dateStr =
                              ;
   }
});
```

Check Show answer

Programs that get user input commonly check the input's value to ensure the input's validity. If the input is invalid, meaning the input is improperly formatted or falls outside the expected range, the program should report an alert to the user. The SalaryCalcButtonGuiFx program allows the user to enter any value, positive or negative, in the TextField wageField. Because a negative wage is not valid, the following program improves upon the SalaryCalcButtonGuiFx program by displaying an alert message if the user enters a negative wage.

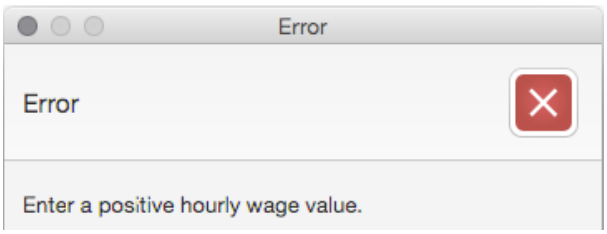
©zyBooks 12/08/22 21:39 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 17.3.3: Displaying an Alert for invalid wage inputs.

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;

public class SalaryCalcButtonErrorAlertGuiFx extends Application {
   private Label wageLabel;     // Label for hourly salary
   private Label salLabel;      // Label for yearly salary
   private TextField salField;  // Displays hourly salary
   private TextField wageField; // Displays yearly salary
   private Button calcButton;   // Triggers salary calculation

   @Override
   public void start(Stage applicationStage) {
      Scene scene = null;         // Scene contains all content
      GridPane gridPane = null;   // Positions components within scene

      gridPane = new GridPane();   // Create an empty pane
      scene = new Scene(gridPane); // Create scene containing the grid
pane

      // Set hourly and yearly salary
      wageLabel = new Label("Hourly wage:");
      salLabel = new Label("Yearly salary:");

      wageField = new TextField();
      wageField.setPrefColumnCount(15);
      wageField.setEditable(true);
      wageField.setText("0");

      salField = new TextField();
      salField.setPrefColumnCount(15);
      salField.setEditable(false);

      // Create a "Calculate" button
      calcButton = new Button("Calculate");

      gridPane.setPadding(new Insets(10, 10, 10, 10)); // Padding around
grid
      gridPane.setHgap(10);                            // Spacing between
columns
      gridPane.setVgap(10);                            // Spacing between
rows

      gridPane.add(wageLabel, 0, 0);  // Add wage label to location (0,
```

```
0)
      gridPane.add(wageField, 1, 0);  // Add wage text field to location
(1, 0)
      gridPane.add(salLabel, 0, 1);   // Add salary label to location (0,
1)
      gridPane.add(salField, 1, 1);   // Add salary text field to
location (1, 1)
      gridPane.add(calcButton, 0, 2); // Add calculate button to location
(0, 2)

      // Set an event handler to handle button presses
      calcButton.setOnAction(new EventHandler<ActionEvent>() {
         /* Method is automatically called when an event
            occurs (e.g, button is pressed) */
         @Override
         public void handle(ActionEvent event) {
            String userInput;
            int hourlyWage;
            int yearlySalary;

            // Get user's wage input and calculate yearly salary
            userInput = wageField.getText();
            hourlyWage = Integer.parseInt(userInput);
            yearlySalary = hourlyWage * 40 * 50;

            if (hourlyWage >= 0) {
               // Display calculated salary
               salField.setText(Integer.toString(yearlySalary));
            }
            else {
               // Display an alert dialog
               Alert alert = new Alert(AlertType.ERROR,
                                       "Enter a positive hourly wage
value.");
               alert.showAndWait();
            }
         }
      });

      applicationStage.setScene(scene);    // Set window's scene
      applicationStage.setTitle("Salary"); // Set window's title
      applicationStage.show();             // Display window
   }

   public static void main(String [] args) {
      launch(args); // Launch application
   }
}
```


Screenshot:



An ***Alert*** is a separate JavaFX window, also known as a dialog or pop-up window, that displays a message to the user. Ex:

```
Alert alert = new Alert(AlertType.ERROR, "Enter a positive hourly wage v
```

creates an Alert object that displays an error message with the text "Enter a positive hourly wage value.". The first argument specifies the Alert's type. Ex: `AlertType.ERROR` specifies that the Alert window should indicate an error.

Alert's showAndWait() method makes the Alert visible to the user and waits for the user's response. The program resumes execution after the user presses the Alert's "OK" button, which closes the Alert window.

An Alert object can display a variety of Alert types, of which some common types are summarized below:

Table 17.3.1: Summary of common Alert types.

Type	Description	Documentation
AlertType.NONE	Configures the Alert to display a basic message.	AlertType.NONE from Oracle's Java Documentation
AlertType.ERROR	Configures the Alert to display an error or failure with an option to confirm.	AlertType.ERROR from Oracle's Java Documentation
AlertType.CONFIRMATION	Configures the Alert to seek confirmation from the user. The displayed message is typically a question with the option to confirm or cancel.	AlertType.CONFIRMATION from Oracle's Java Documentation
AlertType.INFORMATION	Configures the Alert to display an informative message with the option to confirm.	AlertType.INFORMATION from Oracle's Java Documentation
AlertType.WARNING	Configures the Alert to display a message that looks like a warning with an option to confirm.	AlertType.WARNING from Oracle's Java Documentation

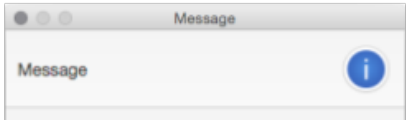
PARTICIPATION ACTIVITY 17.3.2: Common Alert types.

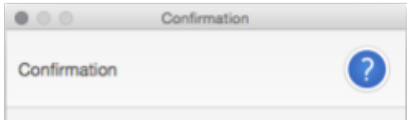
Match the Alert object with the corresponding Alert window.

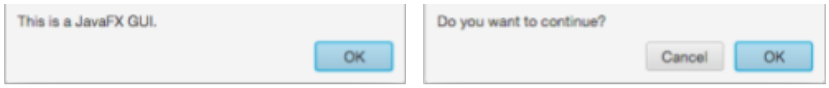
(a)

(b)

(c)







(d) (e)

If unable to drag and drop, refresh the page.

(d) (c) (e) (b) (a)


```
Alert alert = new Alert(AlertType.CONFIR
                        "Do you want to

Alert alert = new Alert(AlertType.INFORM
                        "This is a JavaF

Alert alert = new Alert(AlertType.WARNIN
                        "Use at your own

Alert alert = new Alert(AlertType.ERROR,
                        "Something went

Alert alert = new Alert(AlertType.NONE,
                        "Hello.");
```

Reset

Exploring further:

- JavaFX overview, tutorials, and references from Oracle's Java Documentation
- JavaFX Button class from Oracle's Java Documentation
- JavaFX EventHandler class from Oracle's Java Documentation
- JavaFX ActionEvent class from Oracle's Java Documentation
- JavaFX Alert class from Oracle's Java Documentation

17.4 Basic graphics with JavaFX

This section has been set as optional by your instructor.

JavaFX provides a set of objects for graphical applications. A ***graphical application*** is a program that displays drawings and other graphical objects. Graphical applications display their contents inside a Canvas object that is added to the JavaFX application.

Creating a class for a JavaFX application involves advanced topics, including defining a class and inheritance, which are discussed elsewhere. For now, the below class can be used as a template to create a JavaFX application to draw 2D graphics.

Figure 17.4.1: Template for creating a JavaFX application to draw 2D graphics.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;

public class EmptyCanvasFx extends Application {
   @Override
   public void start(Stage applicationStage) {
      Pane pane = new Pane();                    // Create an empty pane
      Scene scene = new Scene(pane);             // Create a scene
containing the pane
      Canvas canvas = new Canvas(400, 200);      // Create a canvas in
which to draw

      // Get the canvas' graphics context to draw
      GraphicsContext graphicsContext = canvas.getGraphicsContext2D();

      // Write your drawing instructions here

      pane.getChildren().add(canvas);            // Add canvas to pane
      applicationStage.setTitle("Empty canvas"); // Set window's title
      applicationStage.setScene(scene);          // Set window's scene
      applicationStage.show();                   // Display window
   }

   public static void main(String [] args) {
      launch(args); // Launch application
   }
}
```


The highlighted lines, which can be modified to create a custom application, operate as follows:

1. The code defines a class named EmptyCanvasFx that extends Application. An ***Application*** is a JavaFX class that provides the basic functionality for a JavaFX program. The class should be saved to a separate file named with the same name, EmptyCanvasFx.java.
2. A ***Canvas*** is an image onto which graphical objects can be drawn. A Canvas object is created with arguments for the canvas' width and height. Ex: The statement `Canvas canvas = new Canvas(400, 200);` creates a canvas object with a width of 400 pixels and a height of 200 pixels.
3. The statement `GraphicsContext graphicsContext = canvas.getGraphicsContext2D();` gets the canvas' GraphicsContext object. A ***GraphicContext*** is an object that supports drawing shapes on a Canvas.
4. A programmer draws on the canvas using the methods provided by the GraphicsContext object.
5. The title of the application is set using the setTitle() method of the application's Stage object, which contains all objects displayed with a JavaFX application. Ex: `applicationStage.setTitle("Empty canvas");` displays "Empty canvas" in the application's title bar.

PARTICIPATION ACTIVITY 17.4.1: Configuring a JavaFX application.

Select the statements to modify the provided JavaFX application template to implement the following:

1) The class name for the application should be SimpleDrawingAppFX.

- `public class SimpleDrawingAppFx {`
- `public class SimpleDrawingAppFx extends JavaFX {`
- `public class SimpleDrawingAppFx extends Application {`

2) The application should have the title "Simple drawing".

- `applicationStage.setTitle(Simple drawing);`
- `applicationStage.setTitle("Simple drawing");`

3) The application should create a Canvas object with a height of 800 pixels and a width of 300 pixels.

- `Canvas = new Canvas();`
- `Canvas = new Canvas(300, 800);`
- `Canvas = new Canvas(800, 300);`

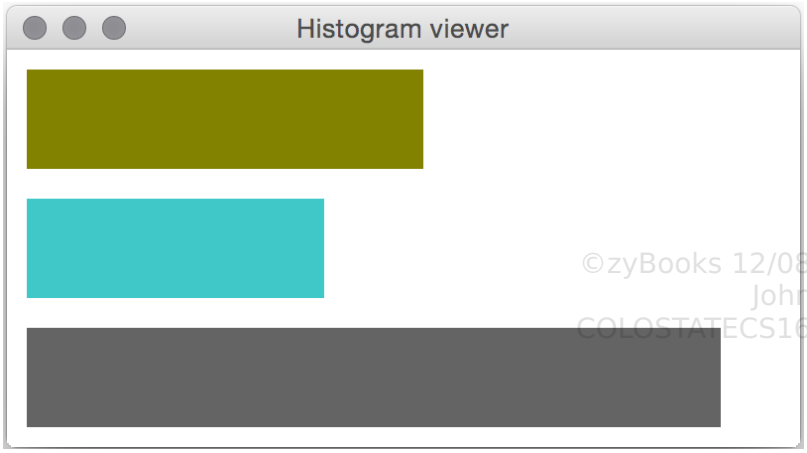
The following program modifies the provided template to draw a simple histogram using rectangles.

©zyBooks 12/08/22 21:39 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

Figure 17.4.2: Drawing a histogram.

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public class HistogramViewerFx extends Application {
   @Override
   public void start(Stage applicationStage) {
      Pane pane = new Pane();                    // Create an empty pane
      Scene scene = new Scene(pane);             // Create a scene
containing the pane
      Canvas canvas = new Canvas(400, 200);      // Create a canvas in
which to draw

      // Get the canvas' graphics context to draw
      GraphicsContext graphicsContext = canvas.getGraphicsContext2D();

      // Draw 1st bin as an olive colored rectangle at (10,10)
      // with width = 200 and height = 50
      Color binColor1 = Color.rgb(128, 128, 0);
      graphicsContext.setFill(binColor1);
      graphicsContext.fillRect(10, 10, 200, 50);

      // Draw 2nd bin as a teal blue rectangle at (10,75)
      // with width = 150 and height = 50
      Color binColor2 = Color.rgb(0, 200, 200);
      graphicsContext.setFill(binColor2);
      graphicsContext.fillRect(10, 75, 150, 50);

      // Draw 3rd bin as a gray rectangle at (10,140)
      // with width = 350 and height = 50
      Color binColor3 = Color.rgb(100, 100, 100);
      graphicsContext.setFill(binColor3);
      graphicsContext.fillRect(10, 140, 350, 50);

      pane.getChildren().add(canvas);                // Add canvas to
pane
      applicationStage.setTitle("Histogram viewer"); // Set window's
title
      applicationStage.setScene(scene);              // Set window's
scene
      applicationStage.show();                       // Display window
   }

   public static void main(String [] args) {
       launch(args); // Launch application
   }
}
```

Screenshot:



The HistogramViewerFX application uses the GraphicsContext and Color objects to draw a simple histogram with three bins, using the operations:

1. *Create a Color object:* A ***Color*** object represents a color. The Color.rgb() method constructs a Color object in the red, green, and blue colorspace. The method takes integer arguments between 0 to 255 for each color channel as specified by the method definition: `Color.rgb(int red, int green, int blue)`. Ex: The statement `Color binColor1 = new Color.rgb(128, 128, 0);` creates a Color object with an olive color.
2. *Set the fill color used by the GraphicsContext object:* GraphicsContext's setFill() method sets the color used for the interior of shapes drawn by the GraphicsContext object. Ex: `graphicsContext.setFill(binColor1);` sets the GraphicsContext's fill color to binColor1.
3. *Draw the shape:* A GraphicsContext object provides different methods for drawing shapes. The fillRect() method draws a rectangle, filling the interior of the rectangle with the GraphicsContext object's current fill color. The fillRect() methods' arguments include the location and size (in pixels) as specified by the method definition: `fillRect(double x, double y, double w, double h)`, where x, y is the location, w is the width, and h is the height.

PARTICIPATION ACTIVITY 17.4.2: Drawing a filled rectangle.

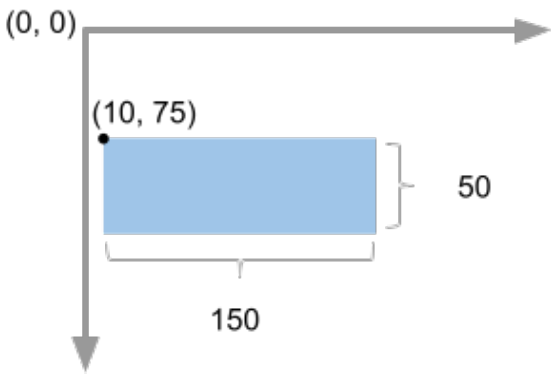
Animation captions:

1. Color.rgb() method creates a Color object representing an RGB color.
2. Set graphicContext's current fill Color.
3. Draw a filled rectangle with graphicContext's current fill color. The rectangle is drawn at coordinates (10, 75) with a width of 150 pixels and a height of 50 pixels.

Alternatively, a programmer can use the setStroke() and strokeRect() methods to draw an outline of a rectangle. setStroke() sets the color used to draw an outline of shapes, and strokeRect() draws an outline of a rectangle with the GraphicsContext's current stroke color.

The programmer needs to know the positioning coordinate system in order to draw shapes in the intended location. As the following figure illustrates, the top-left corner of a Canvas corresponds to coordinates (0, 0). The x-coordinate increases horizontally to the right and the y-coordinate increases vertically downward.

Figure 17.4.3: Canvas coordinate system.



PARTICIPATION ACTIVITY 17.4.3: Drawing colored rectangles.

Which code segment performs the described operation? Type A, B, or C.

A.
```
Color color = Color.rgb(0, 255, 0);
graphicsContext.setFill(color);
graphicsContext.fillRect(0, 0, 150, 100);
```

B.
```
Color color = Color.rgb(255, 0, 0);
graphicsContext.setFill(color);
graphicsContext.fillRect(0, 100, 200, 200);
```

C.
```
Color color = Color.rgb(255, 0, 255);
graphicsContext.setStroke(color);
graphicsContext.strokeRect(0, 100, 50, 150);
```


1) Draws a filled, red square.

Check Show answer

2) Draws the outline of a purple

rectangle 50 pixels wide and 150 pixels in height.

Check Show answer

3) Draws a rectangle whose top-left corner is located at the origin of the coordinate system.

Check Show answer

4) Draws a filled green rectangle.

Check Show answer

A GraphicsContext object can draw a variety of shapes, of which some common shapes are summarized below:

Table 17.4.1: Summary of common shapes for drawing.

Shape	Description	Documentation
Line	strokeLine() draws a line between two coordinate points.	strokeLine() from Oracle's Java Documentation
Rectangle	fillRect() draws a filled rectangle. strokeRect() draws an outline of a rectangle.	fillRect() and strokeRect() from Oracle's Java Documentation
Round rectangle	fillRoundRect() draws a filled rectangle with rounded corners. strokeRoundRect() draws an outline of a rectangle with rounded corners.	fillRoundRect() and strokeRoundRect() from Oracle's Java Documentation
Oval	fillOval() method draws an oval with programmer-specified width, height, and location. strokeOval() method draws an outline of an oval with programmer-specified width, height, and location.	fillOval() and strokeOval() from Oracle's Java Documentation
Polygon	fillPolygon() draws a filled polygon with programmer-specified boundary points. strokePolygon() draws an outline of a polygon with programmer-specified boundary points.	fillPolygon() and strokePolygon() from Oracle's Java Documentation

©zyBooks 12/08/22 21:39 1361995
John Farrell
COLOSTATECS165WakefieldFall2022