What does a GUI framework do?

• Provides ready made visible, interactive, customizable components
• So that, for instance, you don’t have to code your own “window”!

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Window
import javafx.stage.Stage;

public class JavaFXApplication1 extends Application {

    @Override
    public void start(Stage primaryStage) {
        Scene scene = new Scene(root, 300, 250);

        primaryStage.setTitle("Example Window");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```
JavaFX vs. Swing vs. AWT

• When Java was introduced, the GUI classes were bundled in a library known as the Abstract Windows Toolkit (AWT).
  • AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects.
  • In addition, AWT is prone to platform-specific bugs.

• The AWT user-interface components were replaced by a more robust, versatile, and flexible library known as Swing.

• Swing components depend less on the target platform and use less of the native GUI resource.

• With Java 8, Swing is replaced by a completely new GUI platform: JavaFX.
Basic structure of JavaFX

- **`javafx.application.Application`** is the entry point for JavaFX applications
  - JavaFX creates an application thread for running the application start method, processing input events, and running animation timelines.
  - Override the `start(Stage)` method.

- **`javafx.stage.Stage`** is the top level JavaFX container.
  - The primary Stage is constructed by the platform.

- **`javafx.scene.Scene`** class is the container for all content in a scene graph.

- **`javafx.scene.Node`** is the base class for scene graph nodes.
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;

public class MyFirstJavaFX extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Create a button and place it in the scene
        Button btOK = new Button("OK");
        Scene scene = new Scene(btOK, 200, 250);
        primaryStage.setTitle("MyJavaFX"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }

    /**
     * The main method is only needed for the IDE with limited
     * JavaFX support. Not needed for running from the command line.
     */
    public static void main(String[] args) {
        launch(args);
    }
}
// Multiple stages can be added beside the primaryStage
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
public class MultipleStageDemo extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a scene and place a button in the scene
        Scene scene = new Scene(new Button("OK"), 200, 250);
        primaryStage.setTitle("MyJavaFX"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage

        Stage stage = new Stage(); // Create a new stage
        stage.setTitle("Second Stage"); // Set the stage title
        // Set a scene with a button in the stage
        stage.setScene(new Scene(new Button("New Stage"), 100, 100));
        stage.show(); // Display the stage
    }

    public static void main(String[] args) {
        launch(args);
    }
}
Shapes such as Line, Circle, Ellipse, Rectangle, Path, Polygon, Polyline, and Text are subclasses of Shape.

For displaying an image.

UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.
Graphical User Interfaces (ctd.)

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class ButtonInPane extends Application {

    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a scene and place a button in the scene
        StackPane pane = new StackPane();
        pane.getChildren().add(new Button("OK"));
        Scene scene = new Scene(pane, 200, 50);
        primaryStage.setTitle("Button in a pane"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }

    public static void main(String[] args) {
        launch(args);
    }
}
Using Java’ co-ordinate system

- We are used to this:

  - Java’s co-ordinate system:
    - Starts from the upper-left corner
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
public class ShowCircle extends Application {
    @Override
    public void start(Stage primaryStage) {
        Circle circle = new Circle();
        circle.setCenterX(100);
        circle.setCenterY(100);
        circle.setRadius(50);
        circle.setStroke(Color.BLACK);
        circle.setFill(null);
        ...

        // Create a pane to hold the circle
        Pane pane = new Pane();
        pane.getChildren().add(circle);

        // Create a scene and place it on stage
        Scene scene = new Scene(pane, 200, 200);

        // Set the stage title
        primaryStage.setTitle("ShowCircle");

        // Place the scene in the stage
        primaryStage.setScene(scene);
        primaryStage.show(); // Display
    }

    public static void main(String[] args) {
        launch(args);
    }
}
“Binding Properties” in JavaFX

• JavaFX introduces a new concept called **binding property** that enables a target object to be bound to a source object.

• If the value in the source object changes, the target property also changes.
  • **Automatically!**

• The target object is simply called a **binding object** or a **binding property**.

• Resizing the window in the previous example would cover the object:

![Image of a circle within a rectangle, representing the binding concept.](image-url)
public class ShowCircleCentered extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Circle circle = new Circle();
        circle.centerXProperty().bind(pane.widthProperty().divide(2));
        circle.centerYProperty().bind(pane.heightProperty().divide(2));
        circle.setRadius(50);
        circle.setStroke(Color.BLACK);
        circle.setFill(Color.WHITE);
        pane.getChildren().add(circle); // Add circle to the pane
        // Create a scene and place it in the stage
        Scene scene = new Scene(pane, 200, 200);
        primaryStage.setTitle("ShowCircleCentered"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }

    /**
     * The main method is only needed for the IDE with limited
     * JavaFX support. Not needed for running from the command line.
     */
    public static void main(String[] args) {
        launch(args);
    }
}
JavaFX Beans and Binding

- Changes made to one object (the “source”) will be automatically reflected in the other (the “target”).
- A GUI display stays synchronized with the application’s underlying data:
  
  A binding observes its list of dependencies for changes, and updates itself automatically after a change has been detected.

```java
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

public class BindingDemo {
    public static void main(String[] args) {
        DoubleProperty d1 = new SimpleDoubleProperty(1);
        DoubleProperty d2 = new SimpleDoubleProperty(2);
        d1.bind(d2);
        System.out.println("d1 is " + d1.getValue() + " and d2 is " + d2.getValue());
        d2.setValue(70.2);
        System.out.println("d1 is " + d1.getValue() + " and d2 is " + d2.getValue());
    }
}
```

**Output 1:** d1 is 2.0 and d2 is 2.0

**Output 2:** d1 is 70.2 and d2 is 70.2
CSS styling in JavaFX

```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class NodeStyleRotateDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        StackPane pane = new StackPane();
        Button btOK = new Button("OK");
        btOK.setStyle("-fx-border-color: blue;");
        pane.getChildren().add(btOK);
        pane.setRotate(45);
        pane.setStyle("-fx-border-color: red; -fx-background-color: lightgray;";
        Scene scene = new Scene(pane, 200, 250);
        primaryStage.setTitle("NodeStyleRotateDemo"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }
}
```
CSS styling in JavaFX

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class NodeStyleRotateDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        StackPane pane = new StackPane();
        Button btOK = new Button("OK");
        btOK.setStyle("-fx-border-color: blue;")
        pane.getChildren().add(btOK);
        pane.setRotate(45);
        pane.setStyle("-fx-border-color: red; -fx-background-color: lightgray;"");
        Scene scene = new Scene(pane, 200, 250);
        primaryStage.setTitle("NodeStyleRotateDemo"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }
}

The StackPane layout pane places all of the nodes within a single stack
• … with each new node added on top of the previous node.
• This provides an easy way to overlay text on a shape or image and to overlap common shapes to create a complex shape.
Using an “external” CSS style file

// Example to load and use a CSS style file in a scene
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;

public class ExternalCSSFile extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            BorderPane root = new BorderPane();
            Scene scene = new Scene(root,400,400);
            scene.getStylesheets().add(getClass()
                .getResource("application.css")
                .toExternalForm());
            primaryStage.setScene(scene);
            primaryStage.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
Helper classes

• Color

• Font

• Image

• ImageView
Helper classes

- **Color**

  ```java
  javafx.scene.paint.Color
  ```
  - red: double
  - green: double
  - blue: double
  - opacity: double

  ```java
  +Color(r: double, g: double, b: double, opacity: double)
  +brighter(): Color
  +darker(): Color
  +color(r: double, g: double, b: double): Color
  +color(r: double, g: double, b: double,.opacity: double): Color
  +rgb(r: int, g: int, b: int): Color
  +rgb(r: int, g: int, b: int, opacity: double): Color
  ```

  The red value of this Color (between 0.0 and 1.0).
  The green value of this Color (between 0.0 and 1.0).
  The blue value of this Color (between 0.0 and 1.0).
  The opacity of this Color (between 0.0 and 1.0).

  Creates a `Color` with the specified red, green, blue, and opacity values.

  Creates a `Color` that is a brighter version of this `Color`.

  Creates a `Color` that is a darker version of this `Color`.

  Creates an opaque `Color` with the specified red, green, and blue values.

  Creates a `Color` with the specified red, green, blue, and opacity values.

  Creates a `Color` with the specified red, green, and blue values in the range from 0 to 255.

  Creates a `Color` with the specified red, green, and blue values in the range from 0 to 255 and a given opacity.

- **Font**

- **Image**

- **ImageView**
Helper classes

- **Color**

- **Font**

- **Image**

- **ImageView**

---

```
javafx.scene.text.Font

- size: double
- name: String
- family: String

+Font(size: double)
+Font(name: String, size: double)
+font(name: String, size: double)
+font(name: String, w: FontWeight, size: double)
+font(name: String, w: FontWeight, p: FontPosture, size: double)
+getFamilies(): List<String>
+getFontNames(): List<String>
```

The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.

- The size of this font.
- The name of this font.
- The family of this font.

- Creates a `Font` with the specified size.
- Creates a `Font` with the specified full font name and size.
- Creates a `Font` with the specified name and size.
- Creates a `Font` with the specified name, weight, and size.
- Creates a `Font` with the specified name, weight, posture, and size.

- Returns a list of font family names.
- Returns a list of full font names including family and weight.
Helper classes

- Color
- Font
- Image
- ImageView

```java
javafx.scene.image.Image

- error: ReadOnlyBooleanProperty
- height: ReadOnlyBooleanProperty
- width: ReadOnlyBooleanProperty
- progress: ReadOnlyBooleanProperty

+ Image(filenameOrURL: String)
```

The getter methods for property values are provided in the class, but omitted in the UML diagram for brevity.

- Indicates whether the image is loaded correctly?
- The height of the image.
- The width of the image.
- The approximate percentage of image’s loading that is completed.

Creates an Image with contents loaded from a file or a URL.
Helper classes

- Color
- Font
- Image
- ImageView

```java
javafx.scene.image.ImageView

- fitHeight: DoubleProperty
- fitWidth: DoubleProperty
- x: DoubleProperty
- y: DoubleProperty
- image: ObjectProperty<Image>

+ ImageView()
+ ImageView(image: Image)
+ ImageView(filenameOrURL: String)
```

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

The height of the bounding box within which the image is resized to fit.
The width of the bounding box within which the image is resized to fit.
The x-coordinate of the ImageView origin.
The y-coordinate of the ImageView origin.
The image to be displayed in the image view.

Creates an `ImageView`.
Creates an `ImageView` with the specified image.
Creates an `ImageView` with image loaded from the specified file or URL.
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.geometry.Insets;
import javafx.stage.Stage;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;

public class ShowImage extends Application {

    @Override
    public void start(Stage primaryStage) {
        Pane pane = new HBox(10); // Create a pane to hold the image views
        pane.setPadding(new Insets(5, 5, 5, 5));
        Image image = new Image("some-picture.jpg");
        pane.getChildren().add(new ImageView(image));
        ImageView imageView2 = new ImageView(image);
        imageView2.setFitHeight(100);
        imageView2.setFitWidth(100);
        imageView2.setRotate(90);
        pane.getChildren().add(imageView2);
        Scene scene = new Scene(pane);
        primaryStage.setTitle("ShowImage");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
## Panes: to organize nodes in a container

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pane</strong></td>
<td>Base class for layout panes. It contains the <code>getChildren()</code> method for returning a list of nodes in the pane.</td>
</tr>
<tr>
<td><strong>StackPane</strong></td>
<td>Places the nodes on top of each other in the center of the pane.</td>
</tr>
<tr>
<td><strong>FlowPane</strong></td>
<td>Places the nodes row-by-row horizontally or column-by-column vertically.</td>
</tr>
<tr>
<td><strong>GridPane</strong></td>
<td>Places the nodes in the cells in a two-dimensional grid.</td>
</tr>
<tr>
<td><strong>BorderPane</strong></td>
<td>Places the nodes in the top, right, bottom, left, and center regions.</td>
</tr>
<tr>
<td><strong>HBox</strong></td>
<td>Places the nodes in a single row.</td>
</tr>
<tr>
<td><strong>VBox</strong></td>
<td>Places the nodes in a single column.</td>
</tr>
</tbody>
</table>
FlowPane

- **alignment**: ObjectProperty\<Pos>  
- **orientation**:  
  ObjectProperty\<Orientation>  
- **hgap**: DoubleProperty  
- **vgap**: DoubleProperty

**javafx.scene.layout.FlowPane**

- `FlowPane()`  
- `FlowPane(hgap: double, vgap: double)`  
- `FlowPane(orientation: ObjectProperty\<Orientation>)`  
- `FlowPane(orientation: ObjectProperty\<Orientation>, hgap: double, vgap: double)`

The overall alignment of the content in this pane (default: Pos.LEFT).
The orientation in this pane (default: Orientation.HORIZONTAL).
The horizontal gap between the nodes (default: 0).
The vertical gap between the nodes (default: 0).

Creates a default FlowPane.
Creates a FlowPane with a specified horizontal and vertical gap.
Creates a FlowPane with a specified orientation.
Creates a FlowPane with a specified orientation, horizontal gap and vertical gap.
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

public class ShowFlowPane extends Application {
    @Override
    public void start(Stage primaryStage) {
        FlowPane pane = new FlowPane();
        pane.setPadding(new Insets(11, 12, 13, 14));
        pane.setHgap(5);
        pane.setVgap(5);
        // Place nodes in the pane
        pane.getChildren().addAll(new Label("First Name:"),
                                   new TextField(),
                                   new Label("MI:"));
        TextField tfMi = new TextField();
        tfMi.setPrefColumnCount(1);
        pane.getChildren().addAll(tfMi, new Label("Last Name:"), new TextField());
        // Create a scene and place it in the stage
        Scene scene = new Scene(pane, 210, 150);
        primaryStage.setTitle("ShowFlowPane");
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }
}
### javafx.scene.layout GridPane

- **alignment**: `ObjectProperty<Pos>`
- **gridLinesVisible**: `BooleanProperty`
- **hgap**: `DoubleProperty`
- **vgap**: `DoubleProperty`

```java
+GridPane()
+add(child: Node, columnIndex: int, rowIndex: int): void
+addColumn(columnIndex: int, children: Node...): void
+addRow(rowIndex: int, children: Node...): void
+getColumnIndex(child: Node): int
+setColumnIndex(child: Node, columnIndex: int): void
+getRowIndex(child: Node): int
+setRowIndex(child: Node, rowIndex: int): void
+setHAlignment(child: Node, value: HPos): void
+setVAlignment(child: Node, value: VPos): void
```

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

The overall alignment of the content in this pane (default: `Pos.LEFT`).
Is the grid line visible? (default: `false`)
The horizontal gap between the nodes (default: 0).
The vertical gap between the nodes (default: 0).

Creates a `GridPane`.
Adds a node to the specified column and row.
Adds multiple nodes to the specified column.
Adds multiple nodes to the specified row.
Returns the column index for the specified node.
Sets a node to a new column. This method repositions the node.
Returns the row index for the specified node.
Sets a node to a new row. This method repositions the node.
Sets the horizontal alignment for the child in the cell.
Sets the vertical alignment for the child in the cell.
import javafx.application.Application;
import javafx.scene.layout.GridPane;
import javafx.geometry.*;
import javafx.scene.*;
import javafx.scene.control.*;
import javafx.stage.*;
public class ShowGridPane extends Application {
    @Override
    public void start(Stage primaryStage) {
        GridPane pane = new GridPane(); // Create a pane and set its properties
        pane.setAlignment(Pos.CENTER);
        pane.setHgap(5.5);
        pane.setVgap(5.5);

        // Place nodes in the pane at positions column,row
        pane.add(new Label("First Name:"), 0, 0);
        pane.add(new TextField(), 1, 0);
        pane.add(new Label("MI:"), 0, 1);
        pane.add(new TextField(), 1, 1);
        pane.add(new Label("Last Name:"), 0, 2);
        pane.add(new TextField(), 1, 2);
        Button btAdd = new Button("Add Name");
        pane.add(btAdd, 1, 3);
        GridPane.setHalignment(btAdd, HPos.RIGHT);

        Scene scene = new Scene(pane);
        primaryStage.setTitle("ShowGridPane");
        primaryStage.setScene(scene); primaryStage.show();
    }
}
BorderPane

```java
javafx.scene.layout.BorderPane

- top: ObjectProperty<Node>
- right: ObjectProperty<Node>
- bottom: ObjectProperty<Node>
- left: ObjectProperty<Node>
- center: ObjectProperty<Node>

+BorderPane()
+setAlignment(child: Node, pos: Pos)
```

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

The node placed in the top region (default: `null`).
The node placed in the right region (default: `null`).
The node placed in the bottom region (default: `null`).
The node placed in the left region (default: `null`).
The node placed in the center region (default: `null`).

Creates a `BorderPane`.
Sets the alignment of the node in the `BorderPane`. 

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class ShowBorderPane extends Application {
    @Override
    public void start(Stage primaryStage) {
        BorderPane pane = new BorderPane();
        pane.setTop(new CustomPane("Top"));
        pane.setRight(new CustomPane("Right"));
        pane.setBottom(new CustomPane("Bottom"));
        pane.setLeft(new CustomPane("Left"));
        pane.setCenter(new CustomPane("Center"));
        Scene scene = new Scene(pane);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}

class CustomPane extends StackPane {
    public CustomPane(String title) {
        getChildren().add(new Label(title));
        setStyle("-fx-border-color: red");
        setPadding(new Insets(11.5, 12.5, 13.5, 14.5));
    }
}
Hbox and VBox

**HBox**

```
- alignment: ObjectProperty<Pos>
- fillHeight: BooleanProperty
- spacing: DoubleProperty

+ HBox()
+ HBox(spacing: double)
+ setMargin(node: Node, value: Insets): void
```


Creates a default HBox.
Creates an HBox with the specified horizontal gap between nodes.
Sets the margin for the node in the pane.

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

**VBox**

```
- alignment: ObjectProperty<Pos>
- fillWidth: BooleanProperty
- spacing: DoubleProperty

+ VBox()
+ VBox(spacing: double)
+ setMargin(node: Node, value: Insets): void
```


Creates a default VBox.
Creates a VBox with the specified horizontal gap between nodes.
Sets the margin for the node in the pane.

The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.
public class ShowHBoxVBox extends Application {

    @Override
    public void start(Stage primaryStage) {
        BorderPane pane = new BorderPane();
        HBox hBox = new HBox(15);
        hBox.setStyle("-fx-background-color: gold");
        hBox.getChildren().add(new Button("Computer Science"));
        hBox.getChildren().add(new Button("CEWIT"));
        ImageView imageView = new ImageView(new Image("cs14.jpg"));
        hBox.getChildren().add(imageView);
        pane.setTop(hBox);
        VBox vBox = new VBox(15);
        vBox.getChildren().add(new Label("Courses"));
        Label[] courses = {new Label("CSE114"), new Label("CSE214"),
                new Label("CSE219"), new Label("CSE308")};
        for (Label course: courses)
            vBox.getChildren().add(course);
        pane.setLeft(vBox);
        Scene scene = new Scene(pane); primaryStage.setScene(scene);
        primaryStage.show();
    }
}