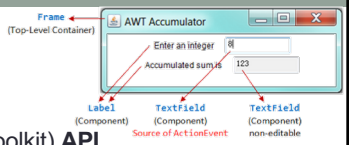
OBJECT-ORIENTED LANGUAGE AND THEORY

9. GUI PROGRAMMING

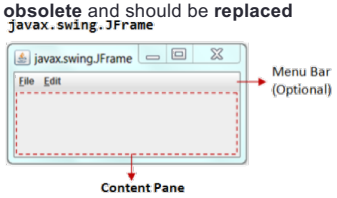
Nguyen Thi Thu Trang
trangntt@soict.hust.edu.vn

Outline

AWT and Swing



- **AWT** (Abstract Windowing Toolkit) **API**
  - From JDK 1.0
  - Most components have **become obsolete** and should be **replaced by Swing** components



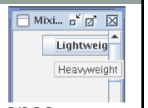
- **Swing** API
  - From JDK 1.1, as a part of **Java Foundation Classes** (JFC)
  - A much more comprehensive set of graphics libraries that enhances the AWT
  - JFC consists of **Swing**, *Java2D, Accessibility, Internationalization, and Pluggable Look-and-Feel Support APIs.*

JavaFX

- **JavaFX** is a software platform for creating and delivering desktop applications, as well as **rich internet applications** (RIAs) that can run across a wide variety of devices.
- **JavaFX** is intended to replace **Swing** as the standard ***GUI library*** *for* ***Java SE***, but both will be included for the foreseeable future.

IFX is just a name, which is normally related with `sound or visual effects` *in the* ***javafx*** *i was in the belief that the* ***fx*** *was function. ...*
FIPS ***stands*** *for the Federal Information Processing Standardization*

Which should we choose?



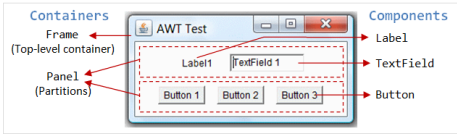
- **AWT**: for simple GUI, but not for comprehensive ones
  - Native OS GUI
  - Flatform-independent and device-independent interface
  - Heavyweight components

- **Swing**: Pure Java code with a more robust, versatile, and flexible library
  - Use AWT for windows and event handling
  - Pure-Java GUI, 100% portable and same across platform
  - Most components are light-weight, different look-and-feel
- **JavaFX:** for developing rich Internet applications
  - Can run across a wide variety of devices
  - More consistent in style and has additional options, e.g. 3D, chart, audio, video…

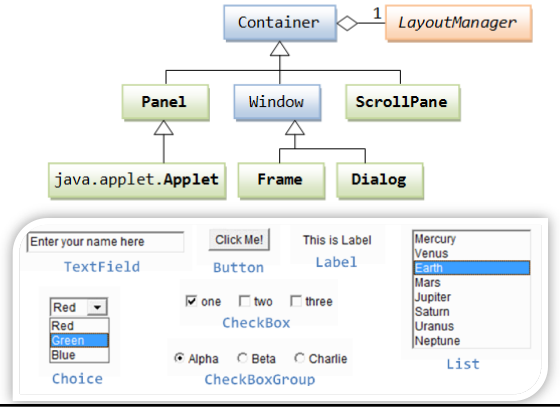
Outline

AWT Containers and Components

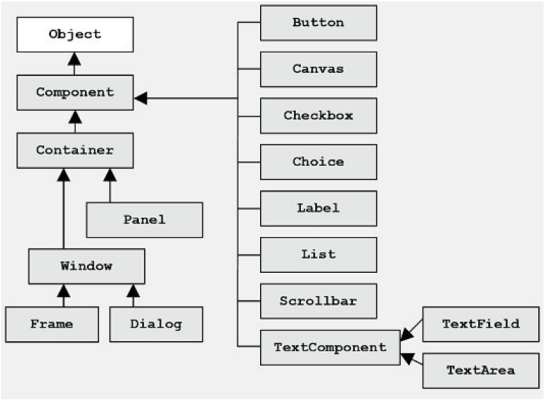
- There are **two types of GUI elements**:
  - ***Component***: Components are elementary GUI entities (e.g. Button, Label, and TextField.)
  - ***Container***: Containers (e.g. Frame, Panel and Applet) are used to *hold components in a specific layout* (such as flow or grid). A container can also hold sub-containers.
  - **GUI components are also called *controls* (Microsoft ActiveX Control), *widgets* (Eclipse's Standard Widget Toolkit, Google Web Toolkit)**, which allow users to interact with the application through these components (*such as button-click and text-entry).*



Hierarchy of the AWT Container Classes



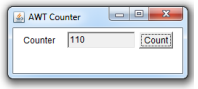
AWT Component Classes

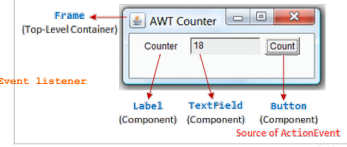


Example: AWT Counter

```
import java.awt.Button;
import java.awt.Frame;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionListener;

public class AWTCounter
        extends Frame implements ActionListener {
   private Label lblCount;
   private TextField tfCount;
   private Button btnCount;
   private int count = 0;      // Counter's value

   // Constructor to setup GUI components and event handling

   public AWTCounter () {
       setLayout(new FlowLayout());

       lblCount = new Label("Counter");
       add(lblCount);

       tfCount = new TextField("0", 10);
       tfCount.setEditable(false); // set to read-only
       add(tfCount);

       btnCount = new Button("Count");
       add(btnCount);

       btnCount.addActionListener(this);
       //Clicking Button source fires ActionEvent
      //btnCount registers this instance as ActionEvent listener

       setTitle("AWT Counter");
       setSize(250, 100);

       setVisible(true);
   }
```

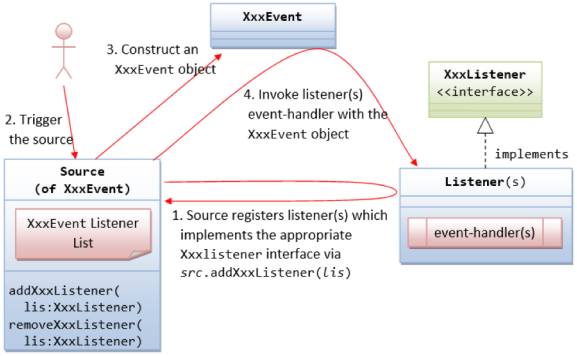
```
   /** The entry main() method */
   public static void main(String[] args) {
      // Invoke the constructor to setup the GUI, by allocating an instance
      AWTCounter app = new AWTCounter();
   }
   /** ActionEvent handler - Called back upon button-click. */
   public void actionPerformed(ActionEvent evt) {
      ++count; // increase the counter value
      // Display the counter value on the TextField tfCount
      tfCount.setText(count + ""); // convert int to String
   }
}
```





AWT Event Handling

Event Driven / Event Delegation



E.g. MouseListener (XxxListener) interface

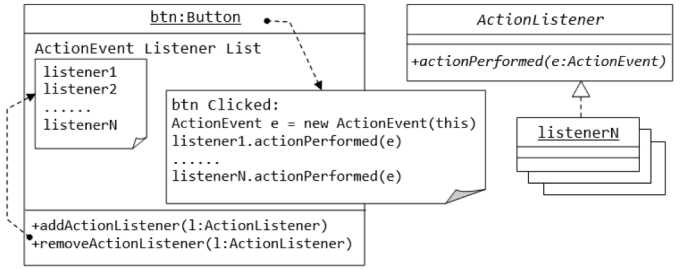
```
//A MouseListener interface, which declares the signature of the handlers
//for the various operational modes.
public interface MouseListener {
   // Called back upon mouse-button pressed
   public void mousePressed(MouseEvent evt);
   // Called back upon mouse-button released
   public void mouseReleased(MouseEvent evt);
   // Called back upon mouse-button clicked (pressed and released)
   public void mouseClicked(MouseEvent evt);
   // Called back when mouse pointer entered the component
   public void mouseEntered(MouseEvent evt);
   // Called back when mouse pointer exited the component
   public void mouseExited(MouseEvent evt);
}
```

Add or remove XxxListener in the source:

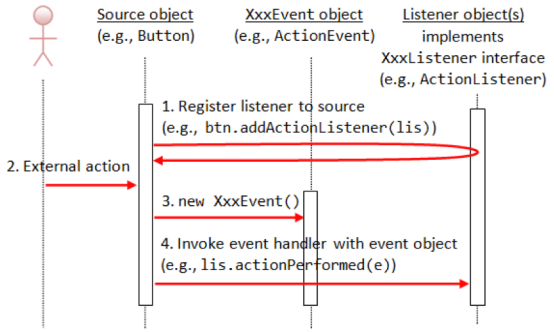
```
public void addXxxListener(XxxListener lis);
public void removeXxxListener(XxxListener lis);
```

```
//An example provides implementation to the event handler methods
class MouseDemo implements MouseListener {
   private Button btn;
   public MouseDemo(){
         //...
         btn.addMouseListener(this);
   }
   @Override
   public void mousePressed(MouseEvent e) {
      System.out.println("Mouse-button pressed!");
   }
   @Override
   public void mouseReleased(MouseEvent e) {
      System.out.println("Mouse-button released!");
   }
   @Override
   public void mouseClicked(MouseEvent e) {
      System.out.println("Mouse-button clicked (pressed and released)!");
   }
   @Override
   public void mouseEntered(MouseEvent e) {
      System.out.println("Mouse-pointer entered the source component!");
   }
   @Override
   public void mouseExited(MouseEvent e) {
      System.out.println("Mouse exited-pointer the source component!");
   }
}
```

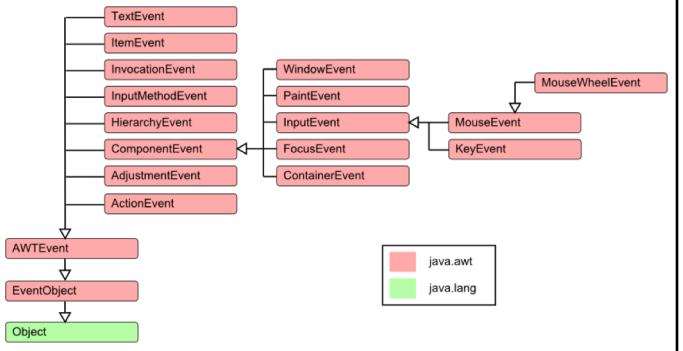

Revisit AWTCounter example



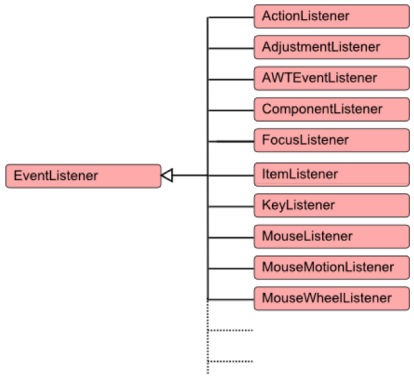
Revisit AWTCounter example



Event Hierarchy



EventListener Hierarchy



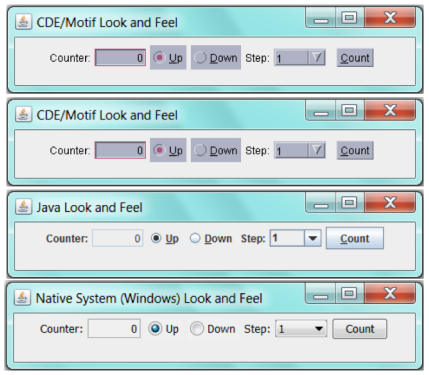
Outline

1. GUI Programming in Java
2. AWT
3. Swing
4. JavaFX

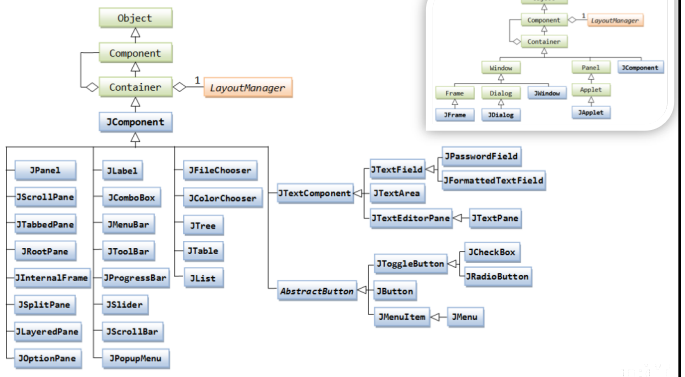
Java Swing

- Light Weight: Pure JAVA code
  - Freelance of native operational System's API
- Use the Swing components with prefix "J", e.g. JFrame, JButton, JTextField, JLabel, etc.
  - Advanced controls like Tree, color picker, table controls, TabbedPane, slider.
- Uses the AWT event-handling classes
- Highly Customizable
  - Often made-to-order in a very simple method as visual appearance is freelance of content.
- Pluggable look-and-feel
  - Modified at run-time, supported by accessible values.

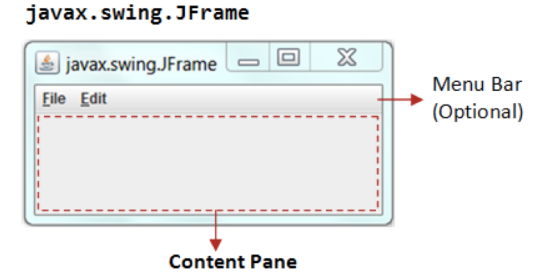
Swing – Different Look & Feel



Swing Container and Component



Containers and ContentPane

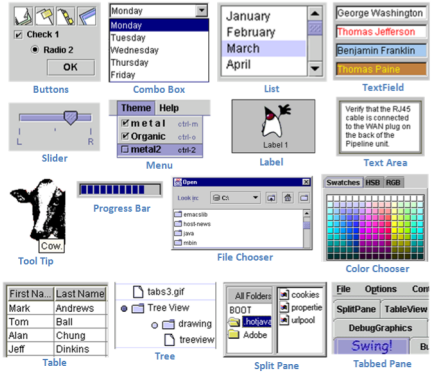



```
Container cp = aJFrame.getContentPane();
aJFrame.setContentPane(aPanel);
```

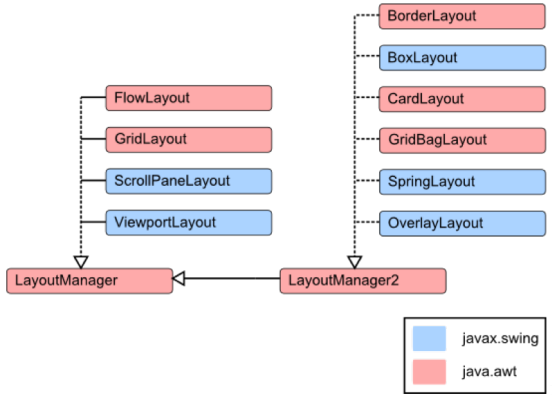

Swing components

Swing is huge (consists of 18 packages of 737 classes as in JDK 1.8) and has great depth

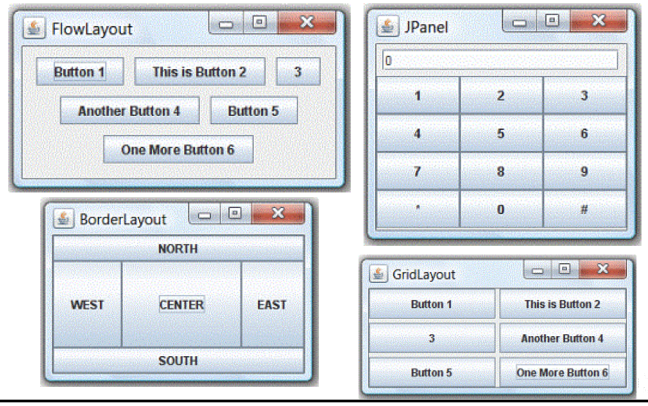
Compare to AWT:
12 packages of 370 classes



Swing Layout



Layout management




```
//A Swing GUI application inherits from top-level container
public class SwingDemo extends JFrame {
   // Private instance variables
   // Constructor to setup the GUI components and event handlers
   public SwingDemo() {
      // top-level content-pane from JFrame
      Container cp = getContentPane();
      cp.setLayout(new ...Layout());

      // Allocate the GUI components
      cp.add(....);

      // Source object adds listener
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     // Exit the program when the close-window button clicked
     setTitle("......");  //"super" JFrame sets title
     setSize(300, 150);   //"super" JFrame sets initial size
     setVisible(true);    // "super" JFrame shows
   }
```


(cont.)

```
   // The entry main() method
   public static void main(String[] args) {
      // Run GUI codes in Event-Dispatching thread
      //for thread-safety
      SwingUtilities.invokeLater(new Runnable() {
          @Override
          public void run() {
             new SwingDemo();
          }
       });
   }
}
```


Outline

1. GUI Programming in Java
2. AWT
3. Swing
4. JavaFX

Why JavaFX?

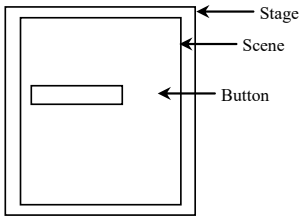
- Additional consistent in its style than Swing
  - Contains WebView supported the popular WebKit browser => Introduce Website within JavaFX
- Design GUI like Web apps with XML and CSS (FXML) than doing in Java code
  - Save building time
- Integrate 3D graphics, charts, and audio, video, and embedded Website inside GUI
  - Easy to develop Game/Media applications
- Light-weight and hardware accelerated

Why JavaFX? (2)

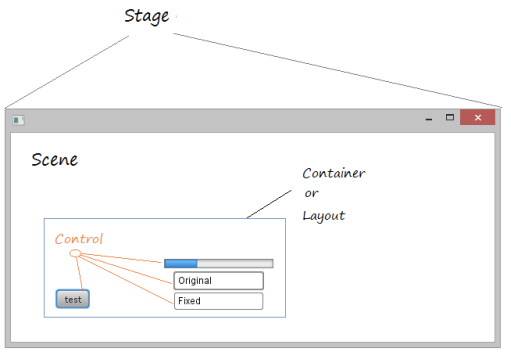
- Support stylish animations
  - Resembling fades, Rotations, Motion ways
  - Custom animations with KeyFrame and Timeline
- Support for modern touch devices
  - Resembling scrolling, swipping, rotating and zooming…
- Many eye-catching controls
  - Collapsible Titled Pane
- Events are higher thought-out and additional consistent

Basic Structure of JavaFX

- Application
- Override the start(Stage) method
- Stage, Scene, and Nodes

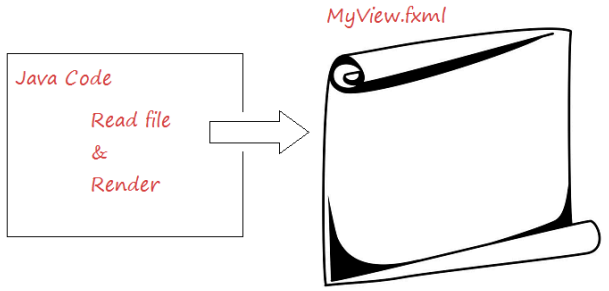


Basic Structure of JavaFX



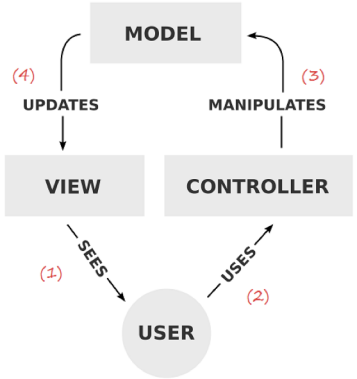
```java
import javafx.application.Application;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            BorderPane root = new BorderPane();
            Scene scene = new Scene(root,400,400);
            scene.getStylesheets().add(getClass().getRes
            primaryStage.setScene(scene);
            primaryStage.show();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

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


JavaFX Scene Builde



MVC pattern

1. After seeing it on VIEW
2. Users use CONTROLLER
3. Manipulate data (Update, modify, delete, ..), the data on MODEL has been changed.
4. Displaying the data of MODEL on VIEW.



View: FXML

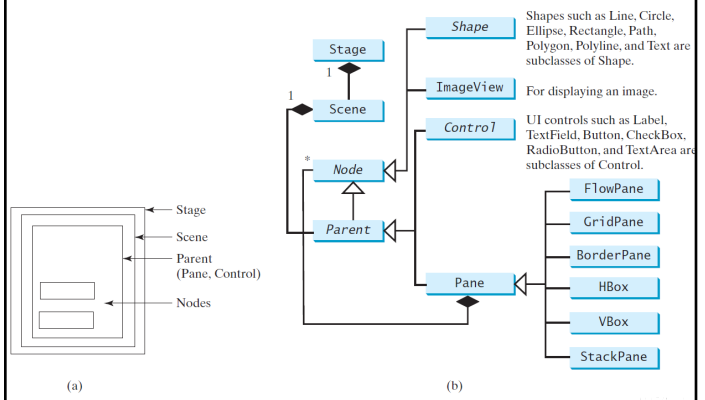
MyScene.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.text.*?>
<?import javafx.scene.control.*?>
<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.AnchorPane?>

<AnchorPane prefHeight="238.0" prefWidth="269.0"
    xmlns="http://javafx.com/javafx/8"
    xmlns:fx="http://javafx.com/fxml/1"
    fx:controller="org.o7planning.javafx.MyController">

  <children>

     <Button fx:id="myButton" layoutX="51.0" layoutY="44.0"
          mnemonicParsing="false"
          onAction="#showDateTime" text="Show Date Time" />

     <TextField fx:id="myTextField" layoutX="28.0"
          layoutY="107.0" prefHeight="25.0" prefWidth="201.0" />

  </children>

</AnchorPane>
```

Controller: Java code

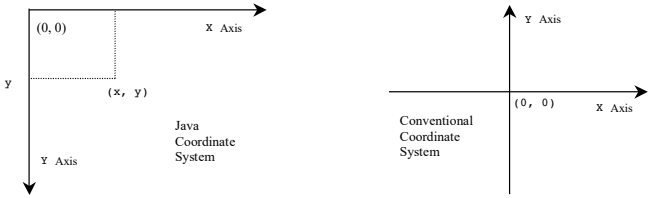
```
public class MyController implements Initializable {
    @FXML
    private Button myButton;
    @FXML
    private TextField myTextField;

    // When user click on myButton, this method will be called
    public void showDateTime(ActionEvent event) {
        System.out.println("Button Clicked!");
        Date now= new Date();
        DateFormat df = new SimpleDateFormat(
                                "dd-MM-yyyy HH:mm:ss.SSS");
        // Model Data
        String dateTimeString = df.format(now);
        // Show in VIEW
        myTextField.setText(dateTimeString);
    }
}
```


Panes, UI Controls, and Shapes



Display a Shape

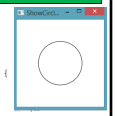
This example displays a circle in the center of the pane.



Binding Properties

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 is also changed automatically. The target object is simply called a *binding object* or a *binding property*.

ShowCircleCentered



```
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 ShowCircleCentered extends Application {
public void start(Stage primaryStage) {
    // Create a pane to hold the circle
    Pane pane = new Pane();
      // Create a circle and set its properties
    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);
  }
}
```

Binding Property: getter, setter, and property getter

```
public class SomeClassName {

  private PropertyType x;

  /** Value getter method */
  public propertyValueType getX() { ... }

  /** Value setter method */
  public void setX(propertyValueType value) { ... }

  /** Property getter method */
  public PropertyType
    xProperty() { ... }
}
```

(a) `x` is a binding property

```
public class Circle {

  private DoubleProperty centerX;

  /** Value getter method */
  public double getCenterX() { ... }

  /** Value setter method */
  public void setCenterX(double value) { ... }

  /** Property getter method */
  public DoubleProperty centerXProperty() { ... }
}
```

(b) `centerX` is binding property

The Color Class

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

javafx.scene.paint.Color	
-red: double	The red value of this Color (between 0.0 and 1.0).
-green: double	The green value of this Color (between 0.0 and 1.0).
-blue: double	The blue value of this Color (between 0.0 and 1.0).
-opacity: double	The opacity of this Color (between 0.0 and 1.0).
+Color(r: double, g: double, b: double, opacity: double)	Creates a Color with the specified red, green, blue, and opacity values.
+brighter(): Color	Creates a Color that is a brighter version of this Color.
+darker(): Color	Creates a Color that is a darker version of this Color.
+color(r: double, g: double, b: double): Color	Creates an opaque Color with the specified red, green, and blue values.
+color(r: double, g: double, b: double, opacity: double): Color	Creates a Color with the specified red, green, blue, and opacity values.
+rgb(r: int, g: int, b: int): Color	Creates a Color with the specified red, green, and blue values in the range from 0 to 255.
+rgb(r: int, g: int, b: int, opacity: double): Color	Creates a Color with the specified red, green, and blue values in the range from 0 to 255 and a given opacity.

The Font Class

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

javafx.scene.text.Font	
-size: double	The size of this font.
-name: String	The name of this font.
-family: String	The family of this font.
+Font(size: double)	Creates a Font with the specified size.
+Font(name: String, size: double)	Creates a Font with the specified full font name and size.
+font(name: String, size: double)	Creates a Font with the specified name and size.
+font(name: String, w: FontWeight, size: double)	Creates a Font with the specified name, weight, and size.
+font(name: String, w: FontWeight, p: FontPosture, size: double)	Creates a Font with the specified name, weight, posture, and size.
+getFamilies(): List<String>	Returns a list of font family names.
+getFontNames(): List<String>	Returns a list of full font names including family and weight.

FontDemo Run

The Image Class

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

javafx.scene.image.Image	
-error: ReadOnlyBooleanProperty	Indicates whether the image is loaded correctly?
-height: ReadOnlyBooleanProperty	The height of the image.
-width: ReadOnlyBooleanProperty	The width of the image.
-progress: ReadOnlyBooleanProperty	The approximate percentage of image's loading that is completed.
+Image(filenameOrURL: String)	Creates an Image with contents loaded from a file or a URL.

The ImageView Class

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.

javafx.scene.image.ImageView

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

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

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.

ShowImage Run

Layout Panes

JavaFX provides many types of panes for organizing nodes in a container.

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

FlowPane

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.

javafx.scene.layout.FlowPane

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

```
+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.

ShowFlowPane Run

GridPane

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.

javafx.scene.layout.GridPane

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

```
+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
+setHalighnment(child: Node,
   value: HPos): void
+setValighnment(child: Node,
   value: VPos): void
```

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.

ShowGridPane Run

BorderPane

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.

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 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`.


ShowBorderPane Run

HBox

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.

javafx.scene.layout.HBox

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

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

The overall alignment of the children in the box (default: `Pos.TOP_LEFT`).
Is resizable children fill the full height of the box (default: `true`).
The horizontal gap between two nodes (default: 0).

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

VBox

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.

javafx.scene.layout.VBox

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

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

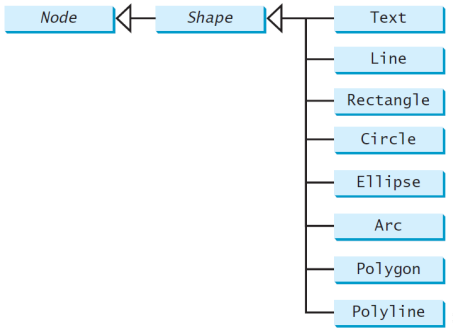
The overall alignment of the children in the box (default: `Pos.TOP_LEFT`).
Is resizable children fill the full width of the box (default: `true`).
The vertical gap between two nodes (default: 0).

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


ShowHBoxVBox Run

Shapes

JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.



Text

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.

javafx.scene.text.Text

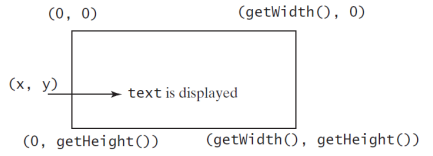
```
-text: StringProperty
-x: DoubleProperty
-y: DoubleProperty
-underline: BooleanProperty
-strikethrough: BooleanProperty
-font: ObjectProperty<Font>
```

```
+Text()
+Text(text: String)
+Text(x: double, y: double,
   text: String)
```

Defines the text to be displayed.
Defines the x-coordinate of text (default 0).
Defines the y-coordinate of text (default 0).
Defines if each line has an underline below it (default `false`).
Defines if each line has a line through it (default `false`).
Defines the font for the text.

Creates an empty Text.
Creates a Text with the specified text.
Creates a Text with the specified x-, y-coordinates and text.

Text Example



(a) Text(x, y, text)

(b) *Three* Text *objects are displayed*

ShowText Run

Line

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.

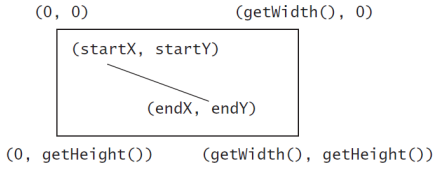
javafx.scene.shape.Line

```
-startX: DoubleProperty
-startY: DoubleProperty
-endX: DoubleProperty
-endY: DoubleProperty
```

```
+Line()
+Line(startX: double, startY:
   double, endX: double, endY:
   double)
```

The x-coordinate of the start point.
The y-coordinate of the start point.
The x-coordinate of the end point.
The y-coordinate of the end point.

Creates an empty `Line`.
Creates a `Line` with the specified starting and ending points.



ShowLine Run

Rectangle

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.

javafx.scene.shape.Rectangle

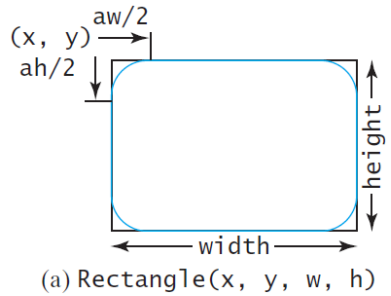
```
-x: DoubleProperty
-y:DoubleProperty
-width: DoubleProperty
-height: DoubleProperty
-arcWidth: DoubleProperty

-arcHeight: DoubleProperty
```

```
+Rectangle()
+Rectanlge(x: double, y:
   double, width: double,
   height: double)
```

The x-coordinate of the upper-left corner of the rectangle (default 0).
The y-coordinate of the upper-left corner of the rectangle (default 0).
The width of the rectangle (default: 0).
The height of the rectangle (default: 0).
The `arcWidth` of the rectangle (default: 0). `arcWidth` is the horizontal diameter of the arcs at the corner (see Figure 14.31a).
The `arcHeight` of the rectangle (default: 0). `arcHeight` is the vertical diameter of the arcs at the corner (see Figure 14.31a).

Creates an empty `Rectangle`.
Creates a `Rectangle` with the specified upper-left corner point, width, and height.

Rectangle Example



(a) Rectangle(x, y, w, h)

ShowRectangle Run

Circle

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.

javafx.scene.shape.Circle

```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radius: DoubleProperty
```

```
+Circle()
+Circle(x: double, y: double)
+Circle(x: double, y: double,
   radius: double)
```

The x-coordinate of the center of the circle (default 0).
The y-coordinate of the center of the circle (default 0).
The radius of the circle (default: 0).

Creates an empty `Circle`.
Creates a `Circle` with the specified center.
Creates a `Circle` with the specified center and radius.

Ellipse

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.

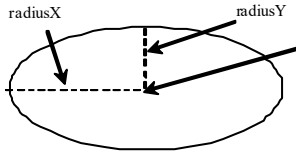
javafx.scene.shape.Ellipse

```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
```

```
+Ellipse()
+Ellipse(x: double, y: double)
+Ellipse(x: double, y: double,
   radiusX: double, radiusY:
   double)
```

The x-coordinate of the center of the ellipse (default 0).
The y-coordinate of the center of the ellipse (default 0).
The horizontal radius of the ellipse (default: 0).
The vertical radius of the ellipse (default: 0).

Creates an empty `Ellipse`.
Creates an `Ellipse` with the specified center.
Creates an `Ellipse` with the specified center and radiuses.



ShowEllipse Run

Arc

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.

javafx.scene.shape.Arc

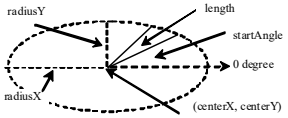
```
-centerX: DoubleProperty
-centerY: DoubleProperty
-radiusX: DoubleProperty
-radiusY: DoubleProperty
-startAngle: DoubleProperty
-length: DoubleProperty
-type: ObjectProperty<ArcType>
```

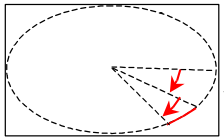
```
+Arc()
+Arc(x: double, y: double,
   radiusX: double, radiusY:
   double, startAngle: double,
   length: double)
```

The x-coordinate of the center of the ellipse (default 0).
The y-coordinate of the center of the ellipse (default 0).
The horizontal radius of the ellipse (default: 0).
The vertical radius of the ellipse (default: 0).
The start angle of the arc in degrees.
The angular extent of the arc in degrees.
The closure type of the arc (`ArcType.OPEN`, `ArcType.CHORD`, `ArcType.ROUND`).

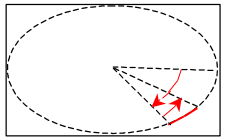
Creates an empty `Arc`.
Creates an `Arc` with the specified arguments.

Arc Examples





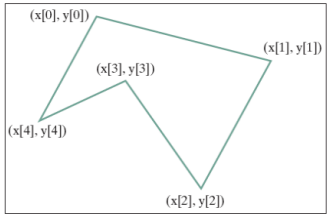
(a) Negative starting angle –30° and negative spanning angle –20°



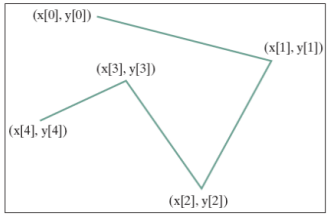
(b) Negative starting angle –50° and positive spanning angle 20°

ShowArc Run

Polygon and Polyline



(a) Polygon



(b) Polyline

ShowArc Run

Polygon

javafx.scene.shape.Polygon	
+Polygon()	Creates an empty polygon.
+Polygon(double... points)	Creates a polygon with the given points.
+getPoints(): ObservableList<Double>	Returns a list of double values as x- and y-coordinates of the points.

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.

ShowPolygon Run

Case Study: The ClockPane Class

This case study develops a class that displays a clock on a pane.

javafx.scene.layout.Panel

ClockPane

-hour: int	The hour in the clock.
-minute: int	The minute in the clock.
-second: int	The second in the clock.
+ClockPane()	Constructs a default clock for the current time.
+ClockPane(hour: int, minute: int, second: int)	Constructs a clock with the specified time.
+setCurrentTime(): void	Sets hour, minute, and second for current time.
+setWidth(width: double): void	Sets clock pane's width and repaint the clock,
+setHeightTime(height: double): void	Sets clock pane's height and repaint the clock,

The `getter` and `setter` methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

ClockPane