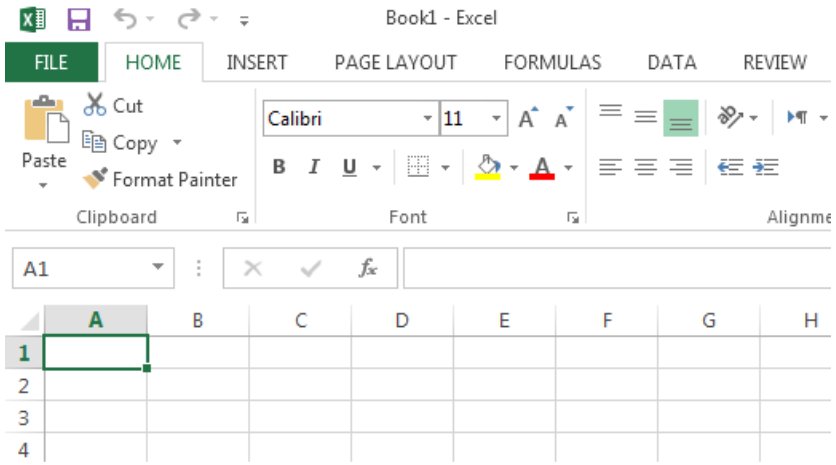
Object-oriented programming

Second semester

Lecture №3

Graphical User Interfaces

GUI Examples



GUI

- Graphical User Interface (GUI)
  - provides user-friendly human interaction
- Building Java GUIs require use of multiple frameworks:
  - JavaFX (part of JSE 8, 2014)
  - An old framework would use:
    - Java's GUI component Libraries
      - **javax.swing.***
    - Java's Event Programming Libraries
      - **java.awt.event.***
      - **Javax.swing.event.***
    - Java's Graphics Programming Libraries
      - **java.awt.***
      - **java.awt.geom.***

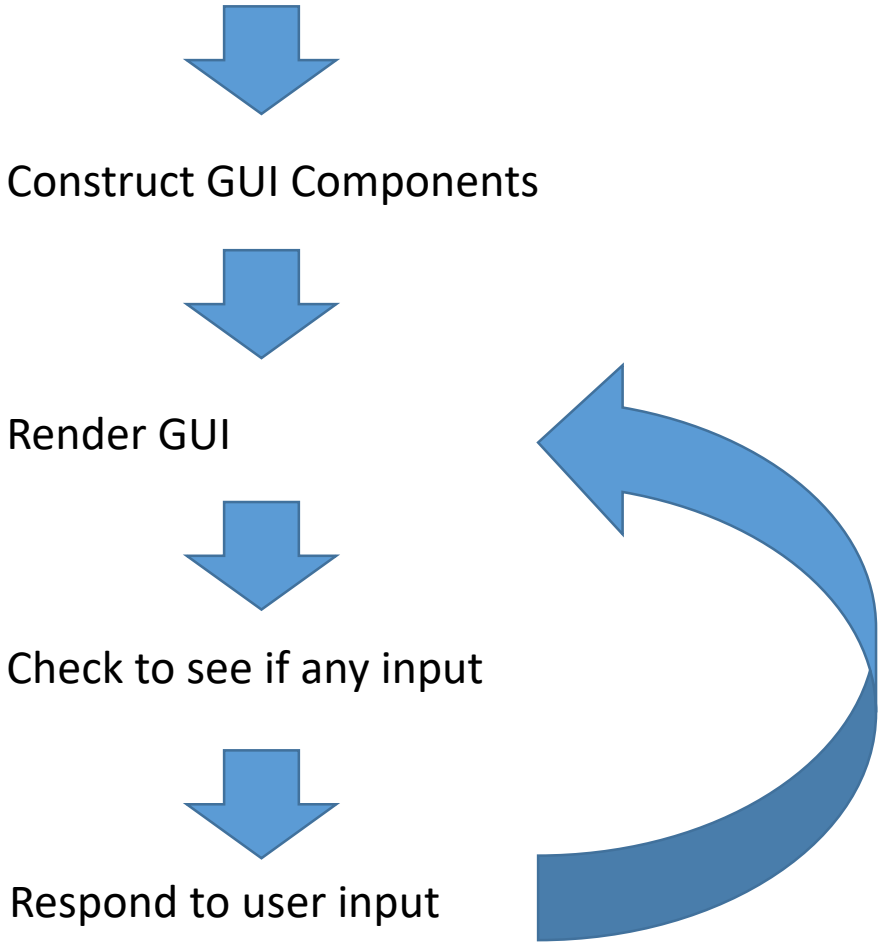
JavaFX vs Swing and AWT

- Swing and AWT are replaced by the JavaFX platform for developing rich Internet applications in JDK8.
  - 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 components.
    - Swing components are painted directly on canvases using Java code.
    - Swing components depend less on the target platform and use less of the native GUI resource.
- **With the release of Java 8, Swing is replaced by a completely new GUI platform: JavaFX.**

Example: a mouse click on a button

- Operating System recognizes mouse click
  - determines which window it was inside
  - notifies that program
- Program runs in loop
  - checks input buffer filled by OS
  - if it finds a mouse click:
    - determines which component in the program
    - if the click was on a relevant component
      - respond appropriately according to handler

How do GUIs work?



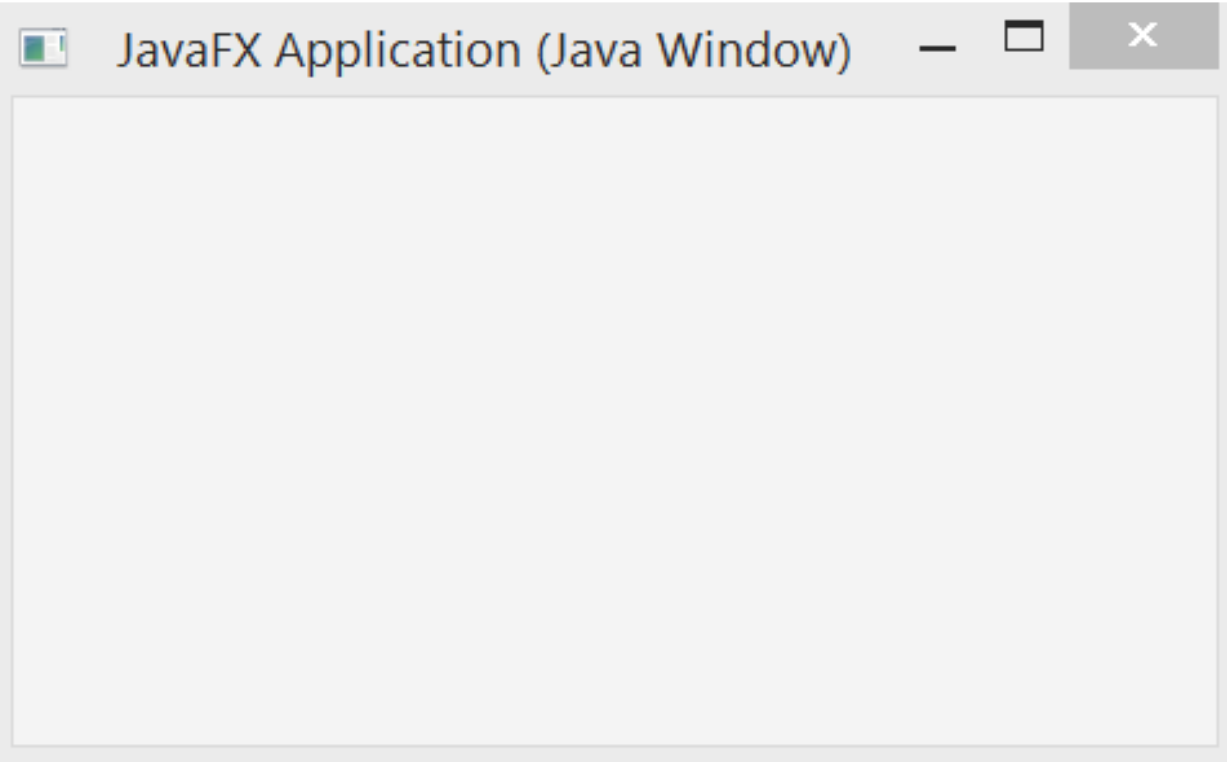
- They loop and respond to events

GUI Look vs. Behavior

- Look
  - physical appearance
  - custom component design
  - containment
  - layout management
- Behavior
  - interactivity
  - event programmed response

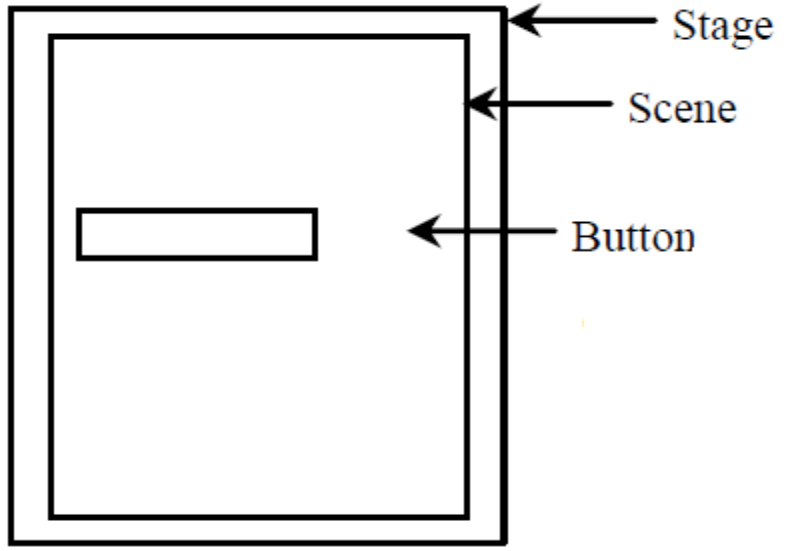
What does a GUI framework do for you?

- Provides ready made visible, interactive, customizable components
- you wouldn’t want to have to code your own window



Basic Structure of JavaFX

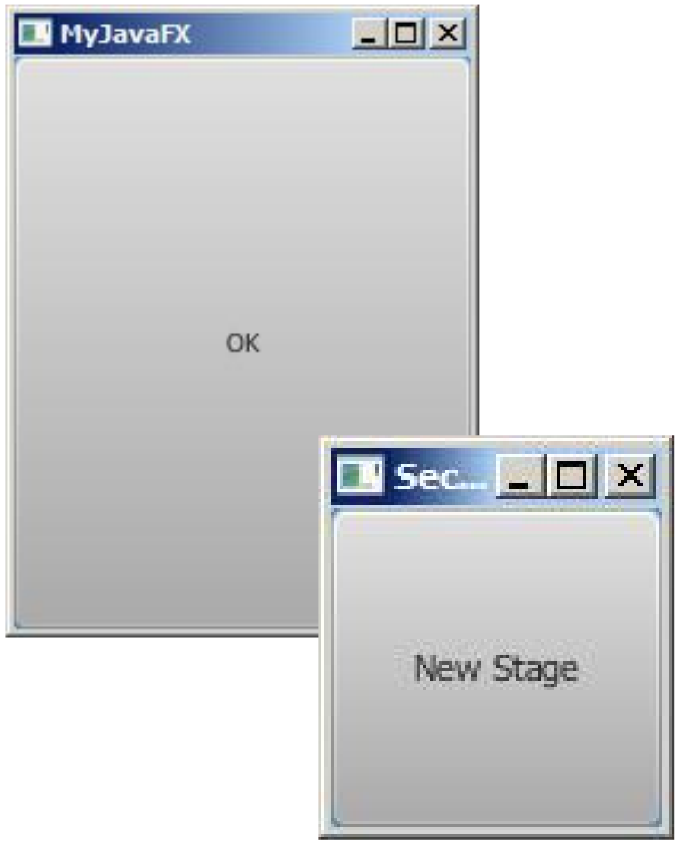
- **javafx.application.Application** is the entry point for JavaFX applications
  - JavaFX creates an application thread for running the application start method, processing input events, and running animation timelines.
  - Override the start(Stage) method!
- **javafx.stage.Stage** is the top level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



My first JavaFX App

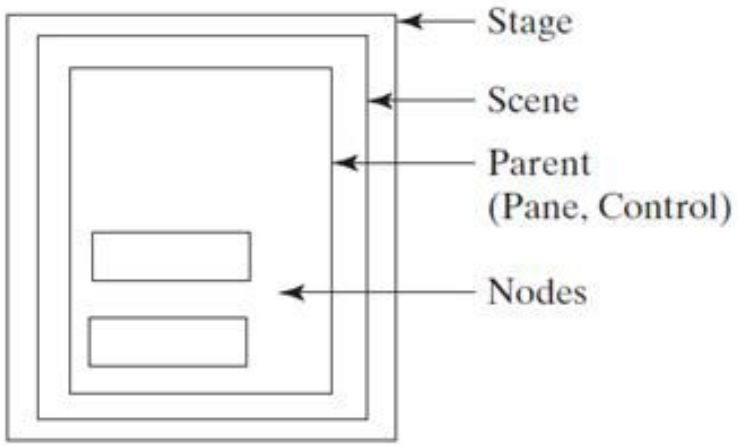

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;

public class MyFirstJavaFX extends Application
{
        @Override // Override the start method in the Application class
        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);
        }
}
```

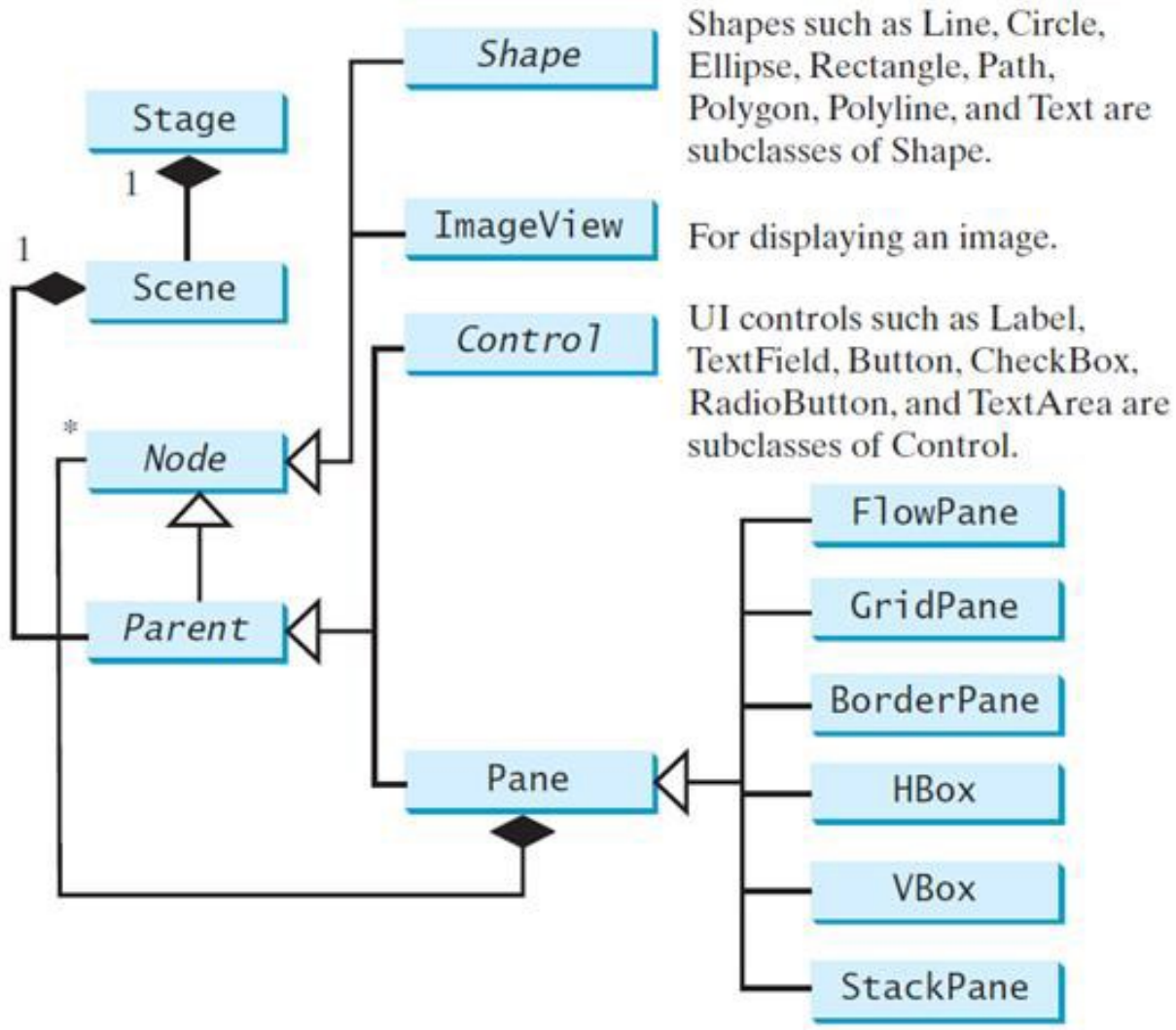
My second JavaFX App




```
// Multiple stages can be added beside the primaryStage
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.Button;
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);
        }
}
```

Panes, UI Controls, and Shapes

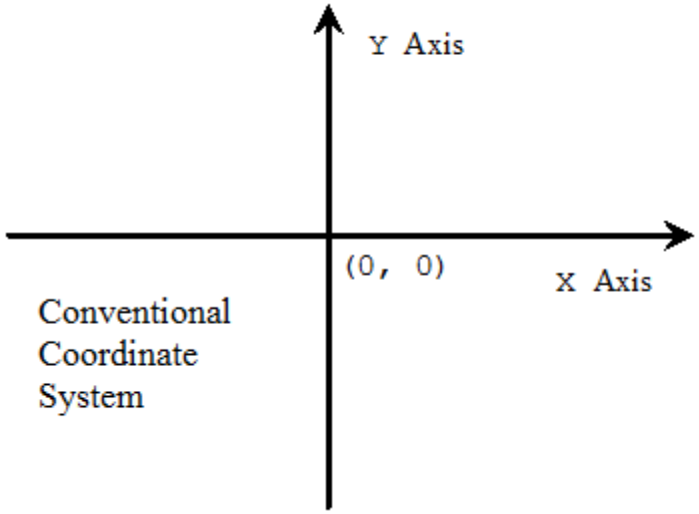




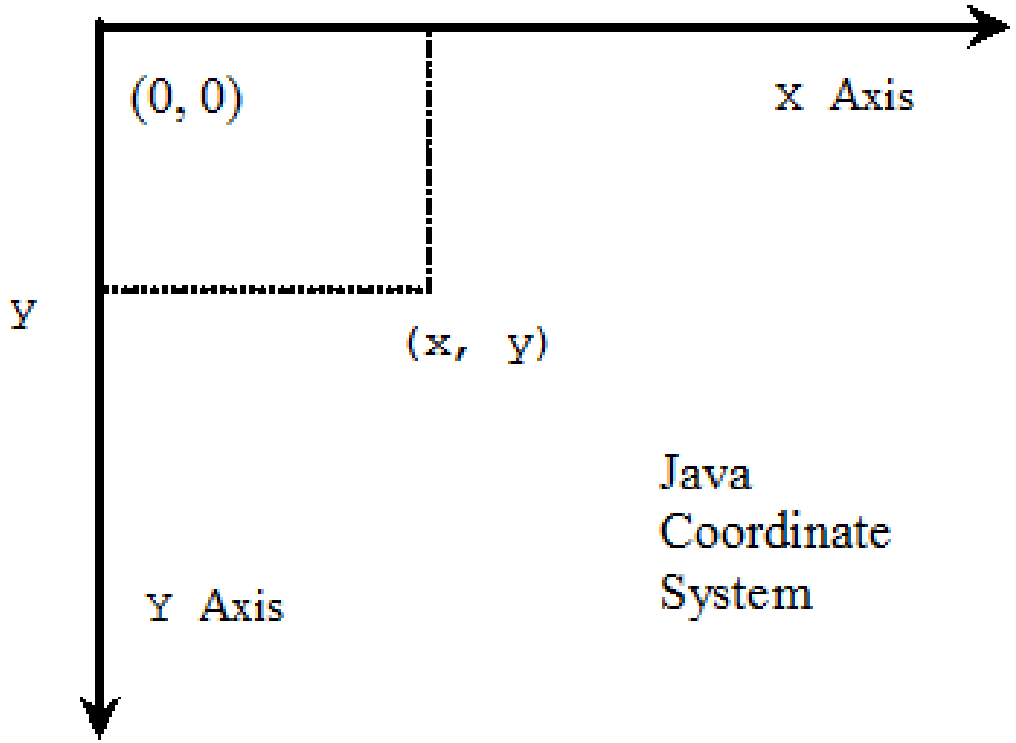
Pane


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

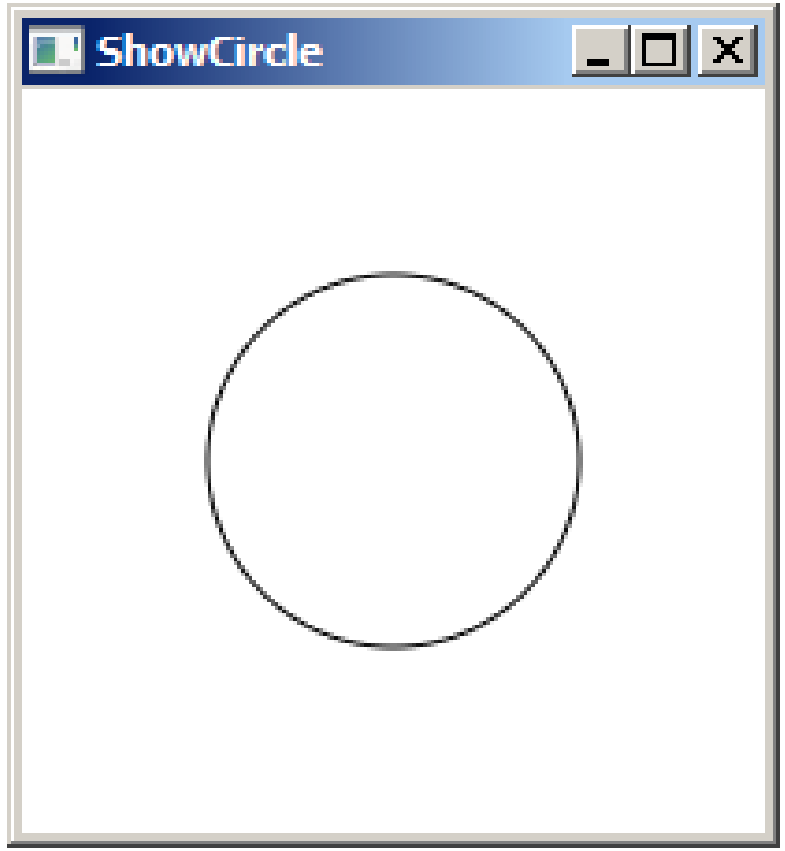
Display a Shape



- Programming Coordinate Systems start from the left-upper corner



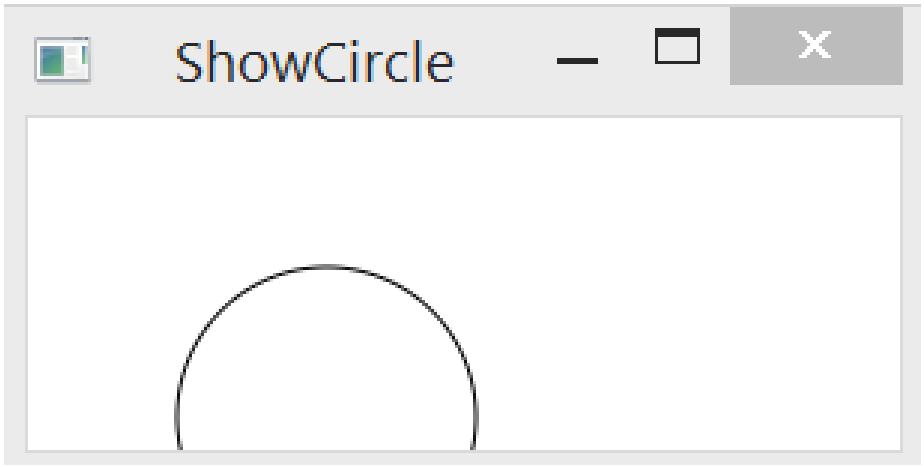
Circle



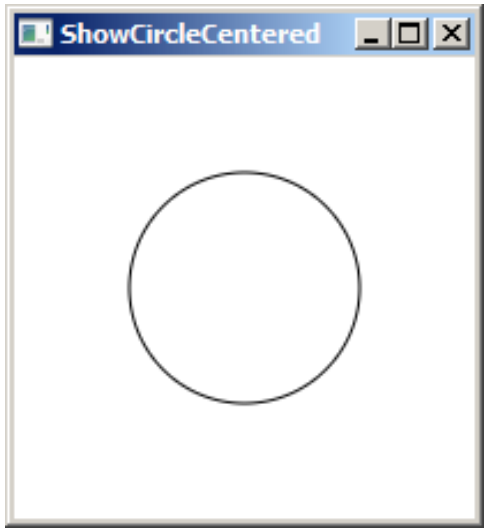

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Circle;
import javafx.scene.paint.Color;
public class ShowCircle extends Application {
		@Override // Override the start method in the Application class
		public void start(Stage primaryStage) {
				// Create a circle and set its properties
				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 in the stage
				Scene scene = new Scene(pane, 200, 200);
				primaryStage.setTitle("ShowCircle"); // 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 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.
- Resizing the window in the previous example would cover the object:



Binding




```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Circle;
import javafx.scene.paint.Color;
public class ShowCircleCentered extends Application {
        @Override // Override the start method in the Application class
        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); }
}
```

JavaFX Beans and Binding

- Changes made to one object will automatically be reflected in another object
  - A graphical user interface automatically keeps its display synchronized with the application's underlying data: a binding observes its list of dependencies for changes, and then updates itself automatically after a change has been detected.

```
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:
d1 is 2.0 and d2 is 2.0
d1 is 70.2 and d2 is 70.2