Chapter 14

JAVAFX BASICS

Objectives

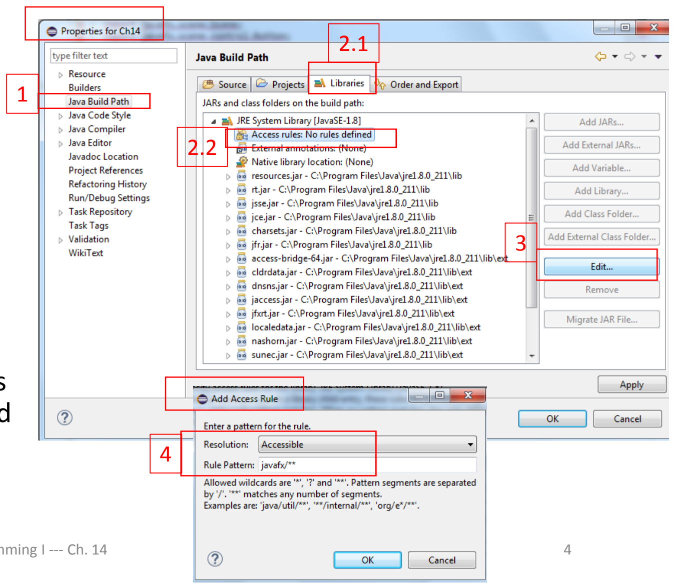
- To understand the relationship among stages, scenes, and nodes
- To create user interfaces using panes, UI controls, and shapes
- To update property values automatically through property binding
- To use the common properties **style** and **rotate** for nodes
- To create colors using the **Color** class
- To create fonts using the **Font** class
- To create images using the **Image** class and to create image views using the **ImageView** class
- To layout nodes using **Pane**, **StackPane**, **FlowPane**, **GridPane**, **BorderPane**, **HBox**, and **VBox**
- To display text using the **Text** class and create shapes using **Line**, **Circle**, **Rectangle**, **Ellipse**, **Arc**, **Polygon**, and **Polyline**

JavaFX: an introduction

- JavaFX is a framework for developing Java GUI programs.
- A JavaFX application can run seemlessly on a desktop and from a Web browser.
- JavaFX has a built-in 2D, 3D, animation support, video and audio playback, and runs as a stand-alone application or from a browser.
- *The abstract* **javafx.application.Application** *class defines the essential framework for writing JavaFX programs.*

Using JavaFX in your Java project on Eclipse

- To use JavaFX in your Java project on Eclipse, you need to enable it.
- To enable it on your Eclipse project,
  1. Go on Properties then Java Build Path.
  2. Choose the tag for JRE System Library and select Access Rules.
  3. Click on Edit on the right menu.
  4. Now, you have to create a new Rule with Accessible resolution and the following Rule Pattern : javafx/** . This setting will enable JavaFX packages and classes for your application.



JARs and class folders on the build path:

```
JRE System Library [JavaSE-1.8]
    Access rules: No rules defined
    External annotations: (None)
    Native library location: (None)
  resources.jar - C:\Program Files\Java\jre1.8.0_211\lib
  rt.jar - C:\Program Files\Java\jre1.8.0_211\lib
  jsse.jar - C:\Program Files\Java\jre1.8.0_211\lib
  jce.jar - C:\Program Files\Java\jre1.8.0_211\lib
  charsets.jar - C:\Program Files\Java\jre1.8.0_211\lib
  jfr.jar - C:\Program Files\Java\jre1.8.0_211\lib
  access-bridge-64.jar - C:\Program Files\Java\jre1.8.0_211\lib\ext
  cldrdata.jar - C:\Program Files\Java\jre1.8.0_211\lib\ext
  dnsns.jar - C:\Program Files\Java\jre1.8.0_211\lib\ext
  jaccess.jar - C:\Program Files\Java\jre1.8.0_211\lib\ext
  jfxrt.jar - C:\Program Files\Java\jre1.8.0_211\lib\ext
  localedata.jar - C:\Program Files\Java\jre1.8.0_211\lib\ext
  nashorn.jar - C:\Program Files\Java\jre1.8.0_211\lib\ext
  sunec.jar - C:\Program Files\Java\jre1.8.0_211\lib\ext
```

Enter a pattern for the rule.

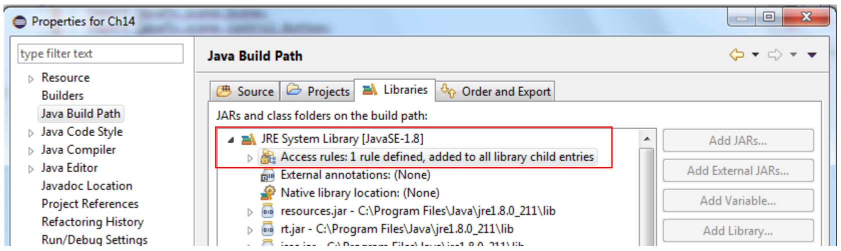
Resolution: Accessible

Rule Pattern: javafx/**

Allowed wildcards are '*', '?' and '**'. Pattern segments are separated by '/'. '**' matches any number of segments.
Examples are: 'java/util/**', '**/internal/**', 'org/e*/**'.

Using JavaFX in your Java project on Eclipse

The following figure shows that the access rule has been added successfully.
You can now use JavaFX in your Java project on Eclipse



Properties for Ch14

type filter text

- Resource
- Builders
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- Project References
- Refactoring History
- Run/Debug Settings

Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

```
JRE System Library [JavaSE-1.8]
  Access rules: 1 rule defined, added to all library child entries
  External annotations: (None)
  Native library location: (None)
  resources.jar - C:\Program Files\Java\jre1.8.0_211\lib
  rt.jar - C:\Program Files\Java\jre1.8.0_211\lib
```

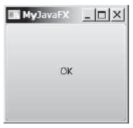
Add JARs... | Add External JARs... | Add Variable... | Add Library...

The Basic Structure of a JavaFX Program

- Every JavaFX program is defined in a in a class that extends **javafx.application.Application**, as shown in Listing 14.1.
- The **launch** method (line 22) is a static method defined in the **Application** class for launching a stand-alone JavaFX application.
- The **main** method (lines 21–23) is not needed if you run the program from the command line. It may be needed to launch a JavaFX program from an IDE with a limited JavaFX support.
- When you run a JavaFX application without a main method, JVM automatically invokes the **launch** method to run the application.
- The main class overrides the **start** method defined in **javafx.application.Application** (line 8).
- After a JavaFX application is launched, the JVM constructs an instance of the class using its **no-arg** constructor and invokes its **start** method.
- The **start** method normally places UI controls in a scene and displays the scene in a stage. A **Stage** object is a window.
- Line 10 creates a **Button** object and places it in a **Scene** object (line 11).
- A **Scene** object can be created using the constructor **Scene(node, width, height)**. This constructor specifies the width and height of the scene and places the node in the scene.

LISTING 14.1 MyJavaFX.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5
6  public class MyJavaFX extends Application {
7    @Override // Override the start method in the Application class
8    public void start(Stage primaryStage) {
9      // Create a scene and place a button in the scene
10     Button btOK = new Button("OK");
11     Scene scene = new Scene(btOK, 200, 250);
12     primaryStage.setTitle("MyJavaFX"); // Set the stage title
13     primaryStage.setScene(scene); // Place the scene in the stage
14     primaryStage.show(); // Display the stage
15   }
16
17   /**
18    * The main method is only needed for the IDE with limited
19    * JavaFX support. Not needed for running from the command line.
20    */
21   public static void main(String[] args) {
22     Application.launch(args);
23   }
24 }
```

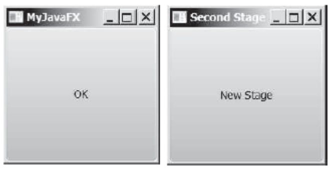


Multiple stages

- You can create additional stages if needed.
- The JavaFX program in Listing 14.2 displays two stages.
- Note that the main method is omitted in the listing.
- By default, the user can resize the stage. To prevent the user from resizing the stage, invoke **stage.setResizable(false)**.

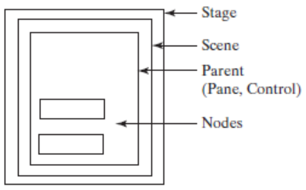
LISTING 14.2 MultipleStageDemo.java

```java
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
  }
}
```



Panes, UI Controls, and Shapes

- When you run MyJavaFX in Listing 14.1, the button is always centered in the scene and occupies the entire window no matter how you resize it. You can fix the problem by setting the position and size properties of a button.
- However, a better approach is to use container classes, called *panes*, for automatically laying out the nodes in a desired location and size.
- You place nodes inside a pane and then place the pane into a scene.
- A *node* is a visual component such as
  - a shape, (text, line, circle, ellipse, rectangle, arc, polygon, polyline, etc.)
  - an image view,
  - a UI control, (a label, button, check box, radio button, text field, text area, etc.)
  - a pane.
- A scene can be displayed in a stage.
- Note that a **Scene** can contain a **Control** or a **Pane**, but not a **Shape** or an **ImageView**.
- A **Pane** can contain any subtype of **Node**.

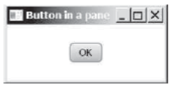


Using Pane: an example

- Listing 14.3 gives a program that places a button in a pane.
- The program creates a **StackPane** (line 11) and adds a button as a child of the pane (line 12).
- The **getChildren()** method returns an instance of **javafx.collections.ObservableList**.
- **ObservableList** behaves very much like an **ArrayList** for storing a collection of elements.
- Invoking **add(e)** adds an element to the list.
- The **StackPane** places the nodes in the center of the pane on top of each other. Here, there is only one node in the pane.
- The **StackPane** respects a node's preferred size. So you see the button displayed in its preferred size.

LISTING 14.3 ButtonInPane.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.stage.Stage;
5  import javafx.scene.layout.StackPane;
6
7  public class ButtonInPane extends Application {
8    @Override // Override the start method in the Application class
9    public void start(Stage primaryStage) {
10     // Create a scene and place a button in the scene
11     StackPane pane = new StackPane();
12     pane.getChildren().add(new Button("OK"));
13     Scene scene = new Scene(pane, 200, 50);
14     primaryStage.setTitle("Button in a pane"); // Set the stage title
15     primaryStage.setScene(scene); // Place the scene in the stage
16     primaryStage.show(); // Display the stage
17   }
18 }
```



Using Pane: another example

- Listing 14.4 gives an example that displays a circle in the center of the pane.
- The program creates a **Circle** (line 12) and sets its center at (100, 100) (lines 13–14), which is also the center for the scene, since the scene is created with the width and height of 200 (line 24). Note that the measurement units for graphics in Java are all in *pixels*.
- Note that the coordinates of the upper left corner of the pane is (**0**, **0**) in the Java coordinate system

LISTING 14.4 ShowCircle.java

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.Pane;
4  import javafx.scene.paint.Color;
5  import javafx.scene.shape.Circle;
6  import javafx.stage.Stage;
7
8  public class ShowCircle extends Application {
9    @Override // Override the start method in the Application class
10   public void start(Stage primaryStage) {
11     // Create a circle and set its properties
12     Circle circle = new Circle();
13     circle.setCenterX(100);
14     circle.setCenterY(100);
15     circle.setRadius(50);
16     circle.setStroke(Color.BLACK);
17     circle.setFill(Color.WHITE);
18
19     // Create a pane to hold the circle
20     Pane pane = new Pane();
21     pane.getChildren().add(circle);
22
23     // Create a scene and place it in the stage
24     Scene scene = new Scene(pane, 200, 200);
25     primaryStage.setTitle("ShowCircle"); // Set the stage title
26     primaryStage.setScene(scene); // Place the scene in the stage
27     primaryStage.show(); // Display the stage
28   }
29 }
```

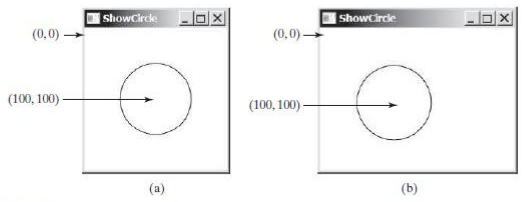


FIGURE 14.5 (a) A circle is displayed in the center of the scene. (b) The circle is not centered after the window is resized.

The Java Coordinate System

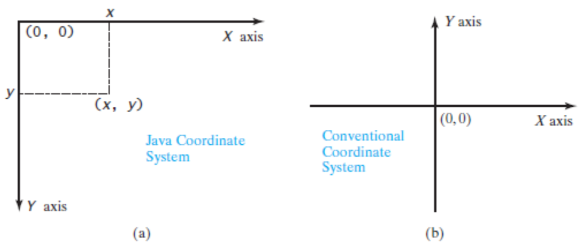
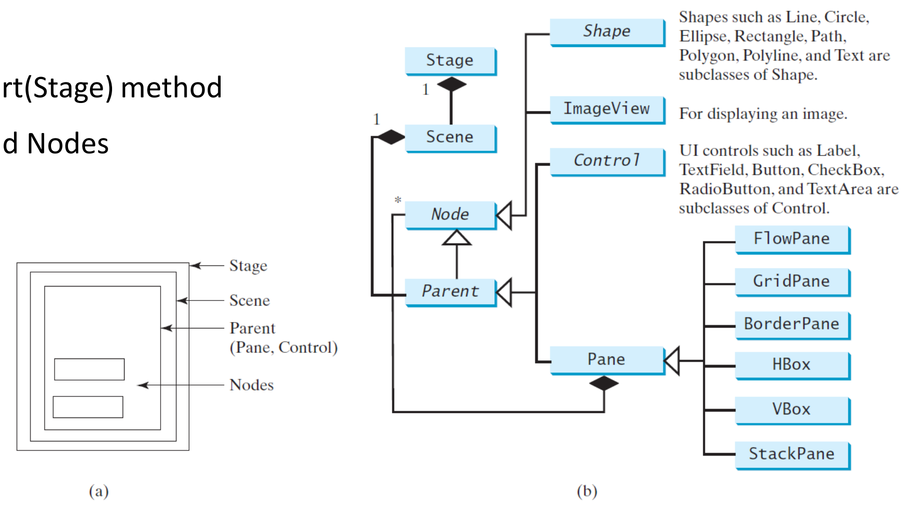


Figure 14.6 The Java coordinate system is measured in pixels, with (0, 0) at its upper-left corner.

Basic Structure of JavaFX

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


Stage
Scene
Parent
(Pane, Control)
Nodes
(a)
Stage
1
1
Scene
*
Node
Parent
Shape
Shapes such as Line, Circle, Ellipse, Rectangle, Path, Polygon, Polyline, and Text are subclasses of Shape.
ImageView
For displaying an image.
Control
UI controls such as Label, TextField, Button, CheckBox, RadioButton, and TextArea are subclasses of Control.
Pane
FlowPane
GridPane
BorderPane
HBox
VBox
StackPane
(b)

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.	See Listing 14.4
StackPane	Places the nodes on top of each other in the center of the pane.	See Listing 14.3
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.	

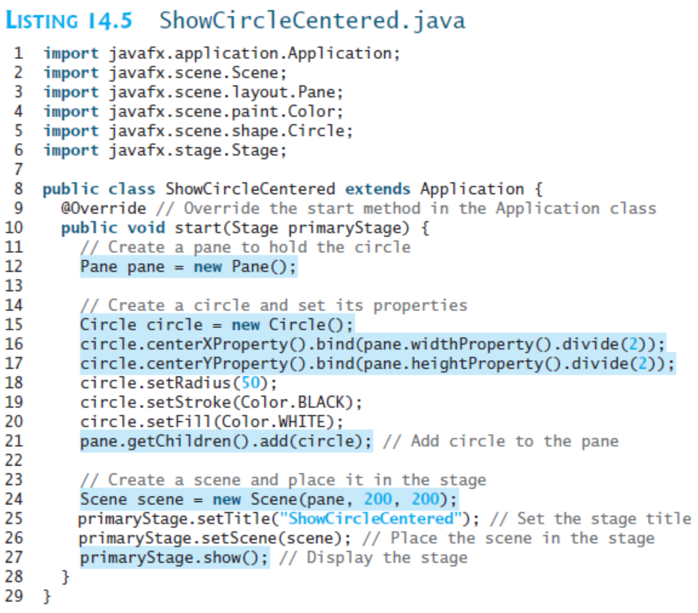
Property Binding

- *You can bind a target object to a source object. A change in the source object will be automatically reflected in the target object.*
- The target object is called a *binding object* or a *binding property* and the source object is called a *bindable object* or *observable object*.
- As discussed in the preceding listing, the circle is not centered after the window is resized.
- In order to display the circle centered as the window resizes, the *x*- and *y*-coordinates of the circle center need to be reset to the center of the pane.
- This can be done by binding the **centerX** with pane's **width/2** and **centerY** with pane's **height/2**, as shown in Listing 14.5.

Property Binding: an example

- A target binds with a source using the **bind** method as follows:

  target.bind(source);



LISTING 14.5 ShowCircleCentered.java

```
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 {
  @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 **Color** Class

- A color instance can be constructed using the following constructor:

  public Color(**double** r, **double** g, **double** b, **double** opacity);

  - in which **r**, **g**, and **b** specify a color by its red, green, and blue components with values in the range from **0.0** (darkest shade) to **1.0** (lightest shade).
  - The **opacity** value defines the transparency of a color within the range from **0.0** (completely transparent) to **1.0** (completely opaque).
  - This is known as the RGBA model. For example,
    Color color = **new** Color(**0.25**, **0.14**, **0.333**, **0.51**);
- Once a **Color** object is created, its properties cannot be changed. The **brighter()** method returns a new **Color** with a larger red, green, and blue values and the **darker()** method returns a new **Color** with a smaller red, green, and blue values.
- Alternatively, you can use one of the many standard colors such as **BEIGE**, **BLACK**, **BLUE**, **BROWN**, **CYAN**, **DARKGRAY**, **GOLD**, **GRAY**, **GREEN**, **LIGHTGRAY**, **MAGENTA**, **NAVY**, **ORANGE**, **PINK**, **RED**, **SILVER**, **WHITE**, and **YELLOW** defined as constants in the **Color** class. The following code, for instance, sets the fill color of a circle to red:

  circle.setFill(Color.RED);

The **Font** Class

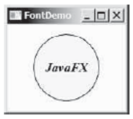
- *A* **Font** *describes font name, weight, and size.*
- A **Font** instance can be constructed using its constructors.
- Once a **Font** object is created, its properties cannot be changed
- A **Font** is defined by its name, weight, posture, and size. Times, Courier, and Arial are the examples of the font names. You can obtain a list of available font family names by invoking the static **getFamilies()** method.
- The font postures are two constants: **FontPosture.ITALIC** and **FontPosture.REGULAR**. For example, the following statements create two fonts.

```
Font font1 = new Font("SansSerif", 16);
Font font2 = Font.font("Times New Roman", FontWeight.BOLD, FontPosture.ITALIC, 12);
```

Font Demo

- Listing 14.8 gives a program that displays a label using the font (Times New Roman, bold, italic, and size 20), as shown in Figure 14.11.

FIGURE 14.11



- The program creates a **StackPane** (line 14) and adds a circle and a label to it (lines 21, 27). These two statements can be combined using the following one statement:

```
pane.getChildren().addAll(circle, label);
```

- A **StackPane** places the nodes in the center and nodes are placed on top of each other.
- As you resize the window, the circle and label are displayed in the center of the window, because the circle and label are placed in the stack pane.
- Stack pane automatically places nodes in the center of the pane.

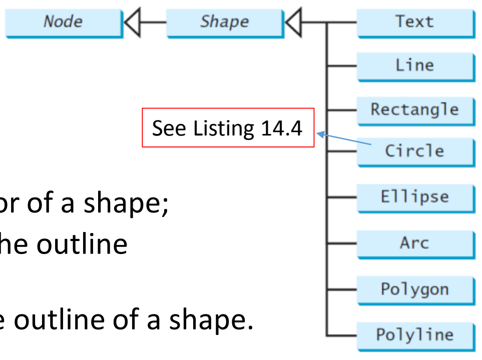
LISTING 14.8 FontDemo.java

```
 1  import javafx.application.Application;
 2  import javafx.scene.Scene;
 3  import javafx.scene.layout.*;
 4  import javafx.scene.paint.Color;
 5  import javafx.scene.shape.Circle;
 6  import javafx.scene.text.*;
 7  import javafx.scene.control.*;
 8  import javafx.stage.Stage;
 9
10  public class FontDemo extends Application {
11    @Override // Override the start method in the Application class
12    public void start(Stage primaryStage) {
13      // Create a pane to hold the circle
14      Pane pane = new StackPane();
15
16      // Create a circle and set its properties
17      Circle circle = new Circle();
18      circle.setRadius(50);
19      circle.setStroke(Color.BLACK);
20      circle.setFill(new Color(0.5, 0.5, 0.5, 0.1));
21      pane.getChildren().add(circle); // Add circle to the pane
22
23      // Create a label and set its properties
24      Label label = new Label("JavaFX");
25      label.setFont(Font.font("Times New Roman",
26        FontWeight.BOLD, FontPosture.ITALIC, 20));
27      pane.getChildren().add(label);
28
29      // Create a scene and place it in the stage
30      Scene scene = new Scene(pane);
31      primaryStage.setTitle("FontDemo"); // Set the stage title
32      primaryStage.setScene(scene); // Place the scene in the stage
33      primaryStage.show(); // Display the stage
34    }
35  }
```

Shapes

- JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines.
- The **Shape** class is the abstract base class that defines the common properties for all shapes.
- Among them are the **fill**, **stroke**, and **strokeWidth** properties.
  - **fill** property specifies a color that fills the interior of a shape;
  - **stroke** property specifies a color used to draw the outline of a shape;
  - **strokeWidth** property specifies the width of the outline of a shape.

A shape is a node. The **Shape** class is the root of all shape classes



See Listing 14.4

Shapes: Text

- The **Text** class defines a node that displays a string at a starting point (**x**, **y**), as shown in Figure 14.27a.
- A **Text** object is usually placed in a pane. The pane's upper-left corner point is (**0**, **0**) and the bottom-right point is (**pane.getWidth()**, **pane.getHeight()**).
- A string may be displayed in multiple lines separated by **\n**.

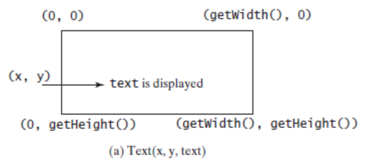
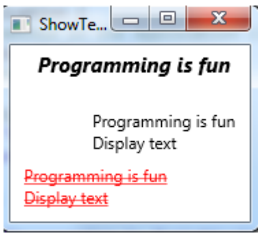


FIGURE 14.27 A Text object is created to display a text.

Shapes: Text – sample code

```
Text text1 = new Text(20, 20, "Programming is fun");
text1.setFont(Font.font("Courier", FontWeight.BOLD, FontPosture.ITALIC, 15));
pane.getChildren().add(text1);

Text text2 = new Text(60, 60, "Programming is fun\nDisplay text");
pane.getChildren().add(text2);

Text text3 = new Text(10, 100, "Programming is fun\nDisplay text");
text3.setFill(Color.RED);
text3.setUnderline(true);
text3.setStrikethrough(true);
pane.getChildren().add(text3);
```



Programming is fun

Programming is fun
Display text

~~Programming is fun~~
~~Display text~~

Three text objects are displayed

- The code creates a **Text**, sets its font, and places it to the pane.
- The code then creates another **Text** with multiple lines and places it to the pane.
- The code then creates the third **Text**, sets its color, sets an underline and a strike through line, and places it to the pane .

Shapes: Line

- The **Line** class defines a line that connects two points with four parameters **startX**, **startY**, **endX**, and **endY**, as shown in Figure 14.29a.
- For example,

```
Line line1 = new Line(10, 10, 50, 50);
line1.setStrokeWidth(5);
line1.setStroke(Color.GREEN);
```

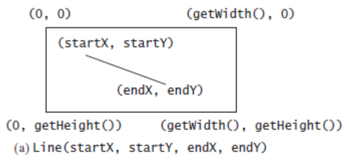
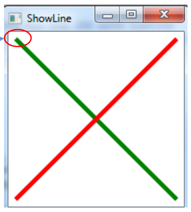


FIGURE 14.29 A Line object is created to display a line.

Shapes: Line – sample code

```
Line line1 = new Line(10, 10, 10, 10);
line1.endXProperty().bind(widthProperty().subtract(10));
line1.endYProperty().bind(heightProperty().subtract(10));
line1.setStrokeWidth(5);
line1.setStroke(Color.GREEN);
pane.getChildren().add(line1);

Line line2 = new Line(10, 10, 10, 10);
line2.startXProperty().bind(pane.widthProperty().subtract(10));
line2.endYProperty().bind(pane.heightProperty().subtract(10));
line2.setStrokeWidth(5);
line2.setStroke(Color.RED);
pane.getChildren().add(line2);
```

Where **startX**, **startY**, **endX**, and **endY** with value of 10



Two lines are displayed across the pane.

Creates two lines and binds the starting and ending points of the line with the width and height of the pane so that the two points of the lines are changed as the pane is resized.

Shapes: Rectangle

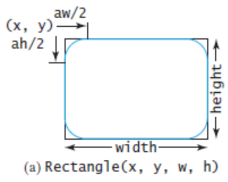
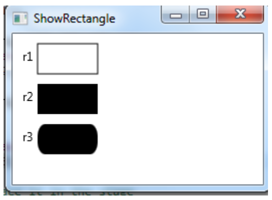


FIGURE 14.31 A Rectangle object

- The **Rectangle** class defines a rectangle with parameters **x**, **y**, **width**, **height**, **arcWidth**, and **arcHeight**, as shown in Figure 14.31a.
- The rectangle's upper-left corner point is at (**x**, **y**) and parameter **aw** (**arcWidth**) is the horizontal diameter of the arcs at the corner, and **ah** (**arcHeight**) is the vertical diameter of the arcs at the corner.
- By default, the fill color is black. So a rectangle is filled with black color. The stroke color is white by default.

Shapes: Rectangle – sample code

```
// Create rectangles and add to pane
Rectangle r1 = new Rectangle(25, 10, 60, 30);
r1.setStroke(Color.BLACK);
r1.setFill(Color.WHITE);
pane.getChildren().add(new Text(10, 27, "r1"));
pane.getChildren().add(r1);

Rectangle r2 = new Rectangle(25, 50, 60, 30);
pane.getChildren().add(new Text(10, 67, "r2"));
pane.getChildren().add(r2);

Rectangle r3 = new Rectangle(25, 90, 60, 30);
r3.setArcWidth(15);
r3.setArcHeight(25);
pane.getChildren().add(new Text(10, 107, "r3"));
pane.getChildren().add(r3);
```



Multiple rectangles are displayed.

- Sets stroke color of rectangle **r1** to black and fill color of rectangle r1 to white.
- Creates rectangle **r3** and sets its arc width and arc height so that **r3** is displayed as a rounded rectangle.

Chapter Summary

- JavaFX is the framework for developing rich Internet applications.
- A main JavaFX class must extend **javafx.application.Application** and implement the **start** method.
- A stage is a window for displaying a scene.
- A **Scene** can contain a **Control** or a **Pane**, but not a **Shape** or an **ImageView**.
- Panes can be used as the containers for nodes (**Shape, ImageView**).
- JavaFX provides many shape classes for drawing texts, lines, circles, rectangles, ellipses, arcs, polygons, and polylines