1 **JavaFX Basics**

CST141

2 **Console vs. Window Programs**

- In Java there is no distinction between *console* programs and *window* programs
  - Java applications can mix (combine) console and window elements
  - Lower case "w" above indicates that windows are not unique to Microsoft, e.g Linux and Mac (remember that Java is transportable)
- The Java Virtual Machine (JVM) always has a command window (console) running

5 **The Application Class (Page 1)**

- The JavaFX classes give developers the flexibility to create *customized* GUI windows
  - Beyond just JOptionPane.showInputDialog() and JOptionPane.showMessageDialog()
- Any class that extends Application is a "JavaFX" application
- Located in the javafx.application package:
  import javafx.application.Application;

6 **The Application Class (Page 2)**

- Format:
  public class *ClassName* extends Application
  { …
- Example:
  public class JavaFX1 extends Application
  { …

7 **The Stage Class (Page 1)**

- The JavaFX Stage class is the "top-level" container for all applications
- Defines a "window" with *Title bar*, and *Minimize*, *Maximize* and *Close* buttons
  - Just the "frame" around it but nothing that is in it
- The window contains the application
- Located in the javafx.stage package:
  import javafx.stage.Stage;

8 **The Stage Class (Page 2)**

- Any class that extends class Application takes a Stage object *parameter* (called the "primary stage") in its start() method:
  public void start(Stage primaryStage)
  { …

9 **The Scene Class (Page 1)**

- A Scene object is the container for all content placed inside the Stage object
  - Placed inside the Stage
- Its content is represented as a hierarchical "scene graph" of nodes (textboxes, buttons, radio buttons, checkboxes, layout managers, etc.)

- Located in the javafx.scene package:
  import javafx.scene.Scene;

10 **The Scene Class (Page 2)**

- A Scene object is instantiated in one of two ways:
  Scene *object* = new Scene(*parent*);
  Scene *object* = new Scene(*parent*, *width*, *height*);
  – *parent* is the object placed into the Scene object (a UI control, a shape, an ImageView or a layout pane)
  – *width* and *height* are the size of the Scene object and therefore its size within the "window"
    • If size is specified, *parent* takes size of scene
    • If size is not specified, scene takes size of *parent*

11 **The Scene Class (Page 3)**

- Examples:
  Scene scene = new Scene(buttonOK, 300, 300);
  Scene scene = new Scene(pane);
  – Whichever object is placed into a Scene totally *fills* that Scene object if width and height are specified
  – If width and height are not specified, the Scene takes the size of the object placed into it

12 **The start() Method (Page 1)**

- When a JavaFX application is launched, the Java Virtual Machine (JVM) *automatically*:
  – Instantiates an object (instance) of the class
  – Launches the start() method (which is why a main() method usually is not required)
  – Passes a Stage object argument (the "primary stage") to the parameter of the start() method

13 **The start() Method (Page 2)**

- The start() method header (signature):
  @Override
  public void start(Stage primaryStage)
  { …
  – *Overrides* abstract start() method from the class Application

14 **The setTitle() Method**

- A method of a Stage object that displays a *title bar* message in the window (Stage)
- Format:
  stageObject.setTitle(*titleString*);
- Example:
  primaryStage.setTitle("JavaFX No. 1");

15 **The setScene() Method**

- A method of a Stage object that places a Scene object in the window (Stage)
- Format:
  stageObject.setScene(*sceneObject*);
- Example:
  primaryStage.setScene(scene);

16 **The show() Method**

- A method of a Stage object that displays the window (Stage)
- Format:
  stageObject.show();
- Example:
  primaryStage.show();

17 **Nodes**

- Nodes are the content which are placed into a Scene object (which is placed in a Stage object) and can include:
  – Control objects—(UI—user interface—controls) such, e.g. TextFields, Buttons, etc.
  – Pane objects—*layout* panes (containers) that are used for positioning objects in a scene
  – Shape objects—lines, circles, text, etc.
  – ImageView controls—for displaying images (files)

18 **The Button Class (Page 1)**

- Button is a JavaFX class used to instantiate UI (user interface) control *command button* objects
- If "event listening" has been activated for a Button control, generates an ActionEvent when the user clicks the button
- Text on the button is called a *button label*
- Located in the javafx.scene.control package:
  import javafx.scene.control.Button;

19 **The Button Class (Page 2)**

- Format to declare a Button object:
  Button *buttonObject* = new Button([*buttonLabel*]);
  – *buttonLabel*—optional argument which is the string that appears as a caption on the button
- Examples:
  Button buttonOK = new Button();
  Button buttonOK = new Button("OK");

20 **The setText() Method**

- Assigns a string value to an object, e.g. a Button if the button label was not assigned when the object was instantiated
- Format:

object.setText(*text*);

- Example:

  buttonOK.setText("OK");

22 **Starting a New JavaFX Project**

- Start a new project as usual
  - In the "Categories" window select "JavaFX"
  - In the "Projects" window select "JavaFXApplication"
- Click the <Next> button as usual
  - <Browse…> to the "Project Location"
  - Assign a "Project Name"
  - Click the <Finish> button

24 **Layout Classes (Page 1)**

- Layout "panes" are container classes (Pane class and its subclasses) that give developers greater control of how nodes are *positioned* within scenes
- Nodes are placed into a pane and then the pane placed into the scene

25 **Layout Classes (Page 2)**

- There are several Pane classes:
  - Pane—the base (super) class for layout panes
  - StackPane—places nodes one on top of another centered in the pane
  - FlowPane—places nodes row-by-row horizontally or column-by-column vertically
  - GridPane—places nodes in two-dimensional grid
  - BorderPane—places nodes in top, right, bottom, left and center regions
  - HBox—places nodes in a single row
  - VBox—places nodes in a single column

26 **The StackPane Class (Page 1)**

- The StackPane layout places in a single "stack" one on top of another
- Provides an easy way to overlay text on a shape or image or to overlap common shapes to create a complex shape
- Default alignment is centered (HPos.CENTER and VPos.CENTER)
- Located in the javafx.scene.layout package:

  import javafx.scene.layout.StackPane;

27 **The StackPane Class (Page 2)**

- Format:

  StackPane *stackPaneObject* = new StackPane([*children*]);
  - *children* are one or more nodes that are initially added to the StackPane
- Examples:

  StackPane pane = new StackPane();

  StackPane pane = new StackPane(new Button(), new Button);

28 **The getChildren() Method (Page 1)**

- Returns the list (*collection*) of children (Node's) for this layout Parent (layout object)
  - Specifically an ObservableList<Node> of the children of this parent
- Inherited from superclass Pane
- The method is required for many (though not all) layout pane classes to be able to add new nodes to those layouts
  - E.g. Pane, FlowPane, StackPane, HBox, VBox

29 **The getChildren() Method (Page 2)**

- Format:
  paneObject.getChildren()
- Example:
  stack.getChildren().add(new Button("OK"));

30 **The add() Method**

- A member of the getChildren() *collection* that adds a node (UI control, shape, pane or image view) to the collection of children in a layout object
- Format:
  paneObject.getChildren().add(*node*);
- Example:
  stack.getChildren().add(new Button("OK"));

31 **The addAll() Method**

- Adds a *series of nodes* (two or more nodes) to the collection of children in a layout object
- Format:
  paneObject.getChildren().addAll(*node1*, *node2*, …);
- Example:
  stack.getChildren().addAll(new Button("OK"), new Button("Cancel"));

32 **The setResizable() Method**

- Method of class Stage that determines if window may be resized
  - By dragging the mouse on one of its borders
- Sets boolean value—default is true
- Format:
  stageObject.setResizable(true/false);
- Example:
  stage.setResizable(false);

34 **The GridPane Class (Page 1)**

- Layout pane that allows the developer to create a flexible grid of *row and columns* to layout nodes
- Nodes can be placed in any cell as needed
  - The GridPane grows horizontally and/or vertically as new cells are added

- Located in the javafx.scene.layout package:
  import javafx.scene.layout.GridPane;

35 **The GridPane Class (Page 2)**

- Constructor format:
  GridPane *gridPaneObject* = new GridPane();
  – Constructor is tot overloaded; sets the layout with hgap/vgap = 0 and TOP_LEFT alignment
- Example:
  GridPane grid = new GridPane();

36 **The add() Method for GridPane**

- The add() method for GridPane adds the node at the specified *columnIndex* and *rowIndex* position
  – Does not use method getChildren()
- Format:
  gridPaneObject.add(*node*, *columnIndex*, *rowIndex*);
  – The *node* is the object added in that cell
- Example:
  grid.add(new TextField(), 1, 0);

37 **The HPos Class Constants**

- A set of Enum's (enumerated *constants*) used for describing horizontal positioning and alignment of JavaFX objects
- Examples:
  – HPos.CENTER
  – HPos.LEFT
  – HPos.RIGHT
- Located in the javafx.geometry package:
  import javafx.geometry.HPos;

38 **The VPos Enum Constants**

- A set of Enum's (enumerated constants) used for describing vertical positioning and alignment of JavaFX objects
- Examples:
  – VPos.BASELINE
  – VPos.BOTTOM
  – VPos.CENTER
  – VPos.TOP
- Located in the javafx.geometry package:
  import javafx.geometry.VPos;

39 **The setHalignment() Method**

- A static method that sets the horizontal alignment for a node contained within a

GridPane object
- Can call method either *before or after* adding node to the GridPane
- Format:
  GridPane.setHalignment(*node*, HPos.CENTER | HPos.LEFT | HPos.RIGHT);
- Example:
  GridPane.setHalignment(buttonAdd, HPos.CENTER);

40 **The setValignment() Method**
- A static method that sets the vertical alignment for a node contained within a GridPane object
- Can call method either *before or after* adding node to the GridPane
- Format:
  GridPane.setValignment(*node*, VPos.BASELINE | VPos.BOTTOM | VPos.CENTER | VPos.TOP);
- Example:
  GridPane.setValignment(buttonSubtract, VPos.BASELINE);

41 **The setHgap() Method**
- A method that sets the width measured in pixels of the horizontal gaps between *columns* in a GridPane object
- Format:
  gridPaneObject.setHgap(*doubleValue*);
- Example:
  grid.setHgap(2);

42 **The setVgap() Method**
- A method that sets the height measured in pixels of the vertical gaps between *rows* in a GridPane object
- Format:
  gridPaneObject.setVgap(*doubleValue*);
- Example:
  grid.setVgap(5);

43 **JavaFX UI Controls (Page 1)**
- Windows-type UI (user interface) *controls* with which users interact by using the mouse or keyboard
- Some basic JavaFX component classes are:
  – Button—a command button that when *clicked* can trigger an "event"
  – TextField—an input textbox
  – Label—a text element that is *not editable*

44 **JavaFX UI Controls (Page 2)**
- Some basic JavaFX component classes (*con*.):
  – CheckBox—clicked "on" and "off"
  – ComboBox—drop-down list of choices

– List—open "scrollable" area with list of choices

45 **The TextField Class (Page 1)**

- A TextField is a UI control class used to instantiate *text field* objects
- A text field is a single-line text box into which a user may type text from the keyboard
- Located in the javafx.scene.control package:
  import javafx.scene.control.TextField;

46 **The TextField Class (Page 2)**

- Format to declare a TextField object:
  TextField *textFieldObject* = new TextField([*text*]);
  – *text* is default string displayed in the text field (optional—setText() method may be called later)
- Examples:
  TextField firstText = new TextField();
  TextField secondText = new TextField("Enter first number");

47 **The Label Class (Page 1)**

- A Label is UI control class used to instantiate text objects which are *not editable*
- Often used "nearby" another control in window to *indicate its purpose*
- Located in the javafx.scene.control package:
  import javafx.scene.control.Label;

48 **The Label Class (Page 2)**

- Format to declare a Label object:
  Label *labelObject* = new Label([*textString* [, *graphicObject*]]);
  – *textString* is the text displayed for the label
  – *graphicObject* is an optional image displayed with the text
  – Overloaded so that any (and/or all) arguments to the constructor are *optional*

49 **The Label Class (Page 3)**

- Format to declare a Label object:
  Label *labelObject* = new Label([*textString* [, *graphicObject*]]);
- Examples:

```
Label label1 =
    new Label("Enter first number");
Label label2 = new Label();
```

51 **The BorderPane Class (Page 1)**

- A BorderPane has five areas
  – TOP
  – LEFT
  – CENTER
  – RIGHT
  – BOTTOM

- Located in the javafx.scene.layout package
  import javafx.scene.layout.BorderPane;

52 **The BorderPane Class (Page 2)**

- Only one node can be inserted into each area
- If the window is enlarged, the CENTER area gets as much of the available space as possible
  – The other areas expand only as much as necessary to fill all available space
- Sometimes not all the areas of a BorderLayout object are used—perhaps just the top, left and right

53 **The BorderPane Class (Page 3)**

- Constructor format:
  BorderPane *borderLayoutObject* = new BorderPane([centerNode, [*topNode*, *rightNode*, *bottomNode*, *leftNode*]]);
  – The optional *node* objects can be added to the areas when an object first is instantiated

54 **The BorderPane Class (Page 4)**

- Examples:
  BorderPane border = new BorderPane();
  BorderPane border = new BorderPane(new Button(), new TextField(), new Button(), new TextField(), new Button());

55 **The "set" Methods for BorderPane**

- The "set" methods for BorderPane specify which of the five areas to place the node
- Formats:
  borderPaneObject.setTop(*node*);
  borderPaneObject.setLeft(*node*);
  borderPaneObject.setCenter(*node*);
  borderPaneObject.setRight(*node*);
  borderPaneObject.setBottom(*node*);
- Example:
  border.setTop(new TextField("Enter your age here "));

57 **The HBox Class (Page 1)**

- A layout control that places nodes in horizontally a single row (unlimited number of nodes as needed)
- Located in the javafx.scene.layout package
  import javafx.scene.layout.HBox;

58 **The HBox Class (Page 2)**

- Constructor format:
  HBox *hBoxObject* = new HBox([*spacing*,] [*children*]);
  – *spacing* is the horizontal spacing between nodes

– *children* are one or more nodes that are initially added to the HBox

59 **The HBox Class (Page 3)**

- Examples:

```
HBox buttons = new Hbox();
HBox buttons = new Hbox(20);
HBox buttons = new Hbox(new Button() );
HBox buttons = new Hbox(20, new Button() );
```

60 **The VBox Class (Page 1)**

- A layout control that places nodes vertically in a single column (unlimited number of nodes as needed)
- Located in the javafx.scene.layout package

```
import javafx.scene.layout.VBox;
```

61 **The VBox Class (Page 2)**

- Constructor format:

VBox *vBoxObject* = new VBox([*spacing*,] [*children*]);

– *spacing* is the vertical spacing between nodes

– *children* are one or more nodes that are initially added to the VBox

62 **The VBox Class (Page 3)**

- Examples:

```
VBox buttons = new Vbox();
VBox buttons = new Vbox(5);
VBox buttons = new Vbox(new Label() );
VBox buttons = new Vbox(5, new Label() );
```

63 **Adding Elements to HBox and VBox**

- Use the getChildren().add() method to add a node or other elements to add a node to the collection of children of an HBox or VBox object
  – Or use getChildren().addAll()
- Format:

hBoxObject | *vBoxObject*.getChildren().add(*node*);

- Example:

```
labels.getChildren().add( new Label() );
```

64 **Nesting Pane Nodes**

- One layout pane can be *nested* inside an outer layout pane
- Placing one layout pane *inside another* gives greater control and flexibility for scene layouts
- Example:

```
GridPane grid = new GridPane();
HBox buttons = new HBox(20);
buttons.getChildren().addAll(       new Button("+"), new Button("-"), new Button("*"),
```

```
            new Button("/") );
        grid.add(buttons, 1, 2);
```

70 **The Pane Class (Page 1)**

- Super class of all layout panes
  - Not abstract so Pane objects can be instantiated
- Often used for drawing JavaFX graphic objects (good for *absolute* positioning)
- Located in the javafx.scene.layout package

```
import javafx.scene.layout.Pane;
```

71 **The Pane Class (Page 2)**

- Constructor format:

  Pane *paneObject* = new Pane([*children*]);
  - *children* are one or more nodes that are initially added to the Pane
- Example:

```
Pane pane = new Pane();
Pane pane = new Pane( new Line(), new Circle() );
```

72 **The Pane Class (Page 3)**

- Use method getChildren().add() to add nodes to a Pane:

```
pane.getChildren().add( new Button("OK") );
```

- Or use getChildren().addAll()

73 **JavaFX Graphics**

- JavaFX provides an API to draw shapes
  - Both 2D and 3D shapes
- Some 2D shape classes:
  - Line
  - Circle
  - Rectangle
  - Ellipse
  - Arc
  - Polygon
  - Text (drawn as a graphic)

74 **The Line Class (Page 1)**

- Draws a *line* at specific start and end x- and y-coordinate locations
- Direct subclass of abstract class Shape
- Located in the javafx.scene.shape package

```
import javafx.scene.shape.Line;
```

75 **The Line Class (Page 2)**

- Format:

  Line *lineObject* = new Line([*startX*, *startY*, *endX*, *endY*]);
  - *x* and *y* are the start x- and y-coordinates

– *x* and *y* are the end x- and y-coordinates

- Examples:

```
Line line = new Line();
Line line = new Line(20, 50, 40, 100);
```

76 **The Line Class (Page 3)**

- There are *set* methods to set the startX, startY, endX and endY properties of a Line object
- Format:

lineObject.set*CoordinateProperty*(*value*);

– *value* is the start or end property's x- or y-coordinate

- Examples:

```
line.setStartX(20);
line.setStartY(50);
line.setEndX(40);
line.setEndy(100);
```

77 **The setStrokeWidth() Method**

- Sets the stroke width (thickness) of the border of a graphic object
- Inherited from direct superclass Shape

– Applied to any JavaFX shape that takes a stroke width

- Format:

shapeObject.setStrokeWidth(*width*);

- Example:

```
circle.setStrokeWidth(5);
```

78 **The setStroke() Method**

- Sets stroke color of the border of graphic object
- Inherited from direct superclass Shape

– Applied to any JavaFX shape that takes a stroke color

- Format:

shapeObject.setStroke(*color*);

- Example:

```
circle.setStroke(Color.BLUE);
```

79 **The Color Class (Page 1)**

- Creates an Color object that uses the RGB model (values between zero (0%) to 1.0 (100%)

– Represents amount of red, green and blue in makeup of color)

- Located in the javafx.scene.paint package:

```
import javafx.scene.paint.Color;
```

80 **The Color Class (Page 2)**

- Format to declare a Color object:

Color *colorObject* = new Color(*red*, *green*, *blue*, [*opacity*]);

- Examples:

```
Color color = new Color(1.0, 0.0, 0.0);
circle.setFill(color);

circle.setFill( new Color(1.0, 1.0, 0.0) );
```

81 **The Color Constants**

- A series of standard static constants of class Color that return a hexadecimal value representing an RGB color
- A few examples of these constants are BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE and YELLOW
- Example:

```
circle.setFill(Color.PEACHPUFF);
```

85 **The Circle Class (Page 1)**

- Draws a *circle* of specific x- and y-coordinates (center of the circle), and radius
- Circle and several other JavaFX graphic objects are direct subclasses of abstract class Shape
- Located in the javafx.scene.shape package

```
import javafx.scene.shape.Circle;
```

86 **The Circle Class (Page 2)**

- Format:

  Circle *circleObject* = new Circle([[*centerX*, *centerY*,] *radius*]);

  – *centerX* and *centerY* are the x- and y-coordinates of the center of the circle

  – *radius* is the measurement from the center of the circle to its outer edge
- Examples:

```
Circle circle = new Circle();
Circle circle = new Circle(75);
Circle circle = new Circle(50, 25, 75);
```

87 **The Circle Class (Page 3)**

- There are *set* methods to set the centerX and centerY properties of a Circle object
- Formats:

  circleObject.setCenterX(*value*);

  circleObject.setCenterY(*value*);

  – *value* is center x- or y-coordinate property of the Cricle
- Examples:

```
circle.setCenterX(400);
circle.setCenterY(300);
```

88 **The Circle Class (Page 4)**

- There is a *set* method to set the radius property of a Circle object

- Format:
  circleObject.setRadius(*radius*);
  – *value* is the radius property for the Circle
- Example:
  circle.setRadius(75);

89 **The setFill() Method**

- Sets the fill color of a Circle or other object
- Inherited from direct superclass Shape
  – May be applied to any JavaFX shape object that takes a fill color
- Format:
  shapeObject.setFill(*color*);
- Example:
  circle.setFill(Color.PEACHPUFF);

82 **Binding (Page 1)**

- Most properties of most nodes have corresponding *property methods*, e.g.:
  – textProperty()
  – radiusProperty()
  – widthProperty()
  – endXProperty()
- JavaFX property binding allows synchronizing the property values of two nodes so that:
  – Whenever one of the property value changes
  – The value of the other property is updated automatically

83 **Binding (Page 2)**

- The bind() method for a property *synchronizes* that property to the property (or other value) of another node
- Format:
  node1.property().bind(*node2.property*());
  – *method()* represents and returns any value associated with *node2*
- Example:
  label1.textProperty().bind(textField1.textProperty());

84 **JavaFX Properties and Arithmetic**

- Methods add(), subtract(), multiply() and divide() are used in arithmetic operations with JavaFX properties
- Formats:
  node.property().add(*doubleValue*)
  node.property().subtract(*doubleValue*)
  node.property().multiply(*doubleValue*)
  node.property().divide(*doubleValue*)
- Example:

```
line.endXProperty().bind( pane.widthProperty().subtract(100) );
```

90 **The Rectangle Class (Page 1)**

- Draws a *rectangle* at specific upper left x- and y-coordinates, and of specified width and height
- Direct subclass of abstract class Shape
- Located in the javafx.scene.shape package

```
import javafx.scene.shape.Rectangle;
```

91 **The Rectangle Class (Page 2)**

- Format:

```
Rectangle rectangleObject = new Rectangle( [x, y,] width, height);
```

  – *x* and *y* are the x- and y-coordinates of the upper-left corner of the rectangle
  – *width* and *height* are the width and height of rectangle

- Example:

```
Rectangle rectangle = new Rectangle();
Rectangle rectangle = new Rectangle(60, 150);
Rectangle rectangle = new Rectangle(125, 350, 60, 150);
```

92 **The setArcWidth() Method**

- Sets *horizontal diameter* of arc (its radius) at the four corners of the rectangle
- Format:

```
rectangleObject.setArcWidth(doubleValue);
```

- Example:

```
rectangle.setArcWidth(15);
```

93 **The setArcHeight() Method**

- Sets *vertical diameter* of arc (its radius) at the four corners of the rectangle
- Format:

```
rectangleObject.setArcHeight(doubleValue);
```

- Example:

```
rectangle.setArcHeight(30);
```

94 **The setRotate() Method**

- Sets the angle of "clockwise" rotation around a shape's center measured in degrees
- Inherited from its indirect superclass Node
- Format:

```
rectangleObject.setRotate(doubleValue);
```

- Example:

```
rectangle.setArcWidth(45);
```

95 **The Ellipse Class (Page 1)**

- Draws an *ellipse* of specific x- and y-coordinates, and width and height
  – In addition to the more specific Circle class
- Direct subclass of abstract class Shape

- Located in the javafx.scene.shape package
  import javafx.scene.shape.Ellipse;

96 **The Ellipse Class (Page 2)**

- Format:
  Ellipse *ellipseObject* = new Ellipse([[*centerX*, *centerY*,] *radiusX*, *radiusY*]);
  – *centerX* and *centerY* are the x- and y-coordinates of the center of the circle
  – *radius* and *radiusY* are the horizontal and vertical measurements respectively from the center of the ellipse to its outer edge

97 **The Ellipse Class (Page 3)**

- Examples:
  Ellipse ellipse = new Ellipse();
  Ellipse ellipse = new Ellipse(75, 50);
  Ellipse ellipse = new Ellipse(400, 400, 75, 50);

98 **The Arc Class (Page 1)**

- Draws an *arc* (partial ellipse) of specific x- and y-coordinates, and width and height, and start location and length of arc
- Direct subclass of abstract class Shape
- Located in the javafx.scene.shape package
  import javafx.scene.shape.Arc;

99 **The Arc Class (Page 2)**

- Format:
  Arc *arcObject* = new Arc([[*centerX*, *centerY*, *radiusX*, *radiusY*, *startAngle*, *length*]);
  – *centerX* and *centerY* are the x- and y-coordinates of the center of the arc
  – *radius* and *radiusY* are the horizontal and vertical measurements respectively from the center of the arc to its outer edge

101 **The setType() Method (Page 1)**

- Sets the *closure type* for an arc using the constants from the class ArcType:
  – ArcType.OPEN—there are no closure lines (default)
  – ArcType.CHORD—draws a straight line from the start of the arc segment to the end of the arc segment
  – ArcType.ROUND—draws a straight line from start of the arc to the center of the full ellipse and from that point to the end of the arc
- The class with these constants are located in the javafx.scene.shape package
  import javafx.scene.shape.ArcType;

102 **The setType() Method (Page 2)**

- Format:
  arcObject.setType(ArcType.*CLOSURE_TYPE*);
- Examples:
  arc.setType(ArcType.ROUND);

103 **The Polygon Class (Page 1)**

- Draws a *polygon* (multisided shape) of several x- and y-coordinate points
- Direct subclass of abstract class Shape
- Located in the javafx.scene.shape package
  import javafx.scene.shape.Polygon;

104 **The Polygon Class (Page 2)**

- Format:
  Polygon *polygonObject* = new Polygon([*point1X*, *point1Y*, *point2X*, *point2Y*, ...]);
- Examples:
  Polygon polygon = new Polygon();
  Polygon polygon = new Polygon(80, 280, 120, 400, 20, 320, 140, 320, 40, 400);
-

105 **The ObservableList Interface**

- An ObservableList is a *generic* collection (list) with a *subtype* (similar to ArrayList)
- Format to declare:
  ObservableList<*Type*> *listObject*
  – *Type* is the subtype
- Example:
  ObservableList<Double> star = polygon.getPoints();

106 **The getPoints() Method (Page 1)**

- Method getPoints() returns the coordinates of a Polygon object as an ObservableList
- A "helper" method that can be used to instantiate an ObservableList
- Effectively *links* the two objects so that as items are inserted into the ObservableList, the points are added to the Polygon

107 **The getPoints() Method (Page 2)**

- Format:
  ObservableList<*Type*> *listObject* = *polygonObject*.getPoints();
- Example:
  ObservableList<Double> star = polygon.getPoints();

108 **The add() Method**

- The add() method for an ObservableList inserts one item (element) into the list
- Format:
  listObject.add(*value*);
- Examples:
  star.add(80);
  star.add(inReader.nextDouble())

109 **The Text Class (Page 1)**

- "Draws" *text* (actually a graphic) at specific upper left x- and y-coordinates

- Direct subclass of abstract class Shape
- Located in the javafx.scene.text package

```
import javafx.scene.text.Text;
```

110 **The Text Class (Page 2)**

- Format:

```
Text textObject = new Text( [x, y,] text);
```

  – *x* and *y* are the upper-left corner of the "box" in which the text draws
  – *text* is the displayed text String
- Examples:

```
Text text = new Text();
Text text = new Text("JavaFX");
Text text = new Text(20, 50, "JavaFX");
```

111 **The Font Class (Page 1)**

- Creates a Font object that sets properties of a JavaFX object:
  – Typeface (font name)
  – Bold property
  – Posture (italic) property
  – Font size measured in "points"
- Located in the javafx.scene.text package:

```
import javafx.scene.text.Font;
```

112 **The Font Class (Page 2)**

- Format to declare a Font object:

```
Font fontObject = new Font( [name,] size);
```

  – *name* is a String which is the "name" of a typeface (e.g. font) installed on the user's computer
  – *size* is a numeric value of type double which is a font size measured in "points"
- Examples:

```
Font font = new Font(24);
Font font = new Font("Comic Sans MS", 24);
```

113 **The Font.font() Method (Page 1)**

- The static "helper" method font() of class Font that returns a Font object with some or all of the font attributes:
  – Typeface (font name)
  – Bold property
  – Posture (italic) property
  – Font size measured in points

114 **The Font.font() Method (Page 2)**

- Format:

Font.font(*typeFace*, [[FontWeight.*WEIGHT_CONSTANT*,]
[FontPosture.*POSTURE_CONSTANT*,] *size*)

- Examples:

```
Font font = Font.font("Comic Sans MS", 24);
Font font = Font.font("Comic Sans MS", FontWeight.BOLD, FontPosture.ITALIC, 24);
```

115 **The setFont() Method**

- Sets the font for a Text object
- Format:

textObject.setFont(*fontObject*);

- Examples:

```
Font font = Font.font("Comic Sans MS", FontWeight.BOLD, 24);
text.setFont(font);
text.setFont( new Font("Comic Sans MS", 24) );
text.setFont( Font.font("Comic Sans MS", FontWeight.BOLD, 24) );
```

117 **The FlowPane Class (Page 1)**

- Arranges components left to right, top to bottom
  - Like lines of text in a paragraph
  - Especially noticeable when FlowPane object is *resized*
- Located in the javafx.scene.layout package:

```
import javafx.scene.layout.FlowPane;
```

118 **The FlowPane Class (Page 2)**

- Format to instantiate a FlowLayout object:

FlowPane *flowPaneObject* = new FlowPane([*hGap*, *vGap*,] [*children*]);

  - *hGap* is the horizontal gap between nodes
  - *vGap* is the vertical gap between nodes
  - *children* are one or more nodes that are initially added to the FlowPane

119 **The FlowPane Class (Page 3)**

- Examples:

```
FlowPane flow = new FlowPane();
FlowPane flow = new FlowPane( new Circle(), new ImageView(image) );
FlowPane flow = new FlowPane( 10, 12  new Circle(), new ImageView(image) );
```

120 **The FlowPane Class (Page 4)**

- Use methods getChildren().add() to add components to a FlowPane:

```
flow.getChildren().add( new Button("OK") );
```

- Or use getChildren().addAll()

121 **The Image Class (Page 1)**

- Creates an Image object that references a graphics file such as GIF or JPEG or PNG
  - Filename *extensions* are.gif or .jpg or .png
- Located in the javafx.scene.image package:

```
import javafx.scene.image.Image;
```

122 **The Image Class (Page 2)**

- Format to declare an Image object:
  Image *imageObject* = new Image("*path/filename*");
  – *path/location* is a String which is the Windows/DOS *filename* and *disk location* (URL) within the project's class folder
- Examples:
  Image image = new Image("javafx.png");
  Image image = new Image("file:images/javafx.png");

123 **The ImageView Class**

- An object for displaying an Image object
  – The "viewer" for placing it into a pane or scene
- Format:
  ImageView *imageViewObject* = new ImageView(*imageObject*);
- Examples:
  ImageView imageView = new ImageView(image);
  ImageView imageView = new ImageView(new Image("javafx.png"));