§16.1 Introduction

- So far, we have kept our UI simple, in order to demonstrate how to create a UI at all, and how to connect its elements to the code that will operate "behind" the UI's façade.
- We have seen `Text`, `TextField`, and `Button` nodes, as well as various shapes (`Arc`, `Circle`, `Line`, `Rectangle`, `Ellipse`, `Polygon` / `Polyline`)
- There are LOTS of other UI elements we can pull together to make a rich interface to make our software easier to use.
- This chapter explores several of them

§16.1 Introduction

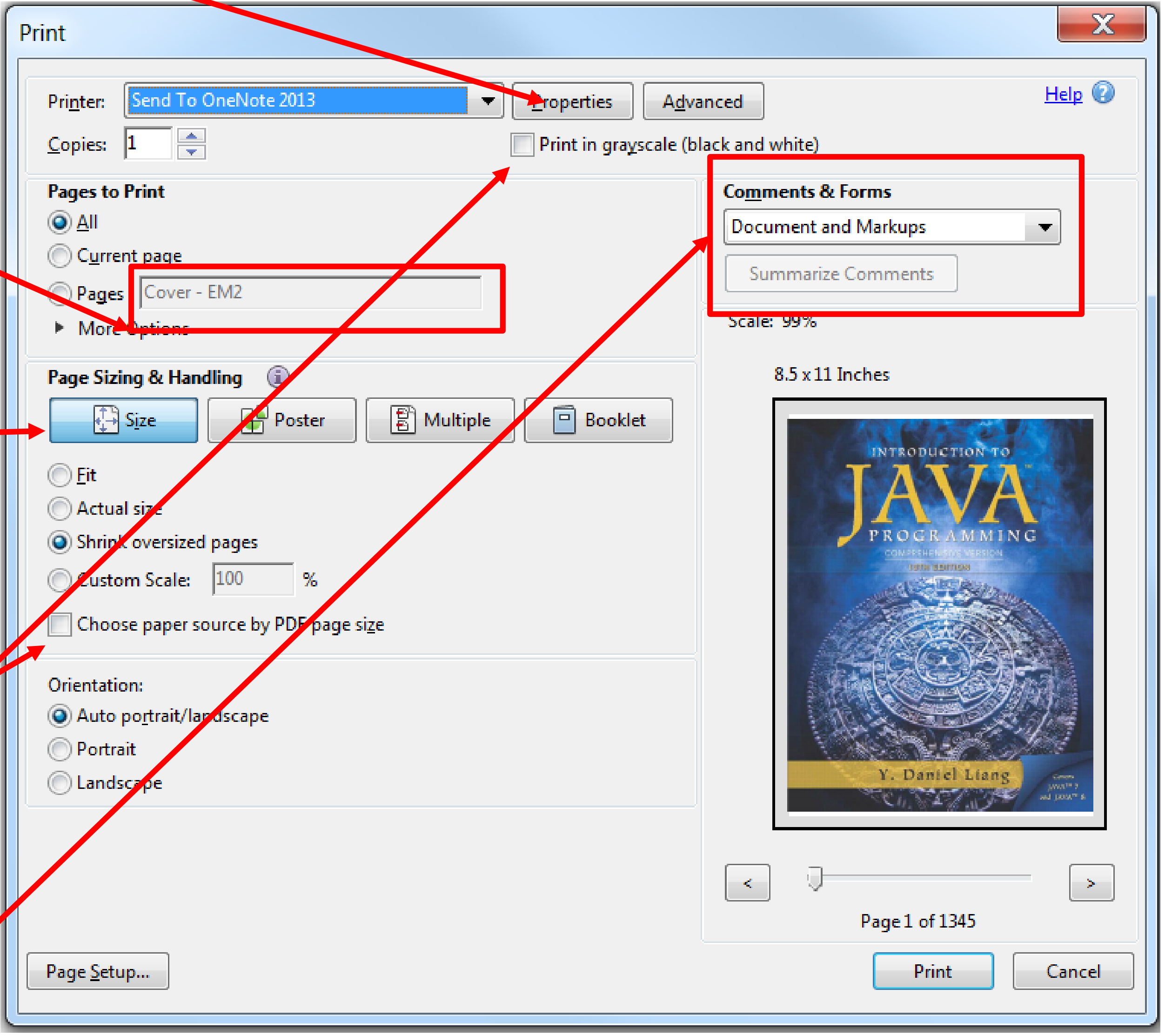
Buttons with a label to tell us what clicking on the button should do

`TextField`, into which the user may type text.

Buttons with both a label *and* an image

`CheckBox` with a *square* box to the left of its label that lets us turn on or off some Boolean value

`ComboBox`, which lets the user drop-down a list of options from which to select



Print

Printer: Send To OneNote 2013 | Properties | Advanced | Help

Copies: 1 | Print in grayscale (black and white)

Pages to Print: [● All] Current page, Pages: Cover - EM2, More Options

Comments & Forms: Document and Markups | Summarize Comments

Scale: 99%

8.5 x 11 Inches

Page Sizing & Handling: Size | Poster | Multiple | Booklet

Fit, Actual size, [● Shrink oversized pages], Custom Scale: 100 %

Choose paper source by PDF page size

Orientation: [● Auto portrait/landscape], Portrait, Landscape

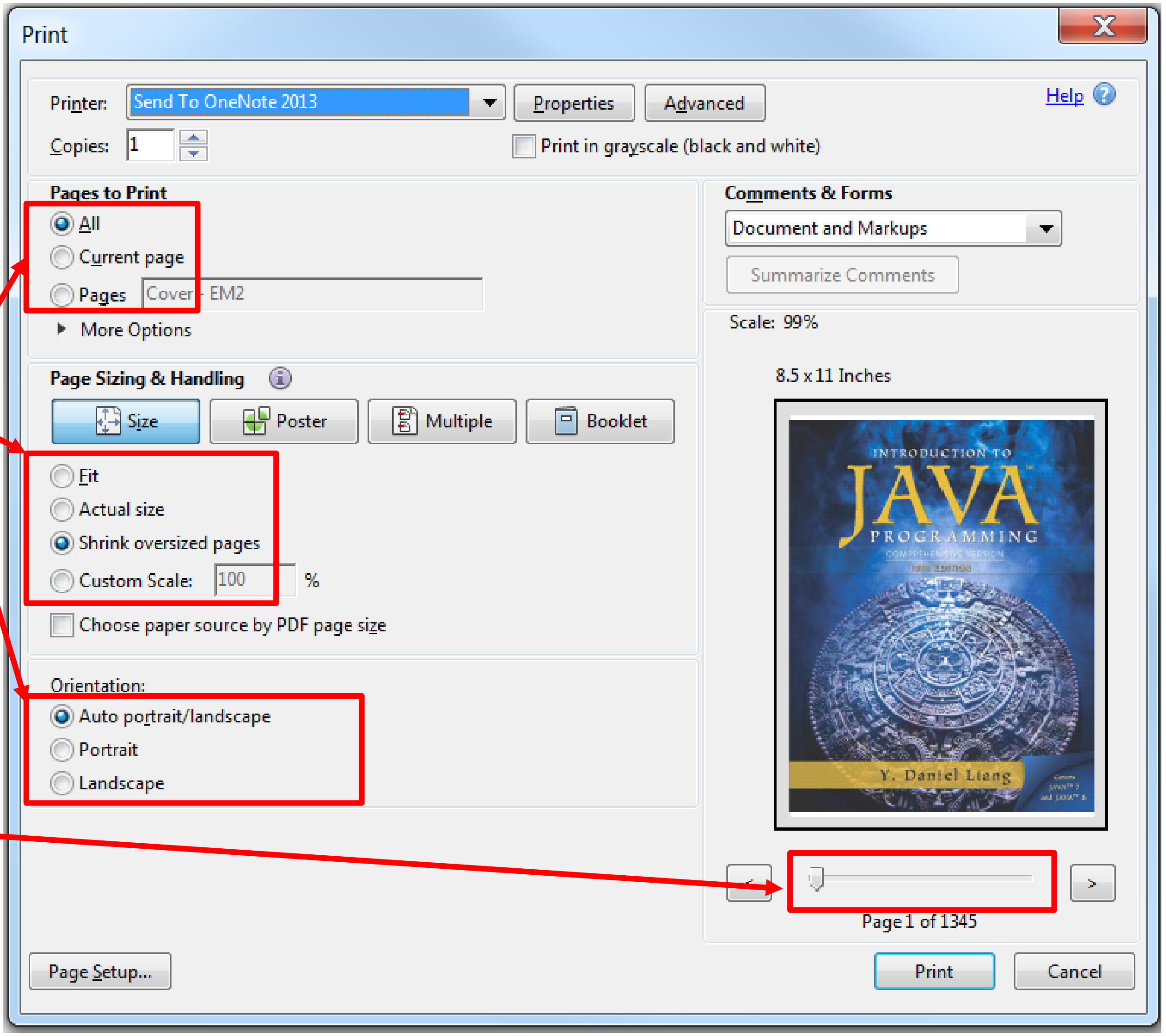
Page 1 of 1345

Page Setup... | Print | Cancel

§16.1 Introduction

- `RadioButtons`, arranged in groups.
- Only one button in a group can be selected (marked) at any one time – selecting one *deselects* the others in the group
- Each `RadioButton` has a *round* "selected" indicator and a label

- `Sliders` are similar to `ScrollBars`
- `Slider` lets us select a value from a range, whereas a `ScrollBar` is usually used to let us scroll content that doesn't fit into its container


Print

Printer: Send To OneNote 2013 | Properties | Advanced | Help

Copies: 1 | Print in grayscale (black and white)

Pages to Print
- (●) All
- ( ) Current page
- ( ) Pages: Cover - EM2
- More Options

Comments & Forms: Document and Markups | Summarize Comments

Scale: 99%

8.5 x 11 Inches

Page Sizing & Handling: Size | Poster | Multiple | Booklet
- ( ) Fit
- ( ) Actual size
- (●) Shrink oversized pages
- ( ) Custom Scale: 100 %
- Choose paper source by PDF page size

Orientation:
- (●) Auto portrait/landscape
- ( ) Portrait
- ( ) Landscape

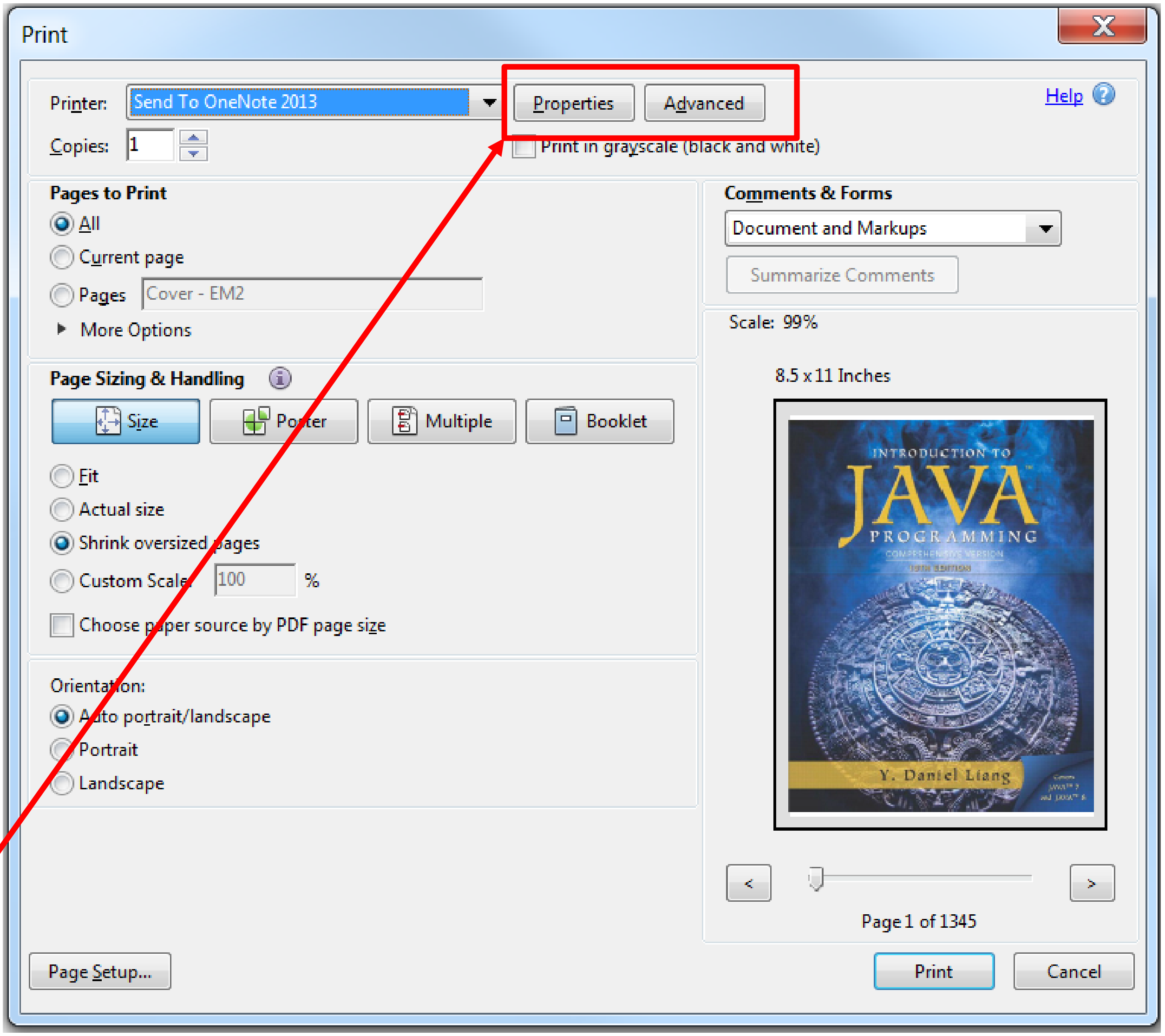
Page 1 of 1345

Page Setup... | Print | Cancel

§16.1 Introduction

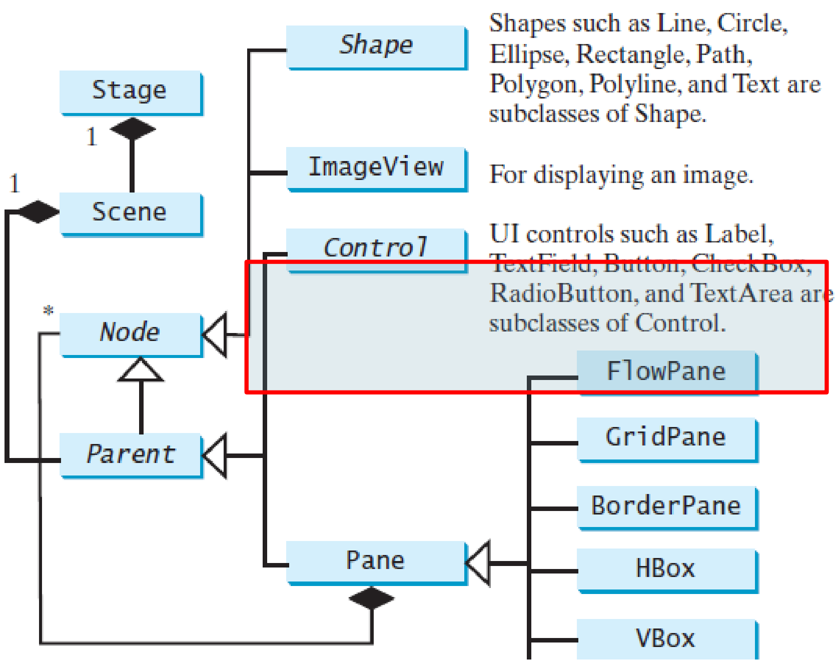
It's usually *easier* to navigate a GUI with the mouse, but it's typically much *faster* to do so it with the keyboard, and there are many keyboard shortcuts available.

First, controls with an underlined letter in their label can be selected by using ALT and the underlined letter (ALT+P from the keyboard is the same as clicking on the "Properties" `Button`)

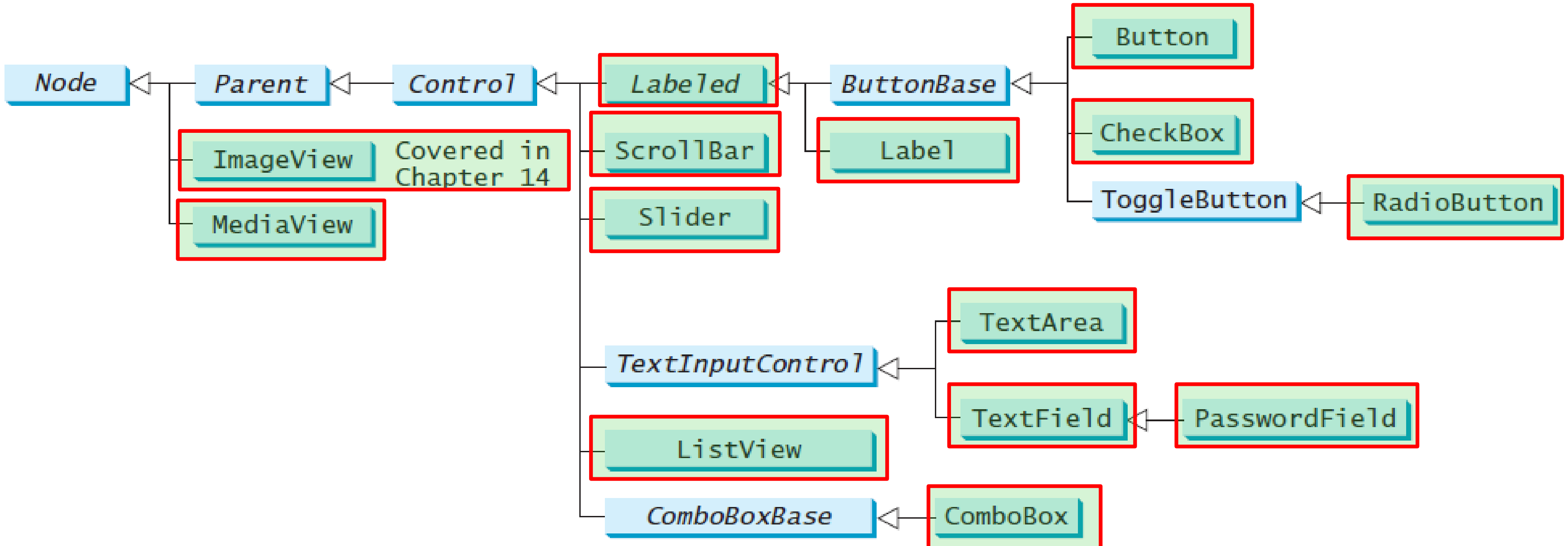


§16.1 Introduction

- We started Chapter 14 this diagram:
- This is great, but that whole branch just says *Control* has a lot of missing pieces
- This chapter goes into the details of what the primary controls *are*, and how we
- can use them to build a richer UI.



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



A Notational Note

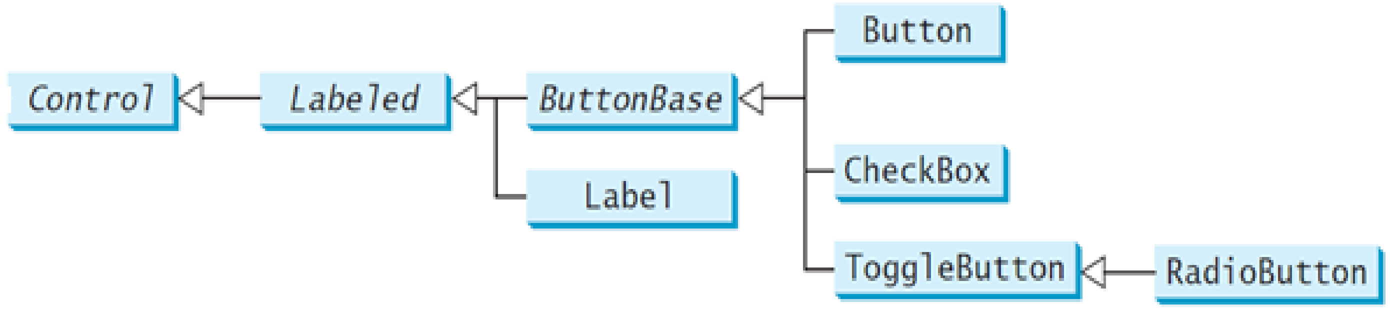
- The author notes that the book will consistently use the following prefixes for the various node types, making it easy to tell by looking at a node's variable name, what type of node it is:

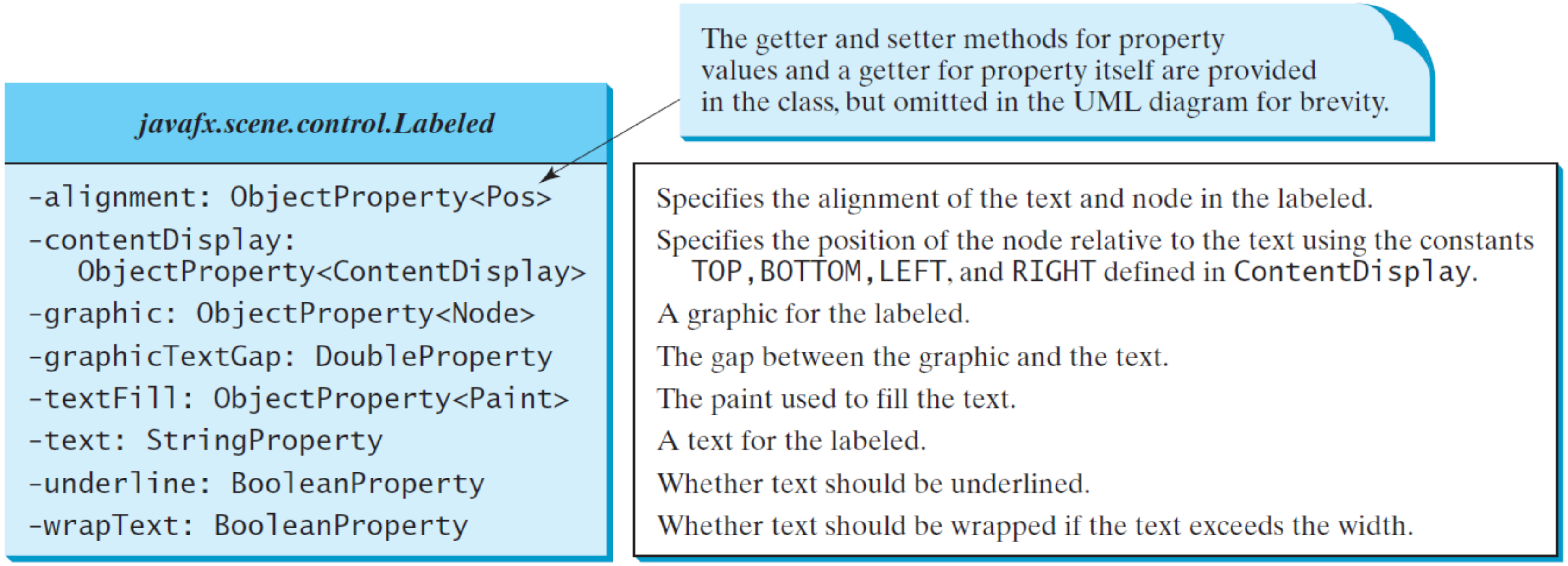
`lbl`	`Label`	`bt`	`Button`
`chk`	`CheckBox`	`rb`	`RadioButton`
`tf`	`TextField`	`pf`	`PasswordField`
`ta`	`TextArea`	`cbo`	`ComboBox`
`lv`	`ListView`	`scb`	`ScrollBar`
`sld`	`Slider`	`mp`	`MediaPlayer`

- This is a good idea. Some programmers even extend this to prefixing most non-obvious variable names with a reminder of their data type (`i`, `sgl`, `dbl`, `c` or `ch`, `bol`, `lng`, `sh`, `byt`) – see here

§16.2 Labeled and Label

- Because a `Button` has a `Label` on it, it's actually a sub-class of the `Labeled` class, which is the parent class of `Label`, `Button`, `CheckBox`, and `RadioButton`.

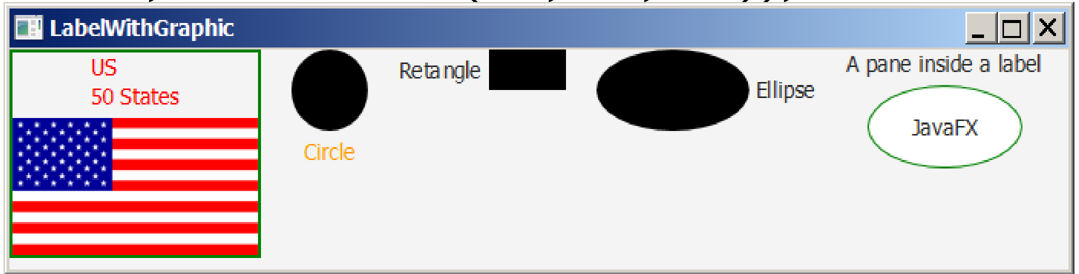



The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

javafx.scene.control.Labeled	
`-alignment: ObjectProperty<Pos>`	Specifies the alignment of the text and node in the labeled.
`-contentDisplay: ObjectProperty<ContentDisplay>`	Specifies the position of the node relative to the text using the constants `TOP`, `BOTTOM`, `LEFT`, and `RIGHT` defined in `ContentDisplay`.
`-graphic: ObjectProperty<Node>`	A graphic for the labeled.
`-graphicTextGap: DoubleProperty`	The gap between the graphic and the text.
`-textFill: ObjectProperty<Paint>`	The paint used to fill the text.
`-text: StringProperty`	A text for the labeled.
`-underline: BooleanProperty`	Whether text should be underlined.
`-wrapText: BooleanProperty`	Whether text should be wrapped if the text exceeds the width.

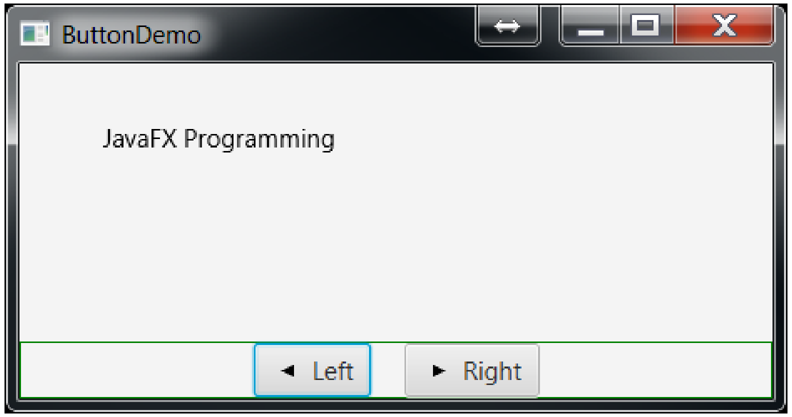
§16.2 Labeled and Label

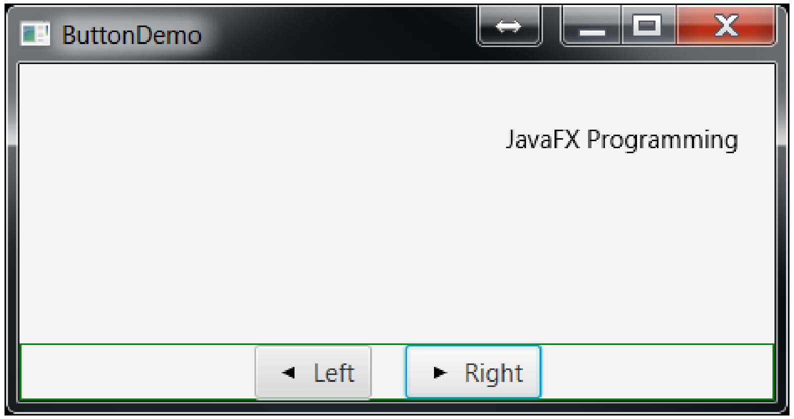
- Label – just a piece of text on the UI to show the user what the control NEXT to it is for
- A Label is the simplest Labeled sub-class
- It typically contains a String, but can contain a graphic instead, or it can contain both.
  - The "graphic" can be an ImageView, or something more complicated, like a whole pane.
- Listing 16.1, pp. 630-631, shows us how to do these "out of the ordinary" labels:
- ImageView us = **new** ImageView(**new** Image(**"image/us.gif"**));
- Label lb1 = **new** Label(**"US\n50 States"**, us);
- Label lb2 = **new** Label(**"Circle"**, **new** Circle(**50**, **50**, **25**));



§16.3 Button

- A Button is a control that fires an ActionEvent when clicked.
- There are several flavors of Button, each of which extends ButtonBase.
- The only thing ButtonBase adds to Labeled is the onAction event handler (which we override!)
- Listing 16.2, pp. 633 – 634 creates a Text element and two Buttons
- Clicking the two Buttons moves the Text to the left and right.



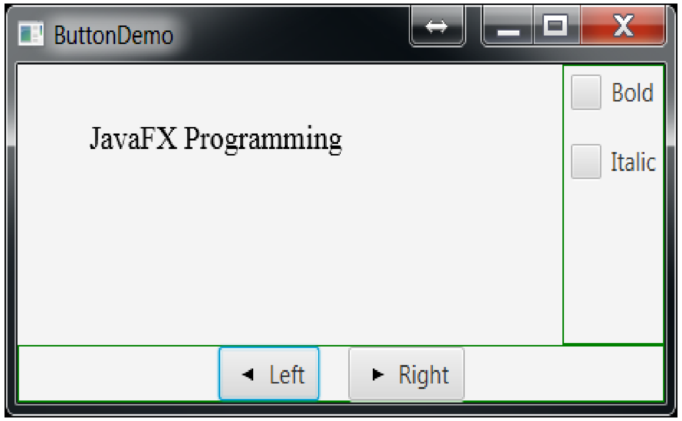


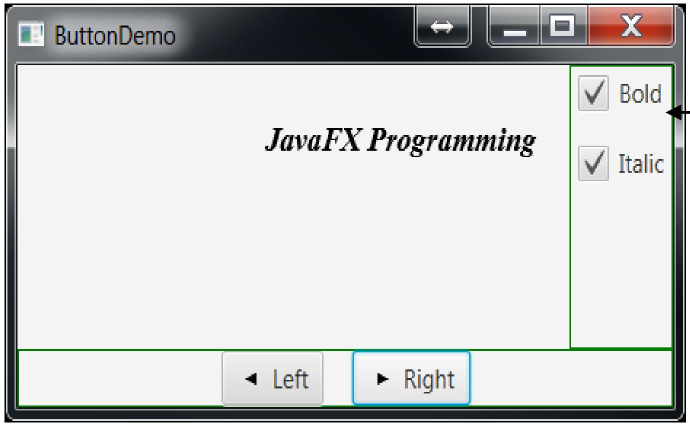
§ Listing 16.2

```
public class ButtonDemo extends Application {
  protected Text text = new Text(50, 50, "JavaFX Programming");

  protected BorderPane getPane() {
    HBox paneForButtons = new HBox(20);
    Button btLeft = new Button("Left",                              create a button
      new ImageView("image/left.gif"));
    Button btRight = new Button("Right",
      new ImageView("image/right.gif"));
    paneForButtons.getChildren().addAll(btLeft, btRight);           add buttons to pane
    paneForButtons.setAlignment(Pos.CENTER);
    paneForButtons.setStyle("-fx-border-color: green");

    BorderPane pane = new BorderPane();                             create a border pane
    pane.setBottom(paneForButtons);                                 add buttons to the bottom

    Pane paneForText = new Pane();
    paneForText.getChildren().add(text);
    pane.setCenter(paneForText);

    btLeft.setOnAction(e -> text.setX(text.getX() - 10));           add an action handler
    btRight.setOnAction(e -> text.setX(text.getX() + 10));

    return pane;                                                    return a pane
  }

  @Override // Override the start method in the Application class
  public void start(Stage primaryStage) {
    // Create a scene and place it in the stage
    Scene scene = new Scene(getPane(), 450, 200);
    primaryStage.setTitle("ButtonDemo"); // Set the stage title
    primaryStage.setScene(scene); // Place the scene in the stage
    primaryStage.show(); // Display the stage
  }
}
```

§16.4 CheckBox

- The CheckBox inherits all of the Label- and Image-displaying abilities from Labeled, and the onAction method from ButtonBase, and adds a Boolean property to tell us whether it is selected (checked) or not, which we can examine via isSelected()
- Listing 16.3 (pp. 635 – 636) extends the ButtonDemo to add CheckBoxes for changing the text to bold and/or italic
- Rather than creating a new pane from scratch, this program extends the ButtonDemo and overrides its getPane() method to produce the new one





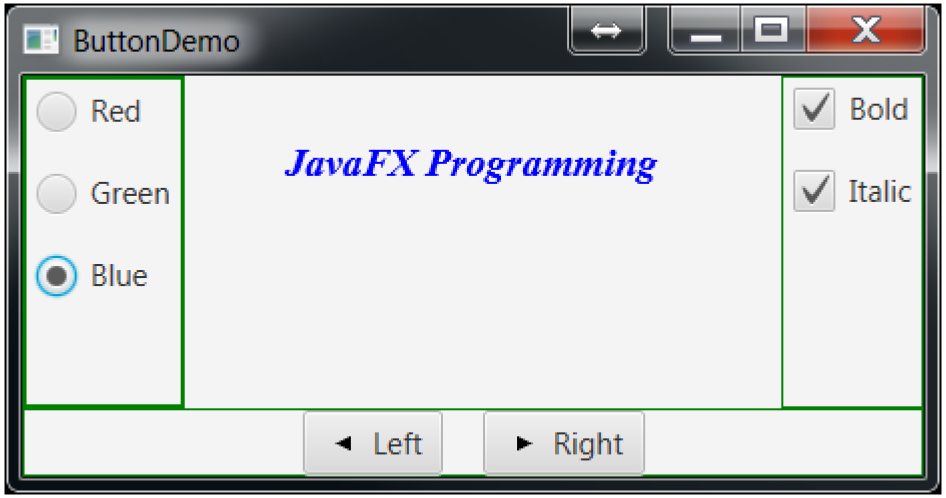
VBox containing two check boxes

§Listing 16.3

```java
text.setFont(fontNormal);

VBox paneForCheckBoxes = new VBox(20);
paneForCheckBoxes.setPadding(new Insets(5, 5, 5, 5));
paneForCheckBoxes.setStyle("-fx-border-color: green");
CheckBox chkBold = new CheckBox("Bold");
CheckBox chkItalic = new CheckBox("Italic");
paneForCheckBoxes.getChildren().addAll(chkBold, chkItalic);
pane.setRight(paneForCheckBoxes);

EventHandler<ActionEvent> handler = e -> {
  if (chkBold.isSelected() && chkItalic.isSelected()) {
    text.setFont(fontBoldItalic); // Both check boxes checked
  }
  else if (chkBold.isSelected()) {
    text.setFont(fontBold); // The Bold check box checked
  }
  else if (chkItalic.isSelected()) {
    text.setFont(fontItalic); // The Italic check box checked
  }
  else {
    text.setFont(fontNormal); // Both check boxes unchecked
  }
};

chkBold.setOnAction(handler);
chkItalic.setOnAction(handler);
```

§16.5 RadioButton

- RadioButtons are sometimes called option buttons
- They let us select (only) one of a group of options, unlike CheckBoxes, in which any (or all) of the boxes can be checked.
- If RadioButton A is selected and we click on RadioButton B, RadioButton A will become *deselected*, and RadioButton B will become *selected*.
- Listing 16.4 (pp. 638 – 639) is an extension of CheckBoxDemo, which was an extension of ButtonDemo. Which adds three RadioButtons, one each for Red, Green, and Blue.
- When the user clicks a RadioButton, the text changes color to match the new selection.



§ Listing 16.4

- In this program, we have three RadioButtons. What if we added three more for something else? How would the UI know that these three are mutually-exclusive, and that the other three are a different set of mutually-exclusive options?
- We create a ToggleGroup for each set, and then add the RadioButtons to the ToggleGroup.

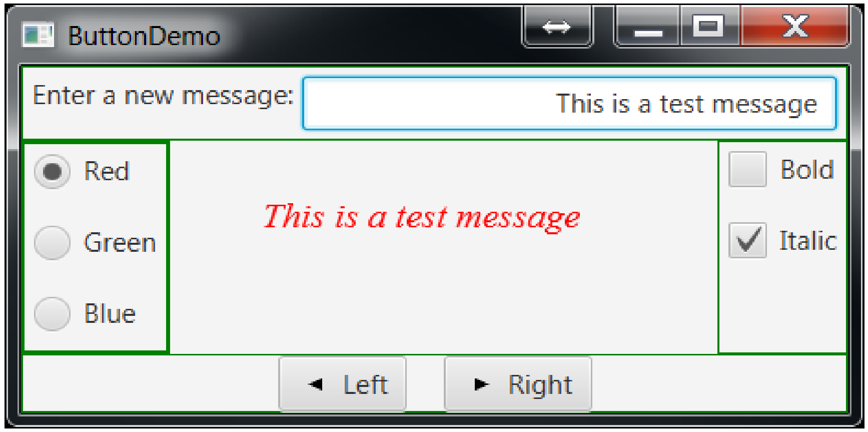
```
    RadioButton rbRed = new RadioButton("Red");
    RadioButton rbGreen = new RadioButton("Green");
    RadioButton rbBlue = new RadioButton("Blue");
    paneForRadioButtons.getChildren().addAll(rbRed, rbGreen, rbBlue);
    pane.setLeft(paneForRadioButtons);

    ToggleGroup group = new ToggleGroup();
    rbRed.setToggleGroup(group);
    rbGreen.setToggleGroup(group);
    rbBlue.setToggleGroup(group);

    rbRed.setOnAction(e -> {
      if (rbRed.isSelected()) {
        text.setFill(Color.RED);
      }
    });

    rbGreen.setOnAction(e -> {
      if (rbGreen.isSelected()) {
        text.setFill(Color.GREEN);
```

§16.6 TextField

- A TextField can be used to either display (uneditable) text, or to create a place the user can type textual information
- Pressing the Enter key inside a TextField fires an ActionEvent
- The only difference between a TextField and a PasswordField is that, as the user is typing text into a PasswordField, rather than showing the characters the user typed, the system displays an asterisk for each character, effectively hiding the text (but not its length)
- Listing 16.5 (pp. 640 – 641) shows yet another extension to the program we've been adding features to all chapter long
- This one adds a TextField to the top section of the BorderPane, so the user can change the contents of the displayed Text. When the user presses Enter, the Text changes

§ Listing 16.5

```
public class TextFieldDemo extends RadioButtonDemo {
  @Override // Override the getPane() method in the super class
  protected BorderPane getPane() {
    BorderPane pane = super.getPane();

    BorderPane paneForTextField = new BorderPane();
    paneForTextField.setPadding(new Insets(5, 5, 5, 5));
    paneForTextField.setStyle("-fx-border-color: green");
    paneForTextField.setLeft(new Label("Enter a new message: "));

    TextField tf = new TextField();
    tf.setAlignment(Pos.BOTTOM_RIGHT);
    paneForTextField.setCenter(tf);
    pane.setTop(paneForTextField);
```



Enter a new message: This is a test message

[● Red] Green Blue

This is a test message

Bold [✓ Italic]

```
    tf.setOnAction(e -> text.setText(tf.getText()));

    return pane;
  }
}
```

§16.7: TextArea

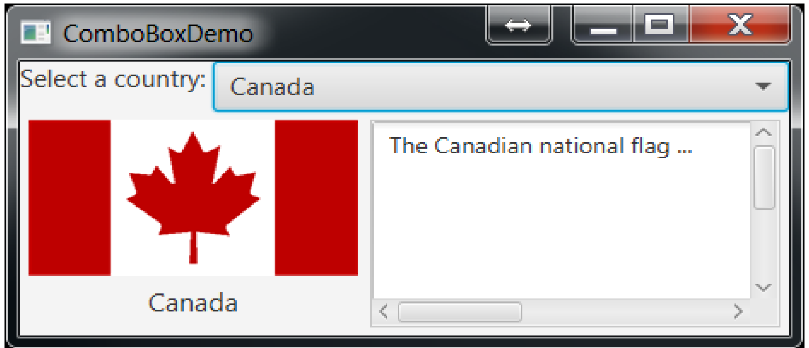
- The TextArea is essentially a TextBox that can contain multiple lines of text, rather than just one.
- It, too, is an extension of TextInputControl
- The code Listing 16.6 (pp. 642 – 643) uses a TextArea inside a ScrollPane to allow us an easy way of scrolling the text for a large TextArea.
- You can place any node in a **ScrollPane**. **ScrollPane** provides vertical and horizontal scrolling automatically if the control is too large to fit in the viewing area.
  - TextArea taDescription = **new** TextArea();
  - taDescription.setWrapText(**true**);   // wrap text
  - taDescription.setEditable(**false**);   // read only
  - // Create a scroll pane to hold the text area
  - ScrollPane scrollPane = **new** ScrollPane(taDescription);

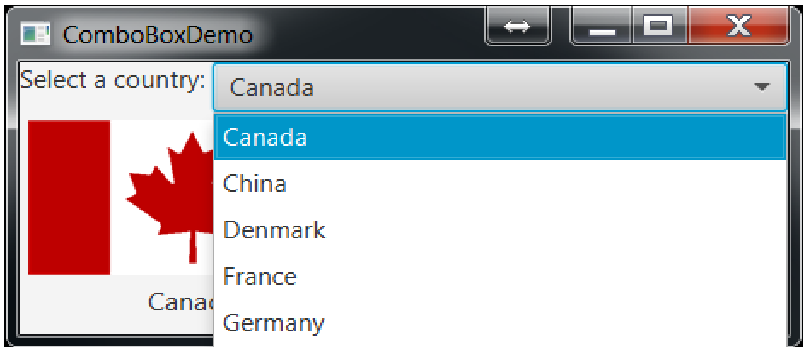
§16.8 : ComboBox

- ComboBox lets the user select one of a pre-defined set of options.
- By only allowing items on the list, we avoid many data validation issues we might have with a TextField.
- A ComboBox fires an ActionEvent when an item on the list is selected
- Listing 16.8 (pp. 646 – 647) show the code for an application that lets the user select a country via a ComboBox, and then displays the country's flag and a descriptive bit of text.
- ComboBox<String> cbo = new ComboBox<>();
- cbo.getItems().addAll("Item 1", "Item 2","Item 3", "Item 4");
- String[] flagTitles = {"Canada", "China", "Denmark", "France", "Germany", "India", "Norway", "United Kingdom", "United States of America"};

ObservableList<String> items = FXCollections.observableArrayList(flagTitles);

- cbo.getItems().addAll(items);


ComboBoxDemo
Select a country:
Canada
The Canadian national flag ...
Canada


ComboBoxDemo
Select a country:
Canada
Canada
China
Denmark
France
Germany
Canad