

Chapter-2 (JavaFX)

Part-1

By: Joseph W.

Outline

- Introduction
- JavaFX App Window Structure
- Layout Containers
- BorderPane
- HBox
- VBox
- GridPane
- FlowPane
- TilePane
- AnchorPane

Introduction

- A graphical user interface (GUI) presents a user-friendly mechanism for interacting with an app.
- A GUI (pronounced “GOO-ee”) gives an app a distinctive “look-and-feel.”
- GUIs are built from GUI components—also called controls or widgets (short for window gadgets).
- Java’s original GUI library was the Abstract Window Toolkit (**AWT**). **Swing** was added to the platform in Java SE 1.2.
- JavaFX 1.0 was released in 2008.
- Prior to version 2.0, developers wrote JavaFX apps in JavaFX Script, which compiled to Java bytecode, allowing JavaFX apps to run on the Java Virtual Machine.
- Starting with version 2.0 in 2011, JavaFX was re-implemented as Java libraries that could be used directly in Java apps.

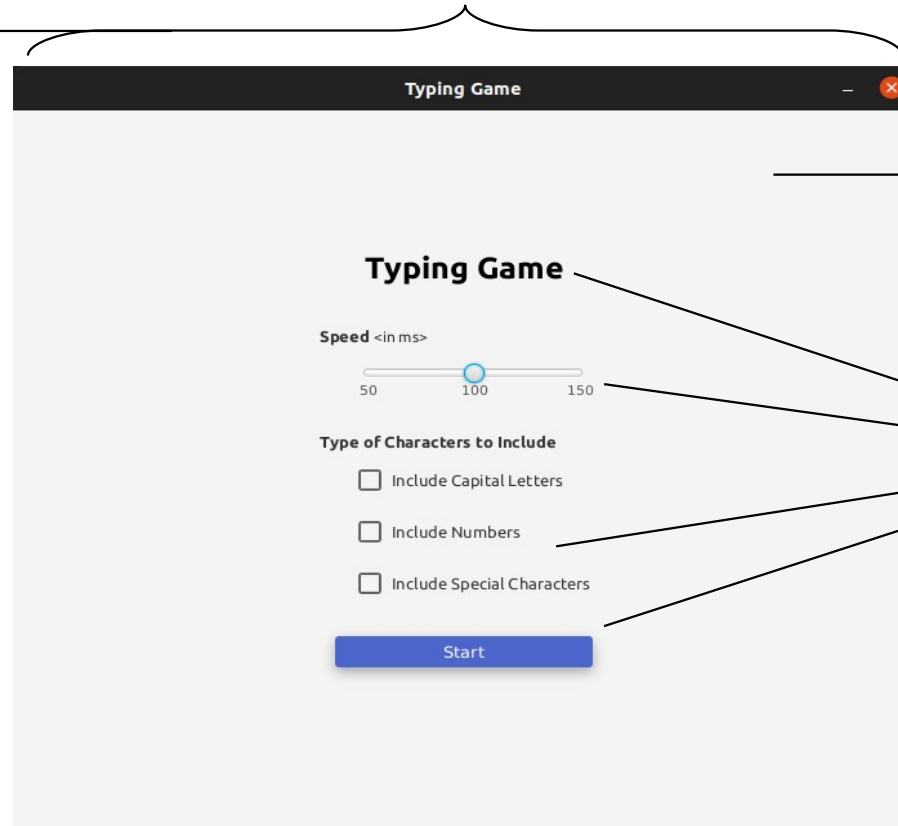
Cont.

- Some of the benefits of JavaFX over Swing include:
 - ✓ JavaFX is easier to use—it provides one API for client functionality, including GUI, graphics and multimedia (images, animation, audio and video). Swing is only for GUIs, so you need to use other APIs for graphics and multimedia apps.
 - ✓ Though Swing components could be customized, JavaFX gives you complete control over a JavaFX GUI's look-and-feel via Cascading Style Sheets (CSS)—the same technology used to style web pages.
 - ✓ JavaFX has better threading support, which is important for getting the best application performance on today's multi-core systems.
 - ✓ JavaFX uses the GPU (graphics processing unit) for hardware-accelerated rendering.
 - ✓ JavaFX supports transformations for repositioning and reorienting JavaFX components, and animations for changing the properties of JavaFX components over time.
 - ✓ JavaFX provides multiple upgrade paths for enhancing existing GUIs

JavaFX App Window Structure

The window is known as stage

The stage contains a scene graph of nodes



The root node of this scene graph is a layout container that arranges the other nodes

Each of the JavaFX components in this GUI is a node in the scene graph

Cont.

→ Controls

- ✓ Controls are GUI components, such as Labels that display text, TextFields that enable a program to receive user input, Buttons that users click to initiate actions, and more.

→ Stage

- ✓ The window in which a JavaFX app's GUI is displayed is known as the stage and is an instance of class Stage (package javafx.stage).

→ Scene

- ✓ The stage contains one active scene that defines the GUI as a scene graph—a tree data structure of an app's visual elements, such as GUI controls, shapes, images, video, text and more. The scene is an instance of class Scene (package javafx.scene).

→ Nodes

- ✓ Each visual element in the scene graph is a node

Cont.

- ✓ A node is an instance of a subclass of Node (package javafx.scene), which defines common attributes and behaviors for all nodes.
 - ✓ With the exception of the first node in the scene graph—the root node—each node in the scene graph has one parent.
 - ✓ Nodes can have transforms (e.g., moving, rotating and scaling), opacity (whether a node is transparent, partially transparent or opaque), effects (e.g., drop shadows, blurs, reflection and lighting).
- **Layout Containers**
- ✓ Nodes that have children are typically layout containers that arrange their child nodes in the scene.
- **Event Handlers**
- ✓ When the user interacts with a control, such as clicking a Button or typing text into a TextField, the control generates an event. Programs can respond to these events—known as event handling—to specify what should happen when each user interaction occurs.

Layout Containers

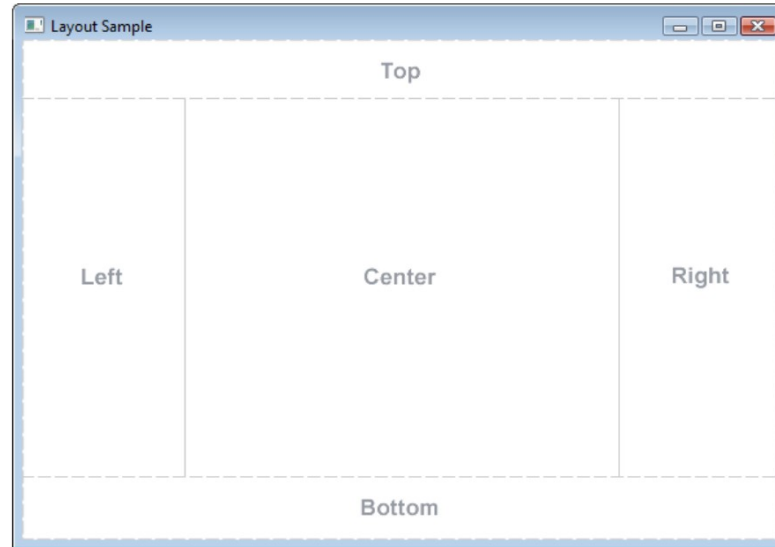
- JavaFX contains layout container classes, called panes, that are available with the JavaFX SDK.
- A JavaFX application can manually lay out the UI by setting the position and size properties for each UI element. However, an easier option is to make use of layout panes.
- The JavaFX SDK provides several layout panes for the easy setup and management of classic layouts such as rows, columns, stacks, tiles, and others.
- As a window is resized, the layout pane automatically repositions and resizes the nodes that it contains according to the properties for the nodes.
- Most JavaFX layout panes use relative positioning—if a layout-pane node is resized, it adjusts its children's sizes and positions accordingly, based on their preferred, minimum and maximum sizes.
- The built in panes include: **BorderPane**, **HBox**, **VBox**, **StackPane**, **GridPane**, **FlowPane**, **TilePane** and **AnchorPane**.

BorderPane

- The BorderPane layout pane provides five regions in which to place nodes: top, bottom, left, right, and center.
- A border pane is useful for the classic look of a **tool bar at the top**, a **status bar at the bottom**, a **navigation panel on the left**, **additional information on the right**, and a **working area in the center**.
- If the window is larger than the space needed for the contents of each region, the **extra space is given to the center** region by default.
- If the window is smaller than the space needed for the contents of each region, the regions might **overlap**.
- The overlap is determined by the **order in which the regions are set**. For example, if the regions are set in the order of left, bottom, and right, when the window is made smaller, the bottom region overlaps the left region and the right region overlaps the bottom region.
- Creating an object of BorderPane class: **BorderPane** border = new **BorderPane**();

Cont.

- Methods **setTop**(node), **setLeft**(node), **setCenter**(node), **setRight**(node) and **setBottom**(node) are used to add or place nodes to the top, left, center, right and bottom regions of the **BorderPane**.
(Note: “node”, will be replaced by child nodes which are controls, other layout panes, shapes and more.)
- If your application does not need one of the regions, you do not need to define it and no space is allocated for it. **Example:** sample BorderPane looks like:



Cont.

→ **Example:** (Create BorderPane)

```
BorderPane border = new BorderPane();  
HBox hbox = addHBox()  
border.setTop(hbox);  
border.setLeft(addVBox());  
addStackPane(hbox);  
border.setCenter(addGridPane());  
border.setRight(addFlowPane());
```

- Some lines in the code fragment above will make more sense as we progress through the rest of the slides.

HBox

- The **HBox** layout pane provides an easy way for arranging a series of nodes in a single row.
- The **padding** property can be set to manage the **distance between the nodes and the edges of the HBox pane**.
- **Spacing** can be set to manage the **distance between the nodes**.
- The style can be set to change the background color.
- **Example:** sample HBox pane looks like:



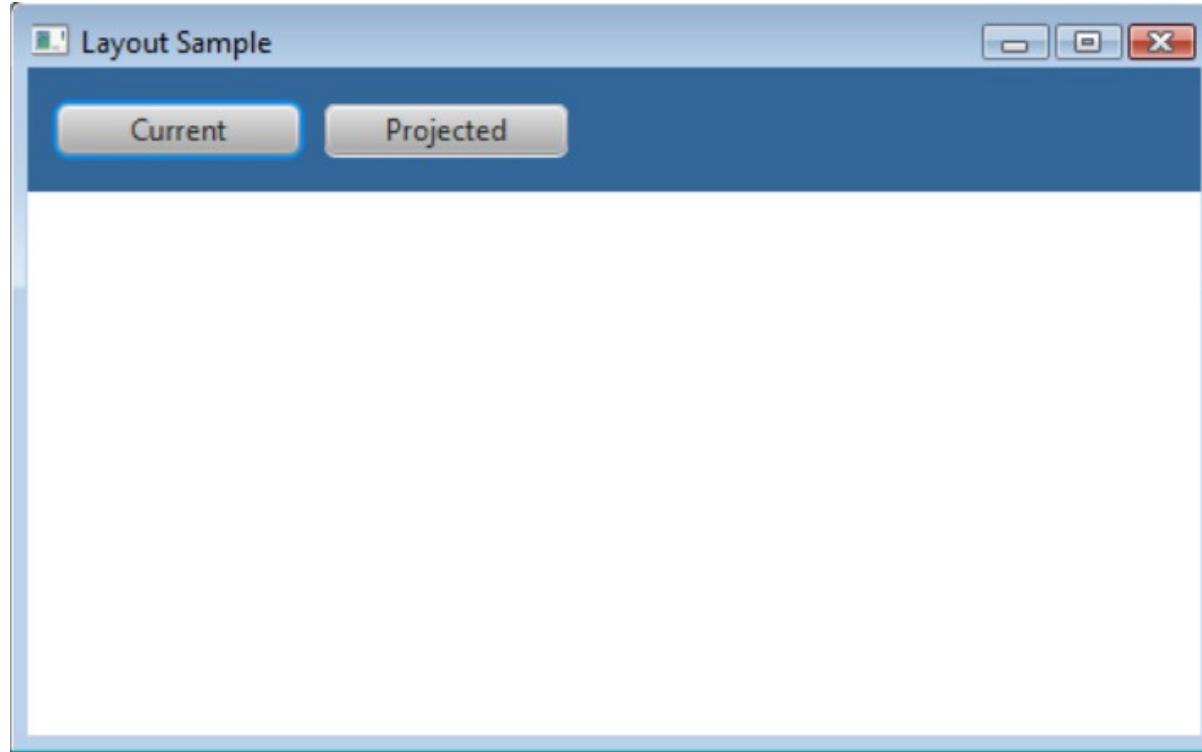
Cont.

→ **Example:** (Create Hbox)

```
public HBox addHBox() {  
    HBox hbox = new HBox();  
    hbox.setPadding(new Insets(15, 12, 15, 12));  
    hbox.setSpacing(10);  
    hbox.setStyle("-fx-background-color: #336699;");  
    Button buttonCurrent = new Button("Current");  
    buttonCurrent.setPrefSize(100, 20);  
    Button buttonProjected = new Button("Projected");  
    buttonProjected.setPrefSize(100, 20);  
    hbox.getChildren().addAll(buttonCurrent, buttonProjected);  
    return hbox;  
}
```

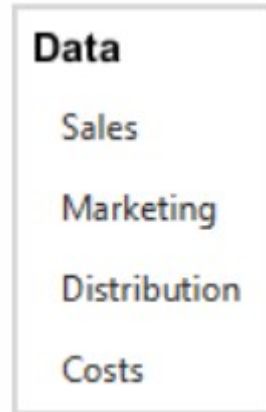
Cont.

- The **HBox** pane in **BorderPane** looks like:



VBox

- The **VBox** layout pane is similar to the HBox layout pane, except that the nodes are **arranged in a single column**.
- The padding property can be set to manage the distance between the nodes and the edges of the VBox pane.
- Spacing can be set to manage the distance between the nodes.
- Margins can be set to add additional space around individual controls.
- Example: sample VBox pane looks like:



Cont.

→ **Example:** (Create VBox)

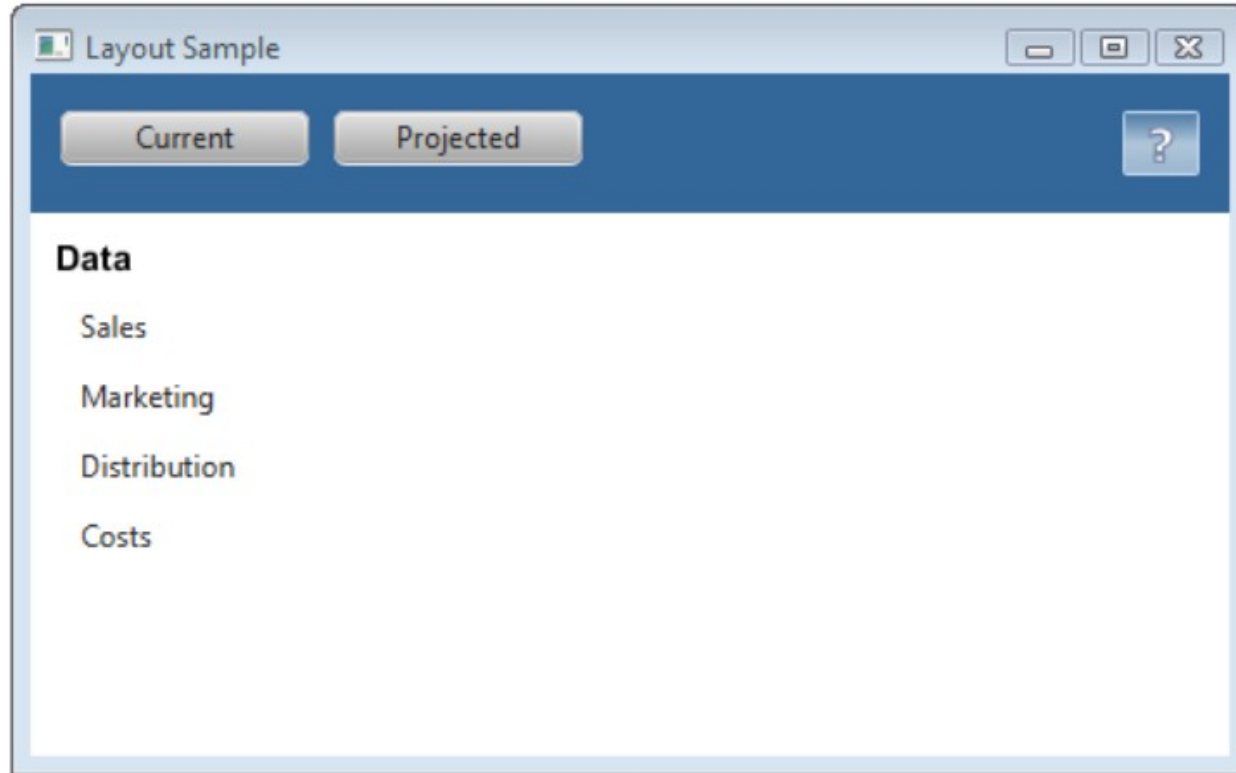
```
public VBox addVBox(); {  
    VBox vbox = new VBox();  
    vbox.setPadding(new Insets(10));  
    vbox.setSpacing(8);  
    Text title = new Text("Data");  
    title.setFont(Font.font("Arial", FontWeight.BOLD, 14));  
    vbox.getChildren().add(title);  
    Hyperlink options[] = new Hyperlink[] {
```

Cont.

```
new Hyperlink("Sales"),  
new Hyperlink("Marketing"),  
new Hyperlink("Distribution"),  
new Hyperlink("Costs"));  
for (int i=0; i<4; i++) {  
    VBox.setMargin(options[i], new Insets(0, 0, 0, 8));  
    vbox.getChildren().add(options[i]);  
}  
return vbox;  
}
```

Cont.

→ The **VBox** pane in **BorderPane** looks like:



StackPane

- The **StackPane** layout pane places all of the nodes within a single stack with each new node added on top of the previous node.
- This layout model provides an easy way to overlay text on a shape or image or to overlap common shapes to create a complex shape.
- Example: below is a help icon that is created by stacking a question mark on top of a rectangle with a gradient background.



- The alignment property can be set to manage how children are positioned in the stack pane.
- This property affects all children, so margins can be set to adjust the position of individual children in the stack.

Cont.

→ **Example:** (Create StackPane)

```
public void addStackPane(HBox hb) {  
    StackPane stack = new StackPane();  
    Rectangle helpIcon = new Rectangle(30.0, 25.0);  
    helpIcon.setFill(new LinearGradient(0,0,0,1, true,  
    CycleMethod.NO_CYCLE,  
        new Stop[]{  
            new Stop(0,Color.web("#4977A3")),  
            new Stop(0.5, Color.web("#B0C6DA")),  
            new Stop(1,Color.web("#9CB6CF")),}));  
    helpIcon.setStroke(Color.web("#D0E6FA"));  
    helpIcon.setArcHeight(3.5);  
    helpIcon.setArcWidth(3.5);  
}
```

Cont.

```
Text helpText = new Text("?");
helpText.setFont(Font.font("Verdana", FontWeight.BOLD, 18));
helpText.setFill(Color.WHITE);
helpText.setStroke(Color.web("#7080A0"));
stack.getChildren().addAll(helpIcon, helpText);
stack.setAlignment(Pos.CENTER_RIGHT);
StackPane.setMargin(helpText, new Insets(0, 10, 0, 0));
hb.getChildren().add(stack);
HBox.setHgrow(stack, Priority.ALWAYS);
}
```

Cont.

→ The **StackPane** in **BorderPane** looks like:

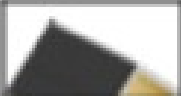
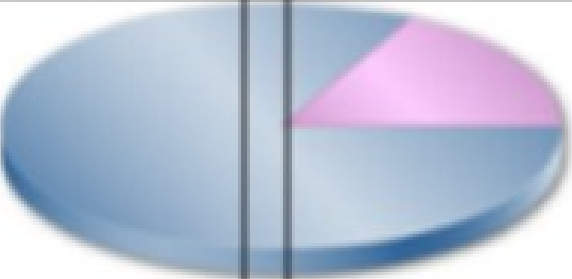


GridPane

- The **GridPane** layout pane enables you to create a flexible grid of **rows** and **columns** in which to lay out nodes.
- Nodes can be placed in any cell in the grid and can span cells as needed.
- A grid pane is useful for creating forms or any layout that is organized in rows and columns.
- Figure in the next slide shows a grid pane that contains an icon, title, subtitle, text and a pie chart. In this figure, the **gridLinesVisible** property is set to display grid lines, which show the rows and columns and the gaps between the rows and columns.
- “**gridLinesVisible**” property is useful for visually **debugging** your GridPane layouts.
- **Gap** properties can be set to manage the spacing between the rows and columns.
- The **padding** property can be set to manage the distance between the nodes and the edges of the grid pane.

Cont.

- The vertical and horizontal alignment properties can be set to manage the alignment of individual controls in a cell.

	Sales:	Current Year	
	Goods and	Services	
Goods 80%			Services 20%

Cont.

→ **Example:** (Create GridPane)

```
public GridPane addGridPane() {  
    GridPane grid = new GridPane();  
    grid.setHgap(10);  
    grid.setVgap(10);  
    grid.setPadding(new Insets(0, 10, 0, 10));  
    Text category = new Text("Sales:");  
    category.setFont(Font.font("Arial", FontWeight.BOLD, 20));  
    grid.add(category, 1, 0);  
    Text chartTitle = new Text("Current Year");  
    chartTitle.setFont(Font.font("Arial", FontWeight.BOLD, 20));  
    grid.add(chartTitle, 2, 0);  
}
```

Cont.

```
Text chartSubtitle = new Text("Goods and Services");
grid.add(chartSubtitle, 1, 1, 2, 1);
ImageView imageHouse = new ImageView(
    new Image(LayoutSample.class.getResourceAsStream("graphics/house.png")));
grid.add(imageHouse, 0, 0, 1, 2);
Text goodsPercent = new Text("Goods\n80%");
GridPane.setValignment(goodsPercent, VPos.BOTTOM);
grid.add(goodsPercent, 0, 2);
ImageView imageChart = new ImageView(
    new Image(LayoutSample.class.getResourceAsStream("graphics/piechart.png")));
grid.add(imageChart, 1, 2, 2, 1);
```

Cont.

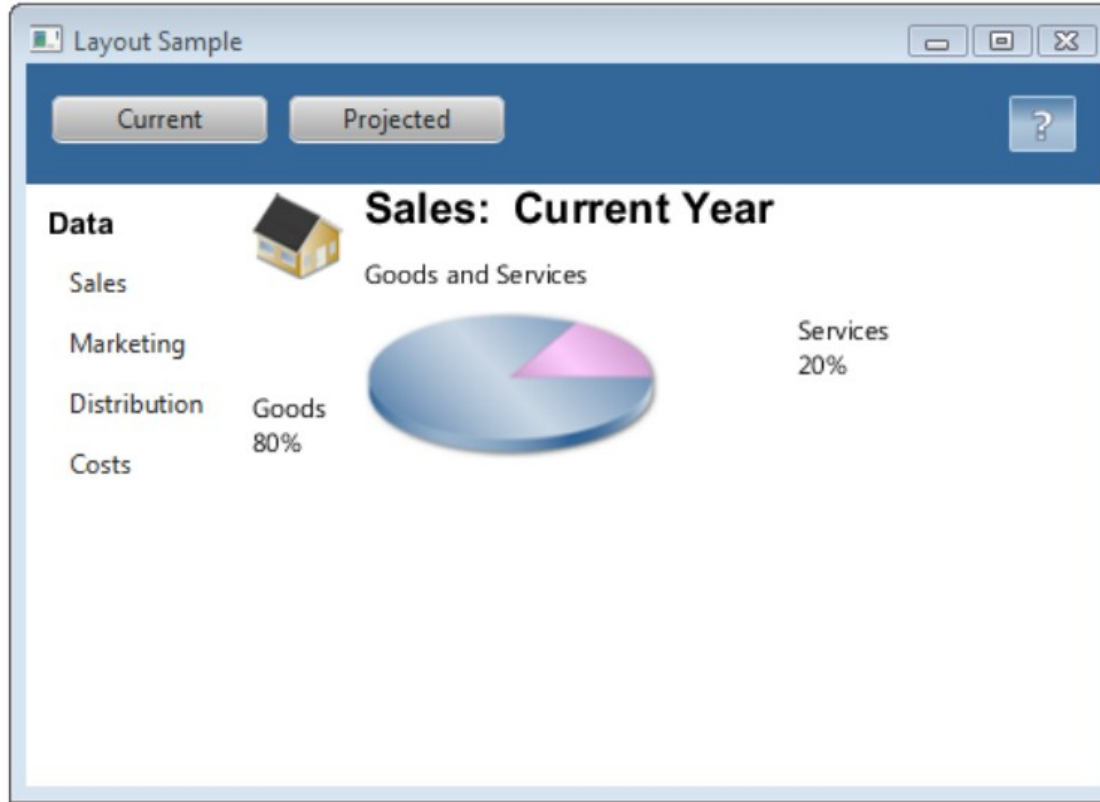
```
Text servicesPercent = new Text("Services\n20%");  
GridPane.setValignment(servicesPercent, VPos.TOP);  
grid.add(servicesPercent, 3, 2);  
return grid;  
}
```

→ Note:

- ✓ the arguments of GridPane's **add** method with two parameters represents **column index** and **row index** respectively.
- ✓ The arguments of GridPane's **add** method with four parameters represents **column index**, **row index**, **column span** and **row span** respectively.

Cont.

→ The **GridPane** in **BorderPane** looks like:



FlowPane

- The nodes within a **FlowPane** layout pane are laid out **consecutively** and **wrap at the boundary set for the pane**.
- Nodes can flow **vertically** (in columns) or **horizontally** (in rows).
- A vertical flow pane wraps at the height boundary for the pane.
- A horizontal flow pane wraps at the width boundary for the pane.
- Figure in the next slide shows a sample horizontal flow pane using numbered icons.
- By contrast, in a vertical flow pane, column one would contain pages one through four and column two would contain pages five through eight.
- Gap properties can be set to manage the spacing between the rows and columns.
- The padding property can be set to manage the distance between the nodes and the edges of the pane.

Cont.



Cont.

→ **Example:** (Create FlowPane)

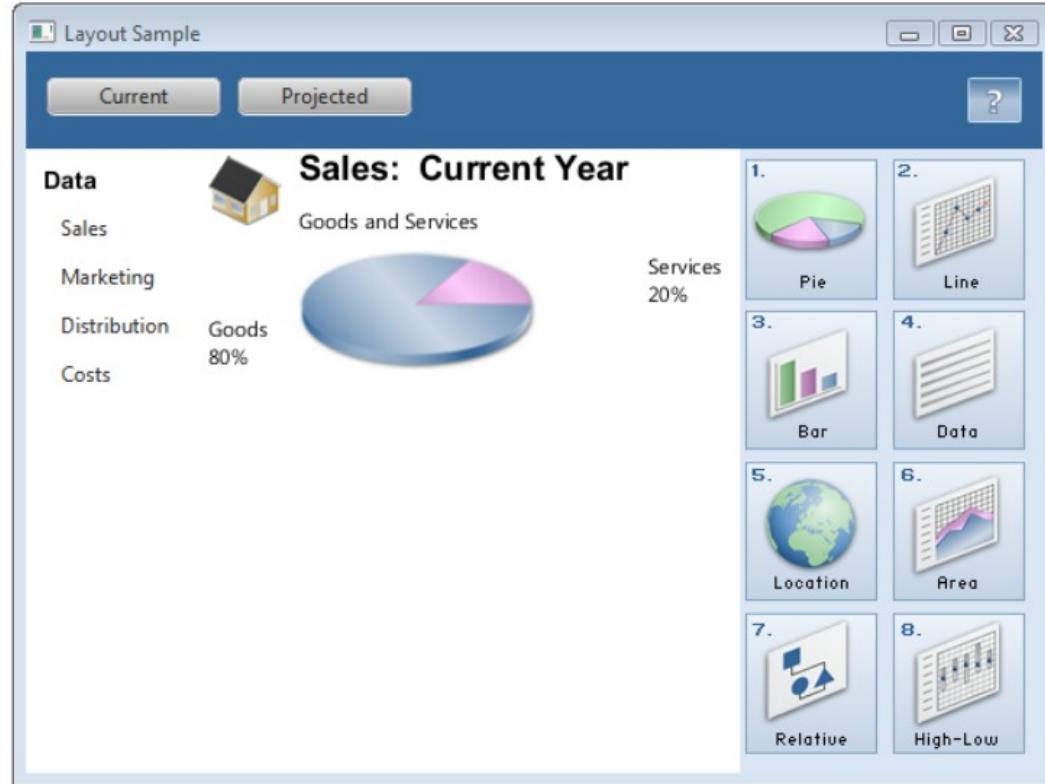
```
public FlowPane addFlowPane() {  
    FlowPane flow = new FlowPane();  
    flow.setPadding(new Insets(5, 0, 5, 0));  
    flow.setVgap(4);  
    flow.setHgap(4);  
    flow.setPrefWrapLength(170);  
    flow.setStyle("-fx-background-color: DAE6F3;");  
    ImageView pages[] = new ImageView[8];
```

Cont.

```
for (int i=0; i<8; i++) {  
    pages[i] = new ImageView(  
        new Image(LayoutSample.class.getResourceAsStream(  
            "graphics/chart_"+(i+1)+".png")));  
    flow.getChildren().add(pages[i]);  
}  
return flow;  
}
```

Cont.

→ The **FlowPane** in **BorderPane** looks like:



TilePane

- A tile pane is similar to a flow pane.
- The TilePane layout pane places all of the nodes in a grid in which each cell, or tile, is **the same size**.
- Nodes can be laid out **horizontally** (in rows) or **vertically** (in columns).
- Horizontal tiling wraps the tiles at the tile pane's width boundary and vertical tiling wraps them at the height boundary.
- Use the **prefColumns** and **prefRows** properties to establish the preferred size of the tile pane.
- Gap properties can be set to manage the spacing between the rows and columns.
- The padding property can be set to manage the distance between the nodes and the edges of the pane.

Cont.

- **Example:** (Create TilePane) (Creates horizontal TilePane)

```
TilePane tile = new TilePane();  
tile.setPadding(new Insets(5, 0, 5, 0));  
tile.setVgap(4);  
tile.setHgap(4);  
tile.setPrefColumns(2);  
tile.setStyle("-fx-background-color: DAE6F3;");  
ImageView pages[] = new ImageView[8];
```

Cont.

```
for (int i=0; i<8; i++) {  
    pages[i] = new ImageView(  
        new Image(LayoutSample.class.getResourceAsStream(  
            "graphics/chart_"+(i+1)+".png")));  
    tile.getChildren().add(pages[i]);  
}
```

AnchorPane

- The **AnchorPane** layout pane enables you to anchor nodes to the **top**, **bottom**, **left** side, **right** side, or **center** of the pane.
- As the window is resized, the nodes **maintain their position** relative to their anchor point.
- Nodes can be **anchored to more than one position** and more than one node can be anchored to the same position.
- Figure in the next slide shows an anchor pane with the grid pane from **GridPane** **anchored to the top** and an **HBox pane with two buttons anchored to the bottom** and the **right side**.

Cont.



Cont.

→ **Example:** (Create AnchorPane)

```
public AnchorPane addAnchorPane(GridPane grid) {  
    AnchorPane anchorpane = new AnchorPane();  
    Button buttonSave = new Button("Save");  
    Button buttonCancel = new Button("Cancel");  
    HBox hb = new HBox();  
    hb.setPadding(new Insets(0, 10, 10, 10));  
    hb.setSpacing(10);  
    hb.getChildren().addAll(buttonSave, buttonCancel);  
    anchorpane.getChildren().addAll(grid, hb);  
}
```

Cont.

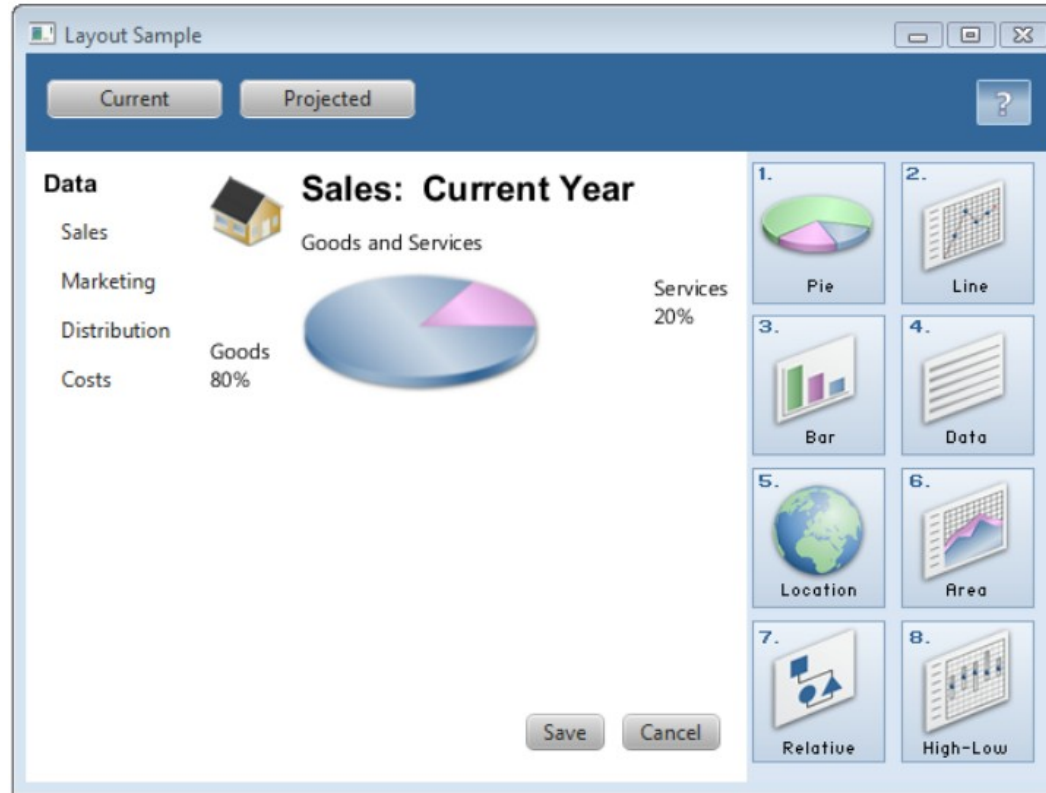
```
AnchorPane.setBottomAnchor(hb, 8.0);  
AnchorPane.setRightAnchor(hb, 5.0);  
AnchorPane.setTopAnchor(grid, 10.0);  
return anchorpane;  
}
```

- The following statement replaces the center region of the border pane with the anchor pane:

```
border.setCenter(addAnchorPane(addGridPane()));
```

Cont.

→ The **AnchorPane** in **BorderPane** looks like:





Thank You!!