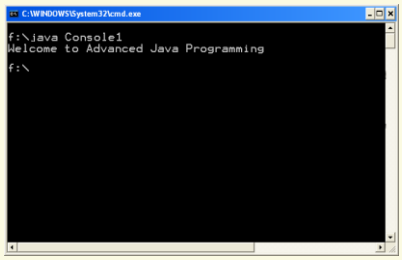
JavaFX Basics

CST141

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 **J**ava **V**irtual **M**achine (JVM) always has a command window (console) running

A Console Application



A Window Application

The javac Compiler (Page 1)

- Sun Microsystem's Java **compiler** is named "javac.exe" and may be run from implicitly from an IDE or from the command line
  - Complier probably is located in folder:
    - "c:\Program Files\Java\jdk1.8.0\bin"
    - The "jdk" numbers will vary base upon your version of Java
- Successfully compiling the java **source file** (.java file) creates a **class** file (.class file)
  - ***Compile errors*** are displayed in console window

The javac Compiler (Page 2)

- At command prompt, type **javac** followed by filename including the "**.java**" extension
- Format:

  ***path*:\javac *SourceFileName*.java**
  - ***SourceFileName*** is case sensitive
  - The **path** environmental variable may need an entry naming the location of the compiler
- Examples:

```
javac Console1.java
c:\Program Files\Java\jdk1.8.0\bin\javac
  Console1.java
```

JVM (Page 1)

- Sun Microsystem's **J**ava **V**irtual **M**achine is the runtime system (application) which ***executes*** the compiled Java class file
- The JVM is named "java.exe"
  - JVM probably also is located in folder:
    - "c:\Program Files\Java\jdk1.8.0\bin"
    - The "jdk" numbers will vary based upon your version of Java

JVM (Page 2)

- To run a Java application from the console, type **java** followed by the class filename
  - The "**.class**" extension should not be included
- Format:

  path*:\java *ClassFileName
  - ***ClassFileName*** is case sensitive
- Examples:

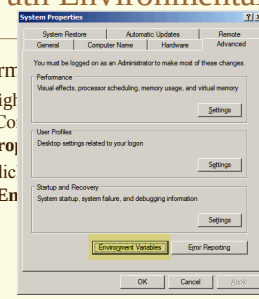
```
java Console1
c:\Program Files\Java\jdk1.8.0\bin\java
  Console1
```

The Path Environmental Variable (Page 1)

- The **path** to a file is basically its address on the computer (the drive plus any directories and sub-directories where file is located)
- The **PATH** environment variable specifies the ***command search path***
- Any file in a directory listed in the PATH variable can be found by direct reference to the name of the file
  - You do not have to specify the file's path

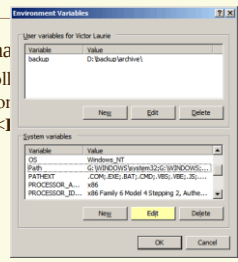
The Path Environmental Variable (Page 2)

- To perm ariable:
  1. Righ mply
     "Co and select
     Pro
  2. Clic e
     <**En**



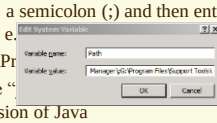
The Path Environmental Variable (Page 3)

- To perma variable (con.):
  3. Scroll window at the
     botto ind it; then click
     the <E



The Path Environmental Variable (Page 4)

- To permanently change the PATH variable (con.):
  4. Type a semicolon (;) and then enter then the added path, e.
     - "c:\Pr in"
     - The " e upon your version of Java



The String[] args Parameter (Page 1)

- In the header for the **main** () method:

```
public static void main(String[] args)
```

- … the parameter **args** is an ***array*** of type **String** and dynamic (undeclared) size
- Method main () is the first method that executes in a Java application
  - ***It never is called*** by another method …
- So how can arguments be passed to it?

Return to String[] args (Page 3)

The String[] args Parameter (Page 2)

- One method is when argument values are passed to application from the ***console*** when the program is executed by the JVM, e.g.

```
java Console3 m 34
```

  - In this instance, **m** and **34** are the arguments …
  - So before method main () begins to execute, **args** would be instantiated as "**String args[2]**" with the following assignments:
    - `args[0] = "m"`
    - `args[1] = "34"`

The **length** Property of an Array

- As with any ***array***, the **length** property stores the size of (number of elements in) the array
- Format:

```
arrayObject.length
```

- Example:

```
if (args.length != 2)
{ …
```

The NetBeans IDE

The Application Class (Page 1)

- The JavaFX classes give developers the flexibility to create customized GUI windows
- A class that **extends** Application is a "JavaFX" application
- Located in the **javafx.application** package:

```
import javafx.application.Application;
```

The Application Class (Page 2)

- Format:

```
public class ClassName extends
  Application { …
```

- Example:

```
public class JavaFX1 extends
  Application { …
```

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
- The window contains the application
- Located in the **javafx.stage** package:

```
import javafx.stage.Stage;
```

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) {
  …
```

The Scene Class (Page 1)

- A Scene object is the container for all content placed inside the Stage object
- Its content is represented as a hierarchical "scene graph" of nodes
- Located in the **javafx.scene** package:

```
import javafx.scene.Scene;
```

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***

The Scene Class (Page 3)

- Examples:

```
Scene scene = new Scene(buttonOK, 300,
  300);
Scene scene = new Scene(pane);
```

The start() Method

- 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

```
@Override
public void start(Stage primaryStage) {
  …
```

  - Overrides **abstract** start() from class **Application**

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");
```

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);
```

The show() Method

- A method of a **Stage** object that displays the window (Stage)
- Format:

```
stageObject.show();
```

- Example:

```
primaryStage.show();
```

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 controls) such, e.g. TextFields, Buttons, etc.
  - Pane objects—layout panes for position objects in a scene
  - Shape objects—lines, circles, text, etc.
  - ImageView controls—for displaying images

The Button Class (Page 1)

- **Button** is a JavaFX class used to instantiate UI 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;
```

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");
```

The setText() Method

- Assigns a string value to an object, e.g. a Button
- Format:

```
object.setText(text);
```

- Example:

```
buttonOK.setText("OK");
```

JavaFX1.java

Layout Classes (Page 1)

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

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

The StackPane Class

- 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)
- Located in the **javafx.scene.layout** package:

```
import javafx.scene.layout.StackPane;
```

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**
- Method is required for many layout pane classes (Pane, FlowPane, StackPane, HBox, VBox) to be able to add new nodes to those layouts

The getChildren() Method (Page 2)

- Format:

```
paneObject.getChildren()
```

- Example:

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

The add() Method

- Adds a node (UI control, shape, pane or image view) to the collection of children from a pane layout object
- Format:

```
paneObject.getChildren().add(node);
```

- Example:

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

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);
```

JavaFX2

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;
```

The GridPane Class (Page 2)

- Constructor format:

```
GridPane gridPaneObject = new GridPane();
```

- Example:

```
GridPane grid = new GridPane();
```

The add() Method for GridPane

- The **add()** method for GridPane adds the node at the specified ***columnIndex*** and ***rowIndex*** position
- Format:

```
gridPaneObject.add(node, columnIndex,
  rowIndex);
```

  - The ***node*** is the object added in that cell
- Example:

```
grid.add( new TextField(), 1, 0);
```

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

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

The setHalignment() Method

- A **static** method that sets the horizontal alignment for a node contained within a GridPane object
- Format:

```
GridPane.setHalignment(node,
  HPos.CENTER/HPos.LEFT/HPos.RIGHT);
```

- Example:

```
GridPane.setHalignment(buttonAdd,
  HPos.CENTER);
```

Set Valignment() Method

- A **static** method that sets the vertical alignment for a node contained within a GridPane object
- Format:

```
GridPane.setValignment(node,
  VPos.BASELINE/VPos.BOTTOM/VPos.CENTER/
  VPos.TOP);
```

- Example:

```
GridPane.setValignment(buttonSubtract,
  VPos.BASELINE);
```

Set Hgap() Method

- A method that sets the width of the horizontal gaps between columns in a GridPane object
- Format:

```
gridPaneObject.setHgap(doubleValue);
```

- Example:

```
grid.setHgap(2);
```

Set Vgap() Method

- A method that sets the height of the vertical gaps between rows in a GridPane object
- Format:

```
gridPaneObject.setVgap(doubleValue);
```

- Example:

```
grid.setVgap(5);
```

JavaFX UI Controls

- Windows-type ***controls*** with which users interact by using the mouse or keyboard
- Some basic GUI 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***
  - CheckBox—clicked "on" and "off"
  - ComboBox—drop-down list of choices
  - List—open "scrollable" area with list of choices

The TextField Class (Page 1)

- A **TextField** is a 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;
```

The TextField Class (Page 2)

- Format to declare a **JTextField** 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");
```

The Label Class (Page 1)

- A **Label** is 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 **javax.scene.control** package:

```
import javax.scene.control.Label;
```

The Label Class (Page 2)

- Format to declare a **JLabel** object:

```
Label labelObject = new Label(
  [textString [, graphicObject ] ] );
```

  - ***textSting*** 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***

The Label Class (Page 3)

- Format to declare a **JLabel** object:

```
Label labelObject = new Label(
  [textString [, graphicObject ] ] );
```

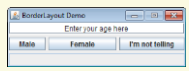
- Examples:

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

JavaFX3

The BorderPane Class (Page 1)

- A **BorderLayout** has five areas
  - TOP
  - LEFT
  - CENTER
  - RIGHT
  - BOTTOM



- Located in the **javafx.scene.layout** package

```
import javafx.scene.layout.BorderPane;
```

The BorderPane Class (Page 2)

- 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 of the areas of a BorderLayout object are use—perhaps just the top, left and right

The BorderPane Class (Page 3)

- Constructor format:

```
BorderPane borderLayoutObject = new
      BorderPane();
```

- Example:

```
BorderPane border = new BorderPane();
```

The "set" Methods for BorderPane

- To "set" methods for **BorderPane** specifiy the area 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 ") );
```

JavaFX4

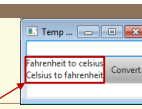
The HBox Class

The VBox Class

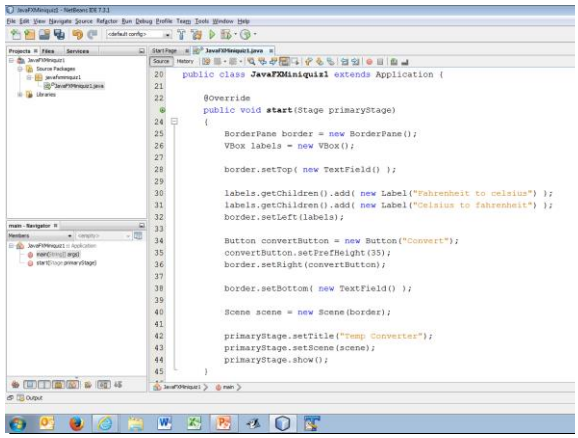
Nesting Pane Nodes

JavaFX5

Mini-Quiz No. 1



- Start a new project named "JavaFXMiniquiz1" as per the image above:
  - The layout manager for the scene is **BorderPane**
  - TextFields are placed in the TOP and BOTTOM areas
  - A **VBox** is placed in the LEFT area with two Labels
  - A Button is placed in the RIGHT area
  - The window properties include title "Temp Converter"
  - ***Do not*** use the CENTER area of BorderPane
  - If you have time research the Java API documentation to discover how to set the "height" of a Button (***Hint***: check the methods of its superclasses)



```
public class JavaFXMiniquiz1 extends Application {

    @Override
    public void start(Stage primaryStage)
    {
        BorderPane border = new BorderPane();
        VBox labels = new VBox();

        border.setTop( new TextField() );

        labels.getChildren().add( new Label("Fahrenheit to celsius") );
        labels.getChildren().add( new Label("Celsius to fahrenheit") );
        border.setLeft(labels);

        Button convertButton = new Button("Convert");
        convertButton.setPrefHeight(35);
        border.setRight(convertButton);

        border.setBottom( new TextField() );

        Scene scene = new Scene(border);

        primaryStage.setTitle("Temp Converter");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

The FlowPane Class (Page 1)

- Arranges components left to right, top to bottom
  - Like lines of text in a paragraph
- Located in the **javafx.scene.layout** package:

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

The FlowPane Class (Page 2)

- Format to instantiate a **FlowLayout** object:

```
FlowPane flowPaneObject = new
  FlowPane();
```

- Example:

```
FlowPane flow = new FlowPane();
```

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

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

The Font Class (Page 1)

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

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

The Font Class (Page 2)

- Format to declare a **Color** object:

```
Font fontObject = new Font(
                  [typeFaceString,]
                  sizeInt);
```

  - ***typeFaceString*** is the name of a font installed on the user's computer
  - ***sizeInt*** is the font size measured in "points"
- Example:

```
Font font = new Font("Comic Sans MS",
                     14);
```

The Font.font() Method (Page 1)

- A **static** method that returns a font object with some or all of the Font attributes:
  - Typeface (font name)
  - Bold property
  - Posture (italic) property
  - Font size

The Font.font() Method (Page 2)

- Format:

```
Font.font(typeFaceString, [
       [FontWeight.WEIGHT_CONSTANT, ]
       [FontPosture.POSTURE_CONSTANT, ]
       sizeInt)
```

- Example:

```
Font font = Font.font("Comic Sans MS",
            FontWeight.BOLD,
            FontPosture.ITALIC, 20);
```

The Circle Class

- Draws a circle of specific radius, and x- and y-coordinates
- Circle and several other graphic objects are direct **abstract** subclasses of **Shape**
- Format:

```
Circle circleObject = new
      Circle([[centerXDouble,
      centerYDouble, ] radiusDouble]);
```

- Example:

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

The Circle.setCenterX() and Circle.setCenterY Methods

- Sets the x- and y-coordinates the center of a Circle object
- Format:

```
circleObject.setCenterX(centerXDouble);
circleObject.setCenterY(centerYDouble);
```

- Example:

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

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;
```

The Color Class (Page 2)

- Format to declare a **Color** object:

```
Color colorObject = new Color(redDouble,
                    greenDouble, blueDouble,
                    [opacityDouble]);
```

- Examples:

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

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

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 theses constants are BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE and YELLOW
- Example:

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

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(colorObject);
```

- Example:

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

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;
```

The Image Class (Page 2)

- Format to declare an Image object:

```
Image iconObject = 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("home.png");
```

The ImageView Class

- An object for displaying (placing into a pane or scene) an Image object
- Format:

```
ImageView imageViewObject = new
          ImageView(imageObject);
```

- Example:

```
ImageView imageView = new ImageView(
          new Image("home.png") );
```

JavaFX6

JavaFX7

The **setBackground** Method

- Sets background color (the color behind) of "swing" GUI components
- Format:

```
jComponentObject.setBackground(
  colorObject );
```

  - The ***colorObject*** may be instantiated from class Color or a Color constant
- Examples:

```
button.setBackground(Color.BLUE);
```

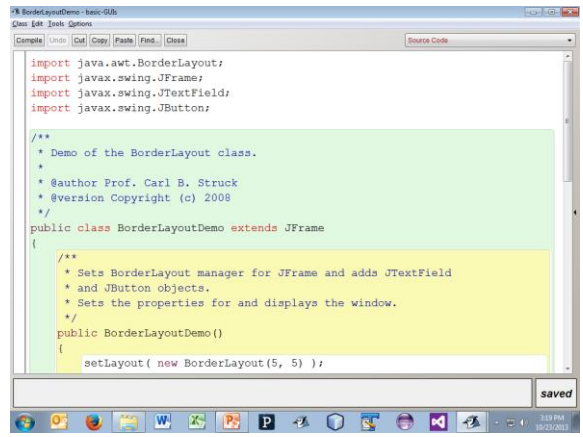
The **setForeground** Method

- Sets foreground color (the font color) of "swing" GUI components
- Format:

```
jComponentObject.setForeground(
  colorObject );
```

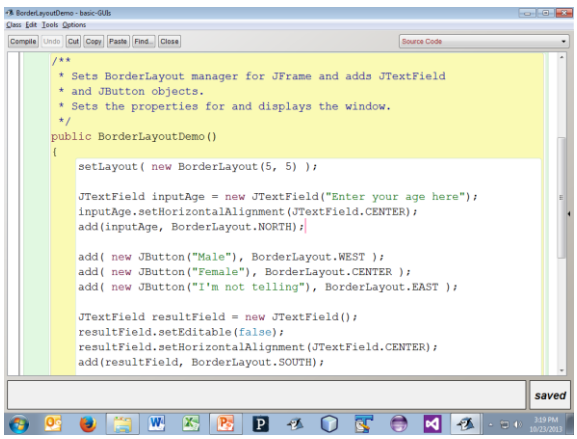
  - The ***colorObject*** may be instantiated from class Color or a Color constant
- Examples:

```
button.setBackground(Color.BLUE);
```



```
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JButton;

/**
 * Demo of the BorderLayout class.
 *
 * @author Prof. Carl B. Struck
 * @version Copyright (c) 2008
 */
public class BorderLayoutDemo extends JFrame
{
    /**
     * Sets BorderLayout manager for JFrame and adds JTextField
     * and JButton objects.
     * Sets the properties for and displays the window.
     */
    public BorderLayoutDemo()
    {
        setLayout( new BorderLayout(5, 5) );
```

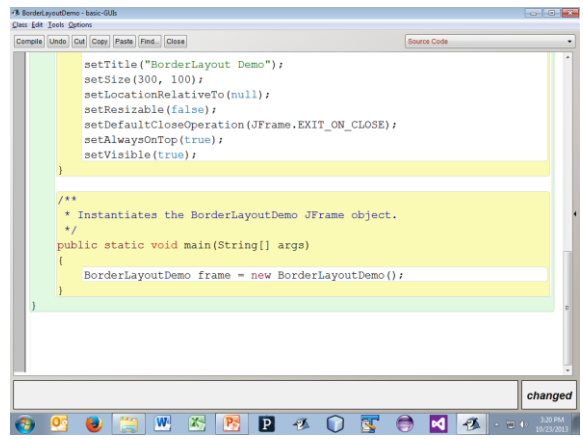


```
    /**
     * Sets BorderLayout manager for JFrame and adds JTextField
     * and JButton objects.
     * Sets the properties for and displays the window.
     */
    public BorderLayoutDemo()
    {
        setLayout( new BorderLayout(5, 5) );

        JTextField inputAge = new JTextField("Enter your age here");
        inputAge.setHorizontalAlignment(JTextField.CENTER);
        add(inputAge, BorderLayout.NORTH);

        add( new JButton("Male"), BorderLayout.WEST );
        add( new JButton("Female"), BorderLayout.CENTER );
        add( new JButton("I'm not telling"), BorderLayout.EAST );

        JTextField resultField = new JTextField();
        resultField.setEditable(false);
        resultField.setHorizontalAlignment(JTextField.CENTER);
        add(resultField, BorderLayout.SOUTH);
```



```
        setTitle("BorderLayout Demo");
        setSize(300, 100);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setAlwaysOnTop(true);
        setVisible(true);
    }

    /**
     * Instantiates the BorderLayoutDemo JFrame object.
     */
    public static void main(String[] args)
    {
        BorderLayoutDemo frame = new BorderLayoutDemo();
    }
}
```

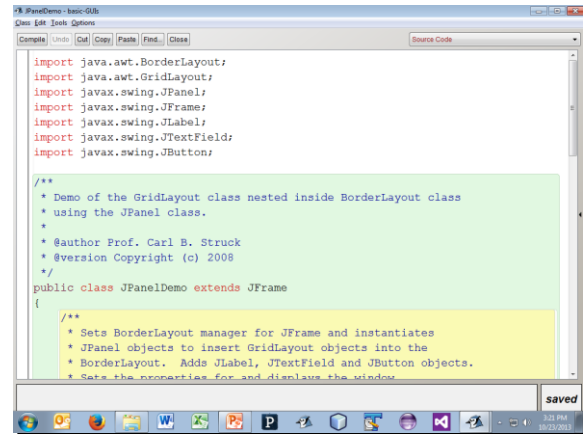
The **add** Method

- Method of class **JFrame** (inherited from class **Container** ← Window ← Frame ← JFrame) that attaches a GUI component object to the window
- Format:

```
[jFrameObject.]add(jGUIObject);
```

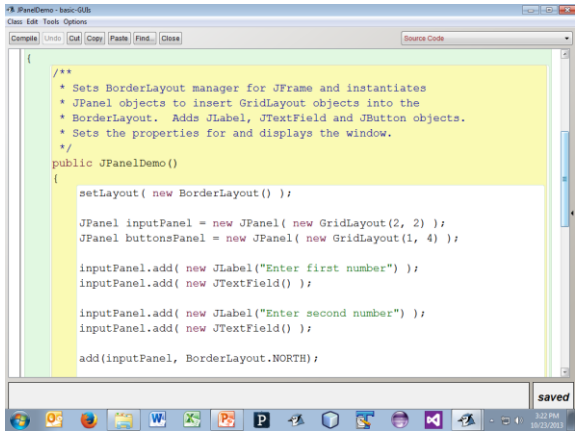
- Example:

```
add( new JTextField(10) );
```



```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;

/**
 * Demo of the GridLayout class nested inside BorderLayout class
 * using the JPanel class.
 *
 * @author Prof. Carl B. Struck
 * @version Copyright (c) 2008
 */
public class JPanelDemo extends JFrame
{
    /**
     * Sets BorderLayout manager for JFrame and instantiates
     * JPanel objects to insert GridLayout objects into the
     * BorderLayout.  Adds JLabel, JTextField and JButton objects.
```



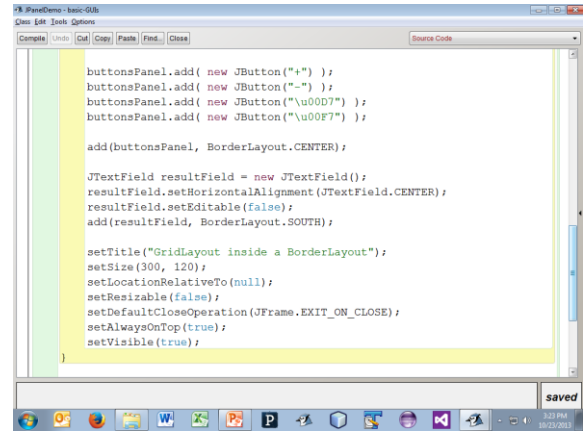
```
{
    /**
     * Sets BorderLayout manager for JFrame and instantiates
     * JPanel objects to insert GridLayout objects into the
     * BorderLayout.  Adds JLabel, JTextField and JButton objects.
     * Sets the properties for and displays the window.
     */
    public JPanelDemo()
    {
        setLayout( new BorderLayout() );

        JPanel inputPanel = new JPanel( new GridLayout(2, 2) );
        JPanel buttonsPanel = new JPanel( new GridLayout(1, 4) );

        inputPanel.add( new JLabel("Enter first number") );
        inputPanel.add( new JTextField() );

        inputPanel.add( new JLabel("Enter second number") );
        inputPanel.add( new JTextField() );

        add(inputPanel, BorderLayout.NORTH);
```

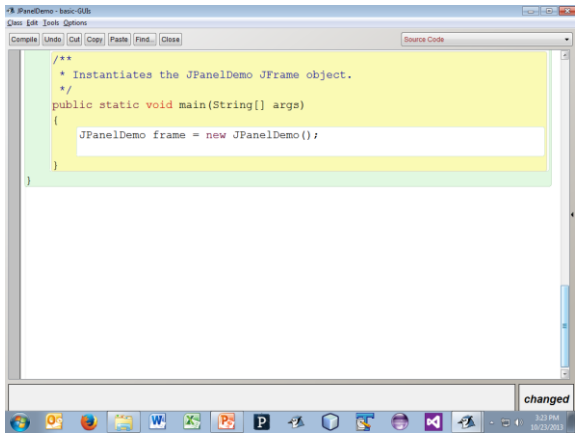


```
        buttonsPanel.add( new JButton("+") );
        buttonsPanel.add( new JButton("-") );
        buttonsPanel.add( new JButton("\u00D7") );
        buttonsPanel.add( new JButton("\u00F7") );

        add(buttonsPanel, BorderLayout.CENTER);

        JTextField resultField = new JTextField();
        resultField.setHorizontalAlignment(JTextField.CENTER);
        resultField.setEditable(false);
        add(resultField, BorderLayout.SOUTH);

        setTitle("GridLayout inside a BorderLayout");
        setSize(300, 120);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setAlwaysOnTop(true);
        setVisible(true);
    }
```



```
    /**
     * Instantiates the JPanelDemo JFrame object.
     */
    public static void main(String[] args)
    {
        JPanelDemo frame = new JPanelDemo();
    }
}
```

http://download.oracle.com/javase/7/docs/api/javax/swing/JCheckBox.html

The **JCheckBox** Class (Page 1)

- ***Subclass*** of the **JToggleButton** class, instantiates ***check box*** objects which have on/off (**true**/**false**) values
- Click once turns it ***on***; next click turns it ***off***
- Located in the **javax.swing** package:

```
import javax.swing.JCheckBox;
```

The **JCheckBox** Class (Page 2)

- Format to declare a **JCheckBox** object:

```
JCheckBox jCheckBoxObject = new JCheckBox
  ([stringObject, [true/false]]);
```

  - ***stringObject*** is the default string displayed to the right of the check box (its caption)
  - **boolean** argument sets button initially ***on*** or ***off***
- Example:

```
JCheckBox bold = new JCheckBox("Bold");
```

http://download.oracle.com/javase/7/docs/api/javax/swing/JRadioButton.html

The **JRadioButton** Class (Page 1)

- ***Subclass*** of **JToggleButton** class, instantiates ***radio*** (option) ***button*** objects which have on/off (**true**/**false**) values
- If ***item listening*** has been activated for the component, generates an **ItemEvent** when the user clicks the button
- Located in the **javax.swing** package:

```
import javax.swing.RadioButton;
```

The **JRadioButton** Class (Page 2)

- Radio buttons are usually ***grouped***
  - ***Only one*** button in the group may be selected at any moment
  - Clicking one radio button in the group turns off any other in the group that is currently on

The **JRadioButton** Class (Page 3)

- Format to declare a **JRadioButton** object:

```
JRadioButton jRadioButtonObject = new
  JRadioButton([stringObject],
  [true/false]);
```

  - ***stringObject*** is the default string displayed to the right of the button
  - **boolean** argument sets button initially ***on*** or ***off***
- Example:

```
JRadioButton size8 = new
  JRadioButton("8", false);
```

http://download.oracle.com/javase/7/docs/api/javax/swing/ButtonGroup.html

The **ButtonGroup** Class (Page 1)

- Creates a ***functional relationship*** between radio buttons
  - ***Not a visual*** GUI component
- The **add** method of objects instantiated from the **ButtonGroup** object places radio buttons into the group
- Located in the **javax.swing** package:

```
import javax.swing.ButtonGroup;
```

The **ButtonGroup** Class (Page 2)

- Format to declare a **ButtonGroup** object:

```
ButtonGroup jButtonGroupObject = new
  ButtonGroup();
```

- Example:

```
ButtonGroup fontGroup = new
  ButtonGroup();
```

The **add** Method for ButtonGroup

- For the **ButtonGroup** object, adds a ***radio button*** to the "logical" group
- Before the **add** method inserts **JRadioButton** into group, it behaves like a **JCheckBox**
- Format:

```
jButtonGroupObject.add(
  jRadioButtonObject );
```

- Example:

```
fontGroup.add(size8);
```

http://download.oracle.com/javase/7/docs/api/javax/swing/JComboBox.html

The **JComboBox** Class (Page 1)

- ***Subclass*** of **JComponent** class, instantiates ***combo box*** (drop-down list) objects from which users may select an item
- If ***item listening*** has been activated for any component, generates an **ItemEvent** when the user selects from the list
- Located in the **javax.swing** package:

```
import javax.swing.JComboBox;
```

The **JComboBox** Class (Page 2)

- Format to declare a **JComboBox** object:

```
JComboBox jComboBoxObject = new
  JComboBox( [stringArray] );
```

  - The **stringArray** provides the items for the list
  - The argument is ***optional*** (however items must then be added ***later*** using the **add** method)
- Example:

```
JComboBox fonts = new
  JComboBox(fontNames);
```

The **setMaximumRowCount** Method

- For a **JComboBox** object, sets maximum ***number of rows*** displayed when the list drops down
- Automatically displays a ***scroll bar*** if the number of items exceeds maximum rows
- Format:

```
jComboBoxObject.setMaximumRowCount(
  numericInt);
```

- Example:

```
fonts.setMaximumRowCount(3);
```

The **setToolTipText** Method

- Method of **JLabel** and other GUI component objects that defines tool tip text for the object
- ***Tool tip*** is the text displayed when the mouse pointer ***hovers*** over the object
- Format:

```
guiObject.setToolTipText(stringObject);
```

- Example:

```
label.setToolTipText("Text only");
```

The **setHorizontalTextPosition** Method

- Method of **JLabel** and other GUI component objects that define where an icon appears ***horizontally*** relative to the object text
- Uses ***constants*** of the **SwingConstants** interface
- Format:

```
jGuiObject.setHorizontalTextPosition(
  SwingConstants.POSITION_CONSTANT);
```

- Examples:

```
label.setHorizontalTextPosition(
  SwingConstants.CENTER);
```

The **setVerticalTextPosition** Method

- Method of **JLabel** and other GUI component objects that define where an icon appears ***vertically*** relative to text
- Uses ***constants*** of the **SwingConstants** interface
- Format:

```
jGuiObject.setVerticalTextPosition(
  SwingConstants.POSITION_CONSTANT);
```

- Examples:

```
label.setVerticalTextPosition(
  SwingConstants.BOTTOM);
```