1 **Java GUI 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

3 **Java "Swing" GUI Components**

- Windows-type *controls* with which users interact by using the mouse or keyboard
- Some basic GUI component classes are:
  - JLabel—a text element that is *not editable*
  - JTextField—an input textbox
  - JButton—command button that triggers an event when *clicked*
  - JCheckBox—clicked "on" and "off"
  - JComboBox—drop-down list of choices
  - JList—open area with list of choices (scrollable)

4 **The JFrame Class (Page 1)**

- Defines a window object with *Title bar*, and *Minimize*, *Maximize* and *Close* buttons
- Any class that extends super class JFrame may contain *GUI components*
- JFrame extends Frame extends Window extends Container extends Component extends Object
- Located in the javax.swing package:

```
import javax.swing.JFrame;
```

5 **The JFrame Class (Page 2)**

- Format to call the overloaded constructor that instantiates a JFrame object:

  JFrame *jFrameObject* = new JFrame("*Titlebar Text*");

  - The "*Titlebar Text*" is the text that appears in the title bar of the window
- Example:

```
JFrame frame = new JFrame("JFrame Demo");
```

6 **The JFrame Class (Page 3)**

- The class that extends JFrame "is a" JFrame and can call all its methods, e.g.

```
public class FlowLayoutDemo extends JFrame
{
    FlowLayoutDemo()
    {
        add( new JLabel("A label") );
        …
```

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

- Instantiates *text field* objects
- A text field is a single-line box into which a user may type text from the keyboard
- Located in the javax.swing package:
  import javax.swing.JTextField;

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

- Format to declare a JTextField object:
  <u>JTextField</u> *jTextFieldObject* = new <u>JTextField</u>([*stringObject*,] [*columns*]);
  – *stringObject* is default string displayed (optional)
  – *columns* is *width* in average character size of current font (optional—default is 1)
- Examples:
  JTextField textField1 = new JTextField(10);
  JTextField textField2 = new JTextField("Enter number", 16);

9 **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*.]<u>add</u>(*jGUIObject*);
- Example:
  add(new JTextField(10));

10 **The setSize Method**

- Method of class JFrame (inherited from class Window ← Frame ← JFrame) that determines the *width* and *height* of the window in pixels
- Format:
  [*jFrameObject*.]<u>setSize</u>(*widthInt*, *heightInt*);
- Examples:
  setSize(200, 75);
  frame.setSize(200, 75);

11 **The setResizable Method (Page 1)**

- Method of class JFrame (inherited from class Frame ← JFrame) that determines if the frame window may be resized
  – By dragging the mouse on one of its borders
- Sets boolean value—default is true

12 **The setResizable Method (Page 2)**

- Format:
  [*jFrameObject*.]<u>setResizable</u>(true/false);
- Example:
  setResizable(false);
  frame.setResizable(false);

13 The setLocationRelativeTo Method

- Method of class JFrame (inherited from class Window ← Frame ← JFrame)
- With single argument null places the window at the center of the screen
- Format:

  [*jFrameObject*.]setLocationRelativeTo(null);
- Examples:

  setLocationRelativeTo(null);

  frame.setLocationRelativeTo(null);

14 The setDefaultCloseOperation Method

- Method that determines what happens when an object window instantiated from the class that extends JFrame is closed
- Example:

  myApp.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  – The constant EXIT_ON_CLOSE from class JFrame causes application to end when the window closes

15 The setAlwaysOnTop Method

- Method of class JFrame (inherited from class Window ← Frame ← JFrame) that sets whether this window is always above other windows
- Sets a boolean value—default is false
- Format:

  [*jFrameObject*.]setAlwaysOnTop(true/false);
- Examples:

  setAlwaysOnTop(false);

  frame.setAlwaysOnTop(true);

16 The setVisible Method (Page 1)

- Method of class JFrame (inherited from class Window ← Frame ← JFrame) that determines if the window is *displayed* (or *not displayed*)
- Sets a boolean value—default is false

17 The setVisible Method (Page 2)

- Format:

  setVisible(true/false);

  – Argument is set to true if the JFrame object is to be displayed; false if *not*
- Examples:

  setVisible(true);

  super.setVisible(true);

18 Try It Out

- BasicGUIs.java—textFieldDemo()

19 **The JButton Class (Page 1)**

- Subclass of the Button class, it instantiates *command button* objects
- If event listening has been activated for the component, generates an ActionEvent when the user clicks the button
- Text on the button is called a *button label*
- Located in the javax.swing package:

  import javax.swing.JButton;

20 **The JButton Class (Page 2)**

- Format to declare a JButton object:

  JButton *jButtonObject* = new JButton([*stringObject*[, *iconObject*]]);

  – *stringObject*—appears as caption on the button
- Example:

  JButton button = new JButton("Click here");

21 **Try It Out**

- BasicGUIs.java—buttonDemo()

22 **The JLabel Class (Page 1)**

- A GUI component class that is used to create text objects displayed in a JFrame window which are *not editable*
- Often used nearby another GUI component in the window to *indicate its purpose*
- Located in the javax.swing package:

  import javax.swing.JLabel;

23 **The JLabel Class (Page 2)**

- Format to declare a JLabel object:

  JLabel *jLabelObject* = new JLabel(*textString* [, *iconObject*, *iconHorizontalPosition*]);

  – *textSting* is the text displayed for the label

  – *iconObject* is an optional image displayed with the text

  – *iconHorizontalPosition* is an int constant indicating where the icon appears relative to text

  – Overloaded so that any (and/or all) arguments to the constructor are *optional*

24 **The JLabel Class (Page 3)**

- Format to declare a JLabel object:

  JLabel *jLabelObject* = new JLabel(*textString* [, *iconObject*, *iconHorizontalPosition*]);
- Examples:

```
JLabel label1 =
    new JLabel("My first label");
JLabel label2 = new JLabel();
JLabel label3 =
    new JLabel("JLabel w/text & icon",
        new ImageIcon("home.gif"),
```

SwingConstants.LEFT);

25 **Try It Out**

- BasicGUIs.java—labelDemo()

26 **The SwingConstants Interface (Page 1)**

- An interface (*not a class*) that defines common int *constants* for use with swing components
- Remember an interface may contain *only*:
  - abstract methods are instantiated, but *must be* developed in classes that implement the interface
  - Declarations of *constants*
- Located in the javax.swing package:
  import javax.swing.SwingConstants;

27 **The SwingConstants Interface (Page 2)**

- Format to reference SwingConstants interface:
  swingObject.*swingMethod*(<u>SwingConstants</u>.*CONSTANT_IDENTIFIER*);
  - Sample constants are SwingConstants.LEFT, SwingConstants.CENTER, SwingConstants.BOTTOM, etc
- Example:
  label1.setHorizontalTextPosition(SwingConstants.CENTER);

28 **The FlowLayout Class (Page 1)**

- Layout managers control how components are added into JFrame objects
- There are several layout classes of which the most basic (*simplest*) is the FlowLayout class
- GUI components are added to a Container object left to right, top to bottom as they *fit*
- Located in the java.awt package:
  import java.awt.FlowLayout;

29 **The FlowLayout Class (Page 2)**

- Format to instantiate a FlowLayout object:
  <u>FlowLayout</u> *flowLayoutObject* = new <u>FlowLayout</u>();
- Example:
  FlowLayout f1 = new FlowLayout();

30 **The setLayout Method (Page 1)**

- Method of a JFrame object that defines a layout manager for the frame …
  - Layout managers determine the *position* and *size* of GUI components
- Layout normally is set *before* GUI components are added to a JFrame

31 **The setLayout Method (Page 2)**

- Format:

setLayout(*layoutObject*);

– The *layoutObject* may be from any layout manager class, not just FlowLayout

- Examples:

```
FlowLayout f1 = new FlowLayout();
setLayout(f1);

setLayout( new FlowLayout() );
```

32 **The setTitle Method**

- Method of a JFrame object that sets a string that is displayed in the window's *title bar*
- Format:

  [*jFrameObject*.]setTitle(string);

  – string will be displayed in the frame's title bar
- Example:

  setTitle("FlowLayout Demo");

33 **Try It Out**

- FlowLayoutDemo.java

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

- A GridLayout object places components in a grid of rows and columns
- Each component takes all available space within its cell, and each cell is exactly the same size …
  - If a window with a GridLayout is resized, the cell size of all objects changes so that the cells are as large as possible, given the space available to the container
- Located in the java.awt package

  import java.awt.GridLayout;

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

- Constructor format:

  GridLayout *flowLayoutObject* = new GridLayout(*rows*, *columns* [, *horizontalGap*, *verticalGap*]));

  – The *rows* and *columns* parameters are the number of rows and columns in the layout

  – The optional *horizontalGap* is the space between columns and *verticalGap* is the space between rows
- Example:

  setLayout(new GridLayout(5, 2, 5, 5));

36 **The add Method for GridLayout Managers**

- Objects are added to the JFrame with a GridLayout left-to-right, top-to-bottom, as they are inserted
- Format:

  add(*jGUIObject*);

- Example:
  add(new JTextField());

37 **Try It Out**

- GridLayoutDemo.java

38 **The BorderLayout Class (Page 1)**

- A BorderLayout has five areas specified by the five BorderLayout constants:
  - NORTH
  - WEST
  - CENTER
  - EAST
  - SOUTH
- Located in the java.awt package
  import java.awt.BorderLayout;

39 **The BorderLayout 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 a JFrame will not use all of the areas of the BorderLayout object—perhaps just the top, left and right

40 **The BorderLayout Class (Page 3)**

- Constructor format:
  BorderLayout *flowLayoutObject* = new BorderLayout([*horizonatalGap*, *verticalGap*]));
  - The optional *horizontalGap* and *verticalGap* is the horizontal and vertical space between the components
- Example:
  setLayout(new BorderLayout(5, 5));

41 **The add Method for BorderLayout Managers**

- To add objects to a BorderLayout container, the area to place the object must be specified
- Format:
  add(*jGUIObject*, BorderLayout.*LOCATION_CONSTANT*);
  - The valid BorderLayout.*LOCATION_CONSTANTS* are NORTH, WEST, CENTER, EAST and SOUTH
- Example:
  add(new JButton("Male", BorderLayout.WEST);

42 **Try It Out**

- BorderLayoutDemo.java

43 **The Panel Class (Page 1)**

- Panel is the simplest GUI *container* class
- It can provide space for an application to attach any other component, including other panels and/or frames
- Located in the java.awt package

  import java.awt.Panel;

44 **The Panel Class (Page 2)**

- For example one possible function of Panel could be to *nest* one layout container inside another …
  - E.g. a GridLayout container object could be placed inside a Panel
  - Then the Panel object could be placed inside a frame with BorderLayout manager

45 **The Panel Class (Page 3)**

- Constructor to format a JPanel with a *layout manager* parameter:

  JPanel *jPanelObject* = new JPanel(new *LayoutManagerClass*());
- Example:

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

46 **The add Method for Panels**

- To add a sub-layout to a JFrame, add individual components to the Panel object (not the JFrame):

  inputPanel.add(new JTextField());
- Once all components are added to the Panel, add the Panel to the JFrame (within its layout manager):

  add(inputPanel, BorderLayout.NORTH);
  - This example assumes that the JFrame is using a BorderLayout layout manager

47 **Try It Out**

- JPanelDemo.java

48 **Mini-Quiz No. 1**

- Start a new project “guis-miniquiz-1” and create a new class “MiniQuiz1” with a JFrame as per the image below:
  - The layout manager for the JFrame is BorderLayout
  - JTextFields are placed in the NORTH and SOUTH areas
  - A 2 row by 1 column (2 x 1) GridLayout with gaps (5, 5) is placed inside a Panel in the WEST area with two JLabels
  - A JButton is placed in the EAST area
  - JFrame window properties include title “Temp Converter” and size (300, 120)

49 **MiniQuiz1.java (Page 1)**

50 **MiniQuiz1.java (Page 2)**

51 **MiniQuiz1.java (Page 3)**

52 **The ImageIcon Class (Page 1)**

- Creates an ImageIcon object that references a graphics file of format GIF or JPEG or PNG
  - – Filename *extensions* are.gif or .jpg or .png
- ImageIcon extends from class Icon
  - – The call to the ImageIcon constructor returns an Icon object reference
- Located in the javax.swing package:

  import javax.swing.ImageIcon;

53 **The ImageIcon Class (Page 2)**

- Format to declare a ImageIcon object:

  <u>ImageIcon</u> *iconObject* = new <u>ImageIcon</u>("*path/filename*");

  – path/location is the Windows (and/or DOS) *filename* and *disk location* of a ".gif" or ".jpg" or ".png" file

54 **The ImageIcon Class (Page 3)**

- Format:

  <u>ImageIcon</u> *iconObject* = new <u>ImageIcon</u>("*path/filename*");
- Examples:

```
ImageIcon image = new ImageIcon("home.gif");
JLabel label = new JLabel(image);

JLabel label = new JLabel(new ImageIcon("home.gif") );
```

55 **Try It Out**

- ImageIconDemo.java

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

- Creates an Color object that use the RGB model (values between zero (0) to 255 that represent the amount of red, green and blue in makeup of color)
- Located in the java.awt package:

  import java.awt.Color;

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

- Format to declare a Color object:

  <u>Color</u> *colorObject* = new <u>Color</u>(*redInt*, *greenInt*, *blueInt*);
- Examples:

```
Color color = new Color(50, 150, 250);
button.setBackground(color);

button.setBackground( new Color(50, 150, 250) );
```

58 **The Color Constants**

- A series of 13 standard static constants of class Color that return an int representing an RGB color
- The constants are BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY,

MAGENTA, ORANGE, PINK, RED, WHITE and YELLOW

- Example:
  button.setBackground(Color.BLUE);

59 ☐ **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);

60 ☐ **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);

61 ☐ **The Font Class (Page 1)**

- Creates an Font object that …
- Located in the java.awt package:
  import java.awt.Color;

62 ☐ **The Font Class (Page 2)**

- Format to declare a Color object:
  Font *fontObject* = new Font(*typeFaceString*, *boldItalicInt*, *sizeInt*);
  – *typeFaceString* is the name of a font installed on the user's computer
  – *boldItalicInt* is an int the of sum Font.BOLD (1) and Font.ITALIC (2) constants that specifies the bold and/or italic style of the font using
  – *sizeInt* is the font size measured in "points"

63 ☐ **The Font Class (Page 3)**

- Example 1:
  Font font = new Color(new Font("Comic Sans MS", Font.BOLD + Font.ITALIC, 24));
  button.setFont(font);
- Example 2:
  button.setFont(new Font("Comic Sans MS", Font.BOLD + Font.ITALIC, 24));

64 ☐ **Try It Out**

- ColorFontDemo.java

65 ☐ **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;

66 **The JCheckBox Class (Page 2)**

- Format to declare a JCheckBox object:
  <u>JCheckBox</u> *jCheckBoxObject* = new <u>JCheckBox</u> ([*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");

67 **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;

68 **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

69 **The JRadioButton Class (Page 3)**

- Format to declare a JRadioButton object:
  <u>JRadioButton</u> *jRadioButtonObject* = new <u>JRadioButton</u>([*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);

70 **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;

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

- Format to declare a ButtonGroup object:

ButtonGroup *jButtonGroupObject* = new ButtonGroup();

- Example:

ButtonGroup fontGroup = new ButtonGroup();

72 **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);

73 **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;

74 **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);

75 **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);

76 **The setText Method (Page 1)**

- Method of JLabel and other GUI component objects that defines the text for the object
- *Alternate method* is to assign the text as an argument in the call to the *constructor* method of the object, e.g.

JLabel label = new JLabel("My label");

77 **The setText Method (Page 2)**

- Format:
  guiObject.setText(*stringObject*);
- Example:
  label.setText("My Label");

78 **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");

79 **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);

80 **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);

81 **Try It Out**

- OptionsDemo.java

82 **Event Handling**

- GUI components generate *events* when users interact with controls
- Typical events include:
  - Clicking the mouse
  - Moving the mouse
  - Typing in a text box (JTextField)
- When an event occurs, information about it is stored in an object of a class that

extends from class AWTEvent

83 **Event Listeners**

- To process an event, the programmer must:
  - Register (declare) an event listener
  - Implement one or more event handler methods
- When an event occurs, GUI component notifies the listener by *calling* the event's handling method(s)

84 **The ActionListener Interface (Page 1)**

- A class file that implements an interface must include all methods "defined" in the interface
  - May be a member of the Java API or written/developed by an application programmer
- ActionListener is an interface used to manage *event listening* and *event handling* for JButton's (and some other GUI components)
- Objects instantiated from a class that implements the ActionListener interface "are" event handlers, e.g. "*Is an* ActionListener"

85 **The ActionListener Interface (Page 2)**

- The method actionPerformed is declared inside the ActionListener interface that *must be defined* in any class that implements it
  - E.g. if a user clicks a JButton object, (and event listening is activated) the actionPerformed event handler method is called automatically
- Imported from java.awt.event package:

  import java.awt.event.ActionListener;

86 **The ActionListener Interface (Page 3)**

- Format:

  private class *EventHandlerClassName* <u>implements</u> <u>ActionListener</u>

  { …

  – implements instead of extends
- Example:

  private class ButtonEventHandler implements ActionListener

  { …

87 **The ActionEvent Class (Page 1)**

- Class that represents the variable *type* of parameter e in the header of the actionPerformed() method
- The variable e is a reference that stores the *event* information about the specific GUI component that *triggered the event*
- Imported from java.awt.event package:

  import java.awt.event.ActionEvent;

88 **The ActionEvent Class (Page 2)**

- Example:
```
private class ButtonEventHandler implements ActionListener
{
    public void actionPerformed(
        ActionEvent e)
    { ...
```

89 **Instantiating an ActionListener Object**

- An ActionListener *class* must have been defined previously
- Format:
  ActionListenerClass actionListenerObject = new *ActionListenerClass*();
- Example:
  ButtonEventHandler h1 = new ButtonEventHandler();

90 **The addActionListener Method (Page 1)**

- A method of a JButton (and other "action listener" GUI components) that assign an ActionListener object to the component
- The ActionListener object is the *argument* to the method
- This method effectively *activates* event listening
- Must be executed for every GUI component that will be an *event listener*

91 **The addActionListener Method (Page 2)**

- Format:
  jGuiComponentObject.addActionListener(*actionListenerObject*);
- Example:
  ok.addActionListener(h1);
  – The GUI component 'ok' is a JButton

92 **Try It Out**

- EventHandlerDemo_1.java

93 **The setEditable Method**

- Sets a boolean value that determines if JTextField object may be edited by a user
- Frequently is set to false if the object will be used for *output* only
- Format:
  jTextFieldObject.setEditable(true/false)
- Example:
  resultField.setEditable(false);

94 **The getText Method**

- Returns the String value currently stored in a JTextField (or another GUI component that has a text property) object
- The text property of the component is the value currently displayed in the text box
- Format:

jGuiObject.getText()
- Example:
  String sFirst = firstNumber.getText();

95 **The setText Method**
- Sets the contents of a JTextField object (or some other GUI component that has a text property) to a *new value*
- Format:
  jGuiObject.setText(*string*)
- Example:
  resultField.setText(resultString);

96 **Try It Out**
- EventHandlerDemo_2.java

97 **The selectAll Method**
- A method of class JTextField (inherited from class JTextComponent ← JTextField) that selects all the text in the object
  - As if it had been selected with a mouse
- Format:
  jComponentObject.selectAll();
- Example:
  inputAge.selectAll();

98 **Mini-Quiz No. 2 (Page 1)**
- Open the class file "BorderLayoutDemo" and modify it as follows:
  - Declare JTextField objects inputAge and resultField as private instance variables
  - Instantiate JButton objects male, female and notTelling and add the objects "by name" to the JFrame (still add them to the same BorderLayout areas as before)
  - Call selectAll() method for the inputAge JTextField

99 **BorderLayoutDemo.java (guis-miniquiz-2—Page 1)**

100 **BorderLayoutDemo.java (guis-miniquiz-2—Page 2)**

101 **BorderLayoutDemo.java (guis-miniquiz-2—Page 3)**

102 **Mini-Quiz No. 2 (Page 2)**
- Create an event handler class "MaleEventHandler" as follows:
  - Declare variable age as type int and retrieve (and convert) the value from the inputAge JTextField
  - Calculate the ideal age of the female spouse as:
    (age / 2 + 7)
  - Display the ideal age to the resultField JTextField in the format:
    Ideal age of spouse is 20
    - (Assuming 20 is the calculated value)

– Add “action listening” to the “male” JButton

103 ☐ **BorderLayoutDemo.java (guis-miniquiz-2—Page 4)**

104 ☐ **BorderLayoutDemo.java (guis-miniquiz-2—Page 2)**

105 ☐ **Mini-Quiz No. 2 (Page 3)**

- Create an event handler class “FemaleEventHandler” as follows:
  - Declare variable age as type int and retrieve (and convert) the value from the inputAge JTextField
  - Calculate the ideal age of the male spouse as:
    (age * 2 – 14)
  - Display the ideal age to the resultField JTextField in the format:
    Ideal age of spouse is 26
    - (Assuming 26 is the calculated value)
  - Add “action listening” to the “female” JButton

106 ☐ **BorderLayoutDemo.java (guis-miniquiz-2—Page 5)**

107 ☐ **BorderLayoutDemo.java (guis-miniquiz-2—Page 2)**

108 ☐ **Mini-Quiz No. 2 (Page 4)**

- Create an event handler class “NotTellingEventHandler” as follows:
  - Display the message to the resultField JTextField:
    No comment!!!
  - Add “action listening” to the “female” JButton

109 ☐ **BorderLayoutDemo.java (guis-miniquiz-2—Page 6)**

110 ☐ **BorderLayoutDemo.java (guis-miniquiz-2—Page 2)**

111 ☐ **The Ternary Operator (Page 1)**

- The ternary operator (?) returns one of two values depending upon the value of a *booleanExpression*
- It can be used as an alternative to Java if/else syntax, but it actually goes beyond that
  - It can even be used on the right hand side of Java assignment statements
- Format:
  booleanExpession ? *valueIfTrue* : *valueIfFalse*

112 ☐ **The Ternary Operator (Page 2)**

- Example:

```
boldInt = ( t1.getFont().isBold() ? Font.BOLD : Font.PLAIN );
```

- Equivalent:

```
if ( t1.getFont().isBold() )
{
   boldInt = Font.BOLD;
}
```

```
else
{
    boldInt = Font.PLAIN;
}
```

113