Lecture 3

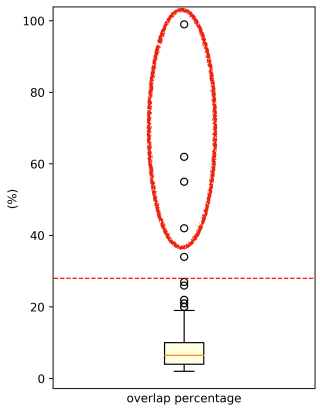
Outline

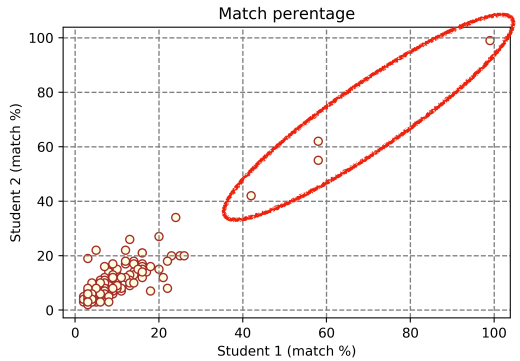
- Graphical User Interfaces (GUIs)
- Swing (I)

Reading:
S: Ch 17

Administrivia

- Plagiarism is a serious problem

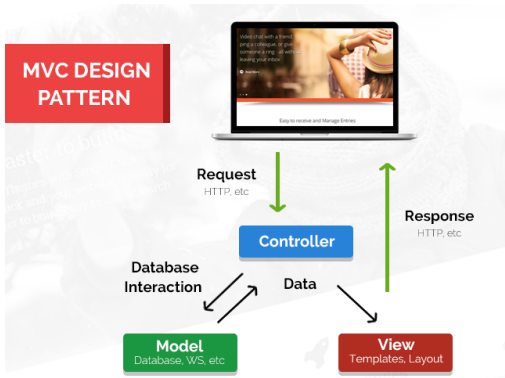




Graphical User Interfaces (GUIs)

Model-View-Controller (MVC)

- MVC - Model for GUI programming (Xerox PARC '78)
  - ✓ Separates GUI into 3 components
    - Model: application data
    - View: visual interface
    - Controller: user interaction



WS: Web service

MVC Model of GUI Design (1)

- Model
  - ✓ Should perform actual work
  - ✓ Should be independent of the GUI
    - But can provide access methods
- Controller
  - ✓ Lets user control what work the program is doing
  - ✓ Design of controller depends on model

MVC Model of GUI Design (2)

- View
  - ✓ Lets user see what the program results
  - ✓ Should not display what controller thinks
    - base display on model, not controller

Programming Models (1)

- Control flow-based Programming
  - ✓ Start at `main()`
  - ✓ Continue until end of program or `exit()`
- Event-driven Programming
  - ✓ Event: Action or condition occurring outside normal flow of control of program
    - e.g., mouse clicks, keyboard input, etc.
  - ✓ Unable to predict time & occurrence of event

Programming Models (2)

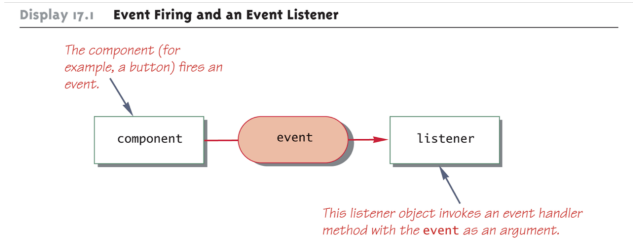
- ✓ signal-and-response approach
  - Start with `main()`
  - Define system elements and register event listeners
  - Await events (& perform associated computation)
- ✓ An event is an <u>object</u> that acts as a signal to another <u>object</u> know as a listener
- ✓ The sending of an event is called <u>firing the event</u>
- ✓ the events determine the execution order !!

Listeners

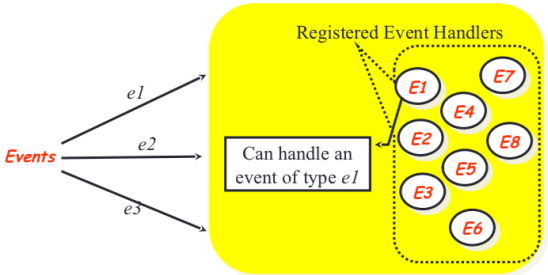
- A listener object performs some action in response to the event
  - ✓ A given component may have any number of listeners
  - ✓ Each listener may respond to a different kind of event
  - ✓ or multiple listeners might may respond to the same events
- Exception Objects: events !!!
  - ✓ The throwing of an exception is an example of firing an event
  - ✓ The listener for an exception object is the catch block that catches the event

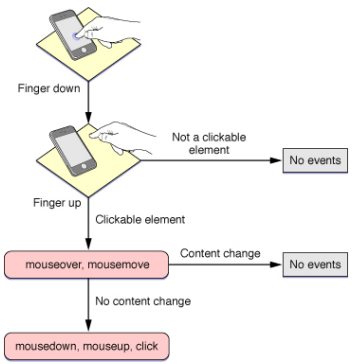
Event Handlers

- A listener object has methods that specify what will happen when events of various kinds are received by it
  - ✓ These methods are called event handlers
- The programmer using the listener object will define or redefine these event-handler methods



Event Handling in Action





GUIs in Java (1)

- AWT (Abstract Window Toolkit) (java.awt.*)
  - ✓ First graphical user interface toolkit for Java
  - ✓ Old GUI framework for Java (Java 1.1)
  - ✓ Reliance on native system libraries
    - ▸ Platform dependence
  - ✓ Responsible for input event mechanisms of the user interface
    - ▸ events that occur in the native window system are received by AWT
    - ▸ then forwarded to Java app. as AWT events

GUIs in Java (2)

- Java 2D
  - ✓ Graphics Library of Java
  - ✓ Introduced in JDK 1.2
  - ✓ Basics and advance drawing operation, image manipulation, and drawing
  - ✓ Handles Swing's Rendering operations

GUIs in Java (3)

- Swing (javax.swing.*)
  - ✓ GUI framework first introduced in JDK 1.2
  - ✓ Includes AWT features plus many enhancements
    - However, it does not completely replace the AWT
    - Some AWT classes are replaced by Swing classes, but other AWT classes are needed when using Swing
  - ✓ Pure Java components (no reliance on native code)
  - ✓ Pluggable look and feel architecture

Create a GUI in Java (1)

- (Step 1) Define a container to hold components
  - ✓ e.g., JFrame, Japplet
- (Step 2) Add GUI components to the container
  - ✓ JButton, JTextField, ...
  - ✓ Use layout manager to determine positions
- (Step3) Add actions to GUI
  - ✓ Add (register) event listeners to GUI components

Create a GUI in Java (2)

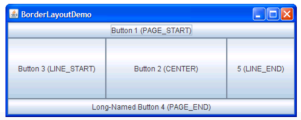
- ✓ Usually one event listener class per widget (the item user sees)
- ✓ Inner class usually utilized to implement listener
- ✓ At run time, java generates event object when events occur
  - Java then passes event object to event listener
- (Step4) Schedule the GUI processing in the EDT (Event-Dispatching Thread)
  - ✓ a background thread used in Java to process events from the AWT event queue

Schedule GUI Processing (1)

- What is a thread?
- Event Dispatching Thread (EDT) is a background thread to process GUI events
  - ✓ These events are mainly updates that
    - Cause components to redraw themselves
    - Represent input events

Schedule GUI Processing (1)

- Swing uses a single-threaded painting model !
  - ✓ EDT is the only valid thread for updating GUI components
  - ✓ <u>Avoid updating GUI components from other threads</u> (a source of bugs)
- Java Layout Manager
  - ✓ Arrange GUI components in container



  - ✓ Translate layout specifications into actual coordinates at runtime, depending on conditions

Beware of Long Computations

- Swing uses a single-threaded model
  - ✓ Long computations in the EDT freezes the GUI !!!!
- Usually you place the computation is a separate thread

<u>Swing — Layout</u>

A Simple Window

- A simple window can consist of an object of the `JFrame` class
  - ✓ includes a border and the usual three buttons
    - minimizing, changing the size of, and closing the window
  - ✓ found in the `javax.swing` package: `JFrame firstWindow = new JFrame();`
- A JFrame can have components added to it, such as buttons
  - ✓ These components can be programmed for action
    - `firstWindow.add(endButton);` `firstWindow.setVisible(true);`

Example

Display 17.2 A First Swing Demonstration Program

```
1  import javax.swing.JFrame;
2  import javax.swing.JButton;

3  public class FirstSwingDemo
4  {
5      public static final int WIDTH = 300;
6      public static final int HEIGHT = 200;

7      public static void main(String[] args)
8      {
9          JFrame firstWindow = new JFrame();
10         firstWindow.setSize(WIDTH, HEIGHT);
```

This program is not typical of the style we will use in Swing programs.

Display 17.2 A First Swing Demonstration Program

```
1  import java.awt.event.ActionListener;
2  import java.awt.event.ActionEvent;

3  public class EndingListener implements ActionListener
4  {
5      public void actionPerformed(ActionEvent e)
6      {
7          System.exit(0);
8      }
9  }
```

This is the file `EndingListener.java`.

(c

Display 17.2 A First Swing Demonstration Program

```
11         firstWindow.setDefaultCloseOperation(
12                                 JFrame.DO_NOTHING_ON_CLOSE);

13         JButton endButton = new JButton("Click to end program.");
14         EndingListener buttonEar = new EndingListener();
15         endButton.addActionListener(buttonEar);
16         firstWindow.add(endButton);

17         firstWindow.setVisible(true);
18     }
19 }
```

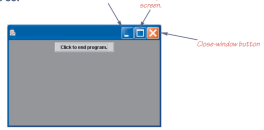
default is `JFrame.HIDE_ON_CLOSE` !!!

This is the file `FirstSwingDemo.java`.

(cc

Display 17.2 A First Swing Demonstration Program

RESULTING GUI



`JFrame` Class (1)

Display 17.3 Some Methods in the Class `JFrame`

The class `JFrame` is in the `javax.swing` package.

`public JFrame()`

Constructor that creates an object of the class `JFrame`.

`public JFrame(String title)`

Constructor that creates an object of the class `JFrame` with the title given as the argument.

(continued)

`JFrame` Class (2)

Display 17.3 Some Methods in the Class `JFrame`

`public void setDefaultCloseOperation(int operation)`

Sets the action that will happen by default when the user clicks the close-window button. The argument should be one of the following defined constants:

`JFrame.DO_NOTHING_ON_CLOSE`: Do nothing. The `JFrame` does nothing, but if there are any registered window listeners, they are invoked. (Window listeners are explained in Chapter 19.)

`JFrame.HIDE_ON_CLOSE`: Hide the frame after invoking any registered `WindowListener` objects.

`JFrame.DISPOSE_ON_CLOSE`: Hide and *dispose* the frame after invoking any registered window listeners. When a window is **disposed** it is eliminated but the program does not end. To end the program, you use the next constant as an argument to `setDefaultCloseOperation`.

`JFrame.EXIT_ON_CLOSE`: Exit the application using the `System exit` method. (Do not use this for frames in applets. Applets are discussed in Chapter 18.)

If no action is specified using the method `setDefaultCloseOperation`, then the default action taken is `JFrame.HIDE_ON_CLOSE`.

Throws an `IllegalArgumentException` if the argument is not one of the values listed above.[2]

Throws a `SecurityException` if the argument is `JFrame.EXIT_ON_CLOSE` and the Security Manager will not allow the caller to invoke `System.exit`. (You are not likely to encounter this case.)

`public void setSize(int width, int height)`

Sets the size of the calling frame so that it has the `width` and `height` specified. Pixels are the units of length used.

(continued)

JFrame Class (3)

Display 17.3 Some Methods in the Class JFrame

`public void setTitle(String title)`
Sets the title for this frame to the argument string.

`public void add(Component componentAdded)`
Adds a component to the JFrame.

`public void setLayout(LayoutManager manager)`
Sets the layout manager. Layout managers are discussed later in this chapter.

`public void setJMenuBar(JMenuBar menubar)`
Sets the menubar for the calling frame. (Menus and menu bars are discussed later in this chapter.)

`public void dispose()`
Eliminates the calling frame and all its subcomponents. Any memory they use is released for reuse. If there are items left (items other than the calling frame and its subcomponents), then this does not end the program. (The method `dispose` is discussed in Chapter 19.)

Ending a Swing Program

- GUI programs are often based on a kind of infinite loop
  - ✓ The windowing system normally stays on the screen until the user indicates that it should go away
  - ✓ If the user never asks the windowing system to go away, it will never go away
- In order to end a GUI program, `System.exit()` must be used when the user asks to end the program
  - ✓ It must be explicitly invoked, or included in some library code that is executed
    - Otherwise, a Swing program will not end after it has executed all the code in the program

A Better Version

Display 17.4 The Normal Way to Define a JFrame

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

public class FirstWindow extends JFrame
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;

    public FirstWindow()
    {
        super();
        setSize(WIDTH, HEIGHT);

        setTitle("First Window Class");

        setDefaultCloseOperation(
                                JFrame.DO_NOTHING_ON_CLOSE);

        JButton endButton = new JButton("Click to end program.");
        endButton.addActionListener(new EndingListener());
        add(endButton);
    }
}
```

This is the file `FirstWindow.java`.

The class `EndingListener` *is defined in Display 17.2.*

(continued)

Display 17.4 The Normal Way to Define a JFrame

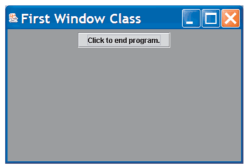
This is the file `DemoWindow.java`.

```
public class DemoWindow
{
    public static void main(String[] args)
    {
        FirstWindow w = new FirstWindow();
        w.setVisible(true);
    }
}
```

(continued)

Display 17.4 The Normal Way to Define a JFrame

RESULTING GUI



Color Class

- In Java, a color is an object of the class `Color`
  - ✓ The class `Color` is found in the `java.awt` package
  - ✓ There are <u>constants</u> in the `Color` class that represent a number of basic colors
- A `JFrame` can not be colored directly
  - ✓ Instead, a program must color something called the content pane of the JFrame
    - the content pane is the "inside" of a JFrame — coloring the content pane has the effect of coloring the inside of the JFrame
    - `getContentPane().setBackground(Color);`

JLabel and Color

Display 17.6 A JFrame with Color

```
1  import javax.swing.JFrame;
2  import javax.swing.JLabel;
3  import java.awt.Color;

4  public class ColoredWindow extends JFrame
5  {
6      public static final int WIDTH = 300;
7      public static final int HEIGHT = 200;

8      public ColoredWindow(Color theColor)
9      {
10         super("No Charge for Color");
11         setSize(WIDTH, HEIGHT);
12         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Display 17.6 A JFrame with Color

```
13         getContentPane().setBackground(theColor);

14         JLabel aLabel = new JLabel("Close-window button works.");
15         add(aLabel);
16     }

17     public ColoredWindow()
18     {
19         this(Color.PINK);
20     }
21 }
```

This is an invocation of the other constructor.

This is the file ColoredWindow.java.

(continued)

Display 17.6 A JFrame with Color

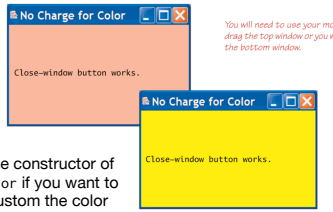
```
1  import java.awt.Color;

2  public class DemoColoredWindow
3  {
4      public static void main(String[] args)
5      {
6          ColoredWindow w1 = new ColoredWindow();
7          w1.setVisible(true);

8          ColoredWindow w2 = new ColoredWindow(Color.YELLOW);
9          w2.setVisible(true);
10     }
11 }
```

This is the file ColoredWindow.java.

(continue

Display 17.6 A JFrame with Color

RESULTING GUI



You will need to use your mouse to drag the top window or you will not see the bottom window.

Use constructor of Color if you want to custom the color

Layout

- ✓ `setLayout(new BorderLayout());`
- ✓ `setLayout(new FlowLayout());`
- ✓ `setLayout(new GridLayout(rows, columns));`

Display 17.10 Some Layout Managers

LAYOUT MANAGER	DESCRIPTION
These layout manager classes are in the `java.awt` package.	
`FlowLayout`	Displays components from left to right in the order in which they are added to the container.
`BorderLayout`	Displays the components in five areas: north, south, east, west, and center. You specify the area a component goes into in a second argument of the add method.
`GridLayout`	Lays out components in a grid, with each component stretched to fill its box in the grid.

BorderLayout

Display 17.7 The BorderLayout Manager

```
1  import javax.swing.JFrame;
2  import javax.swing.JLabel;
3  import java.awt.BorderLayout;

4  public class BorderLayoutJFrame extends JFrame
5  {
6      public static final int WIDTH = 500;
7      public static final int HEIGHT = 400;

8      public BorderLayoutJFrame()
9      {
10         super("BorderLayout Demonstration");
11         setSize(WIDTH, HEIGHT);
12         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Display 17.7 The BorderLayout Manager

```
13         setLayout(new BorderLayout());

14         JLabel label1 = new JLabel("First label");
15         add(label1, BorderLayout.NORTH);

16         JLabel label2 = new JLabel("Second label");
17         add(label2, BorderLayout.SOUTH);

18         JLabel label3 = new JLabel("Third label");
19         add(label3, BorderLayout.CENTER);
20     }
21 }
```

This is the file BorderLayoutJFrame.java.

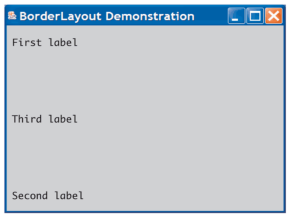
Display 17.7 The BorderLayout Manager

This is the file BorderLayoutDemo.java.

```
1  public class BorderLayoutDemo
2  {
3      public static void main(String[] args)
4      {
5          BorderLayoutJFrame gui = new BorderLayoutJFrame();
6          gui.setVisible(true);
7      }
8  }
```

Display 17.7 The BorderLayout Manager

RESULTING GUI



Grid Layout

Display 17.9 The GridLayout Manager

```
1  import javax.swing.JFrame;
2  import javax.swing.JLabel;
3  import java.awt.GridLayout;

4  public class GridLayoutJFrame extends JFrame
5  {
6      public static final int WIDTH = 500;
7      public static final int HEIGHT = 400;

8      public static void main(String[] args)
9      {
10         GridLayoutJFrame gui = new GridLayoutJFrame(2, 3);
11         gui.setVisible(true);
12     }
```

Display 17.9 The GridLayout Manager

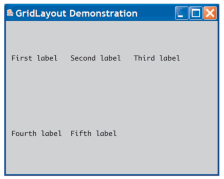
```
13     public GridLayoutJFrame(int rows, int columns )
14     {
15         super();
16         setSize(WIDTH, HEIGHT);
17         setTitle("GridLayout Demonstration");
18         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19         setLayout(new GridLayout(rows, columns ));

20         JLabel label1 = new JLabel("First label");
21         add(label1);
```

Display 17.9 The GridLayout Manager

```
22         JLabel label2 = new JLabel("Second label");
23         add(label2);

24         JLabel label3 = new JLabel("Third label");
25         add(label3);

26         JLabel label4 = new JLabel("Fourth label");
27         add(label4);

28         JLabel label5 = new JLabel("Fifth label");
29         add(label5);
30     }
31 }
```

Display 17.9 The GridLayout Manager

RESULTING GUI



Panels (1)

- A GUI is often organized in a hierarchical fashion
  - ✓ with containers called panels inside other containers
- A panel is an <u>object</u> of the `JPanel` class that serves as a simple container
  - ✓ used to group smaller objects into a larger component (the panel)
  - ✓ One of the main functions of a `JPanel` object is to subdivide a `JFrame` or other container

Panels (2)

Display 17.11 Using Panels

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.FlowLayout;
import java.awt.Color;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class PanelDemo extends JFrame implements ActionListener
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;
```

In addition to being the GUI class, the class `PanelDemo` *is the action listener class. An object of the class* `PanelDemo` *is the action listener for the buttons in that object.*

(continued)

Panels (3)

Display 17.11 Using Panels

```
    private JPanel redPanel;
    private JPanel whitePanel;
    private JPanel bluePanel;

    public static void main(String[] args)
    {
        PanelDemo gui = new PanelDemo();
        gui.setVisible(true);
    }

    public PanelDemo()
    {
        super("Panel Demonstration");
        setSize(WIDTH, HEIGHT);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
```

We made these instance variables because we want to refer to them in both the constructor and the method `actionPerformed`.

(continued)

Panels (4)

Display 17.11 Using Panels

```
        JPanel biggerPanel = new JPanel();
        biggerPanel.setLayout(new GridLayout(1, 3));

        redPanel = new JPanel();
        redPanel.setBackground(Color.LIGHT_GRAY);
        biggerPanel.add(redPanel);

        whitePanel = new JPanel();
        whitePanel.setBackground(Color.LIGHT_GRAY);
        biggerPanel.add(whitePanel);
```

Display 17.11 Using Panels

```
        JPanel bluePanel = new JPanel();
        bluePanel.setBackground(Color.LIGHT_GRAY);
        biggerPanel.add(bluePanel);

        add(biggerPanel, BorderLayout.CENTER);

        JPanel buttonPanel = new JPanel();
        buttonPanel.setBackground(Color.LIGHT_GRAY);
        buttonPanel.setLayout(new FlowLayout());

        JButton redButton = new JButton("Red");
        redButton.setBackground(Color.RED);
        redButton.addActionListener(this);
        buttonPanel.add(redButton);
```

An object of the class `PanelDemo` *is the action listener for the buttons in that object.*

(continued)

Display 17.11 Using Panels

```
        JButton whiteButton = new JButton("White");
        whiteButton.setBackground(Color.WHITE);
        whiteButton.addActionListener(this);
        buttonPanel.add(whiteButton);

        JButton blueButton = new JButton("Blue");
        blueButton.setBackground(Color.BLUE);
        blueButton.addActionListener(this);
        buttonPanel.add(blueButton);

        add(buttonPanel, BorderLayout.SOUTH);
    }
```

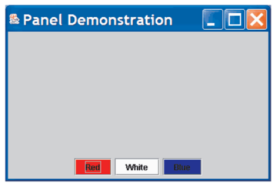
Display 17.11 Using Panels

```
    public void actionPerformed(ActionEvent e)
    {
        String buttonString = e.getActionCommand();

        if (buttonString.equals("Red"))
            redPanel.setBackground(Color.RED);
        else if (buttonString.equals("White"))
            whitePanel.setBackground(Color.WHITE);
        else if (buttonString.equals("Blue"))
            bluePanel.setBackground(Color.BLUE);
        else
            System.out.println("Unexpected error.");
    }
}
```

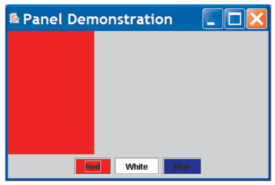
Panels (5)

Display 17.11 **Using Panels**

RESULTING GUI (When first run)

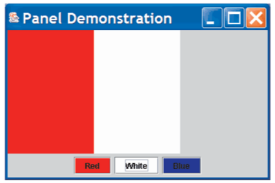


RESULTING GUI (After clicking Red button)

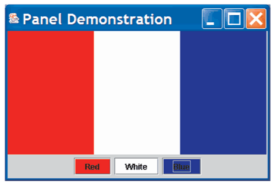


Display 17.11 **Using Panels**

RESULTING GUI (After clicking White button)



RESULTING GUI (After clicking Blue button)



The `Container` Class

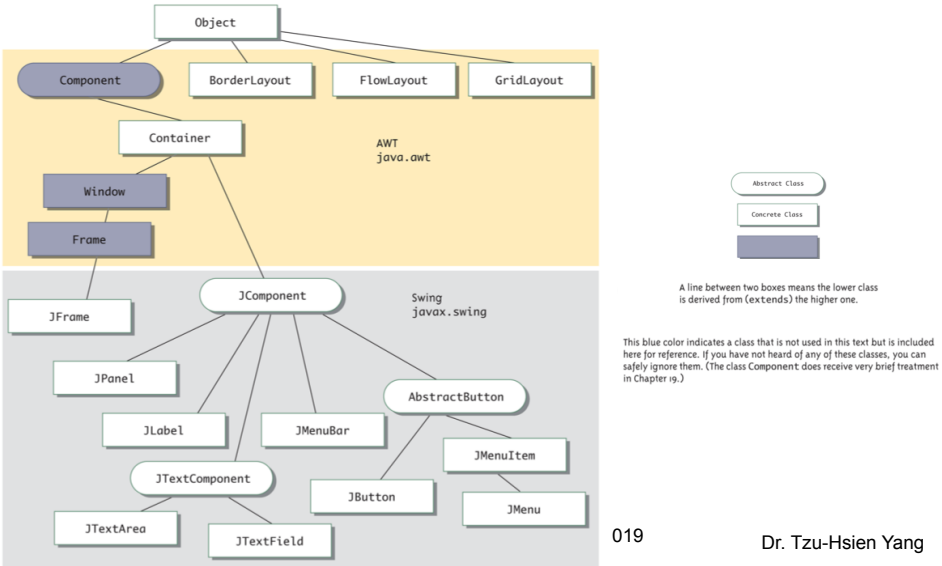
- Any class that is a descendent class of the class `Container` is considered to be a `container` class
  - ✓ found in the `java.awt` package, not in the `Swing` library
- Any object that belongs to a class derived from the `Container` class (or its descendants) can have components added to it
  - ✓ The classes `JFrame` and `JPanel` are descendent classes of the class Container
    - they and any of their descendants can serve as a container

The `JComponent` Class

- Any descendent class of the class `JComponent` is called a component class
  - ✓ `JComponent` objects or component can be added to any `container` class object
  - ✓ derived from the class `Container` — it can be added to another `JComponent`
- GUI built using Swing will be made up of 3 kinds of objects
  - ✓ The container, probably a panel or window-like object
  - ✓ The components added to the container (such as labels, buttons, and panels)
  - ✓ A layout manager to position the components inside the container

Hierarchy of Swing and AWT Classes

Display 17.12 **Hierarchy of Swing and AWT Classes**



Swing Component — Menu

JMenu (1)

Display 17.14 A GUI with a Menu

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.GridLayout;
import java.awt.Color;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
```

Display 17.14 A GUI with a Menu

```
public class MenuDemo extends JFrame implements ActionListener
{
    public static final int WIDTH = 300;
    public static final int HEIGHT = 200;

    private JPanel redPanel;
    private JPanel whitePanel;
    private JPanel bluePanel;

    public static void main(String[] args)
    {
        MenuDemo gui = new MenuDemo();
        gui.setVisible(true);
    }
```

Display 17.14 A GUI with a Menu

```
    public MenuDemo()
    {
        super("Menu Demonstration");
        setSize(WIDTH, HEIGHT);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(1, 3));

        redPanel = new JPanel();
        redPanel.setBackground(Color.LIGHT_GRAY);
        add(redPanel);

        whitePanel = new JPanel();
        whitePanel.setBackground(Color.LIGHT_GRAY);
        add(whitePanel);
```


JMenu (2)

Display 17.14 A GUI with a Menu

```
        bluePanel = new JPanel();
        bluePanel.setBackground(Color.LIGHT_GRAY);
        add(bluePanel);

        JMenu colorMenu = new JMenu("Add Colors");

        JMenuItem redChoice = new JMenuItem("Red");
        redChoice.addActionListener(this);
        colorMenu.add(redChoice);

        JMenuItem whiteChoice = new JMenuItem("White");
        whiteChoice.addActionListener(this);
        colorMenu.add(whiteChoice);

        JMenuItem blueChoice = new JMenuItem("Blue");
        blueChoice.addActionListener(this);
        colorMenu.add(blueChoice);

        JMenuBar bar = new JMenuBar();
        bar.add(colorMenu);
        setJMenuBar(bar);
    }
```

The definition of `actionPerformed` *is identical to the definition given in Display 17.11 for a similar GUI using buttons instead of menu items.*

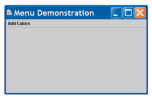
JMenu (3)

Display 17.14 A GUI with a Menu

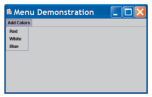
```
    public void actionPerformed(ActionEvent e)
    {
        String buttonString = e.getActionCommand();

        if (buttonString.equals("Red"))
            redPanel.setBackground(Color.RED);
        else if (buttonString.equals("White"))
            whitePanel.setBackground(Color.WHITE);
        else if (buttonString.equals("Blue"))
            bluePanel.setBackground(Color.BLUE);
        else
            System.out.println("Unexpected error.");
    }
}
```

Display 17.14 A GUI with a Menu

Resulting GUI

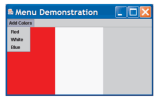


Resulting GUI (after clicking Add Colors in the menu bar)



Display 17.14 A GUI with a Menu

Resulting GUI (after choosing Red and White on the menu)



Resulting GUI (after choosing all the colors on the menu)

Nested Menus

- The class `JMenu` is a descendent of the `JMenuItem` class
  - ✓ Every `JMenu` can be a menu item in another menu
    - Therefore, menus can be nested
  - ✓ Menus can be added to other menus in the same way as menu items

Note

- The classes `JButton` and `JMenuItem` are derived classes of the abstract class named `AbstractButton`
- Objects of the `Dimension` class are used with buttons, menu items, and other objects to <u>specify a size</u>
  - ✓ The Dimension class is in the package `java.awt`
    - `Dimension(int width, int height)`
    - width and height parameters are in pixels

The `setActionCommand` Method

- When a user clicks a button or menu item, an event is fired that normally goes to one or more action listeners
  - ✓ The action event becomes an argument to an `actionPerformed` method
- This action event includes a `String` instance variable called the action command for the button or menu item
  - ✓ The default value is the string written on the button or the menu item
  - ✓ can be retrieved with the `getActionCommand` method
- The `setActionCommand` method can be used to change the action command for a component — useful for name conflicts

Methods in the Class `AbstractButton` (1)

Display 17.15 Some Methods in the Class `AbstractButton`

The abstract class `AbstractButton` is in the `javax.swing` package.
All of these methods are inherited by both of the classes `JButton` and `JMenuItem`.

```
public void setBackground(Color theColor)
```

Sets the background color of this component.

```
public void addActionListener(ActionListener listener)
```

Adds an `ActionListener`.

```
public void removeActionListener(ActionListener listener)
```

Removes an `ActionListener`.

```
public void setActionCommand(String actionCommand)
```

Sets the action command.

(continued)

Methods in the Class AbstractButton (2)

Display 17.15 Some Methods in the Class AbstractButton

```
public String getActionCommand()
```

Returns the action command for this component.

```
public void setText(String text)
```

Makes text the only text on this component.

```
public String getText()
```

Returns the text written on the component, such as the text on a button or the string for a menu item.

```
public void setPreferredSize(Dimension preferredSize)
```

Sets the preferred size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to use the preferred size. The following special case will work for most simple situations. The `int` values give the width and height in pixels.

```
public void setPreferredSize(
                    new Dimension(int width, int height))
```

(continued)

Methods in the Class AbstractButton (3)

Display 17.15 Some Methods in the Class AbstractButton

```
public void setMaximumSize(Dimension maximumSize)
```

Sets the maximum size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to respect this maximum size. The following special case will work for most simple situations. The `int` values give the width and height in pixels.

```
public void setMaximumSize(
                    new Dimension(int width, int height))
```

```
public void setMinimumSize(Dimension minimumSize)
```

Sets the minimum size of the button or label. Note that this is only a suggestion to the layout manager. The layout manager is not required to respect this minimum size.
Although we do not discuss the `Dimension` class, the following special case is intuitively clear and will work for most simple situations. The `int` values give the width and height in pixels.

```
public void setMinimumSize(
                    new Dimension(int width, int height))
```


Swing Component — TextField

Text Field (1)

Display 17.17 A Text Field

```
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JButton;
import java.awt.GridLayout;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Color;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
```

Display 17.17 A Text Field

```
public class TextFieldDemo extends JFrame
                        implements ActionListener
{
    public static final int WIDTH = 400;
    public static final int HEIGHT = 200;
    public static final int NUMBER_OF_CHAR = 30;

    private JTextField name;


    public static void main(String[] args)
    {
        TextFieldDemo gui = new TextFieldDemo();
        gui.setVisible(true);
    }
```

Display 17.17 A Text Field

```
    public TextFieldDemo()
    {
        super("Text Field Demo");
        setSize(WIDTH, HEIGHT);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(2, 1));

        JPanel namePanel = new JPanel();
        namePanel.setLayout(new BorderLayout());
        namePanel.setBackground(Color.WHITE);

        name = new JTextField(NUMBER_OF_CHAR);
```

Display 17.17 A Text Field

```
        namePanel.add(name, BorderLayout.SOUTH);
        JLabel nameLabel = new JLabel("Enter your name here:");
        namePanel.add(nameLabel, BorderLayout.CENTER);

        add(namePanel);

        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new FlowLayout());
        buttonPanel.setBackground(Color.PINK);
        JButton actionButton = new JButton("Click me");
        actionButton.addActionListener(this);
        buttonPanel.add(actionButton);

        JButton clearButton = new JButton("Clear");
        clearButton.addActionListener(this);
        buttonPanel.add(clearButton);
```

(continued)

Text Field (2)

Display 17.17 A Text Field

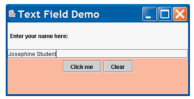
```
47          add(buttonPanel);
48      }

49      public void actionPerformed(ActionEvent e)
50      {
51          String actionCommand = e.getActionCommand();

52          if (actionCommand.equals("Click me"))
53              name.setText("Hello " + name.getText());
54          else if (actionCommand.equals("Clear"))
55              name.setText("");
56          else
57              name.setText("Unexpected error.");
58      }

59  }
```

This sets the text field equal to the empty string, which makes it blank.

(continued)

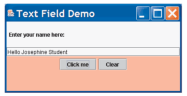
Display 17.17 A Text Field

RESULTING GUI (When program is started and a name entered)



Display 17.17 A Text Field

RESULTING GUI (After clicking the "Click me" button)



Text Areas (1)

- A text area is the same as a text field, except that it allows <u>multiple lines</u>
  - ✓ an object of the class `JTextArea`
  - ✓ `JTextArea theText = new JTextArea(5,20);`
    - 5: number of lines
    - 20: the number of characters per line guaranteed to be visible
  - ✓ `JTextArea theText = new JTextArea("Enter\ntext here.", 5, 20);`

Text Areas (2)

- The line-wrapping policy for a JTextArea can be set using the method `setLineWrap`
  - ✓ `theText.setLineWrap(true);`
    - `true`: any additional characters at the end of a line will appear on the following line of the text area
    - `false`: the extra characters will remain on the same line and not be visible
- A `JTextField` or `JTextArea` can be set so that it can not be changed by the user
  - ✓ `theText.setEditable(false);`

JScrollPane (1)

- horizontal and vertical scrollbar policies
  - ✓ modified by `setHorizontalScrollBarPolicy` and `setVerticalScrollBarPolicy`
  - ✓ to indicate that a scrollbar should always appear: `JScrollPane.VERTICAL_SCROLLBAR_ALWAYS`, `JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS`
  - ✓ to indicate that a scrollbar should appear only if necessary (the defaults): `JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED` / `JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED`
- If policy is set to `HORIZONTAL_SCROLLBAR_NEVER`
  - ✓ a `JTextArea` attached to the JScrollPane will automatically wrap lines

JScrollPane (2)

- `Box` is a subclass of `Container` that uses a BoxLayout to arrange the GUI components horizontally or vertically

Text Areas Example (1)


```java
// Fig. 12.47: TextAreaFrame.java
// Copying selected text from one JText area to another.
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.Box;
import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.JButton;
import javax.swing.JScrollPane;

public class TextAreaFrame extends JFrame
{
   private final JTextArea textArea1; // displays demo string
   private final JTextArea textArea2; // highlighted text is copied here
   private final JButton copyJButton; // initiates copying of text

   // no-argument constructor
   public TextAreaFrame()
   {
      super("TextArea Demo");
      Box box = Box.createHorizontalBox(); // create box
      String demo = "This is a demo string to\n" +
         "illustrate copying text\nfrom one textarea to \n" +
         "another textarea using an\nexternal event\n";
```

Box is a subclass of `Container` that uses a `BoxLayout` to arrange the GUI components horizontally or vertically

Fig. 12.47 | Copying selected text from one `JTextArea` to another. (Part 1 of 2.)

Text Areas Example (2)


```java

      textArea1 = new JTextArea(demo, 10, 15);
      box.add(new JScrollPane(textArea1)); // add scrollpane

      copyJButton = new JButton("Copy >>>"); // create copy button
      box.add(copyJButton); // add copy button to box
      copyJButton.addActionListener(
         new ActionListener() // anonymous inner class
         {
            // set text in textArea2 to selected text from textArea1
            @Override
            public void actionPerformed(ActionEvent event)
            {
               textArea2.setText(textArea1.getSelectedText());
            }
         }
      );

      textArea2 = new JTextArea(10, 15);
      textArea2.setEditable(false);
      box.add(new JScrollPane(textArea2)); // add scrollpane

      add(box); // add box to frame
   }
} // end class TextAreaFrame
```

Fig. 12.47 | Copying selected text from one `JTextArea` to another. (Part 2 of 2.)

Text Areas Example (3)


```java
// Fig. 12.48: TextAreaDemo.java
// Testing TextAreaFrame.
import javax.swing.JFrame;

public class TextAreaDemo
{
   public static void main(String[] args)
   {
      TextAreaFrame textAreaFrame = new TextAreaFrame();
      textAreaFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      textAreaFrame.setSize(425, 200);
      textAreaFrame.setVisible(true);
   }
} // end class TextAreaDemo
```

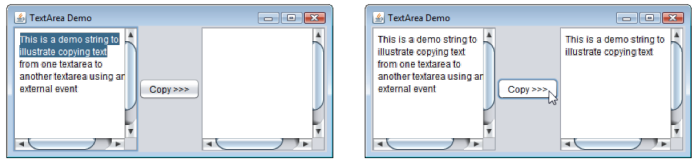


Fig. 12.48 | Testing `TextAreaFrame`.

The Class `JTextComponent`

- Both `JTextField` and `JTextArea` are derived classes of the abstract class `JTextComponent`

Display 17.18 Some Methods in the Class `JTextComponent`

All these methods are inherited by the classes `JTextField` and `JTextArea`.
The abstract class `JTextComponent` is in the package `javax.swing.text`. The classes `JTextField` and `JTextArea` are in the package `javax.swing`.

```
public String getText()
```
Returns the text that is displayed by this text component.

```
public boolean isEditable()
```
Returns `true` if the user can write in this text component. Returns `false` if the user is not allowed to write in this text component.

```
public void setBackground(Color theColor)
```
Sets the background color of this text component.

```
public void setEditable(boolean argument)
```
If `argument` is `true`, then the user is allowed to write in the text component. If `argument` is `false`, then the user is not allowed to write in the text component.

```
public void setText(String text)
```
Sets the text that is displayed by this text component to be the specified `text`.

Labeling a Text Field

- In order to label one or more text fields
  - ✓ Use an object of the class `JLabel`
  - ✓ Place the text field(s) and label(s) in a `JPanel`
  - ✓ Treat the `JPanel` as a single component
- When attempting to input numbers from any Swing GUI, input text must be converted to numbers

of Characters Per Line

- The number of characters per line for a `JTextField` or `JTextArea` object is the number of em spaces
  - ✓ An em space is the space needed to hold one uppercase letter **M**
    - The letter **M** is the widest letter in the alphabet
  - ✓ A line specified to hold 20 **M**'s will almost always be able to hold more than 20 characters

End of Lecture

References

- Lecture slides from CMSC132, University of Maryland

Review Questions (1)

- What is “GUI”? Please also specify the MVC architecture for realizing GUI implementation.
- What is event-driven programming?
- Define the following: (1) event (2) event listener (3) event handler.
- Define the following: (1) AWT (2) Java2D (3) Swing (4) EDT.
- Define the following layout terms in Java: (1) FlowLayout (2) BorderLayout (3) GridLayout

Review Questions (2)

- Describe the 3 basic objects that form a Java Swing GUI.
- Describe the following: (1) JFrame (2) JPanel (3) JComponent (4) Container
- Describe the following: (1) JMenu (2) Jbutton (3) JLabel (4) JMenuItem (5) JTextField (6) JTextArea
- Describe the concept of “em spaces”.