LEWIS UNIVERSITY
A Catholic and Lasallian University

Object-Oriented Programming

Course Number: CPSC-24500

Week: 3

Instructor: Eric Pogue

Object-Oriented Programming

Session: Week 3 Session 1
Instructor: Eric Pogue

LEWIS UNIVERSITY
A Catholic and Lasallian University

Agenda:

1. Review this week's Assignment
2. Introduce the week's Learning Objectives
3. Review Learning Objectives will be covered in this session
4. Present Topics
5. Closing Comments and Next Steps

While everyone is getting logged in an set up for our virtual discussion, I wanted to thank those of you who have posted their background and/or responded to other people's discussion board posting. I have enjoyed reading through your comments.

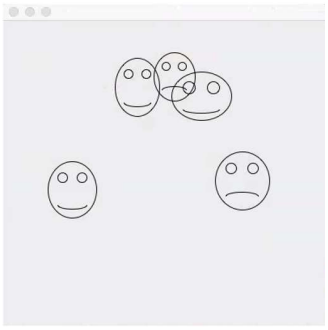
I am going to stat recording so that other can watch later; however, if I think that is impacting our ability to have a good discussion, I will stop the recording and just send out notes later.

FaceDraw Assignment & Learning Objectives

FaceDraw and Learning Objectives: FaceDraw will be a challenging assignment this week. Our Learning Objects, Discussion, Lecture, and Examples will all be focused on helping you complete the assignment and learn the key concepts through delivering a concrete example.

In FaceDraw we will:

1. Create an application in Java that draws random faces on a window
2. Generate a random number between 3 & 10 and draw that many faces
3. Randomly generate reasonable and visually appealing face
4. Position itself to a reasonable size and location
5. Draw all faces so they are entirely within the window.
6. Draw each face with two eyes and a mouth.
7. The mouth should be randomly smiling, frowning, or in-between



I intentionally list our assignment first and our learning objectives second. This week our intent is to support our assignment through our learning objectives, discussion, lecture, and examples… and not the other way around.

It is important to understand the concepts, practices, and principles; however, it is essential that we can utilize them to deliver solutions.
The details of FaceDraw are provided in this week's assignment.

Learning Objectives – Week 3

1. Explain the difference between lightweight and heavyweight components
2. Identify lightweight and heavyweight components
3. Describe how lightweight and heavyweight components render themselves
4. Identify layout managers for heavyweight and lightweight components
5. Explain what a layout manager does and identify and describe three of them
6. Explain how to achieve true Model-View-Controller architecture by removing any reference from the view to the model and from the model to the view
7. Distinguish between extending a class and implementing an interface
8. Explain multiple ways to implement an event handler for a particular object and event (anonymous inner classes vs. named classes vs. having the frame itself serve as the handler)
9. Create an event handler that implements the ActionListener interface to respond to the user clicking on a button
10. Use paint and paintComponent's Graphics object to draw a variety of shapes

Learning Objectives – Week 3 / Session 1

1. Explain the difference between lightweight and heavyweight components
2. Identify lightweight and heavyweight components
3. Describe how lightweight and heavyweight components render themselves
4. Identify layout managers for heavyweight and lightweight components
5. Explain what a layout manager does and identify and describe three of them
6. Explain how to achieve true Model-View-Controller architecture by removing any reference from the view to the model and from the model to the view
7. Distinguish between extending a class and implementing an interface
8. Explain multiple ways to implement an event handler for a particular object and event (anonymous inner classes vs. named classes vs. having the frame itself serve as the handler)
9. Create an event handler that implements the ActionListener interface to respond to the user clicking on a button
10. Use paint and paintComponent's Graphics object to draw a variety of shapes

Lightweight and Heavyweight User Interface Components

UI Components: There are two kinds of graphics components in the Java programming language: **heavyweight** and **lightweight**.

A **heavyweight component** is associated with its own native screen resource. **AWT** Components from the java.awt package, such as Button and Label, are heavyweight components.

Lightweight components such as **Swing components** depend less on the target platform and use less native GUI resource. They also look less native.

```
//*********************************************************
// ShapesView
class ShapeFrame extends JFrame implements ActionListener {
    public ShapeFrame() {
        // Get the JFrame's container... JFrame is a
        //    heavyweigt component.
        Container myContainer = getContentPane();
        myContainer.setLayout(new BorderLayout());

        // Add an empty panel to the center.
        myContainer.add(new JPanel(), BorderLayout.CENTER);

        // Create a panel for Buttons
        JPanel myButtonPanel = new JPanel();
        myButtonPanel.setLayout(new FlowLayout());

        // Add two buttons and implment action listeners
        //     with anonymous inner classes.
        JButton ClickMe = new JButton("ClickMe!?!");
        ClickMe.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Option 1: Inner class");
            }
        });
```

These notes are from the Oracle site. There is also a nice article about on the Oracle site about mixing heavyweight and lightweight components at:
http://www.oracle.com/technetwork/articles/java/mixing-components-433992.html

Java platform graphics have been the Achilles heel of the Java platform since the mid-1990s. Programmers used to joke that AWT stood for Awful Windows Toolkit because the resulting applications were so slow to function and unpleasant to look at. AWT actually stands for Abstract Window Toolkit.

IBM's Standard Widget Toolkit (SWT) toolkit is an alternative to AWT and Swing that has taken root in the Eclipse Integrated Development Environment (IDE) world.
https://en.wikipedia.org/wiki/Standard_Widget_Toolkit

AWT and Swing are excellent learning tools; however, nearly all commercial user interface development has moved on to other tools and libraries. Native user interfaces or Web type interfaces using Java Script is seem to be the direction. Note: Java and Java Script have **NOTHING** to do with each other! It was completely a marketing ploy to give Java Script the Java name.

Java Applets are Java applications that run in a browser. They never really did work… The "write once and run everywhere" philosophy never did play out very well in the industry when it comes to leading edge client applications. Browser based applications seem to have filled that gap while native applications have filled the "rich client" areas. I suspect that trend will continue.

java.awt.*: Graphics, Color, Font
Java.swing: JFrame, JPanel, JButton

Heavyweight Components

- Paint themselves using their “paint(Graphic g)” method.
- Provide methods associate with “Graphic g” such as: drawOval(), fillOval(), drawRect(), fillRect(), and drawstring()
- Own a Content Pane, which is a type of Container
- Implement “getContentPane()” method to access their Content Pane
- Can implement a **Layout Manager** via their Container

The Graphics class is defined in package java.awt.

A window in Java is called a JFrame. JFrames are heavyweight components: graphical components that communicate directly with the operating system and show up directly on the desktop. Heavyweight components contain what are called lightweight components. Lightweight components include panels and buttons. In Java, panels are implemented as JPanels, and buttons are implemented as JButtons.

Lightweight Components

- Paint themselves using their "paintComponent(Graphics g)" method
- Provide methods associate with "Graphic g" such as: drawOval(), fillOval(), drawRect(), fillRect(), and drawstring()
- Are a Container… so no need to call a method to get a Container
- Can implement a **Layout Manager** via their Container

Specifically, a JFrame, a heavyweight component …

1. Can setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) to make the program vanish from memory (i.e. free resources) when closed.
2. Can set size and location using setBounds(x,y,width,height)
3. Can show itself using setVisible(true) and hide itself using setVisible(false)
4. Can give you a reference to its Container through its getContentPane() function. Once you have access to its Container object, you can set the layout of that container so that you can arrange lightweight components in it.

Popular Lightweight Swing UI components include:
Jbutton
JTextArea
JPanel

Layout Managers

Layout Managers arrange controls on the screen so they are visually appealing. The allow developers to be able to focus less on layout details. Both AWT and Swing provide general purpose Layout Managers including:

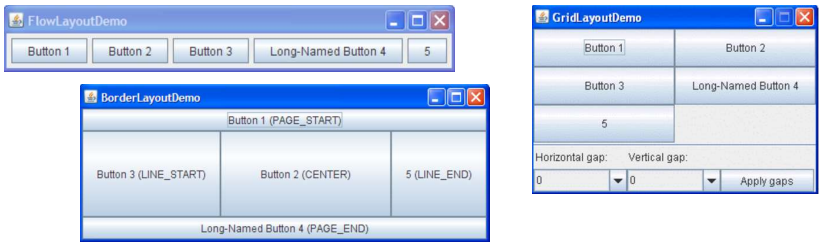
- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout
- GroupLayout
- SpringLayout

This is different from environments where developers (or designers) layout screens pixel by pixel (like Visual Basic for example). With modern UIs that need to fit in all types of sizes and orientations, layout managers are a necessity.

Layout Managers (continued)

Layout Managers arrange controls on the screen so they are visually appealing. We will be focusing on three very common Layout Managers:

- FlowLayout: Arranges components left to right, top to bottom. When a FlowLayout runs out of room horizontally on a row, it places the next component as far left as it can go on the row below.
- BorderLayout: Arranges components in NORTH, SOUTH, EAST, WEST, and CENTER sections.
- GridLayout: Arranges components in an Excel-like table of rows and columns.



This is different from environments where developers (or designers) layout screens pixel by pixel (like Visual Basic for example). With modern UIs that need to fit in all types of sizes and orientations, layout managers are a necessity.

Several AWT and Swing classes provide layout managers for general use:

- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout
- GroupLayout
- SpringLayout

Model-View-Controller

Model-View-Controller (MVC): This week are focus will be in creating our first Graphical view. Specifically we will:

1. Implement our first graphical view using Java AWT and Swing graphics
2. Demonstrate a view that you will be able to build off of for our assignment
3. Implement a more sophisticated view to draw smiling and frowning faces as our assignment for the week
4. Discuss why our command line interface is also a view
5. Discuss how we "render" our data to the command line using the toString() method
6. Discuss pros and cons of "rendering" our data to our graphical view via a "toGraphic()" method

We will be focuses on the View this week, and will keep our references to Models and Controllers to a minimum.

MVC is an important pattern, will be a primary focus of this course, and will be an important pattern for you to master in your career.
Segregation of our Model (data) from our View (user interface) is necessary to effectively develop, enhance, and maintain modern software.

An example would be a system that manages student data. We would want to segregate the Model (data) from the View (UI) for several reasons including that there will likely be many different Views that access the same data including:
student view
faculty view
administrator view,
Web student view,
mobile student view, etc.

Evolution of UI and Data segregation
- Document-View (View was responsible for View-Controller functionality)
- Model-View-Controller
- Model–View–Viewmodel

Interface Implementation vs. Class Extension

Differences include:

1. Interfaces are implicitly abstract and cannot have implementations
2. An interface specifies one or more abstract methods that a class that implements the interface must implement
3. Abstract classes can have both abstract methods and concrete methods
4. A class can descend (extends) only one Class but can implement many interfaces

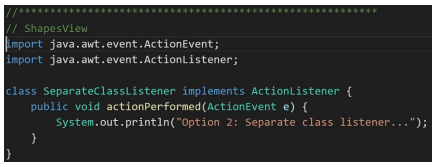
Note: Interfaces are Java's way of avoiding the significant complexities of multiple inheritance while providing most of the benefits.

A little history:
Some languages including C++ implemented multiple inheritance in order to allow Classes to inherit from multiple parent classes. That solution has pros and cons beyond the scope of this course.
Other languages including Java only support single inheritance, but also implemented Interfaces in order to allow class to act like multiple things. An example that we will be discussing this week is our OvalFrame that we would like to be a JFrame but may also want it to act like an ActionHandler to respond to buttons being pushed.

Interface Implementation

Interface implementation is basically agreeing to a contract to implement a given set of methods.

The **ActionalListener** Interface is a good example of an interface that a class must implement in order to receive notification that a button has been pushed by the user.



```
//*******************************************************
// ShapesView
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class SeparateClassListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Option 2: Separate class listener...");
    }
}
```

Class Extension

Class Extension is **Inheritance**. Enough said?

In Java a class can extend from only one parent class, but can implement multiple interfaces.

The code to the right demonstrates ShapeFrame:

- Extending from JFrame so that it can “be a” JFrame and hold lightweight UI components
- Implementing ActionionListener so that it can “act like” a ActionListener and receive notification that a button has been pressed.

```
//*********************************************************
// ShapesView
class ShapeFrame extends JFrame implements ActionListener {
    public ShapeFrame() {
        // Get the JFrame's container... JFrame is a
        //    heavyweigt component.
        Container myContainer = getContentPane();
        myContainer.setLayout(new BorderLayout());

        // Add an empty panel to the center.
        myContainer.add(new JPanel(), BorderLayout.CENTER);

        // Create a panel for Buttons
        JPanel myButtonPanel = new JPanel();
        myButtonPanel.setLayout(new FlowLayout());

        // Add two buttons and implment action listeners
        //     with anonymous inner classes.
        JButton ClickMe = new JButton("ClickMe!?!");
        ClickMe.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println("Option 1: Inner class");
            }
        });

        JButton Cancel = new JButton("Cancel");
        Cancel.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
```

Event Handling

Clicking Event Handling is the way that all graphic user interface environment allow an application to respond to user (or system) events. Possible events include:

- Keyboard strokes
- Clicking buttons
- Dragging a mouse
- Low power warning or automatic system hibernation
- Attaching to an overhead projector
- And many, many others

Event Handling – ActionListener

We will specifically need to implement Java ActionListeners to respond to button clicks. To do that we will need to:

- Understand that **ActionListener**, like other listeners, are an **interface**… not a class
- Implement a Java ActionListeners Interface
- Utilize our ActionListener to receive notifications of an event and respond appropriately
- Understand the three ways we could implement an ActionListener to respond to a user pressing a button contained in a our JFrame:
    1. Implement an ActionListener interface in our JFrame
    2. Utilize an external class that implements ActionListener
    3. Implement an ActionListener utilizing an Anonymous Inner Class

Learning Objectives – Closing Comments

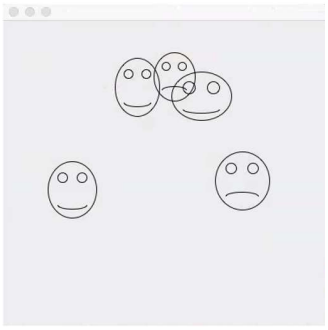
1. Explain the difference between lightweight and heavyweight components
2. Identify lightweight and heavyweight components
3. Describe how lightweight and heavyweight components render themselves
4. Identify layout managers for heavyweight and lightweight components
5. Explain what a layout manager does and identify and describe three of them
6. Explain how to achieve true Model-View-Controller architecture by removing any reference from the view to the model and from the model to the view
7. Distinguish between extending a class and implementing an interface
8. Explain multiple ways to implement an event handler for a particular object and event (anonymous inner classes vs. named classes vs. having the frame itself serve as the handler)
9. Create an event handler that implements the ActionListener interface to respond to the user clicking on a button
10. Use paint and paintComponent's Graphics object to draw a variety of shapes

FaceDraw Assignment - Closing Comments

FaceDraw and Learning Objectives: FaceDraw will be a challenging assignment this week. Our Learning Objects, Discussion, Lecture, and Examples will all be focused on helping you complete the assignment and learn the key concepts through delivering a concrete example.

In FaceDraw we will:

1. Create an application in Java that draws random faces on a window
2. Generate a random number between 3 & 10 and draw that many faces
3. Randomly generate reasonable and visually appealing face
4. Position itself to a reasonable size and location
5. Draw all faces so they are entirely within the window.
6. Draw each face with two eyes and a mouth.
7. The mouth should be randomly smiling, frowning, or in-between



Read the details in the assignment. Keep these in mind as you watch the examples. Start working on this early. It is unlikely to go well if you try to do this at the last minute.

LEWIS UNIVERSITY
A Catholic and Lasallian University

End of Session

Course Number: CPSC-24500

Week: 3

Session: 1

Instructor: Eric Pogue

Graphics Objects

Graphics Objects [link]: We will generally see the Graphics object as part of overriding paintComponent(Graphics g). The "g" object has many useful methods including:

- drawOval(x,y,width,height)
- fillOval(x,y,width,height)
- setColor(Color.BLUE)
- drawRect()
- drawArc() [link]… You will want to use this draw your smiling faces in the assignment this week.
- drawString()
- setFont()

```
//*********************************************************
// CircleDraw
import java.util.ArrayList;
import javax.swing.*;
import java.awt.*;

class ShapePanel extends JPanel {
    private ArrayList<Shape> panelShapesList;

    ShapePanel() {
        panelShapesList = null;
    }

    ShapePanel(ArrayList<Shape> ShapesList) {
        panelShapesList = ShapesList;
    }

    public void paintComponent(Graphics g) {
        // Graphics objects have lots of drawing functions
        super.paintComponent(g); // Draws other components.
        if (panelShapesList != null) {
            for (Shape s : panelShapesList) {

            }
        }
    }
}
```

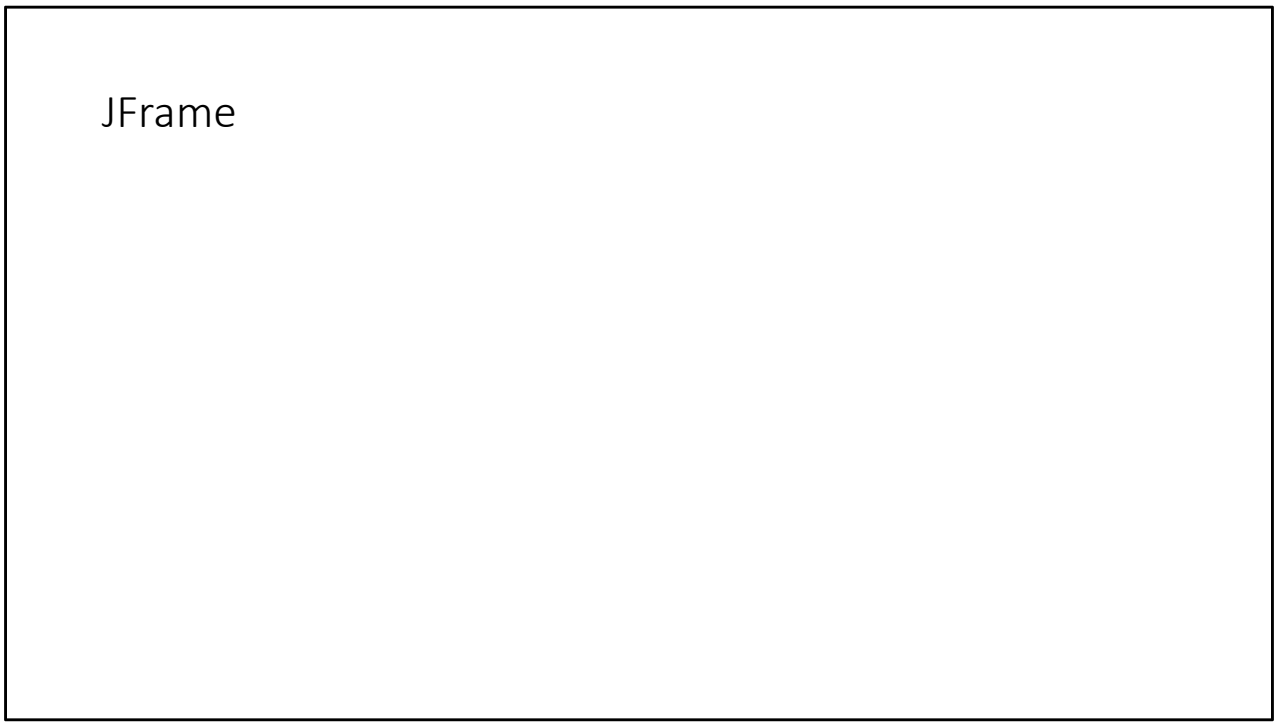
JavaDoc:
https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html

Java Graphics Tutorial:
https://docs.oracle.com/javase/tutorial/2d/basic2d/

drawArc:
http://www.java2s.com/Code/JavaAPI/java.awt/GraphicsdrawArcintxintyintwidthintheightintstartAngleintarcAngle.htm

drawArc Example:
http://www.java-examples.com/draw-arc-applet-window-example

Repaint Sequence:
OS -> Frame -> Frame's paint function -> each LW component paintComponent function


JFrame

Code Sharing & Package Visibility

```
//*******************************************************
// CircleDraw
import java.util.ArrayList;
import javax.swing.*;
import java.awt.*;

class ShapePanel extends JPanel {
    private ArrayList<Shape> panelShapesList;

    ShapePanel() {
        panelShapesList = null;
    }

    ShapePanel(ArrayList<Shape> ShapesList) {
        panelShapesList = ShapesList;
    }

    public void paintComponent(Graphics g) {
        // Graphics objects have lots of drawing functions
        super.paintComponent(g); // Draws other components.
        if (panelShapesList != null) {
            for (Shape s : panelShapesList) {

            }
        }
    }
}
```

**Title

Title: Description

```
//*******************************************************
// Polymorphism
Shape[] shapes = new Shape[3];
shapes[0] = new Circle();
shapes[1] = new Rectangle();
shapes[2] = new Triangle();
for (Shape s : shapes) {
   System.out.println(s.area());
}
```