Object-Oriented Programming

Session: Week 4 Session 1
Instructor: Eric Pogue

Agenda:

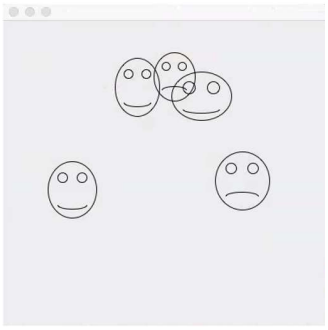
1. Discuss last week's FaceDraw Assignment
2. Review this week's Mosaic Assignment
3. Introduce our Learning Objectives for the week
4. Review each Learning Objective
5. Closing comments and next steps

Discuss FaceDraw Assignment

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'm looking for your feedback!

Review Mosaic Assignment

Mosaic and Learning Objectives: Mosaic will likely not be as challenging for you as FaceDraw was last week. Most of the elements that we will need for Mosaic we already learned last week. Therefore, this week's objectives will be more forward looking.

In Mosaic we will:

1. Draw a set of random tiles
2. Draw random shapes, letters, and colors
3. Respond to a "Randomize" button
4. Display a new set of random tiles when the button is pushed
5. Implement a interesting and unique feature of our own choosing

Note: More details are available our W4Assignment document.

Less theory and more practical development…

Learning Objectives – Week 4

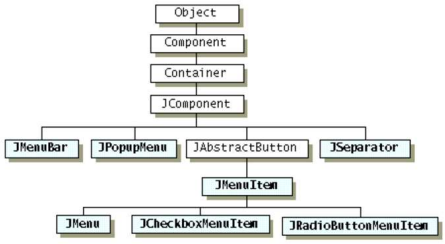
1. Implement a menu system that enables the user to trigger a variety of actions in a familiar way
2. Implement code that responds to of events including: clicking the mouse, moving or dragging the mouse, and typing a key on the keyboard
3. Design multiple intuitive ways for a user to perform a particular task
4. Implement animation using a timer and a corresponding event handler
5. Create interactive applications that adhere to the Model-View-Controller pattern
6. Understand how Java achieves speedier input and output (IO) through a hierarchy file IO classes
7. Design and implement a controller class that outputs data to a text file
8. Design and implement a controller class that inputs data from a text file and builds a collection of objects from the data read data

Any of these items would be a good candidate for your “interesting and unique” feature to add to your Mosaic assignment.

Menu Bars, Menus, and Menu Items

Java Menus are implemented with three related classes:

- **JMenuBar**: the horizontal component across the top of the frame that contains Menus
- **JMenu**: the user interface element that implements "File" in the image below
- **JMenuItem**: the sub-items that can be selected from a Menu like "New" and "Exit" in the example below
- The inheritance hierarchy for menu-related classes:



Overview:

You create the JMenuBar. You create JMenu items that have captions (like File, Edit, etc.). You create JMenuItem objects that have captions (like Open, Close, Save, etc.) and add them to the JMenu's. You associate ActionListener objects with each JMenuItem to describe what should happen when the JMenuItem is clicked. You add each JMenuItem to the JMenu. You add the JMenu to the JMenuBar. You then tell the frame to set its JMenuBar using setJMenuBar.

Mouse Clicking, Mouse Dragging, and Keystrokes

Nearly all graphical applications respond to Mouse Clicking and Keystrokes. Many applications also implement special behavior for Mouse Dragging. With Java in order to respond to these events we implement ActionListeners including:

- **MouseListener:** Interface to implement to respond to Mouse Clicking
- **MouseMotionListener**: Interface to implement to respond to Mouse Dragging
- **KeyListener:** Interface to implement to respond to Keystrokes

MouseListener Interface

The MouseListener interface requires five methods to be implemented:

- mouseClicked(MouseEvent e): Invoked when the mouse button has been clicked on a component
- mouseEntered(MouseEvent e): Invoked when the mouse enters a component
- mouseExited(MouseEvent e): Invoked when the mouse exits a component
- mousePressed(MouseEvent e): Invoked when a mouse button has been pressed on a component
- mouseReleased(MouseEvent e): Invoked when a mouse button has been released on a component

Adapter Classes are an alternative to implementing all five methods. For our examples, I will be implementing all five, even if some of them seem unnecessary. You ware welcome to use Adapters in your assignments if you prefer.

MouseMotionListener Interface

The MouseMotionListener interface includes the following methods:

- mouseDragged(MouseEvent e): Invoked when a mouse button is pressed on a component and then dragged
- mouseMoved(MouseEvent e): Invoked when the mouse cursor has been moved onto a component but no buttons have been pushed

.

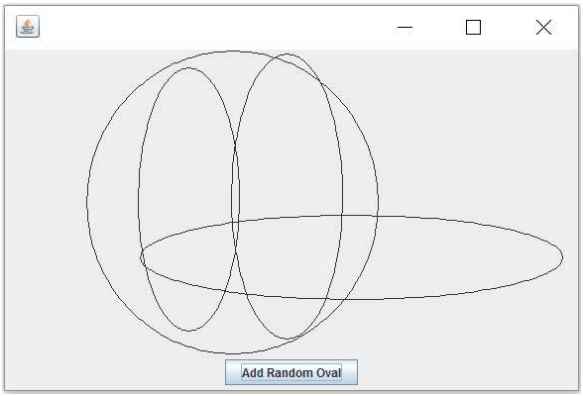
KeyListener Interface

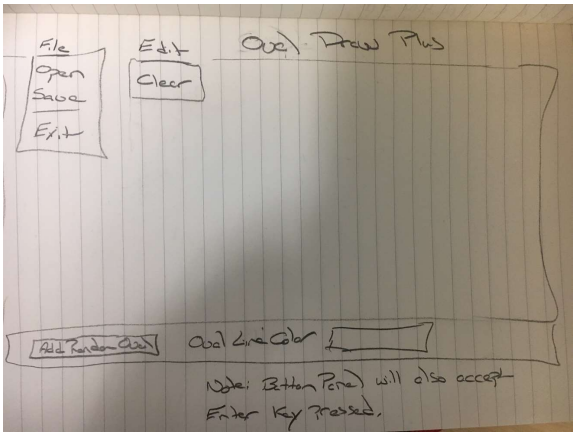
The KeyListener interface includes the following methods:

- keyPressed(KeyEvent e): Invoked when a key has been pressed
- keyReleased(KeyEvent e): Invoked when a key has been released
- keyTyped(KeyEvent e): Invoked when a key has been typed

.

OvalDraw Plus User Interface Design





Yes, this is often what a UI design looks like. In our first coding example this week, we are going to enhance the OvalDraw application to include menu elements to OvalDraw File|Exit and Edit|Clear. In addition, we will implement adding new random ovals with either a line color or a line weight. Finally, we will add file saving and opening and add the related Open and Save menu items to the application… more on that later.

I anticipate that we will also implement something with mouse clicking, mouse dragging, and timer also, but am uncertain what form that may take.

Timers

Timers are pretty straight forward to implement:

- Implement the ActionListener Interface
- Create a new timer passing in the object that implemented the ActionListener

```
//*************************************************************
// Timer

import java.util.Timer

class DrawPanel extends JPanel implements ActionListener {
    private Timer myTimer;

    public DrawPanel {
        myTimer = new Timer(1000,this);
    }

    public void actionPerformed(ActionEvent e) {
        // Do timer stuff.
    }
}
```

We will likely have a final programming example that will show some animation using timers. I'm not sure what form that example will take.

Model-View-Controller

Throughout our design and development activates we are working to create applications that segregate our View from our their Model & Data.

- The Java environment makes this more challenging than it should be at times
- Design tradeoffs need to be considered
- The most import aspect of Model-View-Controller is that the Model has no visibility to the View

Learning Objectives – Week 4

1. Implement a menu system that enables the user to trigger a variety of actions in a familiar way
2. Implement code that responds to of events including: clicking the mouse, moving or dragging the mouse, and typing a key on the keyboard
3. Design multiple intuitive ways for a user to perform a particular task
4. Implement animation using a timer and a corresponding event handler
5. Create interactive applications that adhere to the Model-View-Controller pattern
6. Understand how Java achieves speedier input and output (IO) through a hierarchy file IO classes
7. Design and implement a controller class that outputs data to a text file
8. Design and implement a controller class that inputs data from a text file and builds a collection of objects from the data read data

Any of these items would be a good candidate for your “interesting and unique” feature to add to your Mosaic assignment.

Serialization and Reading/Writing Text Files

Serialization is an object-oriented programming term that means converting an object to a byte steam usually to be written to or read from a file.

To write to a text file:

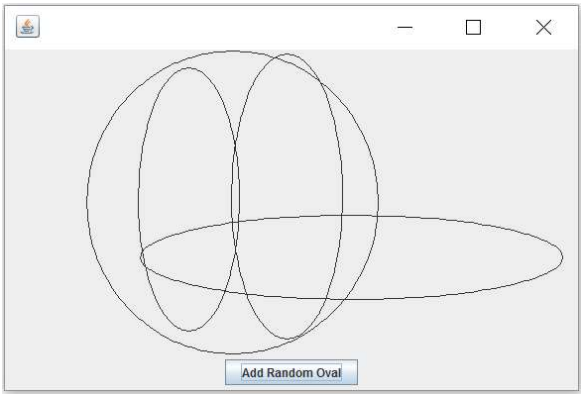
- Create a File object, feeding the file's path to the File class constructor
- Create a FileWriter to access the File
- Create a BufferedWriter to write data to the FileWriter efficiently
- Use BufferedWriter's write and newLine functions to commit the data to the file.

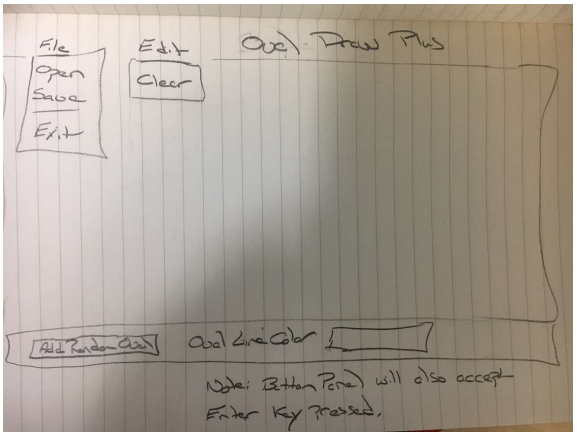
To read from a text file:

- Create a File object, feeding the file's path to the File class constructor
- Attach a Scanner object to it
- Use Scanner's readLine and hasNextLine functions to read the file

A buffered writer or buffered stream efficiently organization reads and writes for optimal disk performance. The danger is that if there is a power failure (or someone accidentally kicks your power strip) you could lose data. It is good to "flush the buffer" at critical times when using a buffered writer on mission critical projects. There is little or no danger in using a buffered reader.

OvalDraw Plus User Interface Design





Our second programming example this week will be implementing Save & Open in our OvalDraw application.

Closing Comments & Next Steps

- As promised, less theory and more development
- Look for three follow-up development examples this week:
  - OvalDraw with User Experience Enhancements
  - OvalDraw reading and writing files
  - Something with Timers and Animation

End of Session

Course Number: CPSC-24500

Week: 4

Session: 1

Instructor: Eric Pogue