

Basic GUI

Introduction to Java
CIS 226
Brad Rippe

Agenda

- Performance Test 2
- Basic GUI
 - Swing
 - JLabel
 - Event Handling
 - JTextField, JButton, JCheckBox, JComboBox, JList
 - Mouse Events
 - Adaptor Classes
 - Keyboard Event Handling
 - Layout Managers
- Assignment 7

Performance Test 2

- Overall performance was good.
- Average grade was a 92.
- Suggestion, if you see answer questions, you should work on those first. There was an additional 10 points of extra credit that most missed.
- You should have been running "GradeMT2" from the beginning of the exam. Make changes to the appropriate classes and re-run "GradeMT2".

Performance Test 2

- "GradeMT2" would display the end result of the exam. This was your final grade.
- Another suggestion, is to learn a few DOS commands. If you plan on beginning a proficient programmer, you need to learn how to navigate through files systems without "Windows Explorer". This is a necessity!
- Learn what the classpath and path environment variables do... this will be on the final.
- Also, learn how to execute java programs in different environments.

Performance Test 2 – Part 1.A

• In the Grid class, add error checking for the put() method. Display a message and return.

```
if( row < 0 || row >= GRID_SIZE ) {
    System.out.println( "ERROR" );
    return;
} else if( col < 0 || col >=
GRID_SIZE ) {
    System.out.println( "ERROR" );
    return;
}
```

Note: that we don't use System.exit(0); this method terminates the entire application. We simple want to exit the method.

Performance Test 2 – Part 1.B

```
if( cell[row][col] != null ) {
    System.out.println( "ERROR" );
    return;
}
```

Note: All cells are initialized to null upon creation of the Grid class. The only time a cell isn't null is when a Ship has been placed in the cell.

cell[0][0] = aShip; // 0, 0 is now a ship and not equal to null!

Performance Test 2 – Part 2

```

for(int i=0; i<cell.length; i++) {
    for(int j=0; j<cell[i].length; j++) {
        if( cell[i][j] != null &&
            cell[i][j].getID() == aShipID )
        {
            foundOnRow = i;
            foundOnCol = j;
            isFoundOnGrid = true;
        }
    }
}

```

Note: We can not call methods on cells that don't have Ships in them. Thus, we must check for null before calling "getID()".

Performance Test 2 – Part 3

- We must **OVERRIDE** the takeaHit() method from the MilitaryShip class. Overridden methods are methods in a subclass with the same name and parameter list as in the base class. Since takeaHit() in the MilitaryShip reduces the weaponLevel by one everytime a Ship is hit, and if its 0 removes the Ship from the Grid. InvincibleShip, simply prints a message, never reduces the weaponLevel and never removes the Ship from the Grid.

```

public void takeaHit() {
    System.out.println( "I AM INVINCIBLE!" );
}

```

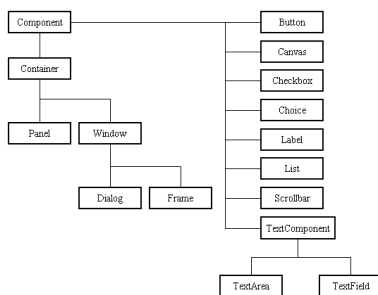
Performance Test 2 – Part 4 & 5

- Create a CommericalShip...
- 5a. Because they are private members of Ship and subclasses do not have access to private members of their super class.
- 5b. Because Grid is a protected member of Ship. MilitaryShip inherits all methods and member of the Ship class, thus we don't need to declare it in the MilitaryShip class.

AWT - Abstract Windowing Toolkit

- ☛ java.awt package contains all the classes for creating user interfaces and for painting graphics and images.
- ☛ All graphical user interface objects stem from a common superclass, Component.
- ☛ By basing almost all of the display classes on the Component class, the designers greatly simplified the interface designer's task. All of the graphical objects that inherit from Component can use the same method calls for basic functions.
- ☛ The original GUI components are tied directly to the local platform's graphical user interface. Thus, each AWT component uses *native code* to display itself on your screen

AWT - Abstract Windowing Toolkit



AWT - Abstract Windowing Toolkit

- ☛ The AWT provides four container classes:
 - Window – A top level display surface (a window)
 - Frame – A top level display surface (a window) with a border and title.
 - Dialog – A top level display surface (a window) with a border and title.
 - Panel – A generic container for holding components.
- ☛ All containers are designed to hold components.
- ☛ Note that the Applet class is a container – it is a subtype of the Panel class and can therefore hold components.

AWT - Abstract Windowing Toolkit

- There are three steps you take to create any GUI application or applet:
 - Compose your GUI by adding components to Container objects.
 - Setup event handlers to respond to user interaction with the GUI.
 - Display the GUI (automatically done for applets, you must explicitly do this for applications).

AWT Simple Example

```
import java.awt.*;

public class AWTEExample extends Frame {
    public AWTEExample() {
        super("AWTEExample");
        add(new TextArea("AWTEExample", 5, 40));
        pack();
    }

    public static void main(String args[]) {
        AWTEExample awt = new AWTEExample();
        awt.show();
    }
}
```

Swing

- JFC – Java Foundation Classes
- Introduced in 1997 at JavaOne Conference
- Known as lightweight components
- Distinguished by the “J” at the beginning of their names.
- These are 100% java components, no native code
- Encouraged use by Sun. JDK 1.0 and 1.1 provide the AWT packages for GUI programming. JDK 1.2 and above provide both Swing and the standard AWT package

Swing

- To add components to Swing Containers, JFrame, JDialog, JApplet, etc. Each top-level container contains a container to manage the visible components called the “content pane”.
- All non-menu components must be added to the content pane like:

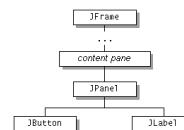
```
getContentPane().add( new JTextField("", 5, 40));
```

JFrame

- Like all other JFC/Swing top-level containers, a JFrame contains a JRootPane as its only child. The **content pane** provided by the root pane should, as a rule, contain all the non-menu components displayed by the JFrame. This is different from the AWT Frame case. For example, to add a child to an AWT frame you'd write:
`frame.add(child);`
- However using JFrame you need to add the child to the JFrame's content pane instead:
`frame.getContentPane().add(child);`

JFrame

- So instead of adding components directly to the JFrame. We will add them to the JFrame's content pane.
- If you don't add non-menu components to the content pane, the `jfm` will throw a runtime error.



JFrameExample

```
import javax.swing.*;
public class JFrameExample extends JFrame {

    public JFrameExample() {
        super("JFrameExample");
        getContentPane().add(new JTextArea("JFrameExample",
        5, 40));
        pack();
    }

    public static void main(String args[]) {
        JFrameExample jframe = new JFrameExample();
        jframe.show();
    }
}
```

JLabel

- JLabels provide text instructions or information on a GUI.
- Displays a single line of read-only text.
- JLabels also support image display as well as text. To display images you must create an object that implements the `Icon` interface. `ImageIcon` is a concrete class that implements `Icon`.
- Vertical and Horizontal alignment of a label can be set with methods, `setHorizontalAlignment()` and `setVerticalAlignment()`

SwingConstants

- This interface is used to position and set up the orientation of components in a container.
- Some of the constants are:
 - TOP
 - BOTTOM
 - CENTER
 - LEFT
 - RIGHT ...
- Ex., To set the orientation of text in a JLabel, we use:
`setHorizontalAlignment(SwingConstants.LEFT);`

Event Handling

- GUIs are event driven
- Every time the user types a character or pushes a mouse button, an event occurs. Any object can be notified of the event. All it has to do is implement the appropriate interface and be registered as an *event listener* on the appropriate *event source*.
- To process a graphical user interface event, the programmer must perform two key tasks – **register an event listener** and **implement an event handler**.

JTextField & JPasswordField

- `JTextField` is a lightweight component that allows the editing of a single line of text.
- `JPasswordField` is a lightweight component that allows the editing of a single line of text where the view indicates something was typed, but does not show the original characters.
- Both extend `JTextComponent` and inherit the functionality of that class.
- *`JTextComponent` implements MVC architecture. This is a design pattern. Design patterns are solutions to recurring problems that arise in software engineering.

JButton

- These are GUI components that resemble push buttons.
- Is a subclass of the `javax.swing.AbstractButton` class. `AbstractButton` class defines common behaviors for all buttons.
- `JButtons` support display of images as well as text button label. Can be achieved with the `ImageIcon` class.
- `JButtons` generate *ActionEvents* when a user clicks on a button and can be handled by an *ActionListener*. An *ActionListener* must define an `actionPerformed()` method.
- Another cool thing about swing components is that they support html parsing.

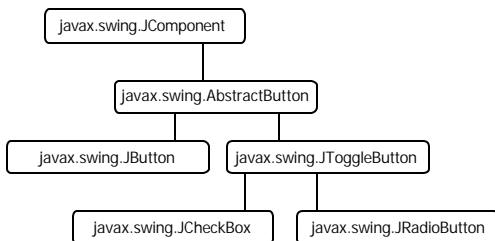
JCheckbox

- an item that can be selected or deselected, and displays its state to the user
- Can have on/off or true/false values.
- Subclass of JToggleButton which is a subclass of AbstractButton, so it inherits all the normal behaviors of a button.
- JCheckboxes generate *ItemEvents* and can be handled by an *ItemListener*. *ItemListeners* must define an *itemStateChanged()* method to handle the item events.

JRadioButton

- An item that can be selected or deselected, and which displays its state to the user. Used with a *ButtonGroup* object to create a group of buttons in which only one button at a time can be selected. (Create a *ButtonGroup* object and use its *add* method to include the *JRadioButton* objects in the group.)
- Each time the user clicks a radio button (even if it was already selected), the button fires an *action event*. One or two *item events* also occur -- one from the button that was just selected, and another from the button that lost the selection (if any). Usually, you handle radio button clicks using an *action listener*.

JButton, JCheckbox, JRadioButton Hierarchy



JComboBox

- A component that combines a button or text field and a drop-down list.
- JComboBoxes generate *ItemEvents* like JCheckboxes and JRadioButtons.
- JComboBox automatically provides a scrollbar that allows the user view all elements in the list.

JList

- A component that allows the user to select one or more objects from a list.
- JList selections can be set in two different modes. One the user can select only one item from the list. Two, the user can select multiple items from the list.
- JLists generate *ListSelectionEvents*. *ListSelectionListeners* must define the *value changed()* method.

JPanels

- The *JPanel* class provides general-purpose containers for lightweight components. By default, panels don't paint anything except for their background; however, you can easily add borders to them and otherwise customize their painting
- Panel the basic container, is a subclass of *java.applet.Applet* and has *FlowLayout* as its default *Layout manager*.

Mouse Events

- An event which indicates that a mouse action occurred in a component. This event is used both for mouse events (click, enter, exit) and mouse motion events (moves and drags).
- Mouse Events
 - a mouse button is pressed
 - a mouse button is released
 - a mouse button is clicked (pressed and released)
 - the mouse cursor enters a component
 - the mouse cursor exits a component
- Mouse Motion Events
 - the mouse is moved
 - the mouse is dragged

MouseListeners

- The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component.
- Declared methods in this interface:
 - [mouseClicked](#)([MouseEvent](#) e)
 - [mouseEntered](#)([MouseEvent](#) e)
 - [mouseExited](#)([MouseEvent](#) e)
 - [mousePressed](#)([MouseEvent](#) e)
 - [mouseReleased](#)([MouseEvent](#) e)

MouseMotionListeners

- The listener interface for receiving mouse motion events on a component.
- Declared methods in this interface:
 - [mouseDragged](#)([MouseEvent](#) e)
Invoked when a mouse button is pressed on a component and then dragged.
 - [mouseMoved](#)([MouseEvent](#) e)
Invoked when the mouse button has been moved on a component (with no buttons no down).

Adaptor Classes

- In the case of the `MouseListener` interface, the programmer must implement all declared methods implement the interface correctly.
- Some times it is not desired, to implement all methods in an event-listener interface. So, Sun has provided the event adaptor classes.
- Adaptor classes implement the event-listener interfaces with methods that have empty bodies. This, allows the programmer to subclass and override only the methods that are needed, instead of defining actions for all the interface methods.

Adaptor Classes

Event Adaptor class	Implements interface
<code>ComponentAdapter</code>	<code>ComponentListener</code>
<code>ContainerAdapter</code>	<code>ContainerListener</code>
<code>FocusAdapter</code>	<code>FocusListener</code>
<code>KeyAdapter</code>	<code>KeyListener</code>
<code>MouseAdapter</code>	<code>MouseListener</code>
<code>MouseMotionAdapter</code>	<code>MouseMotionListene</code>
<code>WindowAdapter</code>	<code>WindowListener</code>

Keyboard Event Handling

- The listener interface for receiving keyboard events (keystrokes).
- Key events are generated when keys on the keyboard are pressed and released.
- Declared 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.

Layout Managers

- Layout managers arrange the components you add to a Container according to certain rules. Different layout managers follow different rules.

FlowLayout	Displays components left to right in the same fashion that you normally write things on a piece of paper
BorderLayout	Displays the components in five areas: north, south, east, west, and center.
GridLayout	Lays components out in a grid with each component stretched to fill its box in the grid

Layout Managers

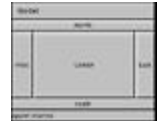
- All Java AWT containers can have an attached LayoutManager object added with the following method:

```
setLayout( new FlowLayout() );  
setLayout( new BorderLayout() );  
setLayout( new GridLayout(3,4) );
```

FlowLayout

- Gives every component its preferred size and does not resize components
- Adds components from left to right and starts a new row when the first one is filled.
- Components that don't fit are simply not shown.

BorderLayout



- Default layout manager for Frame, Window and Dialog containers.
- If you don't specify a constraint when you add a component to a container with a BorderLayout, a constraint of CENTER is assumed. If you add more than one component without a constraint, only the last one will be kept and the others will not be shown in any position.

GridLayout

- The available space is divided into a grid of equal-sized cells and the components are forced to fit.
- The order of addition defines where each component goes, with rows being filled from left to right starting at the top left.

Assignment 7

- Need to construct two classes, Quiz.java and QuizTester.java
- Add the specified member variables and methods to the Quiz class.
- You should have two constructors and the appropriate 10 methods. Two are provided for you.

White Board Explanation

- Check your grades!
- Quiz 5 next week!
- Assignment 7 – due 11/20
- Have a good weekend!