# CIS 3145 Class Notes: Text Chapter 21

## Graphical User Interface (GUI) with Java Swing

**Objectives**

* Describe the Java GUI toolkits
* Describe the Swing inheritance hierarchy
* Create a Form (Frame) and add Controls
* Describe layout managers for forms
* Create a Table Control and a Model for the Table
* Display dialog boxes to display messages and get input

**Introduction**

All programs need a way for humans to interact with them. The study of this process is called Human Computer Interaction (HCI). See [https://www.interaction -design.org/books/hci.html](https://www.interaction-design.org/books/hci.html). Most programs use a GUI interface to allow users to interact with the computer, but that is changing as we start wearing our computers, tapping our tables, and shaking our phones.

Here we learn how to create a graphical user interface in Java code with NetBeans. The interface is called a frame or form. It and all of the controls and components on the form are objects. Remember that objects have **properties**, **methods** and **events**.

There is a **sequence** for working with an interface. **First** create the form and *add the controls* needed for the program. **Second** *change the properties* of the controls, including the ‘**Variable** **Name**’. Then **third**, ONLY after all of the properties are set, we *write the code* for the program.

**Swing Inheritance**

The **Abstract Window Toolkit** (AWT) was the first system for creating a GUI in Java. This system used objects closely *coupled* to the underlying operating system which made it difficult to move a program from one operating system to another. The **Swing** system uses light weight components written in Java making it more portable and popular. All Swing components start with the letter J (JFrame vs. Frame).



**Frame Component**

The JFrame class defines a swing component which displays a window to the user. It includes the standard minimize, maximize, and close buttons in the upper right hand corner of the window. The JFrame object created from the class has **set** methods for properties of the object such as setTitle, setSize, setVisible and setLocation. To make the program adaptable to different operating systems the ‘look and feel’ of the GUI can be set to the default look and feel with the UIManager class

Because all components and controls are containers we can create bundles of controls together in special containers called panels. Panels are invisible and can hold groups of related controls to be added to a frame.

Controls, such as buttons, have set methods to change their properties, such as setText, setEnabled,

Create control objects, add the objects to the panel with the panels add method, then add the panel to the JFrame with the JFrame’s add method.

**Create Event Handlers Methods for active Controls**

An **Event** is something that happens to a control: a user can press a key, click with a mouse, activate a hot key, etc. An **Event Handler** is the sub procedure that has code which runs when the event occurs. Each control has many events that can happen to it.

The default event handler is the **action performed** sub procedure. The **action performed** event is the default event for buttons, radio buttons, checkboxes, and combo boxes.

In order to link the click event of a button to a sub procedure / method the button’s addActionListener method is called. The argument value for this method is an anonymous class. This linking can also be done with a lambda expression.

**Write the code for the Event Handler Methods**

Import statements and packages are used before the class declaration of the Frame class. All code must be inside of the class. Variables are declared the same as before as part of the class or a method, as public or private. In the JFrame class the initComponents() method is used manage the creation of controls in the JFrame object. This method is called from the JFrame constructor.

When we write the code to respond to an **event** the code is typed into the **event handler** method. To access the methods and properties of any control on the form use the **Variable** **Name** for the control with the dot notation.

nameTextField.getText() 🡨will get the text typed by a user in a text field

messageLabel.setText(“Message”) 🡨will set the text property of the label to “Message”

**Describe Common Control Methods**

Any property we set in the design window of the GUI can also be changed dynamically when the program is running. Most properties have both a **get** and **set** method that is used to **read** the value or **change** the value of the property. We can write code to determine if a textbox is set to be ‘Editable’ and then write if statements to do different things depending on the state of the textbox.

**Layout Managers**

**Layout managers** determine where and how controls are displayed in a window. The setLayout method of a component determines which layout manager is used.

The **JFrame** component uses the **Border Layout** by default. This layout has five zones: North, East, South, West and Center. Each zone can only have a single component assigned to it, but that component can hold multiple components. For example, a panel component can have many controls in it and be added to a zone.

The **Jpanel** component uses the **Flow Layout** manager by default. This layout system keeps components in order from left to right but will move them vertically if the window size decreases.

The **Grid Bag Layout** and **Grid Layout** managers create a grid of cells where components are added. With the Grid Layout system each cell is the same size while the Grid Bag system allows each cell to be customized, which makes it very flexible, but more complicated to manage.

The **Card Layout** treats each component of a container as a card in a stack, with only one card (component) visible at a time. The **Box Layout** puts components in a horizontal or vertical row of cells.

This code creates a JFrame object variable named “frame”. The constructor sets the frame title to “Product Manager”. It also sets the size of the frame to 600 by 400 pixels and changes the layout manager to the FlowLayout system. Finally, four buttons are added to the frame.

JFrame frame = new JFrame("Product Manager");

frame.setSize(600, 400);

frame.setLayout(new FlowLayout());

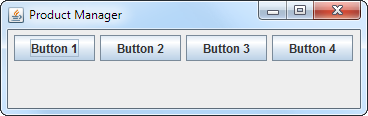
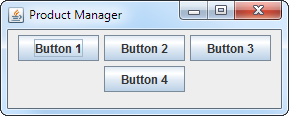
frame.add(new JButton("Button 1"));

frame.add(new JButton("Button 2"));

frame.add(new JButton("Button 3"));

frame.add(new JButton("Button 4"));

Changing the size of the frame will cause the location of the buttons to change.

 versus 

**Tables**

The **JTable** control will display data in a predefined format. It saves us the trouble of creating labels and text fields to display our data. The catch is that the JTable needs a data model class. The data model class must inherit the **Abstract Table Model** class. The data model class that we create is the argument for the Table constructor.

MyTableModel = new MyTableModel();

JTable table = new JTable(MyTableModel);

MyTableModel is a class we create. It inherits the Abstract Table Model and must implement / override a getRowCount, getColumnCount, and getValueAt method. With these methods the JTable class creates the control and populates it with data. The Table Model class acts as an intermediary between the data layer and the presentation layer.

public class MyTableModel extends AbstractTableModel {

@Override

public int getRowCount() {

…}

@Override

public int getColumnCount() {

…}

@Override

public Object getValueAt(int rowIndex, int columnIndex) {

…}

}

**Scroll Bars for Controls**

A scroll bar can be displayed on any control, including the JTable control, by adding the object for the control to a JScrollPane control. The setHorizontalScrollBarPolicy() and setVerticalScrollBarPolicy() methods are used to make the scroll bars appear “Always”, “As Needed”, or “Never”.

JScrollPane myScrollPaneObject = new JScrollPane(productTable);

myScrollPaneObject.setHorizontalScrollBarPolicy(

HORIZONTAL\_SCROLLBAR\_AS\_NEEDED);

**Use the JOptionPane Class to Display Information to a User**

The **JOptionPane** is a popup message box or dialog box used to give users feedback as they interact with the GUI. The **JOptionPane** has a very important method, **showMessageDialog** (), which activates the message box. The first argument for this method is the object of the parent, the next three arguments used to call the showMessageDialog will fill in the **title** bar of the dialog box, the **message** shown in the body of the dialog box, and what type of Icon will be displayed (**messageType**). The **title** and **message** are strings and it is typical to create a variable with the information to be displayed. The message type can be an error, warning, information, question, or no icon.

The **JOptionPane** can also be used to get input from the user. The **showConfirmDialog ()** method is like the message dialog but the messageType argument is used to determine the number and type of button displayed in the dialog box (OK, Yes & NO, Yes & No & Cancel, or OK & Cancel). In addition, the showConfirmDialog will return an integer that represents the button that is pressed. Both the message type and output from the showConfirmDialog can be represented as static JOptionPane constants.

The dialog boxes are modal meaning the user can not switch back to the parent control until the dialog box is closed.