Programming GUI'S with AWT

INTRODUCTION

AWT, or Abstract Window Toolkit, is Oracle's original system for creating GUI's (pronounced "gooey"), or Graphic User Interfaces, in the java programming language.

Why learn AWT?

AWT is not widely used in Java programming anymore because it has been replaced by more specialized systems such as Swing and, more recently, JavaFX However, it is important to learn for two BIG reasons:

1. Older programs use AWT and if you are ever required to debug them or work on them you will need to know how AWT functions.
2. Swing and JavaFX are built on AWT. If you want to understand how they work and what to do when they don't, you need to learn at least the basics of AWT.

Why is AWT not used anymore?

AWT is called a heavyweight application. This means that for every graphic element it actually calls up information from the underlying OS to make the graphic. This causes applications to look and even behave differently on different systems. It makes the multiplatform purpose of Java a little pointless.

AWT is also very generalized. It doesn't have preprogrammed classes for specific elements and a lot of work must be put in to do fancy (or sometimes even simple) graphic work. Two lines of JavaFX could equal 25 lines in AWT.

Disclaimer

Programming GUIs is fun. It's much easier to show off your programming skills to the average person when you have a GUI, but getting there is hard. Using AWT is challenging, especially when trying more complex elements. The good news is that it gets easier and moving to Swing and JavaFX becomes much simpler once you know AWT. Good Luck.

Other Resources

If at any time you have difficulty with this guide try looking at one or more of the following sources:

- Chapters 24, 25, and 26 in *Java: The Complete Reference, 10 Ed.* (the Textbook)
- http://www.splessons.com/lesson/awt/
- https://www.tutorialspoint.com/awt/index.htm

DEFINITIONS

To learn AWT you need to know how the system functions and for that we need to be able to speak using the appropriate language.

Component – Any object having a graphical representation that can be displayed on a screen and interacted with by a user. Examples include buttons, checkboxes, and scrollbars.

Container – An object that can hold other AWT components. Panels and Frames are types of containers.

Frame – A top level window including built in titles, borders, minimize button, maximize button, and close button.

Panel – Space in which components (including other panels) can be attached. Useful for creating custom layouts.

Layout – How user elements are organized on a screen. Specific classes are designed to change how the user interface elements are laid out such as: GridLayout, FlowLayout, or GridBagLayout.

Event – A change in the state of an object. The Event Class is the basis of all GUI's in Java. There are two types:

Foreground Event – Events requiring direct user interaction. Example: Clicking a button.

Background Event – Event that do not require user interaction.

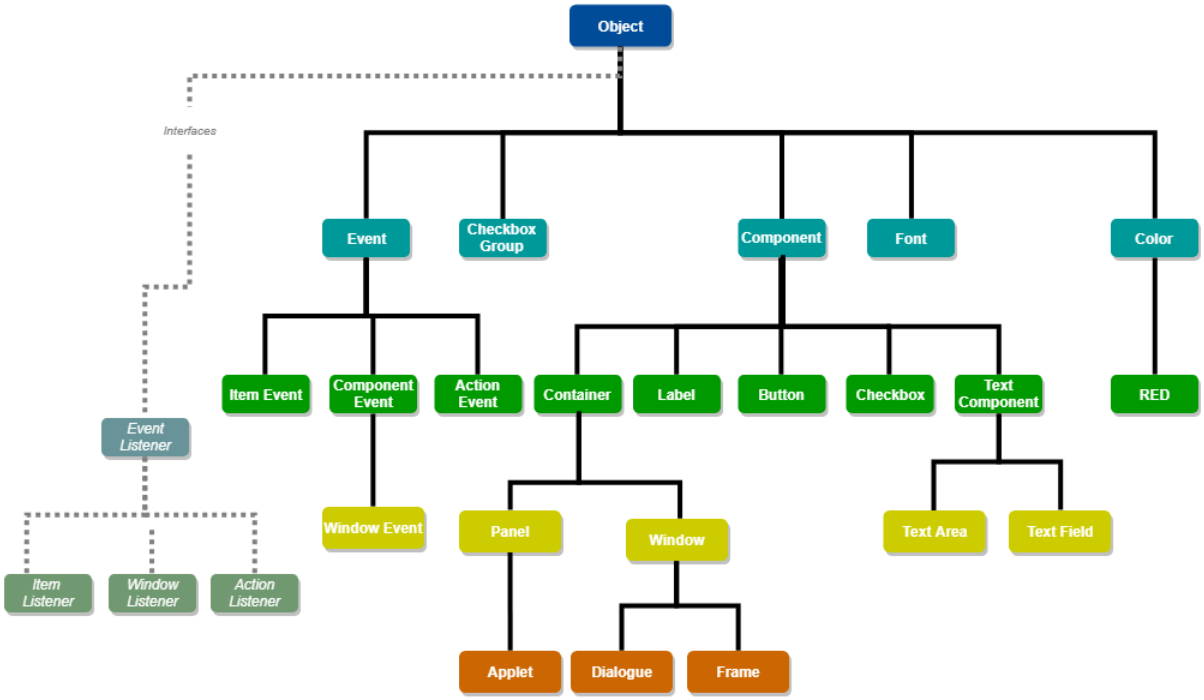
Event Handling – Process of registering an event and deciding what happens when that specific event occurs.

Event Source – Object on which events occur. The objects say "Something happened". AWT has specific classes that handle different types of events, such as: ActionEvent, MouseEvent, and KeyEvent.

Event Listener – Object that is registered to a corresponding event and "listens" in the background waiting to do something if the source tells it something happened.

HIERARCHIES

Part of the reason GUI's haven't been introduced earlier in Java class is because they have complex hierarchies that define how they function. If you don't understand inheritance you are already behind and should go back and review how they work. The other point is that if you were brand new to Java and were told to program in AWT it would melt your brain. Review the hierarchies mapped below and compare them to the above definitions. Not everything is defined and if you don't understand something I recommend you look it up in the API. This is by no means an exhaustive diagram and AWT contains many 0ther classes.



AWT Hierarchy Diagram

THE BASIC PRINCIPLES

The Power of MVC

The strongest aspect of AWT is that the functionality is completely separate from the graphic components. This is a big asset because you can change the look of a the GUI without having to rewrite the code that actually makes the program work. For example, you could completely change the formula that creates a value and the end user will still see exactly the same thing because all they see is the interface, not the implementation behind it.

This whole system is called MVC or Model-View-Controller. It is a Software Architectural Design Pattern (doesn't that sound fancy?), and it is a very commonly used model in all kinds of programming languages including Java, C#, Ruby, and PHP. **Model** is another word for data or functionality. This is what we would call the guts of the program, where all the calculations and methods to work with data occur. The **View** is what we are learning now: GUIs, or how the user sees the program. The **Controller** is the connecting entity between the data and the visual interface. It controls all actions between the View and the Model. In Java this is actually the EventHandlers that we will discuss later. The MVC design pattern is very good at promoting usability and organization in software code.

How To Program GUIs with AWT

Before getting to the programming part, you need to sit down and decide what you are going to make and what it will look like. During the first part of this you are just thinking in terms of Interface. Draw a sketch of what you want the program to look like and think about what a user would want to be able to do with it. With simple programs you don't necessarily need to think about the implementation yet, and it may be useful to have the functionality part of the program done before designing the GUI.

The first part of programming a Java GUI is the window, which is the container that holds every other part of the program. For most programs you will want to use a Frame object which is a subclass of Window. A Frame creates a window with a title, Menu Bar, and a border. It's important to know that the exit Button included in a Frame does not actually close the Window and you will need to program the close function in.

The next part of a program is smaller containers called Panels. Strictly speaking, you don't need Panels, but they make the layout process much easier. Panels are added to the Frame in specific Layouts to divide up the Frame. Each panel also has its own ll~~L~~ayout determining how components that will be added to it will arrange themselves . The default layout for AWT Containers is FlowLayout. FlowLayout arranges components in a line across the Container only moving to the next line when the edge is reached. Other Layouts include: GridLayout, BorderLayout, GridBagLayout, and CardLayout.

You then can create Component Objects. I find it helpful to look at your sketch and name out each element as what kind of Component it is. Highlighters are useful for this step. If you are organizing with multiple methods (and you should) it can be useful to have the Components declared as member variables for easier access between classes. Create each component and add them to the appropriate Panels or the Frame. The order in which you add Components to the Containers is the order in which they appear based on the specified ll~~L~~ayout. During this step a lot of programmers add the EventHandlers, but because we need to ~~teach~~ learn more about how EventHandlers work I'm going to do that in a later step. ~~You~~

At this stage you need to have or create your functionality. This is the guts of the program and except for the fact that you have no way of interacting with the program, it should be complete. For organization purposes, keep your GUI and your guts in separate Classes. Your main method should not contain anything except calls to other methods. This helps keep you organized.

Now, on to Event Handling. For every component you need at least one Event Listener(multiple Event Listeners can be register with one source). The EventListener is going to collect a signal from an Event Source saying the Component was activated and then carry out set lines of code that perform the functions for the Component. The Event Source is automatically created for a Component when you initialized them. You have to register the Listener with the Component so it knows what to listen to exclusively. The interesting (and mildly

irritating) part about AWT is that it doesn't have specific classes for each type of Component. Rather, it has Interfaces that cover groups of Components. You have to override each instance of an Interface to add a Class that processes each Component. Within the overriding class, you write the function of the Component. A button might change the value of a variable or clear a label from the Frame.

List of Steps

Here is a list of what we were talking about above. Remember that somethings can be done in different orders depending on what makes the most sense for you.

1. Design GUI appearance and Function
2. Create Window/Frame
3. Create Panels
4. Create Components
5. Create Functionality
6. Create Event Handlers

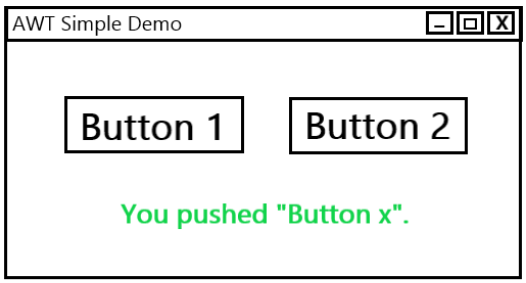
DEMONSTRATION

For practicality, I am going to walk you through how to program a super simple AWT GUI following the steps laid out above. This should help make the above text make more sense. For the purposes of this demonstration, I have declared most of the variables as member variables. I didn't screen shot these individually but you will need the following declared to make the program work.

```
25    public class GUI {
26        private static Frame mainFrame;
27        private static Panel topPanel;
28        private static Panel bottomPanel;
29        private static Button button1;
30        private static Button button2;
31        private static Label instructionLabel;
32        private static Label pushedButton1;
33        private static Label pushedButton2;
34        private static FlowLayout centeredText;
35        private static Font labelFont;
```

1. Design GUI Appearance and Functionality

We are going to make something a bit overly simplified with the goal to model the classes and interfaces you will need later in your own projects. I want to make a Frame with a title. Then I want a couple buttons and labels, and I want the label to change depending on the buttons. My sketch is displayed on the right.



2. Create Window/Frame

First, let's just create the Frame. Remember that for every new structure we add we will need a new import statement(Line 8). I declared mainFrame as a private member variable (Line 14). Private because it's good coding structure, and a member variable because it makes it easier to work with within the GUI class. My main() (Line 16) is only going to hold calls to other methods to keep it clean and easily readable. Line 23 actually creates the Frame. The text in the brackets functions to name the Frame. The name displays in the menu bar at the top left of the window.

```
7   */
8   import java.awt.Frame;
9   /**
10   * A simple program demonstrating how to use the AWT GUI system
11   * @author eliza
12   */
13  public class GUI {
14      private static Frame mainFrame;
15
16      public static void main(String args []){
17          GUIMaker();
18      }
19
20      public static void GUIMaker(){
21          //Create project Frame(title)
22
23          mainFrame = new Frame("AWT Simple Demo");
24
25      }//Close GUIMaker
```

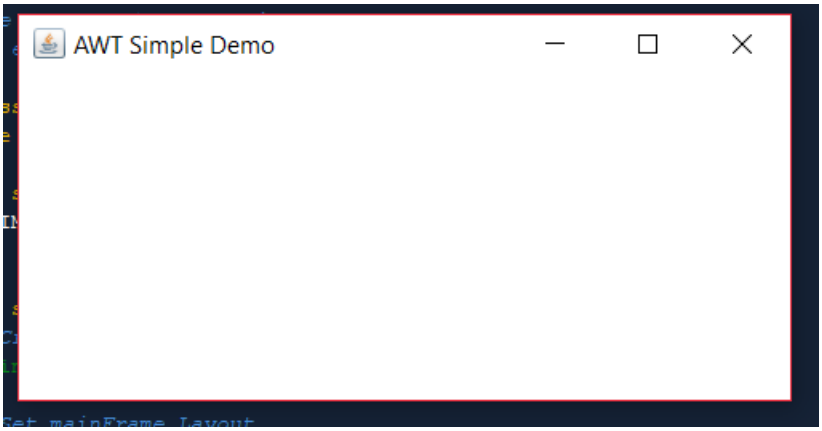
Now we need to set some attributes for the Frame. We need a size and a Layout so the program knows how everything we add to the frame later should be arranged. I picked a GridLayout. GridLayout causes components to be arranged in cells of equal size in a grid system of a defined size. What I'm really trying to do is get two panels stacked on top of each other, but AWT doesn't have a simple way to arrange a verticle stack. Import the GridLayout first.

```
9   import java.awt.GridLayout;
```

Set the mainFrame Layout as GridLayout (Line 26). The numbers in the brackets represent the numbers of columns and rows respectively. In this case I want one column and two rows. Line 29 sets the size of mainFrame to 500 pixels wide by 250 pixels high. The last line of code (Line 32) sets the Frame to visible. If you don't have this line it doesn't matter how good your GUI is, it won't display when you run the program.

```
25          //Set mainFrame Layout
26          mainFrame.setLayout(new GridLayout(1,2));
27
28          //Set mainFrame size
29          mainFrame.setSize(500,250);
30
31          //Set the mainFrame as visible
32          mainFrame.setVisible(true);
33      }//Close GUIMaker
```

If you run the program at this point, it should display this window:



If you try pushing the top right exit button, nothing will happen. So let's fix that. You will need a couple more imports.

```
10  import java.awt.event.WindowAdapter;
11  import java.awt.event.WindowEvent;
```

As you can see, these classes fall under the event superclass so this is actually going to be our first example of event Handling. I'm including it early because it will make testing the program easier while we're creating the GUI. The important line of code here is Line 57 This is a call to close the program. The (0) tells the system that it is closing without error. Different numbers here would indicate different abnormal closing statuses.

```
53          //Program the exit button
54          mainFrame.addWindowListener(new WindowAdapter(){
55              @Override
56              public void windowClosing(WindowEvent windowEvent){
57                  System.exit(0);
58              }//Close windowClosing()
59          });//Close addWindowListener
```

3.Create Panels

Looking at how I want the Window laid out, I think the best way to get that layout is to divide the window into two sections: an upper and a lower. One will hold the buttons and one will hold the label at the bottom. To do this I am going to make two containers called Panels. Each container needs a Layout that tells it how to add components to it. In most cases a container's Layout can be set when you

instantiate it. Here we are using the default FlowLayout but we want to do some more specialized, repeatable things with it. To do this we are creating a FlowLayout object called centeredText that stores the attributes of the desired Layout. We can then apply it to multiple containers.

```
11    import java.awt.Panel;
```

```
61            //create top  & bottom Panels
62            centeredText = new FlowLayout(FlowLayout.CENTER,35,35);
63            topPanel = new Panel(centeredText);
64            bottomPanel = new Panel(centeredText);
```

Our centeredText is handling the Layout within the panel. When we go to add the Panels to the mainFrame we want to set them in an area of the previously defined BorderLayout that was applied to mainFrame earlier. BorderLayout works by having five different regions: NORTH, SOUTH, EAST, WEST, and CENTER, and a component can be assigned to any of those regions. Note that only one component can be assigned to each region. Layouts can also be assigned with separate methods after the instantiation.

```
66            //add Panels to mainFrame
67            mainFrame.add(topPanel, BorderLayout.NORTH);
68            mainFrame.add(bottomPanel, BorderLayout.SOUTH);
69
```

At this point, the output of our GUI is exactly the same as our last step. Everything we did in this step is preparing containers and Layouts so the GUI looks appropriate at the end. Layouts and manipulating the visual aspect of GUIs is the most difficult part.

4. Create Components

Here we get to do the fun part: making buttons. We need to instantiate the buttons and when we do we can set the button's text. You can do this with a separate method later if you want.

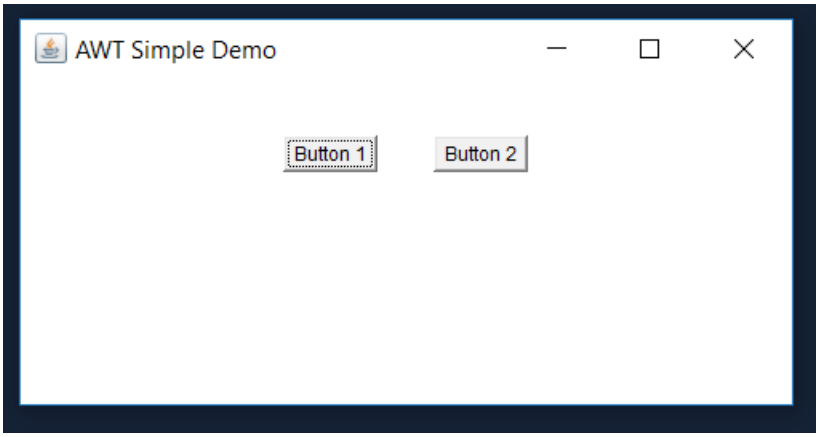
```
14    import java.awt.Button;
```

```
70            //create Buttons
71            button1 = new Button("Button 1");
72            button2 = new Button("Button 2");
73
```

To display the buttons you will need to add them to the top panels.

```
85            //add Buttons to topPanel
86            topPanel.add(button1);
87            topPanel.add(button2);
88
```

After these steps the GUI should look like this:

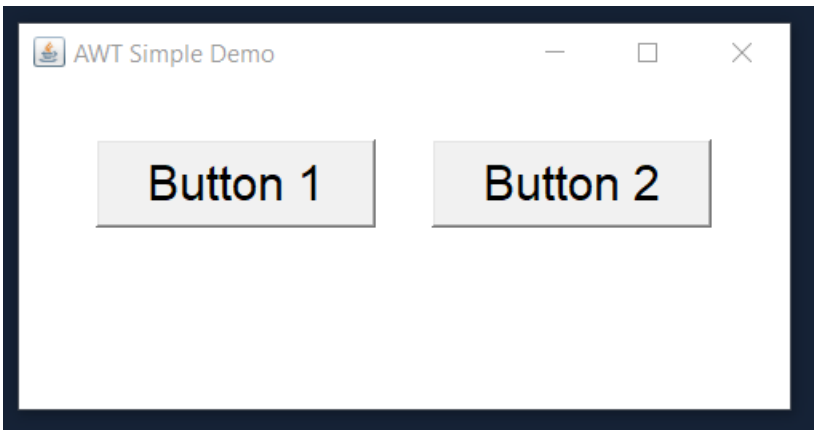


You can just use the default font and button size, which is based on the size of the text inside, but I want something more interesting so we're going to set them independently. I want both buttons to be the same, so I'm going to create a Dimension object called buttonSize to store the information. I'm using the setPreferredSize because I have an applied Layout. If I didn't, I could use another method to set a minimum or maximum size to define the button dimensions. Both Dimension and Font will need their own imports.

```
16    import java.awt.Dimension;
17    import java.awt.Font;
```

```
74            //adjust the size of the Buttons
75            Dimension buttonSize = new Dimension(175,55);
76            button1.setPreferredSize(buttonSize);
77            button2.setPreferredSize(buttonSize);
78
79            //change the font size of the buttons
80            Font buttonFont = new Font("",Font.PLAIN, 30);
81            button1.setFont(buttonFont);
82            button2.setFont(buttonFont);
```

Now we have some slightly more stylized buttons.

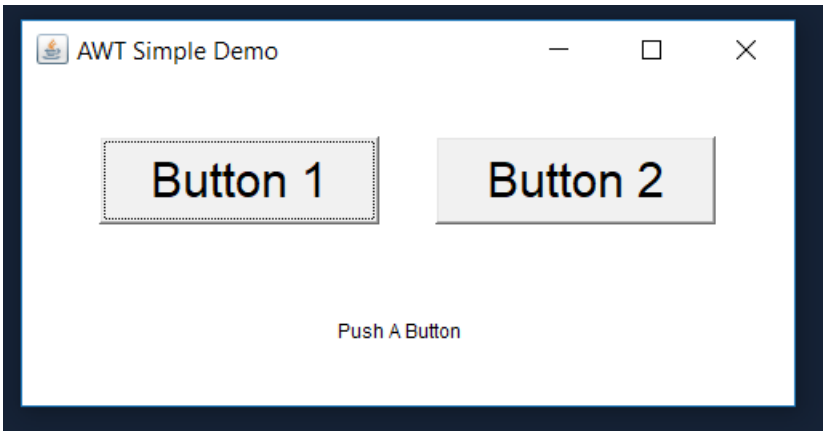


We need some text to display when we click the Buttons so we are going to need some Labels. I want one to display when the program is launched and one for when each button is pushed. Another option would be to change the text of a single Label,

but formatting is more difficult so I did it with separate Labels. You will need yet another import. Create the Label and add the text to it. Then add the first instructionLabel to the bottomPanel so it displays at the program's launch.

```
88          //create Labels
89          instructionLabel = new Label("Push A Button");
90          pushedButton1 = new Label("You pushed Button 1");
91          pushedButton2 = new Label("You pushed Button 2");
92
93          //add instructionLabel to bottomPanel
94          bottomPanel.add(instructionLabel);
```

We have labels now, but they are small and boring.

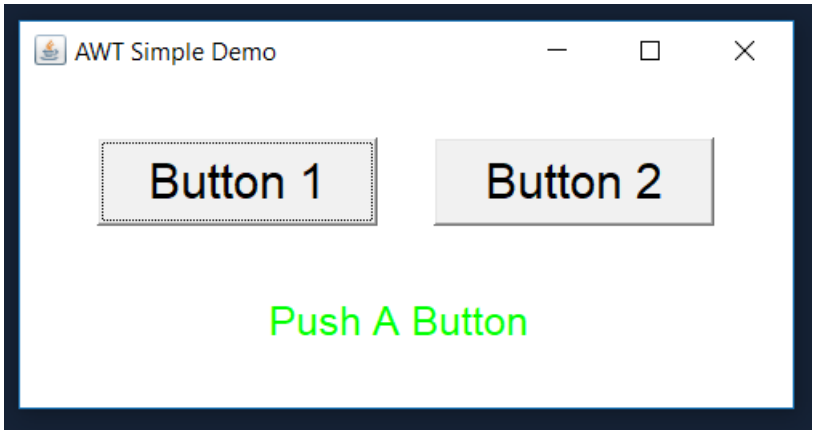


More formatting! We're going to make all the Labels look the same so let's just make a Font object to hold all those variables. To match our ~~picture~~ sketch we want the Label text to be green. The color of text can only be set on initialized objects, so it can't be stored as part of a Font. You will need an import of Color(Line 15).

```
15  import java.awt.Color;
```

```
96          //create and set Label Font
97          labelFont = new Font("",Font.PLAIN,25);
98          instructionLabel.setFont(labelFont);
99          pushedButton1.setFont(labelFont);
100         pushedButton2.setFont(labelFont);
101
102         //make Label text green
103         pushedButton1.setForeground(Color.GREEN);
104         pushedButton2.setForeground(Color.GREEN);
```

Now our GUI should look like this:



At this point all the visual parts of the GUI are finished. Congratulations, you finished messing with the formatting.

5. Create Functionality

This is an important step, but isn't really demonstratable for this program because it's so simple. The principle is that you need to separate the guts of how the program works from the building of the GUI. You should be able to take the function class and stick it into another GUI and it should still work, but with a different look. You can see an example of this division in the program under the "A More Complex Example to Explore" section.

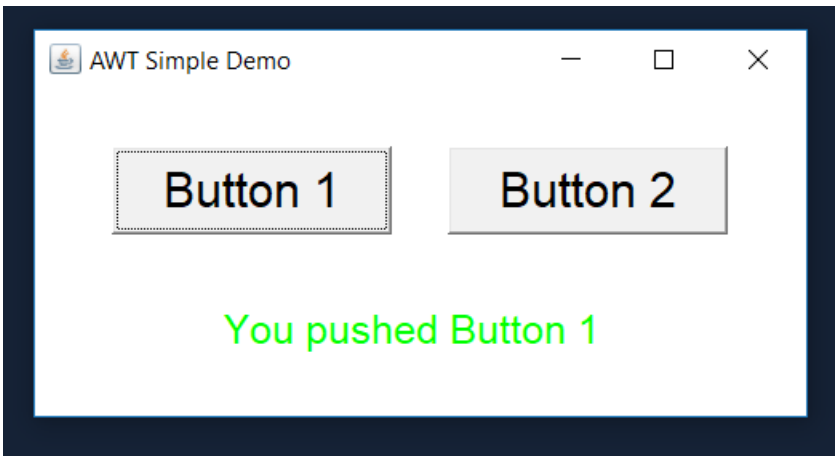
6. Create Event Handlers

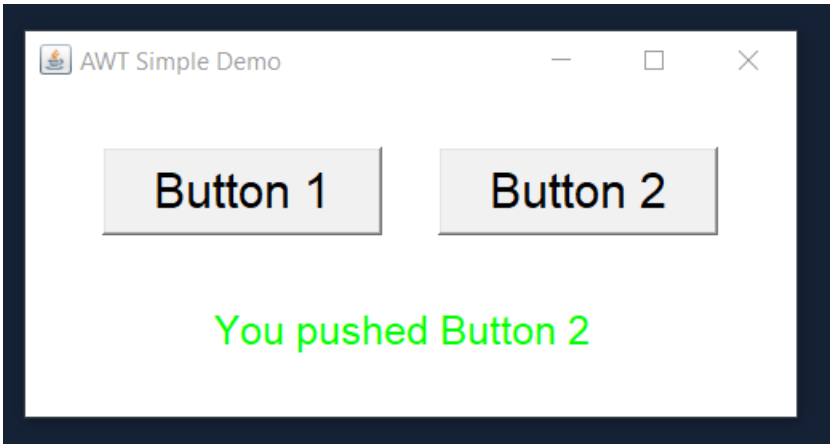
The GUI looks pretty good, but we still need to make it work and do something when we push the Buttons. To do this we need each button to have an EventHandler. For buttons the specified Event is called an ActionEvent. It uses the ActionListener interface. To use the ActionListener we either have to add an Override or use the class implements ActionListener process. I've chosen to use the Override method because it is easier for demonstration purposes. The method actionPerformed is the only method (abstract or otherwise) contained and required for the ActionListener interface. The ActionEvent is what is registered with the ActionListener and tells the whole method to start when the button is pushed.

```
114     public static void eventHandlers(){
115         //EventHandler Button 1
            button1.addActionListener(new ActionListener() {
117             @Override
                public void actionPerformed(ActionEvent ae) {
119                 bottomPanel.removeAll();
120                 bottomPanel.add(pushedButton1);
121                 mainFrame.setVisible(true);
122             }//close actionPerformed
123         });//close button1.addActionListener
124
125         //EventHandler Button 2
            button2.addActionListener(new ActionListener() {
127             @Override
                public void actionPerformed(ActionEvent ae) {
129                 bottomPanel.removeAll();
130                 bottomPanel.add(pushedButton2);
131                 mainFrame.setVisible(true);
132             }//close actionPerformed
133         });//close button2.addActoinListener
```

In these handlers, each time a button is pushed the Panel is wiped clear and then the corresponding Label is added to the panel. Then the whole mainFrame is once again set to visible so everything can be seen. You can put practically any code into the EventHandlers to be performed when a component is activated.

Here we have what the GUI should look like after each button is pushed.





Once you have added the EventHandlers your program is done. This was a simple program but should have taught you the basics of how to create a GUI using AWT. If any part of this was confusing or you had questions, now is the time to ask.

Resources for this program

If you want to view this program completed and in its entirety, click the below link:

https://github.com/efaux01/cit130_ccac_AWT/blob/master/AWTSimpleDemo/src/AWTSimpleDemo/GUI.java

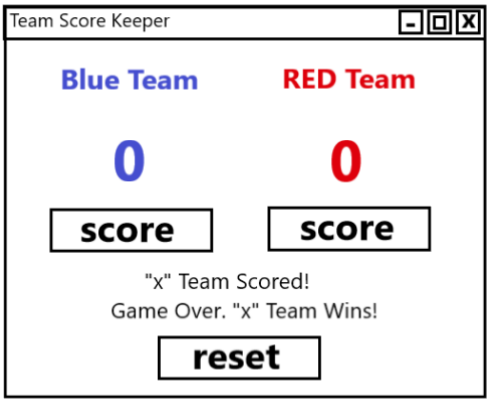
A More Complex Example to Explore

Now that you know what some of the lines of code do, here is a more complex sample program for you to explore. Try playing with sizes, colors, and layouts and see what they do.

https://github.com/efaux01/cit130_ccac_AWT/tree/master/AWTConverter/src/AWTGUIDemo

TRY THIS YOURSELF

Create a program to track the score of two teams. I have created the sketch so everyone is trying to make generally the same thing. The reset button should always reset the scores to zero.



Expansion Ideas

Once you have created the above project (and it works!) you might want to add some of the following features to make it more complicated.

- Score a Game with different point values (such as football or Baseball)
- Customize the text color every time Red and Blue teams are displayed
- Change button, window, or panel colors
- Make the score keeper track best 2 of 3 or 3 of 5 games

PROJECT IDEAS/REQUIREMENTS

Now that you have gone over the basics, try creating a program with a bit more complexity. For this project, please have at least one component that we have not discussed yet. Examples include:

- Combo Box

- List Window
- Drop Down Menu
- Button w/ Icon
- Scroll Bar
- Check Box
- Text Area or Text Field
- Slide Bar

Include the preliminary sketch of your proposed GUI. I would love to see your ideas but here are some potential project ideas to get you started:

- Calculator
- Pig Latin Translator
- Password/User ID checker
- Tic Tac Toe

Don’t try to do something too complicated to start. Start simple and work your way up.