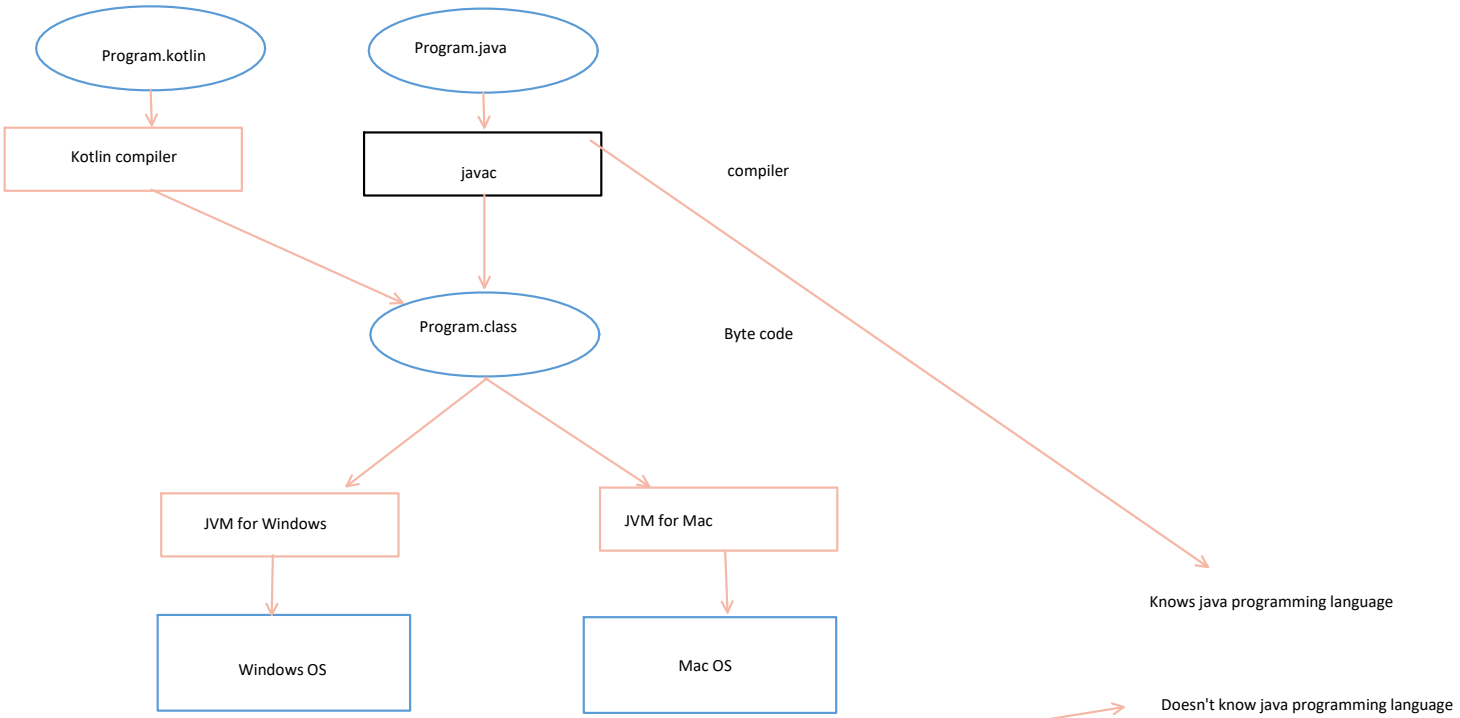
Java Architecture

Monday, 25 May 2020 3:24 PM



How do you run a java program?

C:\>java Program

- You are running a Program.class, **not program.java**
- **Is Program.class a java code?**
  - **No. It's a byte code**
- **That means 'Java' is not running Java Program. Why then we call it java?**

Java Terminology

Monday, 25 May 2020 3:37 PM

Java —> Represents to different and related ideas.

1. It's a … programming language.
   a. But
      - **Android Phone is Not a Java Phone but we using java language**
      - **There were older phone that were java Phone still we never wrote a java code in the phone.**
2. It's an Execution Platform t Execute can Application
   a. Android Phones use Java Programming language to write the program
   b. **But** Andorid Phones don't use Java Execution Platform
      i. They are not considered Java Phones
   c. You may write an Android application in a new programming language called Kotlin and can run it on Java Platform

Java compiler. Knows java

C:\> javac Program.java

C:\>**java Program**

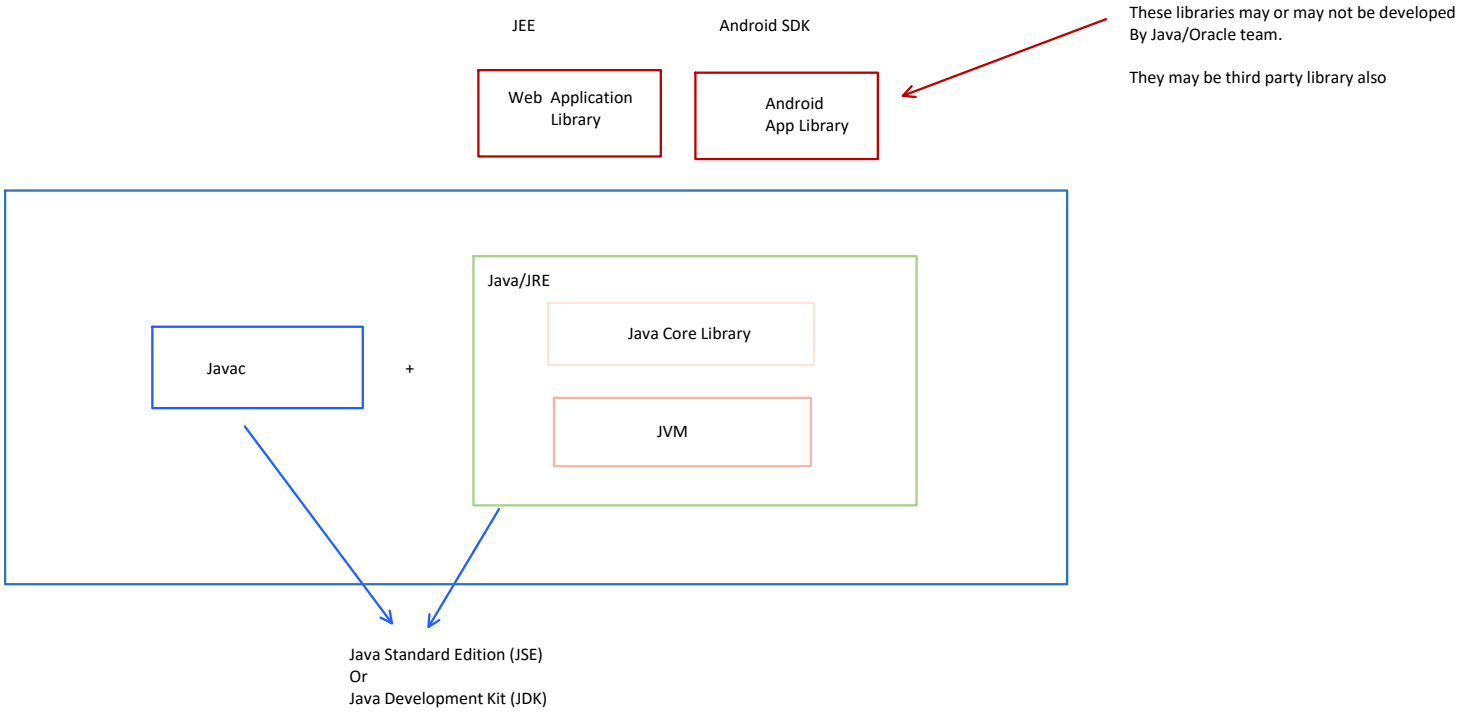
Executes the byte code that is not a java program

3. JRE — Just another word for 'Java Platform'
   a. When you want to run an application on an operating system you need Java installed -> you need JRE installed.
4. JVM — JVM is a component of Java Platform that interprets the entire byte code.

JEE

Java Technology Stack

Monday, 25 May 2020 4:01 PM



Abstract Windowing Toolkit

Monday, 25 May 2020 4:36 PM

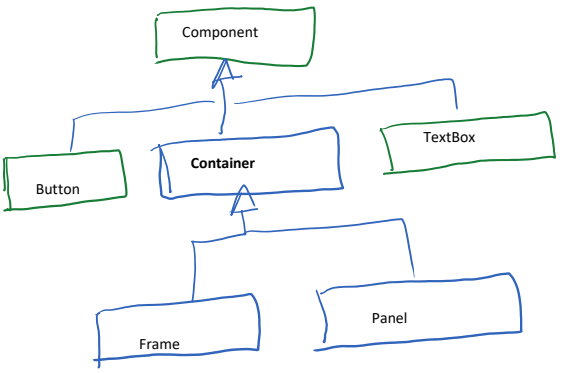
- Original Java Desktop (and Applet) Development Model
- You can develop Java Desktop Application using AWT
- These applications shall be platform Independent
  - On each platform JVM using native OS api to render the UI

AWT Model

AWT Components

- Each UI element such as **Button, TextBox, Checkbox** are individual Objects
- Each of these objects are subtypes of **Component class**

- They are gathered together on a **Frame**
- Frame is also a subclass of **Component**
  - The Exact hierarcy
    - Frame extends **Container** extends Component
- A **Container** is a component that can have childrens
  - Example
    - Frame
    - Panel
    - List
  - A Container can include another container



can be added to a container

outerlevel window
element with visible borders

Invisible container
that can have its
own children

Layout Manager

- AWT doesn't recommend (although allows) absolute positing or sizing of its components
  - The idea is when the UI size is changed the components should be proportionately changed in size and position to make it look good under new size
- This is managed by different Layout managers
- We can associate one (and only one) Layout Manager with a container
- The LayoutManager will decide how a component will be displayed within the container
- The LayoutManager may choose to ignore the dimension and location for individual components

LayoutManager

Monday, May 25, 2020 5:29 PM

- AWT comes with several Layout Managers

1. Flow Layout Manager

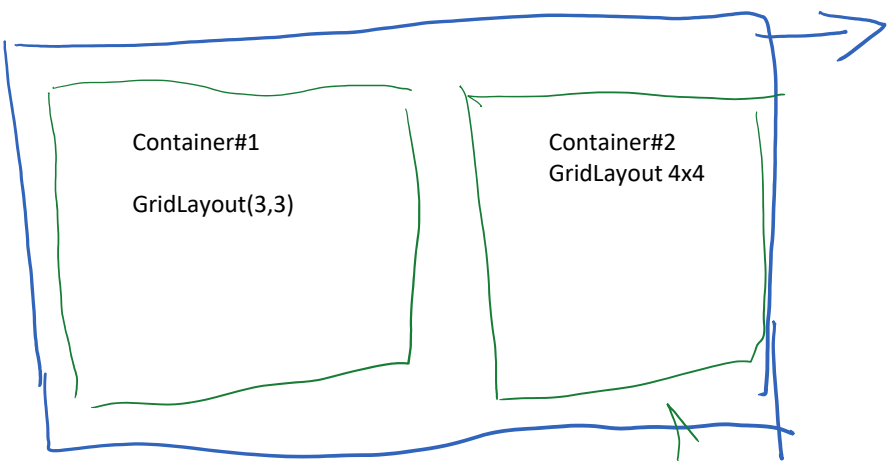
a. Simples of all the layout
b. Positions each component one after another left to right
c. When the line is full horizontally, moves next component to next row
d. The components keep chainging their position as container is resized
e. The Size of the component is not changed.

2. GridLayout

a. divides the container in equal size grids defined by rows and columns
b. Each component is first added row wise
    i. fill each column of row1 , then row 2 etc
c. Resize container will resize all components to maintain optimal space utilization and not breaking the grid formation
d. If the components are more than row X col,
    i. column is ignored,
    ii. number of row is maintained
    iii. number of column is recalculated
e. If number of controls are not a perfect multiple for row x col, some of columns of last row may remain empty.

Working with Multiple Layouts

- A container can have only one layout set at a time
- If you set again the new set call will replace the previous one
- To create complex UI we can add mulitple containers (Panel) over the frame
    - Each container can have its own layout separate from another



Event Handling

Monday, May 25, 2020 5:52 PM

Console Application vs GUI Application

Console Application	GUI Application
1. Program flow is sequential --- It is driven by developer. User defines flow from start to end	1. Program flow is **EventDriven.** Programmer sets the GUI and then what will happen depends on user action -- It is driven by user
2. Program starts with main() and ends with main()	2. Program starts with main where UI is created. End of main is just the beginning of actual program whose lifecycle is decided by user actions
Example **• you want to create a simple calculator** **1. you ask user number1** **2. you ask user number2** **3. you ask user operator (+,-,*,/)** **4. you print result** **5. you ask user if they want a second calculation** **6. end the program if they down want else loop to step 1** **• Problem** **○ You can't change the order of input -- number1, number2, operator** **○ if you want same number1 and number 2 for different operations, you need to enter again and again** **○ If you want to quit after entering number1, you cant unless you enter the other details and reach the prompt on step 5.**	**Example** **• you want to create a simple calculator** **1. Create the UI with two text box and 4 buttons** **2. respond to button clicks** **3. Main ends after intial setup** **• User may choose which number he would enter first and second** **• They can apply multiple operations on the same number** **• They may close application whenever they like** **• There is no nagging prompt if we want ot continue**
*	

Event Handling

- Most important aspect of any GUI application
- How the UI reacts when you interact with it

Import Event Handlng Elements

Event

- When you interact with the GUI, it generates an event
- Interaction may mean
  - clicking a button
  - typing in a text box
  - checking/unchecking a checkbox
  - moving your mouse
  - typing from keyboard
- Each of the interaction generates an Event
- In Awt **Event** is an Object that contains details related to what happened
  - Example
    - TextEvent
      - occurs when you type text in a textbox
      - The event must know these details
        - which text box you typed in (**source of event)**
        - what is typed
    - MouseEvent
      - Occurs when you interact with your mouse
        - MouseMove
        - MouseClick
        - The event must tell you
          - source of the event (component)
          - location where mouse has moved
          - button that is clicked.
  - Each event at least includes the **source**
- **To handle a particular Event** we have Event Listener
- Imortant events
  1. ActionEvent
  2. TextEvent
  3. MouseEvent

Event types

- Events may be **low leve or high level**

Low level event

- something happened with the hardware or divice the purpose is not clear
- Example
  - MouseEvent
  - KeyEvent

High level Event

- Event that gives meaning to action that just happened.
- different low level event may have some common goal
- same low level event may have different goal

4. KeyEvent
5. ComponentEvent
6. ItemEvent
7. WindowEvent

Event Listner

- for every Event **XEvent,** there is a listener interface called **Xlistner**
  - Example
    - for handling **ItemEvent** we have **ItemListener**
    - for handling **ActionEvent** we have **ActionListner**

Component

- A component may produce different events
- All components supports **MouseEvent and ComponentEvent**
- A listbox, checkbox etc also supports **ItemEvent (**when a item is selected or deselected)
- A TextBox also supports **TextEvent** and **ActionEvent**

Handling an Event

- To handle the event of a component we need to take following action
  1. Decide which event you are going to handle say **Xevent**
  2. Create a class that implments **Xlistner**
  3. Create an object of your class
  4. Select the component for which you are handling the event say **comp**
  5. Add the event by calling **addXListner()** method

Some Important Events

Action Event

- defines that user want to perform some action
  - caused by
    - ButtonClick
    - Hitting Enter in TextBox
    - Double Clicking a ListBox Item
- Tex

Text Event

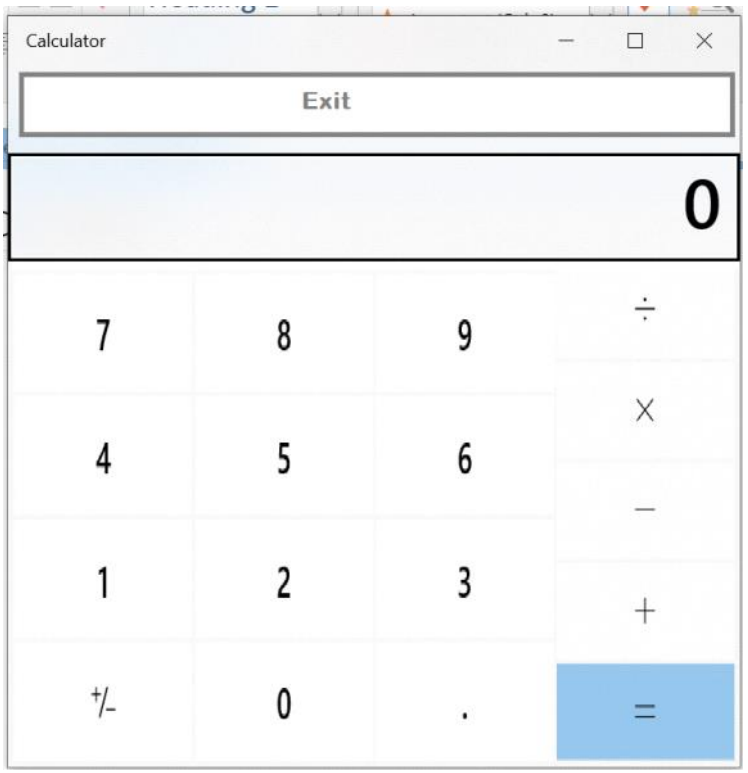
- when text values changes in a **TextField**
  - normal keyboard input causes TextEvent
  - enter key causes ActionEvent

Window Event

- Handles events related to the **Frame**
- The associated interface is **WindowListener**
- WindowListener has 7 methods
  - windowOpened() <— window opens for the first time
  - windowClosed() <— winow has been closed
  - windowActivated <— window gains focus
  - windowDeactivated <— window looses focus
  - windowIconfied<— window minimized
  - windowDeiconfied <— window restored
  - **windowClosing** <— somebody wants to close window (Close Button is clicked or Alt+F4 typed)
    - Window has not closed yet
    - This is a good place to ask if we need to close the window

Assignment -- Create a simple Calculator

Monday, May 25, 2020 7:09 PM

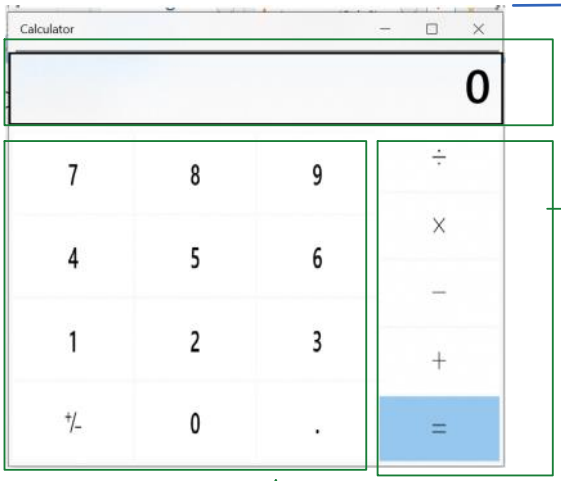


Write the logic to make the calculator work

- Create layout as close as you can get
- Define the event handlers to handle the button clicks
-

Assignment Solution

Tuesday, May 26, 2020 4:20 PM



calculator frame uses Border Layout

Screen to the **North** BorderLayout

A Panel added to the **East** of main Frame
This Pannel has 5 Buttons

What layout should use?

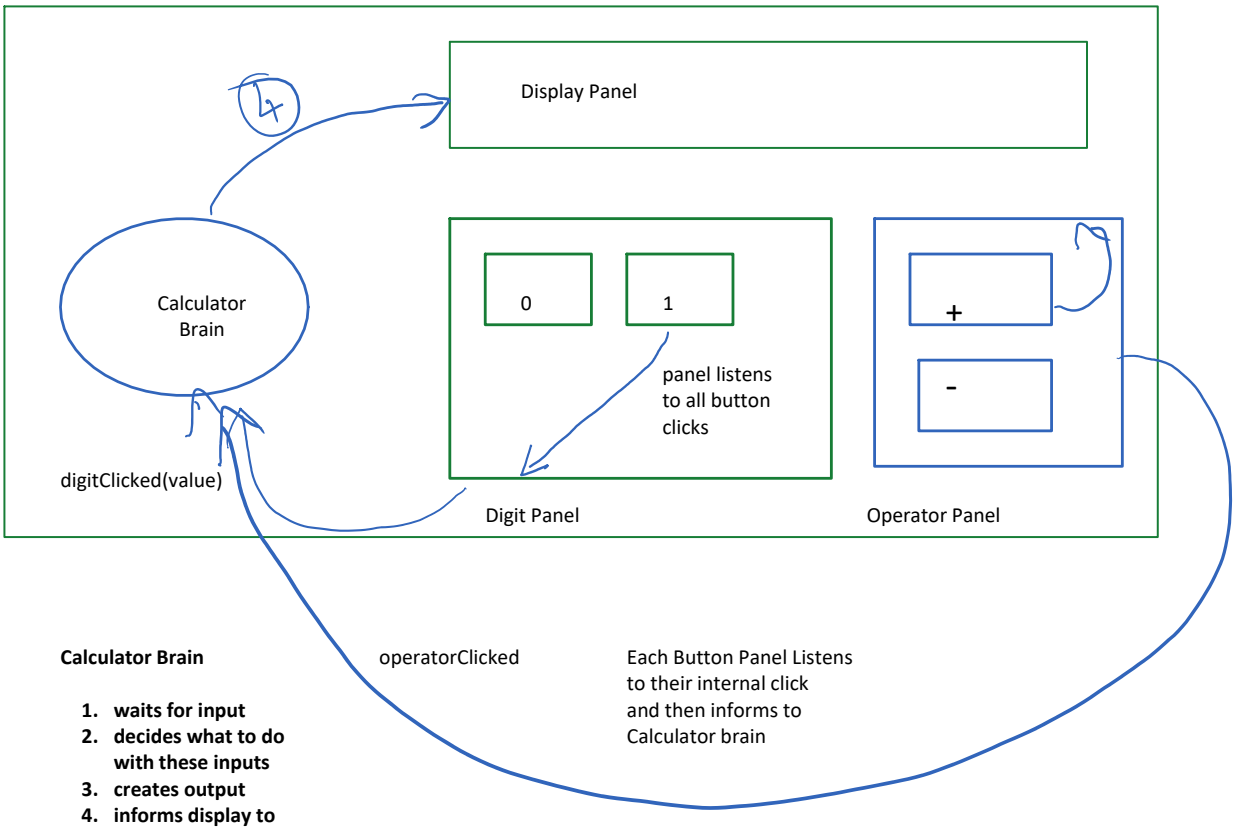
Another Panel with 12 Buttons
The Hight of 4 Buttons in this panel is equal to the Hight 5 buttons in another Panel

Where would you place this panel?

Program Flow

Tuesday, May 26, 2020 6:32 PM

CalculatorWindow

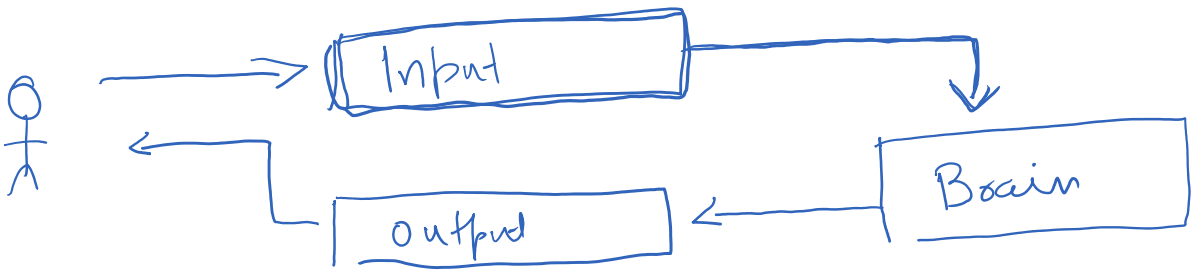


Calculator Brain

1. **waits for input**
2. **decides what to do with these inputs**
3. **creates output**
4. **informs display to display the value**

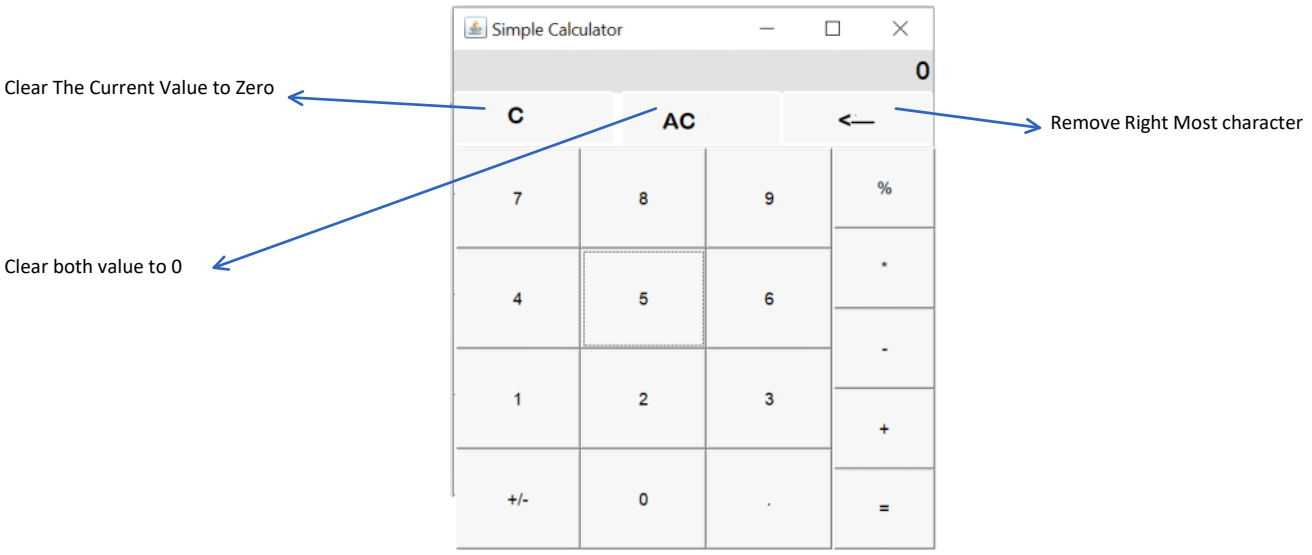
operatorClicked

Each Button Panel Listens to their internal click and then informs to Calculator brain



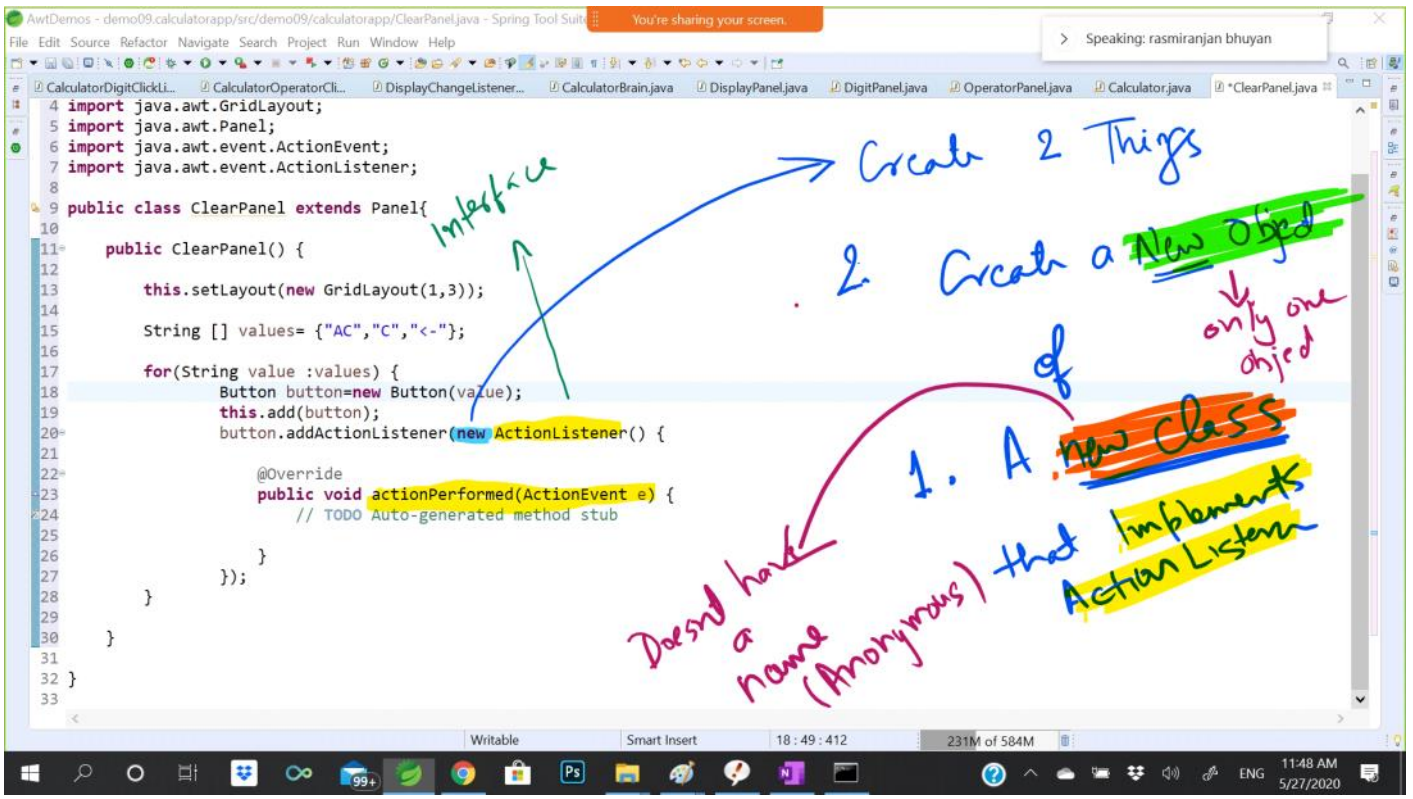
Assignment — New keys

Wednesday, May 27, 2020 10:20 AM



Anonymous Class

Wednesday, May 27, 2020 11:58 AM




```
4  import java.awt.GridLayout;
5  import java.awt.Panel;
6  import java.awt.event.ActionEvent;
7  import java.awt.event.ActionListener;
8
9  public class ClearPanel extends Panel{
10
11     public ClearPanel() {
12
13         this.setLayout(new GridLayout(1,3));
14
15         String [] values= {"AC","C","<-"};
16
17         for(String value :values) {
18                 Button button=new Button(value);
19                 this.add(button);
20                 button.addActionListener(new ActionListener() {
21
22                     @Override
23                     public void actionPerformed(ActionEvent e) {
24                         // TODO Auto-generated method stub
25
26                     }
27                 });
28         }
29
30     }
31
32 }
33
```

Evolution of Java Desktop

Wednesday, May 27, 2020 12:56 PM

1. Java originally shipped with **java.awt** package for desktop development (and browser based applet development)
    a. AWT interacted with underlying UI system of the operating system
    b. Each Awt component is internally translated to a control based on the operating
        i. Its look and feel will vary from one os to another
    c. Because each control must have an equivalent in all operating system AWT supports a very small number of components
        i. You can't have components not defined by all operating system.
    d. AWT is slow in performance.

2. **Java Swing**
    a. Another Java UI Framework
    b. Many concepts are similar in the two
        i. Layoutout
        ii. Event Handling
    c. Java swing doesn't show the native OS components
    d. **It draws all the components**
        i. Think each button and textbox is actually a nice drawing on the screen
    e. Swing is light weight, better performance.
    f. Swing can create more UI components which are not present in all OS
        i. It can even create component not present anywhere
    g. The UI can look the same on all OS

Sun or Oracle the original owner of Java didn't evolve the desktop model beyond their original attempt (almost 2 decades ago)

The core team focused on web and distributed development rather than desktop development

Third party organization developed their own library to create GUI applications.

SWT (Standard Window Toolkit)

Wednesday, May 27, 2020 1:15 PM

- It is a desktop application development library from **Eclipse**
- It is not from **Core Java Team.**
- Eclipse uses this library to develop the **Eclipse IDE**
- Eclipse has released it SWT library in public domain so that you can use this library to develop your own desktop application.
- You need to download SWT library to use in your project.
- You can develop desktop application with SWT
- Just like AWT, SWT also maps its UI to native OS components
- But for some advanced components SWT also draws them like SWING making them avaialble to all operating systems
- There is a huge number of components available in SWT (much more than AWT/SWING)

SWT vs AWT

- Both are desktop development models
- Both have several common concepts
    - Button, TextBox etc
    - Layout
    - Event Handling
    - Both tries to create
- SWT and AWT differs in following ways
    - Both have different components
        - AWT Button and SWT Button is not same
        - AWT Layout and SWT Layout is not same
        - They have different Design Models
    - SWT may be a bit more complex for begineers

Important SWT Elements

Wednesday, May 27, 2020 1:23 PM

1. Widget a.k.a Control

- SWT calls its GUI elements (Button, Label etc) as **widgets**.
- All **widgets** are sub class or **Control** (Awt uses **Component** as the super class)
    - we can use the term **widget** and **Control** interchangabley
        - This is simlar to **java.awt.Component**

2. Shell

- **Shell** is equivalent to **Frame** of AWT
- It represents the MainWindow

3. Display

- Display is an invible element which acts a controller for all Controls and widgets
- It helps you
    - handle events
    - draw on the widgets
    - handle color font etc

Bare minimum SWT Application requires

1. Shell
2. Display