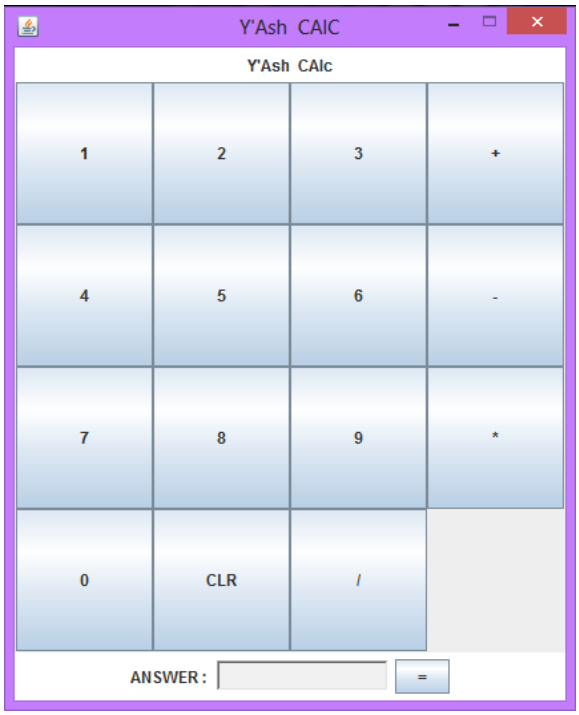
Chapter 1

INTRODUCTION

The goal of the Simple Calculator is to Make a Small JAVA program for Simple Mathematical Calculations.



The Entire Application which is a Simple Calculator is Developed, Designed and coded using Net beans 7.0.1. Most of the mathematical functions have been done using the function in math class in java.

There Are different Buttons included as number integer 0 to 9 ,And one clear button also will be there ,there Are four buttons added for a simple mathematical calculations like Addition, Subtraction, Multiplication, Divide operation,And one Equal To button is included for finding the answer and for displaying it on the Text-area.

Chapter 2

System Analysis

2.1 Package java.awt.* :

Contains all of the classes for creating user interfaces and for painting graphics and images.

Interface Summary

ActiveEvent- An interface for events that know how to dispatch themselves.

Adjustable- The interface for objects which have an adjustable numeric value contained within a bounded range of values.

Composite- The Composite interface, along with CompositeContext, defines the methods to compose a draw primitive with the underlying graphics area.

CompositeContext- The CompositeContext interface defines the encapsulated and optimized environment for a compositing operation.

ItemSelectable-The interface for objects which contain a set of items for which zero or more can be selected.

KeyEventDispatcher-A KeyEventDispatcher cooperates with the current KeyboardFocusManager in the targeting and dispatching of all KeyEvents.

KeyEventPostProcessor- A KeyEventPostProcessor cooperates with the current KeyboardFocusManager in the final resolution of all unconsumed KeyEvents.

LayoutManager- Defines the interface for classes that know how to lay out Containers.

LayoutManager2- Defines an interface for classes that know how to layout Containers based on a layout constraints object.

MenuContainer- The super class of all menu related containers.

Paint- This Paint interface defines how color patterns can be generated for Graphics2D operations.

PaintContext-The PaintContext interface defines the encapsulated and optimized environment to generate color patterns in device space for fill or stroke operations on a Graphics2D.

PrintGraphics- An abstract class which provides a print graphics context for a page.

Shape-The Shape interface provides definitions for objects that represent some form of geometric shape.

Stroke- The Stroke interface allows a Graphics2D object to obtain a Shape that is the decorated outline, or stylistic representation of the outline, of the specified Shape.

Transparency- The Transparency interface defines the common transparency modes for implementing classes.

2.2 Package java.awt.event.* :

Provides interfaces and classes for dealing with different types of events fired by AWT components.

Interface Summary

ActionListener-- The listener interface for receiving action events.

AdjustmentListener-- The listener interface for receiving adjustment events.

AWTEventListener-- The listener interface for receiving notification of events dispatched to objects that are instances of Component or MenuComponent or their subclasses.

ComponentListener-- The listener interface for receiving component events.

ContainerListener-- The listener interface for receiving container events.

FocusListener-- The listener interface for receiving keyboard focus events on a component.

HierarchyBoundsListener-- The listener interface for receiving ancestor moved and resized events.

HierarchyListener-- The listener interface for receiving hierarchy changed events.

InputMethodListener-- The listener interface for receiving input method events.

ItemListener-- The listener interface for receiving item events.

KeyListener--The listener interface for receiving keyboard events (keystrokes).

MouseListener-- The listener interface for receiving "interesting" mouse events (press,release, click, enter, and exit) on a component.

MouseMotionListener-- The listener interface for receiving mouse motion events on a component.

MouseWheelListener-- The listener interface for receiving mouse wheel events on a component.

TextListener-- The listener interface for receiving text events.

WindowFocusListener-- The listener interface for receiving WindowEvents, including WINDOW_GAINED_FOCUS and WINDOW_LOST_FOCUS events.

WindowListener-- The listener interface for receiving window events.

WindowStateListener-- The listener interface for receiving window state events.

2.3 Package javax.swing.* :

Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

Interface Summary

AncestorListener - AncestorListener Interface to support notification when changes occur to a JComponent or one of its ancestors.

CaretListener - Listener for changes in the caret position of a text component.

CellEditorListener- CellEditorListener defines the interface for an object that listens to changes in a CellEditor

ChangeListener - Defines an object which listens for ChangeEvents.

DocumentEvent - Interface for document change notifications.

MenuDragMouseListener -Defines a menu mouse-drag listener.

MenuKeyListener - MenuKeyListener

MenuListener - Defines a listener for menu events.

MouseInputListener- A listener implementing all the methods in both the MouseListener and MouseMotionListener interfaces.

Chapter 3

Design

3.0 ActionListener

Action listeners are probably the easiest — and most common — event handlers to implement. You implement an action listener to define what should be done when an user performs certain operation.To write an Action Listener, follow the steps given below:

Declare an event handler class and specify that the class either implements an ActionListener interface or extends a class that implements an ActionListener interface. For example:

public class MyClass implements ActionListener {

Register an instance of the event handler class as a listener on one or more components. For example:

someComponent.addActionListener(instanceOfMyClass);

Include code that implements the methods in listener interface. For example:

public void actionPerformed(ActionEvent e) {

...//code that reacts to the action... }

In general, to detect when the user clicks an onscreen button (or does the keyboard equivalent), a program must have an object that implements the ActionListener interface. The program must register this object as an action listener on the button (the event source), using the addActionListener method. When the user clicks the onscreen button, the button fires an action event. This results in the invocation of the action listener's actionPerformed method (the only method in the ActionListener interface). The single argument to the method is an ActionEvent object that gives information about the event and its source.

The Action Listener API

The ActionListener Interface

Because ActionListener has only one method, it has no corresponding adapter class.

Method	Purpose
actionPerformed(actionEvent)	Called just after the user performs an action.

The ActionEvent Class

Method	Purpose
String getActionCommand()	Returns the string associated with this action. Most objects that can fire action events support a method called setActionCommand that lets you set this string.
int getModifiers()	Returns an integer representing the modifier keys the user was pressing when the action event occurred. You can use the ActionEvent-defined constants SHIFT_MASK, CTRL_MASK, META_MASK, and ALT_MASK to determine which keys were pressed. For example, if the user Shift-selects a menu item, then the following expression is nonzero: actionEvent.getModifiers() & ActionEvent.SHIFT_MASK
Object getSource() (*in java.util.EventObject*)	Returns the object that fired the event.

3.1 Frames

A Frame is a top-level window with a title and a border. The size of the frame includes any area designated for the border. The dimensions of the border area may be obtained using the getInsets method. Since the border area is included in the overall size of the frame, the border effectively obscures a portion of the frame, constraining the area available for rendering and/or displaying subcomponents to the rectangle which has an upper-left corner location of (insets.left, insets.top), and has a size of width - (insets.left + insets.right) by height - (insets.top + insets.bottom).

A frame, implemented as an instance of the JFrame class, is a window that has decorations such as a border, a title, and supports button components that close or iconify the window. Applications with a GUI usually include at least one frame. Applets sometimes use frames, as well.

To make a window that is dependent on another window — disappearing when the other window is iconified, for example — use a dialog instead of frame.. To make a window that appears within another window, use an internal frame.

Chapter 4
System Design

4.0 Project Objectives

Analysis :

Our project should have the ability to:
_ Interface with a graphic user interface.
_ Communication between the GUI and the Calculator functions
_ Accurate algorithms of calculator functions
_ A storage class for elements provided by the GUI
_ Elements to be used by the calculator and its functions.

4.1 Calculator

The calculator itself consists of two objects, the stack and the calculator object. The stack stores the Elements, and the calculator object has functions which cause it to perform operations to the top numbers on the stack, or to add numbers to the stack, or similar things. This implements a simple calculator, which can be called with simple commands like \Add".

4.2 GUI

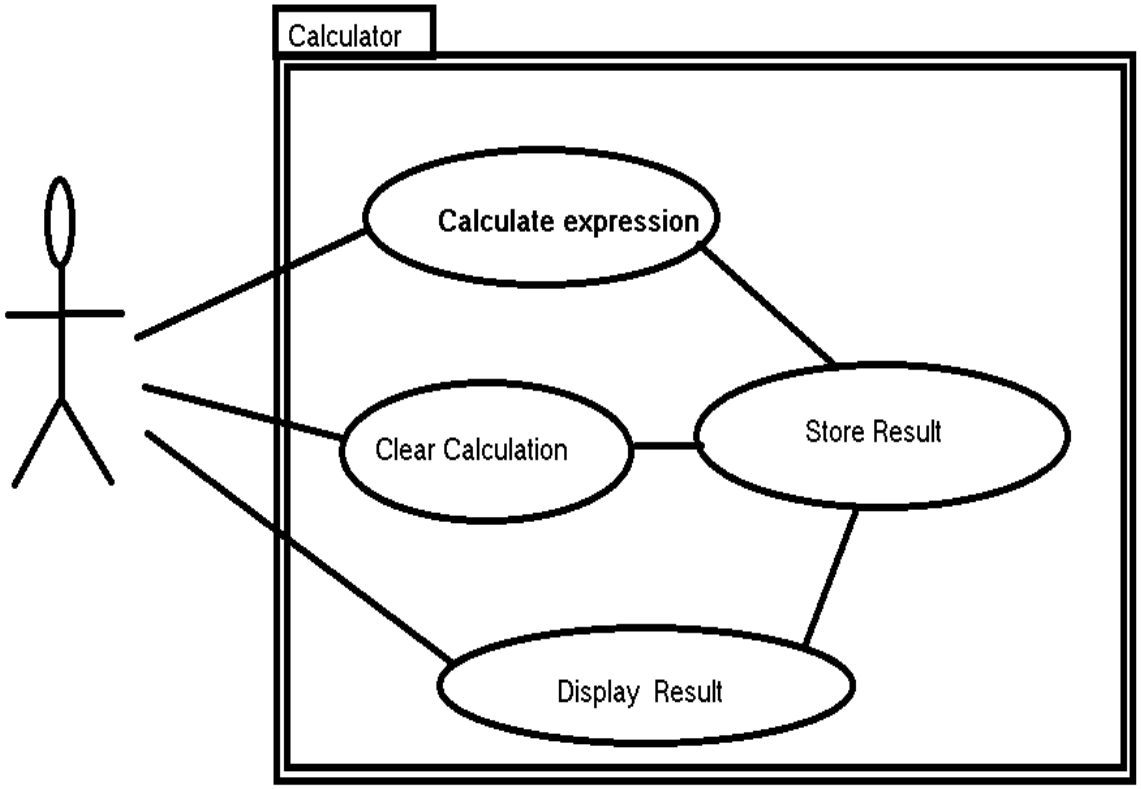
The graphical user interface consists of 2 classes. The _rst is a generic GUI that we can create from existing code. This uses a second class which converts the characters sent by the GUI into commands for the calculator, and updates the display on the GUI. The reason for separating this into 2 classes is that it minimizes the amount of modi_cation needed to existing code, allows for multiple programmers to more easily work on the separate parts, and allows for simple changes to the graphical user interface.

4.3 User Interface

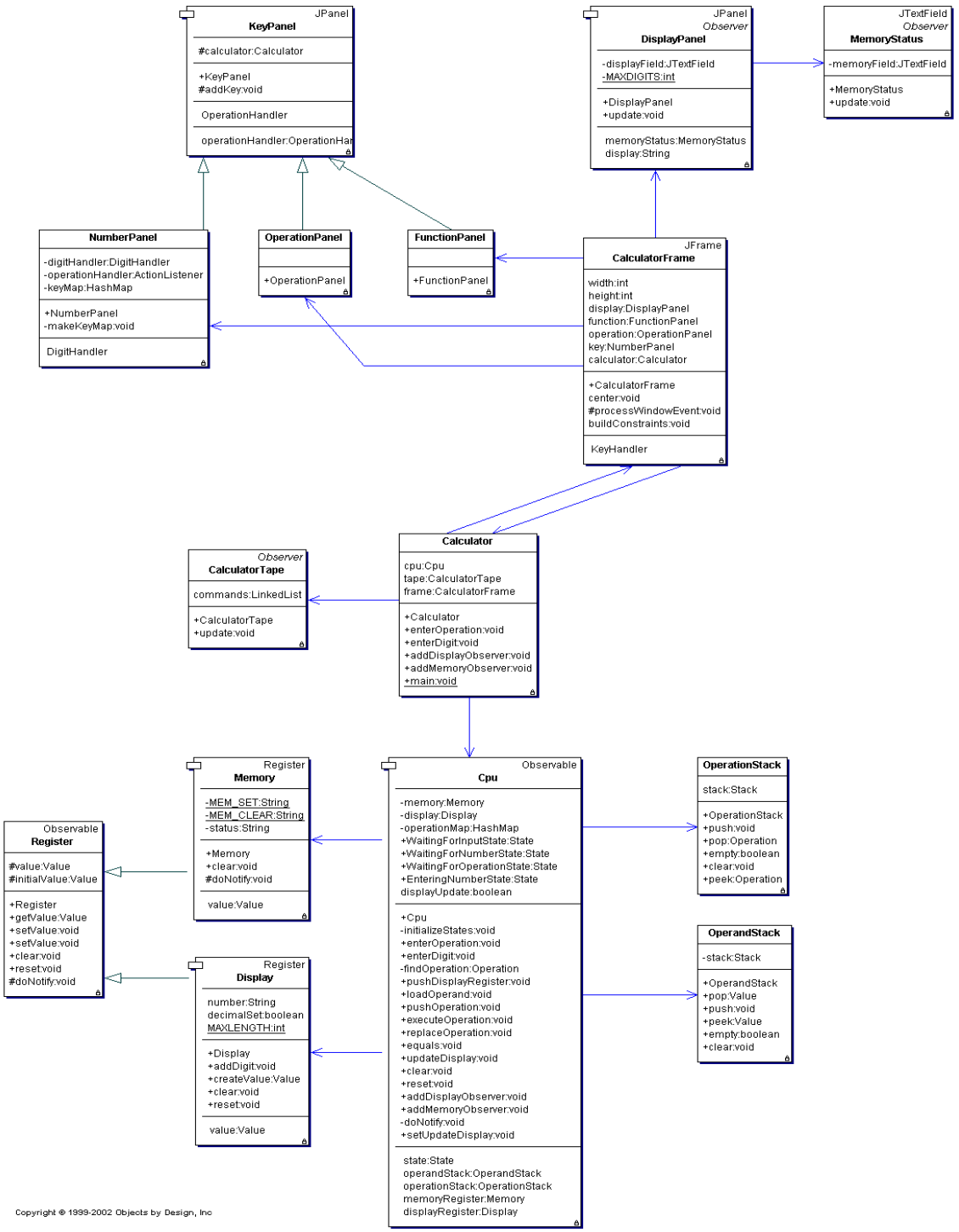
The user interface is a GUI built with the Tk Toolkit, emulating a typical four-function calculator, but with an added stack display and the ability to display operands other than simple integers and oats. The Tk Toolkit was chosen over other graphics packages mainly because it is the standard graphics package distributed with Python, the language chosen for this project, and it is well-documented in that context. It is also less difficult to learn and use than the Qt Toolkit, and more compact and efficient than Java Swing. Tk has been around for a very long time, and it is considered ugly by many who prefer the perfectly sculpted contours of more modern packages, but it is quite adequate for our needs – basic widgets such as buttons and data entry boxes. We decided to minimize the amount of "circuitry" in the display module, and make it basically a passive display, with the ability to capture mouse clicks and send a code indicating which button was pressed. There are two categories of button - data and operations. Typical user interactions are discussed in the analysis section. When the user presses a data key, all that happens is that the display (bottom register) is refreshed with the new key appended to any existing data. When an operation key is pressed, the Controller pops the necessary operands o_ the stack, performs the operation, and refreshes all registers in the display. We also considered an architecture in which the Display module maintained the stack, and made the Controller much simpler. We decided to go with the current design, however, so it would be easier to change the Display if we later add other data types and operations.

Use Case Diagram

Simple Calculator

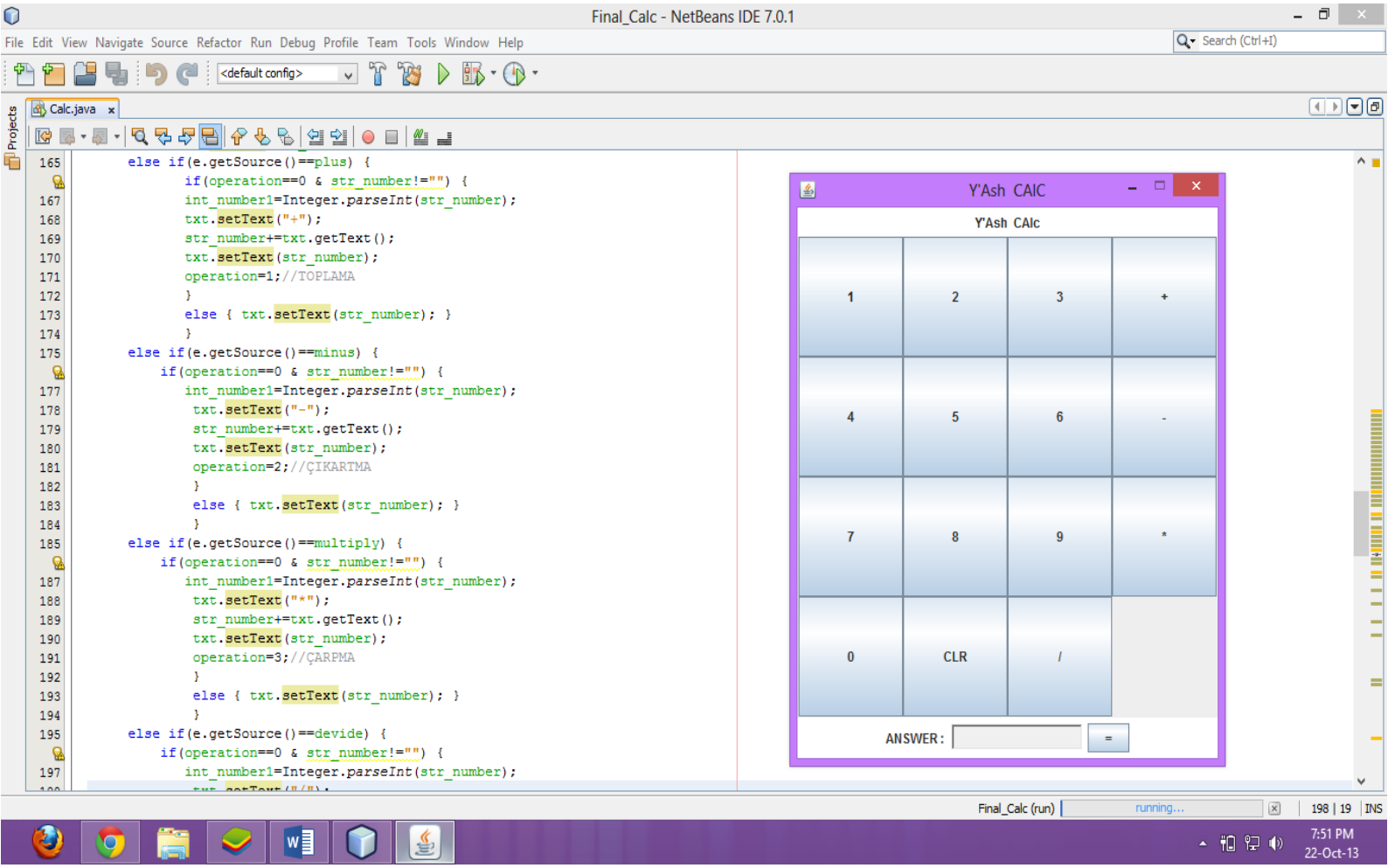


Class Diagram



Chapter 5

Snap Shot:



Source Code main Logical Part That implemented In Net Beans 7.0.1.

Source Code

```
else if(e.getSource()==plus) {
        if(operation==0 & str_number!="") {
        int_number1=Integer.parseInt(str_number);
        txt.setText("+");
        str_number+=txt.getText();
        txt.setText(str_number);
        operation=1;
        }
        else { txt.setText(str_number); }
        }
   else if(e.getSource()==minus) {
      if(operation==0 & str_number!="") {
        int_number1=Integer.parseInt(str_number);
         txt.setText("-");
```

```
            str_number+=txt.getText();
            txt.setText(str_number);
            operation=2;
            }
            else { txt.setText(str_number); }
            }
    else if(e.getSource()==multiply) {
       if(operation==0 & str_number!="") {
          int_number1=Integer.parseInt(str_number);
            txt.setText("*");
            str_number+=txt.getText();
            txt.setText(str_number);
            operation=3;
            }
            else { txt.setText(str_number); }
            }
    else if(e.getSource()==devide) {
       if(operation==0 & str_number!="") {
          int_number1=Integer.parseInt(str_number);
            txt.setText("/");
            str_number+=txt.getText();
            txt.setText(str_number);
            operation=4;
            }
            else { txt.setText(str_number); }
            }
```

5.1 JAR File

The basic format of the command for creating a JAR file is:

jar cf *jar-file input-file(s)*

The options and arguments used in this command are:

- The c option indicates that you want to *create* a JAR file.
- The f option indicates that you want the output to go to a *file* rather than to stdout.
- jar-file is the name that you want the resulting JAR file to have. You can use any filename for a JAR file. By convention, JAR filenames are given a .jar extension, though this is not required.
- The input-file(s) argument is a space-separated list of one or more files that you want to include in your JAR file. The input-file(s) argument can contain the wildcard * symbol. If any of the "input-files" are directories, the contents of those directories are added to the JAR archive recursively.

The c and f options can appear in either order, but there must not be any space between them.

Chapter 6

Conclusion

This Project covers the complete, chronological process of developing a basic Simple Calculator as a Java Applet. By using java applet we created this user friendly Program which is Making simple arithmetical operations . The Calculator is created in java applet which can run on any Type of Operating System use java is platform independent programming language.

Chapter 7

References

i) **www.docs.oracle.com**
ii) **Wikipedia**
iii) **The Complete Reference : JAVA**
iv) **www.techuser.net/calcsam.html**
v) **wwwcsstudents.stanford.edu/~jl/Essays/simplecalc.html**