Government College of Engineering, Jalgaon
(An Autonomous Institute of Government of Maharashtra)

Name : **PRN :**
Subject : CO310U (Application programming Lab) **Sem :** V(Odd)
Class : T.Y. B.Tech **Academic Year :** 2024-25
Date of Performance : **Date of Completion :**

Practical No : 16

Aim: Write a program to draw a form using GUI components to accept details from a customer for a bank.

Required Software: OpenJDK version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)

Java Compiler Version - JAVAC 1.8.0_131

Theory:

The AWT contains numerous classes and methods that allow you to create and manage windows. It is also the foundation upon which Swing is built. In this experiment, you will learn how to create and manage windows, manage fonts, output text, and utilize graphics, pushbuttons, supported by the AWT. It also explains further aspects of Java's event handling mechanism..Although a common use of the AWT is in applets, it is also used to create stand-alone windows that run in a GUI environment, such as Windows.

AWT Classes

The AWT classes are contained in the java.awt package. It is one of Java's largest packages.Fortunately, because it is logically organized in a top-down, hierarchical fashion, it is easier to understand and use than you might at first believe. Figure shows JAVA AWt hierarchy.

Container

The Container is a component in AWT that can contain other components like buttons, text fields, labels etc. The classes that extend Container class are known as containers such as Frame, Dialog and Panel.

Window

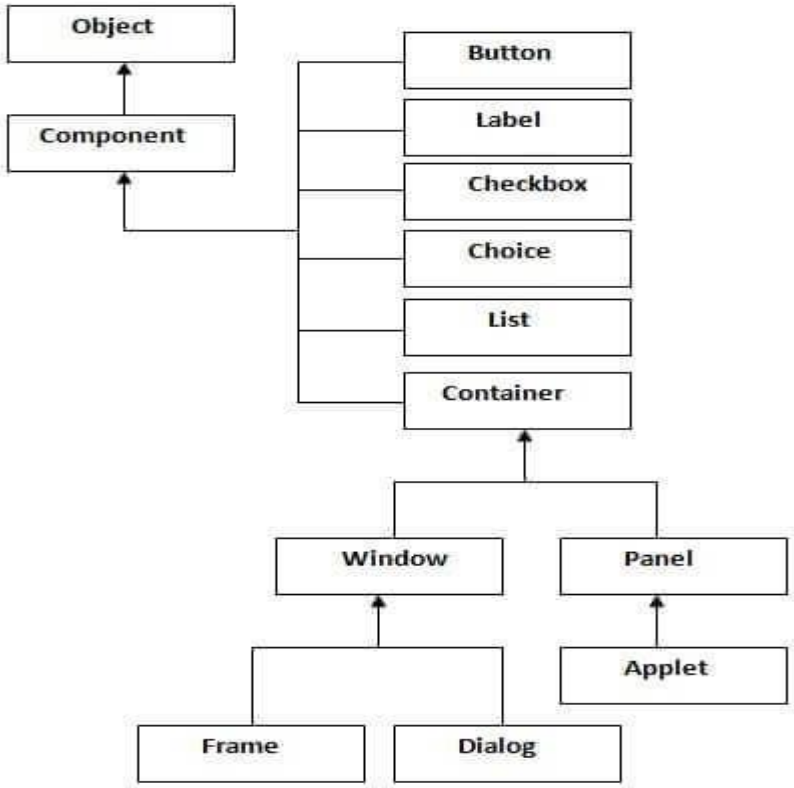
The window is the container that has no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain a title bar and menu bars. It can have other components like buttons, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc



Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Window Fundamentals

The AWT defines windows according to a class hierarchy that adds functionality and specificity with each level. The two most common windows are those derived from Panel, which is used by applets, and those derived from Frame, which creates a standard application window. Much of the functionality of these windows is derived from their parent classes. Thus, a description of the class hierarchies relating to these two classes is fundamental to their understanding. Below figure shows the class hierarchy for Panel and

Frame. Let's look at each of these classes now.

Component

At the top of the AWT hierarchy is the Component class. Component is an abstract class that encapsulates all of the attributes of a visual component. All user interface elements that are displayed on the screen and that interact with the user are subclasses of Component. It defines over a hundred public methods that are responsible for managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting. A Component object is responsible for remembering the current foreground and background colors and the currently selected text font.

Container

The Container class is a subclass of Component. It has additional methods that allow other Component objects to be nested within it. Other Container objects can be stored inside of a Container (since they are themselves instances of Component). This makes for a multileveled containment system. A container is responsible for laying out (that is, positioning) any components that it contains..

Panel

The Panel class is a concrete subclass of Container. It doesn't add any new methods; it simply implements Container. A Panel may be thought of as a recursively nestable, concrete screen component. Panel is the superclass for Applet. When screen output is directed to an applet, it is drawn on the surface of a Panel object. In essence, a Panel is a window that does notcontain a title bar, menu bar, or border. This is why you don't see these items when an appletis run inside a browser. When you run an applet using an applet viewer, the applet viewerprovides the title and border.Other components can be added to a Panel object by its add( ) method (inherited fromContainer). Once these components have been added, you can position and resize themmanually using the setLocation( ), setSize( ), setPreferredSize( ), or setBounds( ) methods defined by Component.

Window

TheWindow class creates a top-level window. Atop-level window is not contained within any other object; it sits directly on the desktop. Generally, you won't create Window objects directly. Instead, you will use a subclass of Window called Frame, described next.

Frame

Frame encapsulates what is commonly thought of as a "window." It is a subclass ofWindowand has a title bar, menu bar, borders, and resizing corners. If you create a Frame object from within an applet, it will contain a warning message, such as "Java Applet Window," to the user that an applet window has been created. This message warns users that the window they see was started by an applet and not by software running on

their computer . When a Frame window is created by a stand-alone application rather than an applet, a normal window is created.

Canvas

Although it is not part of the hierarchy for applet or frame windows, there is one other type of window that you will find valuable: Canvas. Canvas encapsulates a blank window upon which you can draw.

Working with Frame Windows

After the applet, the type of window you will most often create is derived from Frame. You will use it to create child windows within applets, and top-level or child windows for stand-alone applications. As mentioned, it creates a standard-style window.

Here are two of Frame's constructors:

-Frame( )

-Frame(String title)

The first form creates a standard window that does not contain a title. The second form creates a window with the title specified by title. Notice that you cannot specify the dimensions of the window. Instead, you must set the size of the window after it has been created. There are several key methods you will use when working with Frame windows. They are examined here

Setting the Window's Dimensions

The setSize( ) method is used to set the dimensions of the window. Its signature is shown here:

void setSize(int newWidth, int newHeight)

void setSize(Dimension newSize)

The new size of the window is specified by newWidth and newHeight, or by the width and height fields of the Dimension object passed in newSize. The dimensions are specified in terms of pixels. The getSize( ) method is used to obtain the current size of a window. Its signature is shown here: **Dimension getSize( ) -**This method returns the current size of the window contained within the width and height fields of a Dimension object**.**

Hiding a Window

After a frame window has been created, it will not be visible until you call setVisible( ). Its signature is shown here: void setVisible(boolean visibleFlag).The component is visible if the argument to this method is true. Otherwise, it is hidden.

Setting a Window's Title

You can change the title in a frame window using setTitle( ), which has this general form:
void setTitle(String newTitle) Here, newTitle is the new title for the window.

Closing a Frame Window

When using a frame window, your program must remove that window from the screen when it is closed, by calling setVisible(false). To intercept a window-close event, you must implement the windowClosing( ) method of the WindowListener interface. Inside windowClosing( ), you must remove the window from the screen. The example in the next section illustrates this technique.

Control Fundamentals

The AWT supports the following types of controls:

- Labels
- Push buttons
- Check boxes
- Choice lists
- Lists
- Scroll bars
- Text editing

These controls are subclasses of Component

Adding and Removing Controls

To include a control in a window, you must add it to the window. To do this, you must first create an instance of the desired control and then add it to a window by calling add( ), which is defined by Container. The add( ) method has several forms. The following form is the one that is used for the first part of this chapter:

Component add(Component compObj)

Here, compObj is an instance of the control that you want to add. A reference to compObj is returned. Once a control has been added, it will automatically be visible whenever its parent window is displayed. Sometimes you will want to remove a control from a window when the control is no longer needed. To do this, call remove( ). This method is also defined by Container. It has this general form:

void remove(Component obj)

Here, obj is a reference to the control you want to remove. You can remove all controls by calling removeAll( ).

Responding to Controls

Except for labels, which are passive, all controls generate events when they are accessed by the user. For example, when the user clicks on a push button, an event is sent that identifies the push button. In general, your program simply implements the appropriate interface and then registers an event listener for each control that you need to monitor. Once a listener has been installed, events are automatically sent to it

The HeadlessException

Most of the AWT controls described in this practical now have constructors that can throw a HeadlessException when an attempt is made to instantiate a GUI component in a non-interactive environment (such as one in which no display, mouse, or keyboard is present). The HeadlessException was added by Java 1.4. You can use this exception to write code that can adapt to non-interactive environments. (Of course, this is not always possible.)

Using Buttons

Perhaps the most widely used control is the push button. A push button is a component that contains a label and that generates an event when it is pressed. Push buttons are objects of type Button. Button defines these two constructors:

Button( ) throws HeadlessException

Button(String str) throws HeadlessException

The first version creates an empty button. The second creates a button that contains str as a label. After a button has been created, you can set its label by calling setLabel( ). You can retrieve its label by calling getLabel( ). These methods are as follows:

void setLabel(String str)

String getLabel( ) Here, str becomes the new label for the button

Handling Buttons

Each time a button is pressed, an action event is generated. This is sent to any listeners that previously registered an interest in receiving action event notifications from that component. Each listener implements the ActionListener interface. That interface defines the actionPerformed( ) method, which is called when an event occurs. An ActionEvent object is supplied as the argument to this method. It contains both a reference to the button that generated the event and a reference to the action command string associated with the button.
By default, the action command string is the label of the button. Usually, either the button reference or the action command string can be used to identify the button. Here is an example that creates three buttons labeled "Yes", "No", and "Undecided"

Each time one is pressed, a message is displayed that reports which button has been pressed. In this version, the action command of the button (which, by default, is its label) is used to determine which button has been pressed. The label is obtained by calling the getActionCommand( ) method on the ActionEvent object passed to actionPerformed( ).

```
/ Demonstrate Buttons
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ButtonDemo" width=250 height=150>
</applet> */
public class ButtonDemo extends Applet implements ActionListener {
String msg = "";
Button yes, no, maybe;
public void init() {
yes = new Button("Yes");
no = new Button("No");
maybe = new Button("Undecided");
add(yes);
add(no);
add(maybe);
yes.addActionListener(this);
no.addActionListener(this);
maybe.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
String str = ae.getActionCommand();
if(str.equals("Yes"))
 {
msg = "You pressed Yes.";
}
else if(str.equals("No"))
 {
```

```
msg = "You pressed No.";
}
else {
msg = "You pressed Undecided.";
}
repaint();
}
public void paint(Graphics g) {
g.drawString(msg, 6, 100);
}
}
```

Sample output from the ButtonDemo program is shown below

As mentioned, in addition to comparing button action command strings, you can also determine which button has been pressed, by comparing the object obtained from the getSource( ) method to the button objects that you added to the window. To do this, you must keep a list of the objects when they are added.

Conclusion:

Name & sign of Teacher

Program :

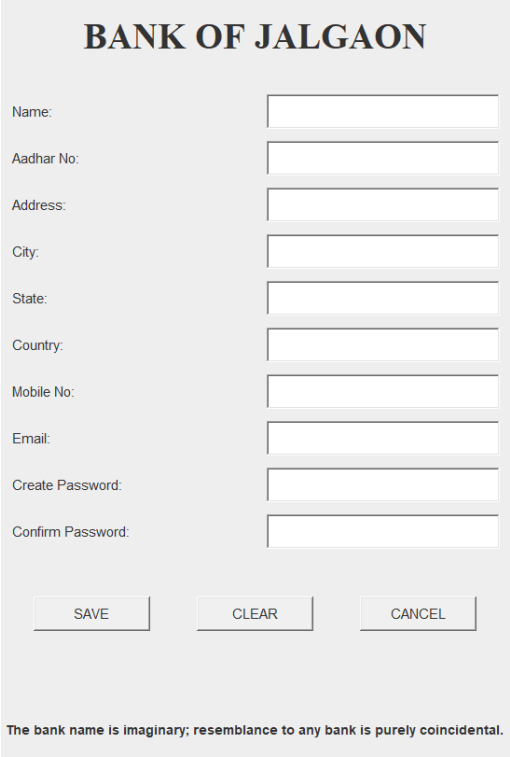
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class BankForm extends JFrame implements ActionListener {

    Label title = new Label("BANK OF JALGAON");
    Label name = new Label("Name:");
    TextField nameField = new TextField();
    Label aadhar = new Label("Aadhar No:");
    TextField aadharField = new TextField();
    Label address = new Label("Address:");
    TextField addressField = new TextField();
    Label city = new Label("City:");
    TextField cityField = new TextField();
    Label state = new Label("State:");
    TextField stateField = new TextField();
    Label country = new Label("Country:");
    TextField countryField = new TextField();
    Label mob = new Label("Mobile No:");
    TextField mobField = new TextField();
    Label email = new Label("Email:");
    TextField emailField = new TextField();
    Label password = new Label("Create Password:");
    TextField passwordField = new TextField();
    Label confirmPassword = new Label("Confirm Password:");
    TextField confirmPasswordField = new TextField();
    Label info = new Label("The bank name is imaginary; resemblance to any bank is purely coincidental.");

    Button saveButton = new Button("SAVE");
    Button clearButton = new Button("CLEAR");
    Button cancelButton = new Button("CANCEL");

    public BankForm() {
        setTitle("Bank Form");
        setSize(1000, 700);
        setLayout(null);

        title.setBounds(310, 50, 380, 40);
        title.setAlignment(Label.CENTER);
        title.setFont(new Font("Serif", Font.BOLD, 30));
        add(title);
```

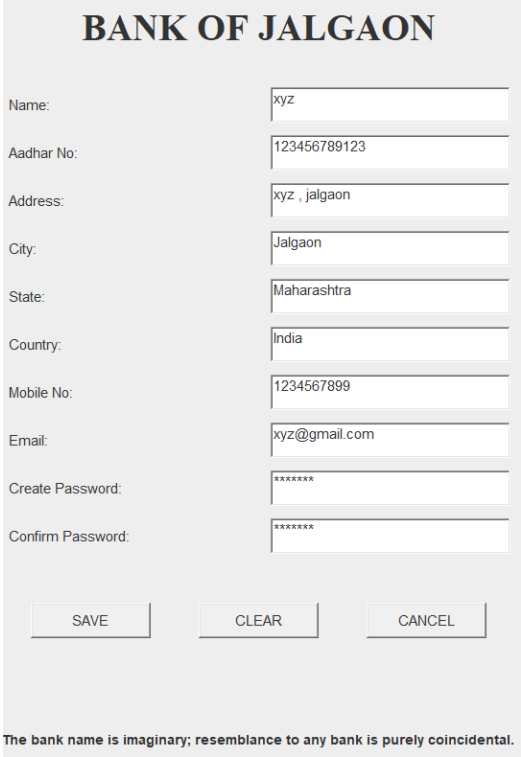
```
    addComponent(name, nameField, 120);
    addComponent(aadhar, aadharField, 160);
    addComponent(address, addressField, 200);
    addComponent(city, cityField, 240);
    addComponent(state, stateField, 280);
    addComponent(country, countryField, 320);
    addComponent(mob, mobField, 360);
    addComponent(email, emailField, 400);

    password.setBounds(290, 440, 200, 30);
    passwordField.setBounds(510, 440, 200, 30);
    passwordField.setEchoChar('*');
    add(password);
    add(passwordField);

    confirmPassword.setBounds(290, 480, 200, 30);
    confirmPasswordField.setBounds(510, 480, 200, 30);
    confirmPasswordField.setEchoChar('*');
    add(confirmPassword);
    add(confirmPasswordField);

    info.setBounds(150, 650, 700, 30);
    info.setAlignment(Label.CENTER);
    info.setFont(new Font("Arial", Font.BOLD, 11));
    add(info);

    addButton(saveButton, 310);
    addButton(clearButton, 450);
    addButton(cancelButton, 590);

    saveButton.addActionListener(this);
    clearButton.addActionListener(this);
    cancelButton.addActionListener(this);

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

private void addComponent(Label label, TextField textField, int yPos) {
    label.setBounds(290, yPos, 200, 30);
    textField.setBounds(510, yPos, 200, 30);
    add(label);
    add(textField);
}
```

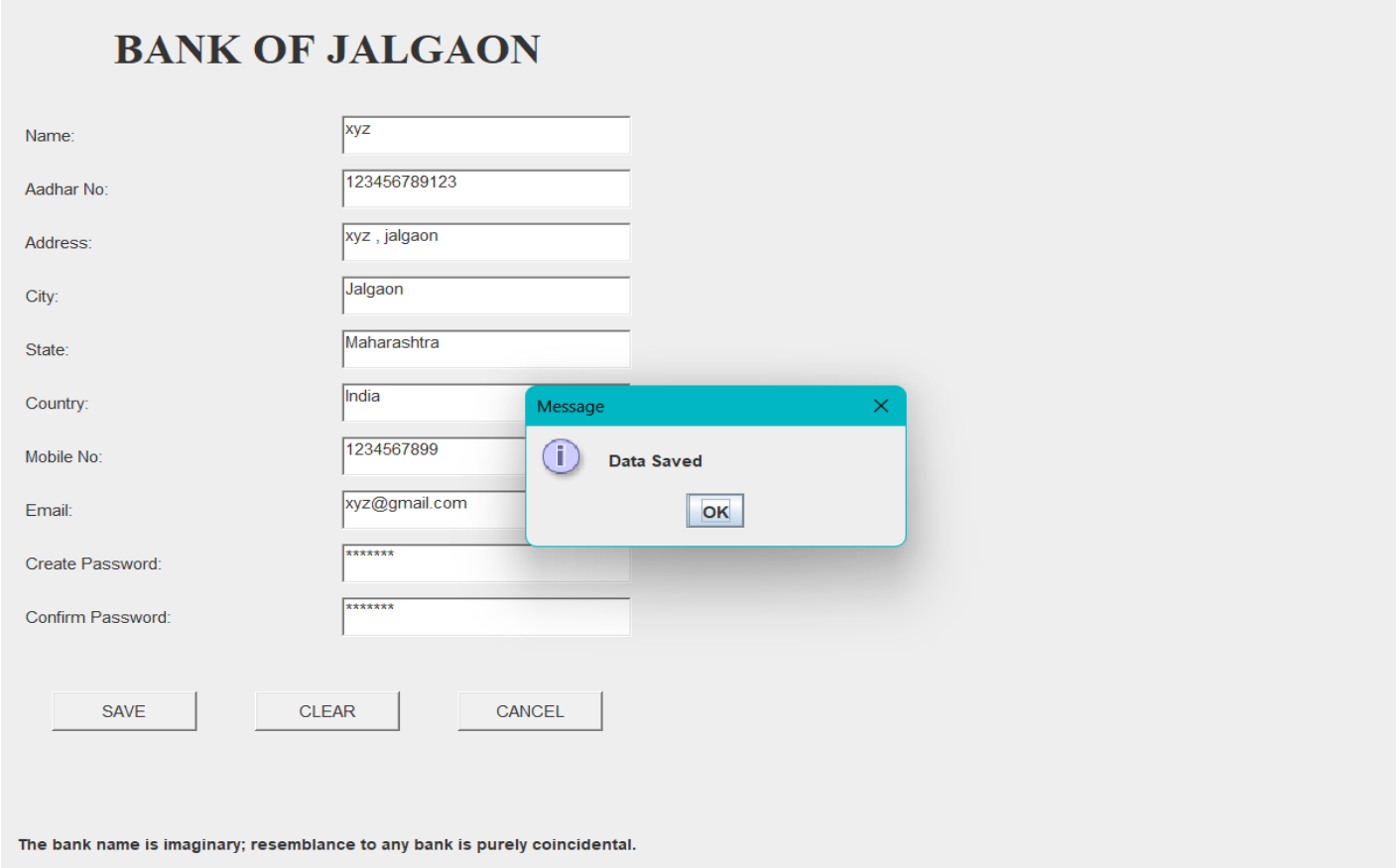
```
private void addButton(Button button, int xPos) {
   button.setBounds(xPos, 550, 100, 30);
   add(button);
}

@Override
public void actionPerformed(ActionEvent e) {
   if (e.getSource() == saveButton) {
      if (validateInputs()) {
         JOptionPane.showMessageDialog(this, "Data Saved");
         clearFields();
      } else {
         JOptionPane.showMessageDialog(this, "Data Not Saved\n Please Enter Correct Data");
      }
   } else if (e.getSource() == clearButton) {
      clearFields();
   } else if (e.getSource() == cancelButton) {
      System.exit(0);
   }
}

private void clearFields() {
   nameField.setText("");
   aadharField.setText("");
   addressField.setText("");
   cityField.setText("");
   stateField.setText("");
   countryField.setText("");
   mobField.setText("");
   emailField.setText("");
   passwordField.setText("");
   confirmPasswordField.setText("");
}

private boolean validateInputs() {
   if (!nameField.getText().matches("[a-zA-Z\\s]+")) {
      JOptionPane.showMessageDialog(this, "Please Enter a Valid Name");
      return false;
   }
   if (!cityField.getText().matches("[a-zA-Z\\s]+")) {
      JOptionPane.showMessageDialog(this, "Please Enter a Valid City");
      return false;
   }
   if (!stateField.getText().matches("[a-zA-Z\\s]+")) {
      JOptionPane.showMessageDialog(this, "Please Enter a Valid State");
```

```
            return false;
        }
        if (!countryField.getText().matches("[a-zA-Z\\s]+")) {
            JOptionPane.showMessageDialog(this, "Please Enter a Valid Country");
            return false;
        }
        if (!passwordField.getText().equals(confirmPasswordField.getText())) {
            JOptionPane.showMessageDialog(this, "Passwords Do Not Match");
            confirmPasswordField.setText("");
            return false;
        }
        return true;
    }

    public static void main(String[] args) {
        new BankForm().setVisible(true);
    }
}
```

Output :


BANK OF JALGAON
Name:
Aadhar No:
Address:
City:
State:
Country:
Mobile No:
Email:
Create Password:
Confirm Password:
SAVE
CLEAR
CANCEL
The bank name is imaginary; resemblance to any bank is purely coincidental.


BANK OF JALGAON
Name: xyz
Aadhar No: 123456789123
Address: xyz , jalgaon
City: Jalgaon
State: Maharashtra
Country: India
Mobile No: 1234567899
Email: xyz@gmail.com
Create Password: *******
Confirm Password: *******
SAVE
CLEAR
CANCEL
The bank name is imaginary; resemblance to any bank is purely coincidental.


BANK OF JALGAON
Name: xyz
Aadhar No: 123456789123
Address: xyz , jalgaon
City: Jalgaon
State: Maharashtra
Country: India
Mobile No: 1234567899
Email: xyz@gmail.com
Create Password: *******
Confirm Password: *******
Message
Data Saved
OK
SAVE
CLEAR
CANCEL
The bank name is imaginary; resemblance to any bank is purely coincidental.

By Mrs. Shrutika S. Mahajan