

ATM GUI

A MINI PROJECT REPORT

Submitted by

RAGAVENDRA.R BE (95072012072)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

FRANCIS XAVIER ENGINEERING COLLEGE

(Autonomous)

TIRUNELVELI - 627 003

FRANCIS XAVIER ENGINEERING COLLEGE

(Autonomous)

TIRUNELVELI-627 003

BONAFIDE CERTIFICATE

Certified that this project report “**ATM GUI**” is the bonafide work of
”**RAGAVENDRA.R (95072012072)**”who carried out the project work under my
supervision.

SIGNATURE

Ms.R.VALARMATHI M.E

SUPERVISOR,

Assistant Professor,

CSE Department,

Francis Xavier Engineering College,

Tirunelveli – 627003.

SIGNATURE

Mrs.P.BRUNDHA M.E (Ph.D)

HEAD OF THE DEPARTMENT,

Assistant Professor,

CSE Department,

Francis Xavier Engineering College,

Tirunelveli – 627003.

Submitted for the B.E Degree Mini Project Viva Voce held on

INTERNAL EXAMINER

ATM GUI



BY
R.RAGAVENDRA

ABSTRACT:

An Automated Teller Machine (ATM) is a safety-critical and real-time system that is highly complicated in design and implementation. This paper presents the formal design, specification, and modeling of the ATM system using a denotational mathematics known as Real-Time Process Algebra (RTPA). The conceptual model of the ATM system is introduced as the initial requirements for the system. The architectural model of the ATM system is created using RTPA architectural modeling methodologies and refined by a set of Unified Data Models (UDMs), which share a generic mathematical model of tuples. The static behaviors of the ATM system are specified and refined by a set of Unified Process Models (UPMs) for the ATM transition processing and system supporting processes. The dynamic behaviors of the ATM system are specified and refined by process priority allocation, process deployment, and process dispatch models. Based on the formal design models of the ATM system, code can be automatically generated using the RTPA Code Generator (RTPA-CG), or be seamlessly transformed into programs by programmers. The formal models of ATM may not only serve as a formal design paradigm of real-time software systems, but also a test bench for the expressive power and modeling capability of exiting formal methods in software engineering.

PROGRAM SOURCE CODE:

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

public class Login extends JFrame {

    String pin = "1234";
    int attempts = 3;
    public Login(){
        //Basic Constructor Setup
        super("Login Screen");
        setLayout(new BorderLayout());
        setResizable(false);
        setLocationRelativeTo(null);
        buildApp();
        pack();
        setSize(250, 110);
        setVisible(true);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

    public static void main(String[] args){
        new Login();
    }

    void buildApp(){
        //=====
        //GUI Setup
        //=====
        JLabel pinInstruction = new JLabel("Please enter your pin number");
        JPasswordField pinText = new JPasswordField(4);
        JButton entPin = new JButton("Enter");
        JButton quit = new JButton("Quit");
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
        buttonPanel.add(quit);
        buttonPanel.add(entPin);
        pinText.setBackground(Color.white);
        add(pinInstruction, BorderLayout.NORTH);
        add(pinText, BorderLayout.CENTER);
        add(buttonPanel, BorderLayout.SOUTH);

        //=====
        //Action Listener Setup
        //=====
        quit.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent event) {
                System.exit(0);
            }
        });
    }
}
```

```

    });

    pinText.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent event) {
            char[] pinGuess = pinText.getPassword();
            String pinString = new String(pinGuess);
            if(pinString.equals(pin)){
                JOptionPane.showMessageDialog(null, "That pin is correct!
Opening Account...");
                dispose();
                new ATM();
            } else{
                if(attempts != 1){
                    attempts--;
                    JOptionPane.showMessageDialog(null, "That pin is
incorrect! \n" + attempts
                    + " attempts remaining!");
                } else {
                    JOptionPane.showMessageDialog(null, "No Attempts
remaining! \n Closing Program");
                    System.exit(0);
                }
            }
        }
    });

    entPin.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent event) {
            char[] pinGuess = pinText.getPassword();
            String pinString = new String(pinGuess);
            if(pinString.equals(pin)){
                JOptionPane.showMessageDialog(null, "That pin is correct!
Opening Account...");
                dispose();
                new ATM();
            } else{
                if(attempts != 1){
                    attempts--;
                    JOptionPane.showMessageDialog(null, "That pin is
incorrect! \n" + attempts
                    + " attempts remaining!");
                } else {
                    JOptionPane.showMessageDialog(null, "No Attempts
remaining! \n Closing Program");
                    System.exit(0);
                }
            }
        }
    });
}

import java.awt.*;

```

```

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

public class ATM extends JFrame {
    //Variables for balance, the input from the user for deposits,
    //and for checking that they have clicked Deposit before clicking Enter
    int balance = 10;
    String[] inputSequence = new String[4];
    String[] transactionHist = new String[100];
    int inputSequenceIndex = 0;
    int transactionIndex = 0;
    boolean readyToEnter = false;
    //Basic Constructor Setup - Setting the input array as empty
    //Other basic setup options - Setting size, location etc, building app
    public ATM(){
        super("ATM");
        for(int i=0; i <= 3; i++){
            inputSequence[i] = "";
        }
        setResizable(false);
        setLocationRelativeTo(null);
        buildApp();
        pack();
        setSize(600, 350);
        setVisible(true);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

    protected void buildApp(){
        //=====
        //GUI Setup (VIEW)
        //=====
        //Basic Panel layout setup
        JLabel displayArea = new JLabel("<html>Instruction Area: <br> Please select a function
from the buttons below <br> Current Balance: \u00A3" + balance + "</html>");
        displayArea.setOpaque(true);
        displayArea.setBackground(Color.white);
        displayArea.setPreferredSize(new Dimension(100, 100));
        JPanel bottomArea = new JPanel();
        bottomArea.setLayout(new BorderLayout(0,0));
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new BorderLayout(0,0));
        JLabel inputDisplay = new JLabel("Input Area:");
        inputDisplay.setBorder(BorderFactory.createLineBorder(Color.black));
        bottomArea.add(inputDisplay, BorderLayout.NORTH);

        //Grid layout setup for buttons
        GridBagLayout buttonGrid = new GridBagLayout();
        buttonPanel.setLayout(buttonGrid);
        GridBagConstraints bPConst = new GridBagConstraints();

```

```

//Buttons SETUP
bPConst.weightx= 0.1;
bPConst.weighty= 0.1;
JButton withDraw1 = new JButton("Withdraw \u00A35");
bPConst.gridx = 0;
bPConst.gridy = 0;
bPConst.insets = new Insets(0, 0, 5, 5);
withDraw1.setSize(new Dimension(200, 30));
buttonPanel.add(withDraw1, bPConst);
JButton withDraw2 = new JButton("Withdraw \u00A310");
bPConst.gridx = 0;
bPConst.gridy = 1;
withDraw2.setSize(new Dimension(200, 30));
buttonPanel.add(withDraw2, bPConst);
JButton withDraw3 = new JButton("Withdraw \u00A320");
bPConst.gridx = 0;
bPConst.gridy = 2;
withDraw3.setSize(new Dimension(200, 30));
buttonPanel.add(withDraw3, bPConst);
JButton deposit = new JButton("Deposit");
bPConst.gridx = 0;
bPConst.gridy = 3;
deposit.setSize(new Dimension(200, 30));
buttonPanel.add(deposit, bPConst);
JButton quit = new JButton("Quit");
bPConst.gridx = 0;
bPConst.gridy = 4;
bPConst.anchor = GridBagConstraints.PAGE_END;
quit.setSize(new Dimension(200, 30));
buttonPanel.add(quit, bPConst);
JButton number1 = new JButton("1");
bPConst.gridx = 1;
bPConst.gridy = 0;
number1.setSize(new Dimension(200, 30));
buttonPanel.add(number1, bPConst);
JButton number2 = new JButton("2");
bPConst.gridx = 2;
bPConst.gridy = 0;
number2.setSize(new Dimension(200, 30));
buttonPanel.add(number2, bPConst);
JButton number3 = new JButton("3");
bPConst.gridx = 3;
bPConst.gridy = 0;
number3.setSize(new Dimension(200, 30));
buttonPanel.add(number3, bPConst);
JButton number4 = new JButton("4");
bPConst.gridx = 1;
bPConst.gridy = 1;
number4.setSize(new Dimension(200, 30));
buttonPanel.add(number4, bPConst);
JButton number5 = new JButton("5");

```



```

bPConst.gridx = 2;
bPConst.gridy = 1;
number5.setSize(new Dimension(200, 30));
buttonPanel.add(number5, bPConst);
JButton number6 = new JButton("6");
bPConst.gridx = 3;
bPConst.gridy = 1;
number6.setSize(new Dimension(200, 30));
buttonPanel.add(number6, bPConst);
JButton number7 = new JButton("7");
bPConst.gridx = 1;
bPConst.gridy = 2;
number7.setSize(new Dimension(200, 30));
buttonPanel.add(number7, bPConst);
JButton number8 = new JButton("8");
bPConst.gridx = 2;
bPConst.gridy = 2;
number8.setSize(new Dimension(200, 30));
buttonPanel.add(number8, bPConst);
JButton number9 = new JButton("9");
bPConst.gridx = 3;
bPConst.gridy = 2;
number9.setSize(new Dimension(200, 30));
buttonPanel.add(number9, bPConst);
JButton number0 = new JButton("0");
bPConst.gridx = 1;
bPConst.gridy = 3;
number0.setSize(new Dimension(200, 30));
buttonPanel.add(number0, bPConst);
JButton clear = new JButton("Clear");
bPConst.gridx = 2;
bPConst.gridy = 3;
clear.setSize(new Dimension(200, 30));
buttonPanel.add(clear, bPConst);
JButton enter = new JButton("Enter");
bPConst.gridx = 3;
bPConst.gridy = 3;
enter.setSize(new Dimension(200, 30));
//Adding everything to the layouts above
buttonPanel.add(enter, bPConst);
bottomArea.add(buttonPanel, BorderLayout.CENTER);
add(displayArea, BorderLayout.NORTH);
add(bottomArea, BorderLayout.CENTER);

//=====
//Action Listener Setup (CONTROLLER)
//=====
// 3 Withdraw buttons (Simply minus from balance if balance is not below ammount):
withDraw1.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        if(balance >= 5){

```

```

        balance = balance - 5;
        displayArea.setText("<html>\u00A35 Withdrawn! <br><br>" +
finishedTransaction() + "</html>");
        readyToEnter = false;
        System.out.println("User Has Withdrawn \u00A35");
        updateTransactionHist("User Has Withdrawn \u00A35");
    } else {
        displayArea.setText("<html> Your Balance is below \u00A35.
Unable to Withdraw!! <br><br>"
        + finishedTransaction() + "</html>");
    }
}

});
withDraw2.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        if(balance >= 10){
            balance = balance - 10;
            displayArea.setText("<html>\u00A310 Withdrawn! <br><br>" +
finishedTransaction() + "</html>");
            readyToEnter = false;
            System.out.println("User Has Withdrawn \u00A310");
            updateTransactionHist("User Has Withdrawn \u00A310");
        } else {
            displayArea.setText("<html> Your Balance is below \u00A310.
Unable to Withdraw!! <br><br>"
            + finishedTransaction() + "</html>");
        }
    }
});
withDraw3.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        if(balance >= 20){
            balance = balance - 20;
            displayArea.setText("<html>\u00A320 Withdrawn! <br><br>" +
finishedTransaction() + "</html>");
            readyToEnter = false;
            System.out.println("User Has Withdrawn \u00A320");
            updateTransactionHist("User Has Withdrawn \u00A320");
        } else {
            displayArea.setText("<html> Your Balance is below \u00A320.
Unable to Withdraw!! <br><br>"
            + finishedTransaction() + "</html>");
        }
    }
});
// Quit Button - Return to login
quit.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        JOptionPane.showMessageDialog(null, "Your Receipt: \n" +
printReceipt());

```

```

        JOptionPane.showMessageDialog(null, "Logging Out! Returning to
Login Screen!");
        dispose();
        new Login();
    }
});
// Clear Button - Clear the input array (Method at bottom)
clear.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        inputDisplay.setText("Input Display: ");
        displayArea.setText("<html>Input Area Cleared! <br><br>"
+ finishedTransaction() + "</html>");
        clearInput();
        readyToEnter = false;
    }
});
// Number Buttons - Add a number to the input array
number1.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        inputDisplay.setText("Input Display: " + updateInput("1"));
    }
});
number2.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        inputDisplay.setText("Input Display: " + updateInput("2"));
    }
});
number3.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        inputDisplay.setText("Input Display: " + updateInput("3"));
    }
});
number4.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        inputDisplay.setText("Input Display: " + updateInput("4"));
    }
});
number5.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        inputDisplay.setText("Input Display: " + updateInput("5"));
    }
});
number6.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        inputDisplay.setText("Input Display: " + updateInput("6"));
    }
});
number7.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        inputDisplay.setText("Input Display: " + updateInput("7"));
    }
});

```

```

});
number8.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        inputDisplay.setText("Input Display: " + updateInput("8"));
    }
});
number9.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        inputDisplay.setText("Input Display: " + updateInput("9"));
    }
});
number0.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        inputDisplay.setText("Input Display: " + updateInput("0"));
    }
});
// Deposit Button - Primes program for a deposit, allows user to click enter deposit
deposit.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        displayArea.setText("<html> Deposit Selected! <br> Please
input an amout below or equal to \u00A31000 and click enter! <br><br>"
        + finishedTransaction() + "</html>" );
        readyToEnter = true;
    }
});
// Enter Button - adds input array val to balance
enter.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event) {
        if(readyToEnter == true){
            if(Integer.parseInt(getInputSequence()) > 1000){
                displayArea.setText("That input is greater than
\u00A31000!");
                clearInput();
                inputDisplay.setText("Input Display: ");
                readyToEnter = false;
            } else if(Integer.parseInt(getInputSequence()) == 0 ||
getInputSequence().equals("0000")){
                displayArea.setText("You have not entered a value!");
                clearInput();
                inputDisplay.setText("Input Display: ");
                readyToEnter = false;
            } else {
                updateBalance(Integer.parseInt(getInputSequence()));
                displayArea.setText("<html> You have deposited
\u00A3" + getInputSequence()
                + "! <br><br>" + finishedTransaction() + "</html>");
                System.out.println("User Has Deposited \u00A3" +
getInputSequence());
                updateTransactionHist("User Has Deposited \u00A3" +
getInputSequence());
            }
        }
    }
});

```

```

        clearInput();
        inputDisplay.setText("Input Display: ");
        readyToEnter = false;
    } else {
        displayArea.setText("<html> You have not yet chosen an action!
<br><br>" + finishedTransaction() + "</html>");
        clearInput();
        inputDisplay.setText("Input Display: ");
    }
}

});

}

//=====
//Extra Methods for calculations(MODEL)
//=====
//Clears the string and resets the Input area
void clearInput(){
    for(int i=0; i <= 3; i++){
        inputSequence[i] = "";
    }
    inputSequenceIndex = 0;
}

//Updates the Input area and the string with a new number when button clicked
String updateInput(String n){
    if(inputSequenceIndex <= 3){
        inputSequence[inputSequenceIndex] = n;
        inputSequenceIndex++;

        StringBuilder strBuilder = new StringBuilder();
        for (int i = 0; i < inputSequence.length; i++) {
            strBuilder.append(inputSequence[i]);
        }
        String newString = strBuilder.toString();
        return newString;
    } else{
        StringBuilder strBuilder = new StringBuilder();
        for (int i = 0; i <= 3; i++) {
            strBuilder.append(inputSequence[i]);
        }
        String newString = strBuilder.toString();
        return newString;
    }
}

String getInputSequence(){
    StringBuilder strBuilder = new StringBuilder();
    if(inputSequence[0] == ""){
        return "0000";
    } else {
        for (int i = 0; i < inputSequence.length; i++) {

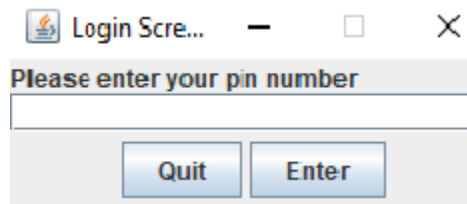
```

```

        strBuilder.append(inputSequence[i]);
    }
    String newString = strBuilder.toString();
    return newString;
}
//Resetting the label to its original state with new balance
String finishedTransaction(){
    return "Instruction Area: <br> Please select a function from the buttons below <br>
Current Balance: \u00A3" + balance;
}
//Updating balance upon deposit
void updateBalance(int l){
    balance += l;
}
//Updating the transaction history for the final receipt
void updateTransactionHist(String t){
    transactionHist[transactionIndex] = t;
    transactionIndex++;
}
//Put the final receipt together and return it for printing
String printReceipt(){
    if(inputSequence[0].equals(null)){
        return "No Transactions Made!";
    } else{
        StringBuilder strBuilder = new StringBuilder();
        for (int i = 0; i < transactionIndex; i++) {
            strBuilder.append(transactionHist[i] + "\n");
        }
        String newString = strBuilder.toString();
        return newString;
    }
}
//Main Method
public static void main (String[] args){
    new Login();
}
}

```

OUTPUT:



Login Scre... — □ ×

Please enter your pin number

Quit Enter

A screenshot of a Windows-style window titled "Login Scre...". It features a text input field with the placeholder text "Please enter your pin number". Below the input field are two buttons: "Quit" and "Enter".



Message ×

i That pin is correct! Opening Account...

OK

A screenshot of a Windows-style message dialog box titled "Message". It contains an information icon (i) and the text "That pin is correct! Opening Account...". At the bottom is an "OK" button.



ATM — □ ×

Instruction Area:
Please select a function from the buttons below
Current Balance: £10

Input Area:

Withdraw £5	1	2	3
Withdraw £10	4	5	6
Withdraw £20	7	8	9
Deposit	0	Clear	Enter
Quit			

A screenshot of a Windows-style window titled "ATM". It has a menu bar with "ATM" and standard window controls. The main area is divided into two sections: "Instruction Area" and "Input Area". The "Instruction Area" contains the text "Please select a function from the buttons below" and "Current Balance: £10". The "Input Area" contains a grid of buttons for transactions (Withdraw £5, £10, £20, Deposit, Quit) and a numeric keypad (0-9, Clear, Enter).

