**1-login class**

**✅ Class Overview**

java

CopyEdit

public class Login extends JFrame implements ActionListener

* **Extends JFrame**: It’s a window in your application.
* **Implements ActionListener**: Allows the class to respond to button clicks.

**📌 UI Components Declared**

java

CopyEdit

JLabel label1 ,label2,label3;

JTextField textField2;

JPasswordField passwordField3;

JButton button1,button2,button3;

* label1, label2, label3: Welcome label, card number label, PIN label.
* textField2: For the **Card Number** input.
* passwordField3: For the **PIN** input.
* button1 ("SIGN IN"), button2 ("CLEAR"), button3 ("SIGN UP").

**🎨 Constructor: Login()**

**🔹 Window Title**

java

CopyEdit

super("Bank Management System");

**🔹 Logo Image**

java

CopyEdit

ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icon/bank.png"));

...

add(image);

* Loads and scales the bank icon from your resources.
* Adds to the frame at position (350, 10).

**🔹 Card Icon (Decoration)**

java

CopyEdit

ImageIcon ii1 = new ImageIcon(ClassLoader.getSystemResource("icon/card.png"));

...

add(iimage);

**🔹 Welcome Label**

java

CopyEdit

label1 = new JLabel("WELCOME TO ATM");

label1.setFont(new Font("AvantGarde",Font.BOLD,38));

...

add(label1);

**🔹 Card Number Field**

java

CopyEdit

label2 = new JLabel("Card No:");

textField2 = new JTextField(15);

**🔹 PIN Field**

java

CopyEdit

label3 = new JLabel("PIN:");

passwordField3 = new JPasswordField(15);

**🔹 Buttons**

* **SIGN IN (button1)** – tries to log in
* **CLEAR (button2)** – clears input fields
* **SIGN UP (button3)** – opens registration window

Each button has:

java

CopyEdit

buttonX.addActionListener(this);

This links it to actionPerformed() method.

**🔹 Background Image**

java

CopyEdit

ImageIcon iii1 = new ImageIcon(ClassLoader.getSystemResource("icon/backbg.png"));

...

add(iiimage);

Sets a background image that covers the entire frame.

**🔹 Frame Setup**

java

CopyEdit

setLayout(null);

setSize(850,480);

setLocation(270,120);

setVisible(true);

**⚙️ Action Handling**

java

CopyEdit

public void actionPerformed(ActionEvent e)

Handles what happens when each button is clicked:

**✅ SIGN IN Button**

java

CopyEdit

Conn c = new Conn();

String cardno = textField2.getText();

String pin = passwordField3.getText();

String q = "select \* from login where card\_number ='"+cardno+"' and '"+pin+"'";

* Executes query to match card number and PIN.
* Conn is your database connection class (likely has Connection and Statement).
* If a match is found, opens a new window:

java

CopyEdit

new main\_Class(pin);

Otherwise:

java

CopyEdit

JOptionPane.showMessageDialog(null, "Incorrect Card Number or Pin");

**🔁 CLEAR Button**

java

CopyEdit

textField2.setText("");

passwordField3.setText("");

Clears the input fields.

**📝 SIGN UP Button**

java

CopyEdit

new Signup();

setVisible(false);

Opens the signup window and hides the login screen.

**2- sign-in class**

## ✅ ****Class Overview****

java

CopyEdit

public class Signup extends JFrame implements ActionListener

* JFrame: Creates a window.
* ActionListener: Handles button clicks (specifically the **Next** button).

## 📌 ****Purpose of Signup Class****

This is **Page 1 of the Sign Up form**, where users fill:

* Personal details like name, father’s name, DOB, gender, email, marital status, address, etc.
* Generates a random **application form number** and saves these details to the database when "Next" is clicked.
* Moves to Signup2 class (likely Page 2 of the form).

## 🔢 ****Form Number Generation****

java

CopyEdit

Random ran = new Random();

long First4 = (ran.nextLong() % 9000L) + 1000L;

String first = "" + Math.abs(First4);

* Generates a **4-digit random number** as a form number (positive only).
* Stored as first and reused later for user tracking.

## 🖼️ ****UI Layout Setup****

* Absolute layout with manual positioning.
* Uses setBounds() and add() to place each component.
* Custom colors and fonts.

### 🧍‍♂️ ****User Input Fields****

| **Label** | **Component** | **Purpose** |
| --- | --- | --- |
| Name | textName | User name |
| Father’s Name | textFname | Father's name |
| Date of Birth | dateChooser | Calendar picker |
| Gender | r1, r2 | Radio buttons (Male, Female) |
| Email | textEmail | Email input |
| Marital Status | m1, m2, m3 | Radio buttons (Married, Unmarried, Other) |
| Address | textAdd | Full address |
| City | textCity | City |
| Pincode | textPin | Area PIN code |
| State | textstate | State |

#### ✅ ****Radio Button Groups****

java

CopyEdit

ButtonGroup buttonGroup = new ButtonGroup();

buttonGroup.add(r1);

buttonGroup.add(r2);

This ensures only one gender/marital option is selected.

## ▶️ ****Next Button****

java

CopyEdit

next = new JButton("Next");

next.addActionListener(this);

* Proceeds to the next signup form (Signup2) after saving data.

## ⚙️ ****actionPerformed() Method****

Triggered when "Next" button is clicked.

### 🟢 Step-by-step Logic:

1. **Extract Input** from form fields.
2. **Assign Gender & Marital Status** based on selection.
3. **Validate Input**:

java

CopyEdit

if (textName.getText().equals("")) {

JOptionPane.showMessageDialog(null, "FILL all the field");

}

1. **Store in Database**:

java

CopyEdit

String q = "insert into signup values(...)";

conn1.statement.executeUpdate(q);

1. **Move to Signup2** page with form number:

java

CopyEdit

new Signup2(first);

## 📂 ****Database Interaction****

java

CopyEdit

Conn conn1 = new Conn();

conn1.statement.executeUpdate(q);

* Assumes a Conn class exists which handles:
  + Database connection setup
  + Exposes a statement object for SQL execution

**2-Signup2 class**

## ✅ ****Purpose of**** Signup2

This class collects:

* Demographic and financial info
* PAN & Aadhaar (with **format validation**)
* Flags like "Senior Citizen" and "Existing Account"
* Then saves all data into a table (signuptwo) and proceeds to Signup3.

## 🧱 ****Class Declaration****

java

CopyEdit

public class Signup2 extends JFrame implements ActionListener

* Inherits from JFrame → it’s a GUI window.
* Implements ActionListener to handle button clicks.

## 🔢 ****Constructor:**** Signup2(String formno)

### 📌 ****Header and Logo****

java

CopyEdit

ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icon/bank.png"));

* Shows logo in the top left.
* Shows current form number passed from Signup1.

### 🧍‍♂️ ****Fields and Inputs Collected****

| **Label** | **Input Component** | **Description** |
| --- | --- | --- |
| Religion | comboBox | Dropdown: Hindu, Muslim, etc. |
| Category | comboBox2 | General, OBC, SC, etc. |
| Income | comboBox3 | Income brackets |
| Education | comboBox4 | Educational qualification |
| Occupation | comboBox5 | Job type |
| PAN Number | textPan | PAN (validated format) |
| Aadhaar Number | textAadhar | Aadhaar (12-digit check) |
| Senior Citizen | r1, r2 | RadioButtons Yes/No |
| Existing Account | e1, e2 | RadioButtons Yes/No |

All are placed using setBounds() with custom layout and design.

## 🎛️ ****Radio Button Grouping****

java

CopyEdit

ButtonGroup buttonGroup = new ButtonGroup();

buttonGroup.add(r1);

buttonGroup.add(r2);

* Prevents both Yes/No from being selected at the same time (like gender in previous class).

## 🎯 ****actionPerformed() Method (Main Logic)****

### ✅ Step-by-Step:

java

CopyEdit

String rel = (String)this.comboBox.getSelectedItem();

...

String pan = this.textPan.getText().trim().toUpperCase();

String aadhar = this.textAadhar.getText().trim();

### ✅ Validation Logic:

java

CopyEdit

String panPattern = "[A-Z]{5}[0-9]{4}[A-Z]";

String aadharPattern = "\\d{12}";

* ✅ Checks:
  + PAN must be **5 letters + 4 digits + 1 letter** (e.g., **ABCDE1234F**)
  + Aadhaar must be **exactly 12 digits**

### 🔄 On Success:

java

CopyEdit

String q = "INSERT INTO Signuptwo VALUES(...)";

c1.statement.executeUpdate(q);

new Signup3(this.formno);

* Saves into signuptwo table.
* Opens **Signup3** (likely account type selection page).
* Hides current frame.

### ❗ On Failure:

* Shows message box for invalid PAN or Aadhaar or if empty.

**4-Signup3 class**

## ✅ Class Summary

**Signup3 extends JFrame implements ActionListener**

* It displays:
  + Account types
  + Auto-generated card number and PIN
  + Checkbox options for services
  + Declaration and Submit button

## 🧱 GUI Components and Layout

| **Component** | **Purpose** |
| --- | --- |
| JRadioButton r1–r4 | For selecting account type |
| JCheckBox c1–c6 | For selecting services like ATM Card, Net Banking |
| JButton s, c | Submit and Cancel buttons |
| String formno | Carries form number from previous pages |
| JLabels | Display layout texts and card info placeholders |

## 🧠 Form Fields & Features

### 🔹 ****Account Type****

java

CopyEdit

r1 = "Saving Account"

r2 = "Fixed Deposit"

r3 = "Current Account"

r4 = "Recurring Deposit"

Grouped with ButtonGroup to ensure only one can be selected.

### 🔹 ****Services Checkboxes****

java

CopyEdit

ATM CARD, Internet Banking, Mobile Banking, Email Alerts, Cheque Book, E-Statement

Collected and appended to fac string.

### 🔹 ****Card Number & PIN (Display Only)****

* Card No: **"XXXX-XXXX-XXXX-4841"** placeholder
* PIN: **"XXXX"**
* Both are auto-generated **when the user submits**.

## 🎯 actionPerformed() Logic

Triggered when:

* Submit (s) button is clicked
* Cancel (c) button is clicked

### ✅ On Submit:

java

CopyEdit

if (atype == null) {

JOptionPane.showMessageDialog(...);

}

➡ Ensures **account type is selected**.

### 🔒 Card Number and PIN Generation (Important Part):

java

CopyEdit

Random ran = new Random();

long first7 = ran.nextLong() % 90000000L + 1409963000000000L; // 16-digit card number

long first3 = ran.nextLong() % 9000L + 1000L; // 4-digit pin

#### 🧪 Uniqueness Check:

java

CopyEdit

ResultSet rs = c1.statement.executeQuery("SELECT \* FROM login WHERE card\_number = ...");

* Ensures that **generated card number and pin** do not exist already.
* If not unique, loop runs again.

### 💾 Database Insertion:

java

CopyEdit

String q1 = "insert into signupthree values(...)";

String q2 = "insert into login values(...)";

* signupthree table → stores account type, card number, pin, selected services.
* login table → stores login credentials (used by Login.java).

### 🧾 Success Feedback:

java

CopyEdit

JOptionPane.showMessageDialog(null, "Card Number : " + cardno + "\n Pin : " + pin);

new Deposit(pin);

setVisible(false);

➡ Shows **Card + PIN to user** and opens the **Deposit** screen.

### ❌ On Cancel:

java

CopyEdit

if (e.getSource() == this.c) {

System.exit(0);

}

Exits the program.

**5-pin class**

**🔰 1. Class Definition and Constructor**

java

CopyEdit

public class Pin extends JFrame implements ActionListener {

String pin;

JButton b1; // CHANGE button

JButton b2; // BACK button

JPasswordField p1; // Field for new PIN

JPasswordField p2; // Field to re-enter new PIN

These variables are used to:

* Accept new PIN input
* Handle user actions (Change/Back)
* Track the user's **current PIN (pin)** passed from previous screen

**🖼️ 2. UI Setup**

java

CopyEdit

ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icon/atm2.png"));

Image i2 = i1.getImage().getScaledInstance(1350, 700, 1);

ImageIcon i3 = new ImageIcon(i2);

JLabel l3 = new JLabel(i3);

l3.setBounds(0, 0, 1050, 690);

this.add(l3);

* Loads background ATM image and adds it to the frame.

**🔡 3. Add Labels and Input Fields**

java

CopyEdit

JLabel label1 = new JLabel("CHANGE YOUR PIN");

label1.setBounds(240, 120, 400, 35);

l3.add(label1);

JLabel label2 = new JLabel("NEW PIN:");

JPasswordField p1 = new JPasswordField();

l3.add(p1);

JLabel label3 = new JLabel("Re-Enter NEW PIN:");

JPasswordField p2 = new JPasswordField();

l3.add(p2);

* Labels and password fields for **new PIN entry and confirmation**.
* These use JPasswordField for security (hides characters).

**🧠 4. Button Setup**

java

CopyEdit

b1 = new JButton("CHANGE"); // Triggers PIN update

b2 = new JButton("BACK"); // Returns to previous screen

b1.addActionListener(this);

b2.addActionListener(this);

* You’ve added two buttons and linked them to actionPerformed() using listeners.

**🧨 5. Main Logic — actionPerformed()**

java

CopyEdit

String pin1 = this.p1.getText();

String pin2 = this.p2.getText();

* Read entered PIN values.

java

CopyEdit

if (!pin1.equals(pin2)) {

JOptionPane.showMessageDialog((Component)null, "Entered PIN does not match");

return;

}

* Ensure both PIN fields match.

**✅ 6. Validations & Update DB**

java

CopyEdit

if (this.p1.getText().equals("")) {

JOptionPane.showMessageDialog((Component)null, "Enter New PIN");

return;

}

* Prevent empty field submission.

java

CopyEdit

Conn c = new Conn();

String q1 = "update bank set pin = '" + pin1 + "' where pin = '" + this.pin + "'";

String q2 = "update login set pin = '" + pin1 + "' where pin = '" + this.pin + "'";

String q3 = "update signupthree set pin = '" + pin1 + "' where pin = '" + this.pin + "'";

* These lines **update the PIN in 3 different tables**.
* this.pin refers to the **current PIN** used for finding the right user.

java

CopyEdit

c.statement.executeUpdate(q1);

c.statement.executeUpdate(q2);

c.statement.executeUpdate(q3);

* All 3 tables are updated to reflect the new PIN.

java

CopyEdit

JOptionPane.showMessageDialog((Component)null, "PIN changed successfully");

this.setVisible(false);

new main\_Class(this.pin);

* Success message and navigation to main\_Class.

**6-Conn class**

**✅ FILE: Conn.java — Explanation**

This class handles **database connection setup** and provides an easy way to run SQL statements.

**1️⃣ Fields**

java

CopyEdit

Connection connection;

Statement statement;

* Connection connection: Represents the DB connection.
* Statement statement: Lets you send SQL commands to the DB.

**2️⃣ Constructor**

java

CopyEdit

public Conn() {

try {

this.connection = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/bankSystem",

"root",

"4hmRahulop@123"

);

this.statement = this.connection.createStatement();

} catch (Exception e) {

e.printStackTrace();

}

}

✅ What happens here:

* **DriverManager.getConnection()** connects to MySQL DB:
  + localhost:3306: local DB server and port.
  + bankSystem: DB name.
  + root: DB username.
  + 4hmRahulop@123: DB password.
* **createStatement()** creates a Statement object to run SQL commands like:

java

CopyEdit

statement.executeUpdate("INSERT INTO ...");

statement.executeQuery("SELECT ...");

* The try-catch ensures:
  + If connection fails (wrong password, DB down, etc.), it shows the error.

**7-Main class**

**🎯 What this class does:**

✅ **Extends JFrame and implements ActionListener.**  
That means:

* It is a Swing-based GUI window.
* It listens for button clicks (actionPerformed() method).

**🧠 Fields (important parts)**

java

CopyEdit

JButton b1, b2, b3, b4, b5, b6, b7;

String pin;

* b1 to b7: Buttons for different actions.
* pin: Stores the logged-in user's ATM PIN.

**🧱 Constructor: main\_Class(String pin)**

✅ **Takes the logged-in pin**  
✅ Creates the UI layout:

* **Background image** (atm2.png)
* **Label** with "Please Select Your Transaction"
* Creates all 7 JButtons with labels like "DEPOSIT", "WITHDRAWL", "PIN CHANGE", etc.
* Sets their positions and colors.
* Adds ActionListener to each button.

💡 this.setVisible(true); finally displays the window.

**🕹️ actionPerformed() — Handling button clicks:**

Each if/else if checks which button was clicked:

| **Button** | **Action** |
| --- | --- |
| b1 (DEPOSIT) | Opens new Deposit(this.pin) screen and hides current. |
| b2 (WITHDRAWL) | Opens new Withdrawl(this.pin) screen and hides current. |
| b3 (FAST CASH) | Opens new FastCash(this.pin) screen and hides current. |
| b4 (MINI STATEMENT) | Opens new mini(this.pin) screen. (Notice: It **does not hide** current screen.) |
| b5 (PIN CHANGE) | Opens new Pin(this.pin) screen and hides current. |
| b6 (BALANCE ENQUIRY) | Opens new BalanceEnquiry(this.pin) screen and hides current. |
| b7 (EXIT) | Calls System.exit(0) to close the app. |

**8- fastcash class**

## 🎯 What this class is:

* Extends JFrame to show a Swing-based screen.
* Implements ActionListener so it can respond to button clicks.
* Provides quick-cash options like Rs. 100, Rs. 500, Rs. 1000, etc.

## 🧠 Fields:

java

CopyEdit

JButton b1, b2, b3, b4, b5, b6, b7;

String pin;

✅ pin: The logged-in user’s ATM PIN — passed from the previous screen (main\_Class).  
✅ b1 to b7: Buttons for different withdrawal amounts and a Back option.

## 🔨 Constructor (FastCash(String pin)):

**Key points in the UI setup:**

* Sets a **background image** (atm2.png).
* Adds a heading label:  
  "SELECT WITHDRAWL AMOUNT"
* Creates 6 fixed-amount withdrawal buttons (Rs. 100, Rs. 500, etc.).
* Creates a Back button to return to the main screen.
* Uses absolute positioning (setBounds) and null layout.
* Adds ActionListener on each button so actionPerformed() is triggered on click.

## ⚡ actionPerformed(ActionEvent e):

**When a button is clicked:**

* ✅ if (e.getSource() == b7):  
  Back was clicked → closes this screen and returns to main\_Class.
* ✅ Otherwise (one of the money options clicked):
  1. Extracts the numeric amount:

java

CopyEdit

String amount = ((JButton)e.getSource()).getText().substring(4); // skips "Rs. "

* 1. Connects to the database (Conn c = new Conn()).
  2. Fetches the user’s transaction history (select \* from bank where pin='...').
  3. Loops over all transactions to **compute balance**:

java

CopyEdit

if (type == "Deposit") balance += amount;

else balance -= amount;

* 1. Checks if balance ≥ requested amount:

java

CopyEdit

if (balance < requestedAmount) show "Insufficient Balance"

* 1. If OK:
     + Insert a withdrawl transaction into bank table.
     + Show success JOptionPane.
  2. Finally close this screen and go back to main\_Class.

## ✅ Good Practices:

* **Balance check** before allowing withdrawal.
* Inserts a new transaction into the bank table with a timestamp.
* Uses JOptionPane for messages.

**9 deposit class**

## 🎯 What this class is for:

**The Deposit screen** lets the user enter an amount and deposit it into their account.

## 🔍 Key Fields:

java

CopyEdit

String pin; // Logged-in user's ATM pin

TextField textField; // Input field for the deposit amount

JButton b1; // "DEPOSIT" button

JButton b2; // "BACK" button

## 🧠 Constructor (Deposit(String pin)):

Here’s what happens in the constructor:

* **Background Image (atm2.png):**  
  Scales and sets the image as the background label.
* **Title Label:**

java

CopyEdit

JLabel label1 = new JLabel("ENETR AMOUNT YOU WANT TO DEPOSIT");

⚠️ There’s a typo — it should be “ENTER AMOUNT YOU WANT TO DEPOSIT.”

* **Amount Input (textField):**  
  Uses TextField for numeric input.
  + Sets background to a teal color and white text.
  + Adds a bold font so the amount is easy to read.
* **Two Buttons:**
  + b1 — DEPOSIT
  + b2 — BACK
* Sets setLayout(null) and positions everything manually with setBounds().  
  ✅ this.setVisible(true); — Finally displays the screen.

## 🧭 actionPerformed(ActionEvent e) logic:

Here’s what happens when a button is clicked:

1. String amount = textField.getText() — Fetch the input.
2. Date date = new Date() — Get the current timestamp.

✅ **When DEPOSIT is clicked (e.getSource() == b1):**

* Check if the amount is blank → Show an error dialog.
* Otherwise:
  + Open DB connection (new Conn()).
  + Run INSERT into the bank table with:

java

CopyEdit

pin, date, "Deposit", amount

* + Show success message.
  + Close this screen and open main\_Class.

✅ **When BACK is clicked (e.getSource() == b2):**

* Simply close this screen and go back to the main screen.

## ✏️ What this does in the app:

* Gives the user a friendly UI to deposit money into their account.
* Records the deposit in the database (bank table) along with the timestamp.

**10 withdrawl class**

## 🎯 What this class is for:

**Withdrawl** is a screen where the user can enter an amount and withdraw it from their account. It checks:

* if the account balance is sufficient, and
* updates the bank table with a withdrawal transaction.

## 🧠 Fields in the class:

java

CopyEdit

String pin; // The logged-in user's PIN

TextField textField; // Input for withdrawal amount

JButton b1; // "WITHDRAW" button

JButton b2; // "BACK" button

## 🧠 What happens in the ****Constructor**** (Withdrawl(String pin)):

✅ Sets up the ATM-style UI with:

* **Background image** (atm2.png)
* Labels like:
  + MAXIMUM WITHDRAWAL IS RS.10,000
  + PLEASE ENTER YOUR AMOUNT
* Input box (textField)
* Two buttons:
  + **WITHDRAW** — Confirm the withdrawal.
  + **BACK** — Return to the main menu.

Styling:

* White text on teal background.
* Fixed positions using setBounds() and null layout.

## 🔄 actionPerformed(ActionEvent e) — Main logic:

This is triggered when a button is clicked.

### 🟢 When WITHDRAW is clicked:

1. Get the amount from textField.
2. If blank → Show a dialog:  
   JOptionPane.showMessageDialog(null, "Please enter the Amount you want to withdraw");
3. Else:
   * Query all transactions (SELECT \* FROM bank WHERE pin = ?)
   * Loop through the ResultSet:
     + If type == Deposit: add to balance
     + Else (Withdrawl): subtract from balance
   * Check balance against the withdrawal amount:
     + If insufficient → Show error
     + If enough:
       - Insert a new withdrawal record into bank table
       - Show success message
       - Go back to main\_Class screen

### 🟡 When BACK is clicked:

* Closes this screen and returns to main\_Class.

**11 balance enquairy**

## 🎯 What this class is for:

**BalanceEnquriy** is a screen that:

* Retrieves all transactions for the logged-in user.
* Calculates the account balance.
* Displays the current balance on the screen.
* Provides a **Back** button to return to the main ATM menu.

## 🔍 Key Fields:

java

CopyEdit

String pin; // User's ATM PIN

JLabel label2; // Shows the calculated balance

JButton b1; // "Back" button

## 🧠 What the ****Constructor**** (BalanceEnquriy(String pin)) does:

✅ Sets up the ATM-style interface:

* Uses a background image (atm2.png)
* Displays the label **Your Current Balance is Rs**
* Sets up a blank label (label2) where the balance will be shown
* Creates a **Back** button that, when clicked, returns the user to the main\_Class

✅ Fetching and Calculating Balance:

java

CopyEdit

int balance = 0;

ResultSet resultSet = c.statement.executeQuery(

"Select \* from bank where pin = '" + pin + "'"

);

while(resultSet.next()) {

if (resultSet.getString("type").equals("Deposit")) {

balance += Integer.parseInt(resultSet.getString("amount"));

} else {

balance -= Integer.parseInt(resultSet.getString("amount"));

}

}

label2.setText("" + balance); // Show the balance

💡 What this does:

* Retrieves all transaction rows (Deposit or Withdrawl)
* Loops and adjusts the balance accordingly
* Finally displays the balance in label2

✅ UI setup:

* All labels and the Back button are positioned using absolute coordinates (setBounds)
* Colors and fonts are styled manually
* The Back button returns the user to the main\_Class

## 🔄 actionPerformed(ActionEvent e) logic:

When the **Back** button is clicked:

java

CopyEdit

this.setVisible(false); // hide the balance screen

new main\_Class(this.pin); // open the main transaction screen

**12 mini statement**

## 🎯 What this class does:

**mini** is a simple **Mini-Statement screen** in your ATM management system.  
✅ It displays:

* The masked **Card Number** of the logged-in user.
* A list of **all transactions** (date, type, amount) as a mini-statement.
* The **current balance**.
* An **Exit** button to close the screen.

## 🔍 Key Fields:

java

CopyEdit

String pin; // The current user's PIN

JButton button; // Exit button

## 🧠 What the ****constructor**** (mini(String pin)) does:

1. **UI Setup:**
   * Sets a pink background (new Color(255, 204, 204)), fixed size (400 x 600), and absolute layout.
   * Adds a few labels:
     + label1: For transaction history.
     + label2: Displays a static title (TechCoder R.S).
     + label3: Will display masked card number.
     + label4: Will display current balance.
   * Adds the **Exit** button.
2. **Display Card Number**:

java

CopyEdit

ResultSet resultSet = c.statement.executeQuery(

"select \* from login where pin = '" + pin + "'");

Fetches the card number for the current user and masks the middle digits like:

javascript

CopyEdit

Card Number: 1234XXXXXXXX5678

1. **Display Transactions and Balance**:

java

CopyEdit

ResultSet resultSet = c.statement.executeQuery(

"select \* from bank where pin = '" + pin + "'");

* + Loops through all transactions in the bank table.
  + Builds a mini-statement by appending each transaction (date, type, amount) to label1.
  + Calculates the balance:
    - If Deposit, add the amount.
    - Otherwise (Withdrawl), subtract the amount.

Finally, updates label4 to show the current balance:

csharp

CopyEdit

Your Total Balance is Rs X

1. **Exit Button**:
   * The Exit button (button) calls this.setVisible(false); on click, closing the mini-statement screen.

## 🎯 The actionPerformed method:

java

CopyEdit

public void actionPerformed(ActionEvent e) {

this.setVisible(false); // closes the screen

}