Title : Creating the BankAcc Class.

File : BankAcc.java

Code : import

```
java.util.ArrayList; import
java.util.List;
public class BankAcc{     private
String accnumb;     private
double balance;     private
List<Transaction> tran;
   public BankAcc(String accnumb, double initbalance){
this.accnumb = accnumb;         this.balance =
initbalance;         this.tran = new ArrayList<>();
   }

   public String getaccnumb(){
return accnumb;
   }
   public double getbal() {
return balance;
   }
   public void deposit(double amount, String type){         balance +=
amount;         tran.add(new Transaction(accnumb, "Deposit (" + type + ")",
amount));
   }
   public boolean withdraw(double amount, String type){
```

```
        if(amount > balance){
return false;
        }
        balance -= amount;         tran.add(new Transaction(accnumb, "Withdrawal
(" + type + ")", amount));
        return true;
    }
public void addtran(Transaction transaction){ tran.add(transaction);
} public void
showtransaction(){
if(tran.isEmpty()){
        System.out.println("No transactions");
    }else {         for(Transaction
transa : tran){
            System.out.println(tran);
        }
    }
}
}
```

Explanation of the Code :

Purpose

Tracks bank accounts with:

1.Account number

2.Current balance

3.Transaction history

Methods

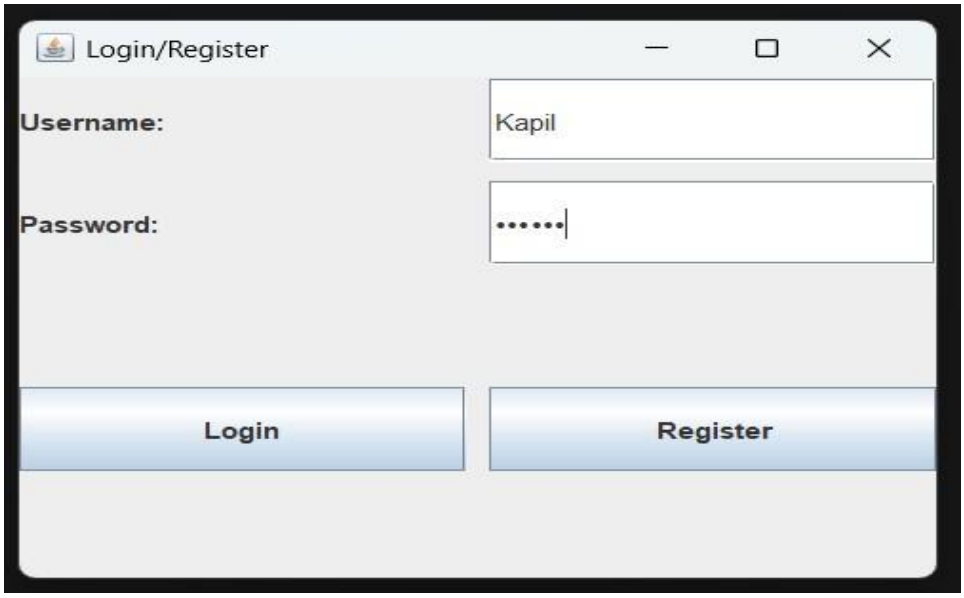
1.Deposit/Withdraw: Updates balance + records transaction

2.Check Balance: getaccnumb(), getbal()

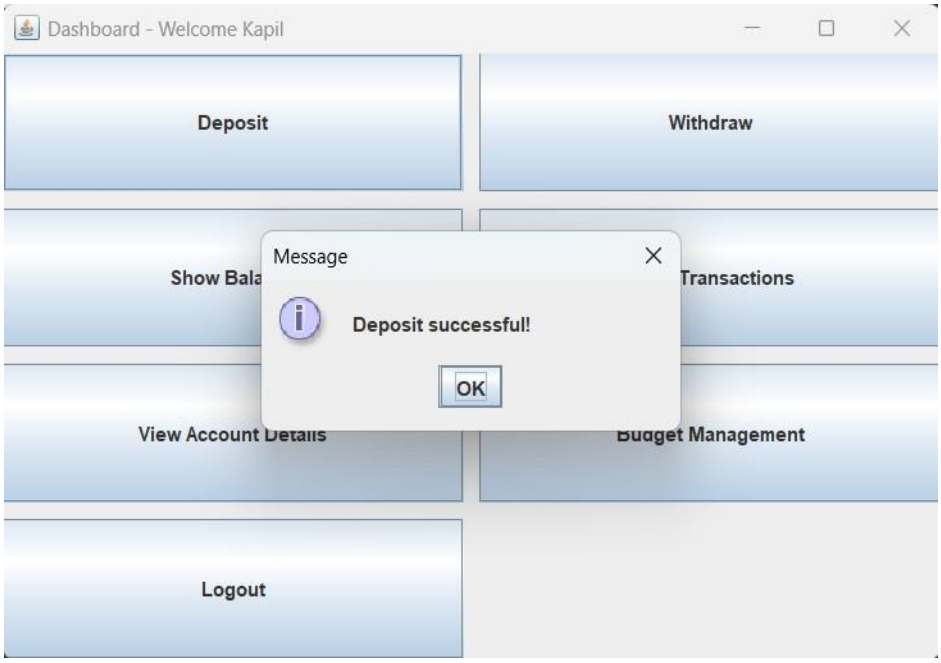
3.View Transactions: showtransaction()

OUTPUT :

We are Registering the New User -



After Registration , We are depositing money-



Title – Creating the BankAccManger Class

File : BankAccManager.java

Use – To handle and manage bank acc's , transaction , doing entries.

Code :

```
import java.io.*; import
java.util.*;
public class BankAccManager {     private final String
ACCOUNTS_FILE = "bank_accounts.txt";     private final String
TRANSACTIONS_FILE = "bank_transactions.txt";
   private List<BankAcc> accounts = new ArrayList<>();
   public BankAccManager() {
      loadacc();
   }
   public void createacc(String accnumb, double initBalance) {
for (BankAcc acc : accounts) {            if
(acc.getaccnumb().equals(accnumb)) {
System.out.println("Account already exists.");
            return;
         }
      }
      accounts.add(new BankAcc(accnumb, initBalance));
      saveacc();
      System.out.println("Account created successfully.");
   }
   public void deposit(String accnumb, double amount, String type) {
for (BankAcc acc : accounts) {            if
```

```
(acc.getaccnumb().equals(accnumb)) {
acc.deposit(amount, type);                saveacc();
logTransaction(accnumb, "Deposit", amount, type);
System.out.println("Deposited successfully.");
            return;
        }
    }
    System.out.println("Account not found.");
  }
  public void withdraw(String accnumb, double amount, String type) {
for (BankAcc acc : accounts) {            if
(acc.getaccnumb().equals(accnumb)) {                if
(acc.withdraw(amount, type)) {                    saveacc();
logTransaction(accnumb, "Withdraw", amount, type);
                System.out.println("Withdrawn successfully.");
            } else {
                System.out.println("Insufficient balance.");
            }
return;
        }
    }
    System.out.println("Account not found.");
  }
  public void showBalance(String accnumb) {
for (BankAcc acc : accounts) {            if
(acc.getaccnumb().equals(accnumb)) {
            System.out.println("Balance: $" + acc.getbal());
```

```
            return;
        }
    }
    System.out.println("Account not found.");
  }
  public void showTransactions(String accnumb) {
    try (BufferedReader reader = new BufferedReader(new
FileReader(TRANSACTIONS_FILE))) {
        String line;          boolean found = false;
while ((line = reader.readLine()) != null) {
if (line.startsWith(accnumb)) {
System.out.println(line);                found =
true;
            }
        }
        if (!found) System.out.println("No transactions found.");
    } catch (IOException e) {
        System.out.println("Error reading transactions.");
    }
  }

  private void logTransaction(String accnumb, String type, double amount, String method) {
try (BufferedWriter writer = new BufferedWriter(new FileWriter(TRANSACTIONS_FILE,
true))) {
        writer.write(accnumb + " - " + type + " - $" + amount + " - " + method);
writer.newLine();
    } catch (IOException e) {
        System.out.println("Error logging transaction.");
```

```
        }
    }
    private void saveacc() {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(ACCOUNTS_FILE)))
{            for (BankAcc acc : accounts) {
writer.write(acc.getaccnumb() + "," + acc.getbal());
writer.newLine();
            }
        } catch (IOException e) {
            System.out.println("Error saving accounts.");
        }
    }
    private void loadacc() {
        File file = new File(ACCOUNTS_FILE);
        if (!file.exists()) return;
        try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
            String line;            while ((line =
reader.readLine()) != null) {                String[]
parts = line.split(",");                if (parts.length
>= 2) {

                String accnumb = parts[0];                    double
balance = Double.parseDouble(parts[1]);
accounts.add(new BankAcc(accnumb, balance));
                }
            }
        } catch (IOException e) {
            System.out.println("Error loading accounts.");
```

```
    }
  }
}
```

Explanation:

1. File Handling

   1.Account data is stored in "bank_accounts.txt"

   2.Transaction records are saved in "bank_transactions.txt"

2. Constructor – BankAccManager()
   Loads existing account data using the loadacc() method at initialization.

3. Creating Account – createacc()
   Adds a new account only if it doesn't already exist. Saves updated account list to file.

4. Depositing Money – deposit()
   Adds the deposit amount to the specified account, logs the transaction, and updates the file.

5. Withdrawing Money – withdraw()
   Attempts to deduct the amount from the account if sufficient balance exists. Updates file and logs transaction.

6. Balance Display – showBalance()
   Prints the current balance for a specified account number.

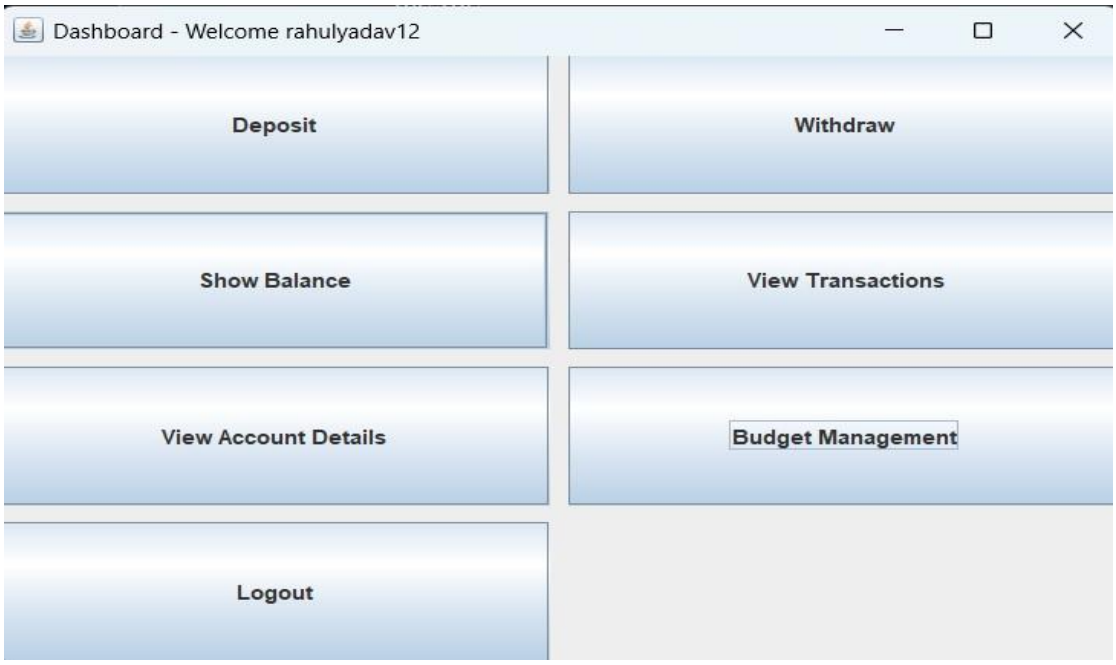
7. View Transactions – showTransactions()
   Reads and displays all transaction logs related to a particular account from the transaction file.

8. Transaction Logging – logTransaction()
   Appends each transaction detail (account, type, amount, method) to "bank_transactions.txt".

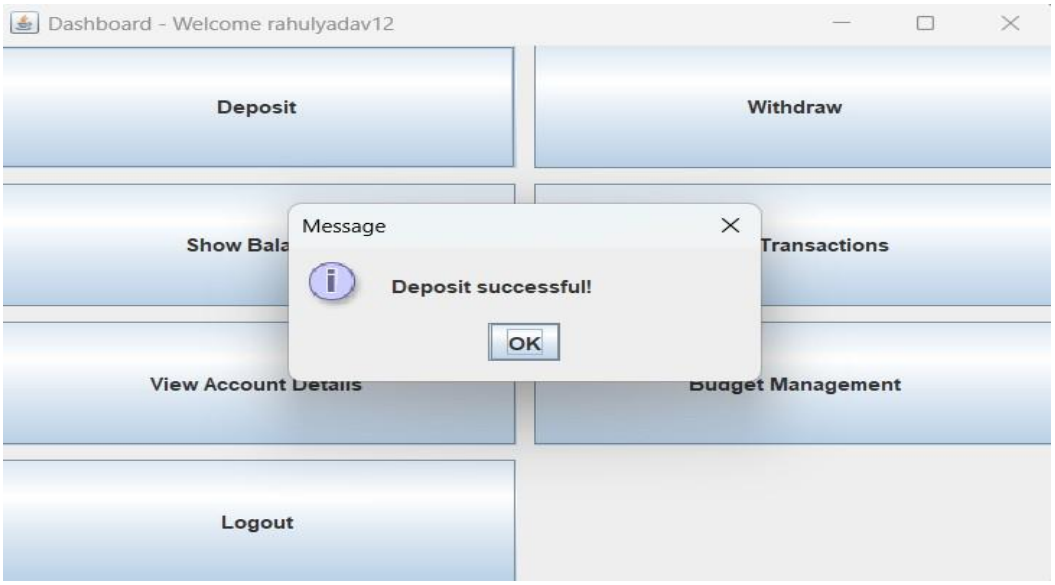
9. Saving Accounts – saveacc()
   Writes all account numbers and their current balances to the "bank_accounts.txt" file.

10. Loading Accounts – loadacc()
    Reads from "bank_accounts.txt" and recreates account objects in memory.

OUTPUT :



Depositing Money :



Title - Budget class represents a budget for a specific category

File : Budget.java

Code :

```
public class Budget{
private String category;
private double limit;
private double totalexp;
   public Budget(String category, double limit, double totalexp){
this.category = category;        this.limit = limit;
this.totalexp = totalexp;
   }
   public String getcate(){
return category;
   }
   public double getlimit(){
return limit;
   }
   public double getexpense(){
return totalexp;
   }
   public void addexp(double amount){
if(totalexp + amount > limit){
          System.out.println("You have exceeded your budget.");
       }
       totalexp += amount;
```

```
    }
}
```

Explanation:

1. Attributes :

   category: Name of the budget category (e.g., Food, Travel).

   limit: Maximum spending allowed for the category.

   totalexp : Total expenses recorded so far for this category.

2. Construcor – Budget(String category, double limit, double totalexp)
   Initializes a new budget with a category name, limit, and existing expense amount.

3. Getter – getcate()
   Returns the name of the budget category.

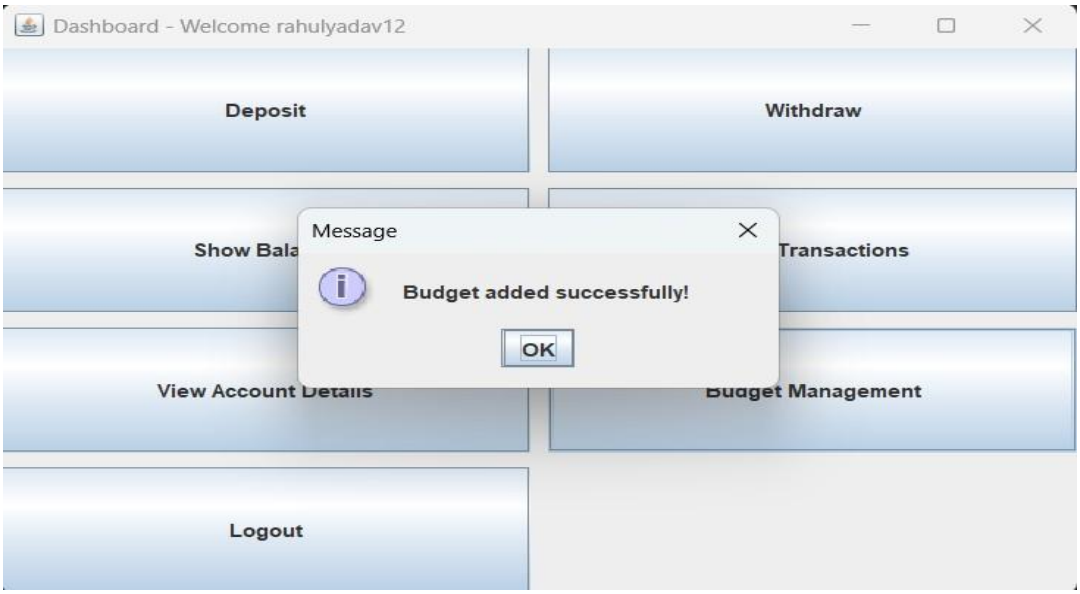
4. Getter – getlimit()
   Returns the budget limit set for the category.

5. Getter – getexpense()
   Returns the current total expense recorded.

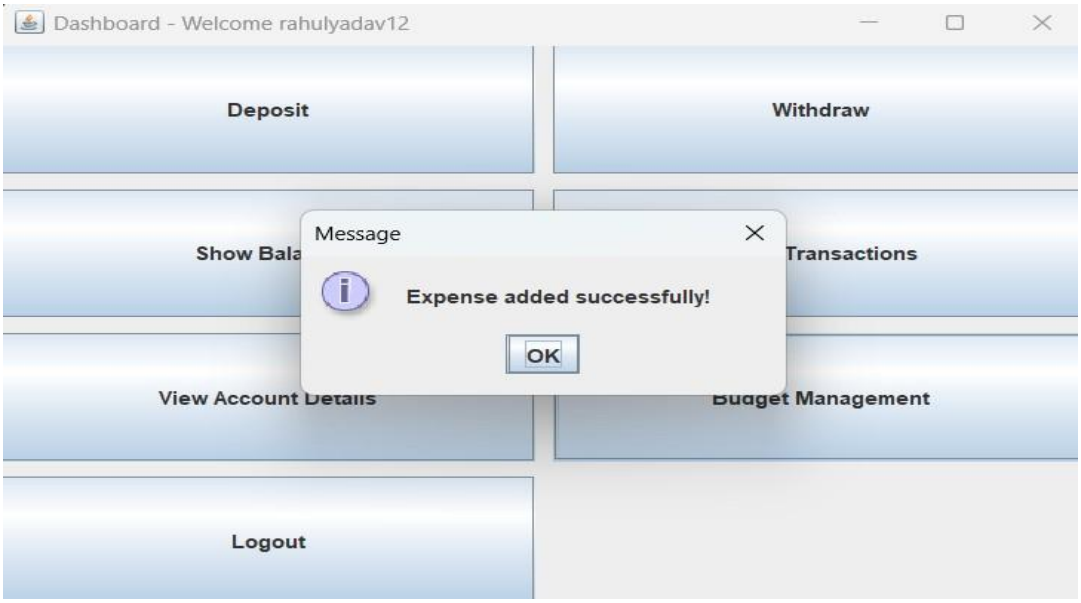
6. Method – addexp(double amount)

OUTPUTS :

Adding Budgets –



Adding Expenses :



Title – Creating the BudgetManager Class to manage multiple budget categories like food , travel .

File – BudgetManager.java

Code :

```
import java.io.*;
import java.util.ArrayList;

public class BudgetManager {     private ArrayList<Budget>
budgets;     private static final String FILE_NAME =
"BUDGETS.txt";

   public BudgetManager() {
this.budgets = new ArrayList<>();
loadBudgets();
   }

   public void addBudget(Budget budget) {
budgets.add(budget);        saveBudgets();
   }

   public void showBudget() {
if (budgets.isEmpty()) {
          System.out.println("No budgets available");
          return;
       }
       for (Budget budget : budgets) {
```

```
            System.out.println("Category : " + budget.getcate()
                    + ", Limit : " + budget.getlimit()
                    + ", Total Expense: " + budget.getexpense());
        }
    }

    private void saveBudgets() {        try (BufferedWriter w = new
BufferedWriter(new FileWriter(FILE_NAME))) {            for (Budget budget :
budgets) {
                w.write(budget.getcate() + "," + budget.getlimit() + "," + budget.getexpense());
w.newLine();
            }
        } catch (IOException e) {
            System.out.println("Error saving budgets: " + e.getMessage());
        }
    }

    private void loadBudgets() {
File f = new File(FILE_NAME);
        if (!f.exists()) return;

        try (BufferedReader r = new BufferedReader(new FileReader(f))) {
            String line;            while ((line = r.readLine()) !=
null) {                String[] parts = line.split(",");                if
(parts.length == 3) {                    String category = parts[0];
double limit = Double.parseDouble(parts[1]);
double expense = Double.parseDouble(parts[2]);
```

```
Budget budget = new Budget(category, limit, expense);
budgets.add(budget);
            }
        }
    } catch (IOException e) {
        System.out.println("Error loading budgets: " + e.getMessage());
    }
  }

  public Budget getBudget(String category) {        for
(Budget budget : budgets) {            if
(budget.getcate().equalsIgnoreCase(category)) {
return budget;
        }        }
return null;
  }

  public void updateExpense(String category, double amount) {
Budget budget = getBudget(category);        if (budget != null) {
budget.addexp(amount);            saveBudgets();
    } else {
        System.out.println("Category not found!");
    }
  }
}
```

Explanation :

addBudget() – Adds a new budget and saves all budgets to file.

showBudget() – Prints all budget details or notifies if empty.

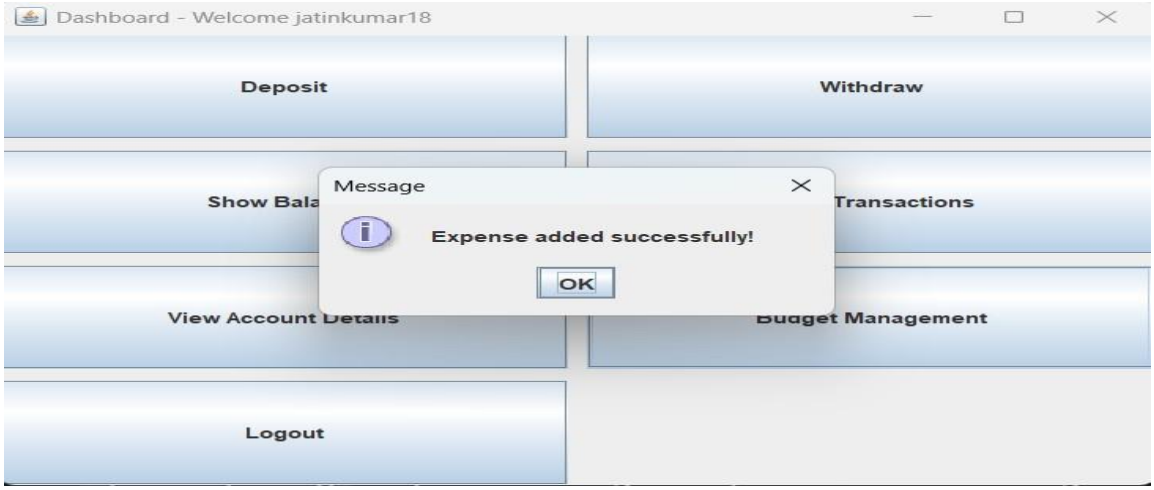
saveBudgets() – Saves all budgets to the file in CSV format.

loadBudgets() – Loads budget data from the file into memory. getBudget()

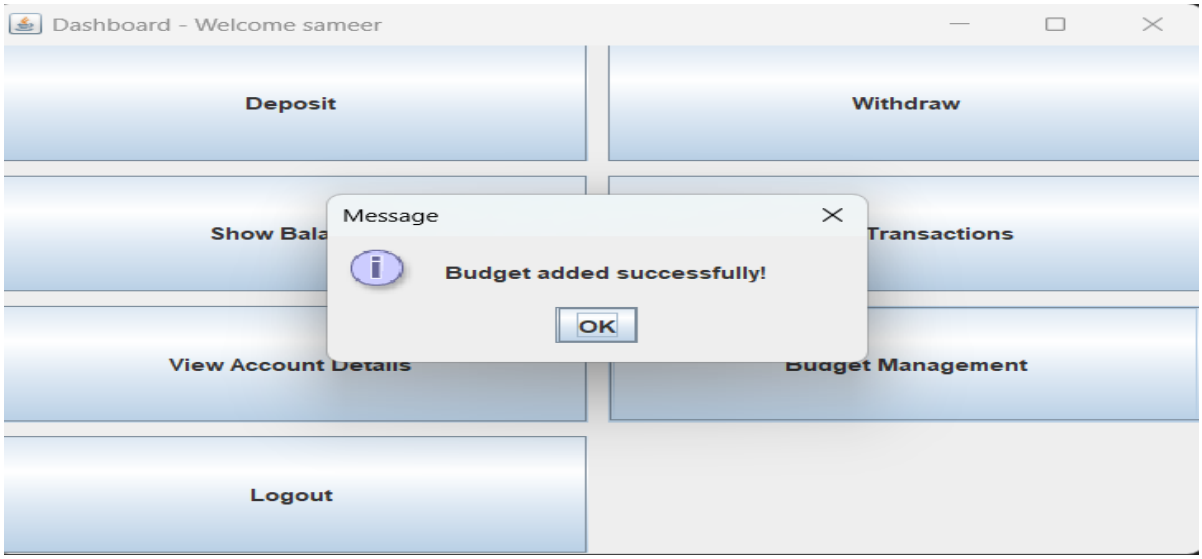
updateExpense() – Updates a budget's expense and saves changes to file.

OUTPUT :

Updating Expenses :



Adding Budgets :



Title – Creating the Transaction Class to represent transactions .
File – Transaction.java

Code :

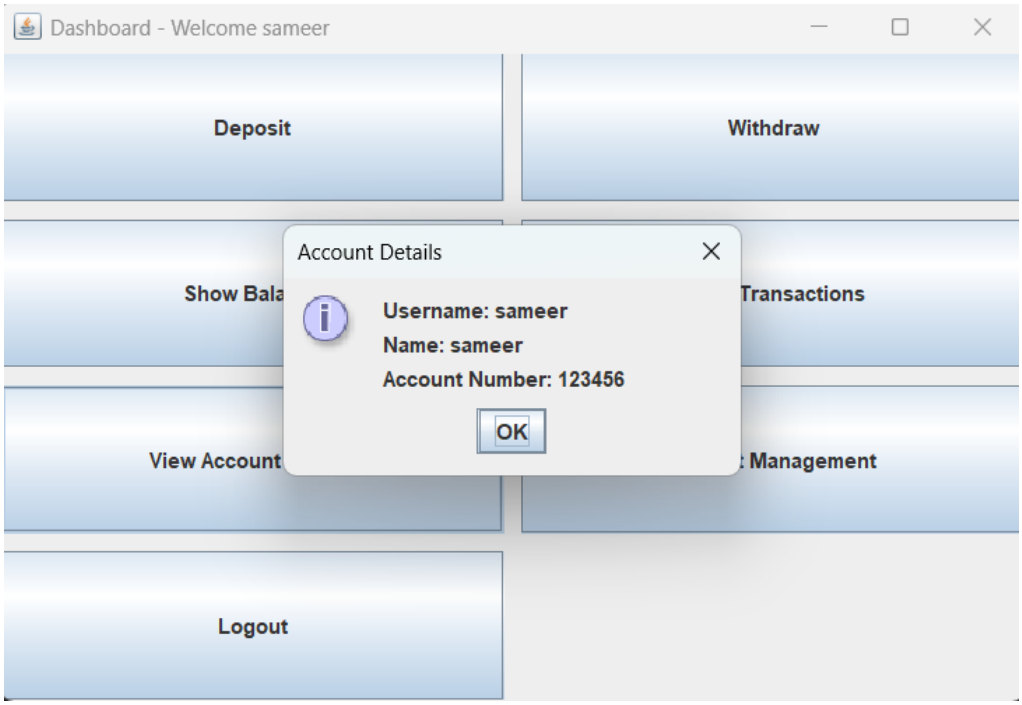
```
public class Transaction {
   private String accnumber;
   private String describe;
   private double amount;

   public Transaction(String accnumber, String describe, double amount){
      this.accnumber = accnumber;
      this.describe = describe;
      this.amount = amount;
   }

   public String getaccnumber() {
      return accnumber;
   }

   public String getdescribe(){
      return describe;
   }

   public double getamount() {
      return amount;
   }

   public String toFileFormat() {
      return accnumber + "," + describe + "," + amount;
   }

}
```

EXPLANATION :

1. accnumber – Stores the account number related to the transaction.
2. describe – Holds a brief description of the transaction
3. amount – Represents the amount involved in the transaction.
4. getaccnumber() – Returns the account number.
5. getdescribe() – Returns the transaction description.
6. getamount() – Returns the transaction amount.

OUTPUT :

Getting the Acc no. -



Transactions.txt :

```
981300100041443 - Deposit - $50000.0 - online
981300100041443 - Withdraw - $4500.0 - cash
981300100041443 - Deposit - $589746.0 - online
981300100041443 - Withdraw - $7795.0 - cash
9813000100041448 - Deposit - $564862.0 - online
```

Title – Creating the User Class to store and provide user account details
File – User.java

Code –

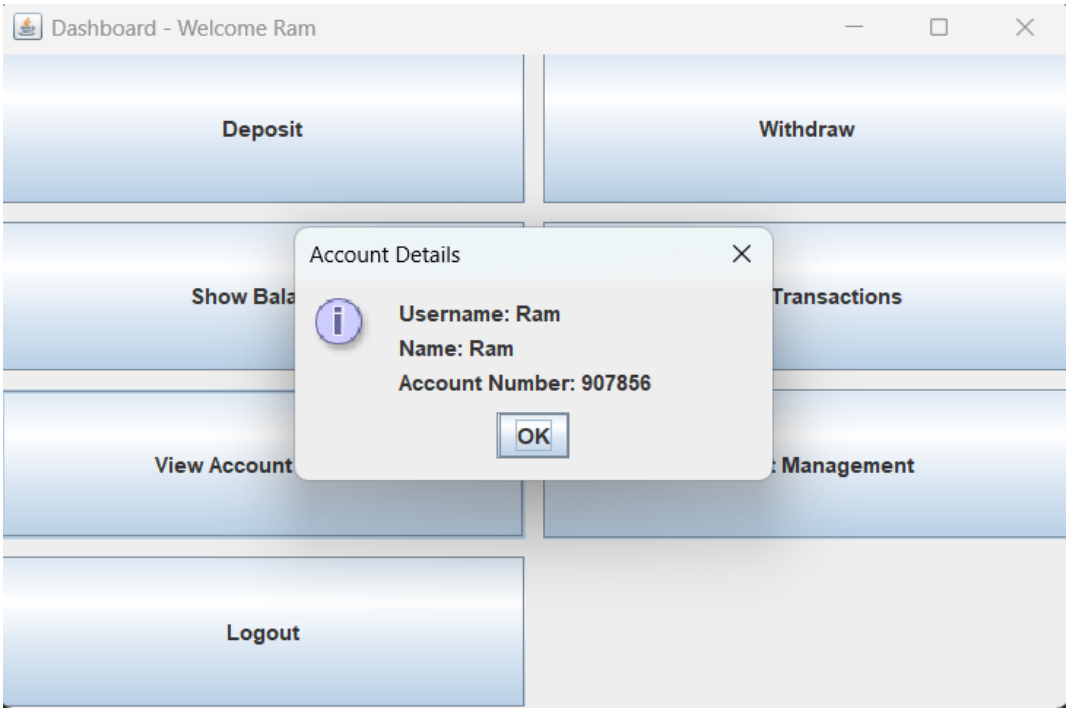
```
import java.io.Serializable;

public class User implements Serializable{
    private String name;
    private String email;
    private String password;
    private String accnum;
    private String phone;

    public User(String name, String email, String password, String accnum){
        this.name = name;
        this.email = email;
        this.password = password;
        this.accnum = accnum;
        // this.phone = phone;
    }

    public String getname(){
        return name;
    }
    public String getemail() {
        return email;
    }
    public String getpassword() {
        return password;
    }
    public String getaccnum() {
        return accnum;
    }
    public String getphone() {
        return phone;
    }

}
```

EXPLANATION:

1.Serializable – Allows User objects to be saved .
2. name, email, password, accnum, phone – Stores the user's personal and account details.
3. getname(), getemail(), getpassword(), getaccnum(), getphone() – Getter methods to access user data from other classes.

OUTPUT :

Showing Details :



User Data :

```
Raghu_01,ragh123@12,Raghvendra Goyal,9813000100041448
Pranjal_01,pra123@123,Pranjal Goyal,9813000100025364
user_01,user123@12,User,98130001002596
sameer,123456,sameer,123456
Ram,456789,Ram,907856
```

Title – Creating the UserManager Class to Manage User Registration and Login

File – UserManager.java

Code –

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JOptionPane;

public class UserManager {
    private static final String FILE_NAME = "Users_data.txt";
    private List<User> users;

    public UserManager() {
        users = new ArrayList<>();
        load();  // custom method to load users from file
    }


    //Duplicate user k liye
    // Check for duplicate username or account number
private boolean isDuplicateUser(String username, String accnum) {
    for (User user : users) {
        if (user.getname().equals(username) || user.getaccnum().equals(accnum)) {
            return true;
        }
    }
    return false;
}

    // Register a new user
public void reguser(String username, String password, String name, String accnum) {
    if (isDuplicateUser(username, accnum)) {
        System.out.println("Error: Username or Account Number already exists. Please try
again with different credentials.");
        return;
    }

    try (BufferedWriter w = new BufferedWriter(new FileWriter(FILE_NAME, true))) {
        w.write(username + "," + password + "," + name + "," + accnum);
        w.newLine();
        users.add(new User(username, password, name, accnum)); // Also add to memory
        System.out.println("User registration successful.");
```

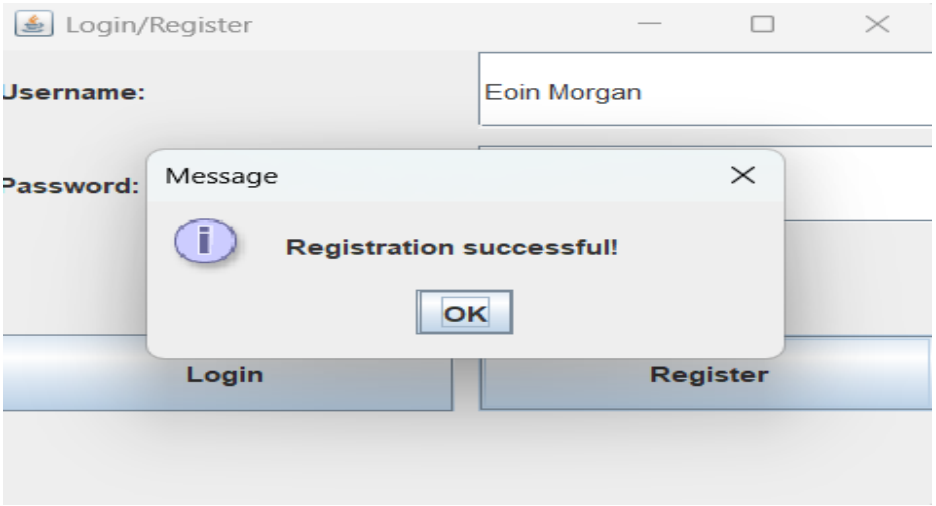
```
    } catch (IOException e) {
        System.out.println("Error while saving user data: " + e.getMessage());
    }
}

// Login method: username + password dono verify hoga
public User login(String username, String password) {
    try (BufferedReader r = new BufferedReader(new FileReader(FILE_NAME))) {
        String line;
        while ((line = r.readLine()) != null) {
            String[] parts = line.split(",");
            if (parts.length == 4 && parts[0].equals(username) && parts[1].equals(password)) {
                return new User(parts[0], parts[1], parts[2], parts[3]);
            }
        }
    } catch (IOException e) {
        System.out.println("Error while reading user data: " + e.getMessage());
    }
    System.out.println("Invalid username or password. Please try again.");
    return null;
}

    // Load all users from file to memory
    private void load() {
        File f = new File(FILE_NAME);
        if (!f.exists()) return;

        try (BufferedReader r = new BufferedReader(new FileReader(f))) {
            String line;
            while ((line = r.readLine()) != null) {
                String[] data = line.split(",");
                if (data.length == 4) {
                    users.add(new User(data[0], data[1], data[2], data[3]));
                }
            }
        } catch (IOException e) {
            System.out.println("Error while loading users: " + e.getMessage());
        }
    }

    // Save all users from memory to file (overwrite)
    private void save() {
        try (BufferedWriter w = new BufferedWriter(new FileWriter(FILE_NAME))) {
            for (User user : users) {
                w.write(user.getname() + "," + user.getpassword() + "," + user.getname() + "," +
user.getaccnum());
```

```
            w.newLine();
        }
    } catch (IOException e) {
        System.out.println("Error while saving user data: " + e.getMessage());
    }
}

// Get user by account number
public User getUserByAccnum(String accnum) {
    for (User user : users) {
        if (user.getaccnum().equals(accnum)) {
            return user;
        }
    }
    return null;
}

public void showAccDetails(String username) {
    try (BufferedReader r = new BufferedReader(new FileReader(FILE_NAME))) {
        String line;
        while ((line = r.readLine()) != null) {
            String[] parts = line.split(",");
            if (parts.length == 4 && parts[0].equals(username)) {
                String message = "Username: " + parts[0] + "\n"
                            + "Name: " + parts[2] + "\n"
                            + "Account Number: " + parts[3];
                JOptionPane.showMessageDialog(null, message, "Account Details",
JOptionPane.INFORMATION_MESSAGE);
                return;
            }
        }
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Error while loading user data: " +
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
    }
    JOptionPane.showMessageDialog(null, "User not found.", "Error",
JOptionPane.ERROR_MESSAGE);
}
}
```

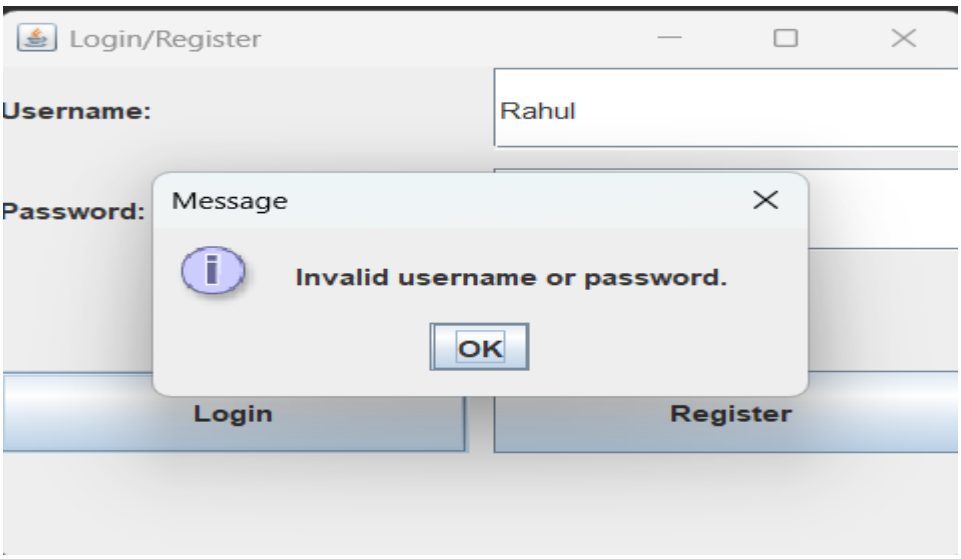
EXPLANATION :

1. UserManager() – Constructor loads existing users from file into memory.
2. isDuplicateUser() – Checks if the username or account number already exists.
3. login() – Authenticates a user by checking the entered username and password.
4. load() – Loads all users from the file Users_data.txt into an in-memory list.
5. getUserByAccnum() – Returns a User object based on the given account number.
6. save() – Saves all users from memory into the file (overwrites existing content).

OUTPUT :
Registration –



Duplicate User Reg . –



NOW HERE ARE THE CODES FOR IMPLEMENTING THE SWING –

Title : This class represents the Dashboard window
File Name: DashboardFrame.java
Code –

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

public class DashboardFrame extends JFrame {
    private User user;
    private UserManager userManager;
    private BankAccManager bankAccManager;
    private BudgetManager budgetManager;

    public DashboardFrame(User user, UserManager userManager, BankAccManager
bankAccManager, BudgetManager budgetManager) {
        this.user = user;
        this.userManager = userManager;
        this.bankAccManager = bankAccManager;
        this.budgetManager = budgetManager;

        setTitle("Dashboard - Welcome " + user.getname());
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        setLayout(new GridLayout(4, 2, 10, 10));

        JButton depositButton = new JButton("Deposit");
        JButton withdrawButton = new JButton("Withdraw");
        JButton balanceButton = new JButton("Show Balance");
        JButton transactionsButton = new JButton("View Transactions");
        JButton accDetailsButton = new JButton("View Account Details");
        JButton budgetButton = new JButton("Budget Management");
        JButton logoutButton = new JButton("Logout");

        add(depositButton);
        add(withdrawButton);
        add(balanceButton);
        add(transactionsButton);
        add(accDetailsButton);
```

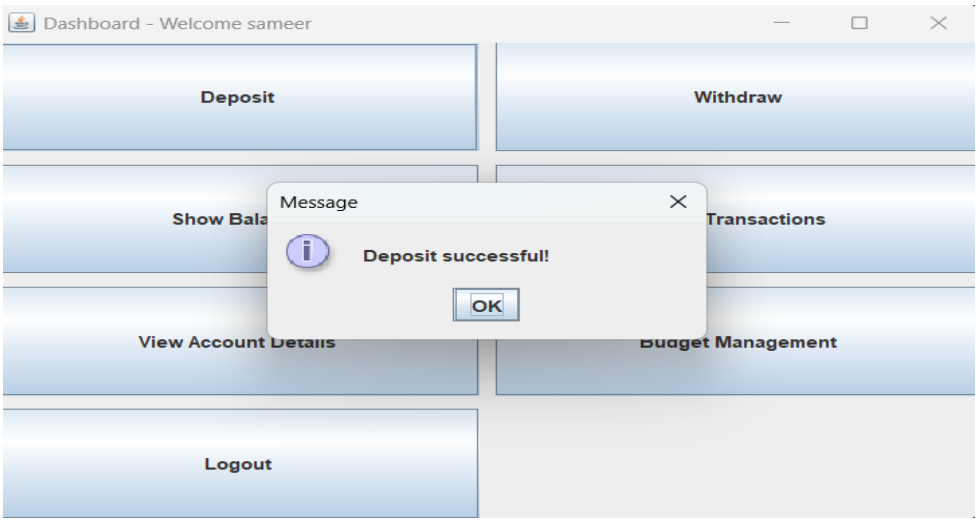
```
    add(budgetButton);
    add(logoutButton);

    depositButton.addActionListener(e -> deposit());
    withdrawButton.addActionListener(e -> withdraw());
    balanceButton.addActionListener(e -> showBalance());
    transactionsButton.addActionListener(e -> showTransactions());
    accDetailsButton.addActionListener(e -> showAccDetails());
    budgetButton.addActionListener(e -> manageBudget());
    logoutButton.addActionListener(e -> logout());

    setVisible(true);
}

private void deposit() {
    String amountStr = JOptionPane.showInputDialog(this, "Enter amount to deposit:");
    String method = JOptionPane.showInputDialog(this, "Enter deposit method:");

    if (amountStr != null && method != null) {
        double amount = Double.parseDouble(amountStr);
        bankAccManager.deposit(user.getaccnum(), amount, method);
        JOptionPane.showMessageDialog(this, "Deposit successful!");
    }
}

private void withdraw() {
    String amountStr = JOptionPane.showInputDialog(this, "Enter amount to withdraw:");
    String method = JOptionPane.showInputDialog(this, "Enter withdrawal method:");

    if (amountStr != null && method != null) {
        double amount = Double.parseDouble(amountStr);
        bankAccManager.withdraw(user.getaccnum(), amount, method);
        JOptionPane.showMessageDialog(this, "Withdrawal successful!");
    }
}

private void showBalance() {
    bankAccManager.showBalance(user.getaccnum());
}

private void showTransactions() {
    bankAccManager.showTransactions(user.getaccnum());
}

private void showAccDetails() {
    userManager.showAccDetails(user.getname());
```

```
    }

    private void manageBudget() {
        String[] options = {"Add Budget", "View Budgets", "Add Expense", "Back"};
        while (true) {
            int choice = JOptionPane.showOptionDialog(this, "Budget Management", "Budget
Menu",
                    JOptionPane.DEFAULT_OPTION, JOptionPane.INFORMATION_MESSAGE,
null, options, options[0]);

            if (choice == 0) {
                String category = JOptionPane.showInputDialog(this, "Enter category:");
                String amountStr = JOptionPane.showInputDialog(this, "Enter budget limit:");
                double amount = Double.parseDouble(amountStr);

                Budget budget = new Budget(category, amount, 0.0);
                budgetManager.addBudget(budget);
                JOptionPane.showMessageDialog(this, "Budget added successfully!");
            } else if (choice == 1) {
                budgetManager.showBudget();
            } else if (choice == 2) {
                String category = JOptionPane.showInputDialog(this, "Enter category:");
                Budget budget = budgetManager.getBudget(category);

                if (budget != null) {
                    String expenseStr = JOptionPane.showInputDialog(this, "Enter expense
amount:");
                    double expense = Double.parseDouble(expenseStr);

                    budget.addexp(expense);
                    JOptionPane.showMessageDialog(this, "Expense added successfully!");
                } else {
                    JOptionPane.showMessageDialog(this, "Budget category not found.");
                }
            } else {
                break;
            }
        }
    }

    private void logout() {
        JOptionPane.showMessageDialog(this, "Logged out successfully.");
        new MainFrame();
        dispose();
    }
}
```

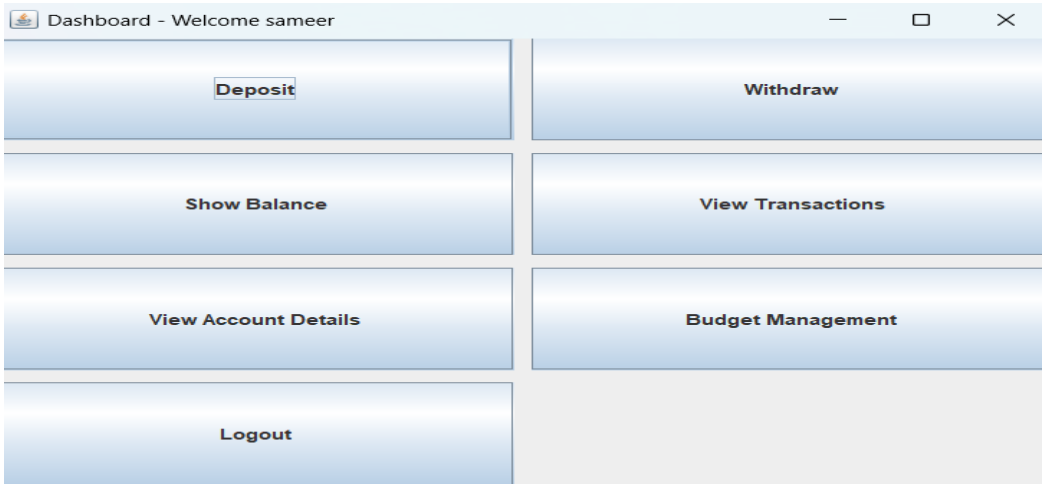
EXPLANATION :

1. DashboardFrame() – Constructor initializes the frame and sets up components.
2. deposit() – Handles the deposit process and updates the account balance.
3. withdraw() – Handles the withdrawal process and updates the account balance.
4. showBalance() – Displays the current balance using BankAccManager.
5.showTransactions() – Displays the transaction history using BankAccManager.
6.showAccDetails() – Displays account details using UserManager.
7.manageBudget() – Manages budget operations like adding budgets, viewing budgets, and adding expenses.
8.logout() – Logs out the user and opens a new MainFrame.

OUTPUT :
Deposit :



Dashboard :



Title : Creating a MainFrame class which give GUI for the registration/login etc.

File : MainFrame.java
Code :

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

public class MainFrame extends JFrame {
    private JTextField usernameField;
    private JPasswordField passwordField;
    private UserManager userManager;
    private BankAccManager bankAccManager;
    private BudgetManager budgetManager;

    public MainFrame() {
        userManager = new UserManager();
        bankAccManager = new BankAccManager();
        budgetManager = new BudgetManager();

        setTitle("Login/Register");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new GridLayout(5, 2, 10, 10));

        JLabel userLabel = new JLabel("Username:");
        usernameField = new JTextField();

        JLabel passLabel = new JLabel("Password:");
        passwordField = new JPasswordField();

        JButton loginButton = new JButton("Login");
        JButton registerButton = new JButton("Register");

        add(userLabel);
        add(usernameField);
        add(passLabel);
        add(passwordField);
        add(new JLabel());
        add(new JLabel());
        add(loginButton);
        add(registerButton);
```

```
        loginButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                login();
            }
        });

        registerButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                register();
            }
        });

        setVisible(true);
    }

private void login() {
    String username = usernameField.getText();
    String password = new String(passwordField.getPassword());

    User user = userManager.login(username, password);

    if (user != null) {
        JOptionPane.showMessageDialog(this, "Login successful!");

        // Open dashboard
        new DashboardFrame(user, userManager, bankAccManager, budgetManager);
        dispose(); // Close login window
    } else {
        JOptionPane.showMessageDialog(this, "Invalid username or password.");
    }
}

    private void register() {
        String username = usernameField.getText();
        String password = new String(passwordField.getPassword());

        if (username.isEmpty() || password.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Username and password cannot be empty.");
            return;
        }

        String name = JOptionPane.showInputDialog(this, "Enter your full name:");
        String accNumber = JOptionPane.showInputDialog(this, "Enter account number:");

        if (name != null && accNumber != null) {
```

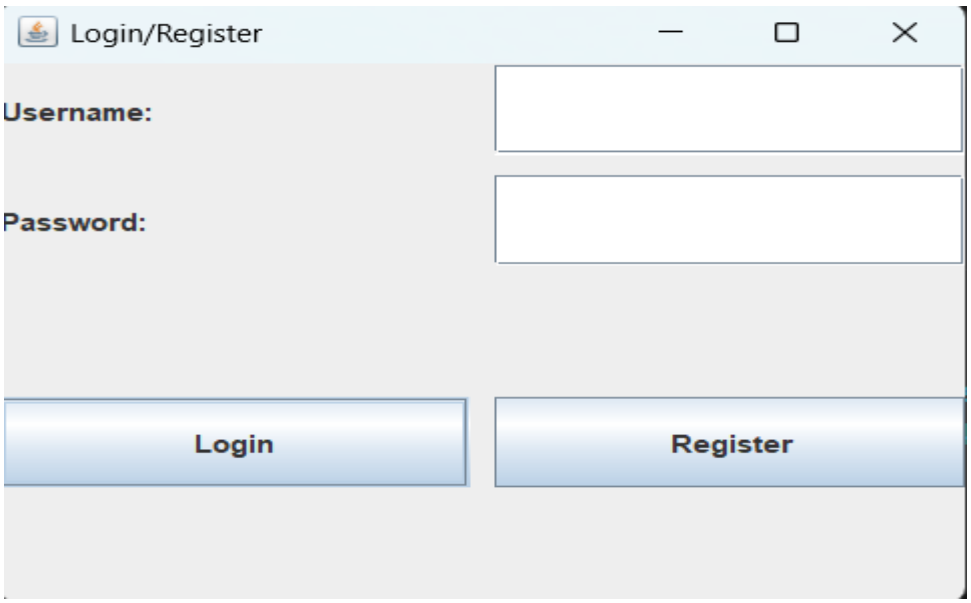
```
            userManager.reguser(username, password, name, accNumber);
            bankAccManager.createacc(accNumber, 0.0);
            JOptionPane.showMessageDialog(this, "Registration successful!");
        } else {
            JOptionPane.showMessageDialog(this, "Registration cancelled.");
        }
    }

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

EXPLANATION :

1.MainFrame() – Initializes the login and registration frame with fields for username and password, along with login and register buttons.
2.login() – Authenticates the user and shows a message dialog: "Login successful!" or "Invalid username or password." Then, opens the DashboardFrame for the user.
3.register() – Registers a new user by asking for a name and account number and shows a message dialog: "Registration successful!" or "Registration cancelled."
4.main() – Creates and shows the main login frame when the application starts.

OUTPUT :

Login/Registration GUI –



Title : Creating the MainApp class.
File : MainApp.java

Code :

```
public class MainApp {
   public static void main(String[] args) {
      new MainFrame();
   }
}
```

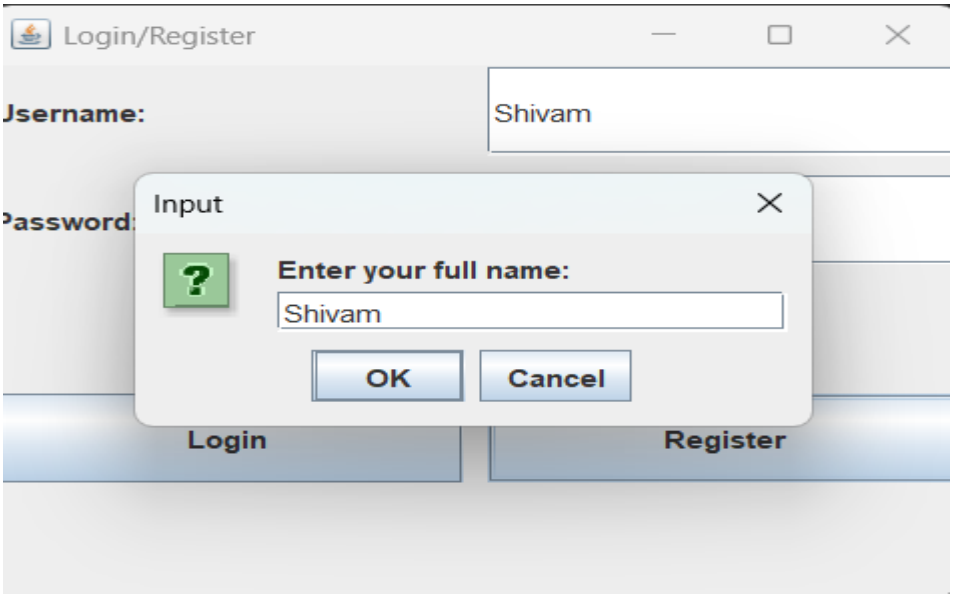
EXPLANATION :

Creates an Instance of MainFrame:
1.The main method calls new MainFrame();, which creates a new instance of the MainFrame class.

2.This will initialize the login and registration window of your application by executing the constructor of the MainFrame class.

OUTPUT :

Registraion GUI :



Dashboard GUI :