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

// Abstract Account class
abstract class Account {
    protected String accountNumber;
    protected double balance;
    protected String branch;

    public Account(String accountNumber, double initialBalance, String branch) {
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
        this.branch = branch;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }

    public double getBalance() { return balance; }
    public String getAccountNumber() { return accountNumber; }
    public String getBranch() { return branch; }

    public abstract void calculateInterest();
    public abstract String getAccountType();
}

// SavingsAccount class
class SavingsAccount extends Account {
    private static final double INTEREST_RATE = 0.0005; // 0.05% monthly

    public SavingsAccount(String accountNumber, double initialBalance, String branch) {
        super(accountNumber, initialBalance, branch);
    }

    @Override
    public void calculateInterest() {
        double interest = balance * INTEREST_RATE;
        balance += interest;
    }

    @Override
    public String getAccountType() { return "Savings"; }
}

// InvestmentAccount class
class InvestmentAccount extends Account {
    private static final double INTEREST_RATE = 0.05; // 5% monthly
    private static final double MIN_INITIAL_DEPOSIT = 500.00;

    public InvestmentAccount(String accountNumber, double initialBalance, String branch) {
        super(accountNumber, initialBalance, branch);
        if (initialBalance < MIN_INITIAL_DEPOSIT) {
            throw new IllegalArgumentException("Investment Account requires minimum BWP500.00");
        }
    }

    public boolean withdraw(double amount) {
        if (amount > 0 && balance >= amount) {
            balance -= amount;
            return true;
        }
        return false;
    }
    @Override
    public void calculateInterest() {
        double interest = balance * INTEREST_RATE;
        balance += interest;
    }

    @Override
    public String getAccountType() { return "Investment"; }
}

// ChequeAccount class
class ChequeAccount extends Account {
    private String companyName;
    private String companyAddress;

    public ChequeAccount(String accountNumber, double initialBalance, String branch,
                        String companyName, String companyAddress) {
        super(accountNumber, initialBalance, branch);
        this.companyName = companyName;
        this.companyAddress = companyAddress;
    }

    public boolean withdraw(double amount) {
        if (amount > 0 && balance >= amount) {
            balance -= amount;
            return true;
        }
        return false;
    }

    public String getCompanyName() { return companyName; }
    public String getCompanyAddress() { return companyAddress; }

    @Override
    public void calculateInterest() {
        // No interest for cheque accounts
    }

    @Override
    public String getAccountType() { return "Cheque"; }
}

// Customer class
class Customer {
    private String firstName;
    private String lastName;
    private String address;
    private ArrayList<Account> accounts;

    public Customer(String firstName, String lastName, String address) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.address = address;
        this.accounts = new ArrayList<>();
    }

    public void addAccount(Account account) {
        accounts.add(account);
    }

    public ArrayList<Account> getAccounts() { return accounts; }
    public String getFirstName() { return firstName; }
    public String getLastName() { return lastName; }
    public String getAddress() { return address; }
    public String getFullName() { return firstName + " " + lastName; }
}

// Main GUI Application
class BankingSystemGUI extends JFrame {
    private ArrayList<Customer> customers;
    private Customer currentCustomer;

    private JTextField firstNameField, lastNameField, addressField;
    private JComboBox<String> accountTypeCombo;
    private JTextField initialDepositField, companyNameField, companyAddressField;
    private JList<String> accountList;
    private DefaultListModel<String> accountListModel;
    private JTextArea outputArea;

    public BankingSystemGUI() {
        customers = new ArrayList<>();
        initializeGUI();
    }

    private void initializeGUI() {
        setTitle("Banking System");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Create panels
        JPanel customerPanel = createCustomerPanel();
        JPanel accountPanel = createAccountPanel();
        JPanel operationsPanel = createOperationsPanel();
        JPanel outputPanel = createOutputPanel();

        // Add panels to main frame
        add(customerPanel, BorderLayout.NORTH);
        add(accountPanel, BorderLayout.WEST);
        add(operationsPanel, BorderLayout.CENTER);
        add(outputPanel, BorderLayout.SOUTH);

        pack();
        setLocationRelativeTo(null);
    }

    private JPanel createCustomerPanel() {
        JPanel panel = new JPanel(new GridBagLayout());
        panel.setBorder(BorderFactory.createTitledBorder("Customer Information"));
        GridBagConstraints gbc = new GridBagConstraints();

        gbc.insets = new Insets(5, 5, 5, 5);

        gbc.gridx = 0; gbc.gridy = 0;
        panel.add(new JLabel("First Name:"), gbc);
        gbc.gridx = 1;
        firstNameField = new JTextField(15);
        panel.add(firstNameField, gbc);

        gbc.gridx = 2;
        panel.add(new JLabel("Last Name:"), gbc);
        gbc.gridx = 3;
        lastNameField = new JTextField(15);
        panel.add(lastNameField, gbc);

        gbc.gridx = 0; gbc.gridy = 1;
        panel.add(new JLabel("Address:"), gbc);
        gbc.gridx = 1; gbc.gridwidth = 2;
        addressField = new JTextField(30);
        panel.add(addressField, gbc);

        gbc.gridx = 3; gbc.gridwidth = 1;
        JButton createCustomerBtn = new JButton("Create Customer");
        createCustomerBtn.addActionListener(e -> createCustomer());
        panel.add(createCustomerBtn, gbc);

        return panel;
    }

    private JPanel createAccountPanel() {
        JPanel panel = new JPanel(new BorderLayout());
        panel.setBorder(BorderFactory.createTitledBorder("Account Management"));

        // Account creation form
        JPanel createPanel = new JPanel(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 5, 5, 5);

        gbc.gridx = 0; gbc.gridy = 0;
        createPanel.add(new JLabel("Account Type:"), gbc);
        gbc.gridx = 1;
        accountTypeCombo = new JComboBox<>(new String[]{"Savings", "Investment", "Cheque"});
        accountTypeCombo.addActionListener(e -> toggleCompanyFields());
        createPanel.add(accountTypeCombo, gbc);

        gbc.gridx = 0; gbc.gridy = 1;
        createPanel.add(new JLabel("Initial Deposit:"), gbc);
        gbc.gridx = 1;
        initialDepositField = new JTextField(10);
        createPanel.add(initialDepositField, gbc);

        gbc.gridx = 0; gbc.gridy = 2;
        createPanel.add(new JLabel("Company Name:"), gbc);
        gbc.gridx = 1;
        companyNameField = new JTextField(10);
        companyNameField.setEnabled(false);
        createPanel.add(companyNameField, gbc);

        gbc.gridx = 0; gbc.gridy = 3;
        createPanel.add(new JLabel("Company Address:"), gbc);
        gbc.gridx = 1;
        companyAddressField = new JTextField(10);
        companyAddressField.setEnabled(false);
        createPanel.add(companyAddressField, gbc);

        gbc.gridx = 0; gbc.gridy = 4; gbc.gridwidth = 2;
        JButton createAccountBtn = new JButton("Create Account");
        createAccountBtn.addActionListener(e -> createAccount());
        createPanel.add(createAccountBtn, gbc);

        panel.add(createPanel, BorderLayout.NORTH);

        // Account list
        accountListModel = new DefaultListModel<>();
        accountList = new JList<>(accountListModel);
        accountList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        JScrollPane scrollPane = new JScrollPane(accountList);
        scrollPane.setPreferredSize(new Dimension(300, 150));
        panel.add(scrollPane, BorderLayout.CENTER);

        return panel;
    }

    private JPanel createOperationsPanel() {
        JPanel panel = new JPanel(new GridBagLayout());
        panel.setBorder(BorderFactory.createTitledBorder("Account Operations"));
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(10, 10, 10, 10);

        gbc.gridx = 0; gbc.gridy = 0;
        JButton depositBtn = new JButton("Make Deposit");
        depositBtn.addActionListener(e -> makeDeposit());
        panel.add(depositBtn, gbc);

        gbc.gridy = 1;
        JButton withdrawBtn = new JButton("Make Withdrawal");
        withdrawBtn.addActionListener(e -> makeWithdrawal());
        panel.add(withdrawBtn, gbc);

        gbc.gridy = 2;
        JButton interestBtn = new JButton("Calculate Interest");
```

```java
        interestBtn.addActionListener(e -> calculateInterest());
        panel.add(interestBtn, gbc);

        gbc.gridy = 3;
        JButton balanceBtn = new JButton("Check Balance");
        balanceBtn.addActionListener(e -> checkBalance());
        panel.add(balanceBtn, gbc);

        return panel;
    }

    private JPanel createOutputPanel() {
        JPanel panel = new JPanel(new BorderLayout());
        panel.setBorder(BorderFactory.createTitledBorder("Output"));

        outputArea = new JTextArea(8, 50);
        outputArea.setEditable(false);
        outputArea.setFont(new Font(Font.MONOSPACED, Font.PLAIN, 12));
        JScrollPane scrollPane = new JScrollPane(outputArea);
        panel.add(scrollPane, BorderLayout.CENTER);

        return panel;
    }

    private void toggleCompanyFields() {
        boolean isCheque = "Cheque".equals(accountTypeCombo.getSelectedItem());
        companyNameField.setEnabled(isCheque);
        companyAddressField.setEnabled(isCheque);
    }

    private void createCustomer() {
        String firstName = firstNameField.getText().trim();
        String lastName = lastNameField.getText().trim();
        String address = addressField.getText().trim();

        if (firstName.isEmpty() || lastName.isEmpty() || address.isEmpty()) {
            showMessage("Please fill all customer fields.");
            return;
        }

        currentCustomer = new Customer(firstName, lastName, address);
        customers.add(currentCustomer);

        showMessage("Customer created: " + currentCustomer.getFullName());

        // Clear fields
        firstNameField.setText("");
        lastNameField.setText("");
        addressField.setText("");

        updateAccountList();
    }

    private void createAccount() {
        if (currentCustomer == null) {
            showMessage("Please create a customer first.");
            return;
        }

        String accountType = (String) accountTypeCombo.getSelectedItem();
        String depositStr = initialDepositField.getText().trim();

        if (depositStr.isEmpty()) {
            showMessage("Please enter initial deposit amount.");
            return;
        }

        try {
            double initialDeposit = Double.parseDouble(depositStr);
            String accountNumber = "ACC" + System.currentTimeMillis();
            String branch = "Main Branch";

            Account account = null;

            switch (accountType) {
                case "Savings":
                    account = new SavingsAccount(accountNumber, initialDeposit, branch);
                    break;
                case "Investment":
                    account = new InvestmentAccount(accountNumber, initialDeposit, branch);
                    break;
                case "Cheque":
                    String companyName = companyNameField.getText().trim();
                    String companyAddress = companyAddressField.getText().trim();
                    if (companyName.isEmpty() || companyAddress.isEmpty()) {
                        showMessage("Please fill company information for Cheque account.");
                        return;
                    }
                    account = new ChequeAccount(accountNumber, initialDeposit, branch, companyName,
companyAddress);
                    break;
            }

            currentCustomer.addAccount(account);
            showMessage("Account created successfully: " + account.getAccountNumber());

            // Clear fields
            initialDepositField.setText("");
            companyNameField.setText("");
            companyAddressField.setText("");

            updateAccountList();

        } catch (NumberFormatException e) {
            showMessage("Invalid deposit amount.");
        } catch (IllegalArgumentException e) {
            showMessage(e.getMessage());
        }
    }

    private void makeDeposit() {
        Account selectedAccount = getSelectedAccount();
        if (selectedAccount == null) return;

        String amountStr = JOptionPane.showInputDialog(this, "Enter deposit amount:");
        if (amountStr != null && !amountStr.trim().isEmpty()) {
            try {
                double amount = Double.parseDouble(amountStr);
                selectedAccount.deposit(amount);
                showMessage("Deposited BWP" + amount + ". New balance: BWP" + selectedAccount.getBalance());
                updateAccountList();
            } catch (NumberFormatException e) {
                showMessage("Invalid amount.");
            }
        }
    }

    private void makeWithdrawal() {
        Account selectedAccount = getSelectedAccount();
        if (selectedAccount == null) return;

        if (selectedAccount instanceof SavingsAccount) {
            showMessage("Withdrawals not allowed from Savings Account.");
            return;
        }

        String amountStr = JOptionPane.showInputDialog(this, "Enter withdrawal amount:");
        if (amountStr != null && !amountStr.trim().isEmpty()) {
            try {
                double amount = Double.parseDouble(amountStr);
```

```java
                boolean success = false;

                if (selectedAccount instanceof InvestmentAccount) {
                    success = ((InvestmentAccount) selectedAccount).withdraw(amount);
                } else if (selectedAccount instanceof ChequeAccount) {
                    success = ((ChequeAccount) selectedAccount).withdraw(amount);
                }

                if (success) {
                    showMessage("Withdrew BWP" + amount + ". New balance: BWP" +
selectedAccount.getBalance());
                    updateAccountList();
                } else {
                    showMessage("Withdrawal failed. Insufficient funds or invalid amount.");
                }
            } catch (NumberFormatException e) {
                showMessage("Invalid amount.");
            }
        }
    }

    private void calculateInterest() {
        if (currentCustomer == null) {
            showMessage("No customer selected.");
            return;
        }

        for (Account account : currentCustomer.getAccounts()) {
            double oldBalance = account.getBalance();
            account.calculateInterest();
            double interest = account.getBalance() - oldBalance;

            if (interest > 0) {
                showMessage("Interest added to " + account.getAccountType() + " Account: BWP" +
                           String.format("%.2f", interest));
            }
        }
        updateAccountList();
    }

    private void checkBalance() {
        Account selectedAccount = getSelectedAccount();
        if (selectedAccount == null) return;

        showMessage("Account Balance: BWP" + selectedAccount.getBalance());
    }

    private Account getSelectedAccount() {
        if (currentCustomer == null) {
            showMessage("No customer selected.");
            return null;
        }

        int selectedIndex = accountList.getSelectedIndex();
        if (selectedIndex == -1) {
            showMessage("Please select an account.");
            return null;
        }

        return currentCustomer.getAccounts().get(selectedIndex);
    }

    private void updateAccountList() {
        accountListModel.clear();
        if (currentCustomer != null) {
            for (Account account : currentCustomer.getAccounts()) {
                accountListModel.addElement(account.getAccountType() + " - " +
                                          account.getAccountNumber() + " (BWP" + account.getBalance() + ")");
            }
        }
    }

    private void showMessage(String message) {
        outputArea.append(message + "\n");
        outputArea.setCaretPosition(outputArea.getDocument().getLength());
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new BankingSystemGUI().setVisible(true);
        });
    }
}
```