PROGRAMMING EXAMPLE: Checking Account Balance

A local bank in your town is looking for someone to write a program that calculates a customer's checking account balance at the end of each month. The data is stored in a file in the following form:

```
467343 23750.40
W 250.00
D 1200.00
W 75.00
W 375.00
D 580.00
I 75.50
.
.
.
```

The first line of data shows the account number, followed by the account balance at the beginning of the month. Thereafter, each line has two entries: the transaction code and the transaction amount. The transaction code `W` or `w` means withdrawal, `D` or `d` means deposit, and `I` or `i` means interest paid by the bank. The program updates the balance after each transaction. If at any time during the month the balance goes below $1000.00, a $25.00 service fee is charged for the month. The program prints the following information: account number, balance at the beginning of the month, balance at the end of the month, interest paid by the bank, total amount of deposit, number of deposits, total amount of withdrawal, number of withdrawals, and service charge if any.

Input: A file consisting of data in the above format

Output: The output is of the following form:

```
Account Number: 467343
Beginning Balance: $23750.40
Ending Balance: $24320.75

Interest Paid: $75.50

Amount Deposited: $2230.50
Number of Deposits: 3

Amount Withdrawn: $1735.65
Number of Withdrawals: 6
```

Note that the output is to be sent to a file.

PROBLEM ANALYSIS AND ALGORITHM DESIGN

The first entries in the input file are the account number and the beginning balance. Therefore, the program first reads the account number and the beginning balance. Thereafter, each entry in the file is of the following form:

```
transactionCode transactionAmount
```

To determine the account balance at the end of the month, you need to process each entry that contains the transaction code and the transaction amount. Begin with the starting balance and then update the account balance after processing each entry. If the transaction code is `D`, `d`, `I`, or `i`, the transaction amount is added to the account balance; if the transaction code is `W` or `w`, the transaction amount is subtracted from the balance. Because the program also outputs the number of withdrawals and deposits, you need to keep separate counts of the withdrawals and deposits. This discussion translates into the following algorithm:

1. Declare the necessary variables and objects.
2. Initialize the variables.
3. Get the account number and beginning balance.
4. Get the transaction code and transaction amount.
5. Analyze the transaction code and update the appropriate variables.
6. Repeat Steps 4 and 5 until there is no more data.
7. Print the result.

VARIABLES AND OBJECTS

The program outputs the account number, beginning balance, balance at the end of the month, interest paid, amount deposited, number of deposits, amount withdrawn, number of withdrawals, and service charge, if any. So far, you need the following variables to store all this information:

```
acctNumber           //variable to store the account number
beginningBalance     //variable to store the beginning balance
accountBalance       //variable to store the account balance
                     //at the end of the month
amountDeposited      //variable to store the total amount
                     //deposited
numberOfDeposits     //variable to store the number of deposits
amountWithdrawn      //variable to store the total amount
                     //withdrawn
numberOfWithdrawals  //variable to store the number of
                     //withdrawals
interestPaid         //variable to store the interest amount
                     //paid
```

Because the program reads the data from a file and the output is stored in a file, the program needs both input and output stream objects. After the first line, the data in each line is the transaction code and the transaction amount; the program needs a variable to store this information.

Whenever the account balance goes below the minimum balance, a service charge for that month is applied. After each withdrawal, you need to check the account balance. If the balance goes below the minimum after a withdrawal, a service charge is applied. You can potentially have several withdrawals in a month; once the account balance goes below the minimum, a subsequent deposit might bring the balance above the minimum, and another withdrawal might again reduce it below the minimum. However, the service charge is applied only once.

To implement this idea, the program uses a `boolean` variable, `isServiceCharged`, which is initialized to `false` and set to `true` whenever the account balance goes below the minimum. Before applying a service charge, the program checks the value of the variable `isServiceCharged`. If the account balance is less than the minimum and `isServiceCharged` is `false`, a service charge is applied. The program needs the following variables:

```
int    acctNumber;
double beginningBalance;
double accountBalance;

double amountDeposited;
int    numberOfDeposits;

double amountWithdrawn;
int    numberOfWithdrawals;

double interestPaid;
char   transactionCode;
double transactionAmount;

boolean isServiceCharged;
```

Suppose the input data is in the file `money.txt`. Also suppose that the output will be stored in the file `money.out`. The following statements create the appropriate input and output stream objects:

```
Scanner inFile = new Scanner(new FileReader("money.txt"));
PrintWriter outFile = new PrintWriter("money.out");
```

NAMED CONSTANTS

The minimum account balance and the service charge amount are fixed, so the program uses two named constants to store them:

```
static final double MINIMUM_BALANCE = 1000.00;
static final double SERVICE_CHARGE = 25.00;
```

Because this program is more complex than previous ones, before writing the main algorithm, the preceding seven steps are described more fully here:

1. **Declare the variables and objects:** Declare the variables and objects as discussed previously.
2. **Initialization:** After each deposit, the total amount deposited is updated and the number of deposits is incremented by one. Before the first deposit, the total amount deposited is `0` and the number of withdrawals is `0`. Therefore, the variables `amountDeposited` and `numberOfDeposits` must be initialized to `0`. Similarly, the variables `amountWithdrawn`, `numberOfWithdrawals`, and `interestPaid` must be initialized to `0`. Also, as discussed earlier, the variable `isServiceCharged` is initialized to `false`. Of course, you can initialize these variables when you declare them.

   Before the first deposit, withdrawal, or interest paid, the account balance is the same as the beginning balance. Therefore, after reading the beginning balance in the variable `beginningBalance` from the file, you need to initialize the variable `accountBalance` to the value of the variable `beginningBalance`.
3. **Get the account number and starting balance:** This is accomplished by the following statements:

   ```
   acctNumber = inFile.nextInt();
   beginningBalance = inFile.nextDouble();
   ```
4. **Get the transaction code and transaction amount:** This is accomplished by the following input statement:

   ```
   transactionCode = inFile.next().charAt(0);
   transactionAmount = inFile.nextDouble();
   ```
5. **Analyze the transaction code and update the appropriate variables:** If `transactionCode` is `'D'` or `'d'`, update `accountBalance` by adding `transactionAmount`, update `amountDeposited` by adding `transactionAmount`, and increment `numberOfDeposits`. If `transactionCode` is `'I'` or `'i'`, update `accountBalance` by adding `transactionAmount` and update `interestPaid` by adding `transactionAmount`. If `transactionCode` is `'W'` or `'w'`, update `accountBalance` by subtracting `transactionAmount`, update `amountWithdrawn` by adding `transactionAmount`, increment `numberOfWithdrawals`, and if the account balance is below the minimum and service charges have not been applied, subtract the service charge from the account balance and mark the service charge as having been applied. If `transactionCode` is other than `'D'`, `'d'`, `'I'`, `'i'`, `'W'`, or `'w'`, then the program outputs an error message. The following `switch` statement does this:

```
switch (transactionCode)
{
case 'D':
case 'd':
    accountBalance = accountBalance + transactionAmount;
    amountDeposited = amountDeposited + transactionAmount;
    numberOfDeposits++;
    break;

case 'I':
case 'i':
    accountBalance = accountBalance + transactionAmount;
    interestPaid = interestPaid + transactionAmount;
    break;

case 'W':
case 'w':
    accountBalance = accountBalance - transactionAmount;
    amountWithdrawn = amountWithdrawn + transactionAmount;
    numberOfWithdrawals++;

    if ((accountBalance < MINIMUM_BALANCE) && (!isServiceCharged))
    {
        accountBalance = accountBalance - SERVICE_CHARGE;
        isServiceCharged = true;
    }
    break;

default:
    System.out.println("Invalid transaction code: "
                    + transactionCode    + " " + transactionAmount);
}//end switch
```

6. **Repeat Steps 4 and 5 until there is no more data:** Because the number of entries in the input file is not known, the program needs an EOF-controlled `while` loop.

7. **Print the result:** This is accomplished by using output statements.

The preceding discussion translates into the following algorithm:

MAIN ALGORITHM

1. Declare and initialize the variables and the input and output stream objects.
2. Get `accountNumber` and `beginningBalance`.
3. Set `accountBalance` to `beginningBalance`.
4. While (not end of input file):
    a. Get `transactionCode`.
    b. Get `transactionAmount`.

c. If `transactionCode is 'D'` or `'d'`:
   i. Add `transactionAmount` to `accountBalance`.
   ii. Add `transactionAmount` to `amountDeposited`.
   iii. Increment `numberOfDeposits`.

d. If `transactionCode is 'I'` or `'i'`:
   i. Add `transactionAmount` to `accountBalance`.
   ii. Add `transactionAmount` to `interestPaid`.

e. If `transactionCode is 'W'` or `'w'`:
   i. Subtract `transactionAmount` from `accountBalance`.
   ii. Add `transactionAmount` to `amountWithdrawn`.
   iii. Increment `numberOfWithDrawals`.
   iv. If `(accountBalance < MINIMUM_BALANCE && !isServicedCharged)`:
      1. `Subtract SERVICE_CHARGE` from `accountBalance`.
      2. Set `isServiceCharged` to `true`.

f. If `transactionCode` is other than `'D'`, `'d'`, `'I'`, `'i'`, `'W'`, or `'w'`, output an error message.

5. Output the results.

COMPLETE PROGRAM LISTING

```
//*********************************************************
// Author: D.S. Malik
//
// Program: Checking Account Balance
// This program calculates a customer's checking account
// balance at the end of the month.
//*********************************************************

import java.io.*;
import java.util.*;

public class CheckingAccountBalance
{
    static final double MINIMUM_BALANCE = 1000.00;
    static final double SERVICE_CHARGE = 25.00;
```

```
public static void main(String[] args)
                        throws FileNotFoundException
{
      //Declare and initialize variables        //Step 1
    int    acctNumber;
    double beginningBalance;
    double accountBalance;

    double amountDeposited = 0.0;
    int    numberOfDeposits = 0;

    double amountWithdrawn = 0.0;
    int    numberOfWithdrawals = 0;

    double interestPaid = 0.0;

    char   transactionCode;
    double transactionAmount;

    boolean isServiceCharged = false;

    Scanner inFile = new Scanner(new FileReader("money.txt"));

    PrintWriter outFile = new PrintWriter("money.out");

    acctNumber = inFile.nextInt();                    //Step 2
    beginningBalance = inFile.nextDouble();           //Step 2

    accountBalance = beginningBalance;                //Step 3

    while (inFile.hasNext())                          //Step 4
    {
        transactionCode = inFile.next().charAt(0);    //Step 4a
        transactionAmount = inFile.nextDouble();      //Step 4b

        switch (transactionCode)
        {
        case 'D':                                     //Step 4c
        case 'd':
            accountBalance = accountBalance
                           + transactionAmount;
            amountDeposited = amountDeposited
                            + transactionAmount;
            numberOfDeposits++;
            break;

        case 'I':                                     //Step 4d
        case 'i':
            accountBalance = accountBalance
                           + transactionAmount;
            interestPaid = interestPaid
                         + transactionAmount;
            break;
```

```
                    case 'W':                                          //Step 4e
                    case 'w':
                        accountBalance = accountBalance
                                       - transactionAmount;
                        amountWithdrawn = amountWithdrawn
                                        + transactionAmount;
                        numberOfWithdrawals++;

                        if ((accountBalance < MINIMUM_BALANCE)
                                        && (!isServiceCharged))
                        {
                             accountBalance = accountBalance
                                            - SERVICE_CHARGE;
                             isServiceCharged = true;
                        }
                        break;

                    default:
                         System.out.println("Invalid transaction code: "
                                        + transactionCode + " "
                                        + transactionAmount); //Step 4f
                    }//end switch

                }//end while

                        //Output results    //Step 5
                outFile.printf("Account Number: %d%n", acctNumber);
                outFile.printf("Beginning Balance: $%.2f %n",
                               beginningBalance);
                outFile.printf("Ending Balance: $%.2f %n",
                               accountBalance);
                outFile.println();
                outFile.printf("Interest Paid: $%.2f %n",
                               interestPaid);
                outFile.println();
                outFile.printf("Amount Deposited: $%.2f %n",
                               amountDeposited);
                outFile.printf("Number of Deposits: %d%n",
                               numberOfDeposits);
                outFile.println();
                outFile.printf("Amount Withdrawn: $%.2f %n",
                               amountWithdrawn);
                outFile.printf("Number of Withdrawals: %d%n",
                               numberOfWithdrawals);
                outFile.println();

                if (isServiceCharged)
                    outFile.printf("Service Charge: $%.2f %n",
                                   SERVICE_CHARGE);

                outFile.close();
            }
        }
```

Sample Run: Contents of the output file `money.out`:

```
Account Number: 467343
Beginning Balance: $23750.40
Ending Balance: $24320.75

Interest Paid: $75.50

Amount Deposited: $2230.50
Number of Deposits: 3

Amount Withdrawn: $1735.65
Number of Withdrawals: 6
```

Input File: `money.txt`

```
467343 23750.40
W 250.00
D 1200.00
W 75.00
W 375.00
D 580.00
I 75.50
W 400.00
W 600.00
D 450.50
W 35.65
```