**COP 3855 – Web Systems Development**

**C# Programming In-Class Activity**

**Loan Application**

This example demonstrates the use of loops in the analysis, design, and implementation of a program. Both pretest and posttest forms of loops are included in the example. The pretest form is used to calculate individual loan details. It is nested inside the posttest through method calls. Static and instance methods, properties, and selection statements are included. Below figure outlines the problem specification.

You should review the problem specification in below figure and make sure you understand the problem definition. Several values are put into the program to represent the loan amount, rate, and the time period of the loan. These values are entered as string variables and then parsed or converted into numeric fields so the calculations can be performed.

Image shows the problem specification for the Loan Application example. The specification is described below.
Create and application that will allow a loan amount, interest rate, and number of finance years to be entered for a given loan. Determine the monthly payment amount. Calculate how much interest will be paid over the life of the loan. Display an amortization schedule showing the new balance after each payment is made. 
Design an object oriented solution. Use two classes. For the Loan class, characteristics such as the amount to be financed, rate of interest, period of time for the loan, and total interest paid will identify the current state of a loan object. Include methods to determine the monthly payment amount, return the total interest paid over the life of the loan, and return an amortization schedule.
In the second class, instantiate an object of the loan class. Allow the user to input data about more than one loan. Display in the Loan App class the payment amount, amortization schedule, and the total amount of interest to be paid.

Two separate classes are to be developed. Creating a separate class for the Loan object enables the class to be used by other applications. The class includes an algorithm for producing an amortization schedule. It also includes a stand-alone method to determine the total interest paid over the life of the loan. Below tables list the data field members needed for the LoanApplication problem.

The class diagram for the Loan Application example is shown.
The name of the class, data members, and methods are listed in boxes stacked on top of each other.
Name of the class: Loan. Data member, loan Amount; Data type, double. Data member, rate; Data type, double. Data member, month Interest; Data type, double. Data member, payment Amount; Data type, double. Data member, total Interest Paid; Data type, double. Data member, balance; Data type, double. Data member, principal; Data type, double. Data member, num Payments; Data type, i n t. Method, Determine Payment Amount of. Method, Return Amortization Schedule of; Return type, string. Method, Determine Total Interest Paid of. Method, Calculate Month Charges of.
Name of the class: Loan App. Data member, Loan; Data type, Loan. Data member, loan Amount; Data type, double. Data member, years; Data type, i n t. Data member, interest Rate; Data type, double. Data member, another Loan; Data type, c h a r. Method, Get Input Values of.

Below table shows instance field members for the Loan class

|  |  |  |
| --- | --- | --- |
| **Data item description** | **Type** | **Identifier** |
| Amount of loan | double | loanAmount |
| Interest rate | double | race |
| Total interest paid | double | totalInterestPaid |
| Monthly payment amount | double | payment Amount |
| Current balance of the loan | double | balance |
| Current amount paid toward principal | double | principal |
| Number of payments | int | numPayments |
| Interest for the current month | double | monthlnterest |

Below table shows local variables for the LoanApp class

|  |  |  |
| --- | --- | --- |
| **Data item description** | **Type** | **Identifier** |
| Amount of loan | double | loanAmount |
| Interest rate | double | interestRate |
| Number of years to finance loan | int | years |
| More calculations (loop state-controlled variable) | char | anotherLoan |

Image lists the forumulas used for the Loan Application example.
Formulas are needed to calculate the following.
1. num Payments = years * 12.
2. term = (1 + rate /12.0) ^ num Payments.
3. payment Amount = loan Amount * rate / 12 * term / (term - 1.0).
4. month Interest = rate / 12 * balance.
5. principal = payment - month Interest.
6. balance = balance - principal.
7. total Interest Paid = total Interest Paid + month Interest.

The desired output is to display the monthly payment amount, an amortization schedule, and the total interest paid over the life of the loan. Below figure shows a prototype for the final output. The xxx’s are placed in the prototype to represent the location in which the calculated values should appear.

Image shows the prototype for the Loan Application example.
The data members are Loan Amount, Interest Rate, Number of Years for Loan, and Monthly payment listed with dummy values. A table is dispalyed with the headings, Month No., Int. Paid, Princ. Paid, and New Balance. The table is popluated with dummy values. Total Interest Paid is listed as another data member. A dummy value has been assigned to it.

Below the Structured English, or pseudocode, is used to design the step-by-step processes for the behaviors of the methods for the LoanApplication example.

Below figure shows behavior of LoanApp class methods

Behavior of the Loan App class methods is shown in pseudocode.
The main method is listed in pseduocode below.
Line 1. Main, left parenthesis, right parenthesis.
Line 2, indented once. begin loop.
Line 3, indented once. left brace.
Line 4, indented twice. Get Input Values, left parenthesis, loan Amount, comma, interest Rate, comma, years, right parenthesis.
Line 5, indented twice. create instance of Loan with loan Amount, comma, interest Rate, comma, years.
Line 6, indented twice. Determine Payment Amount, left parenthesis, right parenthesis.
Line 7, indented twice. Write Payment Amount.
Line 8, indented twice. Write Amortization Schedule, left parenthesis, right parenthesis.
Line 9, indented twice. Determine Total Interest Paid, left parenthesis, right parenthesis.
Line 10, indented twice. Write Total Interest Paid.
Line 11, indented twice. Prompt, open quotes, Do another calculation, question mark, left parenthesis, Y or N, right parenthesis, close quotes.
Line 12, indented twice. Read response.
Line 13, indented once. right brace.
Line 14, indented once. while, left parenthesis, response = = open quotes, Y, close quotes, right parenthesis, semicolon.
The Get Input Values method is listed in pseduocode below.
Line 1. Get Input Values, left parenthesis, double, loan Amount, comma, double, interest Rate, comma, i n t years, right parenthesis.
Line 2, indented once. Prompt, open quotes, Enter loan amount, colon, close quotes.
Line 3, indented once. Read string Loan Amount.
Line 4, indented once. Loan Amount = Convert, left parenthesis, string Loan Amount, right parenthesis.
Line 5, indented once. Prompt, open quotes, Enter interest rate, left parenthesis, as a decimal value, right parenthesis, colon, close qoutes.
Line 6, indented once. Read string Interest Rate.
Line 7, indented once. interest Rate = Convert, left parenthesis, stirng Interest Rate, right parenthesis.
Line 8, indented once. Prompt, open quotes, Enter number of years to finance, colon, close quotes.
Line 9, indented once. Read string Years.
Line 10, indented once. years = Convert, left parenthesis, string, Years, right parenthesis.


Below figure shows pseudocode for behavior of Loan class methods

The behavior of Loan class methods is listed in pseudocode. The pseudocode for the method Determine Total Interest Paid is listed below.
Line 1. Determine Total Interest Paid, left parenthesis, right parenthesis.
Line 2, indented once. set total Interest Paid = 0.
Line 3, indented once. set balance = loan Amount.
Line 4, indented once. set month = 1.
Line 5, loop while, left parenthesis, month < = n u m Payments, right parenthesis.
Line 6, indented once. left brace.
Line 7, indented twice. Calcuate Month Charges, left parenthesis, month, comma, n u m Payments, right parenthesis.
Line 8, indented twice. total Interest Paid = total Interest Paid +.
Line 9, indented 3 times. month Interest.
Line 10, indented twice. month = month + 1.
Line 11, indented once. right brace.
The pseudocode for the method Return Amortization Schedule is listed below.
Line 1. string, colon, Return Amortization Schedule, left parenthesis, right parenthesis.
Line 2, indented once. set a Schedule == hyphen, hyphen, hyphen, heading for columns, hyphen, hyphen, hyphen.
Line 3, indented once. balance = loan Amount.
Line 4, indented once. month = 1.
Line 5, loop while, left parenthesis, month < = n u m Payments, right parenthesis.
Line 6, indented once. left brace.
Line 7, indented twice. Calculate Month Charges, left parenthesis, month, comma, n u m Payments, right parenthesis.
Line 8, set a Schedule, + = month + month Interest, +.
Line 9, indented 3 times. principal + balance + open quotes, back slash, n, close quotes.
Line 10, indented twice. month = month + 1.
Line 11, indented once. right brace.
Line 12, indented once. return a Schedule.
The pseudocode for the method Calculate Month Charges is listed below.
Line 1. Calculate Month Charges, left parenthesis, i n t, month, comma, i n t, n u m Payments, right parenthesis.
Line 2, indented once. set Payment = payment Amount.
Line 3, indented once. set month Interest = rate, division slash, 12 asterisk, balance.
Line 4, indented once. if left parenthesis, month = = n u m Payments, right parenthesis.
Line 5, indented once, left brace.
Line 6, indented twice. set principal = balance.
Line 7, indented twice. set payment = balance + month Interest.
Line 8, indented once. right brace.
Liine 9, indented once. else.
Line 10, indented twice. principal = payment minus month Interest.
Line 11, indented once. balance = balance minus principal.
The pseudocode for the method Determine Payment Amount is listed below.
Line 1. Determine Payment Amount, left parenthesis, right parenthesis.
Line 2, indented once. set term = Math dot Pow, left parenthesis, 1 + rate, division slash, 12, right parenthesis, comma, n u m Payments, right parenthesis.
Line 3, indented once. set payment Amount = left parenthesis, loan Amount, asterisk, rate, division slash, 12, asterisk , term, right parenthesis, division slash.
Line 4, indented 3 times. left parenthesis, term minus 1.0, right parenthesis.

After the algorithm is developed, the design should be checked for correctness. Below table contains values that can be used to verify the correctness of the programming example.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Loan amount** | **Interest rate** | **Years** | **Payment amount** | **Total interest** |
| 50000 | .08 | 10 | 606.64 | 22,796.56 |
| 22000 | .05 | 5 | 415.17 | 2,910.03 |
| 150000 | .055 | 30 | 851.68 | 156,606.06 |
| 12000 | .06 | 3 | 365.06 | 1,142.28 |
| 10000 | .05 | 2 | 438.71 | 529.13 |

After you implement your design, you can compare these results with those obtained from your program output. This is sometimes called a desk run.

After you complete the design and verify the algorithm’s correctness, it is time to translate the design into source code. For this application, you are creating two separate files—one for each class.

The final application listing for both files is as follows:

/\* Loan.cs

\* Creates fields for

\* the amount of loan, interest

\* rate and number of years.

\* Calculates payment amount

\* and produces an amortization

\* schedule.

\*/

using System;

namespace LoanApp

{

public class Loan

{

private double loanAmount;

private double rate;

private int numPayments;

private double balance;

private double totalInterestPaid;

private double paymentAmount;

private double principal;

private double monthInterest;

// Default constructor

public Loan()

{

}

// Constructor

public Loan(double loan, double interestRate,

int years)

{

loanAmount = loan;

if (interestRate < 1)

rate = interestRate;

else //In case directions aren't followed

rate = interestRate / 100; // convert interest rate to decimal

numPayments = 12 \* years;

totalInterestPaid = 0;

DeterminePaymentAmount();

}

// Property accessing payment amount

public double PaymentAmount

{

get

{

return paymentAmount;

}

}

// Property setting and returning loan amount

public double LoanAmount

{

set

{

loanAmount = value;

}

get

{

return loanAmount;

}

}

// Property setting and returning rate

public double Rate

{

set

{

rate = value;

}

get

{

return rate;

}

}

// Property to set the numPayments,

// given years to finance.

// Returns the number of years using

// number of payments.

public int Years

{

set

{

numPayments = value \* 12;

}

get

{

return numPayments / 12;

}

}

// Property for accessing total interest

public double TotalInterestPaid

{

get

{

return totalInterestPaid;

}

}

// Determine payment amount based on

// number of years, loan amount, and rate

public void DeterminePaymentAmount()

{

double term;

term = Math.Pow((1 + rate / 12.0),

numPayments);

paymentAmount = (loanAmount \* rate /

12.0 \* term) / (term - 1.0);

}

// Returns string containing amortization table

public string ReturnAmortizationSchedule()

{

string aSchedule = "Month\t\tInt.\t\tPrin.\t\tNew";

aSchedule += "\nNo.\t\tPd.\t\tPd.\t\tBalance\n";

aSchedule += "-------\t\t-------\t\t--------\t----------\n";

balance = loanAmount;

for (int month = 1; month <= numPayments; month++)

{

CalculateMonthCharges(month, numPayments);

aSchedule += month

+ "\t\t"

+ monthInterest.ToString("N2")

+ "\t\t"

+ principal.ToString("N2")

+ "\t\t"

+ balance.ToString("C")

+ "\n";

}

return aSchedule;

}

// Calculates monthly interest,

// applied principal and new balance

public void CalculateMonthCharges(int month,

int numPayments)

{

double payment = paymentAmount;

monthInterest = rate / 12 \* balance;

if (month == numPayments)

{

principal = balance;

payment = balance + monthInterest;

}

else

{

principal = payment - monthInterest;

}

balance -= principal;

}

// Calculates the amount of interest paid

// over life of loan

public void DetermineTotalInterestPaid()

{

totalInterestPaid = 0;

balance = loanAmount;

for (int month = 1; month <= numPayments; month++)

{

CalculateMonthCharges(month, numPayments);

totalInterestPaid += monthInterest;

}

}

//Return information about the loan

public override string ToString()

{

return "\nLoan Amount: " + loanAmount.ToString("C") +

"\nInterest Rate: " + rate +

"\nNumber of Years for Loan: " + (numPayments / 12) +

"\nMonthly payment: " + paymentAmount.ToString("C");

}

}

}

/\* LoanApp.cs

\* Used for testing Loan class.

\* Prompts user for loan amount,

\* interest rate, and time period

\* for loan. Calls method to display

\* payment amount and amortization

\* schedule. Allows more than

\* one loan calculation.

\* \*/

using System;

using static System.Console;

namespace LoanApp

{

class LoanApp

{

static void Main( )

{

int years;

double loanAmount;

double interestRate;

string inValue;

char anotherLoan = 'N';

do

{

GetInputValues(out loanAmount, out interestRate,

out years);

Loan ln = new Loan(loanAmount, interestRate, years);

WriteLine();

Clear();

WriteLine(ln);

WriteLine( );

WriteLine(ln.ReturnAmortizationSchedule());

ln.DetermineTotalInterestPaid();

WriteLine("Payment Amount: {0:C}", ln.PaymentAmount);

WriteLine("Interest Paid over Life of Loan: {0:C}",

ln.TotalInterestPaid);

Write("Do another Calculation? (Y or N)");

inValue = ReadLine();

anotherLoan = Convert.ToChar(inValue);

}

while ((anotherLoan == 'Y') || (anotherLoan == 'y'));

}

// Prompts user for loan data

static void GetInputValues(out double loanAmount,

out double interestRate,

out int years)

{

Clear( );

loanAmount = GetLoanAmount();

interestRate = GetInterestRate();

years = GetYears();

}

static double GetLoanAmount()

{

string sValue;

double loanAmount;

Write("Please enter the loan amount: ");

sValue = ReadLine();

while ((double.TryParse(sValue, out loanAmount) == false)

|| loanAmount < 1 || loanAmount > 500000)

{

WriteLine("Invalid data entered " +

"for loan amount");

Write("\nPlease re-enter the loan amount (less than 500,000): ");

sValue = ReadLine();

}

return loanAmount;

}

static double GetInterestRate()

{

string sValue;

double interestRate;

Write("Please enter interest rate (as a decimal value " +

"- i.e. .06): ");

sValue = ReadLine();

while ((double.TryParse(sValue, out interestRate) == false)

|| interestRate < 0 || interestRate > 1)

{

Write("\nInvalid data entered " +

"for interest rate (decimal value - i.e. 0.06):");

Write("\nPlease re-enter the interest rate: ");

sValue = ReadLine();

}

return interestRate;

}

static int GetYears()

{

string sValue;

int years;

Write("Please enter the number of years for the loan: ");

sValue = ReadLine();

while ((int.TryParse(sValue, out years) == false)

|| years < 1 || years > 30)

{

Write("\nInvalid data entered " +

"for years");

Write("\nPlease re-enter the years (1-30): ");

sValue = ReadLine();

}

return years;

}

}

}