

## 2

## The Amortization Class

The loan officer at one of the Central Mountain Credit Union's branch offices has asked you to write a loan amortization application to run on her desktop PC. The application should allow the user to enter the amount of a loan, the number of years of the loan, and the annual interest rate. An amortization report should then be saved to a text file.

### Calculations

The credit union uses the following formula to calculate the monthly payment of a loan:

$$Payment = \frac{Loan \times \frac{Rate}{12} \times Term}{Term - 1}$$

where:  $Loan$  = the amount of the loan,  
 $Rate$  = the annual interest rate, and  
 $Term = (1 + Rate/12)^{Years \times 12}$

### Report Requirements

The report produced by the program should show the monthly payment amount and the following information for each month in the loan period: amount applied to interest, amount applied to principal, and the balance. The following report may be used as a model. It shows all the required information on a one-year \$5,000 loan at 5.9 percent annual interest.

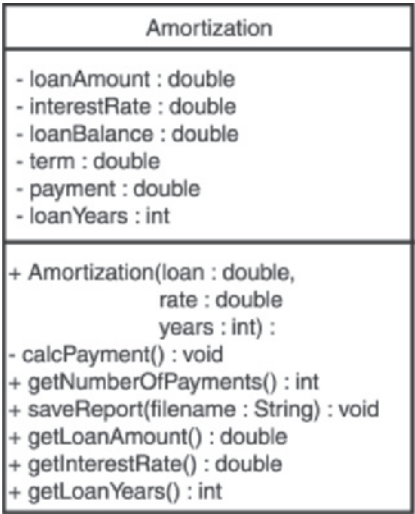
**CS2-2** Case Study 2 The Amortization Class

Monthly Payment: \$430.10			
Month	Interest	Principal	Balance
-----			
1	24.58	405.52	4,594.48
2	22.59	407.51	4,186.97
3	20.59	409.52	3,777.45
4	18.57	411.53	3,365.92
5	16.55	413.55	2,952.37
6	14.52	415.59	2,536.78
7	12.47	417.63	2,119.15
8	10.42	419.68	1,699.47
9	8.36	421.75	1,277.72
10	6.28	423.82	853.90
11	4.20	425.90	428.00
12	2.10	428.00	0.00

The core of the program will be a class, `Amortization`, that holds the primary data, performs the mathematical calculations, and displays the report. Figure CS2-1 shows a UML diagram for the class.

Table CS2-1 lists and describes the class’s fields.

**Figure CS2-1** UML diagram for the `Amortization` class



**Table CS2-1** Amortization class fields

Field	Description
loanAmount	A double variable to hold the amount of the loan.
interestRate	A double variable to hold the annual interest rate.
loanBalance	A double variable to hold the loan balance.
term	A double variable used in the calculation of the monthly payment.
payment	A double variable to hold the amount of the monthly payment.
loanYears	An int variable to hold the number of years of the loan.

Table CS2-2 lists and describes the class's methods.

**Table CS2-2** Amortization class methods

Method	Description
Constructor	The constructor accepts three arguments: the loan amount, the annual interest rate, and the number of years of the loan. These values are stored in their corresponding fields. The private method <code>calcPayment</code> is then called.
<code>calcPayment</code>	A private method that is used to calculate the monthly payment amount. The result is stored in the <code>payment</code> field.
<code>getNumberOfPayments</code>	Returns as an <code>int</code> the number of loan payments.
<code>saveReport</code>	Saves the amortization report to a text file.
<code>getLoanAmount</code>	Returns as a <code>double</code> the amount of the loan.
<code>getInterestRate</code>	Returns as a <code>double</code> the annual interest rate.
<code>getLoanYears</code>	Returns as an <code>int</code> the number of years of the loan.

Code Listing CS2-1 shows the code for the class.

**Code Listing CS2-1** (**Amortization.java**)

```

1 import java.io.*;    // For file-related classes
2
3 /**
4  This class stores loan information and creates a
5  text file containing an amortization report.
6 */

```

## CS2-4 Case Study 2 The Amortization Class

```
7
8 public class Amortization
9 {
10     private double loanAmount;    // Loan Amount
11     private double interestRate;  // Annual Interest Rate
12     private double loanBalance;   // Monthly Balance
13     private double term;          // Payment Term
14     private double payment;       // Monthly Payment
15     private int    loanYears;     // Years of Loan
16
17     /**
18      * The constructor accepts the loan amount, the annual
19      * interest rate, and the number of years of the loan
20      * as arguments. The private method CalcPayment is then
21      * called.
22      * @param loan The loan amount.
23      * @param rate The annual interest rate.
24      * @param years The number of years of the loan.
25      */
26
27     public Amortization(double loan, double rate, int years)
28     {
29         loanAmount = loan;
30         loanBalance = loan;
31         interestRate = rate;
32         loanYears = years;
33         calcPayment();
34     }
35
36     /**
37      * The calcPayment method calculates the monthly payment
38      * amount. The result is stored in the payment field.
39      */
40
41     private void calcPayment()
42     {
43         // Calculate value of Term
44         term =
45             Math.pow((1+interestRate/12.0), 12.0 * loanYears);
46
47         // Calculate monthly payment
48         payment =
49             (loanAmount * interestRate/12.0 * term) / (term - 1);
50     }
51
52     /**
53      * The getNumberOfPayments method returns the total number of
54      * payments to be made for the loan.
```

```

55     @return The number of loan payments.
56 */
57
58 public int getNumberOfPayments()
59 {
60     return 12 * loanYears;
61 }
62
63 /**
64     The saveReport method saves the amortization report to
65     the file named by the argument.
66     @param filename The name of the file to create.
67 */
68
69 public void saveReport(String filename) throws IOException
70 {
71     double monthlyInterest; // The monthly interest rate
72     double principal;       // The amount of principal
73     //DecimalFormat dollar = new DecimalFormat("#,##0.00");
74     FileWriter fwriter = new FileWriter(filename);
75     PrintWriter outputFile = new PrintWriter(fwriter);
76
77     // Print monthly payment amount.
78     outputFile.println(String.format(
79         "Monthly Payment: $%.2f", payment));
80
81     // Print the report header.
82     outputFile.println("Month\tInterest\tPrincipal\tBalance");
83     outputFile.println("-----" +
84         "-----");
85
86     // Display the amortization table.
87     for (int month = 1; month <= getNumberOfPayments(); month++)
88     {
89         // Calculate monthly interest.
90         monthlyInterest = interestRate / 12.0 * loanBalance;
91
92         if (month != getNumberOfPayments())
93         {
94             // Calculate payment applied to principal
95             principal = payment - monthlyInterest;
96         }
97         else // This is the last month.
98         {
99             principal = loanBalance;
100             payment = loanBalance + monthlyInterest;
101         }
102

```

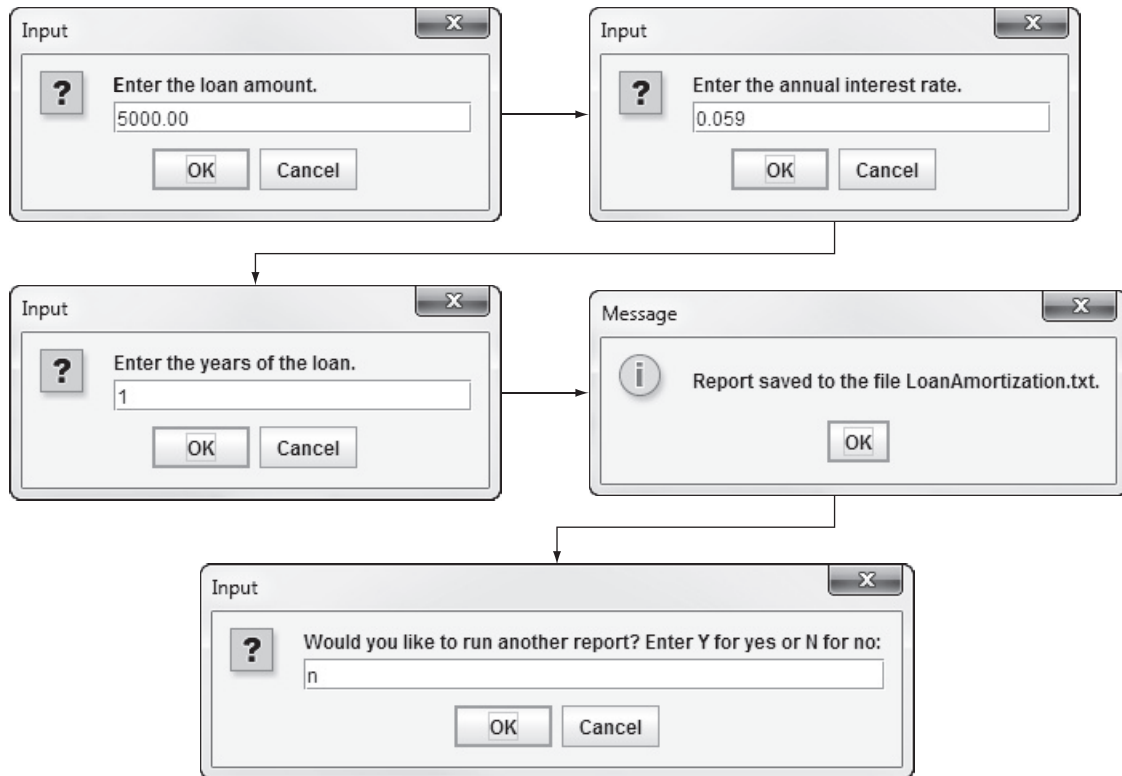
## CS2-6 Case Study 2 The Amortization Class

```
103         // Calculate the new loan balance.
104         loanBalance -= principal;
105
106         // Display a line of data.
107         outputFile.println(String.format("%d\t%.2f\t%.2f\t%.2f",
108                                         month, monthlyInterest, principal,
109                                         loanBalance));
110     }
111
112     // Close the file.
113     outputFile.close();
114 }
115
116 /**
117     The getLoanAmount method returns the loan amount.
118     @return The value in the loanAmount field.
119 */
120
121 public double getLoanAmount()
122 {
123     return loanAmount;
124 }
125
126 /**
127     The getInterestRate method returns the interest rate.
128     @return The value in the interestRate field.
129 */
130
131 public double getInterestRate()
132 {
133     return interestRate;
134 }
135
136 /**
137     The getLoanYears method returns the years of the loan.
138     @return The value in the loanYears field.
139 */
140
141 public int getLoanYears()
142 {
143     return loanYears;
144 }
145 }
```

## The Main Program

The main program code is shown in Code Listing CS2-2. First, it gets the amount of the loan, the annual interest rate, and the years of the loan as input from the user. It then creates an instance of the `Amortization` class and passes this data to the class's constructor. The program then saves the amortization report in the file `LoanAmortization.txt`. It asks the user whether he or she wants to run another report. If so, the program repeats these steps. Figure CS2-2 shows an example of interaction with the program.

**Figure CS2-2** Interaction with the `LoanReport` program



### Code Listing CS2-2 (LoanReport.java)

```

1  import javax.swing.JOptionPane; // For the JOptionPane class
2
3  // The following import statement is required because the main
4  // method has a throws IOException clause. Although this program
5  // doesn't have code that directly performs file I/O, the import
6  // statement is still required because of IOException.
7  import java.io.*;
8
9  /**
10   This program displays a loan amortization report.
11   */
12
13  public class LoanReport
14  {

```

[illegible]



```

63         years = Integer.parseInt(input);
64     }
65
66     // Create and initialize an Amortization object.
67     Amortization am =
68         new Amortization(loan, interestRate, years);
69
70     // Save the report.
71     am.saveReport("LoanAmortization.txt");
72     JOptionPane.showMessageDialog(null, "Report saved to " +
73         "the file LoanAmortization.txt.");
74
75     // Do another report?
76     input = JOptionPane.showInputDialog("Would you like " +
77         "to run another report? Enter Y for " +
78         "yes or N for no: ");
79     again = input.charAt(0);
80
81     } while (again == 'Y' || again == 'y');
82
83     System.exit(0);
84 }
85 }

```

Contents of the file *LoanAmortization.txt*:

Monthly Payment: \$430.10			
Month	Interest	Principal	Balance
1	24.58	405.52	4,594.48
2	22.59	407.51	4,186.97
3	20.59	409.52	3,777.45
4	18.57	411.53	3,365.92
5	16.55	413.55	2,952.37
6	14.52	415.59	2,536.78
7	12.47	417.63	2,119.15
8	10.42	419.68	1,699.47
9	8.36	421.75	1,277.72
10	6.28	423.82	853.90
11	4.20	425.90	428.00
12	2.10	428.00	0.00

First, notice that a do-while loop (in lines 23 through 81) controls everything done in this program. Here is a condensed version of the loop:

```

23     do
24     {

```

(Code to get the loan data and display the report.)

## CS2-10 Case Study 2 The Amortization Class

```
75         // Do another report?
76         input = JOptionPane.showInputDialog("Would you like " +
77             "to run another report? Enter Y for " +
78             "yes or N for no: ");
79         again = input.charAt(0);
80
81     } while (again == 'Y' || again == 'y');
```

During each iteration, this loop first gathers the necessary loan data as input and then writes the amortization report. At the end of the iteration, the user is asked whether he or she wants to run another report. If the user enters Y or y for yes, the loop repeats. Otherwise, the loop terminates and the program ends.

Inside the loop, the code that gathers input from the user also validates the input. For example, here is the code in lines 25 through 37 that gets and validates the loan amount:

```
25         // Get the loan amount.
26         input = JOptionPane.showInputDialog("Enter the " +
27             "loan amount.");
28         loan = Double.parseDouble(input);
29
30         // Validate the loan amount.
31         // (No negative amounts.)
32         while (loan < 0)
33         {
34             input = JOptionPane.showInputDialog
35                 ("Invalid amount. Enter the loan amount.");
36             loan = Double.parseDouble(input);
37         }
```

If the user enters a negative number, the while loop displays an error message and asks the user to enter the loan amount again. The code that gets the annual interest rate and the years of the loan, in lines 39 through 64, also performs similar input validation. Once these values have been correctly entered, an instance of the `Amortization` class is created in lines 67 and 68:

```
Amortization am =
    new Amortization(loan, interestRate, years);
```

Then, the `saveReport` method is called in line 71 to save the amortization report to the file `LoanAmortization.txt`:

```
am.printReport("LoanAmortization.txt");
```



**NOTE:** Notice that Code Listing CS2-1 has the following `import` statement in line 1:

```
import java.io.*;
```

Although this program does not have any code that directly performs file I/O, we still have to have this `import` statement because of the `throws IOException` clause in the main method header. The main method has to have the `throws IOException` clause because it calls the `saveReport` method in the `Amortization` class.