Regis University CC&IS
CS210 Introduction to Programming
Java Programming Assignment 6: Objects and Loops

This assignment will implement three types of loops in Java. You will again use **objects** to store data. Then looping statements will be used to run some of the statements repeatedly.

> WARNING: From this point in the course forward, programs that **do not compile without errors** will **NOT be accepted**. So be sure your code compiles and runs before submitting it.

Problem Summary and Equations

An automobile dealership would like you to write a program to help their customers decide between different car loan options. The loan options are:

- A loan at 4.8 % interest, if the customer puts **nothing** down on the purchase price of the car.
- A loan at 4.2% interest, if the customer puts **5%** down on the purchase price of the car.
- A loan at 3.7% interest, if the customer puts a **10%** down on the purchase price of the car.
- A loan at 3.3% interest, if the customer puts more than 10% down on purchase price of the car.

Interest will be applied monthly. This means that $1/12^{th}$ of the annual interest will be added to the current balance each month.

The program will provide a loan payment schedule to compare **two** of the loan options.

The customer can choose to the amount that they will pay each month for each loan option, but it must be at least $1/72^{nd}$ of the purchase price.

Program Requirements

The program must implement at least **one of each** loop type (**for**, **while** and **do-while**) somewhere in the code.

This program will implement a general-use input reading method called **readMoney** that can be used to read and validate a dollar and cents value within a specified range. The method will include three parameters:

- a String description of what kind of money amount is being read from the user
- the lowest money amount allowed
- the highest money amount allowed

By passing in these values as parameters, you can re-use the method to read any range of money needed for any data value.

For example, if you were to have called the method to read a total savings amount in programming assignment 5, and it was supposed to be between $100 and $1000000, the call would have been:

```
double totalSavings = readMoney(100, 1000000, "total savings");
```

Or to read a discount between $5 and $100, the call would be:

```
double discountAmount = readMoney(5, 100, "discount amount");
```

The string description would be used in the prompt, and the low/high values would be used in the prompt and error messages.

Required Classes and Methods

Two separate classes will be required for this program.

1. Create a new Java NetBeans project, named as follows:

   `LastnameJavaAssn6`
   For example: **`SmithJavaAssn6`**

   With a **main** class, named:

   `LoanComparison`

2. Define a Java **class** with properties and methods for a **Car Loan**.
   The class will have the following **private** properties (data fields):

   - car purchase price (USD $)
   - down payment rate (percentage – e.g. 5%)
   - current loan balance (USD $)
   - annual interest rate (percentage – e.g. 4.5%)
   - monthly payment (USD $)

   Within this class you will define methods, as follows:

   - Define a **constructor**, with parameters that pass in all the initial car loan values, except the current balance.
     - Use the parameter values to initialize the data fields.
     - Calculate and initialize the current balance to be the purchase price minus the down payment.
   - Define an **apply monthly payment** instance method to apply and display a payment for one month (note that there will be ***no loops*** in this method). This method will:
     - Calculate values and display ***one line of output***, containing the data about one month's payment, as follows:
       - Display the current (starting) balance.
       - Use the current balance to calculate the interest to be applied for the month, and add that interest to the current balance.
       - Display the interest charge and current balance (with interest).
       - If the current balance (with interest) is less than the monthly payment,
         - use the current balance (with interest) as the monthly payment.
       - Display the payment that will be paid for the month.
       - Subtract the payment from the current balance (with interest).
       - Display the new current balance (i.e. the ending balance, after the payment has been made).
   - Define a **generate loan payment schedule** instance method to calculate and display information each month until the car loan is paid off. This method will:
     - Have one parameter: the loan option number
     - Display a message saying the payoff schedule for the loan option will be generated.
     - Display the loan option summary (see ***Sample Output*** below).
     - Display headers for each column of the results (see ***Sample Output*** below).

- Use a loop to update and display information about the car loan ***every month***, as follows:
  - Display the month number (starting with month 1, and incrementing by 1 each time the code loops).
  - Call the above defined **apply monthly payment** method to update and display the data values for the month (the figures in each column should line up with each other on the right – see ***Sample Output*** on next page)
- Stop looping when the current balance reaches 0.

NOTE: When one instance method needs to call another instance method, the second instance method can be called using the `this` reference to reference the current object.

3. Within the **LoopComparison** class (containing the **main** method)

- Define the general-use ***static*** **readMoney** method described at the top of this assignment, to prompt for, read, and validate a dollar and cents money amount. This method will:
  - Include the parameters, from the description at the top of this assignment.
  - Prompt for and read a money amount.
    - The prompt should use the parameters to specify the kind of money amount being read (description) and the lowest and highest amounts allowed.
  - As long as the amount entered is not valid (in the range specified by the low and high parameters), issue an error message, and loop to re-prompt the user and read another value
  - Return a valid money amount.

- Define a ***static*** **readPercent** method to prompt for, read, and validate a percentage. This method will:
  - Have one parameter: a percentage specifying the lower bounds that will be accepted
  - Prompt for and read a percentage, specifying that it must be over the lower bounds
  - As long as the percent entered is not valid (i.e. must be over lower bounds and under 100%), issue an error message, and loop to re-prompt the user and read another value
  - Return a valid percentage.

- Define a ***static*** method to **create a loan option**. This method will:
  - Have two parameters: the loan option number and the purchase price
  - Display which loan option is being handled
  - Within a loop, until the user enter a valid choice, let the user choose from a menu of choices on what down payment to make:

    1 – no down payment
    2 – 5% down payment
    3 – 10% down payment
    4 – Some other percent over 10%

    If the user enters 4, call the **readPercent** method to read the down payment percentage to use.
  - Use the option chosen to determine the down payment percent and annual interest rate, as described on page 1.

- Call the **readMoney** method to read the monthly payment for the loan option
  - minimum is 1/72th of the purchase price rounded to the nearest whole dollar
  - maximum is $5,000
- Instantiate and initialize a new car loan object, using the constructor.
- Return the new car loan object.

- Define a **main** method that will:
  - Display a description of what the program will do to the user.
  - Prompt for and read how many customers want to compare car loans
  - In a loop, repeating for the number of customers entered, run the program as follows:
    - Display a message stating what customer number is being handled.
    - Call the **readMoney** method to read the purchase price of the car (minimum of $1000, maximum of $99,000).
    - Call the method to **create a loan option** for the first loan option choice.
    - Call the method to **create a loan option** for the second loan option choice.

    Sample Description and Input

```
This program implements a Car Loan comparison.
It will compute and display the payoff schedule for two loan options

How many customers want to compare loans? 1

FOR CUSTOMER 1:
Enter the purchase price of the car (between 1000 and 99000):  10000

For loan option 1:
What will the down payment will be?
    1 – no down payment
    2 – 5% down payment
    3 - 10% down payment
    4 – Over 10% down payment
Enter choice from menu above: 2
Enter the monthly loan payment (between 139 and 5000): 1000

For loan option 2:
What will the down payment will be?
    1 – no down payment
    2 – 5% down payment
    3 - 10% down payment
    4 – Over 10% down payment
Enter choice from menu above: 3
Enter the monthly loan payment (between 139 and 5000): 800
```

    - Display a couple of blank lines after reading all of the user input
    - Display the customer being handled.

- Call the **generate payoff schedule** method with the first option's CarLoan object.
- Call the **generate payoff schedule** method with the second option's CarLoan object.

Sample Output for one customer

```
LOAN COMPARISONS FOR CUSTOMER 1:

Loan Option 1 loan payment schedule:
$10000.00 car purchased with 5.0% down payment
Loan amount is $9500.00 at a 4.2% annual interest rate
-------------------------------------------------------------
          Initial    Month's     Balance                  End
Month     Balance    Interest    w/Interest    Payment    Balance
-------------------------------------------------------------
    1    9500.00       33.25       9533.25    1000.00    8533.25
    2    8533.25       29.87       8563.12    1000.00    7563.12
    3    7563.12       26.47       7589.59    1000.00    6589.59
    4    6589.59       23.06       6612.65    1000.00    5612.65
    5    5612.65       19.64       5632.30    1000.00    4632.30
    6    4632.30       16.21       4648.51    1000.00    3648.51
    7    3648.51       12.77       3661.28    1000.00    2661.28
    8    2661.28        9.31       2670.59    1000.00    1670.59
    9    1670.59        5.85       1676.44    1000.00     676.44
   10     676.44        2.37        678.81     678.81       0.00

Loan Option 2 loan payment schedule:
$10000.00 car purchased with 10.0% down payment
Loan amount is $9000.00 at a 3.7% annual interest rate
-------------------------------------------------------------
          Initial    Month's     Balance                  End
Month     Balance    Interest    w/Interest    Payment    Balance
-------------------------------------------------------------
    1    9000.00       27.75       9027.75     800.00    8227.75
    2    8227.75       25.37       8253.12     800.00    7453.12
    3    7453.12       22.98       7476.10     800.00    6676.10
    4    6676.10       20.58       6696.68     800.00    5896.68
    5    5896.68       18.18       5914.87     800.00    5114.87
    6    5114.87       15.77       5130.64     800.00    4330.64
    7    4330.64       13.35       4343.99     800.00    3543.99
    8    3543.99       10.93       3554.92     800.00    2754.92
    9    2754.92        8.49       2763.41     800.00    1963.41
   10    1963.41        6.05       1969.46     800.00    1169.46
   11    1169.46        3.61       1173.07     800.00     373.07
   12     373.07        1.15        374.22     374.22       0.00
```

WARNING: The objects, classes, and methods must be implemented exactly as specified above. If your program produces correct output, but you did not create and use the object as specified, and implement the required classes and methods, you will lose a significant number of points.

NOTE: For an example of a Java program implementing loops, see Online Content section 12.13.

4. The program must follow the **CS210 Coding Standards** from Content section 6.10.
   Be sure to ***include*** the following comments:
   - Comments at the ***top of each code file*** describing what the class does

        - Include **tags** with the author's name (i.e. your full name) and the version of the code (e.g. version 1.0, Java Assn 5)
    - Comments at the ***top of each method***, describing what the method does
        - Include **tags** with names and descriptions of ***each*** parameter and return value.

Testing

- Run, debug, and test your Java program with different inputs, until you are sure that all control structures within your program work correctly.

The sample inputs and output can be used as the initial test to test your program.
But be sure you thoroughly test it using other values as well.

Program Submission

This programming assignment is due by midnight of the date listed in the **Course Assignments by Week**.

Programs that **do not compile without errors** will **NOT be accepted**.

Again, you will submit a single **zip** file containing all of the files in your project.

- First export your project from NetBeans:
    - Name your export file in the following format:
      `<lastname>Assn<x>.zip`

      For example:
      `SmithAssn6.zip`

      NOTE: Save this zip file to some other directory, not your project directory.

- Then submit your **.zip** file to the **Java Prog Assn 6** assignment submission folder (located under the **Assignments/Dropbox** tab in the online course).

    - Warning: Only NetBeans export files will be accepted.
      Do not use any other kind of archive or zip utility.

Grading

Your program will be graded using the **rubric** that is linked on the same assignment page from which this program requirements file was downloaded.

WARNING:
*Programs submitted more than 5 days past the due date will **not** be accepted, and will receive a grade of 0.*