CIS 200: Project 7 (50 points)
Due Friday, Apr 8th by midnight

This project can be challenging, which is why you are given **2 Weeks** to complete. I strongly suggest you don't wait 1-2 days before the due date to begin and seek answers for your questions…

Reminder: Programs submitted after the due date/time will be penalized 10% for each day the project is late (not accepted after 3 days, i.e. Midnight, Mon, Apr 11th)

Reminder: As the projects become more challenging, it is a good idea to remind all students about the *Academic Integrity Policy* for this course. ALL projects are **intended to be done individually. Don't share your code with anyone.** *If it is not your work, don't submit it as yours!* Refer to the policy within the course syllabus which states, ***"If two (or more) students are involved in ANY violation of this policy, ALL students involved receive a zero*** for the assignment and the offense is officially reported to the *KSU Honor Council*. The second offense results in a failing grade for the course and possible suspension from the university (this decision is made by the *K-State Honor Council*)."

Assignment Description:

To simulate an actual *real world* programming project, you are given the following *general description* of what is needed by *Big12 Bank Midwest*:

> *A bank needs a program that can be used to figure monthly payment and the total amount to be paid on various mortgage (house) loans. Program should be flexible enough to either allow a bank officer to **enter in all the input** or use a **special advertised promotion** where the customer can get a $250,000 house loan for 20 years at the annual rate of 3.2%. The program will provide a menu that allows the bank officer to choose between the two. Program will continue until the user decides to end the input.*

Develop an *object-oriented solution* to the program requested.
(Note: This is similar to an assigned lab but the assigned lab did ***NOT*** use an *object-oriented* design)

Implementation Requirements:
Your project must contain **two** classes: ***Mortgage*** and ***MortgageApp*** (the application or driver class).

Specifics of the *Defining* Class:

Mortgage will be a general class that represents a *loan mortgage* (i.e. a house loan) and will be used to create *Mortgage* objects. ***At a minimum***, it will contain ***private*** instance variables for the three values needed to calculate a mortgage (*interest rate, term of the loan, amount*) plus the *account number* and the *customer's last name*. It will also contain a *minimum* of two constructors: a *no-argument* constructor and a *multi-argument* constructor (it will be left up to you how you defined and use these constructors and if you want more than two). Lastly, although the class may contain more, it must also define the following methods ***using the method signatures given***:

// Read-in from user, validate, and store the *amount* of the loan
public void storeLoanAmount()

// Read-in from user, validate, and store the *term* of the loan
public void storeTerm()

// Read-in from user, validate, and store the *yearly interest rate* of the loan
public void storeInterestRate()

// Read-in from user and store the *last name* of the customer
public void storeLastName()

// Calculate and return the monthly payment for the loan
private double calcMonthlyPayment()

// Calculate and return the total payment for the loan
private double calcTotalPayment()

// Display formatted output
public String toString()

- For the *account number*, randomly generate a number between 100-10,000 when the object is created and add the first four characters of the user name at the beginning to use as the account number. For example, ***Lang23161***. *You can assume the user has at least 4 characters in their last name.* You do not need to validate *unique* account numbers.
- The first three '*store'* methods are used to get user input, verify the input is valid (as given below) and store result in appropriate private data property:
  1) **Valid loan amounts are $75,000 to 1 million (inclusive)**
  2) **Valid loan terms are 10-40 years (inclusive)**
  3) **Valid interest rates are between 2%-7% (inclusive)**
  4) Within your validation routines, accept *non-numeric* input also as valid input. In other words, your program will NOT crash if *character* input included. For example, a user enters ***$200,000*** for the loan amount or **7.25%** for interest rate or **25 years** for loan term. You can either convert into a valid amount *or* display an error message and have the user re-enter the input. As long as your program doesn't generate a run-time error if *non-numeric* input is entered.
- ***private*** methods *calcMonthlyPayment* and *calcTotalPayment* are called *only* from within the *Mortgage* class (Hint: Call from *toString* method)
- ***toString*** method ***returns a string*** that can be used to display the info in the format shown below. **Do not print within the *toString* method**.

  Account Number: Lang23161
  The monthly payment is $817.08
  The total payment is $147,075.02
- Format all currency values with '$'-signs, commas, and two values after the decimal.
- Since all of your data properties are *private*, you may need to add *get/set* methods to *access/store* values in these data properties.
- The formula for calculating the ***monthly*** payment is given below…notice that *interest* and *term* or *length* of the loan is ***monthly***.

$$M = P \frac{[\, I(1+I)^N \,]}{[\, (1+I)^N - 1 \,]}$$

M = Monthly Payment
P = Mortgage Principal
I = Monthly Interest
N = Number of Months

Specifics of the *Application* Class:

Your driver program (***MortgageApp***) *should* contain only a main method – optional methods are acceptable as long as they are not a required part of the *Mortgage* class, but should be kept to a minimum. Display the menu shown below, validate user's menu choice, then display the result of that loan (as shown below).

Method *main* will include ***creating an array*** that can hold a *maximum* of 10 Mortgage objects (don't allow more), displaying the menu, validating the *menu choice*, calling the methods defined in the *Mortgage* class through the creation of a Mortgage *object*, and validating the input on unique loans (through methods of the Mortgage class). Method *main* should basically be an outline of method calls to methods defined in your *Mortgage* class. You must place objects *in* the array as they are created and call the methods defined in the *Mortgage* class on both types of loans (i.e. call calc*MonthlyPayment* to determine the monthly payment for a *promotional* or *unique* loan.)

Display the result of each loan object as it is entered (*as shown in the screen shot on the previous page*) using your *toString* method. **When user selects 'quit'**, use your *toString* method to display each object in your array. Remember that 10 is the *maximum* number, not the *required* number (i.e. *array may only contain 3 loan objects*)

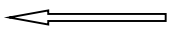
A ***possible*** execution of your program would look like the following (please display your output as shown)

```
Please choose from the following choices below:
        1) Promotional Loan (preset loan amount, rate, term)
        2) Unique Loan (enter in loan values)
        3) Quit (Exit the program)

        Please enter your selection (1-3): 11
                Invalid Choice. Please select 1, 2, or 3: 1

Enter customer's Last Name Only: Smith

PROMOTIONAL LOAN...:
Account number: Smit5419
The monthly payment is $1,411.66
The total payment is $338,797.70
```



These are **calculated** values, not hard-coded

```
Please choose from the following choices below:
        1) Promotional Loan (preset loan amount, rate, term)
        2) Unique Loan (enter in loan values)
        3) Quit (Exit the program)

        Please enter your selection (1-3): 22
                Invalid Choice. Please select 1, 2, or 3: 22
                Invalid Choice. Please select 1, 2, or 3: 2

Enter customer's Last Name Only: Lang

Enter the amount of the loan (Ex:75000): $20000
        Valid Loan Amounts are $75000-$1000000
        Please re-enter loan amount without $ or commas (Ex:75000): $200,000
```

Output example continued on next page…

Enter yearly interest rate (Ex: 8.25): .0725
Valid Interest Rates are 2% - 7%
Please re-enter valid yearly interest rate (Ex: 8.25): 7.25%
Valid Interest Rates are 2% - 7%
Please re-enter valid yearly interest rate (Ex: 8.25): 6.95%

Enter number of years for the loan: 255
Valid Loan Terms are 10-40
Please re-enter valid number of years: 25

UNIQUE LOAN…:
Account number: Lang6560
The monthly payment is $1,407.19
The total payment is $422,155.65

Please choose from the following choices below:
1) Promotional Loan (preset loan amount, rate, term)
2) Unique Loan (enter in loan values)
3) Quit (Exit the program)

Please enter your selection (1-3): 3

PROGRAM COMPLETE
Contents of Array…

Account number: Smit5419
The monthly payment is $1,411.66
The total payment is $338,797.70

Account number: Lang6560
The monthly payment is $1,407.19
The total payment is $422,155.65

Documentation:
At the top of **EACH class**, add the following comment block, filling in the needed information:

```
/**
* <Full Filename>
* <Student Name / Section Day/Time>
*
* <description of the class – i.e. What does this particular class do?>
*/
```

At the top of **EACH method**, add the following comment block, filling in the needed information:

```
/** Method name
  * (description of the method)
  *
  * @param (describe first parameter)
  * @param (describe second parameter)
  * (list all parameters, one per line)
  * @return (describe what is being returned)
  */
```

Requirements

This program will contain TWO classes - *Mortgage.java* **and** *MortgageApp.java*. Your program must compile (by command-line) with the statement: **javac Mortgage.java** and **javac MortgageApp.java**

It must then run with the command: **java MortgageApp**

Submission – *read these instructions carefully or you may lose points*

To submit your project, first create a folder called **Proj7** and move your completed *Mortgage.java* **and** *MortgageApp.java* files into that folder. Then, right-click on that folder and select "*Send To->Compressed (zipped) folder*". This will create file ***Proj7.zip***.

Log-in to Canvas and upload your `Proj7.zip` file. Only a .zip file will be accepted for this assignment in Canvas. **Put your full name and Project 7 in the comments box.**

Important: It is the ***student's responsibility*** to verify that the ***correct*** file is ***properly*** submitted. If you don't properly submit the ***correct*** file, it will not be accepted after the 3-day late period. *No exceptions*.

Grading:

Programs that do not compile or that don't implement two classes will receive a grade of 0. Programs that *do* compile will be graded according to the following rubric:

Requirement	Points
Must properly implement TWO classes to be considered for partial credit.	**-**
EXECUTION (20 pts.)	**-**
Properly displays the menu shown	**1**
All input is correctly validated (menu choice, rate, term, amount)	**5**
Non-numeric input is allowed (doesn't generate a run-time error)	**3**
Output for a *promotional* loan is correctly calculated and formatted (must use *calc* methods of *Mortgage* class)	**3**
Output for a *unique* loan is correctly calculated and formatted (must use *calc* methods of *Mortgage* class)	**5**
Output for all loans objects is properly displayed when user chooses quit	**3**
MortgageApp CODE (14 pts.)	**-**
Other than *main*, methods are kept at a minimum. Must invoke the methods defined in *Mortgage* class to do the *work*. *main* includes displaying menu, validating menu choice, declaring and using array of *Mortgage* objects.	**3**
Array of 10 objects is declared and used properly throughout (i.e. objects are properly placed in the array). Does not allow more than 10 objects to be entered.	**3**
Creates object by calling the proper constructor	**1**
Properly calls all methods through the object for both *promo* and *unique* loans	**3**
Display result by using the *toString* method, which is properly defined – value format exactly matches example ('$'-sign, Comma, two decimal places on all values)	**3**
Properly Loops until user chooses to quit (or maximum number of Objects are created)	**1**
Mortgage CODE (13 pts.) Contains following methods that MUST be called to get any credit	**-**
Contains required ***private*** data properties and a minimum of TWO constructors	**3**
4 "*store*" methods correctly defined using given method signatures	**4**
calcMonthlyPayment method is correctly defined using given method signature (private)	**2**
calcTotalPayment method is correctly defined using given method signature (private)	**2**
toString method is correctly defined using given method signature	**2**
Proper Documentation on both the File Header and ON EACH METHOD	**2**
Proper Submission with full name & project # in Comment box. Correct Filenames (Mortgage and MortgageApp.java)	**1**
Minus Late Penalty (10% per day)	
Total	**50**