CIS 200: Project 7 (60 points + 10% Extra Credit)
Due WED, Mar 26th by 11:59pm

Programs submitted after the due date/time will be penalized 10% for each day the project is late.
Projects are not accepted 2 days after the stated due date – *no exceptions*.

ALL work submitted in this course is to be done BY YOU. Submitting *another's work (human or computer) as your own* is a violation of the Honor Code. ***If you didn't write it, you must cite it*!** If you use ANY code YOU didn't develop *you must cite your source* in a comment above the line(s) of code that are not your own work, otherwise it is considered *plagiarism* (*representing others' work, whether copyrighted or not, as one's own*).

Assignment Description:
Since the *Object-Orienting Programming* paradigm is new for many students, be aware that this project can be especially challenging for those students. So, start soon after the project is made available and seek help and ask clarification questions early in the week (rather the day the assignment is due).

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

A bank needs a program that can be used to figure the monthly payment and 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 promotional loan*** *where the customer can get a $300,000 house loan for 30 years at the annual rate of 5.25%. 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.

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 define 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()

// Generate random number, create account number, and store the *account number* of the customer
public void storeAcctNum()

// 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 (Hint: call your private methods within THIS method)
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 username at the beginning to use as the account number. For example, ***Lang2316***. 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 $100,000 (*inclusive*) to 1 million (*exclusive*)**
  2) **Valid interest rates are between 3.5%-9% (*inclusive*)**
  3) **Valid loan terms are 15-50 years (*inclusive*)**

- **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. ***toString*** method must *NOT* contain any print statements and must be properly declared using the following method signature: ***public String toString()***

Account Number: Lang2316
The monthly payment is $817.08
The total payment is $147,075.02

FYI: *toString* method is NOT covered in zyBook. It will be discussed in Lecture on Tue Week 9. If getting an early start, you can temporarily create a *display* method to use for testing, which can easily be converted to a *toString*. There is also a lecture video in Canvas under Modules-Week 8 discussing *toString*.

- 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:

Important: Although *ArrayLists* would also work for Project 7, the use of an ***Array** of Mortgage* objects is REQUIRED for this Project. It is OK if *ArrayLists* are included in your program as long as an *Array of Mortgage Objects* is used *as described* in the program description and grading rubric.

Your driver program (***MortgageApp***) *should* contain only a main method. Optional methods are allowed if they are not a required part of the *Mortgage* class, but should be kept to a minimum.

- Method *main* must include **creating an array** holding a *maximum* of 10 Mortgage objects. Once 10 is reached, display "*Maximum number of objects reached*" and do not allow any more.
- Display 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.)
- Use constants for the promotional loan amount, rate, and years. Program should still work correctly if ANY or ALL of these values were changed.
- Display the result of each loan object as it is entered (*as shown below*) using your *toString* method.
- When user selects 'quit' OR the maximum of 10 are entered, 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*)

Here is an example run of the program *(user input in red):*

```
Welcome to the Big12 Americana Bank program   ← Note: Only display this ONCE, during initial run of program
created by <programmer's first and last name>

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: 11
                Invalid Choice. Please select 1, 2, or 3: 1

Enter customer's Last Name Only: Smith

PROMOTIONAL LOAN:
Account number: Smit5419      ← 4 digits are randomly generated, so will differ
The monthly payment is $1,656.61
The total payment is $596,380.00     ⇐   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: 22
                Invalid Choice. Please select 1, 2, or 3: 2
```

```
Enter customer's Last Name Only: Lang

Enter the amount of the loan (Ex:120000): 1000000        ← a million
          Valid Loan Amounts are $100000 - $1000000 (non-inclusive)
Please re-enter loan amount without $ or commas (Ex:120000): 20000    ← i.e., $20,000
          Valid Loan Amounts are $100000 - $1000000 (non-inclusive)
Please re-enter loan amount without $ or commas (Ex:120000): 200000   ← i.e., $200,000

Enter yearly interest rate (Ex: 8.25): .0725
          Valid Interest Rates are 3.5% - 9.0%
Please re-enter valid yearly interest rate (Ex: 8.25): 0.725
          Valid Interest Rates are 3.5% - 9.0%
Please re-enter valid yearly interest rate (Ex: 8.25): 7.25

Enter number of years for the loan: 55
          Valid Loan Terms are 15-50
Please re-enter valid number of years: 55
          Valid Loan Terms are 15-50
Please re-enter valid number of years: 50

UNIQUE LOAN:
Account number: Lang6560                ←4 digits are randomly generated, so will differ
The monthly payment is $1,241.79
The total payment is $745,073.27

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 $$1,656.61
The total payment is $596,380.00

Account number: Lang6560
The monthly payment is $1,241.79
The total payment is $745,073.27
```

Documentation: At the top of EACH CLASS, add the following comment block. Each class description will differ since the purpose of each class differs.

```
/**
* <Full Filename>
* <Student Name / Lab Section Day and Time>
*
*   Clearly indicate to the reader of your code what THIS class does (i.e., the purpose of THIS class)
     Must be detailed enough so outside reader of your code can determine the purpose of this class (at least
    3 or 4 sentences)
*/
```

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

```
/**
  * (description of the method)
  * @param (describe first parameter)              …if no parameters, don't include
  * @param (describe second parameter)
  * …(list all parameters, one per line)
  * @return (describe what is being returned)      …if nothing is return, don't include
  * @throws IOException for File I/O…if method doesn't throw an exception, don't include
*/
```

-You WON'T be generating a JavaDoc for Project 7 but still include JavaDoc Comments on ALL methods.

Extra Credit (up to +10%): To be considered for extra credit, you MUST add a comment in your top documentation "EXTRA CREDIT INCLUDED", otherwise the GTA will NOT test for extra credit.

Do either or both of the following modifications:

1) (+2 pts) Declare 6 static final constants (use all uppercase to distinguish) in your *Mortgage* class representing the lower and upper limits of the loan amount, interest rate, and term. Use each of these constants in the data validation within the 3 store methods on p.2.

   For example, if the limits of the term were change to 10-40, then that should be reflected in both your validation check and your error msg if invalid.

2) (+4 pts) Within your validation routines, accept *non-numeric* input also as valid input. In other words, your program will NOT crash if *character* input included. (Range validation should still be included.)

   For example, a user enters ***$200,000*** for the loan amount or **7.25%** for interest rate or **25 years** for loan term. Convert into a valid amount before continuing. Your program will NOT generate a *run-time* error if any of the given *non-numeric* input is entered.

Running Requirements:

This project will contain TWO files - **Mortgage** and **MortgageApp**.

ALL classes must compile (by command-line) with the statement: **javac filename.java**

It must then RUN by command-line with the exact command: **java MortgageApp**

*****Please make sure and test this before submitting, so points are not lost unnecessarily!***

Programs that do not compile/run from the command line with the exact command above will receive a grade of ZERO, regardless of the simplicity or complexity of the error, so make sure you submit the *correct* file that *properly compiles/runs from the command line.*

Submission/Grading – read/follow these instructions carefully or you may get a ZERO on your project

To submit your project, create a folder called *Proj7* and copy BOTH *.java* files *(Mortgage/MortgageApp)* into that folder. Then, right-click on that folder and select "*Send To* → *Compressed (zipped) folder*". This will create the 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. Again, make sure ALL classes compile AND program runs at the command line.

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 2-day late period*. No exceptions.

Grading: Only what is submitted to Canvas before the deadline will be considered for grading, so make sure you submit the CORRECT file. Files can be re-submitted until the deadline but only the LAST submission will be graded.

Programs that compile/run are graded according to the following rubric:

Requirement	**Points**
To be considered for grading, program must include BOTH classes with all '*work*" being done through objects of the *Mortgage* class *stored within an **array*** in the *MortgageApp* class	-
Documentation – includes correct Javadoc documentation on EACH file (class) and above EACH method (including Constructors) in the *Mortgage* class	3
Mortgage class (CODE) (14 pts.)	**-14-**
Contains all required ***private*** data properties and a minimum of TWO constructors	2
5 "*store*" methods correctly defined using given method signatures	5
calcMonthlyPayment method is correctly defined using given method signature (**private**)	2
calcTotalPayment method is correctly defined using given method signature (**private**)	2
toString*: Correctly declared and used to display output. Must be declared:* ***public String toString()*** ***No I/O within this method (i.e. no print statements)***	3
MortgageApp class (CODE) (11 pts.)	**-11-**
Properly declares 3 constants for promotional loan amount, rate, and term	1
ARRAY of 10 *Mortgage* 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 – displays msg.	2
Creates object by calling the proper constructor	2
Properly calls all methods defined in *Mortgage* class to do the *work* through the object for both *promo* and *unique* loans. *main* includes displaying menu, validating menu choice, declaring and using array of *Mortgage* objects.	4
Display objects by correctly calling *toString* method for each Mortgage Object	2
EXECUTION (32 pts.)	**-32-**
Properly displays initial two lines plus the given menu	2
All input is correctly validated (menu choice, rate, term, amount) (2 pts each)	8
Customer Account Number is properly generated and displayed	3
Output for a PROMOTIONAL loan is correctly calculated – value format exactly matches example ('$'-sign, Comma, two decimal places on all values)	4
Output for a UNIQUE loan is correctly calculated – value format exactly matches example ('$'-sign, Comma, two decimal places on all values)	6
Properly Loops until user chooses to quit or maximum number of Objects are created	3
Output for all loans objects in array are properly displayed when user chooses quit	3
Works if values for constant in promotional loan are changed or the array size	3
Extra Credit 1: Works properly if any of the 6 static final constants declared are changed	+2
Extra Credit 2: Non-numeric input is allowed (doesn't generate a run-time error) - $, % years	+4
Minus Late Penalty (10% per day)	
Total	**60 + 6**