CIS 2571: Introduction to Java
Lab Assignment

Name: _____

Lab Assignment	#6 – Objects and Classes (Date, String, and User Created)
Due Date *(beginning of class)*	03/10/2014
Points	Short Answer Questions _____/ 10 pts. **Account Class** Java Program UML diagram source code output uploaded .zip _____/ 60 pts. **Total** _____**/ 70 pts.**

Lab Assignment #6 Activities

1. Answer the questions in the spaces provided or attach a sheet with your answers (be sure to label questions for proper credit) *(2 points each, unless noted otherwise)*:

Question	Your Answer
a. What is the relationship between a **class** and an **object**?	
b. What is the difference between a class **state** and a class **behavior**? How are **class states** and **class behaviors** implemented in Java classes? *(3 points)*	
c. What restrictions are placed on **instance variable** and **static variable** access from **within the definition** of: *(3 points)* • an **instance** method and • a **static** method	
d. How is an **instance** method call different than a **static** method call?	

2. On your own, create a Java program to solve the **Chapter 8 Programming Exercise 8.7** *(The Account Class)* on page 331 of your textbook with the following **additional programming specifications**:

   - The **Account** class should be in a **separate file** from the test program (**TestAccount**). Below are a *few hints* to get you started: *(13 points)*
     - The **annualInterestRate** is shared by **all** account objects; it should be stored as a **static class variable** and initialized to the rate given in the textbook.
     - In place of the **getMonthlyInterestRate**, include a method **getMonthlyInterest** that returns the amount of monthly interest for the current account balance.
     - The **private Date** object can be formatted for use in an output string *(see Date/Time Conversions)*: http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html#syntax
     - You should have void methods for **deposit** and **withdraw** that will be passed an amount to update the balance accordingly.
   - Add the following data field (and the related public **accessor** and **mutator** methods) to the **Account** class: *(2 points)*
     - A private **String/StringBuilder** data field named **name** for the account holder's first and last name (default empty String)
   - Add the following logic to the **TestAccount** program: *(20 points)*
     - The user should be prompted to input their name and beginning account balance. There should be **separate prompts** for their first and last name which are then **concatenated** and stored in a newly created **Account** object.
     - The initial account information is then displayed (use the **static String format** method to create the formatted string for output in the following format). You will note that this string is re-displayed when a deposit or withdrawal is made on the account object.

```
ID:                          1122
Created:                  02/25/13
Owner:                Sam Williams
Opening Balance:        20,000.00
Annual Rate:                 4.50%
```

- The user should be prompted for an amount to **withdraw**. The appropriate method is called and the amount is **deducted** from the account balance and the updated account information displayed to the user *(test case in textbook used)*:

```
ID:                          1122
Created:                  02/25/13
Owner:                Sam Williams
Opening Balance:         20,000.00
Annual Rate:                 4.50%
Withdraw:                 2,500.00
Current Balance:         17,500.00
```

- The user should be prompted for an amount to **deposit**. The appropriate method is called and the amount is **added** from the account balance and the updated account information displayed to the user *(test case in textbook used)*:

```
ID:                          1122
Created:                  02/25/13
Owner:                Sam Williams
Opening Balance:         20,000.00
Annual Rate:                 4.50%
Deposit:                  3,000.00
Current Balance:         20,500.00
```

Use the **console for input and output** of account information.

a. Create the source code files, **Account.java** and **TestAccount.java**. Add a block comment at the top of each file to identify your name, file, date, class, assignment, and short description of the file. *(4 points total, 2 points each)*

b. Compile the source code until no errors are found. The Java bytecode files **Account.class** and **TestAccount.class** should be created.

- Common Errors: http://www.cs.armstrong.edu/liang/intro9e/debug.html

c. Run the Java bytecode and observe the results.

d. Attach a hardcopy printout of your source code.

e. Attach a hardcopy printout of your sample output:

- Use the sample account information from the textbook, use your name for the name. Attach copies of your output session. *(3 points)*

f. Attach a **software generated** hardcopy output of the **UML class diagram** for the **Account** class. Remember to include access specifiers for properties and methods (see Figure 8.17 on pate 320 in textbook). *(15 points)*

- See the following link for an online supplement from the author for UML graphical notations:

  http://www.cs.armstrong.edu/liang/intro9e/supplement/Supplement5dUMLNotation.pdf

g. Create a **.zip** file containing only your **.java** source code files. Upload a copy of this **.zip** file to the appropriate assignment in Blackboard (this will be demonstrated during class). *(3 points)*

- See the following link for a video on submitting assignment in Blackboard:

  http://ondemand.blackboard.com/r91/movies/bb91_student_submit_assignment.htm
- See the following link for a video on creating **.zip** files in Windows XP:
  http://www.youtube.com/watch?v=3xqF56OZo_k
- See the following link for instructions on creating **.zip** files:
  http://condor.depaul.edu/slytinen/instructions/zip.html