CIS 2571: Introduction to Java
Lab Assignment

Name: _____

Lab Assignment	#7 – Classes, Inheritance, and Polymorphism
Due Date *(beginning of class)*	03/24/2014
Points	Short Answer Questions _____/ 25 pts. **Account Class** Java Program UML class diagram source code output uploaded .zip _____/ 45 pts. **Total** _____**/ 70 pts.**

Lab Assignment #7 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 term used to describe a class whose contents cannot be changed?	
b. How can you access a class variable that is hidden by a local variable with the same name?	
c. What is the name of the special type of aggregation relationship where there is an ownership between two objects?	
d. What type of ordering relationship is used to represent the insertion and removal of stack elements?	

e. What are the method names to: • add an element to the stack • remove an element from the stack	
f. Identify the three '**pillars'** of object oriented programming. *(3 points)*	
g. What is the name of the relationship between **superclasses** and **subclasses**? What is the Java keyword used to represent this relationship in the class **definition**?	
h. What Java **keyword** is used to refer to a **superclass** object from a subclass instance method?	
i. How does an **overloaded** method differ from an **overridden** method? Identify the differences between method signatures in your answer. *(4 points)*	
j. A declared type is used to dynamically bind method implementation at compile time (**True** or **False**)?	
k. What is the keyword used to create a class that cannot have any subclasses?	

2. On your own, create a Java program to solve the **Chapter 11 Programming Exercise 11.3** *(Subclasses of Account Class)* on page 442 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 modifications from the previous lab assignment: *(5 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. You should change the access from the previous lab to **protected** so that it is accessible within the subclasses. *(from previous lab)*

     - Class members previously declared **private** in Lab #6 should be modified to **protected** so they are accessible to any subclasses.

     - In place of the **getMonthlyInterestRate**, include a method **getMonthlyInterest** that returns the amount of monthly interest for the current account balance. *(from previous lab)*

     - The **protected Date** object can be formatted for use in an output string: http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html#syntax *(from previous lab)*

     - You should have void methods for **deposit** and **withdraw** that will be **overridden** (if necessary) in the subclass methods. The deposit and withdrawal amounts will be passed to these methods and the balance updated accordingly.

     - A **protected** *(this was **private** in Lab #6)* **String/StringBuilder** data field named **name** for the account holder's first and last name (default empty String).

   - Add the following **subclasses** of the **Account** class: *(this is new for Lab #7) (20 points)*

     - **CheckingAccount** and **SavingsAccount** account (see textbook for specific description of each). Note: The **CheckingAccount** has the ability to withdraw more than the balance (i.e. **overdraft amount**—this should be an additional **private** data member within the **CheckingAccount** class) whereas the **SavingsAccount** cannot have more than the balance withdrawn. *The differing subclass withdraw logic should be implemented in an overriden method of the superclass.*

     - The subclasses should use appropriate calls to the **superclass** within the **constructors**, **toString**, and any other methods that are dependent upon the superclass members.

- Add the following logic to the **TestAccount** program: *(this can be taken and/or modified from Lab #6) (4 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 **CheckingAccount** or **SavingsAccount** object.

    - The initial account information is then displayed **to the console** by overriding the **toString()** method for each user created class and subclass. Use the **static String format** method to create the formatted string for output in the following format: (*Hint: Determine the common output and include that in the Account class. Put the output specific to the Checking or Savings account in the appropriate class. Use the **super** keyword to invoke the superclass method to generate the common output.)*

Output from **Account** toString() method

```
Account Type:              Checking
ID:                            1122
Created:                   03/03/11
Owner:                  Sam Williams
Annual Rate:                  4.50%
Balance:                     550.00
```

Output from **CheckingAccount** toString() method

Output from **Account** toString() method

```
Account Type:              Savings
ID:                            2211
Created:                   03/03/11
Owner:                  Sam Williams
Annual Rate:                  4.50%
Balance:                  24,000.00
```

Output from **SavingsAccount** toString() method

    - The user has the ability to **withdraw** or **deposit** an amount from the account under the previously identified withdrawal requirements stipulated for each account type. Upon successful completion of a deposit or withdrawal, the amount deposited or withdrawn should be displayed with the adjusted balance, for example:

```
Deposit:                      50.00
Current Balance:             550.00
                -OR-
Withdraw:                     50.00
Current Balance:             500.00
```

- Complete the following programming activities:

    a. Create the source code files: **Account.java**, **CheckingAccount.java**, **SavingsAccount.java**, and **TestAccounts.java**. Add a block comment at the top of the file to identify your name, file, date, class, assignment, and short description of the program. *(4 points total, 1 point each)*

b. Compile the source code until no errors are found. The Java bytecode files **Account.class**, **CheckingAccount.class**, **SavingsAccount.class** and **TestAccounts.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. Include output for test cases that show the following *(label each appropriately)*: *(6 points)*
   - **Savings Account**
     - deposit
     - withdraw amount less than balance
     - withdraw amount greater than balance
   - **Checking Account**
     - deposit
     - withdraw amount less than balance
     - withdraw amount greater than balance, but less than overdraft amount
     - withdraw amount greater than balance and overdraft amount

f. Attach a **software generated** hardcopy output of the **UML class diagram** for the **Account**, **CheckingAccount** and **SavingsAccount** classes. Remember to include access specifiers for properties and methods. *(6 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 step is required for **any** assignment 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