Lab 10 – Inheritance (12 pts)

Lab Objectives

- Be able to derive a class from an existing class
- Be able to define a class hierarchy in which methods are overridden and fields are hidden
- Be able to use derived-class objects
- Implement a copy constructor

~~Deliverables~~

~~This lab has two tasks. When you have all tasks done, run the report on Blackboard. The report is a Blackboard test with short-answer, file-response, multiple-answer, and other types of questions.~~

~~In the report, you may be asked to provide code segments, Java source code files (must have extension.*java*), screenshot of program execution, files in PDF format, and your analysis of the results.~~

~~If a short answer question requests a code segment, please ensure that your input is readable – all **new lines and indents** are in place.~~

~~Screenshots in your report **must show a full screen**, so your computer can be identified. Please resize your IDE panels the way that the required dialog or output is visible along with the source code. Show as much source code as possible.~~

~~NOTE:~~

- ~~Use **Blackboard only** to submit your work; **no email** submission unless your instructor directs it.~~
- ~~If Blackboard gives you multiple submission attempts (usually three), the **last one** will be evaluated and graded.~~
- ~~**No late submissions**, **no changes** in your submission after the due date.~~

Introduction

In this lab, you will be creating new classes that are derived from a class called `BankAccount`. A checking account ***is a*** bank account and a savings account ***is a*** bank account as well. This sets up a relationship called inheritance, where `BankAccount` is the superclass and `CheckingAccount` and `SavingsAccount` are subclasses.

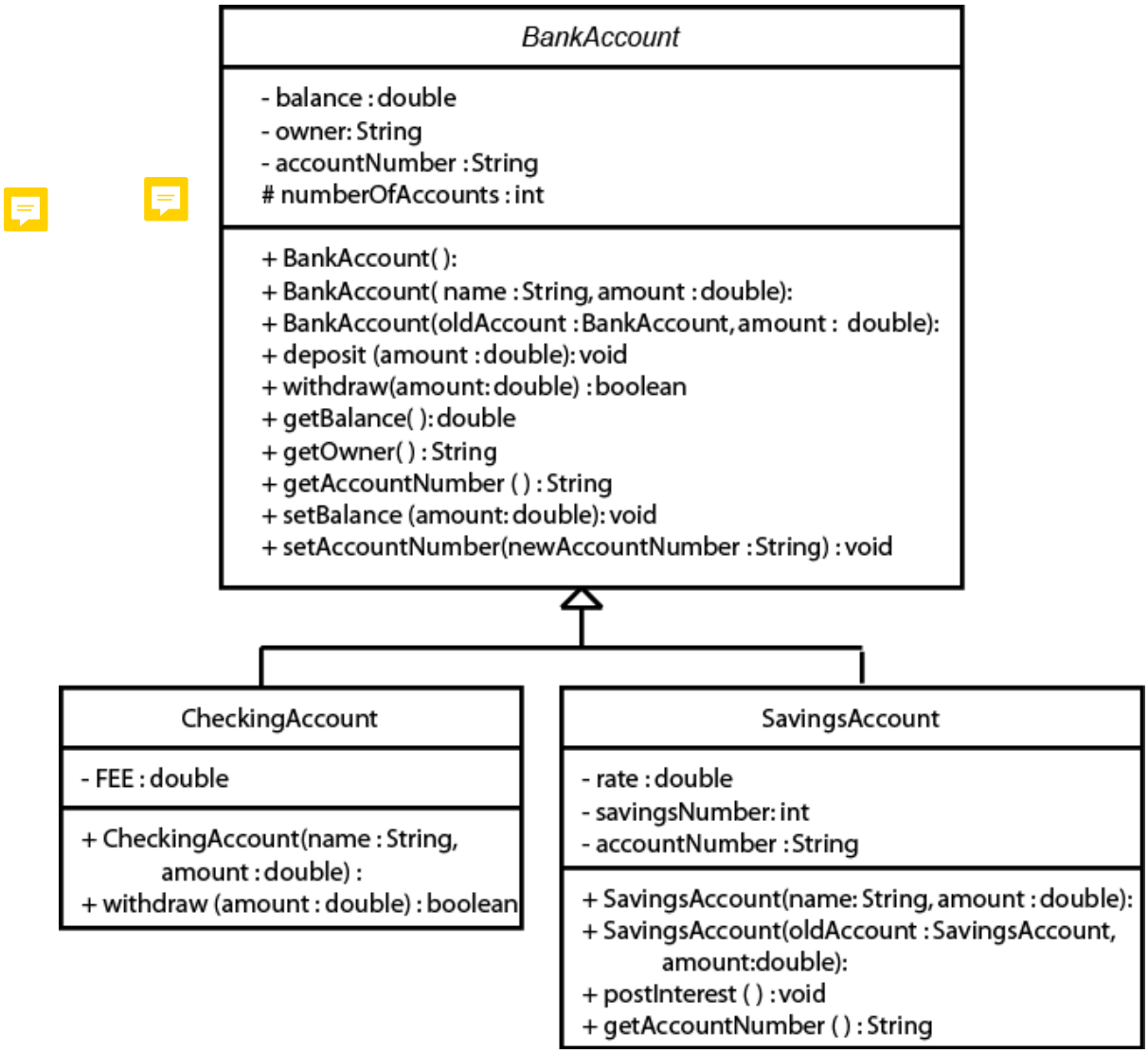
This relationship allows `CheckingAccount` to inherit attributes from `BankAccount` (like owner, balance, and `accountNumber`, but it can have new attributes that are specific to a checking account, like a fee for clearing a check. It also allows `CheckingAccount` to inherit

methods from `BankAccount`, like `deposit`, that are universal for all bank accounts.

You will write a withdraw method in `CheckingAccount` that overrides the `withdraw` method in `BankAccount`, in order to do something slightly different than the original `withdraw` method.

You will use an instance variable called `accountNumber` in `SavingsAccount` to hide the `accountNumber` variable inherited from `BankAccount`.

The UML diagram for the inheritance relationship is as follows:



Task #1 Extending the `BankAccount` Class (6 pts)

1. Copy the files *AccountDriver.java* and *BankAccount.java* as directed by your instructor. *BankAccount.java* is complete and will not need to be modified.
2. Create a new class called `CheckingAccount` that `extends` `BankAccount`.
3. It should contain a `static` constant `FEE` that represents the cost of clearing one check. Set it equal to 15 cents.
4. Write a constructor that takes a name and an initial amount as parameters. It should call the constructor for the superclass. It should initialize `accountNumber` to be the current value in `accountNumber` concatenated with –10 (all checking accounts at this bank are identified by the extension –10). There can be only one checking account for each account number. Remember since `accountNumber` is a private member in `BankAccount`, it must be changed through a mutator method.
5. Write a new instance method, `withdraw`, that overrides the `withdraw` method in the superclass. This method should take the amount to withdraw, add to it the fee for check clearing, and call the `withdraw` method from the superclass. Remember that to override the method, it must have the same method heading. Notice that the `withdraw` method from the superclass returns `true` or `false` depending on if it was able to complete the withdrawal or not. The method that overrides it must also return the same `true` or `false` that was returned from the call to the `withdraw` method from the superclass.
6. Compile and debug this class.

Task #2 Creating a Second Subclass (6 pts)

1. Create a new class called `SavingsAccount` that `extends` `BankAccount`.
2. It should contain an instance variable called `rate` that represents the annual interest rate. Set it equal to 2.5%.
3. It should also have an instance variable called `savingsNumber`, initialized to 0. In this bank, you have one account number, but can have several savings accounts with that same number. Each individual savings account is identified by the number following a dash. For example, 100001-0 is the first savings account you open, 100001-1 would be another savings account that is still part of your same account. This is so that you can keep some funds separate from the others, like a Christmas club account.
4. An instance variable called `accountNumber`, that will hide the `accountNumber`from the superclass, should also be in this class.
7. Write a constructor that takes a name and an initial balance as parameters and calls the constructor for the superclass. It should initialize `accountNumber` to be the current value in the superclass

`accountNumber` (the hidden instance variable) concatenated with a hyphen and then the `savingsNumber`.

5. Write a method called `postInterest` that has no parameters and returns no value. This method will calculate one month's worth of interest on the balance and deposit it into the account.
6. Write a method that overrides the `getAccountNumber` method in the superclass.
8. Write a copy constructor that creates another savings account for the same person. It should take the original savings account and an initial balance as parameters. It should call the copy constructor of the superclass, and assign the `savingsNumber` to be one more than the `savingsNumber` of the original savings account. It should assign the `accountNumber` to be the `accountNumber` of the superclass concatenated with the hyphen and the `savingsNumber` of the new account.
9. Compile and debug this class.
10. Use the `AccountDriver` class to test out your classes. If you named and created your classes and methods correctly, it should not have any difficulties. If you have errors, do not edit the `AccountDriver` class. You must make your classes work with this program.
11. Running the program should give the following output:

```
Account Number 100001-10 belonging to Benjamin Franklin
Initial balance = $1000.00
After deposit of $500.00, balance = $1500.00
After withdrawal of $1000.00, balance = $499.85

Account Number 100002-0 belonging to William Shakespeare
Initial balance = $400.00
After deposit of $500.00, balance = $900.00
Insufficient funds to withdraw $1000.00, balance = $900.00
After monthly interest has been posted, balance = $901.88

Account Number 100002-1 belonging to William Shakespeare
Initial balance = $5.00
After deposit of $500.00, balance = $505.00
Insufficient funds to withdraw $1000.00, balance = $505.00

Account Number 100003-10 belonging to Isaac Newton
```