Chapter 11 Lab

Inheritance

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

Introduction

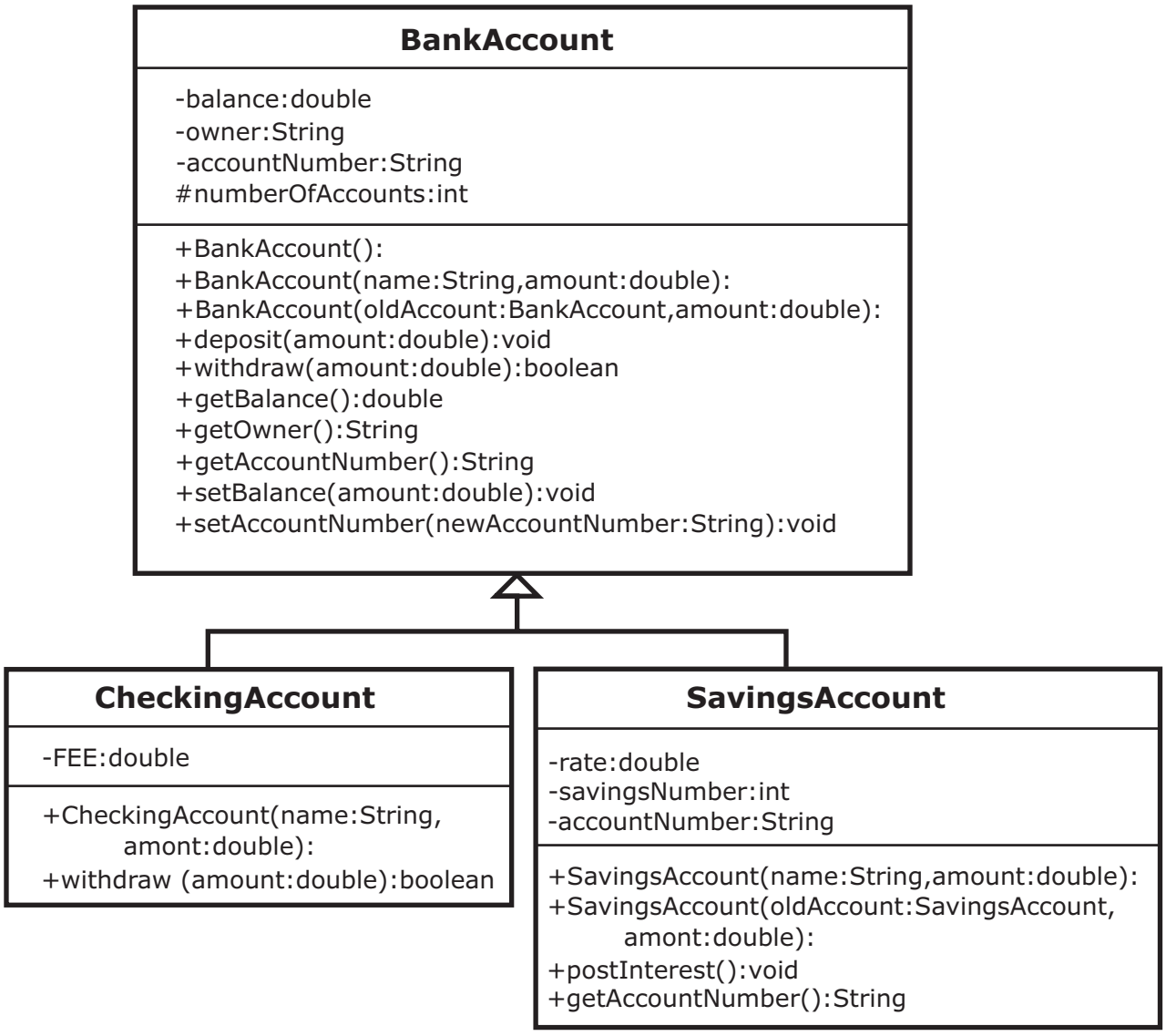
In this lab, you will be creating new classes that are derived from a class called BankAccount. A checking account ***is a*** bank account and a savings account ***is a*** bank account as well. This sets up a relationship called inheritance, where BankAccount is the superclass and CheckingAccount and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

1. Copy the files *AccountDriver.java* (code listing 11.1) and *BankAccount.java* (code listing 11.2) from www.aw.com/cssupport or 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 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

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.
5. 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.
6. 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.
7. 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, 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 hypen 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
Insuffient 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
Insuffient funds to withdraw $1000.00,  balance = $505.00

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

Code Listing 11.1 (AccountDriver.java)

```
import java.text.*;               // to use Decimal Format
/**Demonstrates the BankAccount and derived classes*/
public class AccountDriver
{
     public static void main(String[] args)
     {
          double put_in = 500;
          double take_out = 1000;

          DecimalFormat myFormat;
          String money;
          String money_in;
          String money_out;
          boolean completed;

          // to get 2 decimals every time
          myFormat = new DecimalFormat("#.00");

          //to test the Checking Account class
          CheckingAccount myCheckingAccount =
               new CheckingAccount ("Ben Franklin", 1000);
          System.out.println ("Account Number "
               + myCheckingAccount.getAccountNumber() +
               " belonging to " + myCheckingAccount.getOwner());
          money  = myFormat.format(
               myCheckingAccount.getBalance());
          System.out.println ("Initial balance = $" + money);
          myCheckingAccount.deposit (put_in);
          money_in = myFormat.format(put_in);
          money  = myFormat.format(
               myCheckingAccount.getBalance());
          System.out.println ("After deposit of $" + money_in
               + ",  balance = $" + money);
          completed = myCheckingAccount.withdraw(take_out);
          money_out = myFormat.format(take_out);
```

Code Listing 11.1 continued on next page.

```
money  = myFormat.format(
      myCheckingAccount.getBalance());
if (completed)
{
      System.out.println ("After withdrawal of $" +
            money_out   + ",  balance = $" + money);
}
else
{
      System.out.println ("Insuffient funds to " +
            " withdraw $" + money_out   +
            ",  balance = $" + money);
}
System.out.println();

//to test the savings account class
SavingsAccount yourAccount =
      new SavingsAccount ("William Shakespeare", 400);
System.out.println ("Account Number "
      + yourAccount.getAccountNumber() +
      " belonging to " + yourAccount.getOwner());
money  = myFormat.format(yourAccount.getBalance());
System.out.println ("Initial balance = $" + money);
yourAccount.deposit (put_in);
money_in = myFormat.format(put_in);
money  = myFormat.format(yourAccount.getBalance());
System.out.println ("After deposit of $" + money_in
      + ",  balance = $" + money);
completed = yourAccount.withdraw(take_out);
money_out = myFormat.format(take_out);
money  = myFormat.format(yourAccount.getBalance());
if (completed)
{
      System.out.println ("After withdrawal of $" +
            money_out   + ",  balance = $" + money);
}
else
{
```

Code Listing 11.1 continued on next page

```
System.out.println ("Insuffient funds to " +
      "withdraw $" + money_out   +
      ",  balance = $" + money);
}
yourAccount.postInterest();
money  = myFormat.format(yourAccount.getBalance());
System.out.println ("After monthly interest " +
"has been posted," + "balance = $"   + money);
System.out.println();

// to test the copy constructor of the savings account
//class
SavingsAccount secondAccount =
      new SavingsAccount (yourAccount,5);
System.out.println ("Account Number "
      + secondAccount.getAccountNumber()+
      " belonging to " +
      secondAccount.getOwner());
money = myFormat.format(secondAccount.getBalance());
System.out.println ("Initial balance = $" + money);
secondAccount.deposit (put_in);
money_in = myFormat.format(put_in);
money = myFormat.format(secondAccount.getBalance());
System.out.println ("After deposit of $" + money_in
      + ", balance = $" + money);
secondAccount.withdraw(take_out);
money_out = myFormat.format(take_out);
money = myFormat.format(secondAccount.getBalance());
if (completed)
{
      System.out.println ("After withdrawal of $" +
            money_out + ",  balance = $" + money);
}
else
{
```

Code Listing 11.1 continued on next page

```
                System.out.println ("Insuffient funds to " +
                      "withdraw $" + money_out   +
                      ",  balance = $" + money);
            }
            System.out.println();

            //to test to make sure new accounts are numbered
            //correctly
            CheckingAccount yourCheckingAccount =
                  new CheckingAccount ("Isaac Newton", 5000);
            System.out.println ("Account Number "
                  + yourCheckingAccount.getAccountNumber()
                  + " belonging to "
                  + yourCheckingAccount.getOwner());
      }
}
```

Code Listing 11.2 (BankAccount.java)

```
/**Defines any type of bank account*/
public abstract class BankAccount
{
     /**class variable so that each account has a unique
     number*/
     protected static int numberOfAccounts = 100001;

     /**current balance in the account*/
     private double balance;
     /** name on the account*/
     private String owner;
     /** number bank uses to identify account*/
     private String accountNumber;

     /**default constructor*/
     public BankAccount()
     {
          balance = 0;
          accountNumber = numberOfAccounts + "";
          numberOfAccounts++;
     }

     /**standard constructor
     @param name the owner of the account
     @param amount the beginning balance*/
     public BankAccount(String name, double amount)
     {
          owner = name;
          balance = amount;
          accountNumber = numberOfAccounts + "";
          numberOfAccounts++;
     }

     /**copy constructor creates another account for the same
     owner
     @param oldAccount the account with information to copy
```

Code Listing 11.2 continued on next page.

```
@param the beginning balance of the new account*/
public BankAccount(BankAccount oldAccount, double amount)
{
     owner = oldAccount.owner;
     balance = amount;
     accountNumber = oldAccount.accountNumber;
}

/**allows you to add money to the account
@param amount the amount to deposit in the account*/
public void deposit(double amount)
{
     balance = balance + amount;
}

/**allows you to remove money from the account if
enough money is available,returns true if the transaction
was completed, returns false if the there was not enough
money.
@param amount  the amount to withdraw from the account
@return true if there was sufficient funds to complete
the transaction, false otherwise*/
public boolean withdraw(double amount)
{
     boolean completed = true;

     if (amount <= balance)
     {
          balance = balance - amount;
     }
     else
     {
          completed = false;
     }
     return completed;
}
```

Code Listing 11.2 continued on next page

```
    /**accessor method to balance
    @return the balance of the account*/
    public double getBalance()
    {
         return balance;
    }

    /**accessor method to owner
    @return the owner of the account*/
    public String getOwner()
    {
         return owner;
    }

    /**accessor method to account number
    @return the account number*/
    public String getAccountNumber()
    {
         return accountNumber;
    }

    /**mutator method to change the balance
    @param newBalance the new balance for the account*/
    public void setBalance(double newBalance)
    {
         balance = newBalance;
    }

    /**mutator method to change the account number
    @param newAccountNumber the new account number*/
    public void setAccountNumber(String newAccountNumber)
    {
         accountNumber = newAccountNumber;
    }
}
```