|  |  |
| --- | --- |
|  | |
| Develop Bank and Loan Account Classes | |
| **Updated** | **10.28.2018 2:35 PM** |

# **Problem 1:**

Develop a class called **BankAccount** that stores a name, account number and balance. The constructor for the class must allow the name and account number to be specified, and set the initial balance to 0. The public interface for the class must include methods to perform the following tasks:

1. get name, get account number, get balance
2. deposit (specify an amount to add to balance)
3. withdraw (specify an amount to subtract from balance) -- note that the maximum withdrawal is the current balance, as the balance cannot go below 0
4. an implementation of **toString**(), that will return a string with the account details, in a format like "John Smith: Balance = €1000"

None of these methods should display information on screen -- the display should all happen from the test program, described below. Ensure that all class member variables and methods are defined to be public or private as appropriate.

To test the class, write a program that does the following:

1. Create two **BankAccount** objects, with a name and account number for each
2. Deposit money in both accounts and withdraw some money from one
3. For each account, display a message with the name and balance (use **toString)**

# **Approach**

The question in this case is quite detailed regarding the design of the class so I just followed that specification closely.

# **BankAccount Design**

Declare variables for Name, Account Number and Balance

Constructor allows the name and account number to be specified

Set the initial balance to 0

Methods to get name, get account number, get balance

Deposit Method specifies an amount to add to balance

Withdraw Method

Check for sufficient funds

IF sufficient funds withdraw cash

Update balance

ELSE alert user Insufficient Funds try a smaller amount

Method returns a string with the account details

EXIT

# **Testing**

The program was run several times with different inputs to ensure the code behaves as expected for each possible execution scenario including the input of invalid and out of bound parameters. The program was also tested for scaling.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Test** | **Expected Result** | **Actual Result** |
| 1 | Create **BankAccount**  objects, with a name and account number | **BankAccount**  Object is created with a name and account number | **BankAccount**  Object is created with a name and account number |
| 2 | Deposit money in bank account accounts | Money is deposited in bank account | Money is deposited in bank account |
| 3 | Withdraw money from a bank account. | Money is withdrawn from the bank account | Money is withdrawn from the bank account |
| 4 | Attempt to withdraw more funds than are available in the account | User gets message “Insufficient available funds for transaction” and no money is withdrawn | User gets message “Insufficient available funds for transaction” and no money is withdrawn |
| 5 | Ensure toString returns account details | toString returns account details | toString returns account details |

# **Source Code – BankAccount.java**

|  |
| --- |
| /\*mmcmahon\_wk4\_question1\_BankAccount.java  \* Develop a class called BankAccount that stores a name, account number and balance. The constructor for the class must  \* allow the name and account number to be specified, and set the initial balance to 0. The public interface for the class  \* must include methods to perform the following tasks:  \*  \* 1. get name, get account number, get balance  \* 2. deposit (specify an amount to add to balance)  \* 3. withdraw (specify an amount to subtract from balance) -- note that the maximum withdrawal is the current balance,  \* as the balance cannot go below 0  \* 4. an implementation of toString(), that will return a string with the account details, in a format like  \* "John Smith: Balance = €1000"  None of these methods should display information on screen -- the display should all happen from the test program  Ensure that all class member variables and methods are defined to be public or private as appropriate.  References:  \* http://java.about.com/od/workingwithobjects/a/useinheirtance.htm  \* http://www.uoc.edu/in3/e-math/docs/java/ch13.pdf  \*/  **public** **class** BankAccount {  **private** String name; // declare account name  **private** **int** accountNumber; // declare account number  **protected** **double** balance; // declare the balance    // Account constructor allows the name and account number to be specified  **public** BankAccount( String theName, **int** theAccountNumber)  {  name = theName;  accountNumber = theAccountNumber;  balance = 0;  } // end Account constructor    // Get returns the account name  **public** String getName()  {  **return** name;  } // end method getName    // Get returns the account number  **public** **int** getAccountNumber()  {  **return** accountNumber;  } // end method getAccountNumber    // Get returns the total balance  **public** **double** getBalance()  {  **return** balance;  } // end method getTotalBalance    **protected** **void** setBalance(**double** balance)  {  **this**.balance = balance;  }    // deposit an amount to the account  **public** **void** deposit(**double** amount)  {  balance += amount; // add to balance  } // end method deposit    // withdraw an amount from the account  **public** **void** withdraw( **double** amount )  {  **if** ( amount <= balance) // verify if there is sufficient available funds  {  balance -= amount; // subtract amount from balance  } // end if  **else**  {  System.***out***.println("Insufficient available funds for transaction - Try a smaller amount"); //Throw exception  } // end else    } // end method withdraw    // Return a string with the account details in format like "John Smith: Balance = €1000"  **public** String toString()  {  **return** "Account Name: " + **this**.name +  "\nAccount Number " + **this**.accountNumber +  "\nAccount Balance = €" + **this**.balance;  } // end toString method    } // end class BankAccount |

# **Source Code – TestBankAccount.java**

|  |
| --- |
| /\*mmcmahon\_wk4\_question1\_TestBankAccount.java  \* To test the class, write a program that does the following:  \*  \* 1. Create two BankAccount objects, with a name and account number for each  \* 2. Deposit money in both accounts and withdraw some money from one  \* 3. For each account, display a message with the name and balance (use toString)  \*  \*/  **public** **class** TestBankAccount {  **public** **static** **void** main(String[] args) {    //First BankAccount Object with a name and account number  BankAccount\_Dev1 BankAcount1 = **new** BankAccount\_Dev1("John Smith", 12345);  System.***out***.println("\nDeposit €1000 in Account 1");  BankAcount1.deposit(1000); //Deposit €1000 in the Account  System.***out***.println(BankAcount1.toString()+ "\n"); //Display a message with the name and balance (use toString)      //First BankAccount Object with a name and account number  BankAccount\_Dev1 BankAcount2 = **new** BankAccount\_Dev1("John Doe", 67891);  System.***out***.println("\nDeposit €2000 in Account 2");  BankAcount2.deposit(2000); //Deposit €2000 in the Account  System.***out***.println(BankAcount2.toString()+ "\n"); //Display a message with the name and balance (use toString)  System.***out***.println("\nWithdraw €2500 from Account 2");  BankAcount2.withdraw(2500); //Withdraw €500 from Account  System.***out***.println(BankAcount2.toString()+ "\n"); //Display a message with the name and balance (use toString)  System.***out***.println("\nWithdraw €500 from Account 2");  BankAcount2.withdraw(500); //Withdraw €500 from Account  System.***out***.println(BankAcount2.toString()+ "\n"); //Display a message with the name and balance (use toString)  }  } |

# **Problem 2:**

Develop a new class called **LoanAccount**, based on **BankAccount**. This class has an overdraft facility, so that the user can withdraw money up to the overdraft limit. The class must store an overdraft limit for the account. It must also store the current interest rate, but this is the same for all accounts (it is not part of any one account).

The constructor will require a name and account number to be specified, and will invoke the constructor of the **BankAccount**class and also set the overdraft limit to 0. Also, the **withdraw** method from **BankAccount**will have to be overridden in this class, so that a negative balance is permitted. You will need to add a *protected* **setBalance** method to **BankAccount**, that changes the balance without checking it, since **BankAccount**does not normally allow negative balances.

The public interface must also include the following additional methods:

1. set/get the overdraft limit
2. calculate projected interest (0 if the balance is positive, otherwise multiply current balance by current interest rate)

# **Approach**

The question in this case is quite detailed regarding the design of the class so I just followed that specification closely. Since Loan Account ‘is a’ type of Bank Account I decided to use inheritance.

# **Design**

Add protected setBalance method to BankAccount class

Create a new Loan Account class which extends Bank Account

Set interest rate to 5%,

Set overdraft to 0

Set interest variable

The constructor takes name and account number

Invoke the constructor of the BankAccount class

Set the overdraft limit.

Create setter method for the Overdraft

Create getter method for the Interest Rate

Create getter method for the Overdraft

Create getter method for calculation of available funds

Create getter method for calculation of interest

Override the withdraw method from BankAccount so that a negative balance is permitted.

If balance + overdraft are greater than withdrawal then proceed

Else print "Insufficient available funds for transaction"

If proceed then ensure we only charge fee if account is overdrawn

If not overdrawn proceed as normal and update balance

# **Testing**

Since this was a small code assignment it did not require a lot of testing

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Test** | **Expected Result** | **Actual Result** |
| 1 | Create **BankAccount**  objects, with a name, account number and overdraft | **BankAccount**  Object is created with a name, account number and overdraft | **BankAccount**  Object is created with a name, account number and overdraft |
| 2 | Deposit money in bank account accounts | Money is deposited in bank account | Money is deposited in bank account |
| 3 | Withdraw money from a bank account. | Money is withdrawn from the bank account | Money is withdrawn from the bank account |
| 4 | Withdraw more money than the users balance and use the overdraft | Can withdraw from overdraft | Can withdraw from overdraft |
| 5 | Check that interest is calculated on overdraft amount | Interest is calculated on overdraft amount | Interest is calculated on overdraft amount |
| 6 | Attempt to withdraw more funds than are available in the overdraft | User gets message “Insufficient available funds for transaction” and no money is withdrawn | User gets message “Insufficient available funds for transaction” and no money is withdrawn |
| 7 | Ensure toString returns account details | toString returns account details | toString returns account details |

# **Source Code – LoanAccount.java**

|  |
| --- |
| /\*mmcmahon\_wk4\_question2\_LoanAccount.java  \* Develop a new class called LoanAccount, based on BankAccount. This class has an overdraft facility, so that the user  \* can withdraw money up to the overdraft limit. The class must store an overdraft limit for the account. It must also  \* store the current interest rate, but this is the same for all accounts (it is not part of any one account).  \* The constructor will require a name and account number to be specified, and will invoke the constructor of the BankAccount  \* class and also set the overdraft limit to 0. Also, the withdraw method from BankAccount will have to be overridden in this  \* class, so that a negative balance is permitted. You will need to add a protected setBalance method to BankAccount, that  \* changes the balance without checking it, since BankAccount does not normally allow negative balances.  \*  \* The public interface must also include the following additional methods:  \* set/get the overdraft limit  \* calculate projected interest (0 if the balance is positive, otherwise multiply current balance by current interest rate)  \*  \* References  \* http://java.about.com/od/workingwithobjects/a/useinheirtance.htm  \* http://www.uoc.edu/in3/e-math/docs/java/ch13.pdf  \*/  **public** **class** LoanAccount **extends** BankAccount {    **private** **double** interestRate = 5; //declare interestRate  **private** **double** overdraft = 0; //declare overdraft  **private** **double** interest; //declare interest    /\*Constructor will require a name and account number to be specified, and will invoke the constructor of the BankAccount  \* class and also set the overdraft limit to 0.\*/  **public** LoanAccount(String name, **int** accountNumber, **double** overdraft)  {  **super** (name, accountNumber);  **this**.overdraft = overdraft;  }    //set method for the Overdraft  **public** **void** setOverdraft(**double** overdraftLimit)  {  **this**.overdraft = overdraftLimit;  } // end set method Overdraft    //get method for the Interest Rate  **public** **double** getInterestRate()  {  **return** **this**.interestRate;  } // end get method InterestRate    //get method for the Overdraft  **public** **double** getOverdraft()  {  **return** **this**.overdraft;  } // end get method Overdraft    //get method for calculation of available funds  **public** **double** getAvailableFunds()  {  **return** **super**.getBalance() + **this**.overdraft;  }    //get method for calculation of interest  **public** **double** getInterest()  {  **return** **this**.interest = Math.*abs*(**super**.balance \* **this**.interestRate / 100);  }    /\*The withdraw method from BankAccount is overridden so that a negative balance is permitted\*/  @Override  **public** **void** withdraw( **double** amount )  {  **if** (getAvailableFunds() - (amount + getInterest()) > 0)  {  **if** (balance - amount < 0) // only charge fee if account is overdrawn  {    **super**.balance = balance - (amount + getInterest());  }  **else** {  **super**.balance -= amount;  }  }  **else**  {  System.***out***.println("Insufficient available funds for transaction - Try a smaller amount"); //Throw exception  } // end else  }    // Return a string with the account details in format like "John Smith: Balance = €1000"  **public** String toString()  {  **return** "Account Name: " + **super**.getName() +  "\nAccount Balance = €" + **super**.getBalance() +  "\nAvailable Funds: " + getAvailableFunds() +  "\nOverdraft: " + getOverdraft() +  "\nInterest Rate: " + getInterestRate() +  "\nInterest: " + getInterest();  } // end toString method  } |

# **References:**

Deitel, H. & Deitel, P. (2010). *Java* (9th ed.). Upper Saddle River, NJ: Prentice Hall.

Leahy, P. (2016). *Using Inheritance*. *About.com Tech*. Retrieved 18 September 2016, from <http://java.about.com/od/workingwithobjects/a/useinheirtance.htm>