Intermediate Java II
MISM/MSIT 95-713
Homework 3

There are 2 exercises in this homework. The first exercise is 25 points. The second exercise is 75 points.

The objectives of this homework include:

- Defining classes and creating objects
    - Providing encapsulation
    - Writing constructors
    - Implementing methods specifically the
        - Setter/getter methods
        - toString() method
        - equals() method
- Implementing composition and dependency relationships among classes
- Implementing class hierarchies
- Implementing UML models
- Creating and using packages
- Practicing access modifiers
- Practicing polymorphism
- Practicing overriding
- Practicing abstract classes
- Practicing reflection
- Practicing basic IO
- Providing documentation comments and generating Java documentation files.

For this homework assignment and for all the upcoming homework assignments, follow the commenting and coding convention discussed in the class, unless otherwise specified explicitly. Make sure you follow the minimal class description guidelines for all the classes you introduce, except for the test driver classes that contain the main function. Name your files as specified in each of the exercises below. Put all your java files into a zip file named *yourandrewid*_homework3.zip and submit it to the blackboard before the due date.

1. Enhance the ReflectionTest.java of the textbook in the following way: Copy the code of the ReflectionTest.java to a file named Homework3_1.java. In the *printFields*() method, while printing the field information, print the value of each field along with its name. To do this, you need create an instance of the class through the default constructor, and for each field, call the field.get() method. Enclose the whole code of the printFields method with a *try { … }catch (Exception e) { e.printStackTrace(); }* Introduce an Employee class that has an instance variable *name* of type String and an *id* of type integer. Initialize the name to "Super Employee" and id to 10 within the Employee class. Test the main function with Employee class.

2. Consider the UML conceptual model Figure 1. You will write class definitions for each of the concepts.

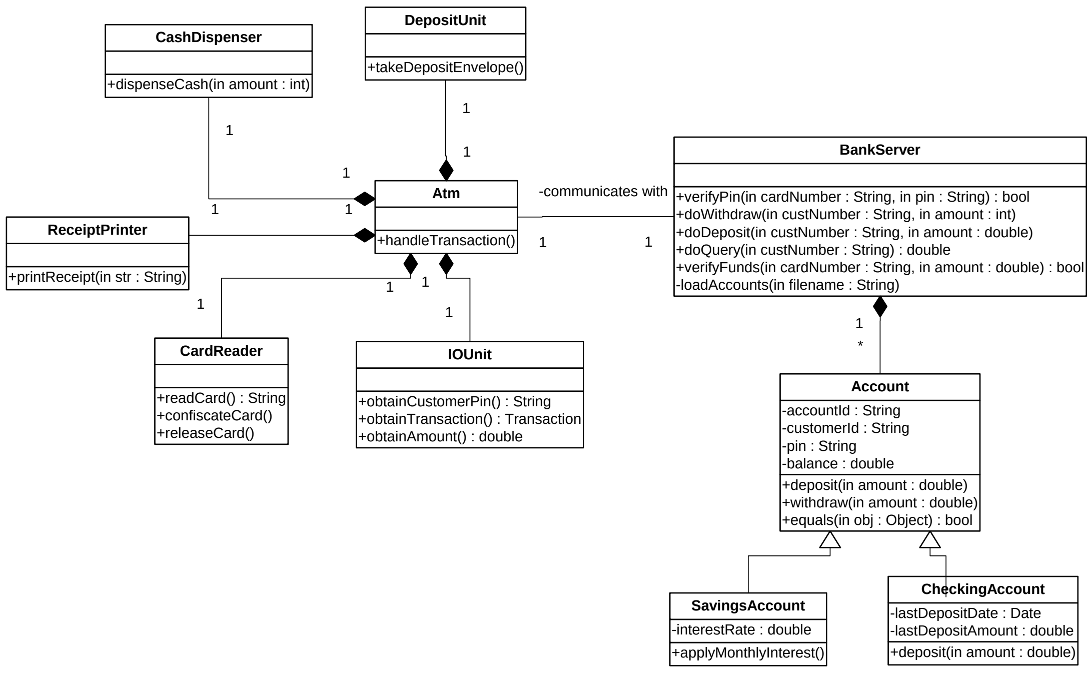
CashDispenser
+dispenseCash(in amount : int)
DepositUnit
+takeDepositEnvelope()
ReceiptPrinter
+printReceipt(in str : String)
Atm
+handleTransaction()
-communicates with
BankServer
+verifyPin(in cardNumber : String, in pin : String) : bool
+doWithdraw(in custNumber : String, in amount : int)
+doDeposit(in custNumber : String, in amount : double)
+doQuery(in custNumber : String) : double
+verifyFunds(in cardNumber : String, in amount : double) : bool
-loadAccounts(in filename : String)
CardReader
+readCard() : String
+confiscateCard()
+releaseCard()
IOUnit
+obtainCustomerPin() : String
+obtainTransaction() : Transaction
+obtainAmount() : double
Account
-accountId : String
-customerId : String
-pin : String
-balance : double
+deposit(in amount : double)
+withdraw(in amount : double)
+equals(in obj : Object) : bool
SavingsAccount
-interestRate : double
+applyMonthlyInterest()
CheckingAccount
-lastDepositDate : Date
-lastDepositAmount : double
+deposit(in amount : double)
1
*

Figure 1 Inheritance Hierarchy for Exercise 2

Have the following classes in a file called Atm.java under the package edu.cmu.heinz.ij95713.Atm: CashDispenser, DepositUnit, ReceiptPrinter, CardReader, IOUnit and Atm. Only the Atm class will be public in this file.

ReciptPrinter has one method, printReceipt(), which takes a parameter of String type and simply prints that parameter to the console.

CardReader has the following methods:

- readCard() asks for the card number from the user, reads it and returns it.
- confiscateCard() simply prints out a message saying that the card is confiscated.
- releaseCard simply prints out a message saying that the card is released.

CashDispenser has one method, dispenseCash(), which takes an integer amount as the parameter and prints a message saying that money in the provided amount is dispensed.

DepositUnit has one method, takeDepositEnvelope(), which prints a message saying that envelope is successfully received.

IOUnit defines a Transaction enum with objects of DEPOSIT, WITHDRAW, QUERY and CANCEL. IOUnit has the following methods:

- obtainCustomerPin() prompts the user for a pin, reads and returns it.

- obtainTransaction() prompts the user to select a transaction from a menu, reads the input and returns the selected transaction.
- obtainAmount() prompts the user for an amount, reads and returns it.

Atm class has the following methods:

- Constructor that takes a BankServer object reference.
- handleTransaction() handles a user transaction. It executes the sequence of following:
    - Read the card.
    - Ask the user to enter the pin and read it.
    - Verify the pin invoking the verifyPin() method of the bank. If the pin is verified, continue with the transaction. Allow user to enter the pin number up to three times. After the three trials, if the user cannot provide the correct pin, confiscate the card and finish the session. If user enters the correct pin, continue with the following steps.
    - Obtain transaction type from user.
    - If transaction is DEPOSIT
        - Obtain amount to be deposited from the user
        - Take deposit envelope
        - Send doDeposit message to bank server with card number and amount
        - Print a receipt including the total amount in the account.
    - If transaction is WITHDRAW
        - Obtain the withdraw amount from the user
        - Verify the availability of funds via the BankServer method verifyFunds(). Continue with the following if funds are verified:
            - Send doWithdraw message to bank server with card number and amount
            - Dispense cash
            - Print a receipt including the total amount in the account.
        - If funds are not verified, print receipt about not having enough funds in the account
    - If transaction is QUERY
        - Send doQuery message to bank server with card number
        - Print a receipt including the current amount in the account
    - If transaction is CANCEL
        - Print a receipt saying transaction is cancelled
    - Release card at the end of any transaction.

Have the following classes in a file named BankServer.java under the package edu.cmu.heinz.ij95713.Bank: Account, SavingsAccount, CheckingAccount and BankServer where only BankServer is a public class. You will define Java classes for Account, SavingsAccount and CheckingAccount concepts. Make sure to define equals() method for these classes. deposit() method increases the balance by the input parameter *amount*. widthdraw() method reduces the balance by the input parameter *amount.* applyMonthlyInterest() will calculate the monthly interest and add it to the balance. Note that deposit() method is overridden in the CheckingAccount class. Implement it so that lastDepositDate and lastDepositAmount fields are updated in addition to the inherited behavior.

BankServer has the following methods:

a. loadAccounts() reads a input file that contains the account information and creates the account objects. A sample input file is shown below
   ```
   S|11111|22221|3333|1000|2.2
   C|11112|22222|3333|5000|102.5|2012|8|12|11|32
   ```
   The first field is either an S for a savings account or a C for a checking account. The remaining fields for savings account are account id, customer id, pin, balance and interest rate. Other fields for checking account are account id, customer id, pin, balance, amount of last deposit, year, month, day, hour and minute of the last deposit date. loadAccount() should read this file and create account objects. Call this method in the constructor of BankServer with the name of the input file.
b. In verifyFunds() method, you will search for the account and if it exists make sure its balance is greater than or equal to the amount. In all of the BankServer methods, use the account number that you read with readCard() method of the IOUnit class as the key for searching the accounts.
c. verifyPin() verifies the pin for the account. If account doesn't exist, it returns false.
d. doWithdraw() invokes withdraw() method on the account, if exists.
e. doDeposit() invokes deposit() method on the account, if exists.
f. doQuery() returns the balance of the account. If account doesn't exist, it returns 0.

Have a class Homework3_2 in file Homework3_2.java under the default package. Define a main function for Homework3_2 that creates a bank server object and an Atm object. Call the handleTransaction() method on the Atm object.