CS212: Paradigms
Dr. Gross, Spring 2016

Lab 4 - Making Bank

1. In your lab project, open Documents/lab04/bank.uml
    a. Go to File-Save As and save it as bank-lastname-lastname.uml in the same folder
    b. Did you save a copy? Don't edit the original!
2. Click on that file in NetBeans and go to Team->Add
3. Go to Team->Commit… and enter "created copy of UML file"
4. Make your design
5. Create a new package called edu.blackburn.cs.cs212.bank.lastnamelastname
6. Copy the files from your lecture project package edu.blackburn.cs.cs212sp16.bankv2 to the new package above in you lab project
7. Click on those files in NetBeans and go to Team->Add
8. Go to Team->Commit… and enter "created copy of base Java classes"

Despite Dakota spoiling the fun surprise for all of you, I am going ahead with the plan to have you finish the bank. There will be no interface, just a series of events. You should print out each event as it happens.

Newb Mode:

1. In Runner create an instance of CheckingAccount, setting the balance when you instantiate it
2. Print the account (hint: what is Account's superclass?)
3. Make a mix of five deposits and five withdrawals; don't worry about fees and overdrafts
4. After each deposit/withdrawal, print out the account
5. Implement the methods that have been stubbed out so far
6. Make sure CheckingAccount works (i.e., it passes the tests you wrote in steps 2 & 3)
7. Make sure your implementation still matches your design
8. Give a shout if you can't figure out how to implement something; do NOT change the design at this point

Easy Mode:

Great, now we have a checking-only system. Let's implement SavingAccount.

1. Create the SavingsAccount class, extend Account, and make method stubs
2. In Runner create an instance of SavingsAccount, setting the balance & interest rate
3. Print the account (hint: what is Account's superclass?)
4. Make a mix of five deposits and five withdrawals; don't worry about fees and overdrafts
5. After each deposit/withdrawal, print out the account
6. Implement the methods that have been stubbed out so far (no need to actually add interest)
7. Make sure SavingsAccount works (i.e., it passes the tests you wrote in steps 2 & 3)
8. Make sure your implementation still matches your design
9. Give a shout if you need help implementing something; do NOT change the design at this point

Hard Mode:

Banks don't just keep track of your balance; they keep track of every transaction (debit, credit). A transaction should just be a record of the old balance, the amount of the change (in positive numbers for a deposit, or negative numbers for a withdrawal), and the final balance.

1. Add what you need to Transaction in the UML
2. Which class's objects should keep track of the Transaction objects?
3. Which methods will create them?
4. Where will you store transactions? (Hint: Account)
5. How will you store them? (Hint: an array, not a good long-term solution)

Nightmare Mode

Create a Loan class.

1. Add a Loan class to your UML
2. It's going to inherit, as you can guess, but from what, and why?
3. Add any methods/attributes you need; keep track of the interest rate, but don't use it
4. Implement the class
5. Is there a method here that doesn't make sense? (Hint: yes) This use of inheritance fails the Liskov substitution principle, but that's OK, we're rebels

Make sure all of your files are added, and then commit them, and push them. Please do NOT push code that doesn't compile.