Hacettepe University
Department of Computer Engineering
Bil138 Programming Laboratory

Subject : OOP (Object Oriented Programming)
Programming Language : Java
Submission Date : 12/04/2011
Deadline : 26/4/2011
Advisors : R.A. Levent Seçkin, Dr. Sevil Şen, Dr. Erkut Erdem
Experiment number : 2

AIM

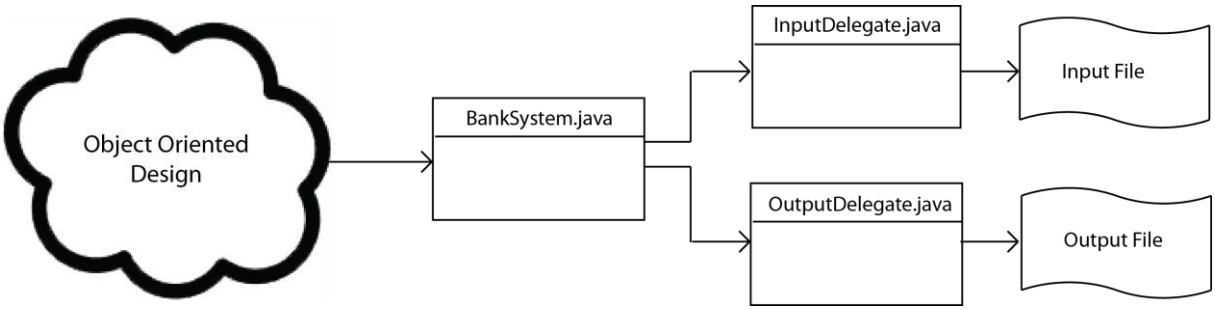
In this experiment you are aimed to get familiar with Inheritance in Object Oriented Programing. You will execute defined commands that are taken from a command file and store the result to output file. Also you will read main data structures from a text file. While doing this you have to develop a **Bank System**.

Inheritance *is a way to compartmentalize and reuse code by creating collections of attributes and behaviors called objects which can be based on previously created objects. In classical inheritance where objects are defined by classes, classes can inherit other classes. The new classes, known as subclasses (or derived classes), inherit attributes and behavior of the pre-existing classes, which are referred to as superclasses (or ancestor classes). The inheritance relationships of classes gives rise to a hierarchy. In prototype-based programming, objects can be defined directly from other objects without the need to define any classes, in which case this feature is called differential inheritance.*[wikipedia]

Experiment

For this experiment you are expected to develop a simple bank system. A bank system can have branchs, which are related with banks, persons, cards, sub-cards, deposit accounts and checking accounts. Accounts will be defined on branches and they should belong to a single person, cards can be defined only on checking accounts. Sub-cards cannot be defined on another sub-card, so they should belong to a regular card. A regular card can have multiple sub-cards. Sub-cards are derived from regular cards for another person. Your application will take input commands from an input file and data fields from a data file. Full path of the input file and data file will be given as program argument. Your application prints the outputs of the commands into an output file. Names of these files will be given as argument in the following order: <data_file> <input_file> <output_file>. In other words your program will be executed as: java BankSystem <data_file ><input_file> <output_file>.

Your application must have full Object Oriented Design and all your different data structures must be defined as a class in a remarkable design. General design of your application have to be suitable with this design:



The cloud named 'Object Oriented Design' will be implemented by yourself. This design is the most important part of your program. You have to remember inheritance can help you while you are doing your design. You have to submit your design in report using the notation above.

Your application should support the following features:

CREATE CHECKING_ACCOUNT <iban> <person_id> <bank> <branch_name> <amount>
Checking account is a simple account which doesn't have any commision ratio or payment date. Checking accounts don't have any specialities. Given amount will be added to this account at the begining.
results:
<iban> named account created. <person_id> has <amount>
ERROR: <person_id> person not found

CREATE DEPOSIT_ACCOUNT < iban > <person_id> <bank> <branch_name> <amount> <ratio>
Deposit acccount is another account which gains money at paydays. In paydays you need to calculate interest of money in the account and you need to add that money to that account.
results:
<iban> named account created by <ratio>. <person_id> has <amount>
ERROR: <person_id> person not found

CREATE CARD <card_id> < iban> <limit>
A main card can be created on a checking account. At payday loan of the card should be decreased from that account.
results:
<card_id> card created by <limit>.
ERROR: <iban> account not found

CREATE SUBCARD <card_id> <main_card_id> <person_id> <limit>
Sub-cards can be defined on a main card for another person. At payday, loan of sub-card will be decreased from main card's account.
results:
<card_id> card created by <limit> on <main_card_id>.
ERROR: <main_card_id> card not found
ERROR:<person_id> person not found.

ADD PAYMENT <card_id> <amount>
To process this command you need to add loan to that card. You need to be carefull about limit of that card.
results:
<card_id> cards payment <amount>. Total loan <total amount>
ERROR: <card_id> card not found
ERROR:<card_id> card doesn't have enough limit for <amount>

DEPOSIT <iban> <amount>
You can add money to an account at any time.
results:
<iban> account deposits <amount>. Total money <total amount>
ERROR:<iban> account not found

WITHDRAW <iban><amount>
You can withdraw money from an account at any time. Amount of the money int the account can not be negative.
results:
<iban> account withdraws <amount>. Total money <total amount>
ERROR:<iban> account not found
ERROR: <iban> account doesn't have enough money

TRANS <source_ iban> <target_ iban> <amount>
Money transfer between checking account is available. But transfer fees are different. You need to decrease calculated amount of money from source account. Between account at same branch, transfers are for free. Same bank different branchs ratio: %2, different banks ratio: %5.
resluts:
<source_ iban> account <amount> transfered <target_ iban> <transfer_fee>
ERROR: <source_iban> account not found
ERROR: <target_iban> account not found
ERROR: <source_iban> account doesn't have enough money

PAYDAY
At paydays you have to check all acount and card for their payment and calculations. You have to check deposit accounts, main cards and sub cards for their deposits and payments.
result: (order is important, you have to write results in order, deposit accounts, cards, sub-cards. For sub orders creation time is parameter)
Payday:
<iban> account deposits <amount>. Total money <total amount>
<card_id> card payment <amount> from <iban >. Total money <total amount>
<sub_card_id> sub-card payment <amount> from <iban >. Total money <total amount>
(these lines must be repated, how many you need)

WEALTH <person_id>

A persons all wealth must be reported by this command. You have to report in order, deposit account, checking account, cards, sub-cards. For sub orders you have to use creation time.
result:
<person_id> persons wealth:
<iban> deposit account: <money amount>
<iban> checking account: <money amount>
<card> main card: <total loan>
<sub-card> sub_card on <owner_name>s account: <total loan>

You can add all your special errors by starting that line with “ERROR:” phrase.
Example:
ERROR: This sub-card related account had already belong to same person.

You need to get data fields from a data file. This data file consumed as if it is database file. You have to read field from that file and construct your project with this data. Data file will not have any error in order. You can read all fields without checking. All values will be splitted by tab character. Three type of data will be given in data file:

PERSON	<person_id>	<name>
BANK	<Bank_name>	
BRANCH	<Bank_name>	<Branch_name>

Example data file:

```
PERSON      12345678950 Ali Demir
PERSON      13245678972 Veli Celik
PERSON      98765432150 Ahmet Altin
PERSON      97865432138 Mehmet Tas
BANK  Ziraat Bankasi
BANK  Inheritance Bank
BANK  HUBBM Bank
BRANCH      Ziraat Bankasi    Beytepe Branch
BRANCH      Ziraat Bankasi    Ayranci Branch
BRANCH      Ziraat Bankasi    Kizilay Branch
BRANCH      Inheritance Bank  Bahceli Branch
BRANCH      Inheritance Bank  Kizilay Branch
BRANCH      HUBBM Bank  Beytepe HeadQuarter
```

Important Notes

- You have to design your application in complete object oriented design approach. You are expected to define a java class for everything that is suitable for a class.
- Using Inheritance is main case of this project.
- Class diagram for full program must be given in your report. You have prepare a basic class diagram so you need to do little research about UML Class Diagrams.
- Another important point is designing your code in encapsulation paradigm. Encapsulation is an obligation in OOP approach.

- You can use *your InputDelegate and OutputDelegate classes from your first experiment* for IO operations. Don't forget using ten dashes.("----------")
- Example input and output files will not be published. You will perform input files and data files to check your program.
- Transfer fee calculations and commission calculations may result in floating point numbers. Make all work on floating point numbers and don't round any number. Only when printing results to the output file you can round the number up until you have digits after dot.

SUBMISSION

- Your submission will be in the format below

< StudentID>
|-- report
report.pdf
|-- source
BankSystem.java
InputDelegate.java
OutputDelegate.java
other *.java files

- You have to use "Online Experiment Submission System".
- http://submit.cs.hacettepe.edu.tr. Other type of submissions especially by e-mail WILL NOT BE ACCEPTED.
- Submission deadline is 26/4/2011, 16.59 pm.
- Do not forget to sign the "submission paper" after you submit your experiment.
- Respect the office hours of your advisor .
  Wednesday 09:00-12:00, Friday 13:00-15:00 Office: 122

REFERENCES

- http://www.google.com
- http://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)
- http://en.wikipedia.org/wiki/Object-oriented_programming
- http://en.wikipedia.org/wiki/Encapsulation_(object-oriented_programming)
- http://web.cs.hacettepe.edu.tr/~ersiner/prj/minio/
- http://en.wikipedia.org/wiki/UML
- http://en.wikipedia.org/wiki/Class_diagram