COMS 3101 - Fall 2013
Perl Homework 4

- Due by start of class (Monday 4pm).
- See submission instructions.

1. Basic Math Module:

Write a traditional module **BasicMath.pm** with 3 functions

- **max** - returns the max of the arguments passed
- **min** - returns the min of the arguments passed
- **med** – returns the median of the arguments passed

In all three functions, the user may pass an arbitrary number of arguments to the function.

Your module should automatically import these functions into the program where it is 'use'ed. In other words, you should be able to call these functions as: 'med(@nums)' rather than 'BasicMath::med(@nums). Hint: use Exporter module.

2. Banking Modules:

Write two modules that implement some simple functionality for a 'bank':

a) The first module (**BankAccount.pm**) defines a class with three attributes: name, accountNo and balance, and methods to retrieve these attributes. It also provides methods to deposit and withdraw money as well as print all account details. The BankAccount object should be implemented using a hash reference to store the data.

Specifically, the BankAccount class should contain the following features:

Attributes:

- **name** - name of account holder
- **accountNo** - account number
- **balance** - account balance

Constructor:

- creates the hash reference to store the object data. Initialize it with 'name' of account holder and account number passed as arguments. The constructor takes arguments and should be called as: BankAccount->new("Alex",9999);

Methods:

- **getName** - retrieves the name
- **getAccountNo** - retrieves the account number
- **getBalance** - retrieves the balance
- **deposit** - amount is passed as an argument and is added to the balance
- **withdraw** - withdraw (deduct) the amount (passed as argument) from the balance
- **printAccountDetails** - prints the name of account holder, account number and balance in separate lines.

b) The second module (**Bank.pm**) defines a class with three attributes: name and accountBase, and methods to add new account, close an account and basic banking operations such as deposit, withdrawal and getBalance, given a specific account number. It should also have a method to apply monthly interests on all accounts, and a listAccounts method to list all accounts. It should also be implemented using a hash reference to store the data.

Specifically, the Bank class should contain the following features:

Attributes:

- **name** - bank's name
- **accountBase** - a reference to a hash, which is keyed by account numbers and values are objects of BankAccount class.

- **nextAccountNumber** - keeps track of account number to use for the next new account added.

Constructor:

- creates the hash reference to store the object data. Initialize it with 'name' of bank passed as an argument. The accountBase should be initialized to an empty hash reference. It shold also initialize attribute nextAccountNumber to a 4 digit value of your choice.

Methods:

- **newAccount** - creates a new account for the person whose name is passed as an argument. You will construct a new BankAccount object using the supplied name and nextAccountNumber variable, and add it to the accountBase hash with it's account number as the key. The attribute nextAccountNumber should be incremented to ensure that the next account gets a new account number. This method returns the account number of the newly created account.
- **getBalance** - given an account number, calls the getBalance function on the corresponding BankAccount object.
- **deposit** - given an account number and amount, calls the deposit function on the corresponding BankAccount object for the amount.
- **withdraw** - given an account number and amount, calls the withdraw function on the corresponding BankAccount object for the amount.
- **applyInterests** - this method should act on all BankAccounts in the bank. The rate of interest is fixed as 12% annual rate, which means, the balance should be increased by 1% every month. So, each call to this method would increase the balance of all accounts by 1%. You can use the deposit method of the account to increase the balance.
- l**istAccounts** - prints details of each account in the Bank in the decreasing order of account numbers using the printAccountDetails method of each account.

c) Test your modules using the **testbank.pl** script. Reviewing this script can also help you with writing the modules in the first place.