**code-on-work-exercises**

1. **Write down the following class and do the exercise given below**

public class Account {

private double bal; //The current balance

private int accnum; //The account number

public Account(int a){

bal=0.0;

accnum=a;

}

public void deposit(double sum){

if (sum>0)

bal+=sum;

else

System.err.println("Account.deposit(...): "

+"cannot deposit negative amount.");

}

public void withdraw(double sum){

if (sum>0)

bal-=sum;

else

System.err.println("Account.withdraw(...): "

+"cannot withdraw negative amount.");

}

public double getBalance(){

return bal;

}

public double getAccountNumber(){

return accnum;

}

@Override

public String toString(){

return "Acc " + accnum + ": " + "balance = " + bal;

}

public final void print(){

//Don't override this,

//override the toString method

System.out.println( toString() );

}

}

Look at the Account class given above and write a main method in a different class to briefly experiment with some instances of the Account class.

* Using the Account class as a base class, write two derived classes called SavingsAccount and CurrentAccount.
* A SavingsAccount object, in addition to the attributes of an Account object, should have an interest variable and a method which adds interest to the account.
* A CurrentAccount object, in addition to the attributes of an Account object, should have an overdraft limit variable.
* Ensure that you have overridden methods of the Account class as necessary in both derived classes.
* Now create a Bank class, an object of which contains an array of Account objects.
* Accounts in the array could be instances of the Account class, the SavingsAccount class, or the CurrentAccount class.
* Create some test accounts (some of each type).
* Write an update method in the bank class. It iterates through each account, updating it in the following ways: Savings accounts get interest added (via the method you already wrote); CurrentAccounts get a letter sent if they are in overdraft.
* The Bank class requires methods for opening and closing accounts, and for paying a dividend into each account.

**Hints:**

* Note that the balance of an account may only be modified through the deposit(double) and withdraw(double) methods.
* The Account class should not need to be modified at all.
* Be sure to test what you have done after each step.

1. **Write down a class Person as shown below**

public class Person {

private String name;

public Person() {

name = "No name yet.";

}

public Person(String n){

name = n;

}

public void setName(String newName){

name = newName;

}

public String getName(){

return name;

}

public void print(){

System.out.println("Name: " + name);

}

public boolean equals(Object p){

return name.equals(p.name);

}

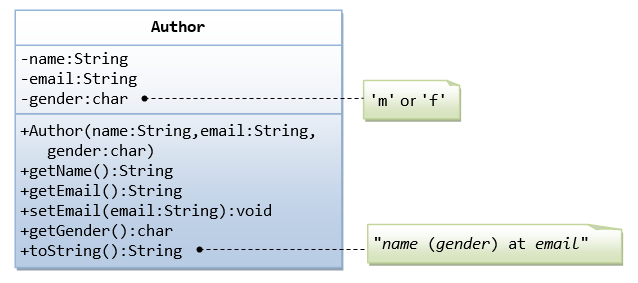
}

Create a class called Employee whose objects are records for an employee. This class will be a derived class of the class Person which you will have to copy into a file of your own and compile. An employee record has an employee's name (inherited from the class Person), an annual salary represented as a single value of type double, a year the employee started work as a single value of type int and a national insurance number, which is a value of type String.  
  
Your class should have a reasonable number of constructors and accessor methods (getters and setters), toString() method, as well as an equals method.

Write another class containing a main method to fully test your class definition.

1. **The Author Class**

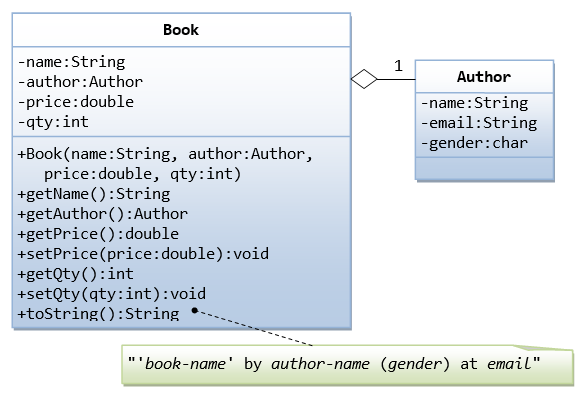
**Create a class called Author by following the picture shown below.**



A class called Author is designed as shown in the class diagram. It contains:

* Three private member variables: name (String), email (String), and gender (char of either 'm' or 'f'.
* A constructor to initialize the name, email and gender with the given values.  
  (There is no *default constructor*, as there is no default value for name, email and gender.)
* Public getters/setters: getName(), getEmail(), setEmail(), and getGender().  
  (There are no setters for name and gender, as these properties are not designed to be changed.)
* A toString() method that returns "*name* (*gender*) at *email*", e.g., "Prahaash (Male) at [prahaash@outlook.com](mailto:prahaash@outlook.com)".

**Create Book class by seeing the following picture**



**Note than now object of Author class is part of Book class.**

Let's design a Book class.

Assume that a book is written by one (and exactly one) author.

The Book class (as shown in the class diagram) contains the following members:

* Four private member variables: name (String), author (an *instance* of the Author class we have just created, assuming that each book has exactly one author), price (double), and qty (int).
* The public getters and setters:

getName(), getAuthor(), getPrice(), setPrice(), getQty(), setQty().

* A toString() that returns "'book-name' by author-name (gender) at email". You could reuse the Author's toString() method, which returns "author-name (gender) at email".

Write a test class BookTest which consists main() method to test the classes you have defined.