EBU5304 Software Engineering: Design Patterns Exercise 1

Consider the following interface type:

```
interface Account
{
 public int getAccNo();
 public String getAccName();
 public double getBalance();
 public void deposit(double amount);
 public void withdraw(double amount);
}
```

It could be implemented by the class `BankAccount` that was used in the exercise in Week 3:

```
class BankAccount implements Account
Rest of code as before
```

or by the class `BasicAccount` given in the exercise in Week 3 that could also be declared as implements `Account` to enable that.

Now write three classes that implement the `Account` interface and use one of the wrapper design patterns:

1) A class that uses the Decorator design pattern to count the number of times the method `deposit` and the method `withdraw` have been called (a separate count for each of them).

2) A class that provides an Immutable View of an `Account` object. It should wrap an `Account` object, such as a `BankAccount` object, and work in a way that means the methods `getAccNo`, `getAccName` and `getBalance` work the same, but if the methods `deposit` or `withdraw` are called they will always just throw an `UnsupportedOperationException`.

3) A class that uses the Composite design pattern to enable a list of `Account` objects to be used as a single `Account` object. It should work by:
    a) The method `getBalance` returns the sum of balances of all the accounts in the list
    b) The method `deposit` divides the amount deposited equally between all the accounts in the list
    c) The method `withdraw` withdraws the amount from the first account in the list that has a balance equal to or greater than the amount withdrawn.

   Objects of the class should have their own account name and account number which are returned by the methods `getAccName` and `getAccNo`. They should also have methods which add and remove `Account` objects from the list.