Design Pattern Exercises

EBU5304 Software Engineering 2018/19

Account Interface

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

Exercise on Wrapper Classes

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

- 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).

- 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`.

- A class that uses the <u>Composite</u> 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withdraws the amount from the first account in the list that has a balance equal to or greater than the amount withdrawn.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.

Account Counter using Decorator Pattern

```
class AccountCounter implements Account
{
 private Account myAccount;
 private int depositCount, withdrawCount;

 public AccountCounter(Account acc) {
  myAccount=acc;
 }

 public int getAccNo() {
  return myAccount.getAccNo();
 }

 public String getAccName() {
  return myAccount.getAccName();
 }

 public double getBalance() {
  return myAccount.getBalance();
 }
```

…

```
…
 public void deposit(double amount) {
  myAccount.deposit(amount);
  depositCount++;
 }

 public void withdraw(double amount) {
  myAccount.withdraw(amount);
  withdrawCount++;
 }

 public String toString(){
  return myAccount.toString();
 }
 public int getDepositCount() {
  return depositCount;
 }

 public int getWithdrawCount() {
  return withdrawCount;
 }
}
```

Immutable View of Account

```
class AccountImmutableView implements Account
{
 private Account myAccount;

 public AccountImmutableView(Account acc) {
  myAccount=acc;
 }

 public int getAccNo() {
  return myAccount.getAccNo();
 }

 public String getAccName() {
  return myAccount.getAccName();
 }

 public double getBalance() {
  return myAccount.getBalance();
 }

…
…

 public void deposit(double amount) {
   throw new UnsupportedOperationException();
 }

 public void withdraw(double amount) {
   throw new UnsupportedOperationException();
 }

 public String toString() {
  return myAccount.toString();
 }
}
```

Account Composition

```
import java.util.ArrayList;

class AccountComposition implements Account
{
 private ArrayList<Account> accounts;
 private int accNo;
 private String accName;

 public AccountComposition(Account acc, String accName, int accNo){
  accounts = new ArrayList<Account>();
  accounts.add(acc);
  this.accNo = accNo;
  this.accName = accName;
 }


…
```

…

```
public int getAccNo() {
 return accNo;
}

public String getAccName() {
 return accName;
}

public double getBalance() {
 double balance = 0.0;
 for(Account acc : accounts)
    balance+=acc.getBalance();
 return balance;
}
```

…

```
…
public void deposit(double amount) {
 amount=amount/accounts.size();
 for(Account acc : accounts)
    acc.deposit(amount);
}

 public void withdraw(double amount) {
 for(Account acc : accounts) {
    if(acc.getBalance()>=amount) {
        acc.withdraw(amount);
        return;
       }
 }
 System.out.println("Withdraw "+amount
               + " unsuccessful. No single account"
               + " with enough available funds");
}

…
```

```
 …


 public boolean addAccount(Account acc) {
  return accounts.add(acc);
 }

 public boolean removeAccount(Account acc) {
  return accounts.remove(acc);
 }

 public String toString()
 {
  double balance=getBalance();
  return "\nAccount number: " + accNo + "\n" + "Account name: "
                  + accName + " \n" + "Balance: " + balance + "\n";
 }
}
```

Account Number Comparator

```
import java.util.Comparator;

class AccNoComparer implements Comparator<BankAccount>
{
 public int compare(BankAccount acc1, BankAccount acc2)
 {
  return acc1.getAccNo()-acc2.getAccNo();
 }
}
```

Account Name Comparator

```
import java.util.Comparator;

class AccNameComparer implements Comparator<BankAccount>
{
 public int compare(BankAccount acc1, BankAccount acc2)
 {
  return acc1.getAccName().compareTo(acc2.getAccName());
 }
}
```

Account Balance Comparator

```
import java.util.Comparator;

class BalanceComparer implements Comparator<BankAccount>
{
 public int compare(BankAccount acc1, BankAccount acc2)
 {
  return (int)(acc1.getBalance()-acc2.getBalance());
 }
}
```

Account Balance Closeness Comparator

```
import java.util.Comparator;

class BalanceClosenessComparer implements Comparator<BankAccount>
{
 private double target;

 public BalanceClosenessComparer(double target)
 {
  this.target=target;
 }

 public int compare(BankAccount acc1, BankAccount acc2)
 {
  double diff1 = Math.abs(acc1.getBalance()-target);
  double diff2 = Math.abs(acc2.getBalance()-target);
  return (int)(diff1-diff2);
 }
}
```