Chapter 6: Object-Oriented Design
Lab Exercises

Topics	Lab Exercises
Method Decomposition	**A Modified MiniQuiz Class**
Overloading	**A Flexible Account Class ***
Static Variables and Methods	**Opening and Closing Accounts ***
	Counting Transactions *
	Transfering Funds*
	Electrical *
	Tax*
GUI Layouts	**Telephone Keypad**

A Flexible Account Class

File *Account.java* contains a definition for a simple bank account class with methods to withdraw, deposit, get the balance and account number, and return a String representation. Note that the constructor for this class creates a random account number. Save this class to your directory and study it to see how it works. Then modify it as follows:

1. Overload the constructor as follows:

   - public Account (double initBal, String owner, long number) – initializes the balance, owner, and account number as specified
   - public Account (double initBal, String owner) – initializes the balance and owner as specified; randomly generates the account number.
   - public Account (String owner) – initializes the owner as specified; sets the initial balance to 0 and randomly generates the account number.

2. Overload the *withdraw* method with one that also takes a fee and deducts that fee from the account.

File *TestAccount.java* contains a simple program that exercises these methods. Save it to your directory, study it to see what it does, and use it to test your modified Account class.

```
//*******************************************************
// Account.java
//
// A bank account class with methods to deposit to, withdraw from,
// change the name on, and get a String representation
// of the account.
//*******************************************************

public class Account
{
  private double balance;
  private String name;
  private long acctNum;

  //-----------------------------------------------
  //Constructor -- initializes balance, owner, and account number
  //-----------------------------------------------
  public Account(double initBal, String owner, long number)
  {
    balance = initBal;
    name = owner;
    acctNum = number;
  }

  //-----------------------------------------------
  // Checks to see if balance is sufficient for withdrawal.
  // If so, decrements balance by amount; if not, prints message.
  //-----------------------------------------------
  public void withdraw(double amount)
  {
    if (balance >= amount)
       balance -= amount;
    else
       System.out.println("Insufficient funds");
  }

  //-----------------------------------------------
  // Adds deposit amount to balance.
  //-----------------------------------------------
```

```
  public void deposit(double amount)
  {
    balance += amount;
  }

  //-----------------------------------------------
  // Returns balance.
  //-----------------------------------------------
  public double getBalance()
  {
    return balance;
  }


  //-----------------------------------------------
  // Returns a string containing the name, account number, and balance.
  //-----------------------------------------------
  public String toString()
  {
      return “Name:” + name +
            “\nAccount Number: ” + acctNum +
            “\nBalance: “ + balance;
  }
}




//*******************************************************
// TestAccount.java
//
// A simple driver to test the overloaded methods of
// the Account class.
//*******************************************************

import java.util.Scanner;

public class TestAccount
{
    public static void main(String[] args)
    {
        String name;
        double balance;
        long acctNum;
        Account acct;

        Scanner scan = new Scanner(System.in);

        System.out.println("Enter account holder's first name");
        name = scan.next();
        acct = new Account(name);
        System.out.println("Account for " + name + ":");
        System.out.println(acct);

        System.out.println("\nEnter initial balance");
        balance = scan.nextDouble();
        acct = new Account(balance,name);
        System.out.println("Account for " + name + ":");
        System.out.println(acct);
```

```
        System.out.println("\nEnter account number");
        acctNum = scan.nextLong();
        acct = new Account(balance,name,acctNum);
        System.out.println("Account for " + name + ":");
        System.out.println(acct);

        System.out.print("\nDepositing 100 into account, balance is now ");
        acct.deposit(100);
        System.out.println(acct.getBalance());
        System.out.print("\nWithdrawing $25, balance is now ");
        acct.withdraw(25);
        System.out.println(acct.getBalance());
        System.out.print("\nWithdrawing $25 with $2 fee, balance is now ");
        acct.withdraw(25,2);
        System.out.println(acct.getBalance());

        System.out.println("\nBye!");
    }
}
```

Opening and Closing Accounts*

File *Account.java* (see previous exercise) contains a definition for a simple bank account class with methods to withdraw, deposit, get the balance and account number, and return a String representation. Note that the constructor for this class creates a random account number. Save this class to your directory and study it to see how it works. Then write the following additional code:

1. Suppose the bank wants to keep track of how many accounts exist.

    a. Declare a private static integer variable numAccounts to hold this value. Like all instance and static variables, it will be initialized (to 0, since it's an int) automatically.
    b. Add code to the constructor to increment this variable every time an account is created.
    c. Add a static method *getNumAccounts* that returns the total number of accounts. Think about why this method should be static – its information is not related to any particular account.
    d. File *TestAccounts1.java* contains a simple program that creates the specified number of bank accounts then uses the getNumAccounts method to find how many accounts were created. Save it to your directory, then use it to test your modified Account class.

2. Add a method void close() to your Account class. This method should close the current account by appending "CLOSED" to the account name and setting the balance to 0. (The account number should remain unchanged.) Also decrement the total number of accounts.

3. Add a static method *Account consolidate(Account acct1, Account acct2)* to your Account class that creates a new account whose balance is the sum of the balances in acct1 and acct2 and closes acct1 and acct2. The new account should be returned. Two important rules of consolidation:

    - Only accounts with the same name can be consolidated. The new account gets the name on the old accounts but a new account number.
    - Two accounts with the same number cannot be consolidated. Otherwise this would be an easy way to double your money!

    Check these conditions before creating the new account. If either condition fails, do not create the new account or close the old ones; print a useful message and return null.

4. Write a test program that prompts for and reads in three names and creates an account with an initial balance of $100 for each. Print the three accounts, then close the first account and try to consolidate the second and third into a new account. Now print the accounts again, including the consolidated one if it was created.

```
//********************************************************
// TestAccounts1
// A simple program to test the numAccts method of the
// Account class.
//********************************************************
import java.util.Scanner;

public class TestAccounts1
{
    public static void main(String[] args)
    {
        Account testAcct;

        Scanner scan = new Scanner(System.in);
```

```
        System.out.println("How many accounts would you like to create?");
        int num = scan.nextInt();

        for (int i=1; i<=num; i++)
            {
                testAcct = new Account(100, "Name" + i);
                System.out.println("\nCreated account " + testAcct);
                System.out.println("Now there are " + Account.numAccounts() +
                                   " accounts");
            }
    }
}
```

Counting Transactions*

File *Account.java* (see **A Flexible Account Class** exercise) contains a definition for a simple bank account class with methods to withdraw, deposit, get the balance and account number, and return a String representation. Note that the constructor for this class creates a random account number. Save this class to your directory and study it to see how it works. Now modify it to keep track of the total number of deposits and withdrawals (separately) for each day, and the total amount deposited and withdrawn. Write code to do this as follows:

1. Add four private static variables to the Account class, one to keep track of each value above (number and total amount of deposits, number and total of withdrawals). Note that since these variables are static, all of the Account objects share them. This is in contrast to the instance variables that hold the balance, name, and account number; each Account has its own copy of these. Recall that numeric static and instance variables are initialized to 0 by default.

2. Add public methods to return the values of each of the variables you just added, e.g., *public static int getNumDeposits()*.

3. Modify the *withdraw* and *deposit* methods to update the appropriate static variables at each withdrawal and deposit

4. File *ProcessTransactions.java* contains a program that creates and initializes two Account objects and enters a loop that allows the user to enter transactions for either account until asking to quit. Modify this program as follows:
   - After the loop, print the total number of deposits and withdrawals and the total amount of each. You will need to use the Account methods that you wrote above. Test your program.
   - Imagine that this loop contains the transactions for a single day. Embed it in a loop that allows the transactions to be recorded and counted for many days. At the beginning of each day print the summary for each account, then have the user enter the transactions for the day. When all of the transactions have been entered, print the total numbers and amounts (as above), then reset these values to 0 and repeat for the next day. Note that you will need to add methods to reset the variables holding the numbers and amounts of withdrawals and deposits to the Account class. Think: should these be static or instance methods?

```
//*******************************************************
// ProcessTransactions.java
//
// A class to process deposits and withdrawals for two bank
// accounts for a single day.
//*******************************************************
import java.util.Scanner;

public class ProcessTransactions
{
    public static void main(String[] args){

        Account acct1, acct2;          //two test accounts
        String keepGoing = "y";        //more transactions?
        String action;                 //deposit or withdraw
        double amount;                 //how much to deposit or withdraw
        long acctNumber;               //which account to access

        Scanner scan = new Scanner(System.in);

        //Create two accounts
        acct1 = new Account(1000, "Sue", 123);
        acct2 = new Account(1000, "Joe", 456);

        System.out.println("The following accounts are available:\n");
        acct1.printSummary();

        System.out.println();
        acct2.printSummary();
```

```
        while (keepGoing.equals("y") || keepGoing.equals("y"))
            {
                //get account number, what to do, and amount
                System.out.print("\nEnter the number of the account you would like
to access: ");
                acctNumber = scan.nextLong();
                System.out.print("Would you like to make a deposit (D) or withdrawal
(W)? ");
                action = scan.next();
                System.out.print("Enter the amount: ");
                amount = scan.nextDouble();

                if (amount > 0)
                    if (acctNumber == acct1.getAcctNumber())
                        if (action.equals("w") || action.equals("W"))
                            acct1.withdraw(amount);
                        else if (action.equals("d") || action.equals("D"))
                            acct1.deposit(amount);
                        else
                            System.out.println("Sorry, invalid action.");
                    else if (acctNumber == acct2.getAcctNumber())
                        if (action.equals("w") || action.equals("W"))
                            acct1.withdraw(amount);
                        else if (action.equals("d") || action.equals("D"))
                            acct1.deposit(amount);
                        else
                            System.out.println("Sorry, invalid action.");
                    else
                        System.out.println("Sorry, invalid account number.");
                else
                        System.out.println("Sorry, amount must be > 0.");


                System.out.print("\nMore transactions? (y/n)");
                keepGoing = scan.next();
            }


        //Print number of deposits
        //Print number of withdrawals
        //Print total amount of deposits
        //Print total amount of withdrawals


    }
}
```

Transfering Funds*

File *Account.java* (see **A Flexible Account Class** exercise) contains a definition for a simple bank account class with methods to withdraw, deposit, get the balance and account number, and print a summary. Save it to your directory and study it to see how it works. Then write the following additional code:

1. Add a method *public void transfer(Account acct, double amount)* to the Account class that allows the user to transfer funds from one bank account to another. If *acct1* and *acct2* are Account objects, then the call *acct1.transfer(acct2,957.80)* should transfer $957.80 from acct1 to acct2. Be sure to clearly document which way the transfer goes!

2. Write a class TransferTest with a main method that creates two bank account objects and enters a loop that does the following:
   - Asks if the user would like to transfer from account1 to account2, transfer from account2 to account1, or quit.
   - If a transfer is chosen, asks the amount of the transfer, carries out the operation, and prints the new balance for each account.
   - Repeats until the user asks to quit, then prints a summary for each account.

3. Add a static method to the Account class that lets the user transfer money between two accounts without going through either account. You can (and should) call the method transfer just like the other one – you are overloading this method. Your new method should take two Account objects and an amount and transfer the amount from the first account to the second account. The signature will look like this:

   public static void transfer(Account acct1, Account acct2, double amount)

   Modify your TransferTest class to use the static transfer instead of the instance version.

Electrical* (Obligatorisk för PÖDET, friviligt för andra)

Create a class that will bundle together several static methods for electrical computation. This class should not have a constructor.
Attributes :
precision – the number of digits to report results with (1 is minimum).

Methods:
computeVoltage (current, resistance) –a static method that returns the voltage given the current and resistance. The result should be rounded to the given number of digits.

Voltage= current * resistance

computeCurrent (voltage, resistance)- same as in the method before but return the current
computeResistance (voltage, current)- same as in the method before but return the resistance
serialResistance (r1, r2)- –a static method that returns the total resistance of the two seroal resistors with the resistance r1, r2 . The total resistance is given by formula r1+r2.
parallelResistance (r1, r2) –a static method that returns the total resistance of the two parallel resistors with the resistance r1, r2 . The total resistance is given by formula 1/ 1/r1+ 1/r2 .

change Precision (new Precision)- a static method that change the number of digits reported for results.

roundTo(value) – a static value that returns the given value rounded to the appropriate number of digits. So if the value is 125.67 and the precision 2 the method will return 130.

Tax* (Obligatoriskt för AP, friviligt för andra)

Create a class that will bundle together several static methods for tax computation. This class should not have a constructor.
Attributes :
basicRate – the basic tax rate as static double variable that starts at 4 percent
luxuryRate- the luxury tax rate as static double variable that starts at 10 percent

Methods:
computeCostBasic(price)- static method that returns the given price plus the basic tax rounded to the nearest penny (öre om du vill).

computeCostLuxury(price)- static method that returns the given price plus the luxury tax rounded to the nearest penny (öre om du vill).

changeBasicRateTo(newRate)- static method that change the basic tax rate

changeLuxuryRateTo(newRate)- static method that change the luxury tax rate

roundToNearestPenny (price)- static method that returns the given price rounded to the nereast penny (öre). For example if the price is 12.567 the method will return 12.56.

MyMethods (Vanliga tenta uppgifter)

Define the methods below in a class called MyMethods. To see if the methods are working well write a driver class called TestMethods and test all the methods.

1. Write a static method called **average** that accepts two integers parameters and returns their average.
2. Overload the average method such that if three integers are provided as parameters, the method returns the average of all three.
3. Write a static method called **multiConcat** that takes a String and an integer as parameters. Return a String that consists of the string parameter concatenated with itself ***cont*** times, where cont is the integerparameter. For exempel if the parameter value is “hi” and 4 , the return value is “hihihihi”. Return the string if the integer value is less than 2.
4. Write a static method called **reverse** that takes a String as parameter and returns a String which is the same string but in revers order. For exempel if the String is “hej” the return value will be “jeh”.
5. Write a static method called **isPalindrome** that takes a String as parameter and returns true if the String is a palindrome and false if not. For exempel the String “anna” is a palindrome

because the string will be the same in revers order. (Reding from the right to the left). Use the method **revers** already defined in exercises 4

Telephone Keypad

Files *Telephone.java* and *TelephonePanel.java* contain the skeleton for a program to lay out a GUI that looks like telephone keypad with a title that says "Your Telephone!!". Save these files to your directory. Telephone.java is complete, but TelephonePanel.java is not.

1. Using the comments as a guide, add code to TelephonePanel.java to create the GUI. Some things to consider:
    a. TelephonePanel (the current object, which is a JPanel) should get a BorderLayout to make it easy to separate the title from the keypad. The title will go in the north area and the keypad will go in the center area. The other areas will be unused.
    b. You can create a JLabel containing the title and add it directly to the north section of the TelephonePanel. However, to put the keypad in the center you will first need to create a new JPanel and add the keys (each a button) to it, then add it to the center of the TelephonePanel. This new panel should have a 4x3 GridLayout.
    c. Your keypad should hold buttons containing 1 2 3, 4 5 6, 7 8 9, * 0 # in the four rows respectively. So you'll create a total of 12 buttons.

2. Compile and run Telephone.java. You should get a small keypad and title. Grow the window (just drag the corner) and see how the GUI changes – everything grows proportionately.

3. Note that the title is not centered, but it would look nicer if it were. One way to do this is to create a new JPanel , add the title label to it, then add the new JPanel to the north area of the TelephonePanel (instead of adding the label directly). This works because the default layout for a JPanel is a centered FlowLayout, and the JPanel itself will expand to fill the whole north area. Modify your program in this way so that the title is centered.

```
//********************************************************
// Telephone.java
//
// Uses the TelephonePanel class to create a (functionless) GUI
// like a telephone keypad with a title.
// Illustrates use of BorderLayout and GridLayout.
//********************************************************
import javax.swing.*;
public class Telephone
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame("Telephone");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new TelephonePanel());
        frame.pack();
        frame.setVisible(true);
    }
}
```

```
//*****************************************************
// TelephonePanel.java
//
// Lays out a (functionless) GUI like a telephone keypad with a title.
// Illustrates use of BorderLayout and GridLayout.
//*****************************************************
import java.awt.*;
import javax.swing.*;

public class TelephonePanel extends JPanel
{
    public TelephonePanel()
    {
        //set BorderLayout for this panel

        //create a JLabel with "Your Telephone" title

        //add title label to north of this panel

        //create panel to hold keypad and give it a 4x3 GridLayout

        //add buttons representing keys to key panel

        //add key panel to center of this panel
    }
}
```