CS 116

Lab # 1

This lab is worth 2 points towards your final grade.

Work on it in class and ask questions if you need help. This Lab assignment should be submitted by Wednesday 2/05 by 10:00 p.m. on Blackboard under assignment "Lab1". Follow the submission instructions listed below. Please make sure that you click the "Submit" button and not just "Save".

If you require an extension you must send an email to me in advance for approval. Indicate the reason. Extensions are limited to two days maximum if approved.

Objectives:

The list below indicates the Java concepts needed for this exercise.

1. Calling static and non static methods from a static method.
2. Reading from a File.
3. Using while loops.
4. Ability to create simple logical conditions.
5. Develop the logical algorithm that will accomplish the required task.
6. Using StringTokenizer.
7. Formatting Numbers.
8. Using Wrapper classes to parse Strings.
9. Scope of variables.
10. Adding data in an array.
11. Packaging a class!!
12. Adding help methods in your classes.
13. Arrays of Objects.
14. Using methods of the String class

THEORY ON Calling static and non static methods from a static method.

We have indicated in lectures that a static method of a class can be invoked in another class simply by using the name of the class. A good example of that are the methods of the library class Math which are static. Thus a Math class method can be invoked as in the example below:

Double d = Math.pow(a,b); where the double number a is raised to the b power and the result is returned as a double stored in identifier d.

What if we wanted to call static or non static methods which are members of the same class from within another member method of the same class which happens to be static? A good example of that is the main method in a Java program which is always declared to be static.

In Java a program interpretation (execution) always begins with the main method. As lines of code are interpreted in the main method we could make a call to another method in which case execution switches to the code of the method that we called and then returns to the main method at the point the call was made.

For example let us assume the following code for main:

```
public class Adding {
        public static void main(String[] args)
        {
                int a=0;
                int b=0;
                int c=0;
                c=sum(a,b);
                System.out.println("The sum="+" "+c);
        }

        public  static int sum(int x, int y)
        {
                int d=x+y;
                return d;
        }
}
```

In the main method we make a direct call to the method sum and we pass it the two integers. The method sum adds them up and returns their sum. The reason we can make this direct call from the main method with the line of code: c=sum(a,b); is because the method sum has been declared to be also static just like the main method.

Rule: If two methods belong to the same class and one method is static then we can make a simple method call, without using an object, to the second method from the first static method, as long as the second method is also static.

But what if we had a static method (like the main method) and we wanted to call another method within the same class which is not static? Then we need to create an object of the class, that both methods belong to, and make an invocation to the non static method using the object (just like when we invoke a method from another class). To demonstrate this technique let us change the code slightly in the above example, so that the method sum is not static:

```
public class Adding {
        public static void main(String[] args)
        {
                int a=0;
                int b=0;
                int c=0;
                Adding add=new Adding();
                c=add.sum(a,b);
                System.out.println("The sum="+" "+c);
        }

        public  int sum(int x, int y)
        {
                int d=x+y;
                return d;
```

```
        }
    }
```

What is different now? First, the method sum does not have the keyword static in its signature. Second, in order to call it from within main, we have to create an object of class Adding (the class that both the main and sum methods belong to) and use it to invoke sum using the dot operator:

```
Adding add=new Adding();
c=add.sum(a,b);
```

Third, we notice that we called the default constructor of class Adding after the new operator, **but we did not explicitly create a default constructor in class Adding.** We are allowed to do that because Java creates a default constructor automatically for us implicitly if there is not a constructor coded in the class. This default constructor is actually empty and does not initialize any variables. It does allow us, however, to create objects whose instance variables have the initial values as declared in the class.

THEORY: OPENING A TEXT FILE FOR READING

When a file is open for reading (or writing) certain tasks can fail. It is the responsibility of the Operating System to provide the resources to open a file. Things can go wrong in the process i.e maybe the file name is misspelled or maybe the file is not in the specified directory path or maybe the file is corrupted. In order to avoid the program from crashing and allow a graceful exit, Java has error handling capabilities via special keywords. In this case the keywords are *try* and *catch*. You will write the code to call for the opening of the file within a try block. The catch block is used to provide a message to the user of the program, in case something went wrong.

```
try
{
        File myfileobject=new File("MyFile.txt");
        Scanner scan=new Scanner(myfileobject);
        ……..
        Loop to read data from file is needed!
        If the scanner is reading one line of data at a time, then the String captured during each loop iteration, represents
        data that may need to be broken down to the individual tokens within the line of data. Remember that the
        scanner class can return the entire line of data as a String by using:

        while(sacn.hasNextLine()) {
        String line = scan.nextLine();
        ……
        For example the String line may have the data : 23, 100, 14, 56, 234, 67
        Each number is a token that needs to be accessed by using the StringTokenizer library class. In creating a
        StringTokenizer object we have to identify the symbol that is used in the data to separate the tokens (the coma in
        our example):
        StringTokenizer strtk= new StringTokenozer(line, ",");
        Notice that the symbol coma (called the delimiter) is shown as a String data type in the constructor of the
        StringTokenizer class.
        To get the first token:
        String   st1=strtk.nextToken();
        int   tok1=Integer.parseInt(st1);
        The identifier tok1 now holds the int value 23 from our example. Every time we invoke nextToken() we get the next
        token. See the API for more methods in the StringTokenizer class that wiil allow you the sense the end of the String
```

that is being tokenized (the String line in our example). Also, sometimes the tokens need to be saved in an array if the values are not to be used immediately or need to be used outside the loop, somewhere else in the program!

}//end of while loop

```
}//end of try block
catch(IOException ioe)
{
        System.out.printl("Something went wrong with the opening of the file);
}
```

The only time that catch is executed is when the code inside the try block fails.

NOTE 1: If arrays are to be used inside the while loop described above, then the size of the array needs to be declared in advanced before we enter the loop that reads the data. We do that by creating another while loop with the scanner object and by just counting the iterations of the loop. That will be the number of lines read as an example, if we are reading data one line at a time. Therefore BEFORE THE LOOP THAT READS THE DATA as described above create a loop:

```
Int count=0;
while(scan.hasNextLine)
{
  scan.nextLine();
 count++;
}
```

The value held by count is the size that we want to use for the array, assuming that we want to store each lien as a String:

```
String  [] myarray=new String[count];
```

This technique can vary depending on the problem as to what data type the array (or arrays) should be.

Note 2: Since the scanner object was used in one loop, the same scanner object can not be used in the second while loop. A new Scanner object can be created (or the previous one can be re set –see API).

THEORY: PACKAGING CLASSES.

Packaging a class means that the compiler will save the bytecodes file (the .class file) in a specified directory path with respect to the current directory where the source code file is located. Keep in mind that the directory path will be created by the compiler!! You do not create the folders of that path manually.

Packaging is used by the library classes and all you have to do is to open the API and take a look at some of the classes. For instance the Scanner class is located in the path java.util, and in order to use it we have to write an import statement at the top of the class like :

import java.util.Scanner; (or import java.util.* if we want all the classes from the package java.util). Therefore, the path java.util is called the package that holds a number of other library classes besides the Scanner class (i.e the StringTokenizer class).

The reason for packaging is that we can isolate classes by their functionality and thus be able to sort out a complex program that has many classes, according to the functionality of those classes.

We can do the same with user defined classes. We can decide in advance on the structure of the packages and where each class in our program should be located. In order to implement packaging we have to do the following:

1. At the top of the class and above any import statements indicate the package that we want the compiler to create by using the keyword *package* followed by the path to be created by the compiler i.e.
   package firstFolder.secondFolder;
   Where in this example we want the compiler to create the path firstFolder/secondFolder with respect to the folder where the source code file is located
2. Finish the code for the class as normal after the above statement. In order to compile the class we have to use a command window and use the proper command. Remember that clicking on the menu bar of the EditPlus text editor will NOT generate the package. It has to be done command line. From a command window in the same path as the source code file type the command:

   C:/MyLABS> javac -d . MyProgram.java

   Where MyLABS in the example, is the folder where the source code file MyProgram.java is located (otherwise known as the current directory). The command option -d . (notice the dot!) means that the compiler should create the package path with respect to the current directory (in other words inside the folder MyLABS in the example. Folder MyLABS is considered the current directory because that is where the source code file is located.
3. Press enter and check to see if the compiler created the folder named firstFolder, inside MyLABS folder. Then click to open folder firstFolder and see if the compiler created the folder secondFolder inside it. Then click the folder secondFolder and check to see if the compiler placed the file MyProgram.class inside it. REMEMBER THAT IT IS THE COMPILER THAT CREATES THE FOLDERS AND NOT YOU!!!!
4. If all the compiled files are there, then the question is how we run a program that has been packaged. Suppose that in the previous example the file MyProgram.java is the file with the main method. Then in order to run the program (remember than you can NOT run it from EditPlus menu bar) we have to use a command window in the path of the current directory (where the source code file is located NOT where the .class file is located!!!!):

   C:/MyLABS> java firstFolder.secondFolder.MyProgram

   Notice that the entire path to the class (firstFolser.secondFolder) with the main method has to be typed in front of the name of the class, where each folder is separated from the next by a dot.

THEORY: FORMATTING NUMBERS:

Number scan be formatted:

1. To restrict the number of decimal places in the accuracy of a double number. Use DecimalFormat library class from package java.text. Use format method of the class. Returns a String version of the number. It can be parsed back to a numerical data type.

2. To convert a number to dollars, in other words a number with 2 decimal places and the $ sign in front of the number. Use NumberFormat class from library package java.text. Use getCurrencyInstance to obtain an object and use the object to invoke format method. Return value is a String.

Please see the text or the lecture notes Lecture2.ppt and look up API.

PROGRAMMING TASK

Please read all steps carefully first, and then start coding. You can use the solutions to practice exercises and any other help including lecture presentations and your text book. The current directory where your source code files are located should be a folder named LAB1.

1. Someone is keeping track of his/her weekly expenses on a text file. The text file **Expenses.txt** contains the expenses recorded by that person and it is given to you as part of the exercise. If you open the text file you will see that there are 4 types of expenses being tracked on monthly basis:
   School, Restaurant, Clothing, Other
   The month of each group of expenses is recorded in the file.
   The following is a sample of how data is recorded in the text file on per month basis:

   Month:January
   Expense:Scool
   45.34,56.76,23.67,89.00,67.00,46.34,100.50,12.23,12.4,56.78,23.45,78.00
   Expense:Restaurant
   34.23,12.56,2.57,45.00,67.00,1.34,34.89,56.00,65.00,27.46,3.23,6.78,7.80
   Expense:Clothing
   45.00,25.00,34.67,14.23,34.56,36.00,56.85,16.00,15.00,30.23,4.34,123.78
   Expense:Other
   34.00,78.00,100.30,120.23,167.56,5.90,1.23,23.45,69.78,100.00,90.23,105.00,200.00,126.68

   In the first line after the word Month and the colon is the name of the month that the expenses occurred.
   The second line is the type of expense after the word Expense and after the colon.
   The third line has all the recorded receipts for the type of expense separated by coma on per day basis (i.e there were 12 days in the month of January where school expenses occurred in that line).
   The fourth line is the next expenses type for the same month and so on.

   The above pattern gets repeated for the next month and so on. The text file has data for 3 months but additional monthly data can be entered. Therefore, your program can not assume that there only 3 months of data. It should work for any amount of data in the text file.
   The same applies to per day records of each expense, the number of entries is not always the same from month to month for the same type of expense.

Also, you can't assume that the order of the expenses is always the same in the file. For instance, on the month of April, there were no Clothing expenses (see text file Expenses.txt), or in another month the Clothing could come first before the School expenses as an example.

2. **Create a <u>service class called Expenses that is packaged in package named: ServiceClasses (0.8 points)</u>.**
   The class represents objects whose attributes are :
   - a String data type representing the month of the expenses,
   - a String representing the type of expenses (School etc),
   - an array of double data types that holds the expenses in each day for type of expenses (i.e from the data
     Expense:School
     45.34,56.76,23.67,89.00,67.00,46.34,100.50,12.23,12.4,56.78,23.45,78.00
     the type of expenses value is School and the values to be held in the array are the double numbers. The size of the array is not fixed in this class!!!! It ha sto be determined when the client class reads the data from the text file).
   - an int data type for the number of days (which for the example line of data given above it will hold the value 12)., in other words the size of the double array above,
   - In addition it has two attributes one static and the other non static for tracking ids of Expenses objects created.

   The default constructor can create a default Expenses object with values:
   "any month", "any expense", a null array, 0, and it advances the id of the object by 1 when used.
   The non default constructor can create an Expense object with given values for the 4 attributes in thelist of arguments. NOTICE THAT THE ID ATTRIBUTES ARE NEVER PASSED AS ARGUMENTS. Both constructors can advance the id of an object.
   There are accessor and mutator methods as well for each attribute.
   There is also an equals method that returns true if the two expenses objects being compared have equal type of expense name and equal number of days in that expense .
   There is a toString method that returns a String that has the value of each attribute including the static id and the non static currentID as well as the name of the attribute in front of its value.
   Compile this class using the proper DOS command in order for the compiler to create the proper package:

   >javac -d . Expenses.java

3. **Create a client class called ExpensesClient packaged in package: ServiceClasses.ClientClass (1.2 points). Remember that since this class and the service class are in two different folders and since this client class is going to use the service class Expenses, the service class needs to be imported in the client class! i.e. import ServiceClasses.Expenses;**
   This class has two methods: a main method and another method. Let us describe first the functionality of the method totalExpenses PerMonth:
   a. Method **totalExpensesPerMonth <u>is not static</u>** and accepts an array of Expenses objects data type as argument. It returns an array of Strings. Each String in the array is the formatted String data type that represents the total of all types of expenses for each month. The number calculated is formatted to represent US dollars with the $ sign included. The number in the String is preceded by the phrase "The total of all expenses for the month of"+ ….. +"is : "+$.....

where the dots represent the name of the month and the number after the dollar sign. That String is entered into an array of Strings to be returned by the method.

public String[] totalExpensesPerMonth(Expenses[] exp)

See the sample output of the program given at the end of this document for additional clarification

4. In the main method of the class do the following:

   Open the file Expenses.txt for reading and read the lines of data according to the need for each attribute of the Expense. Your code should determine the size of needed array from the data Create Expenses objects and save them into an array of Expenses type.

   OUTPUT # 1

   Display the values of the array contents (the values of the attributes of the Expenses objects using the toString method)

   OUTPUT #2

   Make a call to the the help method totalExpensesPerMonth and capture the returned value. Display the contents of the captured returned array.

5. Compile using a DOS window (not from EditPlus) and the special command:
   >javac -d . ExpensesClient.java
6. Interpret using the special command:
   >java nameofpackage.Expenses
   where nameofpackage is the path to the client class.

SAMPLE OUTPUT OF THE PROGRAM:

C:\CS116\SPRING2014\Labs\Lab1>java serviceClasses.ClientClass.ExpensesClient

OUTPUT # 1

The month is: January The type of expenses is School The amounts are 45.34 56.76 23.67 89.0 67.0 46.34 100.5 12.23 12.4 56.78 23.45 78.0 The number of days for data is 12 The expense ID is 1 And the static id value is 15
The month is: January The type of expenses is Restaurant The amounts are 34.23 12.56 2.57 45.0 67.0 1.34 34.89 56.0 65.0 27.46 3.23 6.78 7.8 The number of days for data is 13 The expense ID is 2 And the static id value is 15
The month is: January The type of expenses is Clothing The amounts are 45.0 25.0 34.67 14.23 34.56 36.0 56.85 16.0 15.0 30.23 4.34 123.78 The number of days for data is 12 The expense ID is 3 And the static id value is 15
The month is: January The type of expenses is Other The amounts are 34.0 78.0 100.3 120.23 167.56 5.9 1.23 23.45 69.78 100.0 90.23 105.0 200.0 126.68 The number of days for data is 14 The expense ID is 4 And the static id value is 15
The month is: February The type of expenses is School The amounts are 5.24 56.76 23.61 89.0 67.5 26.34 10.5 12.2 12.45 6.78 23.0 8.0 The number of days for data is 12 The expense ID is 5 And the static id value is 15

The month is: February The type of expenses is Restaurant The amounts are 14.23 10.06 6.57 23.0 7.5 1.84 34.0 56.87 25.0 27.46 12.23 16.78 17.8 The number of days for data is 13 The expense ID is 6 And the static id value is 15

The month is: February The type of expenses is Clothing The amounts are 13.0 5.0 34.67 14.23 34.0 96.8 16.85 19.0 15.83 38.23 9.34 The number of days for data is 11 The expense ID is 7 And the static id value is 15

The month is: February The type of expenses is Other The amounts are 4.0 78.8 70.3 130.13 117.56 25.9 12.23 13.45 59.78 40.5 100.23 185.0 120.0 186.68 The number of days for data is 14 The expense ID is 8 And the static id value is 15

The month is: March The type of expenses is School The amounts are 15.24 6.16 23.81 89.0 67.5 26.34 10.5 12.25 18.45 6.28 29.0 28.0 The number of days for data is 12 The expense ID is 9 And the static id value is 15

The month is: March The type of expenses is Restaurant The amounts are 34.23 12.06 6.1 23.0 7.2 1.44 24.0 16.87 95.0 29.46 18.23 10.78 10.8 2.47 12.45 The number of days for data is 15 The expense ID is 10 And the static id value is 15

The month is: March The type of expenses is Clothing The amounts are 13.0 15.0 24.87 14.23 34.5 16.8 16.95 39.0 15.83 38.23 92.34 The number of days for data is 11 The expense ID is 11 And the static id value is 15

The month is: March The type of expenses is Other The amounts are 4.0 70.8 70.3 30.13 117.56 125.9 10.43 3.45 59.78 40.5 10.23 185.0 120.0 196.68 3.45 The number of days for data is 15 The expense ID is 12 And the static id value is 15

The month is: April The type of expenses is School The amounts are 19.24 6.16 23.81 89.0 67.5 26.34 10.5 12.25 18.45 6.28 29.0 28.0 45.67 2.54 The number of days for data is 14 The expense ID is 13 And the static id value is 15

The month is: April The type of expenses is Restaurant The amounts are 34.23 12.06 6.1 23.0 7.2 1.44 24.0 16.87 95.0 29.46 18.23 10.78 10.8 2.47 12.45 21.3 The number of days for data is 16 The expense ID is 14 And the static id value is 15

The month is: April The type of expenses is Other The amounts are 9.0 70.8 70.3 30.13 107.56 125.95 10.43 8.45 59.78 40.5 10.23 185.0 120.0 296.68 3.45 The number of days for data is 15 The expense ID is 15 And the static id value is 15

OUTPUT # 2

The total of all the expenses for the month of January is: $2,633.35

The total of all the expenses for the month of February is: $2,036.23

The total of all the expenses for the month of March is: $2,005.58

The total of all the expenses for the month of April is: $1,858.39

Submission instructions

- In your submission you must include

    a. The source code files and the compiled files for the program.

- Zip all files and name the zip file using your first name followed by your last name followed by the name of the assignment

    i.e. George_Kay_Lab1.zip or John_Smith_Lab1.zip etc.

- Upload the file on assignment folder: Lab1 on Blackboard.