CS 1113
LAB #5: Nested loops and Methods
Dr. Cline

Lab Goals

1) To show you how to use "methods" to refactor repeated code
2) To give you practice writing nested loops
3) Te present the Object Oriented concept of information hiding

Motivation

In this lab, you will explore the use of methods and nested loops. You have seen methods before. Methods are a convenient way to organize and encapsulate code, so that the complexity hidden inside the method pushed away from the programmer (unless he or she wants to look at it). Methods also permit code reuse. Nested loops are loops inside of loops. These are extremely useful structures. A large number of algorithms use nested loops to get their work done.

Refactoring repeated code into Methods

Often when programming, you come across situations in which you write almost the same block code over and over. Usually, this is a sign that the repetetive code should be "refactored" in a method, and that you should call the method instead of repeating the code. Consider the following program that repeatedly computes and computes the distance (http://en.wikipedia.org/wiki/Distance) between points that are stored as arrays of doubles. (You have already seen String arrays before, in the args array.)

```
// Lab 5a
// <Your name>
// <Your section>

import java.util.*;

public class Lab5a
{
      public static void main(String args[])
      {
            double[] a = {1, 0, 0};
            double[] b = {0, 1, 1};
            double[] c = {1, 1, 1};
            double[] d = {0, 0, 1};

            double ab = Math.sqrt(
                  (a[0]-b[0])*(a[0]-b[0]) +
                  (a[1]-b[1])*(a[1]-b[1]) +
                  (a[2]-b[2])*(a[2]-b[2]) );

            double ac = Math.sqrt(
                  (a[0]-c[0])*(a[0]-c[0]) +
                  (a[1]-c[1])*(a[1]-c[1]) +
                  (a[2]-c[2])*(a[2]-c[2]) );

            double ad = Math.sqrt(
                  (a[0]-d[0])*(a[0]-d[0]) +
                  (a[1]-d[1])*(a[1]-d[1]) +
                  (a[2]-d[2])*(a[2]-d[2]) );
```

```
            System.out.println("ab=" + ab + ", ac=" + ac + ", ad=" + ad);
        }
    }
```

Part 1) First compile and run Lab5a.java. Make sure to put your name and section number in the comment at the top.

Part 2) Next, copy the class to Lab5b.java, and replace the individual distance calculations by calls to a single public static method that computes the distance between two points passed in as parameters. Also, implement the method. Make sure that your modified program compiles and runs, and gives the same output as before. Also, make sure that your new program follows the class coding standards, and that your name is in the comment at the top.

Nested Loops

A "nested loop" is just a loop inside another loop. Nested loops allow you to iterate over high-dimensinal structures, such as 2D grid positions, or lists of lists. Answer the numbered questions in a file called **lab5answers.txt**.

Q1) What does the following nested loop print out? Write the answer in lab5answers.txt.

```
for (int r=1; r<=5; r++)
{
      for (int c=1; c<=5; c++)
      {
            System.out.printf("%4d", (r*c));
      }
      System.out.println();
}
```

Programming with nested loops

In this exercise you will finish a nested loop that reads lines of numbers from a file (using redirection) and computes and prints out the sum, product, and average of each line.

Part 3) Finish the program below that uses a nested loop to compute the sum, product, and average of rows of numbers given in input. The program will quit when the user inputs something that is not a number.

```
// Lab 5c
// <Your name>
// <Your section>

import java.util.*;

public class Lab5c
{
      public static void main(String args[])
      {
            // Scan the input
            Scanner scan = // [Your code here]

            // Process each line separately
            // If the next token is a double, assume there is an input line
```

```
            while (scan.hasNextDouble())
            {
                  // Get a line from the input
                  String line = // [Your code here]

                  // Create a scanner for the line of input you have read
                  Scanner lineScan = // [Your code here]

                  // Write a while loop that will sum all of the
                  // double values on the line (hint: create double
                  // variables called "sum", "product", and "count".
                  // Set sum and count to 0, and product to 1.
                  // Then read as many double values
                  // as you can using the while loop  When you read
                  // a number, add it to sum, multiply product by it,
                  // and add 1 to count.
                  // [Your code here]

                  // Compute the average by dividing sum by count
                  // [Your code here]

                  // Print out the sum, product, and average on one line
                  // [Your code here]
            }
      }
}
```

Part 4) Once you have completed the program, run it on the input file below:

```
1 2 3 4 5 6
4 5 6
7.5 8.5 9
8.1 9.2 10.3
```

Part 5) Make sure that your programs follows the class coding standards, and that it has your name at the top.

Putting the inner loop of a nested loop into a method

While the Lab5c.java is not as long as some of the other programs we have written, the main method is still long enough to benefit from putting part of the functionality in a method.

Part 6) Copy Lab5c.java to Lab5d.java.

Part 7) Rewrite Lab5d.java, moving the functionality to calculate and print the sum, average, and product of a line of values to a method called "processLine". Once you are done, re-run the program to make sure it still works. Use the prototype below:

```
public static void processLine(String line)
```

Notice that now, the nested loop in the original program is replaced by a single loop that calls a method (and the method has a loop inside it). This kind of transformation is one of the main ways programmers manage the mental complexity of writing large programs—they encapsulate bits of that complexity in different methods. When you call a method, you are not forced to think about its complicated internals; you only have to think about what goes in, and what comes out.

Part 8) Run your program on the input file above. Use redirection to send the output to the file "lab5out.txt".

Part 9) Make sure that Lab5c.java follows the coding standards of the class, and that your name is at the top.

Information Hiding in Object Oriented Programming

One of the most important skills in programming is managing the complexity of the code so that it can be understood. We have just seen that one way to manage this complexity is to use methods. Using classes is another way.

In a previous lab, you saw how to create a create a class to bundle together a number of variables and methods. These can either be associated with the class as a whole (static variables), or with instances of the class (non-static variables). Such encapsulation can be an enormous help, but often, we don't want code outside of a class to be able to directly manipulate data inside the class, or even see all of the data. Think about how a car works. Most of you can probably drive cars, but to do so, you don't need to know how your engine and drive train work. There is instead a small interface that you use to control the vehicle. It consists of a steering wheel, two or three pedals, an ignition switch, and a few other controls. This kind of "information hiding" can be extraordinarily useful because it frees your mind to concentrate on more important things.

The information hiding, or encapsulation, idea is fundamental to object oriented programming. To facilitate it, Java allows you to put access modifiers (public, protected, private) on class-level and instance variables and methods. Local variables and parameters cannot have access modifiers. A private variable can only be directly accessed by the code of a class. Protected variables may be accessed by the class and its subclasses. Public variables may be accessed by any code. Typically, variables are made private or protected unless there is a need to make them public. Public methods are then provided to manipulate or get data out of the class. Public methods that change properties in the class are called setters (or mutators), and methods that get property values are called getters (or accessors). Typically, setter method names start with the word set, such as "setName". Getter method names usually start with the word get, such as "getBalance". (See section 5.2 and 5.3 of the book for more info.)

Q2) What is a "getter" method? Write the answer in lab5answers.txt.

Q3) What is a "setter" method?

Q4) Are instance variables typically made private or public?

Q5) Can local variables have access modifiers?

Q6) What code can access the following variables?

a) `public static int myClassVar;`
b) `private int myInstanceVar;`
c) `protected String myString;`

Consider once again the idea of a BankAccount class. Suppose that a savings account bears interest, but that the interest rate is determined by the account balance as follows: if the balance is below $1000.00, then the interest rate is some low value. If the balance is greater than or equal to $1000.00, then the interest rate is some higher value. In this case, there is no variable that stores the interest rate. Instead, it is calculated as a function of the account balance. We can write a method that will get this property, however.

Part 10) Create a .java file called SavingsAccount.java, and complete it, doing the following.

1) Create a public class called SavingsAccount
2) Put your name and section number in a block comment at the top

3) Include a constructor that takes as parameters the owner's name, and an initial account balance.
4) Include **private** non-static variables for the name of the account holder and he account balance.
5) Include **private static** variables for the low and high interest rates.
6) Write **public** getter and setter methods for the account holder name and account balance.
7) Write a **public** getter method for the interest rate, calculated as described above.
8) Include **public static** getter and setter methods for the low and high interest rates.
9) Save the file and make sure it compiles.

Turn in your Lab to D2L

Hand in your lab 5 files to the D2L dropbox:

```
lab5answers.txt
Lab5b.java
Lab5c.java
Lab5d.java
lab5out.txt
SavingsAccount.java
```