**CSc 110 Lab 4**

**Objectives:**

* Become more comfortable with methods, passing parameters and returning values from methods.
* Understand how to write tests for your methods.
* Practice using if-statements

**Activities:**

* Review/New:
  + methods which return values
  + If-Statements
* Write test harness and use it.
* Methods: the method's view
* Paper practice, (designing a solution before implementation)
* More methods and returning values.

**EXERCISES**

**Exercise 0a) Methods which return values (Functional Methods)**

Download and run this program: [Return.java](http://webhome.csc.uvic.ca/%7Ecsc110l/2012_5/Lab4/code/Return.java). Look at the source code and answer these questions.

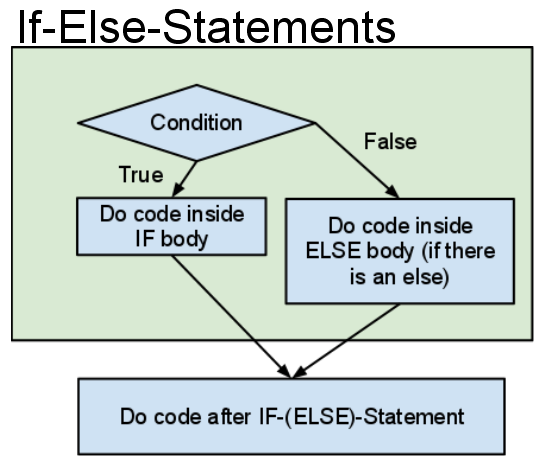
1. What are the String methods used? ([String API](http://download.oracle.com/javase/6/docs/api/java/lang/String.html#method_summary))
2. What are the Math methods used? ([Math API](http://download.oracle.com/javase/6/docs/api/java/lang/Math.html#method_summary))
3. What are the other methods which return a value?
   * What is the return type of each?

**Exercise 0b) Quick coverage of if-statements**

**If-Statements and If-Else-Statements**

These are not loops; they are tools for controlling the flow of a program. We will cover them in more detail in later labs and in lecture.

|  |  |  |
| --- | --- | --- |
| The basics: | | |
| if(condition){    //the true case }else{    //the false case } | if(condition){    //the true case } | if(condition1){    //condition1 is true }else if(condition2){    //condition1 is false but    //condition2 is true }else{    //both conditions are false } |

  
If-Else-Statements can be nested inside each other and conditions can be !negated, and combined with && (and) and || (or). Curly braces are optional around one-line bodies (for example: if(i > 4) System.out.println("no curly braces"); ).

|  |  |  |
| --- | --- | --- |
| Some examples: | | |
| if(guess == answer){    System.out.println("Correct!"); }else{    System.out.println("Sorry, that was wrong."); } | //'in' is a Scanner variable String occupation = in.nextLine(); if(occupation.equals("dentist")){    System.out.println("I'm flossing right now"); } | if(a < 0 || b < 0){    System.out.println("a or b or both are negative");    if(a < 0 && b < 0){       System.out.println("both a and b are negative");    } }else{    System.out.println("a and b are both positive"); } System.out.println("This will always print."); |

**Exercise 1 Testing Your Methods**

Download the file [TestMethods.java](http://webhome.csc.uvic.ca/%7Ecsc110l/2012_5/Lab4/code/TestMethods.java) and compile the given code.  Inspect the code.  The TA will conduct a run through of the methods to get you started.

Your task is to fill in the main method to test the other methods.

You should select values where you know the answer before you conduct a first test.  In later tests you should consider special values, such as zero, or two values that are the same, or where parameters are relatively prime (they don't share factors).

For each method that returns a value to the main method, create a variable of the right type to assign the value and then print the variable, with a meaningful println.  e.g.

int origin = 0;  
int a = -3;  
int b = 2;  
int x = farthest(a, b, origin);  
System.out.println(x + " is the farthest from " + origin);

**For the first method tested use literals as parameters.**  In a test context this is what we usually do.

**In the next method test** create variables, assign values and pass those **variables as parameters**.  This is what is usually the case when the method becomes part of a program.

**Proceed, alternating literals and variables**, until all the methods are tested.

You should compile and test each method as you go.

**Exercise 2 Simple functional methods.**

The TA will have the class develop a header for each method in this exercise. These new methods can go in the same class as the last exercise: TestMethods.java, if you want.

For this exercise (and each that follows) your main method must:

1. Print "Testing method <name>"
2. Define any necessary variables (if using variables instead of literals\*)
3. Call the method and capture the result
4. Print the returned result.

\* Arbitrarily let's say that if the method is given an **even** number of parameters we must **pass literals**, and if there is an **odd** number of parameters we will define **variables** for it.

Parameter type is part of a method signature.

a) Write a method add which, given 2 numbers of type double, returns the sum as double.

b) Write a method add which, given 2 numbers of type int, returns the sum as int.

c) Write a method "concatenate" which, given 3 Strings representing a first name, an initial and a last name, returns a single String with the three put together with spaces between.

**Exercise 3 More functional methods.**

Start as a paper exercise, with pseudocode  
  
Given**:**

int n1 = 1;  
int d1 = 6;  
int n2 = 3;  
int d2 = 4;

Where n1/d1 represents the fraction 1/6 and n2/d2 represents the fraction 3/4, develop the following solution:

1. Define the values n1, d1, n2, and d2 as above
2. Define a new int variable d3
3. Call the lowest common multiple on the denominators (d1 and d2) and store the return value in d3
4. Re-assign n1 such that it is set to d3/d1 \* n1
5. Re-assign n2 such that it is set to d3/d2 \* n2
6. Define n3 and set it equal to n1 + n2
7. Using n1, n2, n3, d1, d2, and d3 create a print statement that will output: 1/6 + 3/4 = 11/12

Implement (on the computer):  
Code a method that will conform to this call: addFractions(1, 6, 3, 4);  
This new method should go in the class: TestMethods.java. It calls the method lcm().  
Just a stub first:

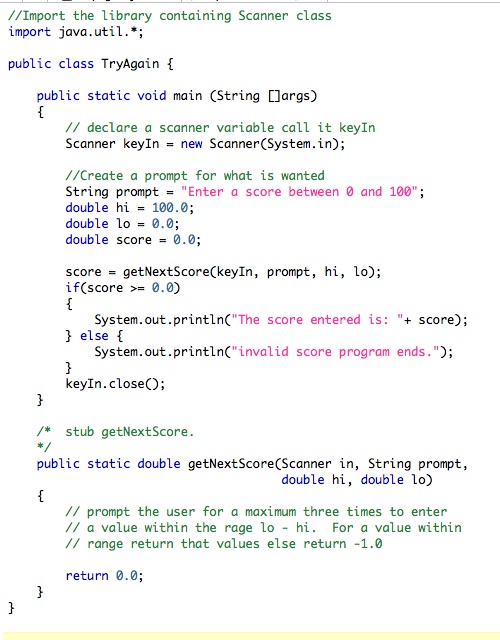
public static void addFractions(...){  
  
  
}

Test it.  Complete it.  Test again.  
Try different values to see if it works with other fractions.   
**What value(s) can make it fail?**

**Exercise 4 Range Checking**

a) Download the [TryAgain.java](http://webhome.csc.uvic.ca/%7Ecsc110l/2012_5/Lab4/code/TryAgain.java) pictured below and test it.

b) Fill in the logic for getNextScore and test it.

(Note one more time that since method variables are a local copy of the parameters they can be called by the same name or by a different name.  It is useful to think of getNextScore as a method that is passed 4 parameters, a Scanner, a String, a double and a double, for it is type, number and order of parameters, in addition to the method's name, that define a method's signature.)  
  
  
You may want to keep the program above for a reference in setting up a Scanner for keyboard input.

c) Create a method, getAvgScore that gives an appropriate prompt and then repeatedly calls getNextScore, eventually returning the overall average score to the main method.

**Exercise 5: Dice Rolling**

Download [Dice.java](http://webhome.csc.uvic.ca/%7Ecsc110l/2012_5/Lab4/code/Dice.java) and follow the steps in the comment below the source code.

The TA will explain how the roll() method works with Math.random to get a random int in the range specified.

**The End**