Exercises: Loops and Conditionals

Start by making a new Eclipse project called basic-syntax-ex or some such. Give it a package. These problems are roughly in order of difficulty.

1. Add a class called LoopTest to your package. Give it a method that will take a number and print out the numbers from 0 up to and including that number. For example, if you pass in 3, it should print 0, 1, 2, and 3. Your method will be very similar to the listNums1 method from the class notes, and your overall code will look roughly like this:

```
package yourpackage;

public class LoopTest {
  public static void main(String[] args) {
    printNums1(3);
  }

  public static void printNums1(int upperLimit) {
    (TODO)
  }
}
```

2. Add a second method to your class. This one should print every other number: i.e., 0, 2, 4, etc., up to the last value that is less than or equal to the argument to the method. For example, if your main method calls printNums2(7), it should print 0, 2, 4, and 6. Remember that "i++" in a loop is the same as if you had written "i = i + 1".

3. Add a third method to your class. This one should be similar to your first method, but should print in reverse order. For example, if your main method calls printNums3(5), it should print 5, 4, 3, 2, and 1. Are there any arguments to printNums3 that would yield bad results?

4. Copy the ArraySum class from the basic-syntax project into your project. Add a method that, given an array of doubles, will return the average value. (Hint: have your new method make use of the existing arraySum1 method, and then use the length property of arrays.) For example, given the array already made inside main (containing 1.1, 2.2, and 3.3), calling arrayAverage(numbers) will output 2.1999... (almost 2.2, but not exactly, due to roundoff error).

5. Add a method that, given an array of doubles, will return the count (int) of how many of them are greater than or equal to zero. For example, For example, given the array already made inside main, calling numPositive(numbers) will output 3. Add a few negative numbers to the array and verify that you still get 3. Add a new positive number and verify that you now get 4.

6. Add a method that, given an array of numbers, a lower bound, and an upper bound, will return the count of how many of the array entries are between the two bounds, inclusive. For example, given the array already made inside main (containing 1.1, 2.2, and 3.3), calling numInRange(numbers, 1.1, 3.2) should return 2.

Exercises:
Strings, Arrays, Math Routines, and Input

Continue adding code to the Java project you made for the previous set of exercises.

1. Make a static method that, given an array of Strings and a potential match, will return true if the array contains an entry matching the potential match, false otherwise. Test with at least one positive match and at least one negative match, and for each, print out whether or not there is a match. Your overall code will look roughly like this:

```
public class StringMatching {
  public static void main(String[] args) {
    String[] testStrings = { "Hello", "Hi", "Hola", "Howdy" };
    if (isStringInArray(testStrings, "Hola")) {
      (TODO)
    if (isStringInArray(testStrings, "Hey")) {
      (TODO)
  }

  public static boolean isStringInArray(String[] strings,
                                        String potentialMatch) {
    (TODO)
  }
}
```

2. Make a new class whose main method flips a coin 10 times, saying “heads” or “tails” each time. Recall that Math.random() returns a double between 0 and 1.

3. Make a new class whose main method creates an array of 4 random doubles (each between 0 and 1). Use one-step array allocation:

```
double[] nums = { ... };
```

Loop down the array and print out the values.

4. Make a new class whose main method creates an array of 100 random numbers. Use two-step array allocation:

```
double[] nums = new double[100];
for(int i=0; i<nums.length; i++) {
  (TODO)
}
```

After the array has been created, loop down the array, calculate the sum of the square roots of the values, and print out the result.

5. Make a variation of the coin-flipping program of problem #2 that flips a coin the number of times the user specifies. Get input from the user using JOptionPane, then convert the String to an int using Integer.parseInt. Do not worry about handling illegal values from the user. Note the required “import” statement at the top of the class (below the package statement but above the start of the class) when you use JOptionPane. See the Input2 example in the slides for details.