Fifth Lesson: Java Continued

Hi again! We'll continue to go through some fundamental concepts in Java today, as well as some additional tips and tricks.

User Input

Like we've been saying, Java is unfortunately much clunkier than Python! In order to get the user input in Java, we have to use the Scanner object (as opposed to the simple `input( )` function in Python). Doing so usually looks like this:

```
//remember to import the correct packages at the beginning of //your code! some developing environments will prompt it for you.
import java.util.Scanner;

public class ScannerExample {

      // don't forget your main method!
    public static void main (String[] args) {

        // creates a Scanner object
        Scanner in = new Scanner(System.in);
        System.out.print("Please enter your given name: ");
        String givenName = in.nextLine();
        System.out.print("Please enter your family name: ");
        String familyName = in.nextLine();
        System.out.println("Your name is " + givenName + " " +
           familyName); // pretend this is all one line!
        // Can you guess what the output will be?
    }
}
```

The three pieces of code (plus duplicate) that are bolded are the most important! You will have to include the first two, regardless of the type of input you want from the user. With the third line, however, you get a little bit more variety with how to call on the Scanner:

```
in.nextLine();  //Returns a whole line
in.next();  //Returns the next 'token', so String with no spaces
in.nextInt();   //Returns an int
in.nextDouble(); //Returns a double
```

Random

The Random class in Java is one of the options you can use to generate random numbers! (Using a random number generator could help when making a guessing game, or to generate random colors, sizes of shapes, etc. You get the point!)

```
import java.util.Random;  //Import new packages when needed

public class RandomNumbers  {

    public static void main(String[] args) {

           Random rand = new Random();  //Declare & initialize

        // generate a random number between 0 & 9
           int randomNum = rand.nextInt(10);

        // for a negative range of numbers...
        //Generates a random number between -10 & -1 (inclusive)
        int randNeg = rand.nextInt(10) - 10;

        // for a range that doesn't start at 0...
        // Generates a random number between 50 & 149 (inclusive)
        int randLarge = rand.nextInt(100) + 50;
     }

}
```

Note: You will see that rand.nextInt(10) only goes up to nine because it is EXCLUSIVE! You might also have noticed that it starts at the first character (at index 0). If you wanted to start at a specific index—say, 2 to the end—then you would have to indicate the index you wish to start at using the format rand.nextInt(2, 10). It's important to remember that the beginning index is inclusive, but the ending index is exclusive.

The Random class has a lot of different applications and is not just limited to integers. You can use the Random class to generate a random boolean, double, or float as well. The method calls are listed below:

```
rand.nextBoolean();   //returns true or false
rand.nextFloat();     //returns a float between 0.0 and 1.0
rand.nextDouble();    //returns a double between 0.0 and 1.0
```

If you want to practice using random plus a bunch of the other stuff that we've already covered, you can try creating a random fortune generator using print statements, a random variable, user input, variables, and if/else statements! Try combining these different elements on your own and see where you get.

Comparing Strings (+ String Methods)

We've already covered the "==" operator, which compares two values and returns a boolean depending on whether or not they are the same. Unfortunately, we can't use the == operator to test if two strings are equal because Java is like that sometimes. But, there's a workaround for that: a very specific method called ".equals()"!

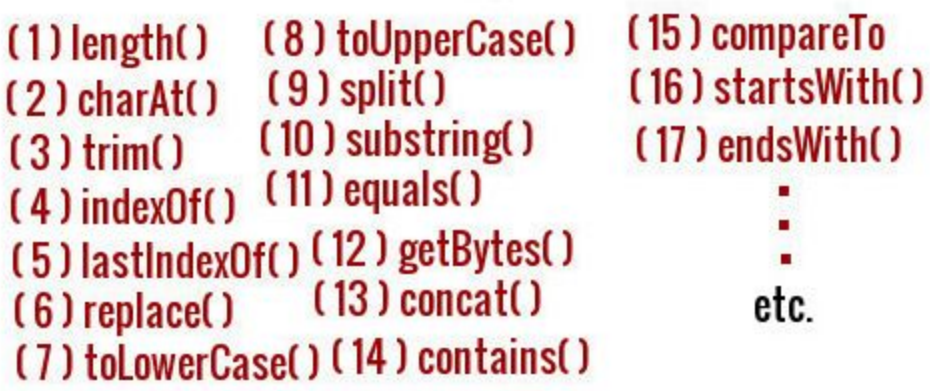
```
String str = "Hello";
String str2 = "Hello";

if (str.equals(str2)) { // MUST use the equals method to compare
                        //two strings to properly run
     System.out.println("Match");
} else {

     System.out.println("No match");
```

It's not particularly difficult! You just have to remember to use this for Strings, while "==" works for most other data types.

This is one example of a string method; the String class in Java contains several of these useful methods, which make life much easier for programmers! Here's a brief list of some of these methods:



(1) gives length of the string (2) provides the character at the inputted index (3) removes extra whitespace (4) provides the first occurrence of an inputted char (5) provides the final occurrence

of an inputted char (6) replaces all the instances of an inputted char in the string with another inputted char (7) makes the entire string lowercase (8) capitalizes the entire string etc…

You'll grow familiar with a lot of these as you work more with strings; it's important to familiarize yourself with these methods because they will save you a lot of time! If you're interested in finding out what (9) - (17) do, there are great resources online that can provide a more in depth explanation with examples. Remember though, that this isn't the limit!

Switch-case Statements

Switch-case statements are sort of similar to if/else statements in the way that they both take a series of options and perform different tasks depending on the choice that is made. Switch-cases tend to be better if you have a large number of options for one variable because they are a bit more concise—not so much to type out. If/else statements, however, can involve multiple variables while switch-cases are limited to one.

```
int x = new Random().nextInt(3);

switch(x) {
      case 0:
            System.out.println("0");
            break;
      case 1:
            System.out.println("1");
            break;
      case 2:
            System.out.println("2");
            break;
      default:
            System.out.println("Something went wrong :(");
            break;
}
```

There's a lot of super-important stuff to unpack here! First and foremost is the switch statement itself. The switch statement essentially states the "thing" that you want to be checked; in this case, we want to check what number `x` is so that we can print out the right number.

Next we have the case statements. The cases are basically the comparisons like in an if statement, except you don't have to constantly retype `if (x == a)`. What goes after **`case`** is your condition, which would simply be what you are trying to compare `x` to. In its body goes what you want to be done if that condition is met; in this case, we want it to print the value of `x`.

Notice that there is a break at the end of each case statement; this is to prevent the code from cycling through every single case. Without the `break`, your code would do exactly that, making the switch-case statement essentially null and void.

Finally, there's the default claim at the end. This is basically your failsafe, to be activated when all other cases are false. You don't necessarily need it, but it's better to have it than not, even if it's just to tell you that your switch statement doesn't work.

Hopefully you've noticed that switch statements only have one definite answer: meaning, you can't use `and or` type things like you would for if-statements.