Syntax and Utilities II

The first three (Lists, Maps, printf) are the most important. I threw in some other problems for developers that already have some experience with these topics.

1. Make a List of Circle objects. Use a random radius. Keep adding circles to the list until Math.random() returns less than 0.01. Then, loop down the list and print out each area.

2. Make a hash table (Map) that associates the following employee IDs with names.

ID	Name
a1234	Steve Jobs
a1235	Scott McNealy
a1236	Jeff Bezos
a1237	Larry Ellison
a1238	Bill Gates

Make some test cases where you test several valid *and* invalid ID's and print the associated name.

3. Change your circle list example (problem 1) so that the areas are printed with the decimal points aligned and with exactly three digits after the decimal point.

4. Make a hash table (Map) that maps numbers (e.g., 2) to words (e.g, "two" or "dos"). Test it out by passing it a few numbers and printing out the corresponding words. Note: hash table keys in Java cannot be primitives; they must be objects. So, technically, you have to convert the numbers to Strings (e.g., String.valueOf(4) returns "4" as a String). But, if you declare the key to be of type Integer, autoboxing will be in effect so that you can use an int and the conversion will occur automatically.

5. Take the state lookup example and modify it so that it works the same no matter what case the user supplies for the state. I.e., Maryland and MARYLAND should work identically.

6. Switch the state lookup example so that it maps state abbreviations to full state names, instead of the other way around.

7. Make a routine that accepts any number of state abbreviations and prints out the corresponding state names.

8. Do some timing tests to compare ArrayList to LinkedList for accessing the middle element. Hints:

- Use System.currentTimeMillis or System.nanoTime to lookup the current time. Compute a delta and divide to get an elapsed time in seconds. Use printf to control the number of decimal places displayed.
- To ensure meaningful results, use very long lists and access the middle element many times.
- Have your program repeat the test several times, and throw out the first result.

9. Do similar timing tests to compare the performance of the two versions of padChar.

10. In the lecture, we talked about how to use classes that *already* support generics. But, it is also possible to create *your own* classes or methods that support generics. Doing so is trickier than simply using preexisting classes, but is not all that hard for the simple cases that account for the majority of uses. See http://docs.oracle.com/javase/tutorial/extra/generics/ (just the first three sections are needed for most cases). If you feel inspired to learn this on your own, try making a static method called RandomUtils.randomElement that takes an array of any non-primitive type, and returns an element of that type, randomly selected from the array. For example:

```
Integer[] nums = { 1, 2, 3, 4 }; // Cannot use int or other primitive
int num = RandomUtils.randomElement(nums); // No typecast: num is 1, 2, 3, or 4
String[] names = { "Joe", "John", "Jane" };
String name = RandomUtils.randomElement(names); // Again, no typecast
```

For those who are interested in this idea but don’t want to read the tutorial, feel free to peek at my solution.