UNIVERSITY OF MICHIGAN
EECS 282

Assignment 4 – Monkey Business. 25 points

Due Date: Feb 11th, 2010, 11:55 PM.

Overview
In this assignment, you will start from your solution to Assignment #3 and make the following changes.

Part 1:

Use jUnit for writing the tests. You should create a separate class, TestMonkey in a file TestMonkey.java, that will contain the unit tests. Start with moving your existing tests from Assignment #3 to the file. These would include the stress tests and the smaller tests. The testinvariants() will remain in the Monkey class, however, as it is called from the Monkey functions, and is not a unit test.

You will have to rename the method stressTest() to testStress() when you move it to TestMonkey. The reason is that jUnit's main program (which is automatically supplied by the framework) looks for methods that start with the word "test". It invokes each of those methods when running the tests.

Test this out to make sure that you got jUnit framework working correctly.

Part 2:

You are going to allow unlimited number of children for a monkey. For that, the current solution of using a fixed-size array is not adequate. The strategy you will use is to wrap the children array into a class MonkeyList. This class should transparently double the size of array when the array's bound is exceeded; it does so by creating a new array and copying the values from the old array to the new one.

Call this class, MonkeyList.java, with a private array attribute, `monkeyArray`, and a `private int length` attribute that tracks the number of monkeys in the list . You can add any other private variables you need. You need to support the following public operations:

- A constructor: it initializes the children array to an initial fixed size. You will start with a size of 2.
- `void add(Monkey m) throws DuplicateMonkeyException:` This will add the monkey m to the array, if it is not already there. If the array bounds are going to be exceeded, you should allocate a new array of double the size, and copy the old array into the new array. Make sure that the `monkeyArray` variable is switched to the new array. If the monkey is already in the array, then it should throw the DuplicateMonkeyException. We have provided you a file DuplicateMonkeyException.java, which you should add to your code. It defines the exception.
- `void remove(Monkey m):` This will remove the monkey m from the array, if it is there.

- `int size()`: get the number of monkeys in the array. This would be >= 0. This should be simply the value of length.

- `Monkey[] getMembers()`: returns a copy of the `monkeyArray`. (Note: It is not really a safe operation to return the `monkeyArray` because the caller would be able to modify it once he get a reference to it, defeating the invariants of this class. So, we are asking you to clone the array and return the cloned array.)

- `boolean contains(Monkey m)`: returns true if the monkey `m` is in the `monkeyArray`. Else returns false.

- `Monkey get(int index)`: return the monkey at the given `index` in the array. `index` starts at 0. If `index` is out of bounds, return null.

Some of the invariants that this class needs to support are:

- Number of non-null monkeys in the `monkeyArray` are equal to the value of `length`
- `length >= 0 and length <= monkeyArray.length`

Add invariant checks at the end of the public functions that modify the state of the list; you are not required to check invariants at entry.

Once you define the MonkeyList class, replace the current children array in the Monkey.java with MonkeyList, something like the following:

```
MonkeyList children;
```

and in the constructor for Monkey:

```
children = new MonkeyList();
```

Modify the rest of your code in Monkey.java to make use of the MonkeyList class. The only difference in specs is that the number of children should be able to grow arbitrarily. For example, you would not need numChildren any more – that can be replaced by children.size() throughout the code. MAX_CHILDREN will also not be needed, since there is now no limit on the number of children.

Modify your tests accordingly. Make sure you test for a situation where the number of children of a node is large.

You should not change the interface to the Monkey class. One design decision that is up to you is whether to have the addChild(m) method pass on the exception DuplicateMonkeyException to the its callers, or to catch it, when adding a duplicate child. The decision is up to you. Other interfaces should not change.

Part 3:

Add two additional methods to the Monkey class:

- boolean isAncestor(m): returns true if m is an ancestor of this monkey, else false. A monkey m is an ancestor if it is either a mom or dad, or an ancestor of mom or dad (note the recursive definition). A monkey is not an ancestor of itself.

- boolean isDescendant(m): returns true if m is a descendant of this monkey, else false. A monkey m is a descendant if it is either a child, or a descendant of a child. A monkey is not a descendant of itself.

Add the following stress test to help check if the above two functions are working.

- Create an array of 50 or more monkeys (randomly male or female), of decreasing age. Arrange them so that monkey 0 has monkey 1 as its child, monkey 1 has monkey 2 as its child, etc. Basically, you will define a long lineage of monkeys.
- Write a loop in which you test for every pair of values i and j, where i and j are array indices in the range 0 to 49, inclusive:
    - if i < j, then monkey j is a descendant of monkey i and monkey i is an ancestor of monkey j.
    - if i == j, then there is no ancestor or descendant relationship between them.
    - if i > j, then monkey j is an ancestor of monkey i and monkey i is a descendant of monkey i.

What to submit: Please submit the files individually, rather than in compressed format.

1. Your source files, Monkey.java, TestMonkey.java, MonkeyList.java, and DuplicateMonkeyException.java.

Note that we may run our own battery of tests on your Monkey class. So, do not modify the names or arguments of public methods – treat it as a contract between you and the 282 teaching staff. You are free to add any private methods or variables. Submit this by attaching each file separately. Do not submit a zip file.

2. The output from running your tests. You can cut/paste the text output from the console into a file output.txt. If you have multiple outputs, submit them in output1.txt, output2.txt, etc.
3. A screen snapshot or text output that shows whether unit tests passed or failed.
4. If your code does not completely work, attach some description in a file "README.txt", which explains the status of the code. Explain the status of the code and the tests.