CS 158 Lab 18 April 8, 2010

The Animal Guessing Game: saving and restoring

1. Make sure that your Animal Guessing Game lab from last time is working completely and correctly. What we want to do now is to be able to save and restore the knowledge a game acquires. The general plan is to first change the end of the program so that it puts the entire tree into an ArrayList object using a preorder traversal. Then we will save that object in a file. Then we'll change the beginning of the program to check for an existing data file and, if found, ask the user if they want to restore it. If they do, read the object back into an **ArrayList**, then remove the nodes, reattaching them to a tree as we go. You should realize that a **Node** object contains reference to other objects in memory. So if these are going to be put into a file, how can they possibly be restored correctly? Any actual memory addresses will be incorrect when the nodes are restored. The answer is that we have to know somehow where they are supposed to go, and reattach them. And we do know where they are supposed to go, because we put them into the **ArrayList** in a special way.

2. The first step is very simple. We are required to assure Java that any objects we put into a file can be reconstructed. To do this reassurance, we simply make each class that is to have objects stored in a file implement the **Serializable** interface. This is extremely easy, since that interface contains no methods. The only class we have that will require this is the **Node** class. ("Serialization" means converting objects to streams of bytes so that they can be stored externally in such a way that they can be recovered as equivalent objects.)

3. Next we will work on saving the tree into a file. First, add and instantiate a static **ArrayList** of nodes as a member variable of the **Game** class. Create a static recursive method called **preorderToArray** that takes a node argument representing the root of a tree, and places that tree into the **ArrayList** in preorder. You should be able to write this method knowing how preorder traversal works. Create a JUnit test for this method. In the test, create five nodes and attach them together as a complete tree, then call **preorderToArray**, then assert that the elements of the **ArrayList** are what you expect them to be. After you get the test to pass, add a call to **preorderToArray** to the main method in the **Game** class after the game ends. Now we need two special stream objects. Add a statement that declares and instantiates a **FileOutputStream** object, and pass in to the constructor the string "animal.dat", which will be the name of the data file we will create. Then declare and instantiate an **ObjectOutputStream** object, and pass in to its constructor the **FileOutputStream** object you just created. Finally, send a **writeObject** message to your **ObjectOutputStream** object, using as the argument the name of the **ArrayList**. Run the game, and then check and see that a file called "animal.dat" was actually created and that it has more than zero bytes in it.

4. Now the hard part: getting it back again. At the beginning of the main method, declare and instantiate a **File** object, passing in to its constructor the string "animal.dat". Next, using that object, you can test if the file exists, and if it does, do all of the following:
    a. Find out if the user wants to restore from the previous game and if so,
    b. declare and instantiate a **FileInputStream** variable, passing in to its constructor the **File** variable you created in this step, and
    c. declare and instantiate an **ObjectInputStream** variable, passing in to its constructor the **FileInputStream** variable you just created, and
    d. invoke the **readObject** method on the **ObjectInputStream** variable, storing what is returned in the **Arraylist** object, and finally
    e. invoke a method yet to be written called **preorderRestoreTree** that will return the restored tree, and store it in the node you used to start the game with.

5. Create the static method **preorderRestoreTree**, which takes no arguments and returns a node object. This will be a recursive method (I hope that wasn't a surprise). Even though this method is the same, conceptually, as **preorterToArray**, there are a couple of extra tricks. You will need three local node variables, the first for the tree to be returned, and the others for the left and right subtrees, which need to be initialized to **null**. Next, check for the base case, which occurs when the **ArrayList** object has nothing in it, and in that case just return **null**. Here is the main part of the method:
    a. Remove the first node from the **ArrayList** and place it in the local tree variable
    b. The subtrees of that node will contain memory addresses for wherever the subtrees were located the last time the program was run. We can't use those addresses, but we CAN use the knowledge of whether they are **null** or not. If the left subtree is not null, make a recursive call to **preorderRestoreTree**, placing the return value in the local left subtree. Either way, if the right subtree is not null, make another recursive call, placing the return value in the local right subtree. You should recognize the preorder traversal that just occurred.
    c. You still have to connect the nodes. So make the actual left subtree of the local tree equal to the local left subtree, and make the actual right subtree of the local tree equal to the local right subtree.
    d. Return the local tree variable.

Run the game enough to verify that it all works correctly, and put the Game.java and Node.java files on the shared drive in a new folder called "animal".