Objective. To implement a non-trivial Java program.

Over the next two weeks we’ll be writing our first Java programs. For the most part, we will focus on the conversion of an example of Python we developed in class—a program the plays “Twenty Questions.”

If you recall, the program makes use of a binary tree container structure, Tree. This class is minimal—it consists of an immutable class that holds a single value. The power of the structure, of course, are a pair of references that enable a recursive definition of a tree that contains an arbitrary number of values.

The q20 application uses the binary tree to maintain a database of questions and guesses. The leaves of the tree—the Tree nodes that make no reference to other Tree nodes—hold the noun phrases that make up the program’s potential guesses. The internal nodes—the Tree nodes that have both left and right references—hold the questions the program might ask to distinguish among the possible guess. Every guess has a unique sequence of questions and answers. Any pair of possible guesses can be distinguished by a different answer to some important question.

Your job is to

1. Implement a Java data structure, Tree, that maintains a binary tree of String data, and
2. Implement q20, a collection of static Java methods that play the game of Twenty Questions in exactly the same manner as the Python program.

The pair of classes—Tree and q20—should be developed and committed by the end of next week’s lab on Monday or Tuesday. The milestones, described below, will help you accomplish this task.

Required Tasks.
As expected, if your CS username is 22xyz9, you can clone the repository for the lab:

```
git clone ssh://22xyz9@davey.cs.williams.edu/cs134/22xyz9/lab10.git ~/cs134/lab10
```

The lab folder contains a number of important items including a copy of the lab handout, the Python version of the program, and a web page resources.html that contains links to Java documentation you may find useful. We’ve also included starter files for Tree.java and q20.java.

The following strategy will allow you to make the most progress while we’re meeting in lab. We hope that you can complete most of this through work during our two remaining lab times, but time outside of lab may also be necessary.

1. You should write the following program, first.java, that prints Hello, world.:

```java
public class first {
    public static void main(String[] args) {
        System.out.println("Hello, world.");
    }
}
```
Over the next week we expect you to be able to explain the reason for each of the keywords used in this iconic program. Compile and run your program:

```
-> javac first.java
-> java first
Hello, world.
```

2. Flesh out the code for the Tree class. Unlike the Python version of this object, this class allows only the storage of String types.

(a) Sign your name in the comment at the top of the Tree.java file.

(b) Add private class instance variables that correspond to the “slot attributes” of the Python Tree class: value, left, and right. In Java, the use of the keyword private makes it impossible for agents outside the Tree class to access these variables. Do not use underscores in the variable names.

(c) Add a public Tree constructor that takes a value and references to two other Tree types.

```
public Tree(String value, Tree left, Tree right)
```

Its purpose is to assign reasonable values to the private instance variables. Recall that the reference to the object is called this instead of self and that it is defined implicitly; the variable this does not appear in the parameter list. Notice also that constructors are not like other methods: they don’t have return types. Their purpose is simply to initialize the instance variables of the object.

(d) Because Java does not have default parameter values like Python, it allows the specification of multiple constructors that are distinguished by the number and types of parameters. If we define

```
public Tree(String value)
// construct a leaf node containing *value*, a String
{
    this(value, null, null);
}
```

this allows us to call the three-parameter version of the constructor with default null (ie. None) Tree references if the left and right parameters are omitted.

(e) Write three accessor methods that allow outside agents to (only) read the instance variables:

```
public String value()
public Tree left()
public Tree right()
```

Note that these methods have the same name as the instance variable they access. Since Java does not support the notion of properties, method calls can always be distinguished from variable accesses. You will find these methods are very easy to write without ambiguity. Always remember, however, to include the parentheses when you call the accessor methods!

(f) Write an isLeaf method that identifies when a Tree node does not refer to subtrees:
public boolean isLeaf()

Be careful: Java uses the keyword null to indicate a non-reference (compare to None from Python). In addition, the == operator in Java compares references (like the is keyword of Python). Java also uses different boolean operators: && corresponds to Python's and, || means or, and ! is used instead of not. The Java expression a != b is equivalent to Python's a is not b.

(g) In Java, the count of objects contained by a class is returned, typically, by a call to size. Write a method, size, that returns the count of values stored in this Tree:

    public int size()
    
This will give you a chance to test your recursive approach to implementing __len__, from Homework 9. If you didn’t solve that problem, now is a good time to develop the solution with a few experiments.

(h) In Java, the toString method of an object returns a String that represents the Tree:

    public String toString()

We'll allow you to do this however you wish, but you will want to make sure it's useful. When you print Tree objects using the println method, it will implicitly call the toString method to obtain a String to print. This is analogous to the __str__ method of Python:

    Tree t = new Tree("a toaster");
    System.out.println(t);

(i) You should now have a minimal working Tree class. It is often helpful to define a main method that exercises the class that contains it. For example,

    public static void main(String[] args)
    {
        Tree alice = new Tree("alice");
        Tree bob = new Tree("bob");
        Tree carol = new Tree("carol", alice, bob);
        System.out.println(carol.size()); // should print 3
    }

This is equivalent to Python's

    if __name__ == "__main__":
        alice = Tree("alice")
        bob = Tree("bob")
        carol = Tree("carol", alice, bob)
        print(len(carol))  # should print 3

(To run this code, compile the Tree.java file and then have Java run Tree.) We'll not be grading the main method, but you would be wise to develop a method that tests your Tree class sufficiently to convince yourself it works correctly!

At this point, you're advised to take a break and commit and push your code.
3. Knowing that you’ve got the basic data structure written for storing the database of objects, you can turn your attention to the development of the q20.java code.

(a) Add your name to the top of the q20.java file.

(b) Review the code we’ve given you to start. We’ve written a bit of code to simulate Python’s input method. In doing that, we’ve had to import a class called Scanner, at the top. Because your Tree definition can be predictably found in a file called Tree.java in the current directory you do not have to explicitly import it. Indeed, when you compile the q20 program, it will notice that you’re making use of the Tree class and it will automatically compile the Tree source as well. To be honest, Java is fairly lazy: if the date of the .class file is later than the .java file, then it knows it has been compiled, and will not compile it again. In this way, the cost of compiling large programs is minimized.

(c) Write a method, main, that simply calls the method twentyQuestions. We will not pickle or serialize the database in a file, so it is not necessary to pass a file name to twentyQuestions:

```java
public static void main(String[] args)
```

It’s very important to note that even though we do not make use of the command line arguments (args), you must still include them in the declaration or signature of the main method. Otherwise, when you attempt to run the program, java will not find an appropriately formed main method. Java is very conservative and will not make any assumptions.

(d) Write the method twentyQuestions,

```java
public static void twentyQuestions()
```

Its purpose is to construct an initial Tree-structured database that describes its knowledge. It then politely asks the user whether they want to play and, if so, plays a round of Twenty Questions. This continues until the user gives up (likely from exhaustion!). Your code should follow, as close as possible, the original Python code. (Warning: note that strings are compared with the equals method. The Python string comparison expression s == t is accomplished in Java with s.equals(t).) You will find the Java input method we have provided behaves exactly as Python’s input method.

(e) Write the play method. This takes a Tree-structured database and, starting at the root of the Tree asks questions until a guess can be made. Because playing the game may involve learning, the database may possibly change. As a result, play should always return the root of the Tree that resulted from the game play:

```java
public static Tree play(Tree db)
```

This should be a direct translation of the Python code into Java. Make sure you understand how the Python code works before you write the same code in Java!

At this point, you’ve completed the lab. Please, make sure you have signed your files and that you have fully tested your program. If you’re satisfied, submit your code for grading.

Congratulations! You’ve completed the final lab!

*