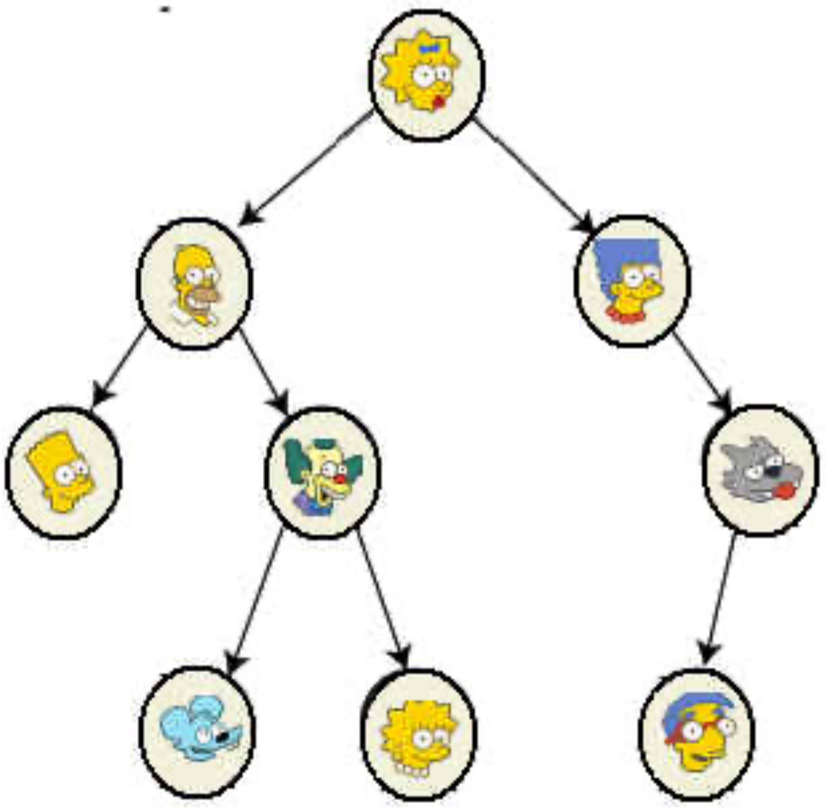
CS 211
Data Structures
Programming assignment 6
Due Oct 23



1. This first program is not really a problem to be solved but simply a demonstration of the use of a binary search tree.

   The file *babynames.txt* contains the most popular baby names during 2020 in Australia. (You can find any data on the web)

   The data in the file is of the form
   name count
   For example
   Arlo 178
   Liam 176

   The count is the number of babies born in 2020 with that name.

Your assignment is to write a program that accepts a name and displays both the name and the count.

First of all, you should make a class

public class Name that implements Comparable with

- two String fields→ one for the name and one for the count
- a default and a \two argument constructor
- an implementation of compareTo based on the name field
- a toString() method that returns the name and count with labels

Next you should create a binary search tree consisting of Name objects. You will use the file babynames.txt to create the tree.

You your program will

- ask for a name,
- search the tree, and
- print the name and count.
- It will also print the height and size of the tree

Here is how it should work:

```
Enter a name end with XXX:  Maddison
Name: Maddison   Count : 50
Enter a name:  Joseph
Name: Joseph   Count : 95
Enter a name:  Ralphie
Name not found
Enter a name:  Liam
Name: Liam Count : 176
Enter a name:  XXX

The height is ----
The size is  ---
```

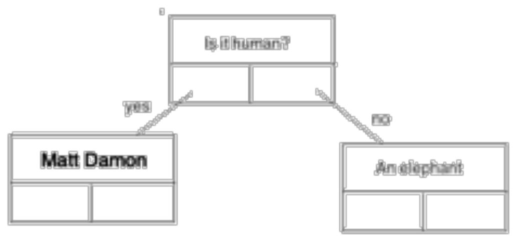
Finally, write the names and counts to a sorted file called *sortedNames.txt* which keeps the names in alphabetical order. You will need a PrinterWriter for that. Remember, you can use a binary search tree for sorting. And don't forget to close the PrintWriter

One last hint: The tree is a tree of Name objects. It is not a tree of String. When you query the user for the name you must create a Name object. The count field can be the empty string. Your search is based on the name field.

Attached are the files that you need.

2. A Guessing Game
This program will take some thought.

Write a program that implements a guessing game such as twenty questions. **However, the more you run the program the more intelligent the program becomes. It learns.** Information is stored in a decision tree. Initially the decision tree is very simple...a root and two leaves. For example,



Here is a sample run. Keep the above tree in mind.

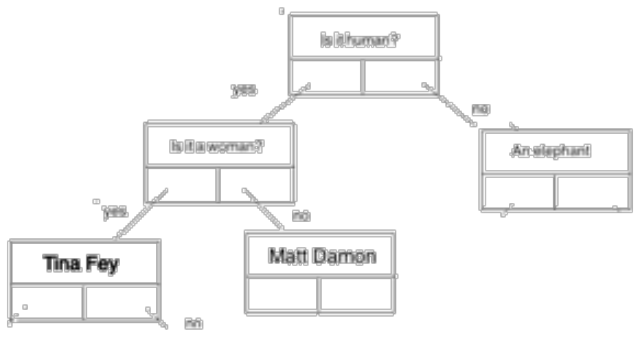
Computer: Think of an Object or person
Computer: Is it human?
User: Y
Is it Matt Damon?
User : N

If the computer guesses incorrectly, it asks the user :

1. **who or what he/she was thinking of and also**
2. **a question that distinguishes that object from the incorrect guess.**

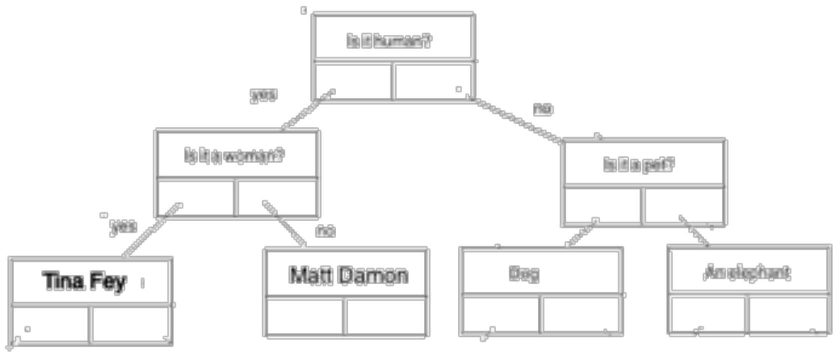
Computer: Who/what were you thinking of?
User: Tina Fey
Computer: Enter a question that distinguishes Matt Damon from Tina Fey
User: Is it a woman?

The new information is added to the tree. (Notice that the answer was yes and Tina Fey is added on the left/yes:



Computer : Play again?
User: Y

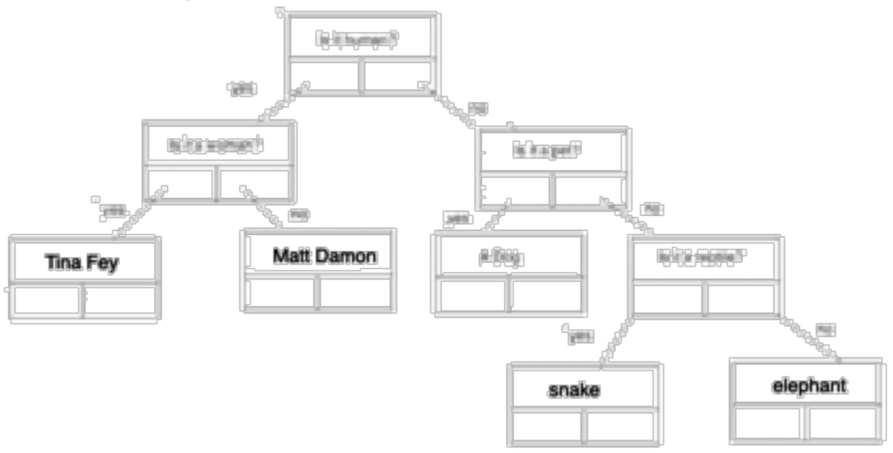
Computer: Think of an Object or person
Computer: Is it human?
User: n
Computer: Is it an elephant?
User: n
Computer: Who/what were you thinking of?
User: A Dog
Computer: Enter a question that distinguishes dog from elephant
User: Is it a pet?



Computer : Play again?
User: y

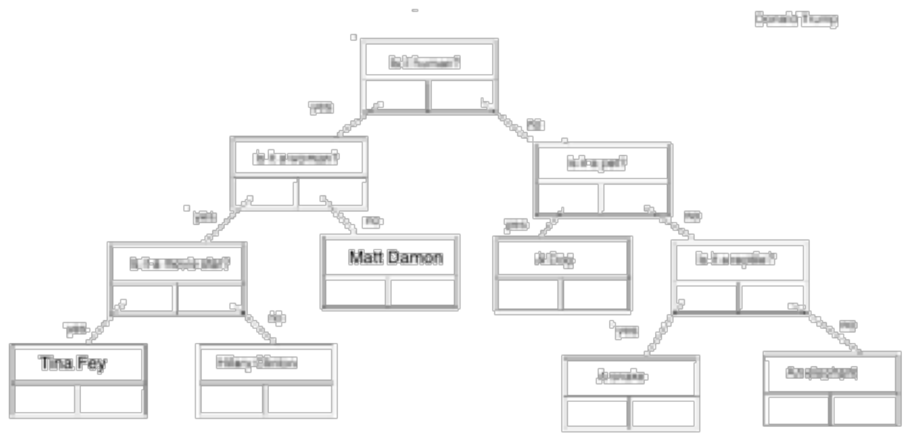
Computer: Think of an Object or person
Computer: Is it human?
User: n
Computer: Is a pet?
User: n
Computer: Is it an elephant?
User: n

Computer: Who/what were you thinking of?
User: A snake.
Computer: Enter a question that distinguishes an elephant from a snake
User: Is it a reptile?



Play again?
User: y

Computer: Think of an Object or person
Computer: Is it human?
User: y
Computer: Is it a woman?
User: y
Computer: Is it Tina Fey?
User: n
Computer: Who/what were you thinking of?
User: Hilary Clinton
Computer: Enter a question that distinguishes Tina Fey from Hilary Clinton
User: Is it a movie star?



Play again?
User: y

Computer: Think of an Object or person
Computer: Is it human?
User: y
Computer: Is it a woman?
User: y
Computer: Is it a movie star?
User: n
Computer: Is it Hillary Clinton?
User: y
Computer: Got it!

Computer: Play again:
User: n
Computer : Bye

Notice;
1. The computer moves left or right from the root depending on whether the answer is yes or no and makes a guess when it reaches a leaf. The guesses are stored in the leaves.

2. The new question is added to the leaf that was used for the computer's guess and two new leaves are added to the tree.

3. If the computer guesses correctly, it just responds "got It" or something similar.

4. When adding the new question, if you always make the answer "yes" or always "no," you will know where to hang the new node... always on the left side or always on the

right side. (If the answer varies, each time, then you should also ask for the answer (yes or no) to decide where the new object goes.)

The program

So how should you structure the program?

1. Make a public class, say GuessNode, that *implements Serializable*. The class will represent one node in the tree. So it will have
   a data(String) field and
   two pointers, call them yes and no, rather than right and left.

   For convenience, you can make the fields of GuessNode public, if you like:

   public class GuessNode implements Serializable

2. Make a class, say, FirstTree, that creates the initial tree (just a root and two leaves) and saves it as a file. You run this **ONCE AND ONLY ONCE**: here is the structure -- put your own questions in it. Notice we are making the class Serializable and using ObjectOutputStream etc. to save the object.

```
import java.io.*;
public class FirstTree implements Serializable
{

        GuessNode root;
        public FirstTree()
        {
                root = new GuessNode("Is it human?"); // or some other
question
                root.yes = new GuessNode("Is it Matt Damon"); // or
someone
                root.no =  new GuessNode("Is an elephant?");
         }
        public static void main(String[] args) throws IOException
        {
                FirstTree root = new FirstTree();
                ObjectOutputStream output = new
ObjectOutputStream(new
FileOutputStream("GuessingTree.dat"));
                output.writeObject(root);
        }
}
```

Notice that you create the first "mini-tree" and then save the object. **Once you run this and create the initial tree, do not run it again unless you want to start from scratch. You have saved the tree object in a file and you can retrieve the tree. That’s what “implement Serializable” does. It allows you to save an object,**

3. Write the program to play the game. Here is a suggestion for the basic structure:

```
import java.io.*;
import java.util.*;
public class GuessingGame implements Serializable
{
   FirstTree t;
   public GuessingGame() throws IOException, ClassNotFoundException
   {
      // Read the tree object from disk
        ObjectInputStream input = new ObjectInputStream(new

FileInputStream("GuessingTree.dat"));
        t = (FirstTree)input.readObject();
   }

   public boolean isLeaf(GuessNode p)
   {
      // are we at a leaf?
   }

   public void play()
   {
       //play the game
   {
 public static void main(String[] args) throws IOException,
                                           ClassNotFoundException
   {
      Scanner in = new Scanner (System.in);
      GuessingGame g = new GuessingGame();
      String answer;
      do
      {
        g.play();
        System.out.print("Play again? Y or y for yes any other key for no:
");
        answer = in.nextLine();
      }while (answer.equals("Y") || answer.equals("y"));

      // save the updated tree to the disk
        ObjectOutputStream output = new ObjectOutputStream(new

FileOutputStream("GuessingTree.dat"));
        output.writeObject(g.t);
```

```
                    }
```

Every time you run the program you will be using the last created tree. The tree will grow each time you run the program. Of course, if the program is to become really good at guessing, the tree will have to be huge or maybe you could restrict the subject of the guesses..."Think of a movie star."