Andrew Hayes
21321503
a.hayes18@nuigalway.ie

CT2109 Assignment 3

2023-05-01

Expandable Binary Tree Guessing Game

1 Problem Statement

The problem of this assignment is to create an expandable binary tree guessing game, not unlike the popular web game "Akinator". There will be a tree which will consist of question nodes. Each node will contain some String data: this will be the question that the node represents. These questions will be "yes" or "no" questions. Each node will have a maximum of two children. These children will represent the next question after the parent has been answered.

The tree will be traversed node by node, starting at the root node. Each node's data will be printed to the user, and they will be prompted to answer the question by entering either "y" or "n". If the user answers "y", then the next node traversed will be the left child of the current node, but if the answer is "n", then the next node will be the right child. Nodes on the left represent an affirmative answer to the parent's question, and nodes on the right will represent a negative answer to the parent's question.

Eventually, a leaf node will be reached (a node with no children). If a node is a leaf node, then this means that this node represents a guess by the game. These guesses will be in the form of a "yes" or "no" question, e.g., "Is it a dog?". If the user enters "y", then the program has won, and the game will be over. If the user enters "n", then the user won the game, and the program will expand it's knowledge by asking the user for a question that distinguishes the game's guess from the correct answer. This question will then be inserted in the tree to replace the leaf node that was the game's guess. The user will then be asked if the answer to the distinguishing question for the correct answer should be "y" or "n", and the initial incorrect guess & the correct answer will be inserted as children of this parent node in the appropriate positions, depending on what the answer to the distinguishing question is.

This program will also implement the ability to save the binary tree to a file and to load a binary tree from a file. The standard way of doing such a thing in Java is to implement the `Serializable` interface. This allows for an object to be written to a file and recovered at a later date, after an indefinite time period has elapsed, regardless of whether the program has been running or not.

2 Analysis & Design Notes

The main method of the program will consist of an infinite loop. The loop will firstly call a `loadTree()` method which does one of three things: load a tree from a file, generate a pre-built tree that's hardcoded in, or use an already defined tree in memory, if one exists. The user will be asked whether or not the tree should be loaded from a file. A case statement will be used to react to the user's inputs, "y" for "yes", "n" for "no". This will loop until a valid input is given. If the answer is "y", then the user will be prompted to enter the name of the file from which the tree should be loaded. This will loop until a valid filename is given (one that exists). Then, the program will attempt to de-serialize the file into a `BinaryTree<String>` object, throwing an error and exiting if any problems are encountered. Because this is not necessarily a safe operation, we presume that the user knows only to supply the program with valid files, and do not check for safety, instead opting to use `@SuppressWarnings("unchecked")` at the top of the class definition. Although it would of course be better practise to ensure file safety, this is somewhat beyond the scope of the assignment, and not really relevant to the theory at hand. If the user opts to not load the tree from a file, one will be generated from some hardcoded values, provided that the existing tree is `null`. If the existing tree is not `null`, then the existing tree is used instead. This will only occur on rematches.

After the tree has been loaded, the `gameplay()` method will be called, and the game will begin. This method will loop over each node, starting at the root node, while the current node is not a leaf node (i.e., while the current node has children). The data of this node (a question String) will be printed out, and the user will enter "y" or "n" to answer the question. If the user enters anything else, it will loop on this question until an appropriate answer is given. Then, depending on the input, the current node will be replaced with the left or the right child of the current node, left for "y", right for "n". An appropriate answer will break out of the loop using a label.

When a leaf node is reached, the loop will end and a guess will be made. The user will be asked to verify the guess, and this will loop until an appropriate answer is given. If the user confirms the guess as correct, the game ends, and the user will be presented with a game menu. Otherwise, the user will be asked what they were thinking of. They will then be asked to provide a question to distinguish what they were thinking of from the program's guess, and the appropriate answer to that distinguishing question for the answer they were thinking of. The current node will then be replaced with the distinguishing question, and the guessed answer & the real answer will become child nodes of this node, placed appropriately on the right or left according to the user's instructions.

Finally, after the `gameplay()` method has completed execution, the user will be presented with the options to play again, quit, or save the tree to a file. If they choose play again, the infinite loop simply repeats, and they are presented with the `loadTree()` operations again. If they choose to quit, the program will exit with code `0`. If they choose to save the tree to a file, the `storeTree()` method will be called. This method prompts the user to enter a filename to which the serialized tree should be saved. The serialized object is then written to this file, overwriting any data that was already in that file, if it existed.

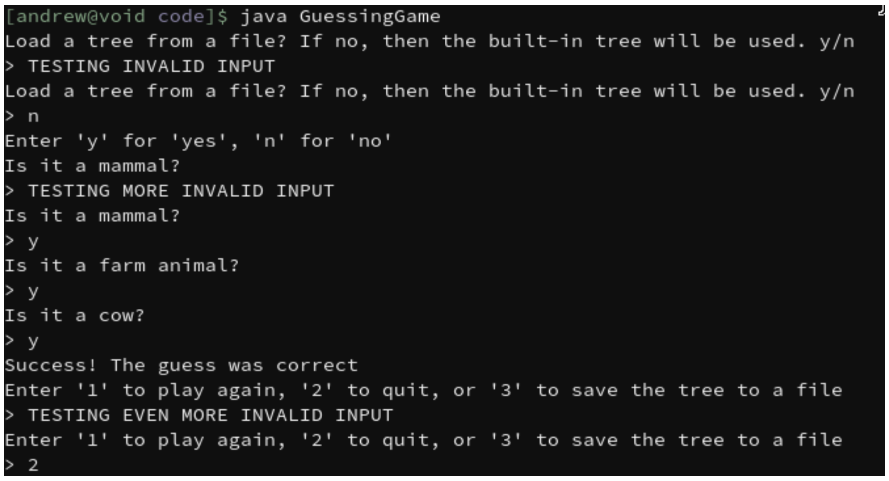
3 Code

```java
import java.util.*;
import java.io.*;

@SuppressWarnings("unchecked")      // ignoring warnings lol
public class GuessingGame implements Serializable {
    private static BinaryTree<String> tree = new BinaryTree<String>();
    private static String input;                         // string to which input will be read
    private static Scanner sc = new Scanner(System.in); // scanner to read in input

    public static void main(String[] args) {
        while (1 == 1) {
            // pick the tree that will be used for the game, either from a file or built-in
            loadTree();

            // play the game
            gameplay();

            playAgainLoop: while (1 == 1) {
                System.out.printf("Enter '1' to play again, '2' to quit, or '3' to save the tree to a
    file\n> ");
                input = sc.nextLine();

                switch (input) {
                    case "1":   // going back to start of loop
                        break playAgainLoop;

                    case "2":
                        System.exit(0);
                        break playAgainLoop;

                    case "3":   // storing the tree, this should not break the loop
                        storeTree();
                        break;

                    default:
                        // loop
                }
            }
        }
    }

    // method that serializes a tree object to a file
    public static void storeTree() {
        // scanning in the name of the file from the user
        System.out.printf("Enter the name of the file that it the tree should be stored as\n> ");
        input = sc.nextLine();

        try {
            // creating output streams
            FileOutputStream fos = new FileOutputStream(input);
            ObjectOutputStream oos = new ObjectOutputStream(fos);

            // writing the tree object ot the file
            oos.writeObject(tree);

            // closing output streams
            oos.close();
            fos.close();
        }
        // catching IOExceptions
        catch (IOException E) {
            System.out.println(E);
            System.exit(1);
        }
    }

    // method to load a tree, either from a file, from memory, or hardcoded in
    public static void loadTree() {
        // looping until an appropriate choice is made
        loadTreeLoop: while (1 == 1) {
            System.out.printf("Load a tree from a file? If no, then the built-in tree will be used. y/n\n
 > ");
            input = sc.nextLine();

            switch (input) {
                case "y":
                    // looping until valid filename is entered
                    while (1 == 1) {
                        System.out.printf("Enter the file from which the tree should be loaded\n> ");
                        input = sc.nextLine();

                        File treefile = new File(input);

                        // breaking if the file exists
                        if (treefile.exists()) {
                            break;
                        }
                    }

                    try {
                        // creating input streams
                        FileInputStream fis = new FileInputStream(input);
                        ObjectInputStream ois = new ObjectInputStream(fis);

                        // deserializing tree object
                        tree = (BinaryTree<String>) ois.readObject();

                        // closing input streams
                        ois.close();
                        fis.close();
                    }
                    // printing errors and crashing
                    catch(IOException E) {
                        System.out.println(E);
                        System.exit(1);
                    }
                    catch (ClassNotFoundException E) {
                        System.out.println(E);
                        System.exit(1);
                    }

                    break loadTreeLoop;

                case "n":
                    // if no tree is defined building the default tree
                    if (tree.getRootNode() == null) {
                        // first the leaves
                        BinaryTree<String> cowTree      = new BinaryTree<String>("Is it a cow?");
                        BinaryTree<String> dogTree      = new BinaryTree<String>("Is it a dog?");
                        BinaryTree<String> fishTree     = new BinaryTree<String>("Is it a goldfish?");
                        BinaryTree<String> geckoTree    = new BinaryTree<String>("Is it a gecko?");

                        // Now the subtrees joining leaves:
                        BinaryTree<String> mammalTree    = new BinaryTree<String>("Is it a farm animal?",
  cowTree, dogTree);
                        BinaryTree<String> notMammalTree = new BinaryTree<String>("Is it a type of fish?"
 , fishTree, geckoTree);

                        // Now the root
                        tree.setTree("Is it a mammal?", mammalTree, notMammalTree);
                    }

                    break loadTreeLoop;

                default:
                    // loop
            }
        }
    }

    public static void gameplay() {
        System.out.println("Enter 'y' for 'yes', 'n' for 'no'");

        BinaryNodeInterface<String> curr = tree.getRootNode();  // current node

        // looping until a leaf node is reached
        while (!curr.isLeaf()) {
            // looping until an appropriate reply is given by the user
            answerloop: while (1 == 1) {
                // printing the question & scanning in the answer
                System.out.printf(curr.getData() + "\n> ");
                input = sc.nextLine();

                switch (input) {
                    case "y":   // continuing via left node if answer to question is yes
                        curr = curr.getLeftChild();
                        break answerloop;

                    case "n":   // continuing via right node if answer to question is no
                        curr = curr.getRightChild();
                        break answerloop;

                    default:
                        // loop
                }
            }
        }

        // making a guess
        // looping until an appropriate reply is given by the user
        guessloop: while (1 == 1) {
            // printing the question & scanning in the answer
            System.out.printf(curr.getData() + "\n> ");
            input = sc.nextLine();

            switch (input) {
                case "y":   // printing a success message if the answer is yes
                    System.out.println("Success! The guess was correct");
                    break guessloop;

                case "n":   // inserting a new question and putting the two guesses beneath it if wrong
                    System.out.printf("Enter the animal that you were thinking of\n> ");
                    String thinkingOf = sc.nextLine();
                    String wrong = curr.getData();

                    // ask the user for a question to distinguish the wrong guess from the correct answer
                    System.out.printf("Enter a question to distinguish %s from '%s'\n> ", thinkingOf, wrong);
                    String question = sc.nextLine();

                    // replacing the current node with the question to distinguish it
                    curr.setData(question);

                    // asking the user for the correct answer to the question for the animal they were thinking of
                    addNodeLoop: while (1 == 1) {   // looping until an appropriate answer is given
                        System.out.printf("Enter the correct answer to the question that you entered for %s (y or n)\n> ", thinkingOf);
                        input = sc.nextLine();

                        switch (input) {
                            case "y":   // adding thinkingOf to the left of the question node if the answer is yes
                                curr.setLeftChild(new BinaryNode<String>("Is it a " + thinkingOf + "?"));
                                curr.setRightChild(new BinaryNode<String>(wrong));
                                break addNodeLoop;

                            case "n":   // adding thinkingOf to the left of the question node if the answer is no
                                curr.setLeftChild(new BinaryNode<String>(wrong));
                                curr.setRightChild(new BinaryNode<String>("Is it a " + thinkingOf + "?"));
                                break addNodeLoop;

                            default:
                                // loop
                        }
                    }

                    break guessloop;

                default:
                    // loop
            }
        }
    }
}
```

GuessingGame.java

4 Testing

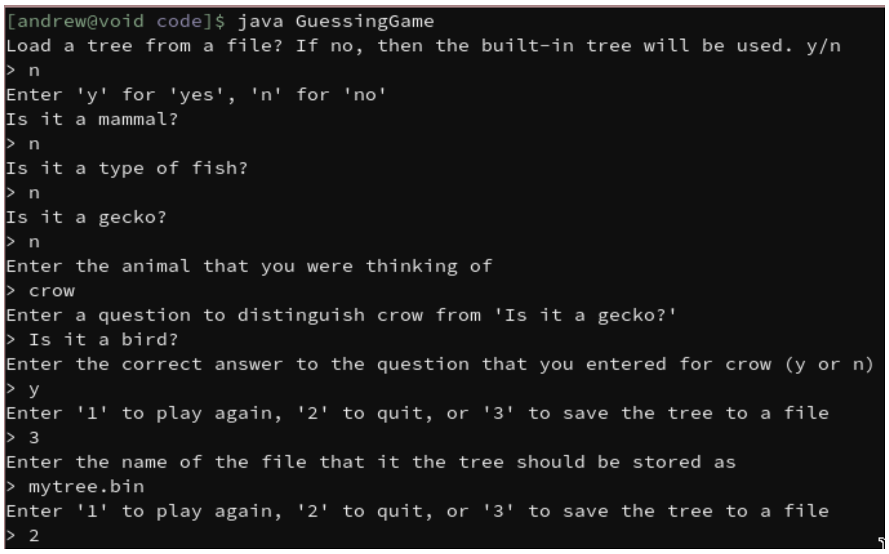
The first thing to be tested is just basic functionality of the game. The screenshot below shows basic testing with valid & invalid input, but only with animals known to the game. Invalid input should just be re-prompted to be entered.



```
[andrew@void code]$ java GuessingGame
Load a tree from a file? If no, then the built-in tree will be used. y/n
> TESTING INVALID INPUT
Load a tree from a file? If no, then the built-in tree will be used. y/n
> n
Enter 'y' for 'yes', 'n' for 'no'
Is it a mammal?
> TESTING MORE INVALID INPUT
Is it a mammal?
> y
Is it a farm animal?
> y
Is it a cow?
> y
Success! The guess was correct
Enter '1' to play again, '2' to quit, or '3' to save the tree to a file
> TESTING EVEN MORE INVALID INPUT
Enter '1' to play again, '2' to quit, or '3' to save the tree to a file
> 2
```

Figure 1: Basic Testing of the Game with the In-Built Tree

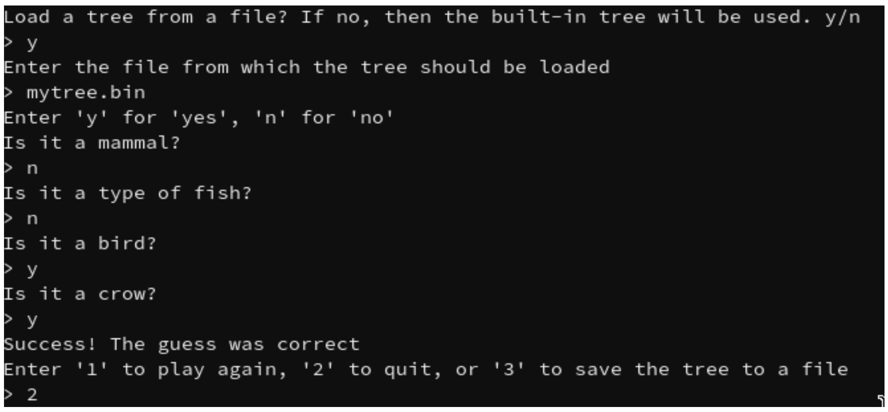
The next bit of testing is testing of an animal that the game doesn't know, adding it to the tree, and saving it to a file, as shown in the screenshot below.



```
[andrew@void code]$ java GuessingGame
Load a tree from a file? If no, then the built-in tree will be used. y/n
> n
Enter 'y' for 'yes', 'n' for 'no'
Is it a mammal?
> n
Is it a type of fish?
> n
Is it a gecko?
> n
Enter the animal that you were thinking of
> crow
Enter a question to distinguish crow from 'Is it a gecko?'
> Is it a bird?
Enter the correct answer to the question that you entered for crow (y or n)
> y
Enter '1' to play again, '2' to quit, or '3' to save the tree to a file
> 3
Enter the name of the file that it the tree should be stored as
> mytree.bin
Enter '1' to play again, '2' to quit, or '3' to save the tree to a file
> 2
```

Figure 2: Testing of the Saving a Tree to a File

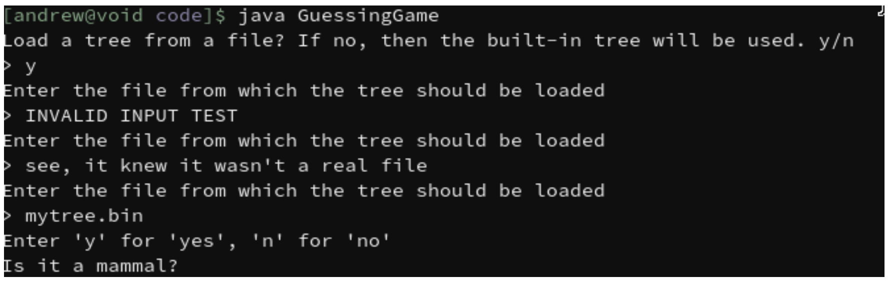
The next bit of testing is testing restoring a binary tree from a file on disk, using the file made above.



```
Load a tree from a file? If no, then the built-in tree will be used. y/n
> y
Enter the file from which the tree should be loaded
> mytree.bin
Enter 'y' for 'yes', 'n' for 'no'
Is it a mammal?
> n
Is it a type of fish?
> n
Is it a bird?
> y
Is it a crow?
> y
Success! The guess was correct
Enter '1' to play again, '2' to quit, or '3' to save the tree to a file
> 2
```

Figure 3: Testing of the Restoring a Tree from a File

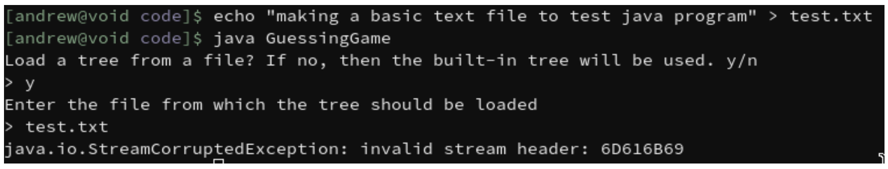
Testing trying to load a tree from a non-existent file on disk. This should loop until a valid input is given.



```
[andrew@void code]$ java GuessingGame
Load a tree from a file? If no, then the built-in tree will be used. y/n
> y
Enter the file from which the tree should be loaded
> INVALID INPUT TEST
Enter the file from which the tree should be loaded
> see, it knew it wasn't a real file
Enter the file from which the tree should be loaded
> mytree.bin
Enter 'y' for 'yes', 'n' for 'no'
Is it a mammal?
```

Figure 4: Testing Trying to Load a Non-Existent Tree File from Disk

Testing trying to load a normal text file as a binary tree file. Should throw an error and exit gracefully.



```
[andrew@void code]$ echo "making a basic text file to test java program" > test.txt
[andrew@void code]$ java GuessingGame
Load a tree from a file? If no, then the built-in tree will be used. y/n
> y
Enter the file from which the tree should be loaded
> test.txt
java.io.StreamCorruptedException: invalid stream header: 6D616B69
```

Figure 5: Testing Trying to Load a Non-Binary Tree File from Disk