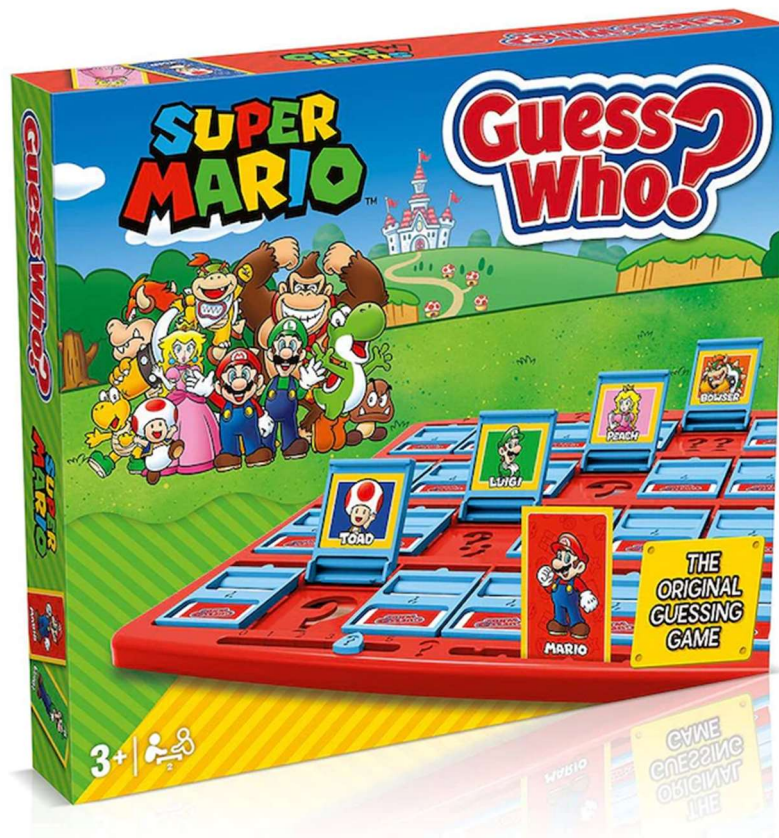


Maxwell Maia

21236277

CT2109 Object Oriented Programming: Data Structures and Algorithms

Assignment 5: Guessing game



Problem Statement

Create a guessing game that uses a tree structure that can be updated, the more times you play it, the more information the tree holds. There should be a way to save the tree object into the persistent storage of the computer and also a way to load the tree object from the persistent storage of the computer. This means the user can grow their tree, save it, close the program and when they come back later they can open it, load the tree and the program will still have their progress. The user will think of a word at the start of the game. The computer will ask the user questions to try and figure out the word. Based on the responses of the user (yes/no), the computer will navigate the tree structure and keep asking questions until it runs out of questions, at which point it will guess what the user is thinking of. The program asks if it was correct. If it was correct, that's the end of the game. From here the user can play again, store or load a tree or quit the game. If the program did not guess the user's word correctly then it will ask for what the word was, save that response, ask for a word that differentiates the computer's guess and the actual correct answer and then update the binary tree with the question and 2 answers. In this way, the knowledge of the guessing computer in the game can infinitely grow using the user's inputs.

Design notes

Game loop

As per the design, the leaf nodes are guesses. When the program navigates the tree, it needs to know when to guess the correct answer. It would be easy to check if the node is a guess by having it that all nodes have a boolean which is true if the content of the node is a guess. If this does not work out, I could check if the node is a leaf node.

Update: After reading the documentation provided, the way that the program will tell that there is a guess is to use the `isLeaf()` method from the provided code. Also I learnt that the nodes only have 1 variable for data. So the boolean idea talked about above will not be used.

The whole program will go in a loop. This is to make the game playable and re-playable without closing the program.

The user will be given some options

1. Play
2. Load a tree
3. Quit

Inside the game loop:

I imagine a loop, starting at the root node and moving down depending on the user's responses, Yes/No. Once the next node contains a guess then exit this navigation loop and handle the guess.

If the guess was correct then give the user the option to

1. Play again

2. Save the tree
3. Load another tree
4. Quit

Else (the guess wasn't correct)

Update the guess node by replacing its content with a question and put the guesses in new nodes as it's children.

Save the user response, creating nodes with the guesses and updating the tree with the nodes.

When the program is started, create an initial binary tree with 4 levels.

The creation the initial tree is done in a different way to how the nodes will be changed later in the program.

Create an initial tree:

The function: createInitialTree takes in a Binary tree object which just contains the root node

The tree is constructed and the root node is added last.

The tree is constructed as follows:

The leaf nodes are created as their own BinaryTree objects by calling the BinaryTree constructor with the data of the leaves, the data being the guesses.

Then the parents of the leaf nodes are created by calling the BinaryTree constructor with the data (the questions) and the children, which are the leaf nodes we just created.

Printing the tree

The function: printBinaryTree takes in a Binary tree and prints out a text representation of its content.

Print all contents of nodes at each level on the tree

Use breadth first traversal

Add root to queue

Loop

Dequeue the front node

Enqueue left child

Enqueue right child

If(isLeaf) The node is a guess; else The node is a question

Print contents

Saving and loading from persistent storage

I will use object serialization

Saving trees:

I will have a method: saveTree.

saveTree takes a BinaryTree<String> object as an argument.

I am going to turn the tree object into text and save it to a .txt file. This will be done using the inbuilt functionality of java, specifically the File, FileOutputStream and ObjectOutputStream packages. The object is saved using the “.writeObject()” method. This is done within a try catch clause because there could be errors occurring when working with files on the system. The exception will handle this and the program will not crash because of an error here.

Loading trees:

I will have a method: loadTree.

loadTree returns a BinaryTree<String> object.

I will read the .txt file using File, FileInputStream and ObjectInputStream packages. The tree object is retrieved from the text file using the “.readObject()” method. This is also done within a try catch clause for the same reason as above.

Menus

There are 2 menu's:

The start menu

And the menu that appears at the end of the game

Both menus have a helper function:

Start menu:

BinaryTree startMenu(BinaryTree tree)

End of game menu:

BinaryTree endGameMenu(BinaryTree tree)

Both menu's take in a tree and return a tree. This allows the tree to be updated if a tree is loaded.

The difference between the menu's is that the end of game menu has the ability to store the tree, where the start menu does not. I think they should be separate for maintainability. Each menu might have many more different features.

Helper function for menu's

getUserResponseMenu method takes in:

String message

and int maximumValidOptions

It will return the user response when prompted with a message, given a number of valid options. The method will check that the response is valid. Returns -1 for an invalid response.

It uses Integer.parseInt and JOptionPane.showInputDialog.

Helper method for playing the game

getUserResponseGame takes in :

String message

It will ask the user a question and accepts an input in a dialog box. It will continue to ask the user the question until the user enters a valid response. A valid responses are 'yes', 'y', 'no', 'n'. Not case sensitive using Java's .toLowerCase method and then checking for valid responses in lower case.

Code

```
import javax.swing.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class Main {
    public static void main(String[] args)
    {
        //Create an initial binary tree
        BinaryTree<String> initialTree = new BinaryTree<String>();
        createInitialTree(initialTree);
    }
}
```

```

BinaryTree<String> tree = null;

//Start menu
startMenu(tree);

while(true)
{
    //If the user did not load a tree, or the tree could not be loaded
    // use the initial tree
    if(tree == null)
    {
        tree = initialTree;
    }

    int yesNoResponse = -1;
    String gameMessage = "";

    //Game loop
    while(true)
    {
        BinaryNodeInterface<String> currentNode = tree.getRootNode();
        //Navigating the tree
        while(!currentNode.isLeaf())
        {
            //Ask question
            System.out.println(currentNode.getData());
            gameMessage = currentNode.getData();

            //Get response
            yesNoResponse = getUserResponseGame(gameMessage);

```

```

//Update current node
if(yesNoResponse == 0)
{
    currentNode = currentNode.getLeftChild();
}
else
{
    currentNode = currentNode.getRightChild();
}
}

//We are at the leaf: got an answer that is either right or wrong
//Present guess to user, and store their answer
System.out.println("Is it a/an " + currentNode.getData() + "?");
String computerGuess = currentNode.getData();
//Get guess
gameMessage = "Is it a/an " + computerGuess + "?";
yesNoResponse = getUserResponseGame(gameMessage);
System.out.println("User response (0 = Yes. 1 = No): " + yesNoResponse + "\n");

//Compare answer to guess. Two possibilities to proceed:
//1. Answer is correct. Display options for user to continue
//1. Play again
//2. Store the tree
//3. Load a stored tree
//4. Quit
if(yesNoResponse == 0)
{
    endGameMenu(tree);
}
else if(yesNoResponse == 1)

```

```

{
    //2. The answer was wrong. We need to expand the tree.

    //Get the correct answer

    String questionMessage = "Well done!\nI couldn't guess your answer.\n\nWhat was the
correct answer?";

    String correctAnswer = JOptionPane.showInputDialog(null, questionMessage);

    //Get new question from user

    questionMessage = "Thank you.\nYou did great!\n";
    questionMessage += "\n\nPlease can you give me a question that will differentiate ";
    questionMessage += "between '" + computerGuess + "' and '" + correctAnswer + "'.\n\n";
    questionMessage += "\n\nAn example question would be: Is it usually found in the
Antarctic?";

    String userQuestion = JOptionPane.showInputDialog(null, questionMessage);

    //User's question gives what answer

    questionMessage = "Your question is: '" + userQuestion + "'\n\nIs this the right answer
for '" + computerGuess + "'?";

    int rightAnswerComputer = getUserResponseGame(questionMessage);

    //Print tree in text representation

    System.out.println("Before updating tree");
    printBinaryTree(tree);

    //Replace the current node with this question

    currentNode.setData(userQuestion);

    //Add left & right children with the answers

    BinaryNode<String> left = new BinaryNode<>(computerGuess, null, null);
    BinaryNode<String> right = new BinaryNode<>(correctAnswer, null, null);

```



```

//If the computer's guess is the answer to the user's question
if(rightAnswerComputer == 0)
{
    currentNode.setLeftChild(left);
    currentNode.setRightChild(right);
}
else
{
    //else the user's answer is the answer to the user's question
    currentNode.setLeftChild(right);
    currentNode.setRightChild(left);
}

//Thank you message
questionMessage = "Okay, thank you.\nYour tree has been updated.\nIf you save your
tree, you can load it at any time,";

questionMessage += " and your question and answer will be a part of the game!";
JOptionPane.showMessageDialog(null, questionMessage);

//Print tree in text representation
System.out.println("After updating tree");
printBinaryTree(tree);

//End of game menu
endGameMenu(tree);
}

}

}

```

```
}
```

```
public static BinaryTree<String> startMenu(BinaryTree<String> tree)
{
    //Get user response
    int userResponse = 0;
    String message = "Enter your response:\n\n1. Play\n2. Load a stored tree\n3. Quit";

    //Start Menu
    //The start menu is different from the end game menu
    // because there is no tree to save in the start menu,
    // hence they are separate.
    //Useful trick to be able to break out of this loop
    startMenuLoop:
    while (true)
    {
        //Get user response
        userResponse = getUserResponseMenu(message, 3);
        //-1. Invalid response
        //1. Play
        //2. Load a stored tree
        //3. Quit
        switch(userResponse)
        {
            case -1:
                //Try again
                System.out.println("Invalid response, please try again.");
                break;
            case 1:
                //Start the game
```

```

        break startMenuLoop;
    case 2:
        //Try to load tree
        tree = loadTree();
        if(tree == null)
        {
            System.out.println("Error loading tree");
        }
        else
        {
            System.out.println("Successfully loaded tree, Ready to play");
        }
        break;
    case 3:
        //Quit the program
        System.out.println("Thanks for playing");
        System.exit(1);
    }
}

return tree;
}

public static BinaryTree<String> endGameMenu(BinaryTree<String> tree)
{
    //Get user response
    int userResponse = 0;

    String message = "CONGRATS! YOU WON!\n\nYou can store the tree to play again later.\n\n\n";
    message += "\n\nEnter your response:\n\n1. Play\n2. Store the tree\n3. Load a stored tree\n4.
Quit";

```

```

//End of Game Menu

//The different menus could have different functionality in the future

//so for maintainability they are separated.

//This menu has an option to save.


//Useful trick to be able to break out of this loop
endGameMenuLoop:
while (true)
{
    //Get user response
    userResponse = getUserResponseMenu(message, 4);

    //-1. Invalid response
    //1. Play
    //2. Store the tree
    //3. Load a stored tree
    //4. Quit
    switch(userResponse)
    {
        case -1:
            //Try again
            System.out.println("Invalid response, please try again.");
            break;
        case 1:
            //Start the game
            break endGameMenuLoop;
        case 2:
            //Try to save the tree
            saveTree(tree);

            //No runtime exception means success
            System.out.println("Successfully saved the tree.");

```

```

        break;
    case 3:
        //Try to load the tree
        tree = loadTree();
        if(tree == null)
        {
            System.out.println("Error loading tree");
        }
        else
        {
            System.out.println("Successfully loaded tree, Ready to play");
        }
        break;
    case 4:
        //Quit the program
        System.out.println("Thanks for playing");
        System.exit(1);
    }
}

return tree;
}

public static int getUserResponseGame(String message)
{
    //Asks the user a question and accepts an input.
    //Will continue to ask the user the question until a valid response is given.
    //Returns a 0 for YES
    //Returns a 1 for NO

    boolean validInput = false;

```

```
int ret = -1;

//Ask the user the first time

String userInput = JOptionPane.showInputDialog(null, message);


//Update message for retries.
message += "\nSorry, that is an invalid input. Try again. Please enter only 'yes' or 'no'.";


while(!validInput)
{
    //Make the text lower case and check in lower case so that the user input is not case-sensitive
    userInput.toLowerCase();

    if(userInput.equals("y") || userInput.equals("yes"))
    {
        ret = 0;
        validInput = true;
    }
    else if(userInput.equals("n") || userInput.equals("no"))
    {
        ret = 1;
        validInput = true;
    }
    else
    {
        //User try again
        userInput = JOptionPane.showInputDialog(null, message);
    }
}

return ret;
}
```

```

public static int getUserResponseMenu(String message, int maximumValidOptions)
{
    //Returns the user response when prompted with a message
    // and given a number of valid options
    //Returns -1 for an invalid response
    int userResponse = Integer.parseInt(JOptionPane.showInputDialog(null, message));

    //Response must be from 1 to maximumValidOptions
    if (userResponse > maximumValidOptions || userResponse < 1)
    {
        userResponse = -1;
    }
    return userResponse;
}

```

```

public static BinaryTree<String> loadTree()
{
    BinaryTree<String> tree = null;

    try
    {
        File f = new File("save.txt");
        FileInputStream fis = new FileInputStream(f);
        ObjectInputStream ois = new ObjectInputStream(fis);
        tree = (BinaryTree<String>)ois.readObject();
        ois.close();
    }
    catch (Exception e)
    {
        throw new RuntimeException(e);
    }
}

```

```
}
```

```
return tree;
```

```
}
```

```
public static void saveTree(BinaryTree<String> tree)
```

```
{
```

```
    try
```

```
    {
```

```
        File f = new File("save.txt");
```

```
        FileOutputStream fos = new FileOutputStream(f);
```

```
        ObjectOutputStream oos = new ObjectOutputStream(fos);
```

```
        oos.writeObject(tree);
```

```
        oos.close();
```

```
    }
```

```
    catch (Exception e)
```

```
    {
```

```
        throw new RuntimeException(e);
```

```
    }
```

```
}
```

```
public static void printBinaryTree(BinaryTree<String> tree)
```

```
{
```

```
    //TLDR; The level indicator only works correctly when for the initial tree.
```

```
    //Note: My print function correctly performs a breadth-first traversal, displaying
```

```
    // the data in each node in the correct order every time.
```

```
    // I also added code which prints out "Level _". When the tree is not full, this "Level _"
    functionality
```

```
    // acts unexpectedly. This is because the height of the two nodes on the same level could be
    different
```

```
    //in a non-full tree.
```



```

System.out.println("\n\nPrinting my binary tree");

//Breadth-first traversal
//Create an ArrayQueue
ArrayQueue q = new ArrayQueue();
//Put the root into the queue
q.enqueue(tree.getRootNode());
BinaryNode currNode = null;
//Variables for finding when the level changes
int prevLevel = 0;
int currLevel = 1;
int treeHeight = tree.getHeight();
while(!q.isEmpty())
{
    //Dequeue the front node
    currNode = (BinaryNode)q.dequeue();

    //Get the current node's level
    currLevel = treeHeight + 1 - currNode.getHeight();

    //Print the current level of the tree
    if(prevLevel != currLevel)
    {
        System.out.println("-----");
        System.out.println("Level: " + currLevel);
        prevLevel = currLevel;
        //The level won't be printed again until current level changes
    }

    //Enqueue the current node's children, if it has any
    if(currNode.hasLeftChild())

```

```

    {
        q.enqueue(currNode.getLeftChild());
    }
    if(currNode.hasRightChild())
    {
        q.enqueue(currNode.getRightChild());
    }

    //Visit the current node
    //Check if the node is a leaf node. Leaves are guesses
    //Output the contents of the node
    if(currNode.isLeaf())
    {
        System.out.println("Guess: " + currNode.getData() + "\t");
    }
    else
    {
        System.out.println("Question: " + currNode.getData() + "\t");
    }
}
System.out.println("-----");
System.out.println("End of tree\n");

}

```

```

public static void createInitialTree(BinaryTree<String> tree)
{
    //When creating a tree,
    // I will create each subtree starting from the leaves,
    // Keep linking subtrees until I reach the root

```

```

//Create leaves

//The leaves are the guesses

//H - O are the guesses, They are the leaf nodes of the initial BinaryTree.

//Level 4 (assuming the root node of the tree we are building is level 1)

BinaryTree<String> hTree = new BinaryTree<String>("H");
BinaryTree<String> iTree = new BinaryTree<String>("I");
BinaryTree<String> jTree = new BinaryTree<String>("penguin");
BinaryTree<String> kTree = new BinaryTree<String>("K");
BinaryTree<String> lTree = new BinaryTree<String>("L");
BinaryTree<String> mTree = new BinaryTree<String>("M");
BinaryTree<String> nTree = new BinaryTree<String>("N");
BinaryTree<String> oTree = new BinaryTree<String>("O");


//Create the subtrees that join the leaves

//Level 3

BinaryTree<String> dTree = new BinaryTree<>("D", hTree, iTree);
BinaryTree<String> eTree = new BinaryTree<>("E: Is it a bird?", jTree, kTree);
BinaryTree<String> fTree = new BinaryTree<>("F", lTree, mTree);
BinaryTree<String> gTree = new BinaryTree<>("G", nTree, oTree);


//Level 2

BinaryTree<String> bTree = new BinaryTree<>("B: Is it a mammal?", dTree, eTree);
BinaryTree<String> cTree = new BinaryTree<>("C", fTree, gTree);


//Level 1 - Root node

tree.setTree("A: Are you thinking of an animal?", bTree, cTree);
}
}

```

Testing

Printing a tree

Initial tree - Letters are placeholder data.

```
Printing my binary tree
-----
Level: 1
Question: A: Are you thinking of an animal?
-----
Level: 2
Question: B: Is it a mammal?
Question: C
-----
Level: 3
Question: D
Question: E: Is it a bird?
Question: F
Question: G
-----
Level: 4
Guess: H
Guess: I
Guess: penguin
Guess: K
Guess: L
Guess: M
Guess: N
Guess: O
-----
End of tree
```

Tree after adding some new data from playing:

Printing my binary tree

Level: 1

Question: A: Are you thinking of an animal?

Level: 2

Question: B: Is it a mammal?

Level: 5

Question: C

Level: 6

Question: D

Level: 3

Question: E: Is it a bird?

Level: 6

Question: F

Question: G

Level: 7

Guess: H

Guess: I

Level: 4

Question: Does it normally live in the Savannah?

Level: 7

Guess: K

Guess: L

Guess: M

Guess: N

Guess: O

Level: 5

Question: Does it's name start with an 'e'?

Level: 7

Guess: penguin

Level: 6

Question: does it's name end with a u?

Level: 7

Guess: Ostrich

Guess: emu

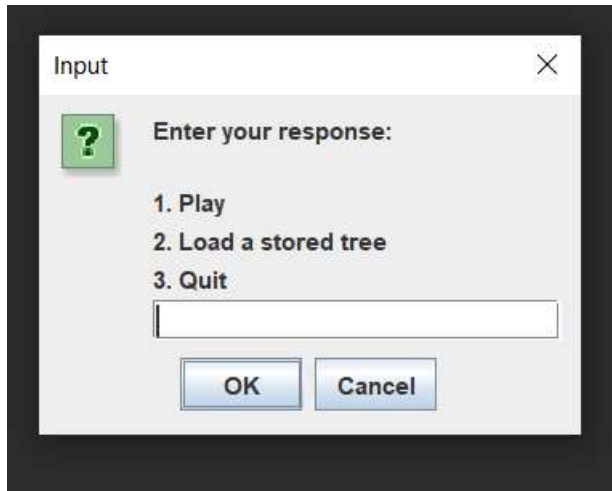
Guess: emuious

End of tree

The order that the Questions and Guesses are displayed is correct.

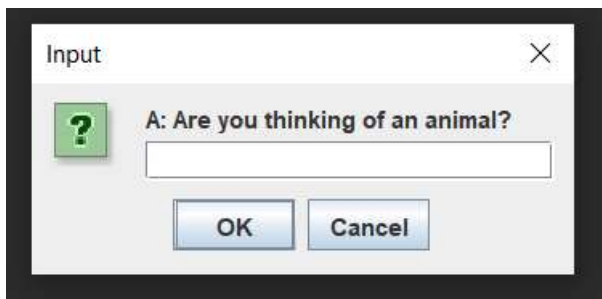
The Level number's are not the correct values.

Start Menu

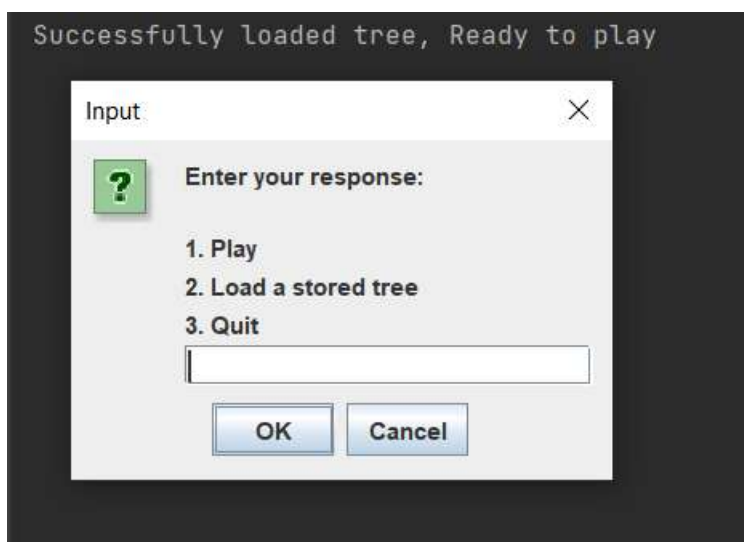


Valid input: 1, 2, 3: Works.

1: Starts the game



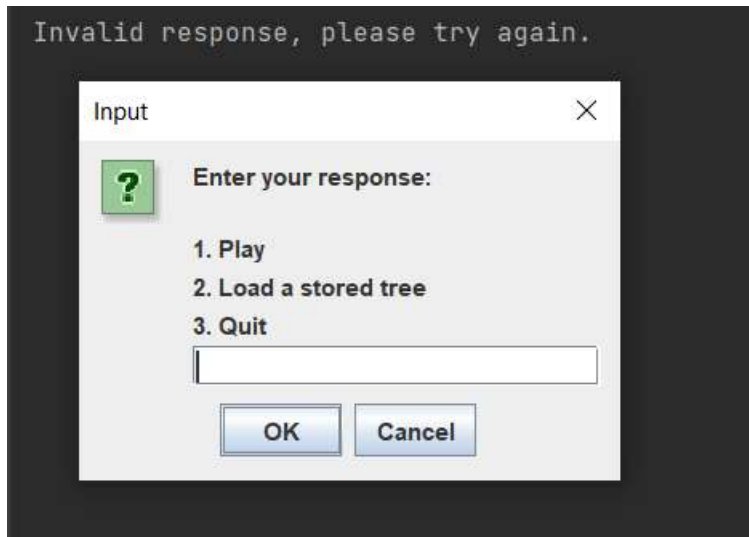
2: Loads a tree from storage



3: Quits the game

```
Thanks for playing  
  
Process finished with exit code 1
```

Invalid input: 4 : Shows message in console and allows the user to retry.



Invalid input: a : program crashes

```
Exception in thread "main" java.lang.NumberFormatException: Create breakpoint : For input string: "a"
```

End of game Menu

Valid input 1, 2, 3, 4 Works. 1, 3, 4 are the same as above.

2: Saves the tree to storage



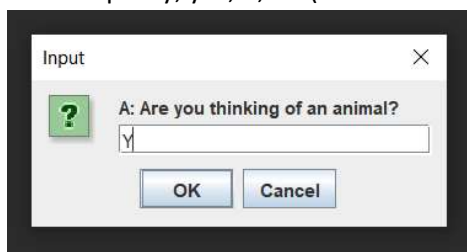
Invalid input: anything higher than 4: Shows message in console and allows the user to retry.

Invalid input: a : program crashes

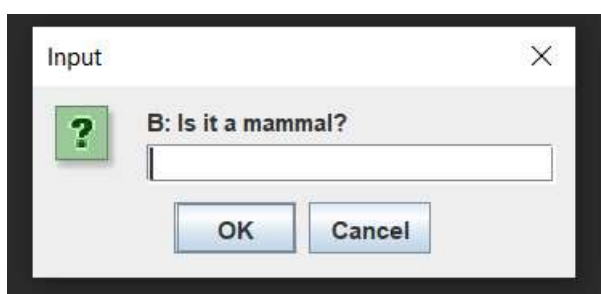
```
Exception in thread "main" java.lang.NumberFormatException: Create breakpoint : For input string: "a"
```

User input during playing the game

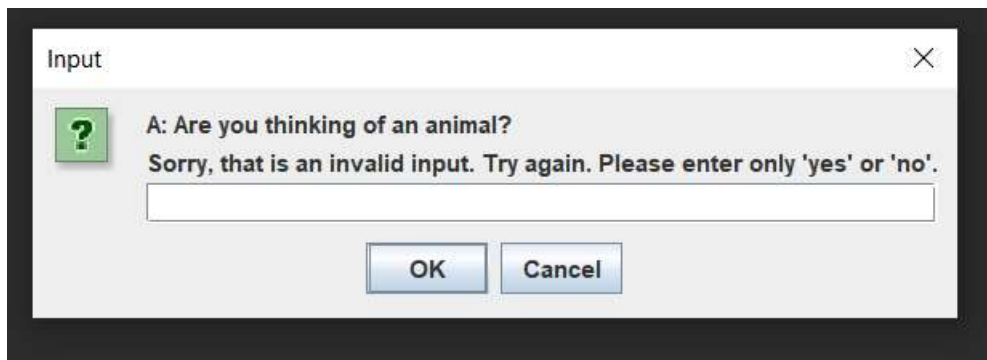
Valid input: y, yes, n, no (case insensitive)



Y: progresses to the next question (this is expected behaviour)

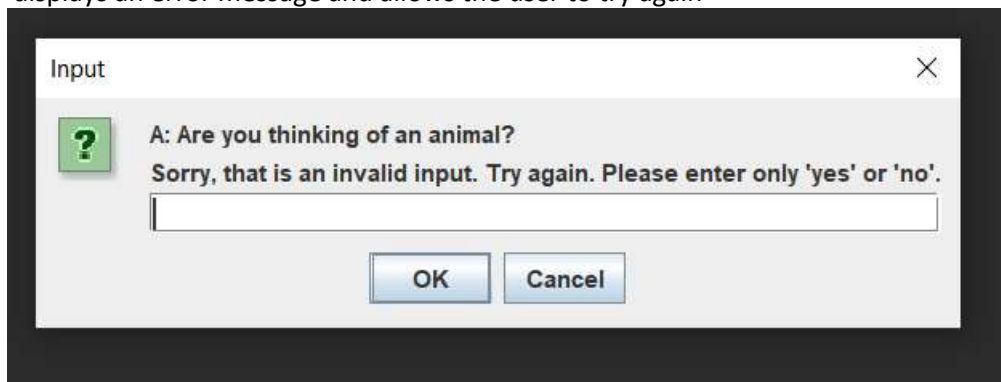


Invalid input: g : displays an error message and allows the user to try again

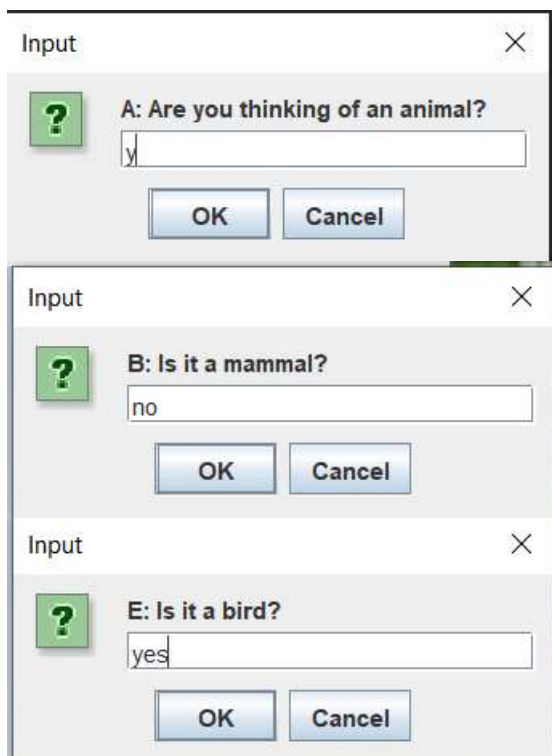


Invalid input: 11

displays an error message and allows the user to try again



Playing the game:



Input

?

Is it a/an penguin?

no

OK Cancel

Input

?

Well done!
I couldn't guess your answer.

What was the correct answer?

Ostrich

OK Cancel

Input

?

Thank you.
You did great!

Please can you give me a question that will differentiate between 'penguin' and 'Ostrich'.

An example question would be: Is it usually found in the Antarctic?

Does it normally live in the Savannah?

OK Cancel

Input

?

Your question is: 'Does it normally live in the Savannah?'

Is this the right answer for 'penguin'?

no

OK Cancel

Message

i

Okay, thank you.
Your tree has been updated.
If you save your tree, you can load it at any time, and your question and answer will be a part of the game!

OK

Input ×

 **CONGRATS! YOU WON!**

You can store the tree to play again later.

Enter your response:

1. Play
2. Store the tree
3. Load a stored tree
4. Quit

Now to see if the tree updated properly. Let's play again.

Input ×

 **CONGRATS! YOU WON!**


You can store the tree to play again later.

Enter your response:

1. Play
2. Store the tree
3. Load a stored tree
4. Quit


Enter 1 here

Input ×

 **A: Are you thinking of an animal?**

yes

Input ×

 **B: Is it a mammal?**

no

Input ✕

? E: Is it a bird?

yes

OK Cancel

Input ✕

? Does it normally live in the Savannah?

yes

OK Cancel

Yes, the tree has been successfully updated.

Input ✕

? Is it a/an Ostrich?

yes

OK Cancel

Ostrich was my answer last time. The tree was updated.

Input ✕

? CONGRATS! YOU WON!

You can store the tree to play again later.

Enter your response:

1. Play
2. Store the tree
3. Load a stored tree
4. Quit

OK Cancel

References

Blackboard code implementations used:

BinaryNode.java

BinaryNodeInterface.java

BinaryTree.java

BinaryTreeInterface.java

TreeInterface.java

ArrayQueue.java

Queue.java

For object serialization:

<https://stackoverflow.com/questions/17293991/how-to-write-and-read-java-serialized-objects-into-a-file>

For depth-first traversal:

Topic 5 - Extra Notes.pdf was used to learn the pseudocode to perform a depth-first traversal.