**Implementation of animal game in java**

The program starts by asking user question containing in the root node. If the response is “no” by the user then the program will follow the left child branch and ask question in the next node, if the response is yes then it will follow the right child branch. This process is repeated with each new node. The leaf node contains sequence of answers that lead from root to leaf. Once the program reaches leaf then it guesses that the user might be thinking of leaf’s object if the leaf’s object matches the user’s object it wins .In case the object do not match the program will learn by asking user yes/no questions which will distinguish the user’s object from leaf object after that the program will place user’s question in appropriate location in the tree.

**Code**

import java.util.Scanner;

public class labOne

{

private static Scanner stdin = new Scanner(System.in);

public static void main(String[ ] args)

{

BTNode<String> root;

instruct( );

root = beginningTree( );

do

play(root);

while (query("Shall we play again?"));

System.out.println("Thanks for teaching me a thing or two.");

}

public static void instruct( )

{

System.out.println("Please think of an animal.");

System.out.println("I will ask some yes/no questions to try to figure");

System.out.println("out what you are.");

}

public static void play(BTNode<String> current)

{

while (!current.isLeaf( ))

{

if (query(current.getData( )))

current = current.getLeft( );

else

current = current.getRight( );

}

System.out.print("My guess is " + current.getData( ) + ". ");

if (!query("Am I right?"))

learn(current);

else

System.out.println("I knew it all along!");

}

public static BTNode<String> beginningTree( )

{

BTNode<String> root;

BTNode<String> child;

final String ROOT\_QUESTION = "Are you a mammal?";

final String LEFT\_QUESTION = "Are you bigger than a cat?";

final String RIGHT\_QUESTION = "Do you live underwater?";

final String ANIMAL1 = "Kangaroo";

final String ANIMAL2 = "Mouse";

final String ANIMAL3 = "Trout";

final String ANIMAL4 = "Robin";

// It creates the root node with the question

root = new BTNode<String>(ROOT\_QUESTION, null, null);

// It creates and attach the left subtree.

child = new BTNode<String>(LEFT\_QUESTION, null, null);

child.setLeft(new BTNode<String>(ANIMAL1, null, null));

child.setRight(new BTNode<String>(ANIMAL2, null, null));

root.setLeft(child);

// It creates and attach the right subtree.

child = new BTNode<String>(RIGHT\_QUESTION, null, null);

child.setLeft(new BTNode<String>(ANIMAL3, null, null));

child.setRight(new BTNode<String>(ANIMAL4, null, null));

root.setRight(child);

return root;

}

public static void learn(BTNode<String> current)

// Precondition: current is a reference to a leaf in a taxonomy tree. This

// leaf contains a wrong guess that was just made.

// Postcondition: Information has been elicited from the user, and the tree

// has been improved.

{

String guessAnimal; // The animal that was just guessed

String correctAnimal; // The animal that the user was thinking of

String newQuestion; // A question to distinguish the two animals

// Sets Strings for the guessed animal, correct animal and a new question.

guessAnimal = current.getData( );

System.out.println("I give up. What are you? ");

correctAnimal = stdin.nextLine( );

System.out.println("Please type a yes/no question that will distinguish a");

System.out.println(correctAnimal + " from a " + guessAnimal + ".");

newQuestion = stdin.nextLine( );

// Puts the new question in the current node, and add two new children.

current.setData(newQuestion);

System.out.println("As a " + correctAnimal + ", " + newQuestion);

if (query("Please answer"))

{

current.setLeft(new BTNode<String>(correctAnimal, null, null));

current.setRight(new BTNode<String>(guessAnimal, null, null));

}

else

{

current.setLeft(new BTNode<String>(guessAnimal, null, null));

current.setRight(new BTNode<String>(correctAnimal, null, null));

}

}

public static boolean query(String prompt)

{

String answer;

System.out.print(prompt + " [Y or N]: ");

answer = stdin.nextLine( ).toUpperCase( );

while (!answer.startsWith("Y") && !answer.startsWith("N"))

{

System.out.print("Invalid response. Please type Y or N: ");

answer = stdin.nextLine( ).toUpperCase( );

}

return answer.startsWith("Y");

}

public static class BTNode<E>

{

private E data;

private BTNode<E> left, right;

public BTNode(E initialData, BTNode<E> initialLeft, BTNode<E> initialRight)

{

data = initialData;

left = initialLeft;

right = initialRight;

}

public E getData( )

{

return data;

}

public BTNode<E> getLeft( )

{

return left;

}

public E getLeftmostData( )

{

if (left == null)

return data;

else

return left.getLeftmostData( );

}

public BTNode<E> getRight( )

{

return right;

}

public E getRightmostData( )

{

if (left == null)

return data;

else

return left.getRightmostData( );

}

public void inorderPrint( )

{

if (left != null)

left.inorderPrint( );

System.out.println(data);

if (right != null)

right.inorderPrint( );

}

public boolean isLeaf( )

{

return (left == null) && (right == null);

}

public void preorderPrint( )

{

System.out.println(data);

if (left != null)

left.preorderPrint( );

if (right != null)

right.preorderPrint( );

}

public void postorderPrint( )

{

if (left != null)

left.postorderPrint( );

if (right != null)

right.postorderPrint( );

System.out.println(data);

}

public void print(int depth)

{

int i;

// Print the indentation and the data from the current node:

for (i = 1; i <= depth; i++)

System.out.print(" ");

System.out.println(data);

// Print the left subtree (or a dash if there is a right child and no left child)

if (left != null)

left.print(depth+1);

else if (right != null)

{

for (i = 1; i <= depth+1; i++)

System.out.print(" ");

System.out.println("--");

}

// Print the right subtree (or a dash if there is a left child and no left child)

if (right != null)

right.print(depth+1);

else if (left != null)

{

for (i = 1; i <= depth+1; i++)

System.out.print(" ");

System.out.println("--");

}

}

public BTNode<E> removeLeftmost( )

{

if (left == null)

return right;

else

{

left = left.removeLeftmost( );

return this;

}

}

public BTNode<E> removeRightmost( )

{

if (right == null)

return left;

else

{

right = right.removeRightmost( );

return this;

}

}

public void setData(E newData)

{

data = newData;

}

public void setLeft(BTNode<E> newLeft)

{

left = newLeft;

}

public void setRight(BTNode<E> newRight)

{

right = newRight;

}

public static <E> BTNode<E> treeCopy(BTNode<E> source)

{

BTNode<E> leftCopy, rightCopy;

if (source == null)

return null;

else

{

leftCopy = treeCopy(source.left);

rightCopy = treeCopy(source.right);

return new BTNode<E>(source.data, leftCopy, rightCopy);

}

}

public static <E> long treeSize(BTNode<E> root)

{

if (root == null)

return 0;

else

return 1 + treeSize(root.left) + treeSize(root.right);

}

}

}

**Output :**

