**Program:**

***Decision tree App class***

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class DecisionTreeApp {

static BufferedReader *keyboardInput* = new

BufferedReader(new InputStreamReader(System.*in*));

static DecisionTree *newTree*;

public static void main(String[] args) throws IOException {

*newTree* = new DecisionTree();

*generateTree*();

System.*out*.println("\nOUTPUT DECISION TREE");

System.*out*.println("====================");

*newTree*.outputBinTree();

*queryTree*();

}

static void generateTree() {

System.*out*.println("\nGENERATE DECISION TREE");

System.*out*.println("======================");

*newTree*.createRoot(1,"Does animal eat meat?");

*newTree*.addYesNode(1,2,"Does animal have stripes?");

*newTree*.addNoNode(1,3,"Does animal have stripes?");

*newTree*.addYesNode(2,4,"Animal is a Tiger");

*newTree*.addNoNode(2,5,"Animal is a Leopard");

*newTree*.addYesNode(3,6,"Animal is a Zebra");

*newTree*.addNoNode(3,7,"Animal is a Horse");

}

static void queryTree() throws IOException {

System.*out*.println("\nQUERY DECISION TREE");

System.*out*.println("===================");

*newTree*.queryBinTree();

*optionToExit*();

}

static void optionToExit() throws IOException {

System.*out*.println("Exit? (enter \"Yes\" or \"No\")");

String answer = *keyboardInput*.readLine();

if (answer.equals("Yes")) return;

else {

if (answer.equals("No")) *queryTree*();

else {

System.*out*.println("ERROR: Must answer \"Yes\" or \"No\"");

*optionToExit*();

}

}

}

}

***Decision Tree class***

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

class DecisionTree {

private class BinTree {

private int nodeID;

private String questOrAns = null;

private BinTree yesBranch = null;

private BinTree noBranch = null;

public BinTree(int newNodeID, String newQuestAns) {

nodeID = newNodeID;

questOrAns = newQuestAns;

}

}

static BufferedReader *keyboardInput* = new

BufferedReader(new InputStreamReader(System.*in*));

BinTree rootNode = null;

public DecisionTree() {

}

public void createRoot(int newNodeID, String newQuestAns) {

rootNode = new BinTree(newNodeID,newQuestAns);

System.*out*.println("Created root node " + newNodeID);

}

public void addYesNode(int existingNodeID, int newNodeID, String newQuestAns) {

if (rootNode == null) {

System.*out*.println("ERROR: No root node!");

return;

}

if (searchTreeAndAddYesNode(rootNode,existingNodeID,newNodeID,newQuestAns)) {

System.*out*.println("Added node " + newNodeID +

" onto \"yes\" branch of node " + existingNodeID);

}

else System.*out*.println("Node " + existingNodeID + " not found");

}

private boolean searchTreeAndAddYesNode(BinTree currentNode,

int existingNodeID, int newNodeID, String newQuestAns) {

if (currentNode.nodeID == existingNodeID) {

if (currentNode.yesBranch == null) currentNode.yesBranch = new

BinTree(newNodeID,newQuestAns);

else {

System.*out*.println("WARNING: Overwriting previous node " +

"(id = " + currentNode.yesBranch.nodeID +

") linked to yes branch of node " +

existingNodeID);

currentNode.yesBranch = new BinTree(newNodeID,newQuestAns);

}

return(true);

}

else {

if (currentNode.yesBranch != null) {

if (searchTreeAndAddYesNode(currentNode.yesBranch,

existingNodeID,newNodeID,newQuestAns)) {

return(true);

}

else {

if (currentNode.noBranch != null) {

return(searchTreeAndAddYesNode(currentNode.noBranch,

existingNodeID,newNodeID,newQuestAns));

}

else return(false);

}

}

return(false);

}

}

public void addNoNode(int existingNodeID, int newNodeID, String newQuestAns) {

if (rootNode == null) {

System.*out*.println("ERROR: No root node!");

return;

}

if (searchTreeAndAddNoNode(rootNode,existingNodeID,newNodeID,newQuestAns)) {

System.*out*.println("Added node " + newNodeID +

" onto \"no\" branch of node " + existingNodeID);

}

else System.*out*.println("Node " + existingNodeID + " not found");

}

private boolean searchTreeAndAddNoNode(BinTree currentNode,

int existingNodeID, int newNodeID, String newQuestAns) {

if (currentNode.nodeID == existingNodeID) {

if (currentNode.noBranch == null) currentNode.noBranch = new

BinTree(newNodeID,newQuestAns);

else {

System.*out*.println("WARNING: Overwriting previous node " +

"(id = " + currentNode.noBranch.nodeID +

") linked to yes branch of node " +

existingNodeID);

currentNode.noBranch = new BinTree(newNodeID,newQuestAns);

}

return(true);

}

else {

if (currentNode.yesBranch != null) {

if (searchTreeAndAddNoNode(currentNode.yesBranch,

existingNodeID,newNodeID,newQuestAns)) {

return(true);

}

else {

if (currentNode.noBranch != null) {

return(searchTreeAndAddNoNode(currentNode.noBranch,

existingNodeID,newNodeID,newQuestAns));

}

else return(false);

}

}

else return(false);

}

}

public void queryBinTree() throws IOException {

queryBinTree(rootNode);

}

private void queryBinTree(BinTree currentNode) throws IOException {

if (currentNode.yesBranch==null) {

if (currentNode.noBranch==null) System.*out*.println(currentNode.questOrAns);

else System.*out*.println("Error: Missing \"Yes\" branch at \"" +

currentNode.questOrAns + "\" question");

return;

}

if (currentNode.noBranch==null) {

System.*out*.println("Error: Missing \"No\" branch at \"" +

currentNode.questOrAns + "\" question");

return;

}

askQuestion(currentNode);

}

private void askQuestion(BinTree currentNode) throws IOException {

System.*out*.println(currentNode.questOrAns + " (enter \"Yes\" or \"No\")");

String answer = *keyboardInput*.readLine();

if (answer.equals("Yes")) queryBinTree(currentNode.yesBranch);

else {

if (answer.equals("No")) queryBinTree(currentNode.noBranch);

else {

System.*out*.println("ERROR: Must answer \"Yes\" or \"No\"");

askQuestion(currentNode);

}

}

}

public void outputBinTree() {

outputBinTree("1",rootNode);

}

private void outputBinTree(String tag, BinTree currentNode) {

if (currentNode == null) return;

System.*out*.println("[" + tag + "] nodeID = " + currentNode.nodeID +

", question/answer = " + currentNode.questOrAns);

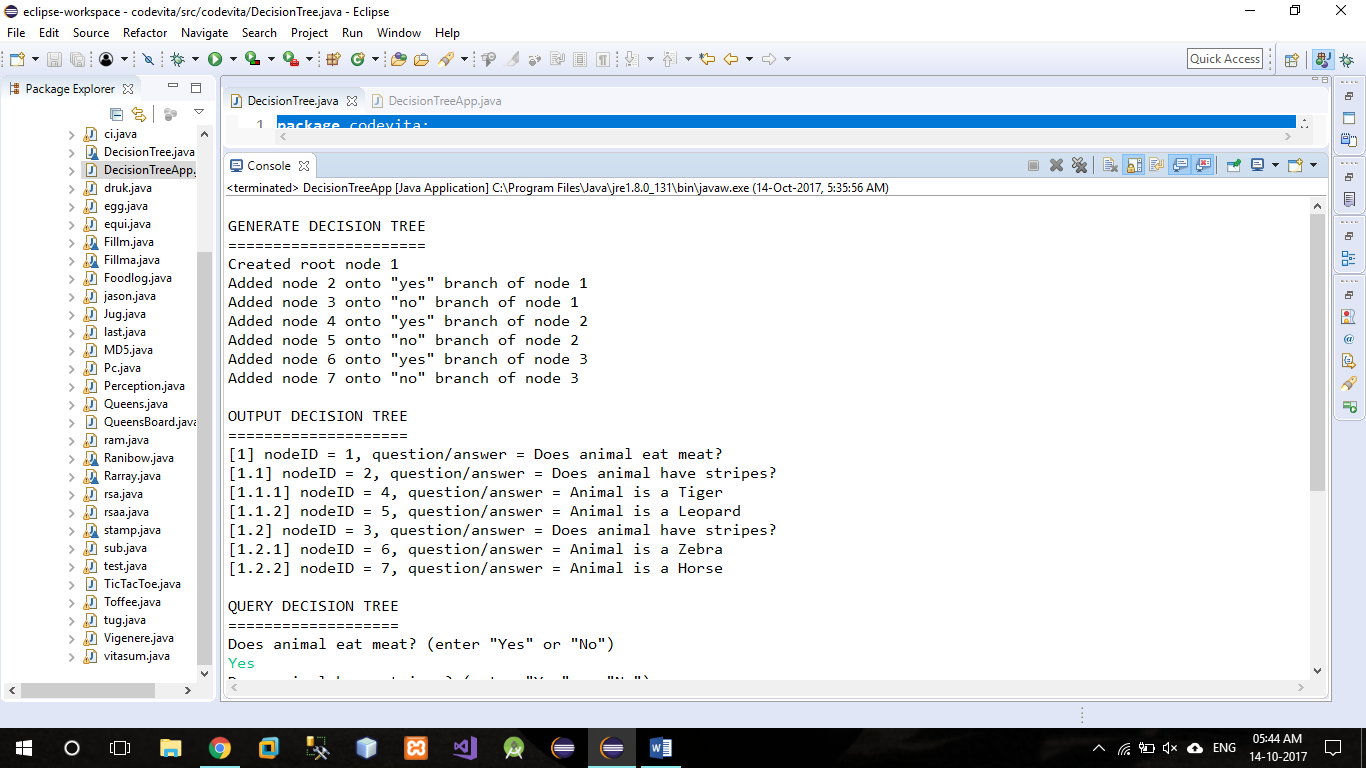
outputBinTree(tag + ".1",currentNode.yesBranch);

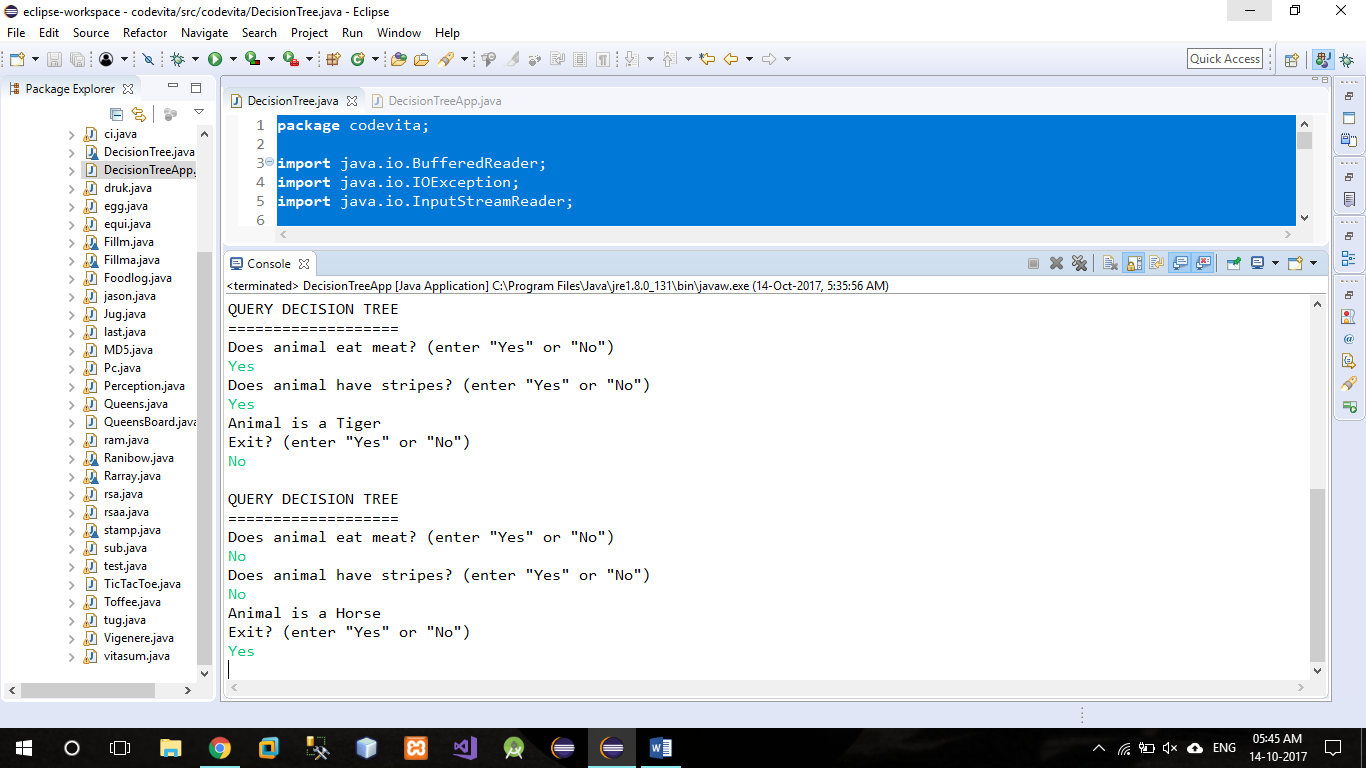
outputBinTree(tag + ".2",currentNode.noBranch);

}

}

**Output:**





**Program:**

#include<stdio.h>

int no\_of\_pred;

int no\_of\_arg[10];

int i,j;

char nouse;

char predicate[10];

char argument[10][10];

void unify();

void display();

void chk\_arg\_pred();

void main()

{

printf("\t=========PROGRAM FOR UNIFICATION=========\n");

printf("\nEnter Number of Predicates:- [ ]\b\b");

scanf("%d",&no\_of\_pred);

for(i=0;i<no\_of\_pred;i++)

{

scanf("%c",&nouse); //to accept "Enter" as a character

printf("\nEnter Predicate %d:-[ ]\b\b",i+1);

scanf("%c",&predicate[i]);

printf("\n\tEnter No.of Arguments for Predicate %c:-[ ]\b\b",predicate[i]);

scanf("%d",&no\_of\_arg[i]);

for(j=0;j<no\_of\_arg[i];j++)

{

scanf("%c",&nouse);

printf("\n\tEnter argument %d:( )\b\b",j+1);

scanf("%c",&argument[i][j]);

}

}

display();

chk\_arg\_pred();

printf("\n");

}

void display()

{

printf("\n\t=======PREDICATES ARE======");

for(i=0;i<no\_of\_pred;i++)

{

printf("\n\t%c(",predicate[i]);

for(j=0;j<no\_of\_arg[i];j++)

{

printf("%c",argument[i][j]);

if(j!=no\_of\_arg[i]-1)

printf(",");

}

printf(")");

}

}

void chk\_arg\_pred()

{

int pred\_flag=0;

int arg\_flag=0;

for(i=0;i<no\_of\_pred-1;i++)

{

if(predicate[i]!=predicate[i+1])

{

printf("\nPredicates not same..");

printf("\nUnification cannot progress!");

pred\_flag=1;

break;

}

}

/\*=====Chking No of Arguments====\*/

if(pred\_flag!=1)

{

for(i=0;i<no\_of\_arg[i]-1;i++)

{

if(no\_of\_arg[i]!=no\_of\_arg[i+1])

{

printf("\nArguments Not Same..!");

arg\_flag=1;

break;

}

}

}

if(arg\_flag==0&&pred\_flag!=1)

unify();

}

void unify()

{

int flag=0;

for(i=0;i<no\_of\_pred-1;i++)

{

for(j=0;j<no\_of\_arg[i];j++)

{

if(argument[i][j]!=argument[i+1][j])

{

if(flag==0)

printf("\n\t======SUBSTITUTION IS======");

printf("\n\t%c/%c",argument[i+1][j],argument[i][j]);

flag++;

}

}

}

if(flag==0)

{ printf("\nArguments are Identical...");

printf("\nNo need of Substitution\n");

}

}

**Output:**

