**A MINI PROJECT**

On

**GUI FOR TOWERS OF HANOI IN JAVA**

(Submitted for partial fulfillment of the requirements for the award of the degree)

**BE 2/4(CSE) MINI PROJECT**

BY

**CH.LIKHITHA (160116733068)**

**G.SAI CHANDANA (160116733079)**

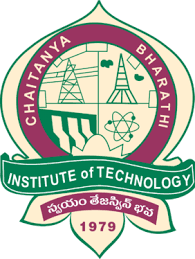
Under the guidance of

**Sri.K.RAMADASU**

Associate Professor

Department of CSE

CBIT, Hyderabad



**Department of Computer Science and Engineering**

**Chaitanya Bharathi Institute of Technology**

**Gandipet, Hyderabad-500075**

**CERTIFICATE**

This is to certify that the project work entitled **“GUI FOR TOWERS OF HANOI USING JAVA”**is bonafide work carried out by **CH.LIKHITHA(160116733068) AND G.SAI CHANDANA(160116733079)** in fulfillment of the award of degree of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING** by the **OSMANIA UNIVERSITY**, Hyderabad, under our guidance and supervision.

The results enclosed in this report are not submitted to any university or organization for the award of any degree or diploma.

**Under the guidance of**

**Sri.K.RAMADASU**  Dr.M.SWAMYDAS

Associate professor Head of the Department

Department of CSE Department of CSE

CBIT, Hyderabad. CBIT,Hyderabad.

**DECLARATION**

This is to certify that the work reported in the present project entitled **“GUI FOR TOWERS OF HANOI USING JAVA”** is a record of work done by us in the Department of Computer Science and Engineering, Chaitanya Bharathi Institute of Technology, Osmania University. The reports are based on the project work done entirely by us and not copied from any source.

**CH.LIKHITHA**

**(160116733068)**

**G.SAI CHANDANA**

**(160116733079)**

**ACKNOWLEDGEMENTS**

We would like to express our heart felt appreciation and gratitude to **Sri.K.RAMADASU**, Assistant Professor, Department of CSE, our project guides ,for their skillful guidance, constant supervision , timely suggestion, keen interest and encouragement in completing the project with in the stipulated time.

We express our profound sense of gratitude to **Dr.M.SWAMYDAS,** Head Of the Department, CSE, who has served as a host of valuable corrections and for providing us time to complete this project.

We gratefully express our thanks to **Dr.P.RAVINDER REDDY,** Principal of our college and the management of **“CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY”** for providing excellent academic and learning environment in the college.

We wish to express our heartfelt gratitude to the **Members of staff** and all others who helped us bringing up our project .We would also like to think the **Lab assistants and programmers** helping us through our project.

**ABSTRACT**

**GUI FOR TOWERS OF HANOI**

Tower of Hanoi is a mathematical puzzle with three rods and ‘n’ numbers of discs; the puzzle was invented by the French mathematician Edouard Lucas in 1883.

The objective of this puzzle is to transfer the entire stack to another rod.

The basic trick behind the solution of Tower of Hanoi puzzle is to break the problem down into a number of smaller problems and further break these into even smaller ones, so that it is made a typical best suited problem for the application of recursive function.

**BY:**

CH.LIKHITHA(160116733068)

G.SAI CHANDANA(160116733079)

**TABLE OF CONTENTS**

**Name of Topic Page No**

Certificate 2

Declaration 3

Acknowledgements 4

Abstract 5

1. Introduction 7

2.Requirements 8

* 1. Hardware Requirements
  2. Software Requirements

3.Design of Project 9

4.Rules of Tower of Hanoi 10

5.Solution of the puzzle Towers of Hanoi 11

6.Towers of Hanoi -Building process 12

7.Recursive algorithm 16

8.The solve method : solve() 17

9.Implementation 19

10.Output 31

11.Conclusion 36

12.References 37

.

**INTRODUCTION**

Towers of Hanoi is an application in which the power of recursion is demonstrated.

Here,in this project we present you a graphical user interface of the operations done in Towers of Hanoi.

There are three poles a

nd 7 discs of different sizes.

Initially, all the discs are placed on the first pole with the largest disc at the bottom and the smallest one at the top.

We need to move all the discs from the first pole to the third pole, with the smallest disc at the top and the largest at the bottom.

We can move only one disc at a time (which should be the topmost disc) and at any point of time, a larger disc cannot be placed over a smaller one i.e. all the discs on a pole must be placed in such a way that the smallest is at the top and the largest at the bottom. The second pole can be used as an intermediate pole to help us in transferring the discs.

**REQUIREMENTS**

**Hardware Requirements:**

* 32 or 64-bit processor
* sufficient RAM to run the application

**Software Requirements:**

* JDK SE-8
* Text Editor(Notepad)
* NetBeans IDE-1.8
* Support for Graphical User Interafaces

**DESIGN OF PROJECT**

The proposed project is implemented using “Graphical User Interface”and is implemented as an applet.

AWT components are used for portraying the stepwise operations in building Towers of Hanoi.

Concepts of threads is used along with graphics in order to animate the project.

The movement of discs is governed by the actions of threads assigned to the respective discs.

The building of a tower is shown by the logical movement of discs according to the proposed rules of the game.

The implementation starts with a fully built tower on the first rod followed by the movement of discs to build a tower on the third rod.

When we close the applet the building process stops.

When we open the applet again the building process starts from the first.

**RULES OF TOWERS OF HANOI**

* Only a single disc is allowed to be transferred at a time.
* Each transfer or move should consists of taking the upper disk from one of the stack and then placing it on the top of another stack i.e. only a top most disk on the stack can be moved.
* Larger disk cannot be placed over smaller disk; placing of disks should be in increasing order.

 The minimal number of moves required to solve a Tower of Hanoi puzzle is 2n − 1, where n is the number of disks.

**SOLUTION TO THE PUZZLE-TOWERS OF HANOI**

This puzzle can be solved using recursion.

For a moment, assume that there are just two discs (disc 1 and 2; disc 2 being the larger one) on the first pole.

The solution consists of three steps.

Step 1: Move disc 1 from pole 1 to pole 2.

Step 2: Move disc 2 from pole 1 to pole 3.

Step 3: Move disc 1 from pole 1 to pole 3.

Now, assume that disc 1 is not a single disc but a collection of discs.

The procedure would be similar to the above three steps, but steps 1 and 3 would be a collection of steps.

Step 1 would be to move the n-1 discs (assuming that there were a total of n discs) from pole 1 to pole 2.

For moving these (n -1) discs, we again follow the same strategy of considering them as 1 disc plus a set of (n-2) discs.

Step 3 would also be similar.This gives us the recursive solution.

**TOWERS OF HANOI-BUILDING PROCESS**

In the following pages,the basic operations that take place for building the Towers of Hanoi are shown pictorially stepwise.

**INITIAL STATE**:

**STEP-I:**

**STEP-II:**

**STEP-III:**

**STEP-IV:**

****

**STEP-V:**

****

**STEP-VI:**

****

**FINAL STATE:**

****

**RECURSIVE ALGORITHM**

The recursive solution to move n discs from the start pole to the end pole using an auxiliary pole is given below.

**Base Case**- When n = 1  
Move the disc from start pole to end pole

**Recursive Case** - When n > 1

**Step 1**: Move (n-1) discs from start pole to auxiliary pole.

**Step 2**: Move the last disc from start pole to end pole.

**Step 3**: Move the (n-1) discs from auxiliary pole to end pole.  
Steps 1 and 3 are recursive invocations of the same procedure.

**THE SOLVE METHOD:Solve()**

The solve method is the backbone of the algorithm as it solely depends upon this method for any recursive functions that constitute a major part of the algorithm.

private void solve(int disks, int from, int to, int spare)

{

if (disks == 1)

moveOne(from,to);

else {

solve(disks-1, from, spare, to);

moveOne(from,to);

solve(disks-1, spare, to, from);

}

}

The method solve() takes four arguments :

disks - the number of discs in the puzzle.

from, to , spare - the names of the three poles which will be used for printing the solution

We first check if the number of poles, n is equal to one.

If so, the base case solution will be used which consists of moving a disc from the start peg to the end peg.

If not, the recursive solution is used which consists of two recursive calls to the same procedure solve().

When we need to move n-1 discs from the start pole to the auxiliary pole, the auxiliary pole becomes the end pole and the end pole becomes the auxiliary pole.

That is why we have written

solve(disks - 1, from, to, spare)

Next we print ' start -> end ' which corresponds to moving the largest disc at the bottom from the “from” peg to the “to” peg.

Finally, we have recursive invocation of solve(). Here, the spare peg becomes the “from” peg and the “from” peg becomes the spare peg.

**IMPLEMENTATION**

import java.awt.\*;

import java.applet.Applet;

//applet tag

/\*

<html>

<applet code ="TowersOfHanoiGUI.class" height=1200 width=1200>

</applet>

</html>

\*/

public class TOHGUI extends Applet implements Runnable

{

private static Color BACKGROUND\_COLOR = new Color(255,255,180);

private static Color BORDER\_COLOR = new Color(100,0,0);

private static Color DISK\_COLOR = new Color(0,0,180);

private static Color MOVE\_DISK\_COLOR = new Color(180,180,255);

private Image OSC; // The off-screen canvas.

private static final int GO = 1, SUSPEND = 2, TERMINATE = 3;

//Values for status.

private int status = GO; // Controls the execution of the thread.

private Thread runner; // A thread to run the animation.

private int[][] tower;

private int[] towerHeight;

private int moveDisk;

private int moveTower;

public void init()

{

// Initialize the applet by setting the background color.

setBackground(BACKGROUND\_COLOR);

}

public void run() {

try {

while (true)

{

tower = null;

if (OSC != null)

{

Graphics g = OSC.getGraphics();

drawCurrentFrame(g);

g.dispose();

}

repaint();

delay(2000);

synchronized(this)

{

tower = new int[3][7];

for (int i = 0; i < 7; i++)

tower[0][i] = 7 - i;

towerHeight = new int[3];

towerHeight[0] = 7;

if (OSC != null)

{

Graphics g = OSC.getGraphics();

drawCurrentFrame(g);

g.dispose();

}

repaint();

delay(2000);

}

solve(10,0,1,2);

delay(4000);

}

}

catch (IllegalArgumentException e){}

}

private void solve(int disks, int from, int to, int spare) {

if (disks == 1)

moveOne(from,to);

else

{

solve(disks-1, from, spare, to);

moveOne(from,to);

solve(disks-1, spare, to, from);

}

}

synchronized private void moveOne(int fromStack, int toStack)

{

moveDisk = tower[fromStack][towerHeight[fromStack]-1];

moveTower = fromStack;

delay(140);

towerHeight[fromStack]--;

putDisk(MOVE\_DISK\_COLOR,moveDisk,moveTower);

delay(80);

putDisk(BACKGROUND\_COLOR,moveDisk,moveTower);

delay(80);

moveTower = toStack;

putDisk(MOVE\_DISK\_COLOR,moveDisk,moveTower);

delay(80);

putDisk(DISK\_COLOR,moveDisk,moveTower);

tower[toStack][towerHeight[toStack]] = moveDisk;

towerHeight[toStack]++;

moveDisk = 0;

}

private void putDisk(Color color, int disk, int t)

{

if (OSC != null)

{

Graphics g = OSC.getGraphics();

g.setColor(color);

g.fillRoundRect(75+140\*t - 5\*disk - 5, 116-12\*towerHeight[t], 10\*disk+10, 10, 10, 10);

g.dispose();

}

repaint();

}

synchronized void drawCurrentFrame(Graphics g)

{

g.setColor(BACKGROUND\_COLOR);

g.fillRect(0,0,430,143);

g.setColor(BORDER\_COLOR);

g.drawRect(0,0,429,142);

g.drawRect(1,1,427,140);

if (tower == null)

return;

g.fillRect(10,128,130,5);

g.fillRect(150,128,130,5);

g.fillRect(290,128,130,5);

g.setColor(DISK\_COLOR);

for (int t = 0; t < 3; t++)

{

for (int i = 0; i < towerHeight[t]; i++)

{

int disk = tower[t][i];

g.fillRoundRect(75+140\*t - 5\*disk - 5, 116-12\*i, 10\*disk+10, 10, 10, 10);

}

}

if (moveDisk > 0)

{

g.setColor(MOVE\_DISK\_COLOR);

g.fillRoundRect(75+140\*moveTower - 5\*moveDisk - 5, 116-12\*towerHeight[moveTower], 10\*moveDisk+10, 10, 10, 10);

}

}

synchronized void delay(int milliseconds)

{

if (status == TERMINATE)

throw new IllegalArgumentException("Terminated");

try {

wait(milliseconds);

}

catch (InterruptedException e) {

}

while (status == SUSPEND)

{

try {

wait();

}

catch (InterruptedException e) {

}

}

if (status == TERMINATE)

throw new IllegalArgumentException("Terminated");

}

synchronized public void start()

{

// When applet is started or restarted, start or

// restart the thread.

status = GO; // Indicates that the applet is running.

if (runner == null || !runner.isAlive())

{

tower = null;

runner = new Thread(this);

runner.start();

}

else

{

notify();

}

}

synchronized public void stop()

{

status = SUSPEND;

notify();

}

synchronized public void destroy()

{

status = TERMINATE;

notify();

}

synchronized public void paint(Graphics g)

{

if (OSC == null)

setupOSC();

if (OSC == null)

drawCurrentFrame(g);

else

g.drawImage(OSC,0,0,this);

}

public void update(Graphics g)

{

paint(g);

}

synchronized private void setupOSC() {

OSC = null; // Free memory currently used by OSC.

try {

OSC = createImage(430,143);

}

catch (OutOfMemoryError e) {

OSC = null;

return;

}

Graphics OSG = OSC.getGraphics();

drawCurrentFrame(OSG);

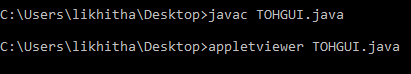
OSG.dispose();

}

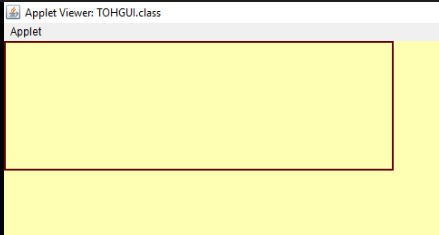
} // end class TowersOfHanoiGUI

**OUTPUT**

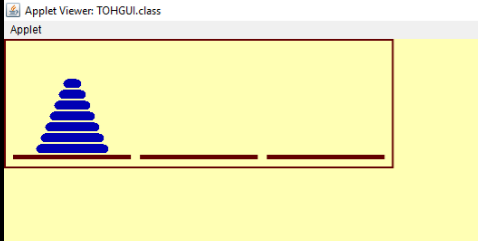
**INITIALIZING THE PROGRAM:**

****

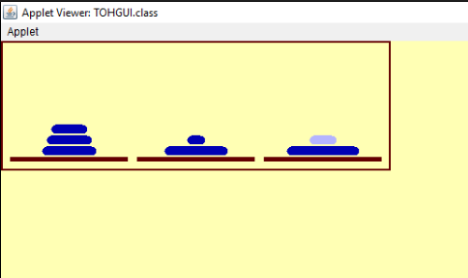
**OUTPUT OF THE PROGRAM:**

****

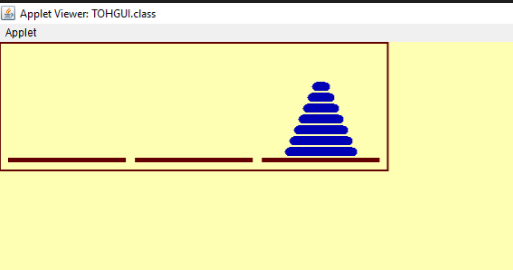
**INITIAL STATE:**

****

**OPERATIONAL STATE:**

****

**FINAL STATE:**

****

**CONCLUSION**

The end output successfully implements the building of Towers of Hanoi considering the best case.

By completing the project we have learned the following concepts successfully.

* The AWT components of java.
* The applications and real power behind the usage of Recursive functions.
* Stack implementation for performing various operations.
* Threads to animate the objects in various situations.

**REFERENCES**

* Javatpoint-www.javatpoint.com
* Wikipedia-www.wikipedia.com
* Google-www.google.com
* Oracle-oracle.net