Instruction:

General:

1. This is a **GROUP** assignment.
2. Based on the problem statement assigned to you, produce a solution following the description attached in this sheet.
3. Copied work will not be graded.
4. Write your name, I.D. number in the space provided and sign the declaration statement on page 2 in Assignment Cover Page and saved as PDF. Each student must submit her copy on LMS(BB or Schoology)

Specific:

1. Suggest a project that will utilize any **simple data structure** (stack, queue, or priority queue).
2. Use this form to submit your project.
3. Fill in the blanks appropriately.
4. Submit the **Project Cover Page (save as PDF after signed)** and the **program code (softcopy)** at the links provided on Blackboard according to the data specified.
5. Write your solution in the space provided. Use the sample as a guideline to write your answer.
6. Due date is **Monday, June 14, 2021** at 4.30 p.m. Late submission is accepted but discounted as 1 point for each day passed.

Problem Statement:

Write a single program demonstrating the uses of STACK from an array(* include all basic operation).

Our project it about Tower of Hanoi, which is a game about moving numbers in decreasing order from the first tower to the last one.

Explain types of data structure used:

- **The data structure we have used is STACK.**
- **The operations we used:**
  - **push(): used to insert an element on top of stack.**
  - **pop(): used to delete an element from top of stack.**
  - **isEmpty(): used to check before delete.**
  - **size(): used to know how many processes of insertion has been taking.**
  - **peek(): used to return the top element in the stack.**

Attach your program code here:

```
import java.util.Stack;
import java.util.EmptyStackException;
public class Player //start of Player class
{
   private String name;
```

```
private int move;
//constructor
public Player(String p)
{
    setName(p);
}
   //set&get method
public void setName(String n)
{
   name=n;
}

 public String getName()
{
   return name;
}
//operations on the movments depending on the rules of the game
public void diskMovment(int t1,int t2,Stack<Integer> n1,Stack<Integer> n2,Stack<Integer> n3)
{
 if (t2==1){
    if(t1==1){}

    if(t1==2)

     if(n2.isEmpty()||n1.isEmpty()||n1.peek()>n2.peek())
       try{//catching the exeption of poping an empty stack
           n1.push(n2.pop());
            System.out.println("                    moves: "+move());
       }catch(EmptyStackException e){
            System.out.println("It is empty. Try again");
       }
     else
        System.out.println("wrong movment");

    if(t1==3 ){

     if(n3.isEmpty()||n1.isEmpty()||n1.peek()>n3.peek())
       try{
           n1.push(n3.pop());
            System.out.println("                    moves: "+move());
       }catch(EmptyStackException e){
            System.out.println("It is empty. Try again");
       }
     else
        System.out.println("wrong movment");
    }
 }

 if (t2==2){
```

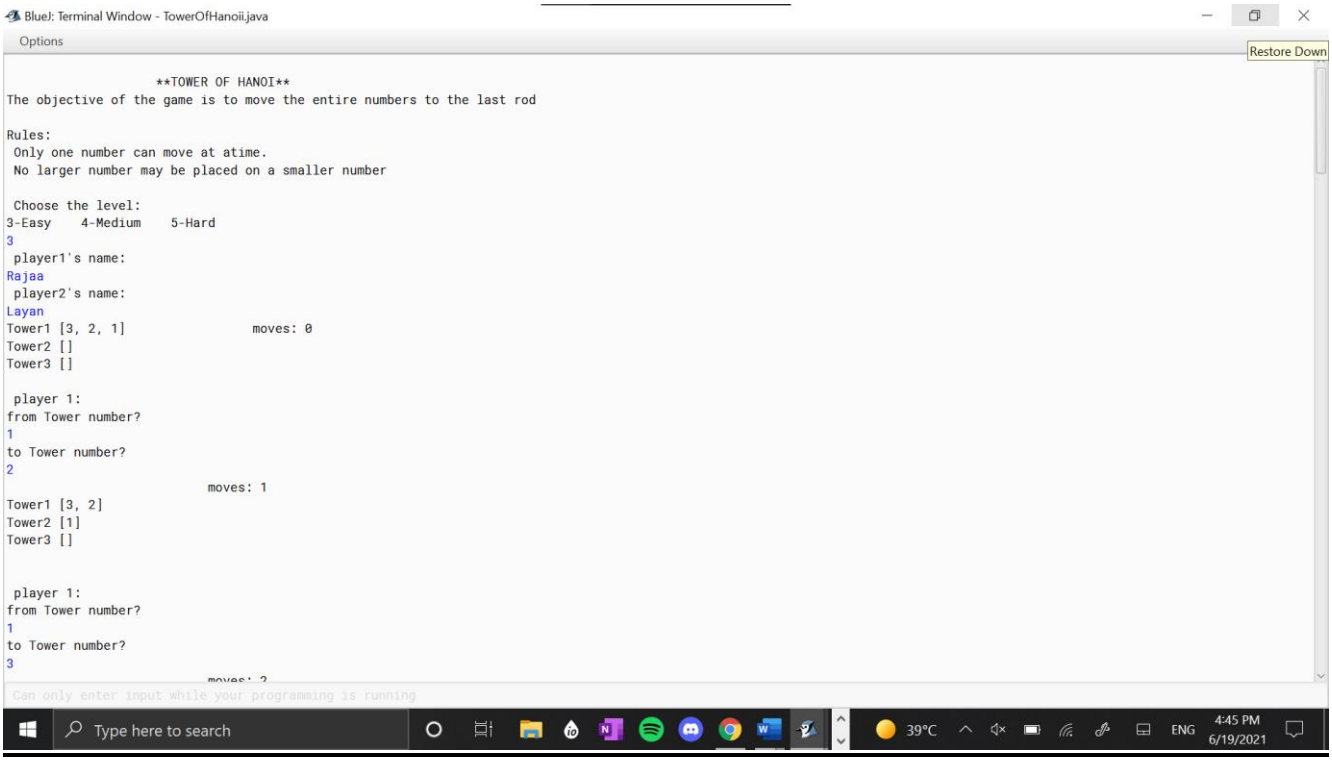
```
    if(t1==1){
     if(n1.isEmpty()||n2.isEmpty()||n2.peek()>n1.peek())
       try{
          n2.push(n1.pop());
           System.out.println("                    moves: "+move());
       }catch(EmptyStackException e){
           System.out.println("It is empty. Try again");
       }
     else
       System.out.println("wrong movment");}

    if(t1==2){}

    if(t1==3 ){

     if(n3.isEmpty()||n2.isEmpty()||n2.peek()>n3.peek())
       try{
          n2.push(n3.pop());
          System.out.println("                    moves: "+move());
       }catch(EmptyStackException e){
          System.out.println("It is empty. Try again");
       }
     else
       System.out.println("wrong movment");
    }
  }

  if (t2==3){
    if(t1==1){

     if(n1.isEmpty()||n3.isEmpty()||n3.peek()>n1.peek()) // r played in it
       try{
          n3.push(n1.pop());
          System.out.println("                    moves: "+move());
       }catch(EmptyStackException e){
          System.out.println("It is empty. Try again");
       }
     else
       System.out.println("wrong movment");}

    if(t1==2)

     if(n3.isEmpty()||n2.isEmpty()||n3.peek()>n2.peek())
       try{
          n3.push(n2.pop());
           System.out.println("                    moves: "+move());
       } catch(EmptyStackException e){
          System.out.println("It is empty. Try again");
       }
       else
```

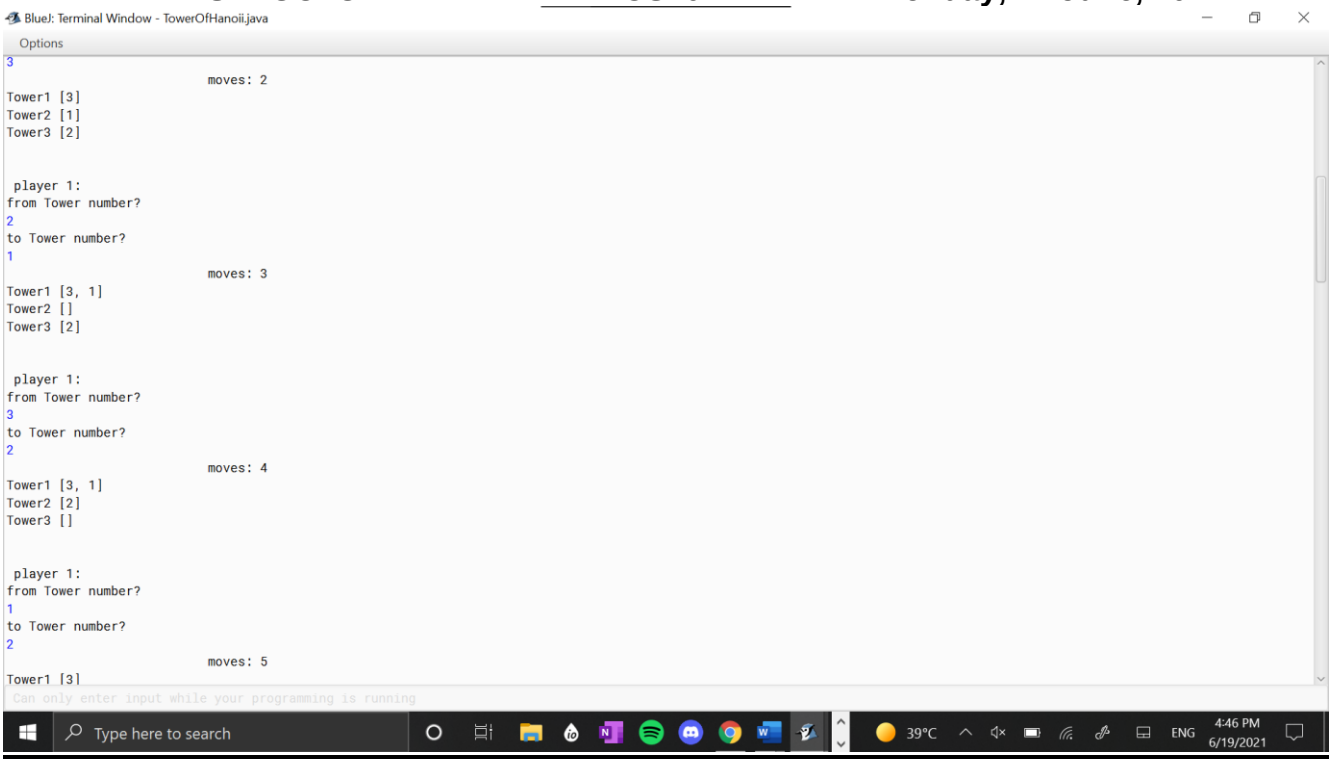
```
            System.out.println("wrong movment");

       if(t1==3 ){}
     }
   }

   public  int move(){
     return move++;//movment counter
   }
} //End of Player class

import java.io.*;
import java.math.*;
import java.util.*;
public class TowerOfHanoi{ //start of TowerOfHanoi class


 public static void main(String args[]){

   Scanner in = new Scanner(System.in);
   int table1,table2,disk,IntMinMove;

   String p1name,p2name;
   // game instructions
   System.out.printf("\n                    **TOWER OF HANOI**\n");
   System.out.println("The objective of the game is to move the entire numbers to the last rod");
   System.out.printf("\nRules:\n Only one number can move at atime.\n No larger number may be
placed on a smaller number\n\n ");
    //Declear 3 stack for the towers
   Stack<Integer> num1 =new<Integer> Stack();
   Stack<Integer> num2 =new<Integer> Stack();
   Stack<Integer> num3 =new<Integer> Stack();
    //let user choose the level
   System.out.println("Choose the level:");
   System.out.println("3-Easy    4-Medium    5-Hard");
   disk=in.nextInt();
    //Players Name
   System.out.println(" player1's name:");
   p1name=in.next();
   Player p1=new Player(p1name);
   System.out.println(" player2's name:");
   p2name=in.next();
   Player p2=new Player(p2name);
    //leval tapy
   if(disk<=5&&disk>=3)
     for(int i=disk;i<=disk&&i>0;i--)
      num1.push(i);
   else
     System.out.println("Minimum number of disks is 3 .Maximum number of disks is 5 ");
```

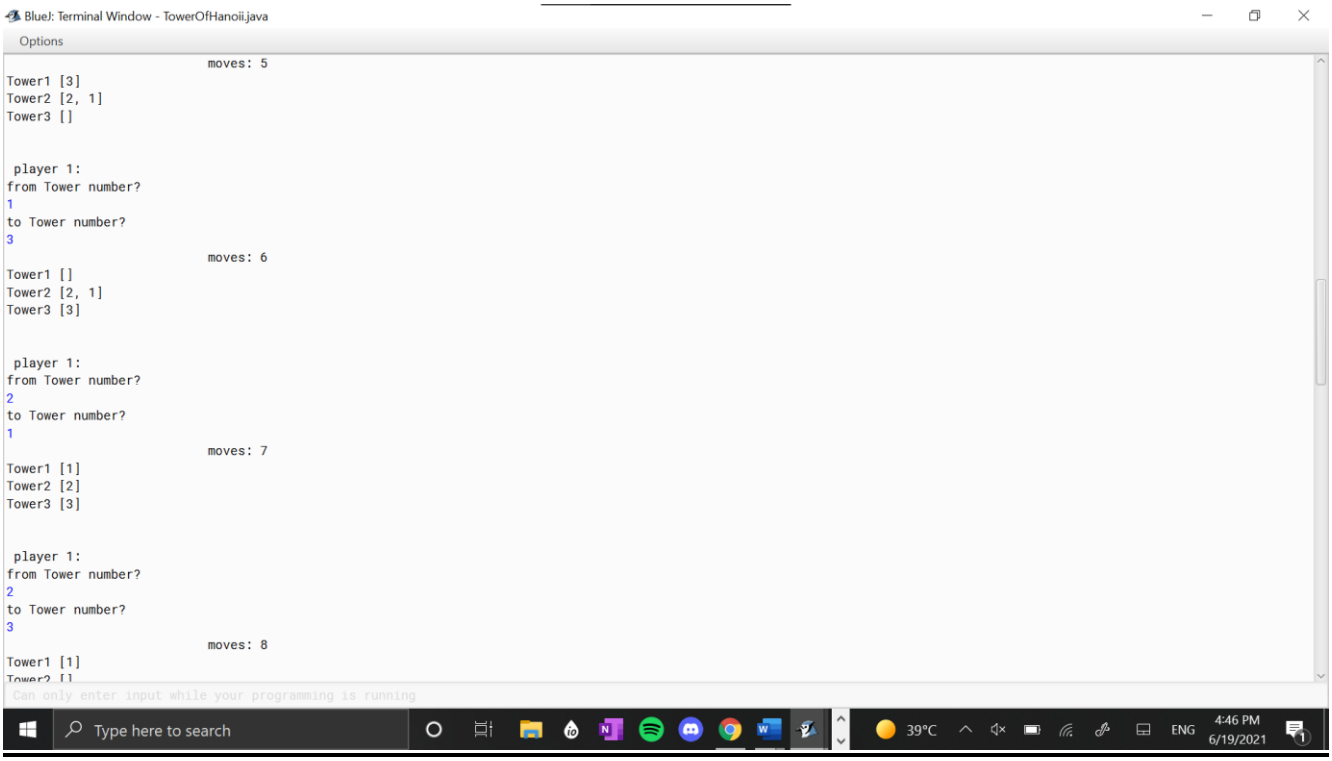
```
System.out.println("Tower1 "+num1+"              moves: "+p1.move());
System.out.println("Tower2 "+num2+"");
System.out.println("Tower3 "+num3+"");

do{
  //Start game player 1
 System.out.println();
 System.out.println(" player 1:  ");
 System.out.println("from Tower number?");
 table1=in.nextInt(); //move from
 System.out.println("to Tower number?");
 table2=in.nextInt(); //move to
 p1.diskMovment(table1, table2, num1, num2, num3);
 System.out.println("Tower1 "+num1           );
 System.out.println("Tower2 "+num2+"");
 System.out.println("Tower3 "+num3+"");
 System.out.println();
}while(num3.size()!=disk);
  //fill stack 1
if(disk<=5&&disk>=3)
  for(int i=disk;i<=disk&&i>0;i--)
  { num1.push(i);
    num3.pop();
  }

else
  System.out.println("Minimum number of disks is 3 .Maximum number of disks is 5 ");

System.out.println("Tower1 "+num1+"              moves: "+p2.move());
System.out.println("Tower2 "+num2+"");
System.out.println("Tower3 "+num3+"");

do{
  //start game for player 2
  System.out.println( );
 System.out.println(" player 2:  ");
 System.out.println("from Tower number?");
 table1=in.nextInt();//move frome
 System.out.println("to Tower number?");
 table2=in.nextInt();//move to
 p2.diskMovment(table1, table2, num1, num2, num3);
 System.out.println("Tower1 "+num1           );
 System.out.println("Tower2 "+num2+"");
 System.out.println("Tower3 "+num3+"");
}while(num3.size()!=disk);

System.out.println("Winner: "+winner(p1,p2));
//find minimum number of movments
double minmove =Math.pow(2,disk);
IntMinMove =(int) minmove-1;
```

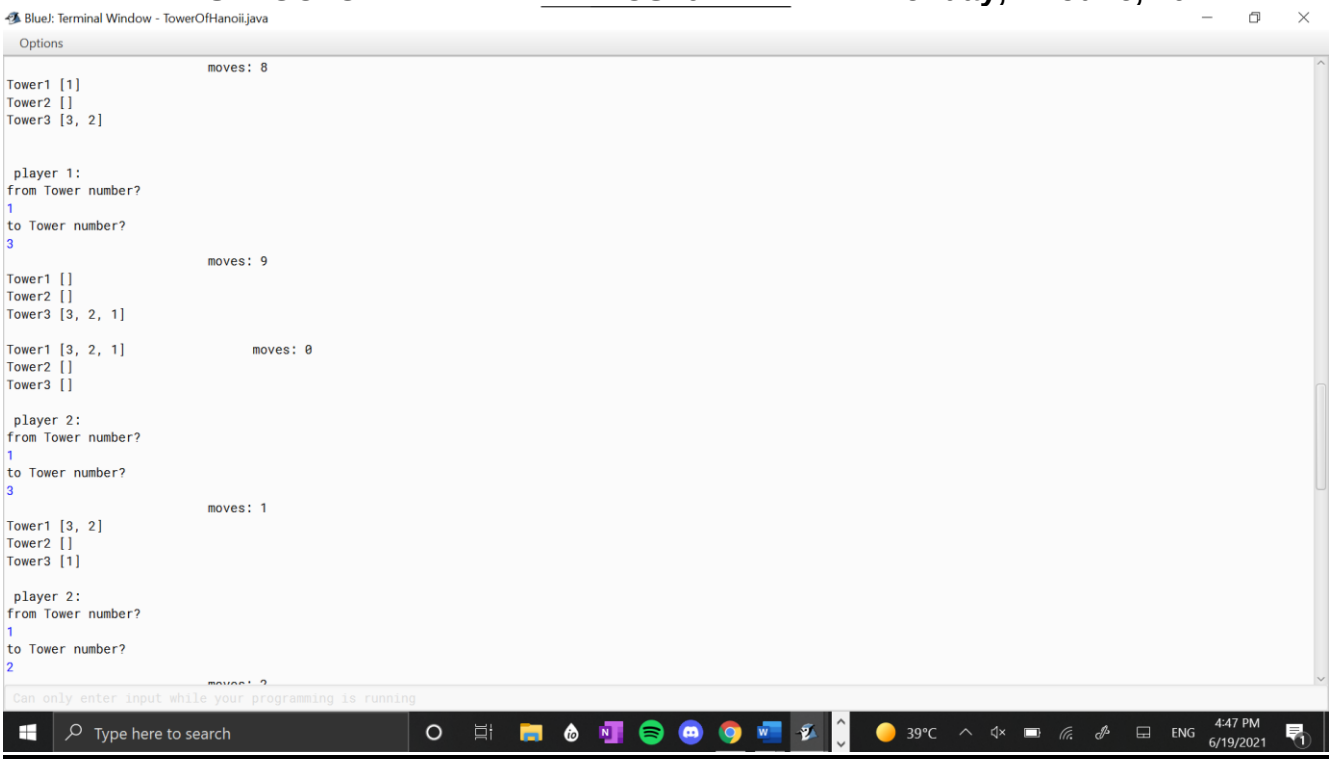
```
   System.out.printf("\n\n GREAT JOB \nMinimum number of movment is %d",IntMinMove);//End
the game
}
//Compare the moves between players
public static String winner(Player p1,Player p2){
    String win=null;
  if(p1.move()<p2.move())
    win= p1.getName();
  else if(p1.move()>p2.move())
    win= p2.getName();
  else if(p1.move()==p2.move())
    win= "no winner";
    return win;
}
 } //End of TowerOfHanoi class
```

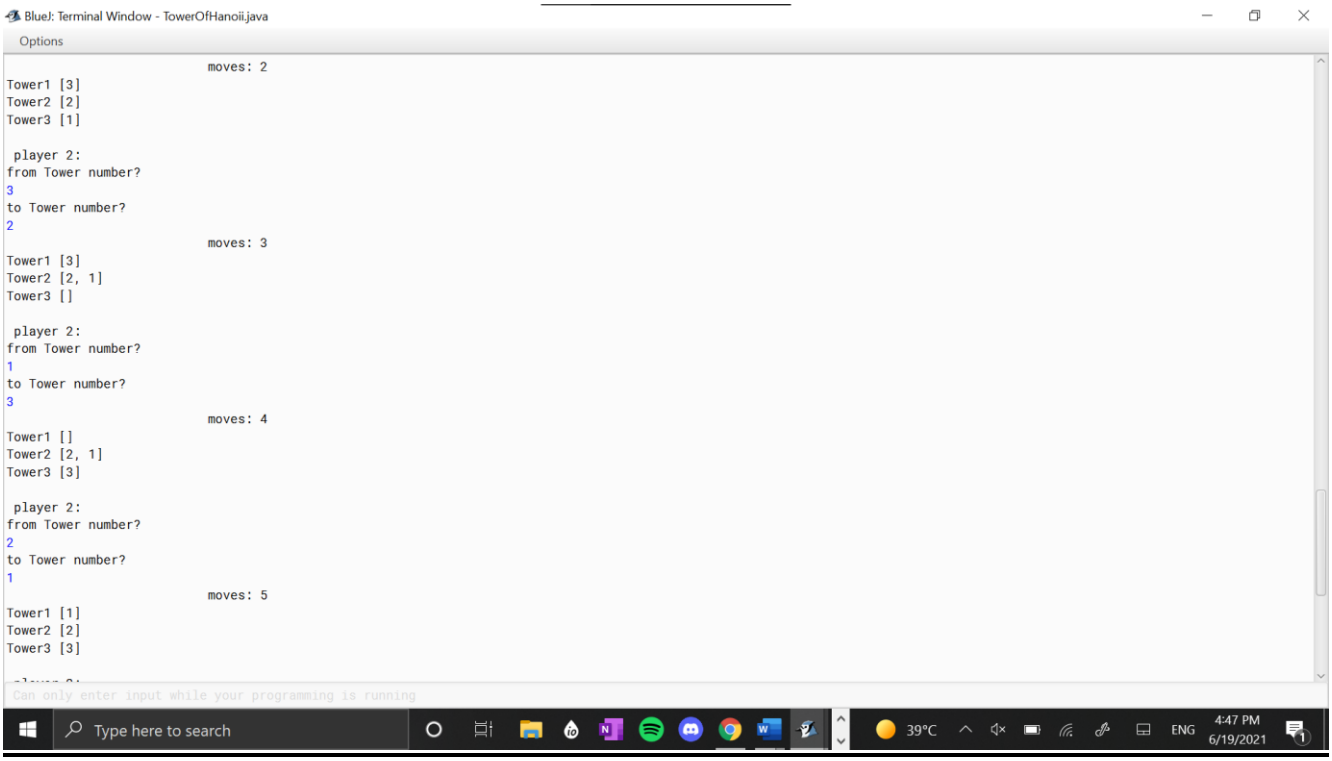
SCREENSHOTS:

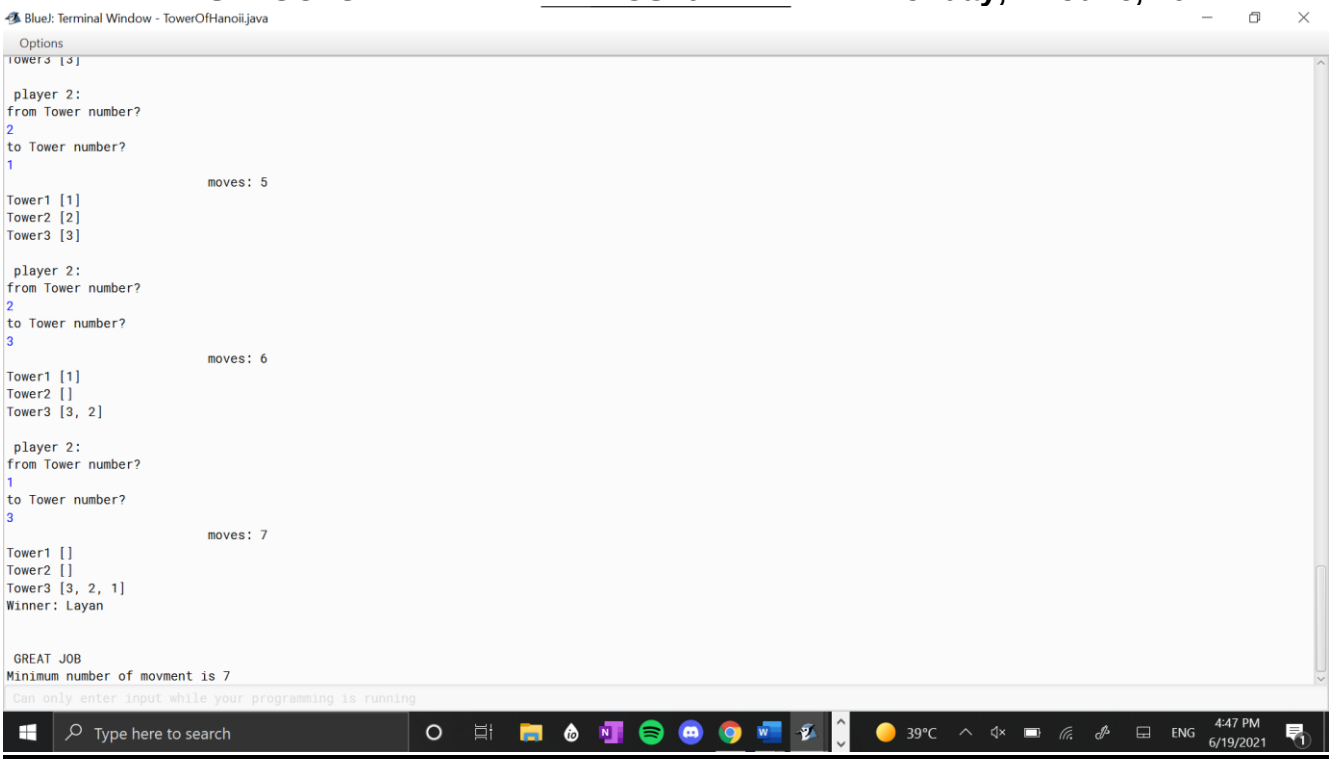



BlueJ: Terminal Window - TowerOfHanoii.java
Options
3
moves: 2
Tower1 [3]
Tower2 [1]
Tower3 [2]
player 1:
from Tower number?
2
to Tower number?
1
moves: 3
Tower1 [3, 1]
Tower2 []
Tower3 [2]
player 1:
from Tower number?
3
to Tower number?
2
moves: 4
Tower1 [3, 1]
Tower2 [2]
Tower3 []
player 1:
from Tower number?
1
to Tower number?
2
moves: 5
Tower1 [3]
Can only enter input while your programming is running
Type here to search
39°C
ENG
4:46 PM
6/19/2021


BlueJ: Terminal Window - TowerOfHanoii.java
Options
moves: 5
Tower1 [3]
Tower2 [2, 1]
Tower3 []
player 1:
from Tower number?
1
to Tower number?
3
moves: 6
Tower1 []
Tower2 [2, 1]
Tower3 [3]
player 1:
from Tower number?
2
to Tower number?
1
moves: 7
Tower1 [1]
Tower2 [2]
Tower3 [3]
player 1:
from Tower number?
2
to Tower number?
3
moves: 8
Tower1 [1]
Can only enter input while your programming is running
Type here to search
39°C
ENG
4:46 PM
6/19/2021


BlueJ: Terminal Window - TowerOfHanoii.java
Options
moves: 8
Tower1 [1]
Tower2 []
Tower3 [3, 2]
player 1:
from Tower number?
1
to Tower number?
3
moves: 9
Tower1 []
Tower2 []
Tower3 [3, 2, 1]
Tower1 [3, 2, 1] moves: 0
Tower2 []
Tower3 []
player 2:
from Tower number?
1
to Tower number?
3
moves: 1
Tower1 [3, 2]
Tower2 []
Tower3 [1]
player 2:
from Tower number?
1
to Tower number?
2
Can only enter input while your programming is running
Type here to search
39°C
ENG
4:47 PM
6/19/2021


BlueJ: Terminal Window - TowerOfHanoii.java
Options
moves: 2
Tower1 [3]
Tower2 [2]
Tower3 [1]
player 2:
from Tower number?
3
to Tower number?
2
moves: 3
Tower1 [3]
Tower2 [2, 1]
Tower3 []
player 2:
from Tower number?
1
to Tower number?
3
moves: 4
Tower1 []
Tower2 [2, 1]
Tower3 [3]
player 2:
from Tower number?
2
to Tower number?
1
moves: 5
Tower1 [1]
Tower2 [2]
Tower3 [3]
Can only enter input while your programming is running
Type here to search
39°C
ENG
4:47 PM
6/19/2021



RUBRIC GUIDELINE:

This is individual project. Each task has several attributes to look at and each will be evaluated using this grading system. Any sign of copy-paste and code sharing will link to **no GRADE.**

ASSESSMENT RUBRIC:

This assignment is to evaluate individual work performance. Each task has several attributes to look at and each will be evaluated using this grading system.

Description:

0.2<0.5- Below level ~ attributes for evaluation not exist or exist but poorly stated or prepared (format not visible, obvious typo errors, obvious grammatical errors.

0.5 < 1.0 Developing level ~ attributes visible but without details. Description may be vague and difficult to understand. Visible minor error.

1.0<1.5 Accepted level ~ attributes visible with simple description and understandable. Exist proper formatting with acceptable errors

1.5< 2.0- Good ~ attribute has complete descriptions with supporting details and materials, in a proper format and fulfill report documentation. Contents are error free.

2.0 - Excellent ~ attributes produced reach to an exceed expectation.

	Attributes checked for evaluation.	**0**	**0.5**	**1.0**	**1.5**	**2**
a)	Program application matched with selected data structure with evident choice of algorithm to manipulate the structure, Able to propose solution using a suitable data structure, Recognize and assign suitable identifiers that reflect the data structure functionality (insert, delete, list, search) (1.02 -D/4)					
b)	Recognize suitable formula and logical steps to process solution, (2.01 -B /1)					
c)	Design shows friendly user interface, Evident meaningful output to reflect program. Evident relevant instruction to guide user for application.(2.01-B/1)					
d)	Program utilize comments to describe system functionality, follow submission instructions and schedule. (3.01-B/6)					
e)	Document neat, complete, proper with comments, follow submission instructions and schedule. Ontime.(4.01-F/6)					
	TOTAL					