Towers of Hanoi

Team Name:

Manager:

Recorder:

Presenter:

Analyst:

This is a Process Oriented Guided Inquiry Learning (POGIL) activity. You and your team will examine a working program. A series of questions will guide you through a cycle of exploration, concept invention, and application. There is strong evidence that this is more effective (and less boring) than a traditional lecture.

By the time you are done with this activity, you and your team should be able to:

- understand and write simple recursive methods.
- think more critically.

Your team's recorder is responsible for writing your team's answers to the numbered questions on this form.

After you complete this activity, please fill out the short survey at

http://goo.gl/forms/HXjyuUb2ou

to improve this activity for future users.

Playing the game

The Towers of Hanoi puzzle was invented in 1883 by Édouard Lucas. Our program is not about allowing humans to solve the puzzle but about generating solutions. To understand the puzzle itself, go to https://www.mathsisfun.com/games/towerofhanoi.html and play. First try to solve the puzzle with three disks, then four.

1. Is everyone done playing and ready to pay attention to the team?

You may need to go back to the game to answer some of the questions to come, but you should do so *deliberately*, because your team's manager assigned one or more people to find something out, not merely because you got bored with the conversation or thought you could answer a question better on your own.

2. What are the limitations on moving disks?

3. Does the puzzle become easier or more difficult with more disks? Why?

If the three towers are labeled A, B, and C, here is a solution for two disks:

A → B
A → C
B → C

4. What is a solution for three disks?

5. What is a solution for four disks?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have.

Hard-coded solutions

Examine Hanoi.java. This program contains several solutions, which we will examine in turn.

As written, the program prints out a solution for one disk by calling `hanoi1HardCoded`.

6. Modify `main` to call `hanoi2HardCoded` instead. You will need three arguments instead of two. Why is the third argument needed?

7. Does `hanoi2HardCoded` produce a correct solution for two disks?

8. Does `hanoi3HardCoded` produce a correct solution for three disks?

9. What is accomplished by the first three lines in `hanoi3HardCoded`?

10. Does `hanoi3HardCoded` produce a correct solution for three disks?

11. What is accomplished by the fourth line in `hanoi3HardCoded`?

12. What is accomplished by the last three lines in `hanoi3HardCoded`?

13. What relevance does the largest disk have while solving this subproblem?

14. These methods are said to have solutions “hard coded” into them: the computer doesn’t so much solve the puzzle as spit out a prerecorded answer. Would this be a good approach for producing, say, a solution for seven disks? Why or why not?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team’s presenter should be prepared to present any of your team’s previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have.

Calling simpler methods

15. Modify `main` to call `hanoi1CallingSimplerMethods`. Does it produce a correct solution for one disk?

16. Does `hanoi2CallingSimplerMethods` produce a correct solution for two disks?

17. Does `hanoi3CallingSimplerMethods` produce a correct solution for three disks?

18. How does `hanoi3CallingSimplerMethods` differ from `hanoi3HardCoded`?

With `main` set up to call `hanoi3CallingSimplerMethods`, place a breakpoint in `hanoi1CallingSimplerMethods` and run the program in the debugger. (If you've forgotten how to use the debugger, review the video at http://screencast.com/t/NEgEMW6sNB2.)

19. Complete the table below summarizing the state of the call stack.

Method	**Arguments**
hanoi3CallingSimplerMethods	start = A, spare = B, end = C
main	args = <array of length 0>

20. Does `start` have the same value throughout the call stack? If so, what is it? If not, why not?

21. Does `spare` have the same value throughout the call stack? If so, what is it? If not, why not?

22. Does `end` have the same value throughout the call stack? If so, what is it? If not, why not?

23. Hit Resume to run to the next time the program reaches the breakpoint. Which argument/ variable values changed in the call stack? Why?

Return to the Java perspective in Eclipse.

24. What is the same between `hanoi3CallingSimplerMethods` and `hanoi2CallingSimplerMethods`?

25. What is different between `hanoi3CallingSimplerMethods` and `hanoi2CallingSimplerMethods`?

26. Can you write `hanoi4CallingSimplerMethods`? Does it work? How do you know?

27. Would this be a good approach for producing, say, a solution for seven disks? Why or why not?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have.

Recursion

Modify `main` to call `hanoi`:

```
hanoi("A", "B", "C", 3);
```

28. What does this version of the program print?

29. What does the fourth argument specify?

30. How can you use hanoi to produce a solution for four disks?

31. What about seven disks?

Now examine the code for the `hanoi` method.

32. How is `hanoi` similar to `hanoi3CallingSimplerMethods`?

33. How is it different?

34. `hanoi3CallingSimplerMethods` called `hanoi2CallingSimplerMethods` to solve subproblems. What does `hanoi` call?

35. This surprising action is called *recursion*. Did some or all of your team previously believe this was impossible? Why?

36. With `main` set to call `hanoi` using three disks, place a breakpoint on line 45 and run the program in the debugger. Complete the table below summarizing the state of the call stack.

Method	Arguments
hanoi	start = A, spare = B, end = C, n = 3
main	args = <array of length 0>

37. `hanoi` calls itself to solve easier problems. In what sense are they easier?

38. For what argument values does `hanoi` *not* recursively call itself?

39. Returning to the Java perspective, comment out lines 44-46 and 50. These make up the base case, the easiest problem. What happens when you run the program without the base case? Why?

40. Restoring those lines, replace `n - 1` with `n` on lines 47 and 49. What happens when you run the program? Why?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have.

Please fill out the survey at http://goo.gl/forms/HXjyuUb2ou.