Anagrams

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:

- trace through a sequence of method calls.
- run JUnit tests and interpret the results.
- process information more effectively.

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

Open the Anagrams project in Eclipse. Run Anagrams.java to play the game. This is a one-player game, so each member of your team can play individually for a while. Each word is an independent puzzle; there is no longer-term winning or losing.

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

You may need to go back and play the game again 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. How does the program know when you are done typing your solution?

3. Does the program let you type things other than letters, such as digits? Does backspace work? What about the arrow keys?

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. If your team is done before other teams, discuss the following open-ended question:

4. How could you change the user interface to make the game easier or more enjoyable to play?

The call stack

Examine Anagrams.java.

5. How many methods are in this program?

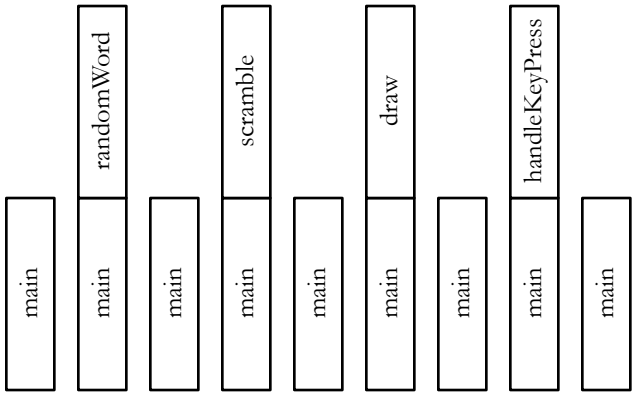
6. What are their names?

7. Which other methods in this program are called directly from `main`?

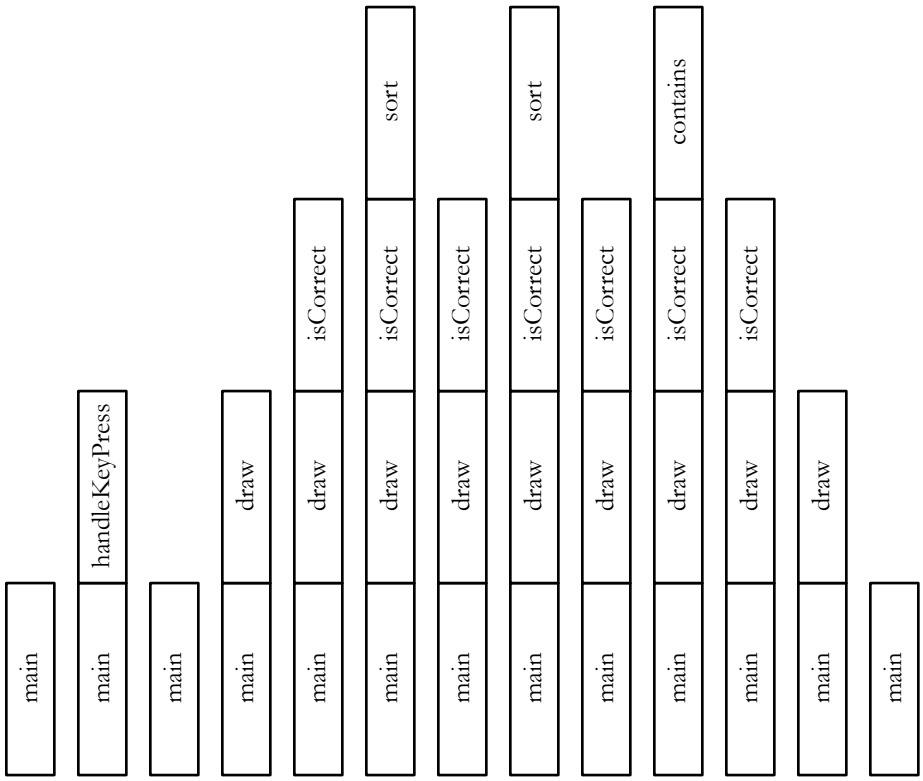
8. Which other methods in this program are called directly from `isCorrect`?

9. Draw a box-and-arrow diagram, similar to the first one in the instructions for *Domineering*, showing which methods call which other methods.

Here is a diagram showing a series of states of the call stack as the program begins:



Here is another diagram showing what happens when the user finishes typing their answer:



10. In what direction does time move in these diagrams (e.g., left to right or top to bottom)?

11. What happens in the diagram when a method is called?

12. What happens in the diagram when a method returns?

13. How did your team reach answers to the two previous questions?

14. After the `randomWord` method finishes running, what line of the program is run next? (Write the code for that line.)

15. After the `handleKeyPress` method finishes running, there are two possibilities for which line of the program is run next. What are they?

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. If your team is done before other teams, discuss the following open-ended question:

16. In general, after a method finishes running, how does Java know which line to run next?

JUnit testing

Running tests on individual methods (*unit tests*), as we did in *Domineering*, is extremely useful but can get a bit tedious. JUnit is a tool to automate the running of unit tests.

The file AnagramsTest.java is a JUnit test suite. Each method here is an automated test. To run the tests, right click on AnagramsTest.java in the Package Explorer and select Run As → JUnit Test.

17. Describe what happened when you did this.

18. How many tests were run?

19. How many errors were there?

20. How many failures were there?

21. How did your team reach answers to the previous three questions?

22. How can you get back to the Package Explorer?

Add the following line at the beginning of the `sort` method in Anagrams.java:

```
StdOut.println(3 / 0);
```

23. Describe what happens when you run the tests again.

24. Why do you think `isCorrectAcceptsCorrectAnswer` did not pass?

25. Click on `sortAlphabetizes` in the JUnit view. What appears in the lower left part of the Eclipse window?

26. Click on `isCorrectAcceptsCorrectAnswer`. Now what appears in the lower left?

27. What happens if, at the lower left, you double click on `Anagrams.sort`?

28. What if you double click on `Anagrams.isCorrect`?

29. Where on the screen is the current state of the call stack displayed?

30. If you were hunting for an error, would it be better to start at the top or the bottom of the call stack? Why?

Remove the erroneous line you added and run the tests again. The reassuring green bar should be back, telling you that all of the tests passed.

Now examine the tests themselves, in AnagramTest.java.

31. What special notation appears before each method?

32. Describe what each of the lines in `sortAlphabetizes` does.

33. Temporarily modify the `contains` method in Anagrams so that it always returns false. What happens when you run the tests?

34. What is the difference between (in JUnit's terminology) an error and a failure?

35. How did you team reach an answer to the previous question?

36. Does AnagramsTest ever draw anything on the screen or get input from the user?

37. If you had a suite of JUnit tests, would you run them more or less often than tests that require manual interpretation?

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. If your team is done before other teams, discuss the following open-ended question:

38. Exchange stories about nasty bugs you have encountered in past computer science courses or side programming projects. Would JUnit tests have helped you find or fix these bugs more quickly?

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

This work was supported by the Google Education and Relations Fund's CS Engagement Small Grant Program grant #TFR15-00411.

The principal investigator, Peter Drake, wishes to thank the following for their useful comments: the members of the CS-POGIL project, specifically Clif Kussmaul and Helen Hu; Maggie Dreyer; and various anonymous reviewers.