Beetle

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:

- interpret classes, including instance variables, methods, and constructors.
- create objects and call methods on them.
- solve problems 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

Beetle, sold commercially as *Cootie*, is a simple game meant to teach children about taking turns, rolling dice, following directions, etc. Our program automates the entire process; since there are no decisions for players to make, the user has nothing do but repeatedly click and observe the result. That said, the underlying code provides an excellent introduction to objects. Each member of your team should run BeetleGame.java a couple of times to get a sense of how the game works.

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. Each outcome of the die roll allows a player to add one piece to their beetle, as long as they have the prerequisite and don't have the maximum number of that piece. Complete the table below.

Roll	Part	Prerequisite	Maximum number
1	body	(none)	1
2			
3			
4	eye	head	2
5			
6			

3. How does a player win the game?

4. Under what circumstances does a player get to roll again?

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:

5. Some people are unable to distinguish red from green. Would such a user have any way of telling which beetle was theirs in this game? What could be done to make the interface friendlier to such users?

Die

Examine Die.java.

6. Which variable is declared outside of any method?

7. In which methods is this variable visible?

8. Where have you previously seen a variable declared outside of any method?

9. How do the methods in this class differ from those you have written previously?

10. How does the `Die` method differ from other methods?

Examine DieTest.java.

11. In `initialTopFaceIs1`, what is the type of the variable `d`?

12. What expression gives `d` its initial value?

13. In `allNumbersAreEquallyLikely`, what does `d.roll()` appear to do?

14. What does `d.getTopFace()` do?

15. In `multipleDiceCanShowDifferentTopFaces`, do `d1.getTopFace()` and `d2.getTopFace()` always, sometimes, or never return the same value?

In object-oriented programming terminology, Die defines a *class* of objects. Each of those objects is an *instance* of the Die class.

16. What expression creates an instance of the Die class?

17. Which method is invoked every time a new instance is created? (You may have to add some printlns to determine this.)

Such a method is called a *constructor*.

18. Does each instance have its own value for the variable `topFace`, or do they all share one?

A variable like this is called an *instance variable* or *instance field*.

19. How would you modify the Die class so that the number of sides on the die was not automatically 6 but was specified when an instance was created?

20. Did your team's first attempt at answering the previous question succeed? If not, what did you learn and apply in subsequent attempts?

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:

21. What happens if you change the declaration of `topFace` from `private int` to `private static int`? What do you think the keyword `static` means in this context?

Beetle

Examine Beetle.java.

22. How many instance variables does this class have?

23. What groups of similar methods are there?

24. Does the `addFeeler` method modify the Beetle instance, return a value, neither, or both?

Examine BeetleTest.java.

25. What instance variable is there in this class?

26. In which method is this instance variable initialized?

27. Is this method called just once, or before each test? (Hint: it is called automatically by JUnit, not explicitly in the code.)

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:

28. How would the other test methods have to modified if the method you just identified were removed?

BeetleGame

Examine BeetleGame.java.

29. What does the `main` method do?

30. What happens if, in `main`, you change `game.run()` to `run()`?

31. Does making `run` static fix this problem? Does it introduce any other problems?

32. What, in general, does the keyword `static` seem to mean?

33. What error message do you get if you remove the line

    ```
    beetles = new Beetle[4];
    ```

    from the constructor and then run the program?

34. What error message do you get if you instead remove the lines below?

```
for (int i = 0; i < beetles.length; i++) {
     beetles[i] = new Beetle();
}
```

35. Explain why both uses of `new` covered in the two previous questions are necessary.

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:

36. Aside from the test classes, there are three classes in this program: Die, Beetle, and BeetleGame. Could any of them be used without the others? Explain.

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.