Asteroid Rally

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

- define nontrivial classes.
- use a UML class diagram to decide on an order in which to write those classes.

Playing the game

Included in the Asteroid Rally module is a file AsteroidRally.jar. This is a compiled version of the working game. To play it, use a terminal to navigate to the directory containing the file and type this on the command line:

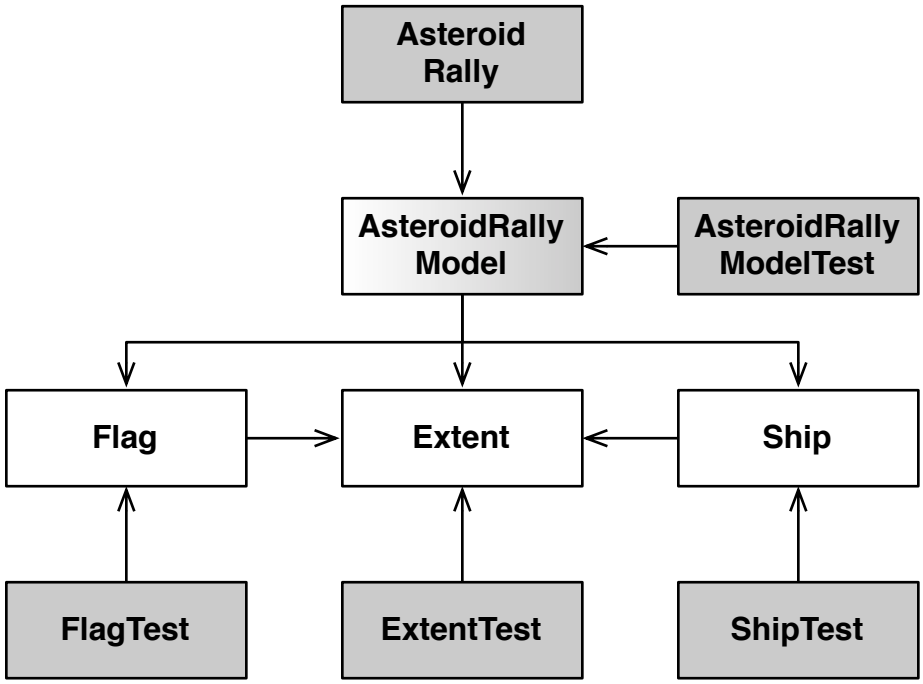
```
java -jar AsteroidRally.jar
```

Asteroid Rally is an original invention, heavily inspired by Atari's 1979 arcade game *Asteroids*. This is a two-player game. You and another student should play a few times to get a sense of how the game works.

Overview

You have only been given some of the files needed to play the game. Your first job is to complete the program so that it behaves *exactly* like AsteroidRally.jar. After that, you are free to make purely graphical enhancements such as a starfield background or flames shooting out of the back of a ship as it accelerates. Any changes to the game logic (more players, moving asteroids, guns, etc.) would interfere with the tests, so do any such exploration after you hand in your program.

The UML class diagram below shows the relationships between classes. The shaded classes have been provided for you; you should not alter them. AsteroidRallyModel is half-shaded because you have been given an incomplete skeleton. The unshaded classes you must write yourself.



Implementation

Take notes as you work. What bugs and conceptual difficulties did you encounter? How did you overcome them? What did you learn?

Looking at the UML class diagram will help you figure out an order in which to write the classes. If a class depends on a second class, that other class has to be working first.

For each of the classes you need to create there is a corresponding test class. These test classes won't compile yet because they depend on classes you need to write! For each class:

1. Write a bare skeleton of the class so that the test will compile. Don't bother providing any instance variables or putting any content in the methods (other than a return statement that might be necessary for the class to compile). You will have to examine the tests to see what methods are needed.

2. Fail the tests.

3. Work through the tests one at a time *in the order in which they appear in in the test class*, writing the necessary code in your class to pass them. Pass one test before moving on to the next one.

You should not modify any of the given tests. If it is helpful for debugging, you may write additional tests.

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.