Snowman

Your name:

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

- write nontrivial methods involving chars, char[]s, Strings, and String[]s.
- create a sequence of images using the StdDraw library.
- use JUnit tests to verify method correctness.

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

http://goo.gl/forms/HXjyuUb2ou

to improve this project for future users.

Playing the Game

Included in the Snowman project is a file Snowman.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 Snowman.jar
```

Play the game a few times so that you understand the rules.

Overview

You have been given an incomplete skeleton file Snowman.java. Your job is to complete the program so that it behaves *exactly* like Snowman.jar. You have quite a bit of creative freedom in drawing the pictures. You may use the snowman you have seen or any other drawing with six parts. For example, you might draw an airplane with a fuselage, two wings, two horizontal stabilizers on the tail, and a propeller.

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

You have been given `main` and part of `draw`. You have also been given a set of tests in SnowmanTest.java.

Start by running all of the tests. They should not pass because they depend on methods you haven't written yet!

Now go through the test *in the order in which they appear in SnowmanTest.java*, writing the necessary code in Snowman.java 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.

containsAcceptsPresentLetter

To pass this test, you will have to write the `contains` method. Some questions to ask as you write this or any other method:

- According to the Javadoc comment, what is the method supposed to accomplish?
- What arguments, if any, does the method take? What do they mean?

- What, if anything, is the method supposed to return? What does the returned value mean?
- Is the method supposed to have any side effects, such as printing or modifying existing data structures?

Other tests involving contains, isComplete, and fillIn

Ask yourself the same questions as you complete the relevant methods to pass these tests.

tests involving randomWord

One of these tests is unusual in that the test is considerably longer than the method (`randomWord`) that you have to write.

The tests use a three-word dictionary. Testing on smaller examples than those used in practice in the program is often a good idea, as it makes the results easier to interpret.

The method you have to write here exists, in exactly the form you need it, in one the previous Learn Java in *N* Games programs. There is no need to re-invent the wheel; you are allowed (nay, encouraged!) to copy that method into this program.

When copying existing code, be careful of two things:

- You must always include a comment saying where you copied it from. In this case, you comment might read, "Copied from the Learn Java in *N* Games program _____." Failure to do this is *presenting someone else's work as your own*, which can have very serious consequence both in academia (where it is called plagiarism) and in industry (where it is sometimes called intellectual property theft).
- Make sure that the code you are copying does what you need to have done. If you find code that is only close, you may have to modify it to suit your needs. You should still include a comment that says something like, "Adapted from _____.")

draw

There is no JUnit test for this method, as it is difficult to write automated tests of graphics. You'll have to test it manually by playing the game repeatedly. Verily, the life of a computer scientist is a difficult one.

You will probably want to use paper or a whiteboard to sketch out your drawing before turning it into code. No rescaling has been done, so *x* and *y* coordinates both range from 0.0 to 1.0.

It is also a good idea to write the code for the first couple of stages and test that before writing the code for the remaining stages. That way, if you've made some mistake like inverting the *y* axis, you'll discover it sooner and have less work to redo.

After you're done, please fill out the survey at http://goo.gl/forms/HXjyuUb2ou.