Problemset 04, Unit Testing

Due to your outstanding performance in programming, you've been hired by the prestigious ACME Inc, Programming division. Your first task is to improve some legacy code present at the company by adding tests. The company still holds on to some snakes and ladders game, as well as a graphical utensil that rotates and scales images.

Mocking frameworks.

You'll need to improve testing in the snakes and ladders game using mock objects. A test coverage tool revealed that in the problemset02 folder, Game#play(Die) currently isn't covered by any unit test. Your new boss claims that this could be easily tested using "mock objects". He can't explain what they are, but he insists you use them. We suggest the following course of action.

Use Eclipse's "Extract interface" refactoring to create an interface for `Die` called `IDie`. In the problemset02 folder, there's a new test case GameTest. There, write a test method that tests Game#play(IDie). As a parameter to the method, pass a mock of IDie. To achieve that, Familiarize yourself with the jMock framework. Then, add one more test case with at least one test method that tests another class using mocks.

Improve the unit tests in the image rotation utensil

The CTO also noticed that the unit tests in the image rotation utensil look somewhat clumsy. He's especially thrown off by the weak error messages that the unit tests return if a point isn't transformed to where it's expected. They just say that an assert failed, but he'd like to see right in the error message, where the point should be and where it is. Also, he doesn't like how the tests look. They look like this:

```
assertTrue(target.distance(new Point2D.Double(4,4)) < 0.001);
```

He maintains that the previoius line is ugly. He'd much rather have it read

```
assertThat(target, closeTo(new Point2D.Double(4,4)));
```

He says that he's seen assertions like that by people using Hamcrest. Familiarize yourself with Hamcrest and implement a matcher that allows the above syntax. Change TransformTest in problemset03 to use this new syntax. Write a test case with several unit tests that thoroughly tests your matcher. Make sure that it prints a useful error message when the assertion fails, like the CTO wants it.

Responsibility-driven design

Read the responsibility-driven design paper in the papers subfolders. Armed with your new knowledge, identify the responsibilities of all classes in problemset02 (except for tests) and note them in a sentence in the class comment of their classes.

Note: It should not be necessary to download hamcrest's or jmock's jar files, as we've downloaded and installed them into problemset02 and problemset03 already.