ROCK PAPER SCISSORS LIZARD SPOCK RADIOACTIVE, POISONOUS, INFESTED EDITION

EE363 DESIGN PROJECT 1

Radioactive, Poisonous, or Infested

In the Radioactive, Poisonous or Infested edition, each weapon may be "augmented" by one of three characteristics. It may be either: radioactive, poisonous, or infested.

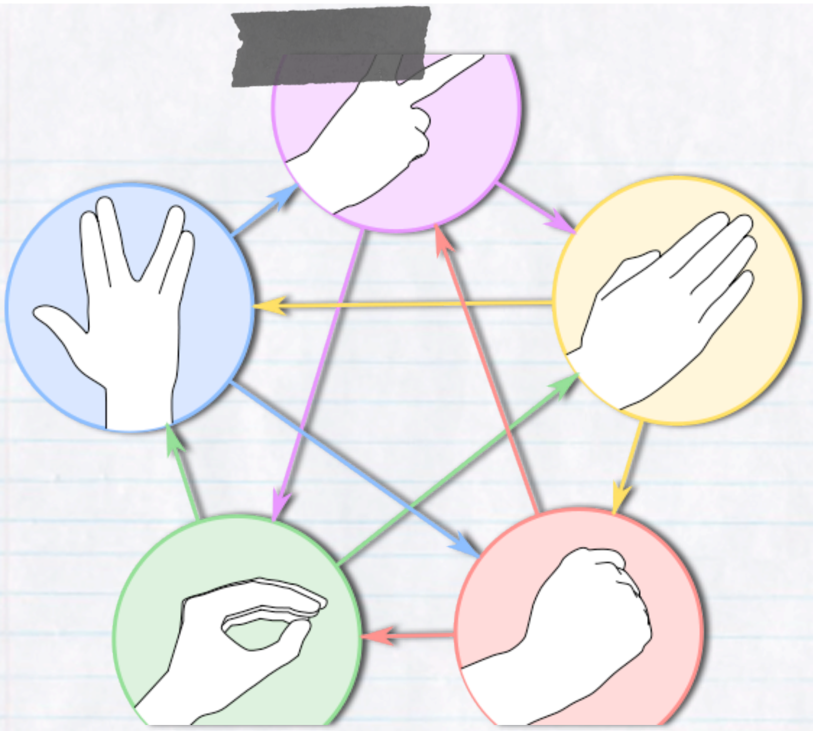
Radioactivity destroys poison; poison destroys infestation, and infestation survives ("beats") radioactivity.

A player indicates his weapon by throwing his right hand, and augments it by throwing his left: 1 finger for radioactivity, 2 for poisonous, and 3 for infested.

Scoring

You can decide to score the game in any number of ways:

- R/P/I first - giving weight to the characteristic of the weapon; the weapon is used to break any ties.
- R/P/s/L/S first - giving weight to the weapon itself, over any characteristic it might have; the characteristic is used to break any ties.



Rock, Paper, Scissors, Lizard, Spock

RPsLS is a five-gesture variant of roshambo (otherwise known as "rock, paper, scissors") in which two or more players chant the weapon names: Rock, Paper, Scissor, Lizard, Spock, then "throw" by making a hand gesture indicating their weapon of choice.

In the five-gesture variant, each weapon defeats two others, and is defeated by two others. A defeated player is out in the next round. Rounds continue until only player remains.

If playing in pairs, that is, only two players, often 3 rounds are played with the winner being chosen by a "best-of" strategy (or two-out-of-three).

Using the Object Oriented design patterns and principles we have learned this semester, design and implement an object model to represent a game of RPsLS/RPI.

Create a "Scoring" object that has a method called: "selectWinner". The method should take an array of gestures, and return the index of the gesture that won.

Provide a suite of unit tests that demonstrate your code works as intended. Test both scoring mechanisms, and all combinations of two-player throws.

Project Requirements

Teams

You may work in teams of **no more than** three (3) people.

Submission and Due Date

The project must be committed to your team's subversion repository no later than 2:00PM on **Friday, September 28, 2012.**

You may choose any one of your team-member's repositories to work on. **You must notify me by 2:00PM Friday Sept 14 of:** (1) your team members, and (2) who's repository you will use so that I can grant access to your team members.

Code Requirements

1. You must conform to Java coding style conventions discussed in class.
2. You must utilize JavaDoc style commenting on all public declarations.
3. You must document uses of all design patterns and principles in your class comments!
4. Your code *must* compile, and *must* pass all unit tests.

Note: that part of your grade will be based on the quality of the unit test coverage - so deleting tests that don't work will hurt you too.

Extra Credit

Provide a GamePlay object that allows rounds to be played interactively, with the following requirements:

1. A player may be a human user, or the computer.
2. A game may either be 2-player, or 3-or-more player. For two player games, play three rounds; for 3-or-more players, play until only one player remains.

You must use the Strategy Pattern to implement the Player and Game structures.

Scoring: Your extra credit will be scored independently as either "no credit," "half-credit" or "full credit," and may be used at the end of the semester to add up to a full letter grade to ANY exam or project of your choice this semester.

OFFICE HOURS SUPPORT

Office hours help will be provided for the project during regular office hours, or by appointment. **The entire team** must be present for office hours help.

If you come to office hours **without UML**, I will ask you to leave.

TEAM ISSUES

If you have issues working with members of your team: you must work them out! I'm happy to facilitate or moderate discussions, but ultimately the entire team is responsible for the work, and will be held accountable as a group.

KEY PATTERNS

- Strategy Pattern
- Decorator Pattern
- Factory Pattern

DESIGN PRINCIPLES

- Isolate things that vary! (p9)
- Program to an Interface! (p11)
- Favor Composition over Inheritance (p23)
- Classes should be open for extension, but closed to change! (p86)