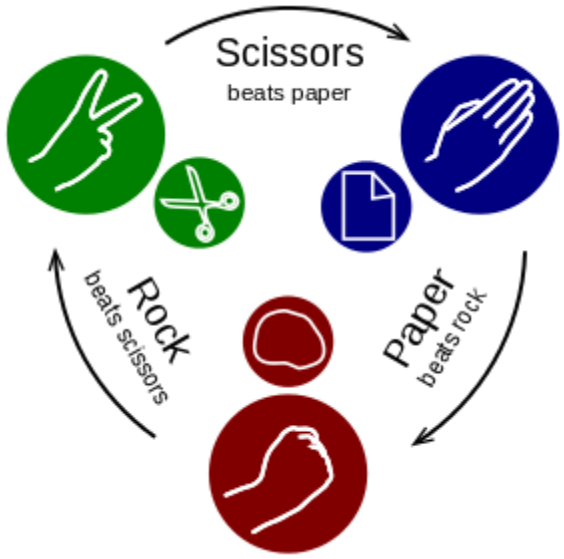
Rock Paper Scissors

Rock-paper-scissors is a hand game usually played by two people, where players simultaneously form one of three shapes with an outstretched hand. The "rock" beats scissors, the "scissors" beat paper and the "paper" beats rock; if both players throw the same shape, the game is tied.



Rock-paper-scissors figure (from http://en.wikipedia.org/wiki/File:Rock-paper-scissors.svg)

Java Implementation

Design assumptions – The following assumptions are made in the Java implementation of this game.

- This is a two player game.
- No 3rd-party library is used except for testing.

The overall implementation design of this game is done using the ***Mediator*** design pattern.

Controller

Based on the *mediator* design pattern, the object that encapsulates how a set of objects interact is `imran.RockPaperScissors`. The class `imran.RockPaperScissor` is also the main application containing the 'main' method. In this method, the game is run using fluent interface. Most of the coding is designed based on loose coupling principle and therefore interaction is done against interfaces instead of concrete instances.

Dependency Injection

Since there are no libraries used with the exception of testing libraries, a class `imran.context.ApplicationContext` is used to apply Inversion of Control (dependency injection) and the clean code principle of using Composition over Inheritance. The Singleton design pattern is used to get an instance of this application context. This application context is not unit tested. In the production code, Spring (http://spring.io/) framework can be used for Inversion of Control.

Model

The `representation` package contains the following representations. Each of the items in the representation defines the data that is used in the interaction with a user of this game.

- Mode – Game mode, which could be a player versus computer or a computer versus computer.
- Move – Move made to play this game such as Rock or Paper or Scissors
- Result – The result of the game. This is assumed that this game is played between two players.
    - Outcome – The outcome of the game, either Win or Lose.
    - Reason – The reason for the result, e.g Rock beats Scissors or Paper beats Rock.
    - First player – The player making the first move.
    - Second player – The player making the second move.

The `model` package contains the following entities:

- Player – The entity containing player name and the move made. This interface is implemented by `RockPaperScissorsPlayer` class.
- Player type – The type of the player, e.g. Person or Computer.
- Rules – The Rules interface defines a method to get result for two players playing this game. The Strategy design pattern is used to implement Rules interface in the class `imran.model.rules.RockPaperScissorsRules`.
- Rule – The Rule interface defines various strategies to get the result of this game. The implementations of this interface are: `RockBeatsScissorsRule`, `PaperBeatsRockRule`, and `ScissorsBeatPaperRule`.

View

The `view` package contains the logic for the interaction with user. This is implemented in `imran.view.InteractiveConsole` which uses java.io.Console for command line interaction.

Validation

Validators in the package `validator` are used to validate input entered by the user of this game. There are two concrete classes to validate game mode selection and move selection. A further improvement to this could be extraction of a Validator interface which uses Java generics to specify parameters for validation.

Providers

Factory design pattern is used to implement providers for the game mode. For move provider, an interactive move provider is used for the end user to select a move and a randomised move provider issued to choose a randomly generated move for the computer.

Service

The `service` package contains the Game interface which allows two players to play this game and returns the game result.

Error

The game errors are implemented using an exception hierarchy with imran.error.GameException as the root class. There are two error conditions that further sub class this to provide error for invalid game mode selection or invalid move.

Unit Testing

Unit testing is done using JUnit, Mockito and Hamcrest and an attempt is made to use BDD-style for unit testing as suggested by the Advance TDD (ATDD) guidelines. There are 56 unit tests.

Acceptance Testing

Using the BDD, acceptance testing is done with the feature "`imran.acceptance.feature.rock-paper-scissors.feature`" which uses Cucumber library with Gherkin and step definition of acceptance tests are implemented using Cucumber-JVM in the `imran.acceptance.step.AcceptanceTestSteps` class. There are 4 scenarios and 21 acceptance tests, which include: Player versus Computer, Computer versus Computer, invalid game mode selection and invalid move.

Further improvements

The following additional improvements can be suggested for a production quality game:

- High quality user interface (requires a front-end developer)
- Dependency injection framework, such as Spring or Guice.
- A 3rd party logging framework, such as SLF4J for debugging.
- Monitoring to keep track of usage KPIs such as statistics for game modes and moves made and win, lost percentages