Vlad Bugayev, Natthapong Choeypant
CSCI 345
7/22/19

Deadwood Program
Coupling and Cohesion Report

Coupling

As we reflect on our design, we notice that there is more coupling than we would like. Multiple classes depend on instances of one another, with the specific instances being shared between multiple classes (such as the Board object for the Deadwood class and the GameMaster class). This makes testing without Mocks difficult and results in data coupling. We also have control coupling present, as the GameMaster class provides approval of and executes specific, risky processes (such as bank payout and player rank upgrades that should be controlled). We have message coupling from exception handling, as the Deadwood class operation depends on other class outputs from try-catch blocks. We chose to use the coupling that we did because it felt like the most logical and easy to follow implementation of the Deadwood game.

Cohesion

Our design exhibits a strong level of functional cohesion, as many of the methods and variables located in classes such as Player.java all provide functionality that the players must have to play the game (methods like PlayerMove, PlayerAct, etc). Communicational cohesion exists within these classes, as all components that operate on the same data are grouped together. We chose to group things by logical objects (such as players, gameboard, etc), and provided functionality within all modules that were needed to execute the game.