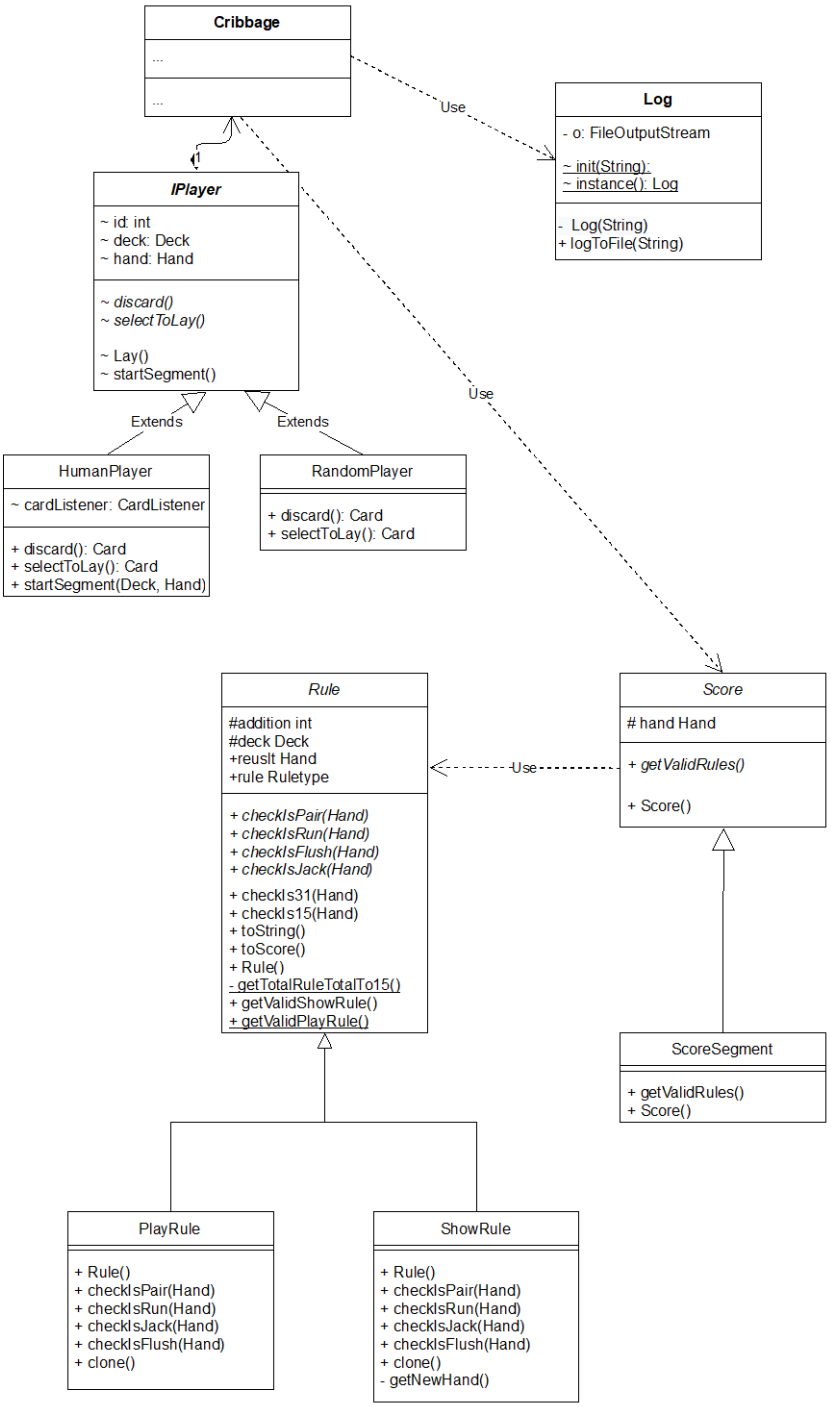
W09 Team 03

In the project, our group uses the following design pattern, such as Factory Method, Singleton, Adapter, Observer. What's more, the principles, such as Open-Closed Principles, Liskov Substitution Principles, Interface Segregation Principle, are also reflected in this assignment. Following graph is our design presentation.



In detail, Yuhang Wang designs the class named Rule which demonstrates the rules of the game. The Rule class is an abstract class which leaves four abstract methods to be implemented. The abstract methods are checkIsPair, checkIsRun, checkIsFlush and checkIsJack, corresponding to the Pair, Run, Flush, Jack rule in the assignments. The PlayRule and ShowRule are derived from the Rule, which are referenced to the two different score way, Play and Show. The PlayRule and ShowRule override four methods, with the implementation described from the doc. What's more, the Rule also defines the interface such as getValidRule to be looked up for every action by player. So the players can easily get the total rules when they choose the next card to play. And Yuhang also overrides the toString for Rule and clone for the PlayRule and Show Rule, so it can easily change the rule to log format and be cloned and pass it to Cribbage. In Yuhang's opinion, this change conforms to the Factory Method of the design pattern and Interface Segregation Principle because the original code in the project is unchanged, and user override the new method to meet the new demand. The Rule accord to the Factory Method, and The PlayRule and Show Rule meet the interface Segregation Principle.

I am Da Zhang. I implemented the new class named Log. The Log is a Singleton, the reason why I choose singleton is that all objects in the runtime should add some log which is tracked to the one Log object, so the Log would only have one instance. And I used the lazy creation method in Singleton, which means that the obj would be created when the method is first to be invoked. In my opinion, Log is a best practice to Singleton of the design pattern. Besides, the implementation of the Log is like a module, and all of methods that operate the data are member functions, which means low coupling. The Log does not depend on other modules or codes, and the elements within the Log are directly related to the functionality that module is meant to provide, so it is high cohesion. Moreover, the Adapter is embodied in the Score class. The Score class provides the interface to lookup the total valid rules both in play stage and show stage. In addition, the scoreSegment is derived from the Score, implementing the real interface for getting valid play rules. I think that meets the Interface Segregation Principle.

From a holistic perspective, zhengtian lu provides the base principles.Zhengtian lu advises the team follows the Liskov Substitution Principles. In our solutions, the Rule is a base class, and they can be replaced with the PlayRule or ShowRule without any change. And zhengtian lu also said we should follow the Dependency Inversion Principle, which means people should program to interface relies on abstraction rather than on concrete. The interface we abstracted in Rule, Score, Log is work fined in the project. Above all, we also keep to Open-Closed Principles. The new method is implemented for new features and the old code is not changed.