Team Number: W05 Team 05 [Wed 05:15PM]

Scoring

For scoring, we chose to use a strategy pattern, this was because of the different ways in which the game is scored, and because of the potential variation in the way the game is played. Protected variation was considered allowing for the introduction or removal of new gameplay rules in the forms of strategies in scoring in the future, adding extendability (see figure 1, IScoringStrategy interface). We then used a composite strategy pattern to model the game's rules being a combination of many smaller rules, which again added flexibility in polymorphism when scoring the game by encapsulating the standard rule-set into a 'normal rules' class (figure 1, compositeNormalRules…. strategies). This allows for the creation of variant rule-sets in the future, further encouraging protected variation.

The reason we split the composite strategies into one for show and one for play is to account for the possibility of a future ruleset involving only an alteration in one of those categories. For example, if one wants to have a variant play rule-set but keep normal show rules, then only a 'variantNormalPlayStrategy' class would need to be created, reducing the amount of work needed to be done to extend the system.

The factory pattern was considered for the initialisation of the strategy patterns in figure 1, but we decided that having the composite strategy instantiate the strategies made sense from the information expert. We also chose not to use this pattern because the composite strategy classes weren't complex or bloated, so assigning them responsibility of creating strategies wouldn't reduce the cohesion of the classes. In addition some cribbage nested classes were moved into their own public class for accessibility via the scoring strategy patterns. Considering the bloatedness of the cribbage class we felt it was appropriate to move some of the card logic outside of the class.

When discussing how cribbage should interact with the scoring subsystem, we considered moving the scoring into player, but figured it would lower the cohesion of the player class so thus decided to use a facade object pattern instead (see figure 1, 'facadeScoring'). This was used to abstract the scoring logic for the system into a single point of contact, adding indirection in the scoring process through pure fabrication (see figures 2 and 3, design sequence diagrams of scoring situations). The facadeScoring class is able to ripple the calls to score down to each individual strategy (figure 1), which is where the scoring logic lies by information expert.

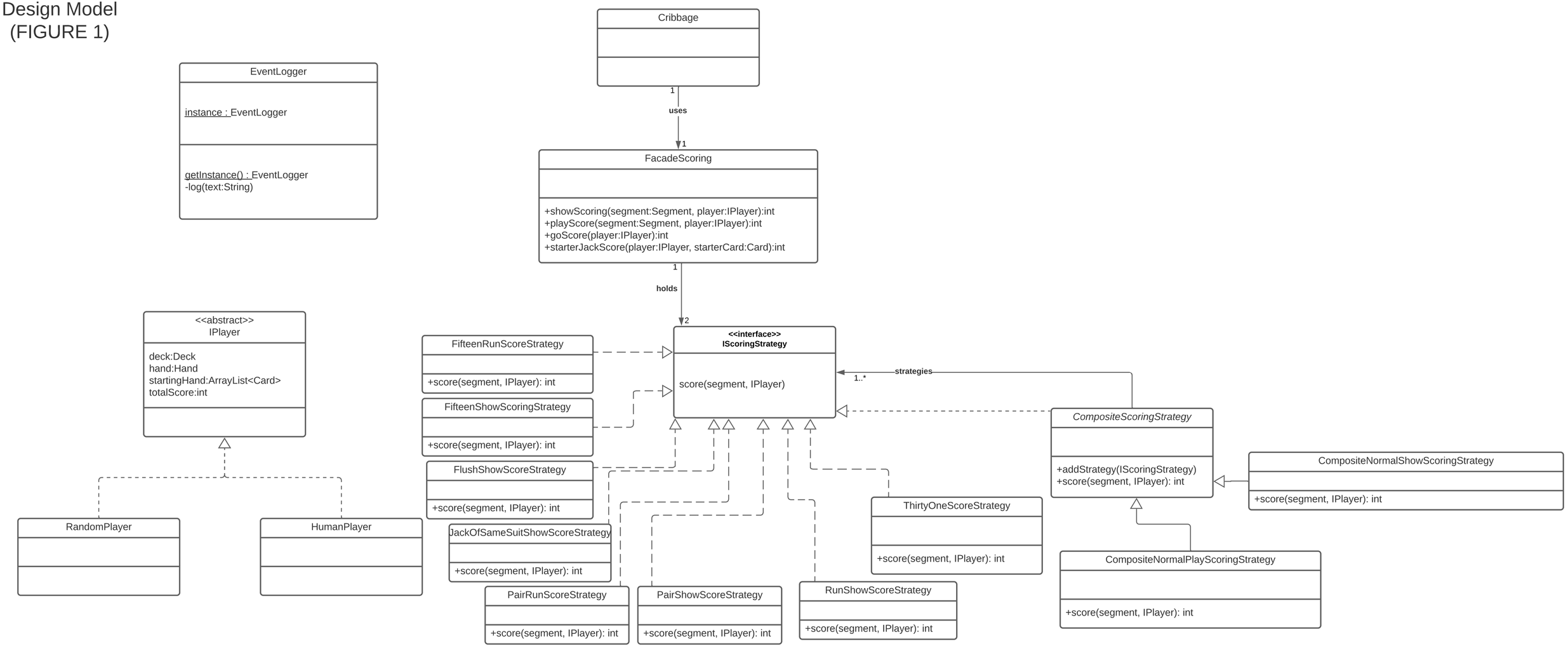
This tremendously lowered the overall coupling of the system; the otherwise complex composite strategy pattern is all wrapped into a single subsystem of scoring, which the Cribbage class accesses via method calls to the facade.

Logging
We initially thought the logging logic would be in Cribbage by the information expert principle, but the class would be too bloated so we moved all logging into eventLogger, a pure fabrication singleton to be used to encapsulate all the logging logic for the different rules, maintaining high cohesion (see figures 4 and 5 to see how logging system works). This also separates the logging logic into its own subsystem (see figure 1) which further increases cohesion.
We also considered putting the logging logic and log calls inside the scoring strategy classes by the information expert principle. However, putting together these two very separate concepts lowers cohesion, thus we kept them separate.

We also discussed potentially using an observer pattern to observe all instances of scoring and then log accordingly, however this approach - with considerations of information expert - added far too much coupling and dependencies into the system, so we decided against it.

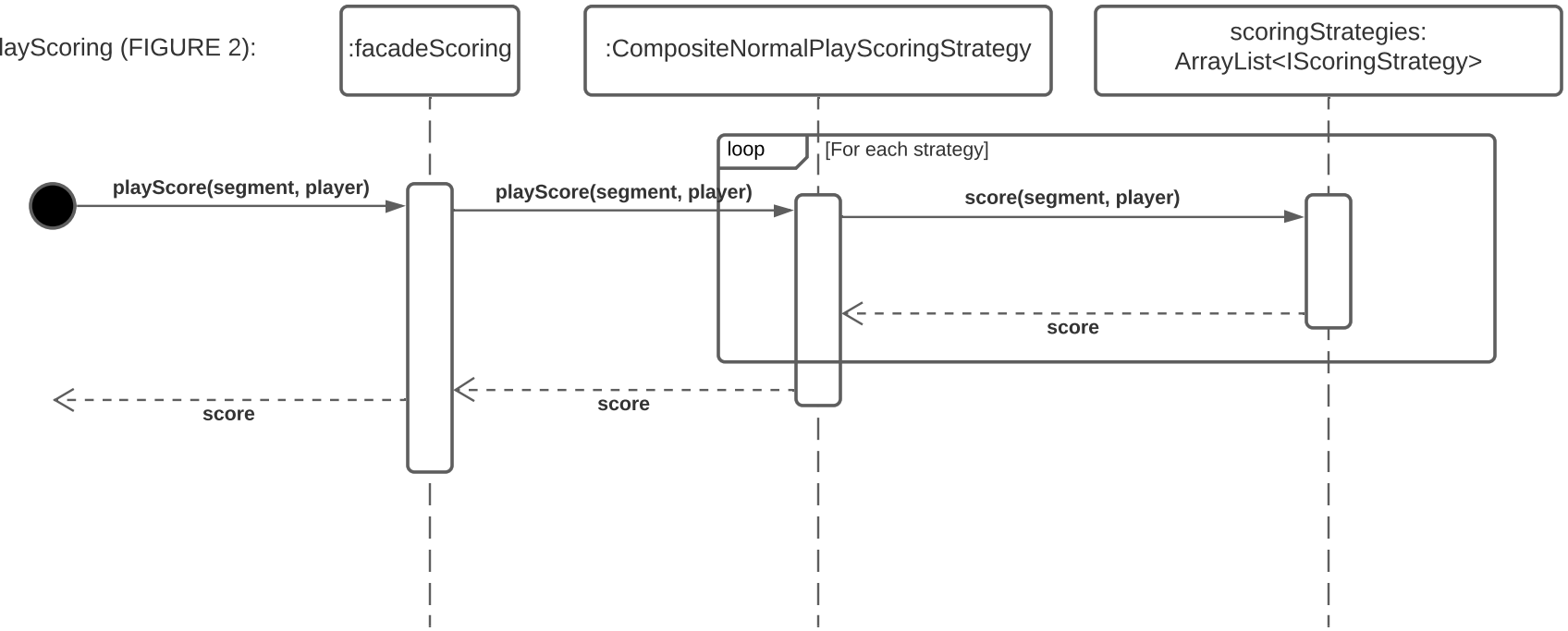
Assumptions made
There's an assumption that the player who plays the last card of the game scores ‘go’ regardless of circumstance. For instance, if the last card played brings the total to 31 and player 1 plays the last card, player 1 will score 2 points for 31 and 1 for go, in that order.

Design Model
(FIGURE 1)

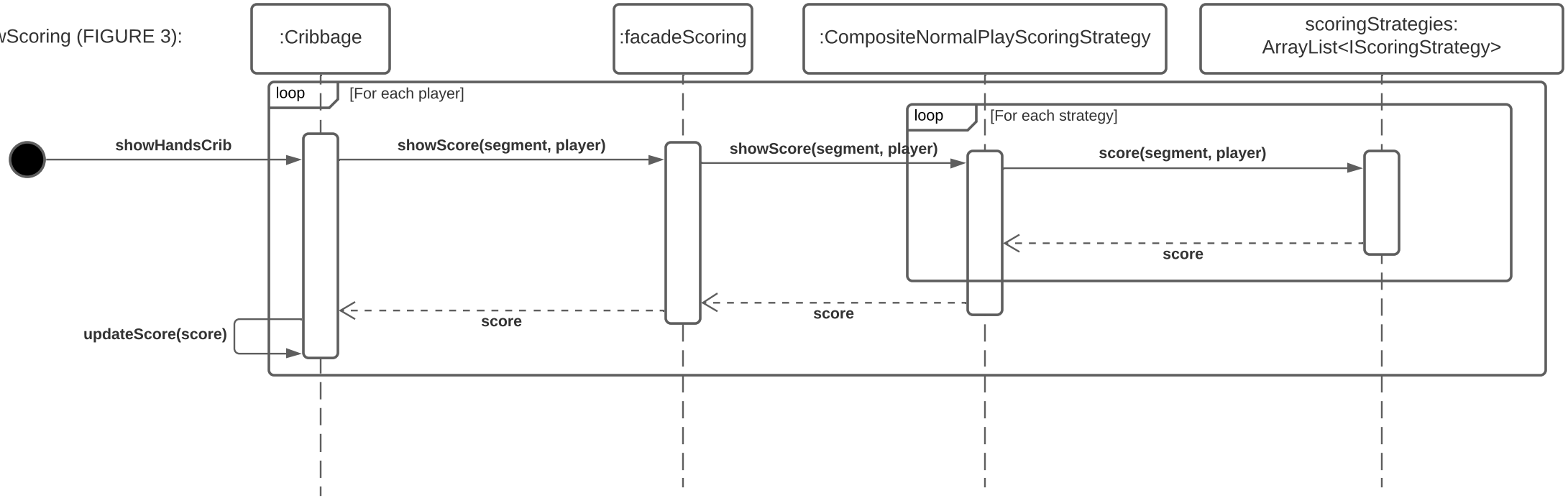


Scoring Design Sequence Diagram

PlayScoring (FIGURE 2):

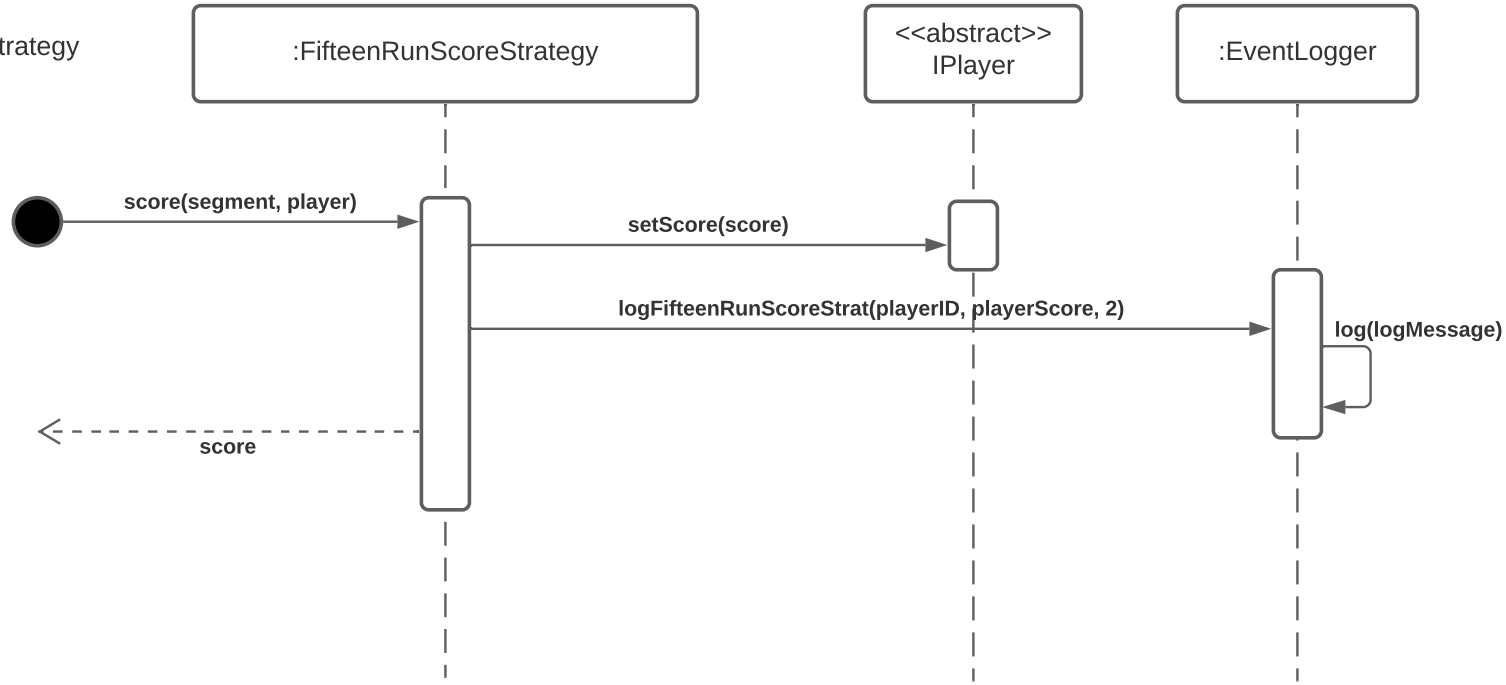


ShowScoring (FIGURE 3):



Logging Design Sequence Diagram

LoggingForScoringStrategy (FIGURE 4):



LoggingForDeal (FIGURE 5):