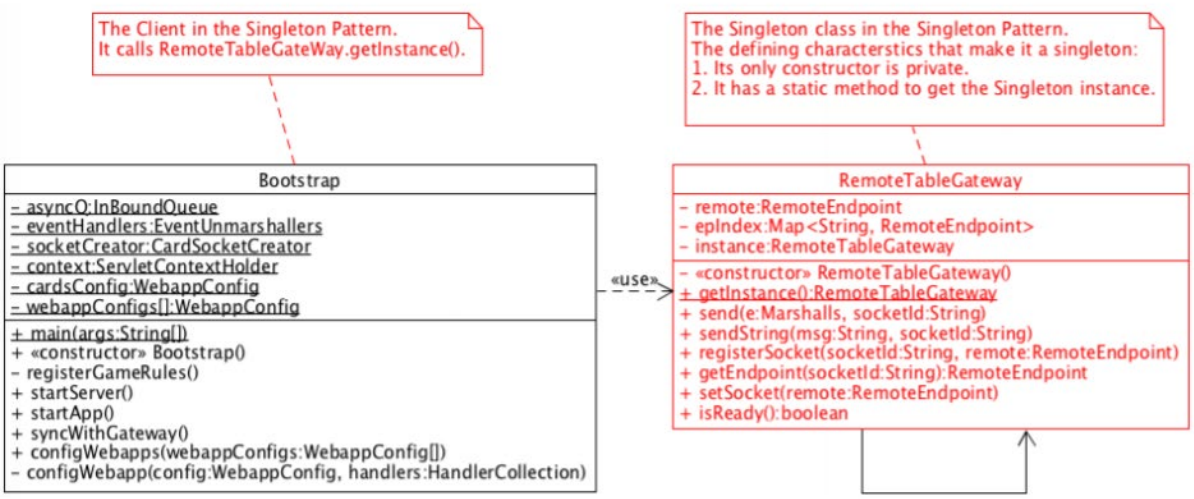
Steven Sheets

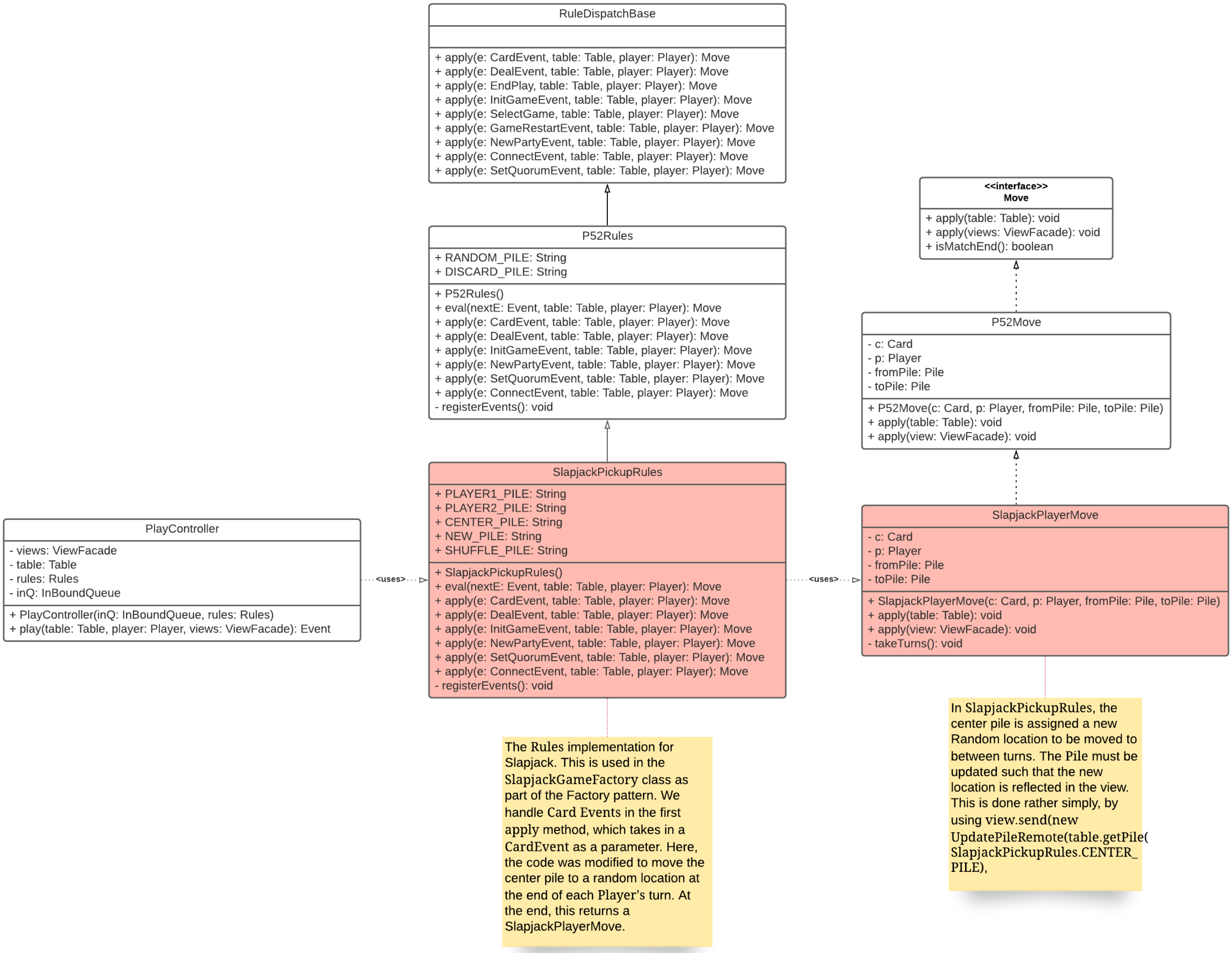
4/20/2021

COM S 362: Object-Oriented Analysis and Design – Group Project, Iteration 3

1. (40 points) Provide a UML class diagram that clearly describes one design pattern in the cards framework you directly used (e.g., added or modified a class that extends an interface in the pattern) to implement one of your individual features. The diagram should show the current state of the project and contain notes that label the parts of the design pattern (e.g., this is the `Client` class, this is the `Singleton` class, etc.) Include only the classes/interfaces required to fully describe the design pattern in the context of the cards framework. Clearly mark what you added or modified. For example, suppose you created a singleton called `RemoteTableGateway` (that class already exists, but for the purpose of the example suppose it didn't), your diagram would look something like this.


The Client in the Singleton Pattern.
It calls RemoteTableGateWay.getInstance().
The Singleton class in the Singleton Pattern.
The defining characterstics that make it a singleton:
1. Its only constructor is private.
2. It has a static method to get the Singleton instance.
Bootstrap
- asyncQ:InBoundQueue
- eventHandlers:EventUnmarshallers
- socketCreator:CardSocketCreator
- context:ServletContextHolder
- cardsConfig:WebappConfig
- webappConfigs[]:WebappConfig
+ main(args:String[])
+ «constructor» Bootstrap()
- registerGameRules()
+ startServer()
+ startApp()
+ syncWithGateway()
+ configWebapps(webappConfigs:WebappConfig[])
- configWebapp(config:WebappConfig, handlers:HandlerCollection)
«use»
RemoteTableGateway
- remote:RemoteEndpoint
- epIndex:Map<String, RemoteEndpoint>
- instance:RemoteTableGateway
- «constructor» RemoteTableGateway()
+ getInstance():RemoteTableGateway
+ send(e:Marshalls, socketId:String)
+ sendString(msg:String, socketId:String)
+ registerSocket(socketId:String, remote:RemoteEndpoint)
+ getEndpoint(socketId:String):RemoteEndpoint
+ setSocket(remote:RemoteEndpoint)
+ isReady():boolean

- See next page…


RuleDispatchBase
+ apply(e: CardEvent, table: Table, player: Player): Move
+ apply(e: DealEvent, table: Table, player: Player): Move
+ apply(e: EndPlay, table: Table, player: Player): Move
+ apply(e: InitGameEvent, table: Table, player: Player): Move
+ apply(e: SelectGame, table: Table, player: Player): Move
+ apply(e: GameRestartEvent, table: Table, player: Player): Move
+ apply(e: NewPartyEvent, table: Table, player: Player): Move
+ apply(e: ConnectEvent, table: Table, player: Player): Move
+ apply(e: SetQuorumEvent, table: Table, player: Player): Move
<<interface>>
Move
+ apply(table: Table): void
+ apply(views: ViewFacade): void
+ isMatchEnd(): boolean
P52Rules
+ RANDOM_PILE: String
+ DISCARD_PILE: String
+ P52Rules()
+ eval(nextE: Event, table: Table, player: Player): Move
+ apply(e: CardEvent, table: Table, player: Player): Move
+ apply(e: DealEvent, table: Table, player: Player): Move
+ apply(e: InitGameEvent, table: Table, player: Player): Move
+ apply(e: NewPartyEvent, table: Table, player: Player): Move
+ apply(e: SetQuorumEvent, table: Table, player: Player): Move
+ apply(e: ConnectEvent, table: Table, player: Player): Move
- registerEvents(): void
P52Move
- c: Card
- p: Player
- fromPile: Pile
- toPile: Pile
+ P52Move(c: Card, p: Player, fromPile: Pile, toPile: Pile)
+ apply(table: Table): void
+ apply(view: ViewFacade): void
PlayController
- views: ViewFacade
- table: Table
- rules: Rules
- inQ: InBoundQueue
+ PlayController(inQ: InBoundQueue, rules: Rules)
+ play(table: Table, player: Player, views: ViewFacade): Event
<uses>
SlapjackPickupRules
+ PLAYER1_PILE: String
+ PLAYER2_PILE: String
+ CENTER_PILE: String
+ NEW_PILE: String
+ SHUFFLE_PILE: String
+ SlapjackPickupRules()
+ eval(nextE: Event, table: Table, player: Player): Move
+ apply(e: CardEvent, table: Table, player: Player): Move
+ apply(e: DealEvent, table: Table, player: Player): Move
+ apply(e: InitGameEvent, table: Table, player: Player): Move
+ apply(e: NewPartyEvent, table: Table, player: Player): Move
+ apply(e: SetQuorumEvent, table: Table, player: Player): Move
+ apply(e: ConnectEvent, table: Table, player: Player): Move
- registerEvents(): void
<uses>
SlapjackPlayerMove
- c: Card
- p: Player
- fromPile: Pile
- toPile: Pile
+ SlapjackPlayerMove(c: Card, p: Player, fromPile: Pile, toPile: Pile)
+ apply(table: Table): void
+ apply(view: ViewFacade): void
- takeTurns(): void

The Rules implementation for Slapjack. This is used in the SlapjackGameFactory class as part of the Factory pattern. We handle Card Events in the first apply method, which takes in a CardEvent as a parameter. Here, the code was modified to move the center pile to a random location at the end of each Player's turn. At the end, this returns a SlapjackPlayerMove.

In SlapjackPickupRules, the center pile is assigned a new Random location to be moved to between turns. The Pile must be updated such that the new location is reflected in the view. This is done rather simply, by using view.send(new UpdatePileRemote(table.getPile(SlapjackPickupRules.CENTER_ PILE),

2. Provide a link to your team's repository.

- https://git.ece.iastate.edu/andmarek/cards362

3. List the two features that you attempted and describe the current state of each (e.g., completed with no known errors, mostly complete but some errors, not complete, not attempted, not able to test due to team integration issues, etc.)

- At the start of play there should be a deal button and the title of the game should be set to "Slapjack"
  - Completed with no known errors.
    - Game starts, deal button exists, is clickable, title of game is "Slapjack".
- Pressing the deal button results in two even piles of shuffled face down cards. There are many simple shuffle algorithms, any is fine.
  - Completed with no known erros.
    - Clicking the deal button results in two piles, each containing 26 shuffled face down cards.

4. List of all files that you created or modified to complete the two features. Give the full package and class names (e.g., `coms362.cards.fiftytwo.P52InitCmd`).

- `coms362.cards.slapjack.SlapjackInitCmd`
- `coms362.cards.slapjack.SlapjackDealCommand`
- `coms362.cards.slapjack.SlapjackPickupRules`

5. What aspects of the project went well?

- There were a few team members that went pretty above and beyond (however, this did have some problems, as will be noted in the next question). We managed to completely implement everything, including the optional requirements, so rather happy with that. However, our implementation certainly wasn't optimal, as it feels too complex.

6. What aspects of the project didn't go well?

- Perhaps the biggest problem was the lack of a common vision. At some points during the project it felt like we had completely different notions as to how things should be setup. E.g., I thought the

SJMove class should have 2 Players, but we ended up going with only 1. This resulted in some weird implementation, but it works for the most part.

- Our communication could have been better. I *was* able to meet with every team member, however, I would say that only 3 of them were down for regular meetings. The other two were incredibly busy. Or, emotionally not in a good place, as these times seem to generate.

7. Provide additional feedback about the project if any.

- I understand why the demo would be cancelled – presumably it's a big ask of the TAs, and you definitely aren't wrong. With that being said, though, it's still a bit awkward not having a demo for this. Namely because there was *so much* concurrency with what was created, as much of it was done sitting together in Carver. Additionally, we wanted a demo because we're mildly confused as to how it's graded. Initially, the intention was that we would each be assigned 2 features, but after working on it, some proved to be far easier than others. For example, the first feature described in Q3 was insanely easy to implement, and, as one may expect, didn't really add any complexity to the system. I was "assigned" this. What we ended up doing was a "everyone work on everything" system, and it was… not very well organized, but it worked, for the most part.
- This may sound silly, considering I only attended a couple office hours, but yea, generally having more office hours for a project like this would be useful. As noted earlier, we had conflicting visions for how this project should work – it certainly isn't optimally designed. More opportunities to discuss with someone who knows an optimal implementation would be great to have the confidence to push a certain way of doing it.
- If, when this is all said and done, if a sample solution could be posted, that would be awesome, as it would enable us to see the shortcomings in our design.