Other Considered Design Class Diagram

<<Interface>> IObserverable

+ registerObserver(observer: IObserver): void
+ removeObserver(observer: IObserver): void
+ notifyObserver(data: String): void

observes

<<Interface>> IObserver

+ update(data: String): void

1 · · · 0..*

GameLog   1

- instance: GameLog
- gameLog: FileWriter

+ getInstance():GameLog
- update(data:String): void
- endLog():void

<<Interface>> IConfigurationObservable

+ OBSERVERS: ArrayList<IObserver>

<<Interface>> IPasureObservable

+ OBSERVERS: ArrayList<IObserver>

<<Interface>> IPlayerObservable

+ OBSERVERS: ArrayList<IObserver>

extends / extends / extends / has a

implements

Configuration   1

- SEED_KEY: String = "Seed"
- ANIMATE_KEY: String = "Animate"
- PLAYER0_KEY: String = "Player0"
- PLAYER1_KEY: String = "Player1"
- configuration: Configuration = null

- seed: int
- animate: boolean
- player0class: String
- player1class: String

- OBSERVERS: ArrayList<IObserver>

+ getInstance(): Configuration
+ setUp(): void
+ registerObserver(observer:IObserver):void
+ removeObserver(observer:IObserver):void
+ notifyObserver(data: String):void

Pasur

+ VERSION: String = "1.0"
+ ON_RESET: String = "onReset"
+ ON_UPDATE_SCORE: String = "onUpdateScore"
+ ON_CARD_TRANSFER: String = "onCardTransfer"
+ ON_GAME_END: String = "onGameEnd"

+ random

- SCORE_TO_WIN:int = 62
- N_HAND_CARDS: int = 4
- N_PLAYERS:int

- paused:boolean = true
- gamestarted:boolean = false

- DECK:Deck
- deckHand:Hand
- poolHand:Hand
- players: Player[]

- log:GameLog
- propertyChangePublisher:
- OBSERVERS: ArrayList<IObserver>

+ pauseGame():void
+ resumeGame():void
+ play():void
+ isAsur(playedCard:Card,isLastRound):boolean):boolean
+ reset():void
+ updateScores():void
+ dealingOutToPlayers(currentStartingPlayerPos: int): void
+ dealingOutToPool():void
+ transfer(cards:List<Card>,h Hand,sortAfterTransfer:boolean)

<<Abstract>> Player

- TARGET_VALUE: int = 11
id: int
hand: Hand
pickedCards: Hand
surs: Hand

scoreBeforeRound : int
scoreFacade : ScoreFacade
- OBSERVERS: ArrayList<IObserver>

+ PLAY_CARD(pool:Hand): Map.Entry<Card,Set<Card>>
# pickCards(pool:Hand, playedCards:Card): Set<Card>
# chooseBestCandidate(candidateSetsOfCardsToPick: List<Set<Card>>):Set<Card>
- findSetsOfCardsSummingToTarget(cards: List<Cards>, targetValue: int, setOfCards:List<Set<Card>>)
- _findSetsOfCardsSummingToTarget(cards: List<Cards>, setOfCards:List<Set<Card>>, targetValue: int, partial:List<Card>)
+ reset: void
+ toString(): String
+ getScore(): int
+ registerObserver(observer:IObserver):void
+ removeObserver(observer:IObserver):void
+ notifyObserver(data: String)

+ abstract selectToPlay: Card

uses

ScoreFacade

- allScoringMethods:ArrayList<IGetScore>

+ instantiateScoringMethods(): void
+ getCurrentScore(pickedCards:Hand,Surs:Hand)

<<enumeration>> Rank

ACE(1)
KING(13)
QUEEN(12)
JACK(11)
TEN(10)
NINE(9)
EIGHT(8)
SEVEN(7)
SIX(6)
FIVE(5)
FOUR(4)
THREE(3)
TWO(2)

- VALUE(): int
+ getValue(): int
+ getCardValuesArray(): int[]
+ canonical():String

<<enumeration>> Suit

CLUBS
DIAMONDS
HEARTS
SPADES

- VALUE(): int
+ getValue(): int
+ getCardValuesArray(): int[]
+ canonical():String

RandomPlayer

+ selectToPlay(): Card

extends

<<Interface>> IGetScore

+ getScore(pickedCards:Hand,Surs:Hand): int

Aces

+ getScore(pickedCards:Hand,Surs:Hand): int

Jacks

+ getScore(pickedCards:Hand,Surs:Hand): int

MostClubs

+ CLUBS_NEEDED: int = 7

+ getScore(pickedCards:Hand,Surs:Hand): int

Surs

+ SURSCORE:int = 5

+ getScore(pickedCards:Hand,Surs:Hand): int

TenOfDiamonds

+ getScore(pickedCards:Hand,Surs:Hand): int

TwoOfClubs

+ getScore(pickedCards:Hand,Surs:Hand): int