System Design and Architecture
Project Black Jack

by Andrea Kaminski, Dajana Berthold, Michael Freiwald, Andreas Mayer, Matthias Müller-Brockhausen, Daniel Sikeler

Building Blocks (Base Models)

For our Blackjack Implementation we have three base models which build upon each other. The primary model is our Blackjack game session which consists of two or more players (secondary model). Each of these players has a hand with cards (cards are our third model) and there is a card deck from which the players draw the cards.

Data structures and functional components of the models

There is a class called Card which represents an individual Card within the Game. A Card has a Symbol (saved as int), a color (saved as int) and a numeric value to determine which card it is (7, king etc.). Of course there is an exception in the card values for the ace card which can be counted as 1 or 11 towards the score of the players hand.

Then there is the hand which is basically a list of the cards (simple array of card class) that were drawn. Additionally it is used to save the wager of the current player during this game session. The Hand also takes care of some game logic implementation. For example it has Methods to check whether another Card may be drawn, whether or not the wager may be incremented and whether certain actions like double down or split are currently allowed.

For the Game itself there is the Deck Class. This is the list of the Cards that can be given to the players. This Class is responsible for making sure each card only gets into the hands of the players once per shuffling.

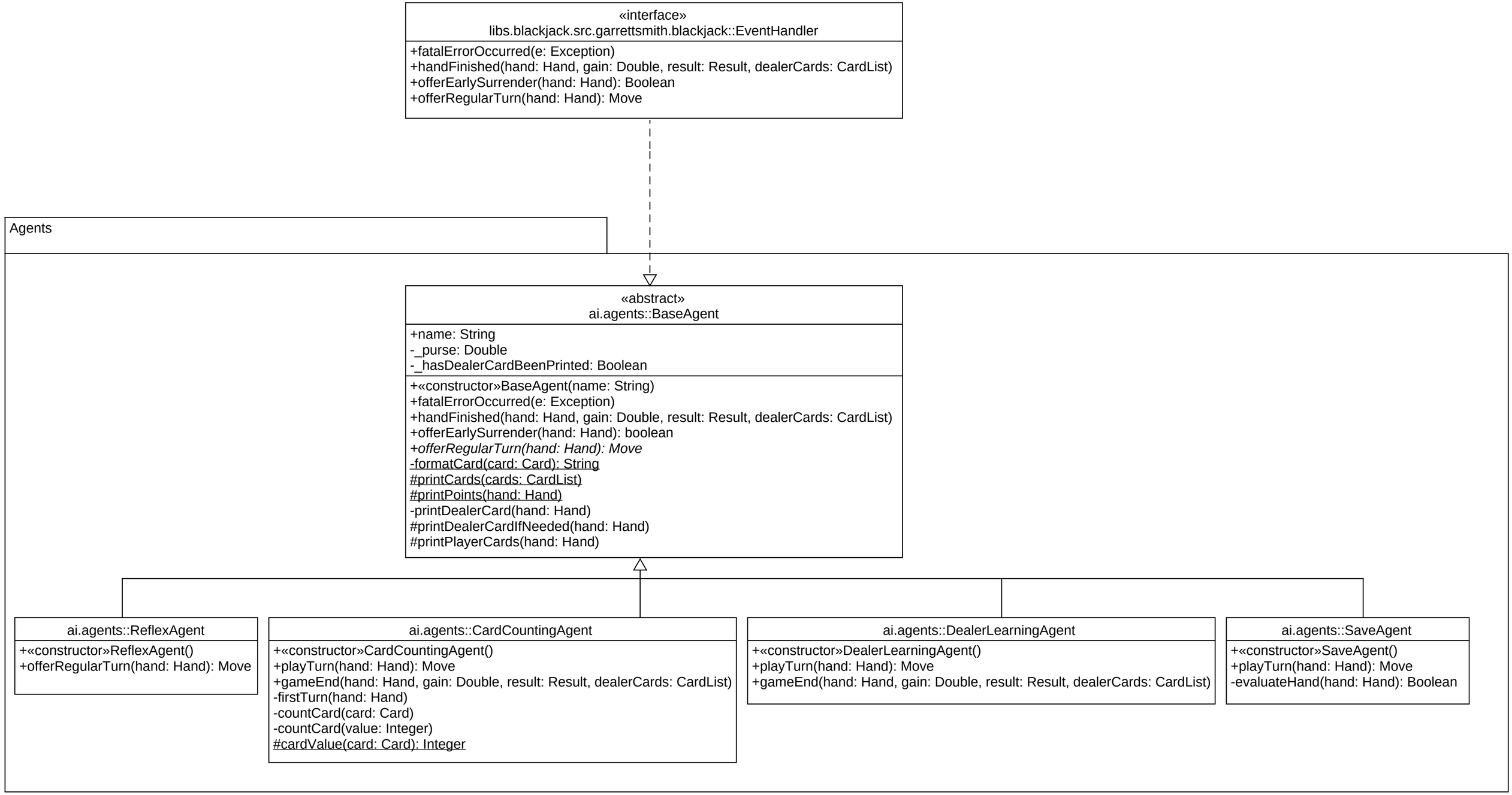
We also have a Class that represents the Players called BaseAgent. BaseAgents are the “players” of the Game. The BaseAgent is our “Interface” to the Game to give the ability to use different Agenttypes and choose between them at the start of the Game. In the Method offerRegularTurn all Logic is done and the Answer is of the enum Move which tells the Blackjack Class what Action the Player wants to take.

The Move Enum contains all valid Actions such as Hit, Stand, Surrender, Douple, and Split.

Then there is our Blackjack Class. This class acts as the Dealer for the Game and hence has a datafield of the Deck class. It also keeps Information about which Players participate and takes care of the play order. The Playerinteraction is given through the offerRegularTurn Method mentioned earlier.

UML-Diagram displaying our Application Design and Architecture

(First Picture only our implementation Part, Second Picture the complete Architecture)


«interface»
libs.blackjack.src.garrettsmith.blackjack::EventHandler
+fatalErrorOccurred(e: Exception)
+handFinished(hand: Hand, gain: Double, result: Result, dealerCards: CardList)
+offerEarlySurrender(hand: Hand): Boolean
+offerRegularTurn(hand: Hand): Move
Agents
«abstract»
ai.agents::BaseAgent
+name: String
-_purse: Double
-_hasDealerCardBeenPrinted: Boolean
+«constructor»BaseAgent(name: String)
+fatalErrorOccurred(e: Exception)
+handFinished(hand: Hand, gain: Double, result: Result, dealerCards: CardList)
+offerEarlySurrender(hand: Hand): boolean
+offerRegularTurn(hand: Hand): Move
-formatCard(card: Card): String
#printCards(cards: CardList)
#printPoints(hand: Hand)
-printDealerCard(hand: Hand)
#printDealerCardIfNeeded(hand: Hand)
#printPlayerCards(hand: Hand)
ai.agents::ReflexAgent
+«constructor»ReflexAgent()
+offerRegularTurn(hand: Hand): Move
ai.agents::CardCountingAgent
+«constructor»CardCountingAgent()
+playTurn(hand: Hand): Move
+gameEnd(hand: Hand, gain: Double, result: Result, dealerCards: CardList)
-firstTurn(hand: Hand)
-countCard(card: Card)
-countCard(value: Integer)
#cardValue(card: Card): Integer
ai.agents::DealerLearningAgent
+«constructor»DealerLearningAgent()
+playTurn(hand: Hand): Move
+gameEnd(hand: Hand, gain: Double, result: Result, dealerCards: CardList)
ai.agents::SaveAgent
+«constructor»SaveAgent()
+playTurn(hand: Hand): Move
-evaluateHand(hand: Hand): Boolean

Game

libs.blackjack.src.garrettsmith.playingcards::Card

-_suit: Suit
-_value: Value
-_color: Color
-_hashCode: int

+«constructor»Card()
+«constructor»Card(value: Value, suit: Suit)
+equals(object Object): Boolean
+getColor(): Color
+getSuit(): Suit
+getValue(): Value
+hashCode(): Integer
+toString(): String

+Color
+Suit
+Value

libs.blackjack.src.garrettsmith.playingcards::CardList

-serialVersionUID: Long

+«constructor»CardList()
+«constructor»CardList(capacity: Integer)
+«constructor»CardList(cards: CardList)

+add(o: Card): Boolean
+getCard(index: Integer): Card
+removeCard(i: Integer): Card

libs.blackjack.src.garrettsmith.blackjack::Hand

-_cards: CardList
-_blackjack: Blackjack
-_isPlayerDone: Boolean
-_isDealerPlayNeeded: Boolean
-_wager: Double
-_splitCounter: SplitCounter
-_isEvaluationNeeded: Boolean

~«constructor»Hand(wager: Double, blackjack: Blackjack, cards: CardList)
-«constructor»Hand(wager: Double, blackjack: Blackjack, newCards: CardList, splitCounter: SplitCounter)
+getBestValue(): Integer
+getValue(): Integer
+getCards(): CardList
+getDealerCard(): Card
+getDealerValue(): Integer
+isBusted(): Boolean
+isDoubleDownAllowd(): Boolean
+isStandAllowed(): Boolean
+getWager(): Double
+isMoveAllowed(move: Move): Boolea
+isHitAllowed(): Boolean
+isSoft(): Boolean
+isSplitAllowed(): Boolean
+toString(): String
+isSurrenderAllowed(): Boolean
~doubleDown() {throws NotAllowedException, NoMoreCardsException}
~isInPlay(): Boolean
~isEvaluationNeeded(): Boolean
~surrender()
~isDealerPlayNeeded(): Boolean
~getSplitCount(): Integer
~hit() {throws NotAllowedException, NoMoreCardsException}
~isPlayerDone(): Boolean
~split(): Hand
~stand() {throws NotAllowedException}
~addCard()
-hasSplit(): Boolean

-SplitCounter

libs.blackjack.src.garrettsmith.blackjack::Blackjack

-_cards: CardContainer
-_dealerCards: CardList
-_rules: Rules
-_handsToPlay: ArrayList<Hand>
-_handsToEvaluate: ArrayList<Hand>

+«constructor»Blackjack()
+«constructor»Blackjack(rules: Rules)
+calculateBestValue(cards: CardList): Integer
~cardsToString(cards: CardList): String
~cardToString(card: Card): String
+calculateValue(card: Card): Integer
+calculateValue(cards: CardList): Integer
+getRules(): Rules
+isNonAceFaceCard(card: Card): Boolean
+isBlackjack(cards: CardList): Boolean
+isSoft(cards: CardList): Boolean
+playGame(handler: EventHndler, wager: Double)
+setRules(rules: Rules)
~getDealerCard(): Card
~getCard(): Card {throws NoMoreCardsException}
~setCards(cards: CardContainer)
-isDealerPlayNeeded(hands: ArrayList<Hand>): Boolean
-evaluateOutcome(dealerValue: Integer, handler: EventHandler)
-isBlackjack(hand: Hand, handler: EventHandler): Boolean
-offerEarlySurrender(hand: Hand, handler: EventHandler): Boolean
-dealInitialCards(wager: Double): Hand
-playDealerHand(): Integer
-playTurn(handler: EventHandler, hand: Hand)

«interface»
libs.blackjack.src.garrettsmith.blackjack::Move

+HIT: Move
+STAND: Move
+SURRENDER: Move
+DOUBLE: Move
+SPLIT: Move

+value(): Integer
+execute(hand: Hand, handler: EventHandler, dealerCards: CardList)

«abstract»
BaseMove {implements Move}

+value(): Integer
+hashCode(): Integer
+equals(o: Object): Boolean
+execute(hand: Hand, handler: EventHandler, dealerCards: CardList): Hand
/#doMove(hand: Hand, handler: EventHandler, dealerCards: CardList): Hand

Hit {extends BaseMove}
Stand {extends BaseMove}
Surrender {extends BaseMove}
Double {extends BaseMove}
Split {extends BaseMove}

«interface»
libs.blackjack.src.garrettsmith.playingcards::Deck

+getAllCards(): List<Card>
+size(): Integer
+getNumCardsOfType(color: Card.Color)
+getNumCardsOfType(suit: Card.Suit)
+getNumCardsOfType(value: Card.Value)

libs.blackjack.src.garrettsmith.playingcards::StandardDeck

_cards: List<Card>
+DECK: StandardDeck

-«constructor»StandardDeck()
+getAllCards(): List<Card>
+size(): Integer
+toString(): String
+getNumCardsOfType(color: Color): Integer
+getNumCardsOfType(suit: Suit): Integer
+getNumCardsOfType(value: Value): Integer

«interface»
libs.blackjack.src.garrettsmith.blackjack::EventHandler

+fatalErrorOccurred(e: Exception)
+handFinished(hand: Hand, gain: Double, result: Result, dealerCards: CardList)
+offerEarlySurrender(hand: Hand): Boolean
+offerRegularTurn(hand: Hand): Move

Agents

«abstract»
ai.agents::BaseAgent

+name: String
-_purse: Double
-_hasDealerCardBeenPrinted: Boolean

+«constructor»BaseAgent(name: String)
+fatalErrorOccurred(e: Exception)
+handFinished(hand: Hand, gain: Double, result: Result, dealerCards: CardList)
+offerEarlySurrender(hand: Hand): boolean
+offerRegularTurn(hand: Hand): Move
-formatCard(card: Card): String
#printCards(cards: CardList)
#printPoints(hand: Hand)
-printDealerCard(hand: Hand)
#printDealerCardIfNeeded(hand: Hand)
#printPlayerCards(hand: Hand)

ai.agents::ReflexAgent

+«constructor»ReflexAgent()
+offerRegularTurn(hand: Hand): Move

ai.agents::CardCountingAgent

+«constructor»CardCountingAgent()
+playTurn(hand: Hand): Move
+gameEnd(hand: Hand, gain: Double, result: Result, dealerCards: CardList)
-firstTurn(hand: Hand)
-countCard(card: Card)
-countCard(value: Integer)
#cardValue(card: Card): Integer

ai.agents::DealerLearningAgent

+«constructor»DealerLearningAgent()
+playTurn(hand: Hand): Move
+gameEnd(hand: Hand, gain: Double, result: Result, dealerCards: CardList)

ai.agents::SaveAgent

+«constructor»SaveAgent()
+playTurn(hand: Hand): Move
-evaluateHand(hand: Hand): Boolean