Blackjack! Presentation

By Nolawee, Shawn, Steve, and Jose

What we did: *(made Blackjack!)*

Demo

2 Use Cases:

1) Play and win a round against a dealer—win a bet

2) Bet everything—lose to the dealer, results in a game over

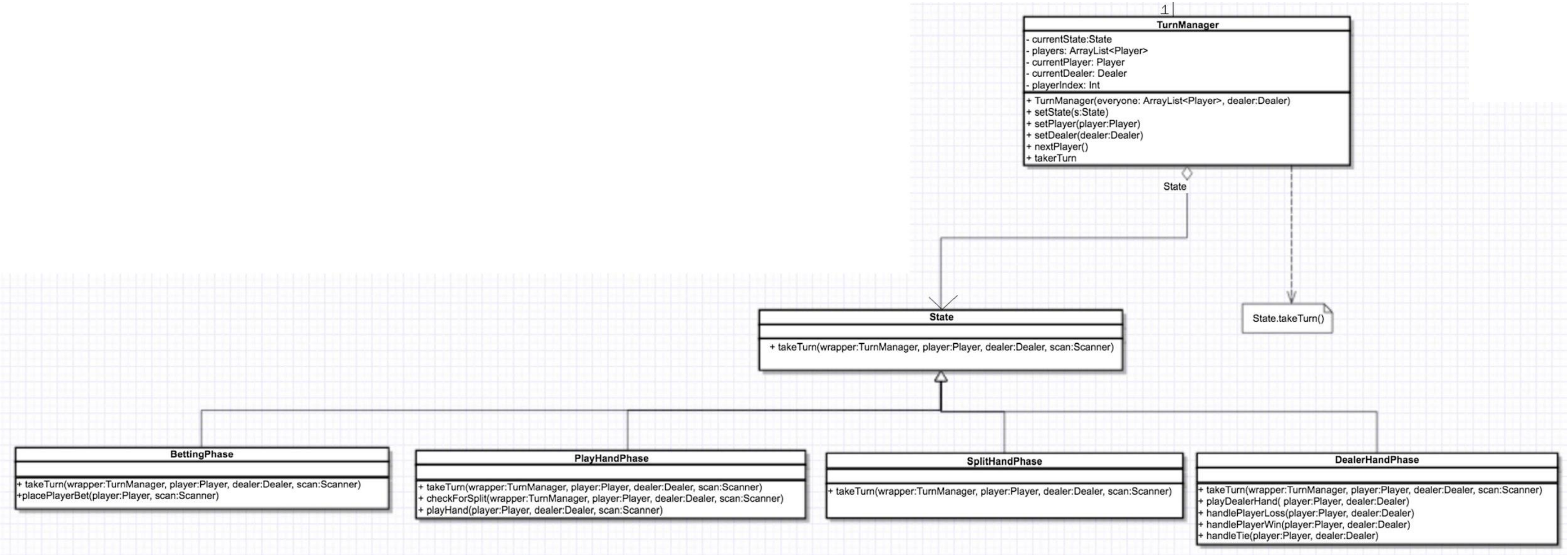
Demo video

Design Patterns

Design Patterns:

- State
    - Handle turns and phases (betting, play hand, etc.)
- Memento
    - Save/load game state

- **STATE**


1
TurnManager
- currentState:State
- players: ArrayList<Player>
- currentPlayer: Player
- currentDealer: Dealer
- playerIndex: Int
+ TurnManager(everyone: ArrayList<Player>, dealer:Dealer)
+ setState(s:State)
+ setPlayer(player:Player)
+ setDealer(dealer:Dealer)
+ nextPlayer()
+ takerTurn
State
State.takeTurn()
State
+ takeTurn(wrapper:TurnManager, player:Player, dealer:Dealer, scan:Scanner)
BettingPhase
+ takeTurn(wrapper:TurnManager, player:Player, dealer:Dealer, scan:Scanner)
+placePlayerBet(player:Player, scan:Scanner)
PlayHandPhase
+ takeTurn(wrapper:TurnManager, player:Player, dealer:Dealer, scan:Scanner)
+ checkForSplit(wrapper:TurnManager, player:Player, dealer:Dealer, scan:Scanner)
+ playHand(player:Player, dealer:Dealer, scan:Scanner)
SplitHandPhase
+ takeTurn(wrapper:TurnManager, player:Player, dealer:Dealer, scan:Scanner)
DealerHandPhase
+ takeTurn(wrapper:TurnManager, player:Player, dealer:Dealer, scan:Scanner)
+ playDealerHand(player:Player, dealer:Dealer)
+ handlePlayerLoss(player:Player, dealer:Dealer)
+ handlePlayerWin(player:Player, dealer:Dealer)
+ handleTie(player:Player, dealer:Dealer)

● MEMENTO

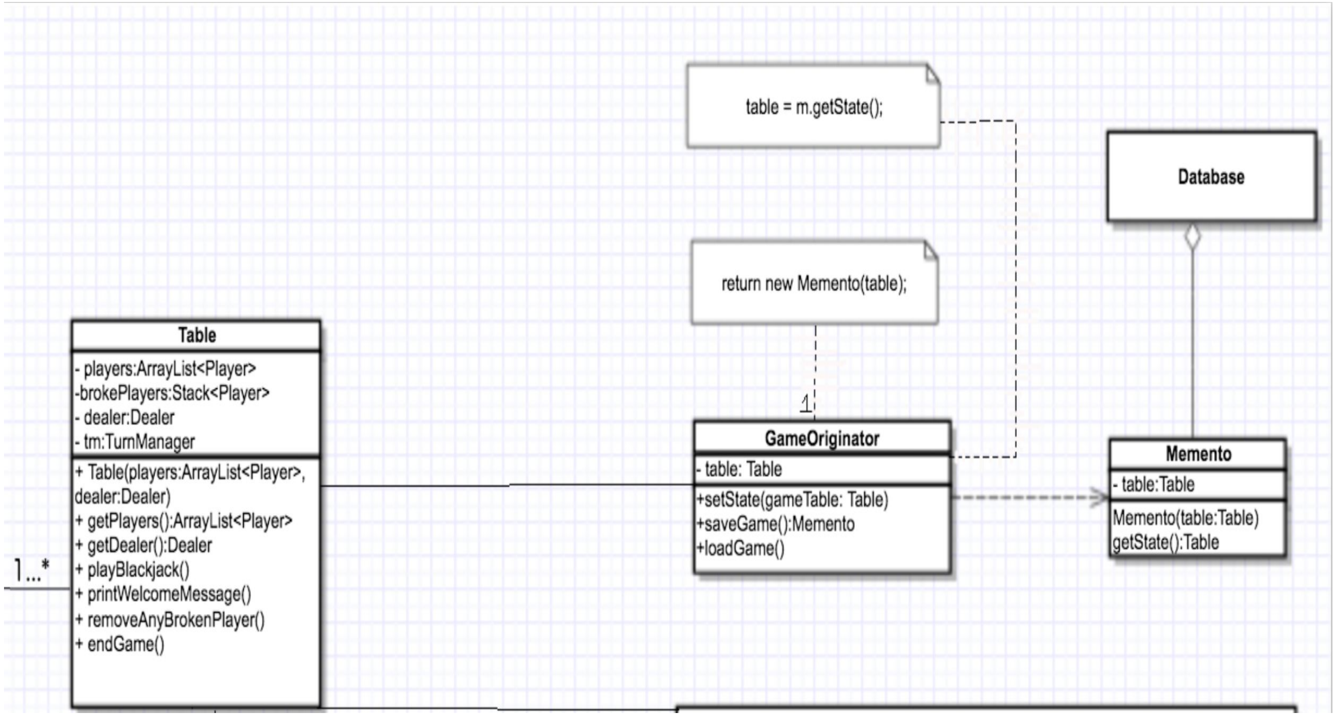


table = m.getState();

return new Memento(table);

Database

Table

- players:ArrayList<Player>
-brokePlayers:Stack<Player>
- dealer:Dealer
- tm:TurnManager

+ Table(players:ArrayList<Player>, dealer:Dealer)
+ getPlayers():ArrayList<Player>
+ getDealer():Dealer
+ playBlackjack()
+ printWelcomeMessage()
+ removeAnyBrokenPlayer()
+ endGame()

1...*

1

GameOriginator

- table: Table

+setState(gameTable: Table)
+saveGame():Memento
+loadGame()

Memento

- table:Table

Memento(table:Table)
getState():Table

Interesting Stuff

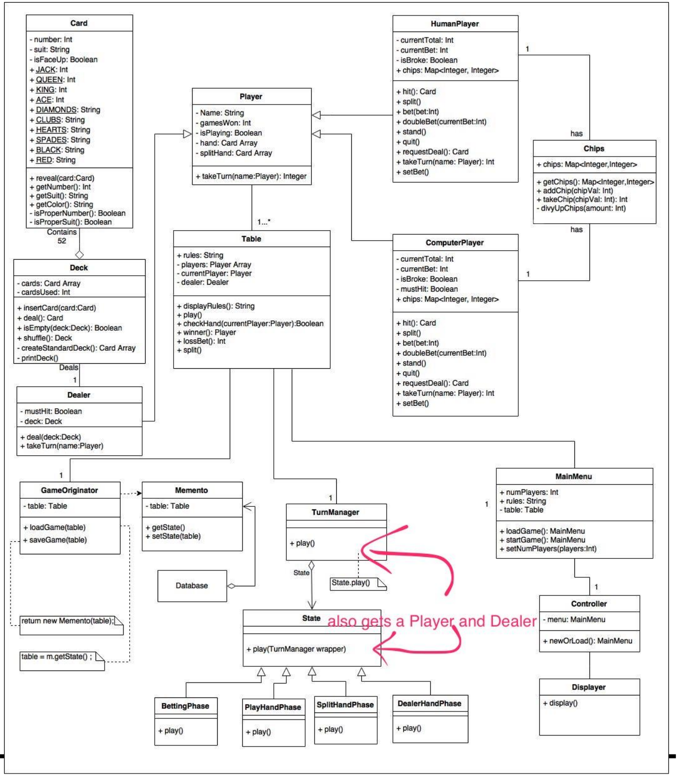
Interesting stuff people might learn from:

- Modeling the real word
- Readable code
- High-level simplicity

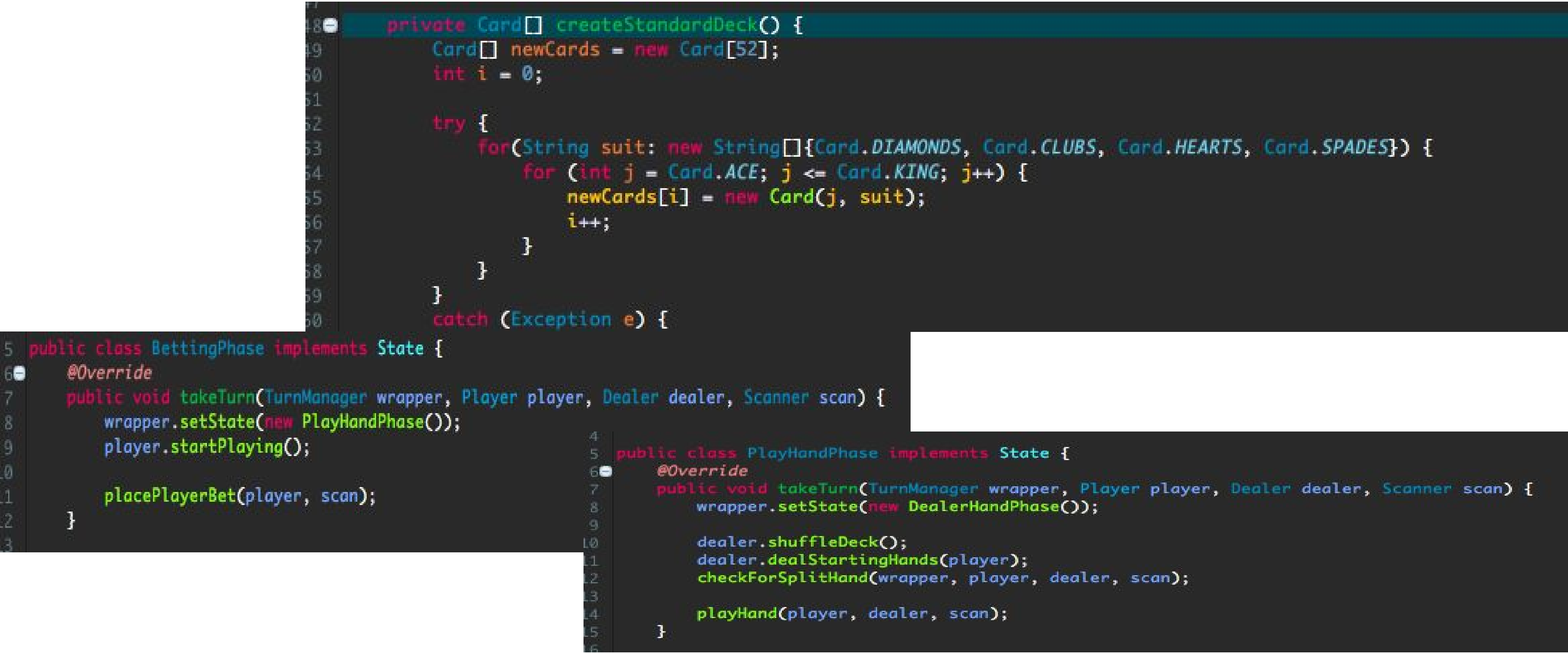
● Modeling the real word

- *(**Use updated class diagram.** And of course, zoom in on specific nodes because Boese said **not** to show whole diagram at once)*

- *(Talk about how classes faithfully represent real-world objects)*
  - *(e.g. how we bet by moving around chips instead of using simple numbers)*



- **Readable code**



```
    private Card[] createStandardDeck() {
        Card[] newCards = new Card[52];
        int i = 0;

        try {
            for(String suit: new String[]{Card.DIAMONDS, Card.CLUBS, Card.HEARTS, Card.SPADES}) {
                for (int j = Card.ACE; j <= Card.KING; j++) {
                    newCards[i] = new Card(j, suit);
                    i++;
                }
            }
        }
        catch (Exception e) {
```

```
public class BettingPhase implements State {
    @Override
    public void takeTurn(TurnManager wrapper, Player player, Dealer dealer, Scanner scan) {
        wrapper.setState(new PlayHandPhase());
        player.startPlaying();

        placePlayerBet(player, scan);
    }
```

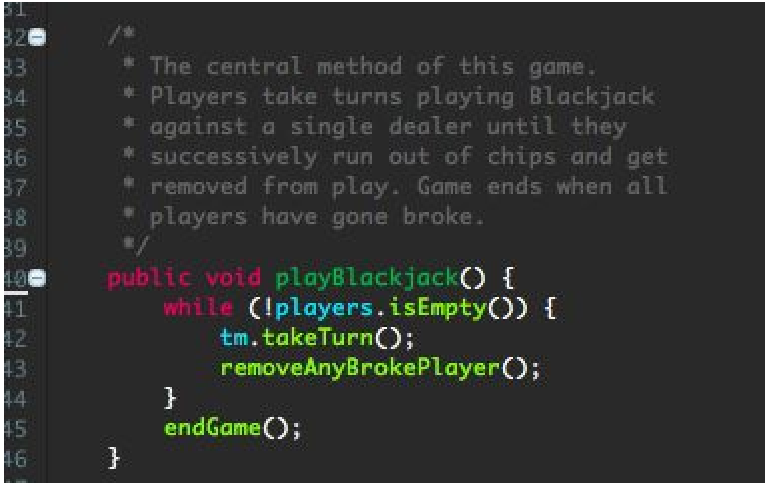
```
public class PlayHandPhase implements State {
    @Override
    public void takeTurn(TurnManager wrapper, Player player, Dealer dealer, Scanner scan) {
        wrapper.setState(new DealerHandPhase());

        dealer.shuffleDeck();
        dealer.dealStartingHands(player);
        checkForSplitHand(wrapper, player, dealer, scan);

        playHand(player, dealer, scan);
    }
```

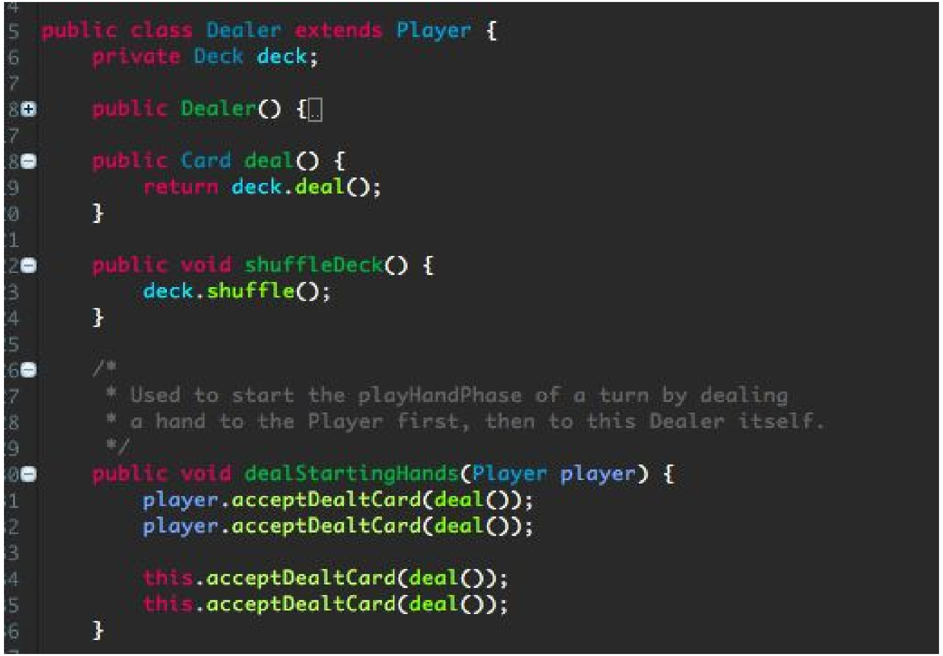
- **High-level simplicity**

TABLE



```
/*
 * The central method of this game.
 * Players take turns playing Blackjack
 * against a single dealer until they
 * successively run out of chips and get
 * removed from play. Game ends when all
 * players have gone broke.
 */
public void playBlackjack() {
    while (!players.isEmpty()) {
        tm.takeTurn();
        removeAnyBrokePlayer();
    }
    endGame();
}
```

DEALER



```
public class Dealer extends Player {
    private Deck deck;

    public Dealer() {...

    public Card deal() {
        return deck.deal();
    }

    public void shuffleDeck() {
        deck.shuffle();
    }

    /*
     * Used to start the playHandPhase of a turn by dealing
     * a hand to the Player first, then to this Dealer itself.
     */
    public void dealStartingHands(Player player) {
        player.acceptDealtCard(deal());
        player.acceptDealtCard(deal());

        this.acceptDealtCard(deal());
        this.acceptDealtCard(deal());
    }
```

FIN. *(...and the crowd goes wild…)*