Different approaches to tacking design:

- Structured Approach.
- Agile approach.
- Prototyping Approach.
- Rapid Application Development Approach.
- End-User Approach.

We chose the structured approach. This involves defining, planning, building, checking and modifying the solution. The defining part has been done by the project requirements. We then plan by deciding which design pattern to use, drawing class diagram and standardising coding practices. We then build the solution and check using test cases to make sure the solution is working correctly. The solution is then tested by playing the game and using different strategies (aggressive, passive) and checking the results, which calls for modification to be done. This is an iterative process. We chose this approach as it is the least complex approach given that this is a small-scale project.

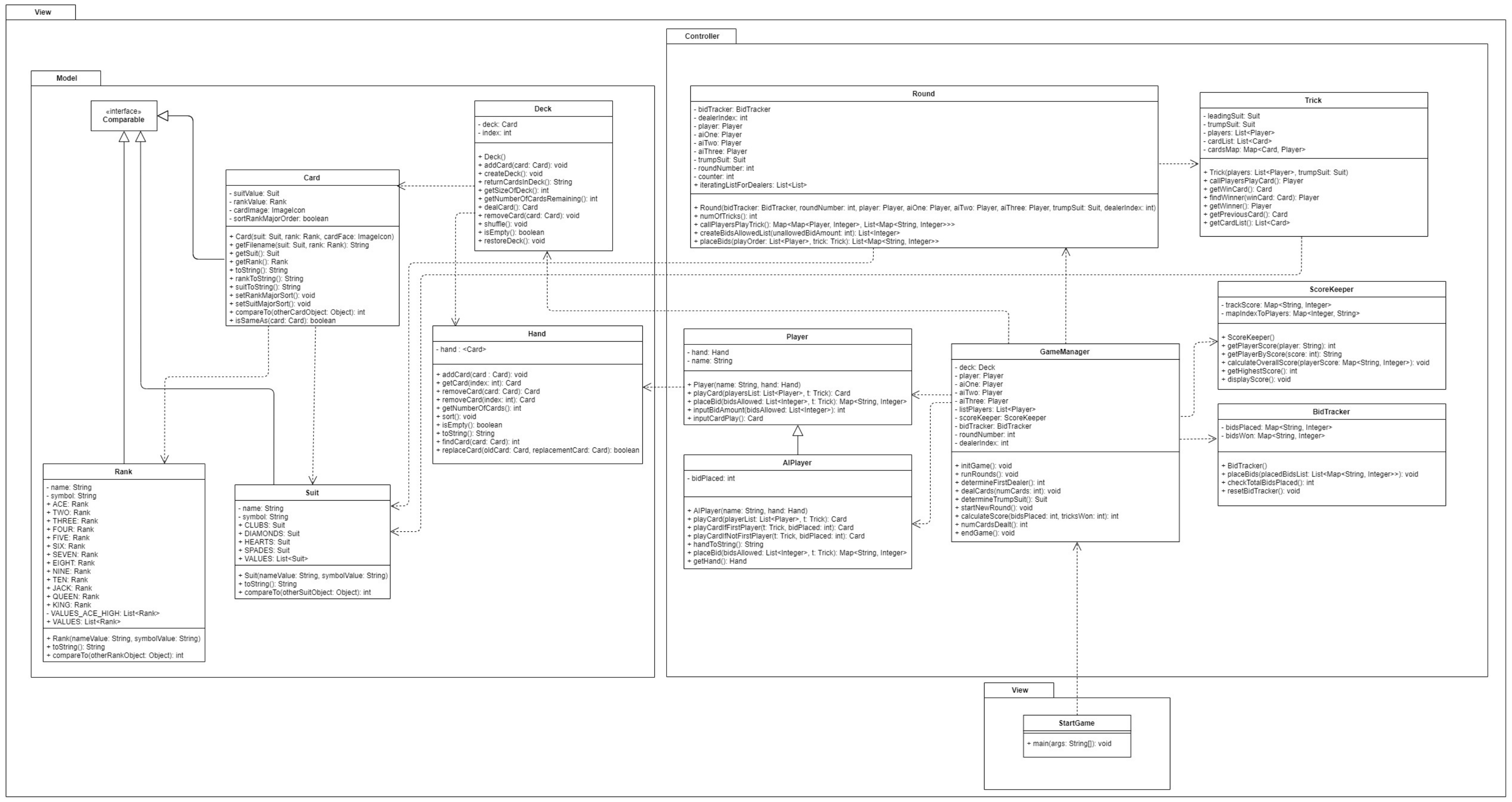
Design patterns applied:

- Model-view-controller paradigm

This is a paradigm where the model contains only the pure application data, it contains no logic describing how to present the data to a user; The View presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it; and The Controller exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. One example is the way we organise our classes. Base classes that only contain data without logic (eg. Card, Rank, Suit) are used as models. Controller manipulates these base classes (eg. Player plays Card) and view allows the user to access the model's data (StartGame allows user to play Card and view scores through ScoreKeeping).

- Creational

This involves class and object creation, using inheritance (eg. AIPlayer inherits Player) for classes and delegation (Rank is used as instance variable in Round) for objects. Cohesion is used within modules and loose coupling is applied across modules. For example, all functions tightly related to Player is contained within Player itself, while Player (plays cards) has separate purposes from Round (records cards played).



View
Model
«interface»
Comparable
Deck
- deck: Card
- index: int
+ Deck()
+ addCard(card: Card): void
+ createDeck(): void
+ returnCardsInDeck(): String
+ getSizeOfDeck(): int
+ getNumberOfCardsRemaining(): int
+ dealCard(): Card
+ removeCard(card: Card): void
+ shuffle(): void
+ isEmpty(): boolean
+ restoreDeck(): void
Card
- suitValue: Suit
- rankValue: Rank
- cardImage: ImageIcon
- sortRankMajorOrder: boolean
+ Card(suit: Suit, rank: Rank, cardFace: ImageIcon)
+ getFilename(suit: Suit, rank: Rank): String
+ getSuit(): Suit
+ getRank(): Rank
+ toString(): String
+ rankToString(): String
+ suitToString(): String
+ setRankMajorSort(): void
+ setSuitMajorSort(): void
+ compareTo(otherCardObject: Object): int
+ isSameAs(card: Card): boolean
Hand
- hand : <Card>
+ addCard(card : Card): void
+ getCard(index: int): Card
+ removeCard(card: Card): Card
+ removeCard(index: int): Card
+ getNumberOfCards(): int
+ sort(): void
+ isEmpty(): boolean
+ toString(): String
+ findCard(card: Card): int
+ replaceCard(oldCard: Card, replacementCard: Card): boolean
Rank
- name: String
- symbol: String
+ ACE: Rank
+ TWO: Rank
+ THREE: Rank
+ FOUR: Rank
+ FIVE: Rank
+ SIX: Rank
+ SEVEN: Rank
+ EIGHT: Rank
+ NINE: Rank
+ TEN: Rank
+ JACK: Rank
+ QUEEN: Rank
+ KING: Rank
- VALUES_ACE_HIGH: List<Rank>
+ VALUES: List<Rank>
+ Rank(nameValue: String, symbolValue: String)
+ toString(): String
+ compareTo(otherRankObject: Object): int
Suit
- name: String
- symbol: String
+ CLUBS: Suit
+ DIAMONDS: Suit
+ HEARTS: Suit
+ SPADES: Suit
+ VALUES: List<Suit>
+ Suit(nameValue: String, symbolValue: String)
+ toString(): String
+ compareTo(otherSuitObject: Object): int
Controller
Round
- bidTracker: BidTracker
- dealerIndex: int
- player: Player
- aiOne: Player
- aiTwo: Player
- aiThree: Player
- trumpSuit: Suit
- roundNumber: int
- counter: int
+ iteratingListForDealers: List<List>
+ Round(bidTracker: BidTracker, roundNumber: int, player: Player, aiOne: Player, aiTwo: Player, aiThree: Player, trumpSuit: Suit, dealerIndex: int)
+ numOfTricks(): int
+ callPlayersPlayTrick(): Map<Map<Player, Integer>, List<Map<String, Integer>>>
+ createBidsAllowedList(unallowedBidAmount: int): List<Integer>
+ placeBids(playOrder: List<Player>, trick: Trick): List<Map<String, Integer>>
Trick
- leadingSuit: Suit
- trumpSuit: Suit
- players: List<Player>
- cardList: List<Card>
- cardsMap: Map<Card, Player>
+ Trick(players: List<Player>, trumpSuit: Suit)
+ callPlayersPlayCard(): Player
+ getWinCard(): Card
+ findWinner(winCard: Card): Player
+ getWinner(): Player
+ getPreviousCard(): Card
+ getCardList(): List<Card>
ScoreKeeper
- trackScore: Map<String, Integer>
- mapIndexToPlayers: Map<Integer, String>
+ ScoreKeeper()
+ getPlayerScore(player: String): int
+ getPlayerByScore(score: int): String
+ calculateOverallScore(playerScore: Map<String, Integer>): void
+ getHighestScore(): int
+ displayScore(): void
Player
- hand: Hand
- name: String
+ Player(name: String, hand: Hand)
+ playCard(playersList: List<Player>, t: Trick): Card
+ placeBid(bidsAllowed: List<Integer>, t: Trick): Map<String, Integer>
+ inputBidAmount(bidsAllowed: List<Integer>): int
+ inputCardPlay(): Card
AIPlayer
- bidPlaced: int
+ AIPlayer(name: String, hand: Hand)
+ playCard(playerList: List<Player>, t: Trick): Card
+ playCardIfFirstPlayer(t: Trick, bidPlaced: int): Card
+ playCardIfNotFirstPlayer(t: Trick, bidPlaced: int): Card
+ handToString(): String
+ placeBid(bidsAllowed: List<Integer>, t: Trick): Map<String, Integer>
+ getHand(): Hand
GameManager
- deck: Deck
- player: Player
- aiOne: Player
- aiTwo: Player
- aiThree: Player
- listPlayers: List<Player>
- scoreKeeper: ScoreKeeper
- bidTracker: BidTracker
- roundNumber: int
- dealerIndex: int
+ initGame(): void
+ runRounds(): void
+ determineFirstDealer(): int
+ dealCards(numCards: int): void
+ determineTrumpSuit(): Suit
+ startNewRound(): void
+ calculateScore(bidsPlaced: int, tricksWon: int): int
+ numCardsDealt(): int
+ endGame(): void
BidTracker
- bidsPlaced: Map<String, Integer>
- bidsWon: Map<String, Integer>
+ BidTracker()
+ placeBids(placedBidsList: List<Map<String, Integer>>): void
+ checkTotalBidsPlaced(): int
+ resetBidTracker(): void
View
StartGame
+ main(args: String[]): void