Definizione delle responsabilità assegnate alle entità/concetti emersi:

Giocatore

- Caratteristiche
  - Nome
  - Carte che possiede (in mano e in tavola)
  - Punteggio
    - Un giocatore bot inoltre ha una difficoltà
- Responsabilità
  - Puo scegliere una carta
  - Può giocare il suo turno

Gioco

- Caratteristiche
  - Giocatori minimi e massimi
  - Tipologia di mazzo utilizzato
- Responsabilità
  - Far cominciare una partita
  - Dare un certo numero di carte per giocatore
  - Dichiarare il vincitore
  - Dare un certo valore alle carte utilizzate (poiché la stessa carta ha valori differenti in base al gioco scelto)

Carta

- Caratteristiche
  - Seme
  - Valore (che gli viene attribuito a seconda del gioco che si è scelto)
  - Numero rappresentato nella carta (che può non corrispondere al valore)
  - La carta può essere visibile a tutti (e quindi scoperta) oppure può essere coperta e quindi visibile solamente a chi la possiede

Mazzo

- Caratteristiche
  - Il mazzo è un'entità differente dalla carta perché rappresenta tutte le carte che possono essere utilizzate nel gioco, perciò in primis una sua caratteristica è l'insieme di carte con cui si gioca
  - Numero di carte attualmente nel mazzo
- Responsabilità
  - Dal mazzo è possibile prendere la carta in cima
  - Dal mazzo è possibile rimuovere una carta specifica
  - Il mazzo può essere mescolato

Queste entità e relative responsabilità sono poi state utilizzate per sviluppare la struttura di una libreria generica che rappresenti tutti i giochi di carte.

Dettagli di implementazione delle responsabilità emerse:

Le **interfacce** e le **classi** presenti nel package della libreria sono poi state specializzate nell'implementazione del gioco **Briscola**. Successivamente sono descritti i comportamenti focali nell'esecuzione e nel corretto funzionamento del gioco di carte implementato.

Scegliere e lanciare la carta (giocatore interattivo)

Un giocatore interattivo può scegliere la carta tramite il metodo .choseCard(). Considerando che un giocatore può scegliere la carta quando è il suo turno osservando le carte in tavola, questo metodo viene chiamato dal .playTurn()

```
@Override
public Card chooseCard() {
    Scanner input = new Scanner( System.in );
    int cardIndex;
    System.out.println(super.getName()+" choose a card [1, 2 or 3]");
    cardIndex = input.nextInt();
    while(cardIndex < 1 || cardIndex > 3) {
        System.out.println("Invalid number! Choose a card [1, 2 or 3]");
        cardIndex = input.nextInt();
    }
    return super.getHandCards().get(cardIndex-1);
}
```

Il metodo .playTurn() chiama infine il metodo .throwThisCard(Card card) poiché la fase successiva nel turno di un giocatore di briscola è lanciare la carta scelta precedentemente

```
public void playTurn() {
    Card chosenCard = this.chooseCard();
    this.throwThisCard(chosenCard);
}
```

Il metodo .throwThisCard(Card card) appunto sposta la carta dalle sue carte in mano alle sue carte in tavola, scoprendola.
La scelta di non rendere il "lanciare la carta" un comportamento a livello di libreria viene dal fatto che potrebbe esistere un gioco in cui non è necessario spostare carte in tavolo ma le carte rimangono solamente nelle mani del giocatore

```
private void throwThisCard(Card card){
    super.getHandCards().remove(card);
    super.getTableCards().add(card);
    card.setIsCovered(false);
}
```

Scegliere e lanciare la carta (giocatore bot)

Un giocatore bot a differenza di uno interattivo, tra le sue carte in mano ne sceglie una randomica.

```
public Card chooseCard() {
    Random r = new Random();
    return super.getHandCards().get(r.nextInt((2) + 1));
}
```

Svolgimento una partita (responsabilita del gioco)

Una partita nella Briscola non finisce finchè il mazzo di carte ha almeno una carta, perciò ad ogni giocatore viene mostrata la situazione del gioco con .printPlayerSituation(player), poi il giocatore gioca il suo turno e viene invocato .playTurn() e se il giocatore è il primo a giocare la sua carta sarà la carta il cui segno comanda la mano. Una volta che tutti i giocatori hanno fatto il proprio turno viene calcolato il vincitore, l'ordine dei giocatori viene ricalcolato, e se ci sono ancora abbastanza carte distribuisce una carta per giocatore, altrimenti viene dichiarato il vincitore della partita in base ai punti

```
public void startGame() {
    turnNumber = 1;
    while(super.deck().getSize() > 0){
        for (Player player : super.getCurrPlayers()){
            this.printPlayerSituation(player);
            player.playTurn();
            if(player.equals(super.getCurrPlayers().get(0))) this.turnCard = player.getTableCards().get(0);
        }
        this.calculateTurnWinner();
        this.reorderPlayer();
        turnNumber++;
        if(super.deck().getSize() >= super.getCurrPlayers().size()) super.giveCardsPerPlayer(1);
        else break;
    }
    this.getWinner();
}
```

Calcolo del vincitore di una mano (responsabilità del gioco)

La singola mano viene calcolata da un metodo privato poiché è solamente un metodo utile al calcolo del vincitore. In base a quante briscole ci sono in tavola viene chiamato il relativo metodo al calcolo del vincitore.
Successivamente viene calcolato il punteggio delle carte in tavola, le carte in tavola vengono coperte e viene incrementato il punteggio di chi ha vinto il turno.

```
private void calculateTurnWinner(){
    Player winnerPlayer = super.getCurrPlayers().get(0);
    int briscolaInTable = this.howManyBriscoleInTable();
    int scoreToAdd = 0;
    if(briscolaInTable == 0) winnerPlayer = this.noBriscoleCase();
    if(briscolaInTable == 1) winnerPlayer = this.oneBriscolaCase();
    if(briscolaInTable > 1) winnerPlayer = this.moreThanOneBriscolaCase();
    for (Player player : super.getCurrPlayers())
        if(player.getTableCards().size() > 0)
            for (Card card : player.getTableCards())
                if(!card.isCovered())
                    scoreToAdd += card.getValue();
    for (Player player : super.getCurrPlayers())
        for (Card card : player.getTableCards())
            card.setIsCovered(true);
    winnerPlayer.setScore(winnerPlayer.getCurrScore() + scoreToAdd);
    this.lastTurnWinner = winnerPlayer;
}
```

nel caso in cui non ci siano briscole si cerca per ogni giocatore, una carta scoperta in tavola che abbia stesso seme della carta che comanda la mano e che abbia valore maggiore. Si salva ad ogni esecuzione la carta con valore maggiore e viene restituito il giocatore che la possiede

```
private Player noBriscoleCase() {
    Player tmpPlayer = super.getCurrPlayers().get(0);
    int higherValue = this.turnCard.getValue();
    for (Player player : super.getCurrPlayers())
        for (Card card : player.getTableCards())
            if(!card.isCovered())
                if (card.getSuit().getName().equals(this.turnCard.getSuit().getName()))
                    if (card.getValue() > higherValue) {
                        tmpPlayer = player;
                        higherValue = card.getValue();
                    }
    return tmpPlayer;
}
```

nel caso ci sia una briscola nel tavolo, allora per ogni giocatore si cerca la carta che abbia stesso seme della briscola e viene salvato il giocatore attuale che poi viene restituito in output

```
private Player oneBriscolaCase(){
    Player playerToReturn = super.getCurrPlayers().get(0);
    for (Player player : super.getCurrPlayers())
        for (Card card : player.getTableCards())
            if (!card.isCovered())
                if (card.getSuit().getName().equals(this.briscolaCard.getSuit().getName()))
                    playerToReturn = player;
    return playerToReturn;
}
```

nel caso in cui le briscole sono piu di una vengono calcolate in questo modo: per ogni carta scoperta in tavola di ogni giocatore, se quella carta ha stesso seme della carta che comanda il gioco e valore maggiore allora vince chi possiede quella carta

```
private Player moreThanOneBriscolaCase(){
    Card higherCard = this.briscolaCard;
    Player playerToReturn = super.getCurrPlayers().get(0);
    for (Player player : super.getCurrPlayers())
        if(player.getTableCards().size() > 0)
            for (Card card : player.getTableCards())
                if(!card.isCovered())
                    if(card.getSuit().getName().equals(higherCard.getSuit().getName())
                        && card.getValue() > higherCard.getValue()){
                        playerToReturn = player;
                        higherCard = card;
                    }
    return playerToReturn;
}
```

ad ogni esecuzione viene utilizzato questo metodo tante volte quante è necessario spostare posizione al fine di far giocare per primo il giocatore che ha vinto l'ultima mano.

```
private void shiftRight(){
    Player tmp = super.getCurrPlayers().get(super.getCurrPlayers().size()-1);
    for(int i = super.getCurrPlayers().size()-1; i > 0; i--)
        super.getCurrPlayers().set(i,super.getCurrPlayers().get(i-1));
    super.getCurrPlayers().set(0, tmp);
}
```

Istruzioni necessarie alla valutazione del progetto:

- Esecuzione del gioco implementato
    - Da terminale nella directory principale del progetto eseguire:
        - ./gradlew build
        - ./gradlew run
- Esecuzione di una partita tra **tre giocatori bot**
    - Modificare riga 26 del file *build.gradle* il *mainClassName* in:
        - mainClassName = 'it.unicam.cs.briscolaImplementation.MainClassExample'
    - Da terminale nella directory principale eseguire:
        - ./gradlew build
        - ./gradlew run