

# Programma DeckLibrary

## Gennaio-Marzo 2022 - Relazione Progetto Programmazione Avanzata

Il seguente programma permette di eseguire il gioco di carte chiamato *briscola*, grazie allo sviluppo di una libreria in grado di implementare giochi di carte.

Per avviare il gioco di carte bisogna scrivere due comandi:

1. *Gradle run* per effettuare il build;
2. *Gradle build* per effettuare il run.

Successivamente sarà visualizzata una console dove giocare tramite terminale.

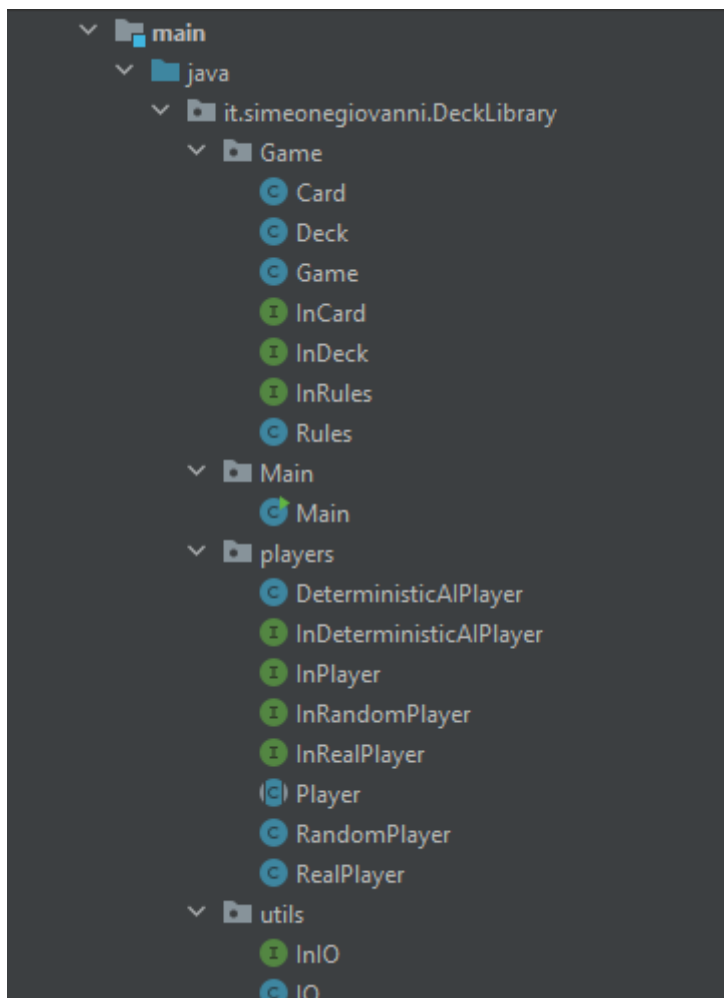


Figura 1

La figura 1 riportata a sinistra presenta le classi del programma.

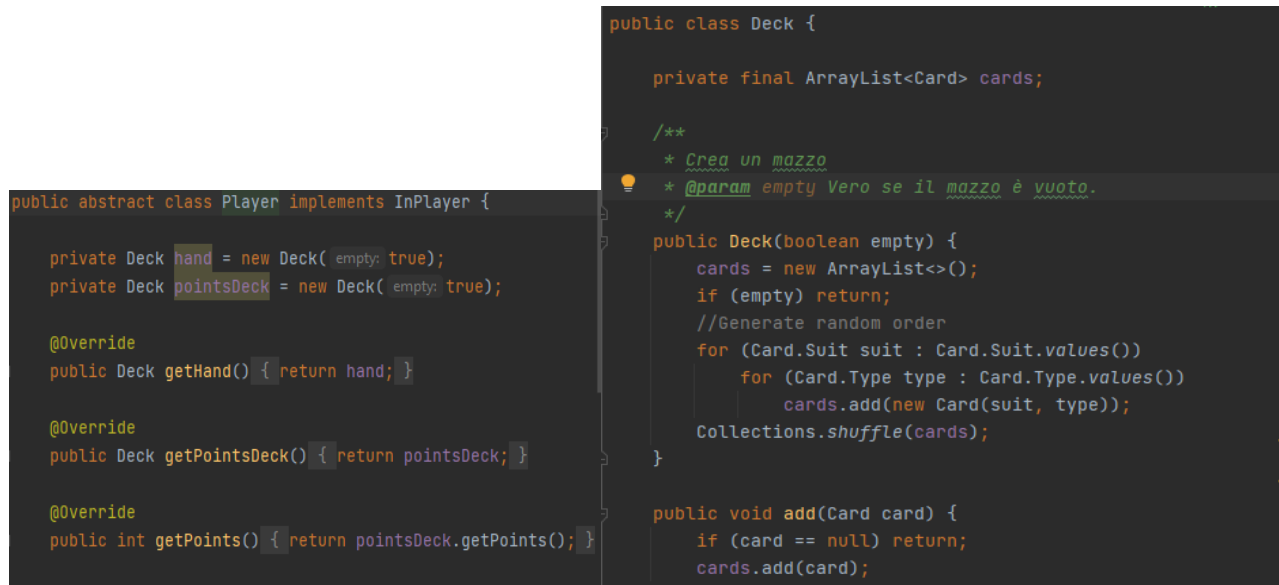
A seguire definiamo il compito di ogni singola classe, come è stato rispettato secondo il **Principio della Singola Responsabilità**:

- **Deck**: è la libreria Java che ha il compito di implementare *giochi di carte*;
- **Card**: è la classe che utilizza la libreria;
- **Deck**: per creare il mazzo di carte *Napoletane*;
- **Game**: è la classe che simula l'intero gioco della *Briscola*;
- **Rules**: è la classe che determina la carta vincente della mano;
- **Main**;
- **DeterministicAIPlayer**: ha il compito di

creare l'IA del bot player;

- **Player:** gestisce le carte dei giocatori;
- **RandomPlayer:** si occupa di dare delle carte casuali nelle mani dei due giocatori;
- **RealPlayer:** gestisce la mano del giocatore reale e le scelte delle carte che gioca;
- **IO:** gestisce le eccezioni.

Riportiamo vari esempi in cui si rispetta il principio Open-closed:



```
public abstract class Player implements InPlayer {  
  
    private Deck hand = new Deck( empty: true);  
    private Deck pointsDeck = new Deck( empty: true);  
  
    @Override  
    public Deck getHand() { return hand; }  
  
    @Override  
    public Deck getPointsDeck() { return pointsDeck; }  
  
    @Override  
    public int getPoints() { return pointsDeck.getPoints(); }  
  
}
```

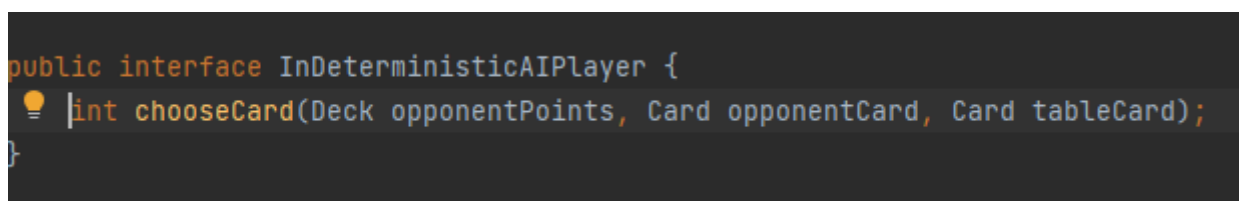
```
public class Deck {  
  
    private final ArrayList<Card> cards;  
  
    /**  
     * Crea un mazzo  
     * @param empty Vero se il mazzo è vuoto.  
     */  
    public Deck(boolean empty) {  
        cards = new ArrayList<>();  
        if (empty) return;  
        //Generate random order  
        for (Card.Suit suit : Card.Suit.values())  
            for (Card.Type type : Card.Type.values())  
                cards.add(new Card(suit, type));  
        Collections.shuffle(cards);  
    }  
  
    public void add(Card card) {  
        if (card == null) return;  
        cards.add(card);  
    }  
  
}
```

Figura 2 Figura 3

Come riportato nella *figura 2* abbiamo un esempio di Open-Closed in cui la classe '**Player**' e la classe '**Deck**':

- sono chiuse perché possono essere compilate, salvate in una libreria ed utilizzate, successivamente, da una persona terza per qualsiasi utilizzo potrebbe essergli utile;
- sono aperte poiché è possibile utilizzarle come classi base oppure aggiungere funzionalità in più se si necessita.

Come è stato presentato inizialmente nella *figura 1* si possono vedere le varie interfacce presenti nel programma. Come è rispettato secondo il **Principio di segregazione delle Interfacce** sono state implementate più interfacce, anziché una singola. La *figura 4* riporta un esempio di interfaccia creata nel programma.



```
public interface InDeterministicAIPlayer {  
  
    int chooseCard(Deck opponentPoints, Card opponentCard, Card tableCard);  
  
}
```

Figura 4

**Il Principio di Inversione di Dipendenza** viene rispettato nel progetto. Infatti possiamo avere molti esempi in cui un implementazione dipende da un interfaccia, come nel caso dell'interfaccia '**InPlayer**'. Si hanno dei metodi dell'astrazione che vengono utilizzati da differenti classi come nel caso della classe '**RandomPlayer**'.

Rispettando il **Principio di Sostituzione di Liskov**, nel progetto, le classi derivate sono in grado di estendere le loro classi base senza modificare il loro comportamento. L'unica classe estesa '**Player**' viene estesa in tre classi diverse:

- **RandomPlayer;**
- **DeterministicAIPlayer;**
- **RealPlayer;**

In tutte e tre le classi non viene mai violato il principio solidi per cui non si subisce nessun cambiamento.

