Mau Mau Projekt

im Modul

komponentenbasierte Entwicklung

Dozent: Prof. Dr. Martin Kempa

Team 6:  
Dustin Lange  
Jörg Lehmann  
Christian Fiebelkorn

Datenverarbeitungskonzept eines betrieblichen Informationssystems

Inhaltsverzeichnis

[1. Komponentenschnitt 3](#_Toc384036)

[2. Schnittstellenbeschreibung 4](#_Toc384037)

[2.1. KartenService 4](#_Toc384038)

[2.2. RegelService 5](#_Toc384039)

[2.3. SpielerService 6](#_Toc384040)

[2.4. SpielService 7](#_Toc384041)

[2.5. KiService 9](#_Toc384042)

[3. Konzeptionelles Datenmodell 10](#_Toc384043)

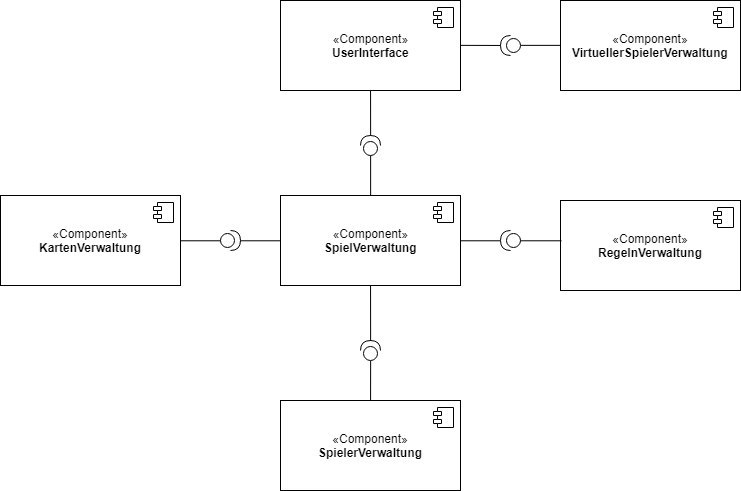
[4. Präsentationsschicht 10](#_Toc384044)

[5. Frameworks 15](#_Toc384045)

[6. Ablaufumgebung 15](#_Toc384046)

# 1. Komponentenschnitt

Das Informationssystem besteht aus Komponenten, die in der folgenden Abbildung ersichtlich sind.



# 2. Schnittstellenbeschreibung

Die Schnittstellen liegen jeweils im Ordner „export“ der jeweiligen Komponentenverwaltung.

## 2.1. KartenService

*/\*\*  
 \** ***@author*** *Joerg Lehmann, Christian Fiebelkorn, Dustin Lange  
 \** ***@version*** *20181113  
 \*/*package de.htwberlin.maumau.kartenverwaltung.export;  
  
import de.htwberlin.maumau.kartenverwaltung.entity.Farbe;  
import de.htwberlin.maumau.kartenverwaltung.entity.Karte;  
  
import java.util.List;  
  
public interface KartenService {  
  
  
 */\*\*Diese Methode legt einen Kartenstapel von 52 Karten an  
 \*  
 \** ***@return*** *Liste der Karten von 52  
 \*/* List<Karte> anlegenKartenstapel();  
  
 */\*\*  
 \* Diese Methode mischt den Ziehstapel einmalig durch.  
 \*  
 \** ***@param*** *obersteKarteBleibt - boolean, der angibt, ob die oberste Karte des Ablagestapels bleibt.  
 \** ***@return*** *Gibt eine Liste an Karten zurueck, diese Liste ist dann durchgemischt  
 \*/* List<Karte> mischenKartenstapel(List<Karte> karten, boolean obersteKarteBleibt);  
  
 */\*\*  
 \* Diese Methode legt neue Karten an  
 \*  
 \** ***@param*** *farbe - Die Farbe der neuen Karte (Enum Farbe beachten)  
 \** ***@param*** *wert - Den Wert der neuen Karte  
 \** ***@return*** *Die neue Karte  
 \*/* Karte erstellenNeuerKarte(Farbe farbe, String wert);  
  
}

## 2.2. RegelService

*/\*\*  
 \** ***@author*** *Joerg Lehmann, Christian Fiebelkorn, Dustin Lange  
 \** ***@version*** *20181113  
 \*/*package de.htwberlin.maumau.regelnverwaltung.export;  
  
import de.htwberlin.maumau.kartenverwaltung.entity.Farbe;  
import de.htwberlin.maumau.kartenverwaltung.entity.Karte;  
  
public interface RegelnService {  
  
 */\*\*  
 \* Diese Methode prueft ob die zulegende Karte gelegt werden darf  
 \* Sollte die letzte Karte ein Bube sein, muss die Wunschfarbe beachtet werden  
 \*  
 \** ***@param*** *letzteKarteKartenstapel - die letzte Karte, die auf dem Ablagestapel liegt  
 \** ***@param*** *legendeKarte - die Karte, die gelegt werden soll  
 \** ***@param*** *farbe - Farbe aus dem Spiel, die durch die Fahrbwahl eines Buben ausgeloest wurde  
 \** ***@return*** *gibt einen boolean zurueck, der angibt, ob die Karte gelegt werden darf  
 \*/* boolean darfKartegelegtwerden(Karte letzteKarteKartenstapel, Karte legendeKarte, Farbe farbe);  
  
 */\*\*  
 \* Diese Methode prueft ob die letzte Karte ein Bube ist und somit eine Farbwahl noetig ist  
 \*  
 \** ***@param*** *gelegteKarte - gerade gelegte Karte  
 \** ***@return*** *boolean, der angibt ob der Spieler sich eine Farbe wuenschen muss  
 \*/* boolean mussSichFarbeWuenschen(Karte gelegteKarte);  
  
 */\*\*  
 \* Methode, die prueft ob der naechste Spieler zwei Karten ziehen muss  
 \*  
 \** ***@param*** *gelegteKarte - gerade gelegte Karte  
 \** ***@param*** *zuziehendeKarte - Anzahl zu ziehende Karten, damit mehrere nacheinander Spieler den naechsten zwei Karten ziehen lassen  
 \* ziehen lassen koennen und eine Addition moeglich ist  
 \** ***@return*** *neue Anzahl der zuziehenden Karten durch den naechsten Spieler  
 \*/* int mussZweiKartenZiehen(Karte gelegteKarte, int zuziehendeKarte);  
  
 */\*\*  
 \* Diese Methode prueft, wenn der naechste Spieler dran ist, ob er ueberhaupt ablegen darf  
 \*  
 \** ***@param*** *gelegteKarte - die letzte Karte die gelegt wurde (also die oberste auf dem Ablagestapel)  
 \** ***@return*** *boolean, der angibt ob der aktuelle Spieler spielen darf  
 \*/* boolean mussRundeAussetzen(Karte gelegteKarte);  
  
 */\*\*  
 \* Diese Methode prueft, ob ein Richtungswechsel ausgeloest werden muss  
 \*  
 \** ***@param*** *gelegteKarte - gerade abgelegte Karte  
 \** ***@return*** *boolean, der angibt ob ein Richtungswechsel ausgeloest werden muss  
 \*/* boolean richtungWechsel(Karte gelegteKarte);  
}

## 2.3. SpielerService

*/\*\*  
 \** ***@author*** *Joerg Lehmann, Christian Fiebelkorn, Dustin Lange  
 \** ***@version*** *20181212  
 \*/*package de.htwberlin.maumau.spielerverwaltung.export;  
  
import de.htwberlin.maumau.kartenverwaltung.entity.Karte;  
import de.htwberlin.maumau.spielerverwaltung.entity.Spieler;  
  
public interface SpielerService {  
  
 */\*\*  
 \* Diese Methode fuegt einem Spieler eine Karte zu seinen Handkarten hinzu  
 \*  
 \*****@param*** *karte - Die Karte die Hinzugefuegt werden soll  
 \*****@param*** *spieler - Der Spieler, dem die Karte hinzugefuegt werden soll  
 \*/* Spieler karteZuHandblatthinzufuegen(Karte karte, Spieler spieler);  
  
 */\*\*  
 \* Diese Methode entfernt eine Karte aus den Handkarten eines Spielers  
 \*  
 \*****@param*** *karte - zu entfernende Karte  
 \*****@param*** *spieler - Der Spieler, aus dessen Handkarten die Karte entfernt werden soll  
 \*/* Spieler karteausHandblattentfernden(Karte karte, Spieler spieler);  
  
 */\*\*  
 \* Legt einen neuen spielerverwaltung an  
 \*  
 \*****@param*** *name - Name des neuen spielerverwaltung  
 \*****@return*** *der neue spielerverwaltung  
 \*/* Spieler neuerSpielerAnlegen(String name);  
}

## 2.4. SpielService

*/\*\*  
 \** ***@author*** *Joerg Lehmann, Christian Fiebelkorn, Dustin Lange  
 \** ***@version*** *20190302  
 \*/*package de.htwberlin.maumau.spielverwaltung.export;  
  
import de.htwberlin.maumau.kartenverwaltung.entity.Farbe;  
import de.htwberlin.maumau.kartenverwaltung.entity.Karte;  
import de.htwberlin.maumau.regelnverwaltung.export.RegelnService;  
import de.htwberlin.maumau.spielerverwaltung.entity.Spieler;  
import de.htwberlin.maumau.spielverwaltung.entity.Spiel;  
  
import java.util.List;  
  
public interface SpielService {  
  
 */\*\*  
 \* Diese Methode legt ein Spiel an  
 \*  
 \** ***@param*** *spielerliste Die Liste aller Spielern  
 \** ***@param*** *erweiterteRegeln Welche Regeln fuer das Spiel verwendet wird  
 \** ***@return*** *Gibt das neu angelegte Spiel zurück  
 \*/* Spiel anlegenSpiel(List<String> spielerliste, boolean erweiterteRegeln);  
  
 */\*\*  
 \* Diese Methode entnimmt eine Karte aus dem Ziehstapel  
 \*  
 \** ***@param*** *spiel - Das Spiel  
 \** ***@return*** *- Das veraenderte Spiel  
 \*/* Spiel ziehenKarteVomZiehstapel(Spiel spiel);  
  
 */\*\*  
 \* Prueft ob die Karte gelegt werden darf und wenn ja, fuegt diese hinzu  
 \*  
 \** ***@param*** *zulegendeKarte Die Karte, die gelegt werden will  
 \** ***@param*** *spieler Der Spieler der die Karte legen will  
 \** ***@param*** *spiel Das spielende Spiel  
 \** ***@return*** *Das Spiel  
 \*/* Spiel legeKarte(Karte zulegendeKarte, Spieler spieler, Spiel spiel);  
  
 */\*\*  
 \* Setzt die Farbe im Spiel um  
 \*  
 \** ***@param*** *spiel Das Spiel  
 \** ***@param*** *farbe Die Farbe, die gesetzt wurde  
 \** ***@return*** *Das Spiel  
 \*/* Spiel farbeGewaehlt(Spiel spiel, Farbe farbe);  
  
 */\*\*  
 \* Setzt den Nächsten Spieler als Aktiv  
 \*  
 \** ***@param*** *spiel Das Spiel  
 \** ***@return*** *Das Spiel  
 \*/* Spiel naechsterSpieler(Spiel spiel);  
  
 */\*\*  
 \* Diese Methode zieht Karten vom Ablagestapel und sorgt dafuer, dass der Spieler sie als Handkarten bekommt  
 \*  
 \** ***@param*** *anzahl - Anzahl der Karten, die gezogen werden sollen  
 \** ***@param*** *karteStapel - Liste an Karten, von denen gezogen werden soll (Ziehstapel)  
 \** ***@param*** *spieler - Spieler, der die Karten ziehen soll  
 \** ***@return*** *Liste der restlichen Karten, stellt Ziehstapel des Spieles dar  
 \*/* List<Karte> karteZiehen(int anzahl, List<Karte> karteStapel, Spieler spieler);  
  
 */\*\*  
 \* Prueft am Ende eines Zuges, ob das Spiel zu Ende ist  
 \*  
 \** ***@param*** *spieler - aktueller Spieler  
 \** ***@return*** *boolean, der angibt, ob das Spiel zu Ende ist  
 \*/* boolean ermittleSpielende(Spieler spieler);  
  
 */\*\*  
 \* Prueft ob der Spieler Mau geklickt hat  
 \* und falls nicht, bekommt der Spieler auch die Strafkarten auf die Hand  
 \*  
 \** ***@param*** *spiel - Das Spiel  
 \** ***@return*** *Das Spiel, nachdem ggfs. Strafkarten dem Spieler zugefuehrt wurden  
 \*/* Spiel pruefeAufMau(Spiel spiel);  
  
 */\*\*  
 \* Setzt Mau bei Spieler  
 \*  
 \** ***@param*** *spieler - bei dem Mau gesetzt werden muss  
 \** ***@param*** *neuerZustand - Der Zustand den der Mauzustand danach haben soll,  
 \*/* void setzeMau(Spieler spieler, boolean neuerZustand);  
  
 */\*\*  
 \* Prüft, ob Karten gemischt werden muessen, und tut es ggf.  
 \*  
 \** ***@param*** *spiel Das Spiel  
 \** ***@return*** *Das Spiel  
 \*/* Spiel mussGemischtWerden(Spiel spiel);  
  
 */\*\*  
 \* Diese Methode fuegt die gewuenschte Regelfassung dem Spiel hinzu  
 \*  
 \** ***@param*** *erweiterteRegeln - gibt an ob mit erweiterten Regeln gespielt werden soll  
 \** ***@return*** *- die Class des gewuenschten Regelsatzes  
 \*/* RegelnService regelwerkHinzufuegen(boolean erweiterteRegeln);  
  
}

## 2.5. KiService

*/\*\*  
 \** ***@author*** *Joerg Lehmann, Christian Fiebelkorn, Dustin Lange  
 \** ***@version*** *20190201  
 \*  
 \*/*package de.htwberlin.maumau.virtuellerspielerverwaltung.export;  
  
import de.htwberlin.maumau.kartenverwaltung.entity.Farbe;  
import de.htwberlin.maumau.spielerverwaltung.entity.Spieler;  
  
public interface KiService {  
  
 */\*\*  
 \* Diese Methode prueft ob die KI mau sagen muss  
 \* und entscheidet ob die KI Mau sagt  
 \*  
 \** ***@param*** *spieler - Die Ki die geprueft werden soll  
 \** ***@return*** *- boolean, der angibt ob die Ki Mau gesagt hat  
 \*/* boolean mauSetzen(Spieler spieler);  
  
 */\*\*  
 \* Diese Methode generiert den Namen fuer einen KI Spieler  
 \*  
 \** ***@param*** *kiZaehler - Nummer des wie vielten gewuenschten KI Spielers  
 \** ***@return*** *- String der den Namen der Ki darstellt  
 \*/* String kiAnlegen(int kiZaehler);  
  
 */\*\*  
 \* Methode entscheidet, welche Farbe sich die KI wuenscht  
 \*  
 \** ***@return*** *- gewuenschte Farbe  
 \*/* Farbe kiMussFarbeWuenschen();  
}

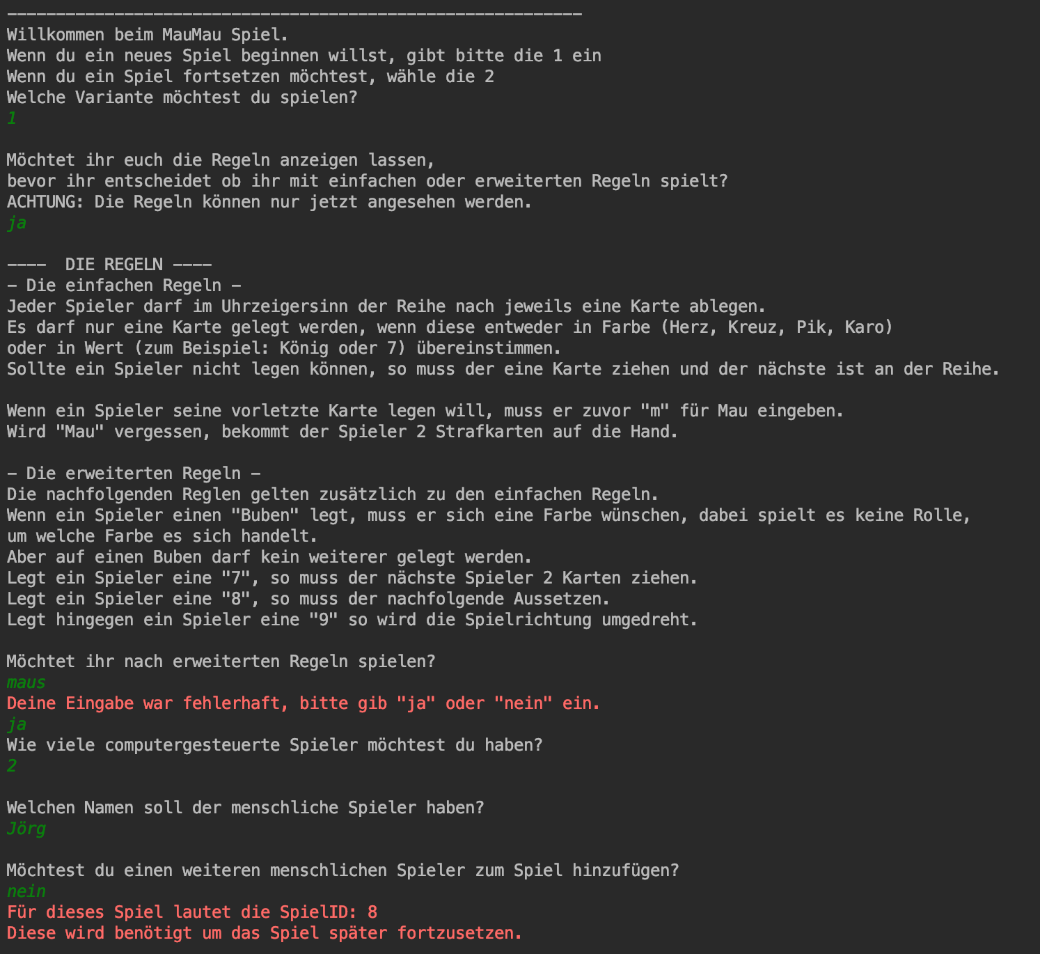
# 3. Konzeptionelles Datenmodell

Es wurde konzeptionell mit verschiedenen Entitäten und dem daraus ergebenen relationalen Modell gearbeitet, da aber ein objektorientiertes Datenmodell (ObjectDB) zur Speicherung genutzt wird, ist dies überflüssig. Es werden alle Attribute in einem Objekt zusammen abgelegt. Es wird nicht denormalisiert. Der Controller ist alleinig für die Verwaltung des Datenmodells zuständig.

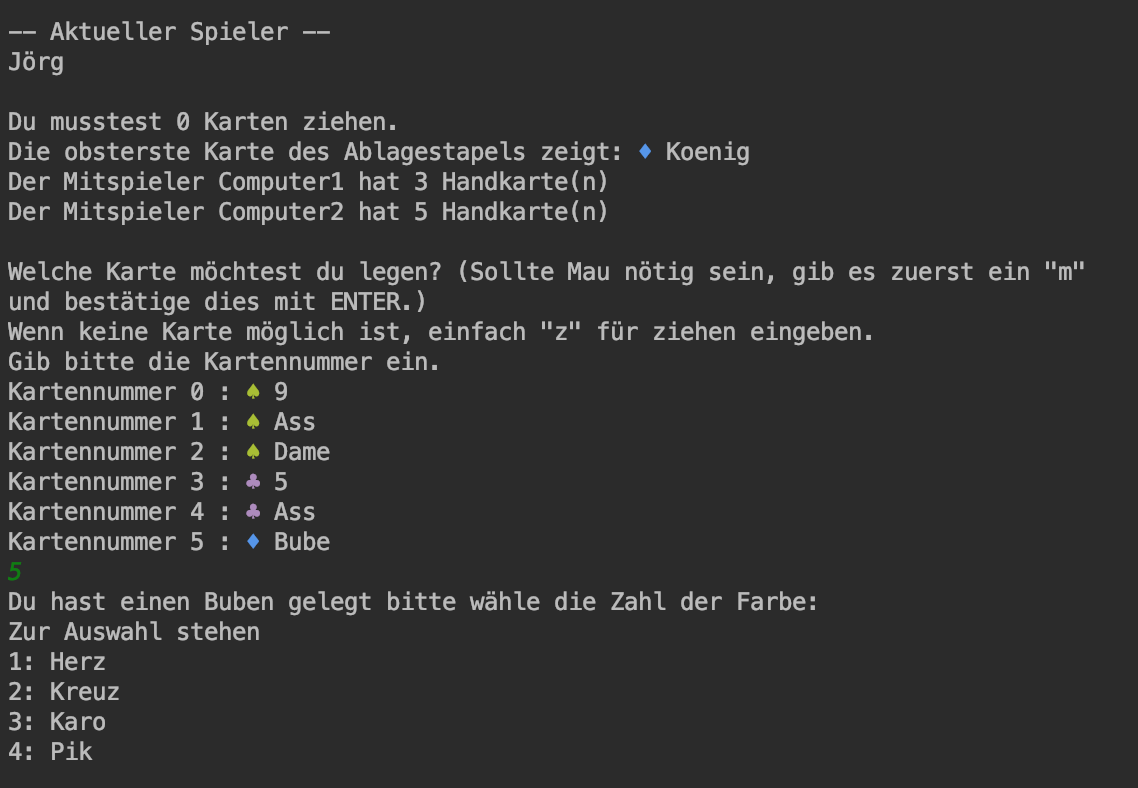
# 4. Präsentationsschicht

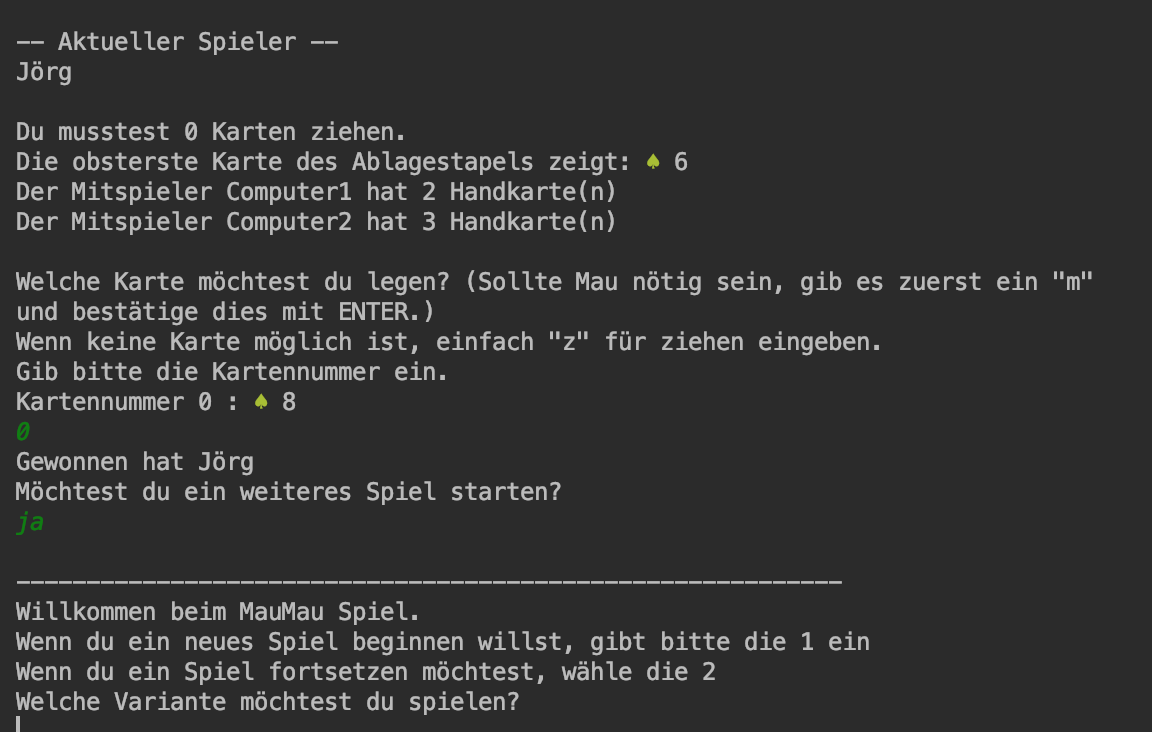
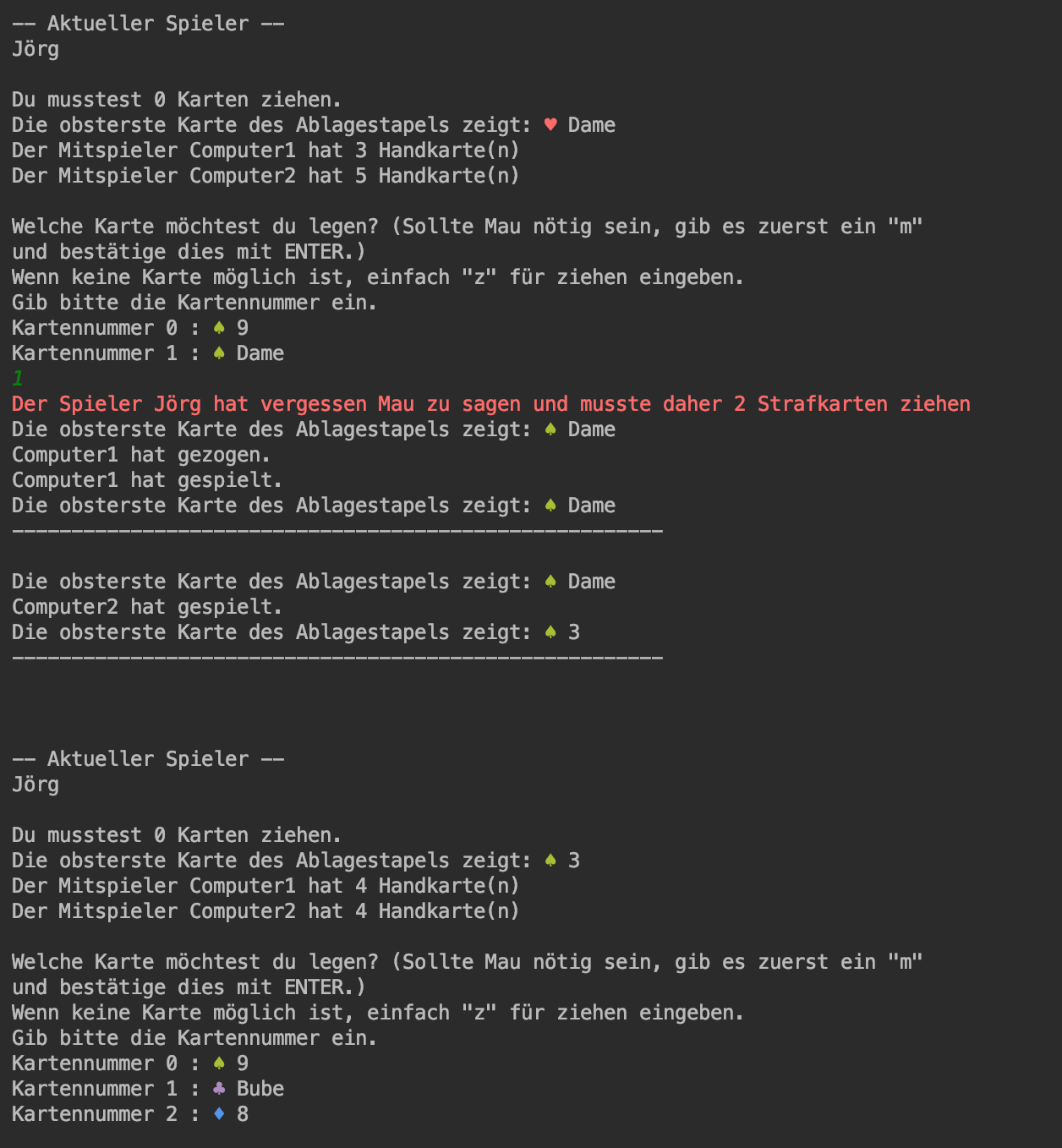
Die Anwendung enthält aktuell noch kein GUI, stattdessen wird ein CLI (Command Line Interface) über die Java-Console bereitgestellt. Eingaben des Nutzers werden durch die Scannerklasse (java.util.Scanner) zur Laufzeit aufgenommen.

Folgende Abbildungen veranschaulichen die wesentlichen Dialoge und Meldungen, welche in der Anwendung auftreten können. Der Spielablauf wird durch diese Abbildungen ebenfalls sequenziell veranschaulicht.









Folgende Services aus den Schnittstellen werden aufgerufen:

SpielControllerImpl.java 🡪 SpielService, KiService, SpielViewer  
SpielServiceImpl.java 🡪 KartenService, RegelnService, SpielerService

# 5. Frameworks

1. @RunWith – Annotation oberhalb der Klassendefinition gibt an, welcher Test-Runner genutzt wird. 🡪 @RunWith(MockitoJUnitRunner.class)
2. @InjectMocks – Annotation markiert das Attribut, welches die zu testende Klasse innehält.
3. Mockito 🡪 Annotationen der Attribute mit @Mock
4. Setzen der für den eigentlichen Test benötigten Returnwerte von Methoden
5. (Mockito.when(<**ZU\_MOCKENDE\_METHODE**><**BENÖTIGTER\_RÜCKGABEWERT**>)Gewöhnlicher JUnit-Test





# 6. Ablaufumgebung

Aktuell ist zum Ausführen der Anwendung eine IDE erforderlich, da bisher keine ausführbare Datei erzeugt wurde. Dies wäre aber mit zwei Schritten möglich.

Die Anwendung kann lokal auf jedem System ausgeführt werden, sofern Java Runtime Machine installiert und ausführbar ist. Es ist von keinen weiteren Systemen, wie z.B. DB-Server abhängig, da für die notwendige Persistenz eine integrierte Object-DB Einsatz findet.