Datenverarbeitungskonzept eines betrieblichen Informationssystems

Mau Mau Kartenspiel Projekt

im Studiengang

Wirtschaftsinformatik

im Fach

Komponentenbasierte Entwicklung

am Fachbereich

Informatik, Kommunikation und Wirtschaft (FB4)

an der Hochschule für Technik und Wirtschaft Berlin

vorgelegt von:

Petrova Olga

Matrikelnr.

Tel.

E-Mail

Berlin, 30.09.2020

Inhaltsverzeichnis

Vorwort

Dieses Dokument beschäftigt sich mit dem Datenverarbeitungskonzept, das zur Fortführung des Fachkonzeptes bei der Programmierung dienen muss. Es beschreibt die relevanten Daten und deren Verarbeitung.

Ausgehend von den Anforderungen wird die Realisierung des DV-Konzepts für Mau Mau Kartenspiel (Java Programm für dieses Kartenspiel) vorgestellt. Zunächst geht es um die Beschreibung von dem Komponentenschnitt und Schnittstellen. Vorstellung von dem konzeptionellen Datenmodell sollte danach einen kurzen Einblick in Datenbankstrukturen geben. Explizit wird auf die für den Benutzer transparente Dialogstruktur eingegangen und die Präsentationsschicht erläutert. Der nächste Teil dieses Dokuments beschreibt die im Projekt verwendeten Frameworks und ihre Konfiguration mit Anwendungsbeispielen. Abschließend wird die technische Ablaufumgebung kurz erläutert.

1. Komponentenschnitt

1.1. Einleitung

Komponenten-Architektur für das Kartenspiel Mau-Mau besteht aus sechs separaten Komponenten. Alle Daten und Funktionen innerhalb jeder Komponente sind semantisch verwandt.

Die Komponente *View_Management* ist für Benutzeroberflache verantwortlich und bietet dem Benutzer, der später im Spiel als echter Spieler auftritt, eine Schnittstelle, die er benutzen kann. Diese Komponente importiert Schnittstellen wie GameService, RulesService, CardService und verwendet auch Datentypen, die in den Komponenten *Real_Player_Management* und *Virtual_Player_Management* definiert sind. Es gibt keine Datenübergabe oder Schnittstellenmethodenaufruf, es besteht lediglich eine Abhängigkeit von den Komponentendaten. Daher wurde die "use"-Notation verwendet.

Die Komponente *Game_Management* exportiert GameService Interface und auch importiert Schnittstellen RealPlayerService, VirtualPlayerService, CardService, RulesService.

Die Komponente *Real_Player_Management* stellt die Schnittstelle RealPlayerService zur Verfügung, und die *Virtual_Player_Management* -Komponente stellt dementsprechend die Schnittstelle VirtualPlayerService bereit. Diese beiden Komponenten werden zu einer *Player_Management*-Komponente zusammengefasst, die eine Abhängigkeit von der *Card_Management*-Komponente hat, da der Spieler Karten auf der Hand hält.

Die Komponente *Rules_Management* exportiert die Schnittstelle RulesService und benutzt Datentypen aus Komponenten *Card_Management*, *Real_Player_Management*, *Virtual_Player_Management,* und ist somit von diesen Komponenten abhängig. Die *Rules_Management* Komponente für die Spielregeln ist austauschbar und durch Implementierungen mit verschieden Sonderregelvarianten bereitgestellt werden kann.

Die Komponente *Card_Management* stellt die Schnittstelle CardService bereit, aber selbst benutzt keine andere Schnittstellen.

Die Komponente *Configuration_management* ist für Bereitstellung von Spring ApplicationContext für die gesamte Anwendung und andere Konfigurationseinstellungen zuständig. Um diese Aufgabe zu erledigen, wird eine Schnittstelle ConfigurationService bereitgestellt. Da die Schnittstelle ConfigurationService für das gesamte Anwendung erforderlich ist, hat das *Configuration_management* Abhängigkeiten zu allen registrierten Komponenten. Dafür das das Diagramm übersichtlich wird, wurde die Komponente im Diagramm weggelassen.

1.2. Komponentendiagramm

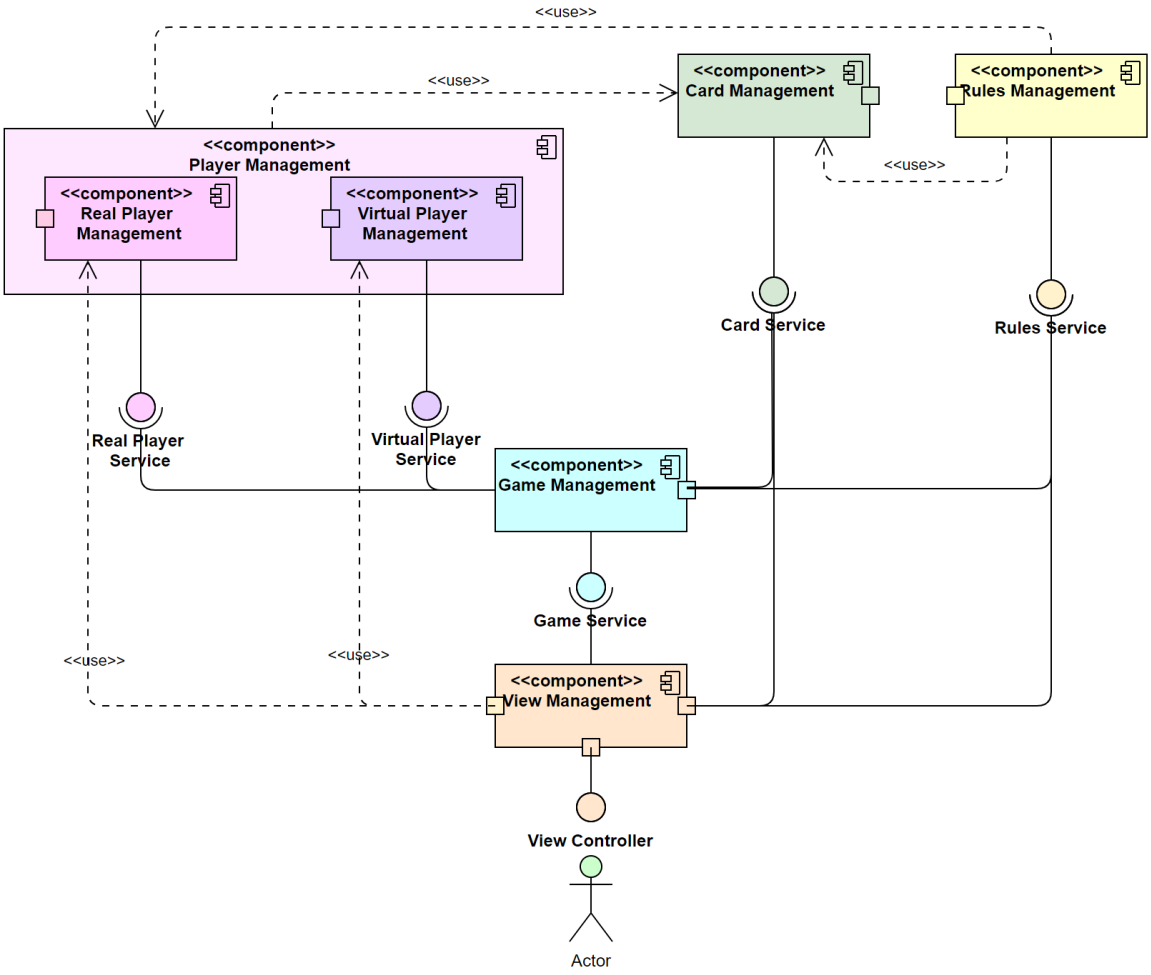

Abbildung 1: Komponentendiagramm

2. Schnittstellenbeschreibung

Dieser Abschnitt enthält eine Beschreibung der Schnittstellen, darunter den Namen und Zweck der nach außen verfügbaren Methoden, Bedeutung und Wertebereiche von Übergabe- und Rückgabeparametern. Diese Daten können der Dokumentation entnommen werden, die mit dem JavaDoc-Tool generiert wurde.

2.1. ViewController

htwberlin.mau_mau.view_management.controller

Interface ViewController

All Known Implementing Classes:

ViewControllerImpl

`public interface ViewController`

The interface ViewController provides actions for the user interface (view) to interact with other components of the game.

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
`void`	`run()` Action to start the new game.

Method Detail

run

`void run()`

Action to start the new game.

2.2. GameService

htwberlin.mau_mau.game_management.service

Interface GameService

All Known Implementing Classes:

GameServiceImpl

`public interface GameService`

The interface Game Service provides operations to control the main game flow.

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
`int`	`countPenalty-Cards(PostAction postAction)` Counts the number of penalty cards.

void	**dealCardsToPlayers**(**GameData** gameData) Deals cards to players, i.e.
PostAction	**getPostAction**(**RulesResult** rulesResult) Gets post action from RulesService if the player can not make a move.
RulesResult	**makeGameMoveForRealPlayer**(int cardPosition, **GameData** gameData) Makes real player move and store it in the game data.
RulesResult	**makeGameMoveForVirtualPlayer**(**Card** openCard, **Deck** hand, **GameData** gameData) Makes virtual player move and store it in the game data, i.e.
void	**saveToDB**(**GameData** gameData) Saves gameData object to persistent storage.
GameData	**setupNewGame**(java.lang.String name, int numberOfVirtualPlayers, **GameRulesId** gameRulesId) Sets up new game.

Method Detail

setupNewGame

```
GameData setupNewGame(java.lang.String name,
                      int numberOfVirtualPlayers,
                      GameRulesId gameRulesId)
```

Sets up new game.

Parameters:

name - the name of the real player that this player entered in ui
numberOfVirtualPlayers - the virtual players
gameRulesId - the game rules id

Returns:

gameData new game data object representing current game state

dealCardsToPlayers

```
void dealCardsToPlayers(GameData gameData)
```

Deals cards to players, i.e. save cards in Hand for each Player and store it in gameData.

Parameters:

gameData - the game data object representing current game state

makeGameMoveForRealPlayer

```
RulesResult makeGameMoveForRealPlayer(int cardPosition,
                                      GameData gameData)
```

Makes real player move and store it in the game data.

Parameters:

cardPosition - the card position in the player's hand, selected by the player in ui
gameData - the game data object representing current game state

Returns:

RulesResult object that contains rules validation result

makeGameMoveForVirtualPlayer

```
RulesResult makeGameMoveForVirtualPlayer(Card openCard,
                                         Deck hand, GameData gameData)
```

Makes virtual player move and store it in the game data, i.e. select a card from the virtual player's hand which can be played against the open card at the top of the playing stack.

Parameters:
openCard - the open card at the top of the playing stack
hand - Deck object representing cards in the virtual player's hand
gameData - the game data object representing current game state
Returns:
RulesResult object that contains rules validation result

getPostAction

PostAction getPostAction(RulesResult rulesResult)
Gets post action from RulesService if the player can not make a move.
Parameters:
rulesResult - object containing rules validation state and result
Returns:
the post action object indicating an action to be done in case the move can't be made

countPenaltyCards

int countPenaltyCards(PostAction postAction)
Counts the number of penalty cards.
Parameters:
postAction - the post action object indicating an action to be done in case the move can't be made
Returns:
the int number of penalty cards that should be dealt to the player

saveToDB

void saveToDB(GameData gameData)
Saves gameData object to persistent storage.
Parameters:
gameData - the game data object representing current game state

2.3. RealPlayerService

htwberlin.mau_mau.real_player_management.service

Interface RealPlayerService

All Known Implementing Classes:
RealPlayerServiceImpl

---

public interface **RealPlayerService**
The interface RealPlayerService provides operations relating to the real player's attributes.

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
RealPlayer	createRealPlayer(java.lang.String name) Creates real player data object using given name.
boolean	isSaidMau(RealPlayer player) Gets value of saidMau boolean.
boolean	isSaidMauMau(RealPlayer player) Gets value of saidMauMau boolean.
void	setSaidMau(boolean saidMau, RealPlayer player) Sets value of saidMau boolean.

void	**setSaidMauMau**(boolean saidMauMau, **RealPlayer** player) Sets value of saidMauMau boolean

Method Detail

isSaidMau

boolean isSaidMau(RealPlayer player)
Gets value of saidMau boolean.

Parameters:
player - the real player object

Returns:
the boolean: true, if the real player has said 'Mau!', when it is his turn and he only has two cards

setSaidMau

void setSaidMau(boolean saidMau, RealPlayer player)
Sets value of saidMau boolean.

Parameters:
saidMau - the boolean: true, if the real player has said 'Mau!'
player - the real player object

isSaidMauMau

boolean isSaidMauMau(RealPlayer player)
Gets value of saidMauMau boolean.

Parameters:
player - the real player object

Returns:
the boolean: true, if the real player has said 'Mau-Mau!', if he has only the last card

setSaidMauMau

void setSaidMauMau(boolean saidMauMau, RealPlayer player)
Sets value of saidMauMau boolean

Parameters:
saidMauMau - the boolean: true, if the real player has said 'Mau Mau!'
player - the real player object

createRealPlayer

RealPlayer createRealPlayer(java.lang.String name)
Creates real player data object using given name.

Parameters:
name - the name

Returns:
the real player object

2.4. VirtualPlayerService

htwberlin.mau_mau.virtual_player_management.service

Interface VirtualPlayerService

All Known Implementing Classes:
VirtualPlayerServiceImpl

---

public interface **VirtualPlayerService**
The interface VirtualPlayerService provides operations related to the virtual player.

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
VirtualPlayer	createVirtualPlayer() Creates virtual player with a random name from the predefined list.

Method Detail

createVirtualPlayer

VirtualPlayer createVirtualPlayer()
Creates virtual player with a random name from the predefined list.
Returns:
the virtual player object

2.5. ConfigurationService

htwberlin.mau_mau.configuration_management.service
Interface ConfigurationService
All Known Implementing Classes:
ConfigurationServiceImpl

---

public interface **ConfigurationService**
The interface Configuration Service provides an instance of Spring dependency injection container.

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
org.springframework.context. ConfigurableApplicationContext	context() Context configurable application context (DI container)

Method Detail

context

org.springframework.context.ConfigurableApplicationContext context()
Configurable application context (DI container)
Returns:
the configurable application context

2.6. CardService

htwberlin.mau_mau.card_management.service
Interface CardService
All Known Implementing Classes:
CardServiceImpl

---

public interface **CardService**
The interface CardService provides operations over the cards and decks.

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
Deck	createDrawingStack() Creates initial drawing deck of cards.
Deck	createPlayingStack(Deck drawingStack) Create initial playing stack deck.
Card	drawCard(Deck drawingStack, Deck playingStack, Deck hand) Picks the new card from the top of drawing stack and add it to the player's hand.
Card	getCardByPositionFrom-Hand(int position, Deck hand) Gets card by position from player's hand.
Card	getOpenCard(Deck playingStack) Gets the open card from top of the the playing stack.
void	playCard(Deck hand, int cardPosition, Deck playingStack) Removes one card from the player's hand and add the played card to the top of the playing stack.
Deck	shuffleDrawingDeck(Deck drawingStack) Shuffles a drawing deck of cards by placing them in random order.

Method Detail

createDrawingStack

```
Deck createDrawingStack()
```

Creates initial drawing deck of cards.

Returns:

Deck the deck

createPlayingStack

```
Deck createPlayingStack(Deck drawingStack)
                 throws EmptyDrawingStackException,
                        EmptyPlayingStackException
```

Create initial playing stack deck.

Parameters:

drawingStack - Deck object representing the drawing stack

Returns:

Deck object representing the playing stack

Throws:

EmptyDrawingStackException - the empty drawing stack exception
EmptyPlayingStackException - the empty playing stack exception

playCard

```
void playCard(Deck hand,
              int cardPosition,
              Deck playingStack)
throws IncorrectCardPositionException
```

Removes one card from the player's hand and add the played card to the top of the playing stack.

Parameters:

hand - Deck object representing cards in the player's hand
cardPosition - card position in hand
playingStack - Deck object representing the playing stack

Throws:
IncorrectCardPositionException - the incorrect card position exception

drawCard

```
Card drawCard(Deck drawingStack,
              Deck playingStack,
              Deck hand)
       throws EmptyDrawingStackException,
         EmptyPlayingStackException
```

Picks the new card from the top of drawing stack and add it to the player's hand. When the drawing stack is empty, turns over and shuffles the playing stack to serve as new drawing stack.

Parameters:
drawingStack - Deck object representing the drawing stack
playingStack - Deck object representing the playing stack
hand - Deck object representing cards in the player's hand

Returns:
Card the card

Throws:
EmptyDrawingStackException - the empty drawing stack exception
EmptyPlayingStackException - the empty playing stack exception

getCardByPositionFromHand

```
Card getCardByPositionFromHand(int position,
                               Deck hand)
```

Gets card by position from player's hand.

Parameters:
position - the position of card in player's hand, returned by View
hand - Deck object representing cards in the player's hand

Returns:
Card the card

getOpenCard

```
Card getOpenCard(Deck playingStack)
```

Gets the open card from top of the the playing stack.

Parameters:
playingStack - Deck object representing the playing stack

Returns:
Card the open card on the top of the playing stack.

shuffleDrawingDeck

```
Deck shuffleDrawingDeck(Deck drawingStack)
```

Shuffles a drawing deck of cards by placing them in random order.

Parameters:
drawingStack - Deck object representing the drawing stack

Returns:
Deck object representing the shuffled drawing stack

2.7. RulesService

htwberlin.mau_mau.rules_management.service

Interface RulesService

All Known Implementing Classes:
RulesServiceSpecial, RulesServiceStandard

```
public interface RulesService
```

The interface RulesService provides operations to check game conditions according to the set of rules.

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
Player	defineNextPlayer(RulesResult rulesResult, java.util.List<Player> players)
PostAction	definePostAction(RulesResult rulesResult) Defines the post action if the player can not make a move.
RulesResult	setUpRules(Card openCard) Sets up rulesResult object at the start of the game.
RulesResult	validatePlayerMove(Card card, Card openCard, RulesResult rulesResult, int numberOfPlayers) Validates player move to check if the card can be played.

Method Detail

validatePlayerMove

RulesResult validatePlayerMove(Card card,
Card openCard,
RulesResult rulesResult,
int numberOfPlayers)

Validates player move to check if the card can be played.

Parameters:

card - the card chosen by player to play with
openCard - the open card on the top of the playing deck
rulesResult - object containing rules validation result and text message
numberOfPlayers - number of players in game

Returns:

rulesResult object containing validation result and text message

definePostAction

PostAction definePostAction(RulesResult rulesResult)

Defines the post action if the player can not make a move.

Parameters:

rulesResult - object containing rules validation result and text message

Returns:

the post action object indicating an action to be done in case the move can't be made

setUpRules

RulesResult setUpRules(Card openCard)

Sets up rulesResult object at the start of the game. Checks, if the open card is special, then sets up related rules.

Parameters:

openCard - the open card on the top of the playing deck

Returns:

rulesResult object containing validation result and text message

defineNextPlayer

Player defineNextPlayer(RulesResult rulesResult,
java.util.List<Player> players)

3. Konzeptionelles Datenmodell

3.1. Auswahl des Datenmodells

Ein relationales Modell wurde als ein Datenmodell in diesem Projekt ausgewählt. Diese Wahl wird durch das spezifische Datenbanksystem bestimmt. Laut Hansen ist es notwendig, das Endkonzeptionelle Datenmodell in ein konkretes Datenmodell umzuwandeln, das von dem entsprechenden Datenbanksystem unterstützt wird.

Als Datenbanksystem verwendet das Projekt die H2 Database Engine, kurz H2, die ein in der Programmiersprache Java geschriebenes relationales Datenbankmanagementsystem ist. Anstelle von H2 in dem Projekt könnte jedoch eine andere relationale Datenbank verwendet werden.

Da beschlossen wurde, die Hibernate-Technologie zu verwenden, wurde aus diesem Grund das relationale Datenmodell angewandt. Hibernates Hauptaufgabe ist die objektrelationale Abbildung (englisch OR-Mapping). Dies ermöglicht es, gewöhnliche Objekte mit Attributen und Methoden in relationalen Datenbanken zu speichern. Beziehungen zwischen Objekten werden auf entsprechende Datenbank-Relationen abgebildet.

3.2. Abbildung des konzeptionellen Datenmodells

Um das konzeptuelle Modell auf physische Implementierungskonstrukte abzubilden, beschreibt das interne Schema, wie die logischen Objekte des konzeptuellen Schemas physisch gespeichert werden sollen. Zu diesen Objekten im Mau-Mau-Projekt gehören Tabellen, die auf der Basis der Java-Klassen mit Daten generiert wurden.

Das Abbilden von Java-Klassen auf Datenbanktabellen in dem Projekt wird mit **Java-Annotation** (@Entity, @Enumeration, @Id usw.) bewerkstelligt. Andere Alternative wäre das mittels einer XML-Datei (Mapping File) zu realisieren. Somit ist das Schema selbst in einer formalen Sprache definiert, so dass sich Daten automatisch darauf überprüfen lassen, ob sie dem Schema entsprechen.

3.3. Denormalisierung

Bei diesem Projekt wird die Denormalisierung, die als bewusste Rücknahme der Normalisierung verstanden wird, um das Laufzeitverhalten einer Datenbankanwendung zu verbessern, angewendet. Der Grund dafür ist, dass die Vererbung im Projekt implementiert ist, die es ermöglicht, Hierarchien von Klassen zu definieren, die verschiedene Implementierungen bieten.

Von den drei verschiedenen Strategien zur Abbildung der Objekte auf Tabellen verwendet der Projekt eine **SINGLE_TABLE** -Vererbungs-Mapping. Diese JPA-Strategie wird als Standard verwendet, dabei eine ganze Vererbungshierarchie des Domänenmodells in eine

einzige Datenbanktabelle gebündelt wird. Das heißt, die Angabe in der obersten Basisklasse gilt für alle Subklassen.

Obwohl es sich um einen *Verstoß gegen die dritte Normalform* handelt, ist diese Strategie performant, da Polymorphismus einschließlich polymorpher Assoziationen einfach zu realisieren ist. Da nur eine Tabelle zum Speichern von Entitäten verwendet wird, sind sowohl Lese- als auch Schreibvorgänge schnell. Selbst wenn eine @OneToMany- oder eine @OneToOne-Basisklassenzuordnung verwendet wird, benötigt Hibernate eine einzige Verknüpfung zwischen Eltern- und Kindtabellen.

Wichtig zu beachten ist, dass die Spalten von Kindklassen müssen nullable sein. Die Überprüfung der Datenintegritätseinschränkungen (data integrity constraints) kann in der Datenzugriffsschicht erfolgen. Die Bean-Validierung kann @NotNull-Attribute zur Laufzeit validieren. JPA definiert auch Callback-Methoden (z.B. @PrePersist, @PreUpdate) sowie Entity-Listener (z.B. @EntityListeners), die eine Ausnahme auslösen können, wenn eine Nicht-Null-Beschränkung verletzt wird. Diese Ansätze wurden jedoch aufgrund von Zeitmangel nicht in das Projekt implementiert.

3.4. Datenhohheit

Die Komponente *Game_Management* ist für die Klassen des Datenmodells wie GameData, GameStatus und GameRulesId zuständig. Klassen mit Daten wie RulesResult, RulesResultSpecial und RulesResultStandard liegen im Verantwortungsbereich der *Rules_Management*-Komponente. Die Komponente *Card_Management* umfasst die Daten von Deck, Card, Rank und Suit Klassen.

Die Komponente *Real_Player_Management* ist für die Klasse RealPlayer bzw. die Komponente *Virtual_Player_Management* für die Klasse VirtualPlayer zuständig, dabei die beide Klassen von der abstrakten Klasse Player geerbt werden und in die Komponente *Player_Management* zusammengefasst..

3.5. Klassendiagramm

Das konzeptionelle Modell wird unter Verwendung solcher Notation wie UML für die Objektmodellierung beschrieben. In der UML-Notation wird das konzeptionelle Modell mit einem Klassendiagramm erläutert, in dem Klassen Konzepte darstellen, Assoziationen Beziehungen zwischen Konzepten darstellen usw.

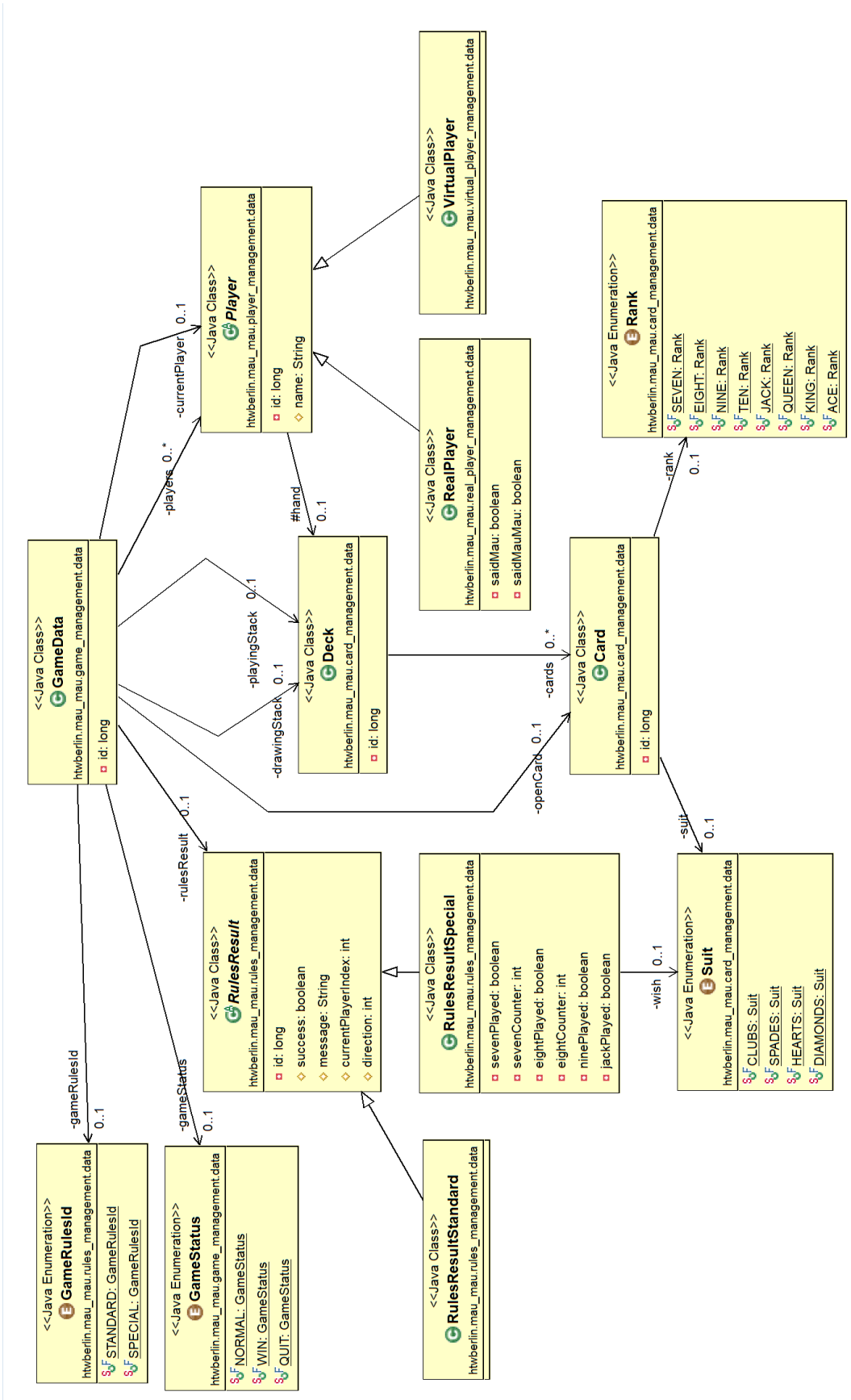

Abbildung 2: Klassendiagramm

4. Präsentationsschicht

4.1. Architektur der GUI

Die Interaktion mit dem Benutzer wird über eine Standard-Java-Textkonsole implementiert. Für die Implementierung der Benutzeroberfläche wird das Architekturmuster Model View Controller (MVC, englisch für Modell-Präsentation-Steuerung) verwendet.

Modell wird durch GameData.class dargestellt. Das Objekt von dieser Klasse wird von der GameSerivce-Komponente empfangen und enthält alle Spieldaten.

View ist View.class, die die direkte Interaktion mit dem Benutzer durch führt, zeigt Text an und empfängt Benutzereingaben.

Controller ist ViewControllerImpl.class. In dieser Klasse wird die Logik der Benutzerschnittstelle und die Verbindung mit der Spiellogik behandelt.

4.2. Dialoge im Spiel mit Standardregeln

4.2.1. Meldungen bezüglich des menschlichen Spielers

Sowohl während des Entwicklungsprozesses als auch für die Kommunikation mit dem Benutzer wurde Englisch verwendet. Der folgende Dialog findet statt, wenn ein Spieler ein neues Spiel nach den Standardregeln beginnt:

```
Welcome to the Mau-Mau Palace! Here we offer you your favourite Mau-Mau
game against virtual players with standard and special rules.
Please enter your name:
>Max
Hello, Max!
Enter number of virtual players (1-3):
>2
Select game rules: 1 - standard rules, 2 - special rules:
>1
OK, Max, let's play Mau-Mau game with 2 other players, using standard
rules!
Mau-Mau is a card came for 2-4 players. The first player with no cards left
wins.  The card you play must match the open card on the top of the playing
stack. Either suit or rank have to match! You can choose a card to play
from your hand, entering a number of card. If you have no suitable card to
play, you must draw a card - enter 'D'. The turn of a player ends directly
after drawing a card. Notice, that you must announce your last card saying
'Mau!' or you get 2 penalty cards. Similarly, you must declare your win
with 'Mau-Mau!' before you play your last card or game continues with 2
penalty cards for you and this win is not accepted. You enter 'M' to say
'Mau!' or 'MM' to say 'Mau-Mau!' To quit the game prematurely, enter 'Q'.
```

Dieser Text wird angezeigt, wenn der Spieler einen nicht gültigen Namen eingibt:

```
Incorrect input. Empty name entered.
Please enter your name:
```

Wenn der Spieler eine ungültige Zahl eingibt, wird die Meldung ausgegeben:

```
Incorrect input. Expected value between 1 and 3, but received r.
```

```
Enter number of virtual players (1-3):
```

Das folgende Beispiel enthält den Dialog, der dem echten Spieler vor jedem Zug angezeigt wird:

```
Players' cards:
Charles Bukowski has 5 cards.
O. Henry has 5 cards.
YOUR TURN NOW:
Cards in your hand:
1 - JACK of HEARTS
2 - EIGHT of DIAMONDS
3 - EIGHT of SPADES
4 - TEN of DIAMONDS
5 - ACE of DIAMONDS
Open card: ACE of HEARTS
Enter (1-5) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
```

Wenn der echte Spieler einen falschen Zug eingibt, wird der Text ausgegeben::

```
Incorrect input. Expected value between 1 and 5, 'D', 'M', 'MM', or 'Q',
but received 0.
Enter (1-5) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
```

Dieser Text wird ausgegeben, wenn ein echter Spieler eine falsche Karte spielt:

```
You can't play this card now. Choose another card from your hand or draw a
new card from the drawing stack, please.
```

Dieser Text wird angezeigt, wenn der echte Spieler eine Karte zieht:

```
Enter (1-5) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
>D
You drew a card. It's NINE of DIAMONDS
```

Wenn der echte Spieler zur falschen Zeit 'Mau' sagt, erhält er diese Meldung:

```
Enter (1-6) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
>M
You said 'Mau!', but it's not the time yet - you have more than two cards.
```

Folgender Text ist zu sehen, wenn der Spieler zur falschen Zeit 'Mau-Mau' sagt:

```
Enter (1-6) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
>MM
You said 'Mau-Mau!', but it's not the time yet - you have more than one
card.
```

Dieser Text wird angezeigt, wenn der echte Spieler eine richtige Karte ausspielt:

```
Enter (1-6) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
>3
You played QUEEN of DIAMONDS against SEVEN of DIAMONDS.
Nice move.
```

Damit das Kartenspiel den Teilnehmer noch ein bisschen unterhält, wird nach jedem erfolgreichen Spielzug unmittelbar nach der Ankündigung der Karte eine zusätzliche Lobbotschaft auf dem Bildschirm angezeigt. Dieses Wort wird zufällig aus der Liste ausgewählt:

```
private static final String[] messages = {"Success! ", "Cool! ", "Great! ",
"WOW! ", "Prima! ", "Nice move. ", "OK. ",  "Phew! ", "Awesome! ", "Fine.
", "Alright! ", "Not bad. ", "Good. ", "Okay. ", "Super! ", "Well done. ",
"Quite good. ", "Acceptable. "};
```

Folgender Dialog wird angezeigt, wenn der echte Spieler zur richtigen Zeit 'Mau' sagt:

```
Enter (1-2) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
>M
You said 'Mau!' The end of the game is approaching! Now, play the card
which matches the open card.
```

Der folgende Dialog wird verwendet, wenn der echte Spieler 'Mau-Mau' sagt und das Spiel gewinnt:

```
Enter (1-1) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
>MM
You said 'Mau-Mau!' Luck is on your side. Now, play your last card which
matches the open card and celebrate your victory!
Enter (1-1) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
>1
You played TEN of HEARTS against QUEEN of HEARTS. Well done.
*** CONGRATULATIONS, Max! ***
 *** YOU HAVE WON! ***
Press enter to finish.
```

Wenn der Spieler das Spiel beendet:

```
Enter (1-5) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
>Q
 So, you decided to end the game, Max! Well, if you want to play  another
time, the Mau-Mau Palace is waiting for you. May the luck be with you!
Press enter to finish.
```

4.2.2. Meldungen bezüglich des virtuellen Spielers

Der Name des virtuellen Spiels zu Beginn eines Spiels wird zufällig aus der Liste ausgewählt, so dass es nicht zwei virtuelle Spieler mit demselben Namen gibt. Im Spiel kann man von 1 bis 3 virtuellen Spielern teilnehmen, diese Zahl bestimmt den realen Spieler.

Wenn der virtuelle Spieler einen Zug macht, wird der Text angezeigt:

```
O. Henry MAKES THE MOVE:
O. Henry played QUEEN of CLUBS against QUEEN of DIAMONDS.
Success!
```

Folgender Text wird angezeigt, wenn der virtuelle Spieler eine Karte zieht:

```
O. Henry MAKES THE MOVE:
O. Henry drew a card.
```

Dieser Text wird angezeigt, wenn der virtuelle Spieler 'Mau' sagt: 'Mau':

```
Ernest Hemingway MAKES THE MOVE:
Ernest Hemingway said 'Mau!' The end of the game is approaching! It means,
there are only two cards left in the player's hand and one of them is
played now.
Ernest Hemingway played KING of SPADES against NINE of SPADES.
Awesome!
```

Folgender Text ist zu sehen, wenn der virtuelle Spieler 'Mau-Mau' sagt und das Spiel gewinnt:

```
O. Henry MAKES THE MOVE:
O. Henry said 'Mau-Mau!' and celebrates the discarding of the last card.
O. Henry played QUEEN of SPADES against JACK of SPADES.
Not bad.
*** O. Henry HAS WON! ***
Good luck next time, Max!
Press enter to finish.
```

4.3. Dialoge im Spiel mit Sonderregeln

4.3.1. Allgemeine Meldungen

Wenn ein menschlicher Spieler das Spiel mit Sonderregeln beginnt, wird ihm zunächst der gleiche Text mit Grüßen und Informationen zu den allgemeinen Regeln angezeigt, genau wie beim Spielstart mit den Standardregeln, aber schließlich kann man noch den Zusatztext sehen:

```
Welcome to the Mau-Mau Palace! Here we offer you your favourite Mau-Mau
game against virtual players with standard and special rules.
/.../
Special rules:
* A SEVEN forces the next player to take two cards from the drawing stack -
unless he can counter the attack with his own SEVEN. If SEVEN is played
again,  next player must play SEVEN too or draw four cards. Enter 'D' once
to draw and you will be given a required number of cards. After that, the
game continues as usual.
* An EIGHT means: skip a round, entering 'S' - unless you can counter the
attack with your own EIGHT and let the following player skip. This rule is
no longer valid if anyone skips a round or if a second player counters with
an EIGHT.
* A NINE means: The direction of play changes. If there are only two play-
ers, the card means  a one-time skipping a round for the next player.
* A JACK is a wish card, it gives a player the right to request a suit of
card. The next player have to play with the wished suit or draw 2 cards.
Moreover, JACK can be played on any card regardless of rank or suit. But
JACK on JACK is forbidden. JACK does not neutralise a SEVEN or an EIGHT.
```

4.3.2. Dialoge mit Sonderkarte Sieben

Dieser Text wird angezeigt, wenn ein echter Spieler eine 'Sieben' spielt:

```
Enter (1-6) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
>4
You played SEVEN of SPADES against SEVEN of DIAMONDS.
SEVEN is played! Next player must play SEVEN or draw two cards!
```

Wenn ein virtueller Spieler eine 'Sieben' spielt, wird analog angezeigt:

```
James Joyce MAKES THE MOVE:
James Joyce played SEVEN of HEARTS against JACK of HEARTS.
SEVEN is played! Next player must play SEVEN or draw two cards!
```

Wenn die Karte 'Sieben' ein zweites Mal gespielt wird, wird dieser Text angezeigt.

```
Dylan Thomas MAKES THE MOVE:
Dylan Thomas played SEVEN of CLUBS against SEVEN of HEARTS.
SEVEN is played again! Next Player must play SEVEN or draw four cards!
```

Wenn Karte 'Sieben' zum dritten Mal gespielt wird:

```
O. Henry MAKES THE MOVE:
O. Henry played EIGHT of HEARTS against EIGHT of DIAMONDS.
SEVEN is played third time! Game continues as usual.
```

Wenn ein echter Spieler eine falsche Karte spielen möchte, nachdem 'Sieben' gespielt wurde:

```
Open card: SEVEN of CLUBS
Enter (1-4) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
>2
Play SEVEN or draw 2! Choose another card or enter 'D' to draw (you will be
given four cards).
```

Oder

```
Play SEVEN or draw 4! Choose another card or enter 'D' to draw (you will be
given four cards).
```

4.3.3. Dialoge mit Sonderkarte Acht

Wenn ein virtueller Spieler eine 'Acht' spielt, wird angezeigt:

```
Dylan Thomas MAKES THE MOVE:
Dylan Thomas played EIGHT of CLUBS against KING of CLUBS.
EIGHT is played! Next player must play EIGHT or skip a round!
```

Dieser Text wird angezeigt, wenn ein echter Spieler eine 'Acht' spielt:

```
You played EIGHT of DIAMONDS against EIGHT of CLUBS.
EIGHT is played again! Next Player must play EIGHT or skip a round!
```

Wiederholtes Spiel mit einer 'Acht' führt zu ähnlichen Botschaften wie bei einem Karten-spiel mit einer 'Sieben'.

Wenn ein Spieler einen Zug überspringt:

```
Open card: EIGHT of HEARTS
Enter (1-5) to play a card, 'S' - to skip the round, 'M' - to say 'Mau!',
'MM' - to say 'Mau-Mau!', 'Q' - to quit the game.
>S
You skipped this round.
```

4.3.4. Dialoge mit Sonderkarte Neun

Folgender Text wird angezeigt, wenn 'Neun' gespielt wird:

```
Dylan Thomas MAKES THE MOVE:
Dylan Thomas played NINE of HEARTS against QUEEN of HEARTS.
NINE is played! The direction of play changes.
```

4.3.5. Dialoge mit Sonderkarte Bube

Wenn der virtuelle Spieler einen 'Buben' spielt, wird der Text angezeigt::

```
O. Henry MAKES THE MOVE:
O. Henry played JACK of HEARTS against NINE of HEARTS.
JACK is played! Player will now make a wish for a suit.
Next player must play a card with that suit or draw two cards.
O. Henry requested  SPADES.
```

Dieser Text wird angezeigt, wenn der virtuelle Spieler einen Wunsch erfüllt:

```
Charles Bukowski MAKES THE MOVE:
Charles Bukowski played QUEEN of SPADES against JACK of HEARTS.
SPADES played! Wish fulfilled. OK.
```

Dieser Dialog wird verwendet, wenn ein echter Spieler 'Bube' spielt und einen Wunsch äußert:

```
You played JACK of SPADES against QUEEN of SPADES.
JACK is played! Player will now make a wish for a suit. Next player must
play a card with that suit or draw two cards.
Please choose a suit. Enter 'S' for SPADES, 'C' for CLUBS,
'H' for HEARTS, 'D' for DIAMONDS.
>S
You requested  SPADES.
```

Wenn ein echter Spieler auf Wunsch einen falschen Wert eingibt, diese Meldung ist zu sehen:

```
Please choose a suit. Enter 'S' for SPADES, 'C' for CLUBS, 'H' for HEARTS,
'D' for DIAMONDS.
>Q
Incorrect input. Expected value 'S', 'C', 'H', or 'D', but received Q.
Please choose a suit. Enter 'S' for SPADES, 'C' for CLUBS, 'H' for HEARTS,
'D' for DIAMONDS.
```

Wenn ein echter Spieler nach dem Wunsch von anderer Spieler eine falsche Karte versucht zu spielen:

```
Open card: JACK of DIAMONDS
Enter (1-4) to play a card, 'D' - to draw a card, 'M' - to say 'Mau!', 'MM'
- to say 'Mau-Mau!', 'Q' - to quit the game.
>1
Play HEARTS or draw 2! Choose HEARTS if you have it or enter 'D' to draw
(you will be given two cards).
```

5. Frameworks

Dieser Abschnitt beschreibt die im Projekt eingesetzten Frameworks, ihre Konfiguration und Verwendung, veranschaulicht durch Code-Beispiele.

5.1. Spring

5.1.1. Beschreibung

Das Projekt verwendet Spring-Context zur Behandlung von Dependency Injections, um die Abhängigkeit zwischen Komponenten zu verarbeiten. Spring-Context ist ein Teil des Spring Frameworks, das ein Open-Source Framework für die Java-Plattform ist und unter der Apache Licence 2.0 vertrieben wird. Ziel des Spring Frameworks ist es, die Entwicklung mit Java zu vereinfachen, dabei steht die Entkopplung der Applikationskomponenten im Vordergrund.

5.1.2. Verbindung

Im Mau-mau-Projekt ist der Spring-Context als Maven-Abhängigkeit enthalten:

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.1.1.RELEASE</version>
</dependency>
```

5.1.3. Schnittstelle und Konfiguration

Um Dependency-Injection zu verwenden, muss man die Spring Context-Schnittstelle definieren und implementieren.

Definition:

```
public interface ConfigurationService {
    /**
     * Configurable application context (DI container)
     *
     * @return the configurable application context
     */
    public ConfigurableApplicationContext context();
}
```

Implementation:

```
@Configuration
public class ConfigurationServiceImpl implements ConfigurationService {
    @Override
    public ConfigurableApplicationContext context(){
        return new AnnotationConfigApplicationContext("htwberlin");
    }
```

Die gemeinsam genutzten Komponenten können dann mit einem @Bean-Annotation versehen werden:

```
@Bean(name = "entityManager")
private static EntityManager createEntityManager() {
    EntityManagerFactory entityManagerFactory = Persis-
tence.createEntityManagerFactory("mau_mau");
    EntityManager entityManager = entityManagerFacto-
ry.createEntityManager();
    return entityManager;
}
```

5.1.4. Injektion

Dependencies sind mittles @Autowired-Annotation injiziert. Instanzen von gemeinsam genutzten Komponenten werden innerhalb der Spring Configuration-Klasse gespeichert. Um Dependency Injection zu verwenden, sollte die Anwendungsklasse mit @Component Annotation markiert werden.

```
@Component
public class GameServiceImpl implements GameService {

    @Autowired
    private EntityManager entityManager;

    @Autowired
    private RealPlayerService realPlayerService;

    @Autowired
    private VirtualPlayerService virtualPlayerService;

    @Autowired
    private CardService cardService;

    @Autowired
    private RulesProvider rulesProvider;
```

5.1.5. Diagram

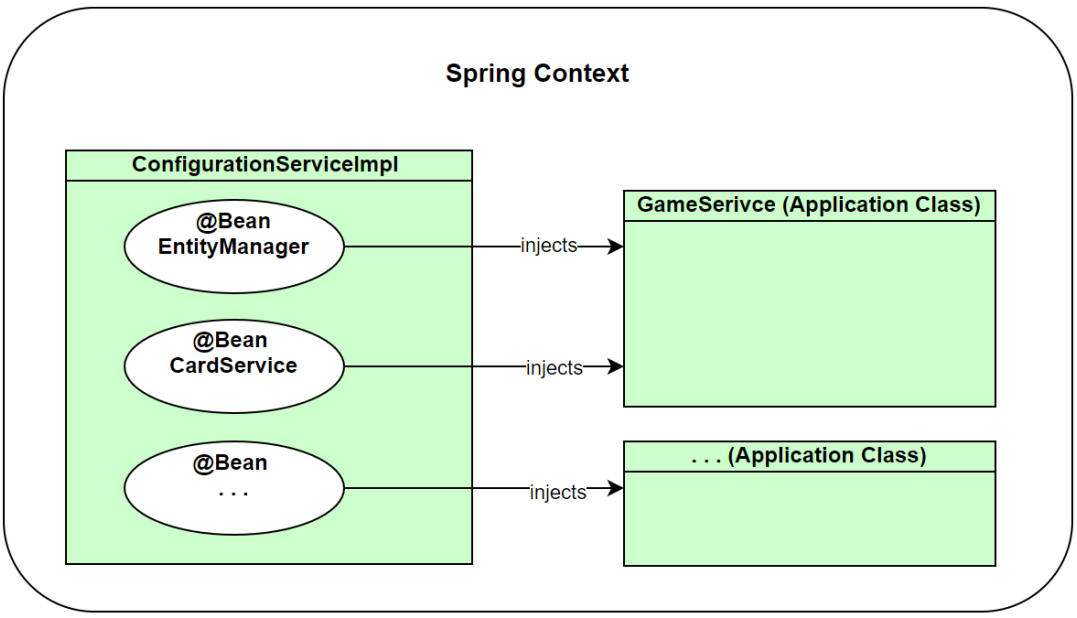

Abbildung 3: Spring Context

5.2. Log4j

5.2.1. Beschreibung

Für die Protokollierung der Anwendungsmeldungen des Programms im Projekt wurde der Log4j Framework verwendet. Das ist freie Software, die unter der Apache Licence 2.0 vertrieben wird. Es ermöglicht die Ausgabe von Protokollanweisungen an verschiedene Ausgabeziele und Protokollebenen. Log4j ist ein De-facto-Standard in der Java-Entwicklung.

5.2.2. Verbindung

Log4j wird dem Projekt mittels Maven-Abhängigkeit hinzugefügt:

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.13.3</version>
</dependency>
```

5.2.3. Verwendung

Instanz der Klasse Logger wird nach Komponente erstellt.

```
private static final Logger LOGGER = LogMana-
ger.getLogger(ViewControllerImpl.class);
```

Dann kann diese Instanz zur Protokollierung von Meldungen verwendet werden

```
LOGGER.error(String.format(USER_INPUT_EXCEPTION + " %s", e.toString()));
```

5.2.4. Protokollebenen

Es stehen 8 Protokollebenen zur Verfügung. Nachrichten mit unterschiedlichem Schweregrad können in verschiedene Protokollebenen gestellt werden. Von Log4j unterstützte Protokollebenen sind ALL, DEBUG, INFO, WARN, ERROR, FATAL, OFF, TRACE

Die erforderliche Log-Stufe kann in der Log4j-Konfiguration eingestellt werden. Nachrichten mit niedrigerem Log-Level werden nicht protokolliert. Das Projekt hat derzeit: `logger.app.level = error`

Im Prozess der Anwendungsentwicklung wurde der Grad die Stufe DEBUG intensiv genutzt:

```
if (LOGGER.isDebugEnabled()) {
LOGGER.debug(String.format("*** EIGHT STATUS BEFORE PLAYER MOVE: %s ",
        isEightPlayed));
}
```

5.3. Hibernate

5.3.1. Beschreibung

Hibernate bietet die Möglichkeit, ein objektorientiertes Domänenmodell auf eine relationale Datenbank abzubilden. Beziehungen zwischen Objekten werden auf entsprechende Datenbank-Relationen abgebildet. Hibernate ist freie Software, die unter der GNU Lesser General Public License 2.1 vertrieben wird.

5.3.2. Verbindung

In diesem Projekt ist Hibernate über die Maven-Abhängigkeit verknüpft.

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>5.4.10.Final</version>
</dependency>
```

5.3.3. Konfiguration

Hibernate wird über die Konfigurationsdatei `persistence.xml` konfiguriert. Jede Klasse, die in der Datenbank gespeichert werden soll, muss in `<class>` Sektion in persistence.xml vorhanden sein.

```
<persistence-unit name="mau_mau">
    <class>htwberlin.mau_mau.game_management.data.GameData</class>
```

5.3.4. Verwendung

Datenbanktabellen, Entitäten und Beziehungen werden von Hibernate automatisch generiert. Anmerkungen werden verwendet, um Java-Klassen und -Felder zu markieren.

`@Entity`: Markierung von Klassen, die in der Datenbank gespeichert werden sollen
`@Id`: Als Primärschlüsselfeld wird ein numerische Attribut innerhalb der Klasse markiert.
`@OneToOne`: Markierung von Feldern, die auf andere Klassen verweisen
`@OneToMany`: Markierung von Sammlungen, zum Beispiel Listen.
`@ManyToMany`: Annotation wird verwendet, wenn eine bidirektionale Beziehung zwischen zwei Objekttypen erforderlich ist.

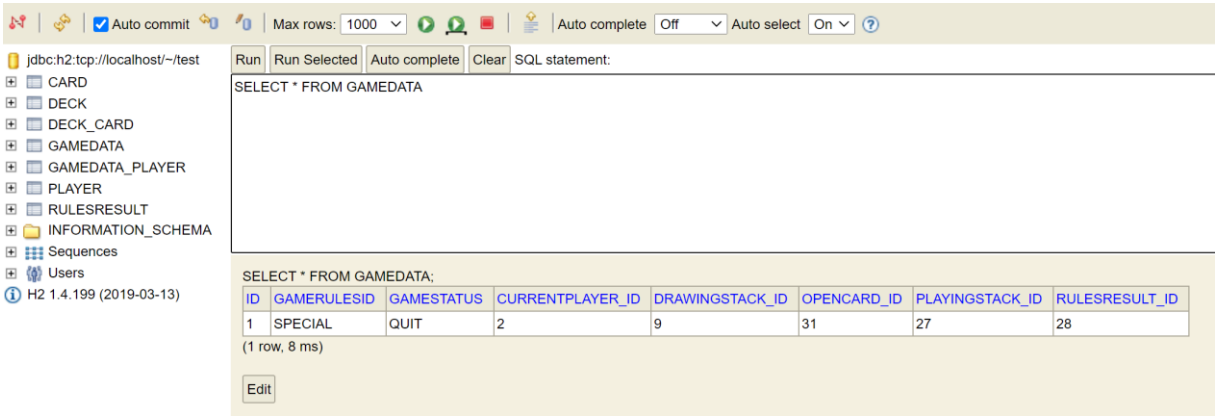

Abbildung 4: H2 Database

5.4. JUnit

5.4.1. Beschreibung

Zum Testen der Anwendung wird die weit verbreitete JUnit - Framework verwendet, die sich besonders für automatisierte Tests einzelner Units unter isolierten Bedingungen eignet. Um eventuell auftretende Probleme zu isolieren, werden Komponenten (Klassen oder Methoden) unabhängig voneinander getestet. Ersatzkomponenten wie Methodenstubs und Mock-Objekte werden benutzt, um Module isoliert zu testen.

5.4.2. Verbindung

In diesem Projekt ist JUnit ebenso über eine Maven-Abhängigkeit verknüpft

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
```

5.4.3. Verwendung

JUnit Tests sind als Java-Klassen implementiert und werden unabhängig oder von Maven während der Projekterstellung ausgeführt.

Grundstruktur des JUnit-Tests:

```
public class RealPlayerServiceTest {
    private RealPlayerService realPlayerService;

    @Test
    public void testSetSaidMau() {
        //Arrange
        RealPlayerService realPlayerService = new RealPlayerServiceImpl();
        RealPlayer player = new RealPlayer("testName");
        //Act
        realPlayerService.setSaidMau(true, player);
        boolean result = realPlayerService.isSaidMau(player);
        //Assert
        Assert.assertTrue(result);
    }
    /.../
}
```

5.5. EasyMock

5.5.1. Beschreibung

In Kombination mit Junit wurde im Projekt EasyMock verwendet, ein Framework zur Emulation von Objekten, um vollständig isolierte Tests implementieren zu können. Mock Objects werden erstellt und konfiguriert, bevor der Test ausgeführt wird.

5.5.2. Verbindung

EasyMock wird dem Projekt über eine Maven-Abhängigkeit in pom.xml hinzugefügt.

```
<dependency>
  <groupId>org.easymock</groupId>
  <artifactId>easymock</artifactId>
  <version>4.2</version>
  <scope>test</scope>
</dependency>
```

5.5.3. Verwendung

EasyMock-Objekte werden in mehreren Schritten verwendet. Zuerst sollte das Mock-Objekt erstellt werden:

```
@Mock(MockType.NICE)
private CardService cardServiceMock;
```

Dann wird das erwartete Verhalten definiert und die Mock-Objekte werden wiedergegeben:

```
expect(cardServiceMock.createDrawingStack()).andReturn(drawingStack);
replay(cardServiceMock);
```

Nachdem der Test durchgeführt wurde, sollte das Scheinobjekt verifiziert werden, um zu überprüfen, ob die erwarteten Methoden ausgeführt wurden:

```
verify(cardServiceMock);
```

6. Ablaufumgebung

Die Architektur der technischen Infrastruktur beschäftigt sich im Wesentlichen mit der Auswahl und Festlegung der „großen" Softwarekomponenten wie Datenbank (…) usw. sowie mit der Festlegung der Zielplattform i.S. Betriebssystem, ggf. Server-Hardware und Netzwerkinfrastruktur. Die technische Architektur zeigt auf, wie diese Infrastruktur zusammenspielt und wie fachliche Usecases auf die verwendeten Technologien abgebildet werden.

Die Mau-mau-Anwendung ist so konzipiert, dass sie auf einem Computersystem mit folgenden Anforderungen läuft:

- Betriebssystem, auf dem Java Runtime Environment 1.8 laufen kann
- Text-Konsole
- H2-Datenbank (lokal bereitgestellt) https://www.h2database.com/

Es gibt keine zusätzlichen Systemabhängigkeiten wie Netzwerkinfrastruktur oder Serverkonfiguration.

Die Anwendung Mau Mau Kartenspiel wird auf folgenden Rechner in folgender Umgebung eingesetzt:

- Microsoft Surface Book 2 Laptop mit Intel® Core™ i7-6600U CPU 2.60 GHz, 2.81 GHz, eingebauter RAM 8,00 GB
- Windows 10 Pro, Version 2004, Betriebssystem Build 19041.508, Systemtyp: 64-Bit-Betriebssystem, x64-basierter Prozessor

Die integrierte Entwicklerumgebung:

- IntelliJ Idea 2020 Community Edition
- JVM 1.8

Als Versionskontrollsystem wurde Git verwendet:

https://github.com/olgapetrova-git/mau_mau

Abbildungsverzeichnis