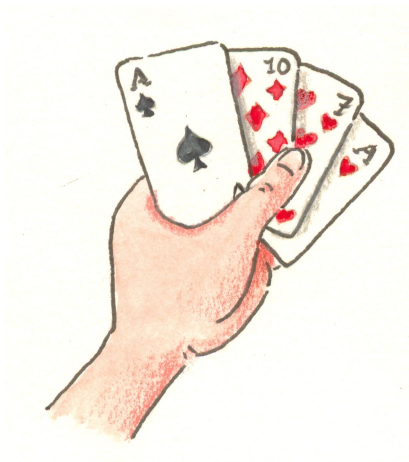


# SPEZIFIKATION

12. November 2013



## NET-WIZHEARTS

Phase	Verantwortlicher	E-Mail
Pflichtenheft	Alina Meixl	alina@meixl.de
Entwurf	Viktoria Witka	witkaviktoria@freenet.de
Spezifikation	Daniel Riedl	dariedl14@yahoo.de
Implementation	Andreas Altenbuchner	a.andi007@gmail.com
Verifikation	Patrick Kubin	kubin@fim.uni-passau.de
Präsentation	w	w

## Inhaltsverzeichnis

<b>1</b>	<b>Hierarchie-Verzeichnis</b>	<b>2</b>
1.1	Klassenhierarchie . . . . .	2
<b>2</b>	<b>Klassen-Verzeichnis</b>	<b>4</b>
2.1	Auflistung der Klassen . . . . .	4
<b>3</b>	<b>Klassen-Dokumentation</b>	<b>8</b>
3.1	ClientController Klassenreferenz . . . . .	8
3.1.1	Ausführliche Beschreibung . . . . .	8
3.2	ClientMain Klassenreferenz . . . . .	9
3.2.1	Ausführliche Beschreibung . . . . .	9
3.2.2	Dokumentation der Elementfunktionen . . . . .	9
3.3	ClientModel Klassenreferenz . . . . .	9
3.3.1	Ausführliche Beschreibung . . . . .	10
3.3.2	Beschreibung der Konstruktoren und Destruktoren . . . . .	10
3.3.3	Dokumentation der Elementfunktionen . . . . .	11
3.4	ClientState Enum-Referenz . . . . .	17
3.4.1	Ausführliche Beschreibung . . . . .	17
3.5	MessageListenerThread Klassenreferenz . . . . .	17
3.5.1	Ausführliche Beschreibung . . . . .	18
3.5.2	Dokumentation der Elementfunktionen . . . . .	18
3.6	ChooseCards Klassenreferenz . . . . .	18
3.6.1	Ausführliche Beschreibung . . . . .	18
3.6.2	Dokumentation der Elementfunktionen . . . . .	19
3.7	Chooseltem Klassenreferenz . . . . .	19
3.7.1	Ausführliche Beschreibung . . . . .	19
3.7.2	Dokumentation der Elementfunktionen . . . . .	19
3.8	CreateGame Klassenreferenz . . . . .	19
3.8.1	Ausführliche Beschreibung . . . . .	20
3.8.2	Dokumentation der Elementfunktionen . . . . .	20
3.9	DiscardPile Klassenreferenz . . . . .	21
3.9.1	Ausführliche Beschreibung . . . . .	21
3.10	DrawDeck Klassenreferenz . . . . .	21
3.10.1	Ausführliche Beschreibung . . . . .	21
3.11	Game Klassenreferenz . . . . .	21
3.11.1	Ausführliche Beschreibung . . . . .	22
3.11.2	Beschreibung der Konstruktoren und Destruktoren . . . . .	22
3.11.3	Dokumentation der Elementfunktionen . . . . .	22
3.12	GameLobby Klassenreferenz . . . . .	23

3.12.1 Ausführliche Beschreibung . . . . .	23
3.12.2 Dokumentation der Elementfunktionen . . . . .	23
3.13 GamePanel Klassenreferenz . . . . .	25
3.13.1 Ausführliche Beschreibung . . . . .	25
3.14 InputNumber Klassenreferenz . . . . .	25
3.14.1 Ausführliche Beschreibung . . . . .	25
3.14.2 Dokumentation der Elementfunktionen . . . . .	26
3.15 Language Enum-Referenz . . . . .	26
3.15.1 Ausführliche Beschreibung . . . . .	26
3.16 Lobby Klassenreferenz . . . . .	26
3.16.1 Ausführliche Beschreibung . . . . .	27
3.16.2 Dokumentation der Elementfunktionen . . . . .	27
3.17 Login Klassenreferenz . . . . .	28
3.17.1 Ausführliche Beschreibung . . . . .	29
3.17.2 Dokumentation der Elementfunktionen . . . . .	29
3.18 OtherPlayer Klassenreferenz . . . . .	30
3.18.1 Ausführliche Beschreibung . . . . .	30
3.19 OwnHand Klassenreferenz . . . . .	30
3.19.1 Ausführliche Beschreibung . . . . .	30
3.20 Password Klassenreferenz . . . . .	30
3.20.1 Ausführliche Beschreibung . . . . .	31
3.20.2 Dokumentation der Elementfunktionen . . . . .	31
3.21 ScoreWindow Klassenreferenz . . . . .	31
3.21.1 Ausführliche Beschreibung . . . . .	32
3.21.2 Dokumentation der Elementfunktionen . . . . .	32
3.22 ViewCard Klassenreferenz . . . . .	32
3.22.1 Ausführliche Beschreibung . . . . .	32
3.22.2 Beschreibung der Konstruktoren und Destruktoren . . . . .	32
3.22.3 Dokumentation der Elementfunktionen . . . . .	33
3.23 Warning Klassenreferenz . . . . .	33
3.23.1 Ausführliche Beschreibung . . . . .	33
3.23.2 Dokumentation der Elementfunktionen . . . . .	33
3.24 ViewNotification Enum-Referenz . . . . .	34
3.24.1 Ausführliche Beschreibung . . . . .	34
3.25 ComBeenKicked Klassenreferenz . . . . .	34
3.25.1 Ausführliche Beschreibung . . . . .	34
3.25.2 Beschreibung der Konstruktoren und Destruktoren . . . . .	35
3.25.3 Dokumentation der Elementfunktionen . . . . .	35
3.26 ComChatMessage Klassenreferenz . . . . .	35
3.26.1 Ausführliche Beschreibung . . . . .	36

3.26.2	Beschreibung der Konstruktoren und Destruktoren	36
3.26.3	Dokumentation der Elementfunktionen	36
3.27	ComClientLeave Klassenreferenz	36
3.27.1	Ausführliche Beschreibung	37
3.27.2	Dokumentation der Elementfunktionen	37
3.28	ComClientQuit Klassenreferenz	37
3.28.1	Ausführliche Beschreibung	37
3.28.2	Dokumentation der Elementfunktionen	37
3.29	ComCreateGameRequest Klassenreferenz	38
3.29.1	Ausführliche Beschreibung	38
3.29.2	Beschreibung der Konstruktoren und Destruktoren	38
3.29.3	Dokumentation der Elementfunktionen	39
3.30	ComGameEnd Klassenreferenz	40
3.30.1	Ausführliche Beschreibung	40
3.30.2	Beschreibung der Konstruktoren und Destruktoren	40
3.30.3	Dokumentation der Elementfunktionen	40
3.31	ComInitGameLobby Klassenreferenz	41
3.31.1	Ausführliche Beschreibung	41
3.31.2	Beschreibung der Konstruktoren und Destruktoren	41
3.31.3	Dokumentation der Elementfunktionen	42
3.32	ComInitLobby Klassenreferenz	42
3.32.1	Ausführliche Beschreibung	43
3.32.2	Beschreibung der Konstruktoren und Destruktoren	43
3.32.3	Dokumentation der Elementfunktionen	43
3.33	ComJoinRequest Klassenreferenz	44
3.33.1	Ausführliche Beschreibung	44
3.33.2	Beschreibung der Konstruktoren und Destruktoren	44
3.33.3	Dokumentation der Elementfunktionen	44
3.33.4	Dokumentation der Datenelemente	46
3.34	ComKickPlayerRequest Klassenreferenz	46
3.34.1	Ausführliche Beschreibung	46
3.34.2	Beschreibung der Konstruktoren und Destruktoren	46
3.34.3	Dokumentation der Elementfunktionen	47
3.35	ComLobbyUpdateGamelist Klassenreferenz	47
3.35.1	Ausführliche Beschreibung	48
3.35.2	Beschreibung der Konstruktoren und Destruktoren	48
3.35.3	Dokumentation der Elementfunktionen	48
3.36	ComLoginRequest Klassenreferenz	48
3.36.1	Ausführliche Beschreibung	49
3.36.2	Beschreibung der Konstruktoren und Destruktoren	49

3.36.3 Dokumentation der Elementfunktionen . . . . .	49
3.37 ComObject Schnittstellenreferenz . . . . .	50
3.37.1 Ausführliche Beschreibung . . . . .	50
3.37.2 Dokumentation der Elementfunktionen . . . . .	50
3.38 ComRuleset Klassenreferenz . . . . .	51
3.38.1 Ausführliche Beschreibung . . . . .	51
3.38.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	51
3.38.3 Dokumentation der Elementfunktionen . . . . .	51
3.39 ComServerAcknowledgement Klassenreferenz . . . . .	52
3.39.1 Ausführliche Beschreibung . . . . .	52
3.39.2 Dokumentation der Elementfunktionen . . . . .	52
3.40 ComStartGame Klassenreferenz . . . . .	52
3.40.1 Ausführliche Beschreibung . . . . .	53
3.40.2 Dokumentation der Elementfunktionen . . . . .	53
3.41 ComUpdatePlayerlist Klassenreferenz . . . . .	53
3.41.1 Ausführliche Beschreibung . . . . .	53
3.41.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	53
3.41.3 Dokumentation der Elementfunktionen . . . . .	55
3.42 ComWarning Klassenreferenz . . . . .	55
3.42.1 Ausführliche Beschreibung . . . . .	56
3.42.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	56
3.42.3 Dokumentation der Elementfunktionen . . . . .	56
3.43 MsgCard Klassenreferenz . . . . .	56
3.43.1 Ausführliche Beschreibung . . . . .	57
3.43.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	57
3.43.3 Dokumentation der Elementfunktionen . . . . .	57
3.44 MsgCardRequest Klassenreferenz . . . . .	58
3.44.1 Ausführliche Beschreibung . . . . .	58
3.44.2 Dokumentation der Elementfunktionen . . . . .	58
3.45 MsgGameEnd Klassenreferenz . . . . .	58
3.45.1 Ausführliche Beschreibung . . . . .	59
3.45.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	59
3.45.3 Dokumentation der Elementfunktionen . . . . .	59
3.46 MsgMultiCards Klassenreferenz . . . . .	60
3.46.1 Ausführliche Beschreibung . . . . .	60
3.46.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	60
3.46.3 Dokumentation der Elementfunktionen . . . . .	60
3.47 MsgMultiCardsRequest Klassenreferenz . . . . .	61
3.47.1 Ausführliche Beschreibung . . . . .	61
3.47.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	61

3.47.3 Dokumentation der Elementfunktionen . . . . .	61
3.48 MsgNumber Klassenreferenz . . . . .	62
3.48.1 Ausführliche Beschreibung . . . . .	62
3.48.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	62
3.48.3 Dokumentation der Elementfunktionen . . . . .	62
3.49 MsgNumberRequest Klassenreferenz . . . . .	63
3.49.1 Ausführliche Beschreibung . . . . .	63
3.49.2 Dokumentation der Elementfunktionen . . . . .	63
3.50 MsgSelection Klassenreferenz . . . . .	64
3.50.1 Ausführliche Beschreibung . . . . .	64
3.50.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	64
3.50.3 Dokumentation der Elementfunktionen . . . . .	64
3.51 MsgSelectionRequest Klassenreferenz . . . . .	66
3.51.1 Ausführliche Beschreibung . . . . .	66
3.51.2 Dokumentation der Elementfunktionen . . . . .	66
3.52 MsgUser Klassenreferenz . . . . .	67
3.52.1 Ausführliche Beschreibung . . . . .	67
3.52.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	67
3.52.3 Dokumentation der Elementfunktionen . . . . .	67
3.53 RulesetMessage Schnittstellenreferenz . . . . .	68
3.53.1 Ausführliche Beschreibung . . . . .	68
3.53.2 Dokumentation der Elementfunktionen . . . . .	68
3.54 Card Schnittstellenreferenz . . . . .	69
3.54.1 Ausführliche Beschreibung . . . . .	69
3.54.2 Dokumentation der Elementfunktionen . . . . .	69
3.55 ClientHearts Klassenreferenz . . . . .	69
3.55.1 Ausführliche Beschreibung . . . . .	70
3.55.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	70
3.55.3 Dokumentation der Elementfunktionen . . . . .	70
3.56 ClientRuleset Klassenreferenz . . . . .	71
3.56.1 Ausführliche Beschreibung . . . . .	71
3.56.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	71
3.56.3 Dokumentation der Elementfunktionen . . . . .	72
3.57 ClientWizard Klassenreferenz . . . . .	74
3.57.1 Ausführliche Beschreibung . . . . .	74
3.57.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	75
3.57.3 Dokumentation der Elementfunktionen . . . . .	76
3.58 Colour Enum-Referenz . . . . .	77
3.58.1 Ausführliche Beschreibung . . . . .	77
3.59 GameClientUpdate Klassenreferenz . . . . .	77

3.59.1 Ausführliche Beschreibung . . . . .	77
3.59.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	77
3.59.3 Dokumentation der Elementfunktionen . . . . .	78
3.60 GamePhase Enum-Referenz . . . . .	79
3.60.1 Ausführliche Beschreibung . . . . .	79
3.61 GameState Klassenreferenz . . . . .	79
3.61.1 Ausführliche Beschreibung . . . . .	80
3.61.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	80
3.61.3 Dokumentation der Elementfunktionen . . . . .	80
3.62 HeartsCard Enum-Referenz . . . . .	83
3.62.1 Ausführliche Beschreibung . . . . .	83
3.62.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	83
3.62.3 Dokumentation der Elementfunktionen . . . . .	84
3.63 HeartsData Klassenreferenz . . . . .	84
3.63.1 Ausführliche Beschreibung . . . . .	84
3.64 OtherData Klassenreferenz . . . . .	84
3.64.1 Ausführliche Beschreibung . . . . .	85
3.64.2 Dokumentation der Elementfunktionen . . . . .	85
3.65 PlayerState Klassenreferenz . . . . .	86
3.65.1 Ausführliche Beschreibung . . . . .	86
3.65.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	86
3.65.3 Dokumentation der Elementfunktionen . . . . .	86
3.66 RulesetType Enum-Referenz . . . . .	87
3.66.1 Ausführliche Beschreibung . . . . .	87
3.67 ServerHearts Klassenreferenz . . . . .	87
3.67.1 Ausführliche Beschreibung . . . . .	88
3.67.2 Dokumentation der Elementfunktionen . . . . .	88
3.68 ServerRuleset Klassenreferenz . . . . .	89
3.68.1 Ausführliche Beschreibung . . . . .	90
3.68.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	90
3.68.3 Dokumentation der Elementfunktionen . . . . .	91
3.69 ServerWizard Klassenreferenz . . . . .	95
3.69.1 Ausführliche Beschreibung . . . . .	96
3.69.2 Dokumentation der Elementfunktionen . . . . .	96
3.69.3 Dokumentation der Datenelemente . . . . .	98
3.70 WizardCard Enum-Referenz . . . . .	98
3.70.1 Ausführliche Beschreibung . . . . .	99
3.70.2 Beschreibung der Konstruktoren und Destrukturen . . . . .	99
3.70.3 Dokumentation der Elementfunktionen . . . . .	99
3.71 WizData Klassenreferenz . . . . .	99

3.71.1 Ausführliche Beschreibung . . . . .	100
3.71.2 Dokumentation der Elementfunktionen . . . . .	100
3.72 GameServer Klassenreferenz . . . . .	100
3.72.1 Ausführliche Beschreibung . . . . .	101
3.72.2 Beschreibung der Konstruktoren und Destruktoren . . . . .	101
3.72.3 Dokumentation der Elementfunktionen . . . . .	101
3.72.4 Dokumentation der Datenelemente . . . . .	103
3.73 GameServerRepresentation Klassenreferenz . . . . .	104
3.73.1 Ausführliche Beschreibung . . . . .	104
3.73.2 Beschreibung der Konstruktoren und Destruktoren . . . . .	104
3.74 LobbyServer Klassenreferenz . . . . .	104
3.74.1 Ausführliche Beschreibung . . . . .	105
3.74.2 Dokumentation der Elementfunktionen . . . . .	105
3.75 LobbyServer.ClientListenerThread Klassenreferenz . . . . .	107
3.75.1 Ausführliche Beschreibung . . . . .	107
3.76 Player Klassenreferenz . . . . .	108
3.76.1 Ausführliche Beschreibung . . . . .	108
3.76.2 Beschreibung der Konstruktoren und Destruktoren . . . . .	108
3.76.3 Dokumentation der Elementfunktionen . . . . .	108
3.77 Server Klassenreferenz . . . . .	109
3.77.1 Ausführliche Beschreibung . . . . .	109
3.77.2 Dokumentation der Elementfunktionen . . . . .	110
3.78 ServerMain Klassenreferenz . . . . .	111
3.78.1 Ausführliche Beschreibung . . . . .	111
3.78.2 Dokumentation der Elementfunktionen . . . . .	111
<b>4 JUnit-Tests . . . . .</b>	<b>111</b>
4.1 isValidWizardMove . . . . .	111
4.2 isValidHeartsMove . . . . .	113
4.3 getWinner . . . . .	115
4.4 QuitPlayer . . . . .	119
4.5 Chat . . . . .	120
<b>5 Implementierungsplan . . . . .</b>	<b>123</b>
5.1 Milestone 1 . . . . .	123
5.2 Milestone 2 . . . . .	123
5.3 Milestone 3 . . . . .	124
5.4 Finale Version . . . . .	124
5.5 Gantt-Diagramme . . . . .	125



**Index****127****1 Hierarchie-Verzeichnis****1.1 Klassenhierarchie**

Die Liste der Ableitungen ist -mit Einschränkungen- alphabetisch sortiert:

<b>ClientController</b>	<b>8</b>
<b>ClientMain</b>	<b>9</b>
<b>ClientModel</b>	<b>9</b>
<b>ClientState</b>	<b>17</b>
<b>MessageListenerThread</b>	<b>17</b>
<b>ChooseCards</b>	<b>18</b>
<b>ChooseItem</b>	<b>19</b>
<b>CreateGame</b>	<b>19</b>
<b>DiscardPile</b>	<b>21</b>
<b>DrawDeck</b>	<b>21</b>
<b>Game</b>	<b>21</b>
<b>GameLobby</b>	<b>23</b>
<b>GamePanel</b>	<b>25</b>
<b>InputNumber</b>	<b>25</b>
<b>Language</b>	<b>26</b>
<b>Lobby</b>	<b>26</b>
<b>Login</b>	<b>28</b>
<b>OtherPlayer</b>	<b>30</b>
<b>OwnHand</b>	<b>30</b>
<b>Password</b>	<b>30</b>
<b>ScoreWindow</b>	<b>31</b>
<b>ViewCard</b>	<b>32</b>
<b>Warning</b>	<b>33</b>
<b>ViewNotification</b>	<b>34</b>
<b>ComObject</b>	<b>50</b>
<b>ComBeenKicked</b>	<b>34</b>
<b>ComChatMessage</b>	<b>35</b>

ComClientLeave	36
ComClientQuit	37
ComCreateGameRequest	38
ComGameEnd	40
ComInitGameLobby	41
ComInitLobby	42
ComJoinRequest	44
ComKickPlayerRequest	46
ComLobbyUpdateGamelist	47
ComLoginRequest	48
ComRuleset	51
ComServerAcknowledgement	52
ComStartGame	52
ComUpdatePlayerlist	53
ComWarning	55
RulesetMessage	68
MsgCard	56
MsgCardRequest	58
MsgGameEnd	58
MsgMultiCards	60
MsgMultiCardsRequest	61
MsgNumber	62
MsgNumberRequest	63
MsgSelection	64
MsgSelectionRequest	66
MsgUser	67
Card	69
HeartsCard	83
WizardCard	98
ClientRuleset	71
ClientHearts	69
ClientWizard	74

Colour	77
GameClientUpdate	77
GamePhase	79
GameState	79
OtherData	84
HeartsData	84
WizData	99
PlayerState	86
RulesetType	87
ServerRuleset	89
ServerHearts	87
ServerWizard	95
GameServerRepresentation	104
LobbyServer.ClientListenerThread	107
Player	108
Server	109
GameServer	100
LobbyServer	104
ServerMain	111

## 2 Klassen-Verzeichnis

### 2.1 Auflistung der Klassen

Hier folgt die Aufzählung aller Klassen, Strukturen, Varianten und Schnittstellen mit einer Kurzbeschreibung:

ClientController	
ClientController	8
ClientMain	
ClientMain	9
ClientModel	
ClientModel	9
ClientState	
ClientState	17
MessageListenerThread	
MessageListenerThread	17

ChooseCards	
ChooseCards	18
ChooseItem	
ChooseItem	19
CreateGame	
CreateGame	19
DiscardPile	
DiscardPile	21
DrawDeck	
DrawDeck	21
Game	
Game	21
GameLobby	
GameLobby	23
GamePanel	
GamePanel	25
InputNumber	
InputNumber	25
Language	
Language	26
Lobby	
Lobby	26
Login	
Login	28
OtherPlayer	
OhterPlayer	30
OwnHand	
OwnHand	30
Password	
Password	30
ScoreWindow	
ScoreWindow	31
ViewCard	
ViewCard	32
Warning	
Warning	33
ViewNotification	
ViewNotification	34
ComBeenKicked	
ComBeenKicked	34
ComChatMessage	
ComChatMessage	35

ComClientLeave	
ComClientLeave	36
ComClientQuit	
ComClientQuit	37
ComCreateGameRequest	
ComCreateGameRequest	38
ComGameEnd	
ComGameEnd	40
ComInitGameLobby	
ComInitGameLobby	41
ComInitLobby	
ComInitLobby	42
ComJoinRequest	
ComJoinRequest	44
ComKickPlayerRequest	
ComKickPlayerRequest	46
ComLobbyUpdateGamelist	
ComLobbyUpdateGamelist	47
ComLoginRequest	
ComLoginRequest	48
ComObject	
ComObject	50
ComRuleset	
ComRuleset	51
ComServerAcknowledgement	
ComServerAcknowledgement	52
ComStartGame	
ComStartGame	52
ComUpdatePlayerlist	
ComUpdatePlayerlist	53
ComWarning	
ComWarning	55
MsgCard	
MsgCard	56
MsgCardRequest	
MsgCardRequest	58
MsgGameEnd	
MsgGameEnd	58
MsgMultiCards	
MsgMultiCards	60
MsgMultiCardsRequest	
MsgMultiCardsRequest	61

MsgNumber	
MsgNumber	62
MsgNumberRequest	
MsgNumberRequest	63
MsgSelection	
MsgSelection	64
MsgSelectionRequest	
MsgSelectionRequest	66
MsgUser	
MsgUser	67
RulesetMessage	
RulesetMessage	68
Card	
Card	69
ClientHearts	
ClientHearts	69
ClientRuleset	
ClientRuleset	71
ClientWizard	
ClientWizard	74
Colour	
Colour	77
GameClientUpdate	
GameClientUpdate	77
GamePhase	
GamePhase	79
GameState	
GameState	79
HeartsCard	
HeartsCard	83
HeartsData	
HeartsData	84
OtherData	
OtherData	84
PlayerState	
PlayerState	86
RulesetType	
Ruleset	87
ServerHearts	
ServerHearts	87
ServerRuleset	
ServerRuleset	89

ServerWizard	
ServerWizard	95
WizardCard	
WizardCard	98
WizData	
WizData	99
GameServer	
GameServer	100
GameServerRepresentation	
GameServerRepresentation	104
LobbyServer	
LobbyServer	104
LobbyServer.ClientListenerThread	
ClientListenerThread	107
Player	
Player	108
Server	
Server	109
ServerMain	
ServerMain	111

### 3 Klassen-Dokumentation

#### 3.1 ClientController Klassenreferenz

##### Private Attribute

- [ClientModel](#) **clientModel**
- [ChooseCards](#) **chooseCards**
- [ScoreWindow](#) **scoreWindow**
- [InputNumber](#) **inputNumber**
- [ChooseItem](#) **chooseColour**
- [Lobby](#) **lobby**
- [Set< Login >](#) **login**
- [CreateGame](#) **createGame**
- [GameLobby](#) **gameLobby**
- [Game](#) **game**
- [Password](#) **password**
- [Language](#) **language**
- [Set< Warning >](#) **warning**

##### 3.1.1 Ausführliche Beschreibung

Der [ClientController](#) enthaelt alle ActionListener der View und leitet durch diese Benutzereingaben an das [Client-Model](#) weiter. Sobald der [ClientController](#) von der ClientMain-Klasse erzeugt wird, erzeugt er wiederum das [Client-Model](#) und die Fenster der ClientView, wobei zunaechst nur das Login Fenster sichtbar ist.

## 3.2 ClientMain Klassenreferenz

### Öffentliche, statische Methoden

- static void [main](#) (final String[] args)

### Private Attribute

- [ClientController](#) [clientController](#)

### 3.2.1 Ausführliche Beschreibung

Die [ClientMain](#) Klasse startet den Spielclient und initialisiert dessen Komponenten.

### 3.2.2 Dokumentation der Elementfunktionen

#### 3.2.2.1 static void main ( final String[] args ) [static]

#### Parameter

<i>args</i>	
-------------	--

## 3.3 ClientModel Klassenreferenz

Abgeleitet von Observable.

### Öffentliche Methoden

- [ClientModel](#) ([MessageListenerThread](#) [netIO](#)) throws [IllegalArgumentException](#)
- void [leaveWindow](#) ()
- void [terminateProgram](#) ()
- void [receiveMessage](#) ([ComChatMessage](#) msg)
- void [receiveMessage](#) ([ComInitLobby](#) msg)
- void [receiveMessage](#) ([ComInitGameLobby](#) msg)
- void [receiveMessage](#) ([ComRuleset](#) msg)
- void [receiveMessage](#) ([ComServerAcknowledgement](#) ack)
- void [receiveMessage](#) ([ComBeenKicked](#) msg)
- void [receiveMessage](#) ([ComUpdatePlayerlist](#) update)
- void [receiveMessage](#) ([ComLobbyUpdateGamelist](#) update)
- void [receiveMessage](#) ([ComObject](#) com)
- List< String > [getPlayerlist](#) ()
- List< [GameServerRepresentation](#) > [getLobbyGamelist](#) ()
- List< [Card](#) > [getPlayedCards](#) ()
- List< [Card](#) > [getOwnHand](#) ()
- List< String > [getOtherPlayerData](#) ()
- int [getOwnScore](#) ()
- void [setLanguage](#) (final [Language](#) language)
- [Language](#) [getLanguage](#) ()
- void [kickPlayer](#) (final String name)
- void [hostGame](#) (String gameName, boolean hasPassword, String password, [RulesetType](#) ruleset)
- void [send](#) ([ComObject](#) object)
- void [send](#) ([RulesetMessage](#) msg)
- int [getPlayerCount](#) ()



- String `getWindowText ()`
- List< Card > `getChooseCards ()`
- void `giveChosenCards` (List< Card > cards)
- void `openChooseCards` (List< Card > cards, String text)
- List< Object > `getChooselItems ()`
- void `giveChosenItem` (Object item)
- void `openChooselItem` (List< Object > items, String text)
- void `giveInputNumber` (int number)
- void `openInputNumber` (String text)
- void `sendChatMessage` (final String msg)
- void `joinGame` (final String name, final String password)
- void `startGame ()`
- void `makeMove` (Card card)
- void `createConnection` (final String username, final String serverAdress, final int port)
- String `getWarningText ()`
- RulesetType[] `getRulesets ()`

#### Private Methoden

- void `initGame ()`
- void `informView` (ViewNotification note)

#### Private Attribute

- String `playerName`
- ClientRuleset `ruleset`
- Language `language`
- ClientState `state`
- List< String > **`playerList`**
- String **`chatMessage`**
- Set< GameServerRepresentation > **`gameList`**
- MessageListenerThread `netIO`

### 3.3.1 Ausführliche Beschreibung

Das `ClientModel` ist die Schnittstelle zwischen dem `MessageListenerThread`, dem `ClientRuleset` und der View. Das Model prüft Nachrichten, welche es vom `MessageListenerThread` über die Methode `receiveMessage()` bekommt. RulesetMessages werden an das `ClientRuleset` weitergeleitet. Weiterhin informiert es seine Observer über Veränderungen und stellt ihnen Methoden zu Verfügung um spielrelevante Daten zu lesen. Weiterhin kann das `Client-Model` ComMessages and den Server schicken, um Kommandos des `ClientRulesets` oder Eingaben des Controllers weiterzugeben.

### 3.3.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.3.2.1 ClientModel ( MessageListenerThread netIO ) throws IllegalArgumentException

Erstellt ein `ClientModel` und erwartet als Argument einen `MessageListenerThread` fuer die Netzwerkanbindung.

#### Parameter

---

<i>netIO</i>	<a href="#">MessageListenerThread</a> fuer die Netzwerkverbindung.
--------------	--

## Ausnahmebehandlung

<i>IllegalArgumentException</i>	Wird geworfen bei fehlerhaftem <a href="#">MessageListenerThread</a> Argument.
---------------------------------	--

Benutzt ClientModel.netIO.

## 3.3.3 Dokumentation der Elementfunktionen

## 3.3.3.1 void leaveWindow ( )

Wird aufgerufen, wenn der User die GameLobby verlaesst.

Der Client gelangt zurueck in die Lobby.

## 3.3.3.2 void receiveMessage ( ComChatMessage msg )

Sendet eine eingehende Chatnachricht direkt an alle Observer weiter.

## Parameter

<i>msg</i>	die ankommende ComChatMessage Nachricht
------------	---

## 3.3.3.3 void receiveMessage ( ComInitLobby msg )

Diese Methode wird aufgerufen, falls der Server den Spieler erfolgreich in die Lobby hinzugefügt hat.

Empfaengt die ComInitGameLobby Nachricht, die eine Liste aller Spieler enthaelt, die sich in der Lobby befinden. Speichert diese Liste und benachrichtigt die Observer mit der loginSuccessful [ViewNotification](#).

## Parameter

<i>msg</i>	die ankommende ComInitLobby Nachricht
------------	---------------------------------------

## 3.3.3.4 void receiveMessage ( ComInitGameLobby msg )

Diese Methode wird aufgerufen, falls der Server den Spieler erfolgreich in die GameLobby hinzugefügt hat.

Empfaengt die ComInitGameLobby Nachricht, die eine Liste aller Spieler enthaelt, die sich in der GameLobby befinden. Speichert diese Liste und benachrichtigt die Observer mit der joinGameSuccessful [ViewNotification](#).

## Parameter

<i>msg</i>	die ankommende ComInitGameLobby Nachricht
------------	---

## 3.3.3.5 void receiveMessage ( ComRuleset msg )

Diese Methode wird aufgerufen, falls eine Nachricht für das Regelwerk ankommt.

Die darin enthaltene RulesetMessage wird dem ClientRuleset zur Verarbeitung uebergeben.

## Parameter

<i>msg</i>	Die ankommende ComRuleset Nachricht
------------	-------------------------------------

## 3.3.3.6 void receiveMessage ( ComServerAcknowledgement ack )

Diese Methode wird aufgerufen, falls ein Server Acknowledgement auftritt.

Dabei ist es von Bedeutung, in welchem Zustand sich der Client befindet.

**Parameter**

<i>ack</i>	Eine Bestätigung durch den Server.
------------	------------------------------------

**3.3.3.7 void receiveMessage ( ComBeenKicked msg )**

Diese Methode wird aufgerufen, falls der Spieler aus der Spiellobby durch einen Spielleiter entfernt wurde.

Der Client gelangt zurueck in die Lobby, die Observer werden mit windowChangeForced benachrichtigt.

**Parameter**

<i>msg</i>	die ankommende ComBeenKicked Nachricht
------------	--

**3.3.3.8 void receiveMessage ( ComUpdatePlayerlist update )**

Diese Methode wird aufgerufen, falls auf dem Server ein neuer Spieler die Lobby/GameLobby betreten hat oder sie von einem Spieler verlassen wurde.

Empfaengt die ComUpdatePlayerlist Nachricht, die die Information enthaelt, ob und welcher Spieler hinzugefuegt oder entfernt werden muss. Die Spielerliste wird dementsprechend bearbeitet und die Observer mit playerList-Update informiert.

**Parameter**

<i>update</i>	die ankommende ComLobbyUpdatePlayerlist Nachricht
---------------	---

**3.3.3.9 void receiveMessage ( ComLobbyUpdateGamelist update )**

Diese Methode wird aufgerufen, falls auf dem Server ein neues Spiel erstellt wurde oder ein Spiel geschlossen/beendet wurde.

Empfaengt die ComLobbyUpdateGamelist Nachricht, die die Information enthaelt, ob und welches Spiel hinzugefuegt oder entfernt werden muss. Die Spielliste wird dementsprechend bearbeitet und die Observer mit gameList-Update informiert.

**Parameter**

<i>update</i>	die ankommende ComLobbyUpdateGamelist Nachricht
---------------	---

**3.3.3.10 void receiveMessage ( ComObject com )**

Standard receiveMessage Methode, die ComObjekte zur Weiterverarbeitung identifiziert.

**Parameter**

<i>com</i>	Das auflaufende ComObjekt.
------------	----------------------------

**3.3.3.11 List<String> getPlayerlist ( )**

Liefert eine Liste der Namen der Spieler in der Lobby oder GameLobby.

**Rückgabe**

Liste von Spielernamen

**3.3.3.12 List<GameServerRepresentation> getLobbyGamelist ( )**

Liefert eine Liste der Spiele, die aktuell auf dem Server offen sind oder gerade gespielt werden.

**Rückgabe**

Liste aller Spiele der Lobby.

**3.3.3.13 List<Card> getPlayedCards ( )**

Gibt eine Liste aller bereits ausgespielten Karten zurueck.

**Rückgabe**

List<Card>. Eine Liste der gespielten Karten.

**3.3.3.14 List<Card> getOwnHand ( )**

Gibt eine Liste der Handkarten des Spielers zurueck.

**Parameter**

<i>Liste</i>	aller Handkarten des Spielers
--------------	-------------------------------

**3.3.3.15 List<String> getOtherPlayerData ( )**

Gibt zusaetzliche Daten der anderen Spieler zurueck.

**Rückgabe**

Liste der Stringrepräsentationen der OtherData der Spieler

**3.3.3.16 int getOwnScore ( )**

Gibt den Punktestand des Spielers zurueck.

**Rückgabe**

der eigene Punktestand.

**3.3.3.17 void setLanguage ( final Language language )**

Setzt die Sprache der GUI.

**Parameter**

<i>language</i>	Enumerator der die Spielsprache anzeigt.
-----------------	--

Benutzt ClientModel.language.

**3.3.3.18 Language getLanguage ( )**

Liefert die Sprache der GUI.

**Rückgabe**

language Enumerator der die Spielsprache anzeigt.

Benutzt ClientModel.language.

**3.3.3.19 void kickPlayer ( final String name )**

Entfernt einen Spieler aus der GameLobby.

**Parameter**

<i>Name</i>	des Spielers, der entfernt werden soll
-------------	--

**3.3.3.20 void hostGame ( String *gameName*, boolean *hasPassword*, String *password*, RulesetType *ruleset* )**

Erstellt ein neues Spiel.

Sendet dazu eine ComCreateGameRequest Nachricht an den Server.

**Parameter**

<i>gameName</i>	String Name des Spieles.
<i>hasPassword</i>	true, wenn das Spiel ein Passwort hat
<i>password</i>	String Passwort zum sichern des Spieles.
<i>ruleset</i>	das zu verwendende Regelwerk

**3.3.3.21 void send ( ComObject *object* )**

Sendet erstellte ComObjects an den Server.

**Parameter**

<i>object</i>	ComObject, das verschickt wird
---------------	--------------------------------

Wird benutzt von ClientModel.send() und ClientModel.sendChatMessage().

**3.3.3.22 void send ( RulesetMessage *msg* )**

Sendet eine RulesetMessage an den Server.

Erstellt dazu eine ComRuleset, die die RulesetMessage enthaelt.

**Parameter**

<i>msg</i>	die RulesetMessage, die an den Server geschickt werden soll
------------	---

Benutzt ClientModel.send().

**3.3.3.23 int getPlayerCount ( )**

Die die Anzahl der Spieler eines Spieles zurueck.

**Rückgabe**

int Die Spielerzahl eines Spieles.

**3.3.3.24 String getWindowText ( )**

Gibt den Text zurueck, der in einem Sonderfenster (InputNumber, Chooseltem, ChooseCards) angezeigt werden soll.

**Rückgabe**

String

**3.3.3.25 List<Card> getChooseCards ( )**

Gibt die Karten zurueck, aus denen gewaehlt werden soll.

**Rückgabe**

Karten, aus denen gewahlt werden kann

**3.3.3.26 void giveChosenCards ( List< Card > cards )**

Uebergibt die Karten, die vom User gewahlt wurden.

Diese werden dann dem Regelwerk weitergegeben. Akzeptiert dieses die gewählten Karten nicht, wird nochmal openChooseCards aufgerufen.

Parameter

<i>cards</i>	Karten, die der User gewahlt hat
--------------	----------------------------------

**3.3.3.27 void openChooseCards ( List< Card > cards, String text )**

Benachrichtigt die Observer mit der openChooseCards [ViewNotification](#) und speichert die Liste der Karten sowie den Anzeigetext des Regelwerks zwischen.

Parameter

<i>cards</i>	Liste der Karten, von denen gewaehlt werden kann
<i>text</i>	Text, der dem User angezeigt werden soll

**3.3.3.28 List<Object> getChooseltems ( )**

Gibt die Items zurueck, aus denen eines gewaehlt werden soll.

Rückgabe

Items, aus denen gewahlt werden kann

**3.3.3.29 void giveChosenItem ( Object item )**

Uebergibt das Item, das vom User gewahlt wurden.

Dieses wird dann dem Regelwerk weitergegeben. Akzeptiert dieses das gewählte Item nicht, wird nochmal openChooseltem aufgerufen.

Parameter

<i>item</i>	Item, das der User gewahlt hat
-------------	--------------------------------

**3.3.3.30 void openChooseltem ( List< Object > items, String text )**

Benachrichtigt die Observer mit der openChooseltem [ViewNotification](#) und speichert die Liste der Items, von denen eines gewaehlt werden soll, sowie den Anzeigetext des Regelwerks zwischen.

Parameter

<i>items</i>	Liste der Items, von denen eines gewaehlt werden soll
<i>text</i>	Text, der dem User angezeigt werden soll

**3.3.3.31 void giveInputNumber ( int number )**

Uebergibt die Zahl, die vom User gewahlt wurde.

Diese wird dann dem Regelwerk weitergegeben. Akzeptiert dieses die gewählte Zahl nicht, wird nochmal openInputNumber aufgerufen.

Parameter

<i>number</i>	Zahl, die vom User gewählt wurde
---------------	----------------------------------

### 3.3.3.32 void openInputNumber ( String text )

Benachrichtigt die Observer mit der openInputNumber [ViewNotification](#) und speichert den Anzeigetext des Regelwerks zwischen.

#### Parameter

<i>text</i>	Text, der dem User angezeigt werden soll
-------------	--

### 3.3.3.33 void sendChatMessage ( final String msg )

Nimmt vom [ClientController](#) eine Chatnachricht entgegen und sendet diese an den Server.

#### Parameter

<i>msg</i>	die Chatnachricht, die an den Server geschickt werden soll
------------	--

Benutzt ClientModel.send().

### 3.3.3.34 void joinGame ( final String name, final String password )

Versucht einem Spiel beizutreten.

Sendet dazu eine ComJoinRequest Nachricht an den Server. Wird diese bestaetigt, gelangt der Client in die Game-Lobby. Wird die Nachricht nicht bestaetigt, wird eine Fehlermeldung ausgegeben und die Observer mit openWarning informiert.

#### Parameter

<i>name</i>	String Der Name des Spiels.
<i>password</i>	String Passwort eines Spieles.

### 3.3.3.35 void startGame ( )

Versucht das erstellte Spiel zu starten.

Sendet dazu eine ComStartGame an den Server. Wenn der Client der Spielleiter des Spiels ist, gelangt er ins Spiel. Wenn der Client nicht der Spielleiter des Spiels ist, wird eine Fehlermeldung ausgegeben.

### 3.3.3.36 void initGame ( ) [private]

Diese Methode wird innerhalb des ClientModels aufgerufen wenn ein Spiel vom Spielleiter gestartet wurde.

Der Client gelangt ins Spiel Die Observer werden über die gameStarted [ViewNotification](#) benachrichtigt.

### 3.3.3.37 void makeMove ( Card card )

Versucht eine Karte auszuspielen.

Laesst dazu vom ClientRuleset ueberpruefen ob, die ausgewaehlte Karte gespielt werden darf. Wenn ja, wird sie im ClientRuleset weiterbehandelt. Wenn nein, wird eine Fehlermeldung ausgegeben und dazu die Observer mit openWarning informiert.

#### Parameter

<i>die</i>	ID der gespielten Karte
------------	-------------------------

### 3.3.3.38 void informView ( ViewNotification note ) [private]

Hilfsmethode die alle verbundenen Observer der GUI kontaktiert.

## Parameter

<i>note</i>	Enum der die Art des Aufrufes bestimmt.
-------------	---

3.3.3.39 void createConnection ( final String *username*, final String *serverAdress*, final int *port* )

Erstellt den [MessageListenerThread](#) und fuehrt den Benutzerlogin durch.

## Parameter

<i>username</i>	String der eindeutige Benutzername der für den Login verwendet wird.
<i>serverAdress</i>	String die Adresse des spielservers.
<i>port</i>	Integer der Port des Spielservers.

## 3.3.3.40 String getWarningText ( )

Gibt den Text aus der bei einer Spielwarnung angezeigt wird.

## Rückgabe

String Text der Warnung.

## 3.3.3.41 RulesetType [] getRulesets ( )

Liefert ein Array mit allen implementierten Regelwerken.

## Parameter

<i>RulesetType[]</i>	Array von unterstützten Regelwerken.
----------------------	--------------------------------------

## 3.4 ClientState Enum-Referenz

## Öffentliche Attribute

- [LOGIN](#)
- [SERVERLOBBY](#)
- [GAMECREATION](#)
- [PASSWORDREQUEST](#)
- [GAMELOBBY](#)
- [GAME](#)
- [USERREQUEST](#)
- [ENDING](#)

## 3.4.1 Ausführliche Beschreibung

Dieser Enumerator enthält alle Zustaende in denen sich der Client befinden kann.

## 3.5 MessageListenerThread Klassenreferenz

Abgeleitet von Thread.

## Öffentliche Methoden

- [MessageListenerThread](#) ()
- void [startConnection](#) ([ClientModel](#) model, Socket connection) throws [IllegalArgumentException](#), [IOException](#)



- void `closeConnection` ()
- void `send` (`ComObject` object)
- void `run` ()

#### Private Attribute

- Socket **socket**
- ObjectInput **in**
- ObjectOutput **out**
- boolean **run** = false
- `ClientModel` **model**

#### 3.5.1 Ausführliche Beschreibung

Diese Klasse implementiert die Netzwerkanbindung des Clients an den Server. Sie enthaelt den dazu noetigen Socket und ObjektStream Reader und Writer.

#### 3.5.2 Dokumentation der Elementfunktionen

##### 3.5.2.1 void `startConnection` ( `ClientModel` *model*, Socket *connection* ) throws `IllegalArgumentException`, `IOException`

Initialisiert die ObjectStreams und speichert den TCP Socket im Thread.

#### Parameter

<i>model</i>	<code>ClientModel</code> , Das Model das den Spielablauf und Serverkommunikation steuert.
<i>connection</i>	Socket, der Socket über den die TCP Verbindung laeuft.

#### Ausnahmebehandlung

<i><code>IllegalArgumentException</code></i>	Wird geworfen bei falschen <code>ClientModel</code> oder Socket Argumenten.
<i><code>IOException</code></i>	Wird geworfen beim fehlerbehafteten Erstellen der ObjectStreams.

### 3.6 ChooseCards Klassenreferenz

Abgeleitet von Observer.

#### Öffentliche Methoden

- void `update` (Observable o, Object arg)

#### Private Attribute

- `OwnHand` **playerHandPanel**

#### 3.6.1 Ausführliche Beschreibung

In diesem Fenster muss der Benutzer eine vorbestimmte Menge Karten auswaehlen.

## 3.6.2 Dokumentation der Elementfunktionen

## 3.6.2.1 void update ( Observable o, Object arg )

Wird durch notify() im [ClientModel](#) aufgerufen.

Je nach dem in arg uebergebenen Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

## Parameter

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet: openChooseCards, chooseCardsSuccessful

## 3.7 Chooseltem Klassenreferenz

Abgeleitet von Observer.

## Öffentliche Methoden

- void [update](#) (Observable arg0, Object arg1)

## Private Attribute

- Object **itemComboBox**

## 3.7.1 Ausführliche Beschreibung

Dieses Fenster ermoeoglicht es dem Spieler aus einer Liste von Items eines auszuwaehlen.

## 3.7.2 Dokumentation der Elementfunktionen

## 3.7.2.1 void update ( Observable arg0, Object arg1 )

Wird durch notify() im [ClientModel](#) aufgerufen.

Je nach dem in arg uebergebenen Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

## Parameter

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet: openChooseltem, chooseltemSuccessful

## 3.8 CreateGame Klassenreferenz

Abgeleitet von JFrame.

## Öffentliche Methoden

- [CreateGame](#) ()
- void [addPanelMouseListener](#) (MouseListener m)
- void [addRulesetSelectionListener](#) (ItemListener i)
- void [addCreateButtonListener](#) (ActionListener a)

- void [addLeaveButtonListener](#) (ActionListener a)
- void [setLanguage](#) (Language l)

#### Öffentliche, statische Methoden

- static void **main** (String[] args)

#### Private Methoden

- void **updateLanguage** ()

#### Private Attribute

- [Language](#) **lang**
- JTextField **nameField**
- BufferedImage **image**
- JTextField **passwordField**
- JPanel **imagePanel**
- JLabel **lblSelect**
- JComboBox< String > **rulesetBox**
- JCheckBox **chckbxPassword**
- JButton **btnLeave**
- JButton **btnCreate**
- JLabel **lblGameName**

#### Statische, private Attribute

- static final long **serialVersionUID** = -2893031560688870723L

### 3.8.1 Ausführliche Beschreibung

Das Fenster [CreateGame](#) dient dem Benutzer zur Erstellung eines neuen Spieles. Es bietet alle Komponenten, um ein Regelwerk zu waehlen, einen Spielnamen festzulegen und das Spiel durch ein Passwort zu schuetzen. In der Spielerstellung wird ein Titelbild des ausgewaehlten Spiels und eine kurze Beschreibung angezeigt. ueber 'Leave' kehrt der Spieler in die [Lobby](#) zurueck und mit 'Create' wird das Spiel erstellt.

### 3.8.2 Dokumentation der Elementfunktionen

#### 3.8.2.1 void addPanelMouseListener ( MouseListener m )

Fuegt einen MouseListener zum ImagePanel des [CreateGame](#) Fensters hinzu, der zur Anzeige des MouseOver--Texts verwendet wird.

##### Parameter

<i>m</i>	ein MouseListener
----------	-------------------

#### 3.8.2.2 void addRulesetSelectionListener ( ItemListener i )

Fuegt einen Listener fuer die Regelwerk-Auswahl des [CreateGame](#) Fensters hinzu.

## Parameter

<i>i</i>	ein ItemListener
----------	------------------

3.8.2.3 void addCreateButtonListener ( ActionListener *a* )

Fuegt einen ActionListener fuer den 'Create' Button hinzu.

## Parameter

<i>a</i>	ein ActionListener
----------	--------------------

3.8.2.4 void addLeaveButtonListener ( ActionListener *a* )

Fuegt einen ActionListener fuer den 'Leave' Button hinzu.

## Parameter

<i>a</i>	ein ActionListener
----------	--------------------

3.8.2.5 void setLanguage ( Language *l* )

Aendert die Sprache des Fensters.

## Parameter

<i>l</i>	Sprache in Form des Language-Enums
----------	------------------------------------

## 3.9 DiscardPile Klassenreferenz

## Private Attribute

- Set< [ViewCard](#) > **card**

## 3.9.1 Ausführliche Beschreibung

Stellt einen Ablagestapel dar, dieser kann sowohl für jeden Spieler einzeln oder für alle Spieler gemeinsam in der Mitte des Spielfeldes angezeigt werden.

## 3.10 DrawDeck Klassenreferenz

## 3.10.1 Ausführliche Beschreibung

Stellt einen Aufnahmestapel dar.

## 3.11 Game Klassenreferenz

Abgeleitet von JFrame und Observer.

## Öffentliche Methoden

- [Game](#) () throws IOException
- void [makeTrickGameBoard](#) (int playercount)
- void [update](#) (Observable o, Object arg)
- void [update](#) (Observable o, String arg)

## Öffentliche, statische Methoden

- static void **main** (String[] args) throws IOException

## Private Attribute

- JPanel **contentPane**
- JTextField **textField**
- [GamePanel](#) **panel**

## Statische, private Attribute

- static final long **serialVersionUID** = -2655520138213745249L

### 3.11.1 Ausführliche Beschreibung

Im [Game](#) Fenster laeuft das Spiel ab. Es enthaelt den Spielchat und ein [GamePanel](#). Ausserdem koennen ueber ein Dropdown-Menue Aenderungen an Hintergrundbild und Kartenhintergruenden vorgenommen werden. Schliesseen beendet das Spiel und der Spieler wird in die [Lobby](#) zurueckgeleitet.

### 3.11.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.11.2.1 **Game** ( ) throws IOException

Erstellt das [Game](#) Fenster.

#### Ausnahmebehandlung

<i>IOException</i>	
--------------------	--

### 3.11.3 Dokumentation der Elementfunktionen

#### 3.11.3.1 void makeTrickGameBoard ( int *playercount* )

Arrangiert die Elemente der Spielfeld-Oberflaeche für ein Kartenspiel, bei dem Stiche gemacht werden.

Hierfuer hat jeder Spieler einen eigenen Ablagestapel vor sich. Es koennen 3, 4, 5, oder 6 Spieler gewaehlt werden.

#### Parameter

<i>playercount</i>	Anzahl der Spieler, wobei 3 <= playercount <=6 einzuhalten ist
--------------------	--

#### 3.11.3.2 void update ( Observable *o*, Object *arg* )

Wird durch notify() im [ClientModel](#) aufgerufen.

Je nach dem in arg uebergebenen ViewNotification-Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

#### Parameter

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet: playedCardsUpdate, otherDataUpdate, moveAcknowledged, gameStarted

#### 3.11.3.3 void update ( Observable *o*, String *arg* )

Wird durch notify() im [ClientModel](#) aufgerufen, wenn eine Chatnachricht uebergeben wird.

## Parameter

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet eine Chatnachricht in String-Form

### 3.12 GameLobby Klassenreferenz

Abgeleitet von JFrame und Observer.

## Öffentliche Methoden

- [GameLobby](#) ()
- void [addStartButtonListener](#) (ActionListener a)
- void [addRemoveButtonListener](#) (ActionListener a)
- void [addLeaveButtonListener](#) (ActionListener a)
- void [addChatMessageListener](#) (KeyListener k)
- void [setLanguage](#) ([Language](#) l)
- void [update](#) (Observable o, Object arg)
- void [update](#) (Observable o, String arg)

## Öffentliche, statische Methoden

- static void **main** (String[] args)

## Private Methoden

- void **updateLanguage** ()

## Private Attribute

- JPanel **contentPane**
- JTextField **messageField**
- [Language](#) **lang**
- JButton **btnRemovePlayer**
- JButton **btnLeave**
- JTextArea **chatlog**
- JButton **btnStartGame**

## Statische, private Attribute

- static final long **serialVersionUID** = -1899311213351027436L

#### 3.12.1 Ausführliche Beschreibung

Die [GameLobby](#) modelliert das Wartefenster, in dem beigetretene Spieler auf den Start des Spieles durch den Spielleiter warten. Der Spielleiter kann Spieler mit dem Remove Player Button entfernen. ueber Leave kehren die Spieler in die [Lobby](#) zurueck. Der spielinterne Chat ist ab hier verfuegbar.

#### 3.12.2 Dokumentation der Elementfunktionen

##### 3.12.2.1 void addStartButtonListener ( ActionListener a )

Fuegt einen ActionListener fuer den 'Start [Game](#)' Button hinzu.

**Parameter**

<i>a</i>	ein ActionListener
----------	--------------------

**3.12.2.2 void addRemoveButtonListener ( ActionListener *a* )**

Fuegt einen ActionListener fuer den 'Remove Player' Button hinzu.

**Parameter**

<i>a</i>	ein ActionListener
----------	--------------------

**3.12.2.3 void addLeaveButtonListener ( ActionListener *a* )**

Fuegt einen ActionListener fuer den 'Leave' Button hinzu.

**Parameter**

<i>a</i>	ein ActionListener
----------	--------------------

**3.12.2.4 void addChatMessageListener ( KeyListener *k* )**

Fuegt einen KeyListener fuer das Nachricht-Senden-Feld der [Lobby](#) hinzu.

**Parameter**

<i>k</i>	
----------	--

**3.12.2.5 void setLanguage ( Language *l* )**

Aendert die Sprache des Fensters.

**Parameter**

<i>l</i>	Sprache in Form des Language-Enums
----------	------------------------------------

**3.12.2.6 void update ( Observable *o*, Object *arg* )**

Wird durch notify() im [ClientModel](#) aufgerufen.

Je nach dem in *arg* uebergebenen ViewNotification-Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

**Parameter**

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet: joinGameSuccessful, playerListUpdate, windowChangeForced, gameStarted

**3.12.2.7 void update ( Observable *o*, String *arg* )**

Wird aufgerufen, wenn eine String-Nachricht im notify() uebergeben wird.

Dieser wird als Chatnachricht interpretiert und dem Chatlog angefuegt.

**Parameter**

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet einen String, der eine Chatnachricht darstellt

### 3.13 GamePanel Klassenreferenz

Abgeleitet von JPanel.

#### Öffentliche Methoden

- [GamePanel](#) ()
- void [makeTrickGameBoardThreePlayers](#) ()
- void [makeTrickGameBoardFourPlayers](#) ()
- void [makeTrickGameBoardFivePlayers](#) ()
- void [makeTrickGameBoardSixPlayers](#) ()
- void **paintComponent** (Graphics g)

#### Private Attribute

- [OwnHand](#) **ownHand**
- Object **ownScoreLabel**
- Set< [OtherPlayer](#) > **otherPlayer**
- [DrawDeck](#) **drawDeck**
- Set< [DiscardPile](#) > **discardPiles**
- BufferedImage **background**

#### Statische, private Attribute

- static final long **serialVersionUID** = -1041218552426155968L

#### 3.13.1 Ausführliche Beschreibung

Das [GamePanel](#) ist die Komponente des Game-Fensters, welche das eigentliche Spiel darstellt. Es besteht aus verschiedenen Panelobjekten, welche je nach Regelwerk auf das Spielfeld gezeichnet werden. Dazu gehören die eigenen Karten, eventuell ausgewählte Karten, ein Textfeld z.B. zur Anzeige der Anzahl der restlichen Karten der Mitspieler und den Ablagestapel. Nach jeder Runde wird der Punktestand aktualisiert.

### 3.14 InputNumber Klassenreferenz

Abgeleitet von Observer.

#### Öffentliche Methoden

- void [update](#) (Observable o, Object arg)

#### Private Attribute

- Object **numberTextfield**

#### 3.14.1 Ausführliche Beschreibung

In diesem Fenster, kann der Benutzer eine Zahl eingeben.



### 3.14.2 Dokumentation der Elementfunktionen

#### 3.14.2.1 void update ( Observable o, Object arg )

Wird durch notify() im [ClientModel](#) aufgerufen.

Je nach dem in arg uebergebenen Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

Parameter

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet: openInputNumber, inputNumberSuccessful

## 3.15 Language Enum-Referenz

Öffentliche Attribute

- **German**
- **English**
- **Bavarian**

### 3.15.1 Ausführliche Beschreibung

[Language](#) stellt Repraesentationen verschiedener Sprachen dar, die von der GUI verwendet werden, um festzustellen welche Anzeigesprache verwendet werden soll.

## 3.16 Lobby Klassenreferenz

Abgeleitet von JFrame und Observer.

Öffentliche Methoden

- [Lobby](#) ()
- void [addJoinButtonListener](#) (ActionListener a)
- void [addHostButtonListener](#) (ActionListener a)
- void [addLeaveButtonListener](#) (ActionListener a)
- void [addChatMessageListener](#) (KeyListener k)
- void [setLanguage](#) ([Language](#) l)
- void [update](#) (Observable o, Object arg)
- void [update](#) (Observable o, String arg)

Öffentliche, statische Methoden

- static void **main** (String[] args)

Private Methoden

- void **updateLanguage** ()

## Private Attribute

- JPanel **contentPane**
- JTextField **messageField**
- JList **playerList**
- JList **gameList**
- JScrollPane **scrollPane**
- JButton **btnHostGame**
- JButton **btnJoinGame**
- JButton **btnLeave**
- JTextArea **chatlog**
- [Language](#) **lang**

## Statische, private Attribute

- static final long **serialVersionUID** = 1L

## 3.16.1 Ausführliche Beschreibung

Diese Klasse erzeugt die Ansicht der ServerLobby auf der Client Seite, in der die Spieler neue Spiele erstellen oder offenen beitreten koennen. In der [Lobby](#) werden die Benutzernamen der sich in der [Lobby](#) befindenden Spieler, sowie offene Spiele angezeigt. In der [Lobby](#) koennen Chatnachrichten gesendet und empfangen werden. ueber 'Leave' verlaesst der Spieler das Spiel. ueber 'Host [Game](#)' wird der Spieler zum CreateGame-Fenster weiter geleitet und mit 'Join [Game](#)' kann einem bereits erstellten Spiel beigetreten werden.

## 3.16.2 Dokumentation der Elementfunktionen

## 3.16.2.1 void addJoinButtonListener ( ActionListener a )

Fuegt einen ActionListener fuer den 'Join' Button hinzu.

## Parameter

<i>a</i>	ein ActionListener
----------	--------------------

## 3.16.2.2 void addHostButtonListener ( ActionListener a )

Fuegt einen ActionListener fuer den 'Host' Button hinzu.

## Parameter

<i>a</i>	ein ActionListener
----------	--------------------

## 3.16.2.3 void addLeaveButtonListener ( ActionListener a )

Fuegt einen ActionListener fuer den 'Leave' Button hinzu.

## Parameter

<i>a</i>	ein ActionListener
----------	--------------------

## 3.16.2.4 void addChatMessageListener ( KeyListener k )

Fuegt einen KeyListener fuer das Nachricht-Senden-Feld der [Lobby](#) hinzu.

**Parameter**

<i>k</i>	
----------	--

**3.16.2.5 void setLanguage ( Language l )**

Ändert die Sprache des Fensters.

**Parameter**

<i>l</i>	Sprache in Form des Language-Enums
----------	------------------------------------

**3.16.2.6 void update ( Observable o, Object arg )**

Wird durch notify() im [ClientModel](#) aufgerufen.

Je nach dem in arg uebergebenen ViewNotification-Befehl wird ein Update des Fensters ausgeführt oder eine Fehlermeldung angezeigt.

**Parameter**

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet: joinGameSuccessful, windowChangeForced, playerListUpdate, gameListUpdate, chatMessage

**3.16.2.7 void update ( Observable o, String arg )**

Wird aufgerufen, wenn eine String-Nachricht im notify() uebergeben wird.

Dieser wird als Chatnachricht interpretiert und dem Chatlog angefügt.

**Parameter**

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet einen String, der eine Chatnachricht darstellt

**3.17 Login Klassenreferenz**

Abgeleitet von JFrame und Observer.

**Öffentliche Methoden**

- [Login](#) ()
- void [addConnectButtonListener](#) (ActionListener a)
- void [addLanguageSelectionListener](#) (ItemListener i)
- void [setLanguage](#) (Language l)
- void [update](#) (Observable o, Object arg)

**Öffentliche, statische Methoden**

- static void **main** (String[] args) throws IOException

**Private Methoden**

- void **updateLanguage** ()

## Private Attribute

- JPanel **contentPane**
- JTextField **nameField**
- JTextField **serverField**
- JComboBox< [Language](#) > **languageComboBox**
- JButton **btnConnect**
- [Language](#) **lang**
- JLabel **lblNickname**
- JLabel **lblHostIp**
- JLabel **lblLanguage**

## Statische, private Attribute

- static final long **serialVersionUID** = -2516577977746181978L

## 3.17.1 Ausführliche Beschreibung

Das Login-Fenster repräsentiert den initialen Dialog zwischen Benutzer und Client. In diesem Fenster kann der Benutzer seinen Namen und die Adresse des Servers eingeben. Ausserdem ist ueber den [Login](#) die Auswahl der Sprache moeglich. ueber den Login-Button wird die Verbindung zum Server hergestellt.

## 3.17.2 Dokumentation der Elementfunktionen

## 3.17.2.1 void addConnectButtonListener ( ActionListener a )

Fuegt einen Listener fuer den 'Connect' Button des [Login](#) Fensters hinzu.

## Parameter

<i>a</i>	ein ActionListener
----------	--------------------

## 3.17.2.2 void addLanguageSelectionListener ( ItemListener i )

Fuegt einen Listener fuer die Sprachauswahl des [Login](#) Fensters hinzu.

## Parameter

<i>i</i>	ein ItemListener
----------	------------------

## 3.17.2.3 void setLanguage ( Language l )

Aendert die Sprache des Fensters.

## Parameter

<i>l</i>	Sprache in Form des Language-Enums
----------	------------------------------------

## 3.17.2.4 void update ( Observable o, Object arg )

Wird durch notify() im [ClientModel](#) aufgerufen.

Je nach dem in arg uebergebenen ViewNotification-Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

## Parameter

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet: loginSuccessful

### 3.18 OtherPlayer Klassenreferenz

## Private Attribute

- Object **name**
- Object **info**

#### 3.18.1 Ausführliche Beschreibung

Zeigt die Informationen über die anderen Spieler an, also den Namen, ein Symbol für die verdeckte Hand und das Label für zusätzliche Angaben.

### 3.19 OwnHand Klassenreferenz

## Private Attribute

- Object **cards**
- Set< [ViewCard](#) > **card**

#### 3.19.1 Ausführliche Beschreibung

Stellt die Karten dar, die der Spieler auf der Hand hat. Der Spieler kann eine Karte durch Anklicken auswählen und durch einen zweiten Klick ausspielen.

### 3.20 Password Klassenreferenz

Abgeleitet von JFrame und Observer.

## Öffentliche Methoden

- [Password](#) ()
- void [addJoinButtonListener](#) (ActionListener a)
- void [setLanguage](#) ([Language](#) l)
- void [update](#) (Observable o, Object arg)

## Öffentliche, statische Methoden

- static void **main** (String[] args)

## Private Methoden

- void **updateLanguage** ()

## Private Attribute

- JPanel **contentPane**
- JTextField **textField**
- JButton **btnJoin**
- JLabel **lblEnterPasswordPlease**
- JButton **btnLeave**
- [Language](#) **lang**

## Statische, private Attribute

- static final long **serialVersionUID** = 799479782389332727L

## 3.20.1 Ausführliche Beschreibung

Dieses Fenster ermöglicht die Eingabe eines Passwortes um einem Passwortgeschuetztem Spiel beizutreten oder per 'Leave' wieder in die [Lobby](#) zurueckzukehren.

## 3.20.2 Dokumentation der Elementfunktionen

## 3.20.2.1 void addJoinButtonListener ( ActionListener a )

Fuegt einen ActionListener fuer den 'Join' Button hinzu.

## Parameter

<i>a</i>	ein ActionListener
----------	--------------------

## 3.20.2.2 void setLanguage ( Language l )

Aendert die Sprache des Fensters.

## Parameter

<i>l</i>	Sprache in Form des Language-Enums
----------	------------------------------------

## 3.20.2.3 void update ( Observable o, Object arg )

Wird durch notify() im [ClientModel](#) aufgerufen.

Je nach dem in arg uebergebenen ViewNotification-Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

## Parameter

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet: openWarning, passwordAccepted

## 3.21 ScoreWindow Klassenreferenz

Abgeleitet von Observer.

## Öffentliche Methoden

- void [update](#) (Observable o, Object arg)

### 3.21.1 Ausführliche Beschreibung

Dieses Fenster zeigt den momentanen Punktestand nach jeder Runde und den Gesamtpunktestand am Ende des Spieles an.

### 3.21.2 Dokumentation der Elementfunktionen

#### 3.21.2.1 void update ( Observable o, Object arg )

Wird durch notify() im [ClientModel](#) aufgerufen.

Je nach dem in arg uebergebenen Befehl wird ein Update des Fensters ausgefuehrt oder eine Fehlermeldung angezeigt.

Parameter

<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet: showScore

## 3.22 ViewCard Klassenreferenz

Abgeleitet von JPanel.

Öffentliche Methoden

- [ViewCard](#) (String s, int n)
- int [getID](#) ()
- void **paintComponent** (Graphics g)

Private Attribute

- String **path**
- int **id**
- BufferedImage **face**

Statische, private Attribute

- static final long **serialVersionUID** = 8733682958484899430L

### 3.22.1 Ausführliche Beschreibung

[ViewCard](#) ist die View-seitige Repraesentation einer Karte. Sie wird verwendet um einzelne Karten auf das Spielfeld zu zeichnen. Dazu enthaelt sie die Pfadangabe zu dem Ordner, in dem die Bilder der Karten gespeichert sind, und eine ID, um das genaue Bild zu spezifizieren.

### 3.22.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.22.2.1 ViewCard ( String s, int n )

Erstellt eine neue Karte fuer die Anzeige und zeichnet dafuer das Bild, das durch die Pfadangabe s und seine Kardinaliaet n im Ordner angegeben ist.

Die Pfadangabe wird durch das Regelwerk bestimmt.

## Parameter

<i>s</i>	Pfadangabe zum zu zeichnenden Bild
<i>n</i>	ID der Karte

## 3.22.3 Dokumentation der Elementfunktionen

3.22.3.1 `int getID ( )`

Gibt die ID der Karte zurueck.

## Rückgabe

ID der Karte

## 3.23 Warning Klassenreferenz

Abgeleitet von Observer.

## Öffentliche Methoden

- void `setText` (String text)
- void `update` (Observable o, Object arg)

## Private Attribute

- String **warningText**

## 3.23.1 Ausführliche Beschreibung

Das Warning-Fenster zeigt dem Benutzer Fehlermeldungen bzw. Hinweise an, welche vom `ClientModel` uebergeben wurden. Es wird nur im Fehlerfall angezeigt.

## 3.23.2 Dokumentation der Elementfunktionen

3.23.2.1 `void setText ( String text )`

Setzt den Warnhinweis des Fensters.

## Parameter

<i>text</i>	Warnhinweis, der angezeigt werden soll
-------------	--

3.23.2.2 `void update ( Observable o, Object arg )`

Wird durch `notify()` im `ClientModel` aufgerufen.

Je nach dem in arg übergebenen Befehl wird ein Update des Fensters ausgeführt oder eine Fehlermeldung angezeigt.

## Parameter



<i>o</i>	erwartet ein Objekt von der Klasse <a href="#">ClientModel</a>
<i>arg</i>	erwartet: openWarning

### 3.24 ViewNotification Enum-Referenz

#### Öffentliche Attribute

- **moveAcknowledged**
- **chooseCardsSuccessful**
- **inputNumberSuccessful**
- **chooseItemSuccessful**
- **playerListUpdate**
- **gameListUpdate**
- **chatMessage**
- **loginSuccessful**
- **joinGameSuccessful**
- **gameStarted**
- **passwordAccepted**
- **playedCardsUpdate**
- **otherDataUpdate**
- **windowChangeForced**
- **openChooseCards**
- **openChooseItem**
- **openInputNumber**
- **openWarning**
- **showScore**

#### 3.24.1 Ausführliche Beschreibung

Enum, das vom [ClientModel](#) ueber notify an seine Observer geschickt wird, um mitzuteilen, welche Veraenderung stattgefunden hat.

### 3.25 ComBeenKicked Klassenreferenz

Abgeleitet von [ComObject](#) und Serializable.

#### Öffentliche Methoden

- [ComBeenKicked](#) (String message)
- String [getMessage](#) ()
- void [process](#) ([ClientModel](#) model)
- void [process](#) ([Player](#) player, [Server](#) server)

#### Private Attribute

- String **message**

#### 3.25.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Die Nachricht wird an einen Spieler gesendet, wenn er aus einem Spiel entfernt wurde. Dies geschieht, wenn ein Spieler ein Spiel verlaesst oder wenn der Spielleiter das Wartefenster verlaesst.

## 3.25.2 Beschreibung der Konstruktoren und Destruktoren

## 3.25.2.1 ComBeenKicked ( String message )

Dies ist der Kontruktor fuer eine neue ComBeenKicked-Nachricht.

Parameter

<i>message</i>	ist die Nachricht.
----------------	--------------------

## 3.25.3 Dokumentation der Elementfunktionen

## 3.25.3.1 String getMessage ( )

Diese Methode liefert die Nachricht, die an den Spieler gesendet wird, wenn er entfernt wird.

Rückgabe

die Nachricht.

## 3.25.3.2 void process ( ClientModel model )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

Parameter

<i>model</i>	ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	--

Implementiert [ComObject](#).

## 3.25.3.3 void process ( Player player, Server server )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

Parameter

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

## 3.26 ComChatMessage Klassenreferenz

Abgeleitet von [ComObject](#) und Serializable.

Öffentliche Methoden

- [ComChatMessage](#) (String message)
- String [getChatMessage](#) ()
- void [process](#) (ClientModel model)
- void [process](#) (Player player, Server server)

Private Attribute

- String **chatMessage**

### 3.26.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie enthaelt eine Chatnachricht in Form eines Strings.

### 3.26.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.26.2.1 ComChatMessage ( String *message* )

Dies ist der Kontruktor fuer eine neue ComChatMessage-Nachricht.

Parameter

<i>message</i>	ist die Chatnachricht, die versendet wird.
----------------	--

### 3.26.3 Dokumentation der Elementfunktionen

#### 3.26.3.1 String getChatMessage ( )

Hier kann die versendete Nachricht von anderen Klassen ausgelesen werden.

Rückgabe

die Chatnachricht, die versendet wurde.

#### 3.26.3.2 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

Parameter

<i>model</i>	ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	--

Implementiert [ComObject](#).

#### 3.26.3.3 void process ( Player *player*, Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

Parameter

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

## 3.27 ComClientLeave Klassenreferenz

Abgeleitet von [ComObject](#) und Serializable.

Öffentliche Methoden

- [ComClientLeave](#) ()
- void [process](#) (ClientModel *model*)
- void [process](#) (Player *player*, Server *server*)

## 3.27.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie wird zur Benachrichtigung gesendet, wenn ein Spieler ins naechste Fenster moechte und aus dem alten entfernt werden soll.

## 3.27.2 Dokumentation der Elementfunktionen

## 3.27.2.1 void process ( ClientModel model )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

Parameter

<i>model</i>	ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	--

Implementiert [ComObject](#).

## 3.27.2.2 void process ( Player player, Server server )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

Parameter

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

## 3.28 ComClientQuit Klassenreferenz

Abgeleitet von [ComObject](#) und Serializable.

Öffentliche Methoden

- [ComClientQuit](#) ()
- void [process](#) ([ClientModel](#) model)
- void [process](#) ([Player](#) player, [Server](#) server)

## 3.28.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Die Nachricht wird verschickt, wenn der Spieler ein Fenster schließt.

## 3.28.2 Dokumentation der Elementfunktionen

## 3.28.2.1 void process ( ClientModel model )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>model</i>	ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	--

Implementiert [ComObject](#).

**3.28.2.2 void process ( Player player, Server server )**

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

**3.29 ComCreateGameRequest Klassenreferenz**

Abgeleitet von [ComObject](#) und Serializable.

**Öffentliche Methoden**

- [ComCreateGameRequest](#) (String name, Enum ruleset, boolean hasPassword, String password)
- String [getGameName](#) ()
- Enum [getRuleset](#) ()
- boolean [hasPassword](#) ()
- String [getPassword](#) ()
- void [process](#) (ClientModel model)
- void [process](#) (Player player, Server server)

**Private Attribute**

- String **gameName**
- Enum **ruleset**
- boolean **hasPassword**
- String **password**

**3.29.1 Ausführliche Beschreibung**

Diese Klasse ist ein spezielles Kommunikations-Objekt. Diese Nachricht wird versendet, wenn ein neues Spiel erstellt werden soll.

**3.29.2 Beschreibung der Konstruktoren und Destruktoren****3.29.2.1 ComCreateGameRequest ( String name, Enum ruleset, boolean hasPassword, String password )**

Dies ist der Kontruktor fuer eine neue ComCreateGameRequest-Nachricht.

Wurde kein Passwort gesetzt, bleibt dieses leer.

## Parameter

<i>name</i>	ist der Name des Spiels.
<i>ruleset</i>	ist die der Spieltyp, der erstellt werden soll.
<i>hasPassword</i>	sagt, ob ein Passwort gesetzt wurde.
<i>password</i>	ist das Passwort, das gesetzt wurde.

Benutzt ComCreateGameRequest.hasPassword().

## 3.29.3 Dokumentation der Elementfunktionen

## 3.29.3.1 String getName ( )

Diese Methode gibt den Namen des Spiels zurueck.

## Rückgabe

den Spielnamen.

## 3.29.3.2 Enum getRuleset ( )

Diese Methode gibt das Regelwerk zurueck, das benutzt werden soll.

## Rückgabe

das Regelwerk, welches benutzt wird.

## 3.29.3.3 boolean hasPassword ( )

Diese Methode gibt an, ob eine Passwort fuer ein Spiel gesetzt wurde.

## Rückgabe

ob es ein Passwort gibt.

Wird benutzt von ComCreateGameRequest.ComCreateGameRequest().

## 3.29.3.4 String getPassword ( )

Gibt das Passwort zurueck.

Sollte keines gesetzt sein, wird null zurueck gegeben.

## Rückgabe

das Passwort.

## 3.29.3.5 void process ( ClientModel model )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>model</i>	ist das ClientModel, welches bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	---

Implementiert [ComObject](#).

## 3.29.3.6 void process ( Player player, Server server )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObject weitergibt.

Implementiert [ComObject](#).

**3.30 ComGameEnd Klassenreferenz**

Abgeleitet von [ComObject](#) und [Serializable](#).

**Öffentliche Methoden**

- [ComGameEnd](#) (String winner, Map< String, Integer > points)
- String [getWinner](#) ()
- Map< String, Integer > [getPoints](#) ()
- void [process](#) ([ClientModel](#) model)
- void [process](#) ([Player](#) player, [Server](#) server)

**Private Attribute**

- String **winner**
- Map< String, Integer > **points**

**3.30.1 Ausführliche Beschreibung**

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie liefert den Gewinner eines Spiels und eine Auflistung der Spieler mit ihren erspielten Punkten und wird versendet, wenn ein Spiel zu Ende ist oder eine Runde. In diesem Fall wird der Gewinner einfach leer gelassen, da es noch keinen gibt.

**3.30.2 Beschreibung der Konstruktoren und Destruktoren****3.30.2.1 ComGameEnd ( String winner, Map< String, Integer > points )**

Dies ist der Kontruktor fuer eine neue ComGameEnd-Nachricht.

**Parameter**

<i>winner</i>	ist der Gewinner des Spiels.
<i>points</i>	ist eine Auflistung der Spieler mit ihren Punkten.

**3.30.3 Dokumentation der Elementfunktionen****3.30.3.1 String getWinner ( )**

Diese Methode gibt den Gewinner zurueck.

**Rückgabe**

den Gewinner.

## 3.30.3.2 Map&lt;String, Integer&gt; getPoints ( )

Diese Methode gibt die Auflistung der Spieler mit ihren zugehoerigen Punkten zurueck.

## Rückgabe

die Liste der Spieler mit ihren Punkten.

## 3.30.3.3 void process ( ClientModel model )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>model</i>	ist das ClientModel, welches ¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	---

Implementiert [ComObject](#).

## 3.30.3.4 void process ( Player player, Server server )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

## 3.31 ComInitGameLobby Klassenreferenz

Abgeleitet von [ComObject](#) und Serializable.

## Öffentliche Methoden

- [ComInitGameLobby](#) (List< String > playerList)
- List< String > [getPlayerList](#) ()
- void [process](#) (ClientModel model)
- void [process](#) (Player player, Server server)

## Private Attribute

- List< String > **playerList**

## 3.31.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie liefert die Liste der Spieler, die sich bereits beim Betreten des Wartefensters darin befinden.

## 3.31.2 Beschreibung der Konstruktoren und Destruktoren

## 3.31.2.1 ComInitGameLobby ( List&lt; String &gt; playerList )

Dies ist der Kontruktor fuer eine neue ComInitGameLobby-Nachricht.



**Parameter**

<i>playerList</i>	ist die Liste aller Player, die sich im Wartefenster befinden.
-------------------	--

**3.31.3 Dokumentation der Elementfunktionen****3.31.3.1 List<String> getPlayerList ( )**

Diese Methode liefert die Liste der Player, die sich beim Hinzufuegen eines weiteren Spielers bereits im Wartefenster befinden.

**Rückgabe**

die Liste der Spieler, die im Wartefenster sind.

**3.31.3.2 void process ( ClientModel model )**

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>model</i>	ist das ClientModel, welches bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	---

Implementiert [ComObject](#).

**3.31.3.3 void process ( Player player, Server server )**

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

**3.32 ComInitLobby Klassenreferenz**

Abgeleitet von [ComObject](#) und Serializable.

**Öffentliche Methoden**

- [ComInitLobby](#) (List< String > playerList, Set gameList)
- List< String > [getPlayerList](#) ()
- Set< [GameServerRepresentation](#) > [getGameList](#) ()
- void [process](#) (ClientModel model)
- void [process](#) (Player player, Server server)

**Private Attribute**

- List< String > **playerList**
- Set< [GameServerRepresentation](#) > **gameList**

## 3.32.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie synchronisiert den Client mit der Lobby, wenn er sich mit dem Server verbindet oder nach einem Spiel in die Lobby zurueckkehrt. Dazu enthaelt sie sowohl die *playerList*, als auch die *gameList*.

## 3.32.2 Beschreibung der Konstruktoren und Destruktoren

3.32.2.1 ComInitLobby ( List< String > *playerList*, Set *gameList* )

Dies ist der Kontruktor fuer eine neue ComInitLobby-Nachricht.

## Parameter

<i>playerList</i>	ist die Liste der Spieler, die sich in der Lobby befinden.
<i>gameList</i>	ist die Liste der Spiele, die existieren und in der Lobby angezeigt werden.

## 3.32.3 Dokumentation der Elementfunktionen

## 3.32.3.1 List&lt;String&gt; getPlayerList ( )

Die Methode liefert die Liste aller Spieler, die in der Lobby sind.

## Rückgabe

die Liste der Spieler.

## 3.32.3.2 Set&lt;GameServerRepresentation&gt; getGameList ( )

Diese Methode liefert eine Liste aller Spiele, die erstellt wurden, damit sie in der Lobby angezeigt werden koennen.

## Rückgabe

die Liste der Spiele.

3.32.3.3 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>model</i>	ist das ClientModel, welches bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	---

Implementiert [ComObject](#).

3.32.3.4 void process ( Player *player*, Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>player</i>	Der Client welcher den Aufruf startet.
---------------	--

<i>server</i>	Der Server an den sich das ComObjekt weitergibt.
---------------	--

Implementiert [ComObject](#).

### 3.33 ComJoinRequest Klassenreferenz

Abgeleitet von [ComObject](#) und [Serializable](#).

#### Öffentliche Methoden

- [ComJoinRequest](#) (String [gameMasterName](#), String password)
- String [getGameMasterName](#) ()
- void [process](#) ([ClientModel](#) model)
- void [process](#) ([Player](#) player, [Server](#) server)

#### Private Attribute

- String [gameMasterName](#)
- String **password**

#### 3.33.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie ist eine Nachricht, die an den Server gesendet wird, wenn der Spieler einem bestimmten Spiel beitreten will. Dazu enthaelt sie den Namen des Spielleiters als String und ein Passwort, falls dieses von Spielleiter gesetzt wurde.

#### 3.33.2 Beschreibung der Konstruktoren und Destruktoren

##### 3.33.2.1 ComJoinRequest ( String *gameMasterName*, String *password* )

Dies ist der Kontruktor fuer eine neue ConJoinRequest-Nachricht.

Ein Spiel kann durch den eindeutigen Namen der Spielleiters identifiziert werden.

#### Parameter

<i>gameMaster-Name</i>	ist der Name der Spielleiters.
<i>password</i>	fuer das Spiel.

Benutzt [ComJoinRequest.gameMasterName](#).

#### 3.33.3 Dokumentation der Elementfunktionen

##### 3.33.3.1 String [getGameMasterName](#) ( )

Diese Methode gibt den Namen des Spielleiters zurueck.

Dieser ist eindeutig, so kann ein bestimmtes Spiel identifiziert werden.

#### Rückgabe

der Name des Spielleiters.

Benutzt [ComJoinRequest.gameMasterName](#).

#### 3.33.3.2 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>model</i>	ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	--

Implementiert [ComObject](#).

### 3.33.3.3 void process ( **Player** *player*, **Server** *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

### 3.33.4 Dokumentation der Datenelemente

#### 3.33.4.1 String gameMasterName [private]

Der Name der Spielleiters muss enthalten sein um ein Spiel zuzuornen.

Der Spielname ist nicht eindeutig, aber der Spielleiter schon. Somit kann jedes Spiel mit Hilfe des Spielleiters identifiziert werden.

Wird benutzt von ComJoinRequest.ComJoinRequest() und ComJoinRequest.getGameMasterName().

## 3.34 ComKickPlayerRequest Klassenreferenz

Abgeleitet von [ComObject](#) und Serializable.

**Öffentliche Methoden**

- [ComKickPlayerRequest](#) (String [playerName](#))
- String [getPlayerName](#) ()
- void [process](#) (ClientModel [model](#))
- void [process](#) (Player [player](#), Server [server](#))

**Private Attribute**

- String [playerName](#)

### 3.34.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie ist eine Nachricht an den Server, die angibt einen Spieler vom Spiel zu entfernen. Dazu enthaelt sie einen String, der den Namen des Spielers enthaelt.

### 3.34.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.34.2.1 ComKickPlayerRequest ( String *playerName* )

Dies ist der Kontruktor fuer eine neue ComKickPlayerRequest-Nachricht.

Diese enthaelt den Namen des Spielers, der aus den Spiel geloescht werden soll.

## Parameter

<i>playerName</i>	ist der Name des Spielers.
-------------------	----------------------------

Benutzt ComKickPlayerRequest.playerName.

## 3.34.3 Dokumentation der Elementfunktionen

## 3.34.3.1 String getPlayerName ( )

Diese Methode liefert den Namen des Spielers, der aus dem Spiel entfernt werden soll.

## Rückgabe

den Spielernamen.

Benutzt ComKickPlayerRequest.playerName.

## 3.34.3.2 void process ( ClientModel model )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>model</i>	ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	--

Implementiert [ComObject](#).

## 3.34.3.3 void process ( Player player, Server server )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

## 3.35 ComLobbyUpdateGamelist Klassenreferenz

Abgeleitet von [ComObject](#) und Serializable.

## Öffentliche Methoden

- [ComLobbyUpdateGamelist](#) (boolean removeFlag, [GameServerRepresentation](#) gameServer)
- boolean [isRemoveFlag](#) ()
- [GameServerRepresentation](#) [getGameServer](#) ()
- void [process](#) ([ClientModel](#) model)
- void [process](#) ([Player](#) player, [Server](#) server)

## Private Attribute

- boolean **removeFlag**
- [GameServerRepresentation](#) **gameServer**

### 3.35.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie aktualisiert die Gameliste in der Lobby. Dazu enthaelt sie den GameServer und ein RemoveFlag.

### 3.35.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.35.2.1 ComLobbyUpdateGamelist ( boolean *removeFlag*, GameServerRepresentation *gameServer* )

Dies ist der Kontruktor fuer eine neue ComLobbyUpdateGamelist-Nachricht.

Parameter

<i>removeFlag</i>	zeigt an, ob das Spiel geloescht werden soll.
<i>gameServer</i>	ist das Spiel.

### 3.35.3 Dokumentation der Elementfunktionen

#### 3.35.3.1 boolean isRemoveFlag ( )

Diese Methode liefert, ob ein Spiel geloescht werden soll oder nicht.

Rückgabe

ob das Spiel gelöscht wird.

#### 3.35.3.2 GameServerRepresentation getGameServer ( )

Diese Methode liefert das Spiel, das geupdated werden soll.

Rückgabe

das Spiel.

#### 3.35.3.3 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

Parameter

<i>model</i>	ist das ClientModel, welches übergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	---

Implementiert [ComObject](#).

#### 3.35.3.4 void process ( Player *player*, Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

Parameter

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

## 3.36 ComLoginRequest Klassenreferenz

Abgeleitet von [ComObject](#) und Serializable.

## Öffentliche Methoden

- [ComLoginRequest](#) (String name)
- String [getPlayerName](#) ()
- void [process](#) ([ClientModel](#) model)
- void [process](#) ([Player](#) player, [Server](#) server)

## Private Attribute

- String **playerName**

## 3.36.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie ist eine Nachricht, die beim Login an den Server gesendet wird. Dazu enthaelt sie den Namen des Spielers, der sich einloggen moechte.

## 3.36.2 Beschreibung der Konstruktoren und Destruktoren

## 3.36.2.1 ComLoginRequest ( String name )

Dies ist der Kontruktor fuer eine neue ComLoginRequest-Nachricht.

## Parameter

<i>name</i>	ist der Name des Spielers, des sich einloggen moechte.
-------------	--

## 3.36.3 Dokumentation der Elementfunktionen

## 3.36.3.1 String getPlayerName ( )

Diese Methode liefert den Namen des Spielers, des sich einloggen moechte.

Dieser muss auf Eindeutigkeit geprueft werden.

## Rückgabe

den Spielernamen.

## 3.36.3.2 void process ( ClientModel model )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>model</i>	ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	--

Implementiert [ComObject](#).

## 3.36.3.3 void process ( Player player, Server server )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.



**Parameter**

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObject weitergibt.

Implementiert [ComObject](#).

**3.37 ComObject Schnittstellenreferenz**

Basisklasse für [ComBeenKicked](#), [ComChatMessage](#), [ComClientLeave](#), [ComClientQuit](#), [ComCreateGameRequest](#), [ComGameEnd](#), [ComInitGameLobby](#), [ComInitLobby](#), [ComJoinRequest](#), [ComKickPlayerRequest](#), [ComLobbyUpdateGamelist](#), [ComLoginRequest](#), [ComRuleset](#), [ComServerAcknowledgement](#), [ComStartGame](#), [ComUpdatePlayerlist](#) und [ComWarning](#).

**Öffentliche Methoden**

- void [process](#) ([ClientModel](#) model)
- void [process](#) ([Player](#) player, [Server](#) server)

**3.37.1 Ausführliche Beschreibung**

Die Klasse [ComObject](#) ist ein Interface, welches ein Objekt darstellt, das zur Kommunikation genutzt werden kann. Spezielle [ComObject](#) Klassen implementieren diese grundlegenden Klasse.

**3.37.2 Dokumentation der Elementfunktionen****3.37.2.1 void process ( ClientModel model )**

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>model</i>	ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	--

Implementiert in [ComCreateGameRequest](#), [ComInitLobby](#), [ComGameEnd](#), [ComJoinRequest](#), [ComUpdatePlayerlist](#), [ComLobbyUpdateGamelist](#), [ComKickPlayerRequest](#), [ComInitGameLobby](#), [ComBeenKicked](#), [ComRuleset](#), [ComWarning](#), [ComLoginRequest](#), [ComChatMessage](#), [ComClientLeave](#), [ComClientQuit](#), [ComStartGame](#) und [ComServerAcknowledgement](#).

**3.37.2.2 void process ( Player player, Server server )**

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObject weitergibt.

Implementiert in [ComCreateGameRequest](#), [ComInitLobby](#), [ComGameEnd](#), [ComJoinRequest](#), [ComUpdatePlayerlist](#), [ComLobbyUpdateGamelist](#), [ComKickPlayerRequest](#), [ComInitGameLobby](#), [ComBeenKicked](#), [ComRuleset](#), [ComWarning](#), [ComLoginRequest](#), [ComChatMessage](#), [ComClientLeave](#), [ComClientQuit](#), [ComStartGame](#) und [ComServerAcknowledgement](#).

### 3.38 ComRuleset Klassenreferenz

Abgeleitet von [ComObject](#) und [Serializable](#).

#### Öffentliche Methoden

- [ComRuleset](#) ([RulesetMessage](#) rulesetMessage)
- [RulesetMessage](#) getRulesetMessage ()
- void [process](#) ([ClientModel](#) model)
- void [process](#) ([Player](#) player, [Server](#) server)

#### Private Attribute

- [RulesetMessage](#) rulesetMessage

#### 3.38.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie ist die grundlegende Nachricht eines Regelwerkaufwurfes und enthaelt eine verfeinerte Nachricht mit weiteren Informationen, die [RulesetMessage](#).

#### 3.38.2 Beschreibung der Konstruktoren und Destruktoren

##### 3.38.2.1 ComRuleset ( RulesetMessage rulesetMessage )

Dies ist der Kontruktor fuer eine neue ComRuleset-Nachricht.

#### Parameter

<i>rulesetMessage</i>	ist eine Nachricht, die ans Ruleset gesendet werden soll.
-----------------------	---

#### 3.38.3 Dokumentation der Elementfunktionen

##### 3.38.3.1 RulesetMessage getRulesetMessage ( )

Diese Methode gibt die Nachricht zurueck, die ans Ruleset gesendet werden soll.

#### Rückgabe

die Nachricht.

##### 3.38.3.2 void process ( ClientModel model )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

#### Parameter

<i>model</i>	ist das ClientModel, welches ¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	---

Implementiert [ComObject](#).

##### 3.38.3.3 void process ( Player player, Server server )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

**3.39 ComServerAcknowledgement Klassenreferenz**

Abgeleitet von [ComObject](#) und Serializable.

**Öffentliche Methoden**

- void [process](#) ([ClientModel](#) model)
- void [process](#) ([Player](#) player, [Server](#) server)

**3.39.1 Ausführliche Beschreibung**

Diese Klasse ist ein spezielles Kommunikations-Objekt. Diese Nachricht wird vom Server als Bestätigung gesendet.

**3.39.2 Dokumentation der Elementfunktionen****3.39.2.1 void process ( [ClientModel](#) model )**

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>model</i>	ist das ClientModel, welches übergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	---

Implementiert [ComObject](#).

**3.39.2.2 void process ( [Player](#) player, [Server](#) server )**

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

**3.40 ComStartGame Klassenreferenz**

Abgeleitet von [ComObject](#) und Serializable.

**Öffentliche Methoden**

- [ComStartGame](#) ()
- void [process](#) ([ClientModel](#) model)
- void [process](#) ([Player](#) player, [Server](#) server)

### 3.40.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie wird versendet, wenn ein Spiel gestartet werden soll.

### 3.40.2 Dokumentation der Elementfunktionen

#### 3.40.2.1 void process ( ClientModel model )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

Parameter

<i>model</i>	ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	--

Implementiert [ComObject](#).

#### 3.40.2.2 void process ( Player player, Server server )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

Parameter

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObject weitergibt.

Implementiert [ComObject](#).

## 3.41 ComUpdatePlayerlist Klassenreferenz

Abgeleitet von [ComObject](#) und Serializable.

Öffentliche Methoden

- [ComUpdatePlayerlist](#) (String playerName, boolean removeFlag)
- String [getPlayerName](#) ()
- boolean [isRemoveFlag](#) ()
- void [process](#) (ClientModel model)
- void [process](#) (Player player, [Server](#) server)

Private Attribute

- String **playerName**
- boolean **removeFlag**

### 3.41.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie sendet eine Nachricht zum Update der Playerliste in der Lobby und Spiellooby. Dazu enthaelt sie den Player und ein removeFlag.

### 3.41.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.41.2.1 ComUpdatePlayerlist ( String playerName, boolean removeFlag )

Dies ist der Kontruktor fuer eine neue ComUpdatePlayerlist-Nachricht.

Diese beinhaltet den Namen des Spielers und die Angabe ob er gelöscht werden soll.

## Parameter

<i>playerName</i>	ist der Name der Spielers.
<i>removeFlag</i>	zeigt, ob der Spieler geloescht werden soll.

## 3.41.3 Dokumentation der Elementfunktionen

## 3.41.3.1 String getPlayerName ( )

Diese Methode gibt den Namen des Spielers zurueck.

## Rückgabe

den Spielernamen.

## 3.41.3.2 boolean isRemoveFlag ( )

Diese Methode gibt zurueck, ob der Spieler aus der Liste geloescht werden soll oder nicht.

## Rückgabe

ob der Spieler geloescht werden soll.

## 3.41.3.3 void process ( ClientModel model )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>model</i>	ist das ClientModel, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	--

Implementiert [ComObject](#).

## 3.41.3.4 void process ( Player player, Server server )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

## 3.42 ComWarning Klassenreferenz

Abgeleitet von [ComObject](#) und Serializable.

## Öffentliche Methoden

- [ComWarning](#) (String warning)
- String [getWarning](#) ()
- void [process](#) (ClientModel model)
- void [process](#) (Player player, Server server)

## Private Attribute

- String **warning**

### 3.42.1 Ausführliche Beschreibung

Diese Klasse ist ein spezielles Kommunikations-Objekt. Sie soll dem Spieler eine Mitteilung senden und so ueber ein Fehlerevent informieren.

### 3.42.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.42.2.1 ComWarning ( String *warning* )

Dies ist der Konstruktor einer neuen ComWarning-Nachricht.

Er enthaelt eine Warnung an den Spieler, wenn ein Fehler passiert.

#### Parameter

<i>warning</i>	ist die Warnung, die der Spieler erhaelt.
----------------	---

### 3.42.3 Dokumentation der Elementfunktionen

#### 3.42.3.1 String getWarning ( )

Diese Methode gibt die Nachricht zurueck, die dem Spieler den Fehler mitteilt.

#### Rückgabe

die Warnnachricht.

#### 3.42.3.2 void process ( ClientModel *model* )

Diese Methode ist noetig, damit der ClientListenerThread entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

#### Parameter

<i>model</i>	ist das ClientModel, welches ¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
--------------	---

Implementiert [ComObject](#).

#### 3.42.3.3 void process ( Player *player*, Server *server* )

Diese Methode ist noetig, damit der Thread Player entscheiden kann welche Message das Object enthaelt und wie diese verarbeitet werden soll.

#### Parameter

<i>player</i>	Der Client welcher den Aufruf startet.
<i>server</i>	Der Server an den sich das ComObjekt weitergibt.

Implementiert [ComObject](#).

## 3.43 MsgCard Klassenreferenz

Abgeleitet von [RulesetMessage](#) und Serializable.

## Öffentliche Methoden

- [MsgCard](#) ([Card](#) card)
- [Card](#) [getCard](#) ()
- void [visit](#) ([ServerRuleset](#) serverRuleset, String name)
- void [visit](#) ([ClientRuleset](#) clientRuleset)

## Private Attribute

- [Card](#) **card**

## 3.43.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie beinhaltet die ausgespielte Karte eines Spielers.

## 3.43.2 Beschreibung der Konstruktoren und Destruktoren

3.43.2.1 [MsgCard](#) ( [Card](#) card )

Dies ist der Kontruktor fuer eine neue MsgCard-Nachricht.

Diese enthaelt die Information, welche Karte von einem Spieler gespielt wurde.

## Parameter

<i>card</i>	ist die Karte.
-------------	----------------

## 3.43.3 Dokumentation der Elementfunktionen

3.43.3.1 [Card](#) [getCard](#) ( )

Diese Methode gibt die ausgespielte Karte des Spielers zurueck.

## Rückgabe

die Karte.

3.43.3.2 void [visit](#) ( [ServerRuleset](#) serverRuleset, String name )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>serverRuleset</i>	ist das Ruleset, welches bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
<i>name</i>	ist der Name des Spielers.

Implementiert [RulesetMessage](#).

3.43.3.3 void [visit](#) ( [ClientRuleset](#) clientRuleset )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.



**Parameter**

<i>clientRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	--

Implementiert [RulesetMessage](#).

**3.44 MsgCardRequest Klassenreferenz**

Abgeleitet von [RulesetMessage](#) und Serializable.

**Öffentliche Methoden**

- [MsgCardRequest](#) ()
- void [visit](#) ([ServerRuleset](#) serverRuleset, String name)
- void [visit](#) ([ClientRuleset](#) clientRuleset)

**3.44.1 Ausführliche Beschreibung**

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Diese Nachricht wird von Server gesendet, um einem Spieler mitzuteilen, dass er das Spielen einer Karte erwartet.

**3.44.2 Dokumentation der Elementfunktionen****3.44.2.1 void visit ( [ServerRuleset](#) serverRuleset, String name )**

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>serverRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
<i>name</i>	ist der Name des Spielers.

Implementiert [RulesetMessage](#).

**3.44.2.2 void visit ( [ClientRuleset](#) clientRuleset )**

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>clientRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	--

Implementiert [RulesetMessage](#).

**3.45 MsgGameEnd Klassenreferenz**

Abgeleitet von [RulesetMessage](#) und Serializable.

**Öffentliche Methoden**

- [MsgGameEnd](#) (String name)
- String [getWinnerName](#) ()

- void [visit](#) ([ServerRuleset](#) serverRuleset, String name)
- void [visit](#) ([ClientRuleset](#) clientRuleset)

#### Private Attribute

- String **winnerName**

#### 3.45.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie signalisiert dem ClientRuleset, dass das Spiel zu Ende ist.

#### 3.45.2 Beschreibung der Konstruktoren und Destruktoren

##### 3.45.2.1 MsgGameEnd ( String name )

Dies ist der Kontruktor fuer eine neue MsgGameEnd-Nachricht.

#### Parameter

<i>name</i>	ist der Name des Gewinners.
-------------	-----------------------------

#### 3.45.3 Dokumentation der Elementfunktionen

##### 3.45.3.1 String getWinnerName ( )

Diese Methode liefert den Namen des Gewinners eines Spiels.

#### Rückgabe

den Gewinnernamen.

##### 3.45.3.2 void visit ( [ServerRuleset](#) serverRuleset, String name )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

#### Parameter

<i>serverRuleset</i>	ist das Ruleset, welches bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
<i>name</i>	ist der Name des Spielers.

Implementiert [RulesetMessage](#).

##### 3.45.3.3 void visit ( [ClientRuleset](#) clientRuleset )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

#### Parameter

<i>clientRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	--

Implementiert [RulesetMessage](#).

### 3.46 MsgMultiCards Klassenreferenz

Abgeleitet von [RulesetMessage](#) und Serializable.

#### Öffentliche Methoden

- [MsgMultiCards](#) (Set cardList)
- Set< [Card](#) > [getCardList](#) ()
- void [visit](#) ([ServerRuleset](#) serverRuleset, String name)
- void [visit](#) ([ClientRuleset](#) clientRuleset)

#### Private Attribute

- Set< [Card](#) > **cardList**

#### 3.46.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie liefert mehrere Karten zum Tausch fuer das Regelwerk Hearts.

#### 3.46.2 Beschreibung der Konstruktoren und Destruktoren

##### 3.46.2.1 MsgMultiCards ( Set *cardList* )

Dies ist der Kontruktor fuer eine neue MsgMultiCards-Nachricht.

#### Parameter

<i>cardList</i>	ist die Liste der ausgewaehlten Karten.
-----------------	---

#### 3.46.3 Dokumentation der Elementfunktionen

##### 3.46.3.1 Set<Card> getCardList ( )

Gibt die Liste der gewaehlten Karten zurÃ¼ck.

#### Rückgabe

die Liste der Karten.

##### 3.46.3.2 void visit ( [ServerRuleset](#) *serverRuleset*, String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

#### Parameter

<i>serverRuleset</i>	ist das Ruleset, welches Ã¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
<i>name</i>	ist der Name des Spielers.

Implementiert [RulesetMessage](#).

##### 3.46.3.3 void visit ( [ClientRuleset](#) *clientRuleset* )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>clientRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	--

Implementiert [RulesetMessage](#).

### 3.47 MsgMultiCardsRequest Klassenreferenz

Abgeleitet von [RulesetMessage](#) und Serializable.

## Öffentliche Methoden

- [MsgMultiCardsRequest](#) (int *count*)
- int [getCount](#) ()
- void [visit](#) ([ServerRuleset](#) serverRuleset, String name)
- void [visit](#) ([ClientRuleset](#) clientRuleset)

## Private Attribute

- int *count*

#### 3.47.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Diese Nachricht wird gesendet, wenn die Auswahl mehrerer Karten vom Spieler gefordert werden soll.

#### 3.47.2 Beschreibung der Konstruktoren und Destruktoren

##### 3.47.2.1 MsgMultiCardsRequest ( int *count* )

Dies ist der Kontruktor fuer eine neue MsgMultipleCardsRequest-Nachricht.

## Parameter

<i>count</i>	ist die erwartete Anzahl an Karten.
--------------	-------------------------------------

Benutzt [MsgMultiCardsRequest.count](#).

#### 3.47.3 Dokumentation der Elementfunktionen

##### 3.47.3.1 int [getCount](#) ( )

Diese Methode gibt die Anzahl der Karten zurueck, die der Server vom Spieler erwartet.

## Rückgabe

die Anzahl der Karten.

Benutzt [MsgMultiCardsRequest.count](#).

##### 3.47.3.2 void [visit](#) ( [ServerRuleset](#) *serverRuleset*, String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>serverRuleset</i>	ist das Ruleset, welches bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
<i>name</i>	ist der Name des Spielers.

Implementiert [RulesetMessage](#).

**3.47.3.3 void visit ( ClientRuleset clientRuleset )**

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>clientRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	--

Implementiert [RulesetMessage](#).

**3.48 MsgNumber Klassenreferenz**

Abgeleitet von [RulesetMessage](#) und Serializable.

**Öffentliche Methoden**

- [MsgNumber](#) (int number)
- int [getNumber](#) ()
- void [visit](#) ([ServerRuleset](#) serverRuleset, String name)
- void [visit](#) ([ClientRuleset](#) clientRuleset)

**Private Attribute**

- int **number**

**3.48.1 Ausführliche Beschreibung**

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie enthaelt eine Zahl, die ein Spieler zuvor ausgewaehlt hat.

**3.48.2 Beschreibung der Konstruktoren und Destruktoren****3.48.2.1 MsgNumber ( int number )**

Dies ist der Kontruktor fuer eine neue MsgNumber-Nachricht.

**Parameter**

<i>number</i>	ist eine Eingabe eines Spielers.
---------------	----------------------------------

**3.48.3 Dokumentation der Elementfunktionen****3.48.3.1 int getNumber ( )**

Diese Methode liefert die Eingabe eines Spielers.

**Rückgabe**

eine Zahl, die Eingabe des Spielers.

**3.48.3.2 void visit ( ServerRuleset serverRuleset, String name )**

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>serverRuleset</i>	ist das Ruleset, welches bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
<i>name</i>	ist der Name des Spielers.

Implementiert [RulesetMessage](#).

**3.48.3.3 void visit ( ClientRuleset clientRuleset )**

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>clientRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	--

Implementiert [RulesetMessage](#).

**3.49 MsgNumberRequest Klassenreferenz**

Abgeleitet von [RulesetMessage](#) und Serializable.

**Öffentliche Methoden**

- [MsgNumberRequest](#) ()
- void [visit](#) ([ServerRuleset](#) serverRuleset, String name)
- void [visit](#) ([ClientRuleset](#) clientRuleset)

**3.49.1 Ausführliche Beschreibung**

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie Wird gesendet, wenn die Eingabe einer Zahl gefordert werden soll.

**3.49.2 Dokumentation der Elementfunktionen****3.49.2.1 void visit ( ServerRuleset serverRuleset, String name )**

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>serverRuleset</i>	ist das Ruleset, welches bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	---

<i>name</i>	ist der Name des Spielers.
-------------	----------------------------

Implementiert [RulesetMessage](#).

#### 3.49.2.2 void visit ( ClientRuleset clientRuleset )

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

Parameter

<i>clientRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	--

Implementiert [RulesetMessage](#).

### 3.50 MsgSelection Klassenreferenz

Abgeleitet von [RulesetMessage](#) und Serializable.

Öffentliche Methoden

- [MsgSelection](#) (Colour selection)
- [Colour](#) getSelection ()
- void visit (ServerRuleset serverRuleset, String name)
- void visit (ClientRuleset clientRuleset)

Private Attribute

- [Colour](#) selection

#### 3.50.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Diese Nachricht enthaelt eine Kartenfarbe, die der Spieler zuvor ausgewaehlt hat.

#### 3.50.2 Beschreibung der Konstruktoren und Destruktoren

##### 3.50.2.1 MsgSelection ( Colour selection )

Dies ist der Kontruktor fuer eine neue MsgSelection-Nachricht.

Parameter

<i>selection</i>	ist die Farbe der Karte, die der Spieler gewaehlt hat.
------------------	--

#### 3.50.3 Dokumentation der Elementfunktionen

##### 3.50.3.1 Colour getSelection ( )

Diese Methode gibt die Farbe zurueck, die der Spieler gewaehlt hat.

Rückgabe

die gewaehlte Farbe.

#### 3.50.3.2 void visit ( *ServerRuleset serverRuleset*, *String name* )

Diese Methode ist noetig, damit das *ServerRuleset* entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.



**Parameter**

<i>serverRuleset</i>	ist das Ruleset, welches ¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
<i>name</i>	ist der Name des Spielers.

Implementiert [RulesetMessage](#).

**3.50.3.3 void visit ( ClientRuleset clientRuleset )**

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>clientRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	--

Implementiert [RulesetMessage](#).

**3.51 MsgSelectionRequest Klassenreferenz**

Abgeleitet von [RulesetMessage](#) und Serializable.

**Öffentliche Methoden**

- [MsgSelectionRequest](#) ()
- void [visit](#) ([ServerRuleset](#) serverRuleset, String name)
- void [visit](#) ([ClientRuleset](#) clientRuleset)

**3.51.1 Ausführliche Beschreibung**

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Diese Nachricht sendet der Server an einen Spieler, wenn er eine Farbauswahl von diesem erwartet.

**3.51.2 Dokumentation der Elementfunktionen****3.51.2.1 void visit ( ServerRuleset serverRuleset, String name )**

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>serverRuleset</i>	ist das Ruleset, welches ¼bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
<i>name</i>	ist der Name des Spielers.

Implementiert [RulesetMessage](#).

**3.51.2.2 void visit ( ClientRuleset clientRuleset )**

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>clientRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	--

Implementiert [RulesetMessage](#).

### 3.52 MsgUser Klassenreferenz

Abgeleitet von [RulesetMessage](#) und Serializable.

## Öffentliche Methoden

- [MsgUser](#) ([GameClientUpdate](#) gameClientUpdate)
- [GameClientUpdate](#) [getGameClientUpdate](#) ()
- void [visit](#) ([ServerRuleset](#) serverRuleset, String name)
- void [visit](#) ([ClientRuleset](#) clientRuleset)

## Private Attribute

- [GameClientUpdate](#) **gameClientUpdate**

#### 3.52.1 Ausführliche Beschreibung

Diese Klasse ist eine Verfeinerung der RulesetMessage-Klasse. Sie wird dem Client gesendet, um dem Client-Ruleset den aktuellen Spielzustand in Form eines GameClientUpdate zu uebermitteln.

#### 3.52.2 Beschreibung der Konstruktoren und Destruktoren

##### 3.52.2.1 [MsgUser](#) ( [GameClientUpdate](#) *gameClientUpdate* )

Dies ist der Konstruktor einer neuen MsgUser-Nachricht.

## Parameter

<i>gameClient-Update</i>	ist der aktuelle Spielstand.
--------------------------	------------------------------

#### 3.52.3 Dokumentation der Elementfunktionen

##### 3.52.3.1 [GameClientUpdate](#) [getGameClientUpdate](#) ( )

Diese Methode liefert den den aktuellen Spielzustand, der fuer ein Update benoetigt wird.

## Rückgabe

den aktuellen Spielzustand.

##### 3.52.3.2 void [visit](#) ( [ServerRuleset](#) *serverRuleset*, String *name* )

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>serverRuleset</i>	ist das Ruleset, welches bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
<i>name</i>	ist der Name des Spielers.

Implementiert [RulesetMessage](#).

**3.52.3.3 void visit ( ClientRuleset clientRuleset )**

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>clientRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	--

Implementiert [RulesetMessage](#).

**3.53 RulesetMessage Schnittstellenreferenz**

Basisklasse für [MsgCard](#), [MsgCardRequest](#), [MsgGameEnd](#), [MsgMultiCards](#), [MsgMultiCardsRequest](#), [MsgNumber](#), [MsgNumberRequest](#), [MsgSelection](#), [MsgSelectionRequest](#) und [MsgUser](#).

**Öffentliche Methoden**

- void [visit](#) ([ServerRuleset](#) serverRuleset, String name)
- void [visit](#) ([ClientRuleset](#) clientRuleset)

**3.53.1 Ausführliche Beschreibung**

Dieses Interface ist eine Verfeinerung der ComRuleset-Klasse. Es enthält Methoden, die von speziellen Ruleset-Messages implementiert werden müssen.

**3.53.2 Dokumentation der Elementfunktionen****3.53.2.1 void visit ( ServerRuleset serverRuleset, String name )**

Diese Methode ist noetig, damit das ServerRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

**Parameter**

<i>serverRuleset</i>	ist das Ruleset, welches bergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
<i>name</i>	ist der Name des Spielers.

Implementiert in [MsgMultiCardsRequest](#), [MsgSelection](#), [MsgCard](#), [MsgUser](#), [MsgGameEnd](#), [MsgMultiCards](#), [MsgNumber](#), [MsgCardRequest](#), [MsgNumberRequest](#) und [MsgSelectionRequest](#).

**3.53.2.2 void visit ( ClientRuleset clientRuleset )**

Diese Methode ist noetig, damit das ClientRuleset entscheiden kann welche Message es enthaelt und wie diese verarbeitet werden soll.

## Parameter

<i>clientRuleset</i>	ist das Ruleset, welches uebergeben wird, damit die ueberladene Methode richtig gewaehlt wird.
----------------------	--

Implementiert in [MsgMultiCardsRequest](#), [MsgCard](#), [MsgSelection](#), [MsgUser](#), [MsgGameEnd](#), [MsgMultiCards](#), [MsgNumber](#), [MsgNumberRequest](#), [MsgSelectionRequest](#) und [MsgCardRequest](#).

### 3.54 Card Schnittstellenreferenz

Basisklasse für [HeartsCard](#) und [WizardCard](#).

## Öffentliche Methoden

- [int](#) [getValue](#) ()
- [Colour](#) [getColour](#) ()

#### 3.54.1 Ausführliche Beschreibung

Dieses Interface modelliert eine Spielkarte

#### 3.54.2 Dokumentation der Elementfunktionen

##### 3.54.2.1 [int](#) [getValue](#) ( )

Gibt den Wert der Karte zurück.

## Rückgabe

Der Wert der Karte

Implementiert in [WizardCard](#) und [HeartsCard](#).

##### 3.54.2.2 [Colour](#) [getColour](#) ( )

Gibt die Farbe der Karte zurück.

## Rückgabe

Die Farbe der Karte

Implementiert in [WizardCard](#) und [HeartsCard](#).

### 3.55 ClientHearts Klassenreferenz

Abgeleitet von [ClientRuleset](#).

## Öffentliche Methoden

- [ClientHearts](#) ([ClientModel](#) client)
- [boolean](#) [isValidMove](#) ([Card](#) card)
- [void](#) [resolveMessage](#) ([MsgMultiCardsRequest](#) msgMultiCardsRequest)
- [boolean](#) [areValidChooosenCards](#) ([Set](#)< [Card](#) > cards)

## Statische, private Attribute

- static final int `MIN_PLAYERS` = 4
- static final int `MAX_PLAYERS` = 4
- static final `RulesetType RULESET` = `RulesetType.Hearts`

## Weitere Geerbte Elemente

### 3.55.1 Ausführliche Beschreibung

Diese Klasse bildet das Regelwerk für den Clientmodel bei einer Partie Hearts

### 3.55.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.55.2.1 `ClientHearts ( ClientModel client )`

Erzeugt ein `ClientHearts`.

##### Parameter

<i>client</i>	Das Model auf dem gespielt wird
---------------	---------------------------------

Benutzt `RulesetType.Hearts`, `ClientHearts.MAX_PLAYERS` und `ClientHearts.MIN_PLAYERS`.

### 3.55.3 Dokumentation der Elementfunktionen

#### 3.55.3.1 `boolean isValidMove ( Card card ) [virtual]`

Prueft ob ein gemachter Zug in einem Spiel gueltig war.

##### Parameter

<i>card</i>	Die Karte
-------------	-----------

##### Rückgabe

true falls die Karte gueltig ist, false wenn nicht

Implementiert `ClientRuleset`.

#### 3.55.3.2 `void resolveMessage ( MsgMultiCardsRequest msgMultiCardsRequest )`

Verarbeitet die `RulesetMessage` dass der Server von dem Spieler verlangt mehrere Karten anzugeben.

##### Parameter

<i>msgMultiCards-Request</i>	Die Nachricht vom Server
------------------------------	--------------------------

#### 3.55.3.3 `boolean areValidChosenCards ( Set< Card > cards )`

Gibt zuueck ob die Karten die der Client tauschen will, gueltig sind.

##### Parameter

<i>cards</i>	Die zu tauschenden Karten
--------------	---------------------------

#### Rückgabe

true wenn Karten valide sind, false wenn nicht

### 3.56 ClientRuleset Klassenreferenz

Basisklasse für [ClientHearts](#) und [ClientWizard](#).

#### Öffentliche Methoden

- [ClientRuleset](#) ([RulesetType](#) ruleset, int minPlayers, int maxPlayers, [ClientModel](#) client)
- [RulesetType](#) getRulesetType ()
- int getMinPlayers ()
- int getMaxPlayers ()
- [GamePhase](#) getGamePhase ()
- List< [Card](#) > getOwnHand ()
- [OtherData](#) getOwnData ()
- [OtherData](#) getOtherPlayerData (String player)
- [PlayerState](#) getCurrentPlayer ()
- [Card](#) getTrumpCard ()
- void resolveMessage ([RulesetMessage](#) message)
- void resolveMessage ([MsgUser](#) clientUpdate)
- void resolveMessage ([MsgCardRequest](#) msgCardRequest)
- abstract boolean isValidMove ([Card](#) card)

#### Geschützte Methoden

- void send ([RulesetMessage](#) message)

#### Private Attribute

- [ClientModel](#) client
- [GameClientUpdate](#) gameState
- final [RulesetType](#) RULESET
- final int MIN\_PLAYERS
- final int MAX\_PLAYERS
- [GamePhase](#) gamePhase

#### 3.56.1 Ausführliche Beschreibung

[ClientRuleset](#) ist eine abstrakte Klasse und wird zur Regelvorauswertung im Client verwendet. Dazu benutzt es die [isValidMove\(\)](#) Methode. Des Weiteren kann es vom [ClientModel](#) erhaltene [RulesetMessages](#) mit der [resolveMessage\(\)](#) Methode behandeln und neue [RulesetMessages](#) senden.

#### 3.56.2 Beschreibung der Konstruktoren und Destruktoren

##### 3.56.2.1 [ClientRuleset](#) ( [RulesetType](#) ruleset, int minPlayers, int maxPlayers, [ClientModel](#) client )

Erstellt eine [ClientRuleset](#) Klasse.

**Parameter**

<i>ruleset</i>	Das Ruleset zum Spiel
<i>minPlayers</i>	Die minimale Spieleranzahl
<i>maxPlayers</i>	Die maximale Spieleranzahl
<i>client</i>	Das ClientModel auf dem gespielt wird

Benutzt ClientRuleset.client, ClientRuleset.gamePhase, ClientRuleset.MAX\_PLAYERS, ClientRuleset.MIN\_PLAYERS, ClientRuleset.RULESET und GamePhase.Start.

**3.56.3 Dokumentation der Elementfunktionen****3.56.3.1 RulesetType getRulesetType ( )**

Gibt den Typ des Regelwerks zurueck.

**Rückgabe**

Der Typ vom Regelwerk

Benutzt ClientRuleset.RULESET.

**3.56.3.2 int getMinPlayers ( )**

Gibt die Mindestanzahl an Spielern zurueck für dieses Spiel.

**Rückgabe**

Die Mindestanzahl an Spielern

Benutzt ClientRuleset.MIN\_PLAYERS.

**3.56.3.3 int getMaxPlayers ( )**

Gibt die Maximale Anzahl an Spielern zurueck.

**Rückgabe**

Die maximale Anzahl an Spielern

Benutzt ClientRuleset.MAX\_PLAYERS.

**3.56.3.4 GamePhase getGamePhase ( )**

Gibt die momentane Spielphase zurück.

**Rückgabe**

gamePhase Die Spielphase

Benutzt ClientRuleset.gamePhase.

**3.56.3.5 List<Card> getOwnHand ( )**

Gibt die eigenen Handkarten zurueck.

**Rückgabe**

Liste von Karten

**3.56.3.6 OtherData getOwnData ( )**

Gibt die [OtherData](#) des Models zurueck.

**Rückgabe**

Die Otherdata des Models

**3.56.3.7 OtherData getOtherPlayerData ( String *player* )**

Holt die [OtherData](#) eines anderen Spielers.

**Parameter**

<i>Der</i>	Spielername
------------	-------------

**Rückgabe**

otherPlayerData Die [OtherData](#)

**3.56.3.8 PlayerState getCurrentPlayer ( )**

Gibt den Spieler der momentan am Zug ist zurueck.

**Rückgabe**

Der momentane Spieler

**3.56.3.9 Card getTrumpCard ( )**

Holt die aufgedeckte Trumpfkarte.

**Rückgabe**

Eine Karte

**3.56.3.10 void resolveMessage ( RulesetMessage *message* )**

Verarbeitet eine RulesetMessage vom Server.

**Parameter**

<i>clientUpdate</i>	Die Nachricht vom Server
---------------------	--------------------------

**3.56.3.11 void resolveMessage ( MsgUser *clientUpdate* )**

Verarbeitet die RulesetMessage dass der Server ein Spielupdate an den Client schickt.

**Parameter**

<i>clientUpdate</i>	Die Nachricht vom Server
---------------------	--------------------------

**3.56.3.12 void resolveMessage ( MsgCardRequest *msgCardRequest* )**

Verarbeitet die RulesetMessage dass der Server von dem Spieler verlangt eine Karte zu spielen.



## Parameter

<i>msgCard-Request</i>	Die Nachricht vom Server
------------------------	--------------------------

## 3.56.3.13 void send ( RulesetMessage message ) [protected]

Ruft beim Model die send Methode auf und verschickt eine Rulesetmessage.

## Parameter

<i>message</i>	Die Nachricht
----------------	---------------

## 3.56.3.14 abstract boolean isValidMove ( Card card ) [pure virtual]

Prueft ob ein gemachter Zug in einem Spiel gueltig war.

## Parameter

<i>card</i>	Die Karte
-------------	-----------

## Rückgabe

true falls die Karte gueltig ist, false wenn nicht

Implementiert in [ClientHearts](#) und [ClientWizard](#).

## 3.57 ClientWizard Klassenreferenz

Abgeleitet von [ClientRuleset](#).

## Öffentliche Methoden

- [ClientWizard](#) ([ClientModel](#) client)
- boolean [isValidMove](#) ([Card](#) card)
- void [resolveMessage](#) ([MsgNumberRequest](#) msgNumber)
- void [resolveMessage](#) ([MsgSelectionRequest](#) msgSelection)
- boolean [isValidTrickNumber](#) (int number)
- boolean [isValidColour](#) ([Colour](#) colour)

## Statische, private Attribute

- static final int [MIN\\_PLAYERS](#) = 3
- static final int [MAX\\_PLAYERS](#) = 6
- static final [RulesetType](#) [RULESET](#) = [RulesetType.Wizard](#)

## Weitere Geerbte Elemente

## 3.57.1 Ausführliche Beschreibung

Diese Klasse bildet das Regelwerk fuer den Client bei einer Partie Wizard

### 3.57.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.57.2.1 ClientWizard ( ClientModel *client* )

Erzeugt ein [ClientWizard](#).

**Parameter**

<i>client</i>	Das Model auf dem gespielt wird
---------------	---------------------------------

Benutzt ClientWizard.MAX\_PLAYERS, ClientWizard.MIN\_PLAYERS und ClientWizard.RULESET.

**3.57.3 Dokumentation der Elementfunktionen****3.57.3.1 boolean isValidMove ( Card card ) [virtual]**

Prueft ob ein gemachter Zug in einem Spiel gueltig war.

**Parameter**

<i>card</i>	Die Karte
-------------	-----------

**Rückgabe**

true falls die Karte gueltig ist, false wenn nicht

Implementiert [ClientRuleset](#).

**3.57.3.2 void resolveMessage ( MsgNumberRequest msgNumber )**

Verarbeitet die RulesetMessage dass der Server von dem Spieler verlangt eine Stichanzahl anzugeben.

**Parameter**

<i>msgNumber</i>	Die Nachricht vom Server
------------------	--------------------------

**3.57.3.3 void resolveMessage ( MsgSelectionRequest msgSelection )**

Verarbeitet die RulesetMessage dass der Server von dem Spieler verlangt eine Farbe auszuwählen.

**Parameter**

<i>msgSelection</i>	Die Nachricht vom Server
---------------------	--------------------------

**3.57.3.4 boolean isValidTrickNumber ( int number )**

Prüft ob die Anzahl der angesagten Stiche vom Spieler gültig sind.

**Parameter**

<i>number</i>	Die Anzahl der angesagten Sticht
---------------	----------------------------------

**Rückgabe**

true falls die Anzahl der Stiche passen, false wenn nicht

**3.57.3.5 boolean isValidColour ( Colour colour )**

Prüft ob die angesagte Trumpffarbe richtig.

**Parameter**

<i>colour</i>	Die angesagte Trumpffarbe
---------------	---------------------------

**Rückgabe**

true falls die Farbe in Ordnung ist, false wenn nicht

## 3.58 Colour Enum-Referenz

### Öffentliche Attribute

- **NONE**
- **HEART**
- **CLUB**
- **SPADE**
- **DIAMOND**
- **BLUE**
- **RED**
- **YELLOW**

### 3.58.1 Ausführliche Beschreibung

Repräsentiert die Farbe einer Karte

## 3.59 GameClientUpdate Klassenreferenz

### Geschützte Methoden

- [GameClientUpdate](#) ([PlayerState](#) *playerState*, [Map](#)< [String](#), [Card](#) > *discardPile*, [Map](#)< [String](#), [OtherData](#) > *otherPlayerData*, [PlayerState](#) *currentPlayer*, [Card](#) *trumpCard*)
- [List](#)< [Card](#) > [getOwnHand](#) ()
- [Map](#)< [String](#), [Card](#) > [getPlayedCards](#) ()
- [OtherData](#) [getOwnData](#) ()
- [OtherData](#) [getOtherPlayerData](#) ([String](#) *player*)
- [PlayerState](#) [getCurrentPlayer](#) ()
- [Card](#) [getTrumpCard](#) ()

### Private Attribute

- [PlayerState](#) *playerState*
- [Map](#)< [String](#), [Card](#) > *discardPile*
- [Map](#)< [String](#), [OtherData](#) > *otherPlayerData*
- [PlayerState](#) *currentPlayer*
- [Card](#) *trumpCard*

### 3.59.1 Ausführliche Beschreibung

Das [GameClientUpdate](#) wird vom [RuleSet](#) ueber den [GameServer](#) an den Client geschickt und enthaelt alle Aenderungen des [GameState](#), die für den Client relevant sind. Das waeren seine Spielhand, der Ablagestapel sowie die Otherdata von allen Spielern und die Trumpfkarte.

### 3.59.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.59.2.1 [GameClientUpdate](#) ( [PlayerState](#) *playerState*, [Map](#)< [String](#), [Card](#) > *discardPile*, [Map](#)< [String](#), [OtherData](#) > *otherPlayerData*, [PlayerState](#) *currentPlayer*, [Card](#) *trumpCard* ) [protected]

Erstellt ein [GameClientUpdate](#).

**Parameter**

<i>playerState</i>	Der Spielerzustand des Client
<i>discardPile</i>	Der Ablagestapel
<i>otherPlayerData</i>	Die Daten der anderen Spieler
<i>currentPlayer</i>	Der momentan aktive Spieler
<i>trumpCard</i>	Die Trumpffarbe

Benutzt `GameClientUpdate.currentPlayer`, `GameClientUpdate.discardPile`, `GameClientUpdate.otherPlayerData`, `GameClientUpdate.playerState` und `GameClientUpdate.trumpCard`.

**3.59.3 Dokumentation der Elementfunktionen****3.59.3.1 `List<Card> getOwnHand ( )` [protected]**

Holt die Karten die der Client auf der Hand hat.

**Rückgabe**

`ownHand` Die Hand des Clients

**3.59.3.2 `Map<String, Card> getPlayedCards ( )` [protected]**

Holt die gespielten Karten auf dem Ablagestapel.

**Rückgabe**

`discardPile` Die gespielten Karten

Benutzt `GameClientUpdate.discardPile`.

**3.59.3.3 `OtherData getOwnData ( )` [protected]**

Holt die Otherdata des Client als String als Stringrepräsentation.

**Rückgabe**

`ownData` Die Otherdata des Clients

**3.59.3.4 `OtherData getOtherPlayerData ( String player )` [protected]**

Holt die [OtherData](#) eines anderen Spielers als Stringrepräsentation.

**Parameter**

<i>player</i>	Der Name des Spielers
---------------	-----------------------

**Rückgabe**

`otherPlayerData` Die [OtherData](#) der anderen Spieler

**3.59.3.5 `PlayerState getCurrentPlayer ( )` [protected]**

Gibt den Spieler der momentan am Zug ist zurück.

**Rückgabe**

Der momentane Spieler

Benutzt `GameClientUpdate.currentPlayer`.

### 3.59.3.6 Card getTrumpCard ( ) [protected]

Holt die aufgedeckte Trumpfkarte.

#### Rückgabe

trumpCard Die Trumpfkarte

Benutzt GameClientUpdate.trumpCard.

## 3.60 GamePhase Enum-Referenz

### Öffentliche Attribute

- [Start](#)
- [Playing](#)
- [CardRequest](#)
- [MultipleCardRequest](#)
- [TrickRequest](#)
- [SelectionRequest](#)
- [RoundEnd](#)
- [Ending](#)

### 3.60.1 Ausführliche Beschreibung

Die [GamePhase](#) modelliert die verschiedenen Zustände des Spiels im [GameState](#)

## 3.61 GameState Klassenreferenz

### Geschützte Methoden

- [GameState](#) (RulesetType ruleset, List< [Card](#) > deck)
- boolean [addPlayerToGame](#) (String name)
- void [setFirstPlayer](#) (PlayerState player)
- PlayerState [getFirstPlayer](#) ()
- boolean [setCurrentPlayer](#) (PlayerState player)
- PlayerState [getCurrentPlayer](#) ()
- List< [Card](#) > [getCardsLeftInDeck](#) ()
- Map< String, [Card](#) > [getPlayedCards](#) ()
- PlayerState [getPlayerState](#) (String name)
- void [setTrumpCard](#) (Card trumpCard)
- Card [getTrumpCard](#) ()
- int [getRoundNumber](#) ()
- int [getNumberOfPlayedCards](#) ()
- List< [Card](#) > [getPlayerCards](#) (PlayerState player)
- void [shuffleDeck](#) ()
- boolean [dealCards](#) (int number)
- void [nextPlayer](#) ()
- boolean [giveACard](#) (PlayerState player, Card card)
- boolean [playCard](#) (Card card)

## Private Attribute

- List< [PlayerState](#) > [players](#)
- [RulesetType](#) [ruleset](#)
- [PlayerState](#) [firstPlayer](#)
- [PlayerState](#) [currentPlayer](#)
- int [roundNumber](#)
- Map< String, [Card](#) > [discardPile](#)
- List< [Card](#) > [deck](#)
- [Card](#) [trumpCard](#)

### 3.61.1 Ausführliche Beschreibung

Das [GameState](#) modelliert einen aktuellen Spielzustand, es wird vom [GameServer](#) instanziiert und vom [RuleSet](#) bearbeitet. Es enthält die einzelnen [PlayerStates](#), sowie Informationen zum Ablage-, Aufnahmestapel, Rundenanzahl, den momentan aktiven Spieler sowie [GamePhase](#).

### 3.61.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.61.2.1 [GameState](#) ( [RulesetType](#) *ruleset*, List< [Card](#) > *deck* ) [protected]

Erstellt eine [GameState](#)klasse und [trumpCard](#) wird als [WizardCard.Empty](#) instanziiert.

#### Parameter

<i>ruleset</i>	Der Regelwerktyp des Spiels
<i>deck</i>	Das Kartendeck im Spiel

Benutzt [GameState.deck](#), [GameState.discardPile](#), [GameState.players](#), [GameState.ruleset](#) und [GameState.trumpCard](#).

### 3.61.3 Dokumentation der Elementfunktionen

#### 3.61.3.1 boolean [addPlayerToGame](#) ( String *name* ) [protected]

Fügt den Spieler ins Spiel hinein, falls er nicht schon im Spiel ist.

#### Parameter

<i>name</i>	Der Name eines Spielers
-------------	-------------------------

#### Rückgabe

true falls der Spieler noch nicht im Spiel ist, und false sonst

Benutzt [GameState.players](#) und [GameState.ruleset](#).

#### 3.61.3.2 void [setFirstPlayer](#) ( [PlayerState](#) *player* ) [protected]

Setzt einen neuen Spieler als [firstPlayer](#).

#### Parameter

<i>player</i>	Der neue <a href="#">firstPlayer</a>
---------------	--------------------------------------

Benutzt [GameState.firstPlayer](#).

**3.61.3.3 PlayerState getFirstPlayer ( ) [protected]**

Holt den Spieler der als erster am Zug war.

**Rückgabe**

firstPlayer Der Spielzustand des Spielers der als erster am Zug war

Benutzt GameState.firstPlayer.

**3.61.3.4 boolean setCurrentPlayer ( PlayerState player ) [protected]**

Setzt einen neuen Spieler als currentPlayer.

**Parameter**

<i>player</i>	Der neue currentPlayer
---------------	------------------------

Benutzt GameState.currentPlayer.

**3.61.3.5 PlayerState getCurrentPlayer ( ) [protected]**

Holt den Spieler der momentan am Zug ist.

**Rückgabe**

currentPlayer Der Spielzustand des Spielers der grad am Zug ist

Benutzt GameState.currentPlayer.

**3.61.3.6 List<Card> getCardsLeftInDeck ( ) [protected]**

Holt die Karten die noch im Aufnahmestapel sind.

**Rückgabe**

deck Holt die Karten die noch im Aufnahmestapel sind

**3.61.3.7 Map<String,Card> getPlayedCards ( ) [protected]**

Holt die gespielten Karten im Ablagestapel.

**Rückgabe**

discardPile Die gespielten Karten

Benutzt GameState.discardPile.

**3.61.3.8 PlayerState getPlayerState ( String name ) [protected]**

Holt einen bestimmten Spieler.

**Parameter**

<i>name</i>	Der Name des Spielers
-------------	-----------------------

**Rückgabe**

player Der Spielzustand des Spielers

Benutzt GameState.players.

**3.61.3.9 void setTrumpCard ( Card trumpCard ) [protected]**

Setzt die Trumpfkarte.



**Parameter**

<i>trumpCard</i>	Die Trumpfkarte
------------------	-----------------

Benutzt GameState.trumpCard.

**3.61.3.10 Card getTrumpCard ( ) [protected]**

Holt die momentane Trumpfkarte im Spiel.

**Rückgabe**

trumpCard Die momentane Trumpfkarte

Benutzt GameState.trumpCard.

**3.61.3.11 int getRoundNumber ( ) [protected]**

Holt die Anzahl an Runden.

**Rückgabe**

Die Anzahl der Runden

Benutzt GameState.roundNumber.

**3.61.3.12 int getNumberOfPlayedCards ( ) [protected]**

Holt die Anzahl der gespielten Karten.

**Rückgabe**

Die Anzahl der gespielten Karten

**3.61.3.13 List<Card> getPlayerCards ( PlayerState player ) [protected]**

Holt die Karten eines Spielers.

**Parameter**

<i>name</i>	Der Name vom Spieler
-------------	----------------------

**Rückgabe**

Die Karten eines Spielers zurueck, wenn der Spieler nicht gefunden wird, wird eine leere Liste zurückgegeben

Benutzt GameState.players.

**3.61.3.14 boolean dealCards ( int number ) [protected]**

Verteilt eine bestimmte Anzahl an Karten an die Spieler.

**Parameter**

<i>number</i>	Die Anzahl an Karten
---------------	----------------------

**Rückgabe**

True falls ein Spieler keine Karten hat, false sonst

Benutzt GameState.deck und GameState.players.

**3.61.3.15 boolean giveACard ( PlayerState player, Card card ) [protected]**

Gibt eine bestimmte Karte einem Spieler.

## Parameter

<i>name</i>	Der Name des Spielers
<i>card</i>	Die Karte

## Rückgabe

true falls die Karte im Stapel ist, false wenn nicht

## 3.61.3.16 boolean playCard ( Card card ) [protected]

Entfernt eine Karte aus der Hand des currentPlayer und legt sie auf dem Ablagestapel.

## Parameter

<i>card</i>	Die gespielte Karte
-------------	---------------------

## Rückgabe

isInHand Gibt true zurück wenn die gespielte Karte auf der Hand vom Spieler liegt und false sonst

## 3.62 HeartsCard Enum-Referenz

Abgeleitet von [Card](#).

## Öffentliche Methoden

- int [getValue](#) ()
- [Colour](#) [getColour](#) ()

## Öffentliche Attribute

- **Empty** =(0,Colour.NONE)
- **Herz2** =(0,Colour.HEART)
- **Caro3** =(3,Colour.DIAMOND)

## Private Methoden

- [HeartsCard](#) (int [value](#), [Colour](#) [colour](#))

## Private Attribute

- final int [value](#)
- final [Colour](#) [colour](#)

## 3.62.1 Ausführliche Beschreibung

Modelliert eine Heartskarte.

## 3.62.2 Beschreibung der Konstruktoren und Destruktoren

## 3.62.2.1 HeartsCard ( int value, Colour colour ) [private]

Erzeugt eine Heartskarte mit einem Wert und einer Farbe.

**Parameter**

<i>value</i>	Der Wert der Karte
<i>colour</i>	Die Farbe der Karte

**3.62.3 Dokumentation der Elementfunktionen****3.62.3.1 int getValue ( )**

Gibt den Wert der Karte zurück.

**Rückgabe**

Der Wert der Karte

Implementiert [Card](#).

**3.62.3.2 Colour getColour ( )**

Gibt die Farbe der Karte zurück.

**Rückgabe**

Die Farbe der Karte

Implementiert [Card](#).

**3.63 HeartsData Klassenreferenz**

Abgeleitet von [OtherData](#).

**Öffentliche Methoden**

- String [toString](#) ()

**Geschützte Methoden**

- [HeartsData](#) ()

**3.63.1 Ausführliche Beschreibung**

Die Otherdata eines Spielers zum Spiel Hearts

**3.64 OtherData Klassenreferenz**

Basisklasse für [HeartsData](#) und [WizData](#).

**Öffentliche Methoden**

- abstract String [toString](#) ()

## Geschützte Methoden

- `OtherData ()`
- `void madeTrick (Set< Card > tricks)`
- `List< Card > removeTricks ()`
- `int getNumberOfTricks ()`
- `void setPoints (int points)`
- `int getPoints ()`

## Private Attribute

- `Set< Card > madeTricks`
- `int points`

## 3.64.1 Ausführliche Beschreibung

`OtherData` speichert alle Spielinformationen eines Spielers, außer seiner eigenen Spielhand.

## 3.64.2 Dokumentation der Elementfunktionen

3.64.2.1 `void madeTrick ( Set< Card > tricks ) [protected]`

Gibt dem Spieler die Stichkarten die er gemacht.

## Parameter

<i>tricks</i>	Die Stiche
---------------	------------

3.64.2.2 `List<Card> removeTricks ( ) [protected]`

Entfernt die gemachten Stichkarten eines Spielers und fuegt sie wieder in den Kartenstapel.

## Rückgabe

Die Kartenstiche

Benutzt `OtherData.madeTricks`.

3.64.2.3 `int getNumberOfTricks ( ) [protected]`

Gibt die Anzahl der gemachten Stiche des Spielers zurueck.

## Rückgabe

Die Anzahl der gemachten Stiche

3.64.2.4 `void setPoints ( int points ) [protected]`

Setzt den Punktestand eines Spielers.

## Parameter

<i>points</i>	Der neue Punktestand
---------------	----------------------

Benutzt `OtherData.points`.

### 3.64.2.5 `int getPoints ( ) [protected]`

Gibt den Punktestand eines Spielers zurück.

#### Rückgabe

Der Punktestand

Benutzt `OtherData.points`.

## 3.65 `PlayerState` Klassenreferenz

### Öffentliche Methoden

- `PlayerState` (`String name`, `RulesetType ruleset`)

### Geschützte Methoden

- `String getName ( )`
- `List< Card > getHand ( )`
- `OtherData getOtherData ( )`
- `void addCard (Card card)`
- `boolean removeCard (Card card)`

### Private Attribute

- `String name`
- `List< Card > ownHand`
- `OtherData otherData`

### 3.65.1 Ausführliche Beschreibung

Repraesentiert den Spielzustand eines Spielers, und wird unter anderem im `GameState` gespeichert. Sie enthält den Namen des Spielers, seine Handkarten und `OtherData`.

### 3.65.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.65.2.1 `PlayerState ( String name, RulesetType ruleset )`

Erstellt einen `PlayerState`.

#### Parameter

<i>name</i>	Der Name des Spielers
<i>ruleset</i>	Der Typ des Spiels

Benutzt `RulesetType.Hearts`, `PlayerState.name` und `PlayerState.otherData`.

### 3.65.3 Dokumentation der Elementfunktionen

#### 3.65.3.1 `String getName ( ) [protected]`

Holt den Namen eines Spielers.

**Rückgabe**

name Der Name des Spielers

**3.65.3.2 List<Card> getHand ( ) [protected]**

Holt die Kartenhand des Spielers.

**Rückgabe**

ownHand Die Kartenhand des Spielers

**3.65.3.3 OtherData getOtherData ( ) [protected]**

Gibt die [OtherData](#) des Spielers zurück.

**Rückgabe**

otherData Die [OtherData](#) eines Spielers

Benutzt PlayerState.otherData.

Wird benutzt von ServerRuleset.getPoints().

**3.65.3.4 void addCard ( Card card ) [protected]**

Gibt dem Spieler eine Karte.

**Parameter**

<i>card</i>	Die Karte die dem Spieler gegeben wird
-------------	--

**3.65.3.5 boolean removeCard ( Card card ) [protected]**

Entfernt eine Karte aus der Hand des Spielers.

**Parameter**

<i>card</i>	
-------------	--

**Rückgabe**

Gibt true zurueck wenn die Karte in der Hand ist und false sonst

**3.66 RulesetType Enum-Referenz****Öffentliche Attribute**

- [Wizard](#)
- [Hearts](#)

**3.66.1 Ausführliche Beschreibung**

Den Regelwerk Typ den ein Spiel hat.

**3.67 ServerHearts Klassenreferenz**

Abgeleitet von [ServerRuleset](#).

### Öffentliche Methoden

- [ServerHearts](#) ([GameServer](#) s)
- void [resolveMessage](#) ([MsgMultiCards](#) msgMultiCard, String name)

### Geschützte Methoden

- boolean [isValidMove](#) ([Card](#) card)
- int [getEndingPoints](#) ()
- void [calculateRoundOutcome](#) ()
- void [calculateTricks](#) ()
- String [getWinner](#) ()
- [GameClientUpdate](#) [generateGameClientUpdate](#) (String player)

### Private Methoden

- boolean [areValidChosenCards](#) (Set< [Card](#) > cards, String name)

### Statische, private Attribute

- static final int [MIN\\_PLAYERS](#) = 4
- static final int [MAX\\_PLAYERS](#) = 4
- static final [RulesetType](#) [RULESET](#) = [RulesetType](#).Hearts
- static final int [ENDING\\_POINTS](#) = 100

#### 3.67.1 Ausführliche Beschreibung

Diese Klasse erstellt das Regelwerk zum Spiel Hearts. Sie enthaelt zudem weitere Methoden, welche für das Spiel Hearts spezifisch benoetigt werden, wie die Regelung zum Tausch von Karten und die Berechnung der Stichpunkten.

#### 3.67.2 Dokumentation der Elementfunktionen

##### 3.67.2.1 boolean [isValidMove](#) ( [Card](#) card ) [protected],[virtual]

Prueft ob ein gemachter Zug vom currentPlayer in einem Spiel gueltig war, wenn nicht wird an den Spieler erneut eine [MsgCardRequest](#) gesendet.

##### Parameter

<i>card</i>	Die Karte die gespielt wurde
-------------	------------------------------

##### Rückgabe

true falls Zug gueltig und false wenn nicht

Implementiert [ServerRuleset](#).

##### 3.67.2.2 int [getEndingPoints](#) ( ) [protected]

Holt die Anzahl der Punkte die ein Spieler haben kann ab der, das Spiel vorbei ist.

##### Rückgabe

Anzahl der maximalen Punkte

Benutzt [ServerHearts](#).[ENDING\\_POINTS](#).

3.67.2.3 void resolveMessage ( [MsgMultiCards](#) msgMultiCard, String name )

Verarbeitet die RulesetMessage dass mehrerer Karten vom Spieler uebergeben werden.

Die wird dann in areValidChosenCards ueberpueft, bei falsche Eingabe wird eine MsgMultiCardsRequest an den selben Spieler gesendet. Bei richtiger Eingabe geht das Spiel weiter.

Parameter

<i>msgMultiCard</i>	Die Nachricht vom Client
<i>name</i>	Der Name des Spielers

3.67.2.4 boolean areValidChosenCards ( Set< [Card](#) > cards, String name ) [private]

Ueberpueft ob eine uebergebenes Kartenset von einem Spieler gultig ist.

Parameter

<i>cards</i>	Ein Kartenset
--------------	---------------

Rückgabe

true falls das Kartenset gueltig ist, false wenn nicht

3.67.2.5 [GameClientUpdate](#) generateGameClientUpdate ( String player ) [protected],[virtual]

Erzeugt ein [GameClientUpdate](#) welches individuell für jeden Benutzer ist.

Parameter

<i>player</i>	Dem Spieler
---------------	-------------

Implementiert [ServerRuleset](#).

## 3.68 ServerRuleset Klassenreferenz

Basisklasse für [ServerHearts](#) und [ServerWizard](#).

Öffentliche Methoden

- [ServerRuleset](#) ([RulesetType](#) ruleset, int min, int max, [GameServer](#) server)
- [RulesetType](#) getRulesetType ()
- int getMinPlayers ()
- int getMaxPlayers ()
- [GamePhase](#) getGamePhase ()
- void runGame ()
- void resolveMessage ([RulesetMessage](#) message, String name)
- void resolveMessage ([MsgCard](#) msgCard, String name)

Geschützte Methoden

- int getRoundNumber ()
- void setGamePhase ([GamePhase](#) phase)
- void setFirstPlayer ([PlayerState](#) player)
- [PlayerState](#) getFirstPlayer ()
- boolean nextPlayer ()
- boolean setCurrentPlayer ([PlayerState](#) player)
- [OtherData](#) getOtherData ([PlayerState](#) player)



- [PlayerState](#) [getCurrentPlayer](#) ()
- void [addPlayerToGame](#) (String name)
- [PlayerState](#) [getPlayerState](#) (String name)
- List< [Card](#) > [getPlayerCards](#) ([PlayerState](#) player)
- void [send](#) ([RulesetMessage](#) message, String name)
- void [broadcast](#) ([RulesetMessage](#) message)
- void [setPoints](#) ([PlayerState](#) player, int i)
- int [getPoints](#) ([PlayerState](#) player)
- boolean [dealCards](#) (int number)
- boolean [giveACard](#) ([PlayerState](#) player, [Card](#) card)
- boolean [playCard](#) ([Card](#) card)
- void [setTrumpCard](#) ([Card](#) card)
- abstract boolean [isValidMove](#) ([Card](#) card)
- abstract void [calculateTricks](#) ()
- abstract void [calculateRoundOutcome](#) ()
- abstract String [getWinner](#) ()
- abstract [GameClientUpdate](#) [generateGameClientUpdate](#) (String player)

#### Private Methoden

- List< [Card](#) > [createDeck](#) ()

#### Private Attribute

- [GameServer](#) server
- [GameState](#) gameState
- [GamePhase](#) gamePhase
- final [RulesetType](#) RULESET
- final int MIN\_PLAYERS
- final int MAX\_PLAYERS

#### 3.68.1 Ausführliche Beschreibung

Das [ServerRuleset](#) ist eine abstrakte Klasse und fuer den Ablauf und die Einhaltung der Regeln eines Spiels zustaeendig (/L280/). Das [ServerRuleset](#) wird im [GameServer](#) instanziiert und verwaltet die Zustaeende des [GameStates](#) im Server. Mit der Methode [isValidMove\(\)](#) wird eine Eingabe eines Clients auf Regelkonformität ueberprueft und dann das [GameState](#) veraendert. Ueber [resolveMessage\(\)](#) kann eine [GameServer](#)instanz eine [RulesetMessage](#) vom Player an das Ruleset weiterleiten.

#### 3.68.2 Beschreibung der Konstruktoren und Destruktoren

##### 3.68.2.1 [ServerRuleset](#) ( [RulesetType](#) ruleset, int min, int max, [GameServer](#) server )

Erstellt ein [ServerRuleset](#) und erzeugt ein [GameState](#).

#### Parameter

<i>ruleset</i>	Der Rulesettyp vom Server
<i>min</i>	Die minimale Anzahl an Spielern
<i>max</i>	Die maximale Anzahl an Spielern

<i>server</i>	Der Server auf dem gespielt wird
---------------	----------------------------------

Benutzt ServerRuleset.createDeck(), ServerRuleset.gamePhase, ServerRuleset.gameState, ServerRuleset.MAX\_PLAYERS, ServerRuleset.MIN\_PLAYERS, ServerRuleset.RULESET, ServerRuleset.server und GamePhase.Start.

### 3.68.3 Dokumentation der Elementfunktionen

#### 3.68.3.1 `int getRoundNumber ( )` [protected]

Holt die aktuelle Rundenanzahl zurueck.

##### Rückgabe

Die aktuelle Rundenanzahl

#### 3.68.3.2 `RulesetType getRulesetType ( )`

Gibt den Typ des Regelwerks zurueck.

##### Rückgabe

Der Typ vom Regelwerk

Benutzt ServerRuleset.RULESET.

#### 3.68.3.3 `int getMinPlayers ( )`

Gibt die Mindestanzahl an Spielern zurueck für dieses Spiel.

##### Rückgabe

Die Mindestanzahl an Spielern

Benutzt ServerRuleset.MIN\_PLAYERS.

#### 3.68.3.4 `int getMaxPlayers ( )`

Gibt die Maximale Anzahl an Spielern zurueck.

##### Rückgabe

Die maximale Anzahl an Spielern

Benutzt ServerRuleset.MAX\_PLAYERS.

#### 3.68.3.5 `void setGamePhase ( GamePhase phase )` [protected]

Aendert den momentanen Spielphase.

##### Parameter

<i>phase</i>	Die neue Spielphase
--------------	---------------------

#### 3.68.3.6 `GamePhase getGamePhase ( )`

Gibt den momentanen Spielzustand zurueck.

##### Rückgabe

Gibt die momentan Spielphase zurueck

Benutzt ServerRuleset.gamePhase.

**3.68.3.7 List<Card> createDeck ( ) [private]**

Erzeugt ein Kartendeck, abhängig von dem [RulesetType](#).

**Rückgabe**

Gibt ein Kartendeck zurueck

Benutzt ServerRuleset.RULESET.

Wird benutzt von ServerRuleset.ServerRuleset().

**3.68.3.8 void setFirstPlayer ( PlayerState player ) [protected]**

Setzt den Spieler der als Erster am Zug ist, im Gamestate.

**Parameter**

<i>Der</i>	Spielerzustand des Spielers
------------	-----------------------------

**3.68.3.9 PlayerState getFirstPlayer ( ) [protected]**

Holt den Spieler der als erster am Zug war.

**Rückgabe**

firstPlayer Der Spielzustand des Spielers der als erster am Zug war

**3.68.3.10 boolean nextPlayer ( ) [protected]**

Setzt den naechsten Spieler in der List als currentPlayer.

**Rückgabe**

true falls es ein anderer Spieler ist und false wenn es derselbe ist.

**3.68.3.11 boolean setCurrentPlayer ( PlayerState player ) [protected]**

Setzt den Spieler der am Naechsten am Zug ist, im Gamestate.

**Parameter**

<i>player</i>	Der Playerstate eines Spielers
---------------	--------------------------------

**Rückgabe**

false wenn der selbe Spieler nochmal als currentPlayer gesetzt wird

**3.68.3.12 OtherData getOtherData ( PlayerState player ) [protected]**

Die [OtherData](#) eines Spielers.

**Parameter**

<i>player</i>	Der Spielerzustand
---------------	--------------------

**Rückgabe**

Gibt [OtherData](#) zurück

**3.68.3.13 PlayerState getCurrentPlayer ( ) [protected]**

Holt den Spieler der gerade am Zug ist.

**Rückgabe**

currentPlayer Der Spielzustand des Spielers der grad am Zug ist

**3.68.3.14 void addPlayerToGame ( String name ) [protected]**

Fuegt einen Spieler ins Spiel ein.

**Parameter**

<i>name</i>	Der name vom Spieler
-------------	----------------------

**3.68.3.15 PlayerState getPlayerState ( String name ) [protected]**

Holt den Spielerzustand eines Spielers.

**Parameter**

<i>name</i>	Der Name des Spielers
-------------	-----------------------

**Rückgabe**

playerState Spielzustand eines Spielers

**3.68.3.16 List<Card> getPlayerCards ( PlayerState player ) [protected]**

Holt die Spielkarten eines Spielers.

**Parameter**

<i>name</i>	Der Name eines Spielers
-------------	-------------------------

**Rückgabe**

Die Spielkarten des Spielers

**3.68.3.17 void send ( RulesetMessage message, String name ) [protected]**

Schickt eine Nachricht an einen Spieler, über den Gameserver.

**Parameter**

<i>message</i>	Die Nachricht vom Typ RulesetMessage
<i>name</i>	Der Name vom Spieler

**3.68.3.18 void broadcast ( RulesetMessage message ) [protected]**

Schickt eine Nachricht an alle Spieler.

**Parameter**

<i>message</i>	Die Nachricht
----------------	---------------

**3.68.3.19 void resolveMessage ( RulesetMessage message, String name )**

Verarbeitet eine allgemeine RulesetMessage.

**Parameter**

<i>msgCard</i>	Die Nachricht vom Client welche Karte gespielt wurde
<i>name</i>	Der Name des Spielers

**3.68.3.20 void resolveMessage ( MsgCard msgCard, String name )**

Verarbeitet die RulesetMessage dass eine Karte vom Spieler gespielt.

Die wird dann in isValidMove überprüft, bei falsche Eingabe wird eine MsgCardRequest an den selben Spieler geschickt. Bei richtiger Eingabe geht das Spiel weiter.

**Parameter**

<i>msgCard</i>	Die Nachricht vom Client welche Karte gespielt wurde
<i>name</i>	Der Name des Spielers

**3.68.3.21 void setPoints ( PlayerState player, int i ) [protected]**

Setzt den Punktestand eines Spielers.

**Parameter**

<i>player</i>	Der Spieler
<i>i</i>	Punktestand

**3.68.3.22 int getPoints ( PlayerState player ) [protected]**

Holt den Punktestand eines Spielers.

**Parameter**

<i>player</i>	Der Spieler
---------------	-------------

**Rückgabe**

Die Punkte des Spielers

Benutzt PlayerState.getOtherData().

**3.68.3.23 boolean dealCards ( int number ) [protected]**

Verteilt eine bestimmte Anzahl an Karten an die Spieler.

**Parameter**

<i>number</i>	Die Anzahl an Karten
---------------	----------------------

**Rückgabe**

Gibt true zurueck wenn ein Spieler keine Karten hat, false sonst

**3.68.3.24 boolean giveACard ( PlayerState player, Card card ) [protected]**

Gibt einem Spieler eine bestimmte Karte.

**Parameter**

<i>player</i>	Der Name eines Spielers
<i>card</i>	Eine Karte

**Rückgabe**

Gibt true zurück wenn die Karte im Deck ist, false sonst

**3.68.3.25 boolean playCard ( Card card ) [protected]**

Der momentane Spieler spielt eine Karte.

**Parameter**

<i>card</i>	Die gespielte Karte
-------------	---------------------

**Rückgabe**

true falls der Spieler die Karte hat, false wenn nicht

**3.68.3.26 void setTrumpCard ( Card card ) [protected]**

Setzt eine Karte als Trumpf.

**Parameter**

<i>card</i>	Eine Karte
-------------	------------

**3.68.3.27 abstract boolean isValidMove ( Card card ) [protected], [pure virtual]**

Prueft ob ein gemachter Zug vom currentPlayer in einem Spiel gueltig war, wenn nicht wird an den Spieler erneut eine MsgCardRequest gesendet.

**Parameter**

<i>card</i>	Die Karte die gespielt wurde
-------------	------------------------------

**Rückgabe**

true falls Zug gueltig und false wenn nicht

Implementiert in [ServerWizard](#) und [ServerHearts](#).

**3.68.3.28 abstract GameClientUpdate generateGameClientUpdate ( String player ) [protected], [pure virtual]**

Erzeugt ein [GameClientUpdate](#) welches individuell für jeden Benutzer ist.

**Parameter**

<i>player</i>	Dem Spieler
---------------	-------------

Implementiert in [ServerWizard](#) und [ServerHearts](#).

**3.69 ServerWizard Klassenreferenz**

Abgeleitet von [ServerRuleset](#).

**Öffentliche Methoden**

- [ServerWizard](#) ([GameServer](#) s)

- void `resolveMessage` (`MsgNumber` msgNumber, String name)
- void `resolveMessage` (`MsgSelection` msgSelection, String name)

#### Geschützte Methoden

- boolean `isValidMove` (`Card` card)
- void `calculateRoundOutcome` ()
- int `getplayingRounds` ()
- void `calculateTricks` ()
- String `getWinner` ()
- `GameClientUpdate` `generateGameClientUpdate` (String player)

#### Private Methoden

- void `setPlayingRounds` (int rounds)
- boolean `isValidNumber` (int number, String name)
- boolean `isValidColour` (`Colour` colour, String name)

#### Private Attribute

- int `playingRounds`

#### Statische, private Attribute

- static final int `MIN_PLAYERS` = 3
- static final int `MAX_PLAYERS` = 6
- static final `RulesetType` `RULESET` = `RulesetType.Wizard`

### 3.69.1 Ausführliche Beschreibung

Diese Klasse erstellt das Regelwerk zum Spiel Wizard. Sie enthaelt zudem weitere Methoden, welche für das Spiel Wizard spezifisch benoetigt werden, wie das Ansage von Stichen, der Bestimmung von Trumpffarben und die Bestimmung der Rundenanzahl.

### 3.69.2 Dokumentation der Elementfunktionen

#### 3.69.2.1 boolean `isValidMove` ( `Card` *card* ) [protected],[virtual]

Prueft ob ein gemachter Zug vom `currentPlayer` in einem Spiel gueltig war, wenn nicht wird an den Spieler erneut eine `MsgCardRequest` gesendet.

#### Parameter

<i>card</i>	Die Karte die gespielt wurde
-------------	------------------------------

#### Rückgabe

true falls Zug gueltig und false wenn nicht

Implementiert `ServerRuleset`.

#### 3.69.2.2 void `setPlayingRounds` ( int *rounds* ) [private]

Setzt die Anzahl an Runden die es in diesem Spiel gibt.

**Parameter**

<i>rounds</i>	Die Anzahl an Runden
---------------	----------------------

Benutzt ServerWizard.playingRounds.

### 3.69.2.3 `int getplayingRounds ( )` [protected]

Holt die Anzahl der Runden die gespielt werden.

**Rückgabe**

playingRounds Die Anzahl an Runden

Benutzt ServerWizard.playingRounds.

### 3.69.2.4 `void resolveMessage ( MsgNumber msgNumber, String name )`

Verarbeitet die RulesetMessage dass der Spieler eine Stichansage macht.

Die wird dann in isValidNumber überprüft, bei falsche Eingabe wird eine MsgCardRequest an den selben Spieler geschickt. Bei richtiger Eingabe geht das Spiel weiter.

**Parameter**

<i>msgNumber</i>	Die Nachricht vom Client
<i>name</i>	Der Name des Spielers

### 3.69.2.5 `boolean isValidNumber ( int number, String name )` [private]

Ueberprueft ob eine eingegebene Stichangabe eines Spielers gueltig ist.

**Parameter**

<i>number</i>	Die Stichangabe
<i>name</i>	Der Name vom Spieler

**Rückgabe**

true falls die Stichangabe gültig ist, false wenn nicht

### 3.69.2.6 `void resolveMessage ( MsgSelection msgSelection, String name )`

Verarbeitet die RulesetMessage dass mehrerer Karten vom Spieler uebergeben werden.

Die wird dann in isValidColour überprüft, bei falsche Eingabe wird MsgMultiCardRequest an den selben Spieler geschickt. Bei richtiger Eingabe geht das Spiel weiter.

**Parameter**

<i>msgSelection</i>	Die Nachricht vom Client
<i>name</i>	Der Name des Spielers

### 3.69.2.7 `boolean isValidColour ( Colour colour, String name )` [private]

Ueberprueft ob eine eingegebene Trumpffarbe eines Spielers gueltig ist.

**Parameter**



<i>colour</i>	Die Trumpffarbe
<i>name</i>	Der Name des Spielers

#### Rückgabe

true falls die Farbe gueltig ist, false wenn nicht

#### 3.69.2.8 `GameClientUpdate` `generateGameClientUpdate ( String player )` `[protected]`, `[virtual]`

Erzeugt ein `GameClientUpdate` welches individuell für jeden Benutzer ist.

#### Parameter

<i>player</i>	Dem Spieler
---------------	-------------

Implementiert `ServerRuleset`.

#### 3.69.3 Dokumentation der Datenelemente

##### 3.69.3.1 `int playingRounds` `[private]`

Die Anzahl an Runden die gespielt wird.

Ist abhaengig von der Spieleranzahl.

Wird benutzt von `ServerWizard.getplayingRounds()` und `ServerWizard.setPlayingRounds()`.

#### 3.70 WizardCard Enum-Referenz

Abgeleitet von `Card`.

#### Öffentliche Methoden

- `int getValue ()`
- `Colour getColour ()`

#### Öffentliche Attribute

- **Empty** =(0,Colour.NONE)
- **NarrBlau** =(0,Colour.BLUE)
- **ZaubererRot** =(14,Colour.RED)
- **EinsGruen** =(1,Colour.GREEN)
- **ZweiGruen** =(2,Colour.GREEN)
- **DreiGruen** =(3,Colour.GREEN)
- **ZweiRot** =(2,Colour.RED)
- **DreiRot** =(3,Colour.RED)
- **VierRot** =(4,Colour.RED)

#### Private Methoden

- `WizardCard (int value, Colour colour)`

#### Private Attribute

- `final int value`
- `final Colour colour`

### 3.70.1 Ausführliche Beschreibung

Modelliert eine Wizardkarte

### 3.70.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.70.2.1 WizardCard ( int *value*, Colour *colour* ) [private]

Erzeugt eine Wizardkarte.

Parameter

<i>value</i>	Der Wert der Karte
<i>colour</i>	Die Farbe der Karte

### 3.70.3 Dokumentation der Elementfunktionen

#### 3.70.3.1 int getValue ( )

Gibt den Wert der Karte zurück.

Rückgabe

Der Wert der Karte

Implementiert [Card](#).

#### 3.70.3.2 Colour getColour ( )

Gibt die Farbe der Karte zurück.

Rückgabe

Die Farbe der Karte

Implementiert [Card](#).

## 3.71 WizData Klassenreferenz

Abgeleitet von [OtherData](#).

Öffentliche Methoden

- String [toString](#) ( )

Geschützte Methoden

- [WizData](#) ( )
- int [getAnnouncedTricks](#) ( )
- void [setAnnouncedTricks](#) (int annouceTricks)

Private Attribute

- int [announcedTricks](#)

### 3.71.1 Ausführliche Beschreibung

Die Otherdata eines Spielers zum Spiel Wizard

### 3.71.2 Dokumentation der Elementfunktionen

#### 3.71.2.1 `int getAnnouncedTricks ( )` [protected]

Holt die angesagten Stiche des Spielers.

#### Rückgabe

`announcedTricks` Die angesagten Stiche

#### 3.71.2.2 `void setAnnouncedTricks ( int annouceTricks )` [protected]

Beim Spielstart werden die vorausgesagten Stiche des Spieler gespeichert.

#### Parameter

<i>annouceTricks</i>	Die vorausgesagten Stiche des Spielers
----------------------	--

## 3.72 GameServer Klassenreferenz

Abgeleitet von [Server](#).

#### Öffentliche Methoden

- [GameServer](#) ([LobbyServer](#) server, [Player](#) gameMaster, String GameName, [RulesetType](#) ruleset, String password, boolean hasPassword)
- [GameServerRepresentation](#) [getRepresentation](#) ()
- synchronized void [addPlayer](#) ([Player](#) player)
- synchronized void [removePlayer](#) ([Player](#) player)
- void [sendRulesetMessage](#) (String player, RulesetMessage message)
- void [broadcastRulesetMessage](#) (RulesetMessage message)
- void [receiveMessage](#) ([Player](#) player, ComKickPlayerRequest kickPlayer)
- void [receiveMessage](#) ([Player](#) player, ComChatMessage chat)
- void [receiveMessage](#) ([Player](#) player, ComClientLeave leave)
- void [receiveMessage](#) ([Player](#) player, ComClientQuit quit)
- void [receiveMessage](#) ([Player](#) player, ComStartGame start)
- void [receiveMessage](#) ([Player](#) player, ComRuleset ruleset)
- ComInitGameLobby [initLobby](#) ()
- void [handleIOException](#) ([Player](#) player)

#### Private Attribute

- [LobbyServer](#) lobbyServer
- String gameMasterName
- String name
- String password
- int maxPlayers
- int currentPlayers
- boolean hasPassword
- [RulesetType](#) rulesetType
- [ServerRuleset](#) ruleset

## Weitere Geerbte Elemente

## 3.72.1 Ausführliche Beschreibung

Diese Klasse ist fuer die Verwaltung eines Spieles zustaendig. Sie verwaltet die Kommunikation zwischen den Clients waehrend eines Spieles. Die GameServer-Klasse erbt Methoden zur Kommunikation vom [Server](#). Der [GameServer](#) tauscht Nachrichten zwischen Ruleset und [Player](#) aus, um so den Spielablauf zu koordinieren.

## 3.72.2 Beschreibung der Konstruktoren und Destruktoren

### 3.72.2.1 GameServer ( LobbyServer server, Player gameMaster, String GameName, RulesetType ruleset, String password, boolean hasPassword )

Konstruktor des GameServers.

Setzt die Attribute lobbyServer, name, password, hasPasword und rulesetType auf die uebergebenen Werte. Setzt den gameMasterName auf den Namen des gameMaster und fuegt den gameMaster dem Set an Spielern hinzu. Bestimmt mithilfe des Enums RulesetType das Ruleset und erstellt es. Setzt currentPlayers auf eins und maxPlayers je nach Ruleset.

## Parameter

<i>server</i>	ist der <a href="#">LobbyServer</a> der den <a href="#">GameServer</a> erstellt hat.
<i>gameMaster</i>	ist der Name des Spielleiters
<i>GameName</i>	ist der Name des Spiels
<i>ruleset</i>	gibt an, welches Ruleset verwendet wird
<i>password</i>	speichert das Passwort des Spiels
<i>hasPassword</i>	gibt an, ob das Spiel ein Passwort hat

## 3.72.3 Dokumentation der Elementfunktionen

### 3.72.3.1 GameServerRepresentation getRepresentation ( )

Erstellt eine neue [GameServerRepresentation](#) und gibt sie zurueck.

## Rückgabe

Gibt die neue [GameServerRepresentation](#) zurueck

Benutzt GameServer.currentPlayers, GameServer.gameMasterName, GameServer.hasPassword, GameServer.maxPlayers, GameServer.name und GameServer.rulesetType.

### 3.72.3.2 synchronized void addPlayer ( Player player )

Diese Methode fuegt einen [Player](#) dem Set an Playern hinzu, welche der [Server](#) verwaltet.

Es wird vorausgesetzt, dass der [Player](#) gueltig und noch nicht im Set vorhanden ist. Zusaetzlich wird die Zahl der currentPlayers um eins Erhoeht.

## Parameter

<i>player</i>	ist der <a href="#">Player</a> , der hinzugefuegt wird
---------------	--

Benutzt GameServer.currentPlayers.

### 3.72.3.3 synchronized void removePlayer ( Player player )

Diese Methode entfernt einen [Player](#) aus dem Set an Playern, welche der [Server](#) verwaltet.

Es wird vorausgesetzt, dass der [Player](#) gueltig und im Set vorhanden ist. Zusaetzlich wird die Zahl der currentPlayers um eins Verringert.

**Parameter**

<i>player</i>	ist der <a href="#">Player</a> , der entfernt wird
---------------	--

Benutzt `GameServer.currentPlayers`.

#### 3.72.3.4 void `sendRulesetMessage ( String player, RulesetMessage message )`

Diese Methode verpackt eine `RulesetMessage` in ein `ComObject` und verschickt es mit `sendToPlayer()` an einen bestimmten Spieler.

**Parameter**

<i>player</i>	ist der Name des Spielers an den die Nachricht verschickt wird
<i>message</i>	ist die Ruleset Nachricht, die in ein <code>ComObject</code> verpackt wird

Benutzt `Server.sendToPlayer()`.

#### 3.72.3.5 void `broadcastRulesetMessage ( RulesetMessage message )`

Diese Methode verpackt eine `RulesetMessage` in ein `ComObject` und verschickt es mit `broadcast()` an alle Spieler.

**Parameter**

<i>message</i>	ist die Ruleset Nachricht, die in ein <code>ComObject</code> verpackt wird
----------------	--

Benutzt `Server.broadcast()`.

#### 3.72.3.6 void `receiveMessage ( Player player, ComKickPlayerRequest kickPlayer )`

Diese Methode ist dafuer zustandig zu ermitteln, was passiert wenn ein Spieler aus der GameLobby geworfen wird.

Der [Player](#) wird durch Aufruf von `changeServer` an die Lobby zurueckgegeben. An diesen Spieler wird ein `ComInitLobby` und ein `ComWarning` geschickt. Danach wird ein `ComUpdatePlayerlist` Objekt mit `broadcast` an alle Clients im Spiel verschickt.

**Parameter**

<i>player</i>	ist der Thread der die Nachricht erhalten hat
<i>kicked</i>	ist das <code>ComObject</code> , das verarbeitet wird

#### 3.72.3.7 void `receiveMessage ( Player player, ComChatMessage chat )`

Diese Methode ist dafur zustandig eine Chatnachricht an alle Clients im Spiel zu verschicken.

Dafuer wird die `ComChatMessage` mit `broadcast` an alle Spieler im `playerSet` verteilt.

**Parameter**

<i>player</i>	ist der Thread der die Nachricht erhalten hat
<i>chat</i>	ist das <code>ComObject</code> , das die Chatnachricht enthaelt

Benutzt `Server.broadcast()`.

#### 3.72.3.8 void `receiveMessage ( Player player, ComClientLeave leave )`

Diese Methode gibt einen [Player](#), der die GameLobby verlassen will, durch Aufruf von `changeServer` an die Server-Lobby zurueck und schickt ihm ein `ComInitLobby`.

Danach wird ein `ComUpdatePlayerlist` Objekt mit `broadcast` an alle Clients im Spiel verschickt.

## Parameter

<i>player</i>	ist der Thread der die Nachricht erhalten hat
<i>leave</i>	ist das ComObject, welches angibt, dass der Spieler in die Lobby zurueckkehrt

3.72.3.9 void receiveMessage ( Player *player*, ComClientQuit *quit* )

Diese Methode behandelt den Fall, dass ein Spieler das laufende Spiel verlaesst.

Alle Spieler, die sich im Spiel befinden werden durch Aufruf von changeServer an die Lobby zurueckgegeben und bekommen ein ComInitLobby und ein ComWarning. Das Spiel wird aus dem gameServerSet des LobbyServers entfernt.

## Parameter

<i>player</i>	ist der Thread der die Nachricht erhalten hat
<i>quit</i>	ist das ComObject, welches angibt, dass der Spieler das Spiel verlaesst

3.72.3.10 void receiveMessage ( Player *player*, ComStartGame *start* )

Diese Methode sagt dem Ruleset, dass ein neues Spiel gestartet werden soll indem er dessen runGame Methode aufruft.

## Parameter

<i>player</i>	ist der Thread der die Nachricht erhalten hat
<i>start</i>	ist das ComObject, dass angibt, dass das Spiel gestartet werden soll

3.72.3.11 void receiveMessage ( Player *player*, ComRuleset *ruleset* )

Diese Methode gibt das erhaltene ComRuleset durch einen Aufruf von resolveMessage an das Ruleset weiter.

## Parameter

<i>player</i>	ist der Thread der die Nachricht erhalten hat
<i>ruleset</i>	ist das ComObject, das zeigt, dass das Object vom Ruleset bearbeitet werden muss

## 3.72.3.12 ComInitGameLobby initLobby ( )

Baut ein neues ComInitGameLobby Objekt und gibt es zurueck.

## Rückgabe

Gibt das ComInitGameLobby Objekt zurueck

3.72.3.13 void handleIOException ( Player *player* )

Diese Methode legt den Ablauf fest, was passiert, falls die Verbindung zu einem Client verloren gegangen ist.

Der uebergebene [Player](#) wird aus dem playerSet im [GameServer](#), sowie dem names Set im [LobbyServer](#) entfernt. Alle Spieler, die sich im Spiel befinden werden durch Aufruf von changeServer an die Lobby zurueckgegeben und bekommen ein ComInitLobby und ein ComWarning. Das Spiel wird aus dem gameServerSet des LobbyServers entfernt.

## Parameter

<i>player</i>	ist der Tread von dem die IOException kommt
---------------	---

## 3.72.4 Dokumentation der Datenelemente

### 3.72.4.1 String password [private]

Das Passwort, das der Spielleiter beim erstellen gesetzt hat.

Ist NULL, falls es kein Passwort gibt.

## 3.73 GameServerRepresentation Klassenreferenz

### Öffentliche Methoden

- [GameServerRepresentation](#) (String gameMaster, String gameName, int max, int current, [RulesetType](#) type, boolean password)
- String [getGameMasterName](#) ()
- String [getName](#) ()
- int [getMaxPlayers](#) ()
- int [getCurrentPlayers](#) ()
- [RulesetType](#) [getRuleset](#) ()
- boolean [isHasPassword](#) ()

### Private Attribute

- String [gameMasterName](#)
- String [name](#)
- int [maxPlayers](#)
- int [currentPlayers](#)
- [RulesetType](#) [ruleset](#)
- boolean [hasPassword](#)

### 3.73.1 Ausführliche Beschreibung

Dies ist eine Klasse, die Informationen ueber den Zustand eines Spielers bereithaelt. Sie wird dem ComObjekt ComLobbyUpdateGameList angehaengt, um die Spielliste in der GameLobby aktualisieren zu koennen.

### 3.73.2 Beschreibung der Konstruktoren und Destruktoren

#### 3.73.2.1 GameServerRepresentation ( String gameMaster, String gameName, int max, int current, RulesetType type, boolean password )

Der Konstruktor der Klasse [GameServerRepresentation](#) initialisiert die Attribute mit den vom [GameServer](#) übergebenen Werten.

#### Parameter

<i>gameMaster</i>	der Name des Spielleiters
<i>gameName</i>	der Name des Spiels
<i>max</i>	Maximal moegliche Anzahl teilnehmender Spieler
<i>current</i>	Anzahl momentaner Spieler
<i>type</i>	Welches Ruleset verwendet wird
<i>password</i>	ob das Spiel ein Passwort hat

## 3.74 LobbyServer Klassenreferenz

Abgeleitet von [Server](#).

## Klassen

- class [ClientListenerThread](#)

## Öffentliche Methoden

- [LobbyServer](#) ()
- void [addName](#) (String name)
- void [removeName](#) (String name)
- void [addGameServer](#) ([GameServer](#) game)
- void [removeGameServer](#) ([GameServer](#) game)
- void [receiveMessage](#) ([Player](#) player, ComChatMessage chat)
- void [receiveMessage](#) ([Player](#) player, ComClientQuit quit)
- void [receiveMessage](#) ([Player](#) player, ComCreateGameRequest create)
- void [receiveMessage](#) ([Player](#) player, ComJoinRequest join)
- void [receiveMessage](#) ([Player](#) player, ComLoginRequest login)
- ComInitLobby [initLobby](#) ()
- void [handleIOException](#) ([Player](#) player)

## Private Attribute

- Set< String > [names](#)
- Set< [Player](#) > [noNames](#)
- Set< [GameServer](#) > [gameServerSet](#)
- [ClientListenerThread](#) [clientListenerThread](#)
- ServerSocket [socket](#)

## Weitere Geerbte Elemente

## 3.74.1 Ausführliche Beschreibung

Diese Klasse ist fuer die Verwaltung der Spiellobby auf dem [Server](#) verantwortlich. Sie erstellt neue Spiele und verwaltet laufende Spiele. Auch wird der Chatverkehr ueber sie an die verbundenen Spieler weitergeleitet. Die LobbyServer-Klasse erbt Methoden zur Kommunikation vom [Server](#).

## 3.74.2 Dokumentation der Elementfunktionen

## 3.74.2.1 void addName ( String name )

Fuegt einen neuen Benutzennamen in das Namensset ein.

Es wird vorausgesetzt, dass der Name noch nicht im Set vorhanden ist.

## Parameter

<i>name</i>	ist der Name der eingefuegt wird
-------------	----------------------------------

## 3.74.2.2 void removeName ( String name )

Loescht einen Benutzennamen aus dem Namensset.

Es wird vorausgesetzt, dass der Name im Set vorhanden ist.



## Parameter

<i>name</i>	ist der Name der gelöscht wird
-------------	--------------------------------

3.74.2.3 void addGameServer ( *GameServer game* )

Fuegt einen neuen *GameServer* in das *gameServerSet* ein.

## Parameter

<i>game</i>	ist der <i>GameServer</i> der eingefuegt wird
-------------	---

3.74.2.4 void removeGameServer ( *GameServer game* )

Loescht einen *GameServer* aus dem *Gameserverset*.

## Parameter

<i>game</i>	ist der <i>GameServer</i> der gelöscht wird
-------------	---

3.74.2.5 void receiveMessage ( *Player player*, *ComChatMessage chat* )

Diese ueberladene Methode ist da fuer zustaendig eine Chatnachricht an alle Clients im Spiel zu verschicken.

Dafuer wird die *ComChatMessage* mit broadcast an alle Spieler im *playerSet* verteilt.

## Parameter

<i>player</i>	ist der Thread der die Nachricht erhalten hat
<i>chat</i>	ist das <i>ComObject</i> , das die Chatnachricht enthaelt

Benutzt *Server.broadcast()*.

3.74.2.6 void receiveMessage ( *Player player*, *ComClientQuit quit* )

Diese ueberladene Methode schliesst die Verbindung, der *Player* wird aus dem *playerSet* (bzw.

*noNames Set*) entfernt, der Name des Players wird aus dem Set *names* entfernt. War der Spieler im *playerSet*, wird ein *ComUpdatePlayerlist* mit broadcast an alle Clients verschickt.

## Parameter

<i>player</i>	ist der Thread der die Nachricht erhalten hat
<i>quit</i>	ist das <i>ComObject</i> , welches angibt, dass der Spieler das Spiel vollstaendig verlaesst

3.74.2.7 void receiveMessage ( *Player player*, *ComCreateGameRequest create* )

Diese ueberladene Methode erstellt einen neuen *GameServer* fuegt ihm den *Player* durch Aufruf von dessen *changeServer* Methode hinzu.

Der neue *GameServer* wird in das *gameServerSet* eingefuegt. Durch broadcast wird im *LobbyServer* sowohl *ComUpdatePlayerlist* als auch ein *ComLobbyUpdateGamelist* verschickt. Zusaetzlich wird dem Client mit *sendToPlayer* ein *ComInitGameLobby* geschickt.

## Parameter

<i>player</i>	ist der Thread der die Nachricht erhalten hat
<i>create</i>	ist das <i>ComObject</i> , welches angibt, dass der <i>Player</i> ein neues Spiel erstellt hat

3.74.2.8 void receiveMessage ( *Player player*, *ComJoinRequest join* )

Diese ueberladene Methode fuegt einen *Player* dem entsprechenden *GameServer* hinzu.

Falls das Passwort nicht leer ist wird geprueft, ob es mit dem Passwort des Spieles uebereinstimmt, wenn nicht,

wird ein ComWarning an den Client geschickt. Ansonsten wird und der [Player](#) dem, durch Namen des Spielleiters identifizierten, Gameserver, durch Aufruf von changeServer uebergeben. Dem joinendenClient wird mit sendTo-Player ein ComInitGameLobby geschickt. Durch broadcast wird sowohl im [LobbyServer](#) ein ComUpdatePlayerlist verschickt.

#### Parameter

<i>player</i>	ist der Thread der die Nachricht erhalten hat
<i>join</i>	ist das ComObject, welches angibt, dass der <a href="#">Player</a> einem Spiel beitreten will

#### 3.74.2.9 void receiveMessage ( [Player](#) *player*, ComLoginRequest *login* )

Diese ueberladene Methode ueberprueft, ob der Name im Set names vorhanden ist, falls ja, wird ein ComWarning an den Client geschickt, dass der Name bereits vergeben ist, falls nein, wird im [Player](#) setName aufgerufen.

Der [Player](#) wird aus dem noNames Set entfernt und in das playerSet eingefuegt. Der Name wird in das Set names eingefuegt. Dem Client wird ein ComServerAcknowledgement geschickt.

#### Parameter

<i>player</i>	ist der Thread der die Nachricht erhalten hat
<i>login</i>	ist das ComObject, dass den Benutzernamen des Clients enthält

#### 3.74.2.10 ComInitLobby initLobby ( )

Diese Methode baut ein neues ComInitLobby Objekt und gibt es zurueck.

#### Rückgabe

Gibt das ComInitLobby Objekt zurueck

#### 3.74.2.11 void handleIOException ( [Player](#) *player* )

Diese Methode legt den Ablauf fest, was passiert, falls die Verbindung zu einem Client verloren gegangen ist.

Der uebergebene [Player](#) wird aus dem playerSet sowie dem names Set im [LobbyServer](#) entfernt.

#### Parameter

<i>player</i>	ist der Tread von dem die IOException kommt
---------------	---

### 3.75 LobbyServer.ClientListenerThread Klassenreferenz

Abgeleitet von Runnable.

#### Öffentliche Methoden

- [ClientListenerThread](#) ()
- void [run](#) ()

#### 3.75.1 Ausführliche Beschreibung

Diese innere Klasse ist fuer das Zustandekommen von Clientverbindungen zustaendig. Der Thread wartet auf eingehende Clientverbindungen, stellt diese her und instanziiert fuer jede Verbindung eine Klasse [Player](#). Dieser wird dann dem [LobbyServer](#) uebergeben.

### 3.76 Player Klassenreferenz

Abgeleitet von Runnable.

#### Öffentliche Methoden

- **Player** (**Server** lobbyServer, ObjectOutputStream output, ObjectInputStream input)
- void **run** ()
- void **send** (ComObject com)
- void **changeServer** (**Server** newServer)
- String **getName** ()
- void **setName** (String newName)

#### Private Attribute

- String **name**
- **Server** server
- ObjectOutputStream **comOut**
- ObjectInputStream **comIn**

#### 3.76.1 Ausführliche Beschreibung

Die Player-Klasse wird zum Versenden von Java Serializable Objects, sowie zum Annehmen solcher verwendet. Sie verwaltet fuer die Dauer einer Serververbindung die Verbindung zu einem Client.

#### 3.76.2 Beschreibung der Konstruktoren und Destruktoren

##### 3.76.2.1 **Player** ( **Server** lobbyServer, ObjectOutputStream output, ObjectInputStream input )

Konstruktor des Players, in ihm werden die Attribute server, comOut und comIn mit vom ClientListenerThread uebergebenen Werten Instanziert.

#### Parameter

<i>lobbyServer</i>	ist der <b>LobbyServer</b> , der zu Beginn den <b>Player</b> verwaltet.
<i>output</i>	ist der ObjectOutputStream an den entsprechenden Client
<i>input</i>	ist der ObjectInputStream vom entsprechenden Client

Benutzt Player.comIn, Player.comOut und Player.server.

#### 3.76.3 Dokumentation der Elementfunktionen

##### 3.76.3.1 void **run** ( )

Die run-Methode des Thread nimmt eingehende Nachrichten des Client entgegen und uebergibt diese an den **Server** durch Aufruf der Methode resolveMessage() Faengt eine ClassNotFoundException ab, falls die Klasse nicht gefunden werden kann und gibt einen Fehler aus.

Faengt eine IOException ab und ruft im jeweiligen **Server**, dem er zugeteilt ist die handleIOException Methode auf.

Benutzt Player.comIn und Player.server.

##### 3.76.3.2 void **send** ( ComObject com )

Diese Methode schickt ein ComObjekt an den Client.

Sie faengt eine IOException ab und ruft im jeweiligen **Server**, dem er zugeteilt ist die handleIOException Methode auf.

## Parameter

<i>com</i>	ist das ComObject das verschickt wird
------------	---------------------------------------

3.76.3.3 void changeServer ( [Server](#) *newServer* )

Diese Methode wechselt beim [Player](#) den [Server](#) an den er comObjects weiterleiten soll.

Dabei wird er aus dem playerSet des alten Servers entfernt und in das playerSet des neuen Servers eingefuegt. Danach wird vom neuen [Server](#) ein ComUpdatePlayerlist Objekt mit broadcast an alle Clients, die vom [Server](#) verwaltet werden, verschickt.

## Parameter

<i>newServer</i>	ist der neue <a href="#">Server</a>
------------------	-------------------------------------

Benutzt Player.getName() und Player.server.

## 3.76.3.4 String getName ( )

Getter-Methode fuer den Benutzernamen.

## Rückgabe

gibt den Benutzernamen des Spielers zurueck

Benutzt Player.name.

Wird benutzt von Player.changeServer().

3.76.3.5 void setName ( [String](#) *newName* )

Setter-Methode fuer den Benutzernamen.

## Parameter

<i>newName</i>	ist der neue Name
----------------	-------------------

Benutzt Player.name.

## 3.77 Server Klassenreferenz

Basisklasse für [GameServer](#) und [LobbyServer](#).

## Öffentliche Methoden

- void [receiveMessage](#) ([Player](#) player, ComObject com)
- synchronized void [sendToPlayer](#) (String name, ComObject com)
- synchronized void [addPlayer](#) ([Player](#) player)
- synchronized void [removePlayer](#) ([Player](#) player)
- synchronized void [broadcast](#) (ComObject com)
- void [handleIOException](#) ([Player](#) player)

## Geschützte Attribute

- Set< [Player](#) > [playerSet](#)

## 3.77.1 Ausführliche Beschreibung

Ist eine abstrakte Klasse, von der die Klassen [LobbyServer](#) und [GameServer](#) erben. Es stellt Methoden zur Nachrichtenversendung und -verarbeitung bereit, sowie zur Verwaltung von Playern

### 3.77.2 Dokumentation der Elementfunktionen

#### 3.77.2.1 void receiveMessage ( Player player, ComObject com )

Diese Methode dient zur Verarbeitung von eingehenden ComObjects.

Parameter

<i>player</i>	ist der <a href="#">Player</a> von dem die Nachricht kommt
<i>com</i>	ist das ComObject vom Client verschickt wurde

#### 3.77.2.2 synchronized void sendToPlayer ( String name, ComObject com )

Diese Methode wird genutzt, um ein ComObject an einen einzigen Client zu verschicken.

Der [Player](#) der die Nachricht verschicken soll wird Anhand des uebergebenen Benutzernamens identifiziert. Es wird vorausgesetzt, dass der Name und das ComObject gueltig sind.

Parameter

<i>name</i>	ist der Name des Clients, an den der <a href="#">Player</a> die Nachricht verschicken soll
<i>c</i>	ist das ComObject, dass verschickt werden soll

Benutzt Server.playerSet.

Wird benutzt von GameServer.sendRulesetMessage().

#### 3.77.2.3 synchronized void addPlayer ( Player player )

Diese Methode fuegt einen [Player](#) dem Set an Playern hinzu, welche der [Server](#) verwaltet.

Es wird vorausgesetzt, dass der [Player](#) gueltig und noch nicht im Set vorhanden ist.

Parameter

<i>player</i>	ist der <a href="#">Player</a> , der hinzugefuegt wird
---------------	--

#### 3.77.2.4 synchronized void removePlayer ( Player player )

Diese Methode entfernt einen [Player](#) aus dem Set an Playern, welche der [Server](#) verwaltet.

Es wird vorausgesetzt, dass der [Player](#) gueltig und im Set vorhanden ist.

Parameter

<i>player</i>	ist der <a href="#">Player</a> , der entfernt wird
---------------	--

#### 3.77.2.5 synchronized void broadcast ( ComObject com )

Diese Methode wird genutzt, um ein ComObject an alle Clients, die vom [Server](#) verwaltet werden, zu schicken.

Es wird vorausgesetzt, dass das ComObject gueltig ist.

Parameter

<i>com</i>	ist das ComObject, dass verschickt werden soll
------------	--

Benutzt Server.playerSet.

Wird benutzt von GameServer.broadcastRulesetMessage(), LobbyServer.receiveMessage() und GameServer.receiveMessage().

#### 3.77.2.6 void handleIOException ( Player player )

Diese Methode legt den Ablauf fest, was passiert, falls die Verbindung zu einem Client verloren gegangen ist.

## Parameter

<i>player</i>	ist der Tread von dem die IOException kommt
---------------	---

## 3.78 ServerMain Klassenreferenz

## Öffentliche, statische Methoden

- static void [main](#) (String[] args)

## Private Attribute

- [LobbyServer lobbyServer](#)

### 3.78.1 Ausführliche Beschreibung

Diese Klasse startet den [Server](#) und ist fuer die Konfiguration des Servers verantwortlich.

### 3.78.2 Dokumentation der Elementfunktionen

#### 3.78.2.1 static void main ( String[] args ) [static]

Die main-Methode erstellt einen neuen [LobbyServer](#).

## Parameter

<i>args</i>	
-------------	--

## 4 JUnit-Tests

JUnit-Tests werden für die folgenden Klassen geschrieben: ClientModel, LobbyServer, GameServer, ClientHearts, ClientWizard, ServerHearts und ServerWizard. Für die folgenden Fälle wurden bereits Tests implementiert.

### 4.1 isValidWizardMove

```
package Ruleset;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import test.TestGameServer;
import test.TestLobbyServer;
import test.TestPlayer;

import Server.GameServer;
import Server.LobbyServer;
import Server.Player;

public class TestIsValidMoveWizard {
```

```

ServerRuleset ruleset;

GameServer gameServer;

LobbyServer lobbyServer;

Player player;
String player1;
String player2;
String player3;
PlayerState playerState1;
PlayerState playerState2;
PlayerState playerState3;

@Before
public void setUp() throws Exception {
    player1 = "Tick";
    player2 = "Trick";
    player3 = "Track";
    lobbyServer = new TestLobbyServer();
    player = new TestPlayer(lobbyServer, null, null);
    gameServer = new TestGameServer(lobbyServer, player, "Mein
        Spiel", RulesetType.Wizard,
        "", false);
    ruleset = new ServerWizard(gameServer);

    ruleset.addPlayerToGame(player1);
    ruleset.addPlayerToGame(player2);
    ruleset.addPlayerToGame(player3);

    playerState1 = ruleset.getPlayerState(player1);
    playerState2 = ruleset.getPlayerState(player2);
    playerState3 = ruleset.getPlayerState(player3);

    ruleset.setFirstPlayer(ruleset.getPlayerState(player1));
    ruleset.setTrumpCard(WizardCard.VierRot);

    ruleset.giveACard(playerState1, WizardCard.DreiGruen);
    ruleset.giveACard(playerState1, WizardCard.ZaubererRot);
    ruleset.giveACard(playerState1, WizardCard.ZweiBlau);

    ruleset.giveACard(playerState2, WizardCard.ZweiGruen);
    ruleset.giveACard(playerState2, WizardCard.DreiRot);
    ruleset.giveACard(playerState2, WizardCard.ZweiGelb);

    ruleset.giveACard(playerState3, WizardCard.NarrBlau);
    ruleset.giveACard(playerState3, WizardCard.EinsGruen);
    ruleset.giveACard(playerState3, WizardCard.ZweiRot);
}

@Test
public void testSorcerer() {
    ruleset.playCard(WizardCard.ZaubererRot);
    ruleset.setCurrentPlayer(playerState2);

    boolean isValidMove = ruleset.isValidMove(WizardCard.DreiRot);

    assertTrue(isValidMove);
}

@Test
public void testRed3OnGreen3() {

```

```

        ruleset.playCard(WizardCard.DreiGruen);
        ruleset.setCurrentPlayer(playerState2);
        boolean isValidMove = ruleset.isValidMove(WizardCard.DreiRot);

        assertFalse(isValidMove);
    }

    @Test
    public void testGreen2OnGreen3() {
        ruleset.playCard(WizardCard.DreiGruen);
        ruleset.setCurrentPlayer(playerState2);

        boolean isValidMove = ruleset.isValidMove(WizardCard.ZweiGruen);

        assertTrue(isValidMove);
    }

    @Test
    public void testFoolBlueOnGreen2OnGreen3() {
        ruleset.playCard(WizardCard.DreiGruen);
        ruleset.setCurrentPlayer(playerState2);

        ruleset.playCard(WizardCard.ZweiGruen);
        ruleset.setCurrentPlayer(playerState3);

        boolean isValidMove = ruleset.isValidMove(WizardCard.NarrBlau);

        assertTrue(isValidMove);
    }
}

```

---

## 4.2 isValidHeartsMove

---

```

package Ruleset;

import static org.junit.Assert.*;
import static org.junit.Assert.assertEquals;

import org.junit.Before;
import org.junit.Test;

import test.TestGameServer;
import test.TestLobbyServer;
import test.TestPlayer;

import Server.GameServer;
import Server.LobbyServer;
import Server.Player;

public class TestIsValidMoveHearts {
    ServerRuleset ruleset;

    GameServer gameServer;

    LobbyServer lobbyServer;

    Player player;
    String player1;
    String player2;
}

```

---



```

String player3;
String player4;

PlayerState playerState1;
PlayerState playerState2;
PlayerState playerState3;
PlayerState playerState4;

@Before
public void setUp() throws Exception {
    player1 = "Tick";
    player2 = "Trick";
    player3 = "Track";
    player3 = "Duck";
    lobbyServer = new TestLobbyServer();
    player = new TestPlayer(lobbyServer, null, null);
    gameServer = new TestGameServer(lobbyServer, player, "Mein Spiel",
        RulesetType.Hearts, "", false);
    ruleset = new ServerHearts(gameServer);

    ruleset.addPlayerToGame(player1);
    ruleset.addPlayerToGame(player2);
    ruleset.addPlayerToGame(player3);
    ruleset.addPlayerToGame(player4);

    playerState1 = ruleset.getPlayerState(player1);
    playerState2 = ruleset.getPlayerState(player2);
    playerState3 = ruleset.getPlayerState(player3);

    ruleset.setFirstPlayer(playerState1);
}

@Test
public void testIsValidMove() {
    ruleset.giveACard(playerState1, HeartsCard.Herz2);
    ruleset.giveACard(playerState1, HeartsCard.Kreuz9);
    ruleset.giveACard(playerState2, HeartsCard.Caro3);
    ruleset.giveACard(playerState2, HeartsCard.Caro6);
    ruleset.giveACard(playerState3, HeartsCard.Pik4);
    ruleset.giveACard(playerState3, HeartsCard.Pik5);
    ruleset.giveACard(playerState4, HeartsCard.Pik1);
    ruleset.giveACard(playerState4, HeartsCard.Herz7);

    boolean isValidMove = ruleset.isValidMove(HeartsCard.Herz2);

    assertFalse(isValidMove);

    boolean isValidMove2 = ruleset.isValidMove(HeartsCard.Caro3);

    assertTrue(isValidMove2);
}

@Test
public void testIsValidMoveOnlyHearts() {
    ruleset.giveACard(playerState1, HeartsCard.Herz2);
    ruleset.giveACard(playerState1, HeartsCard.Herz5);
    ruleset.giveACard(playerState2, HeartsCard.Caro3);
    ruleset.giveACard(playerState2, HeartsCard.Caro6);
    ruleset.giveACard(playerState3, HeartsCard.Pik4);
    ruleset.giveACard(playerState3, HeartsCard.Pik5);
    ruleset.giveACard(playerState4, HeartsCard.Pik1);
    ruleset.giveACard(playerState4, HeartsCard.Herz7);
}

```

```

        boolean isValidMove = ruleset.isValidMove(HeartsCard.Herz2);

        assertTrue(isValidMove);

        boolean isValidMove2 = ruleset.isValidMove(HeartsCard.Herz5);

        assertTrue(isValidMove2);
    }
}

```

---

### 4.3 getWinner

```

package Ruleset;

import static org.junit.Assert.*;

import java.util.List;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import testKlassen.TestPlayer;
import ComObjects.ComObject;
import ComObjects.ComRuleset;
import ComObjects.MsgGameEnd;
import Server.GameServer;
import Server.LobbyServer;
import Server.Player;

public class TestHeartsWinner {

    LobbyServer lobbyServer;

    GameServer gameServer;

    ServerRuleset heartsServerRuleset;

    TestPlayer blue;

    TestPlayer white;

    TestPlayer orange;

    TestPlayer brown;

    List<ComObject> inputList;

    ComRuleset comObject;

    MsgGameEnd endMsg;

    String winner;

    @Before
    public void setUp() {
        lobbyServer = new LobbyServer();
        blue = new TestPlayer(lobbyServer, null, null);
        white = new TestPlayer(lobbyServer, null, null);
        orange = new TestPlayer(lobbyServer, null, null);
        brown = new TestPlayer(lobbyServer, null, null);
    }
}

```

---

```

    }

    @After
    public void tearDown() {
        blue = null;
        white = null;
        orange = null;
        brown = null;
        lobbyServer = null;
        gameServer = null;
        inputList = null;
        comObject = null;
        endMsg = null;
        winner = null;
    }

    @Test
    public void testGetWinner() {

        gameServer = new GameServer(lobbyServer, blue, "Test Game",
            RulesetType.Wizard, "", false);
        gameServer.addPlayer(white);
        gameServer.addPlayer(orange);
        gameServer.addPlayer(brown);

        heartsServerRuleset = new ServerWizard(gameServer);
        heartsServerRuleset.addPlayerToGame("Mr. Blue");
        heartsServerRuleset.addPlayerToGame("Mr. White");
        heartsServerRuleset.addPlayerToGame("Mr. Orange");
        heartsServerRuleset.addPlayerToGame("Mr. Brown");
        heartsServerRuleset.setPoints(heartsServerRuleset.getPlayerState("Mr.
            Blue"), 80);
        heartsServerRuleset.setPoints(heartsServerRuleset.getPlayerState("Mr.
            White"), 20);
        heartsServerRuleset.setPoints(heartsServerRuleset.getPlayerState("Mr.
            Orange"), 60);
        heartsServerRuleset.setPoints(heartsServerRuleset.getPlayerState("Mr.
            Brown"), 110);
        heartsServerRuleset.setGamePhase(GamePhase.Ending);
        heartsServerRuleset.calculateRoundOutcome();

        assertTrue(heartsServerRuleset.getWinner().equals("Mr. Brown"));

        inputList = blue.getServerInput();
        comObject = (ComRuleset) inputList.get(1);
        endMsg = (MsgGameEnd) comObject.getRulesetMessage();
        winner = endMsg.getWinner();
        assertEquals("Nachricht an Blue", "Mr. Brown", winner);

        inputList = white.getServerInput();
        comObject = (ComRuleset) inputList.get(1);
        endMsg = (MsgGameEnd) comObject.getRulesetMessage();
        winner = endMsg.getWinner();
        assertEquals("Nachricht an White", "Mr. Brown", winner);

        inputList = orange.getServerInput();
        comObject = (ComRuleset) inputList.get(1);
        endMsg = (MsgGameEnd) comObject.getRulesetMessage();
        winner = endMsg.getWinner();
        assertEquals("Nachricht an Orange", "Mr. Brown", winner);

        inputList = brown.getServerInput();
        comObject = (ComRuleset) inputList.get(1);
    }

```

```

        endMsg = (MsgGameEnd) comObject.getRulesetMessage();
        winner = endMsg.getWinner();
        assertEquals("Nachricht an Brown", "Mr. Brown", winner);
    }
}

```

---

```

package Ruleset;

import static org.junit.Assert.*;

import java.util.List;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import testKlassen.TestPlayer;
import ComObjects.ComChatMessage;
import ComObjects.ComObject;
import ComObjects.ComRuleset;
import ComObjects.MsgGameEnd;
import Server.GameServer;
import Server.LobbyServer;

/**
 * Testet ob der richtige Sieger ermittelt wird und ob jedem Mitspieler
 * der richtige Sieger mitgeteilt wird
 */
public class TestWizardWinner {

    LobbyServer lobbyServer;

    GameServer gameServer;

    ServerRuleset wizardServerRuleset;

    TestPlayer blue;

    TestPlayer white;

    TestPlayer orange;

    TestPlayer brown;

    List<ComObject> inputList;

    ComRuleset comObject;

    MsgGameEnd endMsg;

    String winner;

    @Before
    public void setUp() {
        lobbyServer = new LobbyServer();
        blue = new TestPlayer(lobbyServer, null, null);
        white = new TestPlayer(lobbyServer, null, null);
        orange = new TestPlayer(lobbyServer, null, null);
        brown = new TestPlayer(lobbyServer, null, null);
    }

    @After
    public void tearDown() {
        blue = null;
    }
}

```

```

        white = null;
        orange = null;
        brown = null;
        lobbyServer = null;
        gameServer = null;
        inputList = null;
        inputList = null;
        comObject = null;
        endMsg = null;
        winner = null;
    }

    @Test
    public void testGetWinner() {

        gameServer = new GameServer(lobbyServer, blue, "Test Game",
            RulesetType.Wizard, "", false);
        gameServer.addPlayer(white);
        gameServer.addPlayer(orange);
        gameServer.addPlayer(brown);

        wizardServerRuleset = new ServerWizard(gameServer);
        wizardServerRuleset.addPlayerToGame("Mr. Blue");
        wizardServerRuleset.addPlayerToGame("Mr. White");
        wizardServerRuleset.addPlayerToGame("Mr. Orange");
        wizardServerRuleset.addPlayerToGame("Mr. Brown");
        wizardServerRuleset.setPoints(wizardServerRuleset.getPlayerState("Mr.
            Blue"), 80);
        wizardServerRuleset.setPoints(wizardServerRuleset.getPlayerState("Mr.
            White"), 200);
        wizardServerRuleset.setPoints(wizardServerRuleset.getPlayerState("Mr.
            Orange"), 130);
        wizardServerRuleset.setPoints(wizardServerRuleset.getPlayerState("Mr.
            Brown"), 240);
        wizardServerRuleset.setGamePhase(GamePhase.Ending);
        wizardServerRuleset.calculateRoundOutcome();

        assertTrue(wizardServerRuleset.getWinner().equals("Mr. Brown"));

        inputList = blue.getServerInput();
        comObject = (ComRuleset) inputList.get(1);
        endMsg = (MsgGameEnd) comObject.getRulesetMessage();
        winner = endMsg.getWinner();
        assertEquals("Nachricht an Blue", "Mr. Brown", winner);

        inputList = white.getServerInput();
        comObject = (ComRuleset) inputList.get(1);
        endMsg = (MsgGameEnd) comObject.getRulesetMessage();
        winner = endMsg.getWinner();
        assertEquals("Nachricht an White", "Mr. Brown", winner);

        inputList = orange.getServerInput();
        comObject = (ComRuleset) inputList.get(1);
        endMsg = (MsgGameEnd) comObject.getRulesetMessage();
        winner = endMsg.getWinner();
        assertEquals("Nachricht an Orange", "Mr. Brown", winner);

        inputList = brown.getServerInput();
        comObject = (ComRuleset) inputList.get(1);
        endMsg = (MsgGameEnd) comObject.getRulesetMessage();
        winner = endMsg.getWinner();
        assertEquals("Nachricht an Brown", "Mr. Brown", winner);
    }

```

```
}
```

---

## 4.4 QuitPlayer

---

```
ackage Server;

import static org.junit.Assert.*;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import test.TestGameServer;
import test.TestLobbyServer;
import test.TestPlayer;

import ComObjects.*;
import ComObjects.ComWarning;
import Ruleset.RulesetType;

public class QuitGameTest {

    TestLobbyServer lobby;

    TestPlayer player1;

    TestPlayer player2;
    TestPlayer player3;
    TestPlayer player4;

    TestGameServer game;

    ComClientQuit quit;

    @Before
    public void setUp() throws Exception {
        lobby = new TestLobbyServer();

        player1 = new TestPlayer(lobby, null, null);
        player1.setName("MrBlue");
        lobby.addPlayer(player1);

        player2 = new TestPlayer(lobby, null, null);
        player2.setName("MrWhite");
        player3 = new TestPlayer(lobby, null, null);
        player3.setName("MrPink");
        player4 = new TestPlayer(lobby, null, null);
        player4.setName("MrRed");

        game = new TestGameServer(lobby, player1, "MrBluesGame",
            RulesetType.Hearts, null, false);

        game.addPlayer(player2);
```

```

        game.addPlayer(player3);
        game.addPlayer(player4);

        quit = new ComClientQuit();
    }

    @After
    public void tearDown() throws Exception {
        lobby = null;
        player1 = null;
        player2 = null;
        player3 = null;
        player4 = null;
        game = null;
    }

    @Test
    public void testPlayerQuitGame() throws IOException{

        player1.changeServer(game);
        assertTrue(game.initLobby().getPlayerList().contains(player1.getName()));

        ComInitLobby initLobby = lobby.initLobby();
        ComWarning warning = new ComWarning("Ein Spieler hat das Spiel
            verlassen");

        player1.injectComObject(quit);

        assertFalse(lobby.initLobby().getGameList().contains(game));
        assertTrue(lobby.initLobby().getPlayerList().contains(player1.getName()));
        assertTrue(lobby.initLobby().getPlayerList().contains(player1.getName()));
        assertTrue(lobby.initLobby().getPlayerList().contains(player1.getName()));
        assertTrue(lobby.initLobby().getPlayerList().contains(player1.getName()));

        assertTrue(player1.getServerInput().get(0).getClass() ==
            initLobby.getClass());
        assertTrue(player1.getServerInput().get(1).getClass() ==
            warning.getClass());
        assertTrue(player2.getServerInput().get(0).getClass() ==
            initLobby.getClass());
        assertTrue(player2.getServerInput().get(1).getClass() ==
            warning.getClass());
        assertTrue(player3.getServerInput().get(0).getClass() ==
            initLobby.getClass());
        assertTrue(player3.getServerInput().get(1).getClass() ==
            warning.getClass());
        assertTrue(player4.getServerInput().get(0).getClass() ==
            initLobby.getClass());
        assertTrue(player4.getServerInput().get(1).getClass() ==
            warning.getClass());
    }
}

```

---

## 4.5 Chat

```

package chat;

import static org.junit.Assert.*;

import org.junit.Test;
import org.junit.After;

```

---

```

import org.junit.Before;

import testKlassen.TestMessageListenerThread;
import testKlassen.TestObserver;
import Client.MessageListenerThread;
import Client.ClientModel;
import ComObjects.ComChatMessage;

public class ClientModelChatTest {

    ComChatMessage testMessage;

    ClientModel testModel;

    TestObserver testObserver;

    TestMessageListenerThread testNetIO;

    String testText;

    @Before
    public void setUp() {
        testNetIO = new TestMessageListenerThread();
        testObserver = new TestObserver();
        testMessage = new ComChatMessage("Hello Test!");
        testModel = new ClientModel((MessageListenerThread) testNetIO);
        testNetIO.setModel(testModel);
        testModel.addObserver(testObserver);
    }

    @After
    public void tearDown() {
        testNetIO = null;
        testMessage = null;
        testModel = null;
        testObserver = null;
    }

    @Test
    public void testSendChatMessage() {
        String inputText = "Hello Test!";
        testModel.sendChatMessage(inputText);
        testText = ((ComChatMessage)
            testNetIO.getModelInput().get(0)).getChatMessage();
        assertEquals("Vergleich der gesendeten Chatnachrichten", testText,
            inputText);
    }

    @Test
    public void testReceiveChatMessage() {
        testNetIO.injectComObject(testMessage);
        assertTrue("Vergleich der empfangenen Chatnachrichten",
            testObserver.getChatMessage().equals(testMessage.getChatMessage()));
    }
}

```

---

```

package chat;

import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

```



```
import testKlassen.TestPlayer;
import Server.LobbyServer;
import ComObjects.ComChatMessage;

public class LobbyServerChatTest {

    ComChatMessage testMessage;

    LobbyServer testServer;

    TestPlayer player1;

    TestPlayer player2;

    TestPlayer player3;

    String testText1;

    String testText2;

    String testText3;

    @Before
    public void setUp() {
        testMessage = new ComChatMessage("Hello Test!");
        testServer = new LobbyServer();
        player1 = new TestPlayer(testServer, null, null);
        player2 = new TestPlayer(testServer, null, null);
        player3 = new TestPlayer(testServer, null, null);
    }

    @After
    public void tearDown() {
        testMessage = null;
        testServer = null;
        player1 = null;
        player2 = null;
        player3 = null;
        testText1 = null;
        testText2 = null;
        testText3 = null;
    }

    @Test
    public void testReceiveMessagePlayerComChatMessage() {
        String messageToMatch = testMessage.getChatMessage();
        testServer.addPlayer(player1);
        testServer.addPlayer(player2);
        testServer.addPlayer(player3);
        player1.injectComObject(testMessage);
        testText1 = ((ComChatMessage)
            player1.getServerInput()).getChatMessage();
        testText2 = ((ComChatMessage)
            player2.getServerInput()).getChatMessage();
        testText3 = ((ComChatMessage)
            player3.getServerInput()).getChatMessage();
        assertEquals("Nachricht an Spieler 1", messageToMatch, testText1);
        assertEquals("Nachricht an Spieler 2", messageToMatch, testText2);
        assertEquals("Nachricht an Spieler 3", messageToMatch, testText3);
    }
}
```

## 5 Implementierungsplan

Es werden für jeden Milestone die einzelnen Arbeitspakete angegeben. Die angegebenen Klassen werden nicht sofort vollständig implementiert, sondern mit den vom Arbeitspaket und Milestone verlangten Funktionen ausgestattet.

### 5.1 Milestone 1

Für den ersten Milestone werden folgende Funktionen angestrebt:

Der Nutzer kann sich im Login-Fenster anmelden und die Lobby betreten. Er kann ein Spiel erstellen und offenen Spielen beitreten. Das wird in der Lobby angezeigt, man gelangt jedoch noch nicht ins Wartefenster. Nebenläufig dazu wird die Datenschicht der Regelwerke implementiert.

- View(Login+Lobby) Dauer: 8 Std.  
Klassen: Login, Lobby, Warning, ClientController
- Client(Login) Dauer 8 Std.  
Klassen: ClientMain, ClientModel, MessageListener Thread, ClientState, ViewNotification
- Server(Login) Dauer 16 Std.  
Klassen: Server, ServerMain, LobbyServer, Player, ClientListenerThread, ComObject, ComLoginRequest, ComClientQuit, ComServerAcknowledgement, ComWarning
- Ruleset(Daten) Dauer 20 Std.  
Klassen: Card, Colour, HeartsCard, WizCard, OtherData, WizData, HeartsData, GameClientUpdate, GameState, PlayerState, RulesetType
- Client(Lobby) Dauer 8 Std.  
Klassen: ClientModel
- Server(Lobby) Dauer 8 Std.  
Klassen: LobbyServer, ComChatMessage, ComLobbyUpdateGamelist, ComJoinRequest, ComInitLobby, ComUpdatePlayerlist
- View(Create+Join) Dauer 8 Std.  
Klassen: Password, CreateGame, ClientController
- Client(Create+Join) Dauer 8 Std.  
Klassen: ClientModel
- Server(Create+Join) Dauer 12 Std.  
Klassen: LobbyServer, ComJoinRequest, ComCreateGameRequest

### 5.2 Milestone 2

Für den zweiten Milestone werden folgende Funktionen angestrebt:

Beim Beitreten oder Erstellen eines Spiels gelangt man ins Wartefenster. Der Spielleiter kann hier Spieler entfernen. Diese gelangen zurück in die Lobby. Das Spiel kann noch nicht gestartet werden, aber das Regelwerk wird bereits serverseitig implementiert.

- Ruleset(Wizard-Server) Dauer 30 Std.  
Klassen: ServerRuleset, ServerWizard, RulesetMessage, MsgCard, MsgCardRequest, MsgGameEnd, MsgNumber, MsgNumberRequest, MsgSelection, MsgSelectionRequest, MsgUser
- View(GameLobby) Dauer 8 Std.  
Klassen: GameLobby, ClientController

- Client(GameLobby) Dauer 8 Std.  
Klassen: ClientModel
- Server(GameLobby) Dauer 10 Std.  
Klassen: LobbyServer, GameServer, ComBeenKicked, ComClientLeave, ComInitGameLobby, ComKickPlayerRequest, ComStartGame
- View(Game) Dauer 20 Std.  
Klassen: Game, GamePanel, OtherPlayer, OwnHand, ViewCard, DrawDeck, DiscardPile, ScoreWindow, ClientController
- Client(Game) Dauer 14 Std.  
Klassen: ClientModel

### 5.3 Milestone 3

Für den dritten Milestone werden folgende Funktionen angestrebt:  
Es kann schon eine vollständige Partie Wizard gespielt werden.

- Server(Game) Dauer 4 Std.  
Klassen: GameServer, ComStartGame, ComRuleset, ComGameEnd
- Ruleset(Wizard-Client) Dauer 12 Std.  
Klassen: ClientWizard
- View(WizardWindows) Dauer 4 Std.  
Klassen: ChooseItem, InputNumber, ClientController
- Client(Wizard) Dauer 6 Std.  
Klassen: ClientModel
- View(HeartsWindows) Dauer 2 Std.  
Klassen: ChooseCards, ClientController
- Client(Hearts) Dauer 4 Std.  
Klassen: ClientModel
- Ruleset(Hearts-Server) Dauer 16 Std.  
Klassen: ServerHearts

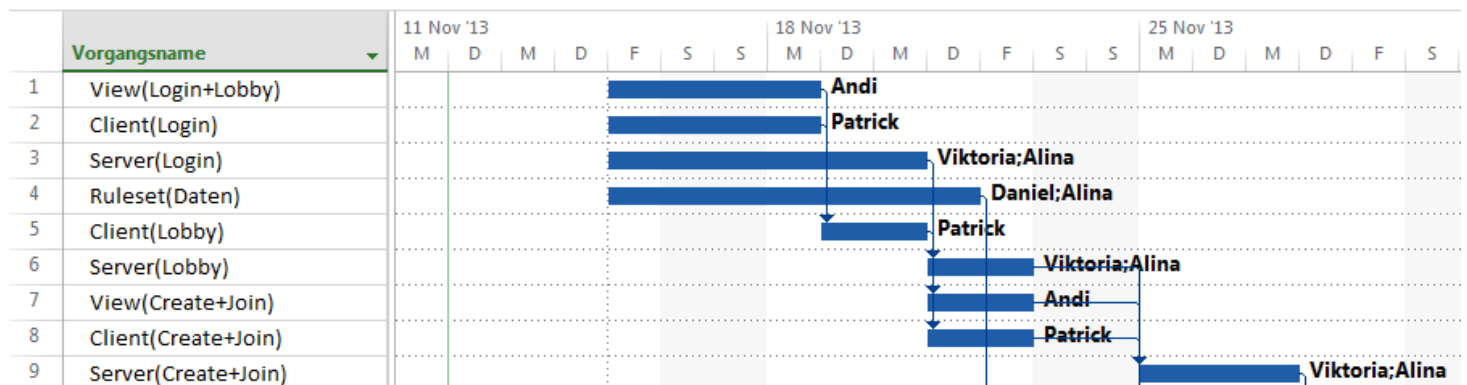
### 5.4 Finale Version

Die finale Version enthält die volle Funktionalität des Programs. Es können also sowohl Wizard als auch Hearts gespielt werden.

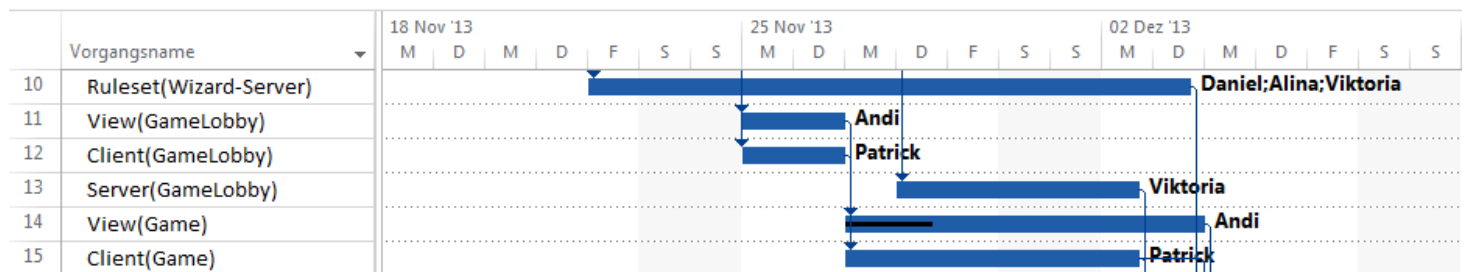
- Ruleset(Hearts-Client) Dauer 10Std.  
Klassen: ClientHearts
- ViewPolishing(evtl Tests) Dauer 10Std  
Verbesserungen an der bisherigen Implementierung. Gegebenfalls Schreiben von zusätzlichen Tests
- ClientPolishing(evtl Tests) Dauer 10Std  
Verbesserungen an der bisherigen Implementierung. Gegebenfalls Schreiben von zusätzlichen Tests
- ServerPolishing(evtl Tests) Dauer 10Std  
Verbesserungen an der bisherigen Implementierung. Gegebenfalls Schreiben von zusätzlichen Tests
- RulesetPolishing(evtl Tests) Dauer 10Std  
Verbesserungen an der bisherigen Implementierung. Gegebenfalls Schreiben von zusätzlichen Tests

## 5.5 Gantt-Diagramme

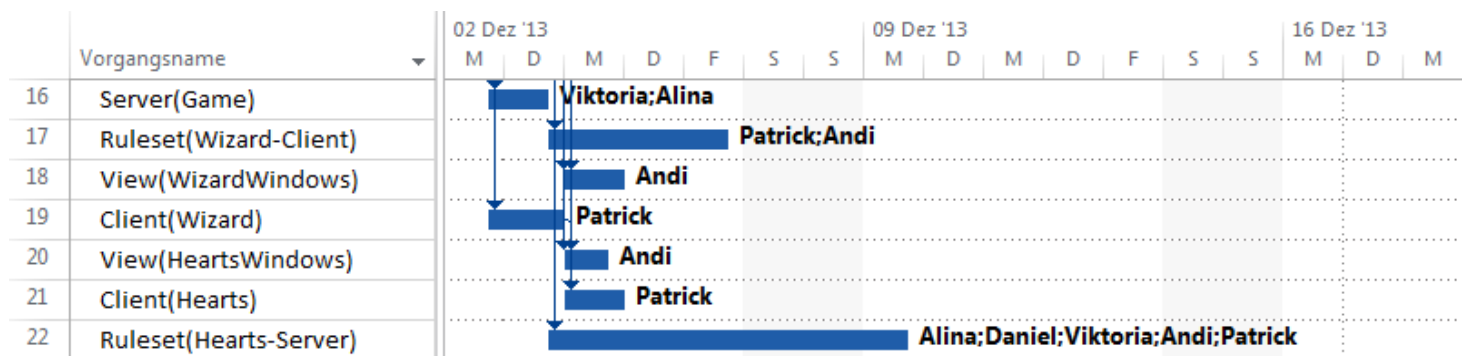
Milestone 1:



Milestone 2:



Milestone 3:



Finale Version:

	Vorgangsname	09 Dez '13						16 Dez '13						
		M	D	M	D	F	S	S	M	D	M	D	F	S
23	Ruleset(Hearts-Client)	Alina;Daniel;Viktoria;Andi;Patrick												
24	ViewPolishing (evtl Tests)												Andi	
25	ClientPolishing (evtl Tests)												Patrick	
26	ServerPolishing (evtl Tests)												Viktoria	
27	RulesetPolishing (evtl Tests)												Alina;Daniel	

## Index

- addCard
  - Ruleset::PlayerState, 87
- addChatMessageListener
  - Client::View::GameLobby, 24
  - Client::View::Lobby, 27
- addConnectButtonListener
  - Client::View::Login, 29
- addCreateButtonListener
  - Client::View::CreateGame, 21
- addGameServer
  - Server::LobbyServer, 106
- addHostButtonListener
  - Client::View::Lobby, 27
- addJoinButtonListener
  - Client::View::Lobby, 27
  - Client::View::Password, 31
- addLanguageSelectionListener
  - Client::View::Login, 29
- addLeaveButtonListener
  - Client::View::CreateGame, 21
  - Client::View::GameLobby, 24
  - Client::View::Lobby, 27
- addName
  - Server::LobbyServer, 105
- addPanelMouseListener
  - Client::View::CreateGame, 20
- addPlayer
  - Server::GameServer, 101
  - Server::Server, 110
- addPlayerToGame
  - Ruleset::GameState, 80
  - Ruleset::ServerRuleset, 93
- addRemoveButtonListener
  - Client::View::GameLobby, 24
- addRulesetSelectionListener
  - Client::View::CreateGame, 20
- addStartButtonListener
  - Client::View::GameLobby, 23
- areValidChosenCards
  - Ruleset::ClientHearts, 70
  - Ruleset::ServerHearts, 89
- broadcast
  - Ruleset::ServerRuleset, 93
  - Server::Server, 110
- broadcastRulesetMessage
  - Server::GameServer, 102
- Card, 69
- changeServer
  - Server::Player, 109
- ChooseCards, 18
- ChooseItem, 19
- Client::ClientMain
  - main, 9
- Client::ClientModel
  - ClientModel, 10
  - createConnection, 17
  - getChooseCards, 14
  - getChooseItems, 15
  - getLanguage, 13
  - getLobbyGamelist, 12
  - getOtherPlayerData, 13
  - getOwnHand, 13
  - getOwnScore, 13
  - getPlayedCards, 13
  - getPlayerCount, 14
  - getPlayerlist, 12
  - getRulesets, 17
  - getWarningText, 17
  - getWindowText, 14
  - giveChosenCards, 14
  - giveChosenItem, 15
  - giveInputNumber, 15
  - hostGame, 14
  - informView, 16
  - initGame, 16
  - joinGame, 16
  - kickPlayer, 13
  - leaveWindow, 11
  - makeMove, 16
  - openChooseCards, 15
  - openChooseItem, 15
  - openInputNumber, 16
  - receiveMessage, 11, 12
  - send, 14
  - sendChatMessage, 16
  - setLanguage, 13
  - startGame, 16
- Client::MessageListenerThread
  - startConnection, 18
- Client::View::ChooseCards
  - update, 19
- Client::View::ChooseItem
  - update, 19
- Client::View::CreateGame
  - addCreateButtonListener, 21
  - addLeaveButtonListener, 21
  - addPanelMouseListener, 20
  - addRulesetSelectionListener, 20
  - setLanguage, 21
- Client::View::Game
  - Game, 22
  - makeTrickGameBoard, 22
  - update, 22
- Client::View::GameLobby
  - addChatMessageListener, 24
  - addLeaveButtonListener, 24
  - addRemoveButtonListener, 24
  - addStartButtonListener, 23
  - setLanguage, 24

- update, 24
- Client::View::InputNumber
  - update, 26
- Client::View::Lobby
  - addChatMessageListener, 27
  - addHostButtonListener, 27
  - addJoinButtonListener, 27
  - addLeaveButtonListener, 27
  - setLanguage, 28
  - update, 28
- Client::View::Login
  - addConnectButtonListener, 29
  - addLanguageSelectionListener, 29
  - setLanguage, 29
  - update, 29
- Client::View::Password
  - addJoinButtonListener, 31
  - setLanguage, 31
  - update, 31
- Client::View::ScoreWindow
  - update, 32
- Client::View::ViewCard
  - getID, 33
  - ViewCard, 32
- Client::View::Warning
  - setText, 33
  - update, 33
- ClientController, 8
- ClientHearts, 69
  - Ruleset::ClientHearts, 70
- ClientMain, 9
- ClientModel, 9
  - Client::ClientModel, 10
- ClientRuleset, 71
  - Ruleset::ClientRuleset, 71
- ClientState, 17
- ClientWizard, 74
  - Ruleset::ClientWizard, 75
- Colour, 77
- ComBeenKicked, 34
  - ComObjects::ComBeenKicked, 35
- ComChatMessage, 35
  - ComObjects::ComChatMessage, 36
- ComClientLeave, 36
- ComClientQuit, 37
- ComCreateGameRequest, 38
  - ComObjects::ComCreateGameRequest, 38
- ComGameEnd, 40
  - ComObjects::ComGameEnd, 40
- ComInitGameLobby, 41
  - ComObjects::ComInitGameLobby, 41
- ComInitLobby, 42
  - ComObjects::ComInitLobby, 43
- ComJoinRequest, 44
  - ComObjects::ComJoinRequest, 44
- ComKickPlayerRequest, 46
  - ComObjects::ComKickPlayerRequest, 46
- ComLobbyUpdateGamelist, 47
  - ComObjects::ComLobbyUpdateGamelist, 48
- ComLoginRequest, 48
  - ComObjects::ComLoginRequest, 49
- ComObject, 50
  - ComObjects::ComObject, 50
- ComObjects::ComBeenKicked
  - ComBeenKicked, 35
  - getMessage, 35
  - process, 35
- ComObjects::ComChatMessage
  - ComChatMessage, 36
  - getChatMessage, 36
  - process, 36
- ComObjects::ComClientLeave
  - process, 37
- ComObjects::ComClientQuit
  - process, 37, 38
- ComObjects::ComCreateGameRequest
  - ComCreateGameRequest, 38
  - getGameName, 39
  - getPassword, 39
  - getRuleset, 39
  - hasPassword, 39
  - process, 39
- ComObjects::ComGameEnd
  - ComGameEnd, 40
  - getPoints, 40
  - getWinner, 40
  - process, 41
- ComObjects::ComInitGameLobby
  - ComInitGameLobby, 41
  - getPlayerList, 42
  - process, 42
- ComObjects::ComInitLobby
  - ComInitLobby, 43
  - getGameList, 43
  - getPlayerList, 43
  - process, 43
- ComObjects::ComJoinRequest
  - ComJoinRequest, 44
  - gameMasterName, 46
  - getGameMasterName, 44
  - process, 44, 46
- ComObjects::ComKickPlayerRequest
  - ComKickPlayerRequest, 46
  - getPlayerName, 47
  - process, 47
- ComObjects::ComLobbyUpdateGamelist
  - ComLobbyUpdateGamelist, 48
  - getGameServer, 48
  - isRemoveFlag, 48
  - process, 48
- ComObjects::ComLoginRequest
  - ComLoginRequest, 49
  - getPlayerName, 49
  - process, 49
- ComObjects::ComObject
  - process, 50
- ComObjects::ComRuleset

- ComRuleset, 51
  - getRulesetMessage, 51
  - process, 51
- ComObjects::ComServerAcknowledgement
  - process, 52
- ComObjects::ComStartGame
  - process, 53
- ComObjects::ComUpdatePlayerlist
  - ComUpdatePlayerlist, 53
  - getPlayerName, 55
  - isRemoveFlag, 55
  - process, 55
- ComObjects::ComWarning
  - ComWarning, 56
  - getWarning, 56
  - process, 56
- ComObjects::MsgCard
  - getCard, 57
  - MsgCard, 57
  - visit, 57
- ComObjects::MsgCardRequest
  - visit, 58
- ComObjects::MsgGameEnd
  - getWinnerName, 59
  - MsgGameEnd, 59
  - visit, 59
- ComObjects::MsgMultiCards
  - getCardList, 60
  - MsgMultiCards, 60
  - visit, 60
- ComObjects::MsgMultiCardsRequest
  - getCount, 61
  - MsgMultiCardsRequest, 61
  - visit, 61, 62
- ComObjects::MsgNumber
  - getNumber, 62
  - MsgNumber, 62
  - visit, 63
- ComObjects::MsgNumberRequest
  - visit, 63, 64
- ComObjects::MsgSelection
  - getSelection, 64
  - MsgSelection, 64
  - visit, 64, 66
- ComObjects::MsgSelectionRequest
  - visit, 66
- ComObjects::MsgUser
  - getGameClientUpdate, 67
  - MsgUser, 67
  - visit, 67, 68
- ComObjects::RulesetMessage
  - visit, 68
- ComRuleset, 51
  - ComObjects::ComRuleset, 51
- ComServerAcknowledgement, 52
- ComStartGame, 52
- ComUpdatePlayerlist, 53
  - ComObjects::ComUpdatePlayerlist, 53
- ComWarning, 55
  - ComObjects::ComWarning, 56
- createConnection
  - Client::ClientModel, 17
- createDeck
  - Ruleset::ServerRuleset, 91
- CreateGame, 19
- dealCards
  - Ruleset::GameState, 82
  - Ruleset::ServerRuleset, 94
- DiscardPile, 21
- DrawDeck, 21
- Game, 21
  - Client::View::Game, 22
- GameClientUpdate, 77
  - Ruleset::GameClientUpdate, 77
- GameLobby, 23
- gameMasterName
  - ComObjects::ComJoinRequest, 46
- GamePanel, 25
- GamePhase, 79
- GameServer, 100
  - Server::GameServer, 101
- GameServerRepresentation, 104
  - Server::GameServerRepresentation, 104
- GameState, 79
  - Ruleset::GameState, 80
- generateGameClientUpdate
  - Ruleset::ServerHearts, 89
  - Ruleset::ServerRuleset, 95
  - Ruleset::ServerWizard, 98
- getAnnouncedTricks
  - Ruleset::WizData, 100
- getCard
  - ComObjects::MsgCard, 57
- getCardList
  - ComObjects::MsgMultiCards, 60
- getCardsLeftInDeck
  - Ruleset::GameState, 81
- getChatMessage
  - ComObjects::ComChatMessage, 36
- getChooseCards
  - Client::ClientModel, 14
- getChooseItems
  - Client::ClientModel, 15
- getColour
  - Ruleset::Card, 69
  - Ruleset::HeartsCard, 84
  - Ruleset::WizardCard, 99
- getCount
  - ComObjects::MsgMultiCardsRequest, 61
- getCurrentPlayer
  - Ruleset::ClientRuleset, 73
  - Ruleset::GameClientUpdate, 78
  - Ruleset::GameState, 81
  - Ruleset::ServerRuleset, 92
- getEndingPoints



- Ruleset::ServerHearts, 88
- getFirstPlayer
  - Ruleset::GameState, 80
  - Ruleset::ServerRuleset, 92
- getGameClientUpdate
  - ComObjects::MsgUser, 67
- getGameList
  - ComObjects::ComInitLobby, 43
- getGameMasterName
  - ComObjects::ComJoinRequest, 44
- getGameName
  - ComObjects::ComCreateGameRequest, 39
- getGamePhase
  - Ruleset::ClientRuleset, 72
  - Ruleset::ServerRuleset, 91
- getGameServer
  - ComObjects::ComLobbyUpdateGamelist, 48
- getHand
  - Ruleset::PlayerState, 87
- getID
  - Client::View::ViewCard, 33
- getLanguage
  - Client::ClientModel, 13
- getLobbyGamelist
  - Client::ClientModel, 12
- getMaxPlayers
  - Ruleset::ClientRuleset, 72
  - Ruleset::ServerRuleset, 91
- getMessage
  - ComObjects::ComBeenKicked, 35
- getMinPlayers
  - Ruleset::ClientRuleset, 72
  - Ruleset::ServerRuleset, 91
- getName
  - Ruleset::PlayerState, 86
  - Server::Player, 109
- getNumber
  - ComObjects::MsgNumber, 62
- getNumberOfPlayedCards
  - Ruleset::GameState, 82
- getNumberOfTricks
  - Ruleset::OtherData, 85
- getOtherData
  - Ruleset::PlayerState, 87
  - Ruleset::ServerRuleset, 92
- getOtherPlayerData
  - Client::ClientModel, 13
  - Ruleset::ClientRuleset, 73
  - Ruleset::GameClientUpdate, 78
- getOwnData
  - Ruleset::ClientRuleset, 72
  - Ruleset::GameClientUpdate, 78
- getOwnHand
  - Client::ClientModel, 13
  - Ruleset::ClientRuleset, 72
  - Ruleset::GameClientUpdate, 78
- getOwnScore
  - Client::ClientModel, 13
- getPassword
  - ComObjects::ComCreateGameRequest, 39
- getPlayedCards
  - Client::ClientModel, 13
  - Ruleset::GameClientUpdate, 78
  - Ruleset::GameState, 81
- getPlayerCards
  - Ruleset::GameState, 82
  - Ruleset::ServerRuleset, 93
- getPlayerCount
  - Client::ClientModel, 14
- getPlayerList
  - ComObjects::ComInitGameLobby, 42
  - ComObjects::ComInitLobby, 43
- getPlayerName
  - ComObjects::ComKickPlayerRequest, 47
  - ComObjects::ComLoginRequest, 49
  - ComObjects::ComUpdatePlayerlist, 55
- getPlayerState
  - Ruleset::GameState, 81
  - Ruleset::ServerRuleset, 93
- getPlayerlist
  - Client::ClientModel, 12
- getPoints
  - ComObjects::ComGameEnd, 40
  - Ruleset::OtherData, 85
  - Ruleset::ServerRuleset, 94
- getRepresentation
  - Server::GameServer, 101
- getRoundNumber
  - Ruleset::GameState, 82
  - Ruleset::ServerRuleset, 91
- getRuleset
  - ComObjects::ComCreateGameRequest, 39
- getRulesetMessage
  - ComObjects::ComRuleset, 51
- getRulesetType
  - Ruleset::ClientRuleset, 72
  - Ruleset::ServerRuleset, 91
- getRulesets
  - Client::ClientModel, 17
- getSelection
  - ComObjects::MsgSelection, 64
- getTrumpCard
  - Ruleset::ClientRuleset, 73
  - Ruleset::GameClientUpdate, 78
  - Ruleset::GameState, 82
- getValue
  - Ruleset::Card, 69
  - Ruleset::HeartsCard, 84
  - Ruleset::WizardCard, 99
- getWarning
  - ComObjects::ComWarning, 56
- getWarningText
  - Client::ClientModel, 17
- getWindowText
  - Client::ClientModel, 14
- getWinner

- ComObjects::ComGameEnd, 40
- getWinnerName
  - ComObjects::MsgGameEnd, 59
- getplayingRounds
  - Ruleset::ServerWizard, 97
- giveACard
  - Ruleset::GameState, 82
  - Ruleset::ServerRuleset, 94
- giveChosenCards
  - Client::ClientModel, 14
- giveChosenItem
  - Client::ClientModel, 15
- giveInputNumber
  - Client::ClientModel, 15
- handleIOException
  - Server::GameServer, 103
  - Server::LobbyServer, 107
  - Server::Server, 110
- hasPassword
  - ComObjects::ComCreateGameRequest, 39
- HeartsCard, 83
  - Ruleset::HeartsCard, 83
- HeartsData, 84
- hostGame
  - Client::ClientModel, 14
- informView
  - Client::ClientModel, 16
- initGame
  - Client::ClientModel, 16
- initLobby
  - Server::GameServer, 103
  - Server::LobbyServer, 107
- InputNumber, 25
- isRemoveFlag
  - ComObjects::ComLobbyUpdateGamelist, 48
  - ComObjects::ComUpdatePlayerlist, 55
- isValidColour
  - Ruleset::ClientWizard, 76
  - Ruleset::ServerWizard, 97
- isValidMove
  - Ruleset::ClientHearts, 70
  - Ruleset::ClientRuleset, 74
  - Ruleset::ClientWizard, 76
  - Ruleset::ServerHearts, 88
  - Ruleset::ServerRuleset, 95
  - Ruleset::ServerWizard, 96
- isValidNumber
  - Ruleset::ServerWizard, 97
- isValidTrickNumber
  - Ruleset::ClientWizard, 76
- joinGame
  - Client::ClientModel, 16
- kickPlayer
  - Client::ClientModel, 13
- Language, 26
- leaveWindow
  - Client::ClientModel, 11
- Lobby, 26
- LobbyServer, 104
- LobbyServer.ClientListenerThread, 107
- Login, 28
- madeTrick
  - Ruleset::OtherData, 85
- main
  - Client::ClientMain, 9
  - Server::ServerMain, 111
- makeMove
  - Client::ClientModel, 16
- makeTrickGameBoard
  - Client::View::Game, 22
- MessageListenerThread, 17
- MsgCard, 56
  - ComObjects::MsgCard, 57
- MsgCardRequest, 58
- MsgGameEnd, 58
  - ComObjects::MsgGameEnd, 59
- MsgMultiCards, 60
  - ComObjects::MsgMultiCards, 60
- MsgMultiCardsRequest, 61
  - ComObjects::MsgMultiCardsRequest, 61
- MsgNumber, 62
  - ComObjects::MsgNumber, 62
- MsgNumberRequest, 63
- MsgSelection, 64
  - ComObjects::MsgSelection, 64
- MsgSelectionRequest, 66
- MsgUser, 67
  - ComObjects::MsgUser, 67
- nextPlayer
  - Ruleset::ServerRuleset, 92
- openChooseCards
  - Client::ClientModel, 15
- openChooseItem
  - Client::ClientModel, 15
- openInputNumber
  - Client::ClientModel, 16
- OtherData, 84
- OtherPlayer, 30
- OwnHand, 30
- Password, 30
- password
  - Server::GameServer, 103
- playCard
  - Ruleset::GameState, 83
  - Ruleset::ServerRuleset, 95
- Player, 108
  - Server::Player, 108
- PlayerState, 86
  - Ruleset::PlayerState, 86
- playingRounds

- Ruleset::ServerWizard, 98
- process
  - ComObjects::ComBeenKicked, 35
  - ComObjects::ComChatMessage, 36
  - ComObjects::ComClientLeave, 37
  - ComObjects::ComClientQuit, 37, 38
  - ComObjects::ComCreateGameRequest, 39
  - ComObjects::ComGameEnd, 41
  - ComObjects::ComInitGameLobby, 42
  - ComObjects::ComInitLobby, 43
  - ComObjects::ComJoinRequest, 44, 46
  - ComObjects::ComKickPlayerRequest, 47
  - ComObjects::ComLobbyUpdateGamelist, 48
  - ComObjects::ComLoginRequest, 49
  - ComObjects::ComObject, 50
  - ComObjects::ComRuleset, 51
  - ComObjects::ComServerAcknowledgement, 52
  - ComObjects::ComStartGame, 53
  - ComObjects::ComUpdatePlayerlist, 55
  - ComObjects::ComWarning, 56
- receiveMessage
  - Client::ClientModel, 11, 12
  - Server::GameServer, 102, 103
  - Server::LobbyServer, 106, 107
  - Server::Server, 110
- removeCard
  - Ruleset::PlayerState, 87
- removeGameServer
  - Server::LobbyServer, 106
- removeName
  - Server::LobbyServer, 105
- removePlayer
  - Server::GameServer, 101
  - Server::Server, 110
- removeTricks
  - Ruleset::OtherData, 85
- resolveMessage
  - Ruleset::ClientHearts, 70
  - Ruleset::ClientRuleset, 73
  - Ruleset::ClientWizard, 76
  - Ruleset::ServerHearts, 88
  - Ruleset::ServerRuleset, 93, 94
  - Ruleset::ServerWizard, 97
- Ruleset::Card
  - getColour, 69
  - getValue, 69
- Ruleset::ClientHearts
  - areValidChosenCards, 70
  - ClientHearts, 70
  - isValidMove, 70
  - resolveMessage, 70
- Ruleset::ClientRuleset
  - ClientRuleset, 71
  - getCurrentPlayer, 73
  - getGamePhase, 72
  - getMaxPlayers, 72
  - getMinPlayers, 72
  - getOtherPlayerData, 73
  - getOwnData, 72
  - getOwnHand, 72
  - getRulesetType, 72
  - getTrumpCard, 73
  - isValidMove, 74
  - resolveMessage, 73
  - send, 74
- Ruleset::ClientWizard
  - ClientWizard, 75
  - isValidColour, 76
  - isValidMove, 76
  - isValidTrickNumber, 76
  - resolveMessage, 76
- Ruleset::GameClientUpdate
  - GameClientUpdate, 77
  - getCurrentPlayer, 78
  - getOtherPlayerData, 78
  - getOwnData, 78
  - getOwnHand, 78
  - getPlayedCards, 78
  - getTrumpCard, 78
- Ruleset::GameState
  - addPlayerToGame, 80
  - dealCards, 82
  - GameState, 80
  - getCardsLeftInDeck, 81
  - getCurrentPlayer, 81
  - getFirstPlayer, 80
  - getNumberOfPlayedCards, 82
  - getPlayedCards, 81
  - getPlayerCards, 82
  - getPlayerState, 81
  - getRoundNumber, 82
  - getTrumpCard, 82
  - giveACard, 82
  - playCard, 83
  - setCurrentPlayer, 81
  - setFirstPlayer, 80
  - setTrumpCard, 81
- Ruleset::HeartsCard
  - getColour, 84
  - getValue, 84
  - HeartsCard, 83
- Ruleset::OtherData
  - getNumberOfTricks, 85
  - getPoints, 85
  - madeTrick, 85
  - removeTricks, 85
  - setPoints, 85
- Ruleset::PlayerState
  - addCard, 87
  - getHand, 87
  - getName, 86
  - getOtherData, 87
  - PlayerState, 86
  - removeCard, 87
- Ruleset::ServerHearts
  - areValidChosenCards, 89

- generateGameClientUpdate, 89
- getEndingPoints, 88
- isValidMove, 88
- resolveMessage, 88
- Ruleset::ServerRuleset
  - addPlayerToGame, 93
  - broadcast, 93
  - createDeck, 91
  - dealCards, 94
  - generateGameClientUpdate, 95
  - getCurrentPlayer, 92
  - getFirstPlayer, 92
  - getGamePhase, 91
  - getMaxPlayers, 91
  - getMinPlayers, 91
  - getOtherData, 92
  - getPlayerCards, 93
  - getPlayerState, 93
  - getPoints, 94
  - getRoundNumber, 91
  - getRulesetType, 91
  - giveACard, 94
  - isValidMove, 95
  - nextPlayer, 92
  - playCard, 95
  - resolveMessage, 93, 94
  - send, 93
  - ServerRuleset, 90
  - setCurrentPlayer, 92
  - setFirstPlayer, 92
  - setGamePhase, 91
  - setPoints, 94
  - setTrumpCard, 95
- Ruleset::ServerWizard
  - generateGameClientUpdate, 98
  - getplayingRounds, 97
  - isValidColour, 97
  - isValidMove, 96
  - isValidNumber, 97
  - playingRounds, 98
  - resolveMessage, 97
  - setPlayingRounds, 96
- Ruleset::WizData
  - getAnnouncedTricks, 100
  - setAnnouncedTricks, 100
- Ruleset::WizardCard
  - getColour, 99
  - getValue, 99
  - WizardCard, 99
- RulesetMessage, 68
- RulesetType, 87
- run
  - Server::Player, 108
- ScoreWindow, 31
- send
  - Client::ClientModel, 14
  - Ruleset::ClientRuleset, 74
  - Ruleset::ServerRuleset, 93
  - Server::Player, 108
  - sendChatMessage
    - Client::ClientModel, 16
  - sendRulesetMessage
    - Server::GameServer, 102
  - sendToPlayer
    - Server::Server, 110
- Server, 109
- Server::GameServer
  - addPlayer, 101
  - broadcastRulesetMessage, 102
  - GameServer, 101
  - getRepresentation, 101
  - handleIOException, 103
  - initLobby, 103
  - password, 103
  - receiveMessage, 102, 103
  - removePlayer, 101
  - sendRulesetMessage, 102
- Server::GameServerRepresentation
  - GameServerRepresentation, 104
- Server::LobbyServer
  - addGameServer, 106
  - addName, 105
  - handleIOException, 107
  - initLobby, 107
  - receiveMessage, 106, 107
  - removeGameServer, 106
  - removeName, 105
- Server::Player
  - changeServer, 109
  - getName, 109
  - Player, 108
  - run, 108
  - send, 108
  - setName, 109
- Server::Server
  - addPlayer, 110
  - broadcast, 110
  - handleIOException, 110
  - receiveMessage, 110
  - removePlayer, 110
  - sendToPlayer, 110
- Server::ServerMain
  - main, 111
- ServerHearts, 87
- ServerMain, 111
- ServerRuleset, 89
  - Ruleset::ServerRuleset, 90
- ServerWizard, 95
- setAnnouncedTricks
  - Ruleset::WizData, 100
- setCurrentPlayer
  - Ruleset::GameState, 81
  - Ruleset::ServerRuleset, 92
- setFirstPlayer
  - Ruleset::GameState, 80
  - Ruleset::ServerRuleset, 92

- setGamePhase
  - Ruleset::ServerRuleset, 91
- setLanguage
  - Client::ClientModel, 13
  - Client::View::CreateGame, 21
  - Client::View::GameLobby, 24
  - Client::View::Lobby, 28
  - Client::View::Login, 29
  - Client::View::Password, 31
- setName
  - Server::Player, 109
- setPlayingRounds
  - Ruleset::ServerWizard, 96
- setPoints
  - Ruleset::OtherData, 85
  - Ruleset::ServerRuleset, 94
- setText
  - Client::View::Warning, 33
- setTrumpCard
  - Ruleset::GameState, 81
  - Ruleset::ServerRuleset, 95
- startConnection
  - Client::MessageListenerThread, 18
- startGame
  - Client::ClientModel, 16
- update
  - Client::View::ChooseCards, 19
  - Client::View::ChooseItem, 19
  - Client::View::Game, 22
  - Client::View::GameLobby, 24
  - Client::View::InputNumber, 26
  - Client::View::Lobby, 28
  - Client::View::Login, 29
  - Client::View::Password, 31
  - Client::View::ScoreWindow, 32
  - Client::View::Warning, 33
- ViewCard, 32
  - Client::View::ViewCard, 32
- ViewNotification, 34
- visit
  - ComObjects::MsgCard, 57
  - ComObjects::MsgCardRequest, 58
  - ComObjects::MsgGameEnd, 59
  - ComObjects::MsgMultiCards, 60
  - ComObjects::MsgMultiCardsRequest, 61, 62
  - ComObjects::MsgNumber, 63
  - ComObjects::MsgNumberRequest, 63, 64
  - ComObjects::MsgSelection, 64, 66
  - ComObjects::MsgSelectionRequest, 66
  - ComObjects::MsgUser, 67, 68
  - ComObjects::RulesetMessage, 68
- Warning, 33
- WizData, 99
- WizardCard, 98
  - Ruleset::WizardCard, 99