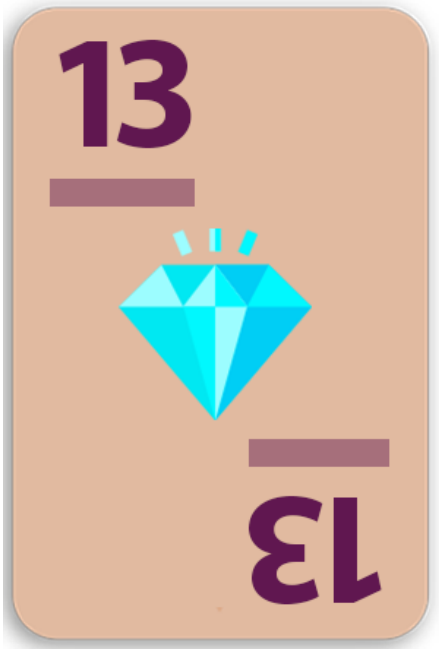


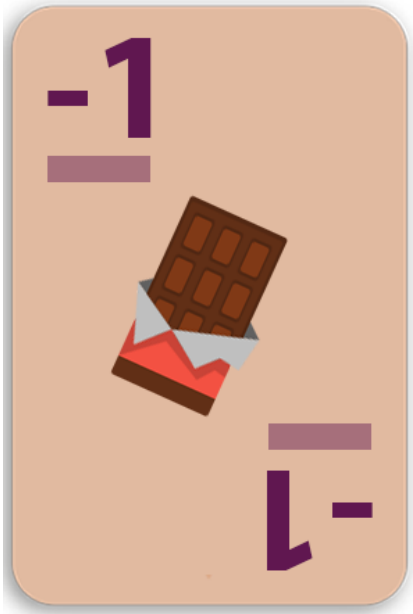


# CABO

Online-Multiplayer-Kartenspiel für
2-4 Spieler

# Aufgabenbereiche - Sophia

- Verbindung zwischen Server und Client (Kommunikation)
- Spielablauf
- Künstliche Intelligenz

# Verbindung zwischen Server und Client

**Server :**

**Spring**

1. Application Klasse
2. Sockethandler Klasse
   - afterConnectionEstablished
   - afterConnectionClosed
   - handleTextMessage
   - sendMessage
3. Gamestate Klasse

# Verbindung zwischen Server und Client

**Client :**
Communicator

- Enthält überschriebene Methoden (onMessage, onOpen etc)
- **Singelton** :
  Somit ist es möglich in allen activities auf das Websocket Object zuzugreifen

**Singelton:**

```
public static Communicator getInstance(Activity activity) {
    if (instance == null) {
        instance = new Communicator(activity);
    }

    return instance;
}
```

**In der ersten Activity, die Verbindung aufbaut:**

```
//connects to server
communicator = Communicator.getInstance(this);
communicator.connectWebSocket();
mWebSocketClient = communicator.getmWebSocketClient();
communicator.setActivity(this);
```

# Verbindung zwischen Server und Client

**Client :**
Communicator

- Enthält überschriebene Methoden (onMessage, onOpen etc)
- **Singelton** :
  Somit ist es möglich in allen activities auf das Websocket Object zuzugreifen

- Zusätzlich im Communicator: bei Android Problem wie bekomme ich dauernd wechselnde Daten von einer Klasse, die keine Activity ist in eine Activity?
  ➡ CommunicatorCallback interface

**Interface und Konstruktor:**

```
public interface CommunicatorCallback extends Serializable {
    public void handleTextMessage(String message) throws JSONException;
}
public Communicator(Activity activity) {
    this.activity = (CommunicatorCallback) activity;
}
```

**Weiterleitung in onMessage im Communicator:**

```
@Override
public void onMessage(String s) {

    try {
        activity.handleTextMessage(s);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

**Jede Activity muss das Interface und die Methode handleTextMessage implementieren:**

```
public class InGameActivity extends AppCompatActivity implements Communicator.CommunicatorCallback {
```

# Verbindung zwischen Server und Client

**Client + Server:**

JSON_command Klasse

- Enthält alle Methoden, die JSON-Objekte „bauen“, welche anschließend gesendet werden

⇩

Nutz Serialisierung und Deserialisierung

```
public static JSONObject useFunctionalitySwap(Card card1, Player player1, Card card2, Player player2) throws JSONException {
    Gson gson= new Gson();

    JSONObject jmsg = new JSONObject();
    JSONObject jsubmsg = new JSONObject();

    jsubmsg.put( name: "card1", gson.toJson(card1));
    jsubmsg.put( name: "player1", gson.toJson(player1));
    jsubmsg.put( name: "card2", gson.toJson(card2));
    jsubmsg.put( name: "player2", gson.toJson(player2));
    jmsg.put( name: "useFunctionalitySwap", jsubmsg);

    return jmsg;
}
```

```
if (jsonObject.has( name: "sendOtherPlayer")) {
    JSONObject js = jsonObject.getJSONObject("sendOtherPlayer");
    if (js.has( name: "player")) {
        String jsonString = js.get("player").toString();
        Gson gson = new Gson();
        Player player = gson.fromJson(jsonString, Player.class);
```

# Spielablauf

- Behandlung aller empfangenen Nachrichten und senden aller Nachrichten
- Logik des Gamestate Objekts: behandelt den gesamten Spielablauf am Server
- **Ab dem Zeitpunkt, in dem ein Spieler den WaitingRoom beitritt, wird dieser Spieler einem freien Gamestate Objekt zugewiesen**
  **➡ ab jetzt werden alle Nachrichten automatisch dem Gamestate Objekt weitergeleitet, dem der Player angehört**

```java
@Override
public void handleTextMessage(WebSocketSession session, TextMessage message)
    throws InterruptedException, IOException {
    JSONObject jsonObject = getMessage(message.getPayload());
    Player currentSender = getPlayerBySessionId(session.getId());

    if (isAssignedToGame(getPlayerBySessionId(session.getId()))) {
        getPlayerBySessionId(session.getId()).getGamestate().handleTextMessage(session, message);
    } else {
```

# Spielablauf

**Im Gamestate:**
Behandlung aller möglichen Nachrichten von chatmessages, über spezifische Spielzüge, interne Statusupdate vom Server, bis hin zu Benachrichtigung über Änderungen von Emojis

**Beispiele:**
- Player Objekte anlegen : Unterscheidung in kein Account und Account
- Kann das Spiel gestartet werden? Sind genug Spieler da ? Hat jeder am Anfang 2 Karten angeschaut und kann somit die erste Runde gestartet werden?
- Festlegen von Spielreihenfolge
- Sicherstellen, dass Spieler diesen Zug gerade überhaupt machen darf

**Beispiele Weiterführung…**
- Beenden eines Zuges: ist das gesamte Spiel aus oder nur eine Runde?
  Neue Runde anfangen ➡ „reset“ auf dem Server für neue Runde
- Client über Veränderungen in anderen Spielerobjekten benachrichtigen
  zB. score ➡ wie viele Punkte haben die anderen Spieler nach der ersten Runde?
- Player Objekte löschen, wenn das Spiel aus ist, aber auch wenn jemand während dem spiel die App beendet/verlässt, sodass die anderen Spieler weiter spielen können
  ➡ User zur „Ursprungs-Activity“ zurückleiten
- Alles zurück setzen

Im SocketHandler: am Ende noch ein bisschen Partylogik

# Künstliche Intelligenz

- Server merkt sich, wenn mit KI gespielt wird.
- Nachrichten werden dann nicht gesendet, sondern an die Methode *handleKI* weitergeleitet
- dann wird anhand der gezogenen Karte entschieden, was die KI macht:
  ➡ zB bei einer sehr niedrigen Karte tauscht sie die gezogenen Karte mit ihrer eigenen
  **! Die KI tauscht die Karte aber nicht mit irgendeiner ihrer Karten, sondern mit der höchsten, die ihr bekannt ist !**
- Sie merkt sich Karten, die sie von sich selbst oder dem realen Spieler kennt in Listen
  ➡ somit kann sie einen klugen Spielzug machen
- Liste der bekannten Karten dated sich auch ab, nachdem der reale Spieler einen Zug beendet hat, zB. realer Spieler tauscht eine seiner Karten aus
  ➡ KI merkt sich, dass sie diese Karte nun nicht mehr kennt
- Damit der reale Spieler auch immer über die Züge der KI informiert wird, wird ein JSON Objekt erstellt und in die *handleTextMessage* Methode gegeben und behandelt
  ➡ **Somit wird simuliert, dass die KI eine Nachricht an den Server schickt.**

```java
public void handleKI(String action) throws IOException {
    if (players != null || players.size() != 0 || players.size() == 1) {
        Player me = players.get("AI");
        if (action.equalsIgnoreCase( anotherString: "nextPlayer")) {
            if (currentPlayerId == me.getId()) {
                KIJsonObject = JSON_commands.sendPickCardKI("");
                try {
                    handleTextMessage( session: null,  message: null);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        if (action.equalsIgnoreCase( anotherString: "decideForMove")) {
```

```java
public JSONObject decideForMove(Player playerKI) throws IOException {
    JSONObject jsonObject = JSON_commands.swapPickedCardWithOwnCardsKI(returnHighestKnownCard(playerKI));
    if (currentPickedCard.getValue() < 5) {
        jsonObject = JSON_commands.swapPickedCardWithOwnCardsKI(returnHighestKnownCard(playerKI));
        updateKnownCardsOfKI(playerKI);
        sendKIsmiely(TypeDefs.tongueOut);
    }
    if (currentPickedCard.getValue() == 5 || currentPickedCard.getValue() == 6 || currentPickedCard.getValue() == 13)
        jsonObject = JSON_commands.playPickedCardKI();
        sendKIsmiely(TypeDefs.smiling);
    }
    if (currentPickedCard.getValue() == 7 || currentPickedCard.getValue() == 8) {
        jsonObject = JSON_commands.useFunctionalityPeekKI(returnPeekCardForKI(playerKI));
        sendKIsmiely(TypeDefs.laughing);
    }
    if (currentPickedCard.getValue() == 9 || currentPickedCard.getValue() == 10) {
        jsonObject = JSON_commands.useFunctionalitySpyKI(returnSpyCardForKI(playerKI), getRealPlayer());
        sendKIsmiely(TypeDefs.laughing);
    }
    if (currentPickedCard.getValue() == 11 || currentPickedCard.getValue() == 12) {
        Card highestKnownCard = returnHighestKnownCard(playerKI);
        Card lowestKnownCardOfOther = returnLowestKnownCardOfOtherPlayer(playerKI);
        Player other = getRealPlayer();
        jsonObject = JSON_commands.useFunctionalitySwap(highestKnownCard, playerKI, lowestKnownCardOfOther, other);
        updateKnownListsAfterSwapping(playerKI, highestKnownCard, lowestKnownCardOfOther);
        sendKIsmiely(TypeDefs.laughing);
    }

    return jsonObject;
}
```

# Künstliche Intelligenz

Die KI soll so menschlich wie möglich wirken:

Deshalb ändert sie die Emojis.
Wenn der reale Spieler zB. eine Karte der KI anschaut, ändert sie den Emoji in einen verärgerten Smiley:

# Fragen?