Methods

**ControllerPlayer**

* protected String doInBackground(Object... command)

Implemented: DELETE and GET command

DELETE: Defines the necessary URL with the player-id as query parameter and sends a request with the DELETE-method to the webservice. As return value you get the response code of the webservice.

GET: Defines the necessary URL and sends a request with the GET-method to the webservice. As return value you get a string in JSON-format which contains all players of the database.

**ControllerOccupation**

* protected String doInBackground(Object... command)

POST: Defines the necessary URL and sends a request with the POST-method and the delivered occupation object as content to the webservice. As return value you get the response code of the webservice.

DELETE: Defines the necessary URL with the player-id and the position-name as query parameters and sends a request with the DELETE-method to the webservice. As return value you get the response code of the webservice.

GET: Defines the necessary URL with the player-id as path parameter and sends a request with the GET-method to the webservice. As return value you get a string in JSON-format which contains all occupations concerning the delivered player.

**Database**

* public void removePlayer(int id, String name) throws Exception

Instantiates a new ControllerPlayer and uses it to perform a DELETE with the delivered player-id. It also removes the specific player from the local ArrayList “allPlayers”. It throws an exception if the webservice does not return the code “200”.

* public void addOccupation(int playerId, String positionName) throws Exception

Instantiates a new ControllerOccupation and uses it to perform a POST with the delivered player-id and position-name (🡪occupation). It throws an exception if the webservice does not return the code “200”.

* public void removeOccupation(int playerId, String positionName) throws Exception

Instantiates a new ControllerOccupation and uses it to perform a DELETE with the delivered player-id and position-name. It throws an exception if the webservice does not return the code “200”.

* public void loadAllPlayers() throws Exception

Instantiates a new ControllerPlayer and uses it to perform a GET. It adds all players it has got from the webservice to the local ArrayList “allPlayers” in a new thread. It throws an exception if the webservice returns “null”.

* public void loadAllOccupations() throws Exception

Calls the “loadOccupations” method for every item of the local ArrayList “allPlayers”.

* private void loadOccupations(Player player) throws Exception

Instantiates a new ControllerOccupation and uses it to perform a GET for the delivered player. It calls the “addPosition” method of the delivered player for every occupation it has got from the webservice with the particular position-name.

**Player**

* public void addPosition(String positionName, boolean calledFromGet) throws Exception

Checks whether it has been called from the get method or not (calledFromGet-parameter), if it has not been called from get and if it’s not already contained in the ArrayList “positions” it calls the “addOccupation” method of the database to deliver it to the webservice (this is used by the ProfileActivity to add occupations to a specific player).

If it has been called from get and is not already contained in “positions” it adds the delivered position to “positions”.

* public void removePosition(String positionName) throws Exception

After checking whether the delivered position already exists for this player it calls the “removeOccupation” method of the database and removes the delivered position from the ArrayList “positions”.

* public void resetPositions() throws Exception

Calls “removePosition” for all four possible positions.

**ProfileActivity**

* protected void onCreate(Bundle savedInstanceState)

Loads the player of whom it has to show the profile (if an admin is logged in: the player selected in the spinner of the MainActivity (name delivered as an extra of the intent), otherwise: currentPlayer from the database). Calls “getViews”. Checks the checkboxes if the position is used by the player.

* private void getViews()

Gets all checkboxes with “findViewById”.

* public void onBtnSave(View view)

Calls “addPosition” of the player if the specific checkbox is cheked and “removePosition” if it is not checked. At the end it closes the activity.

* public void onBtnReset(View view)

Calls “resetPositions” of the player and unchecks all checkboxes. At the end it closes the activity.

**RemoveDialogActivity**

* protected void onCreate(Bundle savedInstanceState)

Fills the textview with the “confirmation”-message.

* public void onBtnYes(View view)

Calls the “removePlayer” method of the database with the delivered player name (as an extra of the intent) and the “handlePlayerRemoved” of the listener if any is registered. At the end it closes the activity.

* public void onBtnNo(View view)

Closes the activity.

**RemovePlayerActivity**

* protected void onCreate(Bundle savedInstanceState)

Calls “setAdapterLvPlayer” and registers a “setOnItemClickListener” for the listview “lvPlayer” which opens the “RemovePlayerActivity” and delivers the id, name and goalDifference to it.

* private void setAdapterLvPlayer()

Fills the listview “lvPlayer” with the return value of “getAllPlayers” of the database.

* public void onResume()

Calls “setAdapterLvPlayer” to guarantee “lvPlayers” is up to date after the user removed a certain player.