**Team Rocket Dao Layer:**

**Quick Reference:**

* **Account(String username, String password)**
  + Gets all user info if the data matches records (checks email, password pair as well)
  + Null if none exist
* **Account(String username, String email, String password, String first, String last)**
  + Creates a new account object with parameters and a new row in database (if username and email are not already in use)
  + Returns null if either username or email is already in use
* **(Account Method) removeTeamByName(String name)**
  + Removes Team from teams variable and database using the teams team\_name variable
* **(Account Method) removeTeamById(int id)**
  + Removes Team from teams variable and database using the teams team\_id variable
* **(Account Method) getTeamByName(String name)**
  + Returns Team from teams variable and database using the teams team\_name variable
* **(Account Method) getTeamById(int id)**
  + Returns Team from teams variable and database using the teams team\_id variable
* **(Account Method) addTeam(String teamName, String visibility)**
  + Adds Team to teams variable and to the database
* **(Team Method ) addPokemon(int pokeId, String name, int level, String move\_1, String move\_2, String move\_3, String move\_4, int teamId)**
  + Adds Pokemon to pokemon variable and to the database
  + Position is given based on what is not already taken
  + If there are already 6 in the team, it will not be added
* **(Team Method) removePokemon(int pos)**
  + Removes the pokemon within the team with the position variable value equal to pos parameter
* **(Team Method) switchPokemon(int pos1, int pos2)**
  + Changes the position variable of the Pokemon with the related position variables pos1 and pos2 to the opposite (i.e Pokemon 1 has Pokemon 2’s position now and vice versa)

**Interfaces:**

1. AccountDao
   1. Methods to allow us to exchange information between us and the server, in regards to the Account table and objects
      1. public Account login(String username, String password)
      2. public Account signUp(String username, String email, String password, String first, String last)
      3. public List<Team> loadTeams(int id)
2. TeamDao
   1. Methods to allow us to exchange information between us and the server, in regards to the Team table and objects
      1. public List<Pokemon> loadPokemon(int id)

**Abstract Classes:**

1. **Transaction**
   1. This Class contains methods for persisting data onto the database and is extended by all the other abstract classes in the project
      1. **public void save()**
         1. calls Session from HibernateUtil, creates a Transaction, and saves the calling object to the session, and then commits it
      2. **public void evict()**
         1. calls Session from HibernateUtil and evicts the calling object from the Session.
            1. Used with switchPositions() in Teams class. Position is part of the primary key on the Pokemon table, and in order to keep from violating the constraint, one of the pokemon objects had to be removed from the session, and then updated using update() from Transactions class
      3. **public void update()**
         1. calls Session from HibernateUtil, creates a Transaction, and updates a detached object
      4. **public void merge()**
         1. calls Session from HibernateUtil, creates a Transaction, and merges the current state of the object with its related table rows in the database
      5. **public void delete()**
         1. calls Session from HibernateUtil, creates a Transaction, and deletes the object from the database
2. **AccountImpl**
   1. This is the Implementation of the AccountDao interface and is used to retrieve a users account information from the database, including the Pokemon Teams related with the account
      1. **public Account login(String username, String password)**
         1. This method calls Session from HibernateUtil and uses it to create a query that checks the credentials provided. If there is an account with the associated credentials (username, password pair **or** email, password pair) an account is created using the retrieved data.
         2. If there is a match and an account object is created, loadTeams() is also called to populate the teams list of the account object
         3. If no match is found, a null value is returned
      2. **public Account signUp(String username, String email, String password, String first, String last)**
         1. This method calls Session from HibernateUtil and uses it to create a query that checks to see if there is already an existing account with the same username or email.
         2. If there is no matches, a new Account object is created and the object then calls save() method to save it to the database
         3. If there is a match, a null value is returned
      3. **public List<Team> loadTeams(int id)**
         1. This method calls Session from HibernateUtil and uses it to create a query that returns all the teams with the userId equal to the id parameter from the Team table.
3. **TeamImpl**
   1. This is the Implementation of the TeamDao interface and is used to retrieve the Pokemon related with the team
      1. **public List<Pokemon> loadPokemon(int id)**
         1. This method calls Session from HibernateUtil and uses it to create a query that returns all the pokemon with the teamId equal to the id parameter from the Pokemon table.

**Concrete Classes:**

1. **Account**
   1. The object related to a specific row of data in the Account table of the database.
      1. **public Account(String username, String email, String password, String first, String last)**
         1. This constructor is used to Sign-Up a new account by calling the inherited signup method. Account will be set to null if the username or email is already in use
      2. **public Account(String username, String password)**
         1. This constructor is used to login to an existing account using the inherited login method. Will return null if the account does not exist
      3. **pubic void removeTeamByName(String name)**
      4. **public void removeTeamById(int id)**
      5. **pubic void getTeamByName(String name)**
      6. **public void getTeamById(int id)**
      7. **public void addTeam(String teamName, String visibility)**
      8. **Getters**
         1. There are getters for every account variable (user\_id, username, password, email, first, last, teams)
      9. **Setters**
         1. There are setters for the following account variables (username, password, email, first, last, teams)
         2. All setter methods (with the exception of teams as it is not necessary) call this.merge() to update the database to the change
2. **Team**
   1. The object related to a specific row of data in the Account table of the database.
      1. **public Team(int user\_id, String team\_name, String visibility)**
         1. Constructor for creating a new Team object
      2. **public void addPokemon(int pokeId, String name, int level, String move\_1, String move\_2, String move\_3, String move\_4, int teamId)**
         1. This method is used to add a Pokemon object to the database and the pokemon list within a team object
         2. If the size of the pokemon list is less than six, the position of the Pokemon object being added is set to the current size of the list plus 1
         3. If there are already six Pokemon objects in the pokemon list, no action is taken
      3. **public void removePoke(int pos)**
         1. removes the pokemon with the corresponding position value
      4. **public void switchPokemon(int pos1, int pos2)**
         1. changes the value of the position variable for the Pokemon object with position value pos1 to pos2, and vise versa for Pokemon object with position value pos2
      5. **Getters**
         1. There are getters for every Team variable (team\_id, user\_id, team\_name, visibility, pokemon)
      6. **Setters**
         1. There are setters for the following Team variables (team\_name, visibility, pokemon)
         2. All setter methods (with the exception of pokemon as it is not necessary) call this.merge() to update the database to the change
3. **Pokemon**
   1. The object related to a specific row of data in the Pokemon table of the database.
      1. **Getters**
         1. There are getters for every Pokemon variable (pokeId, name, level, move\_1-move\_4, position, team\_id)
      2. **Setters**
         1. There are setters for the following Pokemon variables(name, level, move\_1-move\_4)
         2. All setter methods call this.merge() to update the database to the change
         3. The setPosition is set to private so that it cannot be called because in order to maintain the integrity of the database, in order to change the position of a pokemon, you have to use the team method switchPokemon which uses reflection to call this method. This makes sure that we cannot give two or more pokemon within the same team, the same position