**Updated on 10/18/2015**

\*\*\* All words underlined in blue (i.e. [**newEntry**](#newEntry)) are name of variable/ Parse class/ method/ etc. and contain hyperlink

Parse Database

* User Class
  + **Last\_Name [String]** – last name
    - Facebook/ Twitter – Null
  + **First\_Name [String]** – first name
    - Facebook/ Twitter – Null
  + **username [String]** – login name
    - Facebook/ Twitter – Random string
  + **password [String]** – login password
    - Facebook/ Twitter – Null
  + **emailVerified [Boolean]** – email verification
    - Can be ‘false’ now, but not for future
  + **newEntry [Pointer to ‘Friend\_update’ class]** – Point to user data which needs public write permission
    - Required for all users
    - Should not be changed after setup
  + **email [String]** – email address
    - Required for all users
  + **usernameFacebook [String]** – full name for facebook users
    - Normal/ Twitter – Null
  + **usernameTwitter [String]** – full name for twitter users
    - Normal/ Facebook – Null
* Friend\_update Class
  + **newEntry [Boolean]** – indicate if there is any change for friend-related stuff made by friend(s)
    - One of the following will set it to ‘true’:
      * Name/ email/ profile pic change of friend(s) – not yet implemented
      * [Confirmation status](#confirmed) changed by friend(s)
      * Deleted by friend(s)
      * New statement added by friend(s) – not yet implemented
  + **list [List<ParseObject>]** – list of ParseObject from class [**FriendList**](#FriendList) with the corresponding user as field [**userOne**](#userOne) or [**userTwo**](#userTwo)
    - Coexist in phone local memory
  + **offlineFriendList [List<String>]** – list of [string converted](#toString) from object [**Friend**](#Friend)
    - Coexist in phone local memory
* FriendList Class
  + **userOne [Pointer to ParseUser]** – the person who added the friendship
  + **userTwo [Pointer to ParseUser]** – the person who was added by [**userOne**](#userOne)
  + **confirmed [Boolean]** – indicate if the friendship is confirmed by [**userTwo**](#userTwo)
  + **owedByOne [Double]** – the amount of money owed by [**userOne**](#userOne) to [**userTwo**](#userTwo)
  + **owedByTwo [Double]** – the amount of money owed by [**userOne**](#userOne) to [**userTwo**](#userTwo)
* Statement Class (EMPTY)

Code

* AddStatementFragment.java

Variables

* + **dateRecord [Arraylist<Integer>(6)]**– data about previous selected date for DATE and DEADLINE by user
    - *Default data* – Date: 01/01/1899, Deadline: 12/31/2101

Methods

* + **void showDataPickerDialog** – open a window for user to pick date (by creating object [**DatePickerFragment**](#DatePickerFragment) with 2 arguments)
    - *Parameter 1 [int]* – the trigger view, DATE (=0) or DEADLINE (=3)
    - *Argument 1 [int]* – the parament above
    - *Argument 2 [Arraylist<Integerl>(6)]* – [**dateRecord**](#dateRecord)
  + **void setClickedIconText** – set text of a TextView based on the icon that is clicked
    - *Parameter [String]* – text to be displayed
  + **void setDate** – update variable [**dataRecord**](#dateRecord) + display the new date selected by user to DATE or DEADLINE TextView if the selected date is valid
    - *Parament 1 [int]* – new selected year
    - *Parament 2 [int]* – new selected month
    - *Parament 3 [int]* – new selected day
    - *Parament 4 [int]* – the trigger view (DATE/ DEADLINE)
  + **boolean isValidDate** – check if the new selected date is valid
    - *Parament 1 [int]* – new selected year
    - *Parament 2 [int]* – new selected month
    - *Parament 3 [int]* – new selected day
    - *Parament 4 [int]* – the trigger view (DATE/ DEADLINE)
* ContainerActivity.java

Methods

* + **void iconClick** – OnClickListener of all icons in “AddStatement.xml” + generate a string based on the clicked icon and pass it to method [**setClickedIconText**](#setClickedIconText)
    - *Parament 1 [View]* – clicked view (icon)
* ContentActivity.java

\*\* All sidebar-related stuffs are not written \*\*

Methods

* + **void displayAddFriendDialog** – display a window + search friend (ParseUser) using the email user entered, pass it to method [**addFriend**](#addFriend)
    - *No parament*
  + **void addFriend** – add a new ParseObject to class [**FriendList**](#FriendList) + update the content of [**list**](#list) ,[**offlineFriendList**](#offlineFriendList), [**pFriendList**](#pFriendList) + set field [**newEntry**](#newEntry) of the friend to true if the friendship is valid
    - *Parament 1 [ParseUser]* – friend the user is planning to add
  + **boolean isDuplicateFriend** – determined if the friendship between userA and userB is existed
    - *Parament 1 [ParseUser]* – userA
    - *Parament 2 [ParseUser]* – userB
  + **void checkForUpdate** – set variable [**changedRecord**](#changedRecord) to true and re-create variable [**list**](#list) if the field [**newEntry**](#newEntry) is set true (and internet is available)
    - *No parameter*
  + **boolean isNetworkConnected** – check if internet is available
    - *No parameter*
  + **void setEmailFacebookTwitterUser** – display a window + prompt user to link their email address to the account + set field [**newEntry**](#newEntry) of all his/her friends to true
    - *No parameter*
* FriendActivity.java

Custom Objects

* + **ResolveStatementAdaptor** – display data in arraylist<[**Friend**](#Friend)> as ListView based on the item design in ‘friend\_item.xml’
    - *Parament 1 [Context]*
    - *Parament 2 [int]* – layout xml for each ListView item
    - *Parament 3 [List<Friend>]* – list containing data that need to be shown

Methods

* + **boolean isNetworkConnected** – check if internet is available
    - *No parameter*
* LoginActivity.java

Methods

* + **void goToLoggedInPage** – update (function as method [**checkForUpdate**](#checkForUpdate)) if field [**newEntry**](#newEntry) is set true + go to login page
    - *No parameter*
  + **void setUpUsernameTwitter** – set fields two [**newEntry**](#newEnrtyPointer) ([**newEntry**](#newEntry)), [**list**](#list), [**offlineFriendList**](#offlineFriendList) of userA to default values if s/he is first time user + update fields [**usernameTwitter**](#usernameTwitter) + go to logged in page
    - *Parameter 1 [ParseUser]* – userA
  + **void setUpUsernameTwitter** – set fields two [**newEntry**](#newEnrtyPointer) ([**newEntry**](#newEntry)), [**list**](#list), [**offlineFriendList**](#offlineFriendList) of userA to default values if s/he is first time user + update fields [**usernameFacebook**](#usernameFacebook) + go to logged in page
    - *Parameter 1 [ParseUser]* – userA
  + **void goToRegisterPage** - go to registration page
    - *No parameter*
  + **boolean isNetworkConnected** – check if internet is available
    - *No parameter*
* MainActivity.java + RegisterActivity.java

\*\* No defined method nor variable \*\*

* ResolveStatementFragment.java

Custom Objects

* + **FriendAdaptor** – display data in arraylist<[**Friend**](#Friend)> as ListView based on the item design in ‘resolvestatements\_item.xml’
    - *Parament 1 [Context]*
    - *Parament 2 [int]* – layout xml for each ListView item
    - *Parament 3 [List<Friend>]* – list containing data that need to be shown

Methods

* + **boolean isNetworkConnected** – check if internet is available
    - *No parameter*
* Utility.java

Variables

* + **pFriendList [List<Friend>]** – the copy of List<Friend> generated from [**generateFriend-Array**](#generateFriendArray) or [**generateFriendArrayOffline**](#generateFriendArrayOffline)
    - *Default value* – Null
  + **changedRecord [Boolean]** – indicating whether the data is changed
    - *Default value* – true

Custom Objects

* + **DatePickerFragment** – display a date picker with the starting date as either previous selected data (if exists) or current date + return the new selected date
    - *2 parameters [List<Integer>, int] via Bundle (MUST!)*
      * 1) [Arraylist<Integer> with previous selected date](#dateRecord)
      * 2) view that trigger this class
  + **Friend** – the friendship between current user and userA
    - *Parameter 1 [String]* – The object ID of the corresponding ParseObject in class [**FriendList**](#FriendList)
    - *Parameter 2 [ParseUser]* – userA
    - *Parameter 3 [String]* – full name of userA
    - *Parameter 4 [String]* – email of userA
    - *Parameter 5 [Double]* – the amount that current user owes to userA
    - *Parameter 6 [Double]* – the amount that userA owes to current user
    - *Parameter 7 [Boolean]* – the confirmation status
    - *Parameter 8 [Boolean]* – is the current user belonged to field [**userOne**](#userOne)(who added the friendship)
    - Methods
      * **void setConfirm** – set confirmation status variable to true + set field [**confirmed**](#confirmed) of the corresponding ParseObject to true + set field [**newEntry**](#newEntry) of userA to true
      * **void deleteFriend** – delete the corresponding ParseObject from [**list**](#list) + delete that ParseObject from class [**FriendList**](#FriendList) + set field [**newEntry**](#newEntry) of userA to true
      * **void notifyChange** – set field [**newEntry**](#newEntry)of userA to true
      * **String toString** – obtain this [**Friend**](#Friend) object as String

Methods

* + **boolean isNormalUser** – check if userA is a normal user (register via the app)
    - *Parameter 1 [ParseUser]* - userA
  + **boolean isFacebookUser** – check if userA is signed in via Facebook
    - *Parameter 1 [ParseUser]* - userA
  + **boolean isTwitterUser** – check if userA is signed in via Twitter
    - *Parameter 1 [ParseUser]* - userA
  + **String getUserName** – get the full name of userA
    - *Parameter 1 [ParseUser]* - userA
  + **void generateRawFriendList** – generate a list of ParseObject from class [**FriendList**](#FriendList) with the userA as field [**userOne**](#userOne) or [**userTwo**](#userTwo) + store the result to [**list**](#list)
    - *Parameter 1 [ParseUser]* - userA
  + **List<Friend> generateFriendArray** – generate a list of [**Friend**](#Friend) (online version)
    - Two cases:
      * If [**pFriendList**](#pFriendList) isn’t Null and [**changedRecord**](#changedRecord) is false, return [**pFriendList**](#pFriendList)
      * Else, generate List<Friend> from [**list**](#list) + store result to [**pFriendList**](#pFriendList) + convert List<Friend> to List<String> and store to [**offlineFriendList**](#offlineFriendList) + set [**changedRecord**](#changedRecord) to false, return [**pFriendList**](#pFriendList)
    - *No parameter*
  + **List<Friend> generateFriendArrayOffline** – generate a list of [**Friend**](#Friend) (offline version)
    - Two cases:
      * If [**pFriendList**](#pFriendList) isn’t Null, return [**pFriendList**](#pFriendList)
      * Else, generate List<Friend> from [**offlineFriendList**](#offlineFriendList) + store result to [**pFriendList**](#pFriendList) + set [**changedRecord**](#changedRecord) to true, return [**pFriendList**](#pFriendList)
    - *No parameter*
  + **void addToExistingFriendList** – generate a [**Friend**](#Friend) object + add it and its [converted String](#toString) to [**pFriendList**](#pFriendList) and [**offlineFriendList**](#offlineFriendList) + set field [**newEntry**](#newEntry) of userA to true if [**pFriendList**](#pFriendList) isn’t null
    - *Parameter 1 [String]* – the object ID of this new friendship (ParseObject)
    - *Parameter 2 [ParseUser]* – userA
  + **void removeFromExistingFriendList** – remove a [**Friend**](#Friend) object and its [converted String](#toString) from [**pFriendList**](#pFriendList) and [**offlineFriendList**](#offlineFriendList)if [**pFriendList**](#pFriendList) isn’t null
    - *Parameter 1 [*[***Friend***](#Friend)*]* – item needed to be removed
  + **void resetExistingFriendList** – set [**pFriendList**](#pFriendList) to Null
    - *No parameter*
  + **void setChangeRecord** – set [**changedRecord**](#changedRecord) to true
    - *No parameter*
  + **boolean checkNewEntryField** – return the value of field [**newEntry**](#newEntry) of current user
    - No parameter
  + **void editNewEntryField** – modify the value of field newEntry of userA to desired\_value
    - *Parameter 1 [ParseUser]* – userA
    - *Parameter 2 [Boolean]* – desired\_value
  + **ParseObject getRawListLocation** – return the location as a ParseObject where [**list**](#list) and [**offlineFriendList**](#offlineFriendList) located in local memory
    - *No parameter*
  + **void setNewEntryFieldForAllFriend** – set the value of field [**newEntry**](#newEntry) for all friends of the current user
    - *No Parameter*
* ViewStatementFragment.java (EMPTY)