

---

# Class Models

## FH Mobile Application

Version 1.0

Prepared by

Omar Rivera  
Andrew Poirier  
Daven Amin  
Rick Rejeleene

## Table of Contents

Table 1.1 Client Application – Class Description .....	3
Table 2.1 – Client_Application::login – Method Description .....	4
Table 2.2 – Client_Application::resetPassword – Method Description.....	4
Table 2.3 – Client_Application::register– Method Description.....	5
Table 2.4 – Client_Application::refreshBalancesFromServer– Method Description .....	5
Table 2.5 – Client_Application::getHelp– Method Description.....	6
Table 2.6 – Client_Application::updateProfile– Method Description.....	6
Table 3.1 – Server_Application– Class Description .....	7
Table 4.1 – Server_Application::connectClient– Method Description.....	8
Table 4.2 – Server_Application::connectToDatabase– Method Description .....	8
Table 4.3 – Server_Application::validateWorkout– Method Description .....	9
Table 4.4 – Server_Application::getWorkoutsForUser– Method Description .....	9
Table 4.5 – Server_Application::getBalanceForUser– Method Description.....	10
Table 4.6 – Server_Application::updateBalance– Method Description .....	10
Table 4.7 – Server_Application::createOrUpdateUser– Method Description.....	11
Table 4.8 – Server_Application::deleteUser– Method Description.....	11
Table 4.9 – Server_Application::checkForMissedActivities– Method Description .....	12
Table 4.10 – Server_Application::changePassword– Method Description .....	12
Table 4.11 – Server_Application::requestUsersInformation– Method Description.....	13
Table 4.12 – Server_Application::addWorkout– Method Description.....	13
Table 4.13 – Server_Application::getUsersCanValidate– Method Description.....	14
Table 5.1-User_Information-Class Description .....	15
Table 6.1 – User_Information::getBalance– Method Description.....	16
Table 6.2 – User_Information::updateProfile– Method Description.....	16
Table 6.3 – User_Information::setPassword– Method Description .....	17
Table 6.4 – User_Information::addBalance– Method Description .....	17
Table 6.4 – User_Information::addWorkout– Method Description.....	18
Table 6.5 – User_Information::getWorkout– Method Description .....	18
Table 6.6 – User_Information::getUserBMI– Method Description .....	19
Table 6.7 – User_Information::validateWorkout– Method Description.....	19
Table 6.8 – User_Information::getValidatableWorkouts– Method Description .....	20
Table 6.9 – User_Information::createWorkout– Method Description .....	20

Table 7.1 - User_Session-Class Description.....	21
Table 8.1 – User_Session::getSessionID– Method Description .....	22
Table 8.2 – User_Session::getUser– Method Description .....	22
Table 9.1 – Credentials – Class Description.....	23
Table 10.1 – Credentials::getUserID– Method Description .....	24
Table 10.2 – Credentials::getPassword– Method Description.....	24
Table 11.1 – Workout – Class Description.....	25
Table 12.1 – Workout::getStartDate– Method Description.....	26
Table 12.2 – Workout::getNumberOfWeeks– Method Description .....	26
Table 12.3 – Workout::getActivities– Method Description .....	27
Table 13.1 - <<Interface>> Activity – Class Description .....	28
Table 14.1 – Activity::getQuantity– Method Description .....	29
Table 14.2 – Activity::getName– Method Description.....	29
Table 14.3 – Activity::getDescription– Method Description .....	30
Table 14.4 – Activity::getNumberPerWeek– Method Description .....	30

**Table 1.1 Client Application – Class Description**

<b>Name</b>	<i>Client_Application</i>
<b>Base Class</b>	
<b>Purpose</b>	<i>The Purpose of the Client Application is it helps the user login into the application.</i>
<b>States</b>	<i>Empty, full, or neither. Inactive or Active</i>
<b>Constructors</b>	<i>Default</i>
<b>Operators Mutators</b>	login() resetPassword() refreshBalancesFromServer() updateProfile() register()  getHelp()
<b>Accessors</b>	
<b>Fields</b>	<i>User_information</i>

**Table 2.1 – Client\_Application::login – Method Description**

<b>Prototype</b>	Client_Application::login(User_Information)
<b>Purpose</b>	<i>Helps the user login into the client application</i>
<b>Receives</b>	<i>The Login receives a string from the user.</i>
<b>Returns</b>	<i>The Login returns True if the login Id is true or False if the login Id is false.</i>
<b>Remarks</b>	<i>If Login ID matches it logs in to the system.</i>

**Table 2.2 – Client\_Application::resetPassword – Method Description**

<b>Prototype</b>	Client_Application::resetPassword(user:User_Information)
<b>Purpose</b>	<i>The purpose of the reset password is if the user forgets the password, he can reset it.</i>
<b>Receives</b>	<i>The reset password receives the string from the user, in this the userid is the string.</i>
<b>Returns</b>	<i>True if the userid matches False if the userid doesn't match.</i>
<b>Remarks</b>	<i>Helps to reset the password</i>

**Table 2.3 – Client\_Application::register- Method Description**

<b>Prototype</b>	Client_Application::register(user:User_Information)
<b>Purpose</b>	<i>The purpose of Register is for a new user to register into the FH mobile App.</i>
<b>Receives</b>	<i>The Register receives the user name, password which is a string</i>
<b>Returns</b>	<i>void</i>
<b>Remarks</b>	<i>Helps to register for a new user.</i>

**Table 2.4 – Client\_Application::refreshBalancesFromServer- Method Description**

<b>Prototype</b>	Client_Application::refreshBalancesFromServer()
<b>Purpose</b>	<i>The purpose of the Refresh balance is it updates the balance for the user</i>
<b>Receives</b>	<i>The Refresh Balance receives Integers from the Server.</i>
<b>Returns</b>	<i>It returns Balance as Integers.</i>
<b>Remarks</b>	<i>The Refresh Balance helps to check the balance for the user.</i>

R.R.

**Table 2.5 – Client\_Application::getHelp– Method Description**

<b>Prototype</b>	Client_Application::getHelp()
<b>Purpose</b>	<i>The Purpose of the Get Help is it brings up the instructions on how to use the application.</i>
<b>Receives</b>	<i>void</i>
<b>Returns</b>	<i>text</i>
<b>Remarks</b>	<i>Brings up the instructions and how to use the app.</i>

**Table 2.6 – Client\_Application::updateProfile– Method Description**

<b>Prototype</b>	Cilent_Application::updateProfile <i>(User_Information)</i>
<b>Purpose</b>	<i>If the User wants to change his information, he can click the update User information.</i>
<b>Receives</b>	<i>The Update user information receives the input as string</i>
<b>Returns</b>	<i>True if the update is successful</i> <i>False if the update is successful.</i>
<b>Remarks</b>	<i>Helps to modify an already created User Information.</i>

**Table 3.1 – Server\_Application– Class Description**

<b>Name</b>	<i>Server</i>
<b>Base Class(es)</b>	
<b>Purpose</b>	<i>The server is used to connect to the database and store,delete or update information for the user, workouts, and bank in the application.</i>
<b>States</b>	<i>Change/Update, Validate, Create, Delete Active or Inactive</i>
<b>Constructors</b>	<i>Default: makes server ready for data information and processing</i>
<b>Operators Mutators</b>	<i>validateWorkout(user:user_Information, workout:Workout, activity) updateBalance(balance:Integer) createOrUpdateUser(user:User_Information) deleteUser(user:User_Information) checkForMissedActivities() changePassword(User_Session, :string) requestUsersInformation() addWorkout(user:User_Information)</i>
<b>Accessors</b>	<i>connectToDatabase(location: String,db:Credentials) connectClient(Credentials) getWorkoutsForUser(user:User_Information) getBalanceForUser(user:User_Information) getUsersCanValidate(user:User_Information)</i>
<b>Fields</b>	<i>User_Session</i>



**Table 4.1 – Server\_Application::connectClient– Method Description**

<b>Prototype</b>	<i>Bool:Server_Application:connectClient(client:Credentials )</i>
<b>Purpose</b>	<i>To connect the client to a user session in the server</i>
<b>Receives</b>	<i>The clients credentials to authenticate the server</i>
<b>Returns</b>	<i>TRUE if the credentials were valid FALSE if the credentials were invalid</i>
<b>Remarks</b>	<i>The operation may fail if the user forgot to log out last time.</i>

**Table 4.2 – Server\_Application::connectToDatabase– Method Description**

<b>Prototype</b>	<i>Bool:Server_Application:connectToDatabase(location:String,db:Credentials)</i>
<b>Purpose</b>	<i>To connect the user to the database to access the user's saved information</i>
<b>Receives</b>	<i>The database location and database name as a string object The database credentials from the server</i>
<b>Returns</b>	<i>TRUE if the user and server connect to the correct database with the correct credentials FALSE if the user and server do not connect correctly or if the the credentials are invalid</i>
<b>Remarks</b>	<i>The operation may fail if the database cannot be found or the connection is interrupted at anytime with the current connection</i>

**Table 4.3 – Server\_Application::validateWorkout– Method Description**

<b>Prototype</b>	<i>Bool:Server_Application:validateWorkout(user:User_Information, workout:Workout, workout:Activity)</i>
<b>Purpose</b>	<i>To connect the user to the database to access the user's workout history and information</i>
<b>Receives</b>	<i>The server receives the user's information along with the workout and the activity associated with that workout.</i>
<b>Returns</b>	<i>TRUE if the facilitator was able to validate the workout correctly FALSE if the facilitator was unable to find the saved workout or if there are no workouts to validate</i>
<b>Remarks</b>	<i>The operation might fail if the user forgot to save the workout or if the workout does not have any activities associated with it.</i>

**Table 4.4 – Server\_Application::getWorkoutsForUser– Method Description**

<b>Prototype</b>	<i>WORKOUTS:Server_Application:getWorkoutsForUser(user:userInformation)</i>
<b>Purpose</b>	<i>This will connect the server to the database to receive the user's list of workouts from the database.</i>
<b>Receives</b>	<i>The server receives the user's information to find the list of workouts.</i>
<b>Returns</b>	<i>WORKOUTS if the the list of workouts for the user is found in the database NULL if the user does not have any current workouts</i>
<b>Remarks</b>	<i>This operation might fail if the user is not working out for a few weeks or if it the first time and there are no workouts created.</i>

**Table 4.5 – Server\_Application::getBalanceForUser– Method Description**

<b>Prototype</b>	<i>Int:Server_Application:getBalanceForUser(user:User_Information)</i>
<b>Purpose</b>	<i>This will connect the server to the bank database to access the user's current balance.</i>
<b>Receives</b>	<i>The server receives the user's information to find the user in the bank database</i>
<b>Returns</b>	<i>INTEGER if the user's balance was found in the bank database NULL if the user currently does not have any funds available.</i>
<b>Remarks</b>	<i>This operation might fail if the user has not set up bank information with a starting balance.</i>

**Table 4.6 – Server\_Application::updateBalance– Method Description**

<b>Prototype</b>	<i>Bool:Server_Application:updateBalance(balance:Integer)</i>
<b>Purpose</b>	<i>This will connect to the server and will update the users balance off of the current balance that was received</i>
<b>Receives</b>	<i>The server receives the current balance for the user connected</i>
<b>Returns</b>	<i>TRUE if the balance was updated successfully FALSE if the balance was not updated successfully</i>
<b>Remarks</b>	<i>This operation might fail if the user does not have a balance or the user does not have a penalty/reward amount set in the system</i>

**Table 4.7 – Server\_Application::createOrUpdateUser– Method Description**

<b>Prototype</b>	<i>Bool:Server_Application:createOrUpdateUser(user:User_Information)</i>
<b>Purpose</b>	<i>This will allow the user to update or create an account on the server for the application.</i>
<b>Receives</b>	<i>The server receives the current user's information if they are updating. And it will receive new user information if a user is creating an account</i>
<b>Returns</b>	<i>TRUE if the user was created correctly or the information was updated correctly FALSE if the user did not supply a required field when creating an account or the user does not update information with the correct information designated for that field</i>
<b>Remarks</b>	<i>This operation might not work if a user tries to create a user account with the same name that is already on the server.</i>

**Table 4.8 – Server\_Application::deleteUser– Method Description**

<b>Prototype</b>	<i>Bool:Server_Application:deleteUser(user:User_Information)</i>
<b>Purpose</b>	<i>This will allow the user to remove their account and information from the server</i>
<b>Receives</b>	<i>The user's information that is store on the server</i>
<b>Returns</b>	<i>TRUE if the user was deleted successfully FALSE If the user was not deleted successfully.</i>
<b>Remarks</b>	<i>The operation will fail if the user tries to delete a user that does not exist on the server.</i>

**Table 4.9 – Server\_Application::checkForMissedActivities– Method Description**

<b>Prototype</b>	<i>Bool:Server_Application:checkForMissedActivities()</i>
<b>Purpose</b>	<i>This will allow the user to check to see if they missed activities while performing their workout.</i>
<b>Receives</b>	<i>The server will receive the users information to check for missed activities.</i>
<b>Returns</b>	<i>TRUE if the user has missed any activities FALSE if the user has completed all of their activities for the workout</i>
<b>Remarks</b>	<i>The operation will fail if the user does not have any activities listed under a workout</i>

**Table 4.10 – Server\_Application::changePassword– Method Description**

<b>Prototype</b>	<i>Bool:Server_Application:changePassword(user:User_Information ;string)</i>
<b>Purpose</b>	<i>This will allow the user to change the current password saved and used on the server</i>
<b>Receives</b>	<i>The server will receive the user's session information and a sting containing the new password</i>
<b>Returns</b>	<i>TRUE if the password was changed successfully FALSE if the user could not update the password sucessfully(length was not correct or did not pass password specifications)</i>
<b>Remarks</b>	<i>The operation might fail if the user log outs before the success or fail statement is sent back to the user.</i>

**Table 4.11 – Server\_Application::requestUsersInformation– Method Description**

<b>Prototype</b>	<i>UserInformation:Server_Application:requestUsersInformation()</i>
<b>Purpose</b>	<i>This will allow the server to return all users information</i>
<b>Receives</b>	<i>All of the users that are currently on the server</i>
<b>Returns</b>	<i>TRUE if the information was passed correctly FALSE if the user does not exist or the information was not passed correctly</i>
<b>Remarks</b>	<i>The operation might fail if there are no users on the server.</i>

**Table 4.12 – Server\_Application::addWorkout– Method Description**

<b>Prototype</b>	<i>Bool:UserInformation:Server_Application:addWorkout(user:User_Information)</i>
<b>Purpose</b>	<i>This allows the user to add a workout to their account</i>
<b>Receives</b>	<i>The user's information</i>
<b>Returns</b>	<i>TRUE if the workout was added FALSE if the workout could not be added</i>
<b>Remarks</b>	<i>The operation might fail if the user tries to add a workout with an existing workout name</i>

**Table 4.13 – Server\_Application::getUsersCanValidate- Method Description**

<b>Prototype</b>	<i>Bool:UserInformation:Server_Application:getUsersCanValidateWorkout(user:User_Information)</i>
<b>Purpose</b>	<i>Allows the user to set the users that can validate their workout.</i>
<b>Receives</b>	<i>The user's information</i>
<b>Returns</b>	<i>List of users that can validate workout</i>
<b>Remarks</b>	<i>This operation might fail if there are now users listed to validate a workout</i>

**Table 5.1-User\_Information-Class Description**

<b>Name</b>	<i>User_Information</i>
<b>Base Class(es)</b>	
<b>Purpose</b>	<i>A User Information object contains all the user information that can be listed, kept, used by other modules or discarded.</i>
<b>States</b>	<i>Empty, full, or neither</i> <i>Inactive or Active</i>
<b>Constructors</b>	<i>Default: contains user information</i>
<b>Operators</b>	<i>setWorkout(workouts: Workouts)</i>
<b>Mutators</b>	<i>updateProfile(dob:String,height:Integer,weight:Integer)</i> <i>setPassword(newPassword: String)</i> <i>addBalance(balanceToAdd:Integer)</i> <i>addWorkout(new Workout: Workout)</i> <i>setWorkouts(workouts:workout)</i> <i>validateWorkout(User_information, Workout, Activity)</i>
<b>Accessors</b>	<i>getBalance():Integer</i> <i>getUserBMI():Integer</i> <i>getValidatableWorkout():Bool</i>
<b>Fields</b>	<i>userID:String</i> <i>userDOB:integer</i> <i>userHeight: Integer</i> <i>userWeight: Integer</i> <i>password:String</i> <i>workouts:Integer</i> <i>usersCanValidate: List</i>



**Table 6.1 – User\_Information::getBalance– Method Description**

<b>Prototype</b>	<i>Integer:User_Information::getBalance ()</i>
<b>Purpose</b>	<i>Returns the balance of the current user</i>
<b>Receives</b>	<i>Will user information in the database to retrieve recurring user balance.</i>
<b>Returns</b>	<i>INTEGER – Returns the balance of the current user requested by the accessor's functions.</i>
<b>Remarks</b>	<i>The operation may fail if there's not balance available for the current user.</i>

**Table 6.2 – User\_Information::updateProfile– Method Description**

<b>Prototype</b>	<i>Bool User_Information::updateProfile (dob:String,height:Integer,weight:Integer)</i>
<b>Purpose</b>	<i>Update the user information fields for a user</i>
<b>Receives</b>	<i>dob- contains the date of birth of the user height- the height of the current user weight- contains the weight of the current user</i>
<b>Returns</b>	<i>TRUE if the operation succeeded FALSE if the data update information contains invalid values</i>
<b>Remarks</b>	<i>The operation may fail if the DOB,height or weight are invalid values for the profile.</i>

**Table 6.3 – User\_Information::setPassword– Method Description**

<b>Prototype</b>	<i>Bool User_Information::setPassword(newPassword)</i>
<b>Purpose</b>	<i>Add a new password for the current user</i>
<b>Receives</b>	<i>password-the password for the current user security settings</i>
<b>Returns</b>	<i>TRUE if the operation succeeded</i>
<b>Remarks</b>	<i>The operation may fail if the password is not valid (the password must follow security the minnimum ssecurity requirements).</i>

**Table 6.4 – User\_Information::addBalance– Method Description**

<b>Prototype</b>	<i>Integer User_Information::addBalance(balanceToAdd:Integer)</i>
<b>Purpose</b>	<i>Update the balance for the current user.</i>
<b>Receives</b>	<i>balanceToAdd- Contains the balance amount to be assigned for the current user.</i>
<b>Returns</b>	<i>Integer- user balance if the operation succeeded</i>
<b>Remarks</b>	<i>The operation may fail if the balance is not valid a valid amount (the balance must follow system policy).</i>

**Table 6.4 – User\_Information::addWorkout- Method Description**

<b>Prototype</b>	<i>Bool User_Information::addWorkout(newWorkout: Workout)</i>
<b>Purpose</b>	<i>Creates and adds the Workout routine for the current user.</i>
<b>Receives</b>	<i>newWorkout- Contains the workout profile for the user.</i>
<b>Returns</b>	<i>TRUE if the operation succeeded FALSE if the workout routine is not valid</i>
<b>Remarks</b>	<i>The operation may fail if the workout routine is not a (the routine must follow system policy).</i>

**Table 6.5 – User\_Information::getWorkout- Method Description**

<b>Prototype</b>	<i>Bool User_Information::setWorkout(workouts:Integer)</i>
<b>Purpose</b>	<i>Assigns the workout routine for the current user.</i>
<b>Receives</b>	<i>workout- Contains a list of all the workouts balance to be assigned to the current user.</i>
<b>Returns</b>	<i>TRUE if the operation succeeded FALSE if the amount or workout values are not valid</i>
<b>Remarks</b>	<i>The operation may fail if the workout information doesn't contains a valid numerical values (the workout must contain valid data).</i>

**Table 6.6 – User\_Information::getUserBMI– Method Description**

<b>Prototype</b>	<i>double User_Information::getUserBMI()</i>
<b>Purpose</b>	<i>Returns the calculation of the Body/Mass Index of the current user.</i>
<b>Receives</b>	<i>Will utilize user information from the database to calculate the current BMI index.</i>
<b>Returns</b>	<i>DOUBLE if the operation succeeded</i>  <i>NULL if the user information values required for the calculation are not available.</i>
<b>Remarks</b>	<i>The operation may fail if the database doesn't have a valid numerical values for the calculation.</i>

**Table 6.7 – User\_Information::validateWorkout– Method Description**

<b>Prototype</b>	<i>Bool:User_Information::validateWorkout(User_information, Workout, Activity)</i>
<b>Purpose</b>	<i>Validates the current user Workout routine base on the user workout profile and validator input</i>
<b>Receives</b>	<i>User Workout – Contains the user activity profile</i>  <i>Software routine will verify if the user completed the Workout or not.</i>
<b>Returns</b>	<i>TRUE if the operation succeeded</i>  <i>FALSE if the user information values required for the validation are not completed.</i>
<b>Remarks</b>	<i>Validates the workouts/activities of the user</i>

**Table 6.8 – User\_Information::getValidatableWorkouts– Method Description**

<b>Prototype</b>	<i>Workout User_Information::getValidatableWorkouts(User_information)</i>
<b>Purpose</b>	<i>Retrieves the list of workout routines that can be validated by the current user.</i>
<b>Receives</b>	<i>Workout Routine List</i>
<b>Returns</b>	<i>List of Workout routines that can be validated by the current user.</i>
<b>Remarks</b>	<i>Contains a list of workout objects of that can be validated</i>

**Table 6.9 – User\_Information::createWorkout– Method Description**

<b>Prototype</b>	<i>User_Information::createWorkout(User_information, Workout: Integer, Activitiy : Integer)</i>
<b>Purpose</b>	<i>Creates the workout profile for the current user</i>
<b>Receives</b>	<i>User_Information: Workout – the user activities/workout profile input</i>
<b>Returns</b>	<i>TRUE if the operation succeeded</i>  <i>FALSE if the workout profile information values required for the validation are not completed or invalid.</i>
<b>Remarks</b>	<i>Creates the workout profile which will be store in the data base.</i>

**Table 7.1 - User\_Session-Class Description**

<b>Name</b>	<i>User_Session</i>
<b>Base Class(es)</b>	
<b>Purpose</b>	<i>Helps to receive the User Session</i>
<b>States</b>	<i>Empty or full.</i>
<b>Constructors</b>	<i>Default: empty session values</i> <i>New: Contains the new session</i>
<b>Operators Mutators</b>	user
<b>Accessors</b>	getUser() getSessionID()
<b>Fields</b>	<i>sessionID</i>

**Table 8.1 – User\_Session::getSessionID– Method Description**

<b>Prototype</b>	<i>User_Session::getSessionID()</i>
<b>Purpose</b>	<i>Gets the Session ID</i>
<b>Receives</b>	<i>void</i>
<b>Returns</b>	<i>Integer</i>
<b>Remarks</b>	<i>Gets the User Session ID</i>

**Table 8.2 – User\_Session::getUser– Method Description**

<b>Prototype</b>	<i>User_Session::getUser()</i>
<b>Purpose</b>	<i>Gets the User ID from Client Application</i>
<b>Receives</b>	<i>void</i>
<b>Returns</b>	<i>User_Information</i>
<b>Remarks</b>	<i>Gets the User ID from Client Application</i>

**Table 9.1 – Credentials – Class Description**

<b>Name</b>	<i>Credentials</i>
<b>Base Class(es)</b>	
<b>Purpose</b>	<i>Helps client login to the server</i>
<b>States</b>	<i>Empty or full.</i>
<b>Constructors</b>	<i>Default: empty values</i> <i>New: contains user information</i>
<b>Operators Mutators</b>	<i>None</i>
<b>Accessors</b>	getUserID() getPassword()
<b>Fields</b>	userID <i>password</i>



**Table 10.1 – Credentials::getUserID– Method Description**

<b>Prototype</b>	Credentials::getUserID()
<b>Purpose</b>	<i>The Credentials gets the User ID from the User.</i>
<b>Receives</b>	<i>Void</i>
<b>Returns</b>	<i>The Credentials returns the User Id as a String.</i>
<b>Remarks</b>	<i>Gets the User ID from User</i>

**Table 10.2 – Credentials::getPassword– Method Description**

<b>Prototype</b>	Credentials::getPassword()
<b>Purpose</b>	<i>The Credentials gets the password from the User.</i>
<b>Receives</b>	<i>Void</i>
<b>Returns</b>	<i>The Credentials returns the password as a String</i>
<b>Remarks</b>	<i>Gets the User Password from the User.</i>

**Table 11.1 – Workout – Class Description**

<b>Name</b>	<i>Workout</i>
<b>Base Class(es)</b>	
<b>Purpose</b>	<i>Helps to receive workout information</i>
<b>States</b>	<i>Empty or full.</i>
<b>Constructors</b>	<i>Default: empty workout</i> <i>New: contains workout information</i>
<b>Operators Mutators</b>	None
<b>Accessors</b>	getStartDate() getNumberOfWeeks getActivities()
<b>Fields</b>	<i>startDate</i> <i>numberOfWeeks</i> <i>activities</i>

**Table 12.1 – Workout::getStartDate– Method Description**

<b>Prototype</b>	Workout::getStartDate()
<b>Purpose</b>	<i>Gets the Start date for the workout</i>
<b>Receives</b>	<i>Void</i>
<b>Returns</b>	<i>Integer</i>
<b>Remarks</b>	<i>Gives back the workout week</i>

**Table 12.2 – Workout::getNumberOfWeeks– Method Description**

<b>Prototype</b>	Workout::getNumberOfWeeks()
<b>Purpose</b>	<i>Gets the number of weeks for workout</i>
<b>Receives</b>	<i>None</i>
<b>Returns</b>	<i>Integer</i>
<b>Remarks</b>	<i>Gets the number of weeks the workout is going to be made.</i>

**Table 12.3 – Workout::getActivities– Method Description**

<b>Prototype</b>	Workout::getActivities()
<b>Purpose</b>	<i>Gets the type of Workout Activities</i>
<b>Receives</b>	<i>Void</i>
<b>Returns</b>	<i>Activities</i>
<b>Remarks</b>	<i>Gives the type of Activities the user wants to follow.</i>

**Table 13.1 - <<Interface>> Activity - Class Description**

<b>Name</b>	<i>Activity</i>
<b>Base Class</b>	
<b>Purpose</b>	<i>Helps to receive workout activities</i>
<b>States</b>	<i>Empty or full.</i>
<b>Constructors</b>	<i>Default: empty workout value.</i> <i>New contains workout values.</i>
<b>Operators Mutators</b>	
<b>Accessors</b>	getQuantity() getName() getDescription() getNumberPerWeek()
<b>Fields</b>	<i>void</i>

**Table 14.1 – Activity::getQuantity– Method Description**

<b>Prototype</b>	Activity::getQuantity()
<b>Purpose</b>	<i>To get the User Workout quantities from the user</i>
<b>Receives</b>	<i>The User Session receives the workout quantities from the Workout base class.</i>
<b>Returns</b>	<i>INT if there is a quantity False if there is no quantity</i>
<b>Remarks</b>	<i>Gets the Workout quantity</i>

**Table 14.2 – Activity::getName– Method Description**

<b>Prototype</b>	Workout::getName()
<b>Purpose</b>	<i>To Get the Workout names from the Workout</i>
<b>Receives</b>	<i>The Workout name is received as a String from the workout base class</i>
<b>Returns</b>	<i>The Workout name as String</i>
<b>Remarks</b>	<i>Gets the workout Name</i>

**Table 14.3 – Activity::getDescription– Method Description**

<b>Prototype</b>	Workout::getDescription()
<b>Purpose</b>	<i>Gets the Description of the Workout</i>
<b>Receives</b>	<i>Void</i>
<b>Returns</b>	<i>String</i>
<b>Remarks</b>	<i>Gets the Workout Description.</i>

**Table 14.4 – Activity::getNumberPerWeek– Method Description**

<b>Prototype</b>	Workout::getNumberPerWeek()
<b>Purpose</b>	<i>Gets the number of week</i>
<b>Receives</b>	<i>Void</i>
<b>Returns</b>	<i>Integer</i>
<b>Remarks</b>	<i>Gets the number of Weeks for workouts.</i>