



Activity Tracker

1.0

Generated by Doxygen 1.8.14

Contents

1	COMP-2005 Activity Logger Documentation	1
2	Namespace Index	3
2.1	Packages	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Class Index	7
4.1	Class List	7
5	File Index	9
5.1	File List	9
6	Namespace Documentation	11
6.1	Package com	11
6.2	Package com.activitytracker	11

7 Class Documentation	13
7.1 com.activitytracker.ActivityTracker Class Reference	13
7.1.1 Detailed Description	13
7.1.2 Member Function Documentation	14
7.1.2.1 main()	14
7.2 com.activitytracker.CreateUserWindow Class Reference	15
7.2.1 Detailed Description	17
7.2.2 Constructor & Destructor Documentation	17
7.2.2.1 CreateUserWindow()	17
7.2.3 Member Function Documentation	17
7.2.3.1 rootPanel()	17
7.2.3.2 setupActionListeners()	18
7.2.3.3 setupUI()	18
7.2.4 Member Data Documentation	18
7.2.4.1 buttonCancel	19
7.2.4.2 buttonOk	19
7.2.4.3 m_rootPanel	19
7.2.4.4 passwordField	19
7.2.4.5 textFieldEmail	19
7.2.4.6 textFieldHeight	20
7.2.4.7 textFieldName	20
7.2.4.8 textFieldWeight	20
7.3 com.activitytracker.DBManager Class Reference	21
7.3.1 Detailed Description	22
7.3.2 Constructor & Destructor Documentation	23

7.3.2.1 DBManager()	23
7.3.3 Member Function Documentation	23
7.3.3.1 createUser()	24
7.3.3.2 executeQuery()	25
7.3.3.3 executeUpdate()	26
7.3.3.4 getDateOfBirth()	27
7.3.3.5 getRunFloatAttribute()	28
7.3.3.6 getRuns()	29
7.3.3.7 getUserFloatAttribute()	30
7.3.3.8 getUserIDByEmail()	32
7.3.3.9 getUserLastRID()	33
7.3.3.10 getUserPassSalt()	34
7.3.3.11 getUserSex()	35
7.3.3.12 getUserStringAttribute()	36
7.3.3.13 init()	38
7.3.3.14 isEmpty()	40
7.3.3.15 newRun()	40
7.3.3.16 runExists()	42
7.3.3.17 setRun()	43
7.3.3.18 setUserLastRID()	44
7.3.3.19 userExists()	45
7.3.4 Member Data Documentation	46
7.3.4.1 m_conn	46
7.4 com.activitytracker.Iteration3Test Class Reference	46
7.4.1 Detailed Description	47

7.4.2	Member Function Documentation	47
7.4.2.1	main()	47
7.5	com.activitytracker.LoginWindow Class Reference	49
7.5.1	Detailed Description	51
7.5.2	Constructor & Destructor Documentation	51
7.5.2.1	LoginWindow()	51
7.5.3	Member Function Documentation	51
7.5.3.1	rootPanel()	52
7.5.3.2	setupActionListeners()	52
7.5.3.3	setupCreateUserDialog()	53
7.5.3.4	setupUI()	53
7.5.4	Member Data Documentation	53
7.5.4.1	buttonCreateUser	53
7.5.4.2	buttonLogin	54
7.5.4.3	labelLoginMsg	54
7.5.4.4	labelPassword	54
7.5.4.5	labelTitle	54
7.5.4.6	labelUsername	54
7.5.4.7	m_createUserDialog	55
7.5.4.8	m_loginHandler	55
7.5.4.9	m_rootPanel	55
7.5.4.10	passwordField	55
7.5.4.11	textFieldUsername	55
7.6	com.activitytracker.MainWindow Class Reference	56
7.6.1	Detailed Description	57

7.6.2	Constructor & Destructor Documentation	57
7.6.2.1	MainWindow()	57
7.6.3	Member Function Documentation	57
7.6.3.1	rootPanel()	58
7.6.3.2	setupActionListeners()	58
7.6.3.3	setupUI()	59
7.6.4	Member Data Documentation	59
7.6.4.1	buttonAddDevice	59
7.6.4.2	buttonMyActivity	59
7.6.4.3	buttonMyFriends	60
7.6.4.4	contentPanel	60
7.6.4.5	labelProfileIcon	60
7.6.4.6	m_rootPanel	60
7.6.4.7	panelAddDevice	60
7.6.4.8	panelMyActivity	61
7.6.4.9	panelMyFriends	61
7.6.4.10	scrollPaneMyFriends	61
7.6.4.11	tableAvailableDevices	61
7.6.4.12	tableMyActivity	61
7.6.4.13	topPanel	62
7.7	com.activitytracker.Run Class Reference	63
7.7.1	Detailed Description	64
7.7.2	Constructor & Destructor Documentation	64
7.7.2.1	Run()	64
7.7.3	Member Function Documentation	65

7.7.3.1	bulkImport()	65
7.7.3.2	getID()	66
7.7.3.3	getRuns()	66
7.7.3.4	newRunDataPoint()	67
7.7.4	Member Data Documentation	69
7.7.4.1	altitude_ascended	69
7.7.4.2	altitude_descended	69
7.7.4.3	caloriesBurned	70
7.7.4.4	date	70
7.7.4.5	dbManager	70
7.7.4.6	distance	70
7.7.4.7	duration	71
7.7.4.8	id	71
7.8	com.activitytracker.RunAttribute Enum Reference	71
7.8.1	Detailed Description	72
7.8.2	Member Data Documentation	72
7.8.2.1	ALTITUDE_ASCENDED	72
7.8.2.2	ALTITUDE_DESCENDED	72
7.8.2.3	DISTANCE	73
7.8.2.4	DURATION	73
7.9	com.activitytracker.SecureString Class Reference	73
7.9.1	Detailed Description	74
7.9.2	Constructor & Destructor Documentation	74
7.9.2.1	SecureString() [1/2]	74
7.9.2.2	SecureString() [2/2]	75

7.9.3 Member Function Documentation	76
7.9.3.1 equalString()	76
7.9.3.2 generateSalt()	76
7.9.3.3 generateSecureString()	77
7.9.3.4 getSalt()	78
7.9.3.5 toString()	78
7.9.4 Member Data Documentation	79
7.9.4.1 salt	79
7.9.4.2 secureString	79
7.10 com.activitytracker.User.Sex Enum Reference	79
7.10.1 Detailed Description	80
7.10.2 Member Data Documentation	80
7.10.2.1 FEMALE	80
7.10.2.2 MALE	80
7.11 com.activitytracker.User Class Reference	81
7.11.1 Detailed Description	82
7.11.2 Constructor & Destructor Documentation	83
7.11.2.1 User()	83
7.11.3 Member Function Documentation	84
7.11.3.1 createUser()	84
7.11.3.2 getDateOfBirth()	84
7.11.3.3 getEmailAddress()	85
7.11.3.4 getHeight()	85
7.11.3.5 getID()	85
7.11.3.6 getLastRID()	86

7.11.3.7 getName()	86
7.11.3.8 getSex()	86
7.11.3.9 getWeight()	86
7.11.3.10 getLastRID()	87
7.11.4 Member Data Documentation	87
7.11.4.1 dateOfBirth	87
7.11.4.2 dbManager	87
7.11.4.3 emailAddress	87
7.11.4.4 height	88
7.11.4.5 id	88
7.11.4.6 name	88
7.11.4.7 sex	88
7.11.4.8 weight	88
7.12 com.activitytracker.UserAttribute Enum Reference	89
7.12.1 Detailed Description	89
7.12.2 Member Data Documentation	90
7.12.2.1 DATE_OF_BIRTH	90
7.12.2.2 EMAIL_ADDRESS	90
7.12.2.3 HEIGHT	90
7.12.2.4 ID	91
7.12.2.5 NAME	91
7.12.2.6 PASSWORD	91
7.12.2.7 SALT	91
7.12.2.8 SEX	92
7.12.2.9 WEIGHT	92

8 File Documentation	93
8.1 app/src/com/activitytracker/ActivityTracker.java File Reference	93
8.2 app/src/com/activitytracker/CreateUserWindow.java File Reference	93
8.3 app/src/com/activitytracker/DBManager.java File Reference	94
8.4 app/src/com/activitytracker/Iteration3Test.java File Reference	94
8.5 app/src/com/activitytracker/LoginWindow.java File Reference	94
8.6 app/src/com/activitytracker/MainWindow.java File Reference	94
8.7 app/src/com/activitytracker/Run.java File Reference	95
8.8 app/src/com/activitytracker/RunAttribute.java File Reference	95
8.9 app/src/com/activitytracker/SecureString.java File Reference	95
8.10 app/src/com/activitytracker/User.java File Reference	96
8.11 app/src/com/activitytracker/UserAttribute.java File Reference	96
Index	97

Chapter 1

COMP-2005 Activity Logger Documentation

This website contains documentation for all source code contained in the *Activity Logger* application. Class and method documentation may be accessed in HTML format using the left-hand side navigation bar, or the search box at the top right-hand side of the page.

For offline viewing, a precompiled PDF of this documentation has been made available [here](#) Note, however, that this document does *not* contain the full source code which is included in formatted HTML on this website.

More detailed information about contributions, repository branches, and commit history is available by browsing the [GitHub repository](#) for this project.

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

com	11
com.activitytracker	11

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

com.activitytracker.ActivityTracker	13
com.activitytracker.DBManager	21
com.activitytracker.Iteration3Test	46
JDialog	
com.activitytracker.CreateUserWindow	15
JFrame	
com.activitytracker.LoginWindow	49
com.activitytracker.MainWindow	56
com.activitytracker.Run	63
com.activitytracker.RunAttribute	71
com.activitytracker.SecureString	73
com.activitytracker.User.Sex	79
com.activitytracker.User	81
com.activitytracker.UserAttribute	89

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

com.activitytracker.ActivityTracker	13
com.activitytracker.CreateUserWindow	15
com.activitytracker.DBManager	21
com.activitytracker.Iteration3Test	46
com.activitytracker.LoginWindow	49
com.activitytracker.MainWindow	56
com.activitytracker.Run	63
com.activitytracker.RunAttribute	71
com.activitytracker.SecureString	73
com.activitytracker.User.Sex	79
com.activitytracker.User	81
com.activitytracker.UserAttribute	89

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

app/src/com/activitytracker/ActivityTracker.java	93
app/src/com/activitytracker/CreateUserWindow.java	93
app/src/com/activitytracker/DBManager.java	94
app/src/com/activitytracker/Iteration3Test.java	94
app/src/com/activitytracker/LoginWindow.java	94
app/src/com/activitytracker/MainWindow.java	94
app/src/com/activitytracker/Run.java	95
app/src/com/activitytracker/RunAttribute.java	95
app/src/com/activitytracker/SecureString.java	95
app/src/com/activitytracker/User.java	96
app/src/com/activitytracker/UserAttribute.java	96

Chapter 6

Namespace Documentation

6.1 Package com

Packages

- package [activitytracker](#)

6.2 Package com.activitytracker

Classes

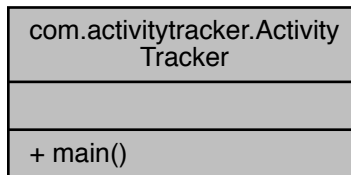
- class [ActivityTracker](#)
- class [CreateUserWindow](#)
- class [DBManager](#)
- class [Iteration3Test](#)
- class [LoginWindow](#)
- class [MainWindow](#)
- class [Run](#)
- enum [RunAttribute](#)
- class [SecureString](#)
- class [User](#)
- enum [UserAttribute](#)

Chapter 7

Class Documentation

7.1 com.activitytracker.ActivityTracker Class Reference

Collaboration diagram for com.activitytracker.ActivityTracker:



Static Public Member Functions

- static void `main` (final String[] args)

7.1.1 Detailed Description

The main program class.

Definition at line 28 of file ActivityTracker.java.

7.1.2 Member Function Documentation

7.1.2.1 main()

```
static void com.activitytracker.ActivityTracker.main (
    final String [] args ) [static]
```

The main program entry point.

Definition at line 33 of file ActivityTracker.java.

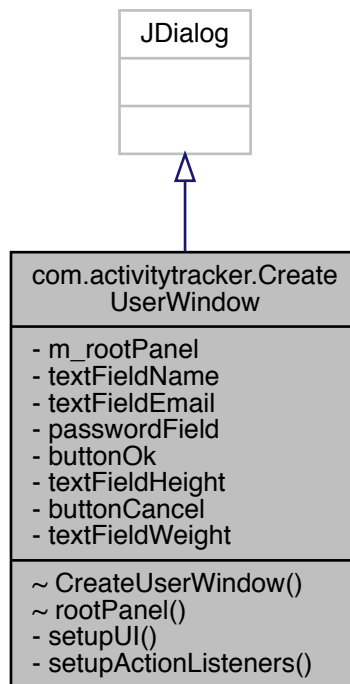
```
33         {
34
35         // Create singleton instance of DBManager
36         DBManager dbManager = new DBManager();
37         if (!dbManager.init("data.db")) {
38             System.err.println("Failed to initialize DBManager");
39             System.exit(1);
40         }
41
42         // Set Look and Feel
43         try {
44             UIManager.setLookAndFeel(new MaterialLookAndFeel());
45         }
46         catch (final UnsupportedLookAndFeelException e) {
47             e.printStackTrace();
48         }
49         // Get desktop resolution of default monitor (in case of multi-monitor setups)
50         final GraphicsDevice gd = GraphicsEnvironment.getLocalGraphicsEnvironment().getDefaultScreenDevice(
51     );
52
53         final JFrame frame = new JFrame("Activity Logger");
54
55         final String logoPath = "./assets/logo.png";
56         ImageIcon imgIcon = new ImageIcon(ActivityTracker.class.getResource(logoPath));
57         frame.setIconImage(imgIcon.getImage());
58         frame.setContentPane(new LoginWindow((Void) -> {
59             frame.setContentPane(new MainWindow().rootPanel());
60             frame.validate();
61             frame.repaint();
62         }).rootPanel());
63         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
64         frame.pack();
65
66         // Set window size to be 1/2 of screen dimensions
67         frame.setSize(gd.getDisplayMode().getWidth() / 2, gd.getDisplayMode().getHeight() / 2);
68         frame.setLocationRelativeTo(null); // Center window
69         frame.setVisible(true);
70     }
```

The documentation for this class was generated from the following file:

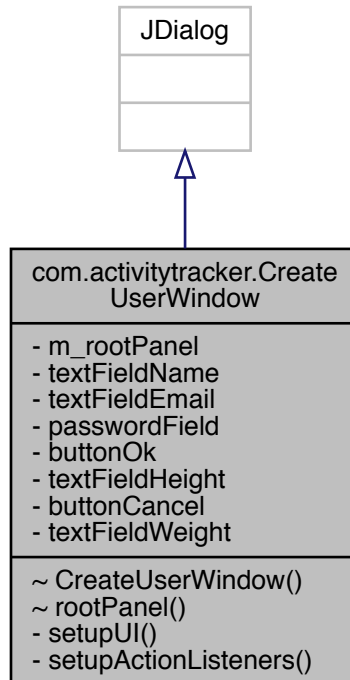
- [app/src/com/activitytracker/ActivityTracker.java](#)

7.2 com.activitytracker.CreateUserWindow Class Reference

Inheritance diagram for com.activitytracker.CreateUserWindow:



Collaboration diagram for `com.activitytracker.CreateUserWindow`:



Package Functions

- [CreateUserWindow \(\)](#)
- `JPanel` [rootPanel \(\)](#)

Private Member Functions

- void [setupUI \(\)](#)
- void [setupActionListeners \(\)](#)

Private Attributes

- `JPanel` [m_rootPanel](#)
- `JTextField` [textFieldName](#)

- JTextField [textFieldEmail](#)
- JPasswordField [passwordField](#)
- JButton [buttonOk](#)
- JTextField [textFieldHeight](#)
- JButton [buttonCancel](#)
- JTextField [textFieldWeight](#)

7.2.1 Detailed Description

Definition at line 10 of file CreateUserWindow.java.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 CreateUserWindow()

com.activitytracker.CreateUserWindow.CreateUserWindow () [package]

Definition at line 20 of file CreateUserWindow.java.

```
20         {
21
22     setupUI();
23     setupActionListeners();
24 }
```

7.2.3 Member Function Documentation

7.2.3.1 rootPanel()

JPanel com.activitytracker.CreateUserWindow.rootPanel () [package]

Definition at line 53 of file CreateUserWindow.java.

```
53         {
54     return m_rootPanel;
55 }
```

7.2.3.2 setupActionListeners()

void com.activitytracker.CreateUserWindow.setupActionListeners () [private]

Definition at line 31 of file CreateUserWindow.java.

```
31         {
32         buttonOk.addActionListener(new ActionListener() {
33             @Override
34             public void actionPerformed(ActionEvent e) {
35
36                 if (textFieldName.getText().isEmpty() ||
37                     textFieldEmail.getText().isEmpty() ||
38                     passwordField.getPassword().length == 0) {
39
40                     return;
41                 }
42             }
43         });
44
45         buttonCancel.addActionListener(new ActionListener() {
46             @Override
47             public void actionPerformed(ActionEvent e) {
48
49             }
50         });
51     }
```

7.2.3.3 setupUI()

void com.activitytracker.CreateUserWindow.setupUI () [private]

Definition at line 26 of file CreateUserWindow.java.

```
26         {
27         MaterialUIMovement.add(buttonCancel, MaterialColors.GRAY_100);
28         MaterialUIMovement.add(buttonOk, MaterialColors.GRAY_100);
29     }
```

7.2.4 Member Data Documentation

7.2.4.1 buttonCancel

`JButton com.activitytracker.CreateUserWindow.buttonCancel [private]`

Definition at line 17 of file CreateUserWindow.java.

7.2.4.2 buttonOk

`JButton com.activitytracker.CreateUserWindow.buttonOk [private]`

Definition at line 15 of file CreateUserWindow.java.

7.2.4.3 m_rootPanel

`JPanel com.activitytracker.CreateUserWindow.m_rootPanel [private]`

Definition at line 11 of file CreateUserWindow.java.

7.2.4.4 passwordField

`JPasswordField com.activitytracker.CreateUserWindow.passwordField [private]`

Definition at line 14 of file CreateUserWindow.java.

7.2.4.5 textFieldEmail

`JTextField com.activitytracker.CreateUserWindow.textFieldEmail [private]`

Definition at line 13 of file CreateUserWindow.java.

7.2.4.6 textFieldHeight

`TextField com.activitytracker.CreateUserWindow.textFieldHeight [private]`

Definition at line 16 of file CreateUserWindow.java.

7.2.4.7 textFieldName

`TextField com.activitytracker.CreateUserWindow.textFieldName [private]`

Definition at line 12 of file CreateUserWindow.java.

7.2.4.8 textFieldWeight

`TextField com.activitytracker.CreateUserWindow.textFieldWeight [private]`

Definition at line 18 of file CreateUserWindow.java.

The documentation for this class was generated from the following file:

- [app/src/com/activitytracker/CreateUserWindow.java](#)

7.3 com.activitytracker.DBManager Class Reference

Collaboration diagram for com.activitytracker.DBManager:

com.activitytracker.DBManager
- m_conn
+ createUser() + userExists() + getUserIDByEmail() + getUserStringAttribute() + getUserFloatAttribute() + getDateOfBirth() + getUserSex() + getUserPassSalt() + getUserLastRID() + setUserLastRID() + newRun() + setRun() + getRunFloatAttribute() + runExists() + getRuns() ~ DBManager() ~ init() - executeQuery() - executeUpdate() - isEmpty()

Public Member Functions

- void [createUser](#) (final String name, final String emailAddress, final int DOBYear, final int DOBMonth, final int DOBDay, final User.Sex sex, final float height, final float weight, final [SecureString](#) securePassword) throws AssertionError
- boolean [userExists](#) (final String emailAddress)
- int [getUserIDByEmail](#) (final String emailAddress)
- String [getUserStringAttribute](#) (final [UserAttribute](#) attribute, final int id)
- float [getUserFloatAttribute](#) (final [UserAttribute](#) attribute, final int id)
- Date [getDateOfBirth](#) (final int id)
- User.Sex [getUserSex](#) (final int id)
- byte [] [getUserPassSalt](#) (final int id)
- int [getUserLastRID](#) (final int id)

- void [setUserLastRID](#) (final int id, final int lastRID)
- int [newRun](#) (final int userID, final java.util.Date date, final float duration, final float distance, final float altitude_ascending, final float altitude_descending)
- void [setRun](#) (final int rID, final float duration, final float distance, final float altitude_ascending, final float altitude_descending)
- float [getRunFloatAttribute](#) (final [RunAttribute](#) attribute, final int rID)
- boolean [runExists](#) (final int rID)
- Vector< Integer > [getRuns](#) (final int userID, final java.util.Date startDate, final java.util.Date endDate)

Package Functions

- [DBManager](#) ()
- boolean [init](#) (final String dbURL)

Private Member Functions

- ResultSet [executeQuery](#) (final String sqlQuery)
- boolean [executeUpdate](#) (final String sqlQuery)
- boolean [isEmpty](#) ()

Private Attributes

- Connection [m_conn](#) = null

7.3.1 Detailed Description

Singleton class for the database. All classes and methods that interact with the database will use a method in this class.

Many times we are faced with the "chicken and egg" problem where we wish to create an object that is populated with information from the database. So the question one faces is, "does the object's constructor query the database (through the [DBManager](#) class, of course) for each attribute of the object that it wishes to retrieve, or do we directly interact with a [DBManager](#) method which will then return a [User](#) or [Run](#) object, for example?" We have decided to use the former methodology, with [DBManager](#) methods being as general as possible, and often accepting enum types which then are put into a switch to create the specific SQL query we wish to execute. This works best when all data returned is of the same data type (for example, the Workout class will have three float attributes at the time of writing so we use one method with return type of float for returning

Workout attributes). This does not work as well when the object requires data of multiple types — for example, the [User](#) class. In this case, we have split the [DBManager](#) methods into a single method for each attribute being returned.

Polymorphism could theoretically be used here to simply have a return type of `Object`, however this is not flexible and requires casting *all* returned data to the correct type in the invoking method.

Definition at line 29 of file `DBManager.java`.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 DBManager()

```
com.activitytracker.DBManager.DBManager ( ) [package]
```

Creates a new [DBManager](#) object.

This should only be called once, from the main program, as [DBManager](#) is meant to be a *singleton* class.

This constructor takes no parameters as verification of the SQLite database is done in the [init\(\)](#) method of this class, which returns information about whether the initialization was successful or not.

Definition at line 46 of file `DBManager.java`.

```
46         {  
47     }
```

7.3.3 Member Function Documentation

7.3.3.1 createUser()

```
void com.activitytracker.DBManager.createUser (
    final String name,
    final String emailAddress,
    final int DOBYear,
    final int DOBMonth,
    final int DOBDay,
    final User.Sex sex,
    final float height,
    final float weight,
    final SecureString securePassword ) throws AssertionError
```

Adds a row for a user to the Users table in the SQLite database for the app.

Requires that the database tables exist and are in the correct format. If the user exists in the database this method raises an AssertionError exception.

Parameters

<i>name</i>	User's name
<i>emailAddress</i>	User's email address; used to authenticate
<i>DOBYear</i>	The year the user was born
<i>DOBMonth</i>	The month the user was born
<i>DOBDay</i>	The day of month the user was born
<i>sex</i>	The user's sex; is either User.Sex.MALE or User.Sex.FEMALE
<i>height</i>	Floating point number of the user's height in metres
<i>weight</i>	Floating point number of the user's weight in kilograms
<i>securePassword</i>	A SecureString object containing the user's password, encrypted

Definition at line 65 of file DBManager.java.

```
67
68
69     if (!userExists(emailAddress)) {
70         String sqlQuery = "INSERT INTO Users (" +
71             "email_address, " +
72             "name, " +
73             "date_of_birth, " +
74             "sex, " +
75             "height, " +
76             "weight," +
77             "password_hash," +
78             "password_salt," +
79             "created_at" +
80             ") VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
81         byte sexByte = sex.equals(User.Sex.MALE) ? (byte) 1 : (byte) 0;
82         java.sql.Date currentTime = new java.sql.Date(System.currentTimeMillis());
```

{

```

83         Calendar c = Calendar.getInstance();
84         c.set(DOBYear, DOBMonth, DOBDay);
85         java.sql.Date dateOfBirth = new java.sql.Date(
86             c.get(Calendar.YEAR),
87             c.get(Calendar.MONTH),
88             c.get(Calendar.DAY_OF_MONTH)
89         );
90
91         try {
92             PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
93             stmt.setString(1, emailAddress);
94             stmt.setString(2, name);
95             stmt.setDate(3, dateOfBirth);
96             stmt.setByte(4, sexByte);
97             stmt.setFloat(5, height);
98             stmt.setFloat(6, weight);
99             stmt.setString(7, securePassword.toString());
100            stmt.setBytes(8, securePassword.getSalt());
101            stmt.setDate(9, currentTime);
102
103            if (stmt.executeUpdate() != 1) {
104                System.err.println("User not added to database.");
105            }
106
107            stmt.close();
108        }
109        catch (final SQLException e) {
110            System.err.println(e.getMessage());
111        }
112    }
113    else {
114        throw new AssertionError("User with email address '" + emailAddress + "' already exists.");
115    }
116 }

```

7.3.3.2 executeQuery()

```

ResultSet com.activitytracker.DBManager.executeQuery (
    final String sqlQuery ) [private]

```

A wrapper method for processing *safe* SQL queries.

By safe we mean that the SQL query string is entirely hard-coded in the program source code. In other words, no user input is added. This is an important distinction as the former may leave the application vulnerable to SQL injection.

In such cases, a SQL PreparedStatement should be used.

Parameters

<i>sqlQuery</i>	The SQL code to be executed. Must be a <i>SELECT</i> statement.
-----------------	---

Returns

This method returns a `ResultSet` containing the returned row(s) and/or column(s) of the SQL query that was executed.

Definition at line 719 of file `DBManager.java`.

```
719                                     {
720         ResultSet res = null;
721
722         try {
723             Statement stmt = m_conn.createStatement();
724             res = stmt.executeQuery(sqlQuery);
725             stmt.close();
726         }
727         catch (final SQLException e) {
728             System.err.println(e.getMessage());
729         }
730
731         return res;
732     }
```

7.3.3.3 executeUpdate()

```
boolean com.activitytracker.DBManager.executeUpdate (
    final String sqlQuery ) [private]
```

A wrapper method for processing *safe* SQL queries.

By safe we mean that the SQL query string is entirely hard-coded in the program source code. In other words, no user input is added. This is an important distinction as the former may leave the application vulnerable to SQL injection.

In such cases, a SQL `PreparedStatement` should be used.

Parameters

<i>sqlQuery</i>	The SQL code to be executed. Must be an <i>INSERT</i> or <i>UPDATE</i> statement.
-----------------	---

Returns

This method returns a boolean indicating if the query was successful.

Definition at line 747 of file `DBManager.java`.

```

747                                     {
748         try {
749             Statement stmt = m_conn.createStatement();
750             stmt.executeUpdate(sqlQuery);
751             stmt.close();
752         }
753         catch (final SQLException e) {
754             System.err.println(e.getMessage());
755             return false;
756         }
757
758         return true;
759     }

```

7.3.3.4 getDateOfBirth()

Date com.activitytracker.DBManager.getDateOfBirth (
 final int id)

Retrieves the user's date of birth (DOB) from the database.

At the time of writing, this method is only being used in the [User](#) constructor.

Parameters

<i>id</i>	Unique ID used to associate information in the database to this user.
-----------	---

Returns

This method returns a Date object containing the user's DOB (i.e., year, month, day).

Definition at line 302 of file DBManager.java.

```

302                                     {
303         Date DOB;
304         java.sql.Date DOBResult;
305         ResultSet res;
306         String sqlQuery = "SELECT date_of_birth FROM Users WHERE id=?";
307         try {
308             PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
309             stmt.setInt(1, id);
310             res = stmt.executeQuery();
311             DOBResult = res.getDate("date_of_birth");
312
313             stmt.close();
314         }
315         catch (final SQLException e) {
316             System.err.println(e.getMessage());
317             return null;

```

```

318     }
319     DOB = new Date(DOBResult.getYear(), DOBResult.getMonth(), DOBResult.getDay());
320
321     return DOB;
322 }

```

7.3.3.5 getRunFloatAttribute()

```

float com.activitytracker.DBManager.getRunFloatAttribute (
    final RunAttribute attribute,
    final int rID )

```

Retrieves a run's attribute as a floating point number, where applicable, from the database.

This method accepts a [RunAttribute](#) enumeration type to specify what attribute it is returning from the database. Only certain attributes are accepted by this method, namely those that are stored as real values. Attributes stored as other data types should use the appropriate accessor method.

Parameters

<i>attribute</i>	<p>The attribute that the method is supposed to query the DB for and return the value of. Note that only certain RunAttribute types are supported in this method.</p> <ul style="list-style-type: none"> • When <i>attribute</i> is RunAttribute.DURATION, the run's duration is returned. • When <i>attribute</i> is RunAttribute.DISTANCE, the run's cumulative distance is returned in metres. • When <i>attribute</i> is RunAttribute.ALTITUDE_ASCENDED, the run's cumulative altitude climbed is returned in metres • When <i>attribute</i> is RunAttribute.ALTITUDE_DESCENDED, the run's cumulative altitude descended is returned in metres
<i>rID</i>	<p>Unique ID corresponding to the row in the Runs table that we wish to query. If such an ID does not exist, <i>0.0f</i> will be returned.</p>

Returns

This method returns a float containing run attribute as specified by the *attribute* parameter.

Definition at line 588 of file DBManager.java.


```

588                                     {
589         ResultSet res;
590         PreparedStatement stmt;
591         String sqlQuery, columnLabel;
592         float attrVal = 0.0f;
593         switch (attribute) {
594             case DURATION:
595                 columnLabel = "duration";
596                 sqlQuery = "SELECT " + columnLabel + " FROM Runs WHERE id=?";
597                 break;
598             case DISTANCE:
599                 columnLabel = "distance";
600                 sqlQuery = "SELECT " + columnLabel + " FROM Runs WHERE id=?";
601                 break;
602             case ALTITUDE_ASCENDED:
603                 columnLabel = "altitude_ascended";
604                 sqlQuery = "SELECT " + columnLabel + " FROM Runs WHERE id=?";
605                 break;
606             case ALTITUDE_DESCENDED:
607                 columnLabel = "altitude_descended";
608                 sqlQuery = "SELECT " + columnLabel + " FROM Runs WHERE id=?";
609                 break;
610             default:
611                 return attrVal;
612         }
613         if (runExists(rID)) {
614             try {
615                 stmt = m_conn.prepareStatement(sqlQuery);
616                 stmt.setInt(1, rID);
617                 res = stmt.executeQuery();
618                 attrVal = res.getFloat(columnLabel);
619             }
620             catch (final SQLException e) {
621                 System.err.println(e.getMessage());
622             }
623         }
624         else {
625             System.err.println("Run " + Integer.toString(rID) + " does not exist. Cannot get " +
columnLabel + ".");
626         }
627
628         return attrVal;
629
630     }

```

7.3.3.6 getRuns()

```

Vector<Integer> com.activitytracker.DBManager.getRuns (
    final int userID,
    final java.util.Date startDate,
    final java.util.Date endDate )

```

Queries the database for all runs by a user with user ID *userID* between *startDate* and *endDate*.

Parameters

<i>userID</i>	The ID of the user whose runs we wish to retrieve.
<i>startDate</i>	The lower bound of the interval we wish to retrieve runs for.
<i>endDate</i>	The upper bound of the interval we wish to retrieve runs for.

Returns

Returns a vector containing run IDs for each run that meets the search criteria.

Definition at line 678 of file DBManager.java.

```
678
    {
679        ResultSet res;
680        Vector<Integer> runs = new Vector<>();
681        try {
682            PreparedStatement stmt = m_conn.prepareStatement(
683                "SELECT id FROM Runs WHERE user_id=? AND date BETWEEN ? AND ?;");
684            stmt.setInt(1, userID);
685            stmt.setLong(2, startDate.getTime());
686            stmt.setLong(3, endDate.getTime());
687
688            res = stmt.executeQuery();
689
690            if (res.isClosed())
691                System.err.println("Result set closed; cannot get any data?");
692
693            while (res.next()) {
694                runs.add(res.getInt("id"));
695            }
696            stmt.close();
697        }
698        catch (final SQLException e) {
699            System.err.println(e.getMessage());
700        }
701
702        return runs;
703    }
```

7.3.3.7 getUserFloatAttribute()

```
float com.activitytracker.DBManager.getUserFloatAttribute (
    final UserAttribute attribute,
    final int id )
```

Retrieves a user's attribute in floating point format, when applicable, from the database's Users table.

This method accepts a [UserAttribute](#) enumeration type to specify what attribute it is returning from the database. Only certain attributes are accepted by this method, namely those that are stored as real values. Attributes stored as other data types should use the appropriate accessor method.

Parameters

<i>attribute</i>	<p>The attribute that the method is supposed to query the DB for and return the value of. Note that only certain UserAttribute types are supported in this method.</p> <ul style="list-style-type: none"> When <i>attribute</i> is UserAttribute.WEIGHT, this method retrieves the user's weight from the database. When <i>attribute</i> is UserAttribute.HEIGHT, this method retrieves the user's height from the database.
<i>id</i>	Unique ID used to associate information in the database to this user.

Returns

Returns a floating point number corresponding to the [UserAttribute](#) passed to the method, for the user specified by *id*.

Definition at line 261 of file DBManager.java.

```

261                                     {
262         float attrVal;
263         ResultSet res;
264         String sqlQuery, columnLabel;
265         switch (attribute) {
266             case WEIGHT:
267                 columnLabel = "weight";
268                 sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";
269                 break;
270             case HEIGHT:
271                 columnLabel = "height";
272                 sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";
273                 break;
274             default:
275                 throw new AssertionError("Incorrect UserAttribute enumeration type passed to method.");
276         }
277         try {
278             PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
279             stmt.setInt(1, id);
280             res = stmt.executeQuery();
281             attrVal = res.getFloat(columnLabel);
282
283             stmt.close();
284         }
285         catch (final SQLException e) {
286             System.err.println(e.getMessage());
287             return 0.0f;
288         }
289
290         return attrVal;
291     }

```

7.3.3.8 getUserIDByEmail()

```
int com.activitytracker.DBManager.getUserIDByEmail (
    final String emailAddress )
```

As we are using the user's email address as their identifying attribute, they will supply this when they log in. Hence, as the database relates everything to the user's unique ID, we must retrieve this ID given the email address.

The logic behind this method relies on the database Users table structure making *email_address* a unique field.

Parameters

<i>emailAddress</i>	The user's email address with which they authenticate.
---------------------	--

Returns

This method returns a unique integer corresponding to the row in the database's Users table that stores user information for user with email address *emailAddress*.

Definition at line 159 of file DBManager.java.

```
159                                     {
160     int id = 0;
161     ResultSet res;
162     String sqlQuery = "SELECT id FROM Users WHERE 'email_address'=?";
163     try {
164         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
165         stmt.setString(1, emailAddress);
166         res = stmt.executeQuery();
167         id = res.getInt("id");
168     }
169     stmt.close();
170 }
171 catch (final SQLException e) {
172     System.err.println(e.getMessage());
173 }
174
175 return id;
176 }
```

7.3.3.9 getUserLastRID()

```
int com.activitytracker.DBManager.getUserLastRID (
    final int id )
```

Retrieves the last workout ID that the user added as an integer from the database.

This is used because of the format in which the data is supplied. As the only way to denote a new workout is by receiving (0, 0, 0) in the input file, if the input is *not* (0, 0, 0), we need to update the previously added workout with the latest line. Hence we need some way of storing an identifier for this workout. As this is unique to each user, we have chosen to store this in the Users table of the database.

Parameters

<i>id</i>	Unique ID used to associate information in the database to this user.
-----------	---

Returns

An integer corresponding to the last row in the Workouts table that the user created.

Definition at line 399 of file DBManager.java.

```
399                                     {
400     int rID = 0;
401     ResultSet res;
402     String columnLabel = "last_run";
403     String sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";
404     try {
405         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
406         stmt.setInt(1, id);
407         res = stmt.executeQuery();
408         rID = res.getInt(columnLabel);
409         stmt.close();
410     }
411     catch (final SQLException e) {
412         System.err.println(e.getMessage());
413     }
414
415     return rID;
416 }
```

7.3.3.10 getUserPassSalt()

```
byte [] com.activitytracker.DBManager.getUserPassSalt (  
    final int id )
```

Retrieves a byte array containing the salt used to encrypt the user's password from the database.

This is necessary because to compare a candidate password supplied by a user to a known (encrypted) password stored in the database, we must encrypt the new candidate password using the same salt as was originally used.

Parameters

<i>id</i>	Unique ID used to associate information in the database to this user.
-----------	---

Returns

This method returns a byte array containing the user's password encryption salt.

Definition at line 370 of file DBManager.java.

```

370                                     {
371     byte[] passSalt;
372     ResultSet res;
373     String sqlQuery = "SELECT password_salt FROM Users WHERE id=?";
374     try {
375         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
376         stmt.setInt(1, id);
377         res = stmt.executeQuery();
378         passSalt = res.getBytes("password_salt");
379         stmt.close();
380     }
381     catch (final SQLException e) {
382         System.err.println(e.getMessage());
383         return null;
384     }
385     return passSalt;
386 }
```

7.3.3.11 getUserSex()

```
User.Sex com.activitytracker.DBManager.getUserSex (
    final int id )
```

Retrieves the user's gender from the database.

We have chosen to represent gender in the SQLite database with the data type BIT(1), where 1 denotes male and 0 denotes female. Hence, if the database contains 1 this method returns [User.Sex.MALE](#) and if the database contains 0 then this method returns [User.Sex.FEMALE](#).

At the time of writing, this method is only being used in the [User](#) constructor.

Parameters

<i>id</i>	Unique ID used to associate information in the database to this user.
-----------	---

Returns

This method returns a [User.Sex](#) enumeration type corresponding to the user's gender.

Definition at line 337 of file DBManager.java.

```
337                                     {
338         byte sex;
339         ResultSet res;
340         String sqlQuery = "SELECT sex FROM Users WHERE id=?";
341         try {
342             PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
343             stmt.setInt(1, id);
344             res = stmt.executeQuery();
345             sex = res.getBytes("sex");
346
347             stmt.close();
348         }
349         catch (final SQLException e) {
350             System.err.println(e.getMessage());
351             return null;
352         }
353
354         if (sex == (byte) 1)
355             return User.Sex.MALE;
356         else
357             return User.Sex.FEMALE;
358     }
```

7.3.3.12 getUserStringAttribute()

```
String com.activitytracker.DBManager.getUserStringAttribute (
    final UserAttribute attribute,
    final int id )
```

This method retrieves a string, varchar, text, or char field, when applicable, from the database's Users table.

This method accepts a [UserAttribute](#) enumeration type to specify what attribute it is returning from the database. Only certain attributes are accepted by this method, namely those that are stored as string-like values. Attributes stored as other data types should use the appropriate accessor method.

Parameters

<i>attribute</i>	<p>The attribute that the method is supposed to query the DB for and return the value of. Note that only certain UserAttribute types are supported in this method.</p> <ul style="list-style-type: none"> When <i>attribute</i> is UserAttribute.PASSWORD, this method retrieves the user's encrypted password from the database. Typically this will be used in the following sequence of calls: <ol style="list-style-type: none"> User attempts to authenticate with email and password Their unique ID is retrieved from the database using DBManager::getUserIDByEmail() Their ID is used to retrieve the hash of their password (i.e., this method is called) The returned string from this method is compared a SecureString generated from the candidate password supplied by the user when authenticating. When <i>attribute</i> is UserAttribute.NAME, this method retrieves the user's full name from the database (e.g., "John Doe"). When <i>attribute</i> is UserAttribute.EMAIL_ADDRESS, this method retrieves the user's email address from the database. Note that this is likely somewhat redundant as the user will always be required to authenticate by providing their email address and hence it will already be available to the User constructor, which is likely what is invoking this method.
<i>id</i>	Unique ID used to associate information in the database to this user.

Returns

This method returns a string containing attribute specified by the *attribute* parameter for the user specified by the *id* parameter.

Definition at line 207 of file DBManager.java.

```

207                                     {
208     String name;
209     ResultSet res;
210     String sqlQuery, columnLabel;
211     switch (attribute) {
212     case PASSWORD:
213         columnLabel = "password_hash";
214         sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";
215         break;
216     case NAME:
217         columnLabel = "name";
218         sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";

```

```

219         break;
220     case EMAIL_ADDRESS:
221         columnLabel = "email_address";
222         sqlQuery = "SELECT " + columnLabel + " FROM Users WHERE id=?";
223         break;
224     default:
225         throw new AssertionError("Incorrect UserAttribute enumeration type passed to method.");
226     }
227     try {
228         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
229         stmt.setInt(1, id);
230         res = stmt.executeQuery();
231         name = res.getString(columnLabel);
232
233         stmt.close();
234     }
235     catch (final SQLException e) {
236         System.err.println(e.getMessage());
237         return null;
238     }
239
240     return name;
241 }

```

7.3.3.13 init()

```
boolean com.activitytracker.DBManager.init (
    final String dbURL ) [package]
```

Initializes a connection to the SQLite database.

As no work is done in the [DBManager\(\)](#) constructor, this method should be called immediately after creating the single instance of [DBManager](#) that the application is to use.

This method will attempt to connect to the database file specified by the *dbURL* parameter, creating the file and all required tables if it/they do not exist. You are encouraged to view the source code of this method for more information about the database schema used.

If all of the above is successful, the method returns True. Otherwise, False is returned.

Parameters

<i>dbURL</i>	A file system path to the SQLite database file.
--------------	---

Returns

This method returns True if the database can be initialized, or False otherwise.

Definition at line 801 of file DBManager.java.

```

801         {
802     try {
803         m_conn = DriverManager.getConnection("jdbc:sqlite:" + dbURL);
804     }
805     catch (final SQLException e) {
806         System.err.println(e.getMessage());
807         return false;
808     }
809     System.out.println("Opened database successfully.");
810
811     if (isEmpty()) {
812         System.out.println("Creating tables...");
813
814         // Create users table
815         String sqlQuery = "CREATE TABLE USERS (" +
816             "    id            INTEGER PRIMARY KEY ASC AUTOINCREMENT NOT NULL," +
817             "    email_address STRING NOT NULL UNIQUE ON CONFLICT FAIL," +
818             "    name          STRING NOT NULL," +
819             "    date_of_birth DATETIME NOT NULL," +
820             "    sex           BIT(1) NOT NULL," +
821             "    height        REAL NOT NULL," +
822             "    weight        REAL NOT NULL," +
823             "    password_hash STRING NOT NULL," +
824             "    password_salt BLOB NOT NULL," +
825             "    last_run      INTEGER NOT NULL DEFAULT 0," +
826             "    created_at    DATETIME NOT NULL" +
827             ")";
828
829         if (!executeUpdate(sqlQuery)) {
830             return false;
831         }
832
833         // Create workouts table
834         sqlQuery = "CREATE TABLE RUNS (" +
835             "    id            INTEGER PRIMARY KEY ASC AUTOINCREMENT NOT NULL," +
836             "    user_id      INTEGER NOT NULL REFERENCES USERS (id)," +
837             "    date          DATETIME NOT NULL," +
838             "    duration      REAL NOT NULL," + // seconds
839             "    distance      REAL NOT NULL," + // metres
840             "    altitude_ascended REAL NOT NULL," + // metres
841             "    altitude_descended REAL NOT NULL" + // metres
842             ")";
843
844         if (!executeUpdate(sqlQuery)) {
845             return false;
846         }
847
848         // Create friends table
849         sqlQuery = "CREATE TABLE FRIENDS (" +
850             "    id            INTEGER PRIMARY KEY ASC AUTOINCREMENT NOT NULL," +
851             "    sender        INTEGER NOT NULL REFERENCES USERS (id)," +
852             "    receiver      INTEGER REFERENCES USERS (id)," +
853             "    send_date     DATETIME NOT NULL," +
854             "    confirm_date  DATETIME DEFAULT NULL" +
855             ")";
856
857         if (!executeUpdate(sqlQuery)) {
858             return false;
859         }
860     }
861 }
862
863 return true;
864 }

```

7.3.3.14 isEmpty()

`boolean com.activitytracker.DBManager.isEmpty () [private]`

Returns a boolean value depending on whether or not the database is populated.

This is done by retrieving tables in the database and checking if this iterator has a `next()`. If not then there are no tables in the database and we consider it to be empty.

Returns

Returns True if there are tables in the database, False otherwise.

Definition at line 769 of file DBManager.java.

```
769             {
770
771         try {
772             final DatabaseMetaData dbmd = m_conn.getMetaData();
773             final String[] types = {"TABLE"};
774             final ResultSet rs = dbmd.getTables(null, null, "%", types);
775
776             return !rs.next();
777         }
778         catch (final SQLException e) {
779             System.err.println(e.getMessage());
780             return true;
781         }
782     }
783 }
```

7.3.3.15 newRun()

```
int com.activitytracker.DBManager.newRun (
    final int userID,
    final java.util.Date date,
    final float duration,
    final float distance,
    final float altitude_ascended,
    final float altitude_descended )
```

Creates a new row in the Runs table with the attributes provided as parameters.

In particular, this method will be called when [Run::newRunDataPoint\(\)](#) receives (0, 0, 0) for (*duration*, *distance*, *altitude*).

Parameters

<i>userID</i>	Unique ID used to associate information in the database to this user.
<i>date</i>	Date that the run was completed.
<i>duration</i>	Duration of the run in seconds.
<i>distance</i>	Distance ran in metres.
<i>altitude_ascended</i>	Cumulative altitude climbed in metres.
<i>altitude_descended</i>	Cumulative altitude descended in metres.

Returns

Returns a unique integer corresponding to the new row in the SQLite Workouts table by which the new entry can be identified.

Definition at line 461 of file DBManager.java.

```

462                                     {
463
464         java.sql.Date date1 = new java.sql.Date(date.getYear(), date.getMonth(), date.getDay());
465
466         int rID = 0;
467         ResultSet res;
468         String sqlInsertQuery = "INSERT INTO Runs (" +
469             "user_id," +
470             "date," +
471             "duration," +
472             "distance," +
473             "altitude_ascended," +
474             "altitude_descended" +
475             ") VALUES (?, ?, ?, ?, ?, ?, ?)";
476         String sqlSelectQuery = "SELECT id FROM Runs WHERE " +
477             "user_id=? AND " +
478             "date=? AND " +
479             "duration=? AND " +
480             "distance=? AND " +
481             "altitude_ascended=? AND " +
482             "altitude_descended=?";
483
484         try {
485             PreparedStatement stmt = m_conn.prepareStatement(sqlInsertQuery);
486             stmt.setInt(1, userID);
487             stmt.setDate(2, date1);
488             stmt.setFloat(3, duration);
489             stmt.setFloat(4, distance);
490             stmt.setFloat(5, altitude_ascended);
491             stmt.setFloat(6, altitude_descended);
492
493             if (stmt.executeUpdate() != 1) {
494                 System.err.println("Run not added to database.");
495             }
496
497             stmt.close();
498
499             // Pass back in the stuff we just created to get the right row ID
500             // Look at a better way of doing this with OUTPUT clause of INPUT statement
501             stmt = m_conn.prepareStatement(sqlSelectQuery);
502             stmt.setInt(1, userID);

```

```

503         stmt.setDate(2, date1);
504         stmt.setFloat(3, duration);
505         stmt.setFloat(4, distance);
506         stmt.setFloat(5, altitude_ascending);
507         stmt.setFloat(6, altitude_descending);
508
509         res = stmt.executeQuery();
510         rID = res.getInt("id");
511     }
512     catch (final SQLException e) {
513         System.err.println(e.getMessage());
514     }
515
516     return rID;
517 }

```

7.3.3.16 runExists()

```
boolean com.activitytracker.DBManager.runExists (
    final int rID )
```

Determines if a given run ID exists in the database.

Parameters

<i>rID</i>	Unique ID corresponding to the row in the Runs table that we wish to check exists.
------------	--

Returns

This method returns True if the run row with ID *WOID* exists in the database, or False otherwise.

Definition at line 639 of file DBManager.java.

```

639         {
640         ResultSet res;
641         String sqlQuery = "SELECT COUNT(*) as count FROM Runs WHERE id=?";
642         boolean exists = false;
643         try {
644             PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
645             stmt.setInt(1, rID);
646             res = stmt.executeQuery();
647             switch (res.getInt("count")) {
648                 case 0:
649                     exists = false;
650                     break;
651                 case 1:
652                     exists = true;
653                     break;

```

```

654         default:
655             exists = true;
656             System.err.println("More than one run for ID " +
657                 Integer.toString(rID) + ". Something isn't right.");
658             break;
659     }
660
661 }
662 catch (final SQLException e) {
663     System.err.println(e.getMessage());
664 }
665
666 return exists;
667 }

```

7.3.3.17 setRun()

```

void com.activitytracker.DBManager.setRun (
    final int rID,
    final float duration,
    final float distance,
    final float altitude_ascended,
    final float altitude_descended )

```

Updates a run entry in the database as new information becomes available from the input file.

In particular, this method is called when [Run::newRunDataPoint\(\)](#) receives non-(0, 0, 0) input for (*duration*, *distance*, *altitude*).

This method will not be called directly by the application, rather it is called from [Run::newRunDataPoint\(\)](#). Hence that method will take care of adding/subtracting to/from the current stored values for *duration*, *distance*, and *altitude* — here we just take the input and put it in the database.

Parameters

<i>rID</i>	Unique ID used to identify a run in the database.
<i>duration</i>	The number of seconds the user's run lasted.
<i>distance</i>	The cumulative number of metres the user ran.
<i>altitude_ascended</i>	The cumulative number of metres the user climbed.
<i>altitude_descended</i>	The cumulative number of metres the user descended.

Definition at line 536 of file DBManager.java.

537

{

```

538     String sqlQuery = "UPDATE Runs SET " +
539         "duration = ?, " +
540         "distance = ?, " +
541         "altitude_ascended=?, " +
542         "altitude_descended=? " +
543         "WHERE id=? ";
544     try {
545         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
546         stmt.setFloat(1, duration);
547         stmt.setFloat(2, distance);
548         stmt.setFloat(3, altitude_ascended);
549         stmt.setFloat(4, altitude_descended);
550         stmt.setInt(5, rID);
551
552         int result = stmt.executeUpdate();
553         System.err.println(Integer.toString(result) + " rows updated in setRun().");
554         if (result != 1) {
555             System.err.println("Run not updated in database.");
556         }
557
558         stmt.close();
559     }
560     catch (final SQLException e) {
561         System.err.println(e.getMessage());
562     }
563
564 }

```

7.3.3.18 setUserLastRID()

```

void com.activitytracker.DBManager.setUserLastRID (
    final int id,
    final int lastRID )

```

Updates a user's last run ID in the database.

This method will be used to update the run that a particular user last created. This is used when creating new run as the format of the input file requires that we maintain a record of what run we must update if the next line in the file is *not* (0, 0, 0).

See [getUserLastRID\(\)](#) for more information on the user of the *last_run* field in the database.

Parameters

<i>id</i>	Unique ID used to associate information in the database to this user.
<i>lastRID</i>	Integer corresponding to the last row in the Workouts table that the user with ID <i>id</i> created.

Definition at line 430 of file DBManager.java.


```

430                                     {
431     String sqlQuery = "UPDATE Users SET last_run=? WHERE id=?";
432     try {
433         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
434         stmt.setInt(1, lastRID);
435         stmt.setInt(2, id);
436         if (stmt.executeUpdate() != 1) {
437             System.err.println("User's last run was not updated correctly.");
438         }
439     }
440     catch (final SQLException e) {
441         System.err.println(e.getMessage());
442     }
443 }

```

7.3.3.19 userExists()

```
boolean com.activitytracker.DBManager.userExists (
    final String emailAddress )
```

The `DBManager::userExists()` method is designed to facilitate the user experience (UX) design choice of users creating one account to the app and logging in with an existing account for future use. This maintains saved (persistent) data and helps enforce the unique constraint placed on the `email_address` field in the database (again, as users are authenticating using their email address as a user name to identify themselves).

Parameters

<i>emailAddress</i>	The user's email address for which we are checking existence. We use email address here because this is what the user uses to log in to the app.
---------------------	--

Returns

True if the user exists in the database, false otherwise.

Definition at line 128 of file DBManager.java.

```

128                                     {
129     String sqlQuery = "SELECT COUNT(*) AS count FROM Users WHERE 'email_address'=?";
130     boolean exists = false;
131
132     try {
133         PreparedStatement stmt = m_conn.prepareStatement(sqlQuery);
134         stmt.setString(1, emailAddress);
135         ResultSet res = stmt.executeQuery();
136         exists = res.getInt("count") > 0;
137     }

```

```
138         stmt.close();
139     }
140     catch (final SQLException e) {
141         System.err.println(e.getMessage());
142     }
143
144     return exists;
145 }
```

7.3.4 Member Data Documentation

7.3.4.1 `m_conn`

Connection `com.activitytracker.DBManager.m_conn = null` [private]

The `m_conn` variable in the [DBManager](#) class is initially assigned the value of *null*.

When [DBManager::init\(\)](#) is invoked, it is made to be the connection to the database and is subsequently used each time a new SQL statement is created.

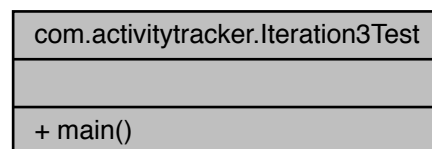
Definition at line 36 of file `DBManager.java`.

The documentation for this class was generated from the following file:

- [app/src/com/activitytracker/DBManager.java](#)

7.4 `com.activitytracker.Iteration3Test` Class Reference

Collaboration diagram for `com.activitytracker.Iteration3Test`:



Static Public Member Functions

- static void [main](#) (String[] args)

7.4.1 Detailed Description

Definition at line 12 of file Iteration3Test.java.

7.4.2 Member Function Documentation

7.4.2.1 main()

```
static void com.activitytracker.Iteration3Test.main (  
    String [] args ) [static]
```

Definition at line 14 of file Iteration3Test.java.

```
14                                     {  
15  
16         // Iteration 1 begins here  
17  
18         User john = null;  
19  
20         DBManager dbManager = new DBManager();  
21         if (!dbManager.init("data.db")) {  
22             System.err.println("Failed to initialize DBManager");  
23             System.exit(1);  
24         }  
25  
26         System.out.println("Attempting to create user...");  
27  
28         if (!dbManager.userExists("jdoe@mac.com"))  
29             User.createUser(  
30                 dbManager,  
31                 "John Doe",  
32                 "jdoe@mac.com",  
33                 1997,  
34                 12,  
35                 12,  
36                 User.Sex.MALE,  
37                 1.6764f,  
38                 54.4310844f,  
39                 "My Very Secure Password"  
40             );  
41         else  
42             System.out.println("User already exists.");  
43  
44  
45     }
```

```
46     if (dbManager.userExists("jdoe@mac.com"))
47         System.out.println("John Doe was created!");
48     else
49         System.out.println("User was NOT created.");
50
51
52     System.out.println("Testing incorrect password...");
53
54     try {
55         john = new User(dbManager,"jdoe@mac.com", "Some Incorrect Password");
56     }
57     catch (final AuthenticationException e) {
58         System.out.println("Incorrect password used; authentication failed.");
59     }
60
61     System.out.println("Authenticating user...");
62
63     try {
64         john = new User(dbManager,"jdoe@mac.com", "My Very Secure Password");
65     }
66     catch (final AuthenticationException e) {
67         System.out.println("Test failed; user could not be authenticated.");
68     }
69
70     // Iteration 1 ended here
71
72     // Iteration 2 begins here
73
74     if (john != null) {
75         Date today = new Date();
76         try {
77             Run.bulkImport(dbManager, john, "/Users/jacobhouse/Google Drive File Stream/My
Drive/Documents/Courses/Computer Science/COMP-2005 Software Engineering/Final
Project/comp2005-activity-tracker/app/InputW0.csv");
78         }
79         catch (final IOException e) {
80             System.err.println(e.getMessage());
81         }
82     }
83     else {
84         System.out.println("John is null. Cannot execute phase 2.");
85     }
86
87     // Iteration 2 ends here
88     // Iteration 3 begins here
89
90     Date date = null;
91     DateFormat sourceFormat = new SimpleDateFormat("dd-MM-yyyy");
92
93     try {
94         date = sourceFormat.parse("01-01-2018");
95     }
96     catch (final ParseException e) {
97         System.err.println(e.getMessage());
98     }
99
100     Vector<Run> runs = Run.getRuns(dbManager, john, date, new Date());
101
102     if (runs == null) {
103         System.out.println("Runs is null.");
104     } else if (runs.size() == 0)
105         System.err.println("No runs in vector.");
106     else
107         for (Run run : runs) {
108             System.out.println("Retrieved run with ID " + Integer.toString(run.getID()));
109         }
110
111
```

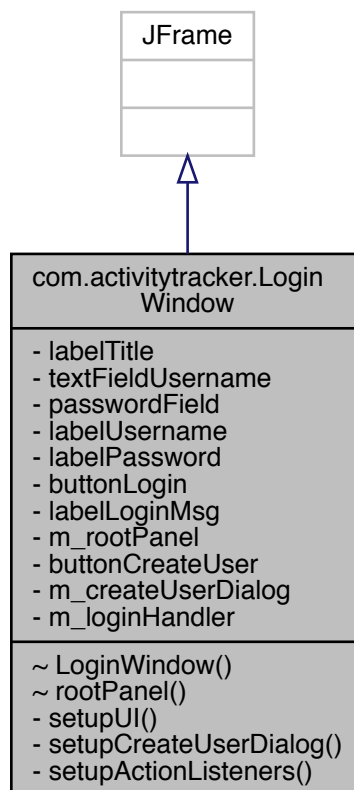
```
112         // Iteration 3 ends here
113
114     }
```

The documentation for this class was generated from the following file:

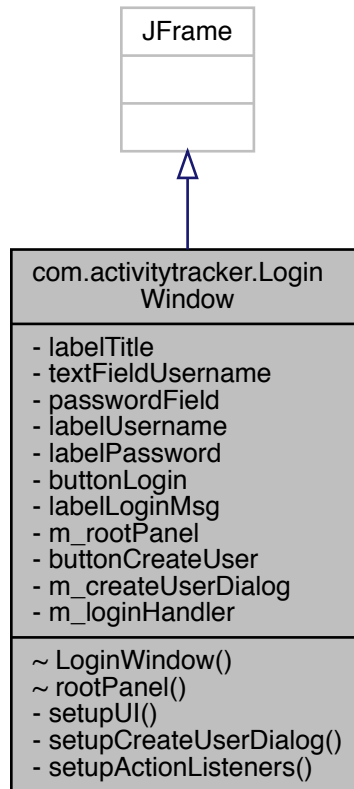
- [app/src/com/activitytracker/Iteration3Test.java](#)

7.5 com.activitytracker.LoginWindow Class Reference

Inheritance diagram for com.activitytracker.LoginWindow:



Collaboration diagram for com.activitytracker.LoginWindow:



Package Functions

- [LoginWindow](#) (java.util.function.Consumer< Void > loginHandler)
- JPanel [rootPanel](#) ()

Private Member Functions

- void [setupUI](#) ()
- void [setupCreateUserDialog](#) ()
- void [setupActionListeners](#) ()

Private Attributes

- JLabel [labelTitle](#)
- JTextField [textFieldUsername](#)
- JPasswordField [passwordField](#)
- JLabel [labelUsername](#)
- JLabel [labelPassword](#)
- JButton [buttonLogin](#)
- JLabel [labelLoginMsg](#)
- JPanel [m_rootPanel](#)
- JButton [buttonCreateUser](#)
- JDialog [m_createUserDialog](#) = null
- java.util.function.Consumer< Void > [m_loginHandler](#)

7.5.1 Detailed Description

Definition at line 11 of file LoginWindow.java.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 LoginWindow()

```
com.activitytracker.LoginWindow.LoginWindow (  
    java.util.function.Consumer< Void > loginHandler ) [package]
```

Definition at line 26 of file LoginWindow.java.

```
26                                     {  
27     m\_loginHandler = loginHandler;  
28  
29     setupUI\(\);  
30     setupCreateUserDialog\(\);  
31     setupActionListeners\(\);  
32 }
```

7.5.3 Member Function Documentation

7.5.3.1 rootPanel()

JPanel com.activitytracker.LoginWindow.rootPanel () [package]

Definition at line 85 of file LoginWindow.java.

```
85         {
86         return m_rootPanel;
87     }
```

7.5.3.2 setupActionListeners()

void com.activitytracker.LoginWindow.setupActionListeners () [private]

Definition at line 54 of file LoginWindow.java.

```
54         {
55
56         // Login button
57         buttonLogin.addActionListener(new ActionListener() {
58             @Override
59             public void actionPerformed(ActionEvent e) {
60
61                 // Do nothing if login fields are empty
62                 if (textFieldUsername.getText().isEmpty() ||
passwordField.getPassword().length == 0) {
63                     return;
64                 }
65
66                 // Change to verifyLogin()
67                 if (true) {
68                     m_loginHandler.accept(null);
69                     return;
70                 }
71
72                 // Display error message
73             }
74         });
75
76         // Create user button
77         buttonCreateUser.addActionListener(new ActionListener() {
78             @Override
79             public void actionPerformed(ActionEvent e) {
80                 m_createUserDialog.setVisible(true);
81             }
82         });
83     }
```


7.5.3.3 setupCreateUserDialog()

```
void com.activitytracker.LoginWindow.setupCreateUserDialog ( ) [private]
```

Definition at line 41 of file LoginWindow.java.

```
41         {
42
43         // Get desktop resolution of default monitor (in case of multi-monitor setups)
44         final GraphicsDevice gd = GraphicsEnvironment.getLocalGraphicsEnvironment().getDefaultScreenDevice(
45         );
46         m_createUserDialog = new JDialog(this, "Activity Logger | Create User", true);
47         m_createUserDialog.setContentPane(new CreateUserWindow().
48         rootPanel());
49         m_createUserDialog.pack();
50         // Set window size to be 1/2 of screen dimensions
51         m_createUserDialog.setSize(gd.getDisplayMode().getWidth() / 2, gd.getDisplayMode(
52         ).getHeight() / 2);
53         m_createUserDialog.setLocationRelativeTo(this); // Center window
54     }
```

7.5.3.4 setupUI()

```
void com.activitytracker.LoginWindow.setupUI ( ) [private]
```

Definition at line 34 of file LoginWindow.java.

```
34         {
35         MaterialUIMovement.add(buttonLogin, MaterialColors.GRAY_100);
36         MaterialUIMovement.add(buttonCreateUser, MaterialColors.GRAY_100);
37
38         labelLoginMsg.setVisible(false);
39     }
```

7.5.4 Member Data Documentation

7.5.4.1 buttonCreateUser

```
JButton com.activitytracker.LoginWindow.buttonCreateUser [private]
```

Definition at line 20 of file LoginWindow.java.

7.5.4.2 buttonLogin

`JButton com.activitytracker.LoginWindow.buttonLogin [private]`

Definition at line 17 of file LoginWindow.java.

7.5.4.3 labelLoginMsg

`JLabel com.activitytracker.LoginWindow.labelLoginMsg [private]`

Definition at line 18 of file LoginWindow.java.

7.5.4.4 labelPassword

`JLabel com.activitytracker.LoginWindow.labelPassword [private]`

Definition at line 16 of file LoginWindow.java.

7.5.4.5 labelTitle

`JLabel com.activitytracker.LoginWindow.labelTitle [private]`

Definition at line 12 of file LoginWindow.java.

7.5.4.6 labelUsername

`JLabel com.activitytracker.LoginWindow.labelUsername [private]`

Definition at line 15 of file LoginWindow.java.

7.5.4.7 m_createUserDialog

```
JDialog com.activitytracker.LoginWindow.m_createUserDialog = null [private]
```

Definition at line 22 of file LoginWindow.java.

7.5.4.8 m_loginHandler

```
java.util.function.Consumer<Void> com.activitytracker.LoginWindow.m_loginHandler [private]
```

Definition at line 24 of file LoginWindow.java.

7.5.4.9 m_rootPanel

```
JPanel com.activitytracker.LoginWindow.m_rootPanel [private]
```

Definition at line 19 of file LoginWindow.java.

7.5.4.10 passwordField

```
JPasswordField com.activitytracker.LoginWindow.passwordField [private]
```

Definition at line 14 of file LoginWindow.java.

7.5.4.11 textFieldUsername

```
JTextField com.activitytracker.LoginWindow.textFieldUsername [private]
```

Definition at line 13 of file LoginWindow.java.

The documentation for this class was generated from the following file:

- [app/src/com/activitytracker/LoginWindow.java](#)

7.6 com.activitytracker.MainWindow Class Reference

Collaboration diagram for com.activitytracker.MainWindow:

com.activitytracker.Main Window
<ul style="list-style-type: none">- m_rootPanel- topPanel- buttonMyActivity- buttonAddDevice- buttonMyFriends- contentPanel- labelProfileIcon- panelMyActivity- panelAddDevice- panelMyFriends- scrollPaneMyFriends- tableAvailableDevices- tableMyActivity
<ul style="list-style-type: none">~ MainWindow()~ rootPanel()- setupUI()- setupActionListeners()

Package Functions

- [MainWindow](#) ()
- JPanel [rootPanel](#) ()

Private Member Functions

- void [setupUI](#) ()
- void [setupActionListeners](#) ()

Private Attributes

- JPanel [m_rootPanel](#)
- JPanel [topPanel](#)
- JButton [buttonMyActivity](#)
- JButton [buttonAddDevice](#)
- JButton [buttonMyFriends](#)
- JPanel [contentPanel](#)
- JLabel [labelProfileIcon](#)
- JPanel [panelMyActivity](#)
- JPanel [panelAddDevice](#)
- JPanel [panelMyFriends](#)
- JScrollPane [scrollPaneMyFriends](#)
- JTable [tableAvailableDevices](#)
- JTable [tableMyActivity](#)

7.6.1 Detailed Description

Definition at line 12 of file MainWindow.java.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 MainWindow()

com.activitytracker.MainWindow.MainWindow () [package]

Definition at line 27 of file MainWindow.java.

```
27         {  
28             setupUI\(\);  
29             setupActionListeners\(\);  
30         }
```

7.6.3 Member Function Documentation

7.6.3.1 rootPanel()

JPanel com.activitytracker.MainWindow.rootPanel () [package]

Definition at line 84 of file MainWindow.java.

```
84         {
85         return m_rootPanel;
86     }
```

7.6.3.2 setupActionListeners()

void com.activitytracker.MainWindow.setupActionListeners () [private]

Definition at line 54 of file MainWindow.java.

```
54         {
55         // My Activity button
56         buttonMyActivity.addActionListener(new ActionListener() {
57             @Override
58             public void actionPerformed(ActionEvent actionEvent) {
59                 panelMyActivity.setVisible(true);
60                 panelAddDevice.setVisible(false);
61                 panelMyFriends.setVisible(false);
62             }
63         });
64         // Add Device button
65         buttonAddDevice.addActionListener(new ActionListener() {
66             @Override
67             public void actionPerformed(ActionEvent actionEvent) {
68                 panelMyActivity.setVisible(false);
69                 panelAddDevice.setVisible(true);
70                 panelMyFriends.setVisible(false);
71             }
72         });
73         // My Friends button
74         buttonMyFriends.addActionListener(new ActionListener() {
75             @Override
76             public void actionPerformed(ActionEvent actionEvent) {
77                 panelMyActivity.setVisible(false);
78                 panelAddDevice.setVisible(false);
79                 panelMyFriends.setVisible(true);
80             }
81         });
82     }
```

7.6.3.3 setupUI()

void com.activitytracker.MainWindow.setupUI () [private]

Definition at line 32 of file MainWindow.java.

```
32         {
33
34         // Apply Material-defined hover effect to buttons
35         Color coolGrey10 = new Color(99, 102, 106);
36         Color coolGrey11 = new Color(83, 86, 90);
37         MaterialUIMovement.add(buttonMyActivity, coolGrey11);
38         MaterialUIMovement.add(buttonAddDevice, coolGrey11);
39         MaterialUIMovement.add(buttonMyFriends, coolGrey11);
40
41         // Load and scale logo into UI
42         String logoPath = "./assets/logo.png";
43         ImageIcon imageIcon = new ImageIcon(getClass().getResource(logoPath));
44         final Image image = imageIcon.getImage(); // transform it
45         final Image newimg = image.getScaledInstance(50, 50, java.awt.Image.SCALE_SMOOTH);
46         imageIcon = new ImageIcon(newimg); // transform it back
47         labelProfileIcon.setIcon(imageIcon);
48
49         panelMyActivity.setVisible(true);
50         panelAddDevice.setVisible(false);
51         panelMyFriends.setVisible(false);
52     }
```

7.6.4 Member Data Documentation

7.6.4.1 buttonAddDevice

JBUTTON com.activitytracker.MainWindow.buttonAddDevice [private]

Definition at line 16 of file MainWindow.java.

7.6.4.2 buttonMyActivity

JBUTTON com.activitytracker.MainWindow.buttonMyActivity [private]

Definition at line 15 of file MainWindow.java.

7.6.4.3 buttonMyFriends

`JButton com.activitytracker.MainWindow.buttonMyFriends [private]`

Definition at line 17 of file `MainWindow.java`.

7.6.4.4 contentPanel

`JPanel com.activitytracker.MainWindow.contentPanel [private]`

Definition at line 18 of file `MainWindow.java`.

7.6.4.5 labelProfileIcon

`JLabel com.activitytracker.MainWindow.labelProfileIcon [private]`

Definition at line 19 of file `MainWindow.java`.

7.6.4.6 m_rootPanel

`JPanel com.activitytracker.MainWindow.m_rootPanel [private]`

Definition at line 13 of file `MainWindow.java`.

7.6.4.7 panelAddDevice

`JPanel com.activitytracker.MainWindow.panelAddDevice [private]`

Definition at line 21 of file `MainWindow.java`.

7.6.4.8 panelMyActivity

JPanel com.activitytracker.MainWindow.panelMyActivity [private]

Definition at line 20 of file MainWindow.java.

7.6.4.9 panelMyFriends

JPanel com.activitytracker.MainWindow.panelMyFriends [private]

Definition at line 22 of file MainWindow.java.

7.6.4.10 scrollPaneMyFriends

JScrollPane com.activitytracker.MainWindow.scrollPaneMyFriends [private]

Definition at line 23 of file MainWindow.java.

7.6.4.11 tableAvailableDevices

JTable com.activitytracker.MainWindow.tableAvailableDevices [private]

Definition at line 24 of file MainWindow.java.

7.6.4.12 tableMyActivity

JTable com.activitytracker.MainWindow.tableMyActivity [private]

Definition at line 25 of file MainWindow.java.

7.6.4.13 topPanel

JPanel com.activitytracker.MainWindow.topPanel [private]

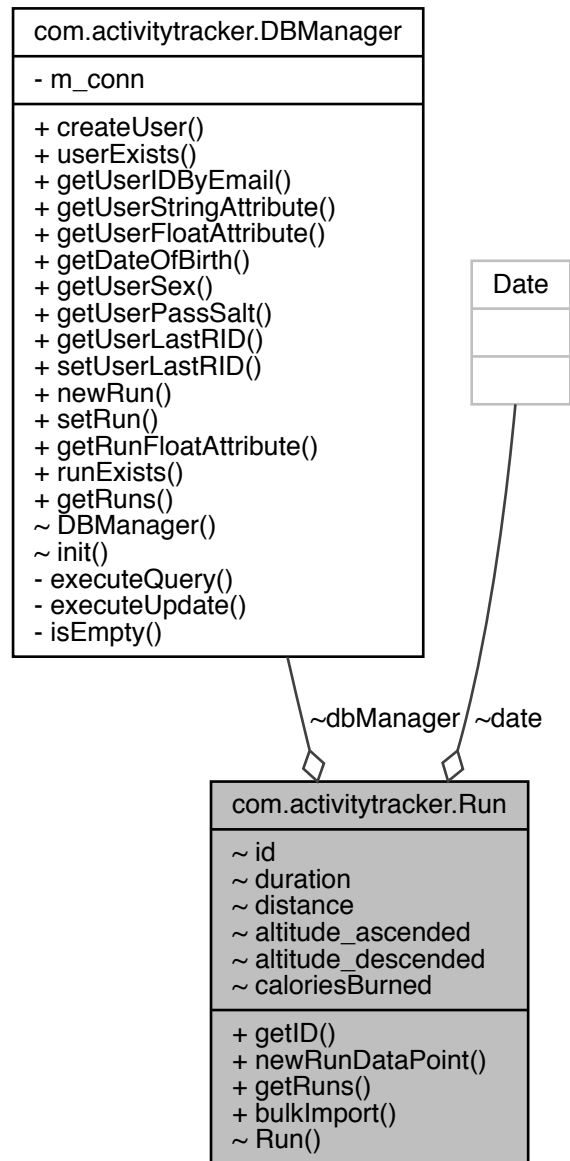
Definition at line 14 of file MainWindow.java.

The documentation for this class was generated from the following file:

- [app/src/com/activitytracker/MainWindow.java](#)

7.7 com.activitytracker.Run Class Reference

Collaboration diagram for com.activitytracker.Run:



Public Member Functions

- `int getID\(\)`

Static Public Member Functions

- static void [newRunDataPoint](#) (final [DBManager](#) dbManager, final [User](#) user, final float [duration](#), final Date [date](#), final float [distance](#), final float altitude)
- static Vector< [Run](#) > [getRuns](#) (final [DBManager](#) dbManager, final [User](#) user, final Date startDate, final Date endDate)
- static void [bulkImport](#) (final [DBManager](#) dbManager, final [User](#) user, final String filePath) throws FileNotFoundException, IOException

Package Functions

- [Run](#) (final [DBManager](#) dbManager, final int rID)

Package Attributes

- int [id](#)
- Date [date](#)
- [DBManager](#) dbManager
- float [duration](#)
- float [distance](#)
- float [altitude_ascended](#)
- float [altitude_descended](#)
- long [caloriesBurned](#) = 0

7.7.1 Detailed Description

Used to logically instantiate a run.

Definition at line 17 of file Run.java.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Run()

```
com.activitytracker.Run.Run (
    final DBManager dbManager,
    final int rID ) [package]
```

The [Run\(\)](#) constructor is used to retrieve workout information from the database and instantiate each row of the Runs table in a logical format.

Parameters

<i>dbManager</i>	The connection to the database.
<i>rID</i>	The run ID used to retrieve information from the database.

Definition at line 60 of file Run.java.

```

60                                     {
61         this.id = rID;
62         this.dbManager = dbManager;
63         this.duration = this.dbManager.getRunFloatAttribute(
RunAttribute.DURATION, rID);
64         this.distance = this.dbManager.getRunFloatAttribute(
RunAttribute.DISTANCE, rID);
65         this.altitude_ascended = this.dbManager.
getRunFloatAttribute(RunAttribute.ALTITUDE_ASCENDED, rID);
66         this.altitude_descended = this.dbManager.
getRunFloatAttribute(RunAttribute.ALTITUDE_DESCENDED, rID);
67     }
```

7.7.3 Member Function Documentation

7.7.3.1 bulkImport()

```

static void com.activitytracker.Run.bulkImport (
    final DBManager dbManager,
    final User user,
    final String filePath ) throws FileNotFoundException, IOException [static]
```

Opens and iterates through a file. The [Run::newRunDataPoint\(\)](#) method is called for each line.

Parameters

<i>dbManager</i>	Database connection with with the method interacts.
<i>user</i>	A User object corresponding to the use whose run(s) is/are being retrieved from the database.
<i>filePath</i>	The file to be iterated through

Exceptions

<i>FileNotFoundException</i>	Thrown if the file path given does not exist.
<i>IOException</i>	Thrown if there is an error reading or opening the file.

Definition at line 177 of file Run.java.

```

178                                     {
179     BufferedReader br = new BufferedReader(new FileReader(filePath));
180     String line = null;
181     Date date = null;
182     while ((line = br.readLine()) != null)
183     {
184         String[] attributes = line.split(",");
185         String buffTime = attributes[0];
186         String buffDistance = attributes[1];
187         String buffAltitude = attributes[2];
188         String buffDate = attributes[3];
189         DateFormat sourceFormat = new SimpleDateFormat("dd-MM-yyyy");
190         try {
191             date = sourceFormat.parse(buffDate);
192         }
193         catch (final ParseException e) {
194             System.err.println(e.getMessage());
195         }
196
197         // Convert strings to floats
198         float fDur = Float.parseFloat(buffTime);
199         float fDist = Float.parseFloat(buffDistance);
200         float fAlt = Float.parseFloat(buffAltitude);
201
202         newRunDataPoint(dbManager, user, fDur, date, fDist, fAlt);
203     }
204 }
```

7.7.3.2 getID()

```
int com.activitytracker.Run.getID ( )
```

Definition at line 207 of file Run.java.

```

207     {
208     return this.id;
209 }
```

7.7.3.3 getRuns()

```
static Vector<Run> com.activitytracker.Run.getRuns (
    final DBManager dbManager,
    final User user,
    final Date startDate,
    final Date endDate ) [static]
```

Retrieves a set of runs from the database. Returns the result as a vector of [Run](#) objects.

Parameters

<i>dbManager</i>	Database connection with which the method interacts.
<i>user</i>	A User object corresponding to the user whose run(s) is/are being retrieved from the database.
<i>startDate</i>	The beginning of the interval for which we are retrieving workouts.
<i>endDate</i>	The end of the interval for which we are retrieving workouts.

Returns

A vector containing instances of [Run](#) corresponding to all entered workouts between the start and end dates specified.

Definition at line 146 of file Run.java.

```

147                                     {
148     Vector<Run> runs = new Vector<>();
149     int rID;
150     Vector<Integer> rIDs = dbManager.getRuns(user.getID(), startDate, endDate);
151
152     if (rIDs != null) {
153         Iterator<Integer> runIDIter = rIDs.iterator();
154         while (runIDIter.hasNext()) {
155             rID = runIDIter.next();
156             runs.add(new Run(dbManager, rID));
157         }
158         return runs;
159     }
160     else {
161         System.err.println("DBManager.getRuns() returned null.");
162         return null;
163     }
164 }

```

7.7.3.4 newRunDataPoint()

```

static void com.activitytracker.Run.newRunDataPoint (
    final DBManager dbManager,
    final User user,
    final float duration,
    final Date date,
    final float distance,
    final float altitude ) [static]

```

Adds a new workout to the database or updates an existing workout with new information that the user imported from the log file.

If *(duration, distance, altitude)* passed to this method is (0, 0, 0) then the intended assumption is that this is the beginning of a new workout. As such, this input will cause a new row to be added to the Runs table in the database and the user's last run ID attribute will be updated accordingly. If the input is non-(0, 0, 0), then three things take place:

1. The *duration* in the database is overwritten by the *duration* provided as input;
2. The *distance* in the database is overwritten by the *distance* provided as input; and
3. Existing values for *altitude_ascended* and *altitude_descended* are retrieved from the database, their difference is compared to the current relative altitude, and depending whether this difference is positive or negative, the appropriate field in the database is updated to reflect the change.

Parameters

<i>dbManager</i>	Database connection with with the method interacts.
<i>user</i>	A User object corresponding to the use whose run is being added to the database.
<i>duration</i>	The length of time in seconds that the user's run lasted.
<i>date</i>	The date the run occurred.
<i>distance</i>	The cumulative distance (in metres) that the user ran as of the current time passed to the method.
<i>altitude</i>	The relative current altitude (in metres) of the user at the time point being entered. Used to compute cumulative altitude ascended and descended throughout the run.

Definition at line 93 of file Run.java.

```

94                                     {
95         int userID = user.getID();
96         int rID;
97         float altitude_ascended;
98         float altitude_descended;
99
100        if (duration == 0f && distance == 0f && altitude == 0f) {
101            altitude_ascended = 0f;
102            altitude_descended = 0f;
103            rID = dbManager.newRun(
104                userID,
105                date,
106                duration,
107                distance,
108                altitude_ascended,
109                altitude_descended
110            );
111            user.setLastRID(rID);
112            System.err.println("Run " + Integer.toString(rID) + " added to database.");
113        } else {
114            rID = user.getLastRID();
115            if (dbManager.runExists(rID)) {
116                altitude_ascended = dbManager.
getRunFloatAttribute(RunAttribute.ALTITUDE_ASCENDED, rID);

```



```
117         altitude_descended = dbManager.  
getRunFloatAttribute(RunAttribute.ALTITUDE_DESCENDED, rID);  
118  
119         if (altitude < 0)  
120             altitude_descended += -1*altitude;  
121         else  
122             altitude_ascended += altitude;  
123  
124         dbManager.setRun(rID, duration, distance,  
altitude_ascended, altitude_descended);  
125         System.err.println("Run " + Integer.toString(rID) + " exists in the database; updating...")  
;  
126     } else {  
127         System.err.println("Run table and User table are inconsistent. No changes made.");  
128     }  
129 }  
130 }
```

7.7.4 Member Data Documentation

7.7.4.1 altitude_ascended

float com.activitytracker.Run.altitude_ascended [package]

The altitude (in metres) that the user climbed throughout the run.

Definition at line 41 of file Run.java.

7.7.4.2 altitude_descended

float com.activitytracker.Run.altitude_descended [package]

The altitude (in metres) that the user descended throughout their run.

Definition at line 45 of file Run.java.

7.7.4.3 caloriesBurned

`long com.activitytracker.Run.caloriesBurned = 0 [package]`

The number of calories that the user burned throughout their run.

Currently this is not being used; it is for future features.

Definition at line 51 of file Run.java.

7.7.4.4 date

`Date com.activitytracker.Run.date [package]`

The date the run occurred.

Definition at line 25 of file Run.java.

7.7.4.5 dbManager

`DBManager com.activitytracker.Run.dbManager [package]`

The run's connection to the database. This is used to add data points and retrieve workout metadata.

Definition at line 29 of file Run.java.

7.7.4.6 distance

`float com.activitytracker.Run.distance [package]`

The distance (in metres) that the user ran.

Definition at line 37 of file Run.java.

7.7.4.7 duration

```
float com.activitytracker.Run.duration [package]
```

The length of the run in seconds.

Definition at line 33 of file Run.java.

7.7.4.8 id

```
int com.activitytracker.Run.id [package]
```

The run's unique ID.

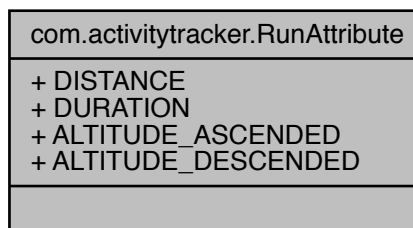
Definition at line 21 of file Run.java.

The documentation for this class was generated from the following file:

- [app/src/com/activitytracker/Run.java](#)

7.8 com.activitytracker.RunAttribute Enum Reference

Collaboration diagram for com.activitytracker.RunAttribute:



Public Attributes

- [DISTANCE](#)
- [DURATION](#)
- [ALTITUDE_ASCENDED](#)
- [ALTITUDE_DESCENDED](#)

7.8.1 Detailed Description

This enumeration type is used to specify the behaviour of generalized methods, particularly in the [DBManager](#) class.

Definition at line 6 of file RunAttribute.java.

7.8.2 Member Data Documentation

7.8.2.1 ALTITUDE_ASCENDED

`com.activitytracker.RunAttribute.ALTITUDE_ASCENDED`

The cumulative altitude (in metres) that the user has climbed throughout their run.

Used in [DBManager::getRunFloatAttribute](#) to specify that ascended altitude should be returned.

Definition at line 24 of file RunAttribute.java.

7.8.2.2 ALTITUDE_DESCENDED

`com.activitytracker.RunAttribute.ALTITUDE_DESCENDED`

The cumulative altitude (in metres) that the user has descended throughout their run.

Used in [DBManager::getRunFloatAttribute](#) to specify that descended altitude should be returned.

Definition at line 30 of file RunAttribute.java.

7.8.2.3 DISTANCE

com.activitytracker.RunAttribute.DISTANCE

The cumulative distance the user has run (in metres).

Used in [DBManager::getRunFloatAttribute](#) to specify that distance should be returned.

Definition at line 12 of file RunAttribute.java.

7.8.2.4 DURATION

com.activitytracker.RunAttribute.DURATION

The duration of the user's run (in seconds).

Used in [DBManager::getRunFloatAttribute](#) to specify that duration should be returned.

Definition at line 18 of file RunAttribute.java.

The documentation for this enum was generated from the following file:

- [app/src/com/activitytracker/RunAttribute.java](#)

7.9 com.activitytracker.SecureString Class Reference

Collaboration diagram for com.activitytracker.SecureString:

com.activitytracker.SecureString
<ul style="list-style-type: none">- secureString- salt
<ul style="list-style-type: none">+ equalString()+ getSalt()+ toString()~ SecureString()~ SecureString()- generateSecureString()- generateSalt()

Public Member Functions

- boolean [equalString](#) (final String other)
- byte [] [getSalt](#) ()
- String [toString](#) ()

Package Functions

- [SecureString](#) (final String plaintext)
- [SecureString](#) (final String plaintext, final byte[] [salt](#))

Private Member Functions

- String [generateSecureString](#) (final String strToSecure, final byte[] [salt](#))

Static Private Member Functions

- static byte [] [generateSalt](#) () throws NoSuchAlgorithmException

Private Attributes

- String [secureString](#)
- byte [] [salt](#)

7.9.1 Detailed Description

This class is used to securely store sensitive string-like information such as user passwords.

Definition at line 10 of file SecureString.java.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 SecureString() [1/2]

```
com.activitytracker.SecureString.SecureString (  
    final String plaintext ) [package]
```

The [SecureString\(\)](#) constructor takes as an argument a plain text string, encrypts it, and stores the encrypted string in the variable [SecureString::secureString](#).

Salt is generated using [SecureString::generateSalt\(\)](#).

Parameters

<i>plaintext</i>	The string to be encrypted. May contain sensitive information.
------------------	--

Definition at line 29 of file SecureString.java.

```
29                                     {
30
31     try {
32         this.salt = generateSalt();
33     }
34     catch (final NoSuchAlgorithmException e) {
35         System.err.println(e.getMessage());
36     }
37     this.secureString = generateSecureString(plaintext, this.
    salt);
38 }
39 }
```

7.9.2.2 SecureString() [2/2]

```
com.activitytracker.SecureString.SecureString (
    final String plaintext,
    final byte [] salt ) [package]
```

The [SecureString\(\)](#) constructor takes as an argument a plain text string and a previously-generated salt, encrypts the plain text string with the provided salt, and stores the encrypted string in the variable [SecureString::secureString](#).

Parameters

<i>plaintext</i>	The string to be encrypted. May contain sensitive information.
<i>salt</i>	Salt that is used to encrypt <i>plaintext</i> . This parameter is used whenever we wish to encrypt using a previously-generated salt for the purpose of encrypted string comparison.

Definition at line 50 of file SecureString.java.

```
50                                     {
51
52     this.salt = salt;
53     this.secureString = generateSecureString(plaintext,
    salt);
54 }
55 }
```

7.9.3 Member Function Documentation

7.9.3.1 equalString()

```
boolean com.activitytracker.SecureString.equalString (
    final String other )
```

Compares the secure string to the *other* parameter for equality.

This method will likely be used to authenticate a user from a password hash existing in the database.

Parameters

<i>other</i>	A (previously encrypted) string with which we compare SecureString::secureString .
--------------	--

Returns

This method returns True if the hashes of both strings are the same, and False otherwise.

Definition at line 66 of file SecureString.java.

```
66                                     {
67
68         return this.secureString.equals(other);
69
70     }
```

7.9.3.2 generateSalt()

```
static byte [] com.activitytracker.SecureString.generateSalt ( ) throws No←
SuchAlgorithmException [static], [private]
```

This method generates salt for encryption of a plain text string.

Returns

Returns a byte array of length sixteen (16) containing the encryption salt.

Exceptions

<i>NoSuchAlgorithmException</i>	Required as <i>SecureRandom.getInstance()</i> may throw this exception and we would like the invoking method to decide how to handle it rather than catching and dismissing it here.
---------------------------------	--

Definition at line 83 of file SecureString.java.

```
83                                     {
84         SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
85         byte[] salt = new byte[16];
86         sr.nextBytes(salt);
87         return salt;
88     }
```

7.9.3.3 generateSecureString()

```
String com.activitytracker.SecureString.generateSecureString (
    final String strToSecure,
    final byte [] salt ) [private]
```

Encrypt string and return secure version.

Due to the importance of securely storing passwords, a "tried and true" method for encrypting passwords found at [this link](#) has been used.

Parameters

<i>strToSecure</i>	The plain text string we wish to encrypt.
<i>salt</i>	The salt with which we will encrypt <i>strToSecure</i> .

Returns

This private method returns the encrypted string to the [SecureString\(\)](#) constructor.

Definition at line 102 of file SecureString.java.

```
102                                     {
103         String generatedPassword = null;
104         try {
105             MessageDigest md = MessageDigest.getInstance("SHA-512");
```

```
106         md.update(salt);
107         byte[] strBytes = md.digest(strToSecure.getBytes());
108         StringBuilder sb = new StringBuilder();
109         for (int i = 0; i < strBytes.length; i++) {
110             sb.append(Integer.toString((strBytes[i] & 0xff) + 0x100, 16).substring(1));
111         }
112         generatedPassword = sb.toString();
113     }
114     catch (final NoSuchAlgorithmException e) {
115         System.err.println(e.getMessage());
116     }
117     return generatedPassword;
118 }
```

7.9.3.4 getSalt()

byte [] com.activitytracker.SecureString.getSalt ()

Returns

Returns the byte array-type salt used to encrypt the text given to the object's constructor.

Definition at line 123 of file SecureString.java.

```
123         {
124         return this.salt;
125     }
```

7.9.3.5 toString()

String com.activitytracker.SecureString.toString ()

Overridden method to return the object as a Java String.

The encrypted string will be returned, though it should be noted for completeness that this is not a full representation of the object since the salt is crucial in arriving at [SecureString::secureString](#) being returned.

Returns

Returns the encrypted string.

Definition at line 136 of file SecureString.java.

```
136         {
137         return this.secureString;
138     }
```

7.9.4 Member Data Documentation

7.9.4.1 salt

```
byte [] com.activitytracker.SecureString.salt [private]
```

The salt that was used to encrypt the plain text string.

Definition at line 19 of file SecureString.java.

7.9.4.2 secureString

```
String com.activitytracker.SecureString.secureString [private]
```

The encrypted string.

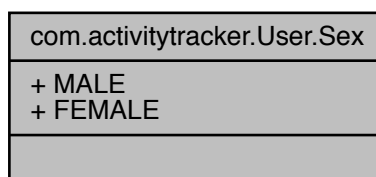
Definition at line 15 of file SecureString.java.

The documentation for this class was generated from the following file:

- [app/src/com/activitytracker/SecureString.java](#)

7.10 com.activitytracker.User.Sex Enum Reference

Collaboration diagram for com.activitytracker.User.Sex:



Public Attributes

- [MALE](#)
- [FEMALE](#)

7.10.1 Detailed Description

Used to represent whether the user is male or female.

Definition at line 12 of file User.java.

7.10.2 Member Data Documentation

7.10.2.1 FEMALE

```
com.activitytracker.User.Sex.FEMALE
```

Used to represent that the user is female.

Recall from the source code included in [DBManager::init\(\)](#) that sex is stored in the database using a data type of BIT(1). If the user is female, we store this in the database by populating this field with a *0*.

Definition at line 26 of file User.java.

7.10.2.2 MALE

```
com.activitytracker.User.Sex.MALE
```

Used to represent that the user is male.

Recall from the source code included in [DBManager::init\(\)](#) that sex is stored in the database using a data type of BIT(1). If the user is female, we store this in the database by populating this field with a *1*.

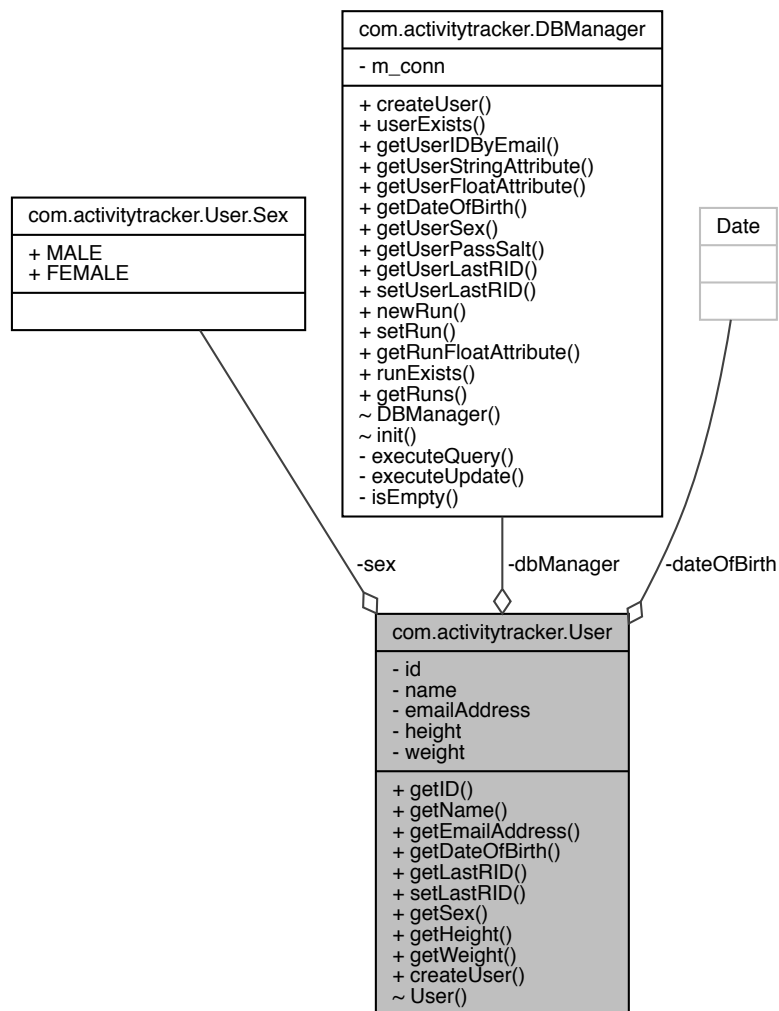
Definition at line 19 of file User.java.

The documentation for this enum was generated from the following file:

- [app/src/com/activitytracker/User.java](#)

7.11 com.activitytracker.User Class Reference

Collaboration diagram for com.activitytracker.User:



Classes

- enum [Sex](#)

Public Member Functions

- int [getID](#) ()
- String [getName](#) ()
- String [getEmailAddress](#) ()
- Date [getDateOfBirth](#) ()
- int [getLastRID](#) ()
- void [setLastRID](#) (final int rID)
- [Sex](#) [getSex](#) ()
- float [getHeight](#) ()
- float [getWeight](#) ()

Static Public Member Functions

- static void [createUser](#) (final [DBManager](#) dbManager, final String [name](#), final String [emailAddress](#), final int DOBYear, final int DOBMonth, final int DOBDay, final User.↵
Sex [sex](#), final float [height](#), final float [weight](#), final String plaintextPassword)

Package Functions

- [User](#) (final [DBManager](#) dbManager, final String [emailAddress](#), final String plaintext↵
Password) throws AuthenticationException

Private Attributes

- int [id](#)
- String [name](#)
- String [emailAddress](#)
- Date [dateOfBirth](#)
- [Sex](#) [sex](#)
- float [height](#)
- float [weight](#)
- [DBManager](#) dbManager = null

7.11.1 Detailed Description

Definition at line 8 of file User.java.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 User()

com.activitytracker.User.User (
 final [DBManager](#) *dbManager*,
 final String *emailAddress*,
 final String *plaintextPassword*) throws [AuthenticationException](#) [package]

Definition at line 38 of file User.java.

```
38
39         {
40             this.dbManager = dbManager;
41             if (this.dbManager.userExists(emailAddress)) {
42
43                 this.id = dbManager.getUserIDByEmail(
44                     emailAddress);
45                 String passHash = this.dbManager.getUserStringAttribute(
46                     UserAttribute.PASSWORD, this.id);
47                 byte[] passSalt = this.dbManager.getUserPassSalt(this.id);
48                 SecureString candidatePassword = new SecureString(plaintextPassword, passSalt);
49                 if (candidatePassword.equalString(passHash)) {
50
51
52
53                     this.name = this.dbManager.getUserStringAttribute(
54                         UserAttribute.NAME, this.id);
55                     // this.emailAddress = this.dbManager.getEmailAddress(this.id);
56                     this.emailAddress = emailAddress;
57                     this.dateOfBirth = this.dbManager.
58                         getDateOfBirth(this.id);
59                     this.sex = this.dbManager.getUserSex(this.id);
60                     this.height = this.dbManager.
61                         getUserFloatAttribute(UserAttribute.HEIGHT, this.id);
62                     this.weight = this.dbManager.
63                         getUserFloatAttribute(UserAttribute.WEIGHT, this.id);
64
65                     System.out.println("Authentication succeeded for " + this.name);
66
67                 }
68                 else {
69
70                     throw new AuthenticationException("Incorrect password.");
71
72                 }
73             }
74             else {
75
76                 throw new NoSuchElementException("No such user exists.");
77             }
78         }
```

7.11.3 Member Function Documentation

7.11.3.1 createUser()

```
static void com.activitytracker.User.createUser (
    final DBManager dbManager,
    final String name,
    final String emailAddress,
    final int DOBYear,
    final int DOBMonth,
    final int DOBDay,
    final User.Sex sex,
    final float height,
    final float weight,
    final String plaintextPassword ) [static]
```

Definition at line 78 of file User.java.

```
80                                                                 {
81
82     SecureString securePassword = new SecureString(plaintextPassword);
83
84
85     dbManager.createUser(
86         name,
87         emailAddress,
88         DOBYear,
89         DOBMonth,
90         DOBDay,
91         sex,
92         height,
93         weight,
94         securePassword
95     );
96
97 }
```

7.11.3.2 getDateOfBirth()

```
Date com.activitytracker.User.getDateOfBirth ( )
```

Definition at line 111 of file User.java.

```
111                                                                 {
112     return this.dateOfBirth;
113 }
```


7.11.3.3 getEmailAddress()

String com.activitytracker.User.getEmailAddress ()

Definition at line 107 of file User.java.

```
107         {  
108             return this.emailAddress;  
109         }
```

7.11.3.4 getHeight()

float com.activitytracker.User.getHeight ()

Definition at line 123 of file User.java.

```
123         {  
124             return this.height;  
125         }
```

7.11.3.5 getID()

int com.activitytracker.User.getID ()

Definition at line 99 of file User.java.

```
99         {  
100             return this.id;  
101         }
```

7.11.3.6 getLastRID()

int com.activitytracker.User.getLastRID ()

Definition at line 115 of file User.java.

```
115 { return this.dbManager.getUserLastRID(this.id); }
```

7.11.3.7 getName()

String com.activitytracker.User.getName ()

Definition at line 103 of file User.java.

```
103         {  
104     return this.name;  
105     }
```

7.11.3.8 getSex()

Sex com.activitytracker.User.getSex ()

Definition at line 119 of file User.java.

```
119         {  
120     return this.sex;  
121     }
```

7.11.3.9 getWeight()

float com.activitytracker.User.getWeight ()

Definition at line 127 of file User.java.

```
127         {  
128     return this.weight;  
129     }
```

7.11.3.10 setLastRID()

```
void com.activitytracker.User.setLastRID (
    final int rID )
```

Definition at line 117 of file User.java.

```
117 { this.dbManager.setUserLastRID(this.id, rID); }
```

7.11.4 Member Data Documentation

7.11.4.1 dateOfBirth

```
Date com.activitytracker.User.dateOfBirth [private]
```

Definition at line 32 of file User.java.

7.11.4.2 dbManager

```
DBManager com.activitytracker.User.dbManager = null [private]
```

Definition at line 36 of file User.java.

7.11.4.3 emailAddress

```
String com.activitytracker.User.emailAddress [private]
```

Definition at line 31 of file User.java.

7.11.4.4 height

`float com.activitytracker.User.height [private]`

Definition at line 34 of file User.java.

7.11.4.5 id

`int com.activitytracker.User.id [private]`

Definition at line 29 of file User.java.

7.11.4.6 name

`String com.activitytracker.User.name [private]`

Definition at line 30 of file User.java.

7.11.4.7 sex

`Sex com.activitytracker.User.sex [private]`

Definition at line 33 of file User.java.

7.11.4.8 weight

`float com.activitytracker.User.weight [private]`

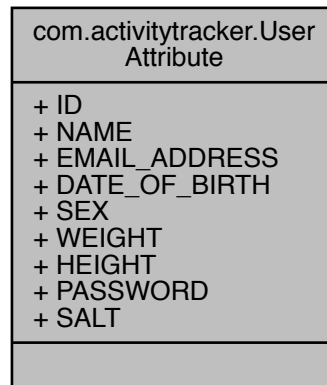
Definition at line 35 of file User.java.

The documentation for this class was generated from the following file:

- `app/src/com/activitytracker/User.java`

7.12 com.activitytracker.UserAttribute Enum Reference

Collaboration diagram for com.activitytracker.UserAttribute:



Public Attributes

- [ID](#)
- [NAME](#)
- [EMAIL_ADDRESS](#)
- [DATE_OF_BIRTH](#)
- [SEX](#)
- [WEIGHT](#)
- [HEIGHT](#)
- [PASSWORD](#)
- [SALT](#)

7.12.1 Detailed Description

This enumeration type is used to specify the behaviour of generalized methods, particularly in the [DBManager](#) class.

Definition at line 6 of file `UserAttribute.java`.

7.12.2 Member Data Documentation

7.12.2.1 DATE_OF_BIRTH

`com.activitytracker.UserAttribute.DATE_OF_BIRTH`

Currently not used as no generalized method retrieves the user's DOB.

Definition at line 26 of file `UserAttribute.java`.

7.12.2.2 EMAIL_ADDRESS

`com.activitytracker.UserAttribute.EMAIL_ADDRESS`

The user's email address.

Used in [DBManager::getUserStringAttribute](#) to specify that the user's email address should be returned.

Definition at line 22 of file `UserAttribute.java`.

7.12.2.3 HEIGHT

`com.activitytracker.UserAttribute.HEIGHT`

The user's height (in metres).

Used in [DBManager::getUserFloatAttribute](#) to specify that the user's email height should be returned.

Definition at line 42 of file `UserAttribute.java`.

7.12.2.4 ID

`com.activitytracker.UserAttribute.ID`

Currently not used as no generalized method retrieves the user's ID.

Definition at line 10 of file `UserAttribute.java`.

7.12.2.5 NAME

`com.activitytracker.UserAttribute.NAME`

The user's full name.

Used in [DBManager::getUserStringAttribute](#) to specify that the user's name should be returned.

Definition at line 16 of file `UserAttribute.java`.

7.12.2.6 PASSWORD

`com.activitytracker.UserAttribute.PASSWORD`

The user's encrypted password hash.

Used in [DBManager::getUserStringAttribute](#) to specify that the user's password hash should be returned.

Definition at line 48 of file `UserAttribute.java`.

7.12.2.7 SALT

`com.activitytracker.UserAttribute.SALT`

Currently not used as no generalized method retrieves the user's password encryption salt.

Definition at line 52 of file `UserAttribute.java`.

7.12.2.8 SEX

`com.activitytracker.UserAttribute.SEX`

Currently not used as no generalized method retrieves the user's sex.

Definition at line 30 of file `UserAttribute.java`.

7.12.2.9 WEIGHT

`com.activitytracker.UserAttribute.WEIGHT`

The user's weight (in kilograms).

Used in [DBManager::getUserFloatAttribute](#) to specify that the user's email weight should be returned.

Definition at line 36 of file `UserAttribute.java`.

The documentation for this enum was generated from the following file:

- [app/src/com/activitytracker/UserAttribute.java](#)

Chapter 8

File Documentation

8.1 app/src/com/activitytracker/ActivityTracker.java File Reference

Classes

- class [com.activitytracker.ActivityTracker](#)

Packages

- package [com.activitytracker](#)

8.2 app/src/com/activitytracker/CreateUserWindow.java File Reference

Classes

- class [com.activitytracker.CreateUserWindow](#)

Packages

- package [com.activitytracker](#)

8.3 app/src/com/activitytracker/DBManager.java File Reference

Classes

- class [com.activitytracker.DBManager](#)

Packages

- package [com.activitytracker](#)

8.4 app/src/com/activitytracker/Iteration3Test.java File Reference

Classes

- class [com.activitytracker.Iteration3Test](#)

Packages

- package [com.activitytracker](#)

8.5 app/src/com/activitytracker/LoginWindow.java File Reference

Classes

- class [com.activitytracker.LoginWindow](#)

Packages

- package [com.activitytracker](#)

8.6 app/src/com/activitytracker/MainWindow.java File Reference

Classes

- class [com.activitytracker.MainWindow](#)

Packages

- package [com.activitytracker](#)

8.7 app/src/com/activitytracker/Run.java File Reference

Classes

- class [com.activitytracker.Run](#)

Packages

- package [com.activitytracker](#)

8.8 app/src/com/activitytracker/RunAttribute.java File Reference

Classes

- enum [com.activitytracker.RunAttribute](#)

Packages

- package [com.activitytracker](#)

8.9 app/src/com/activitytracker/SecureString.java File Reference

Classes

- class [com.activitytracker.SecureString](#)

Packages

- package [com.activitytracker](#)

8.10 app/src/com/activitytracker/User.java File Reference

Classes

- class [com.activitytracker.User](#)
- enum [com.activitytracker.User.Sex](#)

Packages

- package [com.activitytracker](#)

8.11 app/src/com/activitytracker/UserAttribute.java File Reference

Classes

- enum [com.activitytracker.UserAttribute](#)

Packages

- package [com.activitytracker](#)

Index

ALTITUDE_ASCENDED
 com::activitytracker::RunAttribute, 72
ALTITUDE_DESCENDED
 com::activitytracker::RunAttribute, 72
altitude_ascended
 com::activitytracker::Run, 69
altitude_descended
 com::activitytracker::Run, 69
app/src/com/activitytracker/ActivityTracker.java, 93
app/src/com/activitytracker/CreateUserWindow.java, 93
app/src/com/activitytracker/DBManager.java, 94
app/src/com/activitytracker/Iteration3Test.java, 94
app/src/com/activitytracker/LoginWindow.java, 94
app/src/com/activitytracker/MainWindow.java, 94
app/src/com/activitytracker/Run.java, 95
app/src/com/activitytracker/RunAttribute.java, 95
app/src/com/activitytracker/SecureString.java, 95
app/src/com/activitytracker/User.java, 96
app/src/com/activitytracker/UserAttribute.java, 96

bulkImport
 com::activitytracker::Run, 65
buttonAddDevice
 com::activitytracker::MainWindow, 59
buttonCancel
 com::activitytracker::CreateUserWindow, 18
buttonCreateUser
 com::activitytracker::LoginWindow, 53
buttonLogin
 com::activitytracker::LoginWindow, 53
buttonMyActivity
 com::activitytracker::MainWindow, 59
buttonMyFriends
 com::activitytracker::MainWindow, 59
buttonOk
 com::activitytracker::CreateUserWindow, 19

caloriesBurned
 com::activitytracker::Run, 69
com, 11
com.activitytracker, 11
com.activitytracker.ActivityTracker, 13
com.activitytracker.CreateUserWindow, 15
com.activitytracker.DBManager, 21
com.activitytracker.Iteration3Test, 46
com.activitytracker.LoginWindow, 49
com.activitytracker.MainWindow, 56
com.activitytracker.Run, 63
com.activitytracker.RunAttribute, 71
com.activitytracker.SecureString, 73
com.activitytracker.User, 81
com.activitytracker.User.Sex, 79
com.activitytracker.UserAttribute, 89
com::activitytracker::ActivityTracker
 main, 14
com::activitytracker::CreateUserWindow
 buttonCancel, 18
 buttonOk, 19
 CreateUserWindow, 17
 m_rootPanel, 19
 passwordField, 19
 rootPanel, 17
 setupActionListeners, 17
 setupUI, 18
 textFieldEmail, 19

- textFieldHeight, [19](#)
- textFieldName, [20](#)
- textFieldWeight, [20](#)
- com::activitytracker::DBManager
 - createUser, [23](#)
 - DBManager, [23](#)
 - executeQuery, [25](#)
 - executeUpdate, [26](#)
 - getDateOfBirth, [27](#)
 - getRunFloatAttribute, [28](#)
 - getRuns, [29](#)
 - getUserFloatAttribute, [30](#)
 - getUserIDByEmail, [31](#)
 - getUserLastRID, [32](#)
 - getUserPassSalt, [33](#)
 - getUserSex, [35](#)
 - getUserStringAttribute, [36](#)
 - init, [38](#)
 - isEmpty, [39](#)
 - m_conn, [46](#)
 - newRun, [40](#)
 - runExists, [42](#)
 - setRun, [43](#)
 - setUserLastRID, [44](#)
 - userExists, [45](#)
- com::activitytracker::Iteration3Test
 - main, [47](#)
- com::activitytracker::LoginWindow
 - buttonCreateUser, [53](#)
 - buttonLogin, [53](#)
 - labelLoginMsg, [54](#)
 - labelPassword, [54](#)
 - labelTitle, [54](#)
 - labelUsername, [54](#)
 - LoginWindow, [51](#)
 - m_createUserDialog, [54](#)
 - m_loginHandler, [55](#)
 - m_rootPanel, [55](#)
 - passwordField, [55](#)
 - rootPanel, [51](#)
 - setupActionListeners, [52](#)
 - setupCreateUserDialog, [52](#)
 - setupUI, [53](#)
 - textFieldUsername, [55](#)
- com::activitytracker::MainWindow
 - buttonAddDevice, [59](#)
 - buttonMyActivity, [59](#)
 - buttonMyFriends, [59](#)
 - contentPanel, [60](#)
 - labelProfileIcon, [60](#)
 - m_rootPanel, [60](#)
 - MainWindow, [57](#)
 - panelAddDevice, [60](#)
 - panelMyActivity, [60](#)
 - panelMyFriends, [61](#)
 - rootPanel, [57](#)
 - scrollPaneMyFriends, [61](#)
 - setupActionListeners, [58](#)
 - setupUI, [58](#)
 - tableAvailableDevices, [61](#)
 - tableMyActivity, [61](#)
 - topPanel, [61](#)
- com::activitytracker::Run
 - altitude_ascended, [69](#)
 - altitude_descended, [69](#)
 - bulkImport, [65](#)
 - caloriesBurned, [69](#)
 - date, [70](#)
 - dbManager, [70](#)
 - distance, [70](#)
 - duration, [70](#)
 - getID, [66](#)
 - getRuns, [66](#)
 - id, [71](#)
 - newRunDataPoint, [67](#)
 - Run, [64](#)
- com::activitytracker::RunAttribute
 - ALTITUDE_ASCENDED, [72](#)
 - ALTITUDE_DESCENDED, [72](#)
 - DISTANCE, [72](#)
 - DURATION, [73](#)
- com::activitytracker::SecureString
 - equalString, [76](#)
 - generateSalt, [76](#)
 - generateSecureString, [77](#)
 - getSalt, [78](#)
 - salt, [79](#)
 - SecureString, [74](#), [75](#)
 - secureString, [79](#)
 - toString, [78](#)

- com::activitytracker::User
 - createUser, [84](#)
 - dateOfBirth, [87](#)
 - dbManager, [87](#)
 - emailAddress, [87](#)
 - getDateOfBirth, [84](#)
 - getEmailAddress, [84](#)
 - getHeight, [85](#)
 - getID, [85](#)
 - getLastRID, [85](#)
 - getName, [86](#)
 - getSex, [86](#)
 - getWeight, [86](#)
 - height, [87](#)
 - id, [88](#)
 - name, [88](#)
 - setLastRID, [86](#)
 - sex, [88](#)
 - User, [83](#)
 - weight, [88](#)
- com::activitytracker::User::Sex
 - FEMALE, [80](#)
 - MALE, [80](#)
- com::activitytracker::UserAttribute
 - DATE_OF_BIRTH, [90](#)
 - EMAIL_ADDRESS, [90](#)
 - HEIGHT, [90](#)
 - ID, [90](#)
 - NAME, [91](#)
 - PASSWORD, [91](#)
 - SALT, [91](#)
 - SEX, [91](#)
 - WEIGHT, [92](#)
- contentPanel
 - com::activitytracker::MainWindow, [60](#)
- createUser
 - com::activitytracker::DBManager, [23](#)
 - com::activitytracker::User, [84](#)
- CreateUserWindow
 - com::activitytracker::CreateUserWindow, [17](#)
- DATE_OF_BIRTH
 - com::activitytracker::UserAttribute, [90](#)
- DBManager
 - com::activitytracker::DBManager, [23](#)
- DISTANCE
 - com::activitytracker::RunAttribute, [72](#)
- DURATION
 - com::activitytracker::RunAttribute, [73](#)
- date
 - com::activitytracker::Run, [70](#)
- dateOfBirth
 - com::activitytracker::User, [87](#)
- dbManager
 - com::activitytracker::Run, [70](#)
 - com::activitytracker::User, [87](#)
- distance
 - com::activitytracker::Run, [70](#)
- duration
 - com::activitytracker::Run, [70](#)
- EMAIL_ADDRESS
 - com::activitytracker::UserAttribute, [90](#)
- emailAddress
 - com::activitytracker::User, [87](#)
- equalString
 - com::activitytracker::SecureString, [76](#)
- executeQuery
 - com::activitytracker::DBManager, [25](#)
- executeUpdate
 - com::activitytracker::DBManager, [26](#)
- FEMALE
 - com::activitytracker::User::Sex, [80](#)
- generateSalt
 - com::activitytracker::SecureString, [76](#)
- generateSecureString
 - com::activitytracker::SecureString, [77](#)
- getDateOfBirth
 - com::activitytracker::DBManager, [27](#)
 - com::activitytracker::User, [84](#)
- getEmailAddress
 - com::activitytracker::User, [84](#)
- getHeight
 - com::activitytracker::User, [85](#)
- getID
 - com::activitytracker::Run, [66](#)
 - com::activitytracker::User, [85](#)
- getLastRID

- com::activitytracker::User, 85
- getName
 - com::activitytracker::User, 86
- getRunFloatAttribute
 - com::activitytracker::DBManager, 28
- getRuns
 - com::activitytracker::DBManager, 29
 - com::activitytracker::Run, 66
- getSalt
 - com::activitytracker::SecureString, 78
- getSex
 - com::activitytracker::User, 86
- getUserFloatAttribute
 - com::activitytracker::DBManager, 30
- getUserIDByEmail
 - com::activitytracker::DBManager, 31
- getUserLastRID
 - com::activitytracker::DBManager, 32
- getUserPassSalt
 - com::activitytracker::DBManager, 33
- getUserSex
 - com::activitytracker::DBManager, 35
- getUserStringAttribute
 - com::activitytracker::DBManager, 36
- getWeight
 - com::activitytracker::User, 86
- HEIGHT
 - com::activitytracker::UserAttribute, 90
- height
 - com::activitytracker::User, 87
- ID
 - com::activitytracker::UserAttribute, 90
- id
 - com::activitytracker::Run, 71
 - com::activitytracker::User, 88
- init
 - com::activitytracker::DBManager, 38
- isEmpty
 - com::activitytracker::DBManager, 39
- labelLoginMsg
 - com::activitytracker::LoginWindow, 54
- labelPassword
 - com::activitytracker::LoginWindow, 54
- labelProfileIcon
 - com::activitytracker::MainWindow, 60
- labelTitle
 - com::activitytracker::LoginWindow, 54
- labelUsername
 - com::activitytracker::LoginWindow, 54
- LoginWindow
 - com::activitytracker::LoginWindow, 51
- m_conn
 - com::activitytracker::DBManager, 46
- m_createUserDialog
 - com::activitytracker::LoginWindow, 54
- m_loginHandler
 - com::activitytracker::LoginWindow, 55
- m_rootPanel
 - com::activitytracker::CreateUserWindow, 19
 - com::activitytracker::LoginWindow, 55
 - com::activitytracker::MainWindow, 60
- MALE
 - com::activitytracker::User::Sex, 80
- main
 - com::activitytracker::ActivityTracker, 14
 - com::activitytracker::Iteration3Test, 47
- MainWindow
 - com::activitytracker::MainWindow, 57
- NAME
 - com::activitytracker::UserAttribute, 91
- name
 - com::activitytracker::User, 88
- newRun
 - com::activitytracker::DBManager, 40
- newRunDataPoint
 - com::activitytracker::Run, 67
- PASSWORD
 - com::activitytracker::UserAttribute, 91
- panelAddDevice
 - com::activitytracker::MainWindow, 60
- panelMyActivity
 - com::activitytracker::MainWindow, 60
- panelMyFriends
 - com::activitytracker::MainWindow, 61
- passwordField

- com::activitytracker::CreateUserWindow, 19
- com::activitytracker::LoginWindow, 55
- rootPanel
 - com::activitytracker::CreateUserWindow, 17
 - com::activitytracker::LoginWindow, 51
 - com::activitytracker::MainWindow, 57
- Run
 - com::activitytracker::Run, 64
- runExists
 - com::activitytracker::DBManager, 42
- SALT
 - com::activitytracker::UserAttribute, 91
- SEX
 - com::activitytracker::UserAttribute, 91
- salt
 - com::activitytracker::SecureString, 79
- scrollPaneMyFriends
 - com::activitytracker::MainWindow, 61
- SecureString
 - com::activitytracker::SecureString, 74, 75
- secureString
 - com::activitytracker::SecureString, 79
- setLastRID
 - com::activitytracker::User, 86
- setRun
 - com::activitytracker::DBManager, 43
- setUserLastRID
 - com::activitytracker::DBManager, 44
- setupActionListeners
 - com::activitytracker::CreateUserWindow, 17
 - com::activitytracker::LoginWindow, 52
 - com::activitytracker::MainWindow, 58
- setupCreateUserDialog
 - com::activitytracker::LoginWindow, 52
- setupUI
 - com::activitytracker::CreateUserWindow, 18
 - com::activitytracker::LoginWindow, 53
 - com::activitytracker::MainWindow, 58
- sex
 - com::activitytracker::User, 88
- tableAvailableDevices
 - com::activitytracker::MainWindow, 61
- tableMyActivity
 - com::activitytracker::MainWindow, 61
- textFieldEmail
 - com::activitytracker::CreateUserWindow, 19
- textFieldHeight
 - com::activitytracker::CreateUserWindow, 19
- textFieldName
 - com::activitytracker::CreateUserWindow, 20
- textFieldUsername
 - com::activitytracker::LoginWindow, 55
- textFieldWeight
 - com::activitytracker::CreateUserWindow, 20
- toString
 - com::activitytracker::SecureString, 78
- topPanel
 - com::activitytracker::MainWindow, 61
- User
 - com::activitytracker::User, 83
- userExists
 - com::activitytracker::DBManager, 45
- WEIGHT
 - com::activitytracker::UserAttribute, 92
- weight
 - com::activitytracker::User, 88