

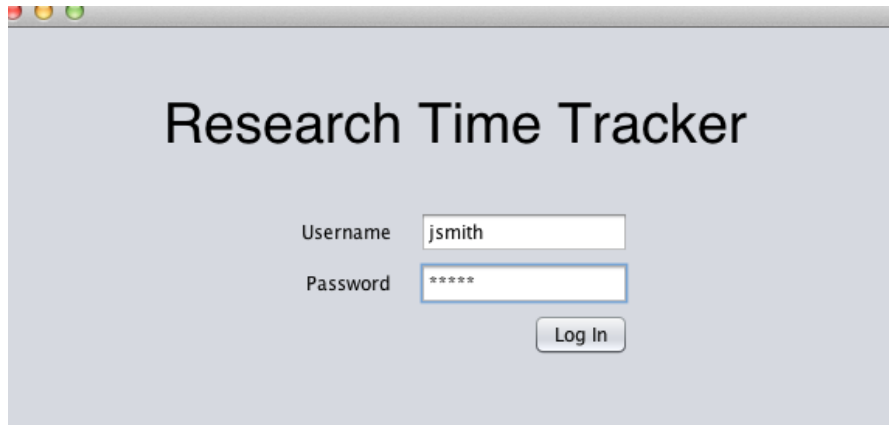
Student Progress Tracker Development

Login Class

All users must be able to login with usernames and passwords.

- Login reads user information from file (See Read class), in order to determine whether user can login.
 - Additionally, if the user can log in, all information is readily available at the start of the program.

Log in differentiates between user types and subsequent interfaces that are triggered by users. This differentiation is based on username/password.



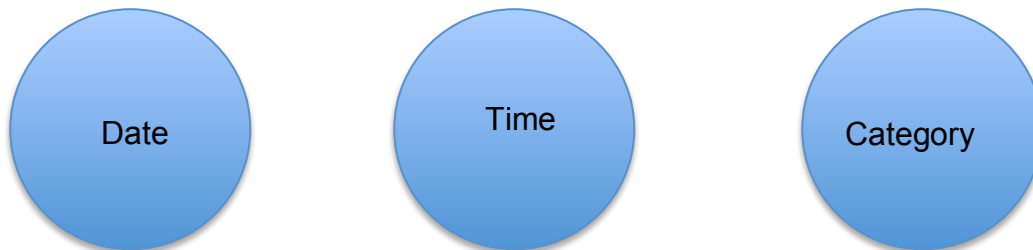
The screenshot shows a window titled "Research Time Tracker" with a light blue background. It features a login form with two input fields: "Username" containing the text "jsmith" and "Password" containing six asterisks "*****". Below the password field is a "Log In" button. The window has a standard macOS-style title bar with red, yellow, and green window control buttons in the top-left corner.

Two types of users: Student and Master. Although these classes share common login methods, these methods are negligible in light of the very different functions of the Student and Master classes. As a result, they were not part of a super/subclass hierarchy.

Student Class

- Can create, edit, and remove time entries. Has calendar type interface for display and management of time entries.

Student **has** three parallel ArrayLists that comprise one time entry at each index.



- Stores date of time entry as a string
- Format 01012014
= January 1, 2014
- Time acquired through activity
- Store as an integer, in minutes
- Category of time acquisition (4 total)
- Stored as an integer (1-4, representing the four methods of time acquisition)

Student **has** a first name, last name, username, password, and grade level, in order to differentiate students from one another.

Student constructors:

- Takes in student info (name, grade, etc.) and time entry ArrayLists
 - For creation of students when reading from file
- Takes in student info
 - For creation of students by “master” user (See Master class)
 - Student **has** boolean “hasLists,” in order to determine whether the user has time entries (whether time entry data needs to be written to file/displayed)

Students have **methods for login/information**:

- set/getUserName
- set/getPassword

Students have **methods for storing data (time entries)**:

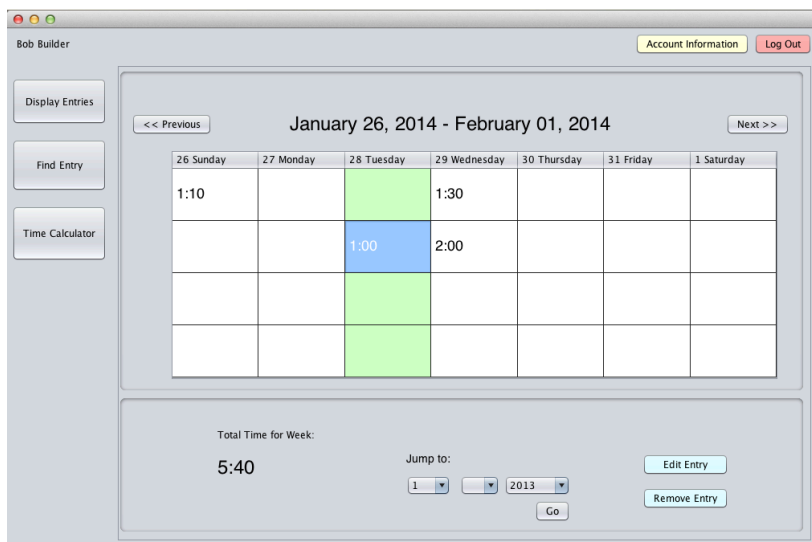
- getDateList, TimeList, CategoryList
- addEntry (takes in a Date object, int time, int category)
 - iterates through ArrayList of String dates (converted to Date—see Display class), if entry of same date/category is not present, adds time entry in chronological order

- Chronology is the reason for conversion to Date object, so the Java Date class' compareTo function can be utilized
- Date objects are stored as Strings in ArrayList<String> for easier write to/read to file function
- getTimeTotal
 - calculates time acquired over from start date to end date (inclusive)
- findEntry
 - finds time entries stored from a start date to an end date (inclusive)

Data is displayed in a weekly calendar type interface, in “**Display Class**”.
(Created with NetBeans) Netbeans was used in order to streamline the process of creating and organizing visual elements in the GUI.

Display has **date tracking/manipulation methods**:

- CalcLowHigh
 - Calculates the “low” date (Sunday=start of the week) and the “high” date (Saturday=end of the week) based on any given day in the week.
 - Display **has** low/high Date variables that are changed by CalcHigh Low
 - Does this using Java Date objects
- getTomorrow—gets the next day
- getYesterday—gets the preceding day
- toDate—String date to Date date
- toFormat—Date date to String date



Display Class interface—weekly calendar

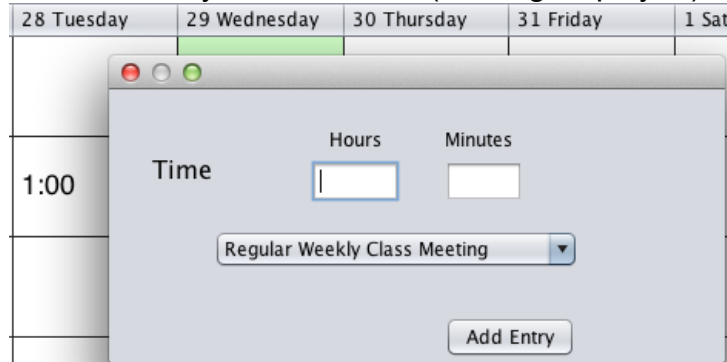
Display **has** a JTable component that comprises the majority of its interface. Each column represents a day, while the four rows represent the four categories of time acquisition.

Display has **JTable display associated methods:**

- setDateLabels—sets column headings (day of month) based on calculation of low/high dates
- refreshModel—refreshes JTable data model whenever time entries or JTable are/is manipulated

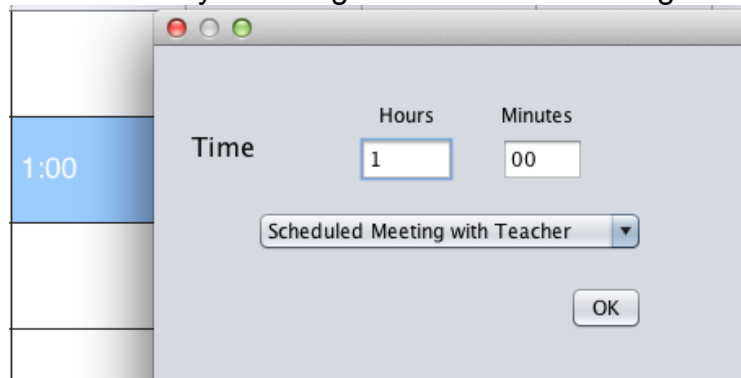
The JTable **default renderer is overridden, mouse event listeners added.**

- On hover, a day (column) is highlighted in green
- On right click, a time entry can be added (JDialog displayed)



Add Entry-JDialog Box—empty input fields

- On left click, entry square is highlighted in blue and “selected”
 - Selected entries can be removed by selecting “remove” button
 - Or edited by selecting “edit” button-->JDialog with entry info



Edit Entry- JDialog Box—fields have original entry data, user must manipulate

These renderer changes (highlighting on selection/hover) were made for a better user interface experience. The user can easily add/change/remove/view data from one screen without confusion.

Additionally, information can be obtained through the **Find Class** and **TimeCalc Class**, which use aforementioned Student methods (findEntry and getTimeTotal, respectively) to achieve their purposes.

Select Start Date:
 1 1 2014

Select End Date:
 1 1 2015

OK

Date	Time	Category
01-01-2014	5:00	3
01-02-2014	3:20	2
01-03-2014	1:40	1
01-22-2014	3:00	1
01-26-2014	1:10	1
01-28-2014	1:00	2

Remove Entry

Find Class interface

Date Input

Select Start Date:
 1 1 2013

Select End Date:
 1 1 2014

OK

Calculated Time

5:00

TimeCalc Class interface

These two classes were made for the purpose for viewing entries/time calculations over time intervals longer than a week. The weekly calendar display is only useful for weekly intervals. A start/end date interface, though clumsier than a visual calendar, is certainly better for data manipulation on longer time intervals.

Master Class

- Serves “administrator” function—for teachers
- Master **has** two static ArrayLists of Student object users and Master object users
 - This is where the Login class reads files to→stores in Master arrays
 - Master can manipulate Students (and other Masters). These changes will(necessarily) be universal, since the ArrayLists are static

Master **has** a first name, last name, username, password, so each master can log in with a personal account. Since ArrayLists of Students/Masters are static, there is really no need for separate accounts, but the function was requested by the end user.

Master **constructors**:

- Takes in master info (name, username, password)
 - For creation of masters when reading from file
- Takes in master name
 - For creation of other masters by master user

Masters have **methods for Master login/information**:

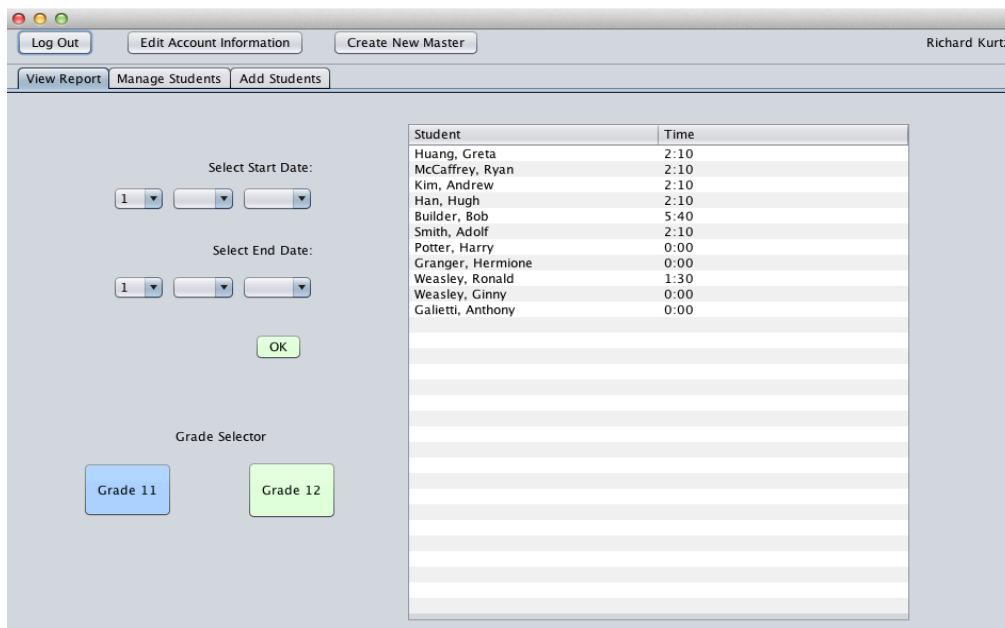
- set/getUserName
- set/getPassword

Masters have **methods for accessing and manipulating Students/Masters**:

- getMasterList/StudentList

- addStudent (one for each Student constructor)—adds new Student to Student ArrayList
- addMaster—(one for each Master constructor) adds new Master to Master ArrayList
- sort methods—overloaded methods for sorting Students in Student List based on name, grade, etc.
- isUser (helper method for Login class)—searches Student list to see if a certain username is present (returns Boolean)
- findUser (helper method for Login class)—searches Student list for a student with a specified username (returns Student)
- isMaster (helper method for Login class)—searches Master list to see if a certain username is present (returns Boolean)
- findMaster(helper method for Login class)—searches Master list for a student with a specified username (returns Master)

The Master Class displays information through the **MasterFrame Class** (the Master Class interface). (Created with NetBeans) It **has** three main panel components on a tabbed pane.



View Report Panel for obtaining reports of total acquired time for each Student

The *View Report* portion (JPanel) of the MasterFrame class **has** a JTable that displays a list of Students and total acquired time entries over a period of time.

- JList defaults to displaying all Students over current week (calculates week using Display Class date tracking methods)
- Using JButton/jComboBox date selectors, one can sort report information

- Allows for multiple inputs of Students at once—useful for beginning of semester, when adding full classes of students to database.
- It **has** a JSlider, which allows the user to select the maximum amount of students to be added.
 - This was created because the computer has to search through the full table for “null” space entries and avoid them, in order to avoid an attempt to add a “null” information Student
 - Creating a large default table model size (of ~100) would slow down the search considerably, thus table model size adjustment was important.

Data for all of these classes are stored in two files—students.txt and masters.txt. Read/write functions are managed by two classes.

These classes are necessary for a data storage program because the program cannot/will not be running at all times. Thus, a place for data storage, such as a text file, is necessary, along with modes of storing/retrieving data.

Write Class

Each class calls to the write class on logout and after editing any data, in order to store information.

Students

Write class first writes user information—

firstname,lastname,grade,username,password,...

Then writes the three ArrayLists of time entries, one index at a time:

date1,time1,category1,date2,time2,category2....

One line per Student user.

- This was done because a line (\n), easily recognized by a human, is no different than a demarcation of “different user” by a string of random symbols used as a sentinel(that is how a computer recognizes a new line command.)
- As a result, the “new line” can be as long of an entry as one would like, but creates clear separation of Student/Master users for humans to review.

Masters

Write class writes user information—firstname,lastname,username,password

One line per Master User

Read Class

Reads in one line at a time (one Student/Master user at a time).

Uses Tokenizer to separate information into tokens based on “,”

Creates a new Student/Master, then adds that Student/Master to their respective ArrayList in the Master Class.


```
Greta,Huang,12,ghuang,515260,01012014,300,3,01022014,200,2,01032014,100,1,01222014,180,1,01262014,70,1,01282014,60,2
Ryan,McCaffrey,11,rmccaffrey,research,01012014,300,3,01022014,200,2,01032014,100,1,01222014,180,1,01262014,70,1,012820
Andrew,Kim,11,akim,research,01012014,300,3,01022014,200,2,01032014,100,1,01222014,180,1,01262014,70,1,01282014,60,2
Hugh,Han,12,hhan,research,01012014,300,3,01022014,200,2,01032014,100,1,01222014,180,1,01262014,70,1,01282014,60,2
Bob,Builder,12,bbuilder,research,01012014,300,3,01022014,200,2,01032014,100,1,01222014,180,1,01262014,70,1,01282014,60
Adolf,Smith,12,asmith,research,01012014,300,3,01022014,200,2,01032014,100,1,01222014,180,1,01262014,70,1,01282014,60,2
Harry,Potter,12,hpotter,research,
Hermione,Granger,12,hgranger,research,
Ronald,Weasley,12,rweasley,research,01272014,90,1
Ginny,Weasley,11,gweasley,research,
Anthony,Galietti,12,agalietti,research,
```

A sample students.txt file

```
1 Richard,Kurtz,rkurtz,research
2 Lorraine,Solomon,lsolomon,research
```

A sample masters.txt file