

Diet Information Manager

Design and implement a simple, easy to use, Diet Information Manager (DIM) application using Java / Swing. DIM should have the following features:

- DIM should maintain an ordered set of daily diet information consisting of Date, Meal, Description (of what was eaten), and Calories (how many calories).
- There should be three views of the diet data should be available: "day", "weekly", and "diet" (total). Use a JTable for the diet information. The user should be able to enter and edit data easily -- consider using table column editors.
- DIM should be private -- that is the application can support many users. Each user has a private set of data and must create and use a password to view and update their data. With your submission there should be the data of two users (provide their passwords) with data for 3 weeks.
- DIM should maintain a persistent storage of user data that can be opened and saved via a file dialog. (This is not a database, or web-services class. Do not use a database for persistence.) Use must use Java serialization for persistence.
- There must be a "Usage" and "Author" menu item. The Usage menu displays information on how to use DIM in a scrollable JEditorPane. The Author menu item displays information about the author (name, email, and picture).
- You can add other information or features to your DIM if you choose.

3 Views

The **day view** should show the time, day of the week (eg: Monday), and the date. The diet information should be shown for each item eaten. A row could have entries for Meal, Description, and Calories. This view is a JTable with 3 columns. For example:

Meal	Description	Calories
brkfst	Bran Flakes	160
brkfst	Banana	100
brkfst	coffee	10
lunch	tuna sand.	350
lunch	yogurt	180
dinner	mac n cheese	640
snack	ice cream	250

The view should also show the total calories eaten for the day. This should be shown as numbers as a graph. Each user of DIM has a daily caloric goal (number of calories to eat each day). The current day's caloric total should be shown in a simple histogram composed of 3 rectangles: green (calories less than goal - 100), yellow (total \leq goal \pm 100), and red (total $>$ goal + 100). The yellow and red rectangles aren't drawn unless the

total calories are in their range. In addition daily diet average (average calories eaten per day for the total diet should be shown in this histogram as a vertical bar drawn through the histogram. For example:



Users should be able to add new items eaten to the “day” view. Users should be able to view and edit any day in the diet as a day view.

The **day of the week view** would then show the total and averages for the days Monday, Tuesday, ..., Sunday. There would be 7 rows for Monday through Sunday. Each row would have the average caloric intake for the day and the total caloric intake for the last instance of this day in the total diet model. This data should also be shown graphically with the average for day in the histogram and the last instance for the day as a vertical line. Note this view does not need to use JTable.

The **diet view** should have a JTable with every row entry for the diet representing all the items eaten that day. The columns should be the date, day of week, meal, description, and calories. Users should be able to add, delete, and edit rows in this view. It should also display the average daily caloric intake for the entire diet.

Managing Views

Views could be managed via a view menu, or, with the use of a JTabbedPane. Views should be scrollable where applicable. I suggest you start with the "diet" view of row items and work out data persistence and using the JFileChooser. Once you have all that stuff working you can then start to implement the other two views of the data.

Accounts

There should be two accounts and users should have passwords using JPasswordField component. These accounts exist only within your program -- they are not computer system accounts! The users / accounts can be determined by the passwords. They can be hard coded into your program. You do not need to be able to create, edit, or delete accounts. Each account could also be the name of a subdirectory where the data for that user is kept. These subdirectories can be created before the program is run -- ie by hand, and be hard-coded into your project. The purpose of the accounts is to have you use JFileChooser to open and save user data. For example a user could start your program, use the File menu to bring up a JFileChooser dialog in the directory of the application. This would show the two subdirectories for the two users -- say "Bob" and "Helen". The user could change to the "Helen" subdirectory and try to open the file "diet.dim". A JPasswordField dialog would be displayed, the user enters the password, "helen" and the diet opens up in its default view (maybe daily – for today). If the user enters "bob" a dialog displays an improper password message and a JPasswordField component for them to try again.

Persistence of data

A user's data should be stored in a subdirectory relative to the application's directory. The data can be stored in one or several files. How this is done depends on how you create the "models" of your application's data. The simplest model is a vector that contains each entry in the diet (date, day of week, meal, description, and calories). This vector could be serialized to a file (helen.dim). There is an example of serialization on the class page. You do not need to serialize the JTable -- only the vector or vectors that is its model. You can serialize the JTable if you wish -- your choice. We have user data persistence so we can use the JFileChooser.

Dates

I recommend you look into the java Date, SimpleDateFormat, and System classes (also possibly Time). You might want to use java Date objects in your model and a SimpleDateFormat in your views. Java's System class has a method currentTimeMillis() that returns the current time in milliseconds. This value can be used in a Date constructor. There is an example DateChooser swing program from a previous class on the class page under Swing examples that you might want to look at. It can be used if appropriate reference / comments are made to the authors.

Implementation

The use of interface builders (IDEs) like Borland's JBuilder or Sun's NetBeans that allow you to "drag and drop" components to make an interface are not the best way to learn how to develop GUIs. These tools facilitate rapid application development for professional developers. They do not by default generate good software design and are not easily reversible. Use them if you wish, but be sure you understand what code they are generating. I recommend building your first GUI application by hand with an editor.

For your initial design consider drawing sketches of the interface for major states and an event-state transition graph. Do not make an overly complex or complete design and then start to implement it. Instead prototype a simple design so you can understand the time and complexity involved in GUI implementation. As you understand the GUI "media" you can design and implement more complex parts of the assignment. Develop a design / implementation plan.

You can work in groups of 2 if you wish. Each group has one submission and grade. If a member is not contributing their "fair-share" to the project the group can divorce before September 30. The group must share all work done up to the divorce. An email must be sent to me and the other member of the group requesting a divorce by 9/29/2005. After 8/30/05 groups can't divorce. This is a "no fault" divorce class.

Submission

Submit the executable application, data for 2 users, and all source files on electronic media (floppy, zip, and cd -- **do not compress any files**) in an 8 by 11.5" folder (I will provide some). If you use netbeans you can submit the netbean jar file as the executable, along with the entire netbean project directory. I still want the source files submitted uncompressed). You do not need to print the source files – I'd rather search them with an editor to find what I want to read. Submit printed versions of:

- UML "lite" class diagram – showing classes, fields, methods (this can be hand drawn, drawn in paint or powerpoint program like the class notes, or with a tool like Visio. There are UML tools installed in the CS dept. labs and many have evaluation downloads.
- a "top level" event state transition table or diagram. This can be a UML state transition diagram.
- a 1 page description of how to run your program
- a printed screen capture of the 3 views
- any other information documentation you actually used to develop the application
- any features you want to point out for my appreciation/evaluation

If you are pointing out parts of your project for me, please also have appropriate comments in your source files so I can search for the relevant code. Be sure the names of all group members are on the envelop and all other submitted material.

I must be able to run your program -- this is your responsibility. I don't want to be compiling (and/or debugging) your programs at the point of evaluation. You must test that your submitted program runs on a system different from the one you developed it on. You cannot assume I have any interface builder installed on the system I will grade your assignment on. I will have the standard jdk1.5.0_04 installation. I should have no problem running any executable built with jdk 1.2.2 or later.