Unit 2 - Diet Manager
Design and Implementation Activity

Introduction

As any scan of recent news articles will attest, people are suffering from an epidemic of overeating (leading to a significant number of overweight people). Your team is going to attack this problem by designing and implementing a prototype software product code-named YADA (Yet Another Diet Assistant). You might consider using your favorite search engine to locate a few of the many diet management systems on the Web; there are many more available as standalone programs for PCs and PDAs.

Detailed Requirements

Programming Language

The system your team develops must use one of the following object oriented languages: Java, C++, Ruby or C#. You are free to propose another language, but your team must be ready to implement the project in the same language. In addition, it must be **approved** by your instructor – instructor may permit or exclude languages at his individual discretion.

As a consequence of this requirement, you may assume that:

- A graphic user interface is not necessary; a command line interface is perfectly acceptable if you find this easier to do. Conversely, having a slick GUI is no guarantee of a good grade – the primary assessment will be based on the quality of the design as a whole.
- Instructor will be less dogmatic about particular style and documentation choices than would be the case in Java; still, teams are expected to adhere to a reasonable & consistent style, including indentation, and to provide basic documentation for all the classes and methods and functions.

Food Database

1. The program must maintain a database of basic foods for consumption (e.g., cheddar cheese, hot dogs, whole milk, etc.). To keep things simple, your prototype need only track the following for each basic food: an identifying string, a list of search keywords used to locate the food, and the calories associated with one serving. It should be obvious from the design how this could be easily expanded to include other nutritional information (protein, carbohydrates, fiber, fat, saturated fat, various minerals, vitamins, etc.).
2. The user must have some way to create composite foods from basic foods (e.g., a peanut butter sandwich from two pieces of bread and a serving of peanut butter, or a peanut butter and jelly sandwich from a peanut butter sandwich and a serving of jelly).
3. The database itself may consist of one file with both basic and composite foods, or two files, one for each category of food. The format of the database file(s) is up to your team with the stipulation that these must be text files that can be viewed (and changed if necessary) in a standard text editor.

4. The database must be loaded when the program is executed and saved at program exit. The user must be able to save the database during a session without terminating the program.
5. The user must be able to add new basic foods via the user interface. For the prototype, new basic foods will be created by entering the identifier, keywords, and calories. However, in the production version users will be able to download food information from a variety of web sites, either those provided by food producers (e.g., restaurants like McDonalds) or by general diet databases. As these sites all have their own idiosyncratic information formats, you must clearly show how your design could be easily extended to handle an additional web site without changes rippling through the system.
6. The user must be able to define new composite foods by selecting one or more existing composite and basic foods and giving the number of servings of each selected food. Each composite, like each basic food, will have an identifying string and list of keywords used in searching. The calories per serving are determined by summing the calories of the component foods.

Daily Logs

1. The program must maintain a log of the foods consumed each day; this log must be read when the program starts, and saved when the program terminates or by a user command. The format of the log is only constrained by the fact that it must be text that can be read and processed by a standard text editor.
2. Users add food to the log by selecting either a composite or basic food and giving the number of servings. Selection can be done by selecting from the whole list of foods, or by narrowing the selection using keywords. The user must have the option of finding foods that match either *all* or *any* of the keywords.

   Note: The same food may be entered several times in the log for a given day.
3. Users must be able to delete foods from the log (changing the number of servings can be done by deleting a food and then reentering it with the new serving count).
4. Users must be able to undo commands to an indefinite depth (that is, there is no predefined limit on how many commands may be undone other than available memory). The undo information is discarded when the program is terminated (that is, it is not carried over from session to session).
5. Users must be able to select, view and update the information for any date in the log, not just the current date.

Diet Goal Profile

1. The program must record the user's gender, height, age, weight and activity level; the latter three must be changeable every day, though the default is to carry over the previous day's information.
2. Based on the information above, the program will compute the target calorie intake for the day. There are several methods for doing this calculation; search the web and incorporate at least two of these in your program. Note that the user must be able to change the method used at any time.
3. At any point, the user can determine the total calories consumed, the target calorie intake, and the difference between these two for whatever date is currently selected. Negative values represent calories available; positive values represent consumption in excess of the target.

Key Design Points

1. New ways of computing target calories must be easy to add without ripple effects throughout the program.
2. New sources of basic food information must be easy to add without ripple effects throughout the program.
3. The log file may grow to be quite large; for this reason, using approaches or patterns that reduce or eliminate duplicate copies of objects is highly desirable (this is a very broad hint).
4. While not all the patterns from Unit 1 and Unit 2 are applicable to this program, many are. Your design will be assessed in large measure on the selection and incorporation of appropriate patterns.

Assessment

Component	**Percent**	**Comments**
Submission	5	Degree to which the project is submitted in accordance with the specification given in the section **Submission Instructions**
Design and Design Document	60	Refer to the **Unit 2 Design Evaluation Rubric**
Implementation	35	Refer to the **Unit 2 Implementation Rubric**. Use proper coding standards.

The Unit 2 Design Evaluation Rubric and Implementation Rubric will be given at an appropriate time.

Submission Instructions

One of the team members will submit a zip file named **Unit2-N.zip**, where **N** is your group number, to the **Unit2 project** drop-box in *Courses portal*. If there are several submissions for a given team, the last submission will be the one that is graded.

The contents of the zip file will include:

1. All the source files in the language you selected that are required to compile, assemble, and run your project.
2. Initial food database(s) and log file sufficient to exercise the program; 12-24 basic food stuffs distributed across several food categories should be sufficient.
3. A *readme.txt* file describing how to run the program and exercise its features.
4. There should be a **single design document** containing all the documentation. Do not submit multiple files containing different types of artifacts. The final design document, in Word format, should include at a minimum:
    a. Title information, including the name of your team, the name of your product, the date, and a list of all the team members.
    b. A short overview section describing the product and the features included.

c. A UML class diagram showing the main classes and interfaces in your design, along with inheritance (generalization), association, aggregation, and composition relationships. Include cardinality and role indicators as you deem appropriate to make the diagram clear. DO NOT include object state information, but include the important methods.

d. A table summarizing the responsibilities of each major class.

e. Other UML diagrams (e.g., sequence charts, collaboration diagrams) that provide insight into the key static and dynamic characteristics of your design.

f. A narrative outlining how the design reflects a balance among competing criteria such as low coupling, high cohesion, separation of concerns, information hiding, the Law of Demeter, extensibility, reusability, etc. This should include a discussion of the design patterns used to achieve this balance in the context of expected product evolution.

g. A short reflection on the two strongest and the two weakest aspects of your design.

5. A copy of the final PowerPoint presentation.