ENSF 409 W22 Final Project

Department of Electrical and Software Engineering
Schulich School of Engineering

The purpose of this final project is to demonstrate your understanding of software development principles using the Java programming language. This assignment is worth 30% of your final grade.

Final Project – Learning Outcomes

- Design and document an object-oriented Java application
- Connect to and import data from a given database
- Process and output data according to user input
- Incorporate GUI interaction with users

At the conclusion of the project, you will have a documented, modelled, and tested application that can be incorporated into an interview portfolio.

Project Logistics

You must work in a team of 3 or 4 members. Team formation guidelines are posted to D2L. You may not switch teams after the project has begun. You must use a **private** GitHub repository to collaborate and record your contributions[1]. Public repositories will be considered a violation of academic integrity. You will need to provide your repository URL and access to the instructors. More information is given in the deliverable descriptions.

Each team is assigned their own D2L dropbox. When submitting, ensure that all names are listed in the description. Only one group member needs to submit, but it is recommended that you complete the submission process together to double-check that nothing is missed! **If the submitting group member uploads the wrong files, it will impact the grade of all members.**

Project Deliverables and Timeline

The project deliverables are designed to give you iterative feedback throughout the development process. Each component contributes to the overall project grade. All deliverables are team-based.

Deliverable	**Due Date**
Early UML Draft (5%)	March 11th at 11:59 PM
Test Design (10%)	March 25th at 11:59 PM
Final Package (85%) • UML Diagram (15%) • Code Implementation (40%) • Documentation (10%) • Unit Tests (25%) • Video Demonstration (10%)	April 12th at 11:59 PM

[1] If students have privacy concerns about using GitHub to host their work, they can contact the course instructors. We can accommodate a self-hosted (by the team) repo as an alternative, but this must be discussed in advance.

Client Problem Definition

Food banks provide support for people who are experiencing food insecurity. Working from supplies which are available, volunteers need to create hampers of nutritionally balanced food to address the needs of clients. Assembling an appropriate hamper for the number of adults and children in a family, for one week, requires knowledge of nutrition and caloric intake needs. To more effectively serve a greater number of people, Example Food Bank needs an application to propose ideal hampers based on current inventory.

Sara, the volunteer manager from Example Food Bank (EFB), explained: "If we calculate a hamper based on one person's needs, we always round up. This means that when we prepare a hamper for a larger family, we can be sure that we are including enough food, and that it is balanced. But including too much food can also be a problem. It means that some food - especially fresh produce and fruits - might go to waste, and we will have less inventory available to help other clients."

She added that the food bank also arranges a regular weekly service for clients with mobility concerns, creating multiple hampers in a single order for delivery.

You are being asked to design an application which will calculate the most efficient combination of inventory which can be used to create a hamper. For example, if a client needs food for three adult males and two children over 8, the application should be able to calculate the combination of items that will meet their nutritional needs for one week, with the least amount of surplus. Furthermore, it should be possible to request multiple hampers of differing configurations as part of a single request. The application should remain active and able to take new requests until a user chooses to exit the program.

Once a request for one or more hampers has been created, the application should generate an order form for the request, ordered by hamper, and remove the items from inventory. An example order form is shown in the design specifications. If it is not possible to fulfill a complete order, the application should not remove any items from inventory and should report on the shortages which caused the order not to be fulfilled. This will allow EFB to use financial resources to procure supplies with nutritional profiles which match the shortfall.

Design Specifications

EFB has provided a sample database that contains information about existing food inventory. The provided file is `inventory.sql`

Different types of clients require different percentages of nutritional minimums and different daily calorie totals. You can assume that the provided client IDs and types (shown with a grey background in the example data) will not change, but all other values should be read from the tables and not hardcoded.

The categories used by EFB are whole grains, fruits & veggies, protein, and other. The second table provides a current list of available food items and their nutritional contribution towards each category. You can assume that the database will be correct (all categories will sum up to 100% and will be rounded to the nearest whole number).

An example of the table data is as follows:

DAILY_CLIENT_NEEDS

ClientID	Client	WholeGrains (%)	FruitsVeggies (%)	Protein (%)	Other (%)	Calories
1	Adult Male	16	28	26	30	2500
2	Adult Female	16	28	26	30	2000
3	Child over 8	21	33	31	15	2200
4	Child under 8	21	33	31	15	1400

AVAILABLE_FOOD

ItemID	Name	GrainContent (%)	FVContent (%)	ProContent (%)	Other (%)	Calories
0023	Tomato Sauce, jar	0	80	10	10	120
1417	Apple, medium	0	100	0	0	52
2349	Granola Bar, box	71	0	6	23	936

Your application must do the following:

1. Use a graphical user interface to take in user input for 1) the number and type of clients in a specific hamper, and 2) several different hampers of varied configurations.
2. Calculate and output the least wasteful combination of food that will fulfill each hamper or specify if an order request cannot be filled.
    - Items cannot be shared between hampers
    - If there are multiple combinations for the most efficient arrangement, select any one combination
    - You are welcome to add food item data to the database for testing purpose or modify the nutritional content
3. If the entire request can be filled, produce a formatted order form in a .txt format. An example order form will be provided with the database files.
4. When an order form is produced, the database should also be updated to specify that the selected items are no longer available in inventory.
5. If a request cannot be fulfilled, the gaps should be identified and output as a message to the user.
6. Additional requests may be submitted until the program is terminated by the user.

The generated order form should include blank spaces where specific name and date information could be filled in later. The order form should include the original hamper requests and the list of food items used to create each hamper. You may choose your own input/output formatting, wording, etc. An example order form is provided in `orderform.txt`

Technical Specifications

- All classes should be public, with access to functionality provided through public methods.
- Each method should have discrete functionality, be free of side effects, and designed to facilitate unit testing, even if it is a private method. Some guidelines:
    - Methods exceeding 24 lines in length should be examined carefully to ensure that they adhere to all the rules.
    - Ensure that the name of the method clearly indicates all expected behavior (to prevent side effects) - if the name is too long, the method is probably not discrete.
    - Consider the different branches that varied input could take through the method. Branches grow exponentially: a single if/else statement has 2 possible branches, but a method with 4 if/else statements has 16 (2*2*2*2). Imagine writing a code for each branch, and if the number is overwhelming, think of how you could break it into smaller pieces: 4 methods, each with one if/else statement, requires only 8 (2+2+2+2) tests to evaluate every branch, compared to the 16 required for one method with 4 if/else statements.
- You may assume that the database data is correct and has already been vetted for invalid inputs, but it cannot be assumed that you will always be able to successfully connect to the database or write to a file, and the code must respond accordingly. However, tests do not need to be written to cover these scenarios.
- Your code should not rely on any packages outside of the `java.*` namespace, with the exception of `junit` and `hamcrest.core`. You cannot expect TAs to install any new software to run your code.
- The program should not exit without the user requesting it (i.e., all errors should be caught before they are thrown by the JVM).
    - Users should be provided with meaningful error messages appropriate to the audience (end user, rather than programmer). Error messages must explain what was wrong and how it can be corrected (e.g., "The number you entered was out of range. Please enter a number between 1 and 10.").
    - Users should be given the opportunity to correct their mistakes, and opportunities for mistakes should be limited. Some techniques for reducing user error are:
        - Providing instructions prior to data entry
        - Normalization of input
        - Limiting choices to only valid options

Deliverable Descriptions

Early UML Draft

Create a legible UML diagram that accurately represents the object-oriented design of your application. You do not need to include your main or GUI interaction components. You may also assume that any database connection/importing will be added later. You will receive feedback on your OOP design - your classes should work together through relationships rather than being a linear sequence of functionality.

When submitting this deliverable, you must include a submission comment with the URL to your main repository.

After receiving feedback, you may modify your UML diagram as necessary before resubmitting as part of the final package.

Test Design

Create a JUnit 4 test suite that can be used to assess the success of your code. You should consider as many boundary cases as possible. You will receive feedback on the thoroughness of your tests and your consideration boundaries, invalid data, exceptions, etc. Your tests will not be run against your code at this point, as your code will still be in development. Instead, it will be compared against your UML diagram and the specifications. If your UML diagram has changed since the first milestone, you can include the updated diagram. Be sure to document your tests well.

After receiving feedback, you may modify your tests as necessary before resubmitting as part of the final package.

Final Package

Your final package will consist of several components to present your solution.

All of the deliverables should be inside a single .zip folder labelled with your ENSF 409 assigned group number. Only one team member needs to upload the solution to D2L. You must include all of your team members' names on each deliverable (i.e. in the code documentation, unit tests, etc.). Please also include all member first and last names in the provided text box when submitting on D2L.

Upon submission, you should also invite `abarcomb-work` and `eamarasco` to contribute to your repository. We will not modify your repository but may use the access to review team contributions.

UML Diagram (15%)

You must submit a UML diagram that accurately represents your final code design. The diagram must be legible and follow proper formatting conventions. You do need to represent exception handling in your diagram.

Code implementation (40%)

All of your solution's .java files should be included. You should use the edu.ucalgary.ensf409 package. Each public class must be placed in its own file.

Your code should be developed using GitHub. Each team member should fork the repository and use pull requests, as this will confirm code contributions in case of team disputes. Suggested workflow:

- Have one person create the project on GitHub. Make sure that the repository is private, and access is granted to team members. People who are not part of the project team or the teaching team should not be able to view the code.
- Each person, including the one who created the project, should fork the repository and work on their fork, sending pull requests to the original project. While we will not enforce this model of working, do note that it is in your interests to follow it, as it ensures that (1) you have a copy of the code for future reference or inclusion in portfolios (in the event of the project owner removing you or deleting the project), and (2) there is a record of your actual contributions in case there is a subsequent dispute about the contributions made by each group member.
- If you pair program, take turns making commits so that there is a record of your contributions.
- Your final code will be submitted via D2L dropbox.
- In the case of any disputes about contributions, we will review your GitHub repository and log. It is to your advantage to ensure that the history accurately reflects your contributions.

Documentation

All of your code should be fully documented and commented. Formatting and naming conventions should follow the standards used in ENSF 409. Your main should include instructions for running and using your code. You may also choose to include a README text file instead.

Unit Tests

You must include unit tests that demonstrate how your solution meets the design specifications. Consider possible boundary cases and exception handling. Your code will be run against your own tests, though values may be changed to ensure that solutions are not hard-coded to specific scenarios.

Video demonstration

Use Zoom or another tool of your choosing to record a short demonstration of your application (5 minutes or less!). All team members should be part of the demonstration. Marks may also be deducted for videos of excessive length. Videos should have clear audio and cameras should be on or you should display a professional-looking photo. This is your opportunity to demonstrate how your application works and why it fulfills the requirements.

Grading Rubric

Each deliverable will be graded using a rubric. You must satisfy all of the criteria in a particular category to earn that category's minimum grade. The grading criteria can be found in the accompanying rubric handout.

Team Assessment

Following the project submission, you will have the opportunity to reflect on the contributions of each team member. You will allocate a percentage of work to each member. For example, in a team of four, if all members contributed equally, you would allocate 25% to each person. If one student took on more responsibilities compared to the others, the allocation might be adjusted to 40%, 20%, 20%, 20%. These allocations may be used to adjust your individual grade on the project in case of uneven contributions. Any discrepancies between allocations will be flagged for the TAs and instructors to investigate further. If you do not submit an evaluation, it will be assumed that you believe the work was evenly distributed.