Lite Byte

Version:
1.0

Date:
2020-04-10

Group:
07

Members:
Andrew Zhu
Jin Sunwoo
Paul Lee

Course:
COMPSCI 2XB3
Computer Science Practice and Experience:
Binding Theory to Practice

Department & University:
Computer Science Department
McMaster University

Revision Page:

Team members:

Andrew Zhu - 400169901
Paul Lee - 400189886
Jin Sunwoo - 400181945

Revision History:

It is written based on git commit history.

Andrew Zhu:

Created project in intelliJ.
Added and completed a dataset CSVs.
Changed project encoding.
Began working on CSV formatter.
Completed working on file getter and setter.
Fixed some CSVs.
Completed CSV formatting function.
Changed to a new dataset.
Worked on skeleton & assigning roles to each member.
Completed getNutrients method in DBReader class.
Fixed bug with project structure.
Completed getNutrient & getNutrients in DBReader class.
Completed working on lexicographical sort.
Fixed workspace.xml.
Added treebuilder skeleton.
Reduced test time.
Clean up code & set up an android studio for the project.
Refactored names to fix naming issues.
Built treebuilder.
Reformatted some code to work with android.
Android Pair breaking getNutrients and logger.
Built XML skeleton.
Updated gradle.
Fixed javafx.util dependency, added a new pair object to compensate for missing dependencies.
Resolved issues with android SDK so that all java based methods are in working order.
Made a change in design.
Removed deprecated code and reorganized activity structure.
Completed overhaul of activity structure.
Added functionality for the log button.
Fixed pop-up window bug.
Tried to fix linter error in android studio and figured out it can be just ignored.

Completed logger functionality.
Worked on a custom adapter & fixed bug.
Killed backstack.
Worked on a logger, ran into fileNotFoundException.
Troubleshooted path bug & fixed manifests.
Saved database files to device memory & organized files.
Completed linking mainActivity to infoActivity.
Cleaned up code & worked on improving speed.
Fixed try-catch/if else bug and two other bugs.
Fixed back buttons & wrong intent.
Worked on reducing CSV accesses.
Centered buttons.
Cleaned up code and separated into more clear classes.
Completed InfoActivity & fixed bug.
Cleaned up code & worked on fixing buttons.
Completed graphView & cleaned up code.
Completed GraphActivity.
Cleaned up code.
Improved activity info to work for all phone sizes. Changed constraints.
Completed writing comments.

Paul Lee:
Added dataset csvs.
Fixed CSV_FORMAT's getColumn function to assign the correct item name, and value.
Fixed CSV_FORMAT getColumn to get names if they aren't in quotations.
Completed DBReader getFoods prototype.
Completed DBReader getFoods method.
Fixed small issue in FOOD NAME.csv formatting.
Completed getfood & getfoods method in DBReader class.
Completed getfoods when inputted category.
Included quotation marks in all food names when sorted lexicographically.
Completed lexicographical sort.
Added Logger.java to log foods and their nutrition info to a file; log.txt.
Added functionality of the buttons except for Back, toGraph, and toinfo buttons.
Worked on the log button to show pop-up.
Added programmatically generated buttons for the categories, and connected those to activity_food_name.
Found error in food_name activity when trying to go back to food_category activity.
Cleaned up FoodCategoryActivity.java.
Added FoodNameActivity once a category is selected.
Removed quotation marks in the names of food and categories when they are displayed.
Added transition from food name activity to popup to logger.

Worked on creating a log.
Completed logging feature.
Fixed small formatting issue in Log.
Tweaked the food name text in the pop up window and log.
Increased text size in Log.
Updated the log format to look more organized.
Removed pop-up and replaced it with toast instead.
Fixed constraints.
Cleaned up code and added measurements to the log for the nutrient information.
Made scroll views size proportional to screen size.
Updated sorting algorithm to lexicographically sort and ignore quotation marks.
Applied sorting to the food category view and name view so that they are all sorted in lexicographical order.
Completed writing comments.

Jin Sunwoo:
Added database csvs.
Completed getCategories & getCategory methods in DBReader class.
Implemented search class with match method.
Worked on UI part - constraint, buttons, text-field, and background.
Made little change in UI parts - background & button colors.
Fixed file path for database reader.
Built a search view in activity_food_name.xml with textView, and connected it with FoodNameActivity and search algorithm.
Modified Search Algorithm and related method “getFoodname” in DBReader.
Created FoodNameSearchActivity & activity_food_name_search layout.
Created a button in activity_main that links from main page to food search by name page.
Made RecyclerViewAdaptor and changed textView used from FoodNameSearchActivity to recyclerview.
Created food_name_item.xml.
Fixed recycler adaptor & match method in search class to adjust for recycler view.
Created FoodNameSearchActivity, InfoSearchActivity, and activity_search_info.xml to link each recyclerview with related nutrient data when clicked by the user.
Linked InfoSearchActivity and LoggerActivity to save nutrition data of the searched food to logger.
Fixed the constraint layout of activity_main.xml.
Created two more different sized buttons in drawable and implemented it in activity_loger.xml.
Changed first getfood method in DBReader class.
Completed writing comments.

By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through CS 2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

Roles and Responsibilities:

Andrew Zhu

Project Executive

Helped make decisions with other project executives on the final expected result.
Worked with developers using the agile method to produce a result that was both attainable and satisfactory.
Worked with the client (professor) to discuss and inquire about project requirements and necessary features.

Project Coordinator

Organized and maintained schedule for project deadlines and milestones.
Assigned tasks in a timely fashion equally among group members.
Made sure tasks that were assigned were properly completed.
Reassigned tasks depending on urgency and rate of completion.

Software Developer

Created models, views/controllers and classes for the application.
Managed and resolved software bugs and issues.
Cleaned up legacy code and unused code fragments.

UI/UX Designer

Wrote up XML files for the user interface that would function as the interface users would be able to interact with.
Worked on creating a user-friendly design in order to provide maximum satisfaction for the user.
Adjusted and modified constraints in order to adapt to mobile devices of all sizes.

Paul Lee

Project Executive

Helped make decisions with other project executives on the final expected result.
Worked with developers using the agile method to produce a result that was both attainable and satisfactory.

Software Developer

Created models, views/controllers and classes for the application.

Managed and resolved software bugs and issues.
Cleaned up legacy code and unused code fragments.

UI/UX Designer

Wrote up XML files for the user interface that would function as the interface users would be able to interact with.
Worked on creating a user-friendly design in order to provide maximum satisfaction for the user.
Adjusted and modified constraints in order to adapt to mobile devices of all sizes.

Jin Sunwoo

Project Executive

Helped make decisions with other project executives on the final expected result.
Worked with developers using the agile method to produce a result that was both attainable and satisfactory.

Software Developer

Created models, views/controllers and classes for the application.
Managed and resolved software bugs and issues.
Cleaned up legacy code and unused code fragments.

UI/UX Designer

Wrote up XML files for the user interface that would function as the interface users would be able to interact with.
Worked on creating a user-friendly design in order to provide maximum satisfaction for the user.
Adjusted and modified constraints in order to adapt to mobile devices of all sizes.

Contributions Page:

Name	Roles	Contributions
Andrew Zhu	Project Exec., Project Coordinator, Software Developer, UI/UX Designer	Overview: ● Group scheduler/coordinator ● Assigned tasks to members ● Helped decide project name ● Code optimization and cleanup ● Constructed initial skeletons for activities ● Set application manifest settings and navigation/action bar settings. Created: - getNutrients() method in DBReader

		- LoggerActivity - GraphActivity - MainActivity - InfoActivity - TreeBuilder - Pair - Associated XML files
Paul Lee	Project Exec., Software Developer, UI/UX Designer	Thought of project name Created: - Both getFoods() methods in DBReader - SortingAlgorithms class and all methods within it - Logger feature - Pop-up activity and functionality (legacy) - FoodCategoryActivity and FoodNameActivity display functionality
Jin Sunwoo	Project Exec., Software Developer, UI/UX Designer	Created: - getCategory & getCategories method in DBReader - Search class with match method - FoodNameSearchActivity - InfoSearchActivity - LoggerSearchActivity (legacy) - RecyclerViewAdaptor - Logger feature that works for food name searching - Round buttons with different sizes

Executive Summary:

Food is a necessity to everyone. The way we get food is from grocery stores. So, in turn, these companies that create the food we need want to know what the demand for certain foods and nutrients are. By having a way to tell what the demand is, companies can prioritize the production of certain foods, which will increase their sales, and leave us, the consumers, happy. Lite Byte provides a way for consumers to track and plan their eating habits so that they can be healthier. This is done by providing information to the consumer about the foods they eat using datasets that contain over five-thousands foods, and over five hundred-thousand nutrition information, so they know what is going into their bodies. Knowing what you're eating makes it much easier to decide what types of foods and nutrients you need to have more of, and less of. Companies can benefit from this by being able to see changes in people's diets and what they're eating, so they can adjust what they are producing to better fit the demand of the consumers.

Table of Contents:

Classes/Modules

MainActivity

This activity class is the “Main Menu” of the app. It is the home screen that the user will view upon launching the app. The class contains the basic functions that Android apps require; onCreate which is initialize upon being displayed, onWindowFocusChanged which calls the hideSystemUI function when the activity is being displayed to the user, and hideSystemUI, which hides the device’s navigation ui, and status bar so the user can see the activity better when it is being displayed. All activity classes contain these three functions.

The onCreate function sets the user’s view to the activity_main.xml view, which is the main menu, or home menu. It then calls the buildTree function.

The buildTree function will create a new TreeBuilder object whenever the app is opened. This TreeBuilder object will generate all of the categories and food names, which makes the app run smoother while it is in use. Due to it getting all of the categories and food names at the start of launch, launching the app will take some time. Any other time the view is changed to the MainActivity class, the treeBuilder variable simply gets the transferred treeBuilder object that is pushed over every time the activity/view changes. Whenever the view is changed from one activity to another, the treeBuilder is pushed to that activity.

The toCategory function initializes the FoodCategoryActivity class, and pushes the treeBuilder to that activity class.

The toLog function initializes the LoggerActivity class, and pushes the treeBuilder to that activity class.

The toFoodSearch function initializes the FoodNameSearchActivity class, and pushes the treeBuilder to that activity class.

DBReader

Used to read the datasets. Contains a function to read each of the datasets. This class was created to maintain the dataset reading aspect of the application so the datasets didn’t have to be reductantly called in the same function.

getFoods reads all the food names in the foodname dataset, or all the food names of a given category by reading both the foodname and foodgroup datasets. The method sets up a scanner for the inputted InputStream, which is supposed to point to the foodname dataset. This cannot be hardcoded into the function itself due to Android Studio reading files differently than normal Java. The scanner skips the first line since the first line in the foodname dataset simply defines what each column defines. It then iterates through each line, and splits them into lists by commas that are outside of quotation marks, since some food names have commas in them, and are thus encased in quotation marks. The function adds the 4th element in the list, which is always the food’s name, removes any quotation marks, capitalizes the first letter, and adds it to the output. It then sorts the output list lexicographically, and outputs that list. The getFoods method that takes foods of a given category works the same way, but it also iterates through the foodgroup dataset to find the ID of the inputted category name. It then uses that ID to get any foods in the foodname dataset that has the same group ID.

getCategories gets all the categories in the foodgroup dataset. Similar to the getFoods function, it creates a scanner of the inputted InputStream that points to the foodgroup dataset. It skips the first line since it is simply used as a label. It then splits the rows by commas that aren't encased with quotation marks, and adds the 2nd element of each list to the output list since the 2nd element is always the category name. It then returns the output list.

getNutrients gets the nutrient information of a given food by reading the nutrientamount and nutrientname datasets. This works similarly to the getFoods function that takes in a category name. A scanner is created for 3 files; the foodname dataset, nutrientname dataset, and the nutrientamount dataset. First, the food ID is taken from the foodname dataset, which is the 0th element in each row when split by commas, and is searched for in the nutrientamount dataset. The food ID is also the 0th element in the nutrientamount dataset. Whenever a row is attached to the food ID, the 2nd and 1st element is taken from the row when split by commas, which is the amount of a nutrient, and ID of the nutrient respectively. Then, the scanner for the nutrientname dataset is initialized, and the ID of the nutrient saved is searched for to find out what nutrient name it belongs to. Once it is found, a Pair object is created with the first element being the nutrient name, and the second element being the amount of that nutrient. That Pair object is added to the output list, and this task is repeated until all of the nutrition info for the inputted food is added to the output list. Then, the output list is returned.

FoodCategoryActivity

This activity class is used for the display of the categories of foods the user can select. This was made to allow the user to select a category of food to make it easier to find a certain food or food type they want to find nutrition information on and/or log. The onCreate, onWindowFocusChanged, and hideSystemUI functions are included.

The onCreate function sets the view to be the activity_food_category.xml view, which displays all of the food categories that the user can choose from. The treebuilder object is called and the list of categories is created from the treebuilder object, and is assigned to the variable, *list*, and is lexicographically sorted. initButtons is then called.

The function toMain is called when the user presses the “Back” button. It pushes the treeBuilder, which is a TreeBuilder object that is transferred throughout all of the activities. It then moves the user to the MainActivity class/display.

The initButtons function is called every time the onCreate function is run. It sets up the buttons on the display to show every single category and, upon being pressed, transfers the user to the FoodNamesActivity class/display that will display the food names of the selected category. In the xml of the food categories activity, one button is already defined, and its text is replaced with the first category in the list. Then, for every other category, the button is duplicated, and the text is set to be another category.

FoodNameActivity

This activity displays the food names of the category that the user selected in FoodCategoryActivity. This was made to allow the user to select a food of a given category so it would be easier to find certain foods. The main three functions; onCreate, onWindowFocusChanged, and hideSystemUI are included.

The onCreate function sets the view to the activity_food_name.xml. It works similar to the FoodCategoryActivitys. However, instead of getting the categories, it gets the foods from the selected category. It also calls initButtons.

The initButtons function works similar to the FoodCategoryActivitys. But instead of displaying categories, it displays the names of foods. Upon clicking on a food, it will get the calories, sugar, sodium, and fat of that food, and carries that info, along with the food name, over to the InfoActivity class. It then changes the view to the InfoActivity view.

FoodNameSearchActivity

This activity class is used for the display of the search bar where the users can search for food names that they want to check on with a full list of all the food name lists under the search bar. Upon searching for the name, the list will be adjusted to only show names that match with the typed words in lexicographical order. The onCreate, onWindowFocusChanged, and hideSystemUI functions are included.

The onCreate function sets the view to the activity_food_name_search.xml and sets searchview with id search_view2 which is in the same xml page. Using DBReader, it reads foodname.csv and gets all the food names in the list. It sets up RecyclerView, RecyclerViewAdaptor, and LinearLayoutManager to use RecyclerView which will be displayed in activity_food_name_search.xml. Using setOnQueryTextListner's onQueryTextChange method to searchView, it first displays a full list of all the food name in RecyclerView using getResult() method which returns a full list, and once the users type some letters, using search() method, it calls match method from Search class to find a list of matched words and displays it.

The function toMain is called when the user presses the "Back" button. It pushes the treeBuilder, which is a TreeBuilder object that is transferred throughout all of the activities. It then moves the user to the MainActivity class/display.

RecyclerViewAdaptor

RecyclerViewAdaptor extends from RecyclerView.Adaptor class and has a clear() method which clear instance variable list and method set(list) which set/reset input list as an instance variable list.

The onCreateViewHolder method sets food_name_item.xml as a view which will be displayed on page as each food name in the list.

The inner class ViewHolder links a button in food_name_item.xml with setOnClickListener method so that upon clicking the button, it can lead to an appropriate nutrient info page which has nutrients values of the chosen button's food name.

InfoActivity

This activity displays the nutrition information of the food that the user selected in FoodNameActivity. This was made to allow the user to view the nutrition info of a desired food, and log it if they desire to. onCreate, onWindowFocusChanged, and hideSystemUI is included in this class.

The onCreate function sets the view to the activity_info xml, and gets the nutrition information that was carried over from when the user selected a food in FoodNameActivity, and displays the food name, calories, sugar, sodium, and fat of that food to the user.

The toCategory function changes the view to the FoodNameActivity view.

The toHome function changes the view to the MainActivity view.

The toLog function removes any quotations in the selected food name and carries it over to the LoggerActivity class, and changes the view to the LoggerActivity view.

InfoSearchActivity

This activity displays the nutrition information of the food that the user selected in FoodNameSearchActivity. This was made to allow the user to view the nutrition info of a desired food, and log it if they desire to. onCreate, onWindowFocusChanged, and hideSystemUI is included in this class.

The onCreate function sets the view to the activity_search_info, and sets each back button & home button with its id and apply finish() & toHome() method so that it can move to the previous page and main page.

The onResume function first gets the nutrient value of the chosen food in a custom pair type, and assigns each value to the right instance variable cal, sugar, na, and fat. Using textViews in activity_info.xml, it displays each nutritional value in a format we designed.

The toHome function changes the view to the MainActivity view.

The toLog function removes any quotations in the selected food name and carries it over to the LoggerActivity class, and changes the view to the LoggerActivity view.

Logger

Was used to save all the user logged foods and their nutrition information by saving the information to a text file called log.txt that was located in the project file. This was created to maintain the creation and addition of logged foods in the user's Log. The Log will save the food name, and the amount of calories, sugar, sodium, and fat of that food. Used the getNutrients method from DBReader to get the nutrient info of the inputted food name. This class was removed, and instead was implemented into the LoggerActivity class due to the fact that reading and writing to a file works differently in Android Studio.

LoggerActivity

This activity adds, displays, and clears the user's logged foods. This was made to allow the user to keep track of what foods they are eating, or want to eat. onCreate, onWindowFocusChanged, and hideSystemUI is included in this class.

The onCreate function sets the view to the activity_logger xml. initStream is then called.

The initStream function checks if there was a food that was carried over from the InfoActivity class. If so, it calls the add function to add that food and it's nutrition info to the log, if not, it just displays the log to the user. First, an InputStreamReader object is created that reads the user's encrypted log.txt file. It decrypts it, and a BufferedReader object is created to read the file. It reads the file line by line and sets each line as a TextView and displays it to the user in a ScrollView.

The add function takes in a food name. It opens the log.txt file, gets the calories, sugar, sodium, and fat of the inputted food name, and the current date, and writes to the file, in a specific format, the food name, calories, sugar, sodium, fat, and date the food was logged. The file is then updated.

The toMain function changes the view to the MainActivity view.

The toGraph function changes the view to the GraphActivity view.

The resetLog function is called when the user presses the “Reset Log” button. It resets the log.txt file by overwriting the current one with a brand new file. It then changes the view to the MainActivity view so the LoggerActivity view is forced to refresh.

GraphActivity

GraphActivity uses the MPAndroidCharts package to plot and graph the data that the user has logged. First, GraphActivity takes all the data in a log.txt file found in the devices’ internal memory. It converts the data into a format that can be plotted out, since the log.txt file already contains information such as date, and calories, and then uses the MPAndroidCharts package to plot out the data.

Pair

A custom defined used to mimic the properties and function of the Java Pair class. Due to the Java Pair class not working the same in Android Studio, we created our own Pair class that does the same thing as the Java Pair class, but for Android Studio.

When created, the Pair object takes in two parameters; the first element, and second element of the pair, and is assigned to the two private variables; x and y. This object is immutable, and there are only two methods that can be called; getFirst() and getSecond(). These methods are fairly self-explanatory. getFirst() returns the first element: x, getSecond() returns the second element: y.

Search

It contains a function match which returns a list which each element of list has a match with input keyword and is organized in lexicographical order. Firstly, the input keyword is turned into all lowercase with no comma and space string. Once checking the length of the input keyword, apply it to a list where it has all the food names. So basically cut each element to the same length with the keyword, and using binary search method in collections, find indexes of elements that match with keyword and go back to a list with all the food names and return a sublist of it with chosen indexes. For instance, when the input keyword is “qui”, then the output list would be such; {“quince,raw”,“quinoa burger patty, frozen”}.

SortingAlgorithms

Contains a function, quickSortLex, that lexicographically sorts the inputted List of Strings via Quicksort. quickSort is the function that is called by anything that needs it. This class was made simply to be called when a list needed lexicographical sorting. This was used just to make the lists of categories and food names displayed to the user to be more organized and easier to find certain foods and categories.

The quickSort function first removes any strings that have quotation marks since quoted strings are sorted before unquoted strings, and some words in the datasets contain quotation marks. It then calls the quickSortLex function, which is where the actual quicksorting occurs.

quickSortLex takes in three parameters; a list of strings, the starting position, and end position. This function recursively calls itself as long as the end position is bigger than the starting position. It first sets and gets the pivot point, which is always chosen to be the last string in the list. It calls the partition function which takes in the same parameters as the quickSortLex function. Once the partition function is finished, it will take the list from the start to the pivot point and recursively call itself to sort that side, and take the list from the pivot point to the end and recursively call itself to sort that side.

The partition function sets the pivot as the last string in the inputted list, and sets the two end positions; left and right, as the inputted start and end positions. It will then increment the left position as long as the string in the corresponding position is lexicographically smaller than or equal to the pivot string and smaller than the end position (the list’s length). It then decrements the right position as long as the string in the corresponding position is lexicographically bigger than the pivot point and bigger than the start position (0). It then swaps the two strings at the left and right positions if the left position did not pass the right position (left < right). This repeats until the left position has passed the right position. Once it has, the left position is returned.

Once all of the sorting is finished, the quickSort function adds back any quotation marks that were removed, and returns the new, sorted list.

TreeBuilder

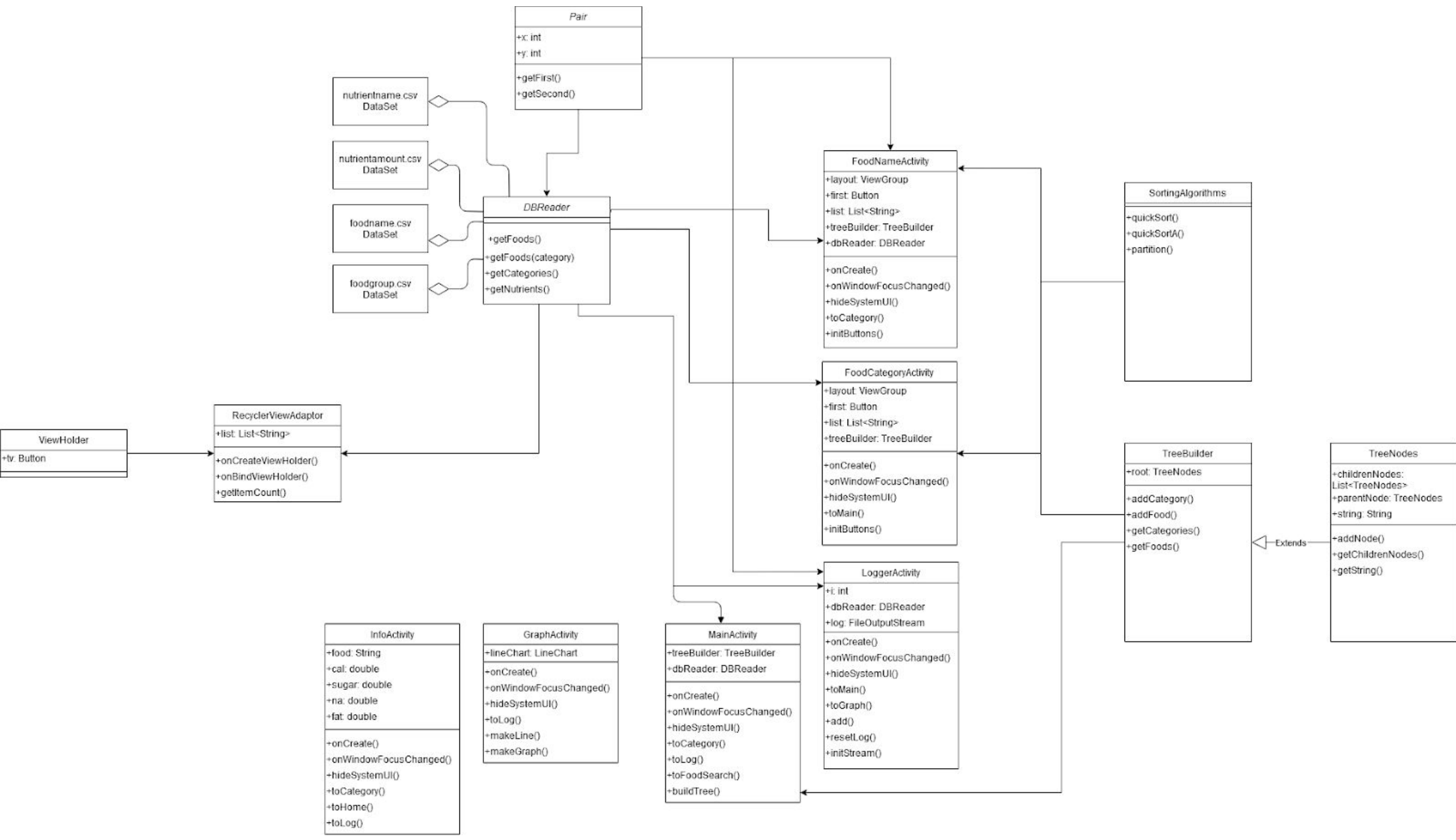
Inherits from Serializable in order function properly in Activity classes. TreeBuilder is a tree representation of the data in the dataset. It first has a root node, which has a list of children nodes which represent the food categories. Then, each of the children nodes has their own lists of children nodes, which represent foods under the food category. For example, milk and cheese would be under dairy.

The TreeBuilder data structure is necessary because without it, every time the user clicked a button to change pages, a new parse would be done and cause a lot of delay. By implementing a TreeBuilder object, the tree would only have to be built once during startup and then the user would never experience the loading delay while using the application.

TreeBuilder has a private class called TreeNodes. TreeNodes is essentially a linked list, except instead of having a single next node it has a list of children nodes. This wat, the nodes “branch out” rather than stay as a single line. TreeNodes has a function that allows the addition of children nodes, getting the children nodes, as well as printing out the name of the current node.

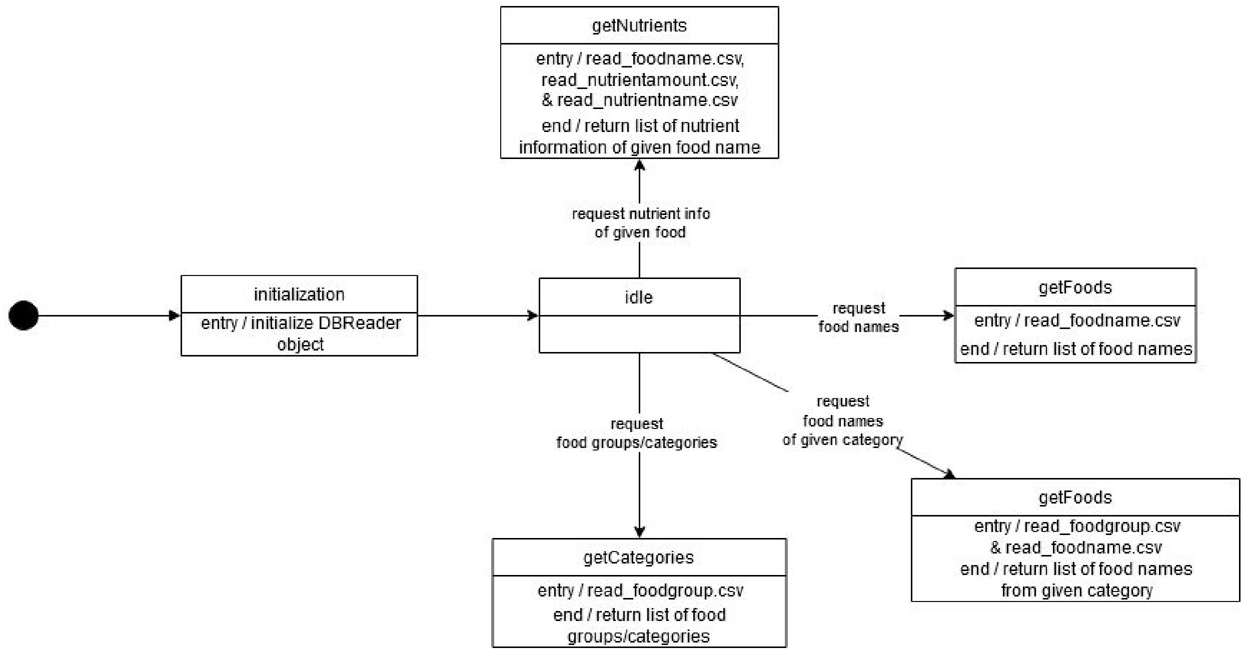
Using TreeNodes, TreeBuilder has the functions addCategory, which adds a new child node directly to the root node, and addFood, which adds a child node to a specified category node. Of course, there are also getters, getCategories which returns the children of the root node, i.e the categories, and getFoods, which returns all the children nodes of a specified category node, i.e the food names in a category.

UML Class Diagram/Uses Relationship



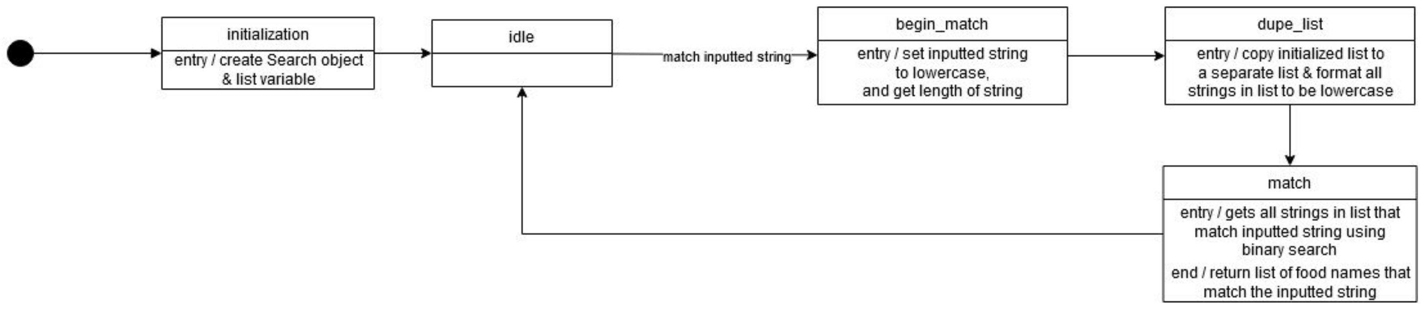
UML State Machine Diagram

DBReader Class



getNutrients
entry / read_foodname.csv, read_nutrientamount.csv, & read_nutrientname.csv
end / return list of nutrient information of given food name
request nutrient info of given food
initialization
entry / initialize DBReader object
idle
request food names
getFoods
entry / read_foodname.csv
end / return list of food names
request food names of given category
request food groups/categories
getFoods
entry / read_foodgroup.csv & read_foodname.csv
end / return list of food names from given category
getCategories
entry / read_foodgroup.csv
end / return list of food groups/categories

Search Class



initialization
entry / create Search object & list variable
idle
match inputted string
begin_match
entry / set inputted string to lowercase, and get length of string
dupe_list
entry / copy initialized list to a separate list & format all strings in list to be lowercase
match
entry / gets all strings in list that match inputted string using binary search
end / return list of food names that match the inputted string

Review/Evaluation of Design

The design adheres to the MVC model. Each of the Activity classes are responsible for providing the user with an interactable, changing interface that is initially built from XML. The activity classes themselves drive the application, and require model objects in order to function. By separating the model and the view/controller, it prevents unexpected side-effects.

The user interface is very user-friendly, and is created to maximize user satisfaction. The application avoids overloading the user with information, yet displays just enough information so that the user doesn't feel that they need to navigate between pages too often.

The application spends a lot of time while loading up in order to build a model of all the data in the CSV files. As a result, this does cause the application to stall as soon as the application begins running, but after the model is built the application runs smoothly with minimal delays. However, this issue could be resolved by using a database file rather than csv files. Since the dataset is massive, multiple files spanning over 500 000 individual data points, and files frequently need to be reread, the actual number of parses done is well into the tens of millions. In situations like this, a CSV reader is simply not ideal, and using a database would be much more efficient. However, because using a database would require many more tools and services, such as servers and .db access tools. Due to the high degree of difficulty, for now the application does not use a database. However, in the future we plan to modify the reader classes to instead use a database in order to speed up the application.

Finally, the application is designed with change in mind. The objects we created have high cohesion and low coupling. They can all be modified rather safely, without causing unexpected side-effects for other objects. We also anticipated that the dataset itself would change. In order to combat this, we created the objects in order to work on ANY dataset with the same layout as the current dataset. This way, if a newer dataset rolls out then the application can easily be updated while causing zero issues to the application.

Overall, the design is very good and was clearly well thought out. Since the developers and the project executives worked together and used the agile method to develop the application, the final product was a little different from what was originally envisioned. There was a feature that was not implemented in the end, the peer-peer health evaluation, but due to budget issues (server costs) the feature was instead changed to a graph of the user’s personal data. This final result however, still far surpassed the project executives’ expectations and ended up as an outstanding product.