Version 042017

Please use this Word file template, following these instructions in gray text, and retain the grayed text. Keep in mind the evaluation criteria, which are at the end of this document. When page limits do not allow you to include supplementary material that you want included, supply it in appendices. These will be read on an as-needed basis only. They should be referred to in the body of the paper.

ASSIGNMENT 1:

BASIC SOFTWARE DESIGN

by Taylor Lapointe

# Pre-existing Project Work

When feasible, build your project on pre-existing work of yours, created either as an individual or as part of team work. Summarize the status of such work in at most half a page. This describes the starting point for your project. Use no more than half a page.

I have no existing project work. I will be starting from scratch. My idea is inspired by a combination of the meal planning app Pepperplate (functionality inspiration) and the agile scrum planning in the JIRA issue management application (UI, process inspiration). In this case it will be agile meal planning, rather than agile sprint planning, and in place of issues in a backlog there will be recipe cards in a recipe box.

# Summary Description

Overall description, in no more than half a page. Your project should be part of a potentially wide scope. For example: a chess-playing program, not tick-tack-toe. The requirements will describe what you plan to build within this wide scope—you are not promising here to build every aspect. Concentrate on ideas that interest you or will be of practical value.

The “Agile Planner” will be a python app to assist users in storing recipes and planning out what to eat each week. Users will be able to add recipes to their recipe box by specifying ingredients, instructions, and supporting meta-data such as prep-time and serving size. Then, the main use case for the app, the user will be able to drag recipe cards from their recipe box into a week calendar layout for each meal time (this will start as command line app, and then move to GUI). Once the planning is complete, the user will be able to indicate planning is complete and the app will generate a grocery shopping list based off of the ingredients listed in each recipe added to the plan. Potential feature enhancements will include having the concept of a pantry where a user can add items that they have in their pantry so that the planner can more intelligently recommend recipes and create the shopping list. Additionally, adding nutritional meta-data to each recipe will allow for factoring in macro-nutrients into a meal planning session for healthier meals.

# I/O Examples

At least two concrete examples of projected output for designated input. Thinking though specific examples will help you to identify the requirements. Use no more than one page.

1. *Input:* A user will select the option to ‘*Add Recipe*.’  
   **Output:** The app will display a series of steps to enter the recipe, followed by the user’s response (application output in bold followed by the user’s input italicized):
   1. **Enter recipe title**: *5-Minute Golden Milk*
   2. **Enter ingredients list** **(comma separated):** *1.5 C light coconut milk, 1.5 cups almond milk, …*
   3. **Enter instructions (comma separated):** *1. To a small saucepan, add coconut milk, almond milk, ground turmeric ..., 2. Whisk to combine…,*
   4. **Enter prep time (in minutes):** *1*
   5. **Enter cook time (in minutes)**: *5*
   6. **How many servings will this make?**: *2*
   7. **Enter source of recipe**: *Minimalist Baker Blog*
   8. **If applicable, enter URL of source:** [*https://minimalistbaker.com/5-minute-vegan-golden-milk/*](https://minimalistbaker.com/5-minute-vegan-golden-milk/)
2. *Input*: The user will scan through the list of recipes provided by the application, and click on “5-Minute Golden Milk”  
   **Output**: The application will prompt the user to what meal they’d like to add the recipe to (Breakfast, Lunch, Dinner, Snack)  
   *Input*: The user will select *Breakfast*  
   **Output**: The application will prompt the user to what day of the week they’d like to add the meal ‘Breakfast: 5-Minute Golden Milk’  
   *Input*: The user will select *Sunday*.  
   **Output:** The application will display the updated meal plan to the user showing Sunday at Breakfast to include ‘5-Minute Golden Milk’
3. *Input*: The user will select *Generate Grocery List*.  
   **Output**: The application will accumulate the shopping list based on ingredients in the added recipes, aggregating common ingredients with updated amounts, and then the application will display the grocery list to the user.   
   *Input*: The user will select remove turmeric from the shopping list as they already have it in their pantry.  
   *Input*: The user will select print shopping list.  
   **Output*:*** The application will send the shopping list to the printer.

# Use Case

Describe a key use case or user story for your proposed application.

The following steps outline the key *Meal Planning* use case of the Agile Planner app.

1. User selects a recipe from their recipe box.
2. User adds a recipe to a meal.
3. User adds the meal to breakfast on one day of the meal plan.
4. Application add the day meal plan to the overall meal plan.
5. Application displays the meal plan to the user.

\*User repeats steps 1-4 until they are satisfied with their meal plan.

1. User selects create grocery list.
2. Application generates and displays the grocery list back to the user.

# Requirements

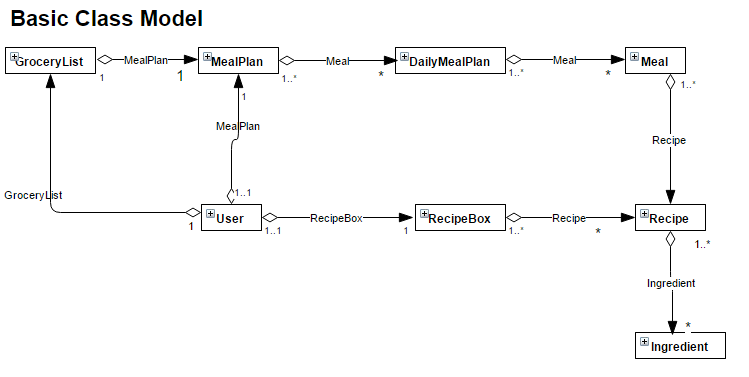
List between 5 and 10 requirements for your project idea. Number them. Use no more than one page. Take care not to make these design or project statements.

1. The application will have an option to ‘Add New Recipe’ to the recipe box.
2. The application, after the user selects ‘Add New Recipe,’ will display a series of prompts for the user to enter the following:
   1. Recipe Title (required)
   2. Ingredients List (required)
   3. Instructions List (required)
   4. Prep Time (not-required)
   5. Cook Time (not-required)
   6. Serving Size (not-required)
   7. Recipe Source (required)
   8. Recipe URL (not-required)
3. A user can add multiple recipes to a meal, then add that meal to a day of the week in their meal plan.
4. The application should generate a grocery list based off of the ingredients of the recipes in the meal plan.
5. The user should be able to edit the grocery list that is displayed back to them to accommodate for ingredients they already have in their pantry.

# Basic Class Model

Produce the core of a class model for your application, containing 5-8 of the most important classes. It should encompass the functionality of the use case described above. Show only non-obvious and key methods.

The below diagram is a basic class model without attributes and methods specified. For a detailed class model including methods and attributes, please see Appendix 1: Detailed Class Model.



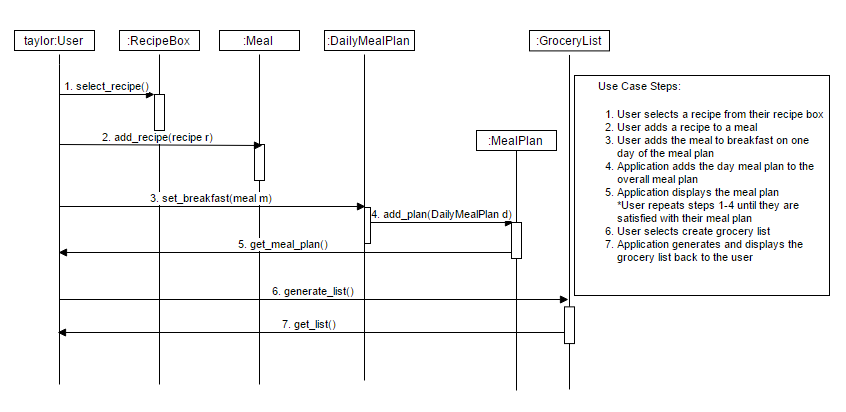
**The main classes for the meal planning application I’ve chosen as follows:**

* MealPlan
  + The MealPlan class will consist of DailyMealPlan objects and a method to add one of those DailyMealPlan objects to the MealPlan class. This will allow the concept of a MealPlan to be flexible in terms of how many days a meal plan will cover.
* DailyMealPlan
  + The DailyMealPlan class will consist of a Meal object for breakfast, lunch, dinner and snack with a date attribute to indicate what day the particular DailyMealPlan object is assigned to. This class will also have a method for setting each of the meals to an instance of the Meal class.
* Meal
  + The Meal class will include a collection of recipe objects. Methods for this class will provide for the ability to add recipes to a meal and to remove recipes from a meal. This class allows for a meal to have multiple recipes.
* Recipe
  + The Recipe class will contain many attributes that make up a recipe such as a description, instructions, and ingredients. The ingredients will be a list of Ingredient class objects in order to accommodate quantity with the ingredient. This class will also have methods to allow for creation of new recipes and a method to add ingredient objects.
* Ingredient
  + The Ingredient class is setup so that an individual ingredient can be listed with quantity and category for grocery shopping. This will also allow for some flexibility in reuse if adding a pantry functionality to the app in future enhancements.
* RecipeBox
  + The RecipeBox class is to hold all of a user’s recipes and will allow for methods to view all recipes and to view selected recipes. This way, the user can browse all of their existing recipes.
* User
  + The user class is setup to allow for user attributes. The user initiates many of the method calls. For expansion in future designs, an abstract user class could be setup so that different types of users such as admin vs general users could be accommodated.
* GroceryList
  + The grocery list consists of items that are generated in a list through a method generate\_list(). This class also offers functionality to the user to add additional grocery shopping items that may not be generated off of the meal plan and for the user to remove items they may already have.

# Sequence Diagram

Give a sequence diagram for your use case above, and relate it to your class model.

The below sequence diagram is a visual representation of the use case documented in step 4 above in a more technical form:

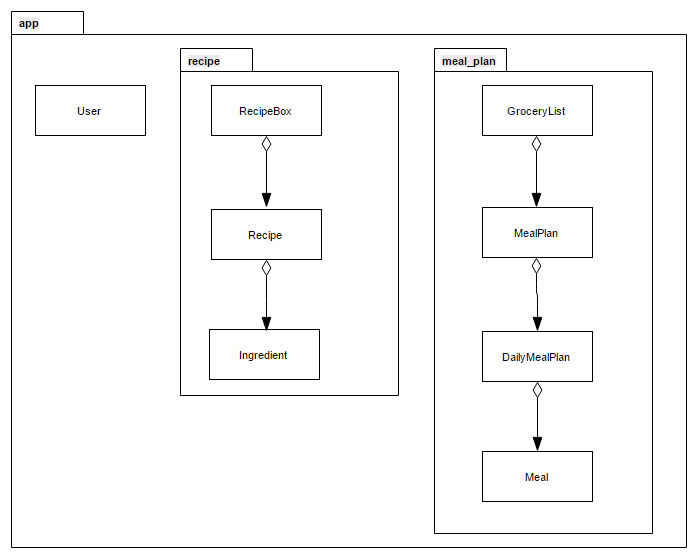


1. The process is started by the instance of the User class, Taylor when Taylor selects a recipe from the recipe box, calling the select\_recipe() function of the RecipeBox class.
2. The user then selects to add the selected recipe to a meal calling the add\_recipe(recipe r) method where the instance of the recipe is passed to the method.
3. The user then selects which meal to add the recipe to a particular meal on a particular day, in this case breakfast, calling the set\_breakfast(meal m) method of the DailyMealPlan class where the meal is passed to the method.
4. The app then sets the daily meal plan that the meal was added to and add that day meal plan to the full meal plan by calling add\_plan(DailyMealPlan d) of the MealPlan class, passing the daily meal plan object to it.
5. The app will then call get\_meal\_plan() of the MealPlan class to return the full meal plan back to the user.
6. When the user is done planning, then can select create grocery list, calling the generate\_list() method of the GroceryList class.
7. The app will call the get\_list() function of the GroceryList class to display the list back to the user.

# Principles of Software Design

Show how your class model can be modularized into packages. Address as many of the software design principles as possible in module 1, and justify, as specifically as possible, why this project is likely to illustrate each of them. Avoid generalities, and focus on your proposed project.

The below diagram displays a possible way the class model can be modularized into packages:



**Explanation of package setup:**

* The app package will contain the user related classes that would be pertinent to the Agile Planner app.
* The recipe package would contain all classes related to recipes. In the case of my class model, this would be the RecipeBox, Recipe and Ingredient classes.
* The meal\_plan package would contain all classes related to meals and the planning of those meals into the final meal plan. In this case, the meal package would contain the Meal, DailyMealPlan, and MealPlan classes, in addition to the GroceryList class.
* With this model, I will be able to add a GUI package for adding a graphical user interface in future enhancements.

**Software design principles and how they will be illustrated in the Agile Planner:**

* Sufficiency
  + As sufficiency is how the software design handles the requirements, I will address this goal by addressing each of the requirements in my design which I have done in the Detailed Class Model found in Appendix 1 by showing the class, methods and attributes that can be implemented to fulfill the requirements listed in part 5 of this document.
* Understandability
  + My software design will fulfill the goal of understandability by adhering to UML standards with provided diagrams and by stating as clearly as possible additional details such as use cases and requirements. I will store these in a docs directory in my GitHub repository for easy access.
* Modularity
  + Modularity, demonstrating that the design is divided into well-defined parts, will be addressed as indicated in the diagram in the first part of this section on packages. My application will be divided into an app package that contains user related info, and two sub-packages: recipe and meal\_plan.
* Cohesion
  + If cohesion is being ‘organized so like-minded elements are grouped together,’ I believe I demonstrate cohesion in my application design in the portion above on organizing my app into packages. Meal planning related classes are grouped into the meal\_plan package and recipe related classes are grouped in the recipe package which is all bundled into an app class that contains user classes.
* Coupling
  + My design does not show any dependent classes, however most are aggregate classes. This may be an area to focus on design improvement.
* Robustness
  + My software design will implement robustness by creating exception classes to handle exceptions and by implementing input validation on all user input fields to ensure that erroneous input is dealt with and that the app continues to run.
* Flexibility
  + I believe the design I have outlined in this document is flexible to handle new requirements that are thrown at it. I believe I could easily add a new package and associated class for a pantry in order for a user to also keep track of items in their pantry. I could add nutritional information for recipes and then use an external API to hook up meal planning with something like FitBit to track calories and macro nutrients. Additionally, I could accommodate different types of users by adding an AbstractUser class that could accommodate basic, premium and admin users. Finally, I believe this to be flexible enough to add a GUI package with associated class to make this have a UI rather than with the current design of being run command-line.
* Reusability
  + I believe the way the design is set now, the recipe and meal\_plan packages described could be reused by others to implement their own meal planning app that would have their own user model and a different form of GUI.
* Information hiding
  + With the design, specifically the input and output that the user provides and sees, I don’t believe that the internal functionality of modules would be displayed to the user.
* Efficiency
  + I don’t believe a meal planning app has any extraordinary speed requirements so executing within an acceptable time and space I would think will be done at implementation phase (reducing nested loops, etc) rather than the design phase.
* Security
  + At this time, this app would run standalone for the user on their local machine and would not require any security implementation beyond input validation as a best practice. If I were to extend this to more users over the web and have user accounts, I would certainly have more robust security requirements and ensure I adhere to defending against the OWASP Top 10. For more, see section 9 below.
* Reliability
  + As the project progresses, I will have exception classes added to ensure that most erroneous behavior is handled appropriately without crashing the app.

# Security Prognosis

Explain the security provisions that should be applied if your application were to be used in real life. Use no more than half a page. Hints: separate security-sensitive elements.

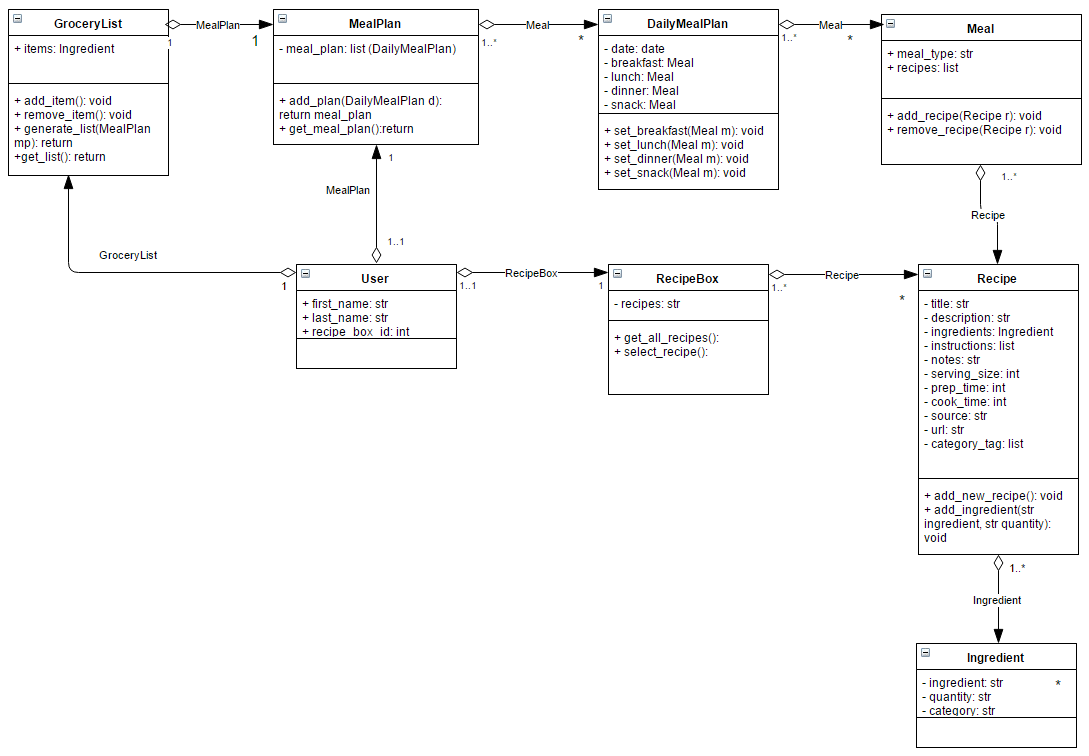
If my application were to be used in real life by the public, I would need to include some sort of authentication and authorization for the application so that users could login and access their unique recipe box and meal plans. I could start by adding an abstract user class so that I could have end users (basic, premium) and admin users. There would be other implementation and configuration steps, possibly including an open source security module to add to the design.

# Appendices

If necessary, supply one or more appendices with material that you want to make available. Appendices will be read on an as-needed basis only. They should be referred to in the body of the paper.

## Appendix 1: Detailed Class Model

To supplement the basic class model above, the below detailed class model includes class attributes and methods:



# Facilitator’s Evaluation

