Southern New Hampshire University

IT 511 Final Project Guidelines and Rubric

Overview

The final project for this course is the creation of a collection manager program. The project is divided into **two milestones**, which will be submitted at various points throughout the course to scaffold learning and ensure quality final submissions. These milestones will be submitted in **Modules Five and Seven.** The final product will be submitted in **Module Nine.**

Substantial software projects are often developed and implemented by utilizing accepted software engineering principles like modularity, encapsulation, and reusability. Throughout this course, you have learned the concepts and processes involved in the development of object-oriented programs. Following established object-oriented principles when writing a program results in modular solutions composed of well-formed classes that can be extended and reused, culminating in an enduring program that solves a problem.

In this final project, you will create a basic program that will help you manage a collection of items that needs to be organized and managed. Your program must meet the requirements of the provided scenario. The creation of this program demonstrates your competency in the use of fundamental data types and more complex data structures and the creation of methods that utilize typical algorithmic control structures, object instantiation, and class definition. In order to make your program enduring, you will be adding inline comments directed toward software engineers about design decisions to facilitate the program's ongoing maintenance, along with generating application programming interface (API) documentation for your programmatic solution that will be directed toward other software developers.

This assessment addresses the following course outcomes:

- Employ suitable data types in object-oriented programs for addressing specific program requirements
- Apply algorithms using appropriate control structures for solving computational problems
- Implement methods that accept parameters and return expected results for addressing given program requirements
- Construct classes with relevant object attributes and behaviors for developing modular, object-oriented solutions
- Utilize appropriate documentation techniques for articulating the purpose and behavior of object-oriented code to specific audiences

Scenario

You will create a program that will help you manage a collection of recipes.

You will implement three classes: one for the main recipe items, one for the ingredients that are part of the recipe, and one for the entire collection of recipes.

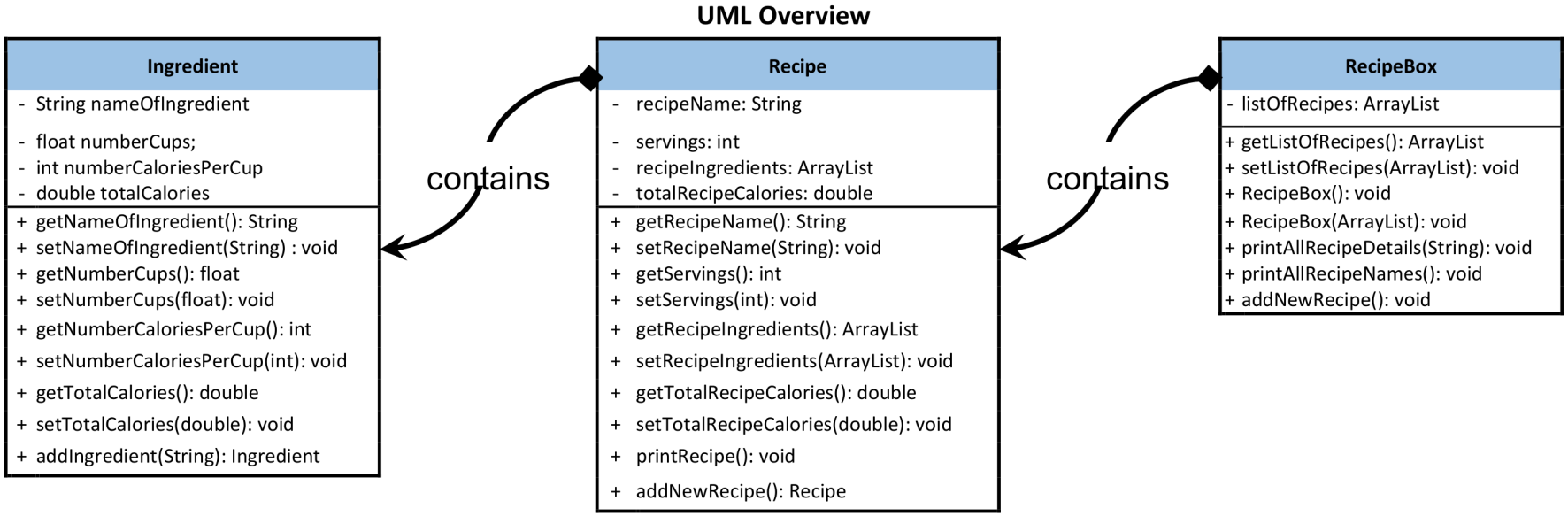
The collection should use a list data structure to store the individual items. Your collection class should include methods like addItem(), printItem(), and deleteItem() that allow you to add, print, or delete items from your collection of items.

Southern New Hampshire University

Your Ingredient class will model the items that will be stored in each recipe in your collection. You will give it some basic attributes (of numeric or string types) and some basic methods (accessors/mutators, printItemDetails(), etc.).

Your Recipe class will start off similar to your Ingredient class, but you will increase its functionality by modifying it to accept the Ingredient objects, containing all the details stored in an Ingredient class object. You will also expand the Recipe class by adding recipe-specific methods to your Recipe class.

The basic, foundational elements are shown in the following Unified Modeling Language (UML) diagram for the required classes:



Finally, you will also write an application driver class that should allow the user to create a new recipe and add it to the collection. In addition, it should allow the user to see a list of items in the collection and then give the user an option to either see more information about a particular item (by retrieving it from the collection) or edit an item that is already in the collection. Finally, your program should allow the user to delete an item from the collection.

Moreover, you will add documentation to the application that will contain inline comments explaining your design decisions as well as documentation comments that will be used to generate API documentation for your programmatic solution for other software developers.

To prepare for the final project, you will complete a series of six stepping stone assignments and two final project milestones that will help you learn the coding skills required for the project. Separate documentation for these assignments is included in the course resources.

Southern New Hampshire University

Prompt

You have been tasked with developing a complete, modular, object-oriented program that will allow for the management of a collection. The scenario provided to you outlines all of the program's requirements. Refer to the provided scenario to make the determinations for all data types, algorithms and control structures, methods, and classes used in your program. Your final submission should be a self-contained, fully functional program that includes all necessary supporting classes. Furthermore, you must provide inline comments in your program design that software engineers would be able to utilize for the ongoing maintenance of your program. Your programmatic solution should also be communicated through application programming interface (API) documentation to other programmers.

Specifically, the following **critical elements** must be addressed:

I. **Data Types:** Your final program should properly employ each of the following data types that meet the scenario's requirements where necessary:
   A. Utilize numerical data types that represent quantitative values for variables and attributes in your program.
   B. Utilize strings that represent a sequence of characters needed as a value in your program.
   C. Populate a list or array that allows the management of a set of values as a single unit in your program.
   D. Utilize inline comments directed toward software engineers for the ongoing maintenance of your program that explain your choices of data types you selected for your program.

II. **Algorithms and Control Structure:** Your final program should properly employ each of the following control structures as required or defined by the scenario where necessary:
   A. Utilize expressions or statements that carry out appropriate actions or that make appropriate changes to your program's state as represented in your program's variables.
   B. Employ the appropriate conditional control structures that enable choosing between options in your program.
   C. Utilize iterative control structures that repeat actions as needed to achieve the program's goal.
   D. Utilize inline comments directed toward software engineers for the ongoing maintenance of your program that explain how your use of algorithms and control structures appropriately addresses the scenario's information management problem.

III. **Methods:** Your final program should properly employ each of the following aspects of method definition as determined by the scenario's requirements where necessary:
   A. Use formal parameters that provide local variables in a function's definition.
   B. Use actual parameters that send data as arguments in function calls.
   C. Create both value-returning and void functions to be parts of expressions or stand-alone statements in your program.
   D. Create unit tests that ensure validity of the methods.
   E. Invoke methods that access the services provided by an object.
   F. Employ user-defined methods that provide custom services for an object.

G. Utilize inline comments directed toward software engineers for the ongoing maintenance of your program that explain the purpose of the methods you implemented in your program.

IV. **Classes:** Construct classes for your program that include the following as required by the scenario where necessary:
   A. Include attributes that allow for encapsulation and information hiding in your program.
   B. Include appropriate methods that provide an object's behaviors.
   C. Create a driver class that instantiates objects for testing the constructed classes.
   D. Utilize inline comments directed toward software engineers for the ongoing maintenance of your program that explain the decisions you made in the construction of the classes in your program.

V. Produce **reference documentation** that communicates your application programming interface (API) to other programmers, using a documentation generator (Javadoc, Doxygen, etc.).

Milestones

Milestone One: *Ingredient Class*

In **Module Five**, you will create a complete class based on Stepping Stone Labs Two and Three and provide it the basic attributes with the appropriate data types. Additionally, you will add code to validate the data type of the user input. This class will be modified for the submission of your final project application; however, it should be functional code that accepts user input for each variable. **This milestone will be graded with the Milestone One Rubric.**

Milestone Two: *Recipe Class*

In **Module Seven**, you will focus your skills finalizing your final project code by submitting a class complete with accessor/mutator, constructor, and "custom" programmer-defined methods. **This milestone will be graded with the Milestone Two Rubric.**

Final Submission: *Collection Manager Program*

In **Module Nine**, you will submit your final project. It should be a complete, polished artifact containing **all** of the critical elements of the final product. It should reflect the incorporation of feedback gained throughout the course. **This submission will be graded with the Final Project Rubric.**

Final Project Rubric

Guidelines for Submission: Your complete program should be submitted as a zip file of the exported project and the reference documentation from your documentation generator.

Southern New Hampshire University

Critical Elements	Exemplary	Proficient	Needs Improvement	Not Evident	Value
Data Types: Numerical		Utilizes numerical data types that represent quantitative values for variables and attributes in the program, meeting the scenario's requirements (100%)	Utilizes numerical data types that represent quantitative values for variables and attributes in the program, but use of data types is incomplete or illogical, contains inaccuracies, or lacks accordance with the scenario's requirements (70%)	Does not utilize numerical data types that represent quantitative values for variables and attributes in the program (0%)	6.34
Data Types: Strings		Utilizes strings that represent a sequence of characters needed as a value in the program, meeting the scenario's requirements (100%)	Utilizes strings that represent a sequence of characters needed as a value in the program, but use of strings is incomplete or illogical, contains inaccuracies, or lacks accordance with the scenario's requirements (70%)	Does not utilize strings that represent a sequence of characters needed as a value in the program (0%)	6.34
Data Types: List or Array		Populates a list or array that allows the management of a set of values as a single unit in the program, meeting the scenario's requirements (100%)	Populates a list or array that allows the management of a set of values as a single unit in the program, but population is incomplete or illogical, contains inaccuracies, or lacks accordance with the scenario's requirements (70%)	Does not populate a list or array that allows the management of a set of values as a single unit in the program (0%)	6.34
Data Types: Inline Comments	Meets "Proficient" criteria and inline comments demonstrate an insightful awareness of adapting documentation techniques to specific audiences (100%)	Utilizes inline comments directed toward software engineers for the ongoing maintenance of the program that explain the choices of data types selected for the program (90%)	Utilizes inline comments that explain the choices of data types selected for the program, but inline comments are incomplete or illogical, contain inaccuracies, or lack applicability toward software engineers for the ongoing maintenance of the program (70%)	Does not utilize inline comments that explain the choices of data types selected for the program (0%)	3.8

Algorithms and Control Structures: Expressions or Statements		Utilizes expressions or statements that carry out appropriate actions or that make appropriate changes to the program's state as represented in the program's variables and meet the scenario's requirements (100%)	Utilizes expressions or statements that carry out actions or that make changes to the program's state as represented in the program's variables, but use of expressions or statements incomplete or illogical, contains inaccuracies, or lacks accordance with the scenario's requirements (70%)	Does not utilize expressions or statements that carry out actions or that make changes to the program's state as represented in the program's variables (0%)	6.34
Algorithms and Control Structures: Conditional Control Structures		Employs the appropriate conditional control structures, as the scenario defines, that enable choosing between options in the program (100%)	Employs the conditional control structures that enable choosing between options in the program, but use of conditional control structures is incomplete or illogical, contains inaccuracies, or lacks accordance with the scenario's definition (70%)	Does not employ the conditional control structures that enable choosing between options in the program (0%)	6.34
Algorithms and Control Structures: Iterative Control Structures		Utilizes iterative control structures that repeat actions as needed to achieve the program's goal as required by the scenario (100%)	Utilizes iterative control structures that repeat actions to achieve the program's goal, but use of iterative control structures is incomplete or illogical, contains inaccuracies, or lacks accordance with the scenario's requirements (70%)	Does not utilize iterative control structures that repeat actions to achieve the program's goal (0%)	6.34
Algorithms and Control Structures: Inline Comments	Meets "Proficient" criteria and inline comments demonstrate an insightful awareness of adapting documentation techniques for specific audiences (100%)	Utilizes inline comments directed toward software engineers for the ongoing maintenance of the program that explain how the use of algorithms and control structures appropriately addresses the scenario's information management problem (90%)	Utilizes inline comments that explain how the use of algorithms and control structures addresses the scenario's information management problem, but inline comments are incomplete or illogical, contain inaccuracies, or lack applicability toward software engineers for the ongoing maintenance of the program (70%)	Does not utilize inline comments that explain how the use of algorithms and control structures addresses the scenario's information management problem (0%)	3.8

Methods: Formal Parameters		Uses formal parameters that provide local variables in a function's definition as determined by the scenario's requirements (100%)	Uses formal parameters that provide local variables in a function's definition, but use of formal parameters is incomplete or illogical, contains inaccuracies, or lacks accordance with the scenario's requirements (70%)	Does not use formal parameters that provide local variables in a function's definition (0%)	3.17
Methods: Actual Parameters		Uses actual parameters that send data as arguments in function calls as determined by the scenario's requirements (100%)	Uses actual parameters that send data as arguments in function calls, but use of actual parameters is incomplete or illogical, contains inaccuracies, or lacks accordance with the scenario's requirements (70%)	Does not use actual parameters that send data as arguments in function calls (0%)	3.17
Methods: Value-Returning and Void Functions		Creates both value-returning and void functions to be parts of expressions or stand-alone statements in the program as determined by the scenario's requirements (100%)	Creates both value-returning and void functions to be parts of expressions or stand-alone statements in the program, but functions are incomplete or illogical, contain inaccuracies, or lack accordance with the scenario's requirements (70%)	Does not create both value-returning and void functions to be parts of expressions or stand-alone statements in the program (0%)	3.17
Methods: Unit Tests		Creates unit tests that ensure validity of the methods as required by the scenario (100%)	Creates unit tests that ensure validity of the methods, but unit tests are incomplete or illogical, contain inaccuracies, or lack accordance with the scenario's requirements (70%)	Does not create unit tests that ensure validity of the methods (0%)	3.17
Methods: Access Services Provided		Invokes methods that access the services provided by an object as required by the scenario (100%)	Invokes methods that access the services provided by an object, but called methods are incomplete or illogical, contain inaccuracies, or lack accordance with the scenario's requirements (70%)	Does not invoke methods that access the services provided by an object (0%)	3.17

Methods: User-Defined Methods		Employs user-defined methods that provide custom services for an object as specified in the program requirements (100%)	Employs user-defined methods that provide custom services for an object, but use of user-defined methods is incomplete or illogical, contains inaccuracies, or lacks accordance with the specifications in the program requirements (70%)	Does not employ user-defined methods that provide custom services for an object (0%)	3.17
Methods: Inline Comments	Meets "Proficient" criteria and inline comments demonstrate an insightful awareness of adapting documentation techniques to specific audiences (100%)	Utilizes inline comments directed toward software engineers for the ongoing maintenance of the program that explain the purpose of the methods implemented in the program (90%)	Utilizes inline comments that explain the purpose of the methods implemented in the program, but inline comments are incomplete or illogical, contain inaccuracies, or lack applicability toward software engineers for the ongoing maintenance of the program (70%)	Does not utilize inline comments that explain the purpose of the methods implemented in the program (0%)	3.8
Classes: Attributes		Includes attributes, as required by the scenario, that allow for encapsulation and information hiding in the program (100%)	Includes attributes that allow for encapsulation and information hiding in the program, but inclusion is incomplete or illogical, contains inaccuracies, or lacks accordance with the scenario's requirements (70%)	Does not include attributes that allow for encapsulation and information hiding in the program (0%)	6.34
Classes: Behaviors		Includes appropriate methods that provide an object's behaviors, as required by the scenario (100%)	Includes methods that provide an object's behaviors, but inclusion of methods is incomplete or illogical, contains inaccuracies, or lacks accordance with the scenario's requirements (70%)	Does not include methods that provide an object's behaviors (0%)	6.34
Classes: Driver Class		Creates a driver class that instantiates objects for testing the constructed classes as specified in the scenario (100%)	Creates a driver class that instantiates objects for testing the constructed classes, but driver class is incomplete or illogical, contains inaccuracies, or lacks accordance with specifications in the scenario (70%)	Does not create a driver class that instantiates objects for testing the constructed classes (0%)	6.34

Classes: Inline Comments	Meets “Proficient” criteria and inline comments demonstrate an insightful awareness of adapting documentation techniques to specific audiences (100%)	Utilizes inline comments directed toward software engineers for the ongoing maintenance of the program that explain the decisions made in the construction of the classes in the program (90%)	Utilizes inline comments that explain the decisions made in the construction of the classes in the program, but inline comments are incomplete or illogical, contain inaccuracies, or lack applicability toward software engineers for the ongoing maintenance of the program (70%)	Does not utilize inline comments that explain the decisions made in the construction of the classes in the program (0%)	3.8
Reference Documentation		Produces reference documentation that communicates the API to other programmers, utilizing a documentation generator (100%)	Produces reference documentation that communicates the API to other programmers, but documentation is incomplete or contains inaccuracies (70%)	Does not produce reference documentation that communicates the API to other programmers (0%)	3.8
Readability	Meets “Proficient” criteria with an organized structure that separates components with different responsibilities and/or groups related code into blocks (100%)	Code follows proper syntax and demonstrates deliberate attention to indentation, white space, and variable naming (90%)	Code follows proper syntax, but there are variations in indentation, white space, or variable naming (70%)	Code does not follow proper syntax (0%)	4.92
				Total	**100%**