Software Documentation

Course: [COP-2805C-86399 Java Advanced](https://hcc.instructure.com/courses/135902)

Assignment: Demonstrate OOP principles for Interfaces

Date: 3/4/2024

Author Details: Jose A. Cañellas

# Requirements

* The program must include 4 dinner classes, BackyardBBQ, ThanksgivingDinner, and HeartyStew. The first class, BackyardBBQ, uses a charcoal or propane grill to cook backyard BBQ. The second class, ThanksgivingDinner, uses an oven and a roasting pan to cook a turkey. The third class, HeartyStew, uses a crockpot or stovetop pot to cook stew. Each of these three classes must implement the interface Meal\_Cook that is supplied with this assignment. You are free to select any attributes and methods you like for these classes, so long as you follow good practices for encapsulation and include a proper toString() method. The fourth class is CookingDinner. This will include a runnable main() method. When running, the program should print out the course, assignment, date, and student name to the screen. Then instantiate an object of each type of dinner and print the status of each object to the screen.
* The BackyardBBQ Class has the following:
  + Setter and Getter methods for:
    - The Menu of choice
    - The Meal of choice
    - The Grill of choice
  + A toString method that displays the choices object.
  + Implemented Cook method from Meal\_Cook that returns the meal of choice and its preparation status.
* The ThanksgivingDinner Class has the following:
  + Setter and Getter methods for:
    - The First Side of choice
    - The Second Side of choice
  + A toString method that displays the choices object.
  + Implemented Cook method from Meal\_Cook that returns the sides of choice and their preparation status.
* The HeartyStew Class has the following:
  + Setter and Getter methods for:
    - The Menu of choice
    - The Stew of choice
  + A toString method that displays the choices object.
  + Implemented Cook method from Meal\_Cook that returns the stew of choice and its preparation status.
* The CookingDinner Class must have a runnable Main method and the following:
  + A way to instantiate objects of each class.
  + A way to call the cook method of all objects to display the preparation status of each.

# Installation and Run Instructions

* The user must have any Operating System compatible with the latest version of Visual Studio Code (Windows 10 and 11, Linux, MacOS 10.15 or above)
* The user must install Visual Studio Code and install the “Coding Pack for Java” in the following link (https://code.visualstudio.com/docs/languages/java). The “Extension Pack” is also recommended.
* Four .java source files will be provided, and the user must open a Java project where they can drag these files into the “src” tab in Explorer.
* On the upper left, click run and then “Start Debugging” or “Run without Debugging” to run CookingDinner.java.

# Design Notes and UML Diagram

* When approaching this, the first thought was what to add to the various classes other than just having them print a preparation status. The initial ideas involved having the user choose from a menu of meals that each type of dinner could have, then the corresponding preparation status will be printed. Yet, it still felt like it could have more, so I expanded the idea to having different menus to pick from:
  + Considering that many restaurants have a regular menu and a gluten free menu, for BackyardBBQ it was decided that there will be two menus (Regular Menu and Gluten Free Menu) with 4 meals each.
  + When it comes to ThanksgivingDinner, a Turkey is a staple of the dinner but that also goes for the sides. Instead of having two different menus it was decided to have a list of 4 side dishes and have the user pick two of those options.
  + For HeartyStew, the instructions mention the use of a Crockpot or a Stovetop Pot both of which affect the preparation of each stew unlike the choice of a Propane or Charcoal Grills in BackyardBBQ. With that in mind, there are 4 stews for each Pot to choose from.
* Another thing that was considered was to facilitate the user with a way to correct their input the same way it was done in Project 4 (Demonstrate OOP principles for interfaces). In that project, a user has the ability to create a number of events and then fill in the details for each one. After the input of each detail is received, a method called inputCorrection(String) or integerInputCorrection(int) asks the user if they wish to change their input and allows as many corrections as they need. Originally those methods were inside a Class called Event from which others inherited these methods, but in the case of this project they were both placed inside their own class called InputCorrection:
  + The method used to correct Strings remains the same with it accepting the parameters of the field name and the user’s input. With those parameters, the method will show the user their input and ask if they wish to correct it. If yes, the user enters the corrected input and is then asked if it is now correct at which point it exits if the user confirms or loops back to request correction if the user says it is not correct. If no, the method simply returns their input and it is assigned to the variable.
  + The method to correct integers had some slight changes. While the process works the same as for Strings, two new parameters were added called minRange(int) and maxRange(int). Since the menus have four options to pick, it was important to keep the possible correction within range of the options to prevent the method from returning anything outside of the options. If a user initially inputs a number outside of the range, an error will be displayed reminding the user to stay within the range of the options. The same thing will happen within the method, if a correction is outside of the range then the input won’t be accepted and an error will be displayed to the user.
* The way BackyardBBQ works is by providing the user with two menus, one with regular meals and the other with Gluten Free meals. These menus include four different meals, each with their own preparation methods/status.
  + First the user is prompted to pick one of the menus and from their choice they are prompted to pick the meal they wish to prepare.
  + With a meal selected, the user is given the option to prepare the meal in a charcoal or propane grill, and their option is concatenated to the String of the meal preparation.
  + With the selections done, the cook method from Meal\_Cook is implemented and it uses an if/else if statement to pick the correct menu and switch statements that will then pick the correct preparation String to return and print for the user.
* HeartyStew works the same way that BackyardBBQ does with the only difference being that the two menus correspond to the preparation method, Crockpot or Stovetop Pot. The implemented cook method from Meal\_Cook will use the same process as BackyardBBQ to return the correct Stew preparation.
* Unlike BackyardBBQ and HeartyStew, ThanksgivingDinner doesn’t have two menus to choose from. Instead, it gives the user the option to pick two sides from the four provided to accompany the Turkey that will go with the dinner.
  + To do this, the user is prompted to pick their first side. When selected, the number of their choice will be assigned to a variable that the set method for the second side will use to make sure their input is not the same one.
  + For the second side choice, the user is prompted for their choice, then the input is checked to make sure it is within the range of choices and not the same as their same choice.
    - In case that the user corrects their input to be that of the first choice, the set method has an if statement that only assigns the input if it is different to that of the first side. If it is the same then an error will be printed and the user is once again asked to pick their side.
* The other difference in ThanksgivingDinner is the way that it implements the cook method from Meal\_Cook. A method called sideDishPreparation includes all the side dish preparation Strings that are returned to cook depending on the side choices. After both side dishes have been assigned their correct preparation String, cook will return a concatenated String with the status of the Turkey, the first side dish, and the second side dish.
* The CookingDinner Class welcomes the user to the program and explains what it is used for. It then creates three objects for each and begins calling the method in each class that calls all of the set methods at once. After the user makes their choices from each Class, the toString method is called to display their choices before moving on to the next Dinner. It is only after all choices have been made for all objects that the preparation status for each Dinner is printed with their implemented cook methods.

**UML Diagram:**

A screenshot of a computer program

Description automatically generated

# Test Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement | Input/Action | Expected | Output | Pass/Fail |
| For the various set methods, user input must stay within the given range of choices. | When correcting selected choice, attempted to input numbers outside the range. | An error will be printed warning the user to stay within the range of numbers given. | The correction method properly caught the attempts to exit range and prints the error before requesting for input. | PASS |
| If the user changes their choice using integerInputCorrection(), the corrected input is correctly returned and processed. | By using breakpoints, the changes in variables were tracked while correcting the input in all set methods. | The integerInputCorrection() method returns the corrected input after the user indicates that they are done correcting their choice. | IDE shows the corrected input was properly returned and the preparation methods were properly chosen. | PASS |
| When a user makes their choices, the implemented cook methods in every class return the proper preparation Strings. | After selecting, corrected every input and kept track to make sure the cook() methods return the correct Strings. | The cook() method of every class prints the correct String for every choice made. | The cook () method of every class correctly printed the String of every dish chosen and corrected. | PASS |
| In ThanksgivingDinner, if the user selects the same dish as the first when choosing the second then the program won’t accept the input. | When selecting the second side dish, attempted to choose the same selection as the first. | An error reminding the user to make a different selection is printed and they are once again prompted to select again. | The error was correctly displayed every time and was prompted for selection. | PASS |
| In ThanksgivingDinner, if the user is correcting the input for the second side dish, they are not allowed to make the same choice as the first side dish. | When prompted for correction, attempted to select the first side dish once again. | integerInputCorrection() returns the input but the loops in setSecondSideDishChoice() detect the similarity and prints an error reminding the user to make a different choice then prompts to select again. | As expected, the loop detects the similarity and prints the error before prompting for selection again. | PASS |
| The sideDishPreparation() method inside ThanksgivingDinner returns the preparation String for the chosen dishes. | Ran the program multiple times to use a variation of different choices for both side dishes. | sideDishPreparation() returns the correct preparation String for each side dish to be concatenated with the string returned by the cook() method. | The correct preparation Strings for each dish were returned and printed by the cook() method. | PASS |
| The Abstract method, cook(), from Meal\_Cook is properly implemented by each Class. | Ran the program multiple times to make different choices for each Class. | When the cook() methods are called, the correct preparation Strings are printed. | cook() correctly prints out the preparation status of each Class. | PASS |
| The new Class, InputCorrection, works when called by each of the Dinner Classes. | Whenever asked for input, watch to see if correction of the input was requested and corrected every input for every choice. | After selecting, the methods within InputCorrection are called which are supposed to ask if the user wishes to make a correction, displays the corrected input, and asks if it is correct, and only returns the input when the user indicates they are done with corrections. | The correction methods ran as intended, allowed for corrections, and returned the corrected inputs. | PASS |

# A screenshot of a computer program Description automatically generatedScreenshots

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated