MET cS665 Summer 2 2018

Abel Gancsos

Abel Gancsos productions | Hawthorne, NJ

AMGVending

2018

Contents

Overview 2

Prerequisites 3

Assumptions 3

1 Application Description 4

1.1 Implementation Details 4

2 Assignment Tasks 5

2.1 Implementation Description 5

2.2 UML Class Diagram 6

2.3 UML Sequence Diagram 7

Bibliography 8

Appendices 9

Appendix A – Setup 9

Appendix A – Building project 9

Appendix B – Running executable 10

# Overview

As more and more retail experiences becoming more automated, HotDrinks Brewing Company of New Jersey is looking towards joining this movement. They have been offering their brewed beverages to the masses by having an onsite barista at many locations; however, they now see the inefficiencies of this approach. They are looking to design an automated vending machine that can provide their current drink offers to patrons, use their proprietary brewing mechanism, and then provide the same drink to customers without having a barista at their location. They already have the hardware designed; however, they now need the software controller that makes the magic happen.

Some requirements of the controller is that it should offer the same drinks as they currently have, such as Coffee Espresso, Coffee Americano, Coffee Latte Macchiato, Black Tea, Green Tea, and Yellow Tea. If this concept works for them, they may add other options as they see fit. The drinks should also be customizable with condiments, such as sugar or milk. For now, condiments will be limited to a quantity of 3 each, so that we can avoid issues brought on by user input. This limit might eventually be raised or lifted completed; however, this is a requirement for now.

# Prerequisites

* C++ compiler with C++11 capability
* CMake version 3.11 or higher
* XCode command-line tools (MacOS only)
* Visual Stuido 2017 or higher (Windows only)

# Assumptions

* The dispensing mechanism works or the software group is not responsible for any mechanical bugs.
* Maybe some day the vending machine is to be run in a batchable mode, therefore, certain switches or flags are built into the controller.
* Additional drink types will be added in the future.
* Additional drink sorts will be added in the future.
* Additional drink condiments will be added in the future.
* The restriction on drink condiment quantity may be raised or lifted completely.
* The payment has already been provided before the controller is used.

# 1 Application Description

## Implementation Details

*The following details are also available in the README.md file*

The following utility, AMGVending, acts as a controller for a new fully automated beverage vending machine. This is to say that run in interactive mode, the utility should be prepared for a customer by having an available prompt, in this case in a shell command window. Once a patron arrives and is ready to order, they select from the menu options the type of drink they are interested in. The controller then prompts for the subtype or sort of drink the patron would like to have, and then it prompts the patron if they would like any toppings or condiments. If the patron selects to add condiments, the controller then displays a menu option for available condiments. The vending machine then prompts the patron for quantity of the condiment, which must be within the valid scope. Then vending machine then prepares the drink and dispenses it to the patron. The controller then goes back to the main menu for the next transaction.

In order to handle future drink types, the controller uses abstract classes for the drink objects. As for the drink sorts, each derived class of the abstract drink type comes with a sort enum, which will be primarily used for input validation. As for the condiments, we use another abstract class as we suspect that additional condiments might be added in the future.

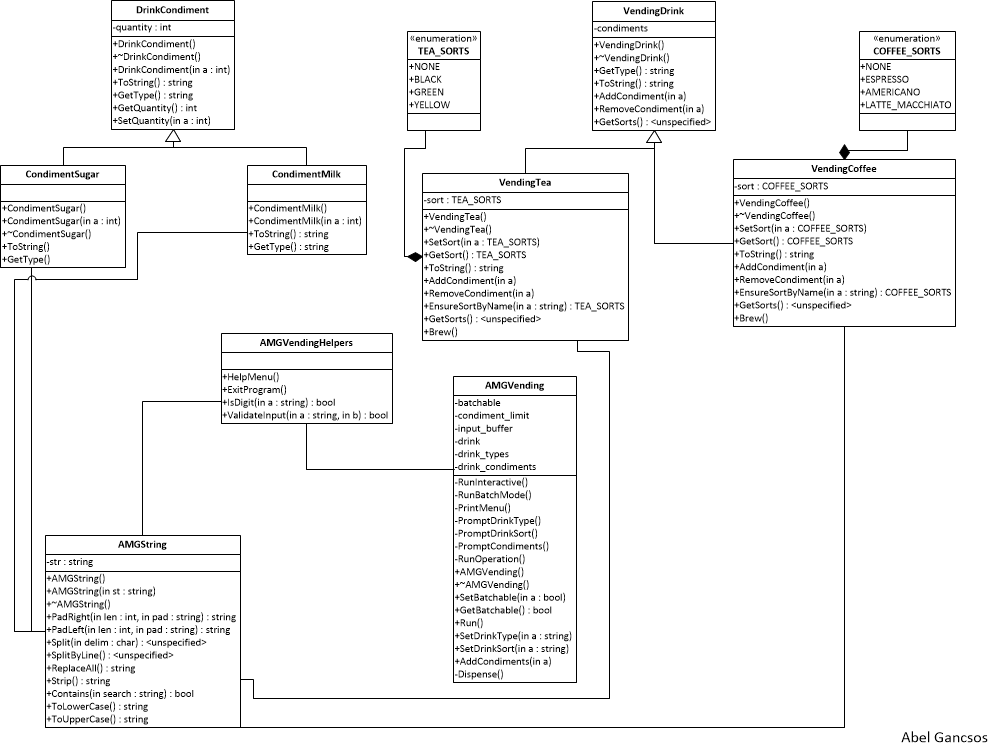
The controller then uses a "session" class as a wrapper to build out the transaction, which holds details of the selected options.

# 2 Assignment Tasks

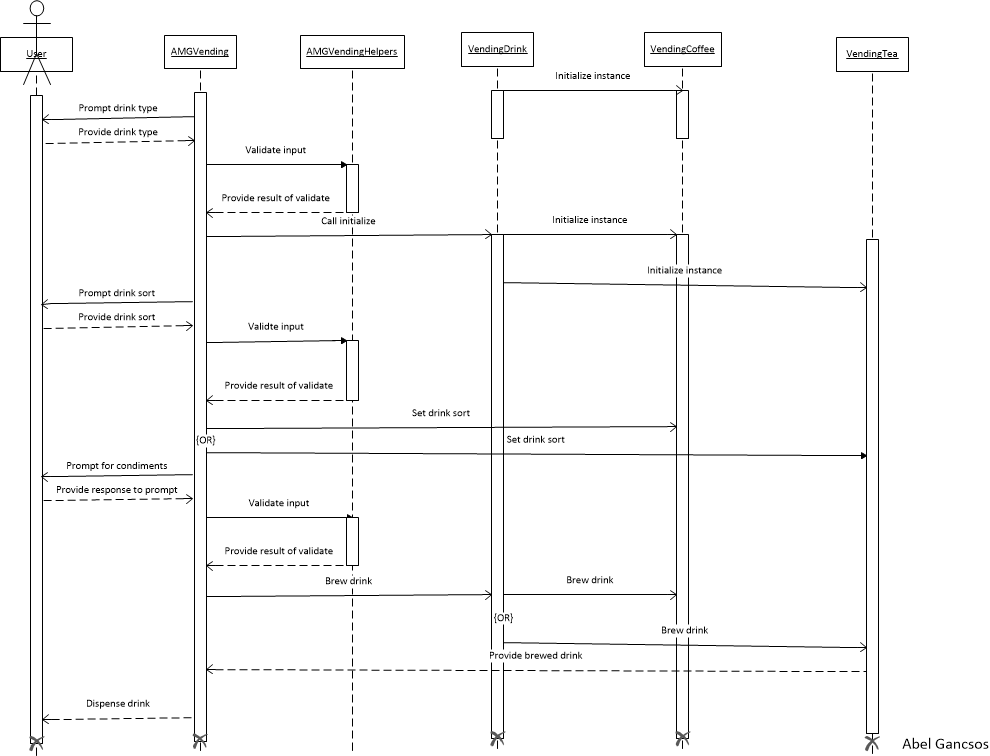
## 2.1 Implementation Description

*The following details are also available in the README.md file*

## 2.2 UML Class Diagram



## 2.3 UML Sequence Diagram



# Bibliography

Döring, A. (2016, 11 07). *minimal\_cmake\_example*. Retrieved 07 03, 2018, from GitHub Web site: https://github.com/krux02/minimal\_cmake\_example

Google Corporation. (2018, 06 19). *Google C++ Style Guide*. Retrieved 07 03, 2018, from Google GitHub styles: https://google.github.io/styleguide/cppguide.html

Kitware Corporation. (n.d.). *Download*. Retrieved 07 03, 2018, from CMake Web site: https://cmake.org/download/

Lucid Software Corporation. (2017). *Class Diagram Tutorial*. Retrieved 7 25, 2017, from Lucid Software Corporation Web site: https://www.lucidchart.com/pages/uml/class-diagram

Microsoft Corporation. (2010). Visio. (2010). Redmond, WA, USA.

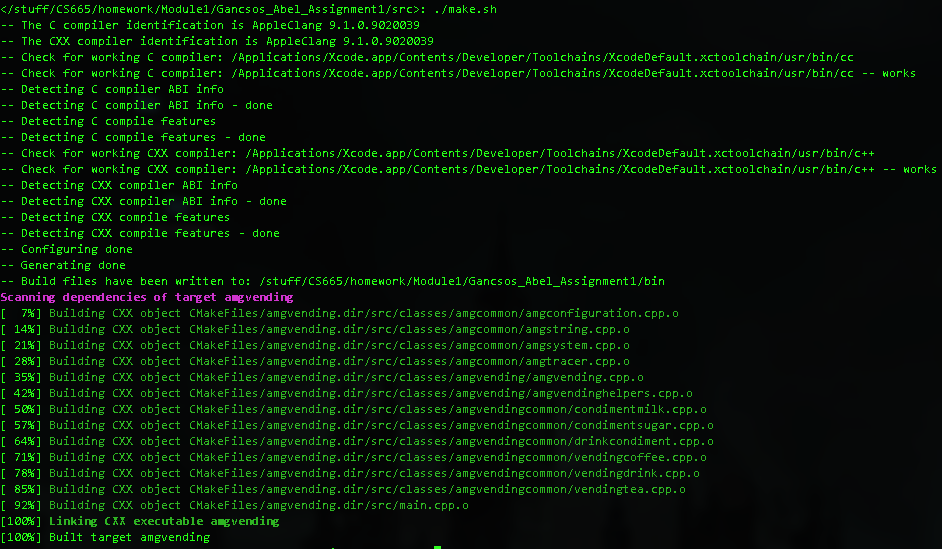
Oracle Corporation. (2017). VirtualBox. (5.1.18). Redwood City, CA, USA.

# Appendices

## Appendix A – Setup

The project can be build using the make.sh or make.bat scripts, depending on the platform. Both of these scripts will run cmake to generate the CMakeFiles and then run the make command in the bin directory. Alternatively, the project can be built manually using the CMakeLists.txt file that comes with the package.

## Appendix A – Building project



## Appendix B – Running executable

