# Problem Overview

Create a simulator of an automatic coffee dispensing machine, called the "*Barista-matic"*.

The machine maintains an inventory of drink ingredients, and is able to dispense a fixed set of possible drinks by combining these ingredients in different amounts.

# Problem Statement in Points

1. Upon startup, the Barista-matic should display a list of its current inventory, followed by a menu to allow the user to select a drink.
2. As drinks are dispensed, the inventory should be updated.
3. Only drinks for which there is sufficient inventory can be dispensed.
4. Initially the Barista-matic should contain 10 units of all ingredients.
5. Restocking the machine should restore each ingredient to a maximum of 10 units.
6. If the user enters an invalid command, then the program should display a single-line message with the following format:

Invalid selection: <characters that were entered>

1. If the user selects a valid drink number, and the machine has the required ingredients to make the drink, then the program should display a single-line message with the following format:

Dispensing: <drink name>

1. if the drink order cannot be completed, then the program should display a single line message with the following format:

Out of stock: <drink name>

# Restrictions for Input and Output Formats

1. The solution should be a command-line program written in Java.
2. The specified input and output formats for the Barista-matic must be followed exactly.
3. All output should be written to the standard output stream.
4. The output generated by program should not have any whitespace at the beginning of a line.
5. The inventory list should be displayed in alphabetic order by ingredient name.
6. The drink menu should be displayed in alphabetic order by drink name.
7. Drinks should be numbered sequentially, starting at 1, in the order they are displayed in the menu.
8. The in-stock indicator should be either "true" or "false".
9. Should read from the standard input stream, one command per line.
10. No prompts or other extraneous user messages should be displayed.
11. Blank input lines should be ignored.
12. The system should re-display the inventory and menu immediately after any applicable message.

# Technical restrictions

* It is not required that the initial machine configuration (inventory counts, available drinks and prices, etc.) be dynamic. In particular, it is acceptable to perform this initialization in code, rather than reading the configuration from an external file or database.
* Your program should be flexible enough to allow new drinks to be added to the menu without requiring extensive code changes.
* Make sure your program works correctly for all combinations of inputs. You may include automated tests
* Extensive inline or method-level comments are not required, unless you want to include them to highlight particular aspects of your design or implementation.

# Main interfaces, classes and Exceptions

* BaristaMachine - an interface which defines public API of Barista-Matic machine.
* BaristaMachineImpl - a general purpose implementation of BaristaMachine interface.
* Drink - Declare public API for drinks that are provided by Barista-Matic machine.
* DrinkImpl - A concrete class representing drink.
* Ingredient - Declare public API for ingredients that go into making the drink.
* IngredientImpl - Concrete class implementation for ingredients that go into making the drink.
* DrinkRefName – A type-safe enumerator to represent reference names of available drinks that are supported by machine.
* IngredientRefName – A type-safe enumerator to represent reference names of available ingredients along with its unit prices.
* DrinkDataService - Provide all necessary data structures that are required for creating the drink through the Drink model.
* IngredientDataService - Provide all necessary data structures that are required for preparing ingredients that go into making drinks through the Ingredient model.
* InventoryService Declare all operations that are need to be done on the inventory to keep tracking the ingredients amounts and to determine its availability.
* InvalidDrinkNumberException - thrown when user enters an invalid command.
* OutOfStockException - thrown when machine doesn't have enough ingredients for the selected drink.

# Data structures used

* Map data structure is used for representing:
  + Ingredient, units pair for drink.
  + Ingredient, unit price pair.
* Collection data structure is used for representing list of available drinks.
* EnumMap data structure is used for representing IngredientRefName, Ingredient pair. I Preferred EnumMap over TreeMap because it's faster and the order is maintained in which the enumerator constants are declared, gives flexibility to change the order from enumerator itself.

# Design decisions

* “java.lang.Enum” is used to represent as reference name for Ingredient, because of following benefits:  
  1- Compile time safety against entering an invalid ingredient name.  
  2- No need to write code for checking if selected ingredient is valid.  
  3- Reusable and well encapsulated.

4- All enumerators are acting as data source for the simulator application.

# Assumptions

* The coffee dispensing machine is able to dispense a fixed set of possible drinks by combining these ingredients in different amounts but ingredients are not customizable i.e. in case of the user asking for “Coffee” drink, he cannot change the amounts of any of the ingredients such as adding one more unit of sugar or removing the sugar at all.

# Reference Documentation

For further reference, please consider the documents folder inside the project folder:

* UML class diagram
* Generated JavaDoc