Implementation Descriptions

A Vending Machine could be used to order more things say eatables along with drinks, so I have implemented it in **modular** way such that more functionalities as per the requirement could be added. Code is **sufficient** along with additional features of milk and sugar.

General Working

- For now I have created functions in a way that it will give two options eat or drink.
- Considering scope of assignment it will directly choose the drink option.
- Then it will ask user whether he will like Tea or Coffee.
- Then user can choose the sort, sugar and milk amount.
- Machine will update user that their order is in process an when it's completed.

There are separate classes for each Beverage which are inheriting general features from parent Beverage Class thus high **cohesion** less **coupling** and **no duplicity**.

Code along with UML is easy to **understand** and **extend**. Suppose if developer wants to add more coffee or tea types, he just have to update the arraylist(flexible data structure) in that particular class. There are separate function for receiving order, preparing and serving, so that if in any stage more functionalities can be added. Also, if more drinks are to be added developer can create a new class inheriting beverages and as most of the functionality is already implemented. Thus code is **efficient, robust and flexible.**

Also, user is shown only the instructions and update messages of the order processing and completed rest most of the information is protected or private, thus **Information Hiding** occurs.

Functions execution order:

```
orderInstructions()
placeOrder()
        chooseMainFoodType()
                orderBeverage()
                        inStartPrepState()
                                askBeverageType()
                                addMilk()
                                addSugar()
                        inPreparingState()
                        inServingState()
```

Other:

addTypesOfBeverage();

UML Class Diagram.