Thank you for the feedback, the service layer (Consumer Service and Supplier Service) has been added to the application to provide more flexibility and decoupling. Factory Method pattern was introduced to support different type of vending machines in the future too.

Once again, thanks for your time and consideration!

1. How to build and run this application
   a) Download from https://github.com/bigzj/VendingMachine or unzip the zip file to any folder
   b) Use Eclipse or any IDE to "import existing Maven project", or build from command line using Maven
   c) Run Unit Test from org.icann.vendingmachine.test.VendingMachineTest

2. High Level Design
   a) Core Java libraries are the only dependencies of the main application; whereas JUnit4 is used for Unit Testing.
   b) com.icann.vendingmachine – the main package of the application
      i. VendingMachine – the abstract class which is the base of all different types of vending machines
   c) com.icann.vendingmachine.impl – the implementations of all different types of vending machines
      i. CoinSodaVendingMachine – the concrete class of vending machine
   d) com.icann.vendingmachine.factory – the factory interface to create different types of vending machines
      i. IVendingMachineFactory – the factory interface
   e) com.icann.vendingmachine.factory.impl – the concrete factory class to create different types of vending machines
      i. VendingMachineFactory – the concrete factory
   f) com.icann.vendingmachine.model – the data models
      i. Product – Enumeration, contains all products that the machine sells.
      ii. Coin – Enumeration, contains all the coin types that the machine takes
      iii. ProductAndChange, encapsulates the return of the confirmPurchase() method, contains the product that the user selected and changes of the coins if any
   g) org.icann.vendingmachine.service – the services of the application
      i. IConsumerService – the interface for consumer services, such as select product, confirmPurchase, etc
      ii. ISupplierService – the interface for supplier services, such as refill product, withdraw, etc
   h) org.icann.vendingmachine.service.impl – the implementation of the services
      i. CoinConsumerService – concrete class for consumer service
      ii. SodaSupplierService – concrete class for supplier service
   i) com.icann.vendingmachine.exception – the business exceptions
      i. NoInventoryException – thrown from the selectProduct() method if the product selected has no inventory
      ii. NoExactChangeException – thrown from the completePurchase() method if no exact change can be returned from the current coins that the machine has
      iii. NotFullyPaidException – thrown from the completePurchase() method if the tenderedAmount is less than the price of the selected product

3. Low Level Design
   a) Factory method pattern is used to support the creation of different type of vending machines
   b) Consumer service and Supplier service is added as the service layer to provide more flexibility to the application for future expansion. Such as a new vending machine for soda but instead of taking coins, it is taking credit cards; we can reuse the soda supplier service

but create a new credit card consumer service class

c) Enumeration is used for product and coin, so we do not have do validation of them all over the place

d) Integer is used to represent coin or money value in cents. It is easier to handle than double or BigDecimal and it is more efficient

e) Class diagram is on the next page as well as in the root directory of the project

4. Assumptions

   a) No database is required; there is no mention of persistence in the requirement. To simplify the build / test process, I decided to not using a database. But a persistence layer can be easily added later

   b) It is a single threaded application, only one user will use the machine at a time, same as real world case

   c) If the previous user did not complete the purchase and left, e.g. inserted some coins and then left, the current user will be able to use the previously inserted coins; same as real world case

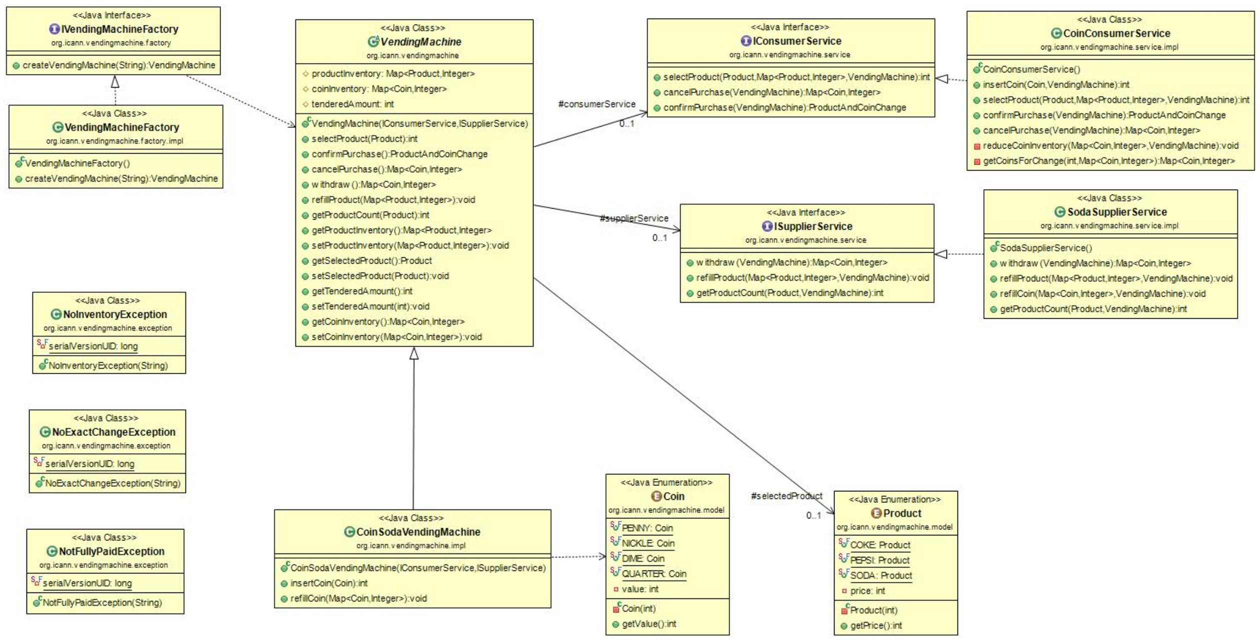
5. Unit Testing

   a) JUnit4 is used to perform the unit testing and it is the only dependency of the application besides core Java libraries

   b) All the public methods in the VendingMachine and CoinSodaVendingMachine class are tested, including alternative flows (exceptional cases)

   c) The sequence of the 9 tests are important, so FixMethodOrder by the method name is used

Once again, thank you for your time and consideration!



<<Java Interface>>
IVendingMachineFactory
org.icann.vendingmachine.factory
createVendingMachine(String):VendingMachine
<<Java Class>>
VendingMachineFactory
org.icann.vendingmachine.factory.impl
VendingMachineFactory()
createVendingMachine(String):VendingMachine
<<Java Class>>
VendingMachine
org.icann.vendingmachine
productInventory: Map<Product,Integer>
coinInventory: Map<Coin,Integer>
tenderedAmount: int
VendingMachine(IConsumerService,ISupplierService)
selectProduct(Product):int
confirmPurchase():ProductAndCoinChange
cancelPurchase():Map<Coin,Integer>
withdraw():Map<Coin,Integer>
refillProduct(Map<Product,Integer>):void
getProductCount(Product):int
getProductInventory():Map<Product,Integer>
setProductInventory(Map<Product,Integer>):void
getSelectedProduct():Product
setSelectedProduct(Product):void
getTenderedAmount():int
setTenderedAmount(int):void
getCoinInventory():Map<Coin,Integer>
setCoinInventory(Map<Coin,Integer>):void
#consumerService
0..1
<<Java Interface>>
IConsumerService
org.icann.vendingmachine.service
selectProduct(Product,Map<Product,Integer>,VendingMachine):int
cancelPurchase(VendingMachine):Map<Coin,Integer>
confirmPurchase(VendingMachine):ProductAndCoinChange
<<Java Class>>
CoinConsumerService
org.icann.vendingmachine.service.impl
CoinConsumerService()
insertCoin(Coin,VendingMachine):int
selectProduct(Product,Map<Product,Integer>,VendingMachine):int
confirmPurchase(VendingMachine):ProductAndCoinChange
cancelPurchase(VendingMachine):Map<Coin,Integer>
reduceCoinInventory(Map<Coin,Integer>,VendingMachine):void
getCoinsForChange(int,Map<Coin,Integer>):Map<Coin,Integer>
#supplierService
0..1
<<Java Interface>>
ISupplierService
org.icann.vendingmachine.service
withdraw(VendingMachine):Map<Coin,Integer>
refillProduct(Map<Product,Integer>,VendingMachine):void
getProductCount(Product,VendingMachine):int
<<Java Class>>
SodaSupplierService
org.icann.vendingmachine.service.impl
SodaSupplierService()
withdraw(VendingMachine):Map<Coin,Integer>
refillProduct(Map<Product,Integer>,VendingMachine):void
refillCoin(Map<Coin,Integer>,VendingMachine):void
getProductCount(Product,VendingMachine):int
<<Java Class>>
NoInventoryException
org.icann.vendingmachine.exception
serialVersionUID: long
NoInventoryException(String)
<<Java Class>>
NoExactChangeException
org.icann.vendingmachine.exception
serialVersionUID: long
NoExactChangeException(String)
<<Java Class>>
NotFullyPaidException
org.icann.vendingmachine.exception
serialVersionUID: long
NotFullyPaidException(String)
<<Java Class>>
CoinSodaVendingMachine
org.icann.vendingmachine.impl
CoinSodaVendingMachine(IConsumerService,ISupplierService)
insertCoin(Coin):int
refillCoin(Map<Coin,Integer>):void
<<Java Enumeration>>
Coin
org.icann.vendingmachine.model
PENNY: Coin
NICKLE: Coin
DIME: Coin
QUARTER: Coin
value: int
Coin(int)
getValue():int
#selectedProduct
0..1
<<Java Enumeration>>
Product
org.icann.vendingmachine.model
COKE: Product
PEPSI: Product
SODA: Product
price: int
Product(int)
getPrice():int