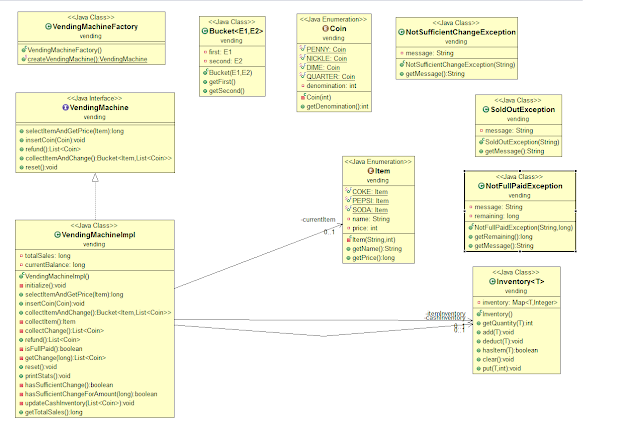
**Vending Machine Design Document in Java**

Here is a sample design document for the Vending Machine problem. Well, it's not that great but conveys my design decisions in text format. Since in a real test, time is critical you need to balance your time between coding, testing, and designing activity, it's better to choose text over an image. If you are good with UML then adding a class diagram would certainly help here.  
  
In short a design document in Java Should include  
- description of the solution  
- design decision and data structures  
- All classes and their responsibility  
- description of the package  
- description of methods  
- the motivation of decision e.g. why this design pattern, why enum, why BigDecimal etc?  
- Flow Chart of a couple of use cases  
- UML Diagram  
- Assumption  
- Risk  
- Benefits



**1. High-level Design**

Includes an overview of the problem, not necessary if you are writing this as part of the test because the evaluator should be familiar with problem specification. Important if your design document is intended for someone who is not very familiar with the problem domain.  
  
    - Main interface, classes, and Exceptions  
          - **VendingMachine**- an interface that defines public API of VendingMachine  
          - **VendingMachineImpl**- a general-purpose implementation of the VendingMachine interface  
          - **Inventory**- A type-safe inventory for holding objects, which is an ADAPTER or WRAPPER over java.util.Map  
          - **Item**- type-safe Enum to represent items supported by vending machines.  
          - **Coin**- type-safe Enum to represent coins, which is acceptable by a vending machine.  
          - **Bucket**- A Holder class for holding two types together.  
          - **SoldOutException**- thrown when the user selects a product that is sold out  
          - **NotSufficientChangeException**- thrown when Vending machine doesn't have enough change to support the current transaction.  
  
          - **NotFullPaidException** - thrown when the user tries to collect an item, without paying the full amount.  
       
    - Data structures used  
          - Map data structure is used to implement cash and item inventories.  
          - The List is used to returning changes because it can contain duplicates, i.e. multiple coins of the same denomination.  
  
     
    - Motivation behind design decisions  
         - Factory design pattern is used to encapsulate the creation logic of VendingMachine.  
         - Adapter pattern is used to create Inventory by wrapping java.util.Map  
         - java.lang.Enum is used to represent Item and Coins, because of the following benefits  
                - compile-time safety against entering an invalid item and invalid coin.  
                - no need to write code for checking if the selected item or entered coin is valid.  
                - reusable and well encapsulated.  
         - long is used to represent price and totalSales, which are the amount because we are dealing in cents.  
           Not used BigDecimal to represent money, because the vending machine can only hold a limited amount, due to the finite capacity of coin inventory.  
  
         -

**2. Low Leven Design**

**1) Methods**  
        -  The getChange() method uses a greedy algorithm to find out whether we have sufficient coins to support an amount.  
  
    - Initialization  
         - When Vending Machine will be created, it's initialized with the default cash amount and item inventory. current with quantity 5 for each coin and item.  
  
  
  
**2) Testing Strategy**  
   - Three primary test cases to testWithExactPrice(), testWithMorePrice() and testRefund() to cover general usecase.  
   - Negative test cases like testing SoldOutException, NotSufficientChangeException or NotFullPaidException  
   -  
  
**3) Benefits**  
   - The design uses Abstraction to create reusable, small classes which are easier to read and test.  
   - Encapsulating Items and Coins in their respective class makes it easy to add new Coins and Items.  
   - Inventory is a general-purpose class and can be used elsewhere, It also encapsulates all inventory operations.  
  
**4) Assumption**  
   - Vending Machine is single-threaded, only one user will operate at a time.  
   - A call to reset() will reset the item and balance i.e. make them zero.

**3 . UML Diagram Vending Machine in Java**

Here is our UML diagram for this solution, generated in Eclipse by using ObjectAid plugin. It let you create a UML class diagram from Java code itself. This diagram shows that VendingMachineImpl extends or implements the VendingMachine interface. It also shows that it is associated with Item and Inventory classes.

