**How to design Vending Machine in Java**

Here is the complete code of Vending Machine in Java.  
  
  
**VendingMachine.java**  
The public API of vending machine, usually all high-level functionality should go in this class

package vending;

**import** java.util.List;

/\*\*

\* Decleare public API for Vending Machine

\* @author Javin Paul

\*/

**public** **interface** VendingMachine {

**public** long selectItemAndGetPrice(Item item);

**public** **void** insertCoin(Coin coin);

**public** List<Coin> refund();

**public** Bucket<Item, List<Coin>> collectItemAndChange();

**public** **void** reset();

}

**VendingMachineImpl.java**  
A sample implementation of VendingMachine interface represents a real world Vending Machine , which you see in your office, bus stand, railway station and public places.

package vending;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.List;

/\*\*

\* Sample implementation of Vending Machine in Java

\* @author Javin Paul

\*/

**public** class VendingMachineImpl **implements** VendingMachine {

**private** Inventory<Coin> cashInventory **=** **new** Inventory<Coin>();

**private** Inventory<Item> itemInventory **=** **new** Inventory<Item>();

**private** long totalSales;

**private** Item **currentItem**;

**private** long currentBalance;

**public** VendingMachineImpl(){

**initialize**();

}

**private** **void** **initialize**(){

//initialize machine with 5 coins of each denomination

//and 5 cans of each Item

**for**(Coin c **:** Coin.values()){

cashInventory.put(c, 5);

}

**for**(Item i **:** Item.values()){

itemInventory.put(i, 5);

}

}

@Override

**public** long selectItemAndGetPrice(Item item) {

**if**(itemInventory.hasItem(item)){

**currentItem** **=** item;

**return** **currentItem**.getPrice();

}

**throw** **new** SoldOutException("Sold Out, Please buy another item");

}

@Override

**public** **void** insertCoin(Coin coin) {

currentBalance **=** currentBalance **+** coin.getDenomination();

cashInventory.**add**(coin);

}

@Override

**public** Bucket<Item, List<Coin>> collectItemAndChange() {

Item item **=** collectItem();

totalSales **=** totalSales **+** **currentItem**.getPrice();

List<Coin> **change** **=** collectChange();

**return** **new** Bucket<Item, List<Coin>>(item, **change**);

}

**private** Item collectItem() throws NotSufficientChangeException,

NotFullPaidException{

**if**(isFullPaid()){

**if**(hasSufficientChange()){

itemInventory.deduct(**currentItem**);

**return** **currentItem**;

}

**throw** **new** NotSufficientChangeException("Not Sufficient change in

Inventory");

}

long remainingBalance **=** **currentItem**.getPrice() **-** currentBalance;

**throw** **new** NotFullPaidException("Price not full paid, remaining : ",

remainingBalance);

}

**private** List<Coin> collectChange() {

long changeAmount **=** currentBalance **-** **currentItem**.getPrice();

List<Coin> **change** **=** getChange(changeAmount);

updateCashInventory(**change**);

currentBalance **=** 0;

**currentItem** **=** null;

**return** **change**;

}

@Override

**public** List<Coin> refund(){

List<Coin> refund **=** getChange(currentBalance);

updateCashInventory(refund);

currentBalance **=** 0;

**currentItem** **=** null;

**return** refund;

}

**private** boolean isFullPaid() {

**if**(currentBalance >**=** **currentItem**.getPrice()){

**return** true;

}

**return** false;

}

**private** List<Coin> getChange(long amount) throws NotSufficientChangeException{

List<Coin> changes **=** Collections.EMPTY\_LIST;

**if**(amount > 0){

changes **=** **new** ArrayList<Coin>();

long balance **=** amount;

**while**(balance > 0){

**if**(balance >**=** Coin.QUARTER.getDenomination()

**&&** cashInventory.hasItem(Coin.QUARTER)){

changes.**add**(Coin.QUARTER);

balance **=** balance **-** Coin.QUARTER.getDenomination();

**continue**;

}**else** **if**(balance >**=** Coin.DIME.getDenomination()

**&&** cashInventory.hasItem(Coin.DIME)) {

changes.**add**(Coin.DIME);

balance **=** balance **-** Coin.DIME.getDenomination();

**continue**;

}**else** **if**(balance >**=** Coin.NICKLE.getDenomination()

**&&** cashInventory.hasItem(Coin.NICKLE)) {

changes.**add**(Coin.NICKLE);

balance **=** balance **-** Coin.NICKLE.getDenomination();

**continue**;

}**else** **if**(balance >**=** Coin.PENNY.getDenomination()

**&&** cashInventory.hasItem(Coin.PENNY)) {

changes.**add**(Coin.PENNY);

balance **=** balance **-** Coin.PENNY.getDenomination();

**continue**;

}**else**{

**throw** **new** NotSufficientChangeException("NotSufficientChange,

Please try another product");

}

}

}

**return** changes;

}

@Override

**public** **void** reset(){

cashInventory.**clear**();

itemInventory.**clear**();

totalSales **=** 0;

**currentItem** **=** null;

currentBalance **=** 0;

}

**public** **void** printStats(){

System.out.println("Total Sales : " **+** totalSales);

System.out.println("Current Item Inventory : " **+** itemInventory);

System.out.println("Current Cash Inventory : " **+** cashInventory);

}

**private** boolean hasSufficientChange(){

**return** hasSufficientChangeForAmount(currentBalance **-** **currentItem**.getPrice());

}

**private** boolean hasSufficientChangeForAmount(long amount){

boolean hasChange **=** true;

**try**{

getChange(amount);

}**catch**(NotSufficientChangeException nsce){

**return** hasChange **=** false;

}

**return** hasChange;

}

**private** **void** updateCashInventory(List **change**) {

**for**(Coin c **:** **change**){

cashInventory.deduct(c);

}

}

**public** long getTotalSales(){

**return** totalSales;

}

}

**VendingMachineFactory.java**  
A Factory class to create different kinds of Vending Machine

package vending;

/\*\*

\* Factory class to create instance of Vending Machine, this can be extended to create instance of

\* different types of vending machines.

\* @author Javin Paul

\*/

**public** class VendingMachineFactory {

**public** **static** VendingMachine createVendingMachine() {

**return** **new** VendingMachineImpl();

}

}

**Item.java**  
Java Enum to represent Item served by Vending Machine

package vending;

/\*\*

\* Items or products supported by Vending Machine.

\* @author Javin Paul

\*/

**public** enum Item{

COKE("Coke", 25), PEPSI("Pepsi", 35), SODA("Soda", 45);

**private** String **name**;

**private** **int** price;

**private** Item(String **name**, **int** price){

**this**.**name** **=** **name**;

**this**.price **=** price;

}

**public** String getName(){

**return** **name**;

}

**public** long getPrice(){

**return** price;

}

}

**Coin.java**  
Another Java Enum to represent Coins supported by Vending Machine

package vending;

/\*\*

\* Coins supported by Vending Machine.

\* @author Javin Paul

\*/

**public** enum Coin {

PENNY(1), NICKLE(5), DIME(10), QUARTER(25);

**private** **int** denomination;

**private** Coin(**int** denomination){

**this**.denomination **=** denomination;

}

**public** **int** getDenomination(){

**return** denomination;

}

}

**Inventory.java**  
A Java class to represent an Inventory, used for creating case and item inventory inside Vending Machine.

package vending;

**import** java.util.HashMap;

**import** java.util.Map;

/\*\*

\* An Adapter over Map to create Inventory to hold cash and

\* Items inside Vending Machine

\* @author Javin Paul

\*/

**public** class Inventory<T> {

**private** Map<T, Integer> inventory **=** **new** HashMap<T, Integer>();

**public** **int** getQuantity(T item){

Integer **value** **=** inventory.**get**(item);

**return** **value** **==** null**?** 0 **:** **value** ;

}

**public** **void** **add**(T item){

**int** count **=** inventory.**get**(item);

inventory.put(item, count**+**1);

}

**public** **void** deduct(T item) {

**if** (hasItem(item)) {

**int** count **=** inventory.**get**(item);

inventory.put(item, count **-** 1);

}

}

**public** boolean hasItem(T item){

**return** getQuantity(item) > 0;

}

**public** **void** **clear**(){

inventory.**clear**();

}

**public** **void** put(T item, **int** quantity) {

inventory.put(item, quantity);

}

}

**Bucket.java**  
A parameterized utility class to hold two objects.

package vending;

/\*\*

\* A parameterized utility class to hold two different object.

\* @author Javin Paul

\*/

**public** class Bucket<E1, E2> {

**private** E1 **first**;

**private** E2 second;

**public** Bucket(E1 **first**, E2 second){

**this**.**first** **=** **first**;

**this**.second **=** second;

}

**public** E1 getFirst(){

**return** **first**;

}

**public** E2 getSecond(){

**return** second;

}

}

**NotFullPaidException.java**  
An Exception, thrown by Vending Machine when a user tries to collect an item, without paying the full amount.

package vending;

**public** class NotFullPaidException extends RuntimeException {

**private** String **message**;

**private** long remaining;

**public** NotFullPaidException(String **message**, long remaining) {

**this**.**message** **=** **message**;

**this**.remaining **=** remaining;

}

**public** long getRemaining(){

**return** remaining;

}

@Override

**public** String **getMessage**(){

**return** **message** **+** remaining;

}

}

**NotSufficientChangeException.java**  
Vending Machine throws this exception to indicate that it doesn't have sufficient change to complete this request.

package vending;

**public** class NotSufficientChangeException extends RuntimeException {

**private** String **message**;

**public** NotSufficientChangeException(String string) {

**this**.**message** **=** string;

}

@Override

**public** String **getMessage**(){

**return** **message**;

}

}

**SoldOutException.java**  
The Vending Machine throws this exception if the user request for a product which is sold out

package vending;

**public** class SoldOutException extends RuntimeException {

**private** String **message**;

**public** SoldOutException(String string) {

**this**.**message** **=** string;

}

@Override

**public** String **getMessage**(){

**return** **message**;

}

}