# CashMan

## Problem Statement:

Given an amount *W*, also a positive integer, to find a set of non-negative (positive or zero) integers {*x*1, *x*2, ..., *xn*}, with each *xj* representing how often the coin with value *wj* is used, with minimal total number of coins, i.e., For (j: 1 to N) *Sum(xj)* should beminimum, subject to *Sum(wj \* xj) = W*.{\displaystyle \sum \_{j=1}^{n}w\_{j}x\_{j}=W.}{\displaystyle \sum \_{j=1}^{n}x\_{j}}

## Glossary and Terminology:

* **Denomination Type:** Represents the valid denomination types supported by CashMan application. (EX: 5$, 10$, 20$, 50$ etc).
* **Denomination Count:** Represents the count of various denomination types available in the system.
* **Denomination:** An entity representation of given *DenominationType* and its total Count Ex: Denomination (5$, 10) represents the count of 5$ available in the CashMan application and its count as 10.
* **Available Total Currency:** Represents the total currency available in the CashMan application, i.e., *Sum(denominationType(J) \* denominationCount(j)).*

## Scope:

### InScope:

* CashMan Withdraw algorithm: Core Algorithm for withdrawing the amount from the given CashMan available currency.
* CashMan DynamoDB repository: DynamoDB based repository for the available currency in CashMan application.
* CashMan Default repository: Repository of available currency persisted and retrieved as part of memory.
* CashMan APIs: Spring boot-based API’s for withdraw, addition, remove denominations from the available currency.

### OutOfScope:

* Support of Transactions. For the current version of document,

## Requirements:

Following are broad use cases and requirements for CashMan application.

* The CashMan application must know how many of each type of bank note it has. It should be able to report back how much of each note it has.
* The CashMan application should be possible to tell it that it has so many of each type of note during initialisation. After initialisation, it is only possible to add or remove notes.
* The CashMan application must support $20 and $50 Australian denominations.
* The CashMan application should be able to dispense legal combinations of notes. For example, a request for $100 can be satisfied by either five $20 notes or 2 $50 notes. It is ***not*** required to present a list of options.
* The CashMan application should report an error condition If a request cannot be satisfied due to failure to find a suitable combination of notes, in some fashion. For example, in an ATM with only $20 and $50 notes, it is not possible to dispense $30.
* The CashMan application should reduce the amount of available cash in the machine post dispensation.
* The CashMan application should not reduce the amount of available cash in case of an error.

## CashMan WithDraw Algorithm:

We will provide the following CashMan WithDraw algorithms for the mentioned CashMan problem.

1. **Greedy Algorithm:** A recursive mechanism to iterate over the available currency, deriving the required denomination at each stage for the provided input.
2. **Optimal Dynamic Programming variant:** Below we provide the dynamic programming algorithm solve the given problem.

* The mentioned problem is a semi NP-hard problem.
* Let’s say the amount to be withdrawn is x. Set WithDrawAmount = x.
* Number of denominations required = min (1 + numberOfCoins(WithDrawAmount-di)), where i <= number of denominations.
* A recursive solution to the mentioned problem finding combinations of outputs, will require semi polynomial time (i.e., 2 power n) and so time consuming. It may not be an optimal solution.
* Instead dynamic programming approach, where we break the original problem in to sub problems and use a matrix to keep track of the optimal solutions to sub problems, and returns each of those sub problems just once, and storing their solutions.
* We reduce the complexity by storing some of the past results for each denomination, so we can avoid recalculating the results we already know.
* We maintain a *denominationTrackingTable* to keep track of the solutions to sub problems. Then before we compute a new minimum, we first check the table to see if a result is already known.
* Set DenominationTypes = [1$, 2$, 5$, 10$, 20$, 100$]. /\*Assign valid list of denominations. \*/
* Set denominationTypesCount = len(DenominationTypes).
* **Set int**[][] denominationTrackingTable = **new int**[denominationTypesCount + 1][withDrawAmount + 1]; //Denomination tracking table with optimal solutions stored for each subproblem.
* For every denomination, we can either include the denomination or exclude it.
* For I in [0, withDrawAmount], do the following:
* Check, if number of denominations for I, is already found and available in denominationTrackingTable.
* If yes, use the available result
* If (denomination value <= WithDrawAmount):
* **Include the denomination**: Verify the count of denominations included for this type. If it is in the available count, reduce the input WithDrawAmount by denomination value, and then use the sub problem solution using (WithDrawAmount - denomination).
* **Exclude the denomination**: solution for the same amount without considering that denomination.
* Else If denomination value is greater than the amount, then we can’t consider that denomination, so solution will be without considering that denomination.

## Synchronization/Concurrency:

We will synchronize the available currency set in the CashMan application for the *withdraw, add, remove* operations. More details about these operations mentioned below. There will only be single instance of CashMan dealing with these operations per given process.

CashMan Repository

CashMan repository stores the available currency of denominations for CashMan application. It by default provides two sets of repositories: DynamoDB based and Default.

* + DynamoDB Repository: Stores and retrieves the denomination information from DynamoDB.
  + Default CashMan Repository: Stores and retrieves the denomination information inside memory as part of application. Initialized through a bean.

Following are the repository API’s.

**public interface** CashManRepository {  
 */\*\*  
 \* Retrieves the initial denomination of currency available from external store say DynamoDB.  
 \* This currency will be used to initialize the CashMan application for further vending.  
 \** ***@return*** *TreeSet with Denomination.  
 \*/* Set<Denomination> retrieveDenomination();  
  
 */\*\*  
 \* Persist the updated denomination of currency available to an external store say DynamoDB.  
 \** ***@param denominationSet*** *Denomination set to be persisted.  
 \*/* **void** persistDenomination(**final** Set<Denomination> denominationSet);  
}

## CashMan Application:

CashMan interface for CashMan application. Following are the API’s, provided as extension point for the CashMan application.

*/\*\*  
 \* Initializes the CashMan application with required currency from repository.  
 \* Provides to the user available denomination types and counts in CashMan application.  
 \*/*

**void** initialize();  
  
*/\*\*  
 \* With draws the given amount from CashMan application and provides the list of Denominations matched with the provided input amount.  
 \** ***@param withDrawAmount*** *\** ***@return*** *\*/*Set<Denomination> withDraw(**final int** withDrawAmount) **throws** CashNotAvailableException;  
  
*/\*\*  
 \* Adds the provided denomination set to the available currency.  
 \** ***@param denominationList*** *\*/***void** addToAvailableCurrency(**final** Set<Denomination> denominationList);  
  
*/\*\*  
 \* Removes the denominations from the available currency.  
 \** ***@param denominationList*** *\*/***void** removeFromAvailableCurrency(**final** Set<Denomination> denominationList);  
  
*/\*\*  
 \* Provides the available denomination count for a given denomination type.  
 \* (EX: count of 5$ notes in the available currency).  
 \** ***@param DenominationType*** *\** ***@return*** *count of denomination for the given type.**\*/***long** getDenominationCount(**final** DenominationType DenominationType);  
  
*/\*\*  
 \* provides the total currency (Set of DenominationType(5$,10$ etc) and their count in CashMan application.  
 \** ***@return*** *set with available total currency in CashMan application.  
 \*/*Set<Denomination> totalAvailableCurrency();

## Pre-Requisites:

* Amazon AWS Java SDK: 1.11.106
* Junit: 4.13, mockito:
* ApacheMaven: 4.x
* Spring: 5.0.3
* SpringBoot: 1.5.10
* AWS Account for DynamoDB (In case you want to use DynamoDB as repository). The implementation as well adds a default repository with predefined denomination for usage.

## Running the CashMan Application

mvn spring-boot:run

Create the Jar:

mvn clean package

Run the jar:

java -jar spring-boot-web-application-example-0.0.1-SNAPSHOT.war

mvn clean dependency:tree

mvn clean compile

## Running executable war from command line

Use the following command to run the war from the CMD.

>java -jar spring-boot-web-application-example-0.0.1-SNAPSHOT.war